SCALING RELATIONSHIPS FOR POWER DEPOSITION AND ION BOMBARDMENT IN RADIO-FREQUENCY PLASMAS

by

Joanne Liu

B. S. Chemical Engineering
University of California, Berkeley
(1986)

M. S. Chemical Engineering Practice
Massachusetts Institute of Technology
(1988)

Submitted to the Department of
Chemical Engineering in Partial Fulfillment of
the Requirements for the
Degree of

DOCTOR OF PHILOSOPHY
in Chemical Engineering
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 1993

Signature of the Author _____

Department of Chemical Engineering
October 1992

Certified by _____

Herbert H. Sawin
Professor of Chemical Engineering, Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by _____

Robert E. Cohen
Chairman, Departmental Committee on Graduate Students
Chemical Engineering

SCALING RELATIONSHIPS FOR POWER DEPOSITION AND ION BOMBARDMENT IN RADIO-FREQUENCY PLASMAS

by

Joanne Liu

Submitted to the Department of Chemical Engineering
on November 17, 1992 in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy in
Chemical Engineering

ABSTRACT

A plasma research reactor has been designed and constructed for the purpose of finding the relationships between plasma properties and operating parameters. The plasma properties considered are the ion energy, angle, and flux on the electrode surface, the plasma sheath width, and the voltage and current waveforms across the plasma. The operating parameters examined are electrode spacing and diameter, gas pressure, power, and magnetic field.

Some unique features in the design of this reactor are one, well confined plasmas at many electrode configurations, and two, the ability to measure ion bombardment properties. A well confined plasma is necessary for defining the plasma volume, and the grounded and powered surface in contact with the plasma. These factors are crucial to understanding where power is deposited within the plasma. The ion analyzer designed and constructed for this thesis is the first instrument to measure the ion bombardment angle.

Scaling relationships for power as a function of current, pressure, magnetic field strength, electrode diameter, and electrode spacing are presented here. Power deposition in the plasmas can be separated into two regions, bulk and sheath power depositions. Bulk power deposition follows

$$\frac{P_b}{Ad^{1/2}} = \kappa_1 \frac{I_o}{A} p^{1/2} \quad ,$$

and with magnetic field

$$P_b - P_b|_{B=0} = \kappa_2 \frac{BI_o}{p^{1/2}} \, .$$

Sheath power deposition follows

$$\frac{P_s}{A} = \frac{\kappa_3}{p^{1/2}} \left(\frac{I_o}{A} \right)^{2.5} \, .$$

Some of the physics associated with these relationships are proposed using a simple plasma model,

and data from argon and SF_6 plasmas are used to show the validity of the proposed relationships.

The argon ion bombardment energy and flux are shown to depend on power, sheath width, and pressure, but not to the electrode geometry. The average ion bombardment energy is proportional to the sheath voltage and decreases with pressure because the ions go through more collisions as the pressure increases. However, the ion energy does not continue to decrease with increasing pressure, but saturates once there are sufficient collisions in the sheath for the energy gained from the electric field to balance the energy lost through collisions. The average ion energy in argon plasmas can be expressed as

$$\frac{\varepsilon}{V_s} = 0.14\ p^{-0.32} \qquad \text{where} \quad p = \text{Torr}.$$

Similarly the average ion bombardment angle increases with increasing pressure, then plateaus when there are sufficient collisions to produce fully-developed distributions.

Monte Carlo techniques are used to predict the ion bombardment energy and angle. These simulations are compared to the measured data, showing that the model and experiment are in agreement in most cases. The model helps explain the multiple peaks in the measured ion energy distributions and is useful for estimating the energy, angle, and flux of energetic neutrals created from collisions with ions. Energetic neutral are shown to be a significant fraction of the energetic particles bombarding the surface in charge-exchange collision dominated plasmas.

Thesis Supervisor: Herbert H. Sawin
Title: Professor

ACKNOWLEDGEMENTS

This is really it, the end of more than six years of graduate school experience, the end of **twenty-two** continuous years of school! The "Graduate School" experience has, without doubt, been the time where I learned the most, both academically and personally.

First, I would like to thank my thesis supervisor, Professor Herb Sawin, who inspired me with his enthusiasm for learning and research. Thanks for all the helpful advice and insight you provided in my project. Thanks are also due to my thesis committee, Professors Robert Brown and Karen Gleason, for the questions and suggestions that helped me complete my project.

Financial support for my first year at MIT was provided by an Ida Green Fellowship. The primary financial support for my research was provided by the East Fishkill facility of the IBM Corporation.

It has been fun and a learning experience being part of the METL group for the past six years. I am especially thankful to Dr. Subodh Kulkarni, Dr. Jean-Philippe Nicolai, and the future Dr. Gerald Gibson for being so smart and willing to share some of the results of that brilliance my way :-). You have been great friends. Gib, life at the lab would not have been as enjoyable had I not had the opportunity to share conversations and insights ranging in topic from politics, science, society, and religion with you. Thank you Linda Kiss for setting the standard in our crazy group. I value the summer tennis games where we got lots of exercise chasing the ball around the court and the privilege of being part of your wedding. Not only have these people been helpful in my project, but they have proofread parts of my thesis with willingness and thoroughness: Vivek, Gil, and Bill. I really appreciate it! The craziness, laughter, maturity, and team spirit all make METL a special group of people; all the above plus Evangelos, Jeff, Tim, Dave, Gavin, John, and Igor.

Some of my best friends who have always wanted the best for me and who were never afraid to tell me things I needed to hear are: Yuri Kinoshita, Vicky Brandt, Ann Black, Susan Park, Kristine Drobot, Elsa Mak, Ethan Wenger, and Bryan Klassen. You have been with me through all the tough times and pointed me in the right direction. I am eternally grateful. I want to thank Kim Oakberg, Peter Rothschild, Dan Zachary, Ted Sung, Randy Saunders, Lisa Chou, Cedric Logan, for taking a genuine interest in my research and stopping by my office for the last three years (talk about consistency!) to offer words of encouragement, commiseration, or food. I am privileged and thankful to have such wonderful friends in Jose Elizondo, Jude "Banana" Federspiel, Harris "fly" Gilliam, Rob Grace, Jonathan Hardy, Rob "Mr. Muscle" Kim, Chris Lee, Dr. Howard Loree, John Oates, Mark Rawizza, Andrew Romain, Jim Ryan, Lisa "Squirt" Sopata, Mark Wintersmith, Jee-Lian Yap, Gillian Gatti, Agnes Gapud, Patti Kellett, Sharon Belville"@athena", and Mike Domroese.

I would not be where I am without the financial and emotional support of my family. My family is the BEST!!!! Many thanks go to my sister, Sharlene, who visited me often in my office and offered me special baked goods and dinners, even as she worked on her thesis at MIT. I am thankful for my sister, Lynette, who sacrificed a Christmas holiday at home to be with me in Boston as I studied for my qualifiers and for her continual words of encouragement through email, wherever she happened to be, Baltimore, Los Angeles, Dallas, or France. My parents are the ones

with the dream and the courage to come to a new country, learn a new language, and build the foundation for our lives. I am indebted to you, Mom and Dad, for the sacrifices you have made and how you have always encouraged me to strive for excellence in whatever I start. But most of all, I am grateful for the love that is always there. I LOVE you both!!! I am the luckiest person in the world because I am blessed with parents like you.

I thank most of all, my Lord and God, for my friends, my loving family, my research, and my job. Thank you for answering so many prayers and for changing me through circumstances and people. Your promise to me is great: "For I know the plans I have for you," declares the Lord, "plans to prosper you and not to harm you, plans to give you a hope and a future." (Jeremiah 29:11) I would never have made it through graduate school without you!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In the microelectronics industry, the goal is always to produce smaller and faster devices with high reliability and fast turn over time. Electronic components with dimensions on the order of microns are fabricated on substrate surfaces. Silicon wafers are the most common substrates used because, not only is silicon an abundant and naturally occurring element, but the technology for fabricating devices on silicon is many times more advance than that for other types of substrates such as gallium arsenide and germanium. Integrated microelectronic fabrication on wafers can be extremely complex, with many layers of insulating, conducting, and semiconducting materials patterned on top of the silicon substrate to form devices. Figure 1.1 shows a simple example of a semiconductor device, the transistor. The fabrication of this device requires many steps, such as doping the drain and source of the transistor, growth of insulating oxide layers between the transistor gate and the channel, deposition of metal lines that connect the transistor to other devices on the substrate, *etc.* Generally, these devices are made by blanketing the entire substrate with a film, *e.g.* aluminum or SiO_2, then selectively removing these films from specified locations to form patterns.

Selective removal of the film is done through a step called etching. Figure 1.2 schematically illustrates the etching process. A photoresist is spun onto the surface of the film. A patterned mask is placed over the film and the photoresist is exposed to x-ray or uv light. The photoresist is developed, leaving a patterned layer of photoresist on top of the film. The areas of the film not covered by the photoresist is etched, thus transferring the photoresist pattern onto the film. The two methods for removing the film in the etching step are wet etching and plasma etching. In wet etching, the wafer is dipped into a chemical bath, and the parts of the film not

covered by the photoresist dissolves away. Figure 1.3 (a) illustrates the wet etching results. Wet etching produces isotropic profiles with undercutting beneath the photoresist because etching occurs at nearly equal rates in all directions. As devices get smaller and are packed closer together, the aspect ratios of the trenches and vias become higher. If the etching step continues to produce undercutting of the photoresist, then the pattern is no longer accurately transferred from the photoresist to the film and the etched features will start to merge together.

For etching high aspect ratio features, plasmas can provide directional ion bombardment perpendicular to the surface to yield anisotropic etching rates. Figure 1.3 (b) shows the results from a plasma etched film. Anisotropic etching in plasma processes is a result of ion enhanced reactions. Winters (1988a) grouped ion-enhanced etching mechanisms into five categories: physical sputtering (Mogab & Levinstein 1980, Gerlach-Meyer *et al.* 1981, Tu *et al.* 1981), chemical sputtering (Tu *et al.* 1981, O'Brien *et al.* 1988), ion-induced mixing (Oostra *et al.* 1986, Winters 1988b), detrapping (Greene *et al.* 1988a), and ion-induced lattice damage (Greene *et al.* 1988a, Donnelly *et al.* 1984, Winters 1985, Bensaoula *et al.* 1987). In these mechanisms, the ions impart energy either to the lattice structure or to the reactant species on the surface, resulting in an increased reaction rate where there is ion bombardment. The etching yield is a function of the ion bombardment angle and energy (Gerlach-Meyer *et al.* 1981, Harper *et al.* 1981, Us *et al.* 1986). Threshold energies between 10 eV and 20 eV for ion-enhanced etching processes have been reported (Us *et al.* 1986, Allen *et al.* 1986).

1.1 PHYSICS OF PLASMAS

The plasmas used in semiconductor fabrication are created by applying radio frequency (rf) power, usually at 13.56 MHz, between two plates or electrodes to partially ionize the gas between the plates. This turns a chemically unreactive gas into reactive radicals, ions, and

electrons. Etching occurs as the plasma species react with the film on the wafer to form volatile products which are pumped away. Some of the plasma properties are listed in Table 1.1.

Table 1.1: Typical Plasma Properties (Sawin 1987)

Property	Range
Pressure	0.001 - 1.0 Torr
Electron Density	10^8 - 10^{12} /cm^3
Average Electron Energy	1 to 10 eV
Average Neutral and Ion Energy	0.025 to 0.035 eV
Ionized Fraction of Gas	10^{-4} to 10^{-7}
Power Dissipation	0.1 to 1 W/cm^2

The pressures used in etching processes are very low compared to fusion plasmas. Also, the ionized fraction of these plasmas are so low that electron-ion collisions are negligible. The electron energy is much higher than the ion and neutral energies. This is because the mass of the electron is many orders of magnitude less than that of other species in the plasma, allowing the time varying electric field to accelerate the electrons to a high energies. The high energy electrons transfer energy to the gas through excitation and ionization reactions, producing ions and radicals. The ions and neutrals are too massive to respond to the time varying field, staying at "low" temperatures. Thus the electrons produce "high" temperature gas phase interactions, while surface reactions involving radicals, occur at "low" temperatures. This plasma is a non-equilibrium plasma because the electron energy is much greater than that of the rest of the gas.

When power is applied between the electrodes, a time varying voltage profile is set up across the plasma. Figure 1.4 shows the voltage profile in an electropositive plasma, with a bulk glow region and a dark sheath region next to each electrode. The bulk of the plasma is quasi-

neutral, while the sheath has very few electrons. As shown in Figure 1.4, most of the applied voltage drops across the sheath. This sheath electric field accelerates the ions toward the surface and the electrons toward the bulk of the plasma. Although the sheath field varies with time at the applied frequency, 13.56 MHz, the ions only "see" the time averaged field that accelerates them toward the surface. This energetic ion flux makes directional etching on the wafer possible.

1.2 MOTIVATION

Control of the etching direction for accurate transfer of the mask pattern onto the film is especially important when etching high aspect ratio features. Some of the problems encountered when etching high aspect ratio features are shown in Figure 1.5. For etching these deep trenches, ion induced etching, whether by physical sputtering or by assisting in surface reactions, is crucial to the production of anisotropic etching profiles. The location of the ion bombardment on the surface will determine where the etching of the film takes place. Competition between chemical etching, which generally occurs at equal rates in both the vertical and horizontal directions, and ion induced etching will determine whether etching beneath the mask occurs. Ions glancing off the walls of the feature can cause bowing to occur or create uneven surfaces at the bottom of features. "RIE (reactive ion etching)" lag occurs when features with larger width etch faster than those with smaller widths. If the flux of ions are not all perpendicular to the surface, then the larger opening features collect more ions at the bottom, enabling etching to proceed at a faster rate. These are some instances where the etching of high aspect ratio features depends greatly on the energy and angle of the bombarding ions.

The ability to control the ion bombardment properties require knowledge of the effect of operating parameters on the energy and angle of the ions striking the electrode where wafers are placed during etching. For example, as plasma pressure increases, the density, and therefore the

flux, of reactive species to the surface increase, increasing etching rate. At the same time, the ion flux becomes less directed perpendicular to the surface because the number of ion collisions with gas particles in the sheath increases, randomizing the ion direction. Therefore, there is a tradeoff between etching rate and directionality of etching if all the other operating parameters are kept constant while changing pressure. Understanding the magnitude of the effect of pressure on the plasma properties is crucial to controlling etching. Other plasma operating variables which can potentially affect etching results are power, type of gas, magnetic field strength, and reactor configuration. To link these operating variables to the etching results require several experimental and theoretical steps, leading to the objective of this thesis.

1.3 OBJECTIVE AND METHODOLOGY

The goal of this research is to link operating parameters to plasma properties, *i.e.* find scaling groups or relationships for plasma properties such as power deposition in the plasma and ion bombardment.

Before starting the experimental work, a plasma reactor with easily adjustable operating parameters is designed and constructed. An important requirement for the apparatus is that the plasma volume must be well defined. The research reactor also has several ports and a sampling chamber for measuring plasma properties. Various associated diagnostic tools are designed to study the plasma.

Figure 1.6 delineates the methodology for accomplishing the goal of this thesis. The operating frequency for all the experiments is set at 13.56 MHz because most commercially available processing plasma reactors use power at this frequency. A symmetric electrode configuration is chosen, that is, the powered and grounded electrode have equal area and are parallel to each other. The gas used in most of the studies is argon. Argon plasmas have very

simple chemistry in that there are only electrons, argon ions, and argon neutrals. Argon plasmas are electropositive plasmas, that is, positive ions dominate over the negative ones. Although argon plasmas are not used very often in microelectronics fabrication, the simplicity of this plasma allows development of basic plasma physics and scaling parameters. Plasmas with greater complexity can use these scaling laws with whatever modifications are necessary. SF_6 plasmas are also studied to see how well the basic scaling parameters developed for argon plasmas apply to more complex systems. The SF_6 plasma is electronegative because a large fraction of the negatively charged species in the plasma are negative ions, rather than electrons.

The experimental work is divided into two groups. The Power Study, as seen in Figure 1.6, links power, pressure, electrode spacing, and electrode area to the measured voltage and current across the electrodes, and the phase between the voltage and current waveforms. The second part of the experimental work, called the Ion Study, links the voltage and current to the ion energy and ion flux striking the electrode surface. Profile simulators (Oldham *et al.* 1980, Smith *et al.* 1986, Yamamoto *et al.* 1987, Economou & Alkire 1988, Ulacia & McVittie 1989, Shaqfeh & Jurgensen 1989, Dalton *et al.* 1992, Jackson & Dalton 1989), which are not part of this thesis work, can then use surface reaction rates and ion bombardment properties to predict etching rates and etching profiles.

In addition to these two major sets of experiments, the spatial and temporal plasma optical emission are also measured to find how the sheath width and plasma ionization location change as a function of pressure and power. Other experiments include applying magnetic fields to the plasma and determining how plasma properties change. The magnetic field decreases electron loss to surfaces, thereby increasing the plasma electron, ion, and radical densities, and decreasing the sheath voltage. The magnetic field strength at various pressures is correlated to power deposition, ion flux, ion energy, and sheath width.

Monte Carlo simulations of the ion kinetics in the sheaths are used to provide a more detailed understanding of the measured ion bombardment energy and angle. These simulations are compared to experimental measurements to see how well they predict properties of real systems. As energetic ions collide with neutrals, they transfer energy to the neutrals. Therefore, energetic neutrals can be a significant fraction of the particles bombarding the surface. These energetic neutrals cannot be easily measured, but Monte Carlo simulations can keep track of the number, energy, and angle of these energetic neutrals.

Throughout this thesis, whenever possible, results are compared to those published in papers or to predictive models. This is a useful confirmation of the accuracy of the measurements, as well as a check for the validity of model assumptions.

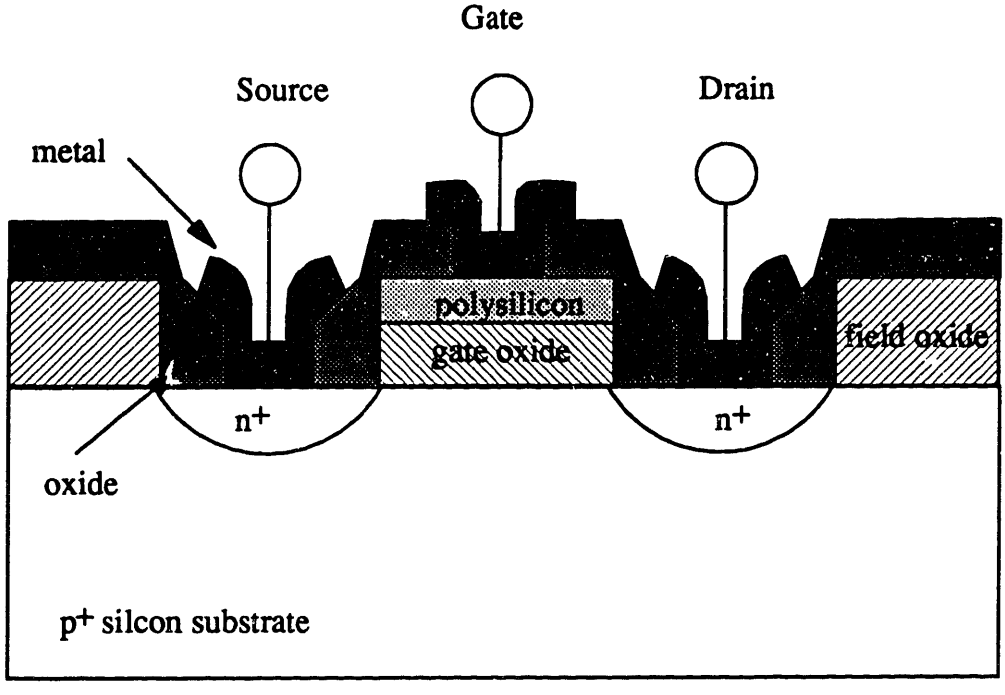


Figure 1.1: An n-p-n metal oxide semiconductor field effect transistor (MOSFET)

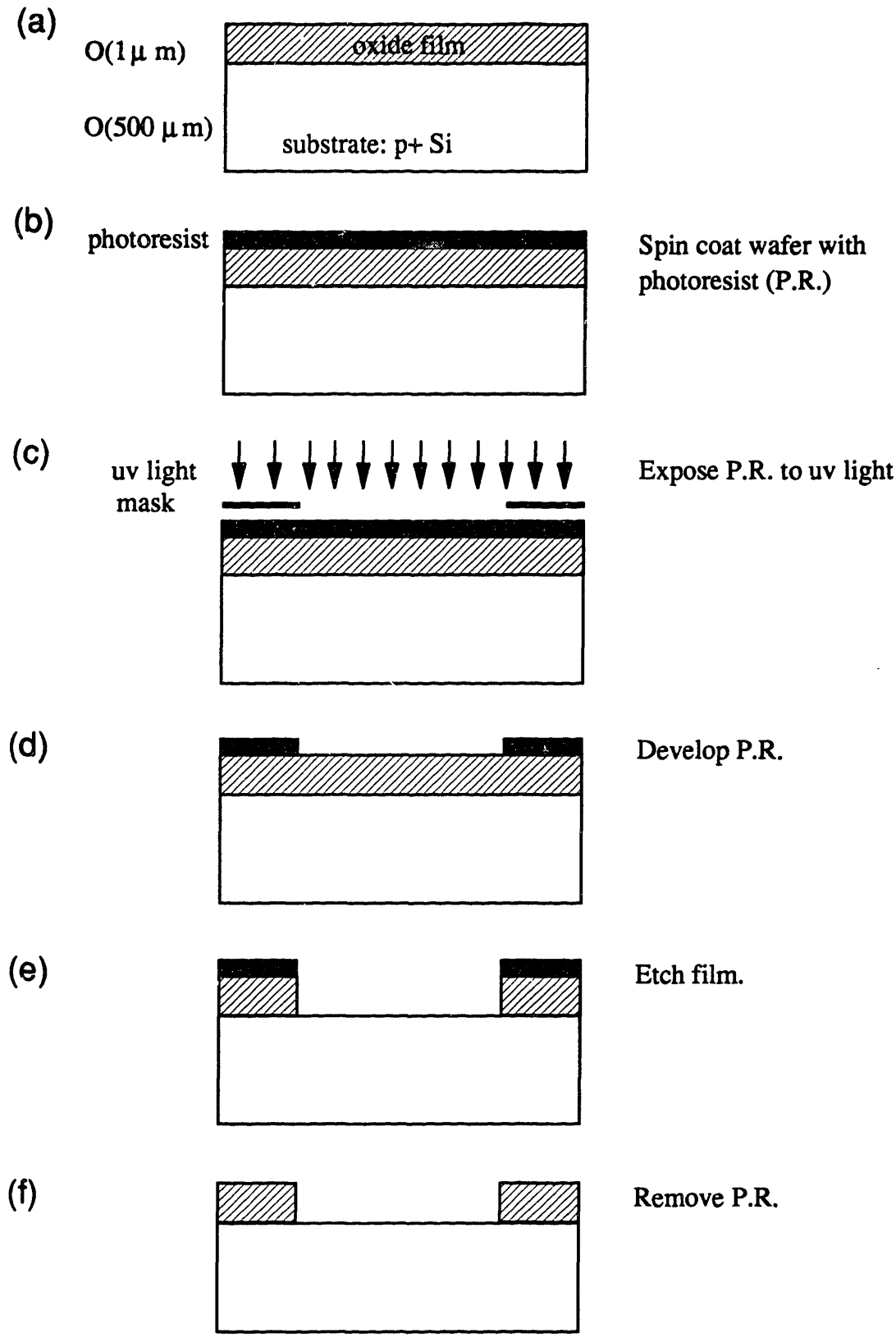


Figure 1.2: Steps for making a patterned oxide film.

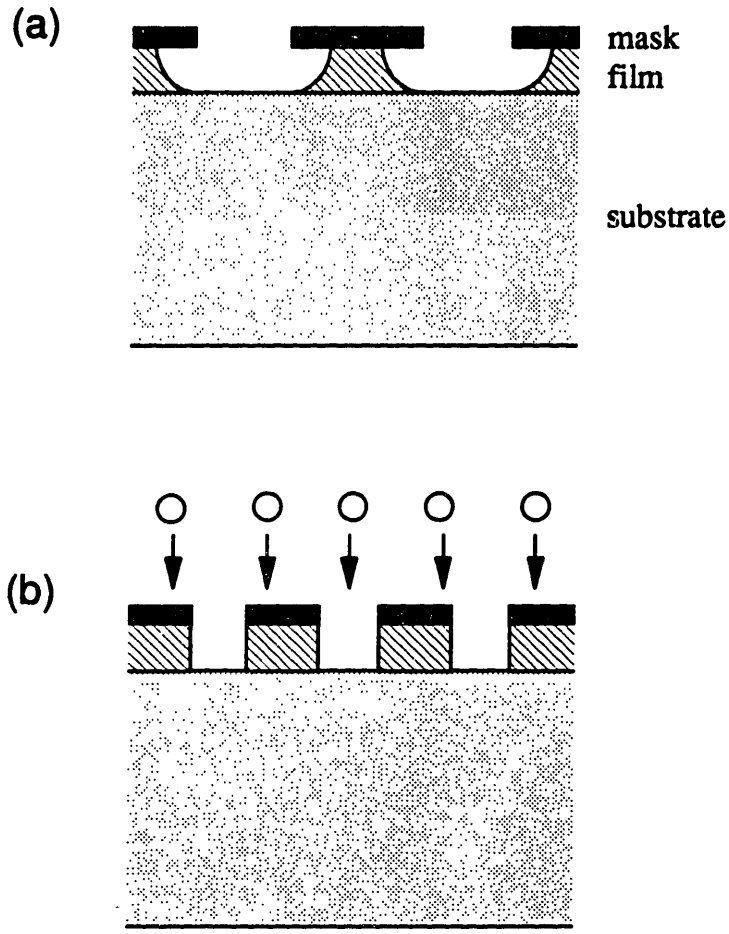


Figure 1.3: Etching profiles: (a) isotropic (b) anisotropic

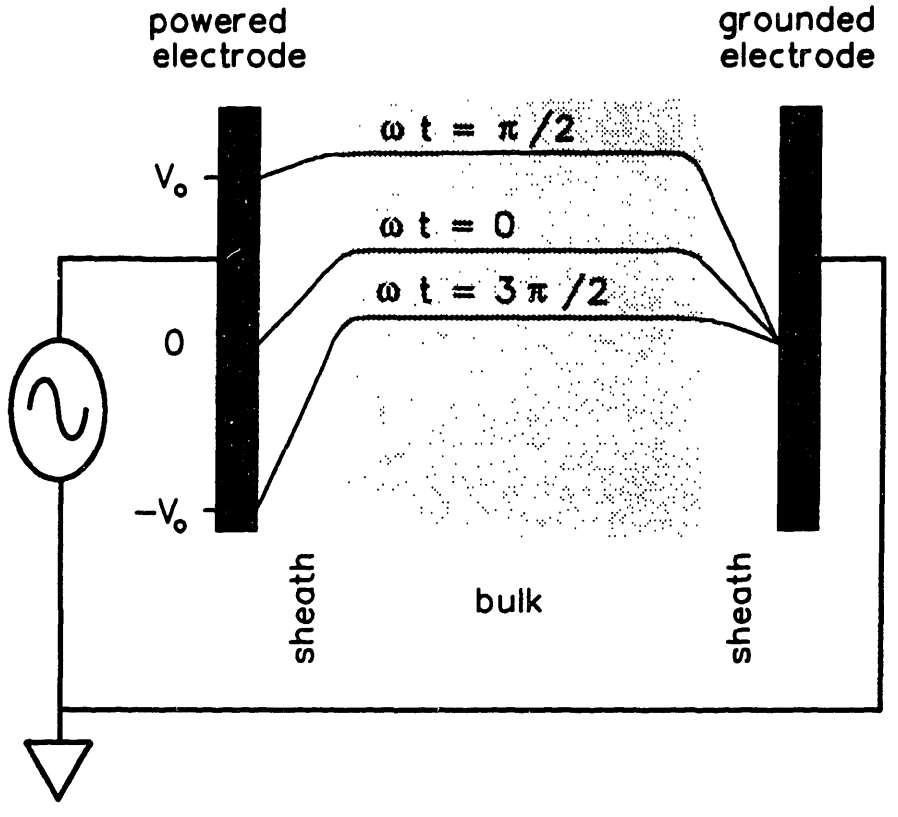


Figure 1.4: Voltage profile between the electrodes at various times in the rf period.

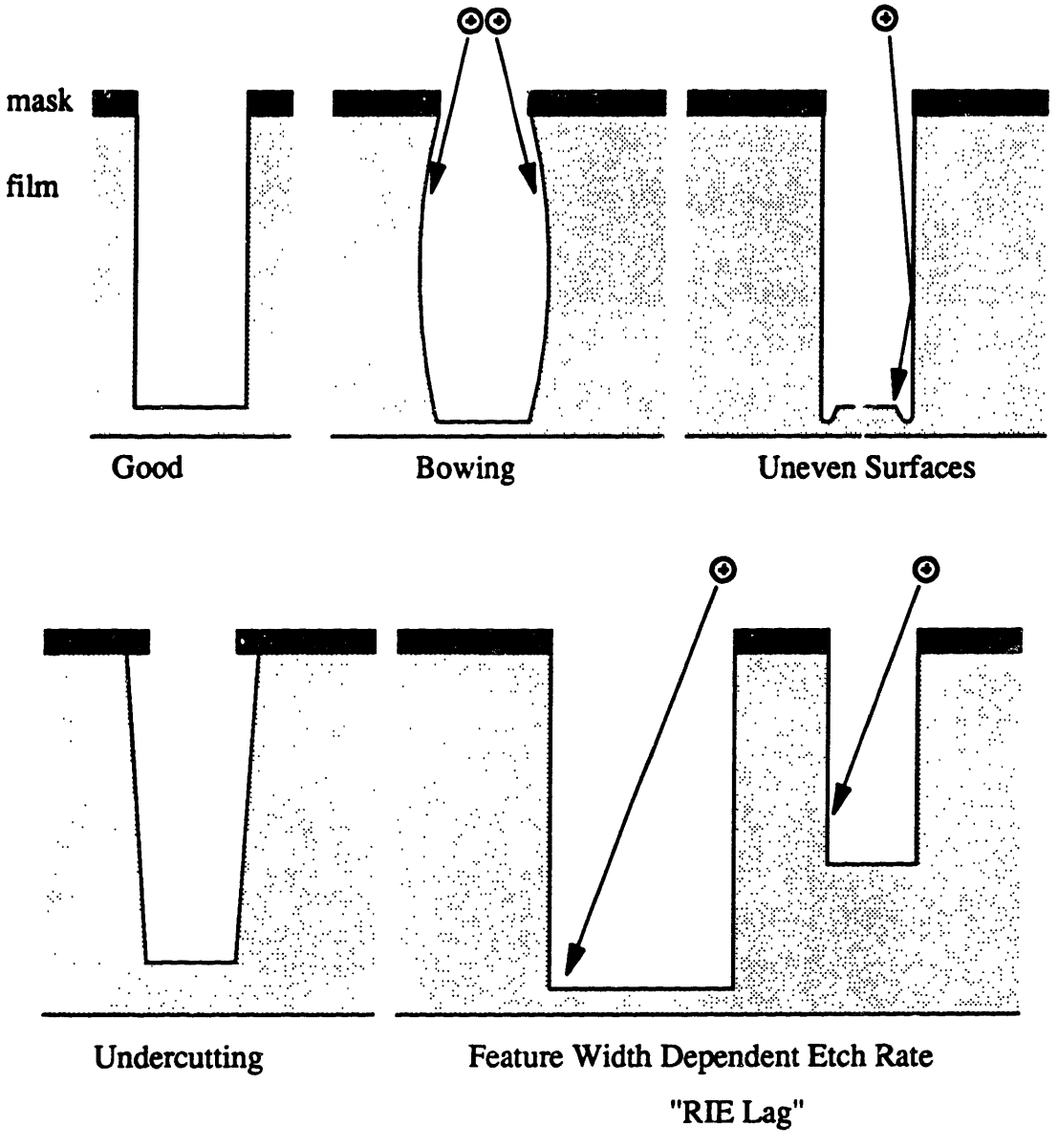


Figure 1.5: Some problems encountered in etching high aspect ratio features.

METHOD OF APPROACH

Use Argon gas
Fixed frequency at 13.56 MHz
Symmetric electrode configuration
Experiments:

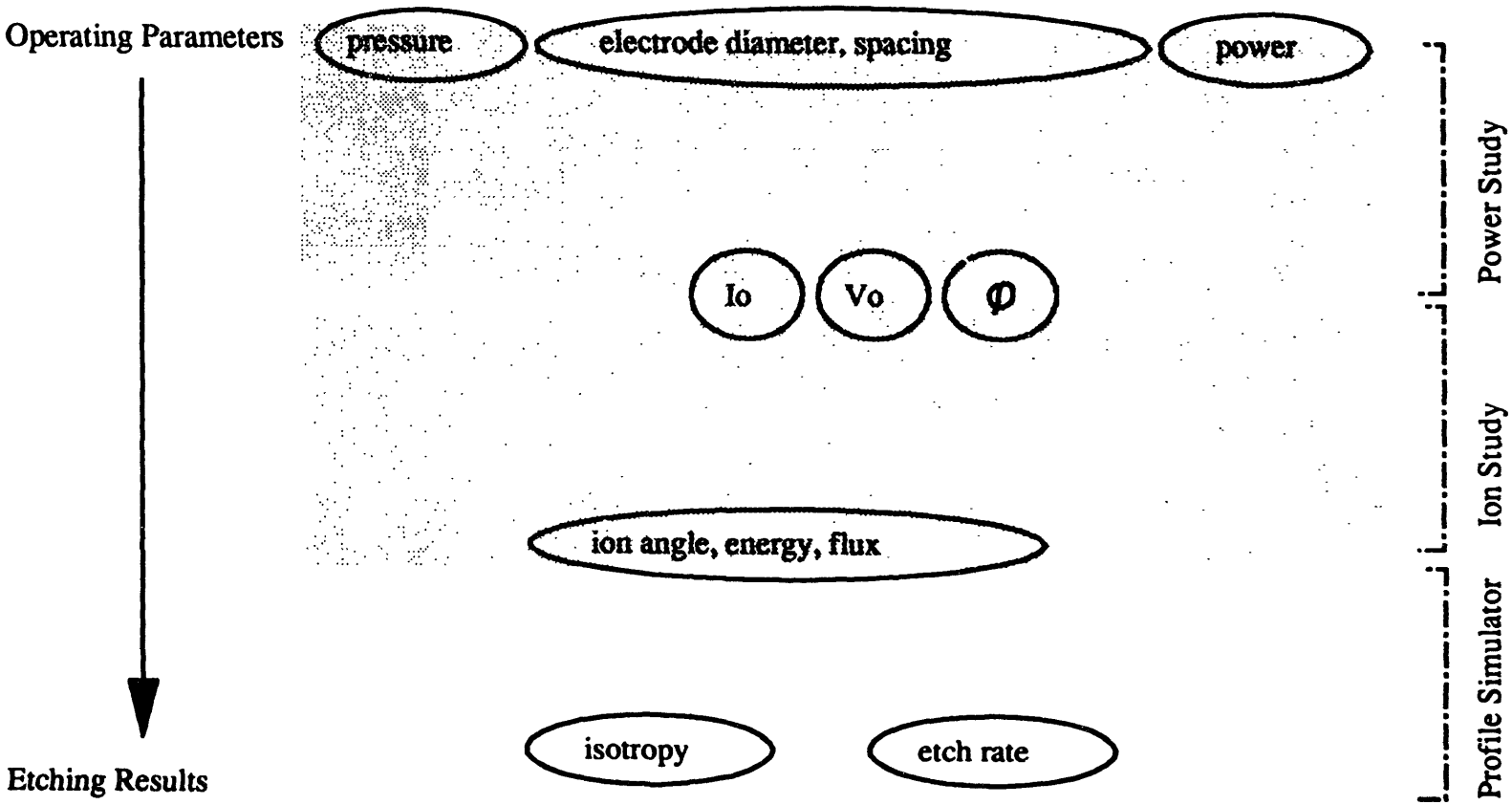


Figure 1.6: Method of approach to link operating parameters to etching results.

CHAPTER 2

EXPERIMENTAL APPARATUS AND DIAGNOSTICS

To accomplish the goals set forth in this thesis, a reactor is designed and constructed so that plasma operating parameters can easily be varied and several plasma properties can be monitored. Appendices A through E contain the fabrication specifications and machine drawings for the plasma reactor and diagnostics, and Appendix F contains a list of the electronic equipment used to run the experiments.

2.1 SCALING REACTOR

Figure 2.1 schematically illustrates the reactor used in the experiments. The reactor has pyrex walls with four windows placed 90° apart for optical diagnostics. The plasma is confined between two parallel plate electrodes and a 1/32" thick teflon cylinder. The teflon cylinder has holes cut out for the four MgF_2 windows which are placed flush to the surface of the teflon. The spacing between the electrodes can be varied by moving the powered electrode up and down using a stepping motor. All the electrodes are water cooled and made of anodized aluminum. The electrode size can be varied using one of three different pairs of electrodes with diameters of 7.6 cm, 11.4 cm, and 15.2 cm and their corresponding chambers with diameters of 9 cm, 13 cm, and 17 cm, respectively. Figure 2.1 shows a symmetric electrode configuration using 7.6 cm diameter electrodes. Asymmetric systems can also be assembled simply with different diameter upper and lower electrodes and using a dielectric, *e.g.* teflon, to make up the surface area difference between the smaller electrode and the wall. Another possible asymmetric electrode configuration is the use of metal walls as part of either the powered or grounded electrode. Figure 2.2 shows two of the possible asymmetric electrode configurations.

In almost all the experiments, the gas used is 99.999 % pure argon. In a few cases, SF_6 gas is also used. Gas enters the plasma region, generally between 5 and 15 sccm, through a shower head arrangement in the upper powered electrode and leaves through eleven 0.25" diameter holes at the bottom perimeter of the pyrex chamber.

13.56 MHz power is capacitively coupled to the plasma through a Π-configuration matching network and a 2000 pF blocking capacitor. The matching network consists of two variable capacitors and one variable inductor. The matching network is adjusted so that the power supply "sees" a resistive load in the matching network and plasma combined. The power supply. besides supplying power to the plasma, also supplies power to a powered shield located outside of the pyrex chamber. Figure 2.1 shows the placement of the power shield with respect to the electrodes and the plasma. This power shield prevents stray plasma from igniting above the powered electrode and eliminates capacitive coupling between the powered electrode and the grounded surfaces around the reactor. A Tektronix P6015 voltage probe and an Ion Physics CM-100-M current probe measure the voltage and current waveforms in the power line approximately one foot away from the powered electrode. The signals from the probes are fed into a LeCroy 9400 dual 125 MHz digital oscilloscope. Appendix H contains the calibration procedure for the voltage probe. Since the voltage and current are not measured directly at the powered electrode, there are stray impedance losses between the measuring point and the powered electrode as indicated in Figure 2.3. To calculate the power deposited in the plasma from the measured voltage and current, a stray impedance de-embedding technique described by Butterbaugh *et al.* (1990) is used. This technique involves placing cells of known impedance between the electrodes and turning on the power without striking a plasma. (The impedance cell design and values are found in Appendix G.) Recording the measured voltage and current for each of the impedance cells, the stray impedance between the measuring point and the power electrode can be de-

embedded from the network. Once the stray impedances are known, the plasma impedance and the power deposited into the plasma can be calculated by "subtracting" the stray impedances from the measured impedance.

Magnetic field lines parallel to the electrode surface are supplied to the plasma using a set of four solenoids placed outside of the vacuum chamber. Magnetic fields are used industrially in plasma processing because they confine the plasma and increases the plasma density. The magnetic field strength to the plasma can be adjusted by varying the current through the coils.

Several plasma properties are measured besides the power deposited into the plasma. The ion flux, energy, and angle striking the electrode are important properties to measure as a function of operating condition since they determine the ion induced etch rate and isotropy in etch features on the wafer. Ions striking the surface are sampled through a small orifice located at the center of the grounded electrode. The ions pass through the orifice and their properties are measured with an ion analyzer. Another measurable plasma property is the time averaged and time resolved plasma emission. The reactor is built on a x-y-z movable stage. Therefore, besides temporal plasma emission, spatial emission between the electrodes can also be measured.

All the data are collected using a PCs Limited XT via an IOtech IEEE 488 and a Data Translation DT2811 Analog and Digital I/O computer interface. A Pascal program called ACQ.PAS, listed in Appendix L, is used for data acquisition of power, ion properties, and plasma emission. For measuring power deposition in the plasma, the computer acquires voltage and current waveforms from the oscilloscope through the IEEE 488 interface. The computer controls the voltage on the ion analyzer grid and three stepping motors used to move the reactor in x, y, and z directions, through the digital to analog I/O interface. The ion current from the ion analyzer and from the photo multiplier are measured with a picoammeter which has a IEEE 488 interface with the computer. All the data are stored and later analyzed using MASSAGE.PAS, another

Pascal program listed in Appendix L.

2.2 GAS FLOW SYSTEM

Gas flows into the plasma via twenty-four evenly distributed, 0.015" diameter holes in the upper electrode. Figure 2.4 shows the shower head arrangement for gas flow into the reactor. The distribution of small diameter holes guarantees that the pressure above the holes is uniform so that the gas flows equally through each of the holes. The majority of the gas is pumped out with a CTI-8 cryogenics pump through eleven 0.25" diameter holes at the bottom perimeter of the pyrex chamber. Some gas flows through a small orifice on the grounded electrode to a lower chamber which is pumped down to 10^{-5} Torr with a Balzers TPU050 turbomolecular pump.

Figure 2.5 shows the vacuum system with the associated pumps. The pressure in the upper chamber, where the plasma resides, is regulated using a pressure feedback control loop. An MKS 220CA capacitance manometer measures the pressure and a Vacuum General 80-2 controller varies the gas flow conductance between the chamber and the pump by changing the flow diameter through a Vacuum General MDV-015S04 motor driven throttle valve.

Since there is a teflon cylinder surrounding the plasma, the plasma pressure is higher than that measured by the capacitance manometer which is located outside the teflon cylinder. Figure 2.6 shows the placement of the capacitance manometer relative to the plasma and the setup used to calibrate the plasma pressure. To correlate the plasma pressure to the measured pressure, a second capacitance manometer is temporarily placed in the chamber beneath the grounded electrode where the ion analyzer usually sits. No gas is pumped out through this chamber. An opening with 1/8" diameter separates the plasma chamber from the lower chamber. This larger opening is used, rather than the small orifices used to sample ions from the plasma, because the pressure recorded by this second capacitance manometer need to quickly equilibrate to the pressure

above the grounded electrode. The pressure from both capacitance manometers are recorded for several argon gas flow rates. From this data, the gas flow conductance between the plasma and pressure measuring point (capacitance manometer #1) can be calculated as a function of gas flow rate. The pumping conductance is calculated for all three diameters of pyrex chambers. The largest pressure drop occurs in the smallest pyrex chamber at high gas flow rates. Appendix J shows how this calculation is done and it also includes calibrations for the reactor volume and the gas flow meter readings.

2.3 MAGNETIC FIELD

The magnetic field strength applied to the plasma never exceeds 200 Gauss. At 200 Gauss, the ion Larmor radius is larger than the electrode dimensions and much larger than the ion mean free path. This means that the ion trajectory in the plasma is unaffected by the presence of magnetic field. On the other hand, the electron mass is approximately five orders of magnitude lighter than the ion mass. At 200 Gauss, the electron Larmor radius is approximately the same order of magnitude as the mean free path at 0.5 Torr, and much smaller at lower pressures. The net result of the addition of the magnetic field parallel to the electrode surface is that the electrons' paths wind around the field lines, decreasing their mobility to the electrode surface. Since the loss of electrons out of the plasma decreases, the plasma density increases, increasing ion flux to the surface.

Magnetic fields are created using four solenoids schematically illustrated in Figure 2.7. The dimensions of these coils are designed using SOLDESIGN, a FORTRAN program written by R. D. Pillsbury (1989). The arrows in the figure indicate the direction of current flow through the copper windings of the solenoids. The two large solenoids have 56 turns and the two smaller solenoids have 48 turns. Appendix B contains the design drawings for these solenoids. The

windings are made from hollow copper tubing. Cooling water runs through the center of the tubing to keep the resistivity of the copper from rising. The maximum expected voltage drop through all four coils combined, assuming the copper remains at 25 °C, is around 30 V. SOLDESIGN calculates the magnetic field at various locations, showing that for the same current running through all four coils, the field is very uniform across the plasma region. Figure 2.8 shows the field lines created by the four solenoids. Over a distance of 8 cm from the center, the field strength only differs by 5.5%. The field strength is proportional to the current through the coils or

$$B\ (\text{Gauss}) = \frac{2}{3}\ \text{I}\ (\text{Amp})\ . \tag{2.1}$$

By electrically connecting all four solenoids in series, only one current supply is needed to create spatially uniform magnetic field strengths up to 200 Gauss. The field strength only varies by 5.7 % from the center to 8 cm away in the axial direction. Measurements of the field strength and uniformity using a Gauss meter have been done after installation of the coils, verifying the SOLDESIGN calculations.

Several water cooling lines for the solenoids are needed. The copper tubing for each of the large coils are 69 m long, with an inner diameter of 0.0057 m for water flow. The tubing for each of the small coils are 24.4 m long, with an inner diameter of 0.004 m. For a large enough water flow rate to keep the coils from overheating, each coil is separated into two windings that are still electrically joined, but have separate water inlets and outlets to keep a reasonable water pressure drop through each winding. This results in a total of eight cooling water sources for all four solenoids. Table 2.1 shows that for an allowed water temperature rise, ΔT, of 25 °C, 300 Amp current through the coils, and a copper resistivity value at 25 °C, the following power dissipation (**P**), required water flow rate ($\dot{\mathbf{F}}_{H2O}$), and water pressure drops ($\Delta \mathbf{p}$) apply.

Table 2.1: Power dissipation and water pressure drop for both the large and small solenoids, each divided into an outer and inner shell for water cooling.

	winding length (m)	R (Ω)	Power (W)	$\dot{F}_{H2O}=P/C_p\Delta T$ (cm^3/sec)	tube diameter (m)	Δp (psi)
Large Solenoid outer shell	37.8	0.019	1640	15.7	0.0057	21.5
Large Solenoid inner shell	31.3	0.015	1360	13.0	0.0057	12.2
Small Solenoid outer shell	13.3	0.012	1050	10.1	0.0040	18.9
Small Solenoid inner shell	11.1	0.010	880	8.4	0.0040	11.1

2.4 ION ANALYZER

The ions striking the surface are sampled through orifices located at the center of the lower electrode. Once the ion passes through the orifice, its incident energy and angle are determined using the ion analyzer represented by Figure 2.9. The analyzer is located directly beneath the grounded electrode in a chamber that is differentially pumped to 10^{-5} Torr. At this pressure, the ion mean free path is much larger than the analyzer dimensions. The analyzer consists of three grids made of concentric semi-spherical wire mesh and a semi-spherical ion collecting plate. The collecting plate has nine electrically isolated annular metal rings machined on it, each ring is ≈ 4.5° wide. With all nine rings combined, all ions with incident angles ≤ 39.5° measured from the electrode surface normal will fall on the collecting plate. The grid mesh consists of 25 μm diameter stainless steel wires woven in a 50 X 50 per square inch pattern. The first and third grids have single mesh layers, resulting in a 90% ion transmission. The second grid is constructed from two mesh layers, separated by 0.4 mm, resulting in an 81% ion transmission. The double mesh layers reduce electric field penetration between the mesh wires, greatly

increasing the ion energy resolution (Huchital & Rigden 1972). The energy resolution in the analyzer is ≈ 1 - 2 eV, depending on the ion energy.

The top grid potential is biased at 0 V to create a field free region between the orifice on the grounded electrode and the analyzer, leaving the ion path unaltered after it passes through the orifice. The third grid is biased at a negative voltage, generally at -50 V, for two reasons. First, if secondary electrons are created when ions strike the collecting plate, they will be returned to the collecting plate, thus preventing a higher apparent ion current reading. Secondary electron emission was observed by Thompson *et al.* (1986) with an ion analyzer that did not have a negatively biased grid above the collecting plate. Second, the negative bias on the third grid prevents electrons from the plasma that pass through the orifice from reaching the collecting plate. For ion energy analysis, the second grid potential was raised from 0 V to 100 V, allowing only ions with energy greater than or equal to the second grid potential to pass through the grid to the collecting plate. The ion energy distribution (IED) can be calculated by differentiating the data taken by measuring the ion current as a function of the second grid potential. Figure 2.10 shows an example of the measured data and the corresponding IED. Greene *et al.* (1988b) used a band pass ion energy analyzer, rather than the high pass energy filter used here, to get a better signal to noise ratio. However, with their analyzer, only ions incident near normal to the surface could be measured. To determine the ion angle distribution (IAD), we set the second grid at 0 V, allowing all ions to pass through to the collecting plate. Depending on its incident angle, an ion strikes one of the nine annular rings on the collecting plate. A Keithley 485 picoammeter is used to measure the current at each annular ring.

The finite thickness of the ion sampling hole necessitates correction of the measured IAD. The opening that an ion "sees" when incident normal to the orifice plane is greater than that of an ion approaching at an off normal angle. The shaded area in Figure 2.11 illustrates the effective

ion collection area for an ion with non-zero incident angle. Appendix I.1 shows the calculations used for this correction. These area corrections vary from 0% at normal incidence to 11% at 37° for a 75 μm diameter and 9 μm thick orifice. All the IADs presented here have been corrected for this effect.

A small diameter orifice is necessary to minimize disturbances to the neutral density and electric field around the orifice. Based on the work of Coburn and Kay (1971), ions undergo collisions downstream of the orifice that separates the plasma from the low pressure ion analysis chamber. In our reactor, using a 75 μm diameter orifice to sample argon ions from a 0.05 Torr plasma, the high neutral density around the downstream side of the orifice caused 1% of the ions to undergo collisions before striking the collecting plate. As pressure is increased to 0.5 Torr, 16% of the ions are scattered. Appendix I.2 shows the details of these calculations. In a 0.5 Torr argon plasma, although 16% of the ions undergo collisions, a large qualitative effect on the shape of the IADs will not be seen since there were also many collisions in the sheath. The measured IEDs, however, are shifted toward lower energies. As Thompson *et al.* (1986) pointed out, this is somewhat mitigated by the reduction of neutral density immediately above the orifice. Greene *et al.* (1988b) attributed the broad structure they saw in the IED when comparing the effect of two orifice diameters, 50 μm versus 200 μm, to collisions after passing through the orifice. However, since they were collecting only ions with near surface normal incident angle, if there were elastic collisions, the ions would be scattered away from the detector. If there were charge-exchange collisions, the new ion created in the field free region would rarely have enough energy to reach the detector. Ingram and Braithwaite (1988) avoided the density change around the orifice in a differentially pumped system by using an ion analyzer with dimensions (0.75 mm) less than a collision mean free path.

Space-charge broadening of the ion beam passing through the orifice under the conditions

used in these experiments are negligible. The low ion flux and small orifice diameters limit the beam broadening; see Appendix I.3 for these calculations.

The electric field distortion caused by the presence of the ion sampling orifice can alter ion trajectories. Ions with energies greater than 5 eV are deflected less than the angular resolution of our analyzer (4.5°). Ions approaching the orifice with low energies, less than 5 eV, can experience a deflection large enough to appreciably distort IAD measurements. The deflection of the ion trajectory as it passes through the orifice increases with the sheath electric field and the orifice diameter. Ion trajectories and field distortions around the ion sampling orifice, predicted using D. A. Dahl's (1985) program, SIMION, are discussed in Appendix I.4.

To ensure that there was negligible deflection of the ion trajectory as the ion passes through the orifice, we used a Kratos Minibeam I ion gun was used to check the ion beam width both before and after all the ion data were taken. Approximately 180 eV ions were directed at right angles (0°) to the electrode surface, toward the orifice array and the ion analyzer. The IAD was measured at several second grid potentials on the ion analyzer. All the ion current was measured at the center ring on the ion collecting plate when the ion energy was greater than the second grid potential by 2 - 3 eV. For ions with energies less than that, approximately 10% of the ions were deflected to the next outer ring, *i.e.* 10% were deflected by about 4.5°.

The ion gun was also used to check the energy resolution of the ion analyzer. Two sets of data were taken: 1) the second grid was used as the ion energy filter and the ions which passed this grid were measured at the collecting plate, and 2) all the grids were grounded while the collecting plate was biased to a positive potential to repel ions with energy less than the potential on the collecting plate. The ion energy measured using the collecting plate to repel ions was taken as the absolute ion energy and used to check the energy measured using the second grid as an energy filter. This comparison showed that the resolution of the ion analyzer is 5% of the ion

energy.

Three orifice sizes were used to evaluate ion bombardment properties. The diameters of the orifices, 25 μm, 50 μm, and 75 μm are much smaller than the sheath thickness. Therefore, the effect of the orifice on the sheath electric field should be small. An example of IADs measured at 0.01 Torr is presented in Figure 2.12. The two sets of data taken with the same diameter orifice, 75μm, but different aspect ratios, 8.6:1 and 3:1, gave the same IAD. However, keeping the same aspect ratio of 3:1 but decreasing the orifice diameter from 75 μm to 25μm changed the IAD. Thompson *et al.* (1986) saw no difference between IEDs taken with a 25 and 50 μm diameter orifices and aspect ratio equal to two. The difference in IADs is presently not well understood. Calculations in Appendix I.4 indicate that they cannot be due to field distortion caused by the presence of the orifice. Uniform charging of the orifice wall also does not appear to be the cause. The observed effect of the orifice diameter may be a result of deposition of an insulating layer or growth of aluminum oxide on the walls, or other experimental artifacts. Figure 2.12 shows that the shift of the IAD measured using the 25 μm orifice from the other two IADs is not large, and therefore, the 25 μm orifice has not significantly altered the IADs. Unless otherwise specified, all IADs presented in this thesis have been measured with the 75 μm diameter, 9μm thick orifice.

The ion incident angle cannot be measured when a magnetic field is applied to the plasma because the field alters the ion trajectory after the ion passes through the ion sampling orifice. Figure 2.13 illustrates the path of a 5 eV ion and a 10 eV ion, with an initial 0° incident angle, as they deviate from a vertical path while traveling from the orifice to the ion collecting plate in a 200 Gauss magnetic field. The ions that experience the maximum deflection are those that have an initial 0° incident angle. To calculate the ion trajectories, the first and second grids of the ion analyzer are kept at 0 V, while the third grid is biased at -50 V. Details of the ion

trajectory calculation can be found in Appendix I.5. For ions with energies equal to or less than 5 eV and 0° initial incident angle, instead of striking the center ring on the collecting plate, they strike at a larger incident angle. Therefore, for plasmas with applied magnetic fields, the IAD cannot be measured directly. The total ion energy distribution can still be measured since most ions will still make it to the collecting plate even if their trajectories change. Since the ion angle depends primarily on the number of collisions it has before striking the surface, a method for estimating the IAD is by using the measured IAD from a plasma without magnetic field, but which has a similar average ion energy and matching number of mean free paths in the sheath. In this case, the ion flux would not be the same since the plasma density is lower without the magnetic field.

2.5 PLASMA EMISSION DETECTION

The plasma emission at various locations in between the electrodes is probed by focussing the optics at one point in space while the entire reactor and the plasma is moved up and down. The emission from the plasma must pass through a MgF_2 window, a S1-UV-B fused silica grade window, and finally through two lenses before entering into the Instruments SA, Inc. HR-640 monochromator. The Products for Research, Inc. PR-1405RF00ℓ photo multiplier converts the monochromatic light into a current signal. For a time averaged spatial or wavelength emission scan, this current is fed into the Keithley 485 picoammeter, and from there, transmitted to the computer. Figure 2.15 shows an example of an argon plasma spatial and wavelength scan. For temporal scans where the emission varies at 13.56 MHz, the signal from the photo multiplier and the signal from a Stanford Research Systems, Inc. (SRS) Model DG535 four channel digital delay/pulse generator are fed into the SRS Model SR250 gated integrator and boxcar averager. The signal is averaged over 15,000 data points for each time step. The gate width of this

detection system is 7 nsec.

The reactor is constructed with optical windows that allow, not only ease of monitoring of both the spatial and temporal plasma emission, but also for using laser induced fluorescence as an analytical technique. The flat windows can be replaced by Brewster windows, shown in Figure 2.14, that minimize loss of laser intensity by reducing light reflection. Though this option of studying the plasma is available, only monitoring of the plasma induced emission is presented in this thesis.

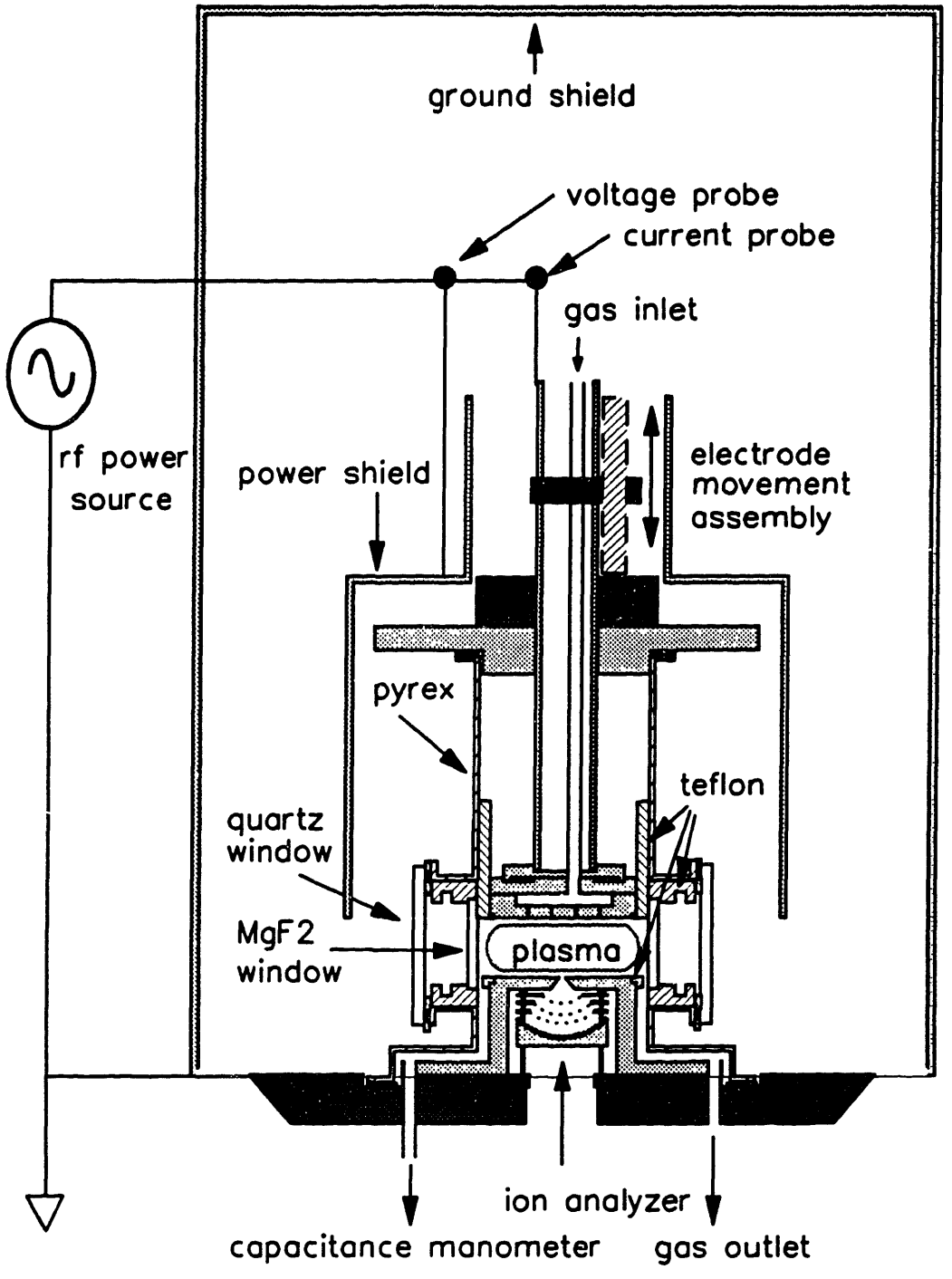


Figure 2.1: Reactor with 9 cm diameter pyrex chamber.

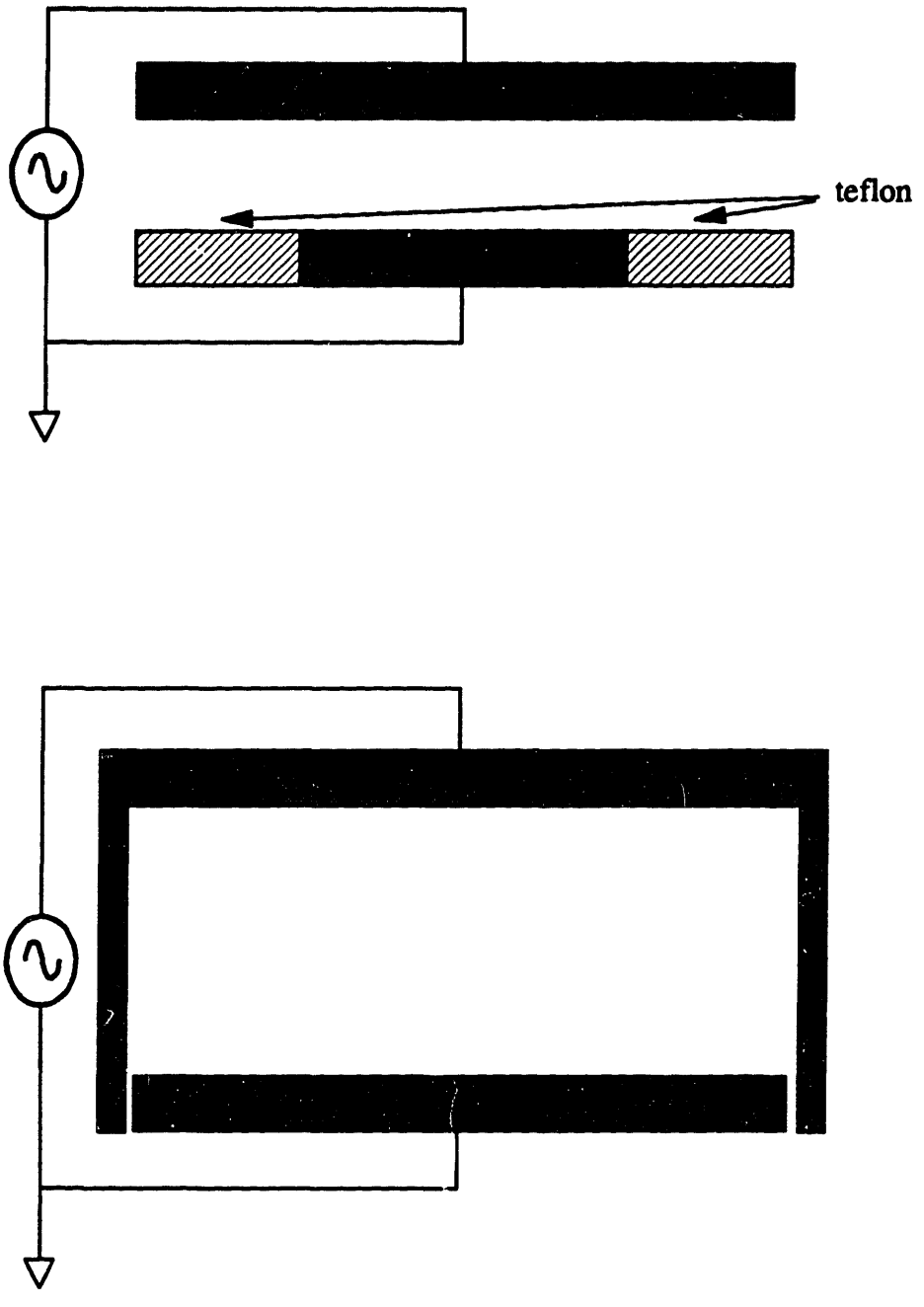


Figure 2.2: Various asymmetric electrode configurations.

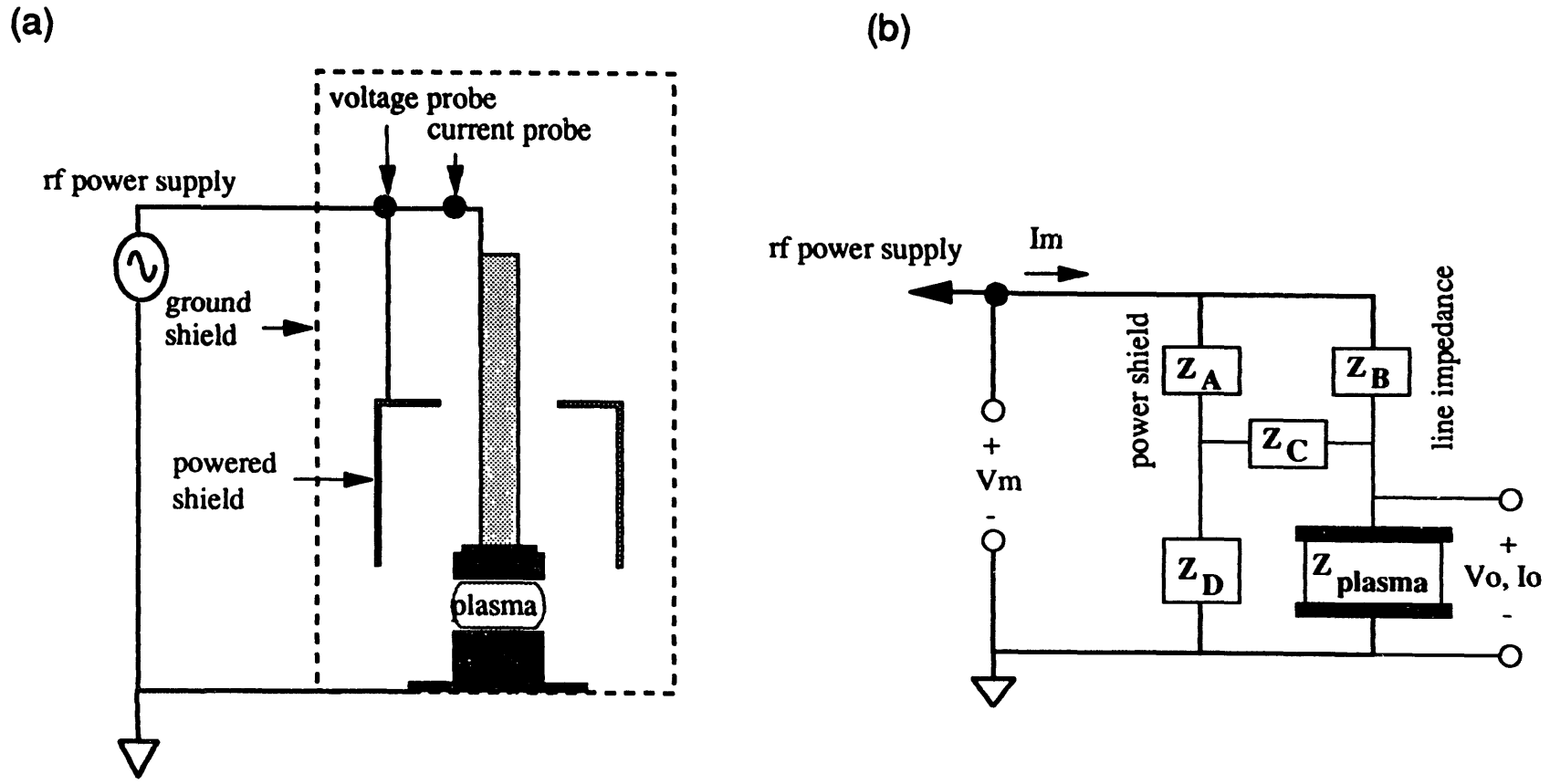


Figure 2.3: (a) Schematic of the plasma reactor showing the powered and grounded areas. (b) Electrical representation of the stray and plasma impedances between the measuring point and the powered electrode.

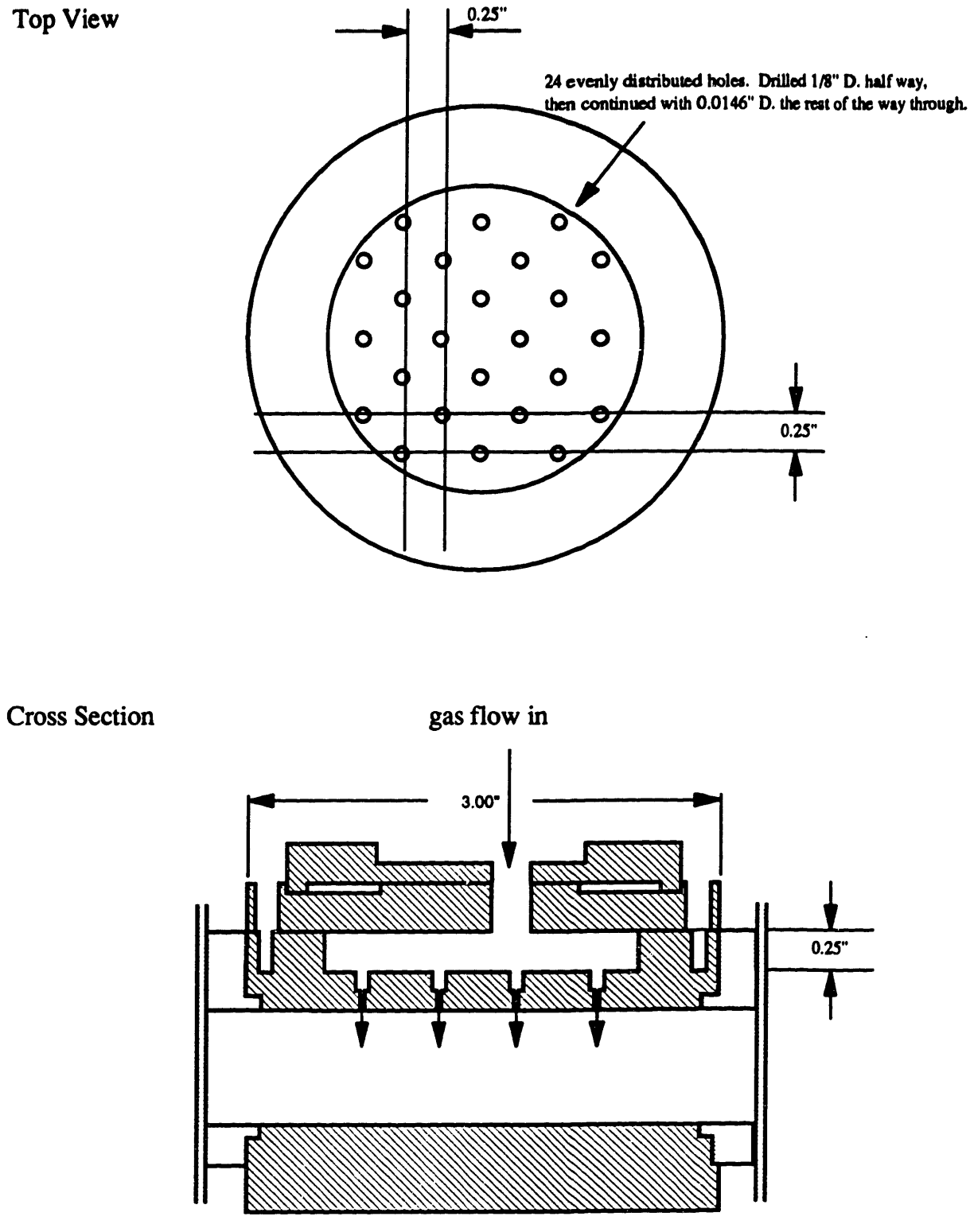


Figure 2.4: Shower head arrangement in powered electrode for gas flow.

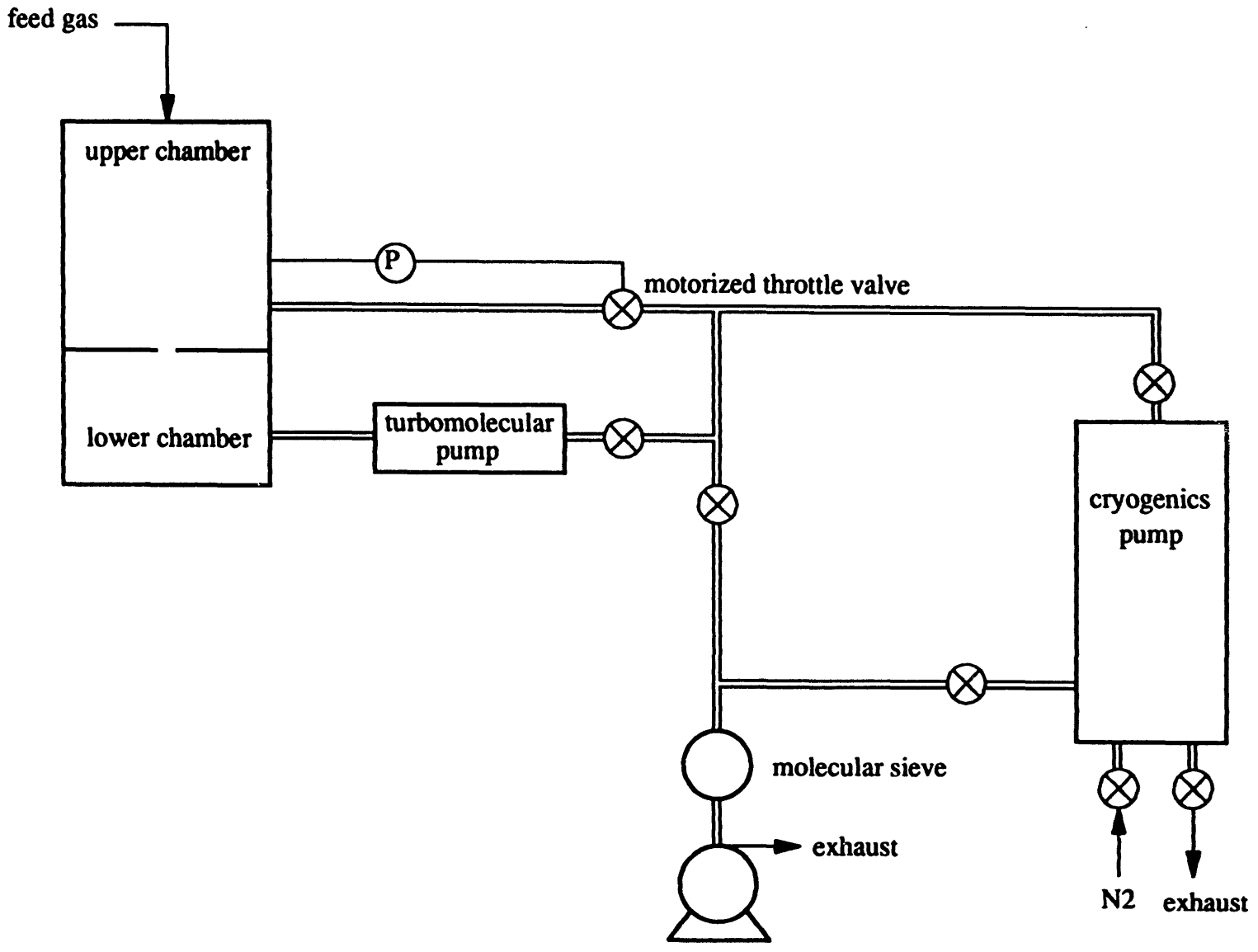


Figure 2.5: Vacuum pumping system.

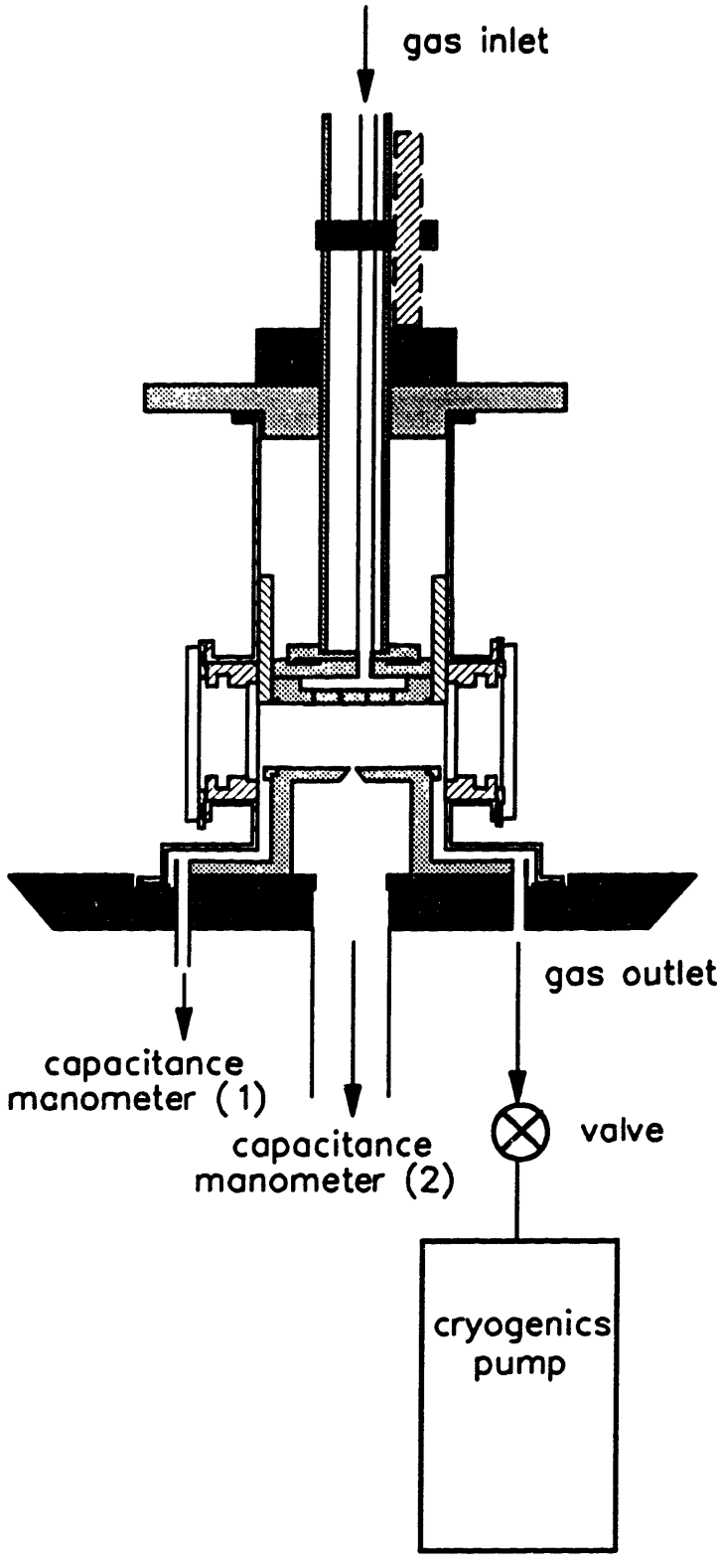


Figure 2.6: Reactor configuration for correlating plasma pressure to measured pressure.

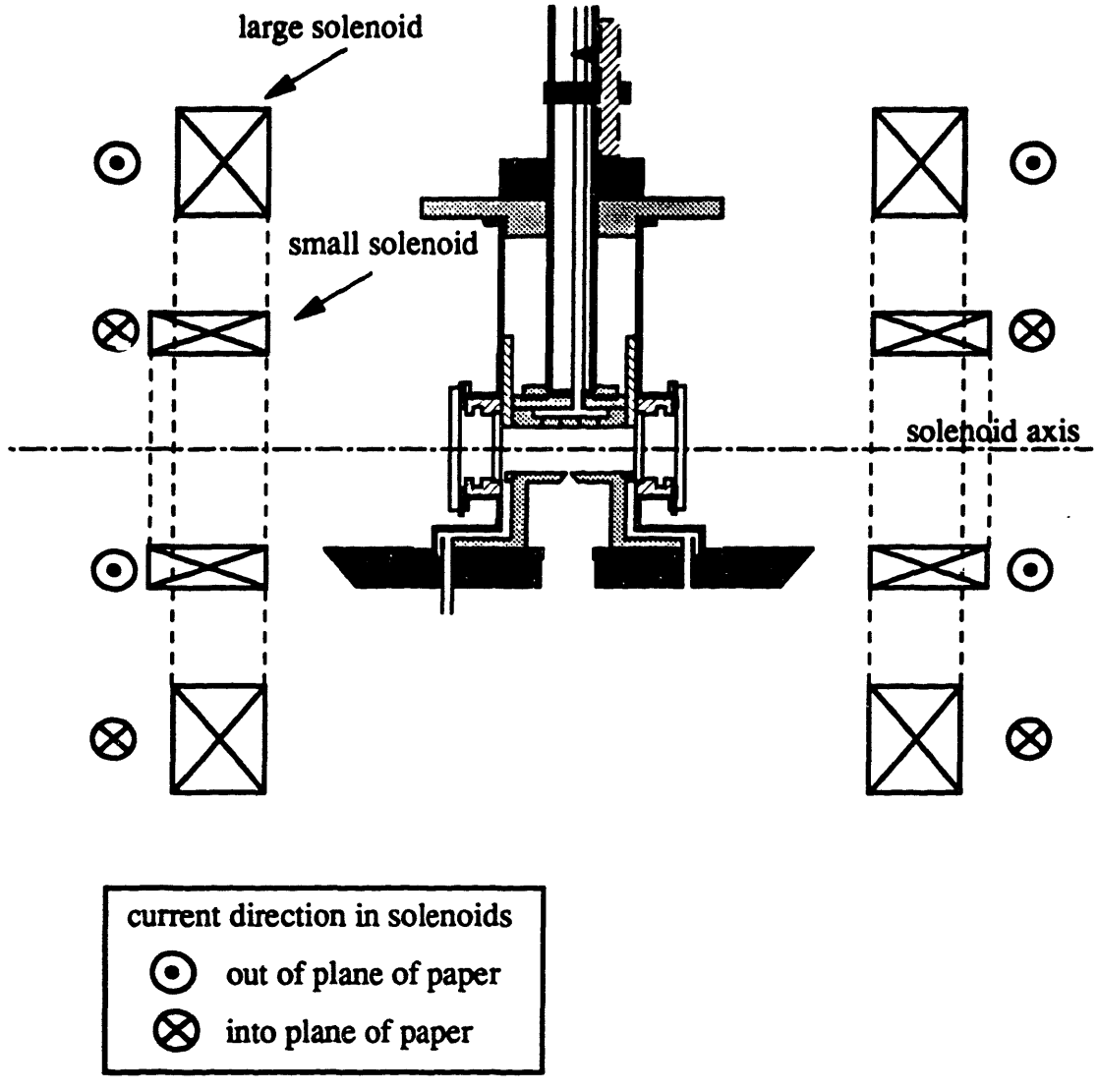


Figure 2.7: Placement of four solenoids with respect to the reactor.

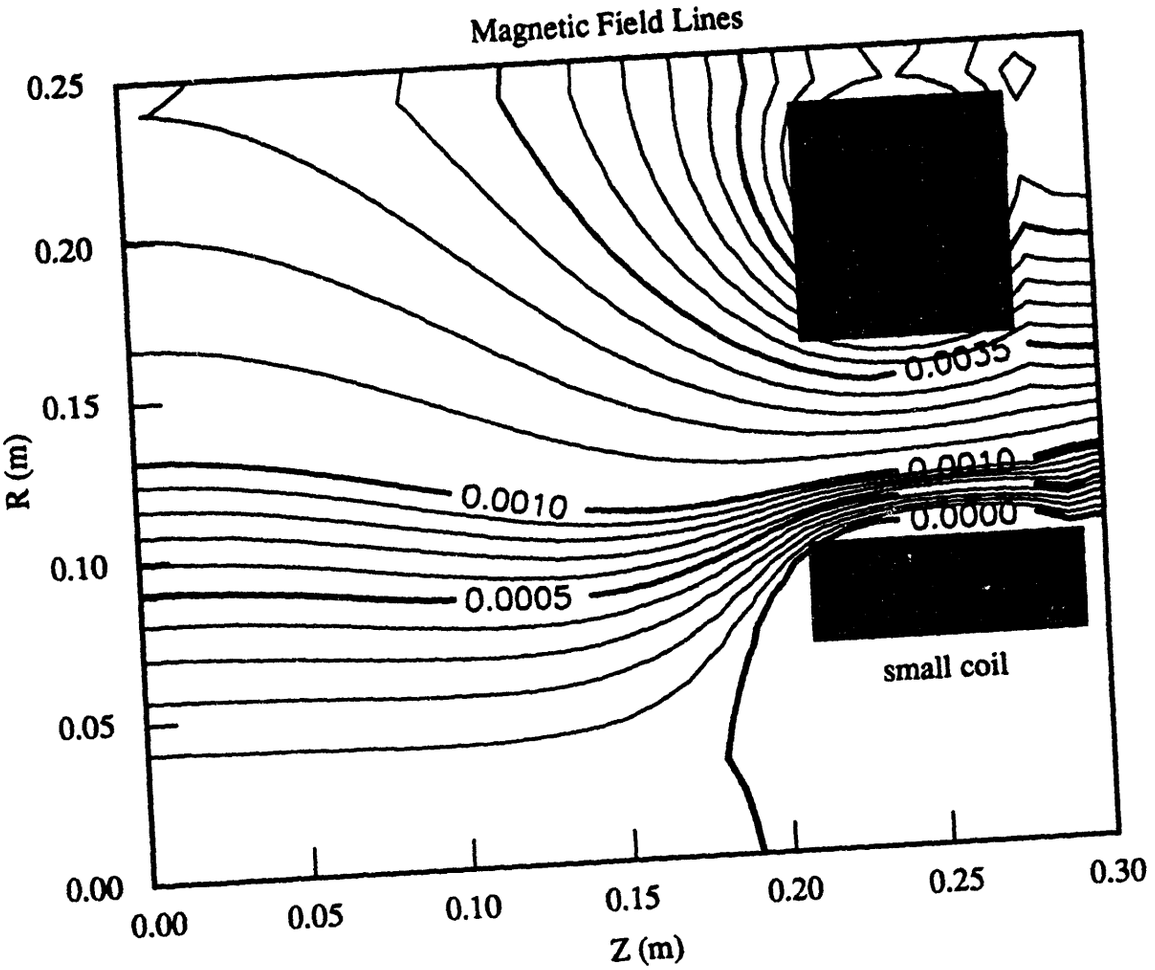


Figure 2.8: Field lines showing uniformity at the center of the solenoid arrangement. The Z-direction corresponds to the axis of all four solenoids and the R-direction is the radial direction. The origin is the center of the solenoid arrangement, corresponding to the center of the electrodes.

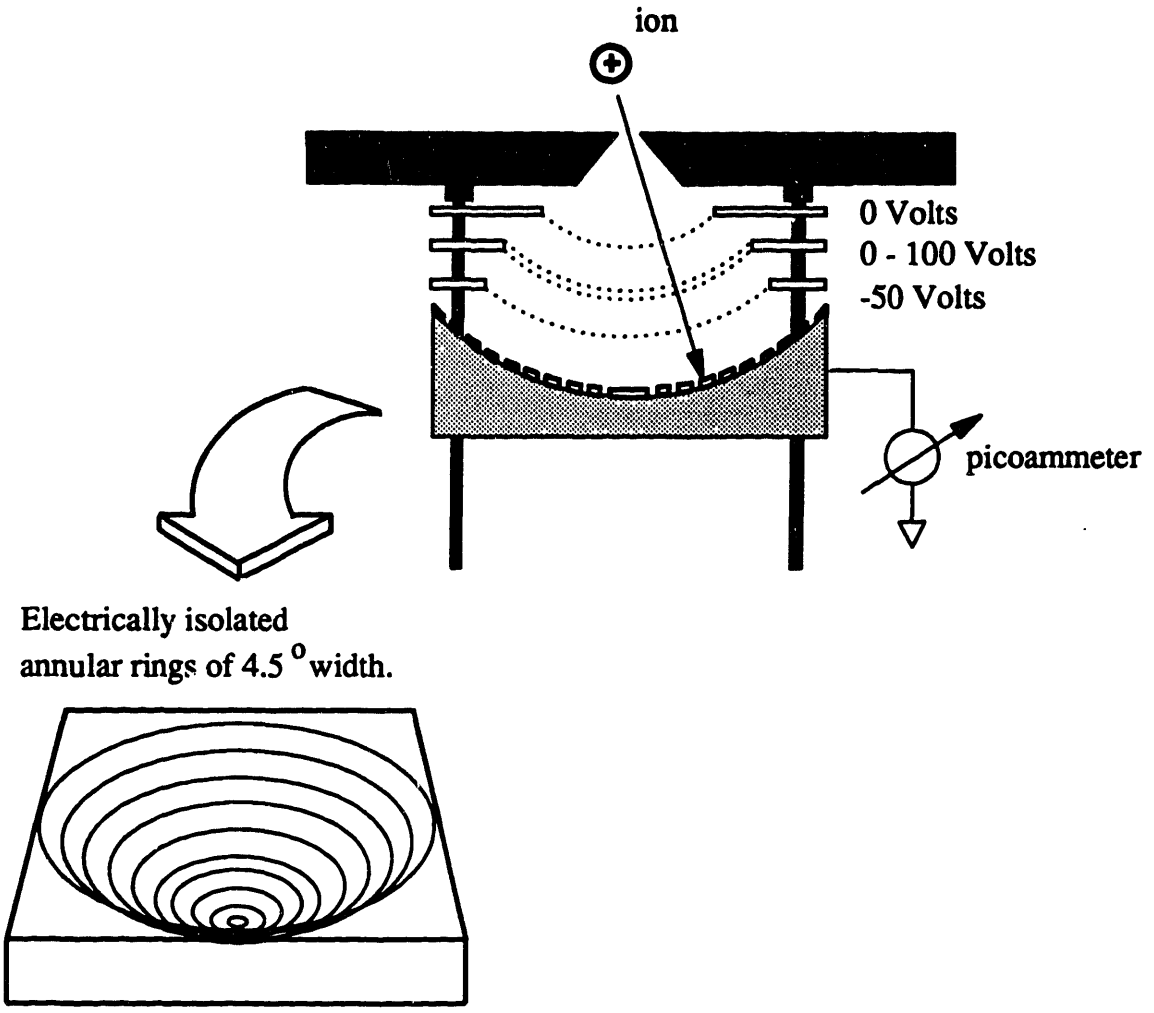


Figure 2.9: Detailed drawing of the ion energy and angle analyzer. The analyzer consists of three spherical grids for repelling ions or electrons and a spherical ion collector with electrically isolated annular rings.

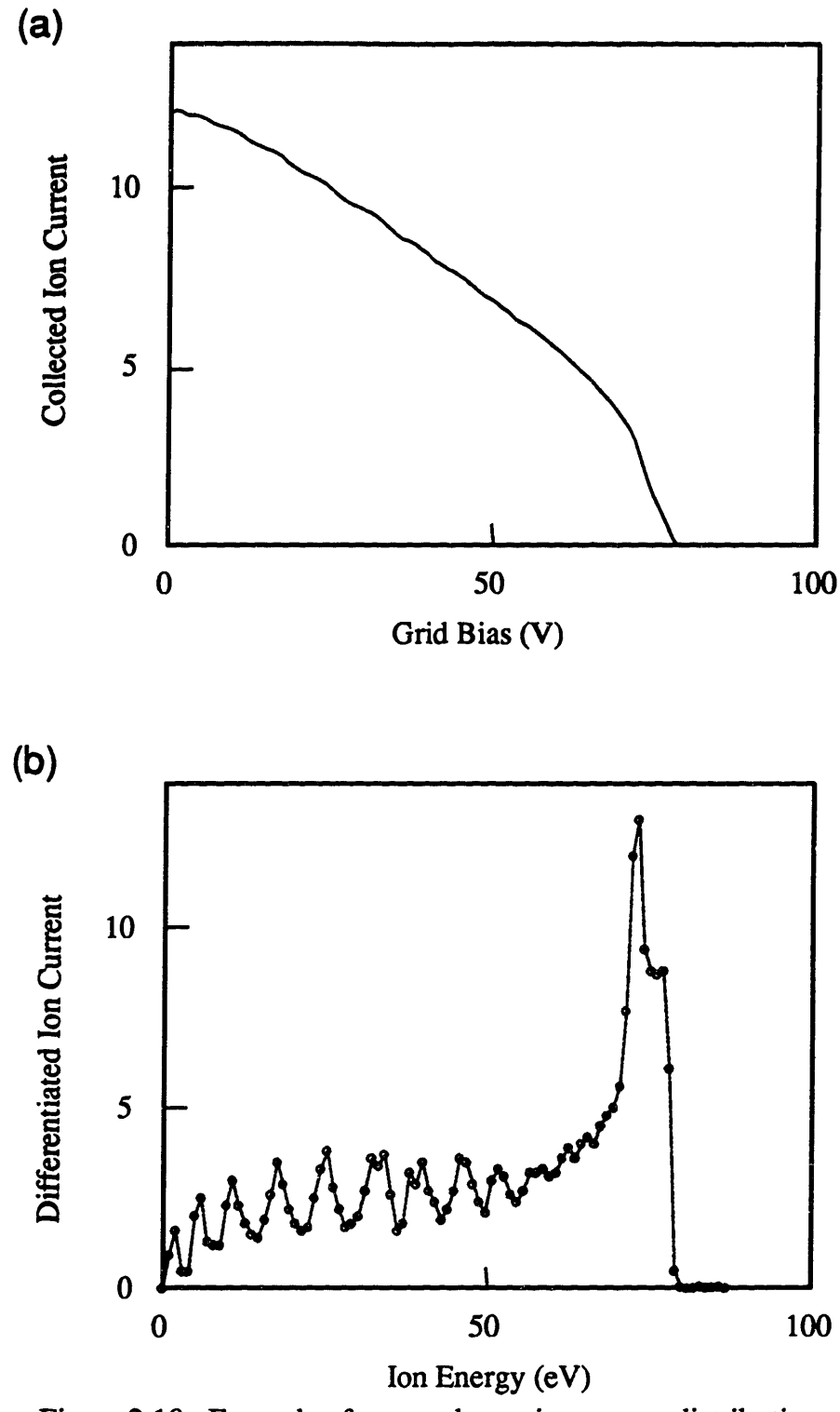
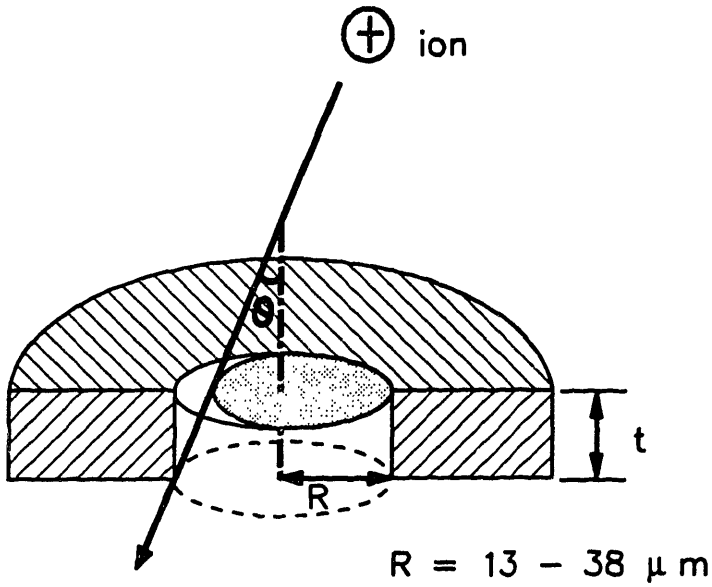


Figure 2.10: Example of argon plasma ion energy distribution. (a) raw data (b) differentiated data.



$$R = 13 - 38\ \mu m$$

$$t = 9\ \mu m$$

$$\text{Effective Area Ratio} = \frac{\text{Collection Area}}{\text{Actual Area}}$$

$$= \frac{\text{Shaded Area}}{\pi R^2}$$

Figure 2.11: Schematic of an orifice cross section made from an aluminum sheet used to sample ions bombarding the electrode. The shaded area is the effective area ions "see" when incident with an angle, θ, off the electrode normal.

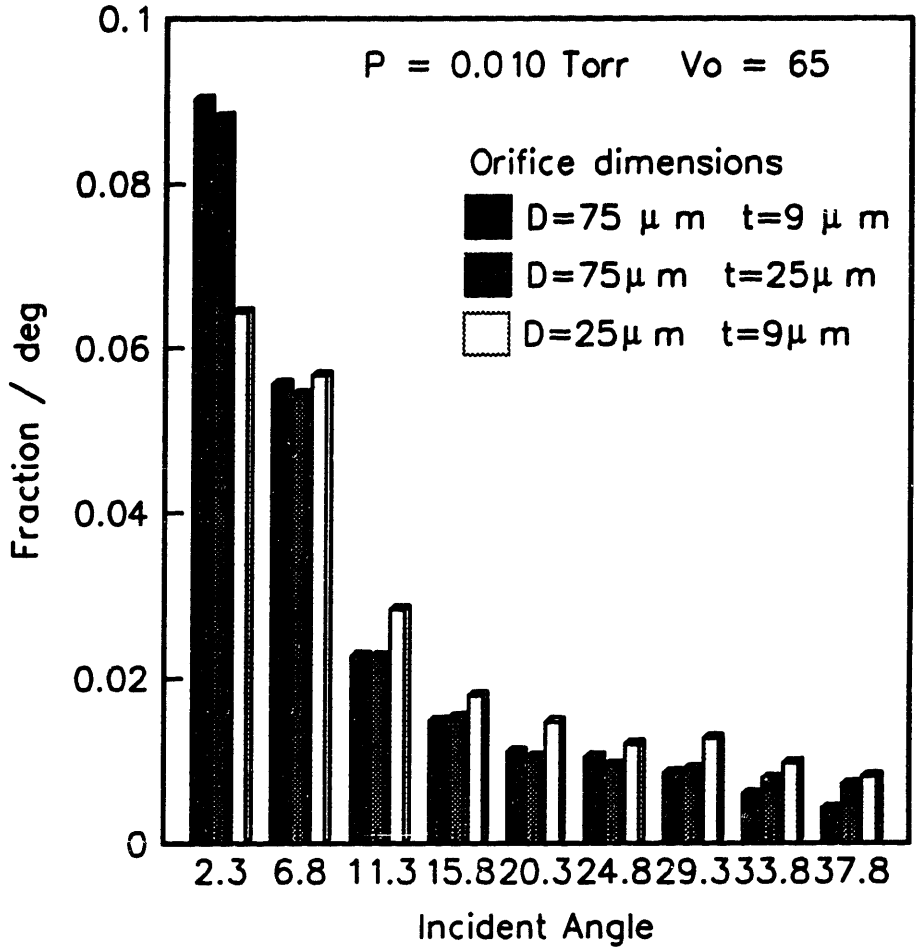


Figure 2.12: Comparison of IADs measured with three orifice sizes after correcting for the effective area.

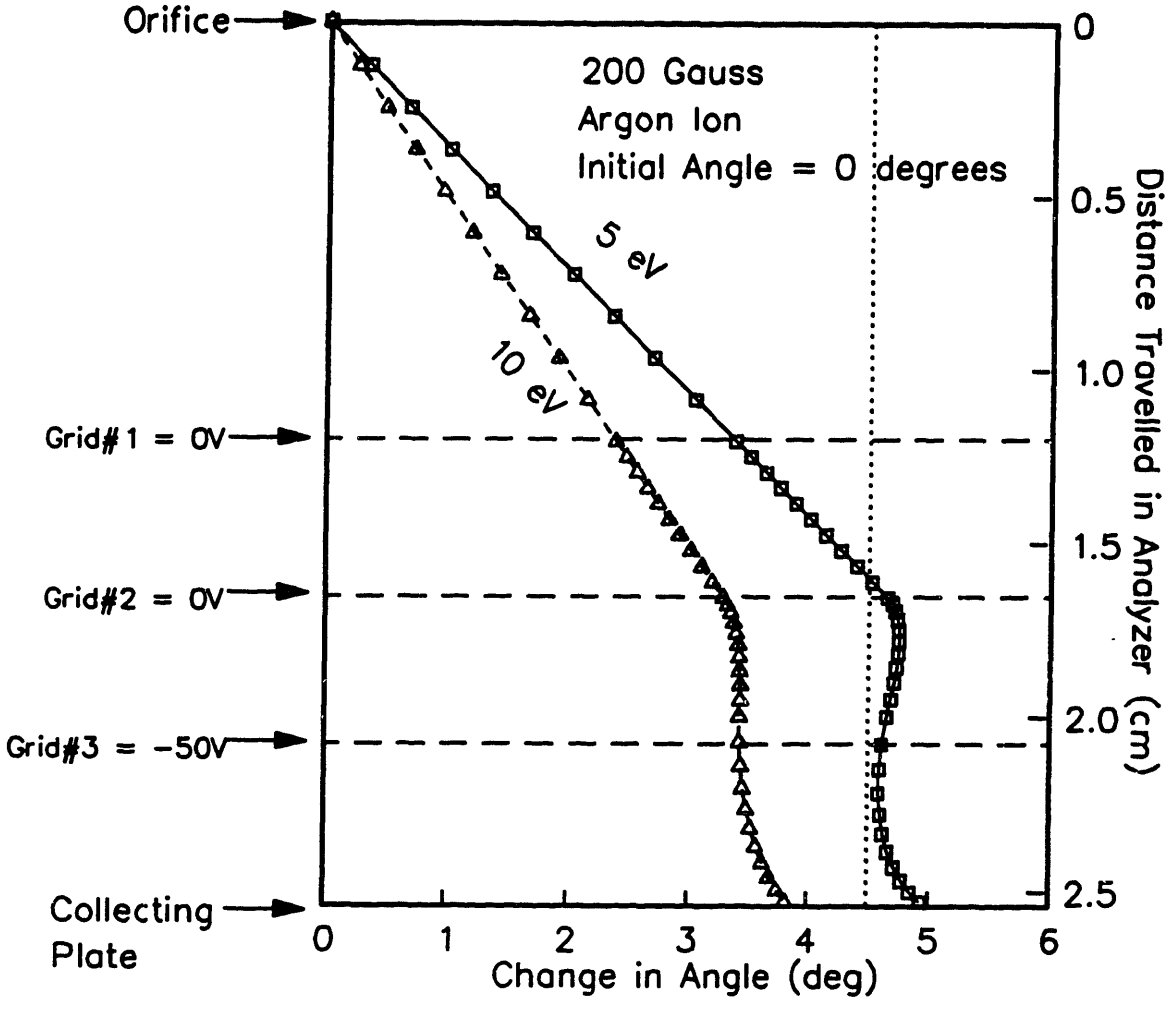


Figure 2.13: Ion trajectory in the ion analyzer with a 200 Gauss magnetic field and no bias on the second grid. Ions with 0 degree initial incident angle and low energy experience the maximum deflection.

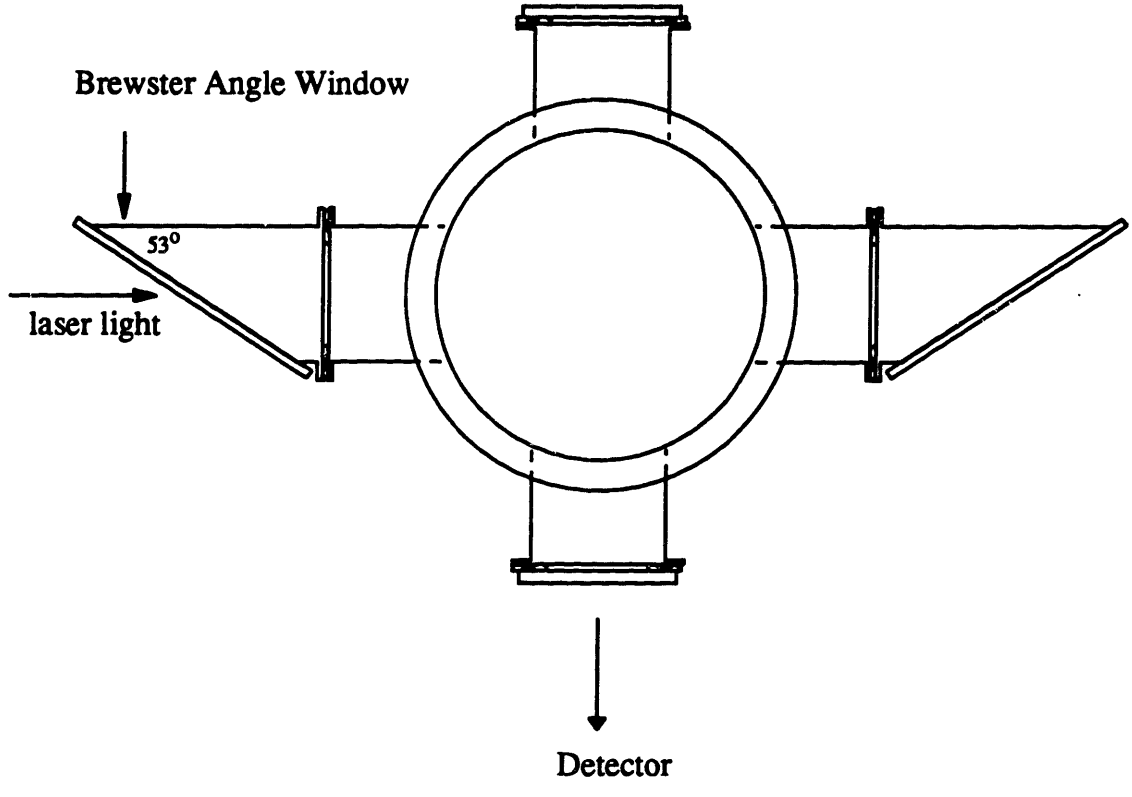
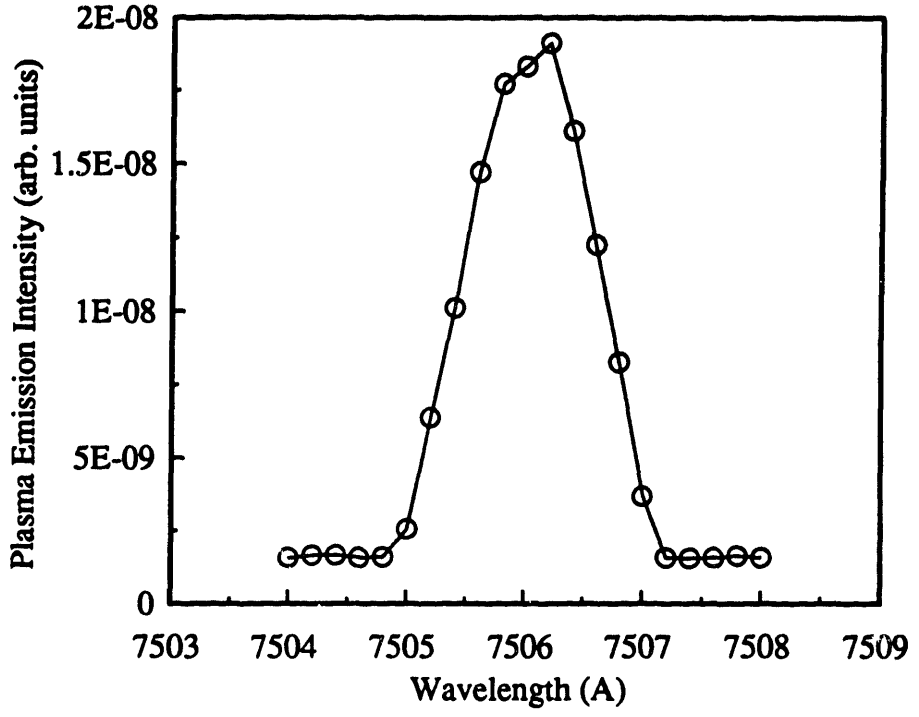


Figure 2.14: Top view of the reactor with Brewster windows attached.



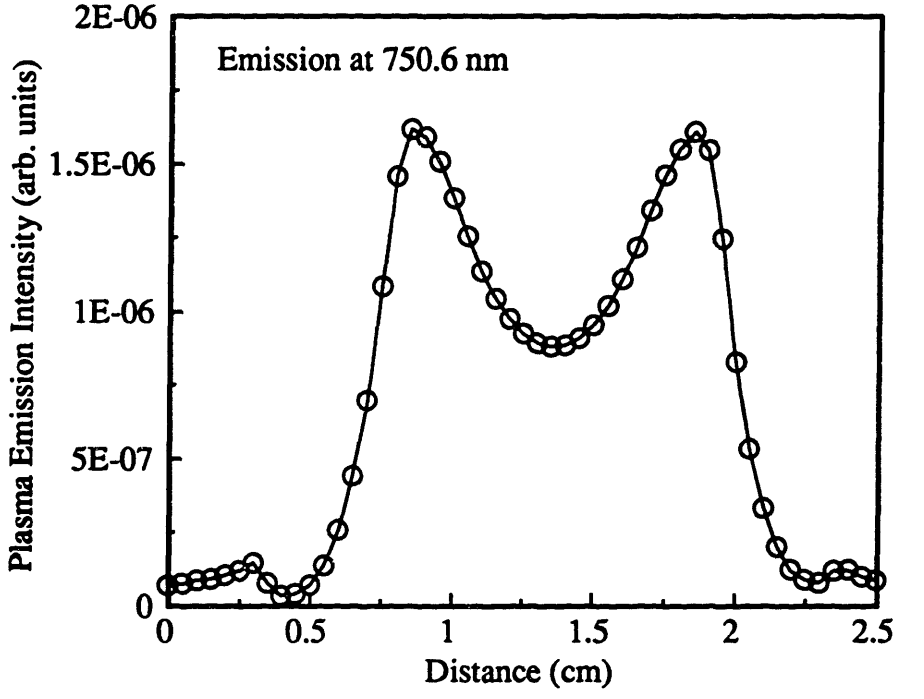


Figure 2.15: Examples of time averaged emission from argon plasmas. (a) wavelength scan from a 0.01 Torr plasma (b) spatial emission scan from a 0.2 Torr plasma.

CHAPTER 3

POWER DEPOSITION SCALING GROUPS FOR ARGON AND SF_6

3.1 INTRODUCTION

Plasma is created by putting power, generally at 13.56 MHz, between two electrodes and breaking the gas down into ions, reactive radicals, and electrons. This gas then etches the film by reacting with the surface species to form volatile products. The amount of power applied to the plasma determines the flux and energy of etchants striking the wafers placed on the electrodes. Therefore, it is important to accurately measure the power deposited into the plasma and study how it varies with operating conditions. In this chapter, the measurements and analysis will focus on plasma properties powered only at 13.56 MHz. The amount of power deposited in the plasma determines the voltage and current across the electrodes and the phase difference between the voltage and current waveforms. The voltage and current, in turn, determine the ion energy and ion flux striking the electrode surface. Profile simulators can then be used to predict etch rates and etch profiles from these ion properties.

This chapter describes how power deposition in the plasma varies with reactor configuration, pressure, and current. Beneking (1990) investigated the variation of power with operating frequency at a fixed pressure of 3 Torr. Godyak *et al.* (1991b) presented power variations, at 13.56 MHz, as a function of pressure. Presented here are power measurements taken at 13.56 MHz as a function of various plasma lengths and diameters. The phase difference and the voltage are related to the power using an electrical analog representation of the plasma. Data from argon and SF_6 plasmas are used to show how well the model relates power, current, voltage, phase, and reactor configuration together.

3.2 POWER VARIATION WITH OPERATING CONDITIONS

The discussion in this section involves power deposition in a symmetric parallel electrode configuration. Power is deposited in two places in the plasma; in the bulk where power primarily goes into electron impact reactions to produce radicals, ions, and excited species, and in the sheath where power is used primarily to accelerate ions toward the electrode surfaces. Godyak *et al.* (1991b) express power as bulk plus sheath power deposition in the following way.

$$P = \frac{1}{2}(I_o V_b + I_o^2 R_s) \tag{3.1}$$

where V_b is the time averaged voltage drop across the bulk of the plasma, R_s is the sheath resistance of both sheaths, and I_o is the total current passing through the discharge. The following discussion of power consumption in the plasma is separated into two parts, bulk and sheath dominated regimes.

3.2.1 Bulk Power Deposition

Power deposition in the bulk of the plasma dominates over that of the sheath when operating in the low power and high pressure regime. The bulk power deposition term in the above equation shows that power should be proportional to I_o for a fixed plasma size. Figure 3.1 is a graph of power versus I_o at 0.5 Torr, showing power measurements for a wide range of plasma diameters (9 cm to 17 cm) and lengths (2 cm to 6.7 cm). In all these cases, power is proportional to I_o^{α}, with $0.85 < \alpha < 1.15$. Since the sheath width is generally small compared to the gap spacing and is independent of the electrode spacing, as verified by spatial optical emission measurements, the plasma bulk length increases proportionally with electrode spacing. This indicates that power should scale with the gap spacing, d, and the area, A. Combining an energy balance for electrons and species balance for ions shows that $V_b/d^{\frac{1}{2}}$ is a constant for a

fixed pressure (Appendix K). Therefore, as the following equation shows, power should be proportional to $Ad^{1/2}$. Dividing both sides of Equation 3.1 by $Ad^{1/2}$ and neglecting sheath power deposition gives:

$$\frac{P_b}{Ad^{1/2}} = \left(\frac{V_b}{2d^{1/2}}\right)\frac{I_o}{A} = \left(\frac{\mathcal{E}_b d^{1/2}}{2}\right)\frac{I_o}{A} \qquad (3.2)$$

$$\text{where} \quad \mathcal{E}_b \text{ (bulk electric field)} \equiv \frac{V_b}{d} .$$

If electron current dominates in the bulk, then $I_o/A = nq\mu\mathcal{E}_b$ can be substituted into the above equation for I_o, where n is the electron density in the bulk, q is the electron charge, and μ is the electron mobility. This substitution would make P_b proportional to n, which is the same result shown in plasma simulations by Surendra and Graves (1991b). Figures 3.2 shows graphs of power/$Ad^{1/2}$ plotted against I_o/A for 0.05 and 0.5 Torr argon plasmas at various plasma geometries. These graphs show that the data points all fall nearly along the same line, demonstrating that power can be scaled by $Ad^{1/2}$ as long as power is deposited in the bulk and V_b is independent of power. In Figure 3.2 (b), the 0.05 Torr data points at high power levels show that power/$Ad^{1/2}$ is not proportional to I_o/A. The next section will show that sheath power deposition is beginning to dominate in this region. According to Equation 3.2, estimates for the electric field in the bulk of the plasma can be deduced from the y-axis intercepts of the graphs in Figure 3.2. They are:

$$\mathcal{E}_b d^{1/2} = 3.0 \text{ V/cm}^{1/2} \quad \text{at 0.05 Torr}$$

$$\mathcal{E}_b d^{1/2} = 8.8 \text{ V/cm}^{1/2} \quad \text{at 0.5 Torr}$$

Electric field values have also been calculated using the same method with data taken at 1 Torr and 2 Torr. Figure 3.3 shows the bulk electric field values and the best fit curve through these points which can be written as:

$$\mathcal{E}_b = 12.5 \left(\frac{p}{d}\right)^{1/4}$$

$$\text{where} \quad \mathcal{E}_b \equiv \frac{V}{cm} \qquad p \equiv \text{Torr} \qquad d \equiv \text{cm} \tag{3.3}$$

The $p^{1/4}$ dependence is expected from electron and ion energy and mass balances.

Using the information given in Equation 3.3, an equation for bulk power deposition can be written which universally applies to various plasma dimensions and pressures. This is done simply by multiplying and dividing the right side of Equation 3.2 by $p^{1/4}$.

$$\frac{P_b}{A d^{1/4}} = \kappa_1 \frac{I_o}{A} p^{1/4}$$

$$\text{where} \qquad \kappa_1 = \frac{\mathcal{E}_b}{2}\left(\frac{d}{p}\right)^{1/4} = \text{constant} \tag{3.4}$$

Figure 3.4 shows data taken at various pressures and gap spacings, showing that Equation 3.4 can be used to predict bulk power deposition. Since all the points collapse roughly onto the same line, this graph also shows that $\mathcal{E}_b d^{1/4}/p^{1/4}$ is constant for argon discharges in the range of pressures, 0.05 Torr to 2 Torr, and powers used in these experiments.

3.2.2 Sheath Power Deposition

As power to the plasma increases or the pressure decreases, power deposition switches from bulk to sheath dominated regime. In this regime, although the power still goes to the bulk of the plasma, a larger fraction of the power goes into accelerating more ions to a higher energy toward the electrode surface (Godyak *et al.* 1991b, Surendra & Graves 1991b). Dividing both sides of Equation 3.1 by the plasma area while neglecting the bulk power deposition term gives:

$$\frac{P_s}{A} = \frac{1}{2}\left(\frac{I_o}{A}\right)^2 AR_s \tag{3.5}$$

From the analysis in the following pages, sheath resistance can be expressed as

$$R_s = \frac{2\kappa_3}{Ap^{1/2}}\left(\frac{I_o}{A}\right)^{1/2} \tag{3.6}$$

where κ_3 depends only on the physical properties of the ion and the operating frequency. Substituting Equation 3.6 into Equation 3.5, gives:

$$\frac{P_s}{A} = \frac{\kappa_3}{p^{1/2}}\left(\frac{I_o}{A}\right)^{2.5} \tag{3.7}$$

From Figure 3.5, the data at 0.05 Torr and various reactor geometries all approach the same line as power increases, showing that power/area approaches $(I_o/A)^{2.5}$ dependence. There is insufficient data at high powers to fully graph out Equation 3.7, but Godyak *et al.* (1991b) do show that power depends on I_o^{2+}. Figure 3.5 shows the transition from bulk to sheath dominated power deposition occurs at about 0.001 Amp/cm^2 at 0.05 Torr, a number that is bounded by the transition values observed in Godyak *et al.*'s (1991b) data at 0.03 Torr and 0.1 Torr.

3.3 ELECTRICAL ANALOG MODEL OF ARGON PLASMA

An electrical analog representation of the plasma can be used to calculate the sheath voltage, V_s, and phase difference between the voltage and current waveforms, ϕ. Being able to relate V_s to I_o and power is important since V_s directly determines the ion energies striking the wafer surface. A commonly accepted electrical analog model of the plasma, shown in Figure 3.6 (a), consists of a capacitor to represent the two sheaths, an inductor to account for the inertia of the charged particles which do not respond instantaneously to the time varying electric field, a

resistor in the bulk of the plasma to represent the charge carrier collisions, and a resistor in the sheath to account for heating of the ions by the sheath electric field. These components can be written as functions of the plasma dimensions (A, d_b, ℓ_s), mass of particle j (m_j), collision frequency (ν_c), operating angular frequency (ω), density of particle j (n_j), and ion to total current ratio (I_i/I_o), using force and energy balances (Appendix K):

$$C_s = \frac{\varepsilon_o A}{2\ell_s} \qquad L_b = \frac{m_j}{n_j q^2}\frac{d_b}{A}$$
$$R_b = \frac{m_j \nu_c}{n_j q^2}\frac{d_b}{A} \qquad R_s = \frac{4\ell_s}{\omega \varepsilon_o A}\frac{I_i}{I_o} . \tag{3.8}$$

In developing the functionality of R_s, the displacement current is assumed to be the dominant type of current through the sheath (Surendra & Graves 1991b) and the sheath power loss was set equal to the ion flux times the time average sheath voltage.

In argon plasmas, the primary charge carrier through the bulk of the plasma are electrons (Surendra & Graves 1991b, Gogolides 1990). In this case, an inductor is not necessary to represent the plasma since electrons respond to the time varying electric field (Godyak *et al.* 1991b). Only R_b, R_s, and C_s, shown in Figure 3.6 (b) are needed to model the argon plasma. The argon plasma impedance is

$$Z_p = R_b + R_s - \frac{i}{\omega C_s} . \tag{3.9}$$

Equations for both V_s and ϕ are derived from this equation in the following two sections.

3.3.1 Voltage

The voltage drop in the sheath of an argon plasma can be estimated from the current amplitude. Current continuity through the plasma requires that the current through the bulk, which

is mostly electron current, has to equal the current through the sheath, which is mostly displacement current. When R_s is much smaller than $1/\omega C_s$, Figure 3.6 (c), the sheath voltage equals the current times the sheath impedance or

$$\frac{I_o}{A} = \left(\frac{\varepsilon_o \omega}{\ell_s}\right) V_s \quad . \tag{3.10}$$

Figure 3.7 illustrates that this relationship holds whenever sheath impedance dominates; that is, I_o/A is proportional to V_s. This equation ignores the voltage drop across R_s which is small compared to the voltage drop across C_s in the range of the measured data. At high powers and low pressures, the voltage drop across R_s may become significant.

With Equation 3.10, the previous scaling laws of power deposition as a function of I_o can now be expressed as a function of V_s. For bulk power deposition, substituting Equation 3.10 into Equation 3.4 gives a relationship between power and voltage for various plasma dimensions and pressures.

$$\frac{P_b}{A d^{1/2}} = \left(\frac{\varepsilon_o \omega}{2\ell_s} \frac{\mathcal{E}_b d^{1/2}}{p^{1/2}}\right) V_s p^{1/2} \quad \text{or} \tag{3.11}$$

$$V_s = \kappa_2 \frac{P_b}{A d^{1/2}} \frac{\ell_s}{p^{1/2}}$$

$$\text{where} \quad \kappa_2 = \frac{2}{\omega \varepsilon_o}\left(\frac{p^{1/2}}{\mathcal{E}_b d^{1/2}}\right) = \text{constant} \; . \tag{3.12}$$

If the following two conditions are met, then κ_2 of Equation 3.12 should be constant and all the data points should fall on the same line: 1) $\mathcal{E}_b d^{1/2}/p^{1/2}$ must remain constant with changes in power and pressure, and 2) the frequency is not varied. Data measured from 0.05 to 2 Torr and various plasma dimensions follow this equation well, as seen in Figure 3.8 (a). The non-zero

intercept exists in the graph but is not accounted for by the model because the simple combination of circuit elements fail to capture some of the plasma physics, in this case, the plasma potential. The bulk plasma potential is greater than the potential at either electrode, creating a time averaged electric field that accelerates ions toward the surface while retarding the electrons. Since the electron mobility is much greater than the ion mobility, this plasma potential develops to maintain an equal ion and electron flux to the surface. Surendra and Graves also show that electron power deposition does depend linearly on V_o where V_o is approximately equal to $2V_s/\sin(\phi)$ for argon plasmas.

An expression for sheath power loss is derived first by expressing the sheath resistance in terms of I_o. To do that, Lieberman's equation which relates ion flux, I_i/A, to sheath width, ℓ_s, ion mean free path, λ_i , and sheath voltage, V_s, is combined with Equation 3.10 to give the ratio of the ion current to the total current on the electrode,

$$\frac{I_i}{I_o} = 2.10 \left(\frac{2q}{m_i}\right)^{1/2} \frac{\lambda_i^{1/2}}{\ell_s \varepsilon_o^{1/2} \omega^{3/2}} \left(\frac{I_o}{A}\right)^{1/2} \quad . \tag{3.13}$$

From this equation, we can see that when pressure and frequency are constant, I_i/I_o is proportional to

$(I_o/A)^{1/2}$. The exponential dependence on I_o/A can vary from 0 to 1 depending on the assumptions made for ion transport in the sheaths (Lieberman 1989). Experimental argon ion flux data found in Godyak *et al.* (1991a) show that for $I_i/I_o \propto (I_o/A)^{\beta}$, β lies between 0.3 and 0.4. Particle simulation results of a 12 MHz Helium discharge show that $I_i \propto V_o^{1.15}$ (Surendra & Graves 1991b) or $I_i/I_o \propto (I_o/A)^{0.15}$.

Substituting Equation 3.13 into Equation 3.8, and using $\lambda_i = (k_b T)/(p\sigma)$ for the ion collision mean free path, the sheath resistance is

$$R_s = \frac{2\kappa_3}{Ap^{1/2}}\left(\frac{I_o}{A}\right)^{1/2}$$

$$\text{where} \quad \kappa_3 = 2.10\left(\frac{2q}{m_i}\right)^{1/2}\left(\frac{k_b T}{\sigma}\right)^{1/2}\frac{2}{\varepsilon_o^{3/2}\omega^{5/2}} \tag{3.14}$$

κ_3 is only a function of the physical properties of the ion and the operating frequency. The sheath resistance calculated by Godyak *et al.* (1991b) for argon plasmas in a fixed reactor diameter and spacing show similar dependence on current density and pressure. Combining this equation with the expression for sheath power loss, Equation 3.5, results in:

$$\frac{P_s}{A} = \frac{\kappa_3}{p^{1/2}}\left(\frac{I_o}{A}\right)^{2.5} \tag{3.15}$$

We can also rewrite the sheath power deposition equation in terms of voltage by substituting Equation 3.10 for I_o into Equation 3.15.

$$V_s = \kappa_4 \ell_s p^{1/5}\left(\frac{P_s}{A}\right)^{0.4}$$

$$\text{where} \quad \kappa_4 = \left[2.10\left(\frac{2q}{m_i}\right)^{1/2}\left(\frac{k_b T}{\sigma}\right)^{1/2} 2\varepsilon_o\right]^{-2/5} \tag{3.16}$$

Simulation results by Surendra and Graves (1991b) show that V_o depends on $P_s^{1/2}$. Figure 3.8 (b) shows that the 0.05 Torr V_s, measured at various plasma dimensions, approaches $(P/A)^{0.4}$ dependence. Again, there is insufficient data to show accurately how power varies with voltage at high powers.

3.3.2 Phase

The calculated phase difference can be compared to the measured values as a function of

operating conditions to check the soundness of the circuit model. Figure 3.9 shows the measured phase difference as a function of I_o/A at 0.05 Torr and 0.5 Torr. Three observations can be made from the data: 1) ϕ decreases with I_o/A, 2) the magnitude of ϕ is smaller for higher pressures and 3) the magnitude of ϕ is smaller for larger gap spacings. ϕ decreases with I_o/A because the charge carrier concentration in the bulk increases with I_o. This leads to a decrease in the bulk plasma resistance, and thus the plasma becomes more capacitive. Lowering the operating pressure also decreases ϕ. As pressure decreases, C_s decreases because ℓ_s increases; and R_b decreases because the number of collisions electrons have traveling through the plasma bulk decreases, both leading to a decrease in ϕ. The gap spacing between the electrodes has little effect on ϕ at low pressures, *e.g.* 0.05 Torr, because the plasma impedance is dominated by C_s, which is independent of gap spacing. The effect of gap spacing on ϕ is more clearly seen at 0.5 Torr where bulk power deposition dominates. Since ℓ_s is independent of gap spacing, C_s remains constant with changes in gap spacing. But since R_b increases linearly with gap spacing, the magnitude of ϕ decreases with gap spacing.

These experimental observations can be explained using the electrical analog model to form equations for ϕ as a function of operating parameters. The following derivations for ϕ are separated into two regimes, bulk and sheath power dominated regimes.

In the bulk power deposition dominated regime, R_s can be neglected from the electrical analog model. If the current through the bulk of the plasma is assumed to be carried solely by electrons and is proportional to the electron density (Gogolides 1990), then the current can be written as a function of the bulk electric field, $I_o/A = nq\mu\mathcal{E}_b$. Combining this with Equation 3.8 to express R_b and C_s in terms of plasma properties results in the following formula for ϕ in the bulk power deposition dominated regime:

$$\tan(\phi) = \left[\frac{\mathrm{Im}(Z_p)}{\mathrm{Re}(Z_p)}\right] = -\frac{1}{\omega R_b C_s} = -\frac{2\ell_s}{\omega \varepsilon_o d_b}\frac{nq^2}{m\nu_c} = -\kappa_5\frac{I_o}{A} \qquad (3.17)$$

$$\text{where} \quad \nu_c = \frac{q}{\mu m} \quad \text{and} \quad \kappa_5 = \frac{2\ell_s}{\omega \varepsilon_o V_b}$$

This equation shows that tan (ϕ) should be proportional to I_o/A. The 0.5 Torr and the lower power points in the 0.05 Torr data does fit this equation, where the sheath width calculated from κ_5 is 3 mm at 0.5 Torr.

If power deposition in the sheath is much greater than in the bulk ($R_b << R_s$), then ϕ is:

$$-\tan(\phi) = \frac{1}{\omega R_s C_s} = \frac{I_o}{I_i} \qquad (3.18)$$

Substituting in Equation 3.13 results in:

$$\tan(\phi) = -\kappa_6\left(\frac{I_o}{A}\right)^{-1/2} \qquad (3.19)$$

$$\text{where} \quad \kappa_6 = \frac{1}{2.10}\left(\frac{m_i \varepsilon_o}{2q}\right)^{1/2}\frac{\ell_s \omega^{3/2}}{\lambda_i^{1/2}}$$

This equation shows that tangent of the phase difference does not depend on the length of the plasma and depends on the square root of the current density. Since the data from Figure 3.9 do not extend far enough to the point where sheath power deposition is dominant, the increasing ϕ with increasing I_o/A trend is not clearly observed. However, from the data given by Godyak *et al.* (1991b), ϕ does increase with I_o/A, as predicted by Equation 3.19.

3.4 SHEATH WIDTH

The only dependent variable in Equations 3.12 and 3.16 not yet related to power, pressure, or plasma dimensions, is the sheath width, ℓ_s. The sheath width can be experimentally linked to

pressure and power by measuring the plasma emission intensity as a function of position between the electrodes.

These plasma emission scans along the length of the argon plasma give information not only on the sheath width, ℓ_s, but also indicate where electron energy may be deposited within the plasma. Typical emission scans are shown in Figures 3.10 and 3.11 for both 0.05 and 0.5 Torr plasmas. The 750.6 nm line is used to find the sheath width because it is one of the strongest emission lines from the argon plasma. Emission from the 565.3 and 549.8 nm are also monitored because their threshold energy for emission are closer to the ionization threshold of 15.7 eV as listed in Table 3.1.

Table 3.1: Argon Emission Threshold Energies

wavelength (nm)	threshold energy (eV)
549.8	15.3
565.3	15.1
750.6	13.5

It is probable that monitoring the emission from wavelengths with threshold energies closer to the ionization threshold will give a closer approximation of where ionization takes place than other plasma emission lines. Bulk power deposition in the argon plasma consists primarily of ionization, and some energy loss as electrons lose momentum in collisions with neutrals or initiate excitation reactions to form metastable atoms. Figures 3.10 and 3.11 indicate that there is power consumption in the bulk of the plasma from ionization at both 0.05 and 0.5 Torr. Since the electron concentration is much higher in the center of the plasma compared to the sheaths, energy loss from electron-neutral collisions will increase power consumption in the bulk of the plasma. Thus, a profile of the energy consumption in the plasma, when bulk power deposition dominates over sheath power deposition, should look similar to the ionization profile, Figure 3.10,

but with the power level at the center region raised by power deposition through other electron-neutral collisions. Figure 3.11 shows that at 0.05 Torr, there is bulk power deposition. At the conditions shown in this figure, the power deposited in the plasma depends on I_o^{2+}, indicating that there is also sheath power deposition. This power goes into accelerating ions toward the electrode surface, and does not appear as emission from the plasma. So although the 0.05 Torr spatial emission profile indicates power is being deposited only in the bulk, there is also significant power deposition in the sheaths.

The sheath width is defined as the distance from the electrode surface to the peak 750.6 nm spatial emission scan. Gogolides (1990) shows that the optical sheath width will overestimate the electrical sheath width at 0.5 Torr, but only by less than 0.7 mm. As pressure decreases, a larger error may result from estimating the electrical sheath width from the optical sheath width. Figure 3.12 shows the measured sheath width as a function of pressure. Experimentally, ℓ_s varies with pressure in the following manner:

$$\ell_s = 3.0 \cdot p^{-0.4} \qquad (3.20)$$
$$\text{where} \quad \ell_s \equiv \text{mm} \quad \text{and} \quad p \equiv \text{Torr}$$

If the sheath width is assumed to be proportional to the electron oscillation amplitude in the electric field near the sheath, then the expected dependence on pressure is $p^{-0.5}$ (Appendix K). Surendra and Graves (1991a) show that ℓ_s is proportional to $p^{-0.4}$ in 120 MHz plasma simulations, and Godyak *et al.* (1991b) use their power measurements to calculate the sheath width, showing $\ell_s \propto p^{-0.5}$. Therefore, the measured ℓ_s dependence on pressure matches the work of others. As power changes, the sheath width does not vary more than the resolution of our optical arrangement, 0.5 mm, except at very low powers, where ℓ_s increases as power decreases. Combining the bulk and sheath power equations, Equations 3.12 and 3.16, with the sheath width equation, we can predict power deposition as a function of pressure and voltage. These are useful

relationships since the voltage is directly determines the maximum and average ion energy striking the electrode surface.

3.5 SF_6 PLASMA POWER DEPOSITION

In SF_6 plasmas, the negative and positive ion concentration is approximately two orders of magnitude larger than the electron concentration. This differs from argon plasmas where the negative ion fraction is negligible. The large ion concentration in SF_6 plasmas makes the ion current through the plasma about the same order of magnitude as the electron and displacement currents (Gogolides 1990). Unlike electrons, the large mass of the ions make them unable to respond instantaneously to the applied 13.56 MHz field. This inertial effect appears in the phase shift between the measured voltage and current waveforms. In the argon plasma, the observed phase shift, ϕ, is always somewhere between -90° and 0°, indicating that the plasma can be represented by resistors and capacitors, and that the dominant types of currents in the plasma are electron and displacement currents. However, in the SF_6 plasmas, the observed phase shift can be greater than 0°, requiring an inductor in the circuit model of the plasma, Figure 3.6 (a). The voltage waveform across a capacitor is -90° phase shifted from that of the current ($I = C\ dV/dt$), whereas the voltage waveform across an inductor is +90° phase shifted from that of the current ($V = L\ dI/dt$.) The relative magnitude of the inductance and resistance of the ions compared to the electrons are estimated using ion and electron densities found in Gogolides (1990) and the following values for ion and electron mobilities.

Substituting these values into Equation 3.8, where $\nu_c = e/(m_j\mu_j)$, gives the following ratios for electron to ion inductance and resistance.

These ratios indicate that the plasma inductance is caused by ions, not electrons, and that both ions and electrons contribute to the plasma resistance.

$$n_e \approx 5X10^8 \text{ cm}^{-3} \quad \text{and} \quad n_{ion} \approx 3X10^{11} \text{ cm}^{-3} \quad \text{(Gogolides 1990)}$$

$$\mu_e p = 30 \frac{m^2 Torr}{V \cdot sec} \quad \text{(Yoshizawa 1979)} \tag{3.21}$$

$$\text{and} \quad \mu_{SF_5^-/SF_5} p = 0.05 \frac{m^2 Torr}{V \cdot sec} \quad \text{(Brand 1983)}$$

$$\frac{L_b(SF_5^-)}{L_b(e)} \approx 390 \quad \text{and} \quad \frac{L_b(SF_3^+)}{L_b(e)} \approx 270 \quad \text{and} \quad \frac{R_b(ion)}{R_b(e)} \approx \frac{1}{2} \tag{3.22}$$

Figure 3.13 shows the measured phase shift between the voltage and current waveforms for SF_6 plasma at 0.03 Torr and 0.3 Torr. This figure shows that as the gap spacing between the electrodes, **d**, increases, the plasma becomes more inductive. This observation is consistent with Equation 3.8 which shows that L_b is proportional to d.

As I_o/A increases, all the measured ϕ's of the same electrode area decrease, converging to the same value. In SF_6 plasmas, besides electron impact reactions with neutrals, there are also positive and negative ion recombination reactions, and electron attachment to neutral reactions. As I_o/A increases, the electron concentration increase linearly, but the ion concentration does not increase as fast because positive and negative ions are lost via recombination, which is a second order reaction with respect to the ion concentration (Gogolides 1990). Thus electron current carriers began to dominate as I_o/A increases, making the plasma less inductive.

The low pressure SF_6 plasmas (*e.g.* 0.03 Torr) appear more like the argon plasmas in that the measured phase shifts lie between -90° and 0°. This is probably because at low pressures, the charge particle concentration is sufficiently low that the displacement current is large compared to the electron and ion currents. Modeling results by Gogolides (1990) show that, at a fixed current, as pressure decreased from 1 Torr to 0.5 Torr, the ion concentration decreased, but the electron concentration remained almost unchanged. As pressure decreases, the mobility and diffusivity of charged species increase, increasing loss to the surfaces. The neutral concentration

also decreases with pressure, making the ion concentration, which depends on electron impact reactions with neutrals, decrease. Similar to increasing I_o/A, the ion concentration does not decrease linearly with pressure because loss through recombination depends on the square of the ion concentration. The net result of decreasing pressure is that, for a constant current density through the discharge, the electron current becomes more important while the ion current decreases. The ion concentration approaches the electron concentration as pressure decreases, making the plasma look more like an Argon discharge. This can be seen, not only in the measured ϕ's, but also by the power dependence of the SF_6 plasma in Figure 3.14 (a) which is similar to that observed in argon plasmas.

Figure 3.14 shows power measurements of SF_6 plasmas at 0.03 and 0.3 Torr. Using Equation 3.1 to describe power deposition in SF_6 plasmas, it would appear that for the range of data collected at 0.03 Torr, power deposition occurs both in the sheaths and bulk of the plasma while bulk power deposition dominates at 0.3 Torr. There is always significant power deposition in the bulk of SF_6 plasmas because of the large electric field in the bulk. The spatial emission profiles of the fluorine atom 7039 Å line corroborate this as seen in Figure 3.15. The spatial emission is a convolution of the electron energy and electron concentration. Although relative power deposition values at various locations in the plasma cannot be extracted from the spatial emission profile, the fact that there is strong emission in the center of the discharge indicates that there are quite a few electron impact reactions in the bulk of the plasma.

Although the data are collected only at two pressures for SF_6, bulk power deposition also seems to follow Equation 3.4 in that power depends on the square root of pressure. The 0.03 Torr and 0.3 Torr data in Figure 3.14 can be plotted together as power/$(Ad^{1/2})$ versus $(I_o/A\ p^{1/2})$, to show that all the data points will fall on the same line. Therefore, like argon plasmas, SF_6 plasmas will follow the scaling groups found in Equation 3.4.

The power dependence on voltage amplitude, V_o, in SF_6 plasmas differs from that in argon plasmas. In Figure 3.16 (a), the characteristics of the 0.03 Torr power-voltage curve are similar to that found in argon. However, the data at 0.3 Torr in Figure 3.16 (b) show that the voltage remains nearly constant with increasing power. V_o is a sum of the voltage drop across the bulk, which is nearly constant with power, and the voltage drop across the sheaths, which varies almost linearly with current. In 0.3 Torr SF_6 plasmas, the large bulk electric field, $\mathcal{E}_b$, cause the voltage drop in the bulk to dominate over the sheath voltage drop. The $\mathcal{E}_b$ in SF_6 is more than ten times the argon $\mathcal{E}_b$ (Gogolides 1990). Therefore, V_o depends mostly on the bulk voltage drop which does not change with power. In a few cases, V_o decreases slightly as power increases, indicating that $\mathcal{E}_b$ decreases. The decreasing $\mathcal{E}_b$ with power phenomenon has also been observed in continuum modeling of SF_6 plasmas (Gogolides 1990). Unlike argon plasmas, the voltage of the 0.3 Torr SF_6 plasmas depends on the electrode area, indicating that power loss to the walls of the reactor is important. The hotter walls of the reactor, compared to when argon plasmas are used, supports this. The measured voltage amplitude is higher for smaller electrode areas, where the chamber wall to electrode area ratio is greater. This indicates that the voltage drop in the bulk of the plasma is higher, and thus power losses in the bulk of the plasma, which includes power loss to the chamber walls, is higher than to the electrode for the smaller area electrode configuration.

3.6 COMPARISON WITH CONTINUUM MODEL RESULTS

The experimental measurements of plasma properties can be used to check the validity of using the continuum method to model the plasma. Several continuum models have been developed to model dc and rf powered plasmas, and electropositive and electronegative plasmas. (Graves & Jensen 1986, E. Zawaideh *et al.* 1986, Boeuf 1987, Paranjpe 1989, Gogolides 1990)

The basis of the continuum model is that all the particles in the plasma can be characterized by their average properties as a group rather than looking at individual particles or their distributions. This requires the length scale of interest to be larger than several collision mean free paths. Therefore, all the comparisons presented here will use plasmas with pressures between 0.1 Torr and 1 Torr.

Although there are several models, this chapter will focus only on comparisons of experimental data with Gogolides's (1990) and Huppert's model[1]. This is a one dimensional model; that is, the powered and grounded electrodes are parallel and the diameters of the electrodes are assumed to be much greater than the electrode separation so that edge effects are negligible. The comparisons between the experiment and model involve only argon plasmas, but for a range of pressures and powers. The plasma properties that can both be predicted by the model and measured experimentally are 1) the electrical properties, such as the voltage, current, and power, 2) the ion bombardment properties, such as the energy and the flux, and 3) the ionization profile between the electrodes. The measured spatially and temporally resolved plasma emission should be similar to the ionization profile because both ionization and plasma emission are electron impact reactions, and the excitation relaxation time is much less than the rf period and the characteristic diffusion time (Wiese & Martin 1980). The electrode gap spacing for both the model and experiments are set at 2.54 cm and the plasma diameters used in the experiments are 9 cm and 17 cm.

Figures 3.17 and 3.18 show how the measured voltage, current, power, and ion properties at the electrode compare with the continuum model results from 0.1 Torr to 1 Torr. In these figures, the symbols represent experimental points and the lines are the model results. Not only

[1]Continuum model results are generated by G. L. Huppert for comparing with experimental measurements.

does the model predict all the right trends, but it also predicts nearly all the correct values for the current, voltage, and ion flux at various pressures.

Since the model results follow the same trend as the experimental data, the scaling groups for power and current can be applied to the model results. Figure 3.17 (b) shows that power is approximately linearly dependent on current, implying that power deposition is in the bulk dominated regime. The model results follow the scaling equation for bulk power deposition, Equation 3.4, very well, as shown in Figure 3.19. Running the model at higher power levels should eventually produce the sheath power dominated regime.

Figures 3.20 and 3.21 compares the plasma emission to the predicted ionization rate as a function of time and distance. The plasma emission is measured at the 750.6 nm argon line with a boxcar integrator to get a resolution of 50 time intervals per rf period. The emission is measured at thirteen locations, evenly distributed between the electrodes. The model compares well with the experiment, with the best comparison at 1 Torr and the worst at 0.1 Torr. This is expected since continuum model assumptions are most applicable at high pressures.

3.7 DATA VERIFICATION

To check the accuracy of the power measurements and the validity of the stray impedance de-embedding technique, power data collected at 0.03 Torr, 0.5 Torr, and 1 Torr are compared to Godyak *et al.*'s (1991b) power measurements and to ion energy measurements.

Figure 3.22 (a) shows power versus voltage measurements from three pressures, taken from the reactor used in this work, Reactor 1, and Godyak *et al.*'s (1990b) reactor, which will hence forth be referred to as Reactor 2. The data from Reactor 1 and 2 are close in value. However, the measured power from Reactor 1 is generally higher for the same voltage and the plasma extinguishes at a higher voltage in Reactor 1 than in Reactor 2. These differences can be

explained by examining the differences between the reactor geometries. Figure 3.23 shows schematics of the two reactors. The electrode separation in Reactor 1 is set equal to that of Reactor 2, at 6.7 cm. However, the two plasma diameters are different, 17 cm and 14.3 cm. Since power is proportional to the plasma area, replotting the data in Figure 3.22 (a), scaled by the area, brings the data between the two reactors closer together, as seen in Figure 3.22 (b). The higher observed extinguishing voltage for Reactor 1 may be caused by the smaller electrode area compared to the chamber area in this reactor. Reactor 2's electrode extends beyond the confined area for the plasma, unlike Reactor 1's which ends within the glass chamber. As voltage is decreased, the fringe fields at the edge of Reactor 1's electrodes get weaker, unable to sustain a plasma. In contrast, Reactor 2's electric fields near the plasma edge remain uniform, thus able to sustain a plasma at a lower voltage than in Reactor 1.

Figure 3.24 compares the measured current and phase versus power from Reactor 1 and Reactor 2. Although power and voltage measurements from Reactor 1 and 2 matched, comparisons for current and phase do not coincide. For the same power deposited into the plasma, a lower current and a higher ϕ is measured in Reactor 1 than in Reactor 2, both of which indicate that the measured impedance from Reactor 1 is less capacitive plasma than that from Reactor 2. This observation may be partially explained by the difference in voltage and current measuring techniques. In Reactor 1, calibrated impedance cells subtract out all the stray impedances that are not from the plasma, which include the electrode and wall capacitance, resulting in a lower current and smaller phase difference than the impedance calculated directly from measurements. The accuracy of the voltage and current measured directly at the powered electrode in Reactor 2 is much better than that measured in Reactor 1 where the voltage and current have to be de-embed from a network of stray impedances, resulting in power measurement errors within 10 %.

The measured voltage amplitude is compared to the maximum ion energy to verify measured voltage accuracy. Ions which do not go through any collisions in the sheath should acquire an energy equal to the time averaged sheath potential. All other ions have energy lower than the sheath potential. Figure 3.25 shows a graph of the maximum ion energy versus sheath voltage calculated from a capacitor-resistor electrical model, Figure 3.6 (c). The data are measured at a wide range of pressures, plasma geometries, and voltages. The maximum ion energies are independent of pressure, electrode spacing, and electrode diameter. In all cases, the maximum ion energies nearly equal the sheath potential or were slightly greater, indicating that voltage is accurately measured. The greater maximum ion energy compared to the sheath voltage is consistent with the existence of the plasma potential which makes the potential in the bulk of the plasma greater than that of either electrode and which is not taken into account in the electrical analog model.

3.8 CONCLUSION

Power measurements in both argon and SF_6 plasmas are presented in this chapter, with the aim of showing how power, voltage, current, and phase vary with plasma dimensions and pressure. The power measurements show reasonable agreement with other independent measurements.

This chapter shows that power can be differentiated into two power deposition regions, the bulk and the sheaths. Bulk power deposition dominates at the high pressure, low power regime, where power goes into electron impact reactions. In the low pressure, high power regime, most of the power goes into the sheath to accelerate ions toward the surface. The plasma can be represented by a combination of resistors, capacitors, and inductors, which can be used to describe these two regimes of power deposition. The electrical analog representation of the argon plasma

is sufficient to provide information on the behavior of current, voltage, and phase, at various powers and pressures.

In argon plasmas, the average plasma bulk electric field, $\mathcal{E}_b$, varies as $(pd)^{1/2}$, with almost no dependence on the power. Using this, scaling relationships for power variations in the bulk and sheath power dominated regimes as a function of pressure and plasma dimensions are deduced. Bulk power deposition in argon plasmas at constant operating frequency follows

$$\frac{P_b}{Ad^{1/2}} = \frac{I_o V_b}{2Ad^{1/2}} = \kappa_1 \frac{I_o}{A} p^{1/2} \quad .$$

Sheath power deposition follows

$$\frac{P_s}{A} = \frac{\kappa_3}{p^{1/2}} \left(\frac{I_o}{A} \right)^{2.5} \quad .$$

SF_6 plasmas, although very different from argon plasma in its charge carrier concentration, also follows the same equation for bulk power deposition.

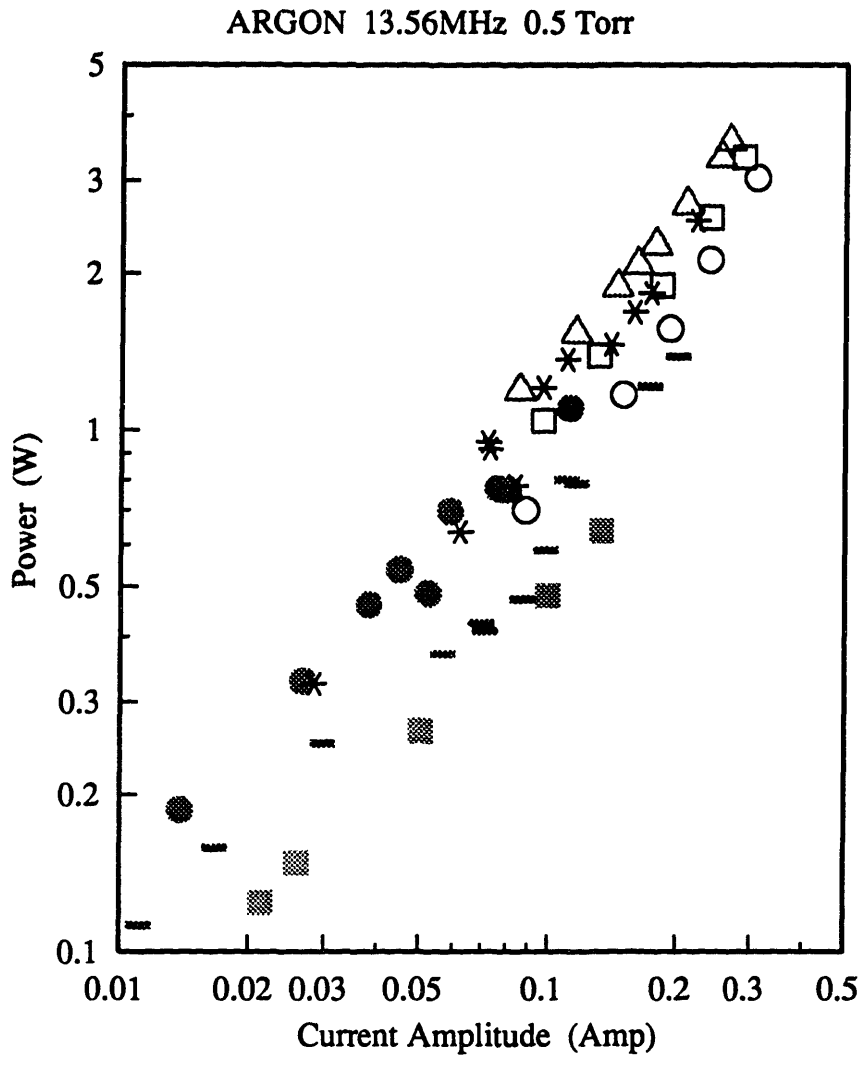
In both of the above equations, I_o can be replaced by V_s using Equation 3.10, as long as displacement current dominates in the sheaths. This is simple to do in argon plasmas since the equation for sheath capacitance can be used to link I_o and V_s.

$$C_s = \frac{\varepsilon_o A}{\ell_s} = \frac{I_o}{\omega V_s} \quad .$$

However, in SF_6 plasmas, the ions play a major role in the conduction of current and the voltage drop in the bulk of the plasma is very large. It is not possible to calculate V_s from the measured voltage and phase using a capacitor to represent the plasma sheath.

3.10 NOMENCLATURE

A	electrode area
C_s	capacitance of both sheaths
D	electrode diameter
d	electrode spacing
d_b	plasma bulk length = d - $2\ell_s$
$\mathcal{E}_b$	electric field in plasma bulk
i	$\sqrt{-1}$
I_o	current amplitude
J_i	ion flux
k_B	Boltzman's constant
L_b	plasma bulk inductance
ℓ_s	sheath width
m_e	electron mass
m_i	ion mass
m_j	mass of species j=i or e
n_j	density of species j
P	power
P_b	bulk power
P_s	sheath power
p	pressure
q	electron charge, 1.6 X 10^{-19} C
R_b	plasma bulk resistance
R_s	resistance of both sheaths
T	gas temperature
V_b	voltage drop across plasma bulk
V_f	floating potential
V_o	voltage amplitude
V_s	time averaged sheath voltage
Z_p	plasma impedance
κ_i	i:1-5, constants
ε_o	8.85 X 10^{-12} F/m
λ_i	ion mean free path
μ_j	mobility of species j
σ	collision cross section
ν_c	collision frequency
ρ_s	sheath resistivity
ϕ	phase difference (ϕ_V - ϕ_I)
ω	applied angular frequency



D = 9 cm	▬ d=3.5cm	● d=5cm	
D = 13 cm	▩ d=2cm	✱ d=5cm	
D = 17 cm	○ d=2.8cm	□ d=4.5cm	△ d=6.7cm

Figure 3.1: Power versus current data of 0.5 Torr argon plasma measured at various plasma lengths and diameters.

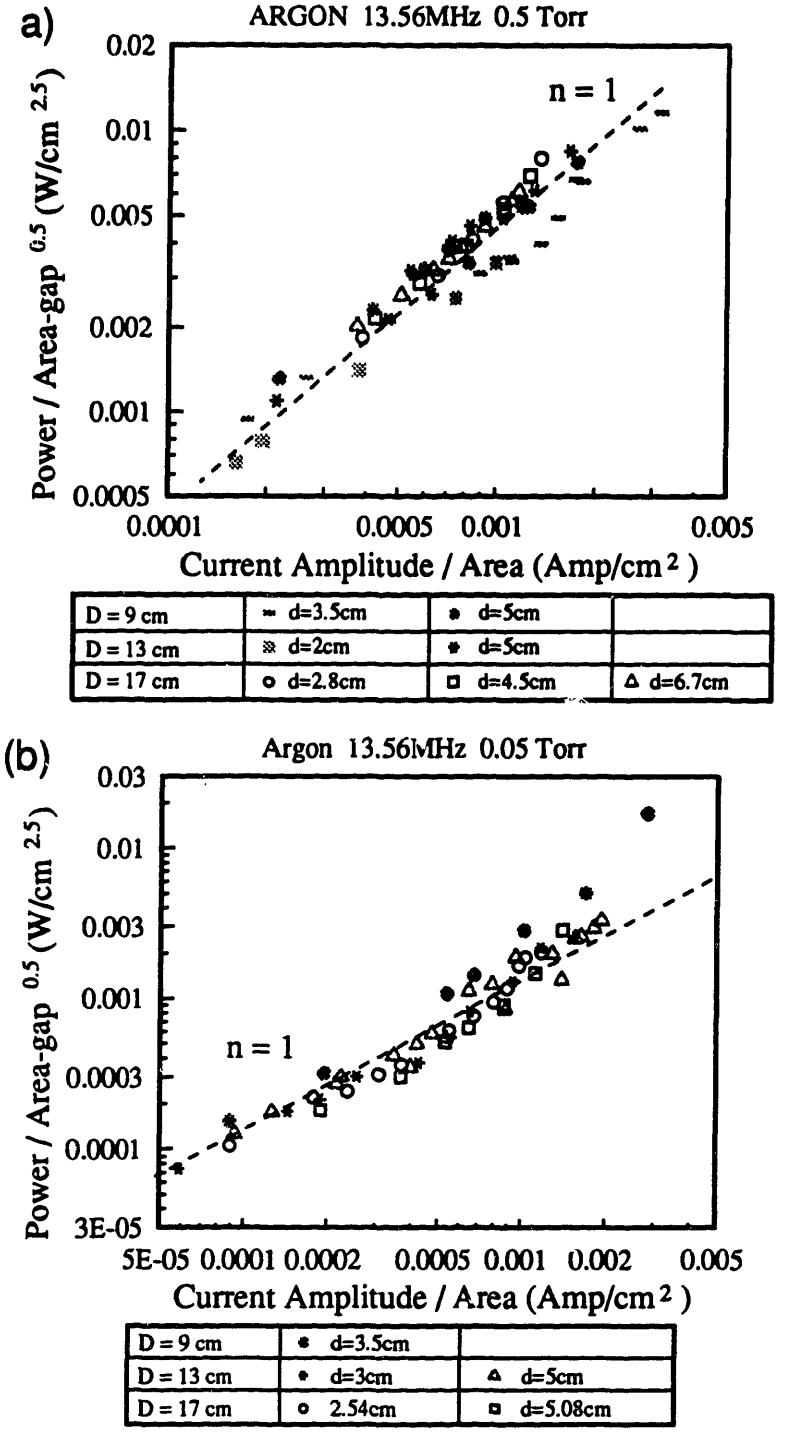


Figure 3.2: (a) 0.5 Torr and (b) 0.05 Torr argon plasmas showing how bulk power deposition depends on plasma size.

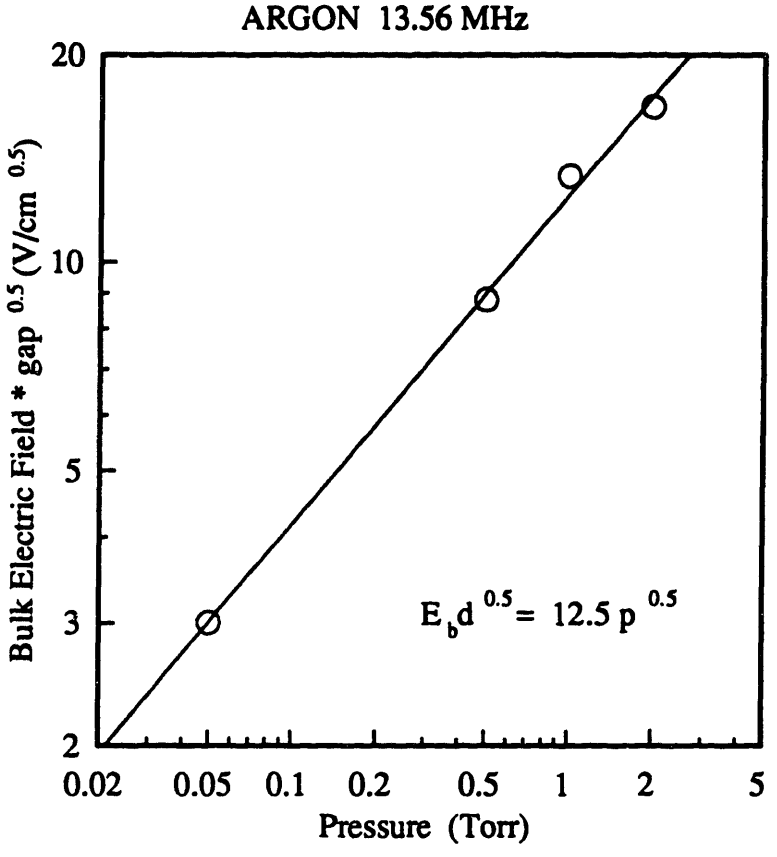


Figure 3.3: Electric field in the bulk region of argon plasmas at various pressures. The circles are data points and the line is found from linear regression.

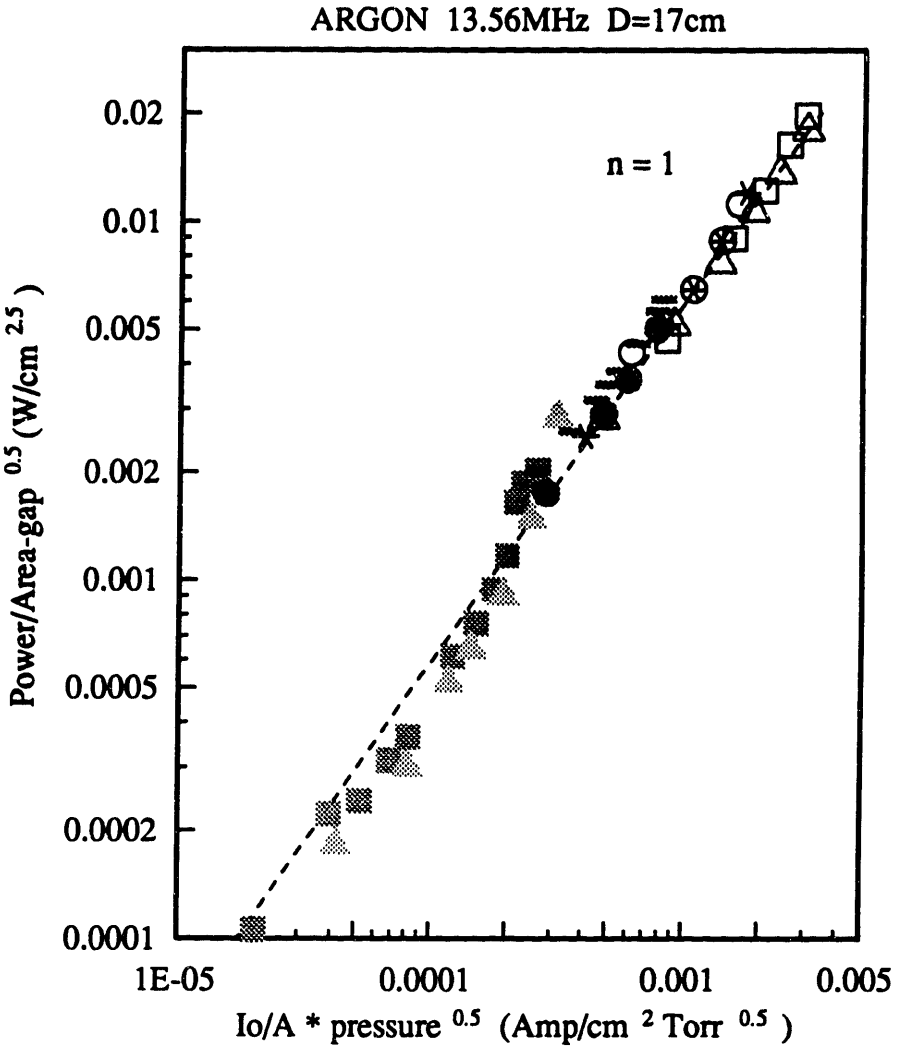
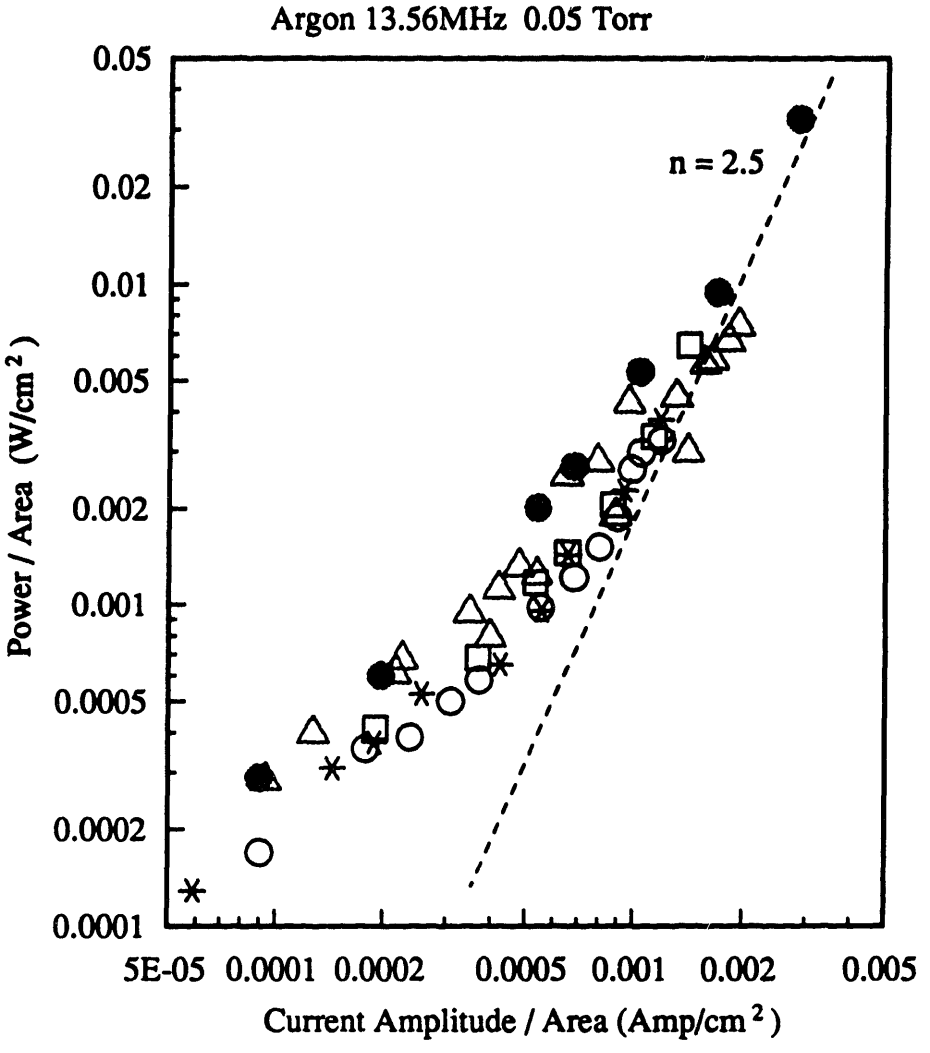


Figure 3.4: Argon power data taken at a wide range of conditions, showing how bulk power deposition varies with pressure and plasma size.



D = 9 cm	● d=3.5cm	
D = 13 cm	* d=3cm	△ d=5cm
D = 17 cm	o 2.54cm	□ d=5.08cm

Figure 3.5: Argon data at 0.05 Torr showing how sheath power deposition should approach $(Io/A)^{2.5}$ dependence.

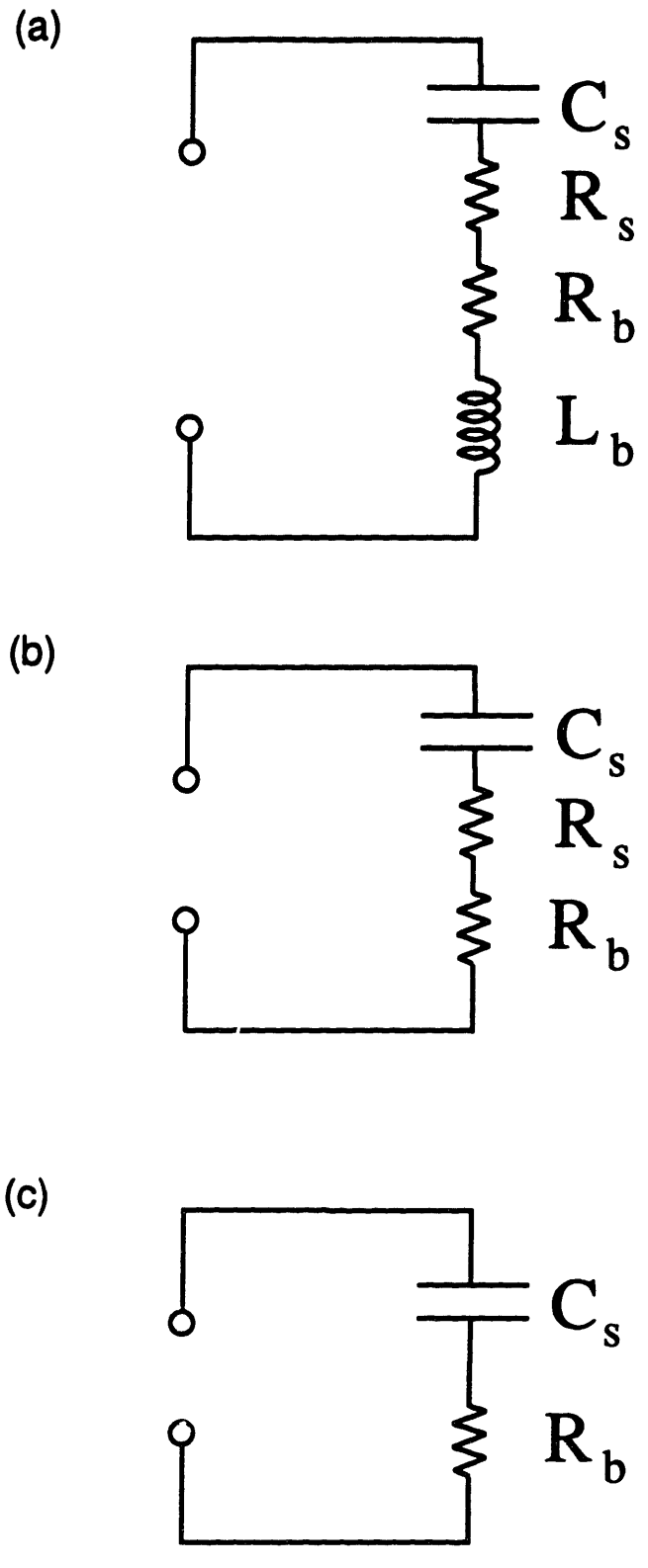


Figure 3.6: Electrical analog models of the plasma.

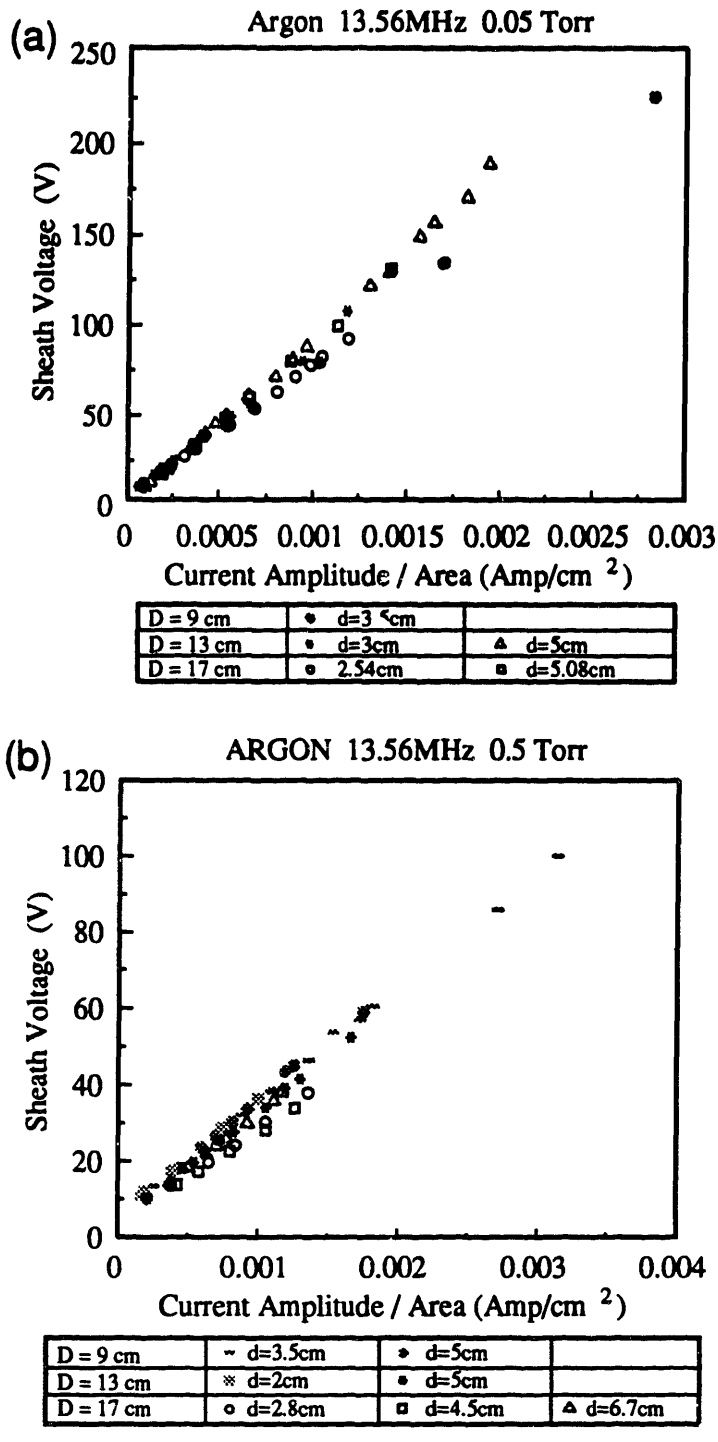


Figure 3.7: (a) 0.05 Torr and (b) 0.5 Torr argon plasmas showing the linear dependence between sheath voltage and Io/A.

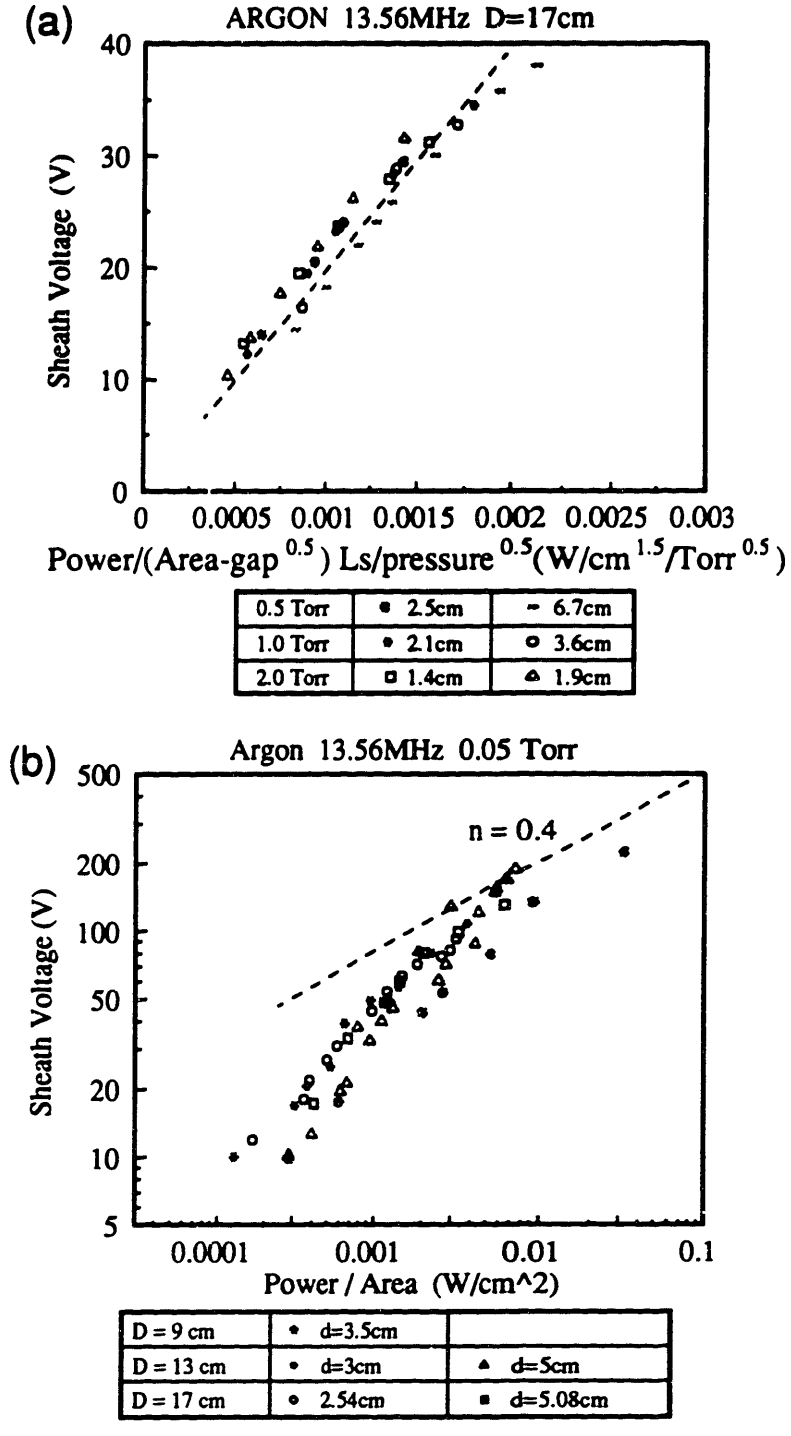


Figure 3.8: Graphs showing how sheath voltage depends on power in the (a) bulk power deposition and (b) sheath power deposition dominated regimes.

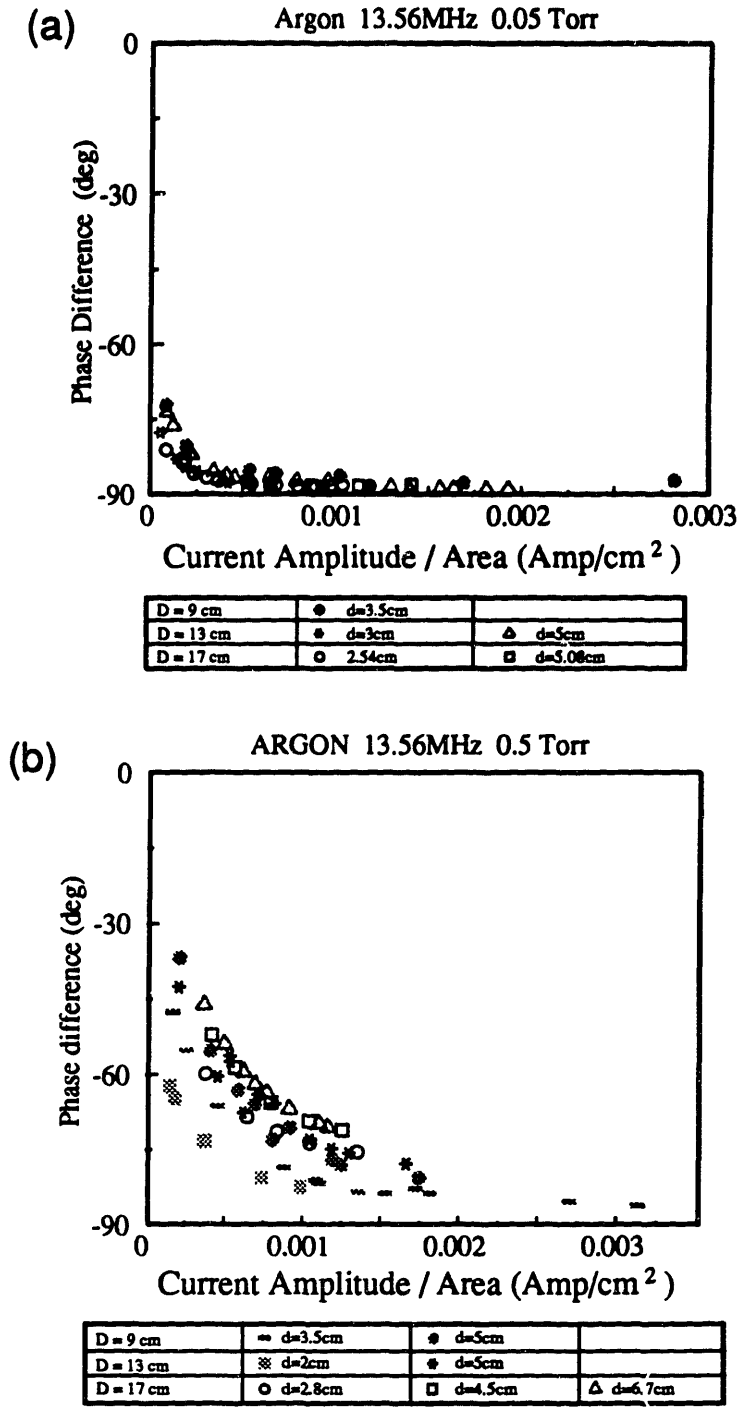


Figure 3.9: The phase difference between the voltage and the current waveforms from argon plasmas at (a) 0.05 Torr and (b) 0.5 Torr.

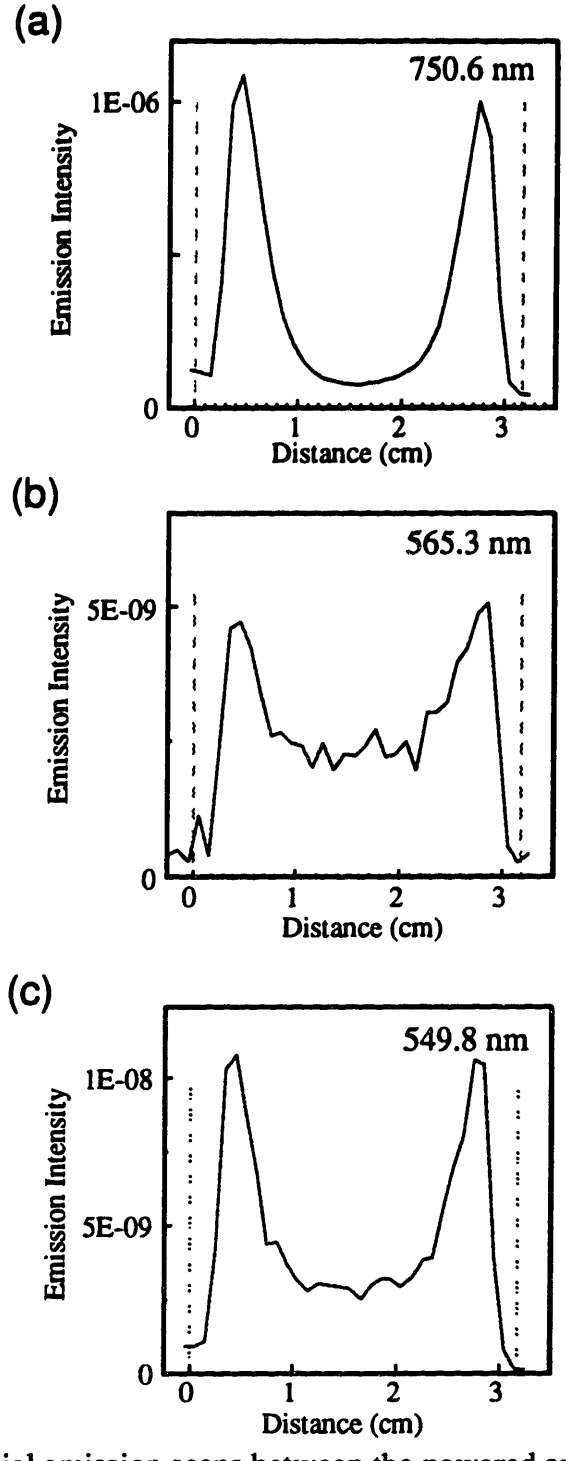


Figure 3.10: Spatial emission scans between the powered and grounded electrodes of the same plasma (0.5 Torr argon, d=3.2 cm, D=17 cm, Vo=63 V), but monitoring the emission at three different wavelengths: (a) 750.6 nm (b) 565.3 nm and (c) 549.8 nm.

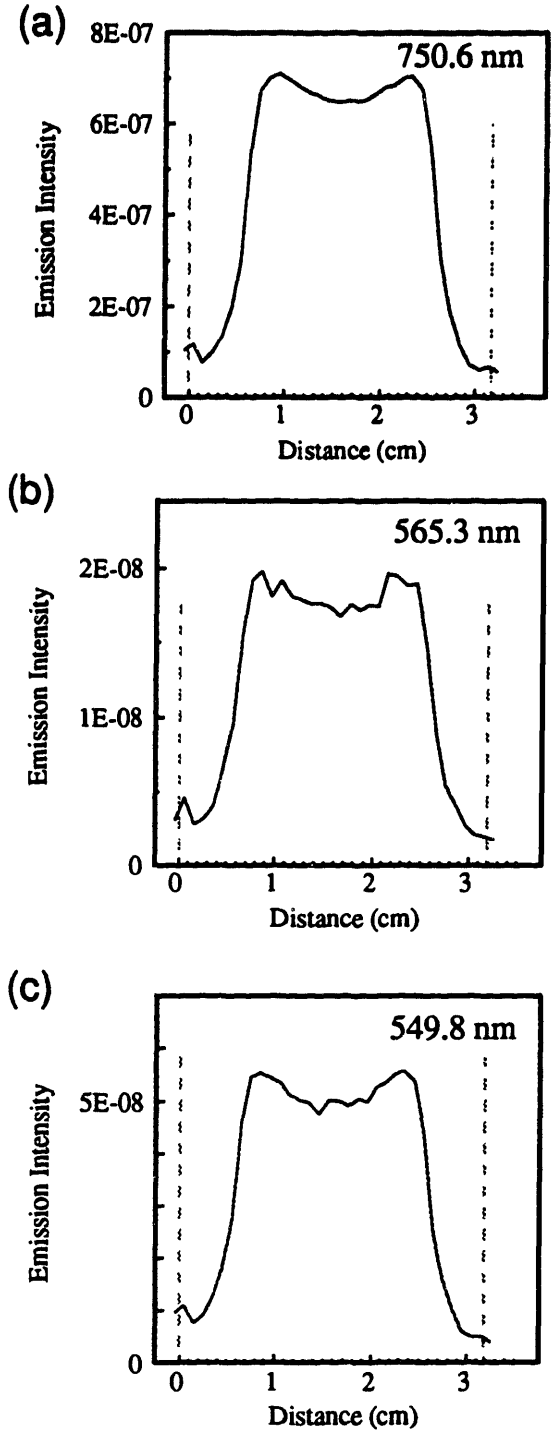


Figure 3.11: Same as Figure 3.10, except that the plasma is at 0.05 Torr, D=9cm and Vo=190 V.

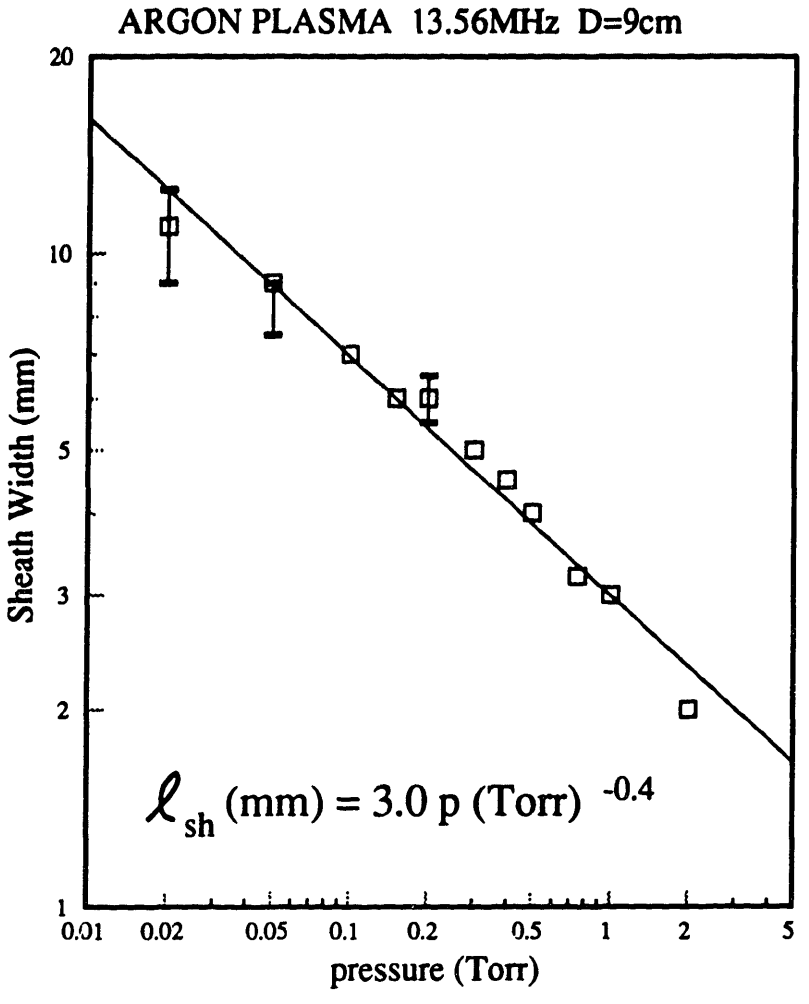


Figure 3.12: Sheath width measured using the optical emission scans at various pressures. The line drawn is the best fit line through the data.

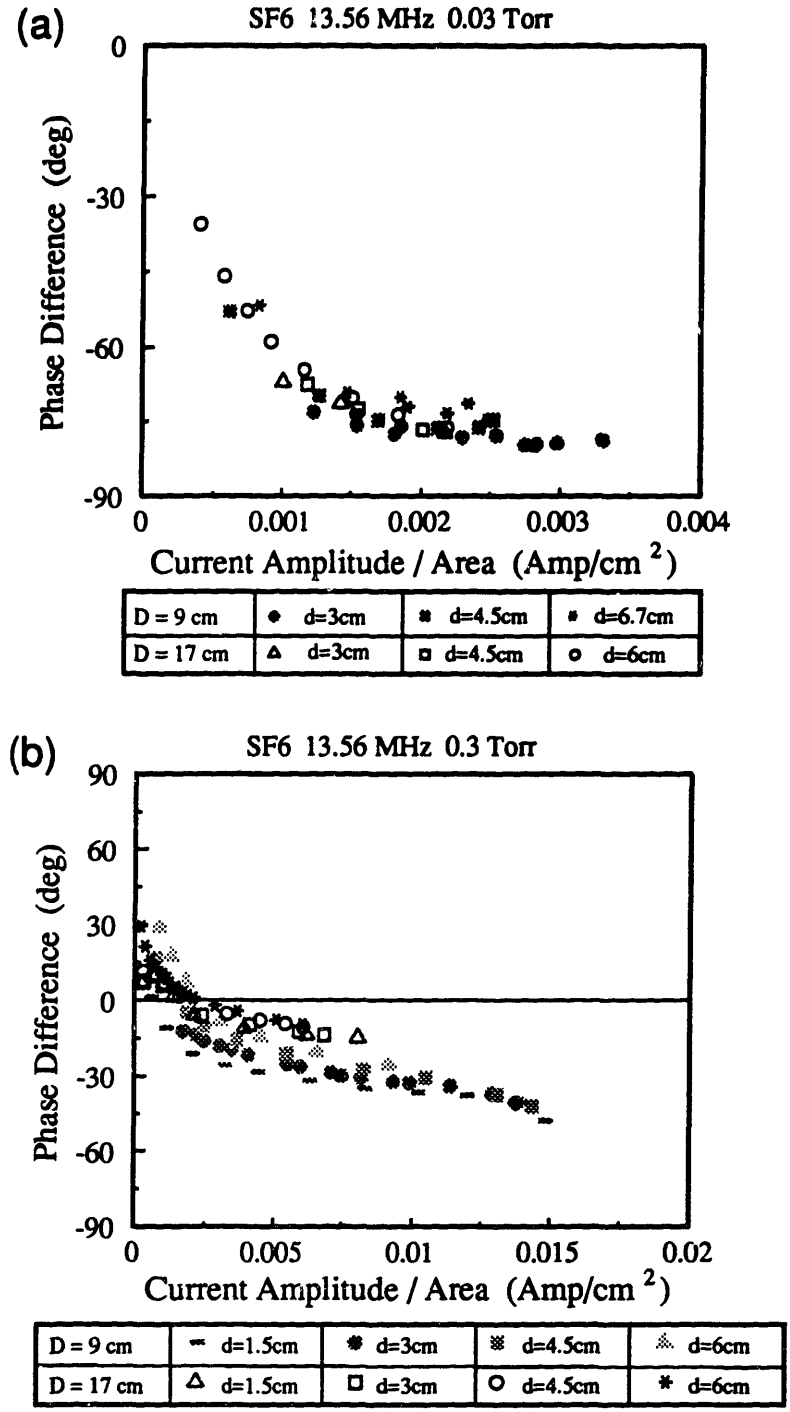


Figure 3.13: The phase difference between the voltage and the current waveforms from SF6 plasmas at (a) 0.03 Torr and (b) 0.3 Torr.

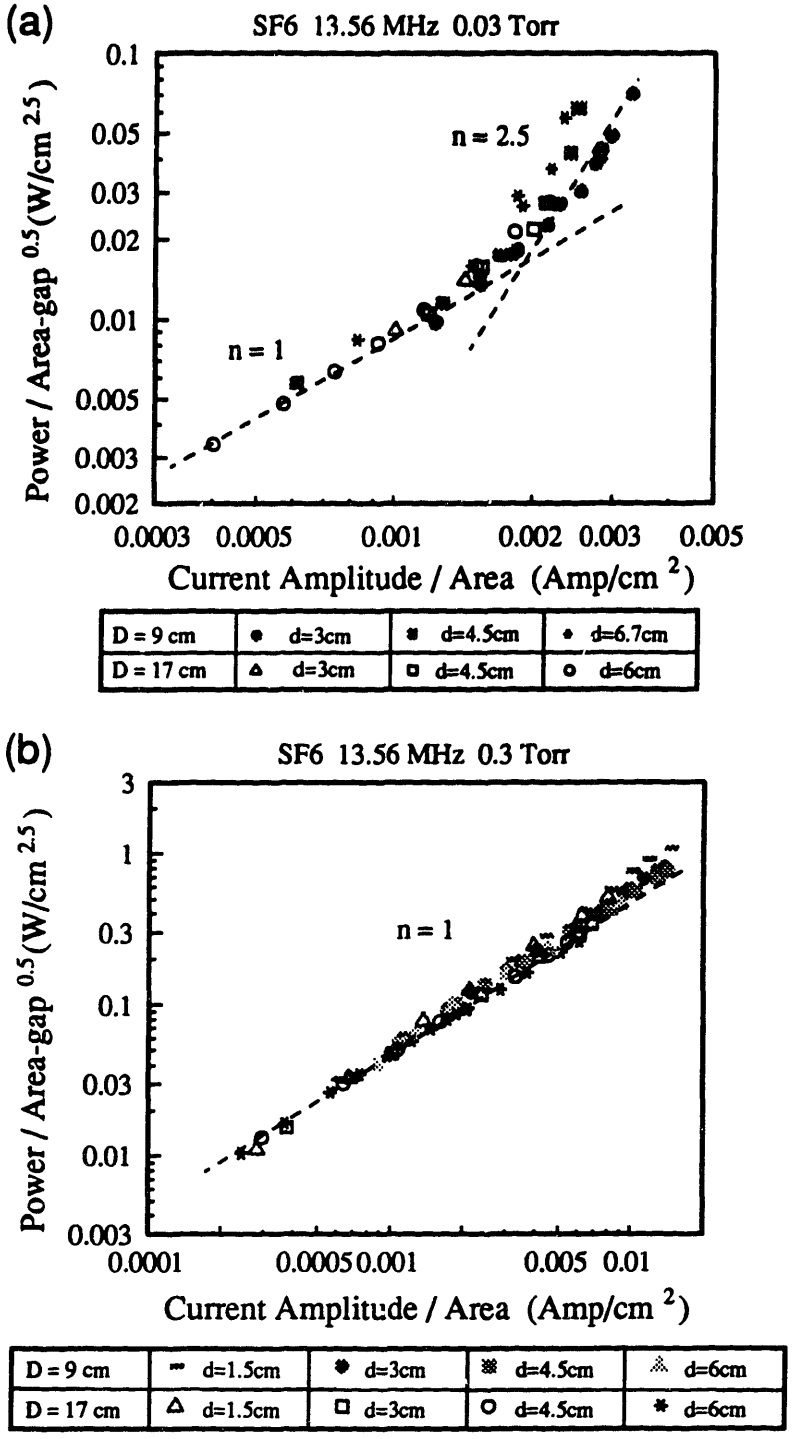


Figure 3.14: SF6 data showing how power varies with plasma dimensions at (a) 0.03 Torr and (b) 0.3 Torr for both bulk and sheath power dominated regimes.

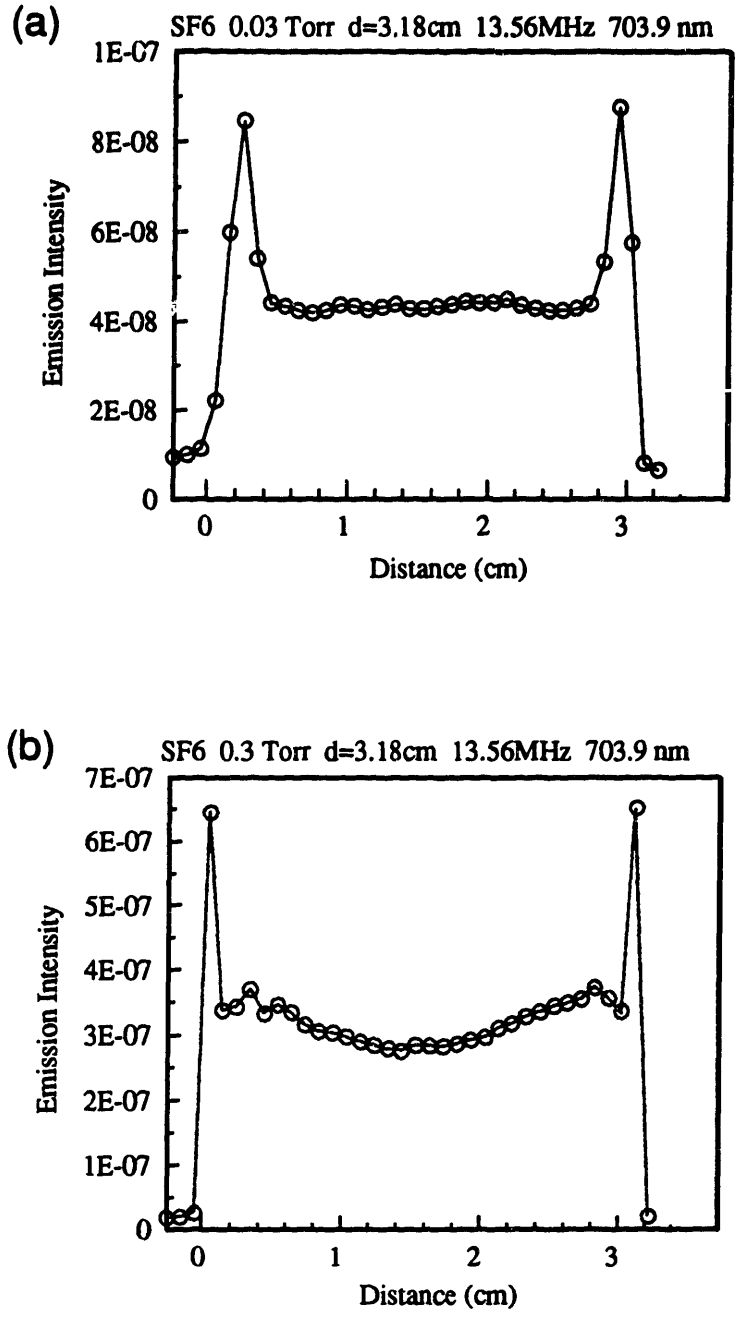


Figure 3.15: Spatial emission scans between the powered and grounded electrodes of the F atom 703.9 nm line in the SF6 plasma at (a) 0.03 Torr and (b) 0.3 Torr.

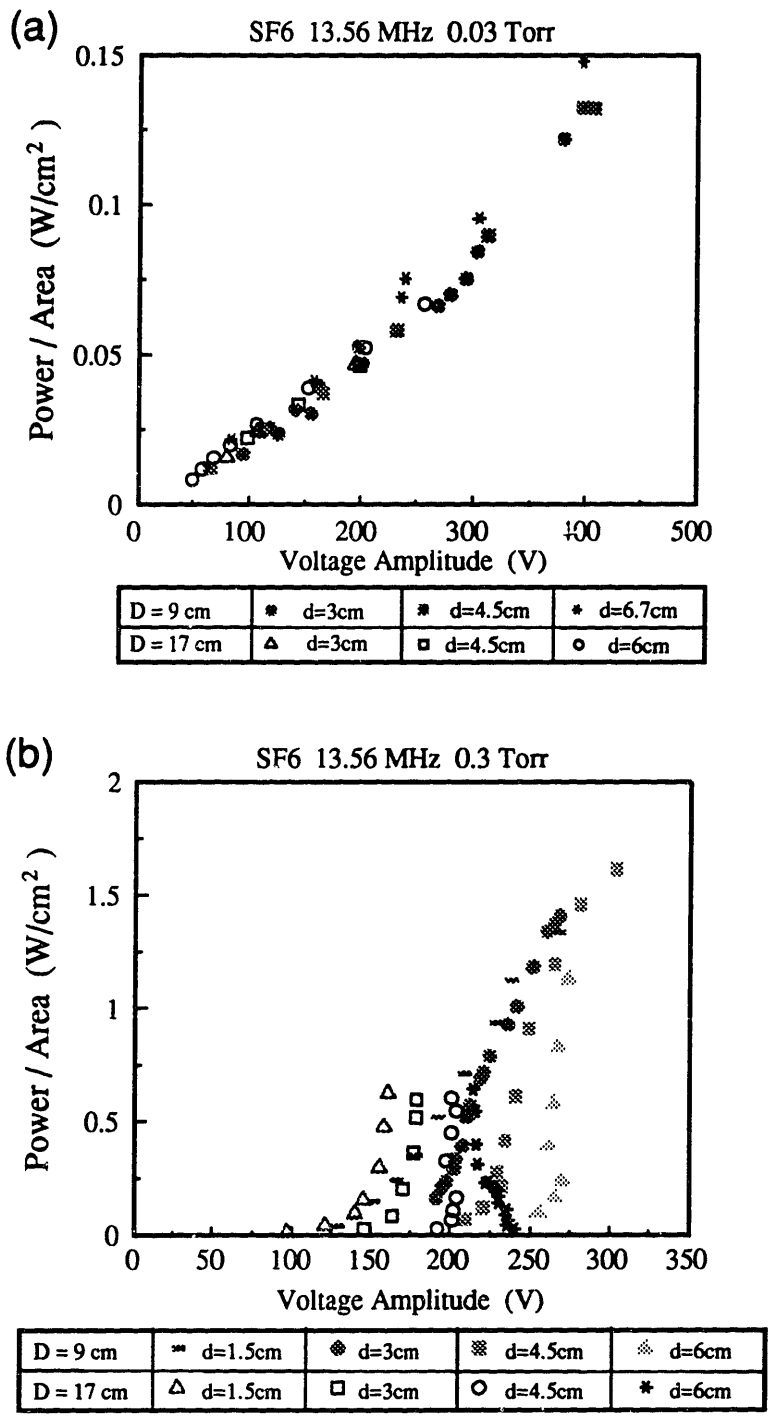


Figure 3.16: Power and voltage relationships in SF6 plasmas at (a) 0.03 Torr and (b) 0.3 Torr.

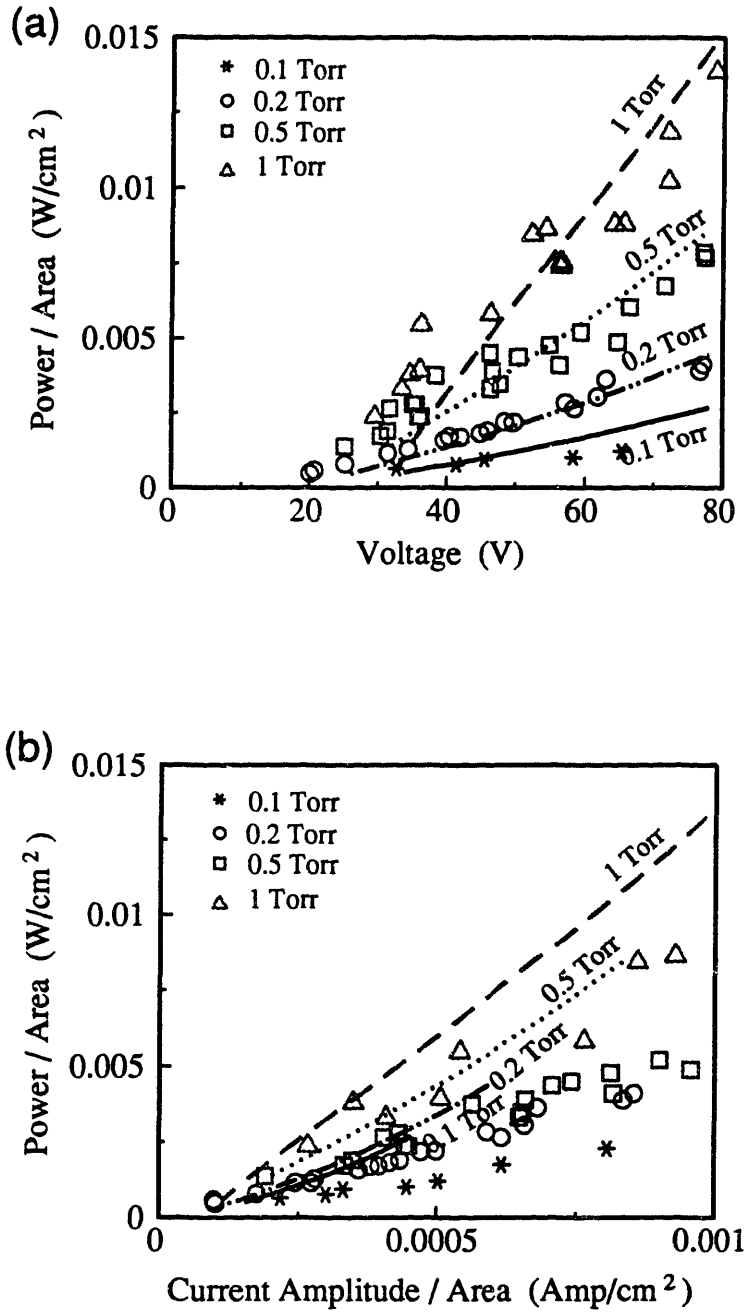


Figure 3.17: Comparison between data and continuum model results of plasma electrical properties of 13.56 MHz argon plasmas. Symbols are data and lines are model results.

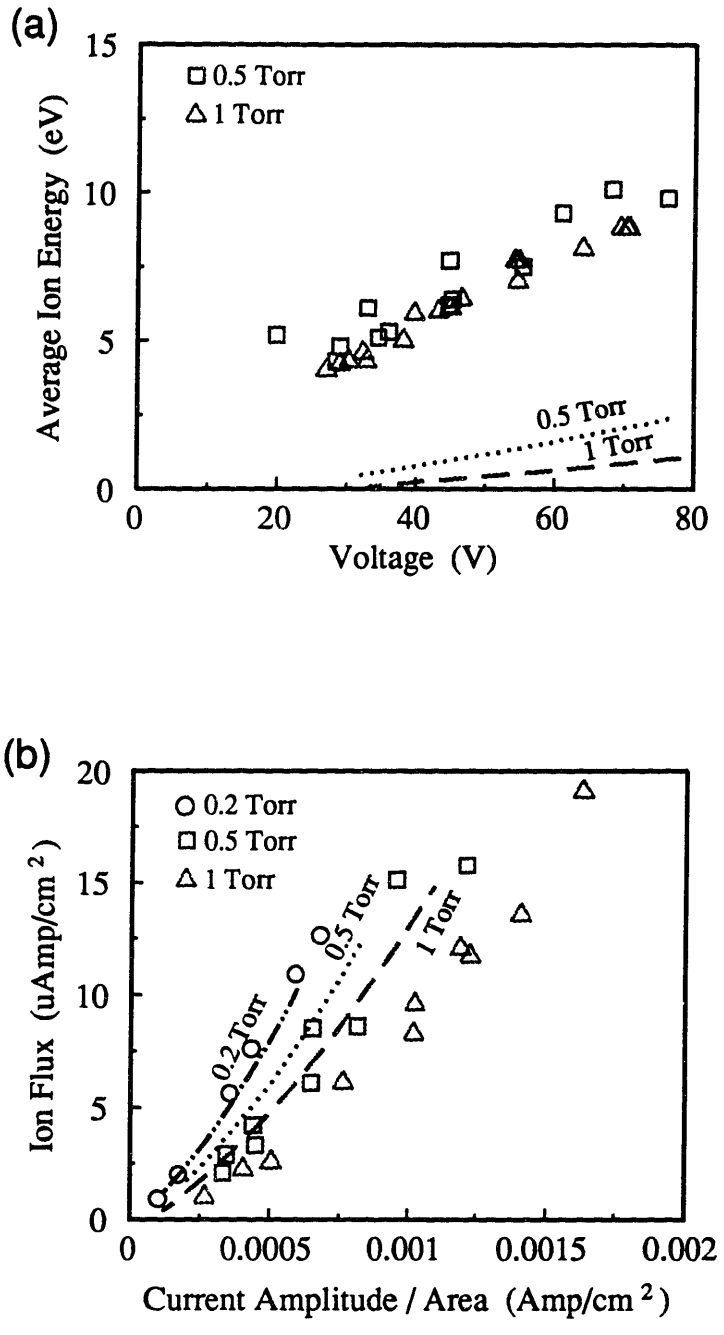
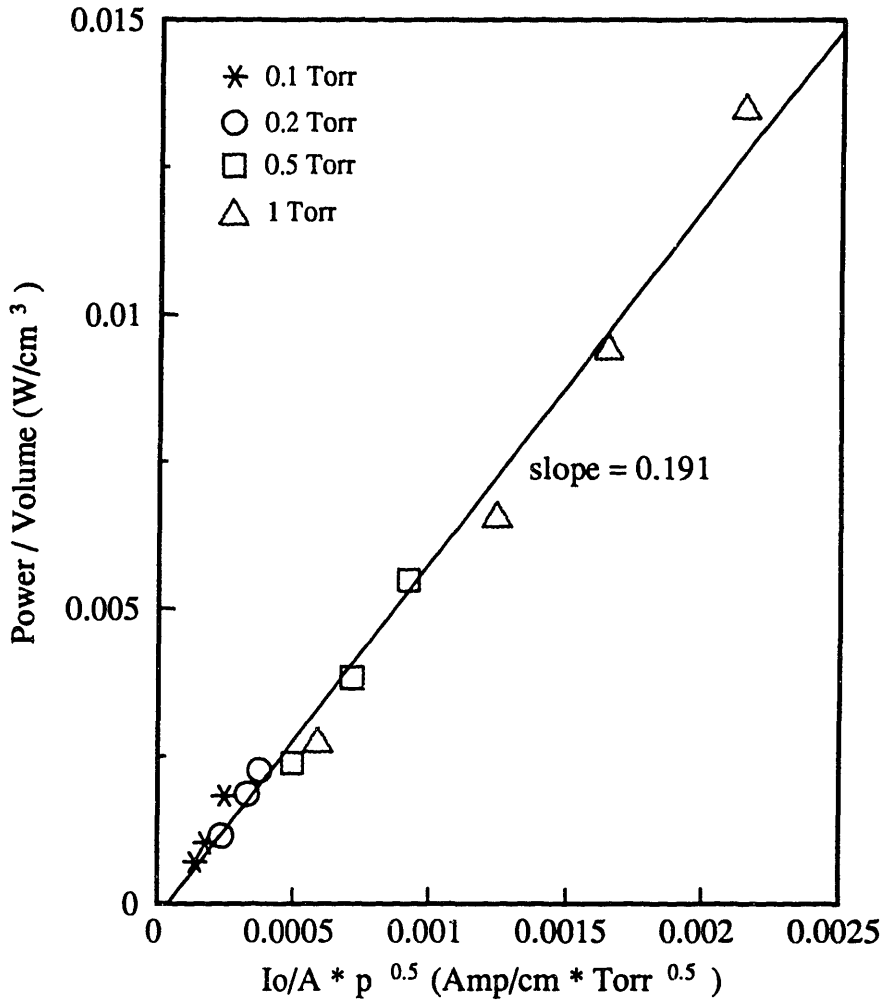


Figure 3.18: Comparison between data and continuum model results of plasma ion bombardment properties of 13.56 MHz argon plasmas. Symbols are data and lines are model results.



Note that experimental data gives a slope of 0.126

Figure 3.19: Graph showing that the continuum modeling data follows proposed scaling relationship for bulk power deposition. The points are generated with a continuum model of 13.56 MHz argon plasmas, at 0.1, 0.2, 0.5, and 1 Torr. Linear regression of these points is also graphed.

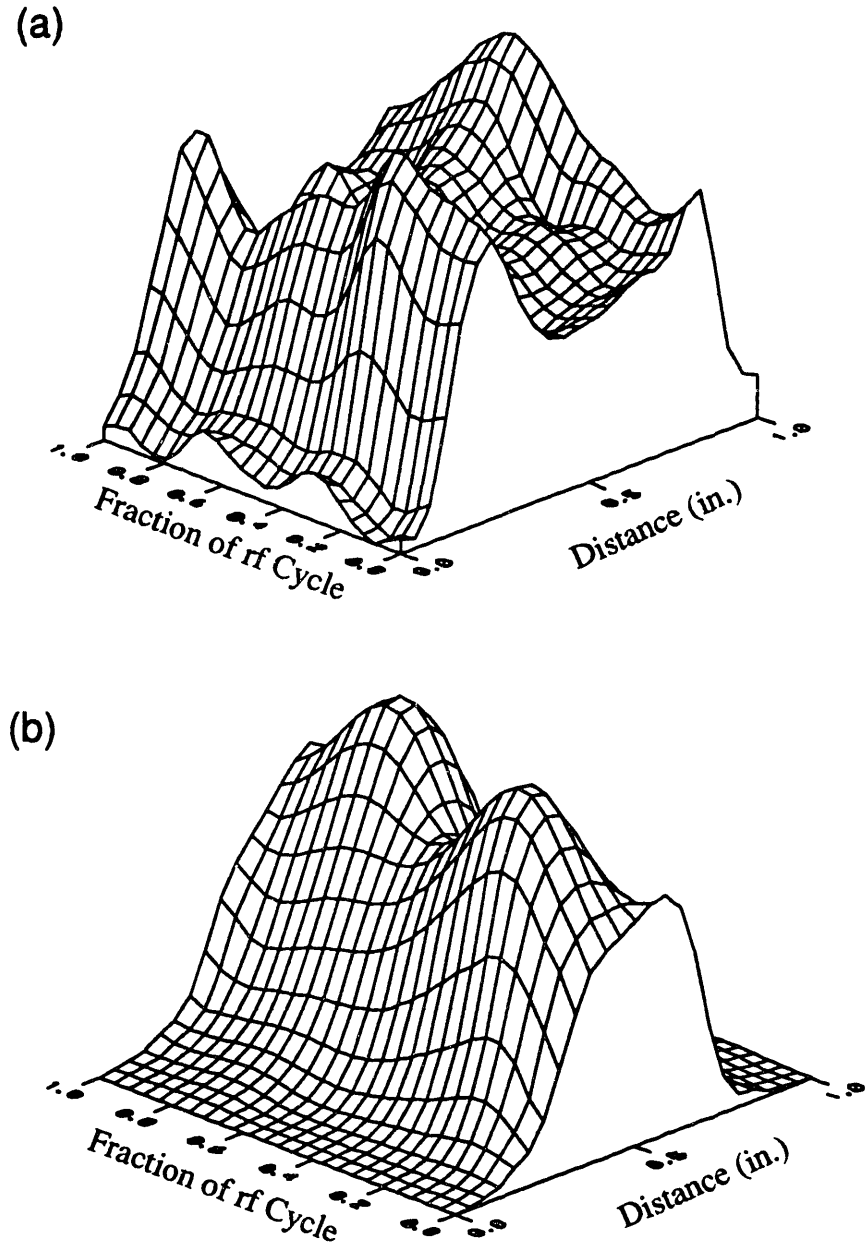


Figure 3.20: Comparison of (a) measured plasma emission against (b) continuum model prediction of ionization rate in a 13.56 MHz, 0.1 Torr argon plasma.

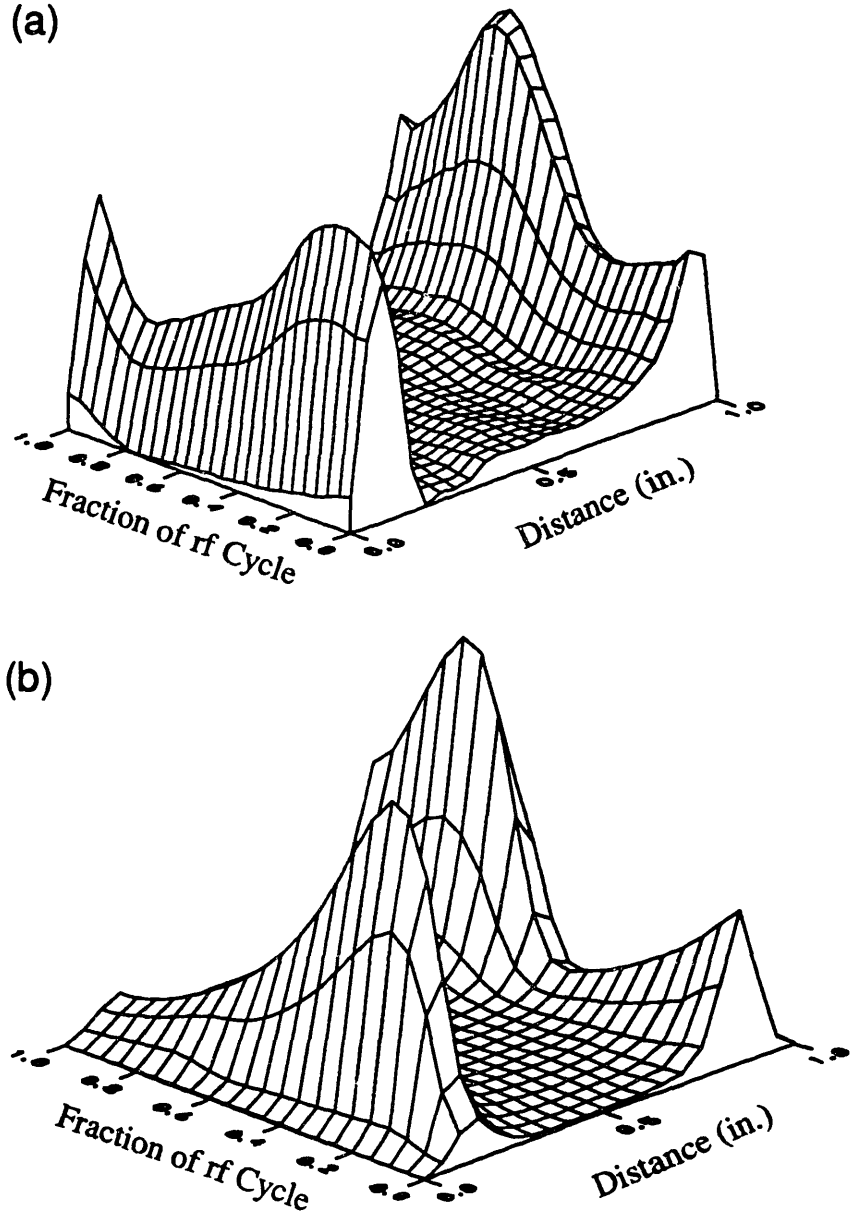


Figure 3.21: Comparison of (a) measured plasma emission against (b) continuum prediction of ionization rate in a 13.56 MHz, 1 Torr argon plasma.

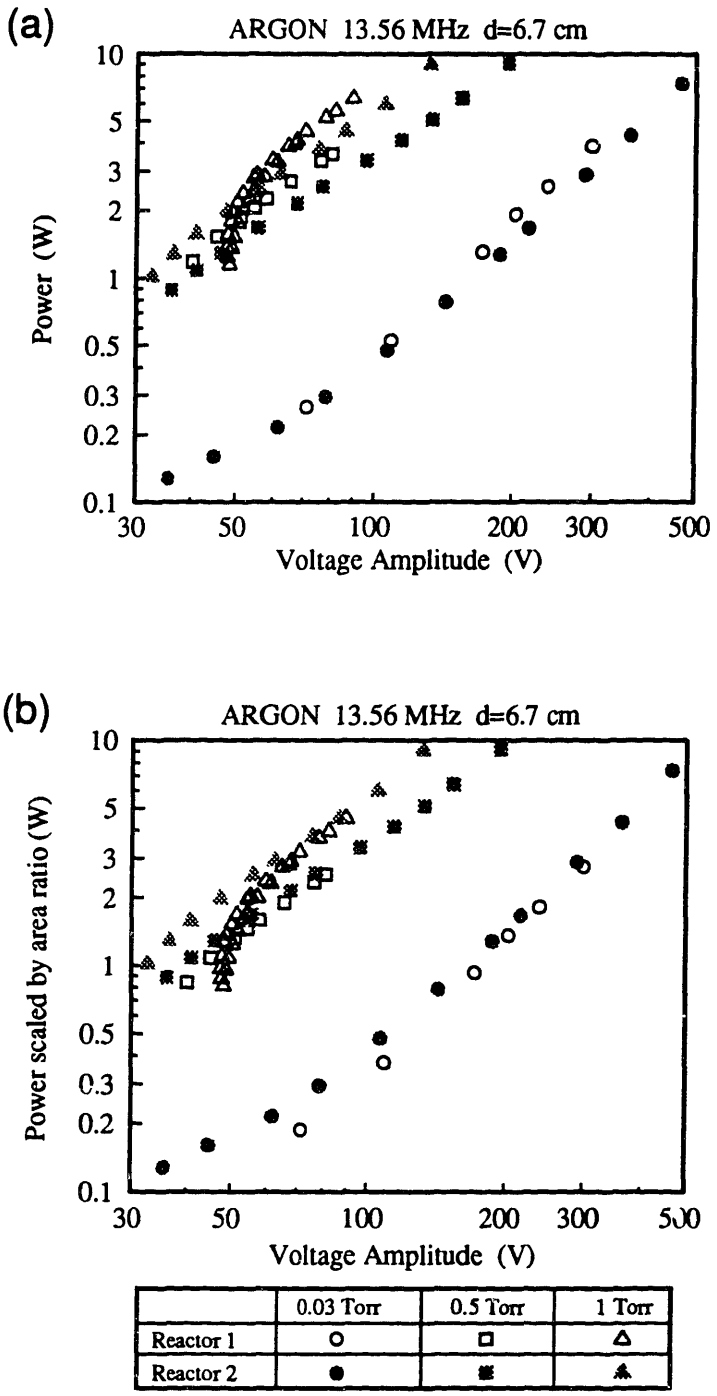


Figure 3.22: Comparison of power measurements between our data and Godyak et al.'s (1991b) data measured with an electrode gap spacing of 6.7 cm.
(a) Original data and (b) our data scaled by the ratio of different plasma areas between the two reactors.

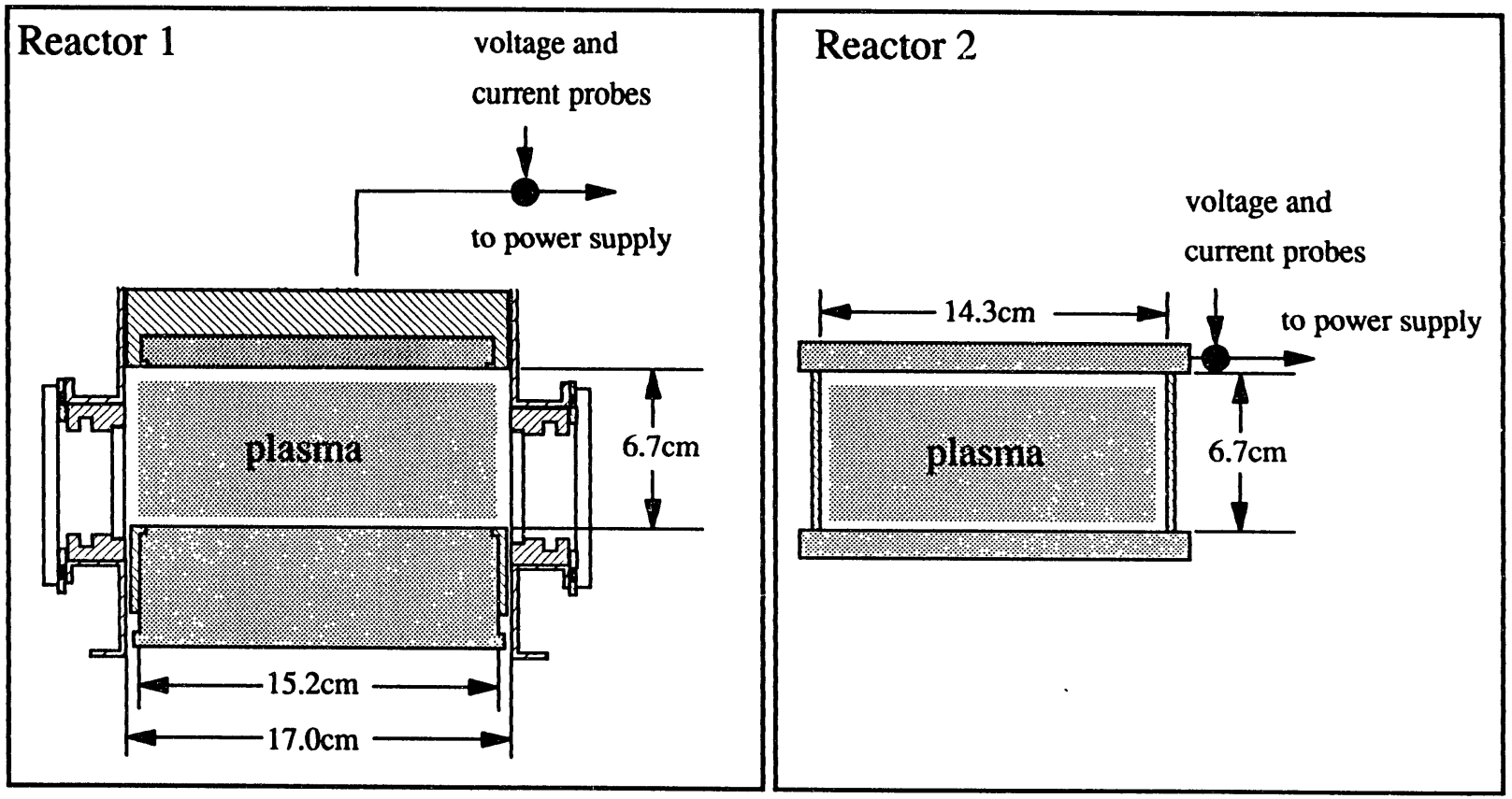


Figure 3.23: Schematic of the two reactors, showing the differences and similarities between them. Reactor 1 = the reactor used in these experiments. Reactor 2 = the reactor used by Godyak et al. (1991b).

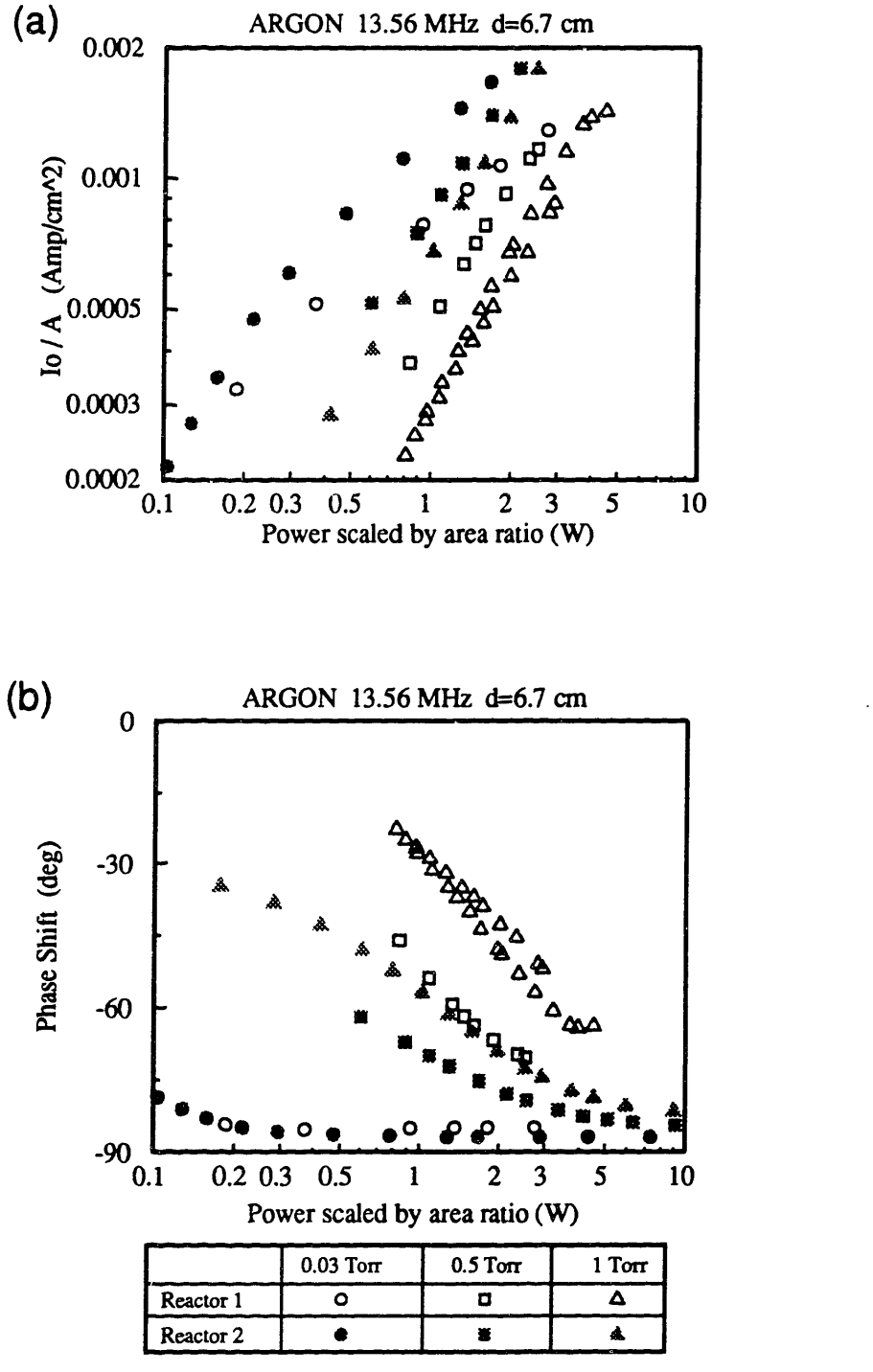
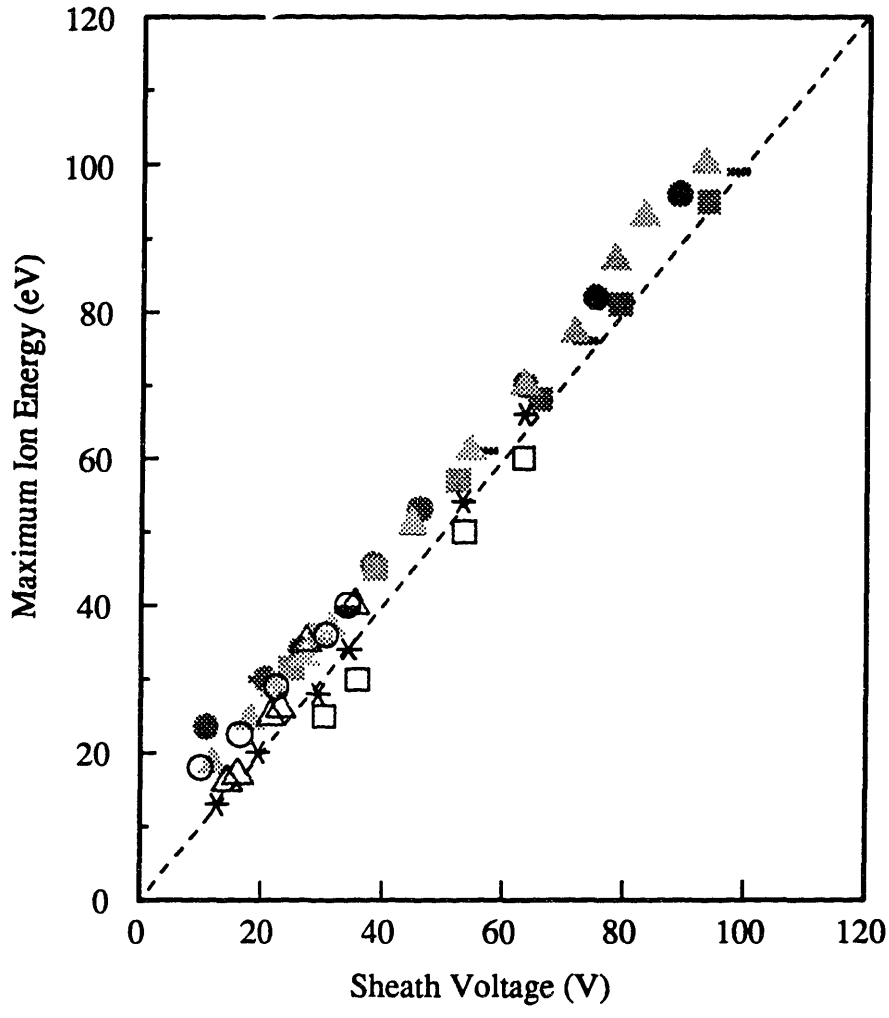


Figure 3.24: Comparison of (a) current and (b) phase shift between our data and Godyak et al.'s (1991b) data measured with an electrode gap spacing of 6.7 cm.



	10 mTorr	50 mTorr	500 mTorr	1 Torr
D = 9 cm	d=2"	d=2"	* d=0.8"	□d=0.8"
D = 17 cm	● d=2"	▲ d=1"	○ d=2"	△ d=1"

Figure 3.25: Comparison of measured maximum ion energy at various pressures and the sheath voltage calculated from voltage measurements and using the electrical analog model in argon plasmas.

CHAPTER 4

ION BOMBARDMENT ENERGY, ANGLE, AND FLUX

This chapter focusses on the ion bombardment properties on the electrode surface in 13.56 MHz argon plasmas. Although argon is not a major industrial etch gas, its simple plasma chemistry (electrons, Ar atoms, and Ar^+ species) makes it an appropriate starting gas for ion bombardment studies. The operating parameters varied in this study were pressure and applied voltage across the electrodes. Measured ion energy and angle distributions are compared with Monte Carlo simulation results in the next chapter.

4.1 REVIEW OF PREVIOUS WORK

The energy and angle of an ion incident on the electrode are determined by the electric field in the sheath, the sheath width to mean-free-path ratio, and the number of radio frequency (rf) cycles required for an ion to cross the sheath. These parameters are determined by operating conditions such as power, frequency, and pressure. With the exception of Thompson *et al.* (1986), who measured the energy of ions incident over a wide collection angle, only ion energy distributions (IEDs) of ions incident near normal to the electrode surface have been measured. However, previous workers, even Thompson *et al.*, measured only the ion energy component perpendicular to the surface rather than the total ion energy. At the time this thesis was written, there has been no previous measurement of ion bombardment angle distributions (IADs).

4.1.1 Collisionless and Collisional dc Sheath

In a direct current (dc) plasma, the ion energy depends on the magnitude of the electric field the ion travels through before striking a surface. The maximum energy an ion can have, if

it does not lose energy through collisions, is approximately equal to the applied potential between the electrodes. Since there are usually collisions in the sheath, a distribution of ion energy forms with a shape that depends on the type and probability of collisions. Charge-exchange collisions can be a major energy loss mechanism, especially in plasmas where symmetric ion-neutral charge transfer occurs. Davis and Vanderslice (1963) observed that the IEDs were skewed toward higher energies when ions had few or no collisions in the sheath, while operating conditions which favored charge-exchange collisions produced IEDs skewed toward low energy. In cases where the ions have multiple collisions, there were no ions with energy equal to the applied potential across the electrodes.

For abnormal dc plasmas, the product of pressure and sheath width, $p\ell_{sh}$, is approximately constant for a fixed potential drop across the electrodes (Davis & Vanderslice 1963, von Engel 1983); increasing pressure does not increase the number of ion-neutral collisions in the sheath. Doughty *et al.* (1987) showed that in 3.5 Torr Helium plasmas, as the applied voltage increases, the sheath width initially decreases, then increases. Therefore, the effect of operating conditions on the IEDs is best described by quantities such as the number of mean free paths in the sheath or the sheath potential drop rather than the pressure or applied voltage.

4.1.2 Collisionless rf Sheath

When a radio frequency (rf) voltage is applied across the electrodes, most of the voltage drop occurs in the sheath, with very little voltage drop across the plasma bulk. The plasma potential in the bulk oscillates at the applied frequency, as illustrated in Figure 4.1. The sheath electric field that accelerates ions toward the surface is just the difference between the plasma potential and the electrode potential across a distance, ℓ_s, where ℓ_s is the sheath width. Therefore, the electric field accelerating ions toward the grounded electrode is the plasma potential minus 0

V and the electric field at the powered electrode is the plasma potential minus the applied potential. The number of rf cycles the ions take to cross the sheath determines whether their final energies are modulated by the time varying sheath electric field.

In a collisionless sheath, the ion transit time or the number of rf cycles an ion takes to cross the sheath, defines the width of the ion energy distribution. If the ion transit time is a fraction of an rf cycle, the ion's energy will depend on the phase of the rf cycle when it enters the sheath. The maximum and minimum energies occur when the ion crosses the sheath during the maximum and minimum sheath potentials. If an ion takes many rf cycles to cross the sheath, its energy will depend on the average sheath potential. Modulation of ion energy is still apparent in cases where the ion transit time is about five rf cycles (Ulacia & McVittie 1989). Okamoto and Tamagawa (1970) derived an expression for the width of the IED or energy dispersion, $\Delta\varepsilon$, using the following assumptions: the sheath has a field determined by space charge, the mean free path is much greater than the Debye length in the sheath, the ion energy at the bulk-sheath interface is negligible, and the ions have no collisions within the sheath. They predicted that $\Delta\varepsilon$ follows:

$$\Delta\varepsilon \propto \frac{1}{\omega}\left(\frac{q}{m}\right)^{1/2}\frac{V_s}{\ell_s} \quad , \tag{4.1}$$

where m is the ion mass, q is the ion charge, ω is the angular excitation frequency, V_s is the magnitude of the time varying component of the sheath voltage, and ℓ_s is the sheath width. From this expression, one can see that higher frequencies, heavier ions, and wider sheaths, all of which increases the ratio of ion transit time to the rf period, should cause $\Delta\varepsilon$ to decrease. Okamoto and Tamagawa experimentally verified that $\Delta\varepsilon$ does scale inversely with frequency from 20 MHz to 80 MHz and decreases with increasing ion mass. However, the range of ion mass studied was not wide enough to clearly show the $m^{-1/2}$ dependence.

Extending the range of ion mass studied by Okamoto and Tamagawa and using a 13.56 Mhz plasma, Coburn and Kay (1972) showed that even at frequencies where the ion transit time is the same order of magnitude as the rf period, $\Delta\varepsilon$ scales with $m^{-1/2}$. However, deviations from the $m^{1/2}$ relationship were seen at high ion masses.

Under operating conditions where the ion transit time is comparable to the rf period, the IEDs are double peaked, with one peak at high energy and one peak at low energy (Thompson *et al.* 1988, Coburn & Kay 1972, Tsui 1968, Kushner 1985, Kuypers &Hopman 1988). Tsui (1968) used macroscopic energy balances, and Thompson *et al.* (1988) and Kushner (1985) used Monte Carlo models, to predict the bimodal IEDs and the decrease in $\Delta\varepsilon$ with increasing ion transit time. The ion energy balance showed that the shape of the IED from a collisionless sheath depends on only the scaling parameter, $(qV_o)/(m\ell_{sh}\omega)$ (Tsui 1968, Kushner 1985, Greene *et al.* 1988).

In low frequency plasmas, *e.g.* 100 KHz, the relative heights of the double peaks in the IEDs depend on the shape of the sheath potential as a function of time. At low frequencies (Kohler *et al.* 1985, Metze *et al.* 1986), the sheath potential is not a temporally symmetric sine wave. Figure 4.1 (b) illustrates the plasma potential drawn with respect to the potential at the surface where ion energy is measured. The difference between the plasma potential and 0 V is the sheath potential. Since the ion transit time is short with respect to the rf cycle, the time the sheath potential is at a minimum relative to the maximum determines the peak heights in the IED. Generally, the sheath potential measured at the larger electrode has a minimum value for a longer time than a maximum value. Metze *et al.* (1989) showed this effect by calculating the sheath potential at the larger surface of an asymmetric electrode configuration. They obtained bimodal IEDs with a larger peak at the low energy end of the spectrum. ArH^+ IEDs measured by Kohler *et al.* (1985) for a 100 kHz plasma also showed a significantly larger peak at low energy

compared to high energy.

For a capacitive plasma at higher frequencies, *e.g.* 13.56 MHz, the sheath potential should vary approximately sinusoidally with time. In this situation, the relative heights of the high and low energy peaks in the IED will depend on the ion transit time relative to the rf cycle time (Thompson *et al.* 1988). Coburn and Kay (1972) measured the IEDs of several ions in a 13.56 MHz argon plasma at the larger electrode. All their IEDs showed a significantly larger peak at the low energy end of the spectrum. Kuypers and Hopman (1988) measured the IED for an argon and an oxygen plasma in a cylindrical reactor at the smaller, powered electrode. The low energy peak is slightly larger than the high energy peak in their argon IED and vice-versa for the peaks in their O_2^+ and O^+ IEDs.

The effect of rf modulation on the ion energy can be seen in the maximum ion bombardment energy. Ion bombardment energy measurements of the ArH^+ ion from a 0.05 Torr, 2.7 MHz argon plasma (Kohler *et al.* 1985) showed that an ion can gain a maximum of 79% of the available sheath potential at this frequency. This indicates that the ion transit time is more than an rf cycle. The ArH^+ energy is a good measure of the maximum, collisionless ion energy because the probability of charge-exchange between ArH^+ and an Ar atom is much smaller than between symmetric ion-neutral charge-exchange. As frequency increases, the ion no longer sees the maximum sheath potential but responds only to the time averaged sheath potential. A graph of maximum ion energy as a function of frequency exhibits a flat region at low frequencies where the ion crosses the sheath in less than an rf period. In this region, the maximum ion energy equals the maximum sheath potential energy. As the operating frequency increases, the maximum ion energy goes through a transition region where the ion energy decreases toward the average sheath potential. This is followed by another flat region where the ion energy equals the average sheath potential (Bruce 1981).

4.1.3 Collisional rf Sheath

Ion-neutral collisions in the sheath, which result in momentum and energy transfer, can significantly alter the IEDs from the collisionless case. Most of the previously measured IEDs from collisional sheaths have been from argon plasmas. The two major collisional processes between argon ions and neutrals are charge-exchange and elastic collisions which have comparable cross sections (Cramer 1959). In charge-exchange collisions, the original ion becomes a neutral but maintains its initial energy before collision, and the original neutral becomes an ion with almost no energy. The operating pressure and the sheath width determine the number of mean free paths in the sheath. If pressure increases and the sheath width remains about the same, the number of collisions increases, lowering the ion energies and randomizing their direction. The IEDs and IADs are fully developed when the shape of the distribution functions no longer change with additional collisions. The number of collisions required to produce a fully developed distribution varies for different collision processes and sheath fields. For elastic processes, an average of three scattering events is sufficient to produce fully developed distributions when the field is dc and spatially uniform, while six scattering events are required for a spatially linear dc electric field (Thompson 1988).

Broadening of the IED can be attributed both to rf modulation and to collisions in the sheath. To separate these two effects, Greene *et al.* (1988) compared the ArH^+ and Ar^+ IEDs from the same argon plasmas. They showed that for ArH^+, which goes through essentially no collisions traveling through the sheath, the $\Delta\varepsilon$ of the IED increased with increasing pressure. This change is due to decreasing sheath width with increasing pressure, resulting in fewer rf periods needed for the ion to cross the sheath. Under the same operating conditions, the Ar^+ IED has a much larger $\Delta\varepsilon$ than that of ArH^+, with multiple low energy peaks in the IED. Therefore, collisions in the sheath broaden the IEDs. One would expect random collisions in the sheath to lower the ion

energy, but not necessarily produce the peaks in the IEDs that Greene *et al.* observed.

Multiple peaks in the IED can occur in rf modulated sheath fields for ions that undergo charge-exchange collisions. In a charge-exchange collision process, an ion transfers charge and a small amount of momentum to a neutral, forming a new ion that has little directed energy perpendicular to the electrode and an energetic neutral from the original ion. For plasmas where the probability for charge-exchange collisions is high, these energetic neutrals can contribute a significant portion of the total energy deposited on the surface (Jurgensen 1988). Measurements of the argon ion energy incident near normal to the electrode by Wild and Koidl (1989), for 13.56 MHz plasmas between 0.0075 and 0.0225 Torr, showed distinct peaks spread throughout the energy spectra at discrete energy intervals. They attributed these peaks to the phase of the rf cycle and the position in the sheath where new ions are formed via ion-neutral charge-exchange collisions. Using a model which depends only on the parameter η, where

$$\eta = \frac{q\, V_{dc}}{m\, \omega^2\, \ell_s^2} \quad , \tag{4.2}$$

and assuming a constant collision cross section, a uniform rate of ion creation with distance in the sheath and with time in the rf cycle, and a sheath electric field of the form $\mathcal{E}_s \propto x^{\nu}[1 - \cos \omega t]$, Wild and Koidl were able to reproduce the multiple peaks by fitting ν and η to their data. V_{dc}, m, ν, and x are the dc bias across the electrodes, mass of the ions, electric field spatial dependence, and distance into the sheath, respectively. They obtained $\nu = 0.55 - 0.7$ for the electric field profile in the sheath.

Ion energy distributions for plasmas other than argon have been studied by Thompson *et al.* (1986). They observed that the average ion energy in the high pressure regime (0.2 to 1 Torr) of SF_6, CF_3Cl, and CF_3Br plasmas increased with power and decreased with increasing pressure. Since there are multiple collisions at these pressures, Thompson *et al.* coupled a plasma

impedance model for the sheath electrical properties with an expression for the ion velocity of a fully-developed velocity distribution to give a scaling group for the average ion energy, ε. The expression is:

$$\varepsilon \propto \frac{I_o}{\omega A p} \quad , \tag{4.3}$$

where I_o is the current amplitude across the electrodes, A is the electrode area, and p is the operating pressure. Thompson *et al.*'s measured average ion energy at various pressures, frequencies, and electrode spacings, scaled well with $I_o/(\omega Ap)$. The scaling factor proposed by Zarowin (1983), $\varepsilon \propto \text{Power}^{0.5}/(\omega p)$, was shown to work well at constant frequency, but failed when frequency was varied.

The average ion incident angle (*i.e.* the angle measured from the surface normal) increases with the number of collisions in the sheath until the distribution becomes fully developed. At this point the average incident angle remains constant with increasing sheath length to mean free path ratio. The thermal energy of the ion at the bulk sheath interface is usually small compared to the sheath potential. If ions have no collisions in the sheath, they will have their velocity directed completely perpendicular to the surface. In situations where there are only charge-exchange collisions and the sheath potential is small (~ 30 eV), the thermal velocity component could produce ions with large incident angles (Kushner 1985). In general, for capacitive plasmas where most of the applied voltage falls across the sheaths, the initial ion thermal energy is negligible and momentum transfer scattering is necessary to produce large ion incident angles. Although ions are scattered in the sheath, most of their energy is still directed toward the electrode since the sheath electric field accelerates them toward the surface. Modeling efforts have shown that the average ion energy is greater at small incident angles than at large incident angles (Ulacia & McVittie 1989, Jurgensen 1988).

4.2 EXPERIMENTAL CONDITIONS

The experimental apparatus and diagnostics used for measuring the ion properties are explained in detail in Chapter 2. This section briefly outlines some of the experimental conditions specific to the data reported here. The data are measured in a symmetric electrode configuration, consisting of two 7.6 cm or 15.2 cm diameter anodized aluminum electrodes. Teflon rings confine the plasma between the electrodes. The electrode spacing is set at 3 cm. The argon gas flow rate used is between 5 and 10 sccm. The ions striking the surface are sampled using orifices located at the center of the lower electrode. These orifices are made from a 9 μm thick aluminum sheet with an array of three 75 μm diameter holes drilled into it, while an array of nineteen orifices, 13 μm diameter and 5μm thick, are used to measure the ion flux.

4.3 RESULTS AND DISCUSSION

In an argon plasma, sheath properties determine the ion bombardment energy and angle at the surface. Argon is a capacitive plasma where most of the applied potential between the electrodes falls across the sheaths. Therefore, the ion energies are directly related to the applied voltage. At the plasma bulk-sheath interface, an ion starts with an energy approximately equal to the average plasma electron energy. The maximum energy an ion can have when it arrives at the surface is equal to the sheath potential. At 13.6 MHz and 0.01 to 0.5 Torr, an ion typically takes ten or more rf cycles to cross the sheath. Consequently, the ion energy depends more on the time averaged sheath potential than the instantaneous sheath potential. The pressures used in these experiments ranged between 0.01 and 0.5 Torr while the time varying voltage amplitude, V_o, varied from 40 V to 75 V.

The number of ion-neutral collisions in the sheath is estimated using the combined charge-exchange and elastic collision cross sections measured in swarm experiments (Cramer 1959). The

measured sheath width, ℓ_s, is 11 mm at 0.01 Torr, 8 mm at 0.05 Torr and 4 mm at 0.5 Torr. These values are consistent with those reported by Wild and Koidl (1989). The applied potential across the electrodes has no observable effect on ℓ_s, within the accuracy of the measurement technique. Kuypers and Hopman (1988) also reported a nearly constant sheath width as a function of power. Using the reported argon collision cross section (Cramer 1959) and the measured ℓ_s found in Chapter 3, the calculated sheath width to mean free path ratios, ℓ_s/λ_m, are approximately 1.5 at 0.01 Torr, 5 at 0.05 Torr, and 25 at 0.5 Torr.

4.3.1 Effect of Pressure on Ion Properties

The shapes of ion energy distributions (IEDs), integrated for all incident angles, are a functions of both pressure and V_o. IEDs at 0.01, 0.05, and 0.5 Torr and $V_o = 65$ are shown in Figures 4.2 (a), (b), and (c). The number and energy of the multiple peaks seen in the 0.01 and 0.05 Torr cases are very reproducible. The separation between each peak varies from 3 to 5 eV.

At low pressures, a significant number of ions experience no collisions when crossing the sheath. These ions contribute to the high energy peak located at the far right of the 0.01 and 0.05 Torr IEDs. The low energy peak can be attributed to ions which undergo several elastic collisions in the sheath and/or to ions which are formed in the sheath via charge-exchange collisions close to the electrode surface. The intermediate peaks in the IEDs result from a combination of charge-exchange collisions and the time varying electric field in the sheath. When a charge-exchange collision occurs between an ion and a neutral in the sheath, the result is a new ion with almost no energy and an energetic neutral. If the new ion is created when the sheath electric field is low, it stays near where it is created until the electric field increases and sweeps it toward the electrode. Ions created when the electric field is high are immediately accelerated toward the electrode. This results in ions travelling through the sheath in groups, where the energy of these groups depend

on the sheath location where the ions are created. The outcome is the multiple peaks in the IEDs.

Other workers have also reported a large fraction of low energy ions in the IEDS. Coburn (1970) reported seeing a low energy tail in the Ar^+ IED from a dc plasma which he attributed to resonance charge-transfer. The IEDs presented by Ingram and Braithwaite (1988) also had a low energy tail, but their ion detector may not have had sufficient energy resolution to resolve the multiple peaks. There are distinct peaks in Greene *et al.*'s (1988) data which they assigned to collisions in the sheath and misattributed to collisions after the ions pass through their 200 μm diameter sampling orifice. Wild and Koidl (1989) observed IED attenuation similar to the ones shown in this thesis for ions incident near normal to the surface in low pressure argon plasmas.

At 0.5 Torr, the ions go through a sufficient number of collisions to form a fully-developed IED. The ions have reached a balance between energy lost through collisions and energy gained by acceleration in the electric field. The shape of the IEDs, after there have been sufficient number of collisions to reach a fully-developed state, do not vary with pressure.

Ion angle distributions (IADs), corresponding to the same pressures and V_o as in Figure 4.2, are shown in Figure 4.3. The incident angle is defined as the angle measured from the electrode surface normal; 0° corresponds to an ion incident perpendicular to the surface. The values on the x-axis are the average collection angle of each annular ring. Figure 4.3 shows that a majority of the ions have incident angles less than 10° at low pressures. At 0.5 Torr, the ions are roughly evenly distributed between incident angles of 0° to 40°.

The average incident energy and angle at V_o = 65 are summarized in Figure 4.4. As pressure decreases, an ion gains more energy accelerating through the sheath electric field before it collides with a neutral. This is seen in the sharp increase in average ion energy and decrease in incident angle as pressure decreases from 0.1 to 0.01 Torr. However, as pressure increases above 0.1 Torr, both the average ion incident energy and angle approach linear dependence on

pressure because the number of collisions in the sheath approaches the number of collisions needed for a fully-developed energy and angle distribution.

All the IEDs shown so far have combined all the ions incident on the ion detector. The ion energy was also measured separately as a function of incident angles. Four IEDs, measured at 0.01 Torr, are shown in Figure 4.5. They represent the number of ions and their energy at 2.3°, 11.3°, 20.3°, and 29.3° incident angles. Ions with near normal incident angles are represented by the curve labeled 2.3°. These are the ions with incident angles between 0° and 4.5°. There is a large peak at the high energy end of the IED, with some ions striking at lower energies. The high energy peak represents those ions which cross the sheath without collisions and have energy equal to the average sheath potential. The lower energy ions are probably created in the sheath through charge-exchange collisions or multiple collision events. For example, an ion created from a charge-exchange collision just above the electrode has an energy equal to the neutral's thermal energy. On the other hand, an ion created at the bulk-sheath interface of the plasma, which undergoes no collisions, accelerates through the entire sheath electric field and gains energy nearly equal to the time averaged sheath potential. Ions created in the sheath via charge-exchange collisions have small incident angles and have energies between 0 eV and the maximum sheath potential. Ions which have gone through a single elastic collisions can have large incident angles and lower energies than those incident near normal to the surface with no collisions, as seen in Figure 4.5. The trajectories of ions with low energies and small incident angles tend to be deflected by the distorted field lines at the orifice. However, their energies should not be altered. (See Appendix C). This may explain why there are a large number of ions with low energies at high incident angles.

The IEDs at various incident angles for a 0.05 Torr argon plasma are shown in Figure 4.6. For ions incident at and near normal to the surface (the 2.3° line in Figure 4.6) the energy ranges

from the maximum energy to near zero energy. The high energy ions crossed the sheath without experiencing a collision. As in the 0.01 Torr case, the lower ion energies result primarily from ions created by charge-exchange with energies corresponding to the sheath position at which they were formed. Ions which have gone through several elastic collisions will have lower energies and larger incident angles, as shown in the 29.3° line in Figure 4.6. The energy distributions of ions with intermediate incident angles have multiple peaks. If charge-exchange collisions were solely an exchange of charge between an energetic ion and a neutral, with no transfer of momentum, then these multiple peaks should be small in IEDs measured at off normal incident angles. However, the IEDs presented in Figure 4.6 show large multiple peaks. Therefore, momentum transfer must also occur. Hasted (1979) discusses the probability of charge-transfer for small impact parameters where momentum transfer is present.

At 0.5 Torr, the ions undergo a large number of collisions in the sheath. Figure 4.7 shows that, as expected for fully-developed distributions, the IEDs are fairly isotropic over all the incident angles; that is, the shape and energies of the ions are independent of incident angle.

The average ion energy at each incident angle for various operating pressures is shown in Figure 4.8. For 0.01 and 0.05 Torr, the ions incident with small angles have the highest energy. This figure illustrates that ions which undergo more hard sphere collisions generally have large incident angles and low energies. At 0.5 Torr, the decrease in average energy with increasing incident angle is not evident, unlike the results for the lower pressure cases, again suggesting that the distributions are much more isotropic. At 0.5 Torr, there are approximately twenty-five mean free paths in the sheath. The energy distributions are thus fully-developed and do not have a strong dependence on the incident angle. In fact, ion energies are slightly lower between 0° and 10° than between 10° and 25°, although the average energies for ions at all incident angles did not change markedly. The slightly lower ion energies at low incident angles

may be influenced by the ions which have charge-exchange collisions. These ions contribute primarily to near normal incident angles and have lost more energy per collision than the elastically scattered ions.

4.3.2 Effect of V_o and reactor size on IEDs

The amplitude of the applied voltage across the electrodes has a significant effect on the width of the IEDs and the magnitude of the peaks in the IEDs at 0.05 Torr. Figures 4.9 (a), (b), and (c) show the IEDs at V_o's ranging from 40 V to 70 V. As the applied voltage across the electrodes increased, fine structure in the IED becomes more prominent. The fine structure at low V_o's are not clearly seen because of the small energy difference between peaks which the ion analyzer cannot clearly resolve.

When there are many collisions in the sheath, the IED shape becomes independent of V_o and pressure. At 0.5 Torr, ions undergo sufficient collisions in the sheath to produce fully-developed IEDs. Figure 4.10 (a), (b), and (c) show that the IED shape is relatively independent of V_o. The ions at this pressure are skewed toward the low energies; few of them arriving at the electrode with energy equal to the average sheath potential.

In argon plasmas, the average ion energy is linearly dependent on the applied voltage across the electrodes. Figure 4.11 shows graphs of average energy versus V_o at 0.01, 0.05 and 0.5 Torr. The graphs shows that the ion energy is not dependent on the reactor diameter or electrode spacing. As pressure decreases, the slope increases because the ions have fewer collisions in the sheath, and hence gain more energy for a given V_o than ions in a higher pressure plasma. The average ion energy is directly related to the potential drop per collision mean free path. Therefore, the average ion energy depends only on the number of mean free paths in the sheath, not the reactor dimensions. For all three pressures, the graphs have a positive intercept

which corresponds to some fraction of the plasma floating potential. The offset caused by the floating potential should be larger at lower pressures since it is not reduced as much as at high pressures by collisions in the sheath. Combining all the information on how the operating parameters affect the average ion energy, the following equation can be used to predict ion energy in 13.56 MHz argon plasmas:

$$\frac{\varepsilon}{V_s} = 0.14\ p^{-0.32}\ , \tag{4.4}$$

where **p** is expressed in Torr.

4.3.3 Ion Flux on the Electrode

Lieberman (1989) proposed the following equation to describe ion current, I_{ion} as a function of plasma parameters in a collisional sheath:

$$I_{ion} = 2.10\ \varepsilon_o \left(\frac{2q}{m}\right)^{1/2} \frac{V_s^{3/2} A \lambda_m^{1/2}}{\ell_s}\ . \tag{4.5}$$

If the sheath in the plasma behaves like a capacitor, that is, the displacement current is greater than the electron and ion currents through the sheaths and the displacement current is proportional to the sheath voltage, then Equation 4.5 becomes:

$$J_{ion} \equiv \frac{I_{ion}}{A} = \kappa \left(\frac{I_o}{A}\right)^{3/2}$$

$$\text{where} \qquad \kappa = 2.10 \left(\frac{2q}{m}\right)^{1/2} \frac{\lambda_m^{1/2}}{\ell_s \varepsilon_o^{1/2}\, \omega^{3/2}} \tag{4.6}$$

Figure 4.12 shows the measured argon ion flux, J_{ion}, on the electrode surface as a function of the current density through the plasma, I_o/A. The data shows that ion flux is proportional to $I_o^{3/2}$.

For a fixed I_o, or V_o since V_o is proportional to I_o, the ion flux is higher at lower pressurs. This observation is explained by Equation 4.6. As pressure increases, $\lambda_m^{1/2}$ decreases faster than the sheath width, ℓ_s, resulting in a higher κ.

4.4 CONCLUSION

In capacitive plasmas such as argon, most of the applied voltage drops across the sheath with only a small fraction lost in the plasma bulk. Therefore, the average ion bombardment energy in these plasmas increases linearly with increasing V_o. For a given V_o, the average ion energy decreases with increasing pressure because the number of collisions in the sheath increases. However, the average ion energy does not continue to decrease with increasing pressure, but saturates once there are sufficient collisions in the sheath for the energy gained from the electric field to balance the energy lost through collisions. Similarly, the average ion bombardment angle increases with increasing pressure, then saturates when there are sufficient collisions to produce fully-developed distributions.

The IED structures vary in shape both as a function of pressure and V_o. For 0.5 Torr plasmas, the IEDs are fully-developed, and thus their shape does not vary with V_o. At lower pressures such as 0.01 and 0.05 Torr, the IEDs are skewed toward higher energy. There are also multiple peaks in the IEDs of the 0.01 and 0.05 Torr plasma that are not seen in the 0.5 Torr plasma. These peaks can be attributed to a combination of charge-exchange and elastic collisions.

The measured average ion energy is proportional to the applied potential across the electrodes. At lower pressures, the fraction of the sheath potential the ion gains traveling across the sheath is greater because there are fewer collisions in the sheath. The average ion incident angle is also smaller at lower pressures, another indication of fewer collisions in the sheath.

There are a significant number of ions incident at wide angles, even at 0.01 Torr. The

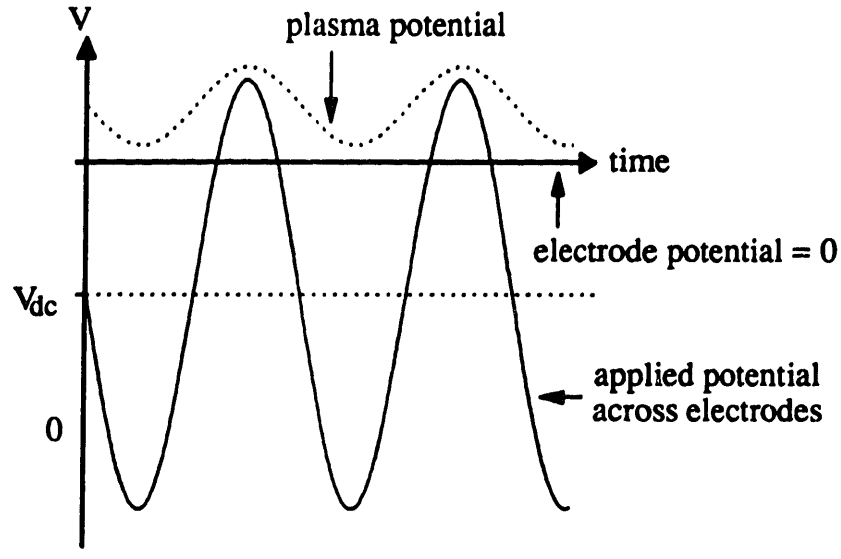
average energy of ions incident at most angles in a 0.5 Torr plasma are approximately equal since there are many collisions in the sheath which randomize the impingement angle. Although there are also ions incident at large angles in the 0.01 and 0.05 Torr plasmas, the average energy of these ions decrease rapidly with angle.

The average ion energy in 13.56 MHz plasmas is independent of the plasma geometry and depends only on the number mean free paths in the sheath and the sheath electric field. Therefore, the average ion energy can be expressed as:

4.5 NOMENCLATURE

A	electrode area
$\mathcal{E}_s$	sheath electric field
I_{ion}	ion current
I_o	current amplitude
J_{ion}	ion flux
ℓ_s	sheath width
m	ion mass
p	pressure
q	ion charge
t	time
V_{dc}	dc voltage across the electrodes
V_o	time varying voltage amplitude across the electrodes
V_s	sheath voltage
ε	average ion energy
$\Delta\varepsilon$	energy width of IEDs or energy dispersion
ε_o	permittivity of vacuum, 8.85×10^{-12} F/m
η	physical constant
λ_m	ion mean free path
ν	spatial dependence of the electrice field, x^ν
ω	angular frequency = $2\pi f$, where f=13.56 MHz

(a) Potential Lines Drawn For High Frequency Plasmas



(b) Potential Lines Drawn For Low Frequency Plasmas

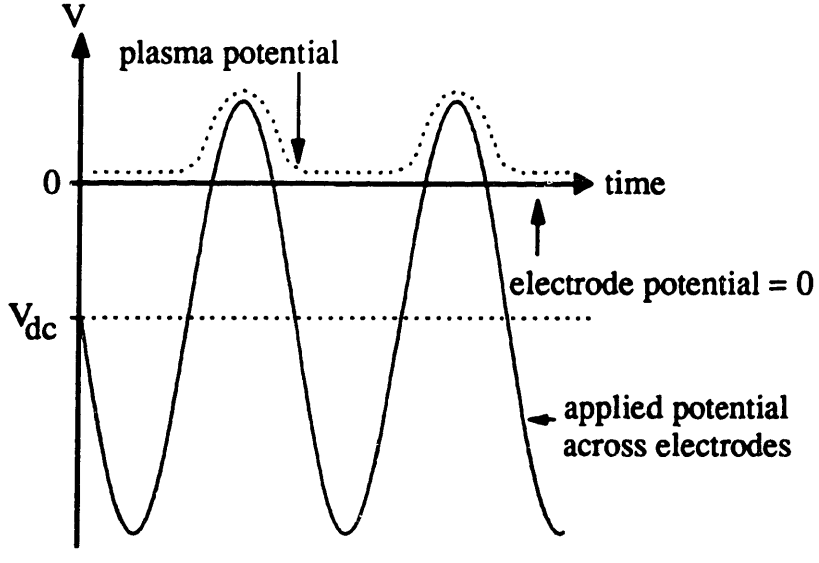


Figure 4.1: Plasma potentials in (a) high frequency (e.g. 13.56 MHz) and (b) low frequency (e.g. 100 kHz) asymmetric plasmas. The sheath potential at the large electrode is the difference between the plasma potential and the x-axis.

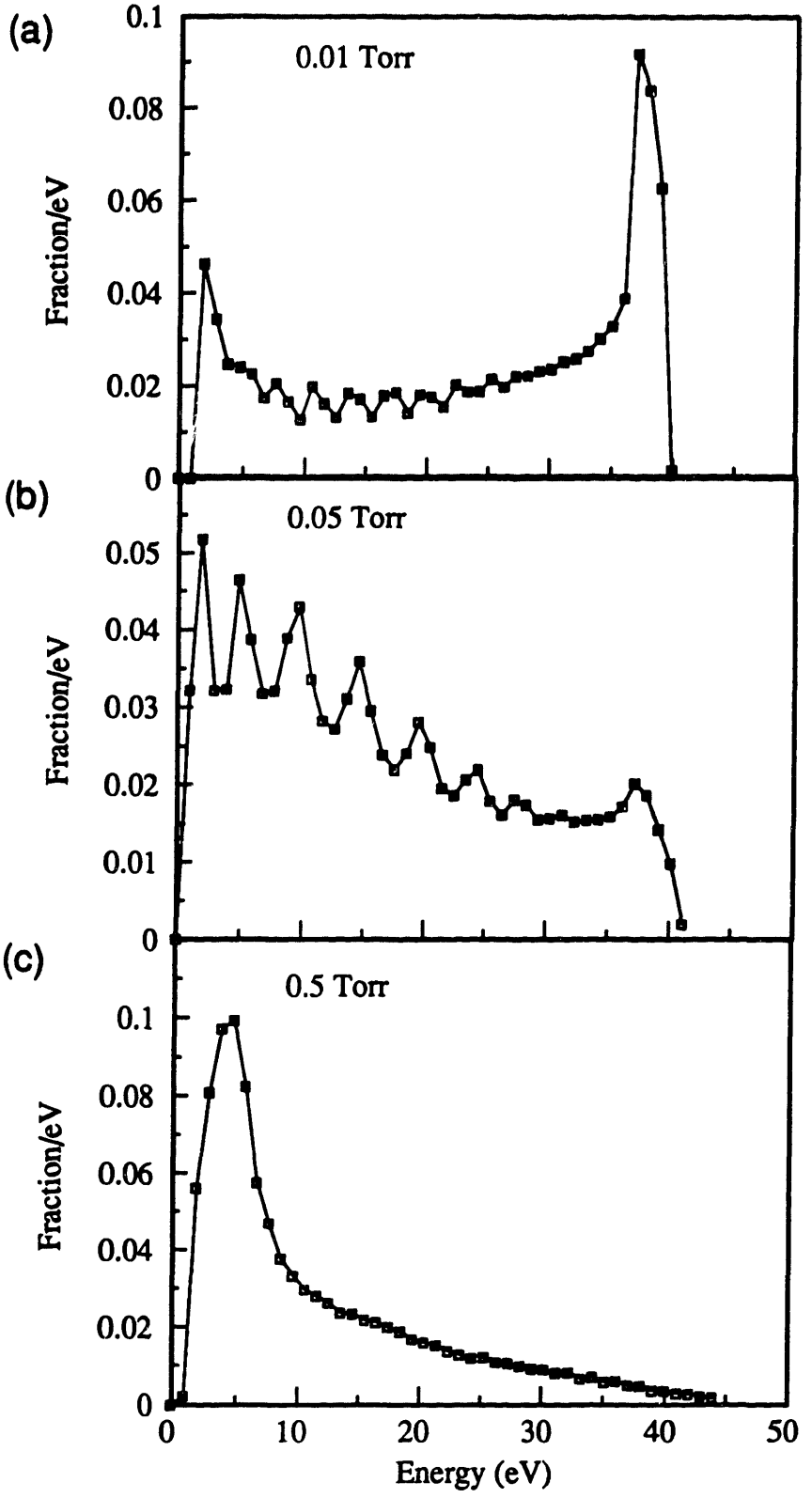


Figure 4.2: Total IED of argon plasmas at Vo=65 and various pressures. All graphs are normalized so that the area under the curves are equal.

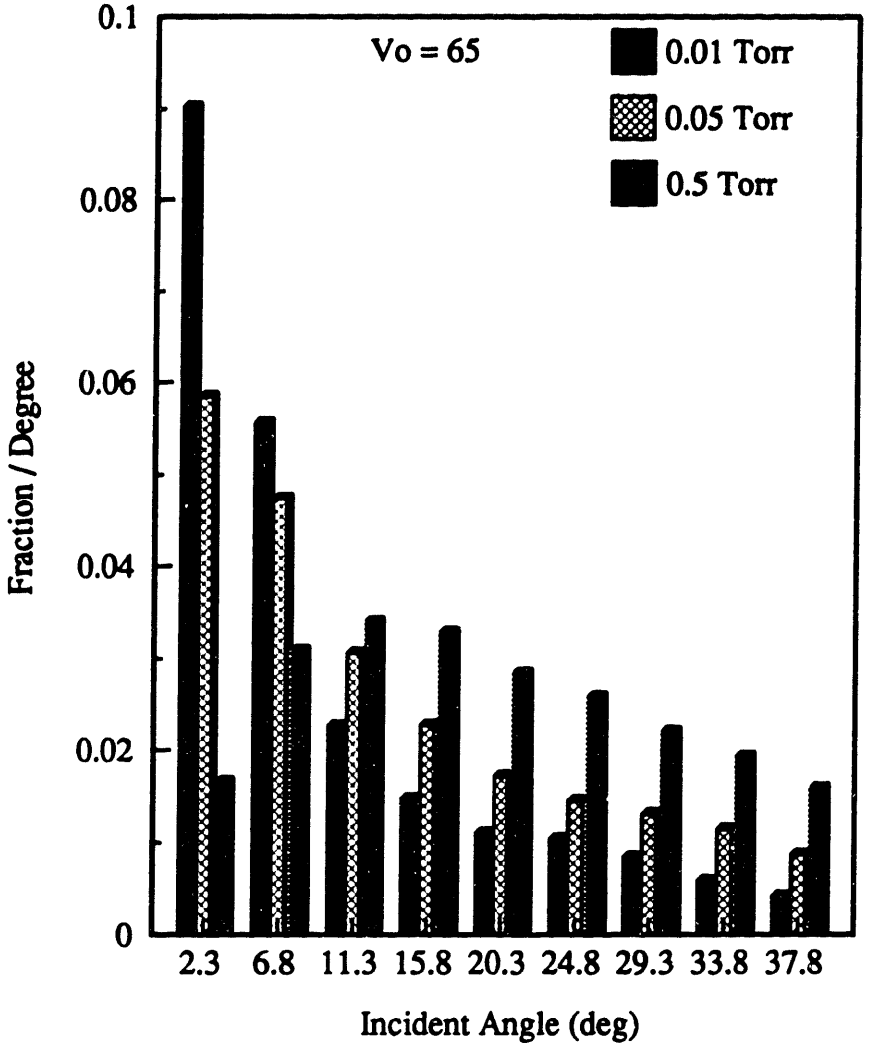


Figure 4.3: Ion angle distributions at Vo=65 for 0.01 Torr, 0.05 Torr, and 0.5 Torr, corresponding to the operating conditions of Figure 4.2. The IADs are azmuthally integrated. The x-axis is the average incident angle of the 4.5 degree angle width of the ion angle detector, with 0 degrees corresponding to normal incidence to the electrode surface. The IADs are measured with the 75μm diameter, 9 μm thick orifice.

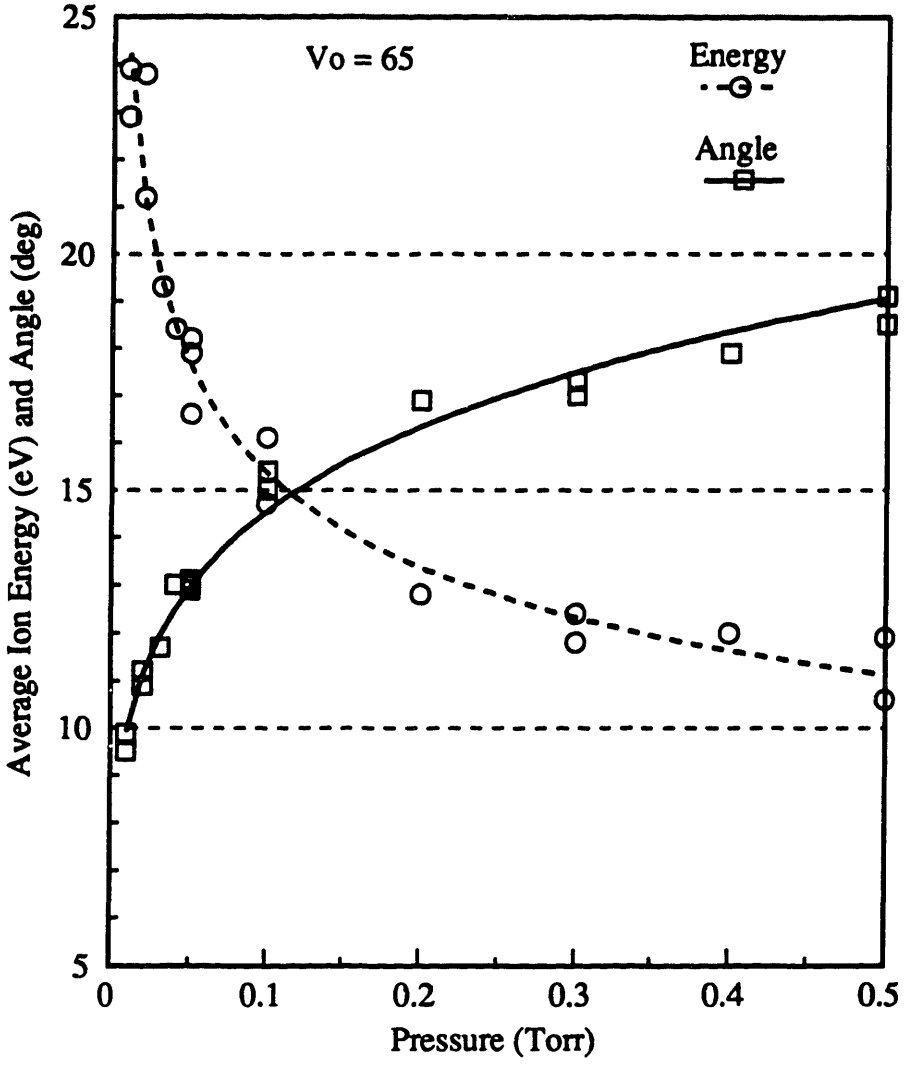


Figure 4.4: Average ion energy and angle at Vo=65 as a function of pressure.

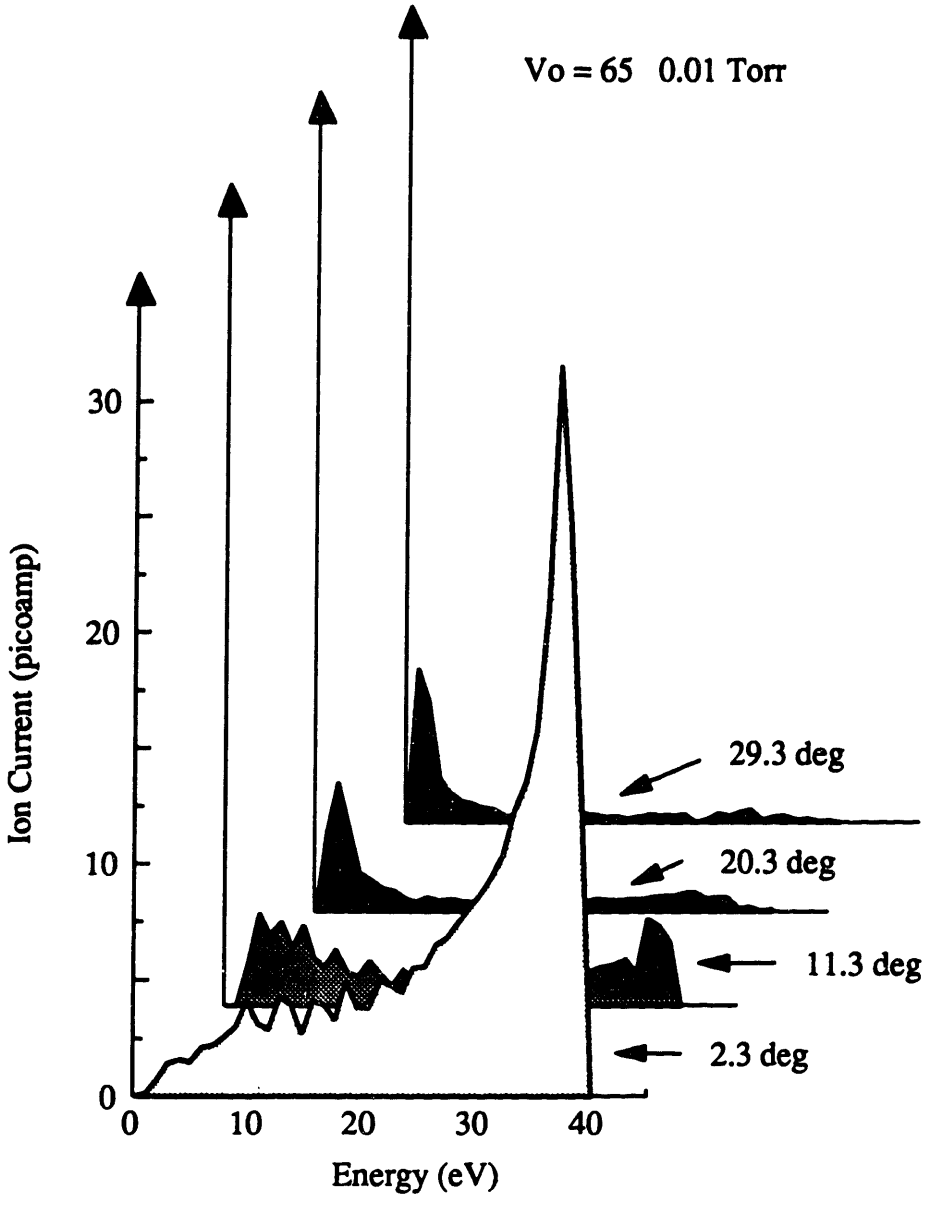


Figure 4.5: IEDs at various incident angles for 0.01 Torr and Vo=65.

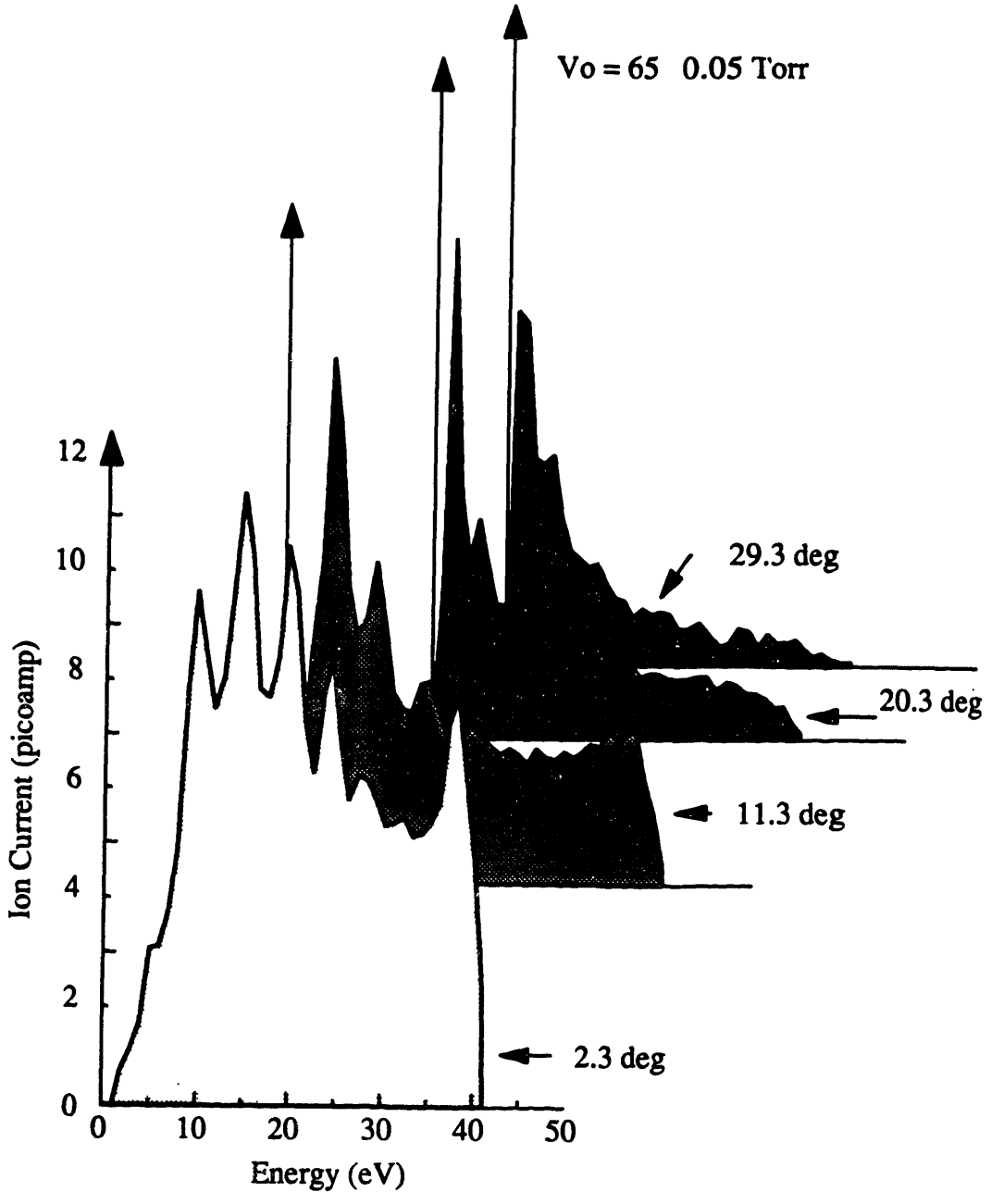


Figure 4.6: IEDs at various incident angles for 0.05 Torr and Vo=65.

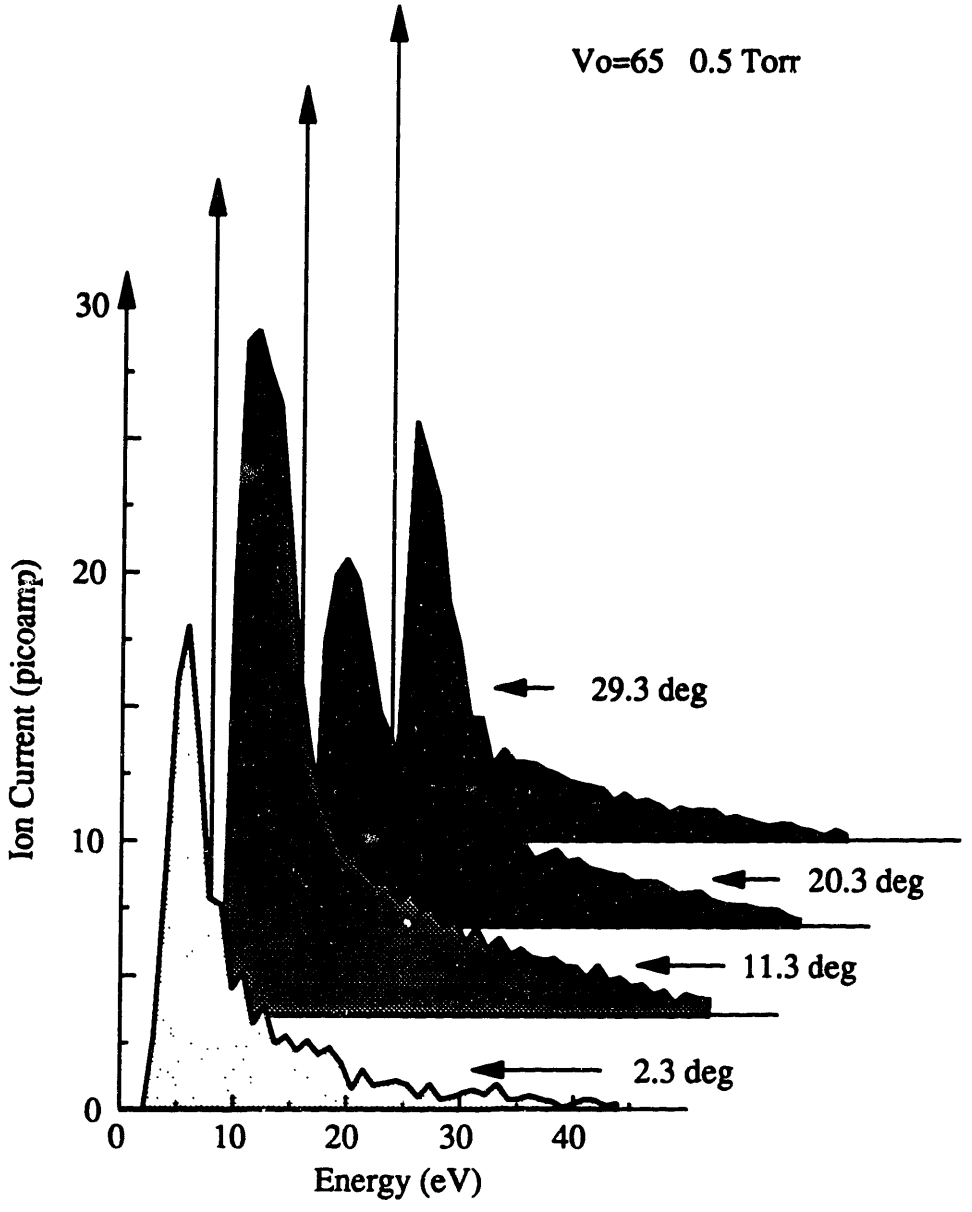


Figure 4.7: IEDs at various incident angles for 0.5 Torr and Vo = 65.

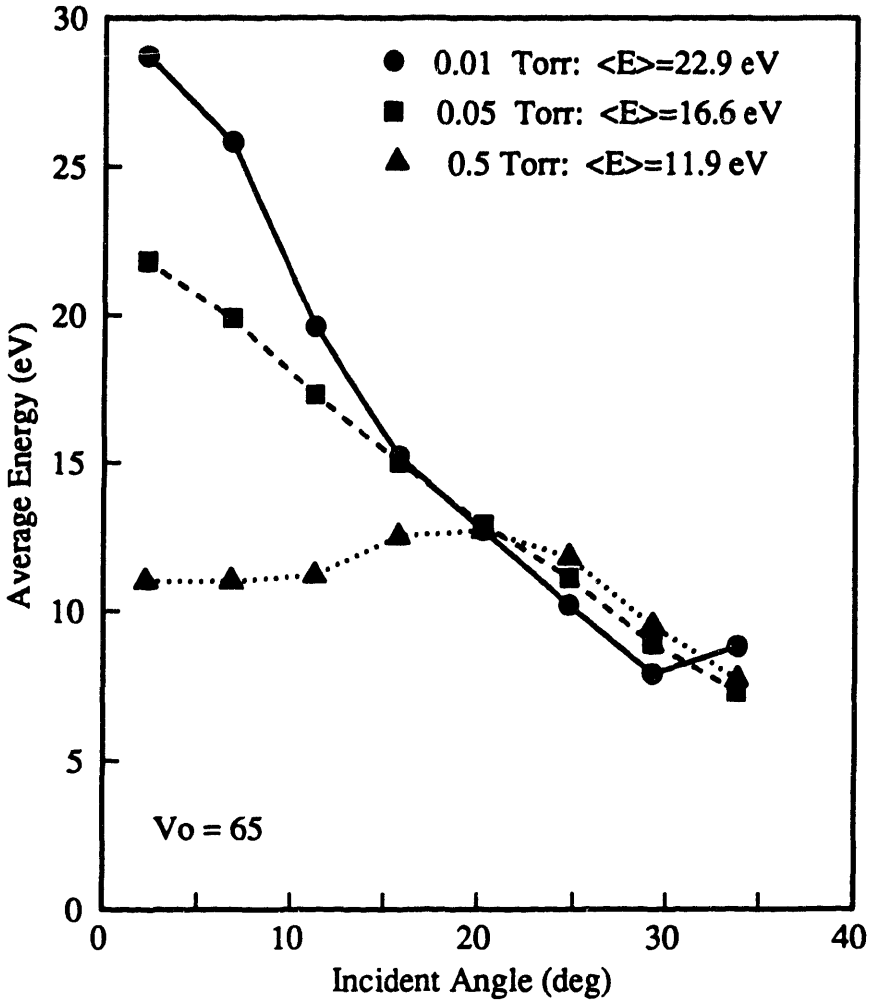


Figure 4.8: Average ion energy as a function of ion incident angle at Vo = 65. The overall average ion energy is written next to the pressure.

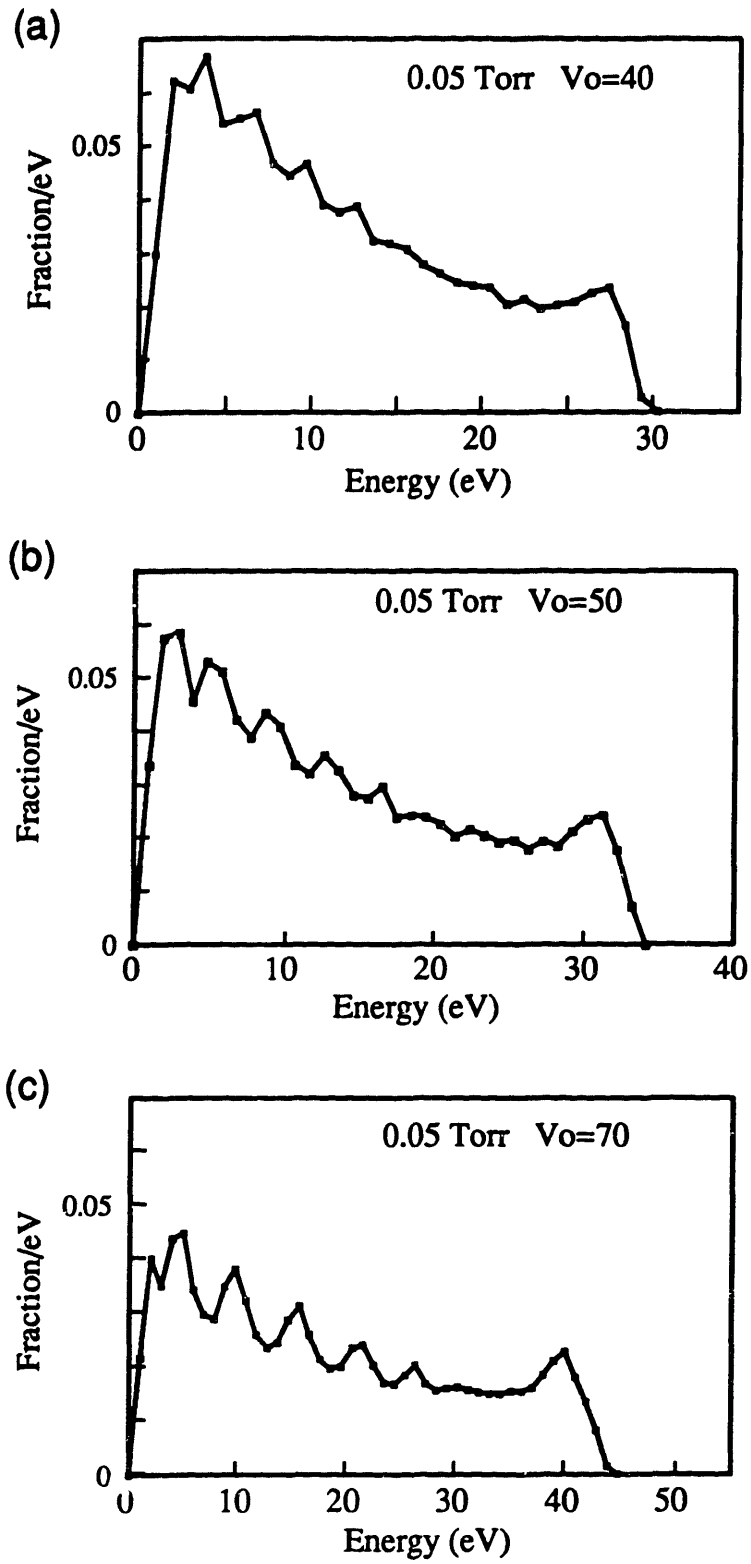


Figure 4.9: IEDs at 0.05 Torr and various Vo's.

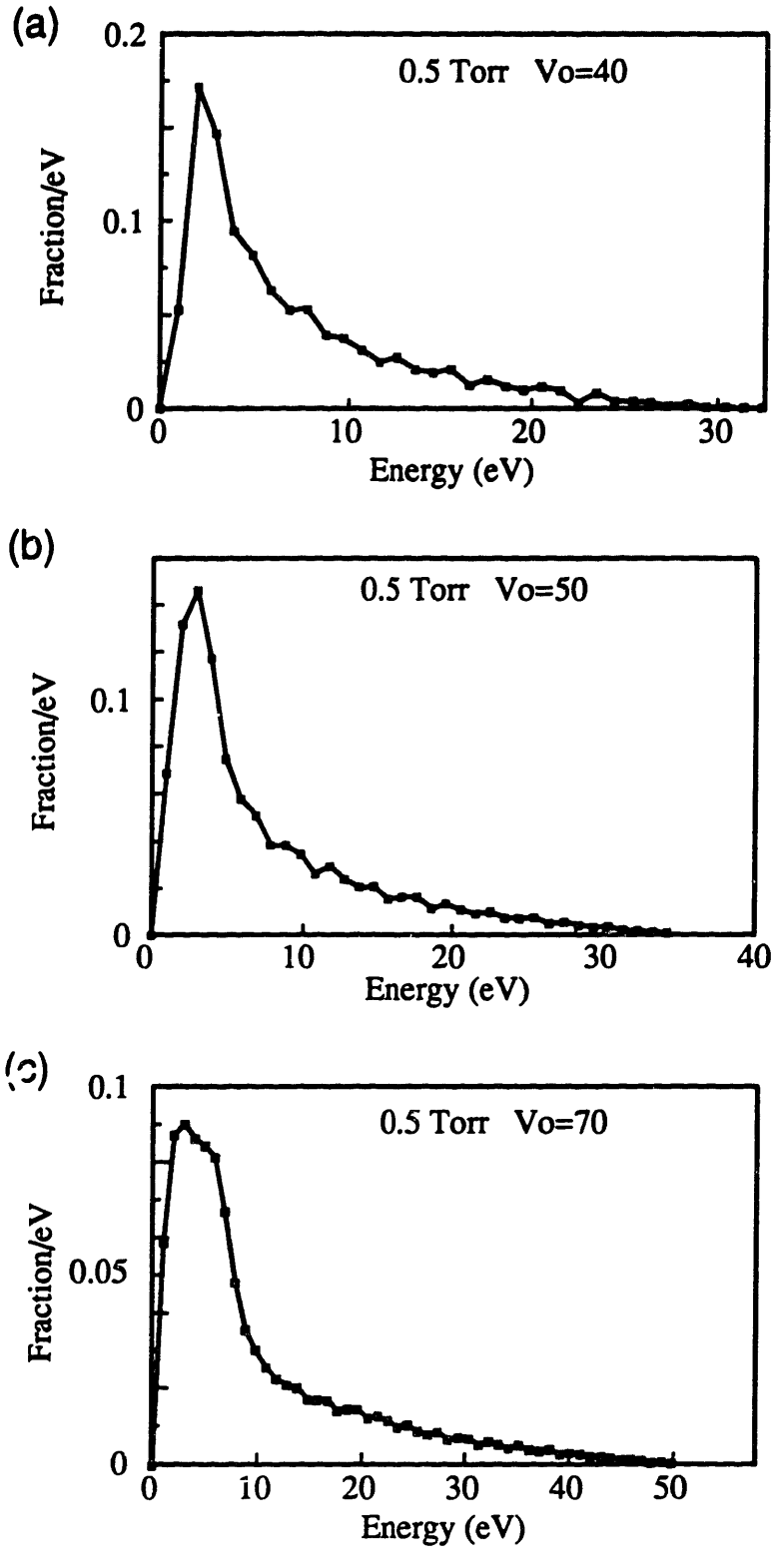
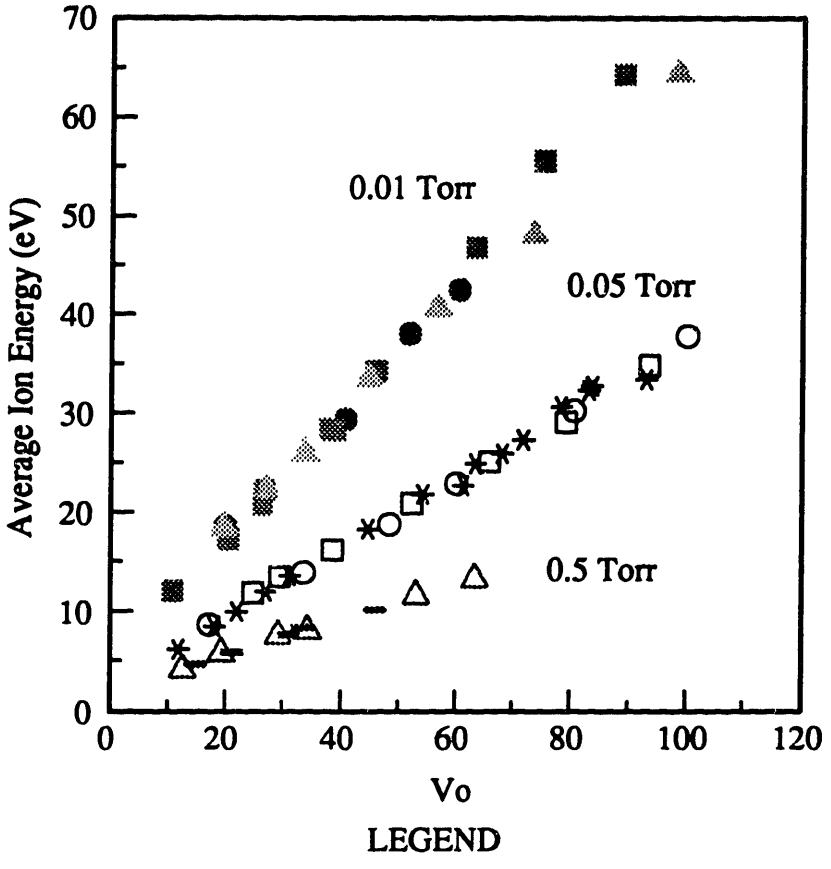


Figure 4.10: IEDs at 0.5 Torr and various Vo's.



LEGEND

0.01 Torr	0.05 Torr	0.5 Torr
D=9cm d=5cm	D=9cm d=3cm	D=9cm d=2cm
D=9cm d=5.1cm	D=17cm d=2.5cm	D=13cm d=3cm
D=17cm d=5.1cm	D=17cm d=5.1cm	

Figure 4.11: Measured average ion energy as a function of Vo at various pressures and reactor geometries.

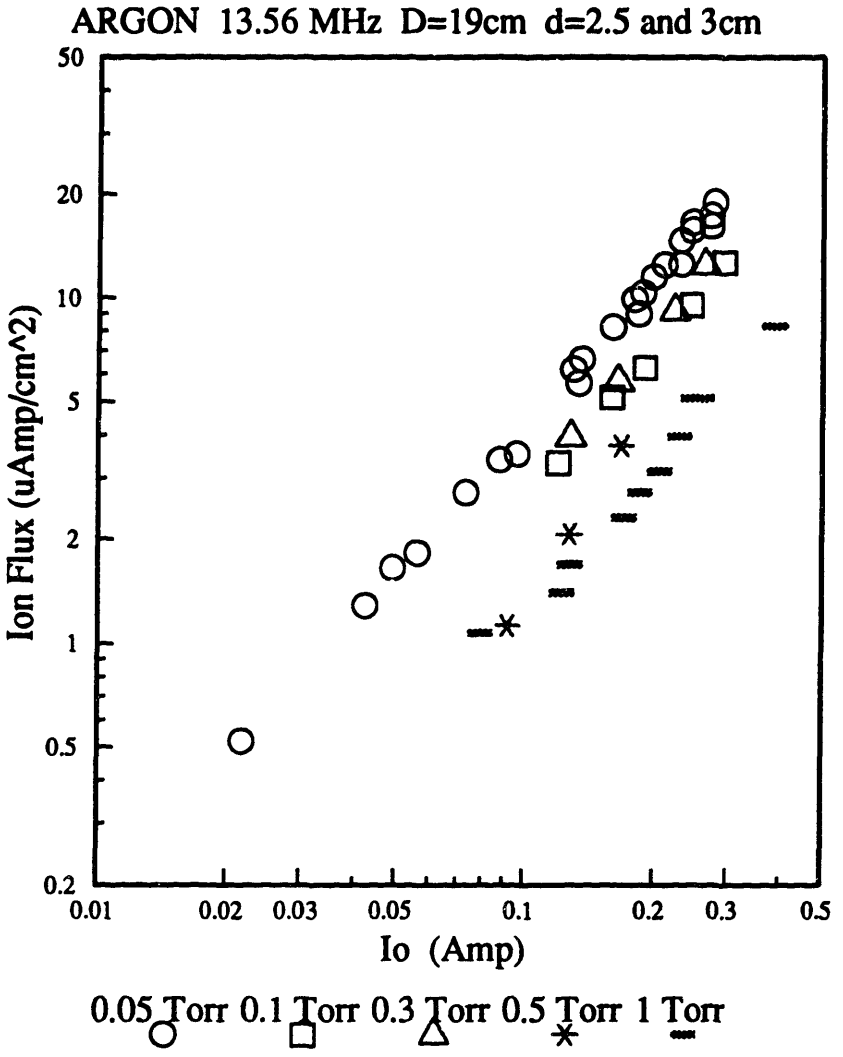


Figure 4.12: Ion flux on the electrode from 13.56 MHz argon plasmas.

CHAPTER 5

MONTE CARLO SIMULATION OF ARGON SHEATH KINETICS

Monte Carlo techniques are used to follow the ion-neutral kinetics in the sheath to predict the energetic particle bombardment energies and angles on the surface. Although the energetic particle bombardment depends on many factors, such as the operating power, pressure, magnetic field strength, and the type of gases used in the plasma, the factors influencing the particle bombardment energy and angle can be consolidated into the following terms:

(1) The magnitude of the sheath electric field, which accelerates ions through the plasma sheath toward the surface, directly determines the energy gained by the ions.

(2) The number of mean free paths in the sheath, which is set by the pressure and magnetic field strength, determines the number of collisions the ions undergo before striking the surface.

(3) The mass ratio of the ion to other particles in the plasma determines the amount of momentum transferred during a collision. These collisions reduce the ions' energies and change their trajectories.

(4) The type of collisions these ions undergo also affects their final energies and angles. Some ions undergo charge-exchange collisions in which the original ion becomes an energetic neutral and a new, low energy ion is created. These collisions are common when the neutral gas particle is the same atom or molecule as the ion, but without charge. Charge-exchange collisions tend to lower the average ion energy, and bring the average incident angle closer to the surface normal than hard sphere collisions.

There are two reasons for modeling ion-neutral sheath kinetics. Monte Carlo simulations

of the particle bombardment energy and angle distributions circumvent the need to measure these properties for every operating condition and every gas used. Secondly, in charge-exchange dominated plasmas, energetic neutrals can compose a large fraction of the total energetic particle bombardment on surfaces, and therefore, neutrals as well as ions, can affect the etching direction and rate. These energetic neutrals are not easily measured, but Monte Carlo simulations keeps track of them.

In this chapter, Monte Carlo simulation results are compared to measured argon ion energy and angle. This work also compares the assumptions of spatially uniform versus linear electric field, shows the effect of various types of scattering probabilities on the modeling results, and evaluates the importance of energetic neutral flux on the surface at various pressures. This work differs from previously reported Monte Carlo simulation results of plasma sheath kinetics in that measured differential cross-sections for ion-neutral collisions are used instead of assuming an isotropic angular scattering probability.

5.1 REVIEW OF PREVIOUS WORK

Several workers have modeled the energy and angle of particles striking the surface. Jurgensen (1988) reported both ion angle and energy distributions for oxygen plasmas, where charge-exchange collisions dominate. In his model, he used a dc sheath electric field because the ion transit time across the sheath was much greater than a 13.56 MHz rf period. He showed that, with hard sphere collisions, a significant number of ions strike the surface at off normal incident angles. He also concluded that energetic neutrals should be considered when charge-exchange is a dominant collision process. Kushner (1985) investigated ion bombardment properties using a Monte Carlo model which assumed that only charge-exchange collisions occur. In his model, the initial thermal energies of the ions created through charge-exchange collisions were the only

mechanism for production of off normal incident angles. Thompson *et al.* (1988) simulated ion energy distributions (IEDs) as a function of the number of mean free paths in the sheath. They showed that various types of ion-neutral scattering processes in the sheath (such as hard sphere, isotropic shrinking sphere, anisotropic shrinking sphere, and 9-6-4 potential interaction collisions) with either spatially uniform or linear sheath electric fields produce roughly equivalent average bombardment angle and non-dimensionalized energy for fully developed distributions.

Monte Carlo methods can simulate not only ion bombardment properties, but also the properties of energetic neutrals created via collisions with ions or other energetic neutrals. Manenschijn and Goedheer (1991) used both charge-exchange and isotropic scattering to follow Ar-Ar^+ and Ar-Ar collisions in the sheath. They compared the flux on the surface of all neutrals with energies above 0.1 eV to that of all the ions. They concluded that for sheath widths greater than one mean free path long, neutrals are a significant fraction of the energetic particle flux on surfaces. May *et al.* (1992) also used the Monte Carlo technique with Cramer's (1959) argon cross sections to show that neutrals can be a large fraction of the energetic ion flux on surfaces.

Sommerer and Kushner (1991) used a hybrid model of the plasma bulk and sheath to study a CF_4 plasma. The energetic neutral considered was the F atom, produced mostly near the bulk sheath interface from electron impact dissociation. This work differs from the previous two works by Jurgensen (1988) and Manenschijn and Goedheer (1991) in that the neutral energy comes, not from charge-exchange or hard sphere collisions in the sheath, but from energy released to the F atom, 8 eV, after a dissociative CF_4 reaction. Monte Carlo simulation, assuming CF_3^+ as the only ionic species in the plasma and a spatially linear, 13.56 MHz time varying sheath electric field, produced trends showing the importance of neutral F atoms at various pressures and applied voltages.

Wild and Koidl (1991) have measured and modeled IEDs in an asymmetric argon plasma.

Ions were assumed to be created uniformly in both space and time in the sheath via charge-exchange collisions. Equations were developed to solve for the energy of ions traveling through a time varying sheath electric field. Two adjustable parameters were used to provide a near perfect fit to the experimental data. These parameters gave the spatial dependence of the electric field in the sheath and the sheath width. This model explained the origin of the multiple peaks in the IEDs commonly observed in rf plasmas dominated by charge-exchange collisions.

5.2 MONTE CARLO METHOD

Monte Carlo simulations of the plasma sheath kinetics, under the experimental conditions, are performed using a modified version of the program previously reported by Thompson *et al.* (1988). The major difference between Thompson's program and the current application is the use of time of flight (TOF) instead of distance of flight (DOF) analysis for generating ion flight paths. The DOF technique inherently requires calculation of not only the final position and velocity of each ion, but also requires the total length of the path traversed to reach the final state. The TOF technique removes this calculation, but increases the number of null-scattering events. The TOF technique is more computationally efficient for most of cases studied here, and thus is used for all simulations. The TOF technique also preserves analytic solutions for position versus time in the simple cases of dc fields and spatially uniform rf fields.[2]

Four input parameters (type of ion-neutral and neutral-neutral collision cross sections, electric field profile in the sheath, sheath thickness to mean free path ratio, and frequency) are required for the Monte Carlo simulations. These parameters are computed from the experimentally measured applied voltage, optical sheath thickness, pressure, applied frequency,

[2]Major modifications to B. E. Thompson's program were done by G. L. Huppert. These modifications and additions include changing from DOF to TOF analysis and an algorithm for tracking the energetic neutrals.

and collision cross section data from Cramer (1959) and Vestal (1978). In all the simulations, the total collision cross section for ion-neutral collisions follows:

$$\sigma\ (cm^2) = 9.64 \times 10^{-15}\ \varepsilon^{-0.131}\ , \qquad (5.1)$$

where ε is the ion energy given in eV. This equation is derived from data regression of Cramer's measured cross sections between 4 eV and 400 eV.

The program monitors ion trajectories and energies as the ions begin at the bulk-sheath interface, travel through the sheath, and eventually strike the surface. The ions are decelerated through collisions with the background gas and accelerated by the sheath electric field. The number of collisions between ions and neutrals depends on the pressure and the sheath width.

The three cases of scattering angle probabilities for ion-neutral collisions used in this chapter are listed in Table 5.1.

Table 5.1: Types of Scattering Probabilities for Ion-Neutral Collisions

Case	Scattering Probability	Comments
1	isotropic	The ion has equal probability of scattering in any direction in the center of mass reference frame after a collision.
2	55% charge-exchange 45% isotropic (Cramer 1959)	The same percent of charge-exchange and isotropic scattering is used for all energies. Charge-exchange is probably higher for ion energies below 5 eV where no data is available.
3	anisotropic (Vestal *et al.* 1978)	

In the Monte Carlo simulations, a charge-exchange collision produces a new ion which has no energy and a neutral with velocities equal to that of the original ion. The scattering probability described by Case 3 eliminates the separation of charge-exchange and hard sphere collisions described by Case 2. Case 3 assumes that there are only elastic scattering events, but that the scattering angle probability follows that reported by Vestal *et al.* (1978) for Ar-Ar^+ collisions

rather than an isotropic differential scattering angle probability. In the center of mass reference frame, the reported differential cross sections show that ions are more likely to be forward and back scattered than in an isotropic scattering probability. The scattering probability at any angle is calculated by multiplying the sine of the scattering angle and the average of Vestal *et al.*'s (1978) measured differential cross-sections over the four reported ion energies between 2 eV and 20 eV. Figure 5.1 shows the angular scattering probability of all three cases.

Both spatially uniform and linear sheath electric fields are used in the modeling results shown in this chapter. Gogolides (1990), using an electrodynamic continuum model, has shown that the sheath electric field is neither spatially uniform, nor linear. This is caused by the time varying sheath width. A current passing through the time varying sheath capacitance produces a nonlinear response. Monte Carlo simulation, assuming a spatially uniform rf electric field in the sheath, underestimates the measured ion energy (Liu *et al.* 1990). A spatially linear sheath electric field raises the predicted ion energy.

Several authors have explored whether the actual sheath electric field is more closely approximated by an uniform or linear field. For the electric field expressed in the form $\mathcal{E} \propto x^n$, where x is the distance into the sheath, all have reported values of n between 0 and 1. For a collisionless sheath, Child's (1911) equations produced $\mathcal{E} \propto x^{1/3}$. For a collisional sheath, where the ion mobility is proportional to the electric field, $\mathcal{E} \propto x^{1/2}$ (Cobine 1941). In the intermediate regime, Lieberman's (1989) equations produced $\mathcal{E} \propto x^{2/3}$. Jurgensen and Shaqfeh's (1988) modeling showed that as the number of mean free paths in the sheath increases, n increases from 1/3 to 2/3 for a dc sheath electric field. Wild and Koidl, inferred from fitting equations to their measured argon IEDs, that $0.55 < n < 0.69$. Therefore, although the spatially uniform (n=0) and linear (n=1) electric field used in the Monte Carlo simulations here, while not exact, gives the bounds for the average values of ion properties.

The oscillating sheath electric field amplitude is set equal to V_s/ℓ_s and $2V_s/\ell_s$ for spatially uniform and linear field, respectively, where V_s is the sheath voltage and ℓ_s is the sheath width. V_s is the sum of a time varying component plus the dc floating potential, V_f. In argon plasmas, the time varying component is generally half the voltage amplitude applied across the electrodes, $\frac{1}{2}V_o$. The dc floating potential, V_f, found by comparing the maximum measured ion energies reported in Chapter 4 to the applied voltage, is 7 V for argon plasmas.

The program tracks the neutrals from the time they are created until they strike the electrode surface or until their energies drop below a specified limit. Energetic neutrals can be created from collisions with ions or with other energetic neutrals. The total collision cross section and the angular scattering probability for neutral-neutral collisions are different from those of ion-neutral collisions. The ion-neutral collision cross section should be larger than the neutral-neutral collision cross section because the long range force from an ion induced dipole falls as r^{-6}, where r is the distance between the ion and neutral particles. Also, the bond strength for Ar_2^+ should be much larger than for Ar_2. Therefore, the total collision cross section for neutral-neutral collisions should be smaller than that described by Equation 5.1. A cross section of 1.0×10^{-15} cm^2 (Chapman and Cowling 1952) is used for neutral-neutral collisions.

In the simulations, the scattering probability used for neutral-neutral collisions is also different from that of ion-neutral collisions. Thompson *et al.* (1988) showed that there is almost no difference in Monte Carlo predictions of ion energy and angle between using hard sphere collisions, where there is only repulsive forces, and using collisions with an interaction potential between the two colliding particles where the attractive force is taken into account. This long range attractive force affects low energy scattering (*e.g.* thermal energies) and the interaction potential between argon atoms is smaller than the combined energies of the colliding neutrals. Therefore, all neutral-neutral collisions are assumed to undergo isotropic scattering in the program

because only high energy neutrals are monitored. If low energy neutral were monitored, then the scattering probability would look more like that for ion-neutral collisions. From both classical and quantum mechanical calculations of elastic scattering (Smith 1980, Buck 1975), the forward scattering probability for a neutral-neutral collision is found to be at least an order of magnitude larger than for angles greater than some critical angle, θ_c, where θ_c decreases as the neutral energy increases. This means that as the neutral energy increases, the angular scattering probability approaches the isotropic scattering probability. Unlike the ion-neutral scattering probability, in neutral-neutral collisions, there is not a greater probability for back scattering to occur over that of other scattering angles. But in the center of mass reference frame, the result of a forward scattered neutral is indistinguishable from that of a back scattered neutral. Therefore, when θ_c is large or the neutral energy is low, the measured angular scattering probability should look somewhat like the ion-neutral angular scattering probability shown by the solid line in Figure 5.1.

Another input parameter to the Monte Carlo program is the sheath width to mean free path ratio. The mean free path is calculated from the pressure and Equation 5.1. The sheath width varies as a function of the pressure and is found from spatial scans of the argon plasma emission. The sheath edge was defined as the peak in the 750.6 nm argon emission intensity scan. The sheath widths at various pressures are listed in Table 5.1.

Table 5.2: Measured Argon Sheath Width at Various Pressures

Pressure (Torr)	0.01	0.05	0.1	0.3	0.5	1	2
Sheath Width (mm)	11	8	7	6	4	3	2

The simulated ion angle distributions (IADs) are presented, azimuthally integrated, as the fraction of the total ions per degree from the surface normal in order to directly compare simulations to experimental results. A large number of ions, 5×10^5, is used in the simulations to assure small statistical scatter in the results.

5.3 COMPARISON WITH EXPERIMENTAL RESULTS

Measured IEDs and IADs used for comparison with Monte Carlo simulation results are found in Chapter 4. Monte Carlo simulations of 0.01, 0.05, and 0.5 Torr IEDs using separation of hard sphere and charge-exchange collision cross sections (Cramer 1959), Case 2, are shown in Figure 5.2 and using Vestal *et al.*'s (1978) differential collision cross sections, Case 3, are shown in Figure 5.3. In all simulation cases, a 13.56 MHz spatially linear electric field is used. The model predicts very similar IED shapes and for the 0.01 and 0.05 Torr cases, approximately the same number of peaks as the measured IEDs found in Figure 4.2. There is not much difference between the shapes of the IEDs generated with Cramer's and Vestal *et al.*'s differential cross sections, although the IEDs generated with Vestal *et al.*'s cross sections is closer to the experimentally measured IEDs.

The large low energy peak in the 0.01 Torr data, which is not seen in the model, may be attributed to lack of accurate collision cross section data used in the model for very low energy charge-exchange and hard sphere collision processes. At very low energies, the charge-exchange cross section should increase much faster with decreasing energy than the $\varepsilon^{-0.131}$ dependence used in the model. Ions which are created by a charge-exchange collision are more likely to go through another charge-exchange collision because they have very low energies. This cascade effect should produce the low energy peak in the 0.01 Torr IED.

There are multiple peaks in the IED generated by the model which are absent in the experimental data taken at 0.5 Torr. In this case, the energy separation between peaks may be too small for the ion analyzer to resolve. At 0.5 Torr, although Monte Carlo simulations with a spatially linear electric field predict higher ion energies than seen in simulations with an uniform electric field (Liu *et al.* 1990), the predicted ion energies are still lower than the experimental values. Experimental errors may result from the non-uniform field at the orifice and the grid used

as a high pass filter. The non-uniform field bends the ion trajectories away from the collecting plate and affects low energy ions the most. If this occurs, then the low energy ions are not measured and thus a seemingly higher fraction of ions arrive at the high energy end of the IED. Since the second grid used as a high pass filter does not produce a perfectly uniform potential between grid wires, ions with energies lower than the grid potential (not less than 90% of the grid potential) can actually make it past the grid, also shifting the fraction of ions toward the high energy end of the IED. However, these two experimental artifacts cannot fully account for the large difference in energies observed between measurement and model. The plasma may have Ar^{2+} and ArH^+ ions which can make the measured ion energy higher than the Monte Carlo simulation results. The Ar^{2+} ion has twice the charge, and therefore twice the energy, of the Ar^+ ion if they both have the same sequence of collisions traveling through the sheath. Davis and Vanderslice (1963) measured IEDs from dc argon plasmas at 0.47 Torr and 0.6 Torr. They showed that the maximum Ar^+ energy is less than the sheath potential while the maximum Ar^{2+} energy is equal to the sheath potential. Although these measurements were done in dc plasmas, their data indicates that the high energy ions presented in the IEDs of this thesis are doubly charged ions. The ArH^+ ion is most likely formed via an attachment reaction in the plasma bulk and it goes through fewer collisions traveling through the sheath than the Ar^+ ion (Greene *et al.* 1988). Thus the average energy of ArH^+ ions should be higher than that of the Ar^+ ions. Although the high energy tail in the IEDs can be attributed to either Ar^{2+} and ArH^+ ions, the IADs presented later in this section indicate that the high energy ions are probably a result of Ar^{2+} bombardment.

A comparison between a Monte Carlo simulation with only charge-exchange collision processes and a simulation with only hard sphere collisions processes will demonstrate that the multiple peaks in the IEDs are a result of charge-exchange collisions. Figure 5.4 is a simulation

of a 0.05 Torr and V_0 = 65 V argon plasma with only charge-exchange collisions, showing multiple peaks in the IED. Thompson *et al.* (1988) simulated IEDs with only hard sphere collisions and their IEDs do not have multiple peaks. This implies that charge-exchange collisions are necessary to produce multiple peaks in the IEDs. These multiple peaks should not be confused with the double-peaked structure at the high energy end of non-collisional IEDs which results simply from the sinusoidal modulation of the applied voltage (Thompson *et al.* 1988, Ulacia & McVittie 1989, Okamoto & Tamagawa 1970, Coburn & Kay 1972, Tsui 1968, Kushner 1985). Charge-exchange collisions, which are included in Cramer's cross sections for Ar^+-Ar collisions, account for the multiple peaks in the Monte Carlo simulated IEDs in Figure 5.2.

Although Vestal *et al.*'s differential collision cross sections does not specifically identify charge-exchange collisions, back scattering events produces charge-exchange like collisions which create the multiple peaks in the IEDs. These collisions look like charge-exchange collisions because the ion loses most of its energy to the neutral and is left standing in the sheath until the electric field accelerates it toward the electrode. To show that the multiple peaks in the IEDs arise from the back scattered ions, the angular scattering probability used in the program was altered so that there was zero probability for back scattering. This probability distribution is shown in Figure 5.5. The IED simulated from this differential angle scattering probability at 0.05 Torr and V_0 = 65, shown in Figure 5.6, has no multiple peaks. The two peaks at the high energy end of the IED consist of ions which have not gone through any collisions and they contribute to one of the two peaks depending on the phase of the time varying field when they enter the sheath. Given sufficient resolution in the energy detector, these peaks are always present in the IEDs of ions accelerated through a time varying sheath electric field. The lack of multiple peaks in the IEDs shown in Figure 5.6 confirms that back scattered ions are equivalent to ions formed by charge-exchange collisions.

Analytical expressions, derived with several simplifying assumptions, describing the ion velocity in the sheath, can aid in explaining the origin of the multiple peaks in the IEDs. Figure 5.7 shows a three dimensional graph of the final ion energy as a function of the position in the sheath and the time in the rf period in which the ion is created. The final ion energies are calculated with several simplifying assumptions: 1) the ions are created, via charge-exchange collisions, uniformly in the sheath in both space and time, 2) they start with no energy, and 3) the sheath electric field is spatially uniform, but varies sinusoidally in time. A force balance on the ion produces the following equations:

$$F = m_{Ar^+} \frac{d^2x}{dt^2} = q\mathcal{E} \tag{5.2 a}$$

$$\frac{d^2x}{dt^2} = \frac{q\mathcal{E}_o}{m_{Ar^+}}[1+\sin(\omega t+\phi)] \tag{5.2 b}$$

$$v = \frac{dx}{dt} = \frac{q\mathcal{E}_o}{m_{Ar^+}}\left[t+\frac{\cos(\omega t)-\cos(\omega t+\phi)}{\omega}\right] \tag{5.2 c}$$

where F is the force, m_{Ar^+} is the mass of the ion, x is the position in the sheath with 0 defined at the electrode surface, t is time, q is the ion charge, $\mathcal{E}$ is the sheath electric field, $\mathcal{E}_o$ is the time varying amplitude of $\mathcal{E}$, ω is the angular frequency, ϕ is the phase the ion is created, and v is the ion velocity. Equation 5.2 (b) is solved to find the time required for an ion to travel a distance x to the electrode surface. Once the time is known, the final ion velocity, and therefore energy, is calculated from Equation 5.2 (c). The solutions to these equations are presented in Figure 5.7 (Huppert 1992), where the three dimensional graph shows the final ion energies at the electrode (z-axis) for ions created via charge-exchange collisions at various locations in the sheath, x (x-axis), and times in the rf period, ϕ (y-axis). Ions created when the sheath electric field is low do not move much, and are bunched together with ions created at higher fields when they are swept

toward the electrode. This creates the flat surfaces in the 3-dimensional graph over the fraction of the rf period when the field is low. The graph on the left shows the number histogram of the final energies of ions at the electrode if the ions are created uniformly in space and time. The flat portions of the 3-dimensional plot correspond to the peaks in the histogram or IED.

The Monte Carlo simulations of the IADs for the same experimental conditions as Figure 4.3 and corresponding to the IEDs in Figures 5.2 and 5.3, are shown in Figure 5.8. The IADs calculated using Vestal *et al.*'s (1978) differential cross section are closer to the measured results than the IADs calculated using Cramer's (1959) separation of charge-exchange and hard sphere collision cross sections. Previous simulations of argon sheath kinetics reported in literature have always used separation of charge-exchange and hard sphere collisions. In Liu *et al.*'s paper (1990), it was postulated that in a charge-exchange collision, there may also be energy transfer so that the newly created ion has some momentum directed non-normal to the surface. The scattering angle probability calculated from Vestal *et al.*'s differential cross sections in Figure 5.1 shows that a large fraction of the ions are back scattered near 180°. These are the ions that go through charge-exchange collisions with some momentum transfer. Using this scattering angle probability rather than that calculated with a division of charge-exchange and hard sphere collisions produces simulated IADs that better match the experimental measurements. The sum of the ions from 0° to 9° in the 0.01 Torr IAD calculated from Vestal *et al.*'s data, Figure 5.8 (b), is only 20% higher than the measured values, while at 0.5 Torr, this sum is roughly the same for the calculated and the measured ions. This indicates that in the measured IADs, there may have been some widening of the measured ion beam as it passed through the non-uniform field near the orifice (See Appendix I.4).

Earlier in this section, it was postulated that ArH^+ and Ar^{2+} ions may strike the surface besides Ar^+ ions because the measured ion energies are higher than the simulation values. If the

higher energy ions are ArH^+ ions which have gone through fewer collisions traveling through the sheath than Ar^+ ions, then the measured IADs should show more ions with near normal incidence than the simulated results. However, this is not observed, indicating that the higher measured ion energy is not a result of fewer collisions in the sheath, but of ions gaining more energy while having the same number of collisions in the sheath. The Ar^{2+} ions would satisfy these criteria.

Figure 5.9 through 5.11 show the Monte Carlo simulation IEDs at various incident angles for 0.01, 0.05, and 0.5 Torr. These correspond to the measured IEDs shown in Figures 4.5 through 4.7. The IEDs are simulated with a 13.56 MHz spatially linear electric field and the scattering angle probability calculated from Vestal *et al.*'s (1978) differential cross sections. At 0.01 Torr and 0.05 Torr (Figures 5.9 and 5.10), the distributions at angles other than 2.3° are multiplied by ten and five, respectively. Multiple peaks in IEDs are present at several incident angles. In contrast, for IEDs simulated with division of hard sphere and charge-exchange cross sections, multiple peaks are observed only for ions with perpendicular incident angle (Liu *et al.* 1990, Manenschijn & Goedheer 1991). Using differential cross sections instead of a distinct separation of hard sphere and charge-exchange collisional events has brought the model simulation results closer to the measured ion IEDs. The measured IEDs have a large peak at the low energy end of the IEDs, not seen in these simulations, because the these ions have been shifted from normal incidence to higher incident angles by the non-uniform field at the orifice (See Appendix I.4.)

At 0.5 Torr (Figure 5.11), the trends predicted by the model agree with the experiments. The major difference is that the predicted magnitude of the ion energies at 0.5 Torr are much lower than the experimental values. As mentioned already, the high energy ions are probably doubly charged argon, which are not used in the simulations.

Figure 5.12 shows the Monte Carlo simulation of 0.05 Torr IEDs at several V_o's. These

correspond to the measured IEDs in Figure 4.9. A 13.56 MHz linear electric field and Vestal *et al.*'s differential cross section are used in the simulation. The number of peaks and peak separation are similar to the experimental results. Changing V_o in the simulations changes the magnitude of the ion energy without affecting the general shape of the IEDs. The comparison, between measured IEDs with Monte Carlo simulations for 0.01 and 0.5 Torr plasmas at several voltages, is similar to the 0.05 Torr case shown here in that the shape of the predicted and measured IEDs do not change with V_o.

5.4 PARAMETER EFFECTS

The spatial dependence of the electric field on the ion properties is explored by running simulations using both spatially uniform and linear electric fields. Figure 5.13 shows the Monte Carlo simulation results using Vestal *et al.*'s differential cross section, along with the measured average ion bombardment energy and angle. Both experimental measurements and simulation results show that increasing pressure causes the average ion energy to decrease and the average ion angle to increase. This is expected because the number of mean free paths in the sheath increases. For both spatially uniform and linear electric fields, the simulation under-predicts the measured ion energy and angle, although the linear electric field comes closer, by a factor of two over that of the uniform electric field at high pressures, to the measured values. On the other hand, the average ion angle is not very sensitive to the spatial form of the electric field used in the simulation. Earlier in this chapter, it was postulated that low energy ions are not detected on the collecting plate of the ion analyzer. In the simulation results, if all ions with 5 eV or less at the electrode are not used to calculate the average ion energy, then the agreement between the model and experiments are brought within 4 eV of each other. On the other hand, there is not much difference between the average angle calculated with and without the low energy ions (ε

< 5 eV). However, Figure 4.2 (c) shows that some low energy ions are detected. Therefore, neglecting all the low energy ions in the averaging is not valid even though the experimental and simulation values match better without the low energy ions. Not measuring all the low energy ions may partially contribute to the high average ion energy measured, but there must be ions that go through fewer collisions or gain more energy traveling through the sheath than Ar^+, as postulated in Section 5.3.

Farouki *et al.* (1991) also used the Monte Carlo technique to model ion transport through the sheath. In their model, rather than assuming a spatial form for the sheath electric field, their model calculates the spatial variation of electric field from the ion density. Although average energy and angle values were not reported, the shapes of the energy and angle distributions look similar to the ones shown here. Therefore, varying the spatial dependence of the electric field changes the magnitude of the average energy and angle, but does not alter the qualitative features of the predicted ion distributions and trends.

Since the ion plasma frequency is lower than the 13.56 MHz driving frequency, it is often assumed that the ions do not respond to the time varying field, but see only the time averaged field. Figure 5.14 shows the IEDs from a simulation at 0.05 Torr and V_o = 65 for both dc and rf spatially linear electric fields. The general shape of the energy and angle distributions are the same for both the dc and rf cases, the only difference being that the dc case has no multiple peaks in the IED. The average energy and average angle are the same for both cases, again showing that a time averaged electric field is sufficient to approximate a 13.56 MHz sheath electric field for an argon plasma.

Three scattering angle probabilities, described in Section 5.2, are used to explore the effect of various cross sections on the predicted ion energy and angle. Refer to Figure 5.1 for the angular scattering probabilities of all three cases. In all these cases, the total collision cross

section used is the same, Equation 5.1. These simulation results are shown in Figure 5.15. The type of angular scattering probability used in the simulation has little effect on the predicted ion energy, but has a profound effect on the predicted angle with which ions strike the surface. With the separation of charge-exchange and hard sphere collisions, Case 2, the average ion incident angle is very low because the ions are redirected normal to the surface in 55% of the collisions. In Case 3, the ion is still preferentially forward and back scattered in the center of mass reference frame, but momentum is also transferred in these collisions. Thus, the predicted average ion bombardment angle is slightly greater than in Case 2. In the isotropic scattering case, Case 1, the probability for forward and back scattering is almost zero, resulting in very large average ion bombardment angle.

If the plasma contains a presheath region which accelerates ions, this effect may be captured in the simulation by giving the ions some energy before they start crossing the sheath. Simulations at 0.01, 0.05, and 0.1 Torr were run with ions having an initial energy of 1 eV directed perpendicular to the surface. The difference in simulation results between ions starting with 1 eV and 0 eV is found only in the 0.01 Torr case, where the average energy is 0.2 eV greater for the higher starting energy. Collisions in the sheath and the accelerating sheath electric field have greater effects on the ion properties than the presheath acceleration.

5.5 NEUTRAL RESULTS

In ion-neutral collisions, if the ions are back scattered, they transfer most of their original energy to the neutrals. These neutrals have been hypothesized to constitute a significant portion of the energetic particle flux on the surface. To verify the importance of the neutral flux onto the surface, energetic neutrals created from collisions with ions are followed through the sheath until they either strike the electrode or drop below a specified minimum energy, E_{min}.

An example of the Monte Carlo simulation output of neutral properties is shown in Figure 5.16. The corresponding ion properties are shown together for comparison. The conditions for the simulation are 0.05 Torr, V_o = 65, the scattering angle probability calculated from Vestal *et al.*'s differential cross sections, and a 13.56 MHz spatially linear electric field. Neutrals have larger incident angles than ions, Figure 5.16 (b), because the sheath electric field does not increase their velocity trajectories normal to the surface as it does for the ions. Unlike the IEDs at various incident angles (Figure 5.10), the neutral energy distributions shown in Figure 5.16 (c) have no multiple peaks because neutrals are not accelerated by the time varying electric field. The neutrals only strike the electrode if they do not go through any collisions or go through glancing collisions after they are formed. Since the sheath electric field does not accelerate the neutrals, if the neutrals go through more than two or three collisions, their energies drop below E_{min} or they do not travel toward the electrode. Therefore, although the simulation results in Figure 5.16 is for a 0.05 Torr plasma, the shape of the distributions, a large peak at low energies and fewer ions at higher energies, are representative of neutral distributions at other pressures and sheath voltages.

There are three numbers which are of interest in comparing neutrals with ions: the neutral to ion flux ratio, the average energy, and the average angle. Both the neutrals and ions are compared on the same basis; that is, neutrals with energies above E_{min} are compared to ions with energies above E_{min}.

Figures 5.17 and 5.18 show the energetic neutral properties compared to ions. In these simulations, V_o is set at 400 V, as a realistic voltage applied across commercial etching reactors. E_{min} is set at 20 eV for most cases because ion induced etching rate depends on the energy of the bombarding particles (Gray 1992), and the most reactions require at least 20 eV. Manenschijn and Goedheer (1991) chose 0.1 eV as the minimum neutral energy to consider because they compared their neutral flux to the properties of all the ions. This is appropriate when considering

the energy flux to a surface, but such a low threshold is not appropriate for ion enhanced surface processes which occur in processing, *e.g.* plasma etching.

The ratio of the energetic neutral to ion flux increases with pressure because there are more ion-neutral collisions where ions transfer energy to neutrals. In Figure 5.17, the ratio increases from about 0.5 at 0.01 Torr, where the sheath width is approximately one mean free path long, to 4.5 for pressures above 0.5 Torr. Even at low pressures, the neutral flux constitutes a significant portion of the energetic particles bombarding the surface. As pressure increases, the increase in numbers of energetic neutrals resulting from collisions with ions is counter balanced by the loss of energy from more neutral-neutral collisions in the sheath. Thus, the neutral to ion ratio for energies above 20 eV eventually plateaus.

A better indication of the importance of neutral particles is not the ratio of neutral to ion flux, but changes in the average bombardment energy and angle when neutrals are included as part of the energetic particle flux to the surface. It is clear from Figure 5.18 (a) that at pressures above 0.01 Torr, although neutrals strike the surface with lower energy than ions, there are still enough energetic neutrals that they must be included with the ions when calculating the average energy of bombarding particles. As pressure increases, the number of low energy particles increases but the average ion and neutral energies plateau at 40 eV and 35 eV, respectively, because only particles with energies above 20 eV are included when calculating the average values. So while the number of energetic neutrals formed increases with pressure, many of the neutrals' energies drop below E_{min} from too many collisions and are not included in calculating the average energies.

Figure 5.18 (b) shows the average angle as a function of pressure. Near 0.01 Torr, the average angle of all particles is close to that of the ions alone. As pressure increases, not only does the average angle increase, but the average angle of all particles approaches that of the neutrals. As in the case of average energy, this is because the ratio of neutrals to ions increase

with pressure.

The effect of the type of scattering angle probability on the simulation results are explored by assuming isotropic scattering probability for ion-neutral collisions. All other collision parameters are the same as before. Isotropic scattering occurs in most plasmas used for etching, such as CF_4 or SF_6. Figures 5.19 and 5.20 show the simulation results, still using a 13.56 MHz spatially linear electric field, E_{min} = 20 eV, V_o = 400 V, and V_f = 7 V. Figure 5.19 shows that the neutral to ion ratio remains near 1.5 above 0.05 Torr, unlike the case shown in Figure 5.17. The lower ratio of neutrals to ions with energies above E_{min} results because ions and neutrals are scattered isotropically in a collision, leading to small energy transfer in the direction perpendicular to the electrode, from ion to neutral. Since neutrals are not accelerated by the sheath electric field and they are more likely to be scattered so that their path to the electrode is longer than if they went through a charge-exchange collision, the number of neutrals with energies above 20 eV arriving at the electrode plateaus sooner than the case shown in Figure 5.17. An interesting trend shown in Figure 5.20 is the decrease in the neutral average angle as pressure increases. At low pressures, most of the energetic neutrals formed after a collision with an ion do not go through another collision. Therefore, even energetic neutrals with wide scattering angles make it to the electrode with energies above 20 eV. As pressure increases, energetic neutrals with wide scattering angles have longer paths to the electrode and thus go through more collisions with the background gas than energetic neutrals with trajectories perpendicular to the electrode. Therefore, the energetic neutrals with wide scattering angles rarely make it to the electrode with energies above 20 eV. The ions with energies above 20 eV correspond to the ones with small incident angles, resulting in a decrease in average neutral angle as pressure increases.

Since an E_{min} of 20 eV is arbitrarily chosen, Figure 5.21 shows how choosing a cut off energy, E_{min}, affects the calculated average energetic particle bombardment energy and angle. The

results shown here are calculated using a 13.56 MHz spatially linear electric field, V_o = 400 V, V_f = 7 V; and ion-neutral collisions obey the scattering angle probability calculated from Vestal *et al.*'s data, while the neutral-neutral collisions obey the isotropic scattering probability. In Figure 5.21 (a), the average energy of all particles with E_{min} = 5 eV, 10 eV, or 20 eV decreases with pressure in a similar way. Choosing a different cut off energy for ions and neutrals merely changes the calculated average energy, but maintains the same trend. The difference in average energies calculated with various E_{min}'s at a given pressure is just slightly greater than the difference in the E_{min}'s themselves, with a lower average energy corresponding to a lower E_{min}. Figure 5.21 (b) shows that the average angle increases when a lower E_{min} is used. Including lower energy ions increases the calculated average angle because ions with lower energies generally have gone through more collisions and thus have high bombardment angles.

5.6 CONCLUSION

Monte Carlo simulations can be used to predict ion energy and angle if the sheath electric field, the sheath width, pressure, and the total and differential collision cross sections are known for all types of collisions in the sheath. The simulation results show the same trends and the multiple peaked IEDs as experimental data. However, the magnitude of the average ion energy and angle depend strongly upon the angular scattering probability and the spatial form of the electric field assumed. Predictions of average ion angle from simulations with a preferentially forward and backward scattered differential cross section and an isotropic differential cross section bound the experimentally measured values. The effect of the spatial form of the electric field on the ion angle is minor in comparison. Argon IADs previously reported in literature have always assumed a separation of charge-exchange and hard sphere collisions. These IADs are not similar to measured IADs. This chapter has shown that simulations using the scattering probability

calculated from Vestal *et al.*'s (1978) measured differential cross sections, where ions are preferentially forward and back scattered, agree better with the experimentally measured IADs than previous simulations. Monte Carlo simulations using either a spatially uniform or linear sheath electric field under-predict the measured ion energy, though predictions using a spatially linear electric field come closer to measurements. This discrepancy may result from the presence of Ar^{2+} ions in the measurements.

Simulations using a dc sheath electric field captures all the trends in the measured IEDs and IADs, and the calculated average ion energy and angle are the same for both dc and 13.56 MHz time varying sheath electric field simulations. The only shortcoming of a dc field simulation is the failure to reproduce the multiple peaked structure in the IEDs.

Monte Carlo simulations show that the number of energetic neutrals can be equal to or greater than the number of energetic ions in plasmas where there are charge-exchange collisions. The simulations for argon plasmas at high pressures show that the number of energetic neutrals is about four and a half times that of ions, when comparing ions and neutrals with energies greater than 10% of the time averaged sheath potential. This ratio decreases to 1.5 when there are no charge-exchange collisions.

The average neutral angle is always higher than the average ion angle because, unlike the ions, the neutrals are not accelerated toward the electrodes. Since for pressures above 0.01 Torr, the neutral flux can be comparable to the ion flux, the neutrals can have a significant effect on the etching direction, creating isotropic profiles. Anisotropic etching profiles require low operating pressures, not only to decrease the ion bombardment angle, but also to decrease the energetic neutral flux to the surface.

The neutral energy distribution at all pressures and sheath voltages all look similar because after an energetic neutral is formed from a collision with an ion, it only loses energy through

collisions. Therefore, all the energy distributions show that there are many particles at the low energy end of the distribution, with the number of neutrals tapering off as energy increases.

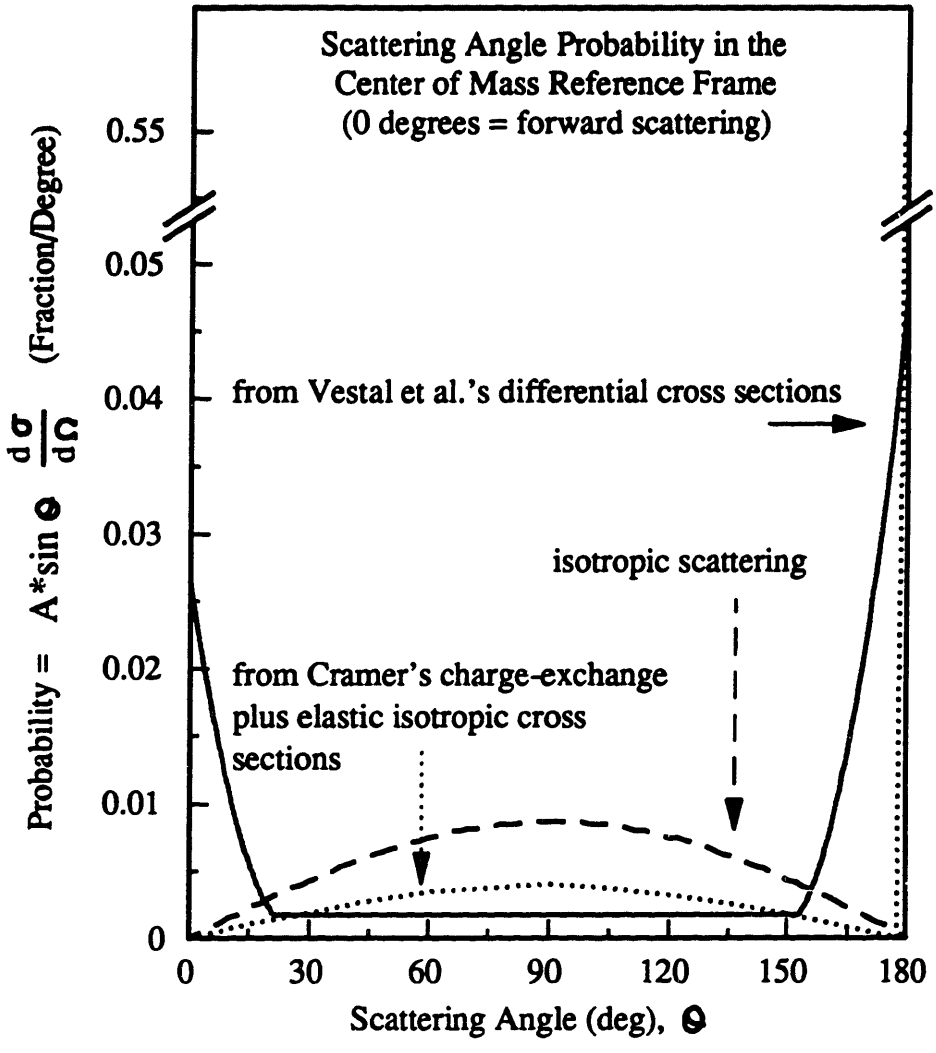


Figure 5.1: The angular scattering probability of ions when they collide with neutrals. The angular scattering is given in the center of mass reference frame; 0 degrees corresponds to the ion being forward scattered. The constant A is found by integrating the probability and setting it equal to 1. The solid, dashed, and dotted lines indicate the scattering probability calculated using Vestal et al.'s (1989) data, assuming isotropic scattering, and using Cramer's separation of charge-exchange and isotropic scattering data, respectively.

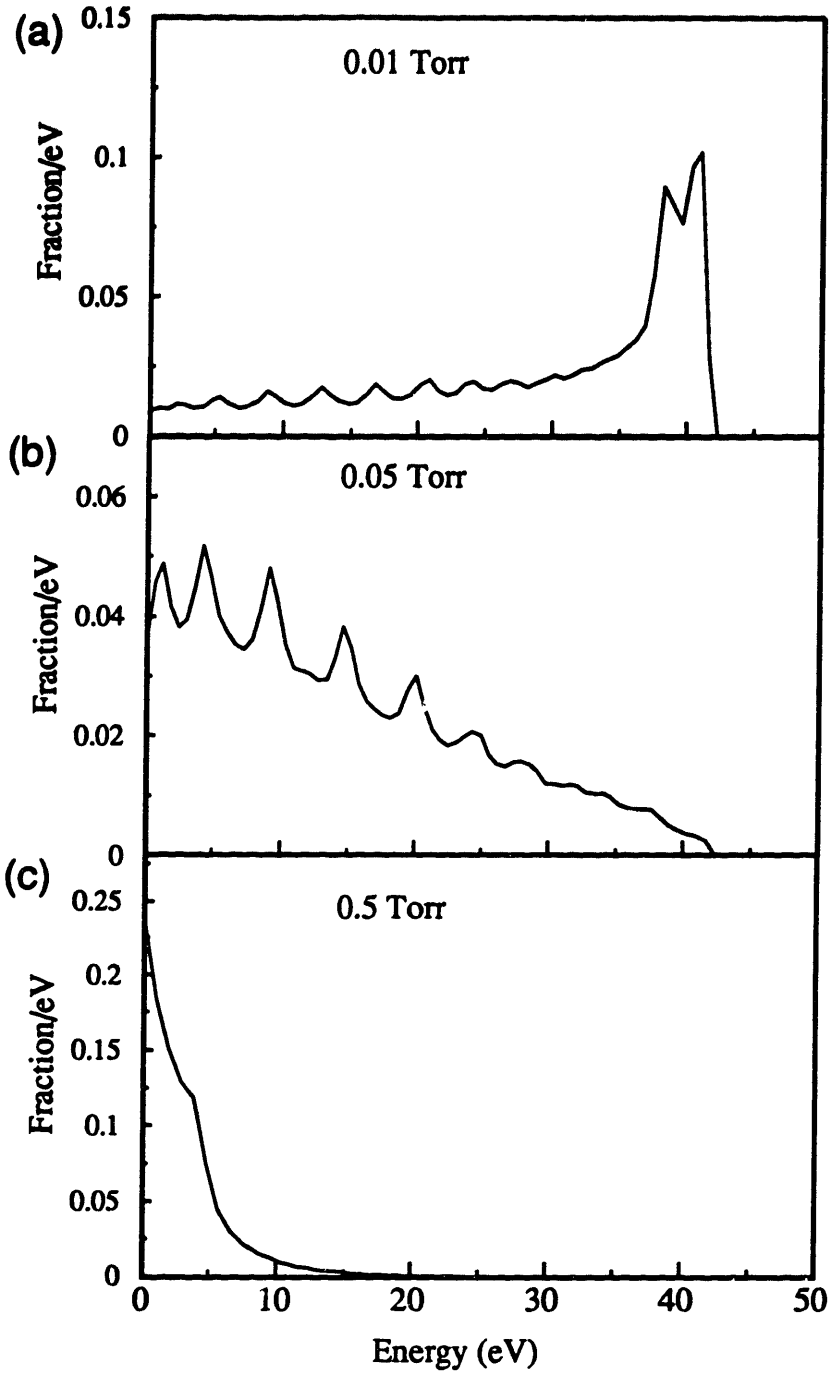


Figure 5.2: Simulated total IEDs at Vo=65 and Vf=7, using a 13.56 MHz spatially linear electric field, and Cramer's (1959) separation of charge-exchange and hard sphere collision cross sections at (a) 0.01 Torr (b) 0.05 Torr and (c) 0.5 Torr.

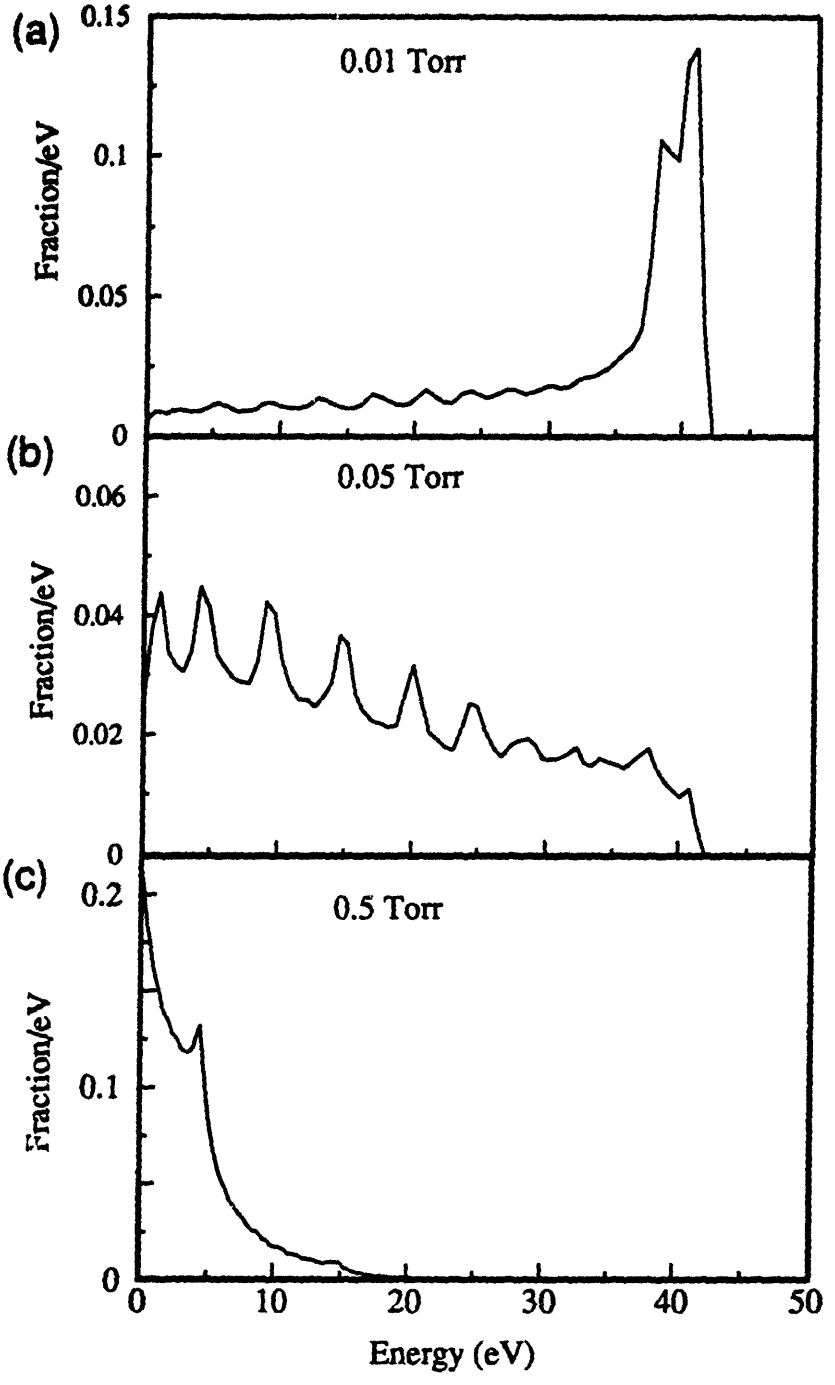


Figure 5.3: Simulated total IEDs at Vo=65 and Vf=7, using a 13.56 MHz spatially linear electric field, and Vestal et al's (1978) differential cross sections at (a) 0.01 Torr (b) 0.05 Torr and (c) 0.5 Torr.

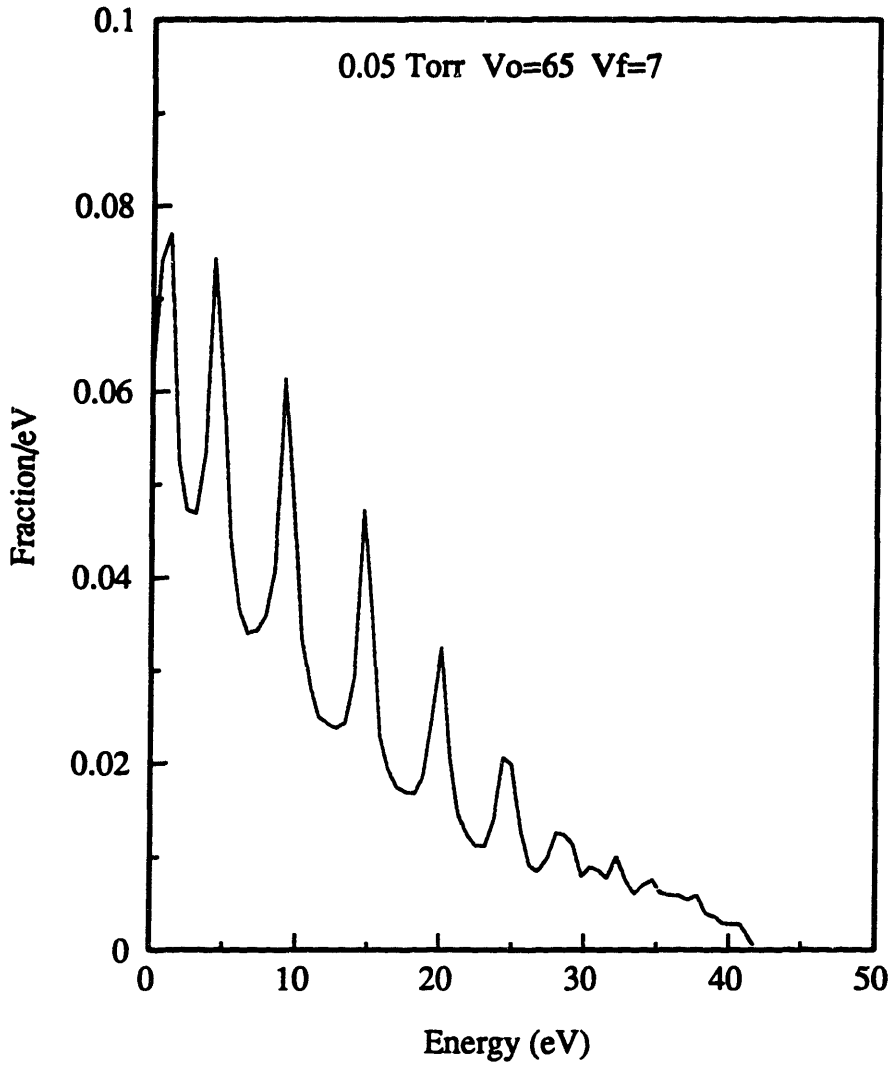


Figure 5.4: Monte Carlo simulation of an IED using only charge-exchange collisions, i.e. no momentum transfer during collisions. Simulation conditions are a 13.56 MHz spatially linear electric field, 0.05 Torr, Vo=65, and Vf=7.

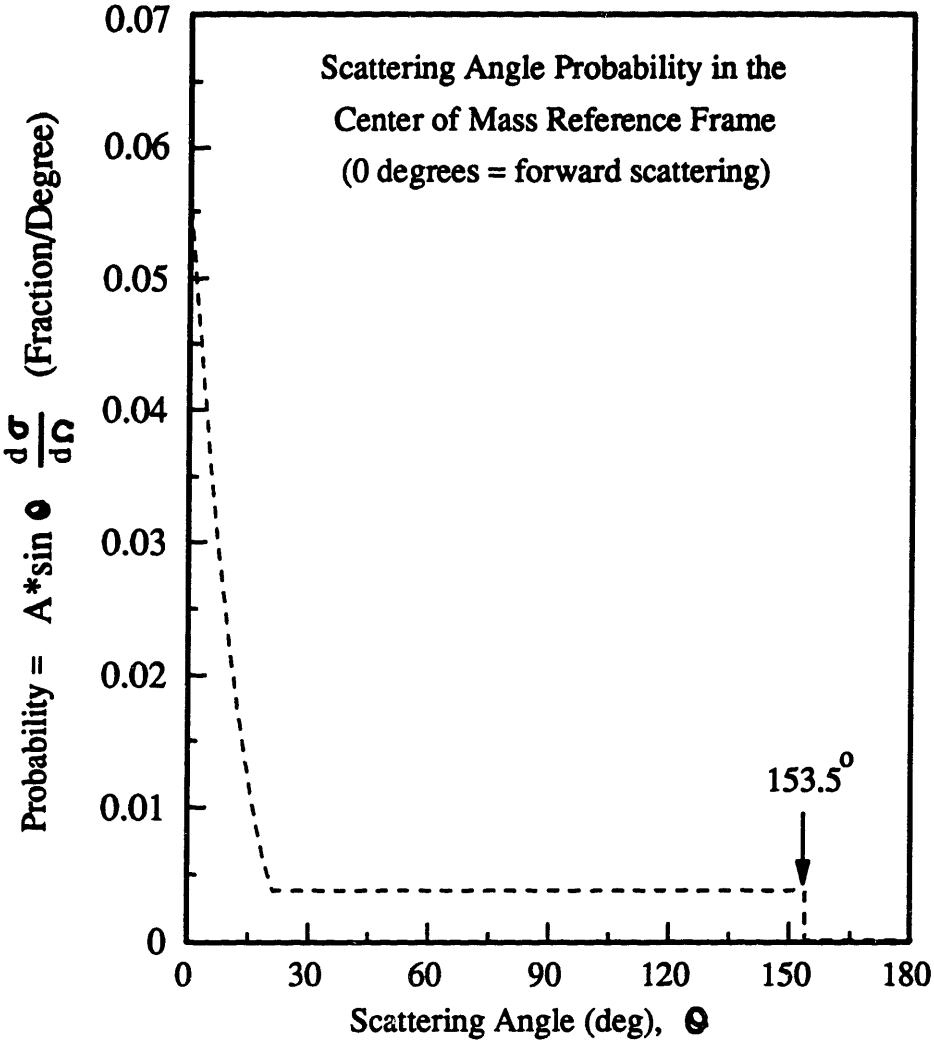


Figure 5.5: The same angular scattering probability as in Figure 5.1, except the back scattering probability from 153.5 to 180 degrees has been set to zero. The angles are given in the center of mass reference frame.

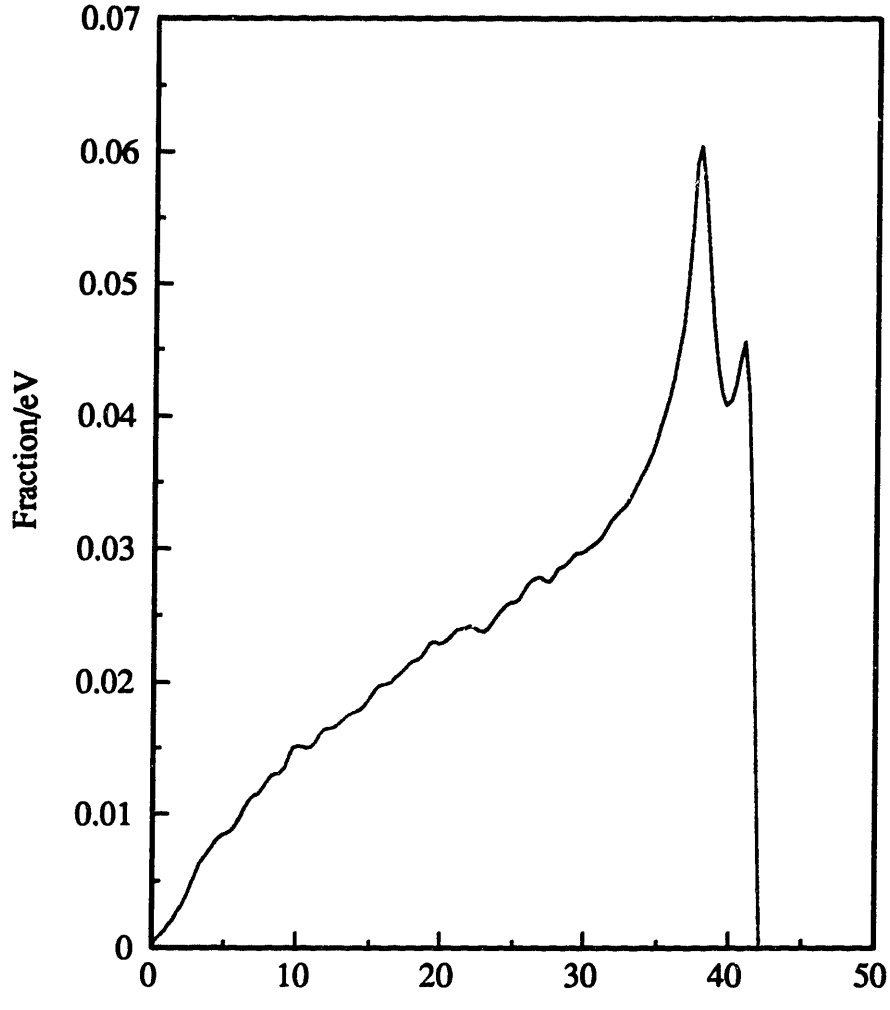


Figure 5.6: IED calculated using the angular scattering probability shown in Figure 5.5. The simulation conditions are 0.05 Torr, Vo=65, Vf=7, and 13.56 MHz spatially linear electric field.

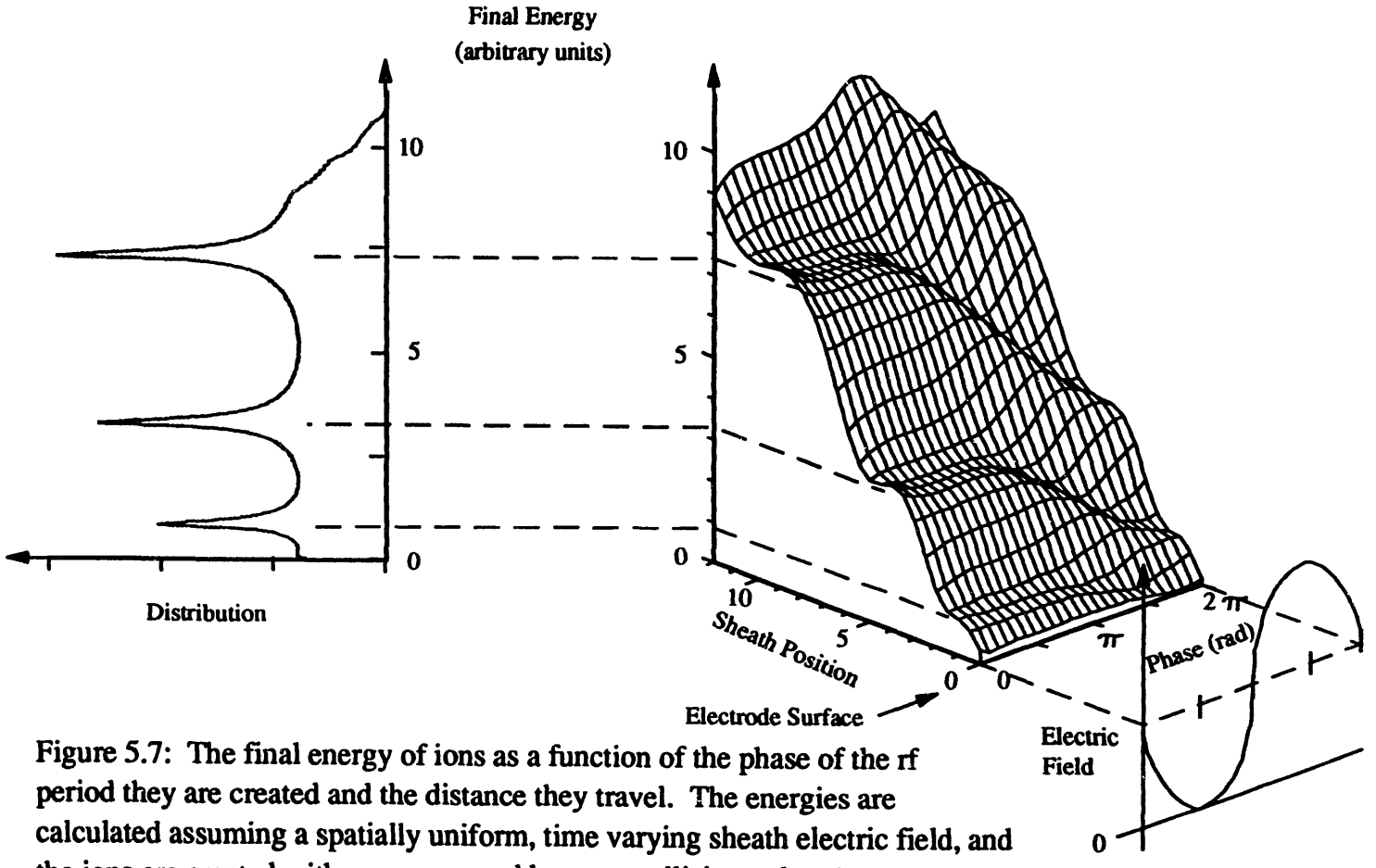


Figure 5.7: The final energy of ions as a function of the phase of the rf period they are created and the distance they travel. The energies are calculated assuming a spatially uniform, time varying sheath electric field, and the ions are created with no energy and have no collisions after they are created. The distribution is derived by assuming the ions are created uniformly in space and time. The sinusoidal curve indicates the magnitude of the electric field at the time the ion is created. (Graphs from Huppert 1992)

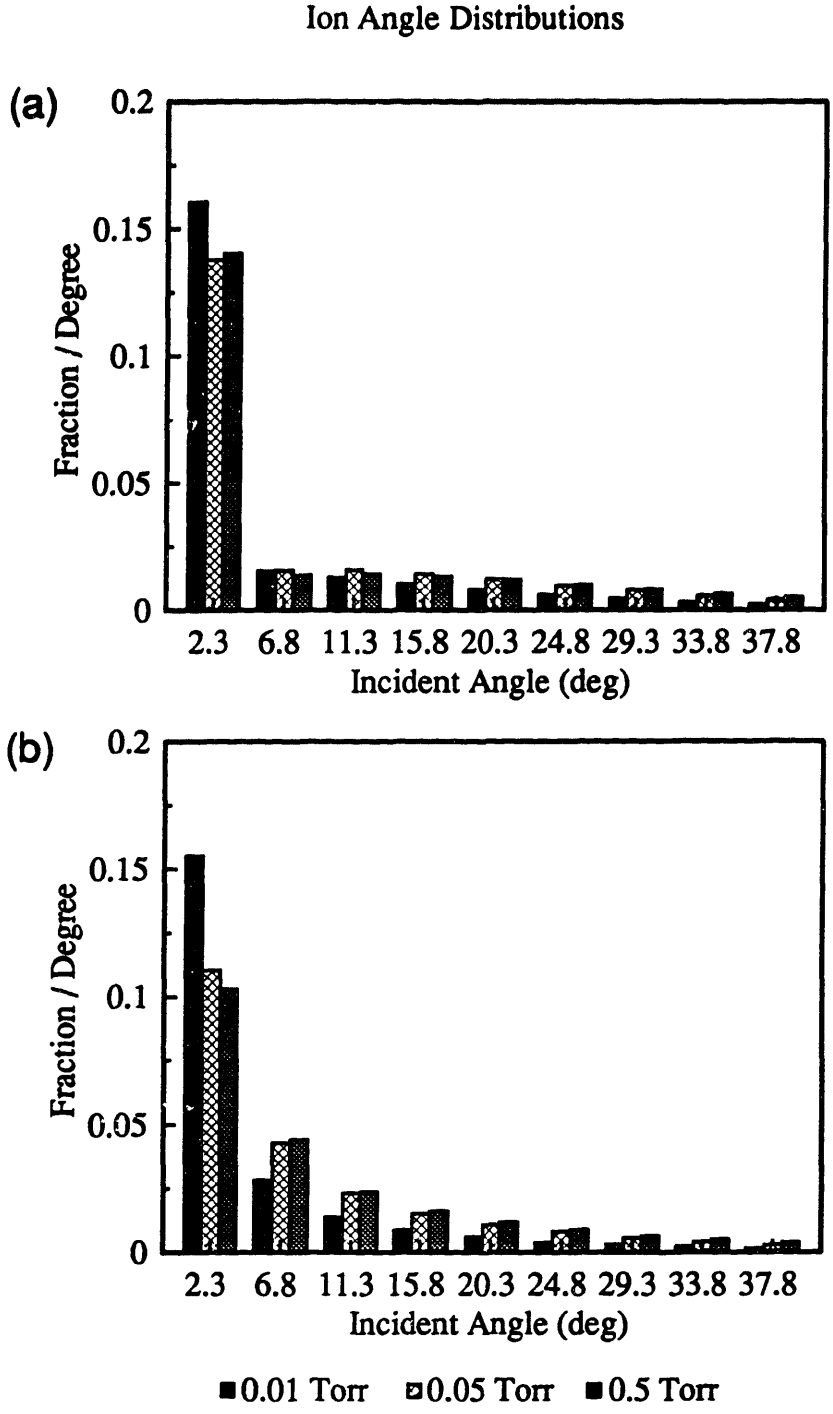


Figure 5.8: Simulated IADs at 0.01, 0.05, and 0.5 Torr, and Vo=65, Vf=7, and 13.56 MHz spatially linear electric field. The distributions are azimuthally integrated. (a) Cramer's (1959) cross sections (b) Vestal et al.'s (1978) differential cross sections.

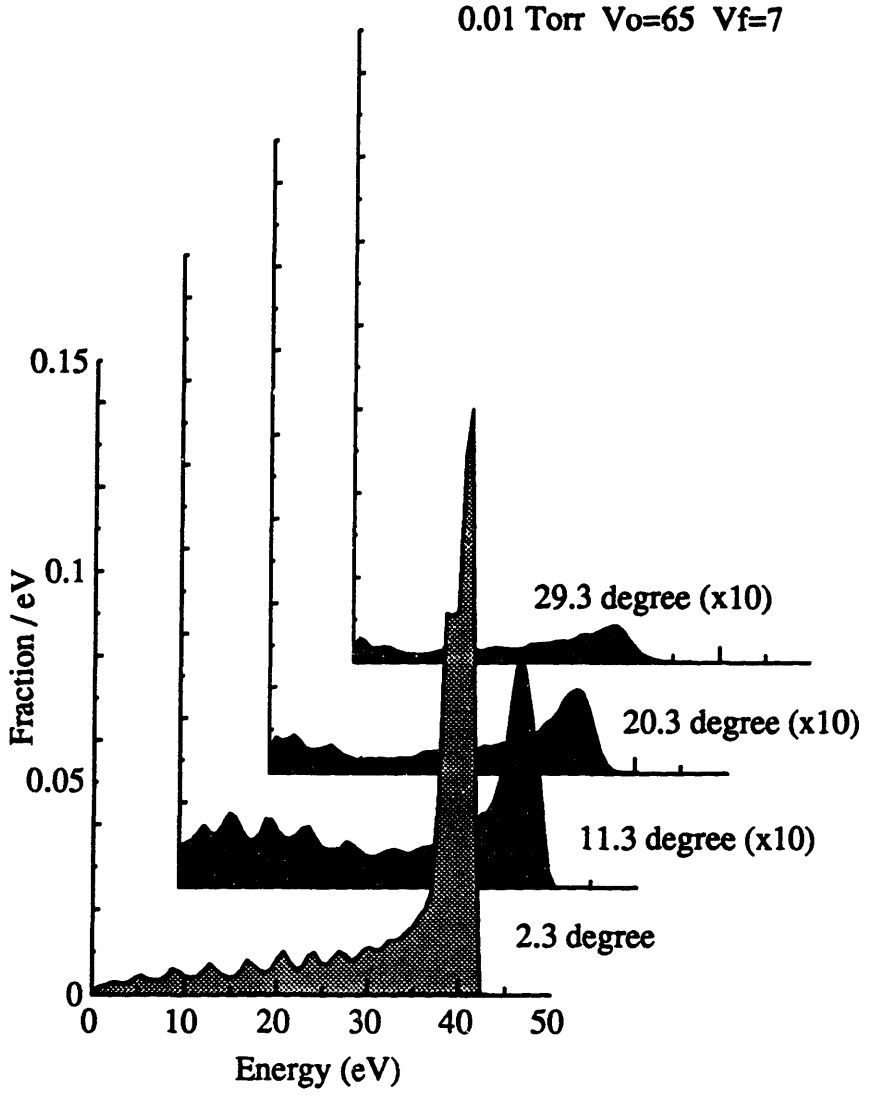


Figure 5.9: IEDs at various incident angles for 0.01 Torr, Vo=65, and Vf=7. The y-axis of the simulated distributions at 11.3, 20.3, and 29.3 degrees are magnified 10 times. The simulation used a 13.56 MHz spatially linear electric field and Vestal et al.'s (1978) differential cross sections.

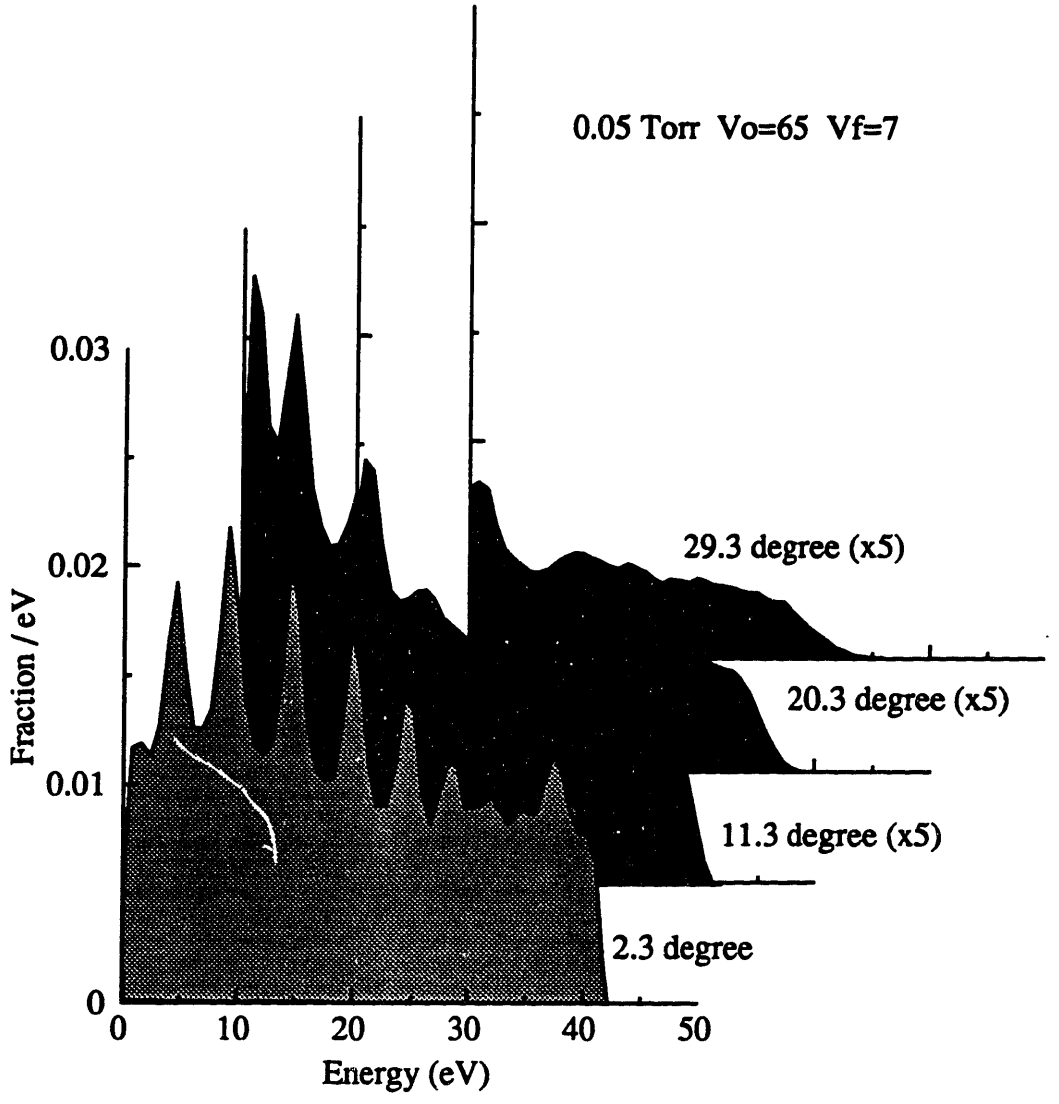


Figure 5.10: IEDs at various incident angles for 0.05 Torr, Vo=65, and Vf=7. The y-axis of the simulated distributions at 11.3, 20.3, and 29.3 degrees are magnified by five. The simulation used a 13.56 MHz spatially linear electric field and Vestal et al.'s (1978) differential cross sections.

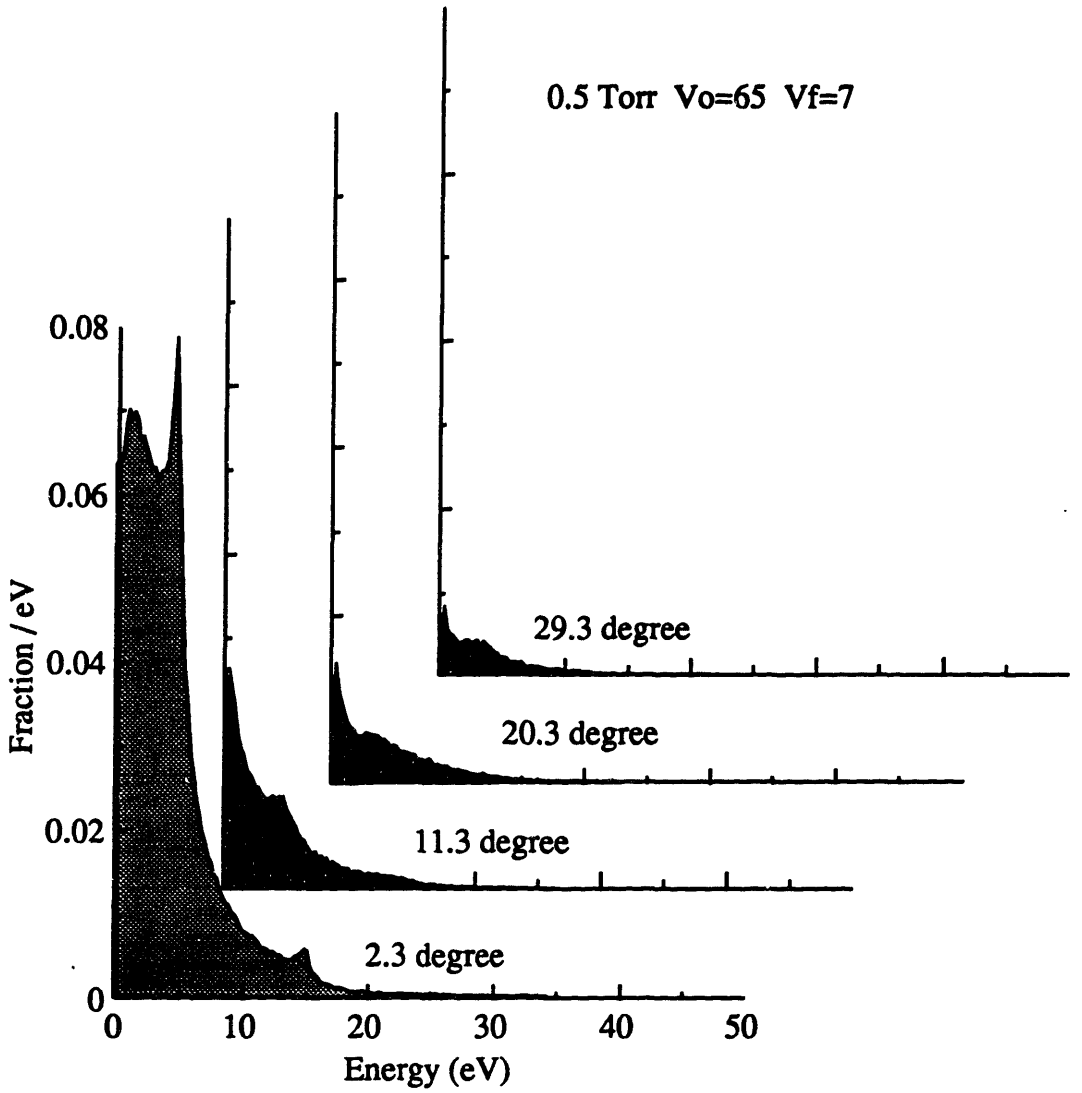


Figure 5.11: IEDs at various incident angles for 0.5 Torr, Vo=65, Vf=7. The simulation used a 13.56 MHz spatially linear electric field and Vestal et al.'s (1978) differential cross sections.

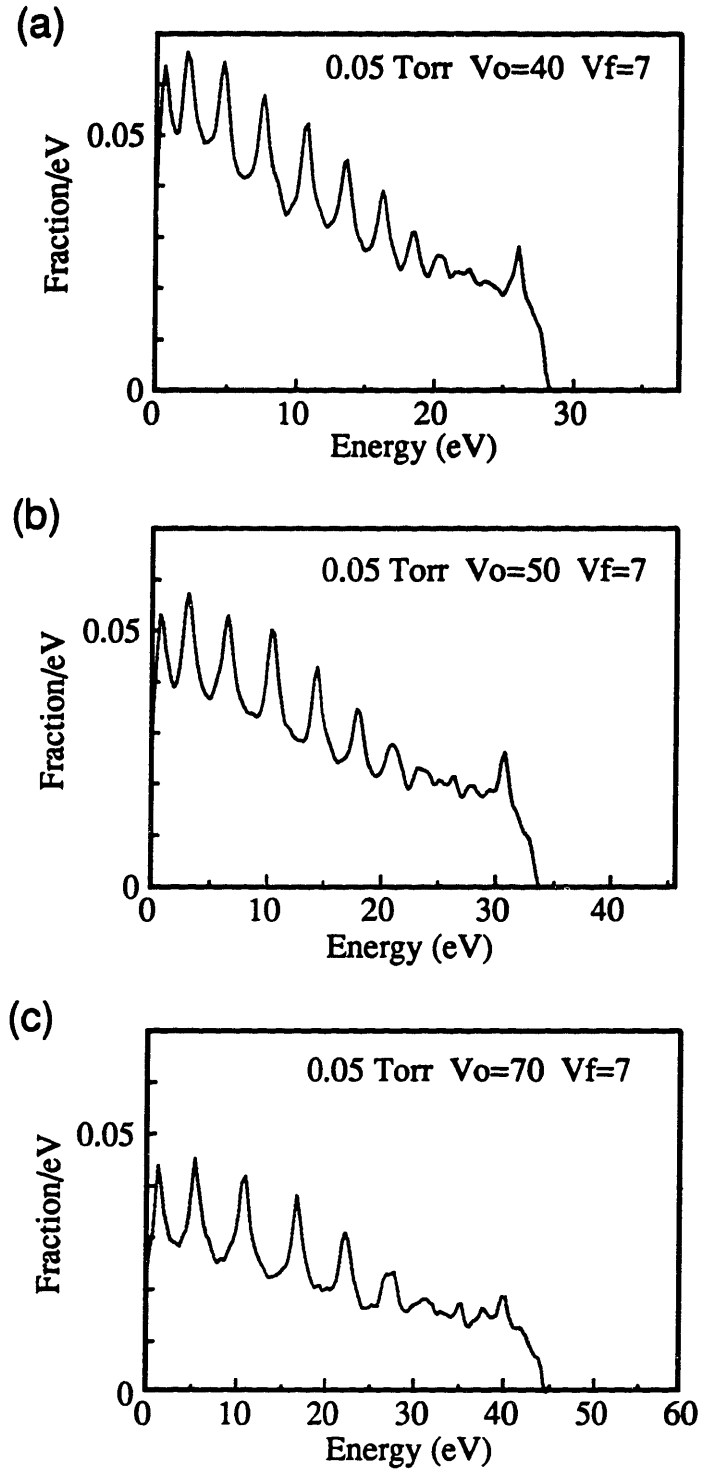


Figure 5.12: IEDs at 0.05 Torr and various Vo's. Simulation conditions are a 13.56 MHz spatially linear electric field and Vestal et al.'s differential cross sections.

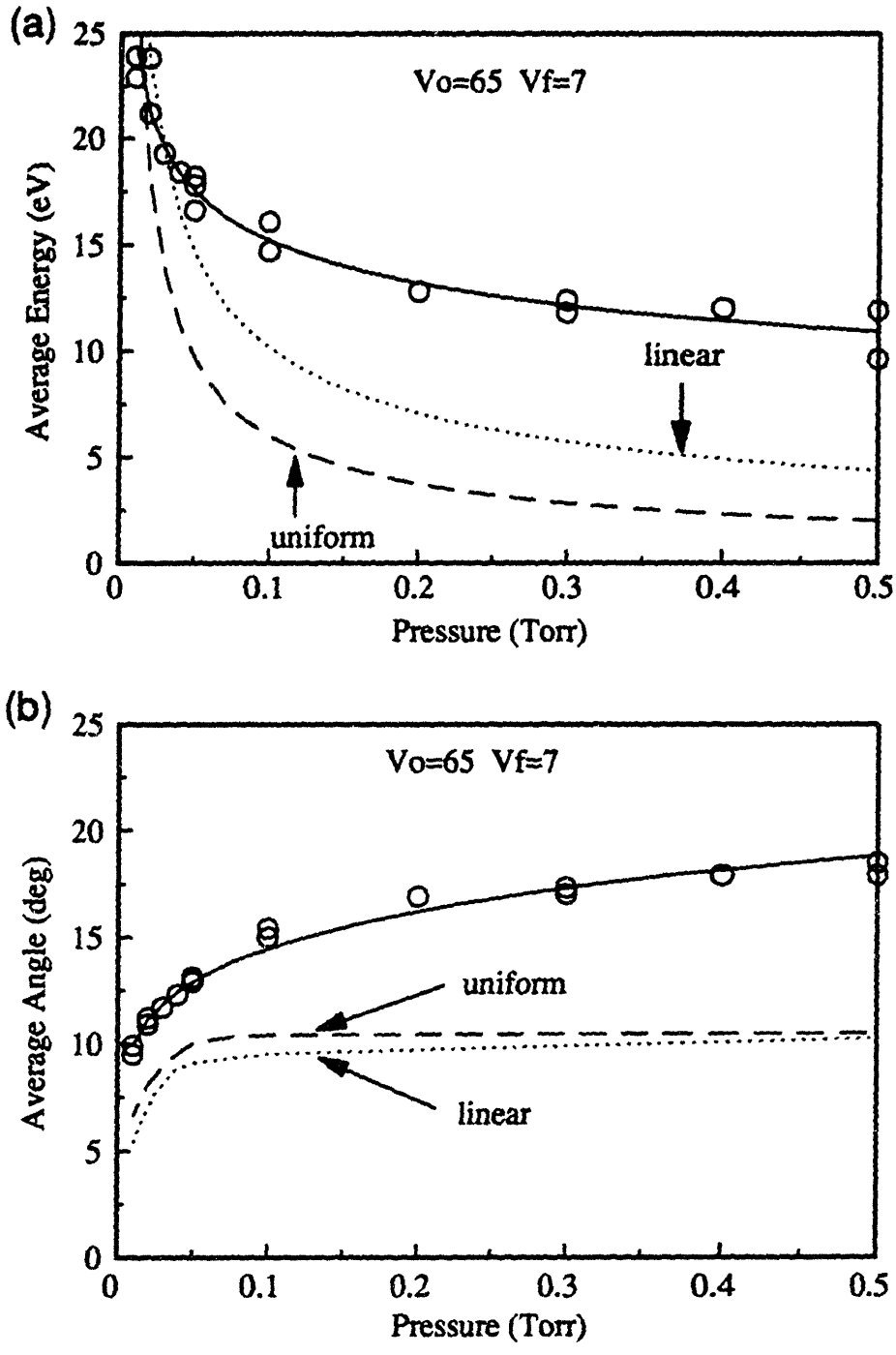


Figure 5.13: The simulated average ion properties compared to experimental data. The simulation conditions are Vo=65, Vf=7, 13.56 MHz field, and Vestal et al.'s (1978) differential cross sections. The dashed line and dotted lines correspond to spatially uniform and linear simulation results, respectively. The circles are data. (a) average ion energy (b) average ion angle.

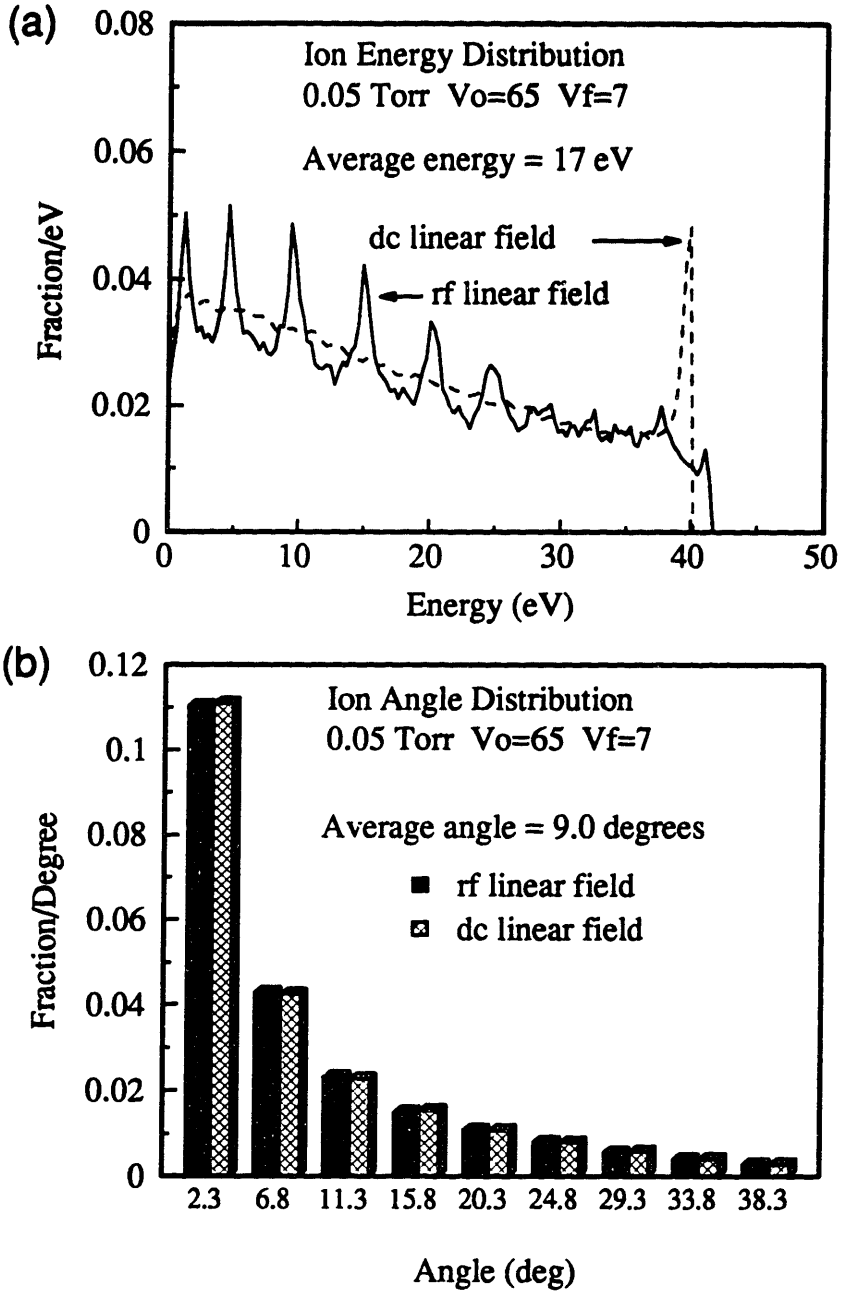


Figure 5.14: Comparison of 13.56 MHz and dc spatially linear electric field simulation results. The simulation conditions are 0.05 Torr, Vo=65, and Vf=7. (a) IEDs; dashed line indicates dc field and solid line indicates 13.56 MHz field results (b) azimuthally integrated IADs; cross-hatched boxes indicate dc field results and solid boxes indicate 13.56 MHz field results.

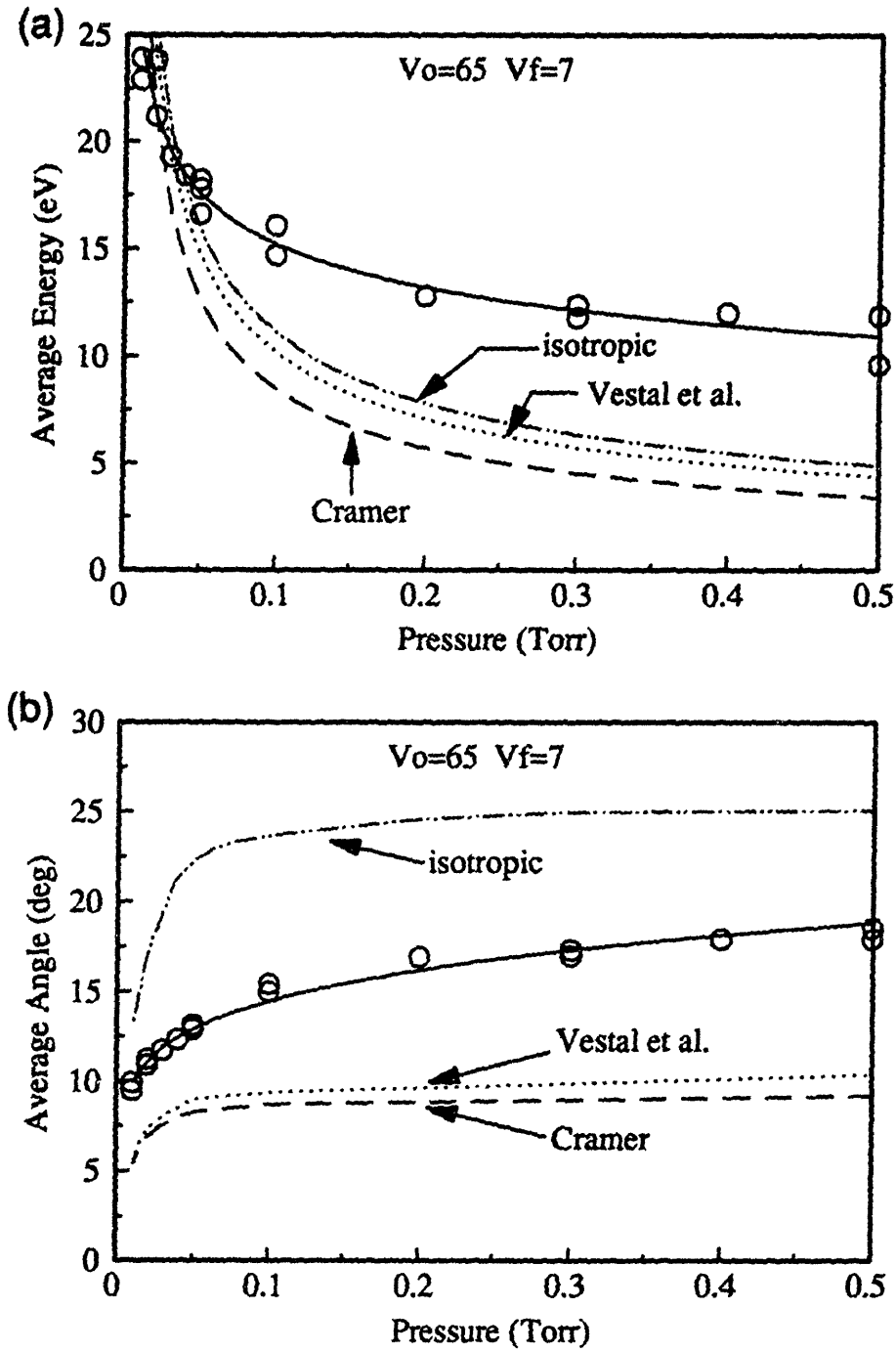


Figure 5.15: The effect of the type of scattering probability on the simulated average ion (a) energy and (b) angle. The simulation conditions are a 13.56 MHz spatially linear electric field, Vo=65, and Vf=7.

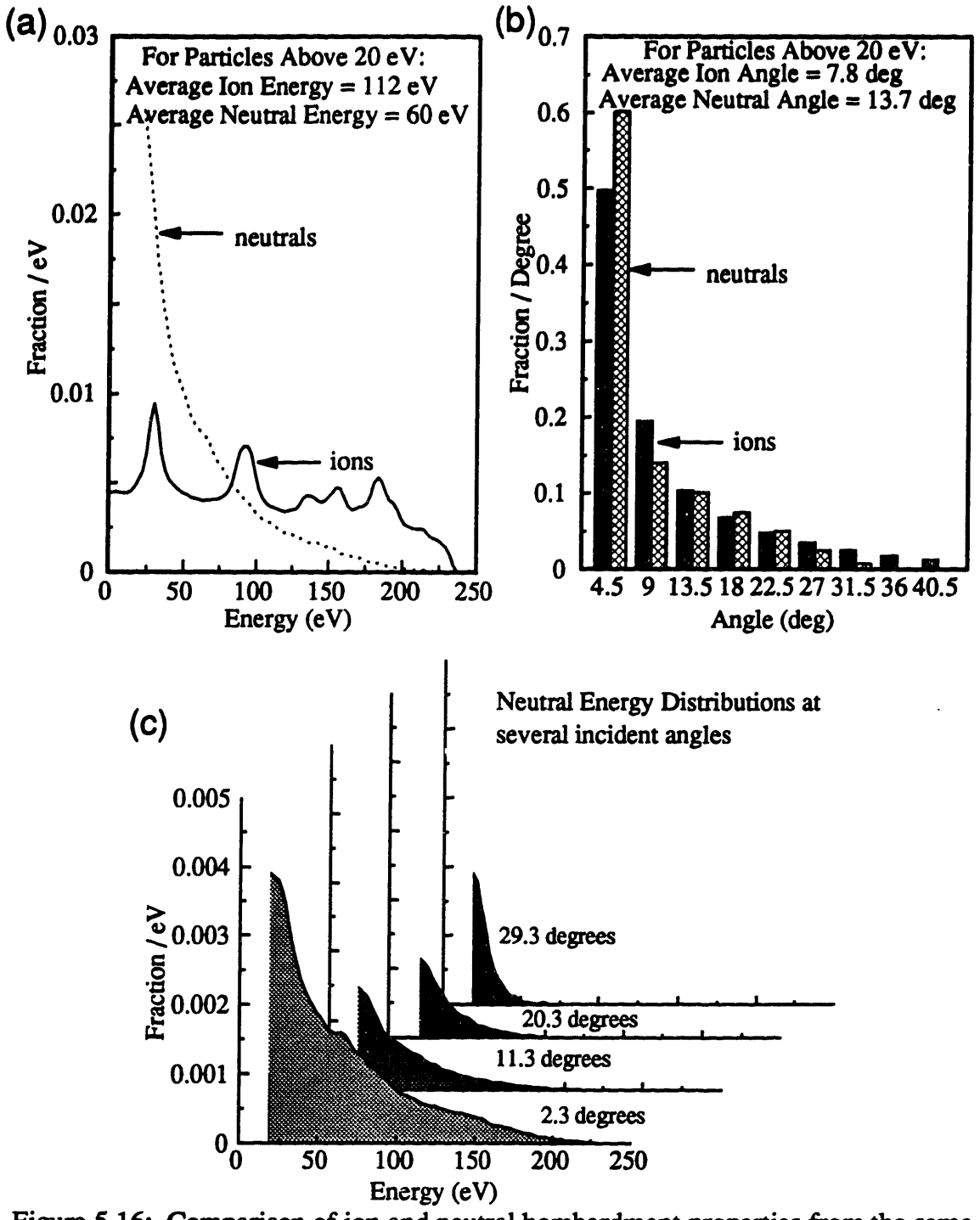


Figure 5.16: Comparison of ion and neutral bombardment properties from the same plasma. Simulation conditions are 0.05 torr, Vo=400, Vf=7, 13.56 MHz spatially linear electric field, and Emin = 20 eV; Vestal et al.'s differential cross sections and Cramer's total cross section for ion-neutral collisions; isotropic scattering angle probability and 10 A^2 total cross section for neutral-netural collisions.

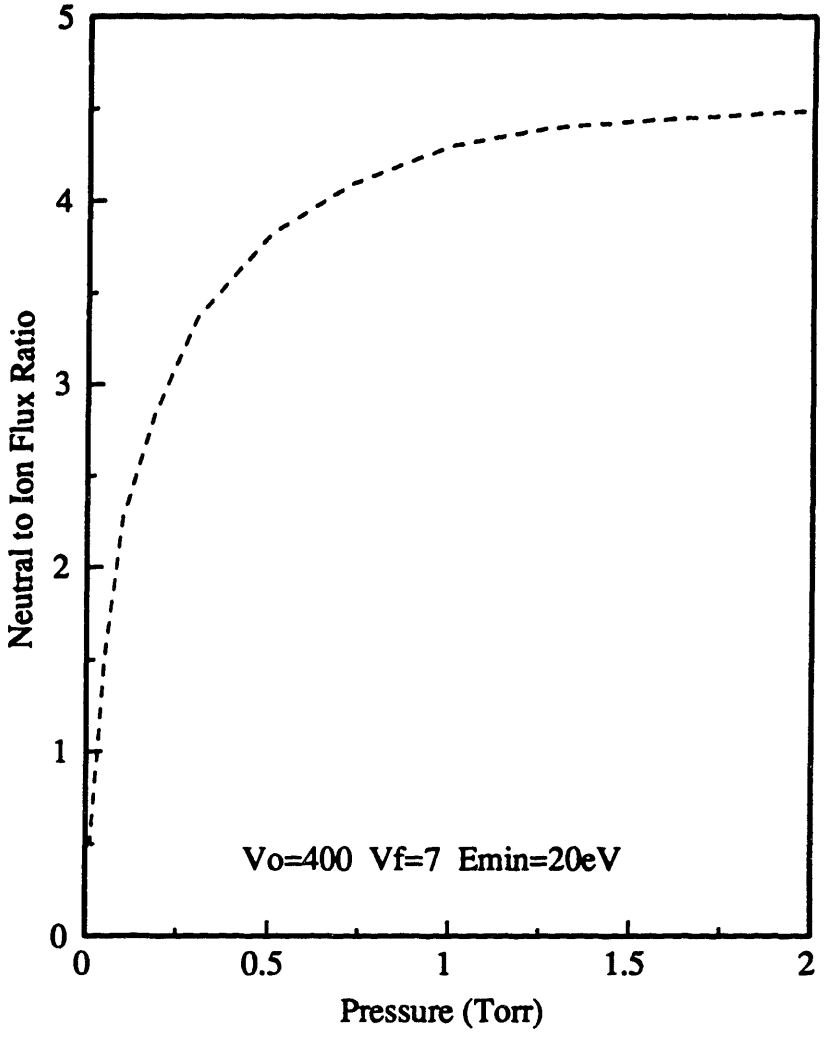


Figure 5.17: Ratio of the number of neutrals to ions with energies above a Emin of 20 eV. The simulation conditions are Vo=400, Vf=7, and 13.56 MHz spatially linear electric field; Vestal et al.'s (1978) differential and Cramer's (1959) total cross sections for ion-neutral collisions; isotropic scattering and 10 A^2 total cross section for neutral-neutral collisions.

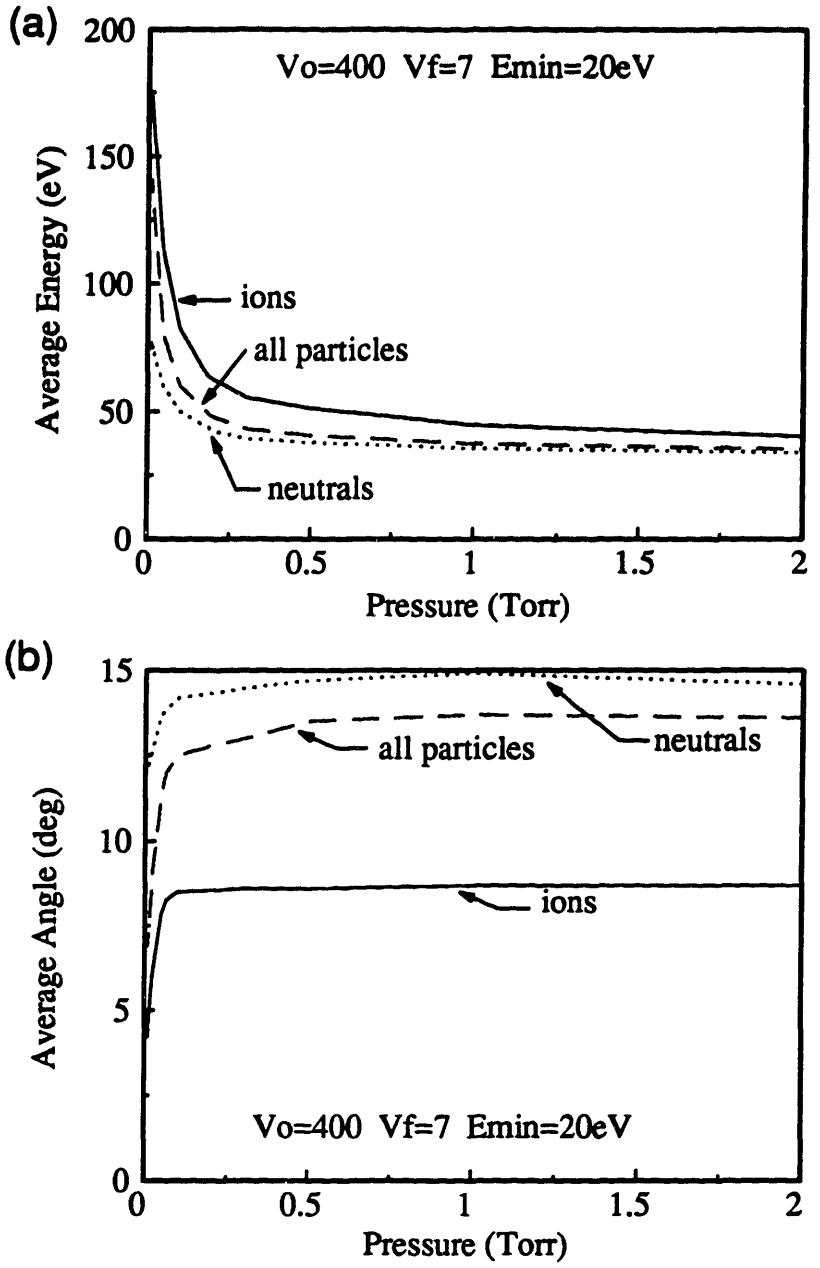


Figure 5.18: The contribution of both ions and neutrals to the total energetic particle bombardment on the surface. The simulation conditions are Vo=400, Vf=7, Emin=20eV, 13.56 MHz spatially linear electric field; Vestal et al.'s (1978) differential and Cramer's (1959) total cross sections used for ion-neutral collisions; isotropic scattering and 10 A^2 cross section used for neutral-neutral collisions. (a) average energy (b) average angle, with 0 degrees corresponding to normal incidence.

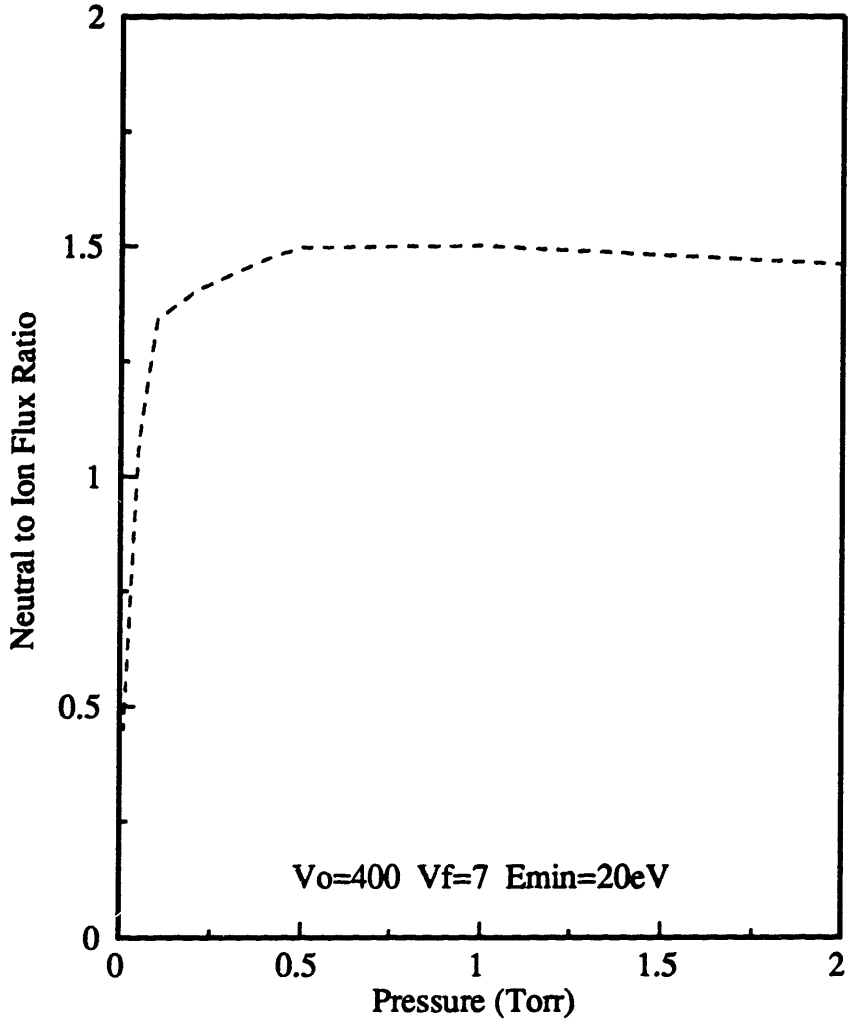


Figure 5.19: Ratio of the number of neutrals to ions with energies above Emin of 20 eV. The simulation conditions are Vo=400, Vf=7, and 13.56 MHz spatially linear electric field; Cramer's (1959) total cross section for ion-neutral collisions, 10 A^2 for neutral-neutral collisions; both types of collisions used isotropic scattering.

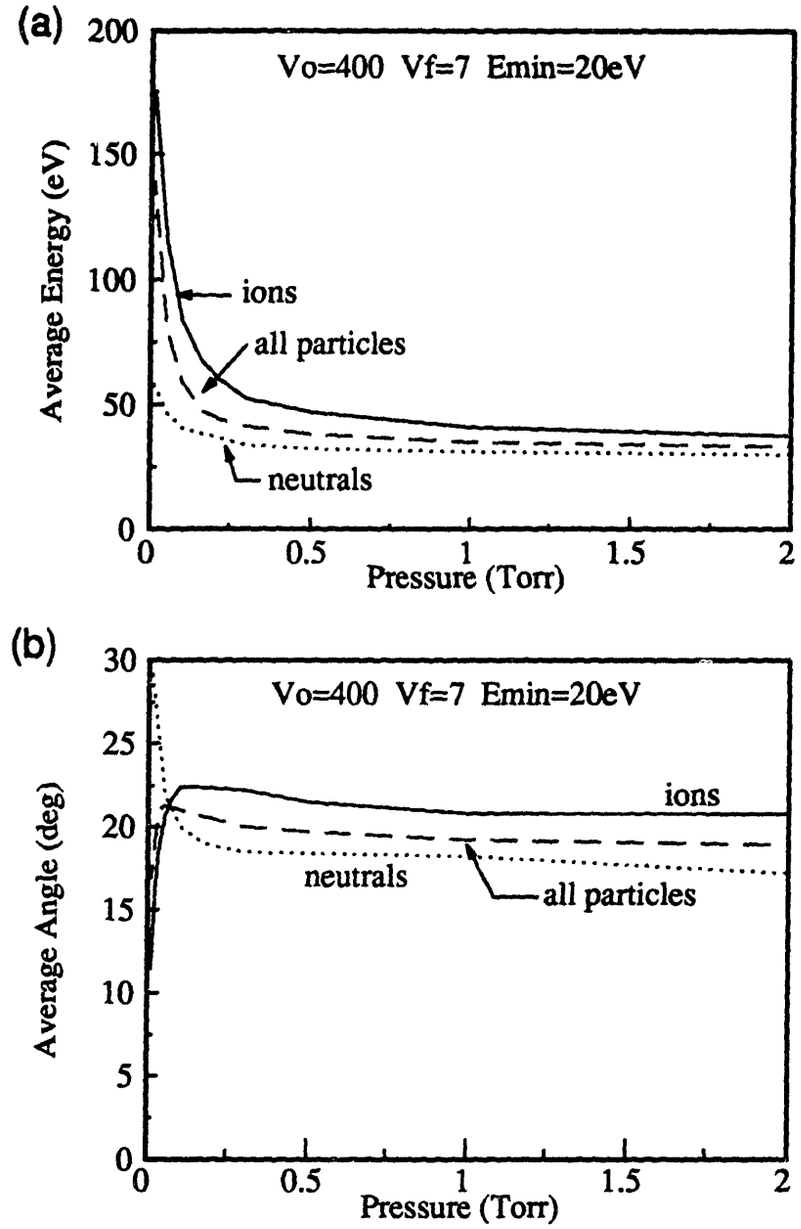


Figure 5.20: The contribution of both ions and neutrals to the total energetic particle bombardment on the surface. The simulation conditions are Vo=400, Vf=7, Emin=20eV, 13.56 MHz spatially linear electric field; Cramer's (1959) total cross section used for ion-neutral collisions and 10 A^2 used for neutral-neutral collisions; both types of collisions used isotropic scattering.
(a) average energy (b) average angle, with 0 degrees corresponding to normal incidence.

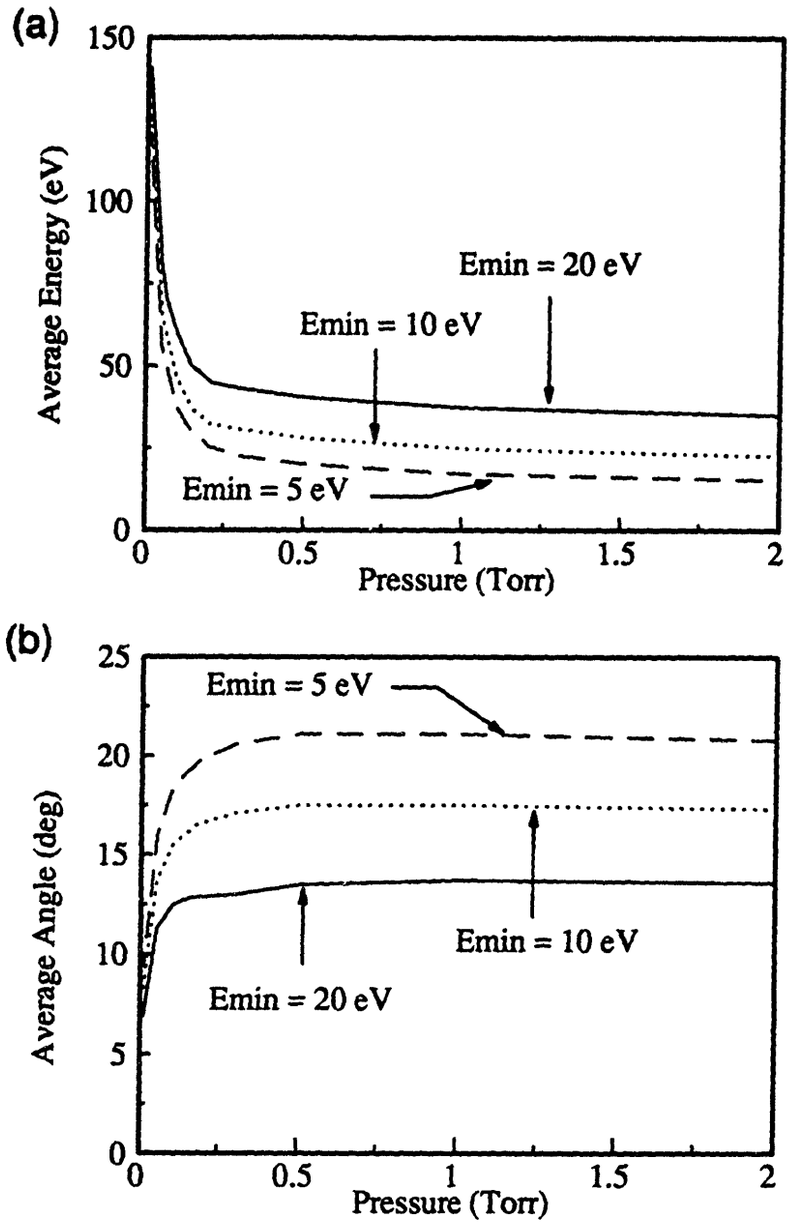


Figure 5.21: The effect of Emin on the calculated average energetic particle bombardment properties. The simulation conditions are Vo=400, Vf=7, and 13.56 MHz spatially linear electric field; Vestal et al.'s differential and Cramer's total cross sections for ion-neutral collisions; 10 A^2 total cross section and isotropic scattering for neutral-neutral collisions. (a) average energy (b) average angle.

CHAPTER 6

MAGNETIC FIELD EFFECTS ON PLASMA PROPERTIES

As microelectronic devices reach submicron dimensions, plasma etching steps require more normally directed energetic particle flux on the wafer to achieve accurate transfer of the mask pattern onto the film. This usually means operating the plasma at very low pressures, *e.g.* 0.01 Torr, to ensure that ions have few if any collisions in the sheath. The resulting energetic ion bombardment is directed perpendicular to the wafer surface. Thus etching rate is enhanced in the vertical direction. However, etching rate decreases as pressure decreases because the plasma density also decreases, resulting in both lower radical and ion flux to the surface. Increased reactant flux to the surface of wafers, and therefore increased etching rate, can be achieved by increasing power dissipation in the plasma or by adding magnetic fields to the plasma. However, increasing power also means increasing the sheath voltage, and thus, the average ion energy striking the surface. High ion energy bombardment on the wafer surface can result in the fabrication of defective devices (Strunk *et al.* 1988). Thus, the use of magnetic fields to increase plasma density without increasing damage is studied.

6.1 BACKGROUND

Increased power is used to increase plasma density. In a low pressure plasma, an alternative is the addition of a magnetic field parallel to the wafer or electrode surface. Adding a magnetic field at a fixed power, as opposed to increasing power, has the advantage of increasing reactant flux to the surface without necessarily increasing ion bombardment energy. Adding the magnetic field decreases electron loss to the electrode surfaces, thereby increasing the electron density in the plasma. This leads to more electron impact reactions such as ionization and

dissociation. For a fixed plasma power consumption, the sheath voltage decreases with magnetic field strength (Yeom *et al.* 1989, McNevin *et al.* 1992). The net result is an increase in the plasma density as well as a decrease in ion bombardment energy. Therefore, the advantage of using a magnetic field in a plasma over increasing power is that the magnetically enhanced plasma creates the possibility of high etching rates without the ion induced damage on wafer surfaces. Increasing the magnetic field increases not only the radical and ion flux to the surface, but it can also increase the ion to radical flux ratio (Heinrich & Hoffmann, 1992). Thus, a second advantage of using a magnetically enhanced plasma is that ion induced etching reactions can dominate on the wafer surface, leading to anisotropic etching profiles.

The magnetic field used in microelectronics fabrication, which ranges from 0 to 200 Gauss, affects only the electrons, not the ions. For the magnetic field to have any effect on the plasma, the Larmor radius of the charged species must be the same order of magnitude or less than the particle's collision mean free path. The Larmor radius, r_L, and the collision mean free path, λ_m, can be written as follows:

$$r_L = \frac{m_j v_j}{qB} \qquad \lambda_m = \frac{1}{\sigma_j N} \tag{6.1}$$

where m_j, v_j, and q are the mass, velocity, and charge of the particle, respectively; σ_j is the collision cross section of the particle; and N is the gas density. The ion Larmor radius at 200 Gauss is greater than the dimensions of most plasma reactors, and is always greater than the ion collision mean free path. Consequently, the magnetic field has no effect on the ion trajectory in the plasma. On the other hand, the electron mass is four to five orders of magnitude lighter than the ion mass, and hence the magnetic field can affect the electron motion in the plasma. For a 200 Gauss magnetic field and a 0.5 Torr plasma, the Larmor radius of the electron is approximately the same order of magnitude as its collision mean free path. Increasing pressure

beyond 0.5 Torr at a fixed 200 Gauss field or decreasing magnetic field strength below 200 Gauss at a fixed pressure of 0.5 Torr, both have the same effect of making the electron mean free path smaller than the Larmor radius, thus negating the effect of the magnetic field on the plasma. The motion of the electrons between collisions is governed by the following force balance equation:

$$F = m\frac{d\vec{v}}{dt} = q\vec{\mathcal{E}}(t) + q\vec{v} \times \vec{B} \; . \qquad (6.2)$$

This equation indicates that the magnetic field forces the electron in the $\vec{v} \times \vec{B}$ direction, that is, the electron path winds around the magnetic field as shown in Figure 6.1. The electrons also drift in the $\vec{\mathcal{E}} \times \vec{B}$ direction parallel to the electrode surface. This $\vec{\mathcal{E}} \times \vec{B}$ drift is seen when the magnetic field is applied to a low pressure plasma, such as argon at 0.05 Torr. Looking along the magnetic field lines, the sheath width is narrower on one side of the electrode than on the other, indicating that the plasma density is greater on one side than the other. This effect can be reversed simply by changing the direction of the magnetic field.

Several workers have measured the effect of the magnetic field on etching rates, confirming that etching rates can be increased with the addition of magnetic field to low pressure plasmas. Nguyen *et al.* (1990) showed that the etching rate of polysilicon in Cl_2 plasmas increased with power and magnetic field strength. For pressures greater than 0.1 Torr, however, the magnetic field had no observable effect on the etching rate, probably because at this pressure and with a maximum field strength of 140 Gauss, the electron mean free path is smaller than its Larmor radius. The wafer surfaces etched in a magnetically enhanced plasma showed less lattice disorder, therefore less substrate damage, than ones etched in a plasma without applied magnetic fields. Yeom and Kushner (1990) measured the fluorine radical and ion concentrations in both CHF_3 and CF_4 plasmas, showing that the concentration of these species increased with magnetic field strength. They also showed that the magnitude of the field accelerating the ions toward the

wafer surface decreased with magnetic field strength. The etching rate of Si and SiO_2 in these plasmas did not continue to increase with increasing magnetic field strength as would be expected from the increased fluorine and ion flux to the wafer surface. The etching rate initially increased with magnetic field strength, but reached a maximum and then decreased with continual increases in magnetic field strength. The increase in etching rate corresponded to the increase reactive species flux to the wafer surface. But as the magnetic field strength increased, the sheath voltage decreased, and therefore, the ion energy striking the surface decreased. With sufficient magnetic field strength, the ion bombardment energy fell below the threshold that was needed to initiate surface reactions, thus decreasing the etching rate. In the etching of polysilicon with 0.015 Torr Cl_2 plasma, a plateau in etching rate at 140 Gauss was also observed (Nguyen *et al.* 1990). The decrease in etching rate was not observed because 140 Gauss was the maximum achievable field strength in that reactor.

6.2 POWER DEPOSITION

Power deposition patterns in magnetically enhanced plasmas should be similar to power deposition in plasmas without magnetic fields. By decreasing the loss of electrons to the electrodes, the magnetic field increases bulk power deposition. Therefore, the transition from bulk power deposition dominated regime to sheath power deposition dominated regime should occur at a higher current.

Argon power deposition trends in magnetically enhanced plasmas are shown in Figures 6.2 through 6.7. All the data are measured in the 19 cm diameter chamber with the electrodes separated by 3 cm. Power is measured in a magnetically enhanced plasma in the same way it was measured in the data reported in Chapter 3. Figure 6.2 shows power deposition for argon plasmas at various pressures, without magnetic field. The trend shown here is the same as that shown in

Chapter 3 for bulk power deposition and is repeated here as a reference for the power data measured under the same conditions, except with a magnetic field. As demonstrated in Chapter 3, and seen in Figure 6.2, bulk power deposition without applied magnetic field can be expressed as:

$$P_b|_{B=0} = \kappa_1 I_o (pd)^{\frac{1}{2}} \quad . \tag{6.3}$$

The power values from 0.05 Torr to 2 Torr all follow the same line initially. As the I_o increases, the power values deviate from Equation 6.3 as power deposition changes from the bulk power deposition to sheath power deposition regime.

Figures 6.3 through 6.7 show the effect of magnetic field on power deposition in plasmas with pressures ranging from 0.05 Torr to 1 Torr. Two immediate observations are found by examining these graphs. The first is that the magnetic field has no effect on power deposition for 1 Torr plasmas (Figure 6.7), and for 0.5 Torr plasmas when the field strength is less than 200 Gauss (Figure 6.6). The second is that there is a transition from bulk to sheath deposition in the plasmas without applied magnetic field. In contrast, for plasmas with magnetic field (seen in the lines marked with Δ's, □'s, and O's), power is proportional to I_o, indicating that all the data is in the bulk power deposition regime (Figures 6.3 through 6.5).

The first observation, that the magnetic field has no effect on power deposition at high pressures, and therefore, the electron Larmor radius must be larger than the its mean free path. At 0.5 Torr, only at 200 Gauss is there a slight increase in power deposition. Looking at Figure 6.6 carefully, power at the lowest I_o values, just before the plasma extinguishes, does depend on the magnetic field strength.

The transition out of bulk power deposition regime is evident in power data taken without applied magnetic field from 0.05 Torr to 0.3 Torr, shown by the line marked with *'s in Figures 6.3 through 6.5. For a fixed current, I_o, through the plasma, the magnetic field increases power

consumption in the bulk at all these pressures. In all these cases, the electron Larmor radius is smaller than the mean free path, thus the magnetic field confines electrons to the bulk, increasing electron impact reactions. At the same time that the magnitude of bulk power deposition increases, the higher plasma density does not necessarily lead to higher sheath power deposition. Although the ion flux to the surface increases linearly with magnetic field strength, as will be shown in the next section, the energy of these ions decreases because the sheath voltage decreases. Figure 6.8 shows that the voltage amplitude, which is approximately proportional to the sheath voltage, at 0.05, 0.1, and 0.3 Torr as a function of the current amplitude. For a fixed current, the voltage decreases approximately linearly with magnetic field strength. Since sheath power deposition is proportional to the product of the ion flux times the sheath voltage, the sheath power changes little with the addition of the magnetic field. For the range of data shown, all the magnetically enhanced plasmas are in the bulk power deposition regime. The transition to sheath power deposition will probably occur, but at higher I_o values than in plasmas without magnetic field.

At low pressures, replotting the power deposition values of Figures 6.3 through 6.5 as a function of magnetic field strength for fixed currents, shows that bulk power deposition scales linearly with the magnetic field strength. Assuming bulk power deposition varies with pressure and current similarly between a plasma with and without magnetic fields, the following equation can be fit to the power data:

$$P_b - P_b|_{B=0} = \kappa_2 \frac{B I_o}{p^{1/2}} . \qquad (6.4)$$

κ_2, which is a function of the gas properties and the length of the plasma, equals 0.0114 when fit to the data at 0.05 Torr, 0.1 Torr, and 0.3 Torr. $P_b|_{B=0}$ is calculated from Equation 6.3. The units of power, magnetic field, current amplitude, and pressure, are expressed as Watts, Gauss,

Amperes, and Torrs, respectively. Figure 6.9 shows that the data follows Equation 6.4 well.

6.3 ION PROPERTIES

The ion properties measured in a magnetically enhanced plasma are the ion flux and ion energy. The ion angle is not measured because the magnetic field alters the ion trajectory in the ion analyzer, as reported in Appendix I. Since the average ion angle depends on the average number of collisions ions have in the sheath, the ion angle can be inferred from plasmas run without magnetic field but with the same ion bombardment energy and number of ion mean free paths in the sheath.

Ion fluxes are shown in Figures 6.10, 6.11, and 6.12 for plasma with pressures ranging from 0.05 Torr to 0.3 Torr and magnetic field strengths ranging from 0 to 200 Gauss. In Chapter 4, equation 4.6 shows that combining Lieberman's (1989) equation for ion flux, J_{ion}, with the assumption that the sheath behaves like a capacitor gives the following relationship:

$$J_{ion} = \kappa_3 J_o^{3/2}$$

$$\text{where} \quad \kappa_3 = 2.10 \left(\frac{2q}{m}\right)^{1/4} \frac{\lambda_m^{1/2}}{\ell_s \varepsilon_o^{1/4} \omega^{3/2}} \quad . \tag{6.5}$$

q and m are the charge and mass of the ion, λ_m is the ion mean free path, ℓ_s is the sheath width, ω is the angular frequency, and J_o is equal to I_o/A. As in plasmas without magnetic field, the ion flux behavior in a magnetically enhanced plasma is described by Equation 6.5: the ion flux is proportional to $I_o^{3/2}$. Since the magnetic field changes the sheath width, the proportionality constant, κ_3, changes. At 0.05 and 0.1 Torr, κ_3 increases with magnetic field strengths, while the opposite trend occurs at 0.3 Torr. At low pressures, the magnetic field decreases the sheath width, thus increasing κ_3. The decrease in ion flux as magnetic field increases for the 0.3 Torr data is not understood.

Ion energy distributions (IEDs) for 0.05 Torr and 0.1 Torr, with and without magnetic field, are shown in Figures 6.13 and 6.14. For comparing IEDs with and without magnetic field, the IEDs are chosen which have approximately the same sheath voltage. The sheath voltage is equal the maximum ion energy (shown in Figures 6.13 and 6.14) because ions that goes through no collisions while travelling through the sheath, therefore have the maximum ion energy, attain an energy equal to the sheath potential. The average number of mean free paths in the sheath, estimated using collision cross sections reported by Cramer 1959, for ions in a 0.05 Torr plasma without magnetic field is five (ℓ_s = 8 mm); while at 200 Gauss, the number of mean free paths in the sheath is approximately one ($\ell_s \approx$ 1.5 mm). It is clear, from the few ions at the low energy end of the 0.05 Torr and 200 Gauss IED, that the ions have very few or no collisions travelling through the sheath. Although measured at the same pressure, the IED for the 0.05 Torr plasma without the magnetic field shows that most of the ions go through collisions while travelling through the sheath. The average ion energy at 0 Gauss and 200 Gauss are 17.3 eV and 32.2 eV, respectively. Similarly, the 0.1 Torr IEDs in Figure 6.14 shows that the magnetic field reduces the number of ion collisions in the sheath. The average ion energy at 0 Gauss and 133 Gauss are 11.8 eV and 21.5 eV, respectively. The estimated number of collisions in the sheath for a 0.1 Torr plasma is eight without magnetic field (ℓ_s = 7 mm), and four with a 133 Gauss field ($\ell_s \approx$ 3.5 mm).

Since the number of ion collisions increases with magnetic field strength for a fixed pressure, and since the mean free path is approximately independent of the magnetic field, the sheath width must decrease with magnetic field. Figure 6.15 shows the effect of magnetic field on the sheath width which is determined using the peaks of the spatial plasma emission scan. At 0.05 Torr, the sheath width decreases with the magnetic field strength, while higher pressures require higher magnetic field strengths before changes in the sheath width are observed. The

sheath width is proportional to the oscillation amplitude of the electron in an rf cycle. Therefore, the decrease in sheath width is expected because the magnetic field lowers the electron mobility toward the electrode, decreasing the electron oscillation amplitude. Higher pressures require higher magnetic fields to change the sheath width because the electron mean free path must be greater than the electron Larmor radius for the magnetic field to affect the plasma.

The average ion energies in 0.05 Torr and 0.1 Torr plasmas, at several magnetic field strengths, are shown in Figures 6.16 (a) and (b). These graphs show that for a given pressure and magnetic field strength, the ion energy is proportional to the sheath voltage, and that as magnetic field increases at a fixed voltage, the average ion energy increases. The positive y-intercept on these graphs corresponds to some fraction of the floating potential, similar to the ion energy data shown in Chapter 4 for plasmas without magnet fields. As pressure decreases and the magnetic field strength increases, the number of collisions in the sheath approaches zero. At this point, the y-intercept of an average ion energy versus V_o graph should approach the plasma floating potential.

6.4 CONCLUSION

Applying magnetic fields, ranging from 0 to 200 Gauss, to 13.56 MHz argon plasmas will change several plasma properties. The magnetic field lowers electron mobility to the electrode surfaces, and thus increase plasma density. This effect is seen in the increased bulk power deposition with magnetic field strength. Bulk power deposition scales as:

$$P_b - P_b|_{B=0} = \kappa_2 \frac{B I_o}{p^{1/2}} .$$

In 0.05 Torr and 0.1 Torr plasmas, keeping I_o constant, the voltage across the sheaths decreases while the ion flux increases with magnetic field strength. The net result of using

magnetic fields in plasmas is a increase in ion flux, but with lower energy ions. The ion flux with and without magnetic field scales linearly with $I_o^{3/2}$.

With magnetic fields, the ion energy still scales with the number of ion mean free paths in the sheath, and the energy is proportional to the sheath voltage. For a fixed voltage and pressure, the ion energy increases with magnetic field strength because the magnetic field decreases the sheath width, thus decreasing the number of collisions in the sheath.

All the magnetic field effects on the plasma properties reported above occur only when the electron mean free path is greater than the Larmor radius. That is, for pressures above 0.5 Torr, magnetic field strength greater than 200 Gauss is needed to change the plasma.

6.5 NOMENCLATURE

A	electrode area
$\vec{B}$	magnetic field strength
d	electrode spacing
$\vec{\mathcal{E}}$	electric field
$\vec{F}$	force
I_o	current amplitude
J_{ion}	ion flux on the surface
J_o	I_o/A
m_j	mass of particle j
N	gas density
P_b	bulk power deposition
$P_b \mid_{\vec{B}=0}$	bulk power deposition without magnetic field
p	pressure
q	charge of the particle
r_L	Larmor radius
t	time
v_j	velocity of particle j
κ_i	i=1,2,3 constants
λ_m	mean free path
σ_j	collision cross section of particle j

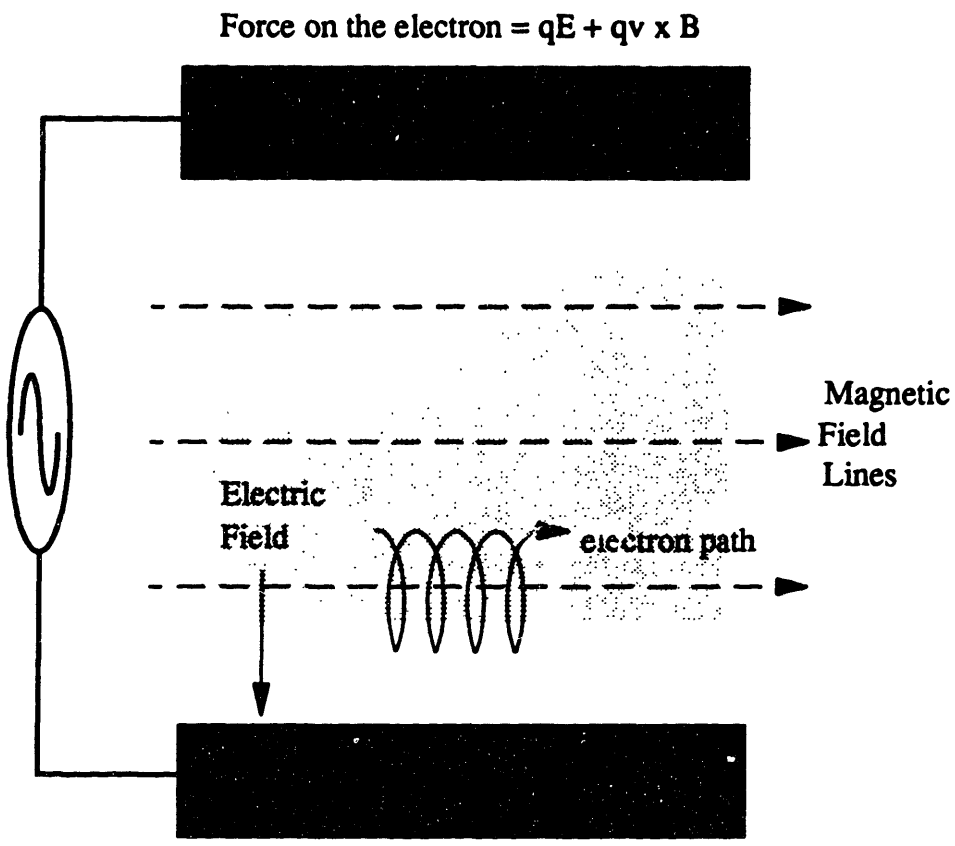


Figure 6.1: Magnetic field effect on electron path.

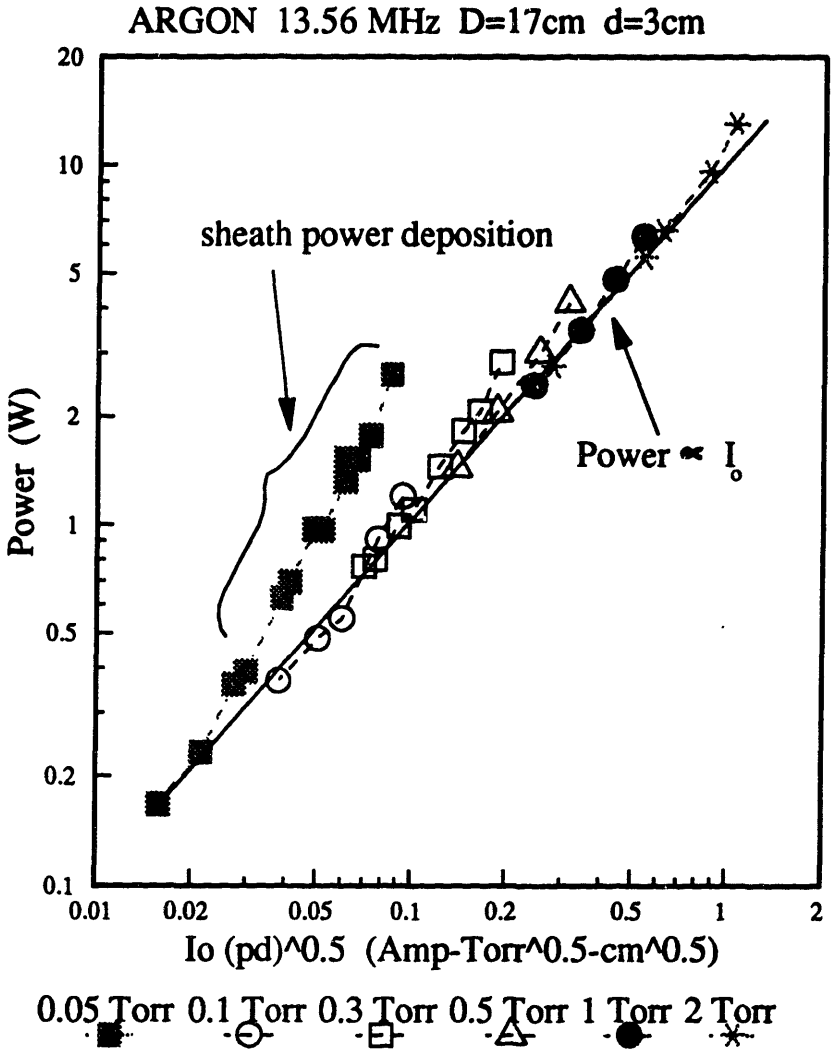


Figure 6.2: Power at various pressures and no magnetic field.

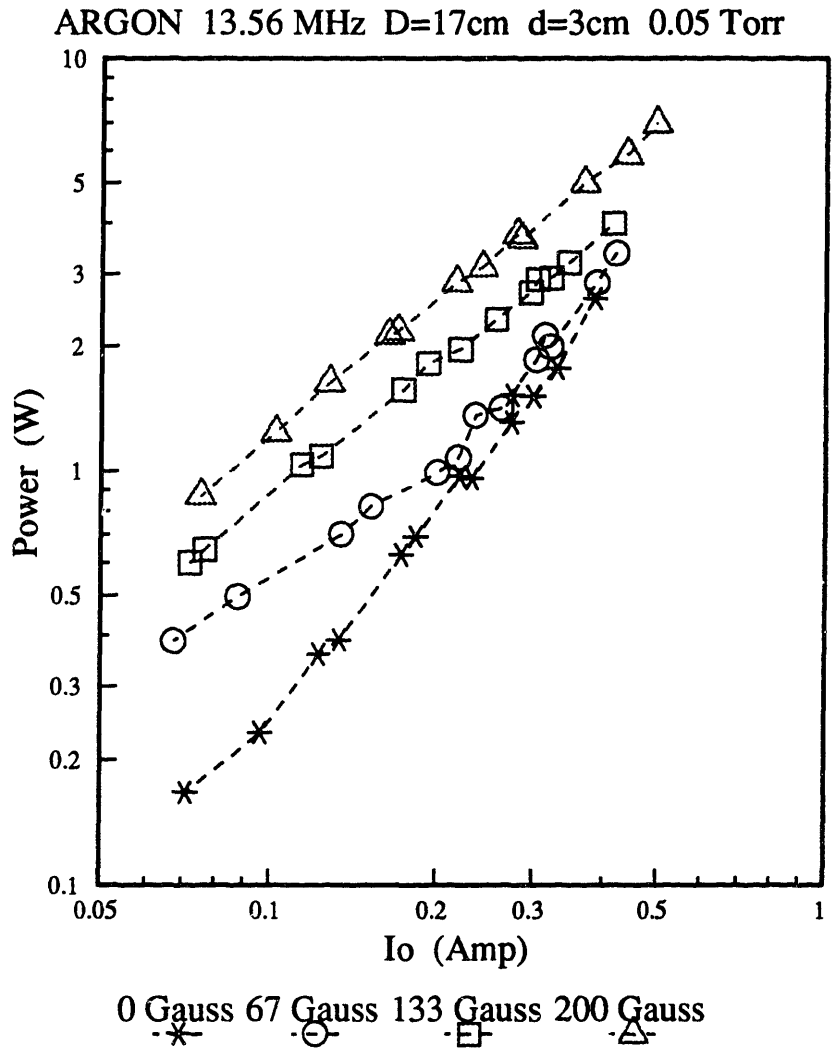


Figure 6.3: The effect of magnetic field on power deposition at 0.05 Torr.

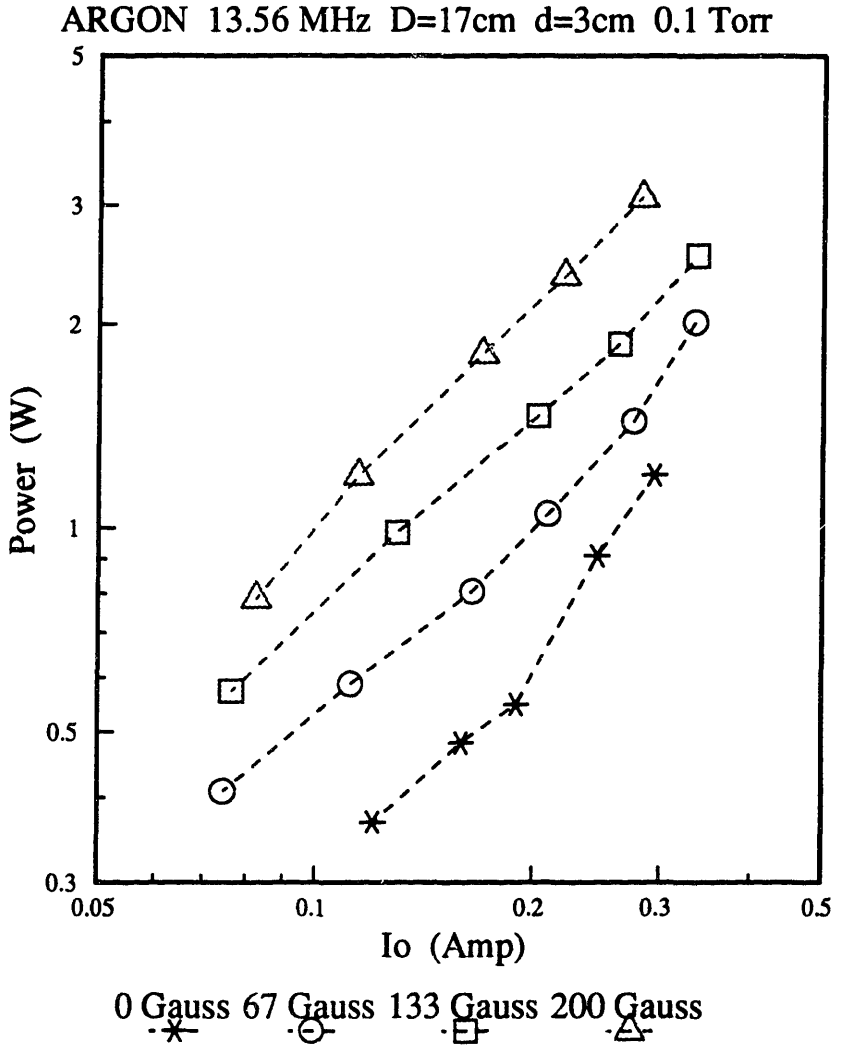


Figure 6.4: The effect of magnetic field on power deposition at 0.1 Torr.

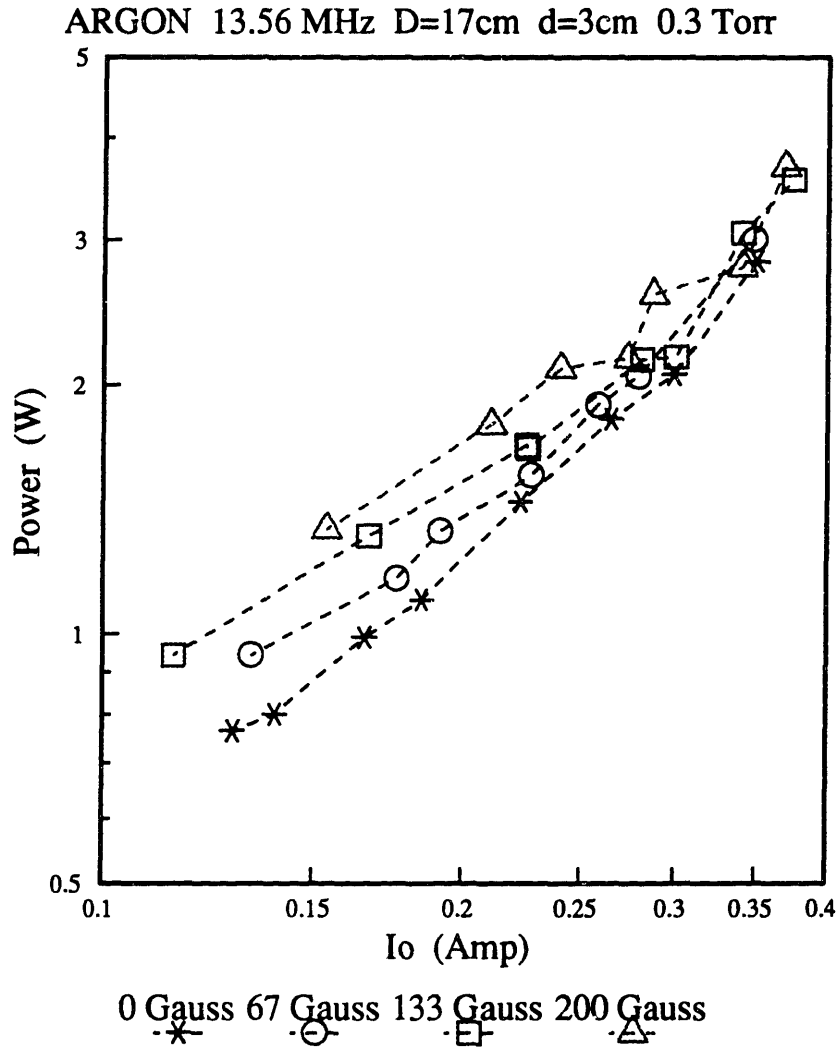


Figure 6.5: The effect of magnetic field on power deposition at 0.3 Torr.

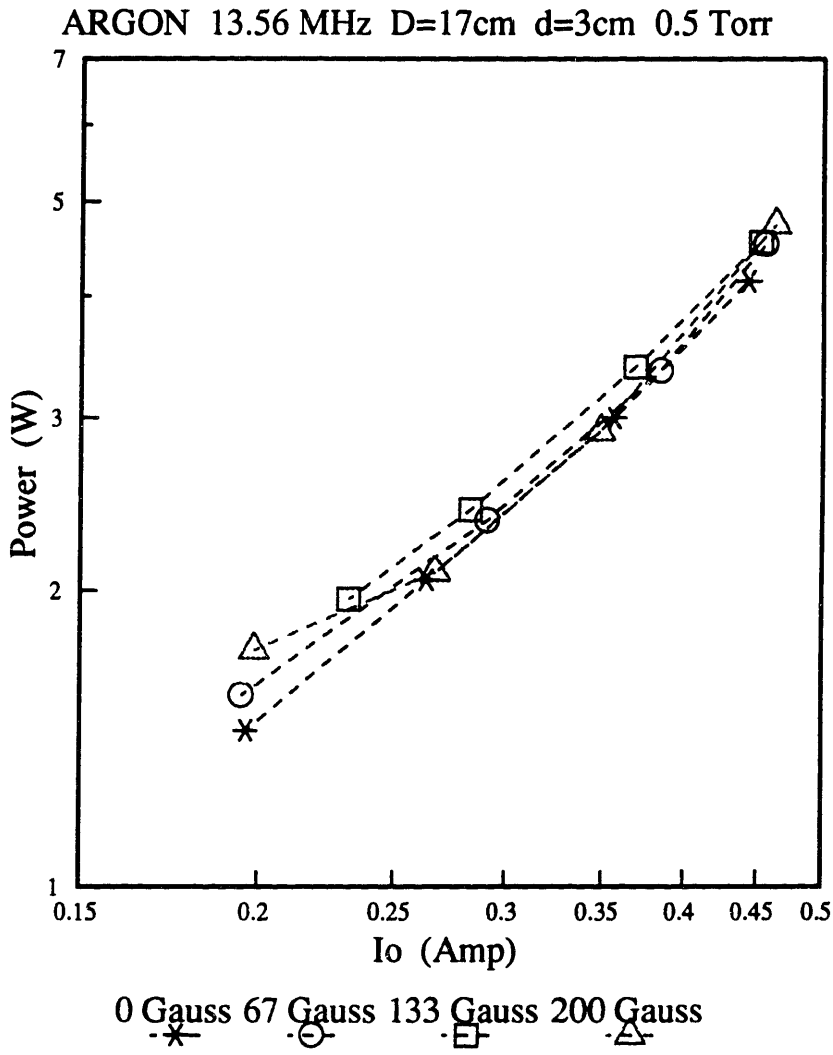


Figure 6.6: The effect of magnetic field on power deposition at 0.5 Torr.

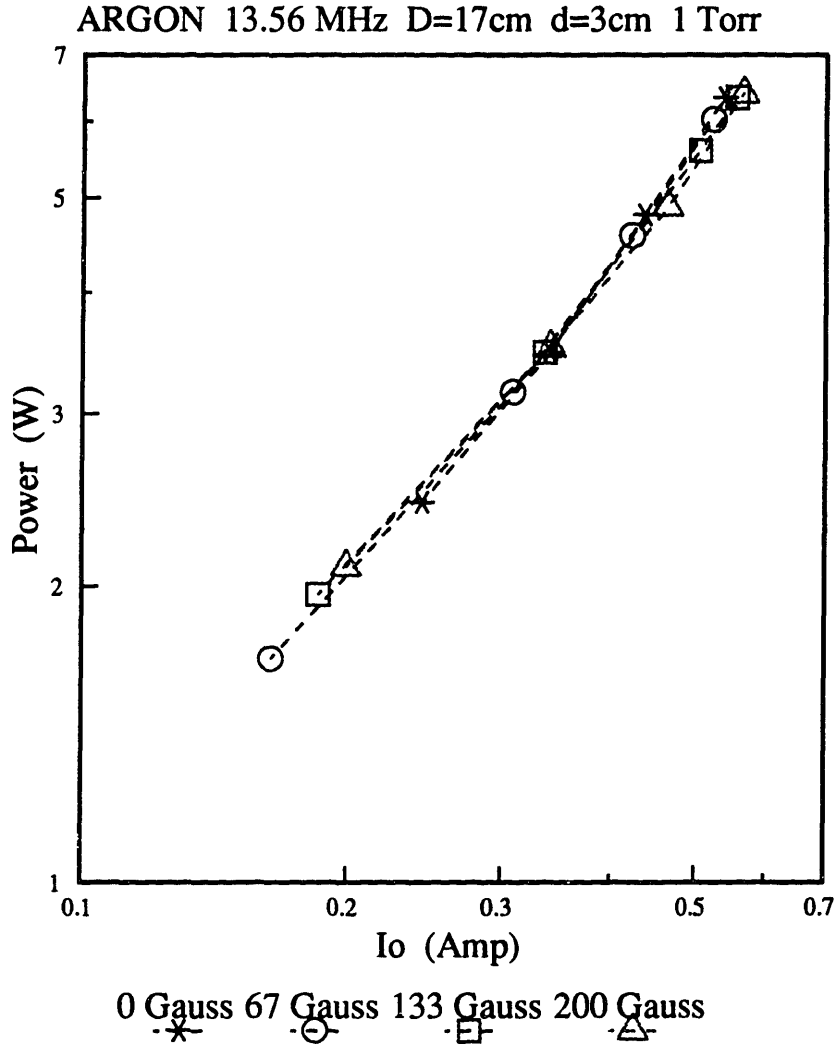


Figure 6.7: The effect of magnetic field on power deposition at 1 Torr.

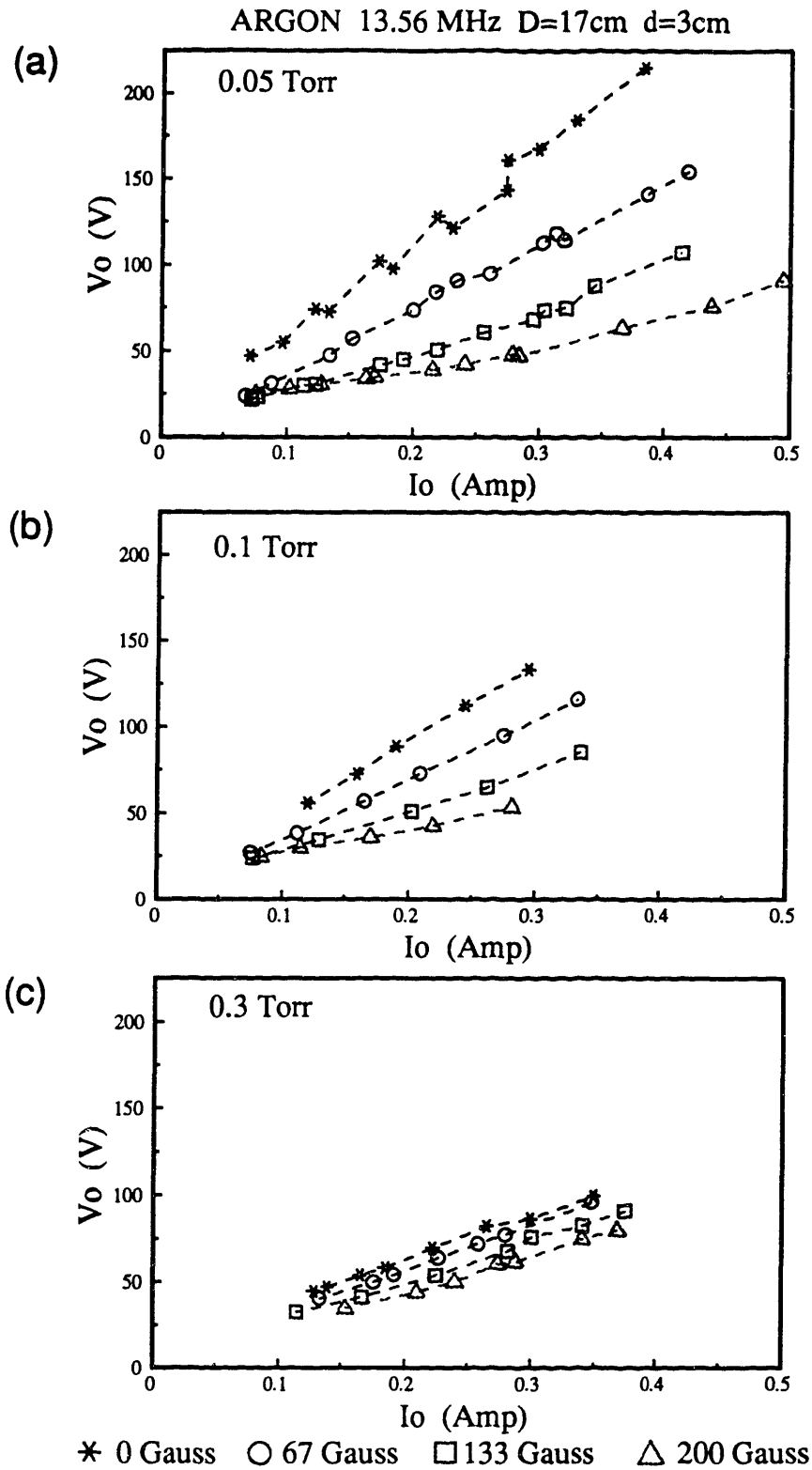


Figure 6.8: The effect of magnetic field on voltage at (a) 0.05 Torr, (b) 0.1 Torr, and (c) 0.3 Torr.

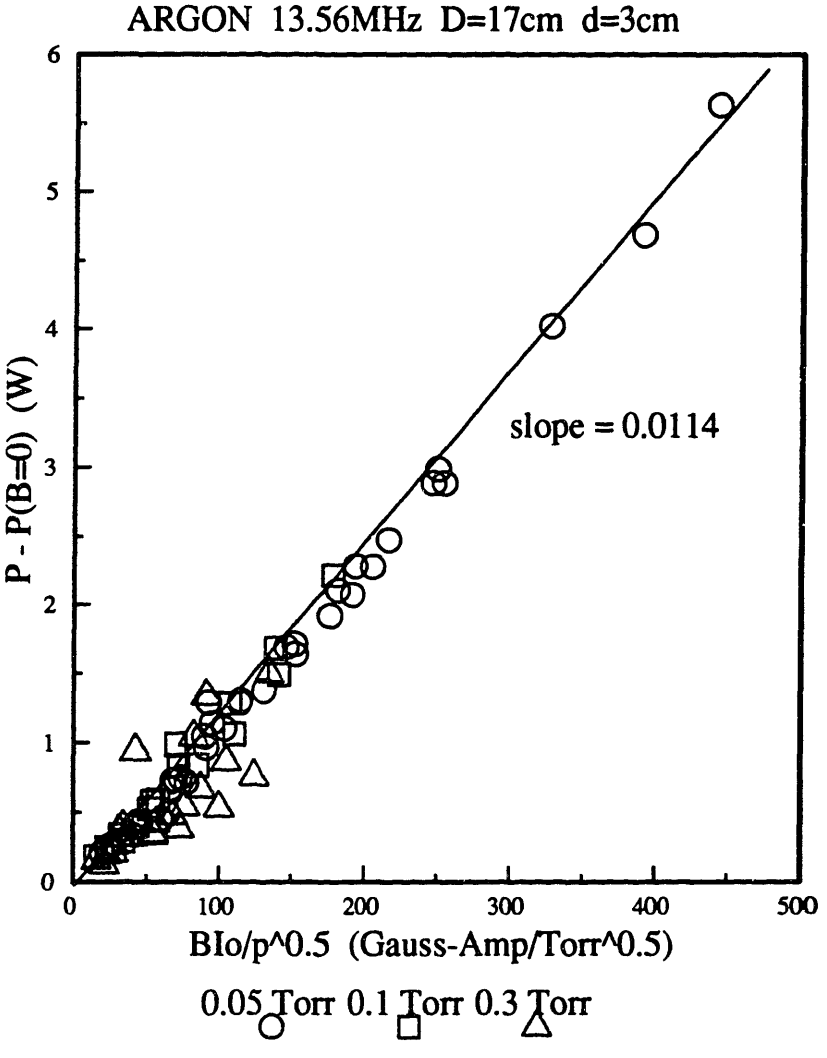


Figure 6.9: Bulk power deposition in plasmas with magnetic fields.

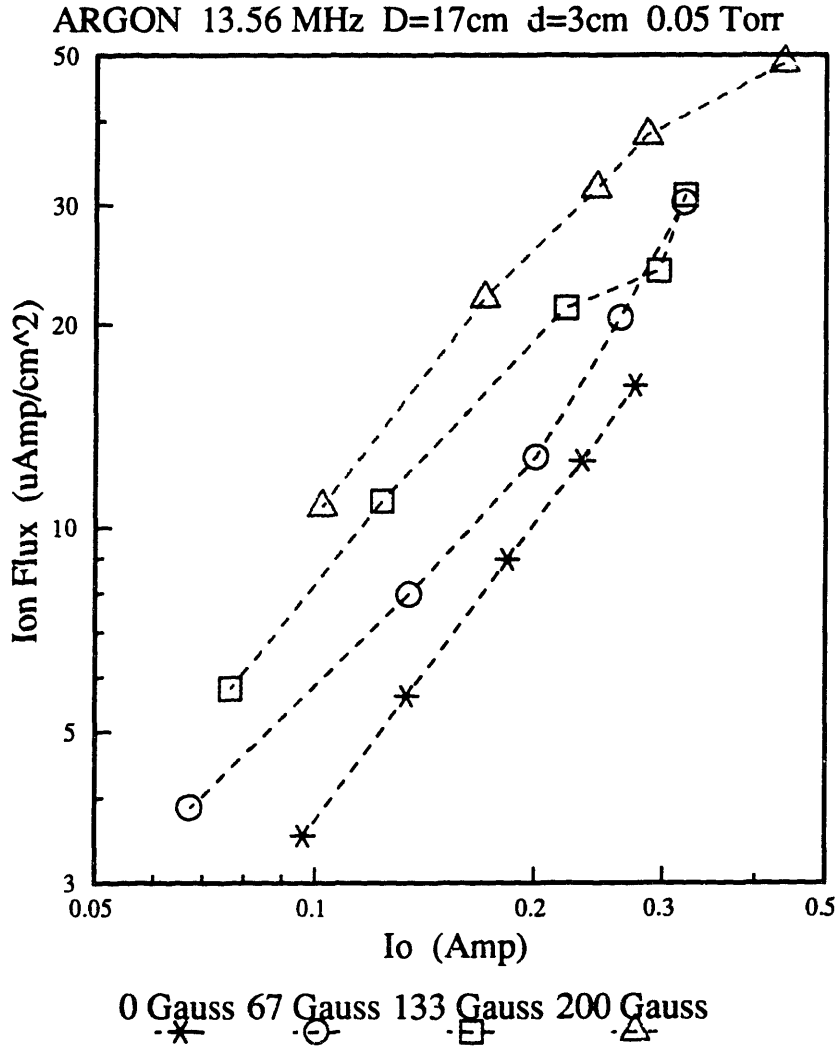


Figure 6.10: Ion flux at various magnetic field strengths and 0.05 Torr.

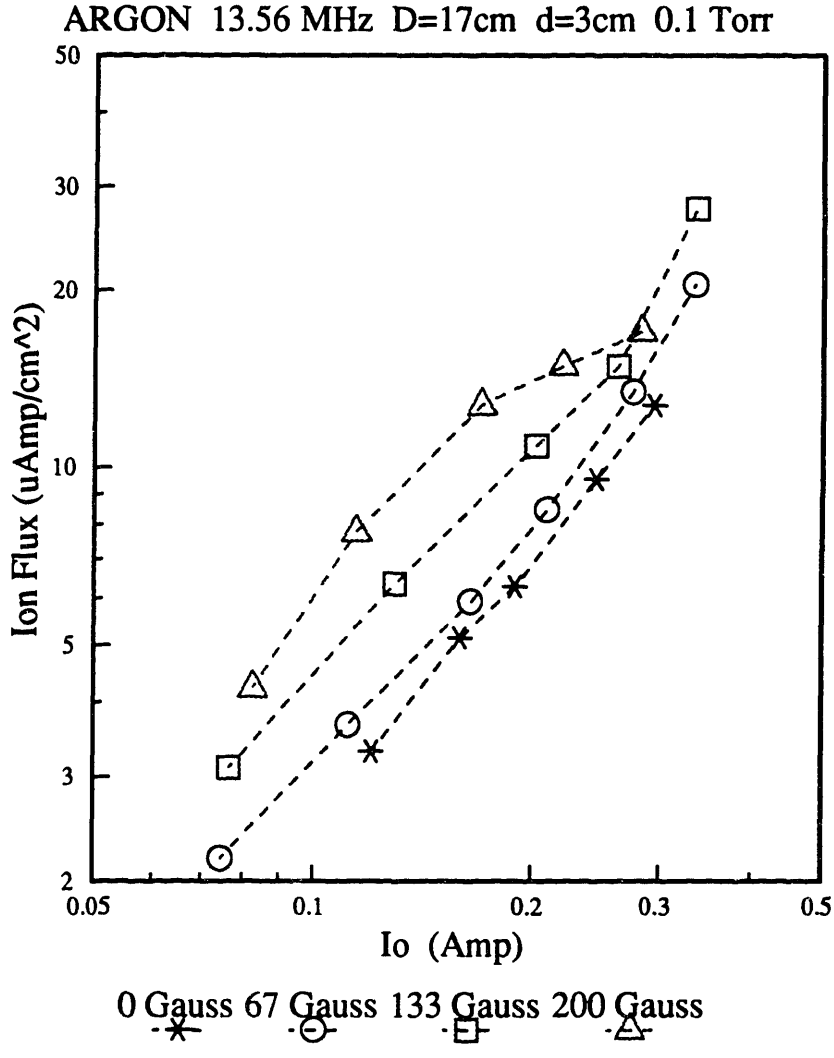


Figure 6.11: Ion flux at various magnetic field strengths and 0.1 Torr.

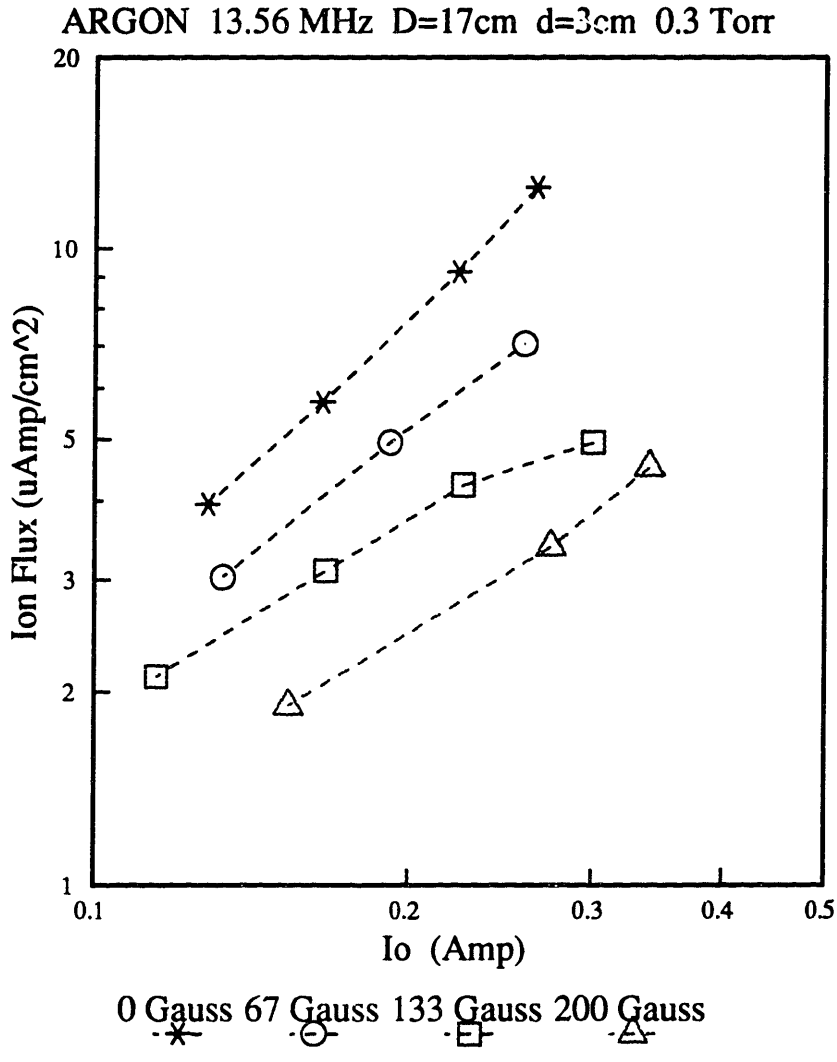


Figure 6.12: Ion flux at various magnetic field strengths and 0.3 Torr.

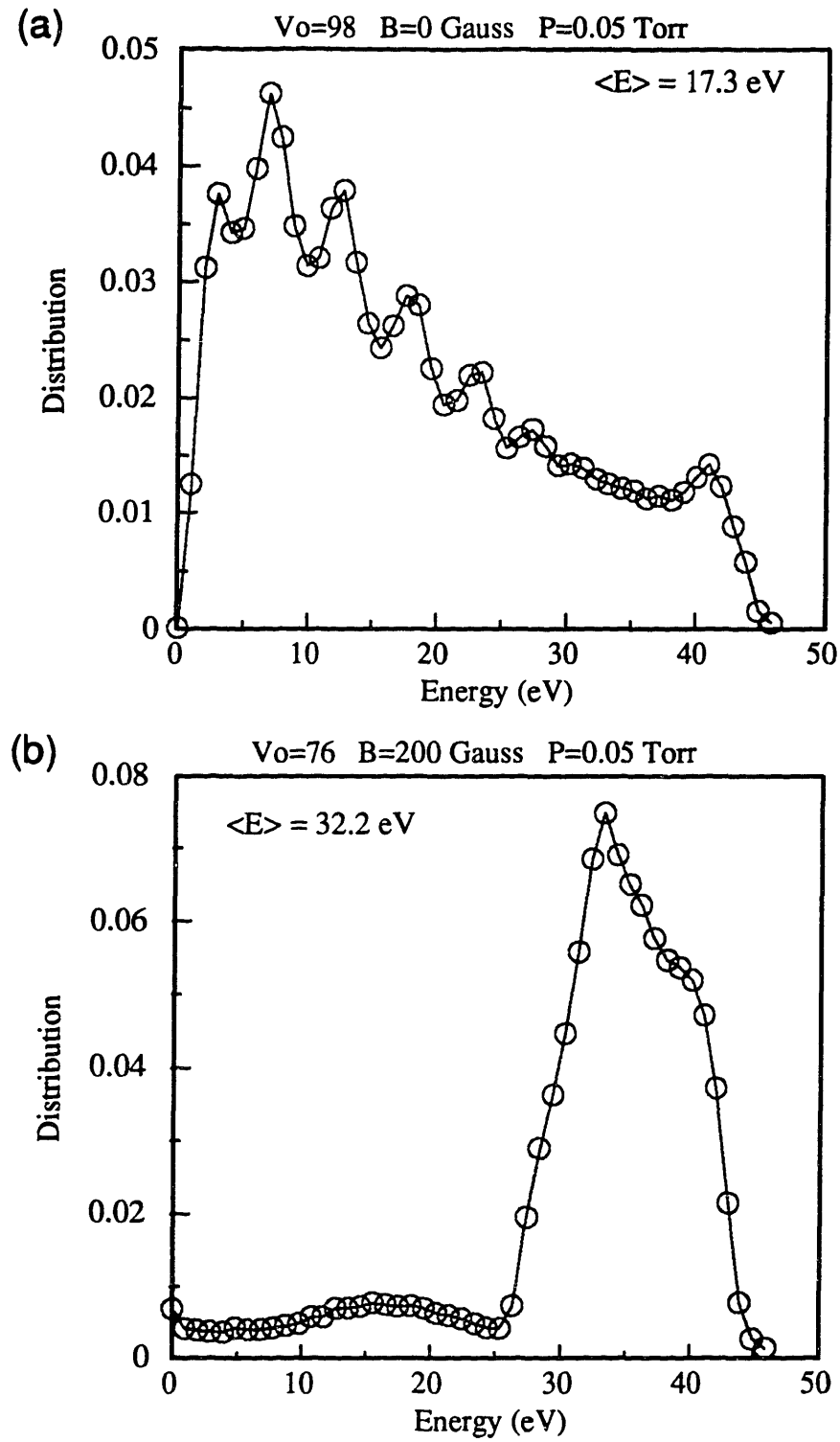


Figure 6.13: Argon ion energy distribution at 0.05 Torr with (a) no magnetic field and with a (b) 200 Gauss field.

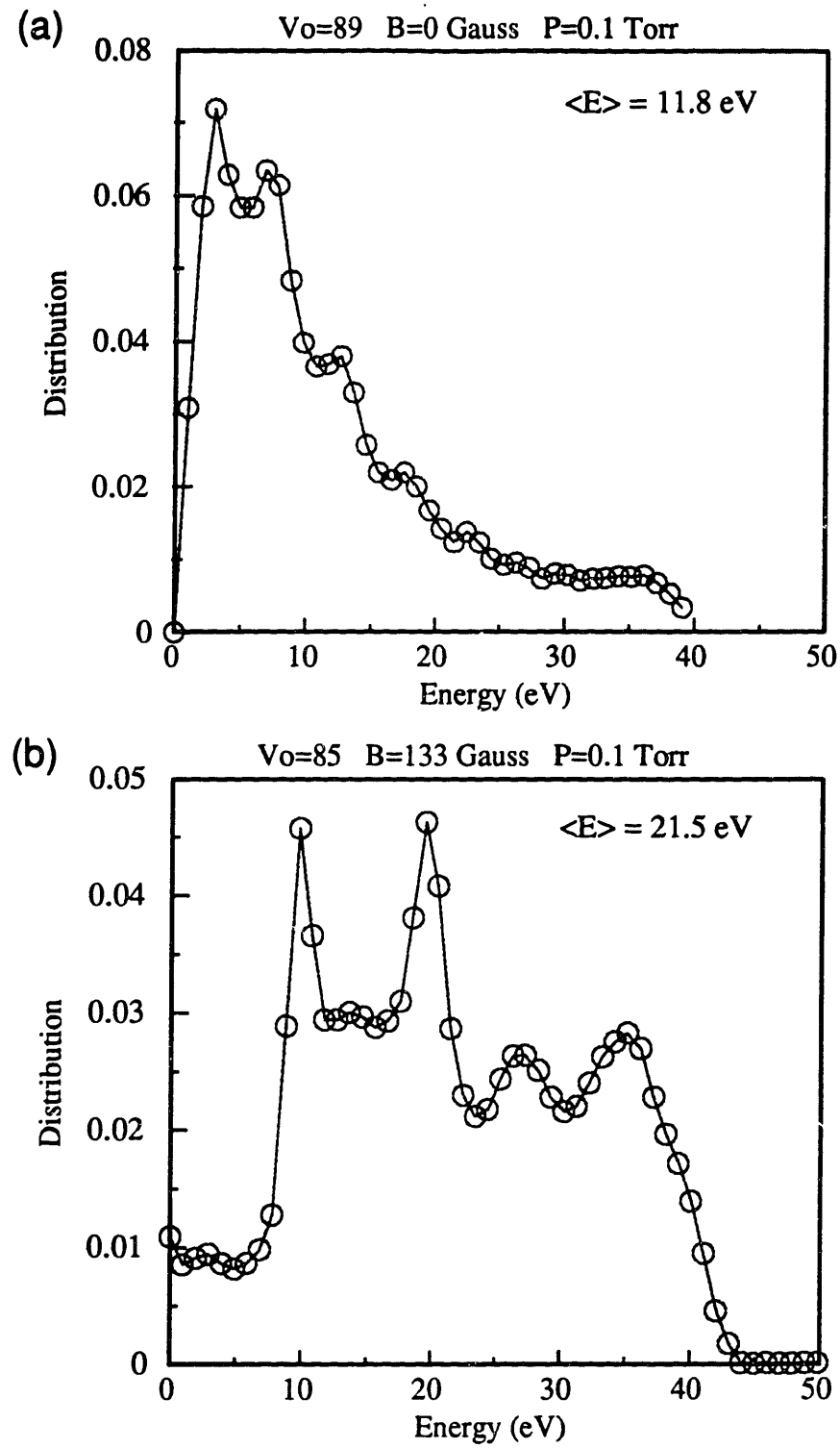


Figure 6.14: Argon ion energy distribution at 0.1 Torr with (a) no magnetic field and with a (b) 133 Gauss field.

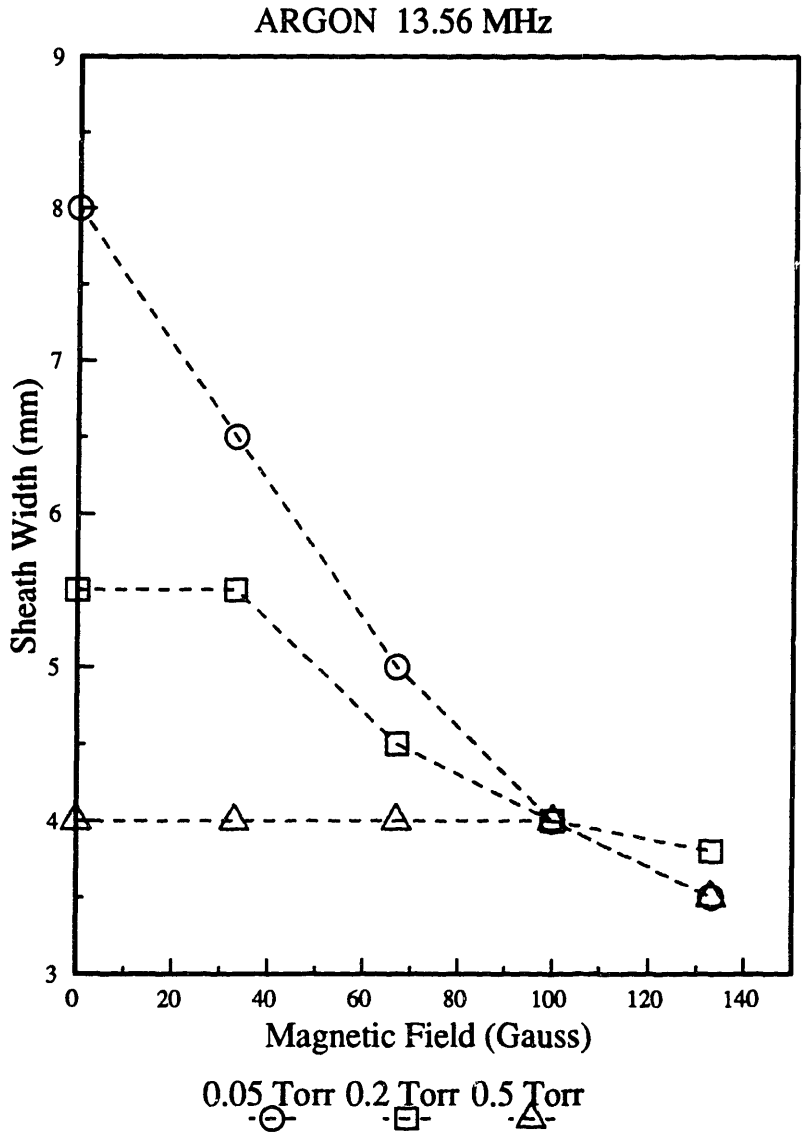


Figure 6.15: The effect of magnetic field on sheath width in argon plasmas.

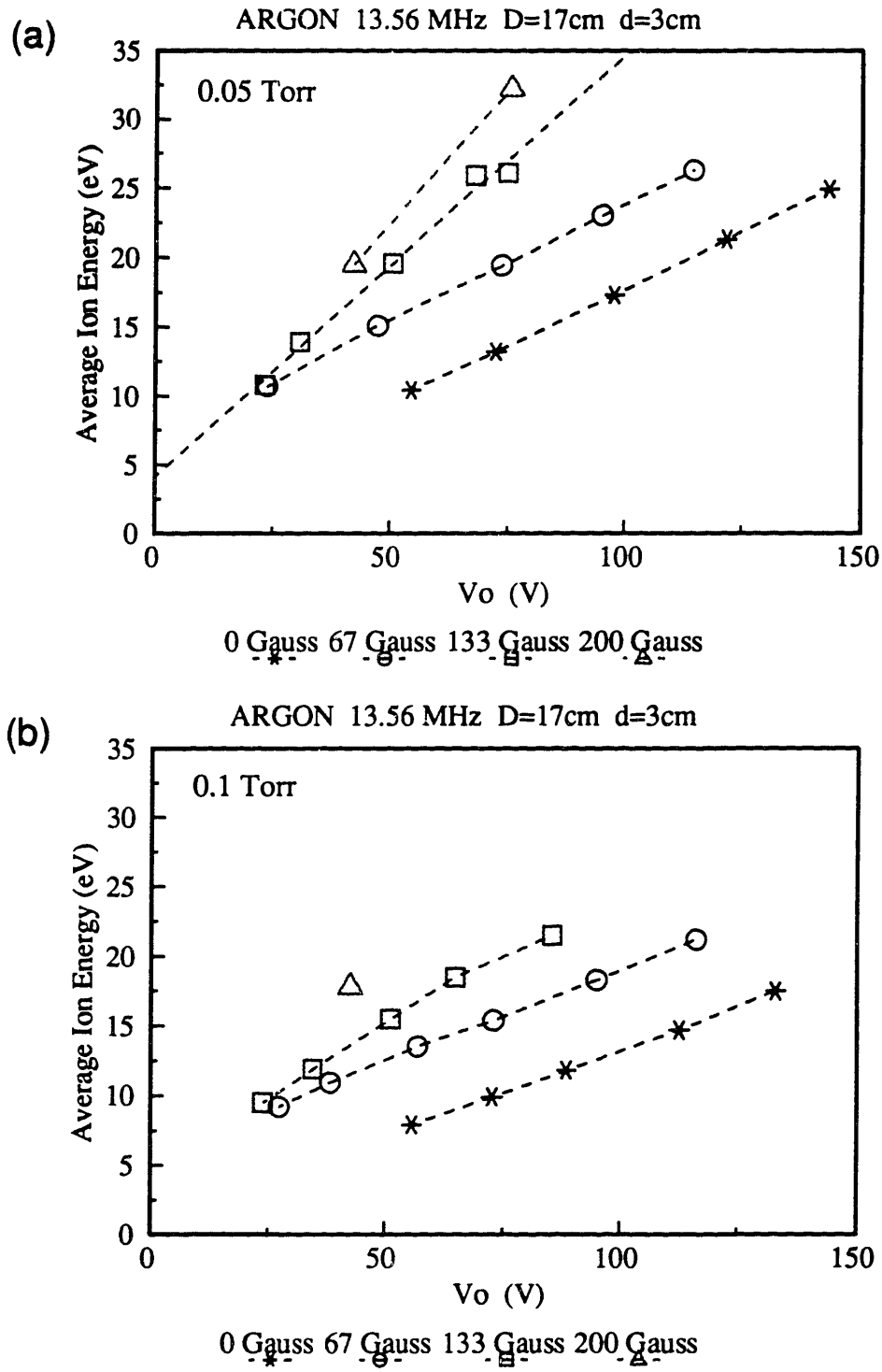


Figure 6.16: The effect of magnetic field on the average ion energy at (a) 0.05 Torr and (b) 0.1 Torr.

REFERENCES

K. D. Allen, H. H Sawin, and A Yokozeki, "The Plasma Etching of Polysilicon with CF_3Cl/Argon Discharges," *Journal of Electrochemical Society: Solid-State Science and Technology* **133**(11), 2331 (1986).

C. Beneking, "Power Dissipation in Capacitively Coupled rf Discharges," *Journal of Applied Physics* **68**(9), 4461 (1990).

A. Bensaoula, J. A. Strozier, A. Ignatiev, J. Yu, and J. C. Wolfe, "Ion Enhanced Reactive Etching of Tungsten Single Crystals and Films with XeF_2," *Journal of Vacuum Science and Technology* **A5**(4), 1921 (1987).

J. P. Boeuf, "Numerical Model of RF Glow Discharges," *Physical Review* **A36**(6), 2782 (1987).

K. P. Brand and H. Jungblut, "The Interaction Potentials of SF_6 Ions in SF_6 Parent Gas Determined from Mobility Data," *Journal of Chemical Physics* **78**(4), 1999 (1983).

I. Brodie and J. J. Muray, *The Physics of Microfabrication*, New York, Plenum Press, 137-143 (1982).

R. H. Bruce, "Ion Response to Plasma Excitation Frequency," *Journal of Applied Physics* **52**(12), 7064 (1981).

U. Buck, *Advances in Chemical Physics. Molecular Scattering: Physical and Chemical Applications*, **XXX** ed. K. P. Lawley, John Wiley and Sons, London (1975).

J. W. Butterbaugh, L. D. Baston, and H. H. Sawin, "Measurement and Analysis of Radio Frequency Glow Discharge Electrical Impedance and Network Power Loss," *Journal of Vacuum Science and Technology* **A8**(2), 916 (1990).

S. Chapman and T. G. Cowling, *The Mathematical Theory of Non-Uniform Gases*, 2nd ed., University Press, Cambridge, 223 (1952).

C. D. Child, "Discharge from Hot CaO," *Physical Review* **32**(5), 492 (1911).

J. D. Cobine, *Gaseous Conductors: Theory and Engineering Applications*, McGraw-Hill Book Co., Inc., New York, 129 (1941).

J. W. Coburn, "A System for Determining the Mass and Energy of Particles Incident on a Substrate in a Planar Diode Sputtering System," *Review of Scientific Instruments* **41**(8), 1219 (1970).

J. W. Coburn and E. Kay, "Pressure Considerations Associated with Ion Sampling from Glow Discharges," *Journal of Vacuum Science and Technology* **8**(6), 738 (1971).

J. W. Coburn and E. Kay, "Positive-Ion Bombardment of Substrates in rf Diode Glow Discharge

Sputtering," *Journal of Applied Physics* **43**(12), 4965 (1972).

W. H. Cramer, "Elastic and Inelastic Scattering of Low-Velocity Ions: Ne^+ in A, A^+ in Ne, and A^+ in A," *Journal of Chemical Physics* **30**(3), 641 (1959).

D. A. Dahl, EG&G Idaho Inc. Idaho National Engineering Laboratory, P. O. Box 1625, Idaho Falls, ID 83415 (1985) - PC Version.

T. J. Dalton, J. C. Arnold, H. H. Sawin, s. Swan, and D. Corliss, "Micro Trench Formation During Plasma Etching, " **700RNP**. *Electrochemical Society Spring Meeting*, St. Louis, Missouri (1992).

W. D. Davis and T.A. Vanderslice, "Ion Energies at the Cathode of a Glow Discharge," *Physical Review* **131**(1), 219 (1963).

V. M. Donnelly, D. L. Flamm, W. C. Dautremont-Smith, and D. J. Werder, "Anisotropic Etching of SiO_2 in Low-Frequency CF_4/O_2 and NF_3/Ar Plasmas," *Journal of Applied Physics* **55**(1), 242 (1984).

D. A. Doughty, E. A. Den Hartog, and J. E. Lawler, "Current Balance at the Surface of a Cold Cathode," *Physical Review Letters* **58**(25), 2668 (1987).

D. J. Economou and R. C. Alkire, " Effect of Potential Field on Ion deflection and Shape Evolution of Trenched during Plasma-Assisted Etching," *Journal of Electrochemical Society* **135**(4), 941 (1988).

A. von Engel, *Electric Plasmas: Their Nature and Uses*, 135-6 (1983).

R. T. Farouki, S. Hamaguchi, and M. Dalvie, "Monte Carlo Simulations of Space-Charge-Limited Ion Transport Through Collisional Plasma Sheaths," *Physical Review* **A44**(4), 2664 (1991).

U. Gerlach-Meyer, J. W. Coburn, and E. Kay, "Ion-Enhanced Gas-Surface Chemistry: the Influence of the Mass of the Incident Ion," *Surface Science* **103**, 177 (1981).

V. A. Godyak, R. B. Piejak, and B. M. Alexandrovich, "Ion Flux and Ion Power Losses at the Electrode Sheaths in a Symmetrical rf Discharge," *Journal of Applied Physics* **69**(6), 3455 (1991a).

V. A. Godyak, R. B. Piejak, and B. M. Alexandrovich, "Electrical Characteristics of Parallel Plate rf Discharges," *IEEE Transactions on Plasma Science* **19**(4), 660 (1991b).

E. Gogolides, *Self-Consistent Continuum Modeling of Radio-Frequency Glow Discharges*, Ph.D. Thesis, MIT Department of Chemical Engineering (1990).

D. B. Graves and K. F. Jensen, "A Continuum Model of DC and RF Discharges," *IEEE Transactions on Plasma Science* **PS14**(2), 78 (1986).

D. C. Gray, *Beam Simulation Studies of Plasma-Surface Interactions in Fluorocarbon Etching of*

Si and SiO_2, Ph.D. Thesis, MIT Department of Chemical Engineering (1992).

W. M. Greene, D. W. Hess, W. G. Oldham, "Ion-Bombardment-Enhanced Plasma Etching of Tungsten with NF_3/O_2," *Journal of Vacuum Science and Technology* **B6**(5), 1570 (1988a).

W. M. Greene, M. A. Hartney. W. G. Oldham, and D. W. Hess, "Ion Transit Through Capacitively Coupled Ar Sheaths: Ion Current and Energy Distribution," *Journal of Applied Physics* **63**(5), 1367 (1988b).

D. Halliday and R. Resnick, *Fundamentals of Physics*, 2nd ed., John Wiley & Sons, New York (1981).

J. M. E. Harper, J. J. Cuomo, P. A. Leary, G. M. Summa, H. R. Kaufman, and F .J. Bresnock, "Low Energy Ion Beam Etching," *Journal of Electrochemical Society: Solid-State Science and Technology* **128**(5), 1077 (1981).

J. B. Hasted, "Ion-Atom Charge Transfer Collisions at Low Energies", *Advances in Atomic and Molecular Physics* **15**, 205 (1979).

F. Heinrich and P. Hoffmann, "Laser-Induced Fluorescence and Emission Spectroscopic Study of Magnetic Field Effects in a Low-Pressure Etch Plasma," *Journal of Applied Physics* **71**(4), 1683 (1992).

D. A. Huchital and J. D. Rigden, "Resolution and Sensitivity of the Spherical-Grid Retarding Potential Analyzer," *Journal of Applied Physics* **43**(5), 2291 (1972).

G. L. Huppert, unpublished data (1992).

S. G. Ingram and N. St. J. Braithwaite, "Ion and Electron Energy Analysis at a Surface in an RF Discharge," *Journal of Physics D: Applied Physics* **21**(10), 1496 (1988).

S. C. Jackson and T. J. Dalton, "A Profile Evolution Model with Redeposition," *Proceedings SPIE - The International Society for Optical Engineering: Dry Processing for Submicrometer Lithography* **1185**, 225 (1989).

Wilhelm Jost, ed., "General Introduction to Kinetics: Gas Reactions," *Physical Chemistry, an Advanced Treatise*, **VIB**, Academic Press, New York, 566 (1975).

C. W. Jurgensen, "Sheath Collision Processes Controlling the Energy and Directionality of Surface Bombardment in O_2 Reactive Ion Etching," *Journal of Applied Physics* **64**(2), 590 (1988).

C. W. Jurgensen and E. S. G. Shaqfeh, "Nonlocal Transport Models of the Self-Consistent Potential Distribution in a Plasma Sheath with Charge Transfer Collisions," *Journal of Applied Physics* **64**(11), 6200 (1988).

K. Kohler, D. E. Horne, and J. W. Coburn, "Frequency Dependence of Ion Bombardment of Grounded Surfaces in rf Argon Glow Discharges in a Planar System," *Journal of Applied Physics*

58(9), 3350 (1985).

M. J. Kushner, "Distribution of Ion Energies Incident on Electrodes in Capacitively Coupled rf Discharges," *Journal of Applied Physics* **58**(11), 4024 (1985).

A. D. Kuypers and H. J. Hopman, "Ion Energy Measurement at the Powered Electrode in an rf Discharge," *Journal of Applied Physics* **63**(6), 1894 (1988).

M. A. Lieberman, "Spherical Shell Model of an Asymmetric rf Discharge," *Journal of Applied Physics* **65**(11), 4186 (1989).

J. Liu, G. L. Huppert, and H. H. Sawin, "Ion Bombardment in rf Plasmas," *Journal of Applied Physics* **68**(8), 3916 (1990).

A. Manenschijn and W. J. Goedheer, "Angular Ion and Neutral Energy Distribution in a Collisional rf Sheath," *Journal of Applied Physics* **69**(5), 2923 (1991).

P. W. May, D. Field, and D. F. Klemperer, "Modeling Radio-Frequency Discharges: Effects of Collisions Upon Ion and Neutral Particle Energy Distributions," *Journal of Applied Physics* **71**(8), 3721 (1992).

S. C. McNevin, N. A. Ciampa, and J. Miner, "SiO_2 / Si Etching with CHF_3 in a High-Field Magnetron," *Journal of Vacuum Science and Technology* **A10**(4), 1227 (1992).

A. Metze, D. W. Ernie, and H. J. Oskam, "Application of the Physics of Plasma Sheaths to the Modelling of rf Plasma Reactors," *Journal of Applied Physics* **60**(9), 3081 (1986).

A. Metze, D. W. Ernie, and H. J. Oskam, "The Energy Distribution of Ions Bombarding Electrode Surfaces in rf Plasma Reactors," *Journal of Applied Physics* **65**(3), 993 (1989).

C. J. Mogab and H. J. Levinstein, "Anisotropic Plasma Etching of Polysilicon," *Journal of Vacuum Science and Technology* **17**(3), 721 (1980).

S. V. Nguyen, G. Chrisman, D. Dobuzinsky, and D. Harmon, "Magnetically Enhanced Reactive Ion Etching of Poly Gate Electrodes Smaller Than 0.5 μm," *Solid State Technology* **33**(10), 73 (1990).

W. L. O'Brien, T. N. Rhodin, and L. C. Rathbun, "Chemically Enhanced Ion Etching on Refractory Metal Silicides," *Journal of Vacuum Science and Technology* **A6**(3), 1384 (1988).

Y. Okamoto and H. Tamagawa, "Energy Dispersion of Positive Ions Effused from an RF Plasma," *Journal of Physical Society of Japan.* **29**(1), 187 (1970).

W. G. Oldham, A. R. Neureuther, C. Sung, J. L. Reynolds, and S. N. Nandgaonkar, "A General Simulator for VLSI Lithography and Etching Processes: Part II - Application to Deposition and Etching," *IEEE Transactions on Electron Devices* **ED-27**(8), 1455 (1980).

D. J. Oostra, A. Haring, and A. E. de Vries, "Sputtering of SiO_2 in a XeF_2 and in a Cl_2 Atmosphere," *Journal of Vacuum Science and Technology* **B4**(6), 1278 (1986).

A. P. Paranjpe, "Studies of Gas Discharges for Dry Etching: Modeling and Diagnostics," Ph.D. Thesis, Stanford University (1989).

R. D. Pillsbury, Jr. , "SOLDESIGN: magnetic field calculation program," ver. 2.3, The Plasma Fusion Center, Massachusetts Institute of Technology, Cambridge, MA 02139. (1989).

H. H. Sawin and L.R. Reif, *Plasma Processing in Integrated Circuit Fabrication*, M.I.T. 10.616J/6.776J Course Notes, Massachusetts Institute of Technology, Cambridge, MA (1987).

E. S. G. Shaqfeh and C. W. Jurgensen, " Simulation of Reactive Ion Etching Pattern Transfer," *Journal of Applied Physics* **66**(10), 4664 (1989).

Ian W. M. Smith, "Molecular Collision Dynamics," *Kinetics and Dynamics of Elementary Gas Reactions*, Butterworths & Co., Ltd., London, 73 (1980).

R. Smith, G. Carter, and M. J. Nobes, "The Theory of Surface Erosion by Ion Bombardment," *Proceedings of the Royal Society, London* **A407**, 405 (1986).

T. J. Sommerer and M. J. Kushner, "Translationally Hot Neutrals in Etching Discharges," *Journal of Applied Physics* **70**(3), 1240 (1991).

H. P. Strunk, H. Cerva, and E. G. Mohr, "Damage to the Silicon Lattice by Reactive Ion Etching," *Journal of Electrochemical Society: Solid-State Science and Technology* **135**(11), 2876 (1988).

M. Surendra and D. B. Graves, "Modeling and Simulation of RF Glow Discharges," *XX International Conference on Phemomena in Ionized Gases*, Barga, Italy, (July 1991a).

M. Surendra and D. B. Graves, "Particle Simulations of Radio-frequency Glow Discharges," *IEEE Transactions on Plasma Science* **19**(2), 144 (1991b).

B. E. Thompson, K. D. Allen, A. D. Richards, and H. H. Sawin, "Ion Bombardment Energy Distributions in Radio-Frequency Glow-Discharge Systems," *Journal of Applied Physics* **59**(6), 1890 (1986).

B. E. Thompson, H. H. Sawin, and D. A. Fisher, "Monte Carlo Simulation of Ion Transport Through rf Glow-Discharge Sheaths," *Journal of Applied Physics* **63**(7), 2241 (1988).

R. T. C. Tsui, "Calculation of Ion Bombarding Energy and Its Distribution in rf Sputtering," *Physical Review* **168**(1), 107 (1968).

Y. Y. Tu, T. J. Chuang, and H. F. Winters, "Chemical Sputtering of Fluorinated Silicon," *Physical Review B* **23**(2), 823 (1981).

J. I. Ulacia F. and J. P. McVittie, "A Two-Dimensional Computer Simulation for Dry Etching

Using Monte Carlo Techniques," *Journal of Applied Physics* **65**(4), 1484 (1989).

N. C. Us, R. W. Sadowski, and J. W. Coburn, "Quartz Crystal Microbalance Simulation of the Directionality of Si Etching in CF_4 Glow Discharges," *Plasma Chemistry and Plasma Processing* **6**(1), 1 (1986).

M. L. Vestal, C. R. Blakley, and J. H. Futrell, "Crossed-Beam Measurements of Differential Cross Sections for Elastic Scattering and Charge Exchange in Low-Energy Ar^+-Ar Collisions," *Physical Review* **A17**(4), 1337 (1978).

W. L. Wiese and G. A. Martin, ed., *Wavelengths and Transition Probabilities for Atoms and Atomic Ions, Part II Transition Probabilities*, NIST, Gaithersburg, MD, 365 (1980).

Ch. Wild and P. Koidl, "Structured Ion Energy Distribution in Radio Frequency Glow-Discharge Systems," *Applied Physics Letters* **54**(6), 505 (1989).

C. Wild and P. Koidl, "Ion and Electron Dynamics in the Sheath of Radio-Frequency Glow Discharges," *Journal of Applied Physics* **69**(5), 2909 (1991).

H. F. Winters, "Etch Products from the Reaction on Cl_2 with Al(100) and Cu(100) and XeF_2 with W(111) and Nb," *Journal of Vacuum Science and Technology* **B3**(1), 9 (1985).

H. F. Winters, "Phenomena Produced by Ion Bombardment in Plasma-Assisted Etching Environments," *Journal of Vacuum Science and Technology* **A6**(3), 1997 (1988a).

H. F. Winters, "Ion-Induced Etching of SiO_2: The Influence of Mixing and Lattice Damage," *Journal of Applied Physics* **64**(5), 2805 (1988b).

S. Wolf and R. N. Tauber, *Silicon Processing for the VLSI ERA: Vol. 1 - Process Technology*, p. 199, Lattice Press, Sunset Beach, CA (1986).

S. Yamamoto, T. Kure, M. Ohgo, T. Matsuzama, S. Tachi, and H. Sunami, "A Two-Dimensional Etching Profile Simulator: ESPRIT," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **CAD-6**(3), 417 (1987).

G. Y. Yeom, J. A. Thornton, and M. J. Kushner, "Cylindrical Magnetron Discharges. II. The Formation of DC Bias in RF-Driven Discharge Sources," *Journal of Applied Physics* **65**(10), 3825 (1989).

G. Y. Yeom and M. J. Kushner, "Si/SiO_2 Etch Properties Using CF_4 and CHF_3 in Radio Frequency Cylindrical Magnetron Discharges," *Applied Physics Letter* **56**(9), 857 (1990).

T. Yoshizawa, Y. Sakai, H. Tagashirea, and S. Sakamoto, "Boltzmann Equation Analysis of the Electron Swarm Development in SF_6," *Journal of Physics D: Applied Physics* **12**(11), 1839 (1979).

C. B. Zarowin, "Plasma Etch Anisotropy - Theory and Some Verifying Experiments Relating Ion

Transport, Ion Energy, and Etch Profiles," *Journal of Electrochemical Society* **130**(5), 1144 (1983).

E. Zawaideh, F. Najmabadi, R. w. Conn, "Generalized Fluid Equations for Parallel Transport in Collisional to Weakly Collisional Plasmas," *Physical Fluids* **29**(2), 463 (1986).

APPENDIX A

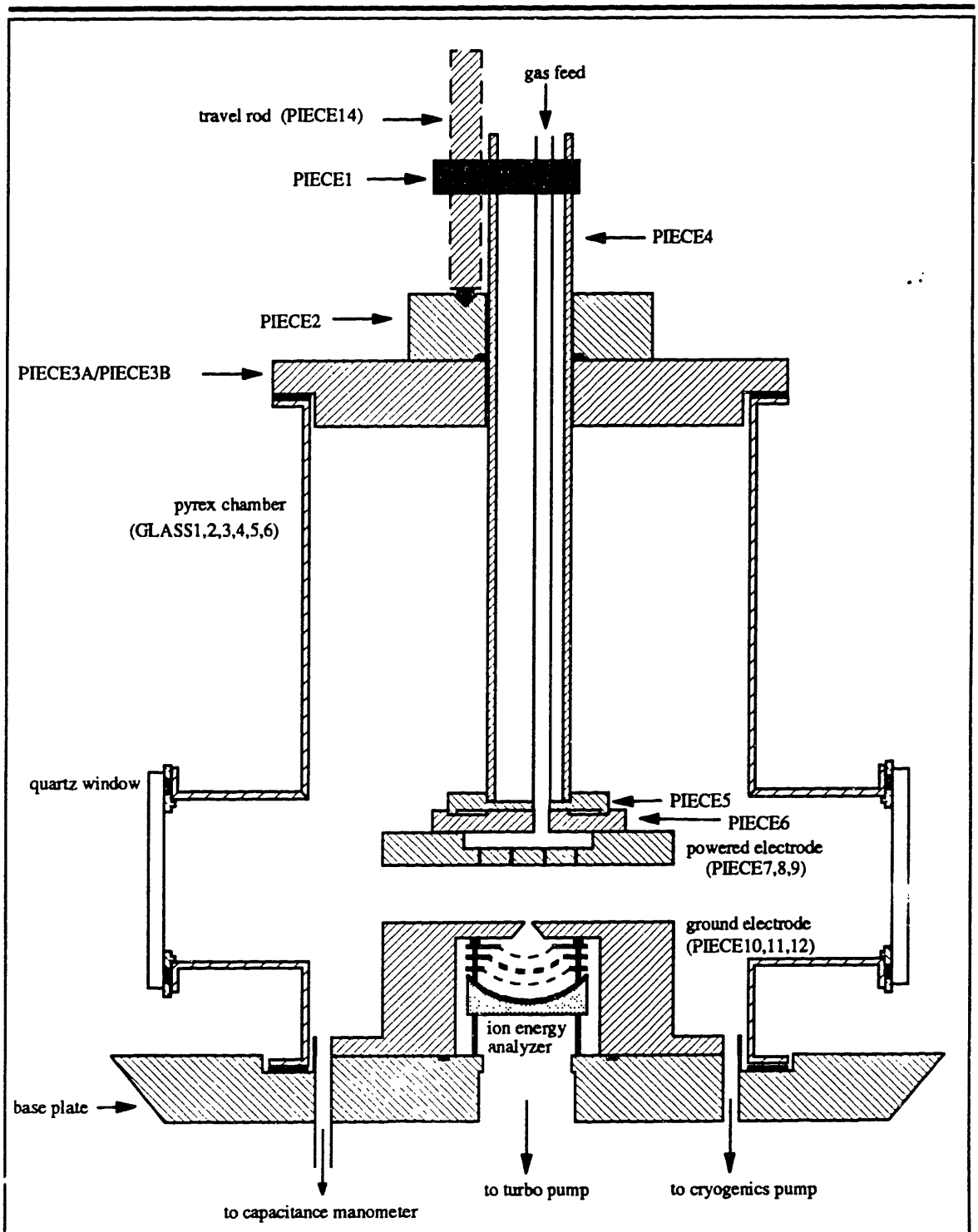
REACTOR DESIGN DRAWINGS

This appendix contains the machine drawings and fabrication specifications for construction of the pyrex reactor.

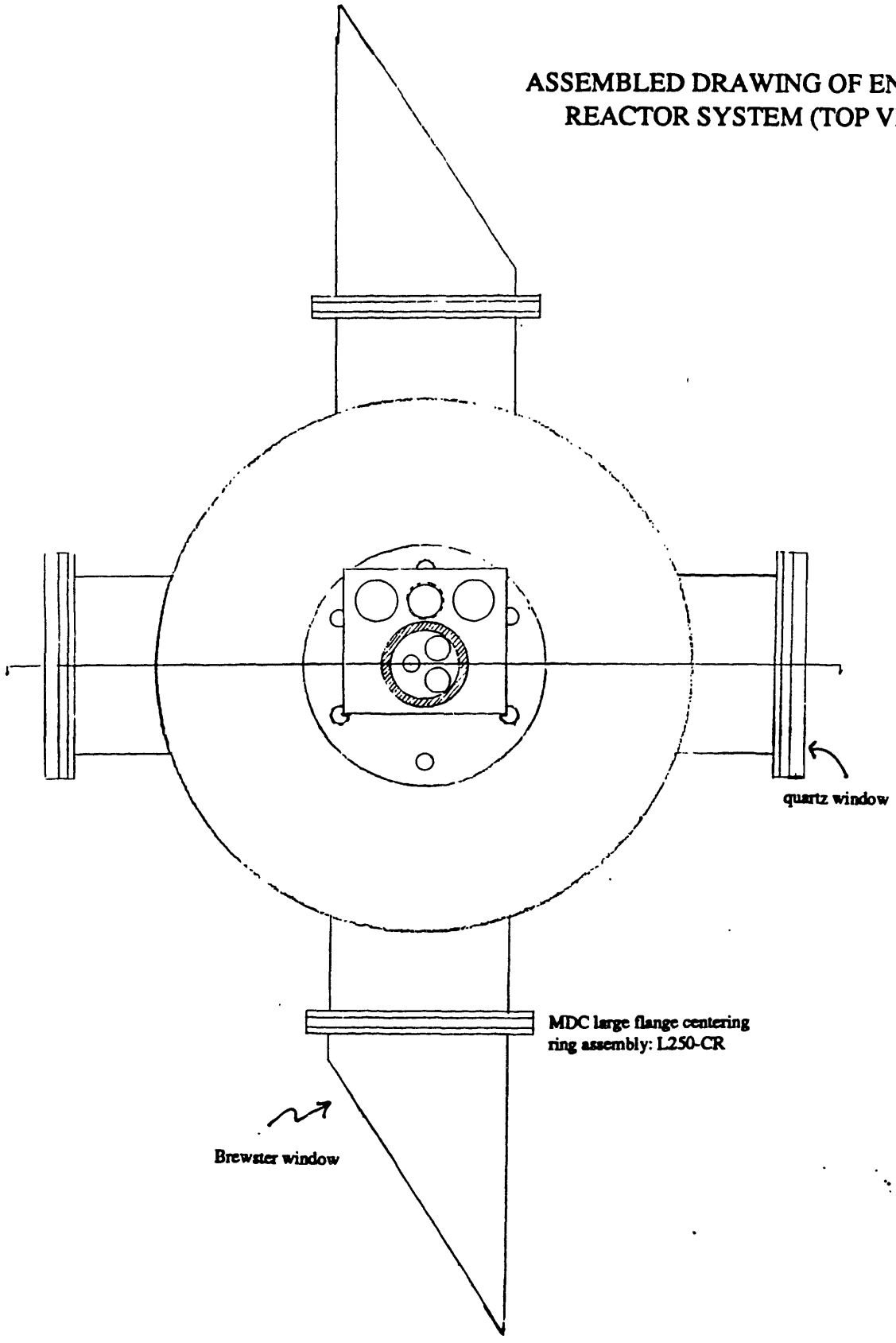
PLASMA REACTOR

(LARGE GLASS CHAMBER - ASSEMBLED DRAWING)

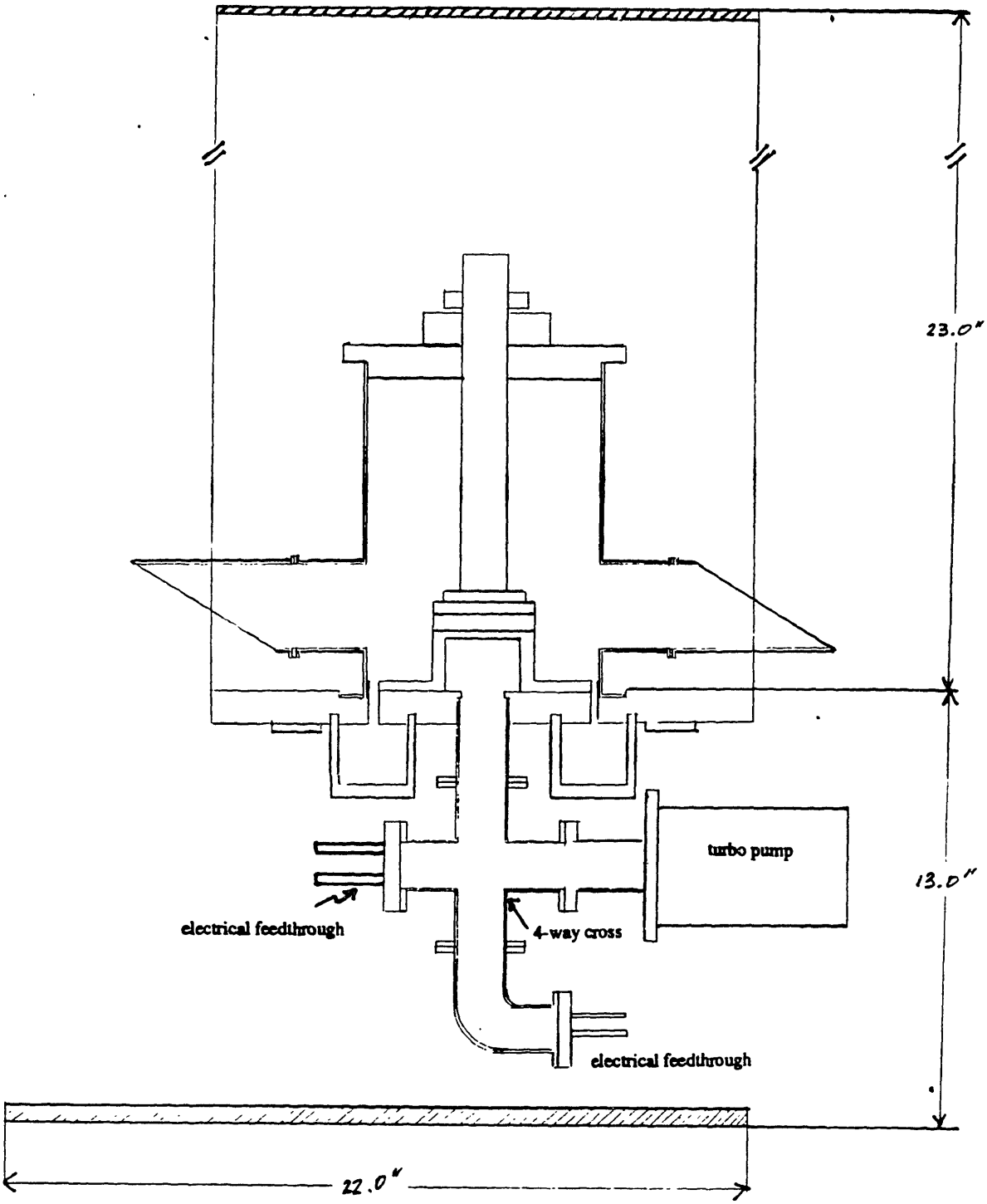
REACTOR1.DRW



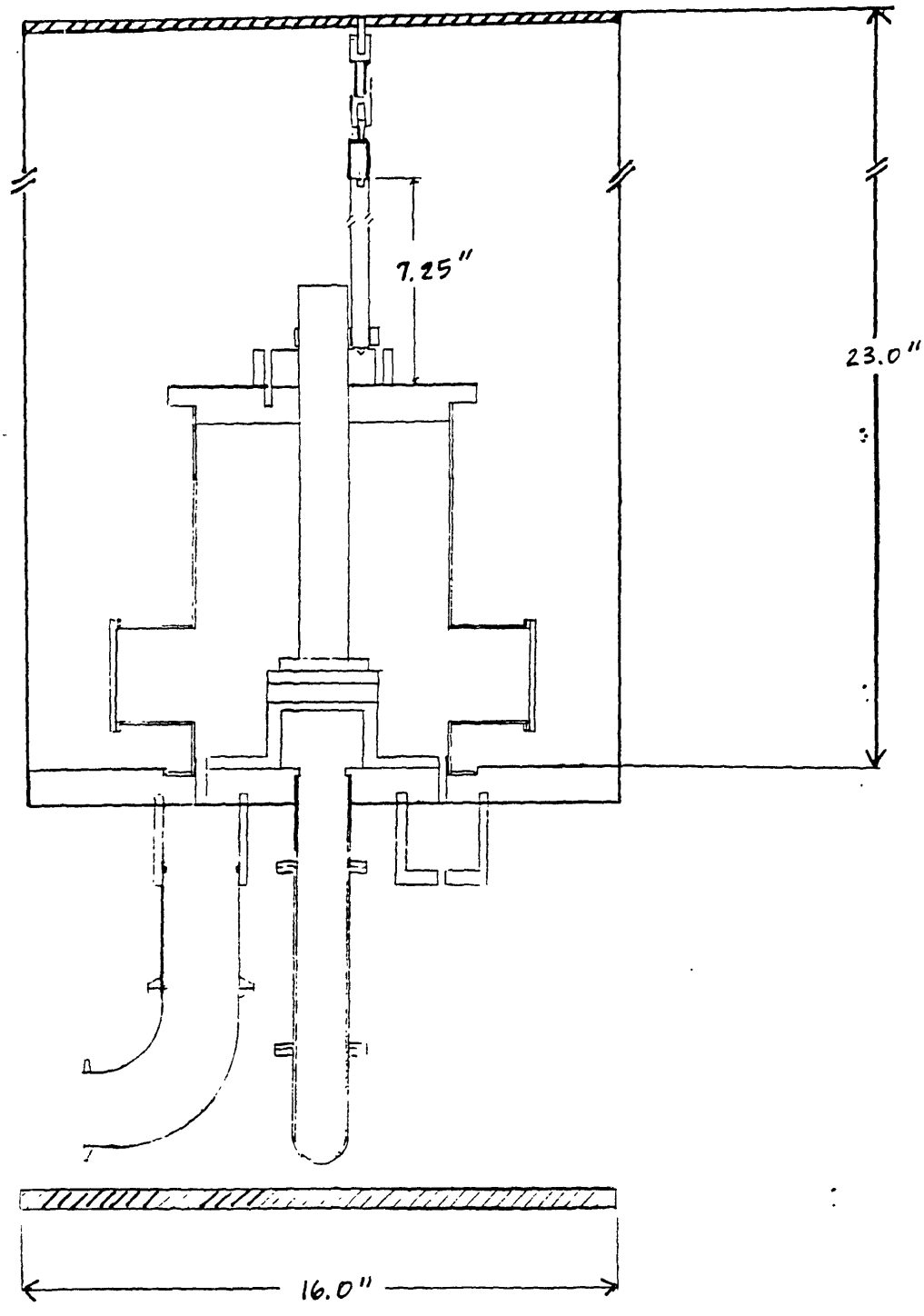
ASSEMBLED DRAWING OF ENTIRE REACTOR SYSTEM (TOP VIEW)



ASSEMBLED DRAWING OF ENTIRE REACTOR SYSTEM (VIEW 1)



ASSEMBLED DRAWING OF ENTIRE
REACTOR SYSTEM (VIEW 2)

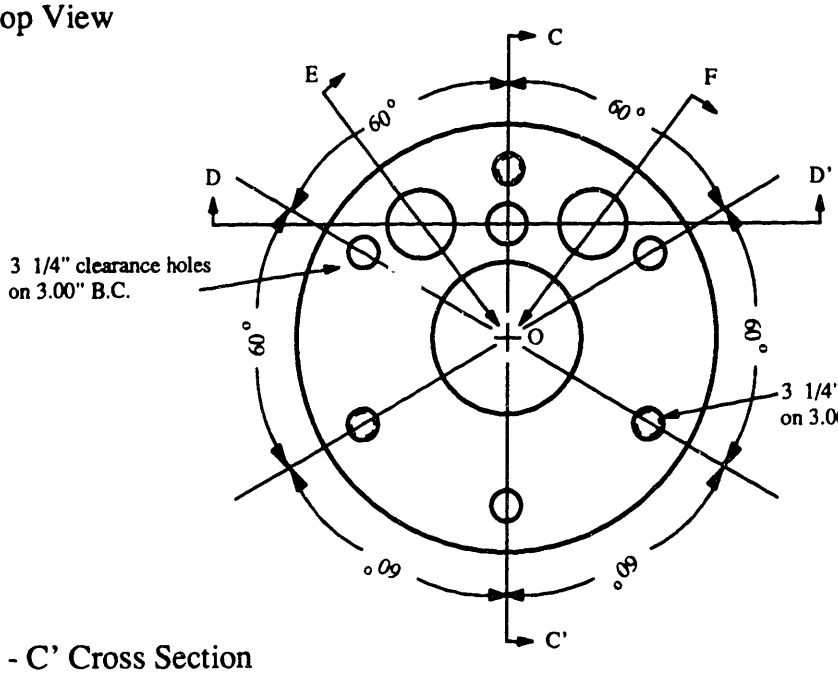


PLASMA REACTOR

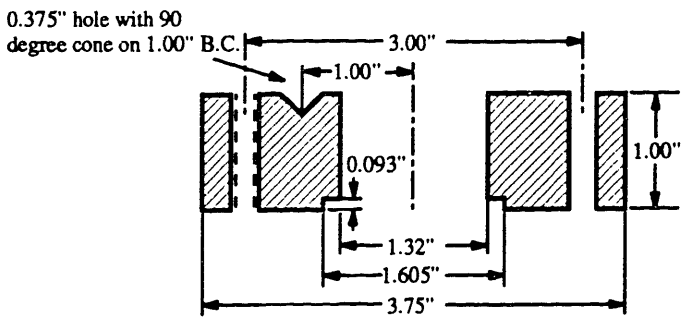
(BRASS)

PIECE2.DRW

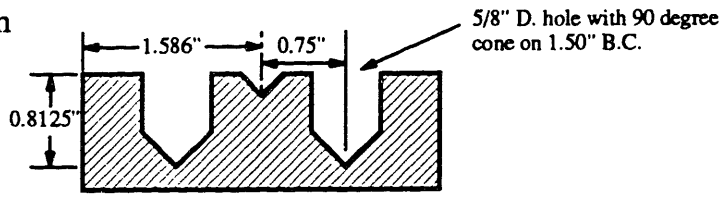
Top View



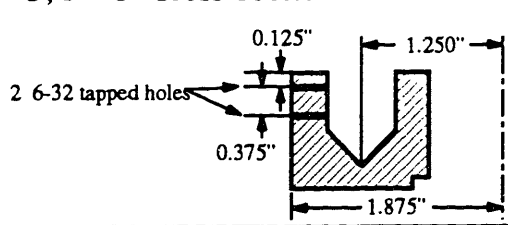
C - C' Cross Section



D - D' Cross Section



E - O, F - O Cross Section

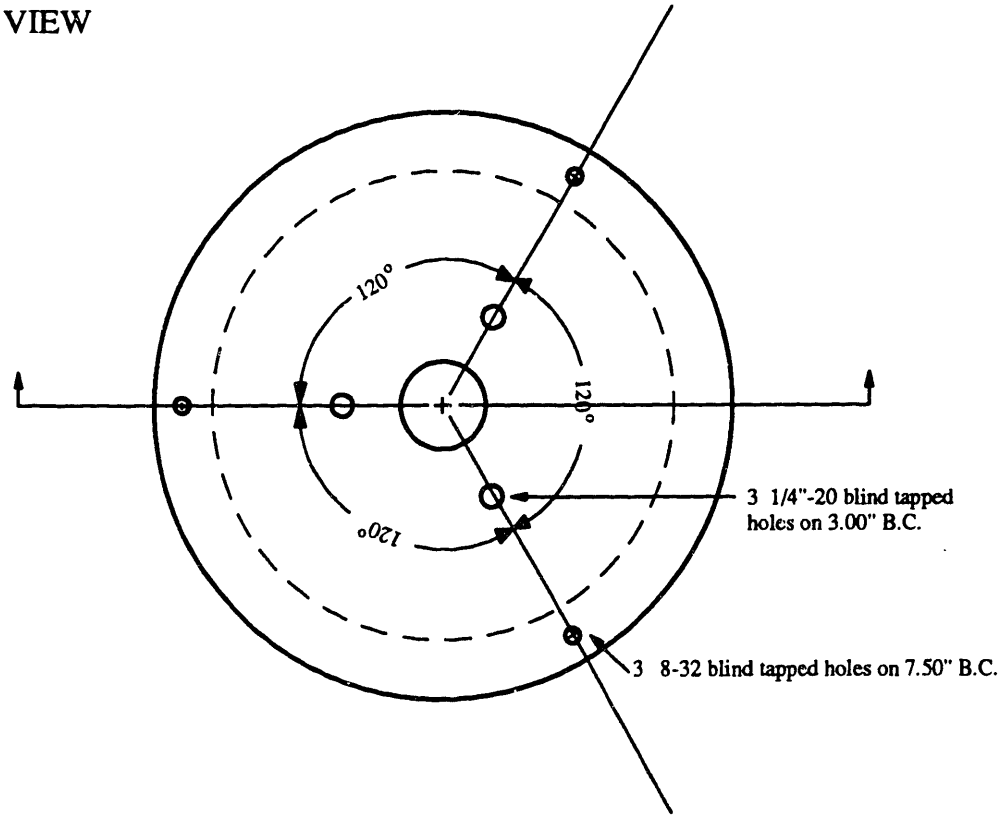


PLASMA REACTOR

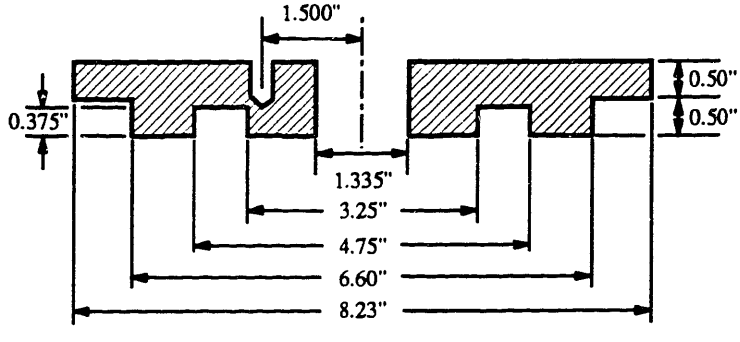
(ALUMINUM LID - TOP OF GLASS REACTOR)

PIECE3A.DRW

TOP VIEW



CROSS SECTION



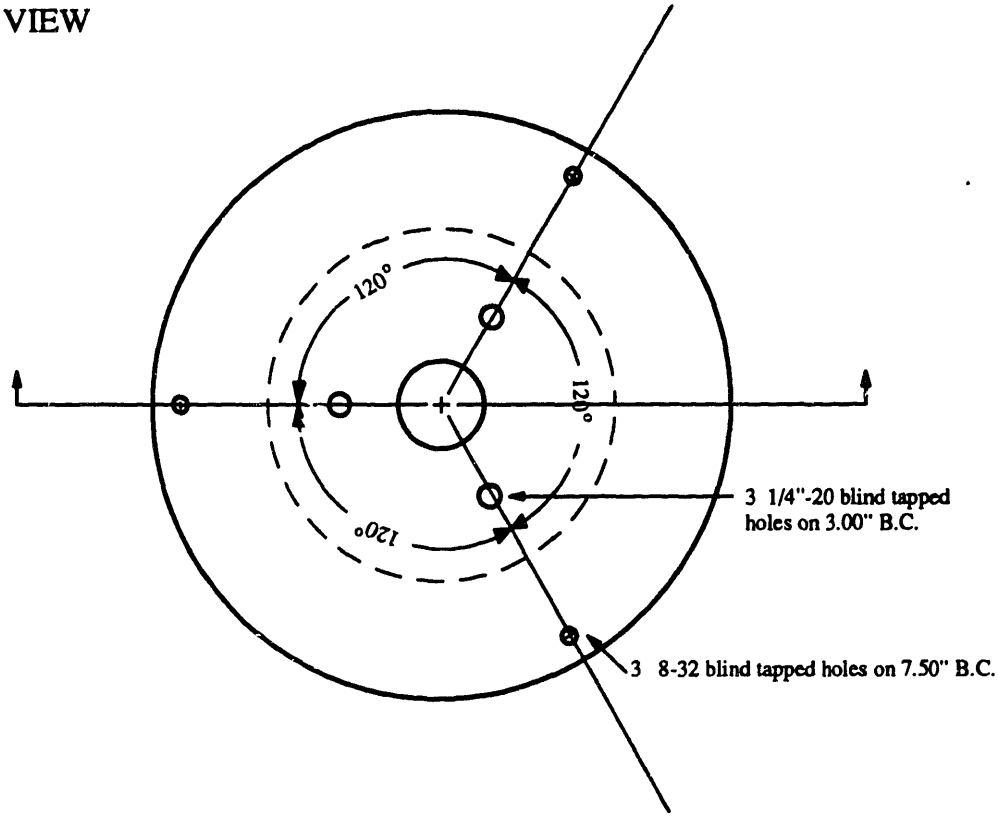
PLASMA REACTOR

Aug. 23, 1990

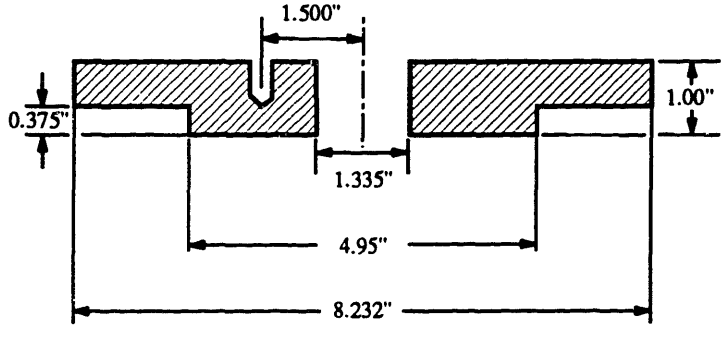
(ALUMINUM LID - TOP OF GLASS REACTOR)

PIECE3B.DRW

TOP VIEW



CROSS SECTION

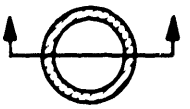


PLASMA REACTOR

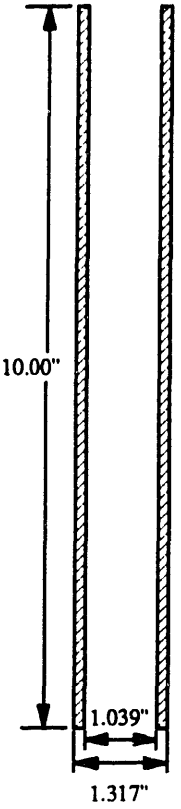
(STAINLESS STEEL TUBE)

PIECE4.DRW

TOP VIEW



CROSS SECTION

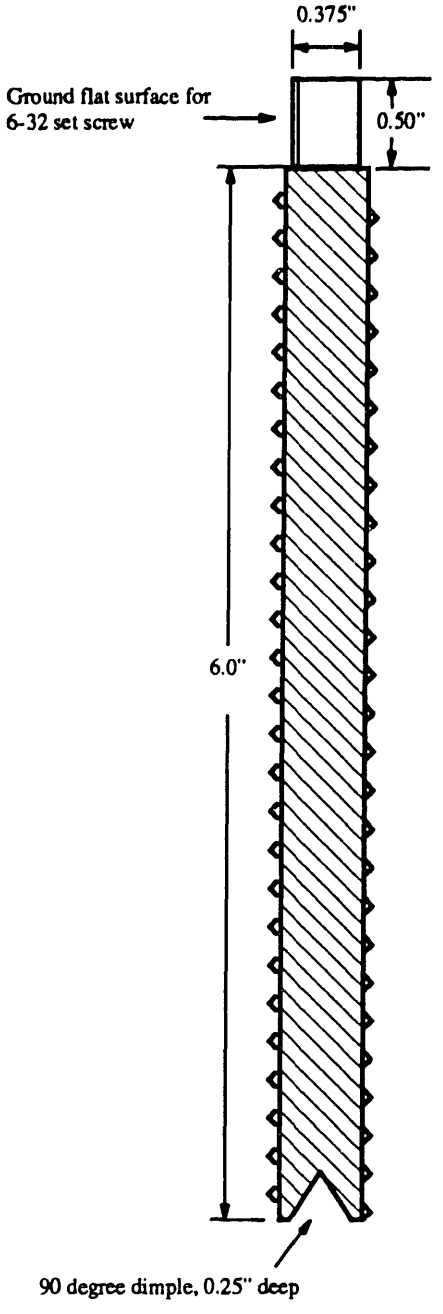


PLASMA REACTOR

(BRASS TRAVEL ROD - 1/2-20 SCREW)

PIECE14.DRW

Scale: 1" = 1"



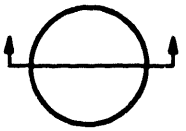
PLASMA REACTOR

(PRECISION GROUND STAINLESS STEEL RODS)

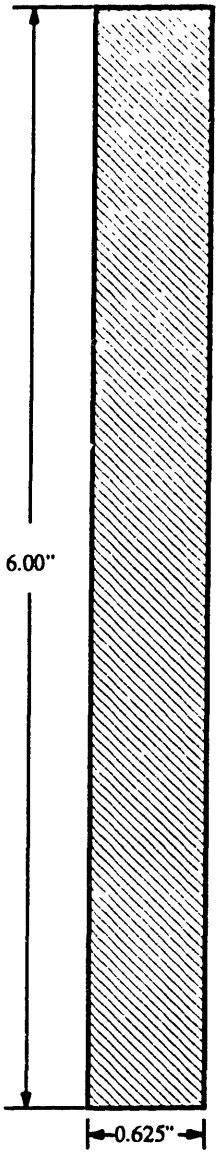
PIECE15.DRW

Scale: 1" = 1"
Make 2 pieces

Top View



Cross Section

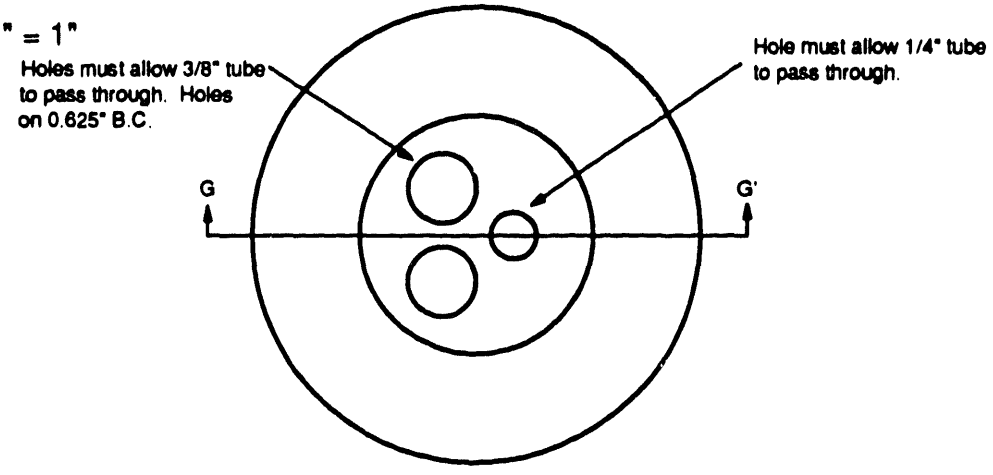


PLASMA REACTOR

(STAINLESS STEEL)

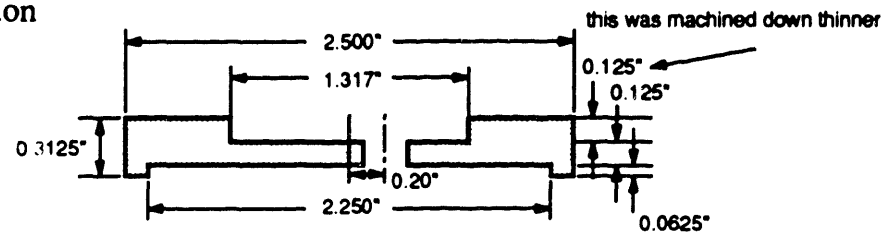
PIECE5.DRW

Scale: 1" = 1"

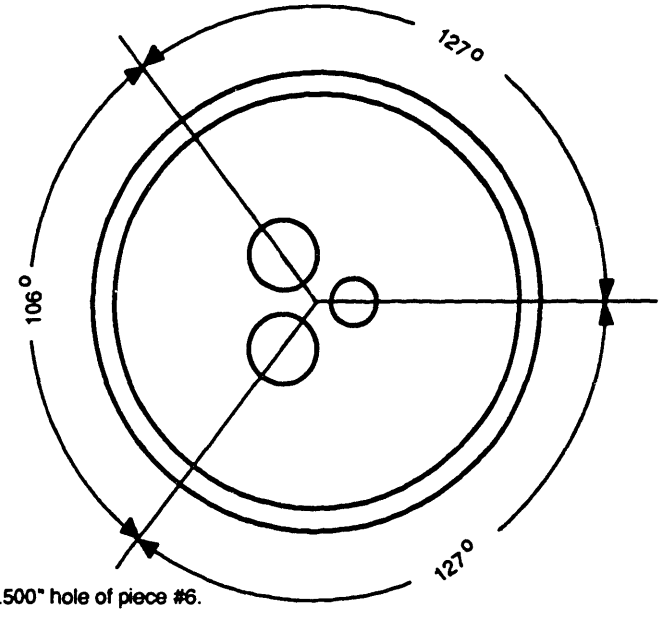


Top View

G - G' Cross Section



Bottom View



Note: This piece fits into the 2.500" hole of piece #6.

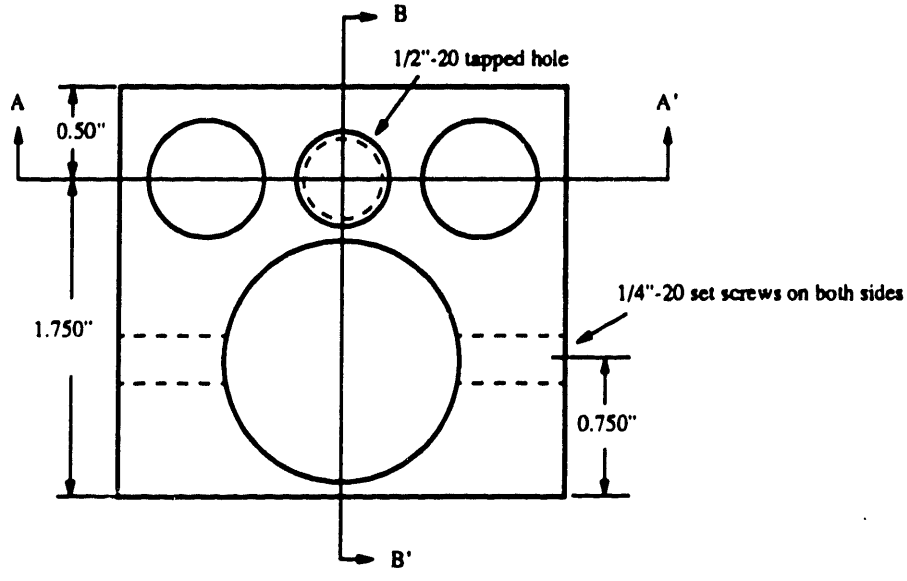
PLASMA REACTOR

(BRASS)

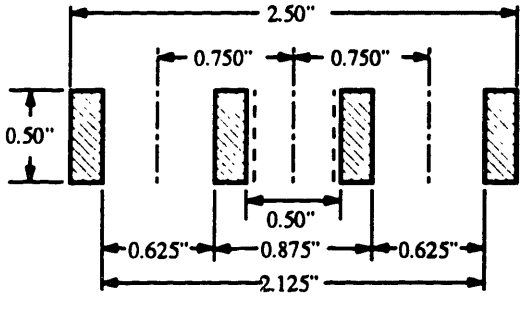
PIECE1.DRW

SCALE: 1"=1"

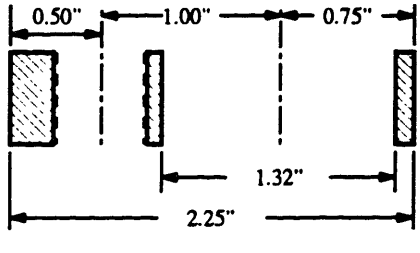
Top View



A - A' Cross Section



B - B' Cross Section

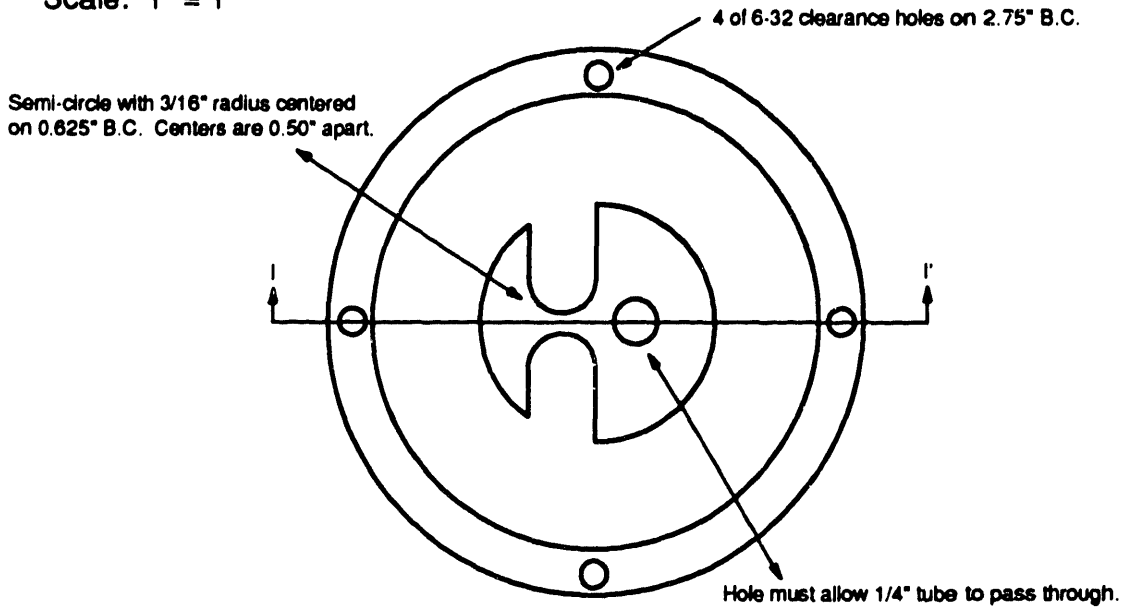


PLASMA REACTOR

(STAINLESS STEEL)

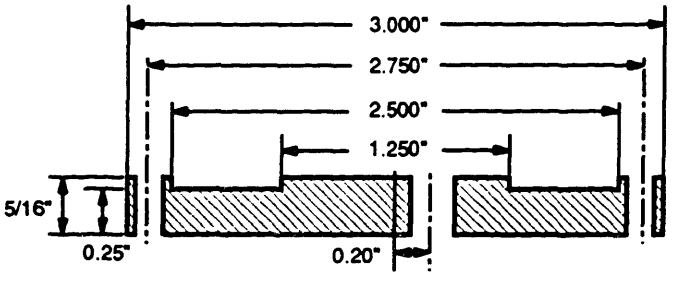
PIECE6.DRW

Scale: 1" = 1"



TOP VIEW

I - I' CROSS SECTION



Note: The 2.500" hole must fit with piece #5

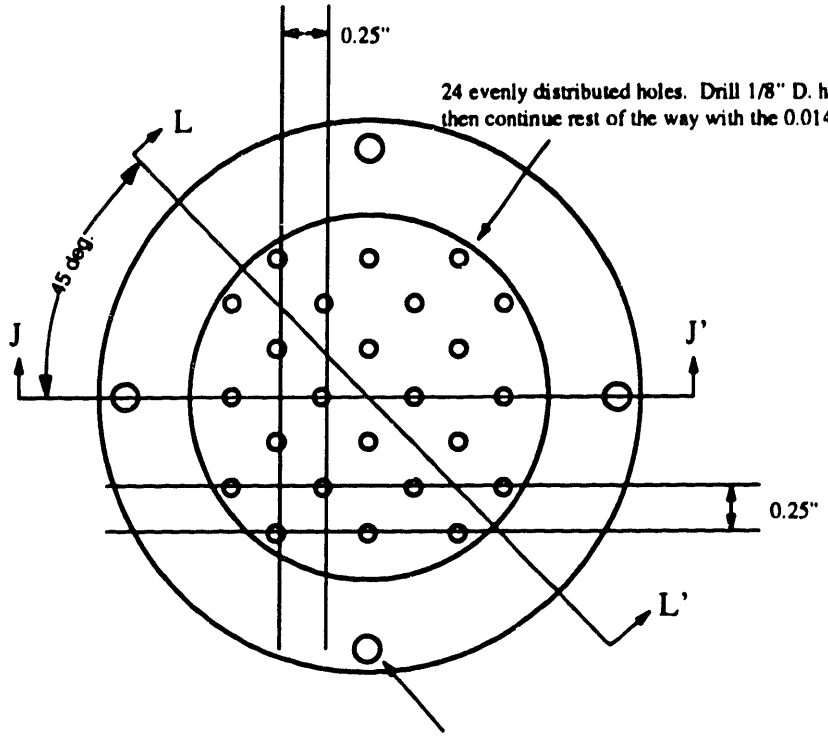
PLASMA REACTOR

(3.0" D. POWERED ELECTRODE)

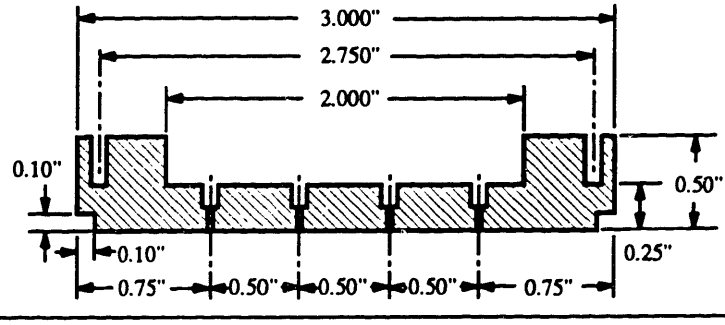
PIECE7.DRW

Top View

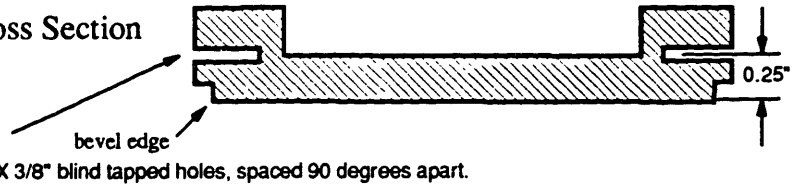
Material: Aluminum
Scale: 1" = 1"



J -J' Cross Section



L - L' Cross Section

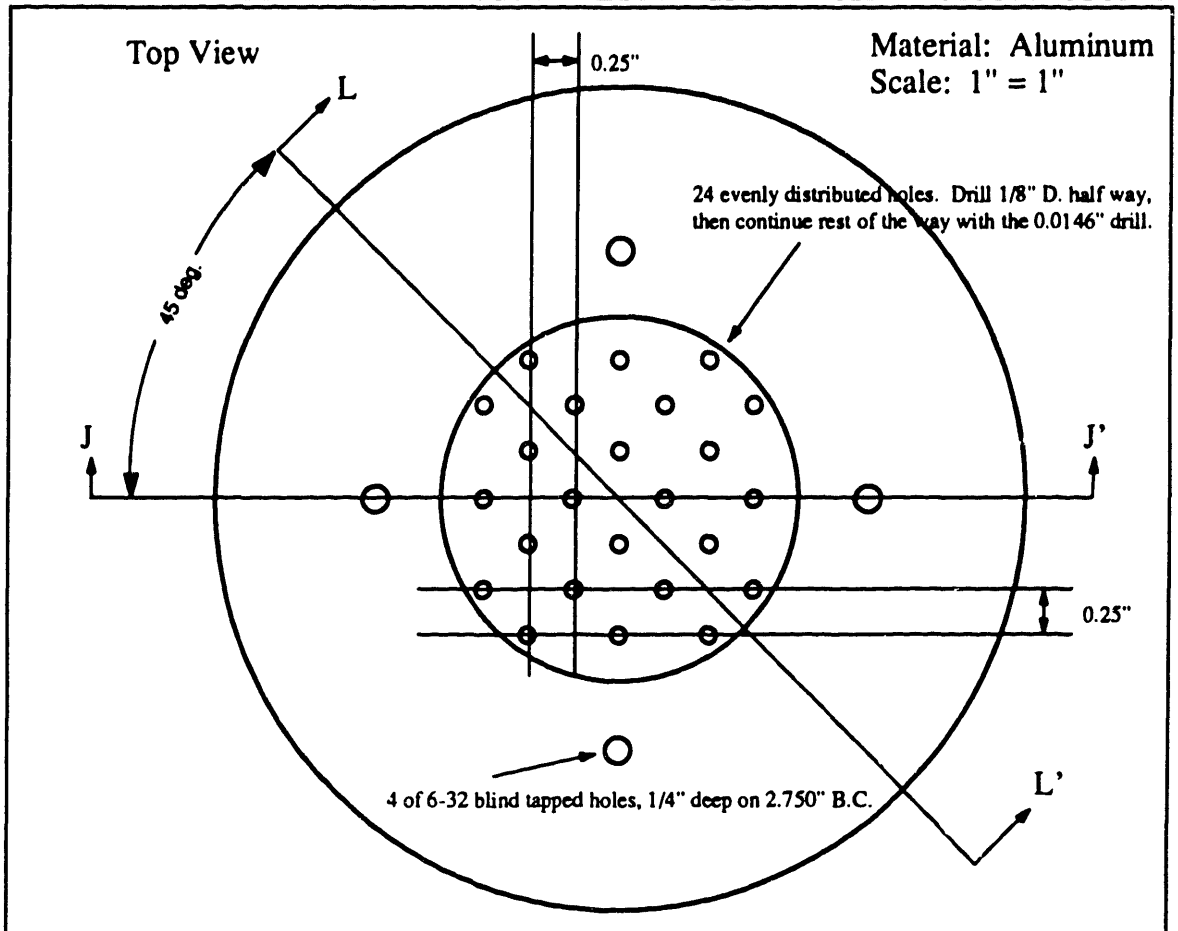


4 of 6-32 X 3/8" blind tapped holes, spaced 90 degrees apart.

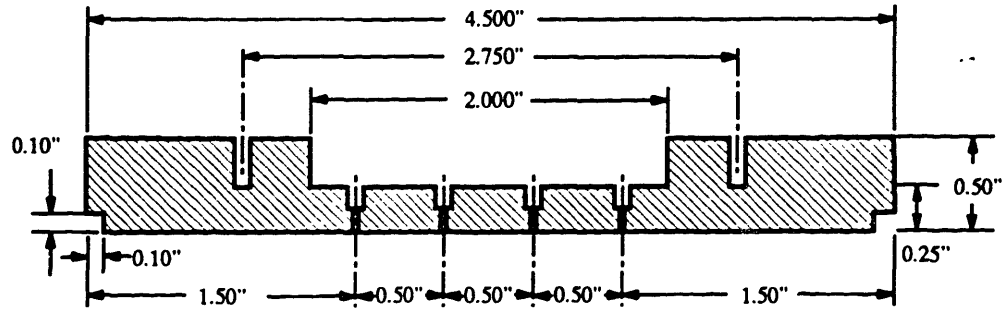
PLASMA REACTOR

(4.5" D. POWERED ELECTRODE)

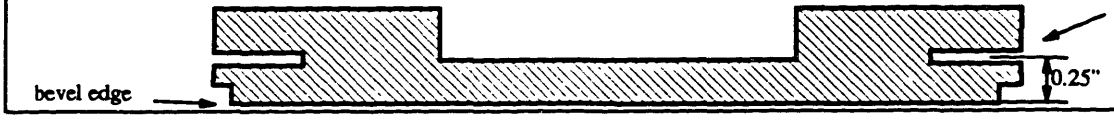
PIECE8.DRW



J - J' Cross Section



L - L' Cross Section



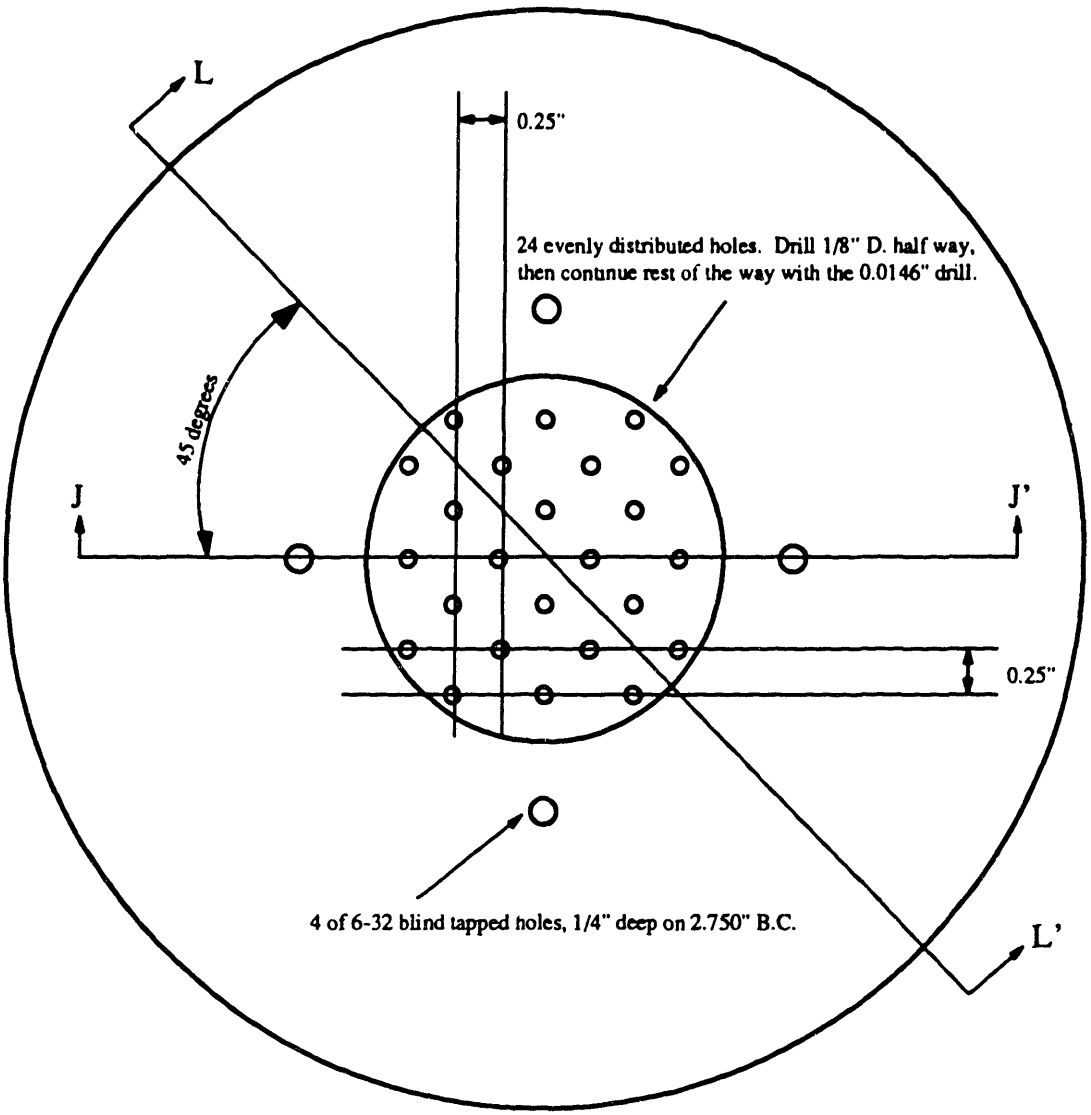
PLASMA REACTOR

(6.0" D. POWERED ELECTRODE)

PIECE9A.DRW

Top View

Material: Aluminum
Scale: 1" = 1"

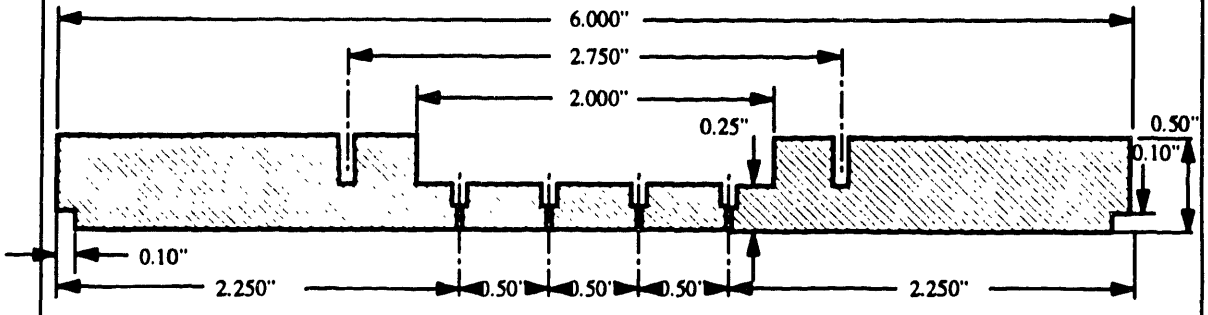


PLASMA REACTOR

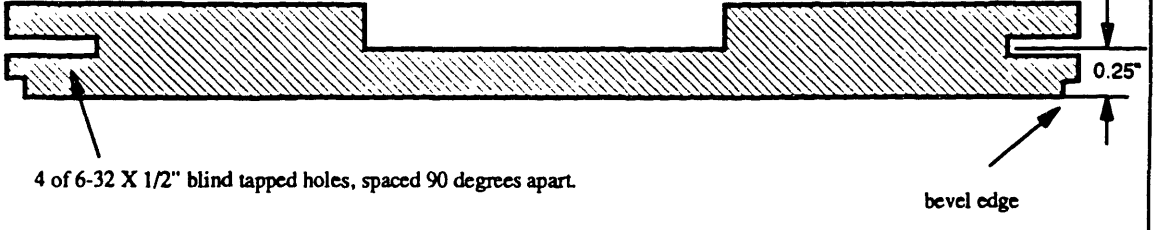
(6.0" D. POWERED ELECTRODE)

PIECE9B.DRW

J - J' Cross Section



L - L' Cross Section



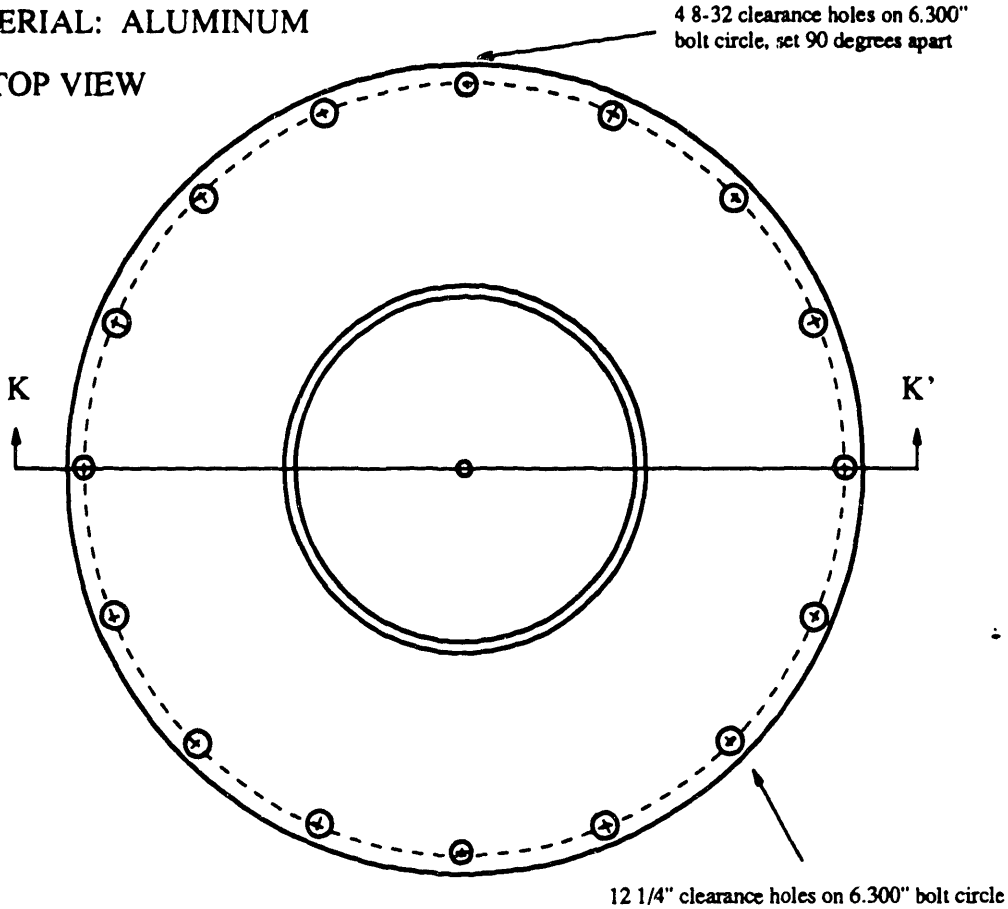
PLASMA REACTOR

(3" D. GROUND ELECTRODE)

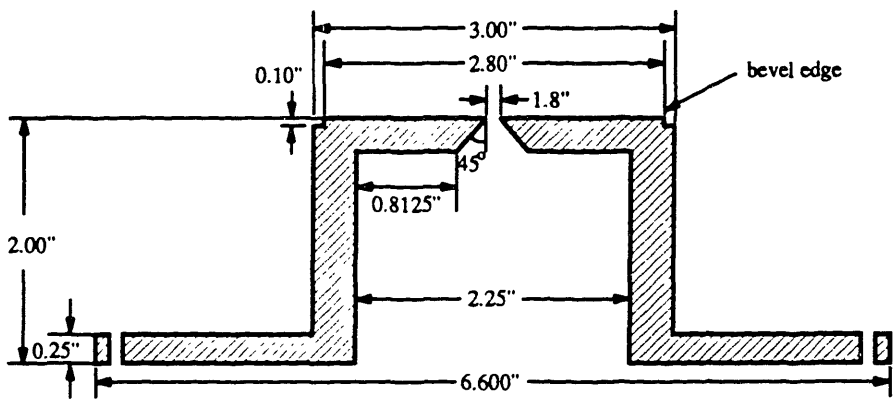
PIECE10.DRW

MATERIAL: ALUMINUM

TOP VIEW



K - K' CROSS SECTION



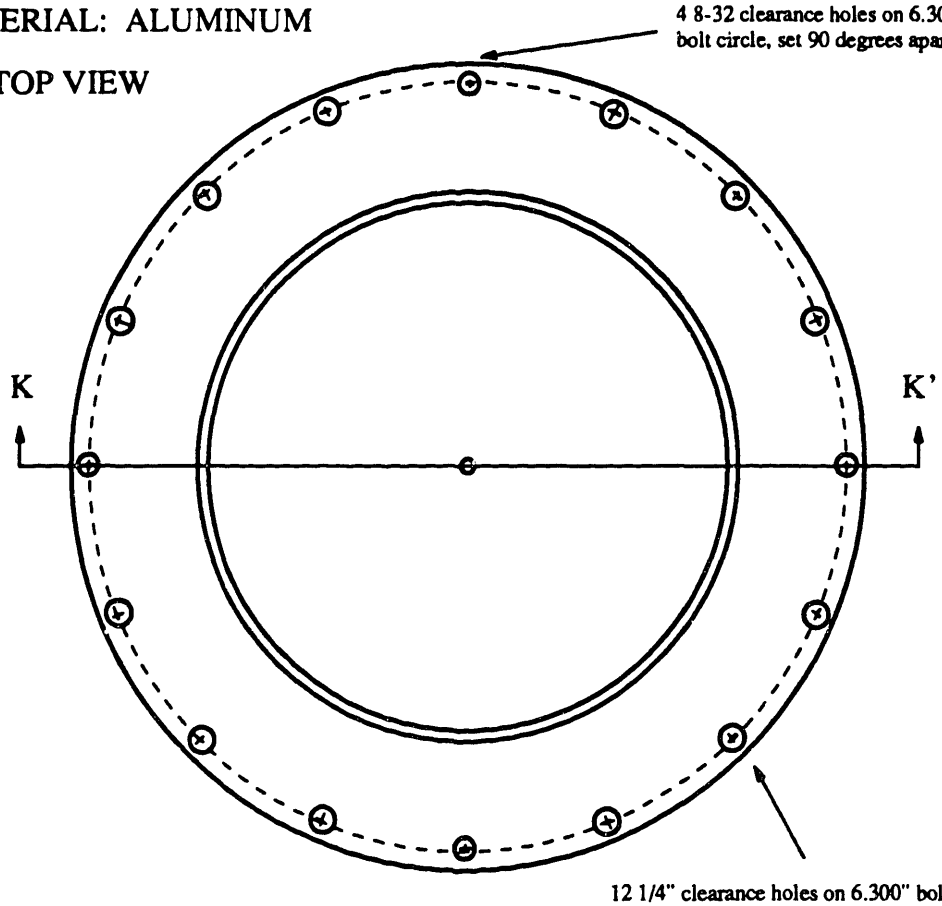
PLASMA REACTOR

(4.5" D. GROUND ELECTRODE)

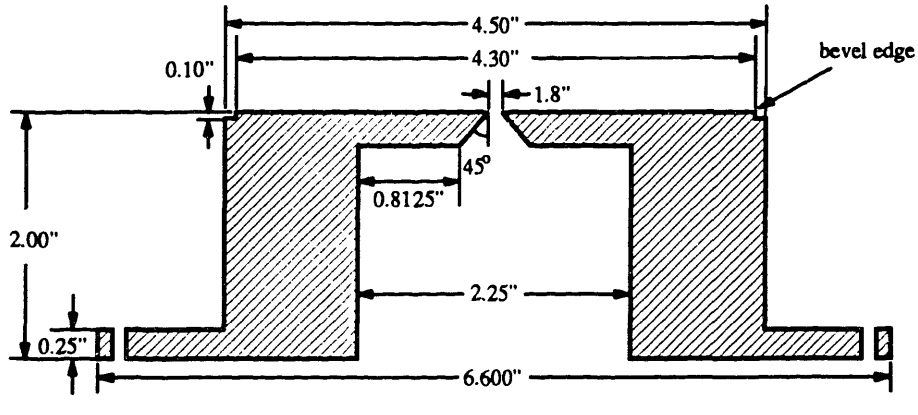
PIECE11.DRW

MATERIAL: ALUMINUM

TOP VIEW



K - K' CROSS SECTION



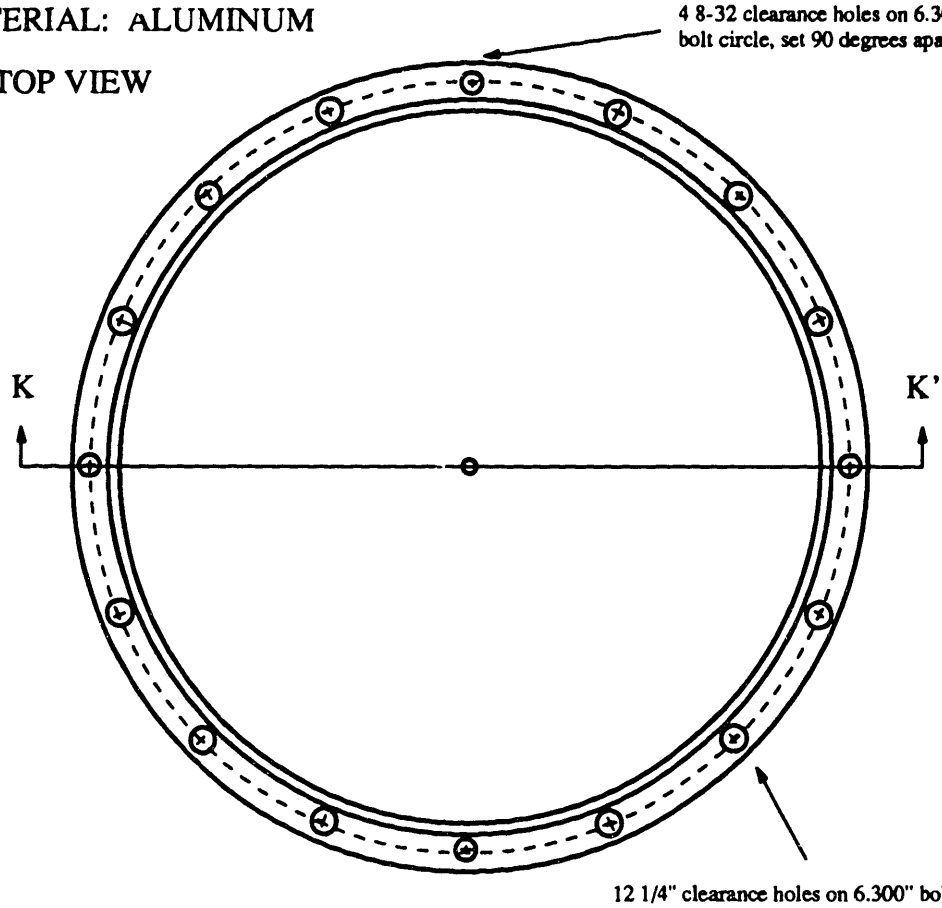
PLASMA REACTOR

(6.0" D. GROUND ELECTRODE)

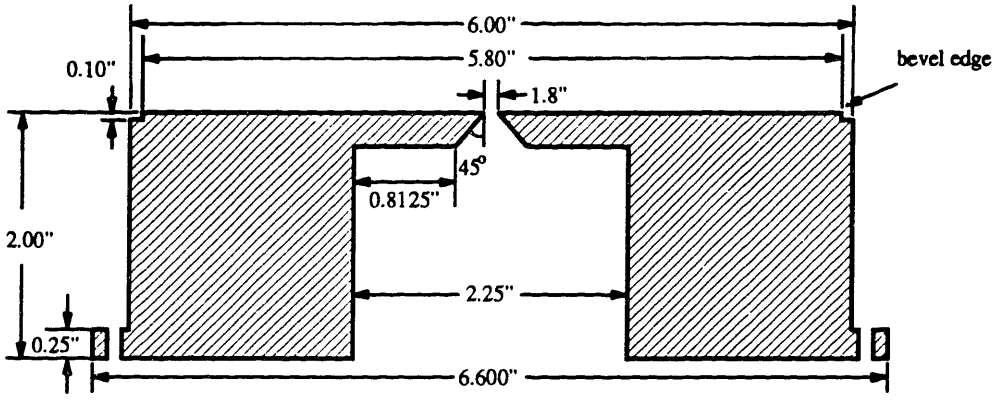
PIECE12.DRW

MATERIAL: ALUMINUM

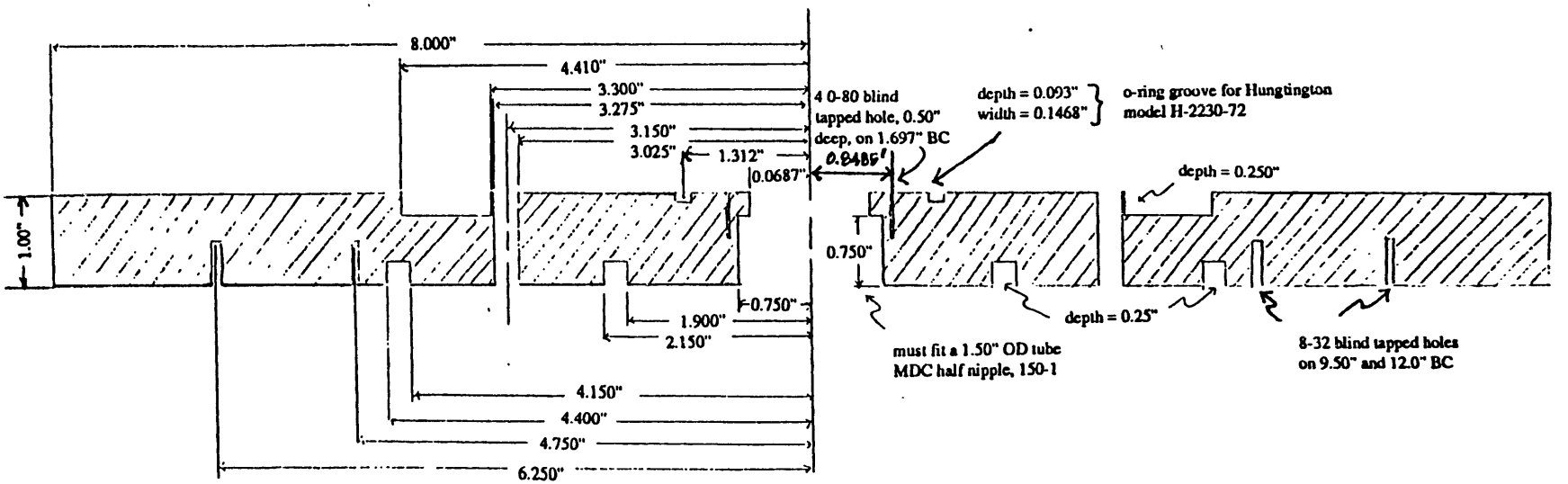
TOP VIEW

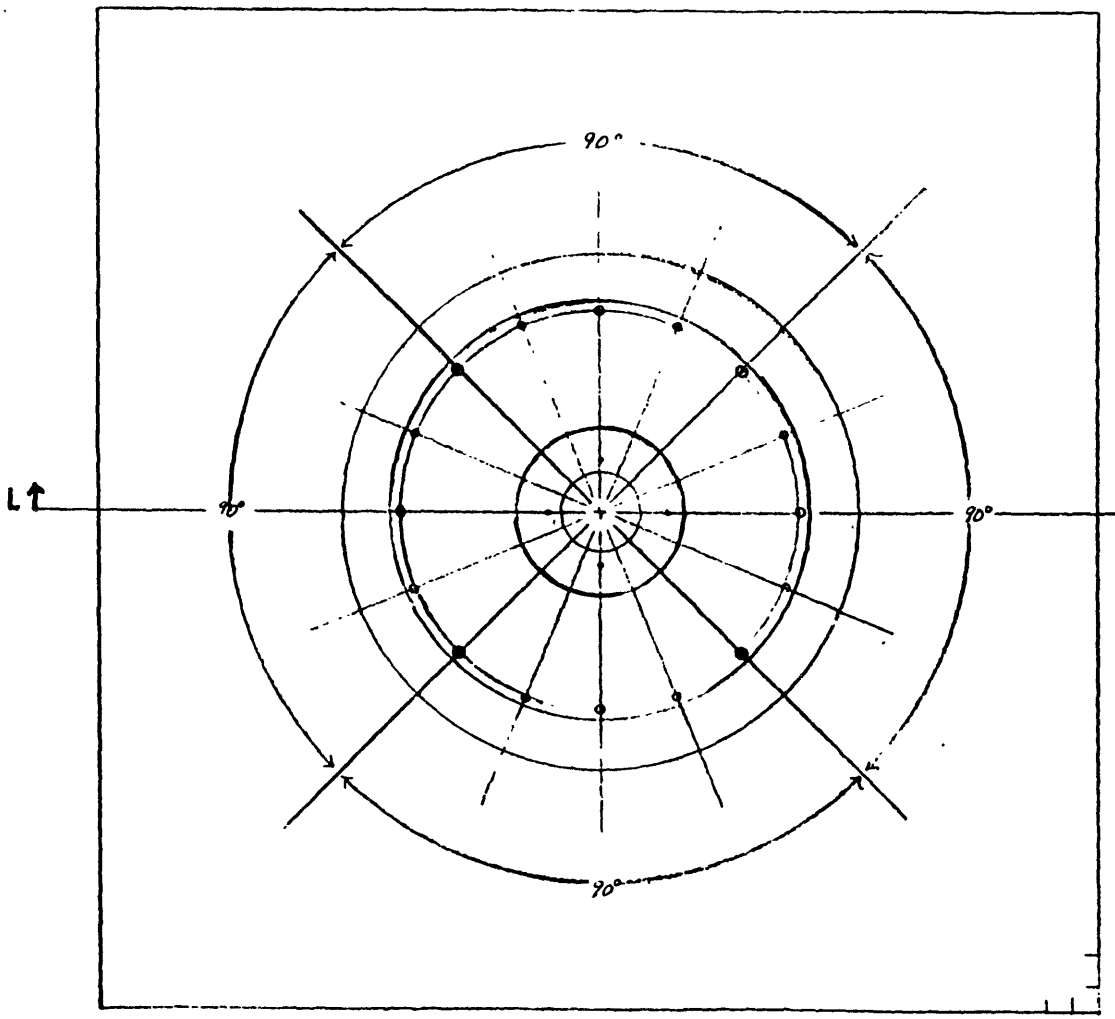


K - K' CROSS SECTION



L - L' CROSS SECTION OF BASE PLATE

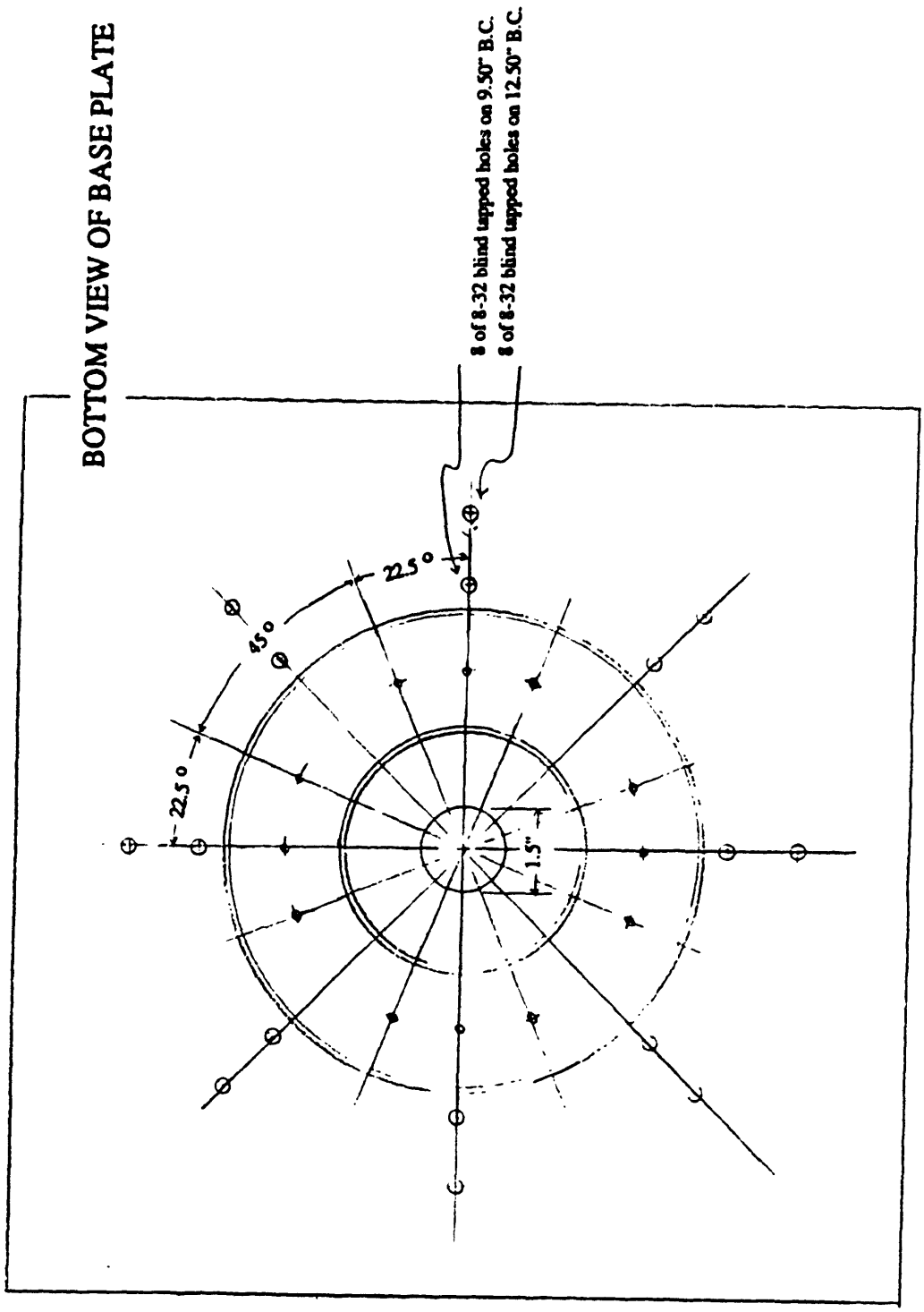




ALUMINUM BASE PLATE
(Top View)

16 equally spaced holes on 6.3" B.C.
4 of them are 8-32 blind tapped holes, set 90 degrees apart.
The rest are 1/4" holes.

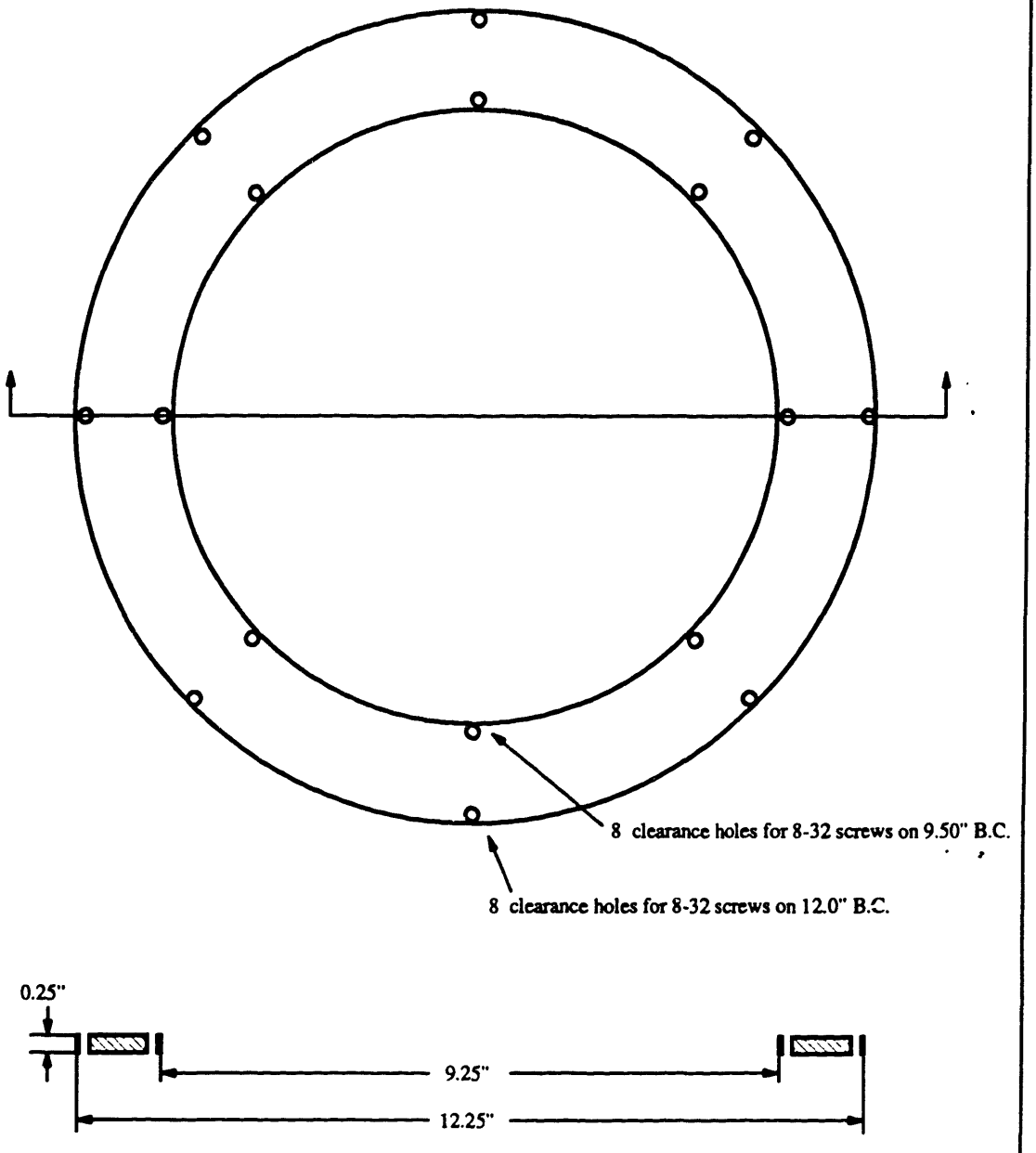
10-32 x 1/5" deep tapped holes. Four at each corner. Pairs of holes are staggered with the other side.


BOTTOM VIEW OF BASE PLATE
8 of 8-32 blind tapped holes on 9.50" B.C.
8 of 8-32 blind tapped holes on 12.50" B.C.
22.5°
45°
22.5°
1.5"

PLASMA REACTOR

(BRASS RING FOR MOUNTING WATER COOLING COILS)

PIECE16.DRW

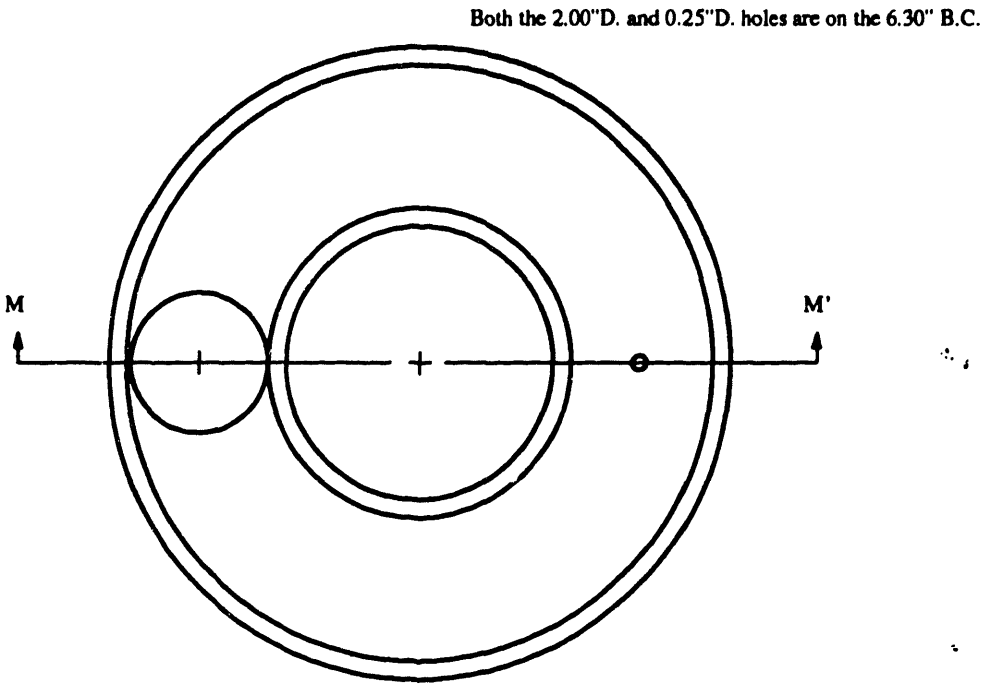


PLASMA REACTOR

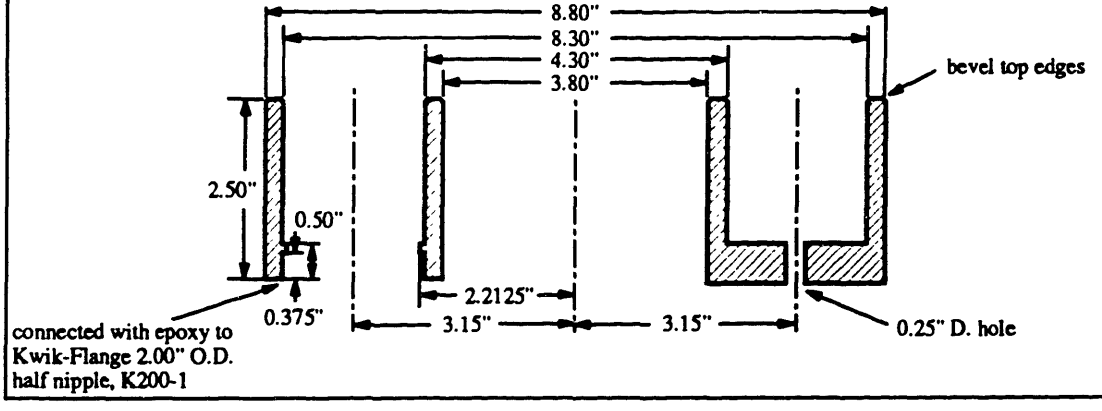
(ALUMINUM ANNULAR RING)

PIECE17.DRW

Top View



M - M' Cross Section



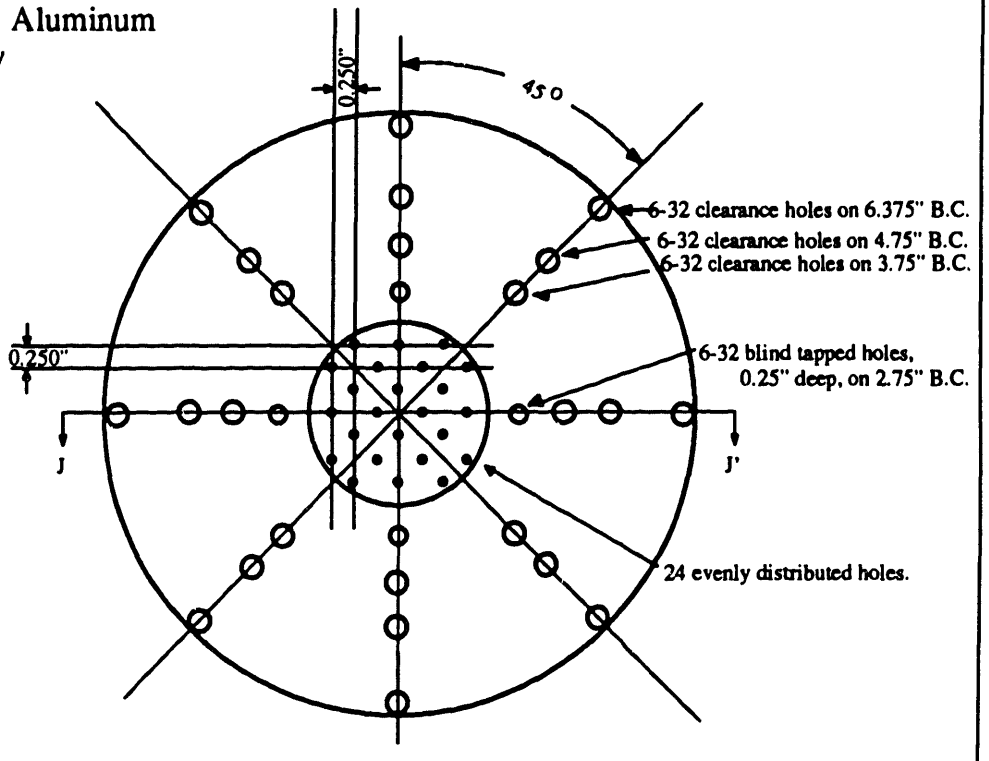
PLASMA REACTOR

May 1, 1989

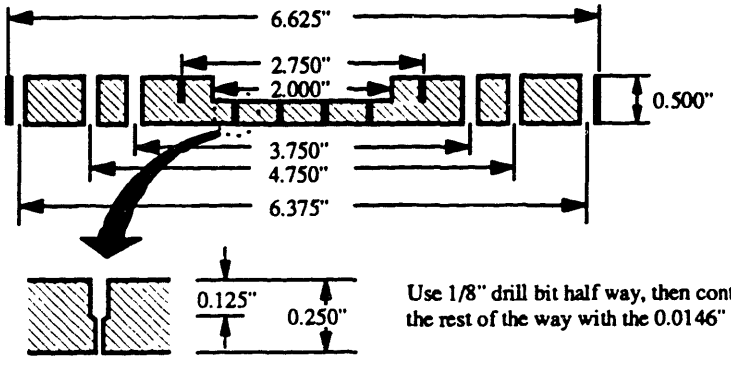
(ASYMMETRIC ELECTRODE CONFIGURATION)

PIECE13.DRW

Material: Aluminum
Top View



J - J' Cross Section



Anodized at Arborway Metal Finishing, Inc.
50 Park St., Dorchester. 288-1200

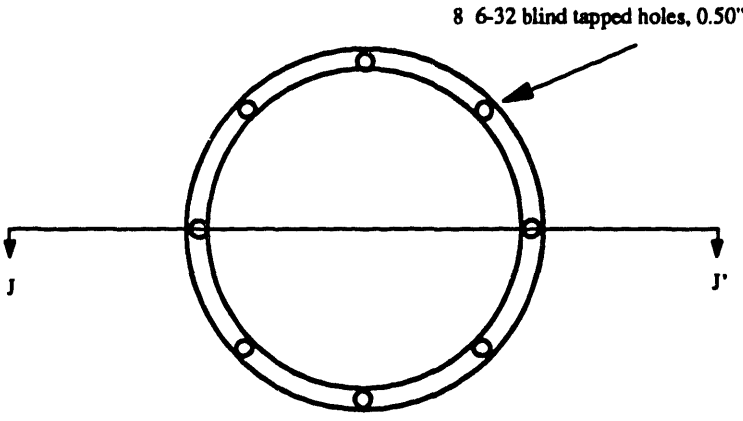
PLASMA REACTOR

May 1, 1989

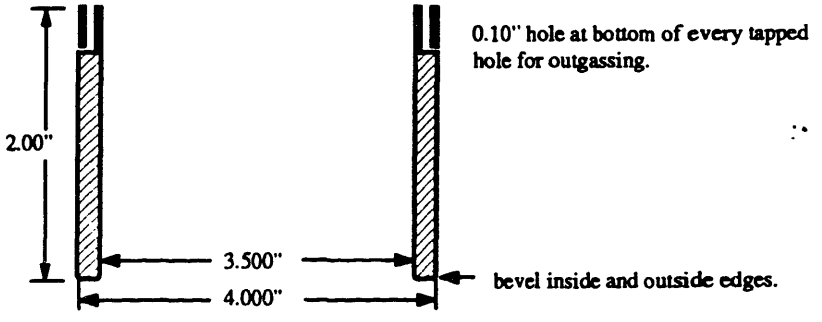
(ASYMMETRIC ELECTRODE CONFIGURATION)

PIECE20.DRW

Material: Aluminum
Top View



J – J' Cross Section



Anodized at Arborway Metal Finishing, Inc.
50 Park St., Dorchester. 288-1200

PLASMA REACTOR

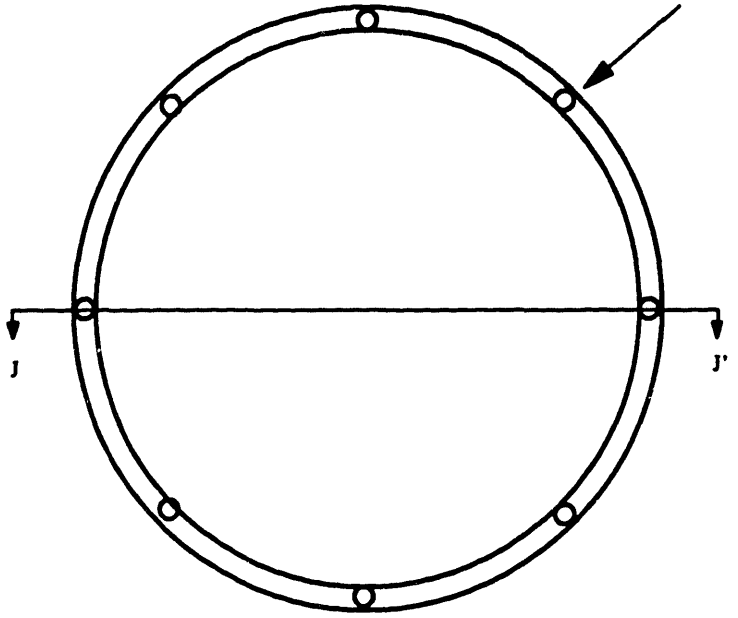
May 1, 1989

(ASYMMETRIC ELECTRODE CONFIGURATION)

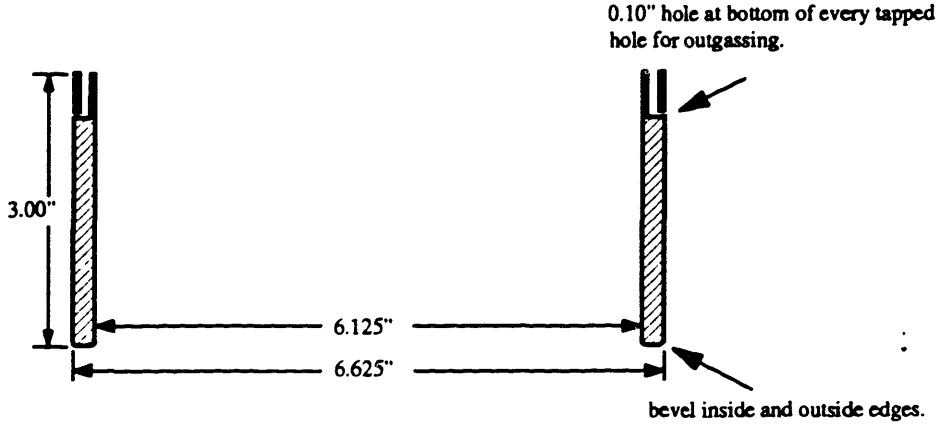
PIECE21.DRW

Material: Aluminum

Top View



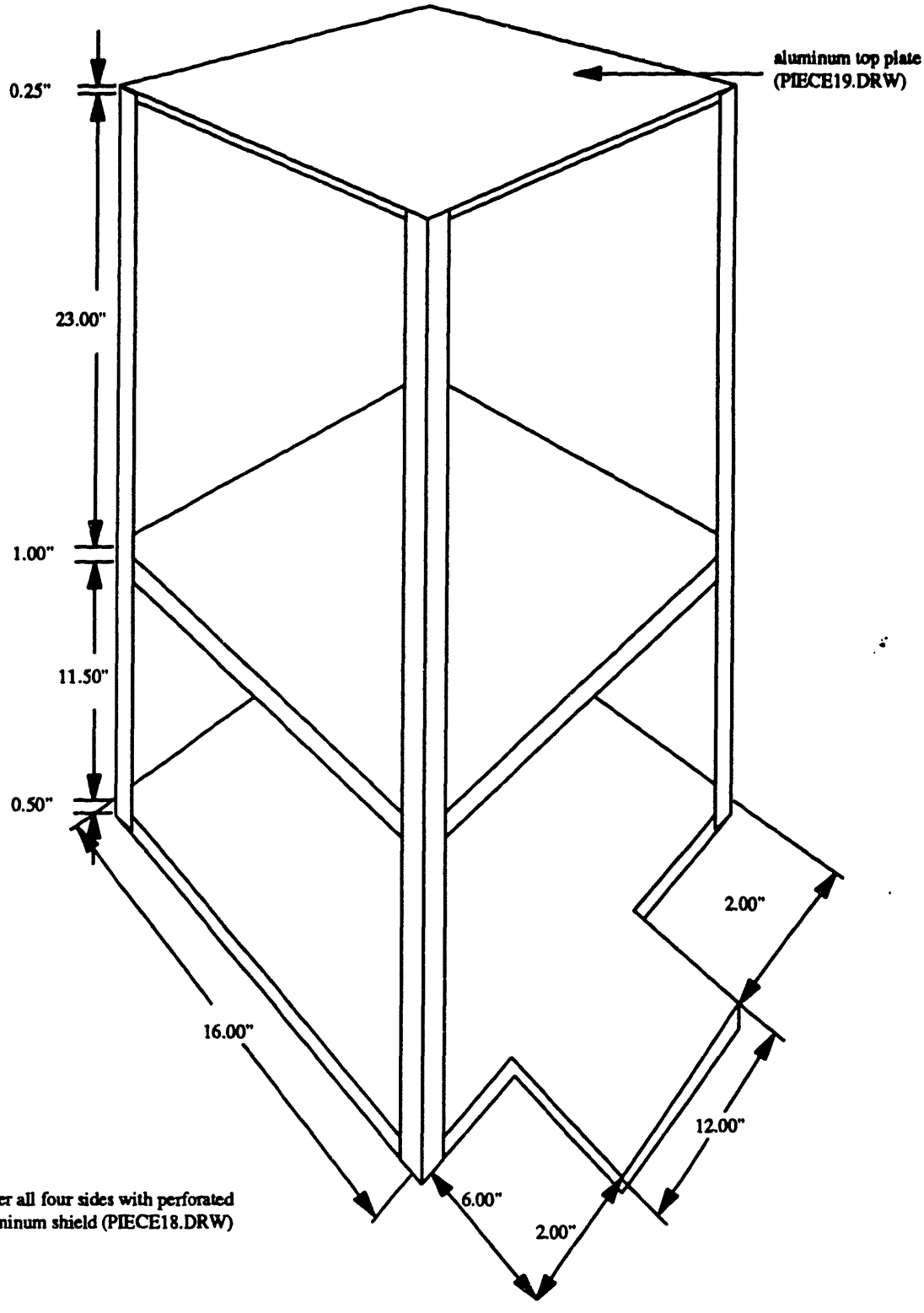
J – J' Cross Section



Anodized at Arborway Metal Finishing, Inc.
50 Park St., Dorchester. 288-1200

CAGE STRUCTURE TO HOLD REACTOR

PIECE27.DRW



Note: all three plates are bolted together to 3/16" thick, 1" wide aluminum angle

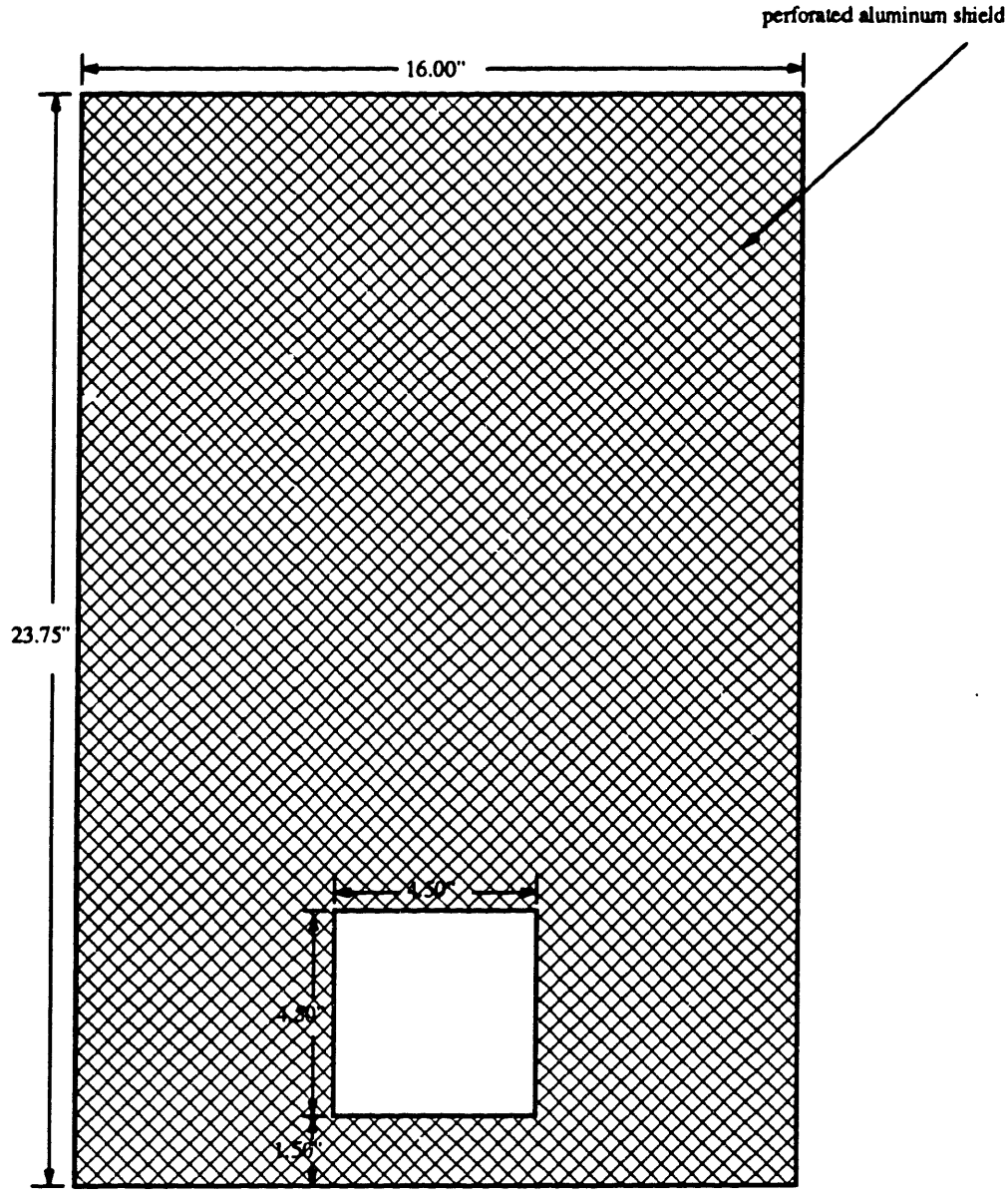
PLASMA REACTOR

(PERFORATED ALUMINUM SHIELD)

PIECE18.DRW

Scale: 1"=4"

Make 4 shields



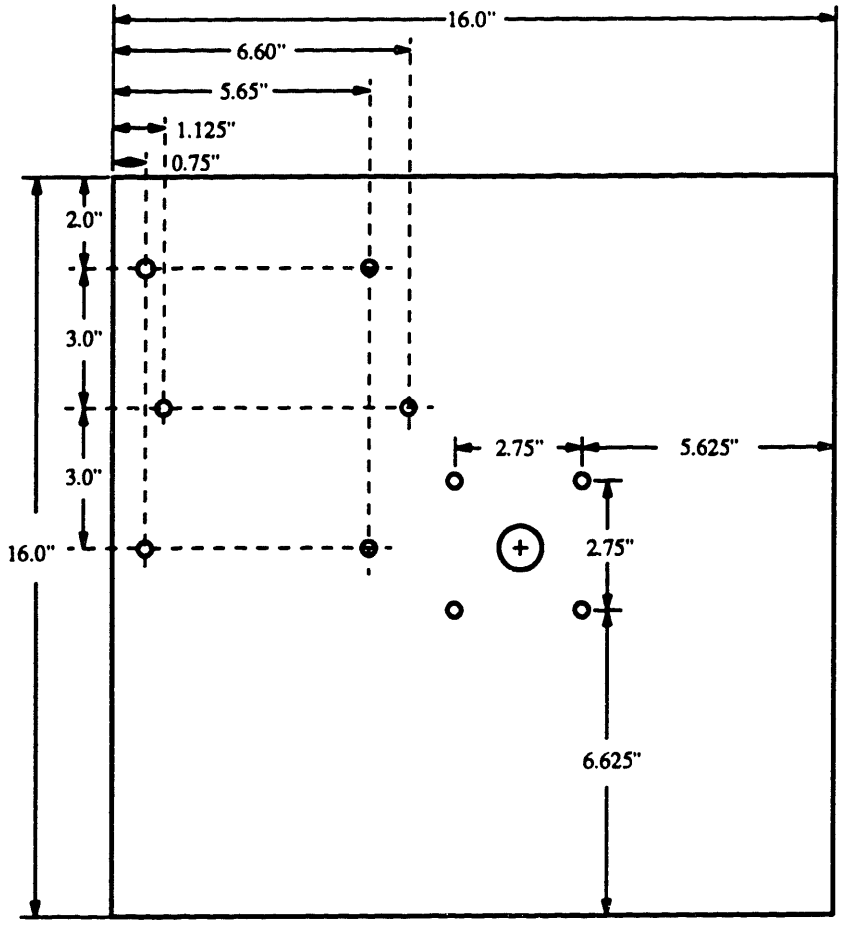
PLASMA REACTOR

(ALUMINUM TOP PLATE, 1/4" thick)

PIECE19.DRW

Scale: 1"=4"

small hole diameter = 0.20"

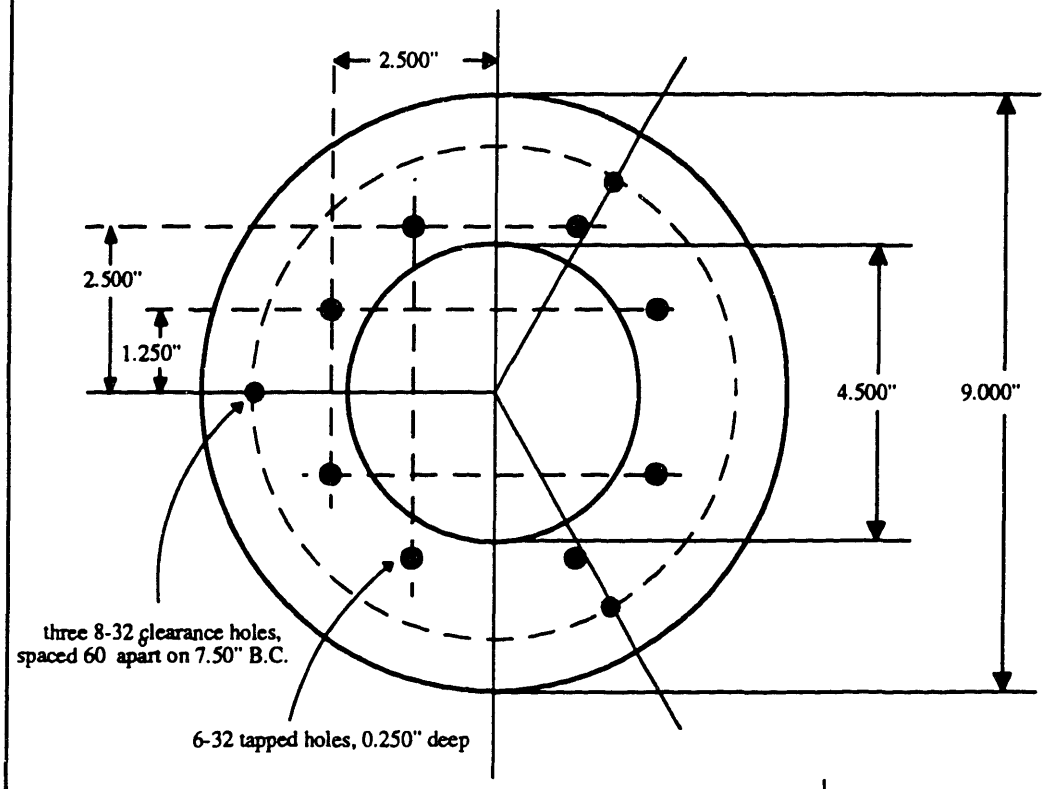


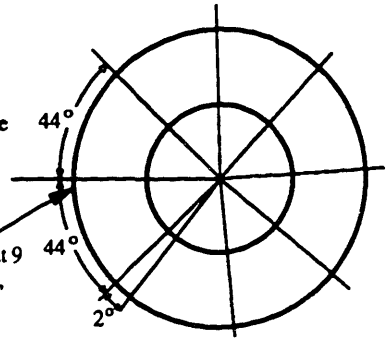
PLASMA REACTOR

(1/4" THICK ALUMINUM PLATE)

PIECE24.DRW

Top View





check with holes on perforated shield before adding holes

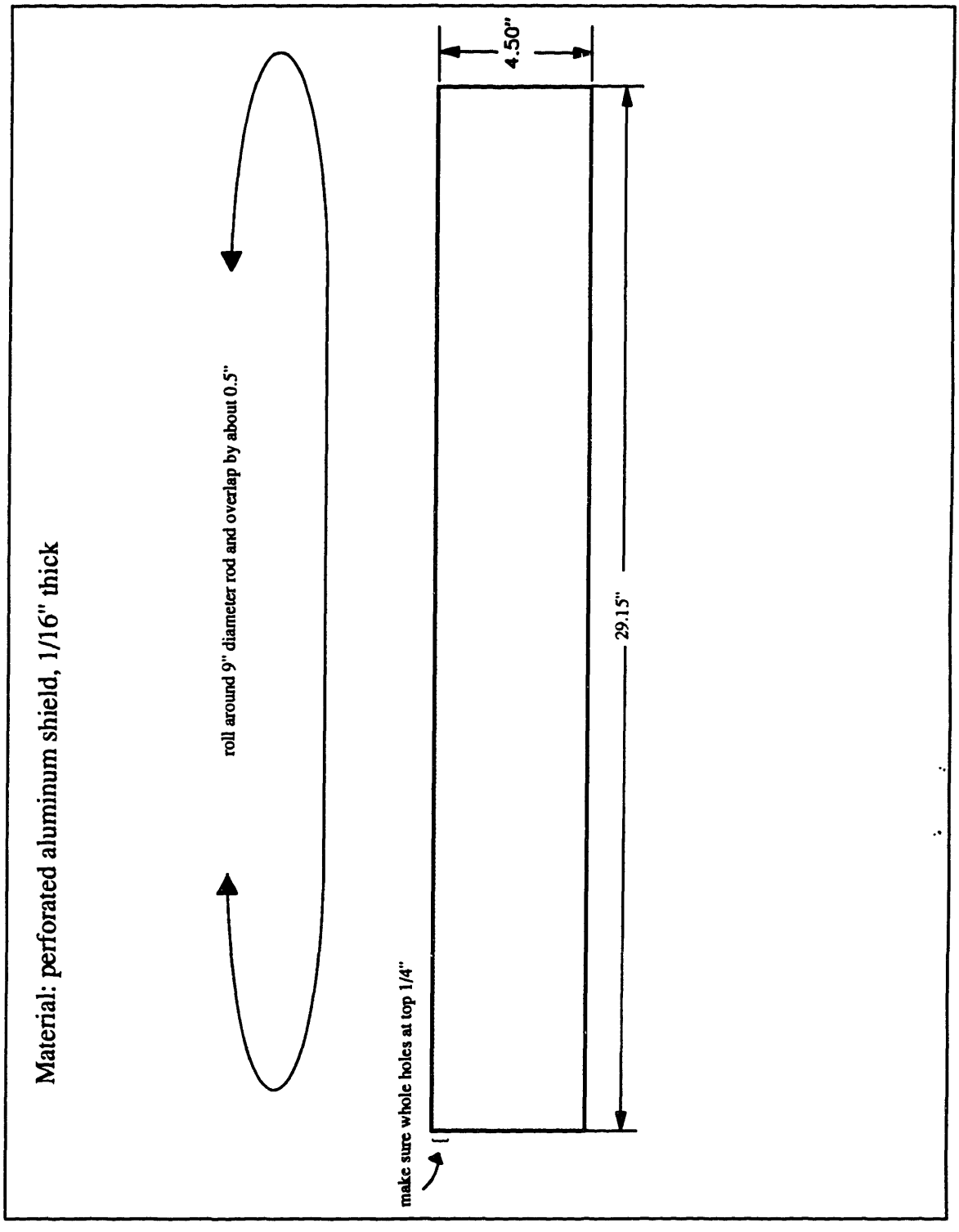
along side rim of aluminum plate, put 9 6-32 blind tapped holes, 0.500" deep, and spaced 44° apart

PLASMA REACTOR

(POWER SHIELD)

Aug. 7, 1990

PIECE22.DRW

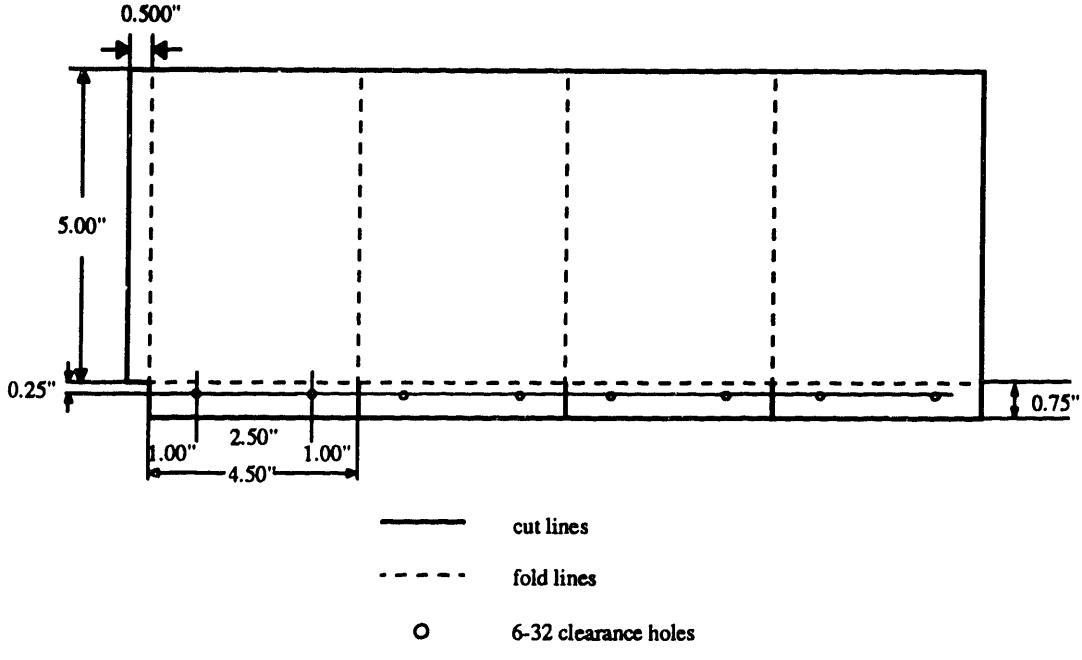


PLASMA REACTOR

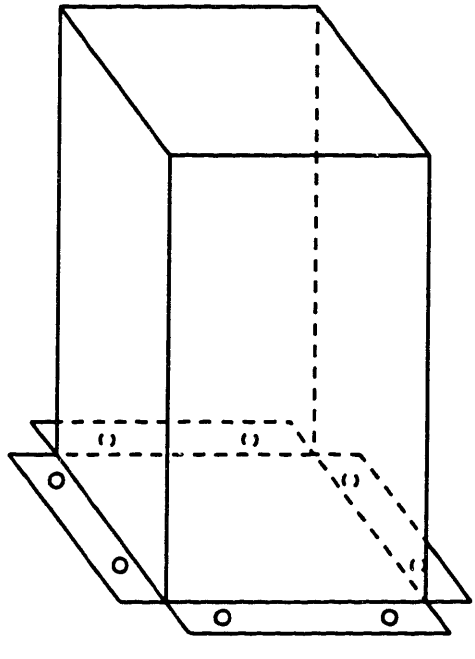
(ALUMINUM CAGE)

Aug. 7, 1990

PIECE23.DRW



3-D drawing of folded aluminum cage



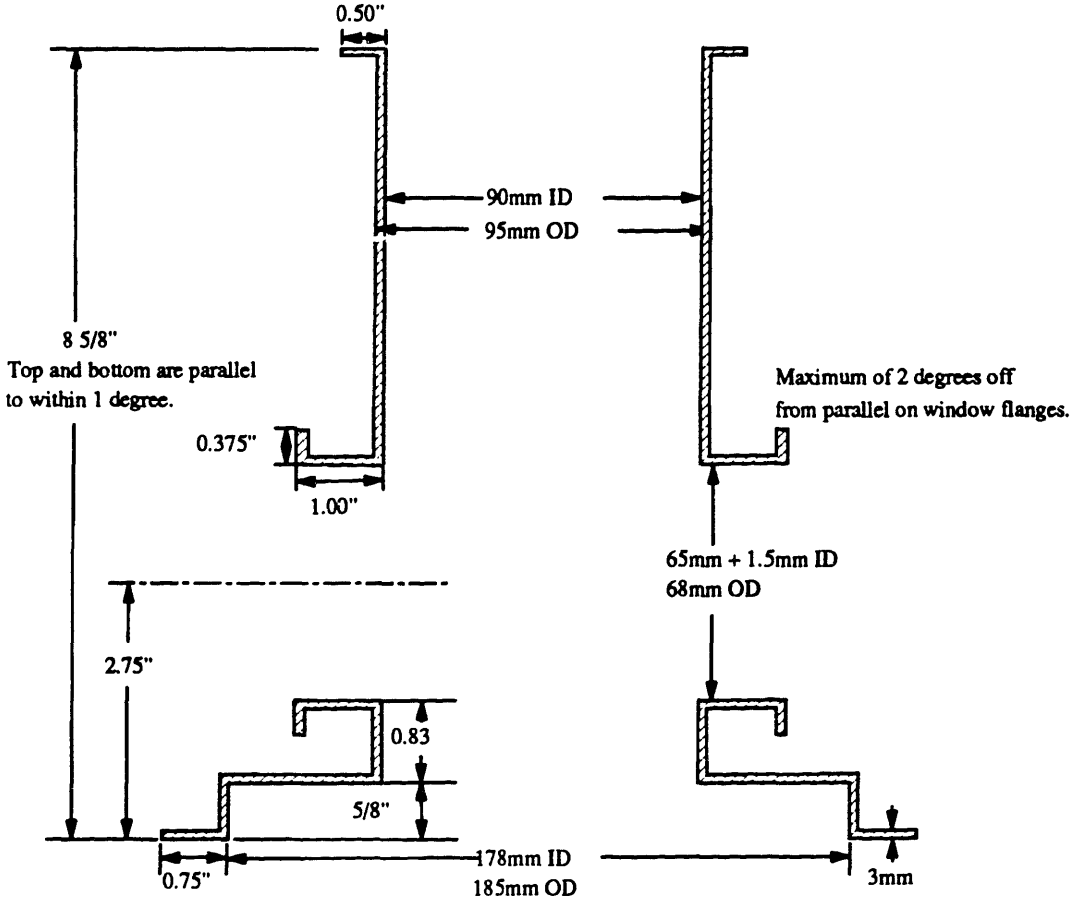
PLASMA REACTOR

(SMALL GLASS CHAMBER)

January 29, 1991

GLASS1.DRW

Cross section of pyrex chamber
Please ground all flange surfaces.



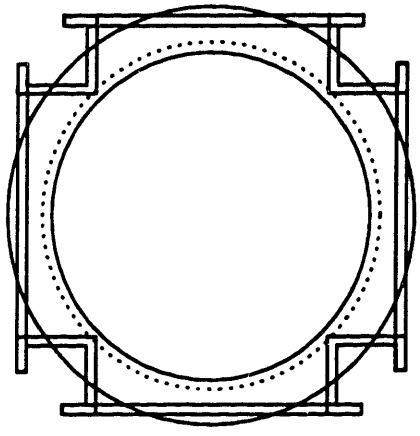
PLASMA REACTOR

(SMALL GLASS CHAMBER)

GLASS2.DRW

Top view of pyrex chamber

Note: Where the four windows cross, the junction should not protrude into the inner diameter of the main chamber, that is 90mm.



PLASMA REACTOR

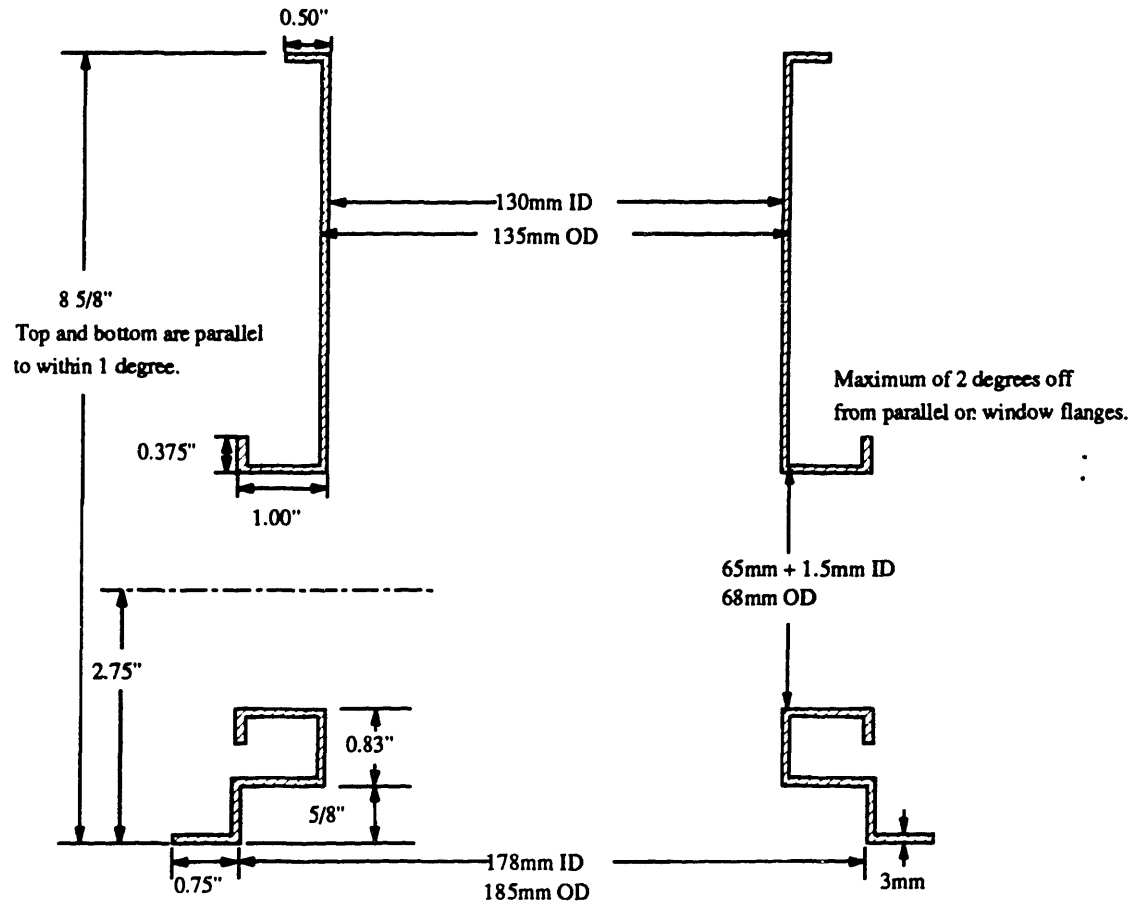
(MEDIUM GLASS CHAMBER)

January 29, 1991

GLASS3.DRW

Cross section of pyrex chamber

Please ground all flange surfaces.



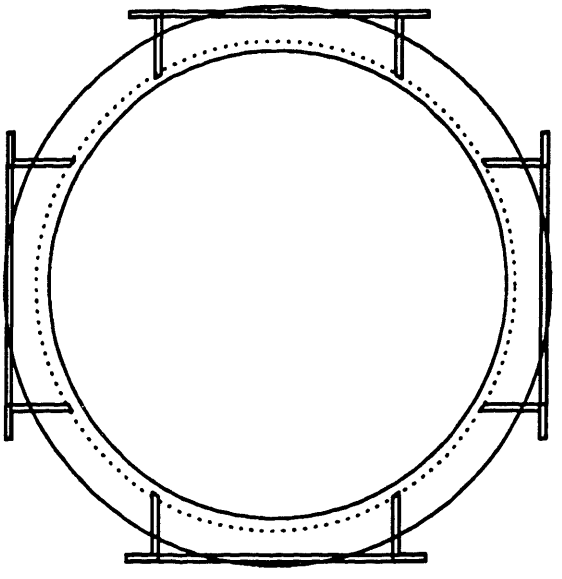
PLASMA REACTOR

(MEDIUM GLASS CHAMBER)

Jan. 29, 1991

GLASS4.DRW

Top view of pyrex chamber



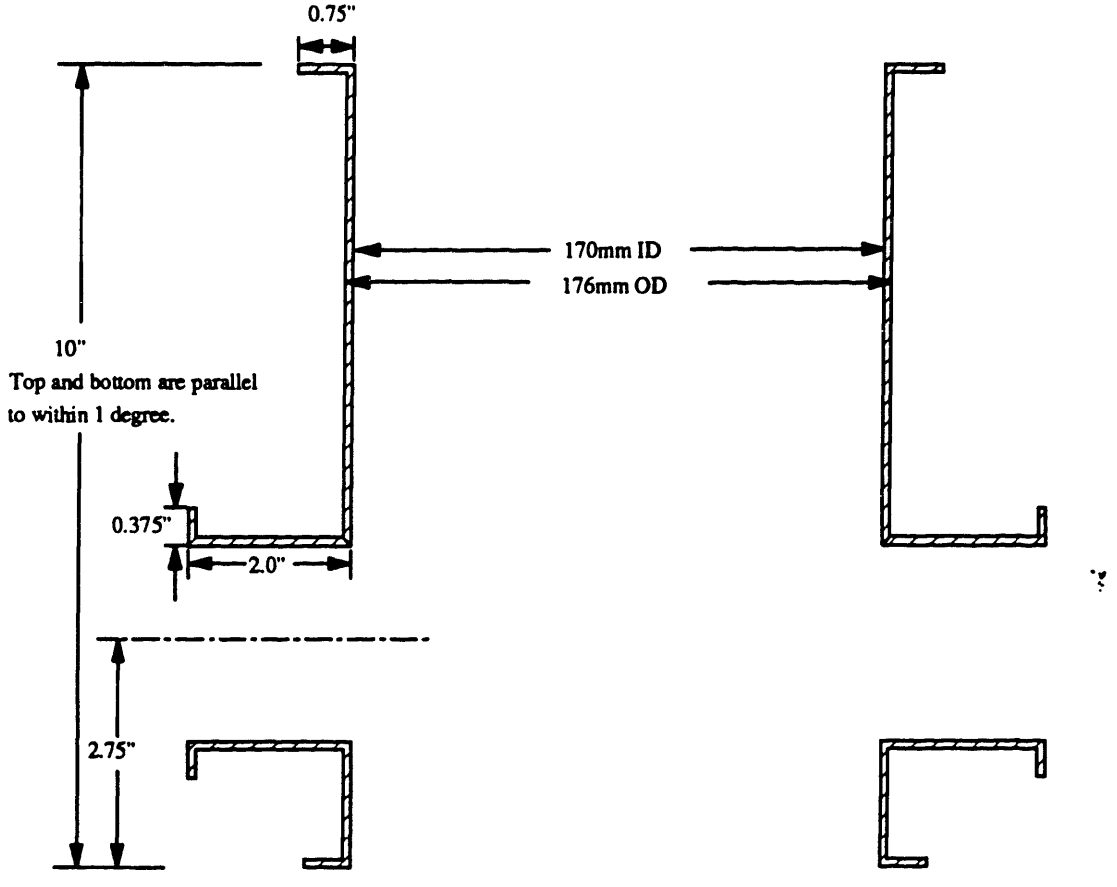
PLASMA REACTOR

(MEDIUM GLASS CHAMBER)

GLASS5.DRW

Cross section of pyrex chamber

Please ground all flange surfaces.

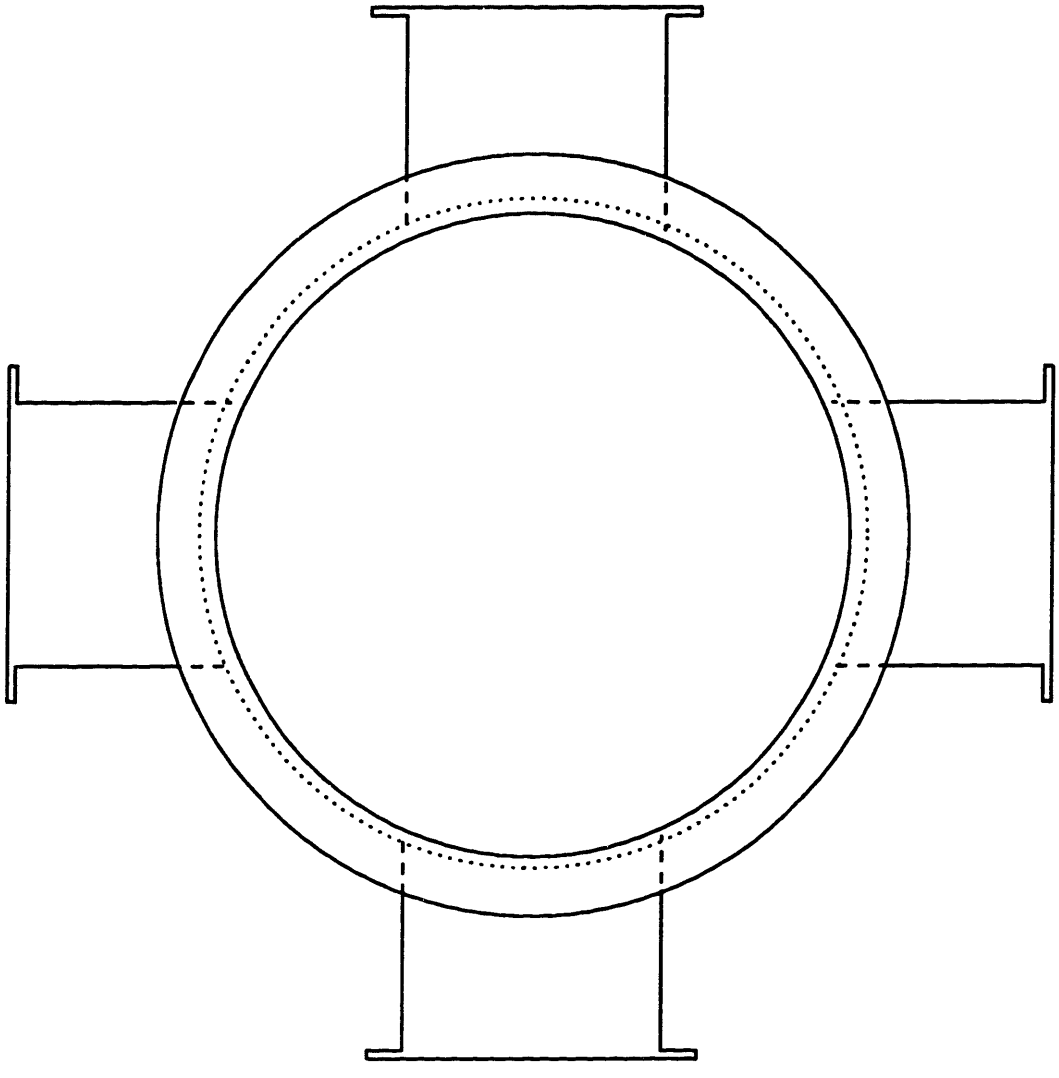


PLASMA REACTOR

(LARGE GLASS CHAMBER)

GLASS6.DRW

Top view of pyrex chamber

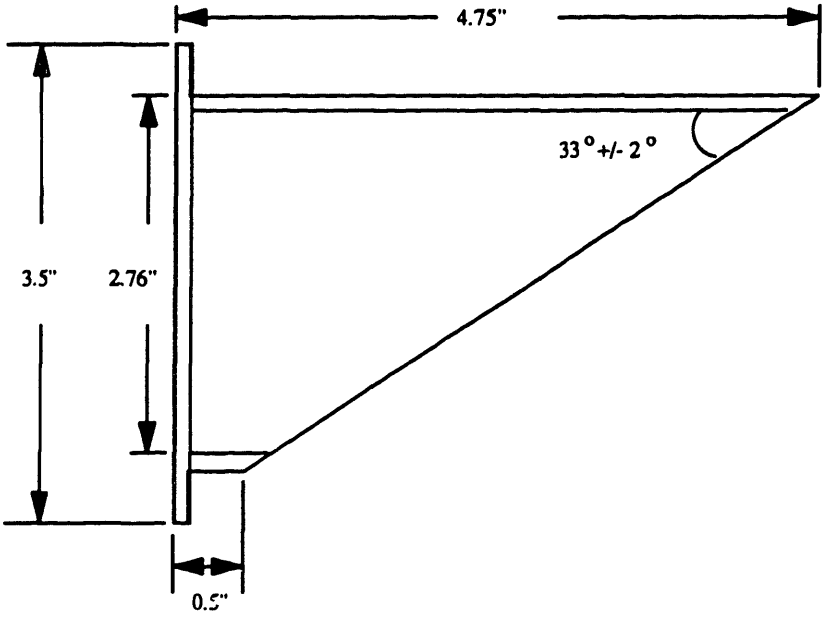


PLASMA REACTOR

(PYREX BREWSTER WINDOW)

GLASS7.DRW

Note: make two of these, ground flange surfaces



PLASMA REACTOR

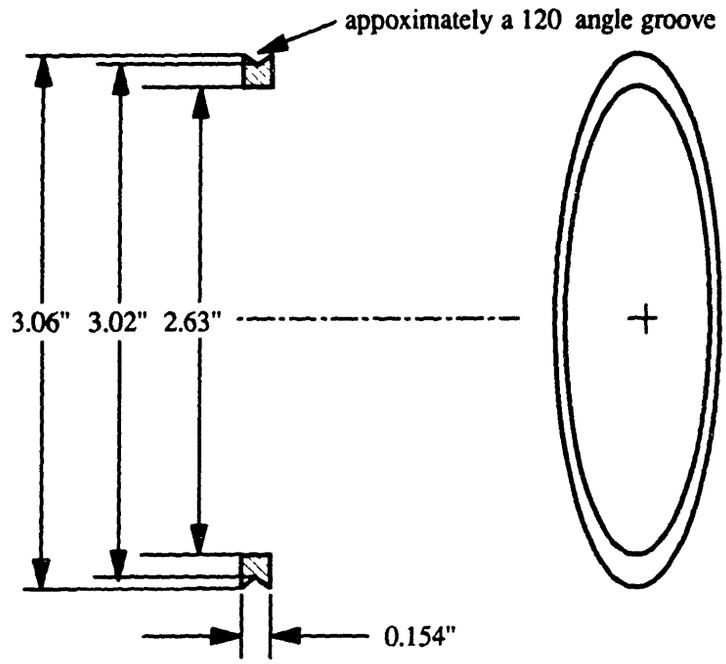
April 1, 1991

(O-RING HOLDER FOR SEAL BETWEEN WINDOWS AND FLANGE)

DELRIN.DRW

Material: DELRIN
Make four

Cross Section



PLASMA REACTOR

Jan. 12, 1988

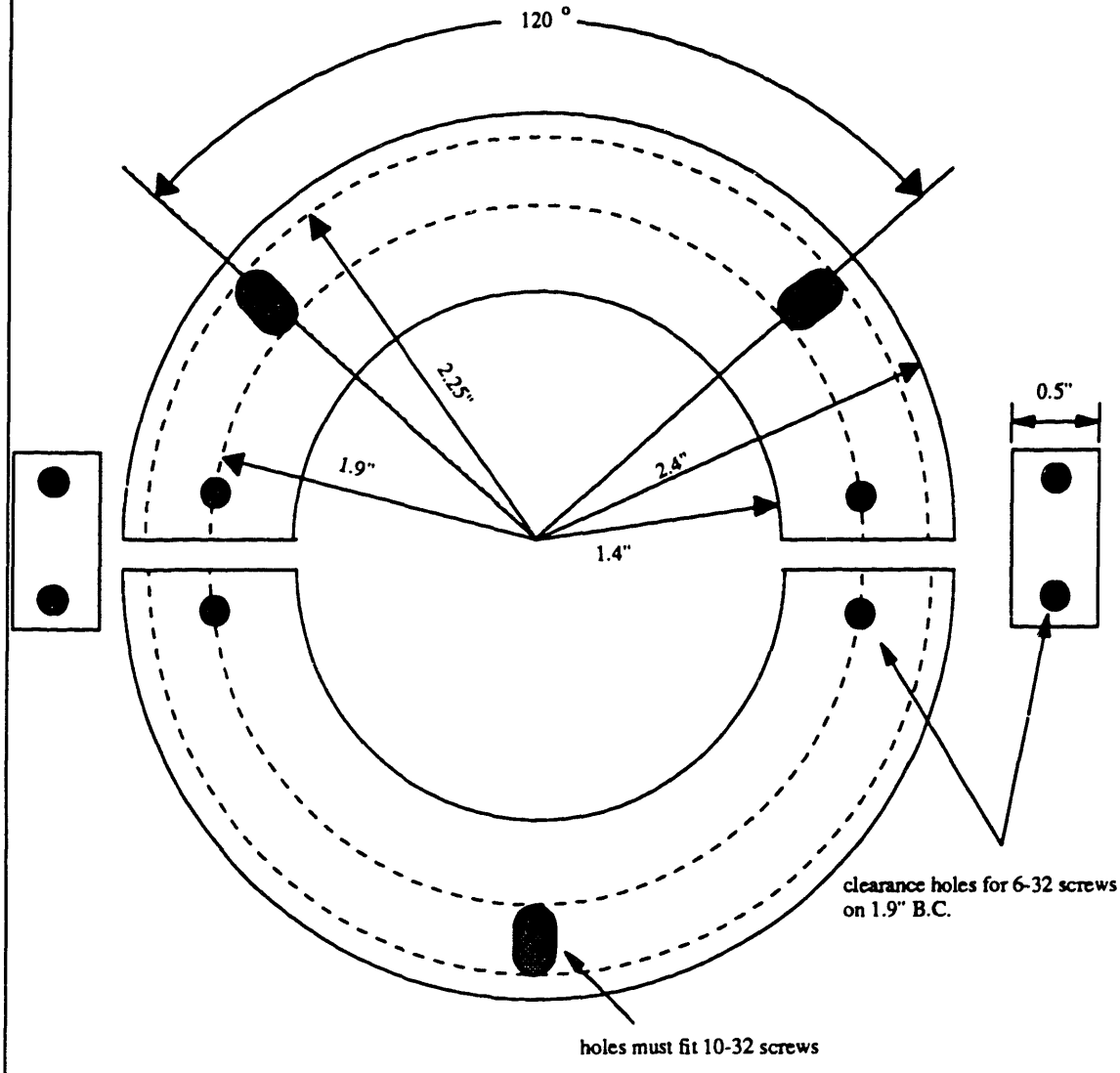
(CLAMPS FOR FLAT AND BREWSTER WINDOWS)

CLAMPS.DRW

Number of sets needed: 8

Material: 1/8" thick plexiglass

Scale: 1" = 1"



SCREWS:

32 of 6-32 5/8" long stainless steel

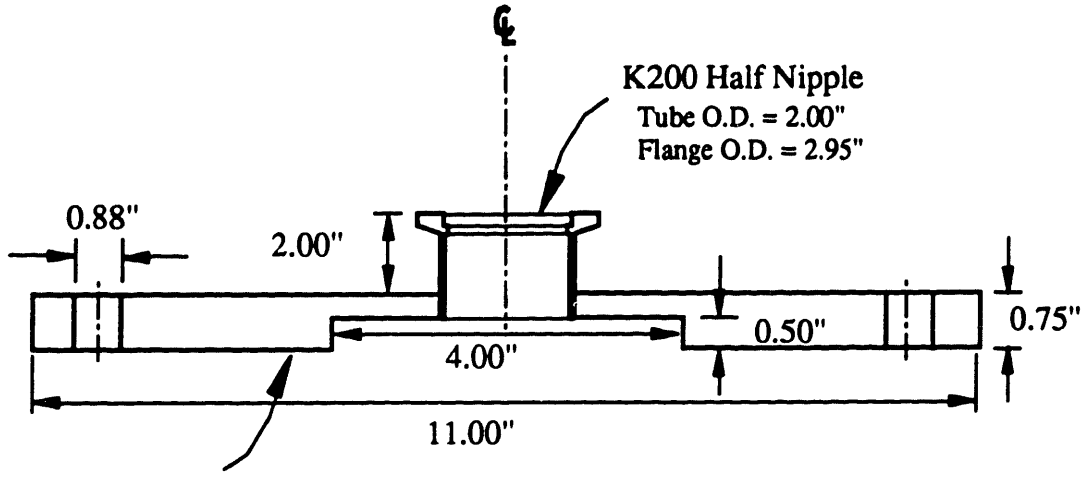
12 of 10-32 1.5" long stainless steel

PLASMA REACTOR

March 23, 1989

(FLANGE FOR CRYOGENICS PUMP)

PIECE26.DRW



ANSI 6" Blank Flange -- Custom Bore

O.D. = 11.00"
Bolt Dia. = 0.88"
Bolt Circle = 9.50"
Thickness = 0.75"
Flat - No O-Ring

Bore Dia. = 4.00"
Bore Depth = 0.50"

* Bore 4.00" Dia. x 0.50" depth on Centerline of Flange
* Bore for and Weld K200 Half Nipple to Flange on Centerline

APPENDIX B

MAGNET DESIGN

Solenoids were constructed to create magnetic fields parallel to the electrodes, with field strength up to 200 Gauss. The solenoids were made by:

Magnecoil Corp.
P. O. Box 3479
Peabody, MA 01960
(508) 535-0008 / (508) 535-1000 / 777-9095
(508) 777-2347 FAX
Location: 11R Newbury St., Danvers, MA 01923
Contacts: Hugh Quinn, Danny Marisseau, John Pryor

The copper tubing were purchased from:

Samll Tube Products
Altoona, PA
(814) 695-4491

This appendix contains the specifications, followed by the construction drawings.

SOLENOID SPECIFICATION

January 1990

1. GENERAL

A total of four coils are needed, two with 56 turns of 0.325" SQ. OD. copper tubing and two with 48 turns of 0.236" SQ. OD. copper tubing. Each coil has two sets of windings which are electrically connected to each other, but with separate water inlets and outlets. The coil characteristics are given in the following tables.

Table B.1: Large Coil Description

Number of coils required	two
Coil resistance (20°C)	
outer half	0.0161 ohm
inner half	0.0133 ohm
Conductor dimensions	
width	0.325"
height	0.325"
coolant hole diameter	0.225"
corner radius	0.060"
copper area, per turn	0.063 $in.^2$
Insulation dimensions	
turn-to-turn	0.028"
winding overwrap	0.020"
coil overwrap	0.030"
Total conductor turns	56
Mean coil length per turn	48.6 in
Max. number of joints	1
Total conductor length per coil	227 ft
Conductor Weight	70 lb
Coil differential coolant pressure	50 psi
Avg. coolant flow per winding	0.4 gal/min
Operating current	300 Amp
Operating voltage	10 V
Operating power	3 kW

Table B.2: Small Coil Description

Number of coils required	two
Coil resistance (20°C)	
outer half	0.0102 ohm
inner half	0.0085 ohm
Conductor dimensions	
width	0.236"
height	0.236"
coolant hole diameter	0.157"
corner radius	0.040"
copper area, per turn	0.035 in^2
Insulation dimensions	
turn-to-turn	0.028"
winding overwrap	0.020"

coil overwrap	0.030"
Total conductor turns	48
Mean coil length per turn	20"
Max. number of joints per winding	0
Total conductor length per coil	80 ft
Conductor weight	15 lb
Coil differential coolant pressure	50 psi
Avg. coolant flow per winding	0.27 gal/min
Operating current	300 Amp
Operating voltage	6.5 V
Operating power	1.9 kW

2. TECHNICAL REQUIREMENTS

2.1 Material

The windings shall be fabricated from fully annealed high conductivity copper. The conductors will be square and will have the following dimensions:

(1) 0.325" ± 0.004" square with a 0.060" corner radius, and a round center hole having a 0.225" diameter.

(2) 0.236" ± 0.004" square with a 0.040" corner radius, and a round center hole having a 0.157" diameter.

2.2 Coil Construction

Each coil shall consist of two sets of windings as shown in the drawing. The windings are wound axially rather than in the usual radial direction, producing a winding of larger diameter surrounding the other. Turn-to-turn insulation shall be provided by a one layer, half lap wrap of 0.007" thick, amino silane treated, fiberglass tape. Dacron sleeving of 0.014" wall is an acceptable alternative. An overwrap consisting of one layer, half lap wrap of 0.010" thick amino silane treated fiberglass tape shall be applied to each set of windings. A half lap wrap of with 0.015" (0.010" is also acceptable) thick amino silane treated fiberglass tape shall then be applied to the entire coil.

Vacuum impregnation of the coil shall be done. A vacuum level of less than 300 microns shall be achieved prior to epoxy impregnation. Individual windings need not be separately impregnated. All cast volumes of epoxy shall be filled with glass roving, chopped fibers, tape, cloth or similar material such that there be no unfilled volume of epoxy. Epoxy shall be clear and the surface unpainted to permit visual inspection.

3. COIL FABRICATION REQUIREMENTS

3.1 Turn Forming

The individual coil turns can be formed either before or after application of turn-to-turn insulation. Keystoning shall be controlled by proper coil forming procedures. Winding in accordance with drawing is expected to result in keystoning of acceptable levels, without requiring subsequent removal. Conductor surfaces shall be free of all burrs or slivers which could penetrate or damage the insulation.

3.2 Joints

Joints between bar lengths shall be formed by ferrule and counterbore as detailed in drawings and shall be silver soldered in accordance with the American Welding Society B CUP 5 procedure. Ferrules shall be fabricated from CDA alloy 102 stock. Alternate joint designs which have been proven to have at least equivalent mechanical and electrical properties will be allowed, subject to the prior written approval of MIT.

All joints shall be formed as specified above and shall be completed in a fashion to assure that any nearby insulated areas are not contaminated. Technicians shall be required to make six consecutive satisfactory sample joints using the specified materials and procedures anticipated for this coil prior to assignment to this task. The internal joint position shall be chosen to lie outside the cross-over region to avoid forming distortion or joint damage.

Given the conductor length and the conductor lengths required per winding, it appears that only the large pancake will require at most one internal joint.

3.3 Surface Cleaning

All surfaces of the internal components of the coil shall be thoroughly cleaned prior to impregnation and shall be protected until impregnation is begun to prevent any contamination from substances such as dirt, oil, grease, or perspiration in order to promote a good bond between the epoxy and the coil components.

The copper conductor shall be cleaned with a chemical and a non-metallic abrasive agent to insure the removal of all copper burrs, grease, and oil. The cleaned coil shall be kept free from contamination.

All surfaces of the glass tape shall be kept free of dirt, oil, moisture, or perspiration. If any of these materials become contaminated by the above prior to impregnation of the coil, it shall be solvent washed and dried before it is used in the coil assembly.

3.4 Application of Insulation and Filler

All insulation and any filler material shall be applied to the coil assembly in a area free of metallic dust, welding fumes, or other sources of contaminants. No resin-rich areas shall be present in the coil. The presence of any of the above inside the structure of a finished coil will lead to rejection of the coil.

4. TEST REQUIREMENTS

The following tests shall be performed on each coil at appropriate points in time during fabrication. MIT shall be notified at least one week before the application of tests in Section 4.3 to Section 4.8 so that these tests may be witnessed by a designated representative of MIT.

4.1 Hydraulic Joints

Each joint in all of the hydraulic circuits shall be leak tested. The preferred test incorporates a halogen spectrometer in a "sniffer mode." To assure that the minimum acceptable leak is not plugged with water, each coolant passage must be pressurized with a suitable low molecular weight gas (nitrogen, helium, etc.) to 150 ± 50 psig before the leak check is done. A standard leak shall be used to demonstrate adequate sensitivity of the halogen leak detector. Alternate procedures shall be subject to the prior written approval of MIT.

4.2 Electrical Joints

A current of no less than 100 Amps and no more than 200 Amps shall be passed through each electrical joint. The voltages between points on the conductor four inches apart and including the joint shall be measured. The voltages across a four inch length of joint-free conductor shall be measured. The joint shall be rejected if it produces a voltage more than 5% higher than that found across the joint-free conductor.

4.3 Cleanliness Test for Coolant Passages

After impregnation and after cleaning and flushing the coolant passages, the following test shall be made.

Each coolant passage shall be connected to a clear potable water supply at a pressure of at least 30 psi. A filter of 50 to 150 micron size shall be installed on both inlet and outlet sides of the coil circuits. The temperature of the water used shall be high enough to prevent the condensation of moisture on the coil surfaces.

After at least 100 gallons of water have passed through each water circuit, there shall be no trace of dirt or grease deposited on the outlet filter.

4.4 Water Flow

The water flow through each individual cooling circuit in a winding shall be measure using a static pressure difference between the circuit inlet and outlet of 100 ± 10 psig. The temperature of the water used shall be high enough to prevent condensation of moisture on the coil surfaces. The differences in flow through each coolant circuit shall be no greater than can be accounted for by the differences in individual circuit length and/or bend radii.

4.5 Water Pressure Application Before Electrical Tests

After impregnation, all water circuits in each winding shall be subjected to 300 ± 10 psig water pressure. The temperature of the water used shall be lower than the ambient temperature and greater than the dew point temperature of the air surrounding the coil. The pressure shall be applied to each of the water circuits through the inlet tube to that circuit with the outlet tubes plugged after all air has been removed. The circuits shall then be isolated from the pressure source. The initial pressure in the isolated circuits shall be read and recorded. After a time of no less than one hour, the pressure shall be read and recorded. If this final reading is lower than the initial reading, the results will be interpreted as indicating a leak in the water circuit and the coil shall be subject to rejection. This test shall be finished no longer than one hour before performing

any part of the electrical tests specified in Section 4.6 and 4.7.

4.6 Induced Voltage Test

An induced voltage resulting in a turn-to-turn voltage difference greater than 10 volts rms shall be applied to the coil to detect any shorts between turns. The voltage difference between the free ends of the coil will be approximately ten times the number of turns in the coil and therefore these ends shall be insulated or otherwise protected to prevent shorting.

The frequency of the induced voltage shall be between 60 and 3000 Hertz. The induced voltage shall be applied for at least one minute. The excitation voltage shall be raised and lowered gradually to avoid damaging voltage transients. The exciting current of the test equipment shall be recorded. For the detection of shorted turns, a volt meter connected to a pickup coil shall be used. The Contractor shall demonstrate that the equipment can detect a 10% short in a single turn in the subject coil.

4.7 DC Voltage Test

4.7.1 Megger, Ground Insulation Test

The windings shall be placed between two copper shields. A 1,000 volt Dc megger test shall be made between the coil conductor and the copper shield. The resistance of the insulation so determined shall be no less than one megohm to be acceptable. This test shall be performed before the test of Section 4.7.2.

4.7.2 Ground Insulation Measurements

2,500 volts DC shall be applied between the conductor and the copper shield. Full voltage shall be held for one minute. The test voltage shall be raised and lowered in steps of 200 volts at a rate not to exceed 200 volts per second. After each step up or down is made, the voltage shall remain unchanged until the leakage current has been stabilized and recorded.

4.8 Resistance Check

After impregnation, the DC resistance of each coil shall be measured and recorded with an absolute error no larger than 0.5%. The temperature of the conductor at the time of this measurement shall also be measured and recorded with an error no larger than 1°F.

4.9 Coil Insulation

Each finished coil shall be inspected carefully and there shall be no cracks in the resin and there shall be no voids, soft spots, extraneous inclusions, or areas of dry, unimpregnated fiberglass.

4.10 Dimensional Checks

The actual coil dimensions shall be measured and recorded on a copy of the coil drawing.

4.11 Test Records

The records taken during each test specified herein, and of all additional tests specified by the Contractor, shall be

submitted to MIT for examination and acceptance prior to shipment of the coils.

5. PACKING AND SHIPPING

Precautions shall be taken by the Contractor to prevent damage to the coils during transportation. All tubing, nozzles and other openings shall be capped or otherwise protected immediately after shop cleaning to prevent the intrusion of dirt or other foreign matter. All water cooling passages shall be protected against freezing by flushing with a sufficient amount of ethyl alcohol if shipment will be subject to freezing temperatures.

6. QUALITY REQUIREMENTS

The Contractor shall maintain an effective and timely quality program that shall include the following activities as a minimum.

6.1 Material Control

The Contractor shall maintain a system of positive control of material from procurement through fabrication. The program shall assure that materials to be used in fabrication or processing of products conform to the applicable physical, chemical, and other technical requirements.

6.2 Drawing and Change Control

The Contractor shall maintain a system which will assure that the latest applicable drawings, technical requirements, and contract change information are used. The system shall also assure that obsolete information is removed from all points of issue and use.

6.3 Record System

The Contractor shall maintain an effective and orderly system of records to provide objective evidence of product quality. Records shall be retained or forwarded to MIT as required.

6.4 Measuring and Testing Equipment

The Contractor shall provide and maintain gauges and other measuring and testing equipment necessary to assure that all supplies conform to technical requirements. A system shall be maintained to provide periodic calibration of these devices using certified standards.

6.5 Subcontract Control

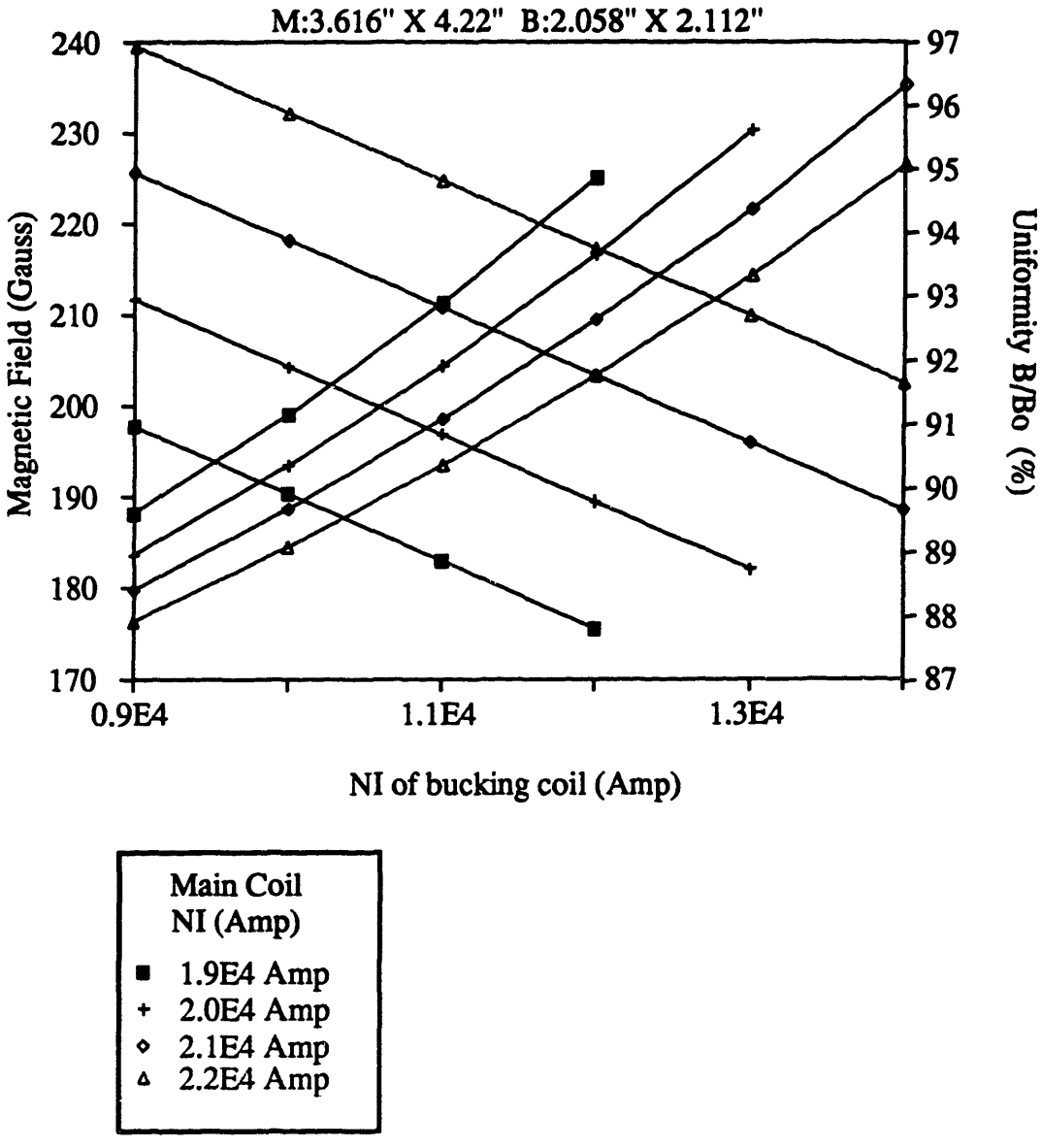
The Contractor shall be responsible for assuring that all supplies and services subcontracted conform to specification requirements. The Contractor shall exercise control over subcontractors to the extent necessary to assure acceptable quality.

Table B.3: Parts List for Solenoids

Part Number	Amt. Required	Description
1		copper conductor 0.325" SQ. OD. with 0.225" Dia. hole
2		copper conductor 0.236" SQ. OD. with 0.157" Dia. hole
3		0.007" thick fiberglass tape over copper conductor
4		0.010" thick fiberglass tape over each winding
5		0.015" thick fiberglass tape over each coil
6	4	connector plate for 0.325" SQ. OD. tubing
7	4	connector plate for 0.236" SQ. OD. tubing
8	8	barbed fitting for 0.325" SQ. OD. tubing, Imperial Eastman KA06-06RL
9	8	barbed fitting for 0.236" SQ. OD. tubing, Imperial Eastman KA04-04RL

Nov. 24, 1989

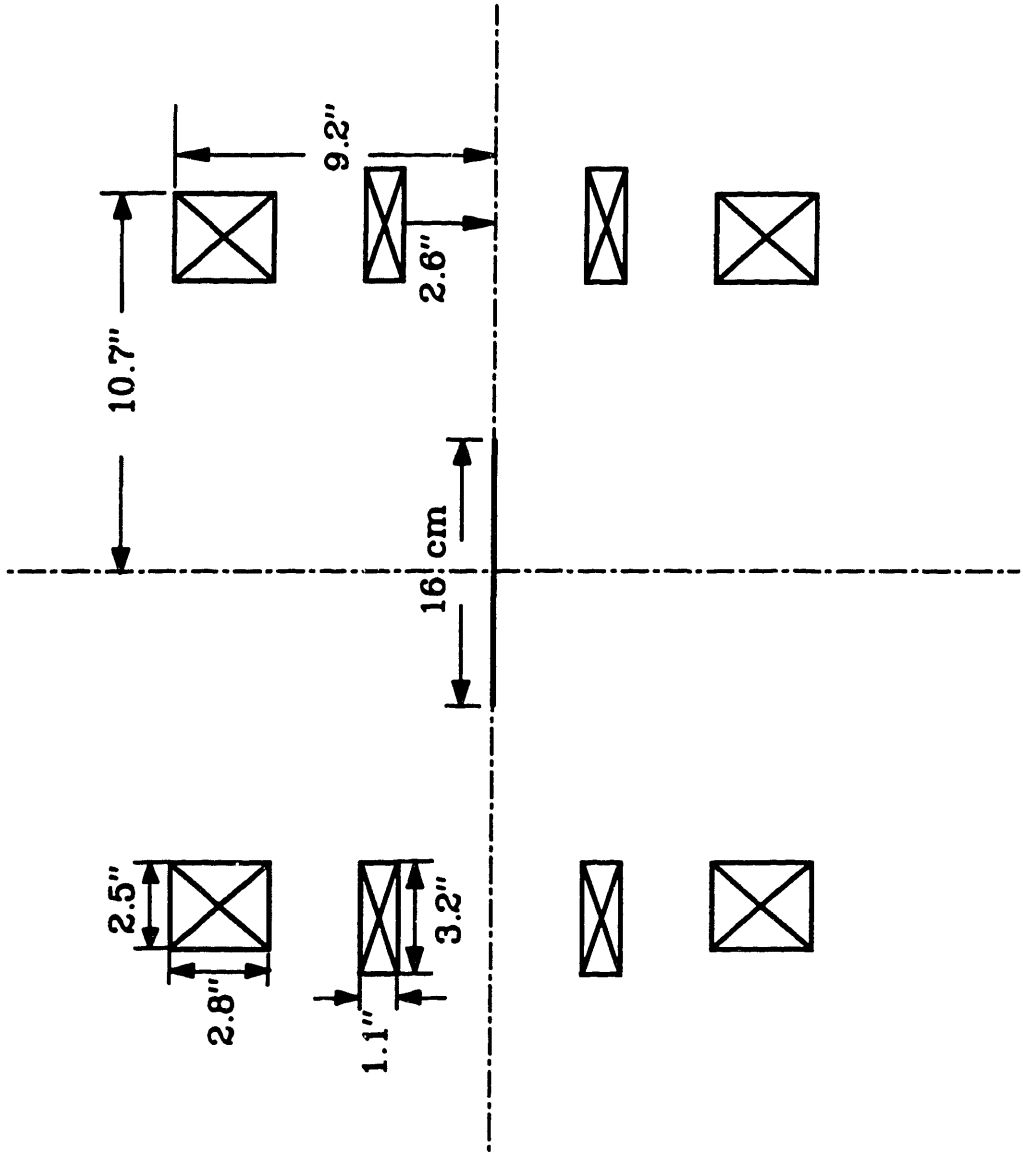
MAGNETIC FIELD AT VARIOUS CURRENTS



MAGNET11.DRW

SOLENOIDS

(SOLENOID CONFIGURATION) MAGNET1.DRW

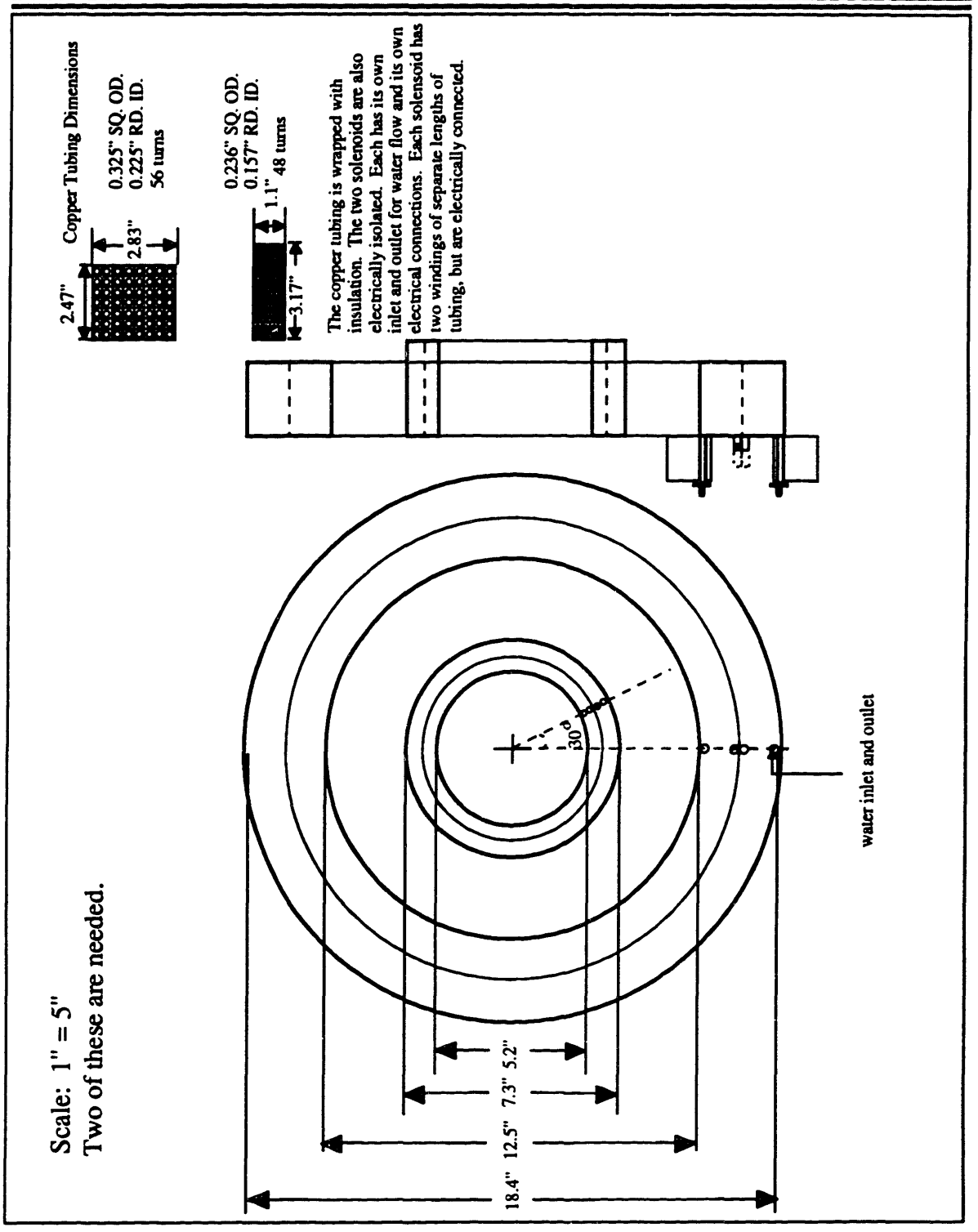


SOLENOIDS

(SOLENOID CONFIGURATION)

December 8, 1989

MAGNET2.DRW



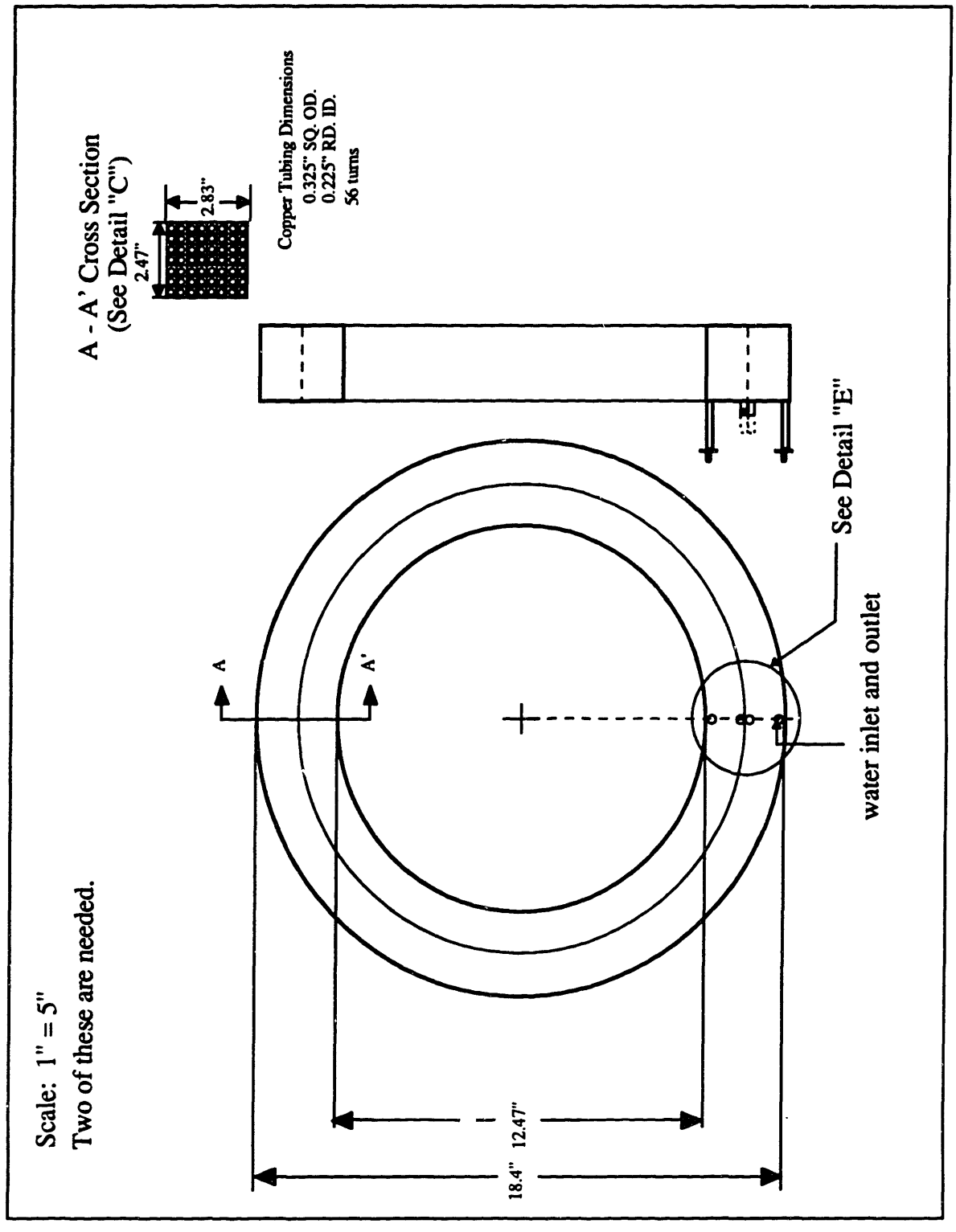
Scale: 1" = 5"

Two of these are needed.

SOLENOIDS

(LARGE COIL)

MAGNET3A.DRW

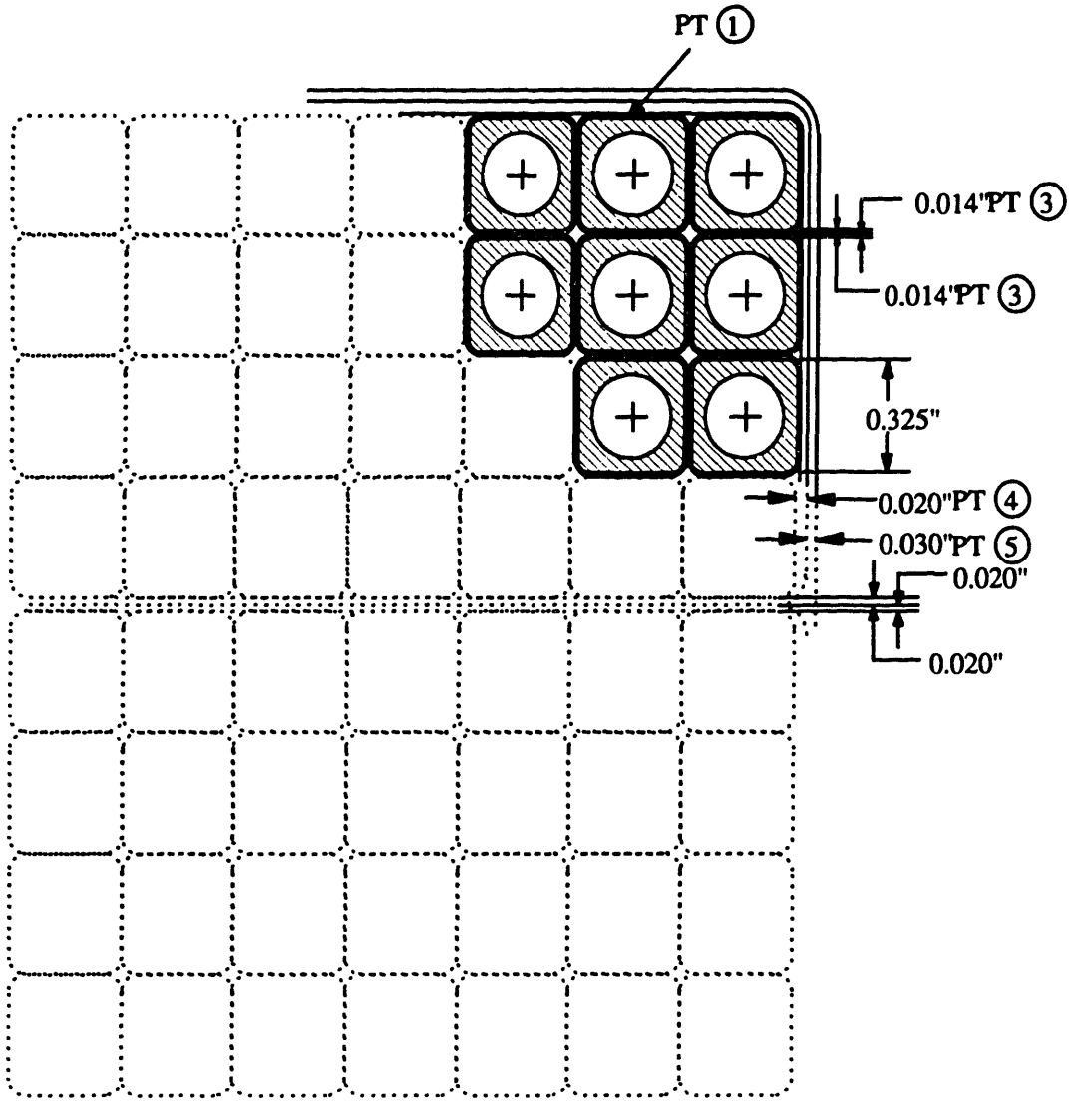


SOLENOIDS

(LARGE COIL - DETAIL "C")

MAGNET4A.DRW

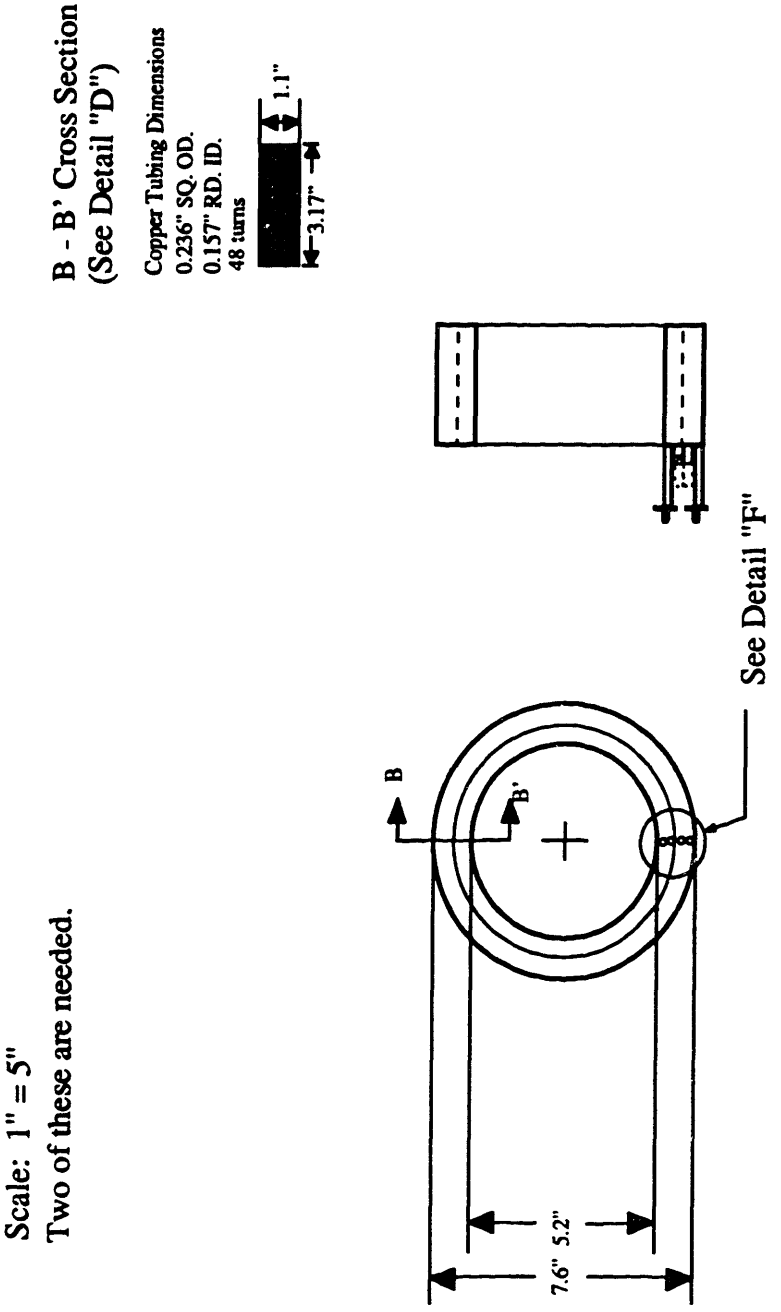
Note: Voids caused by crossover to next layer should be filled with a filler material (e.g. G-10 or chopped fibers).



SOLENOIDS

(SMALL COIL)

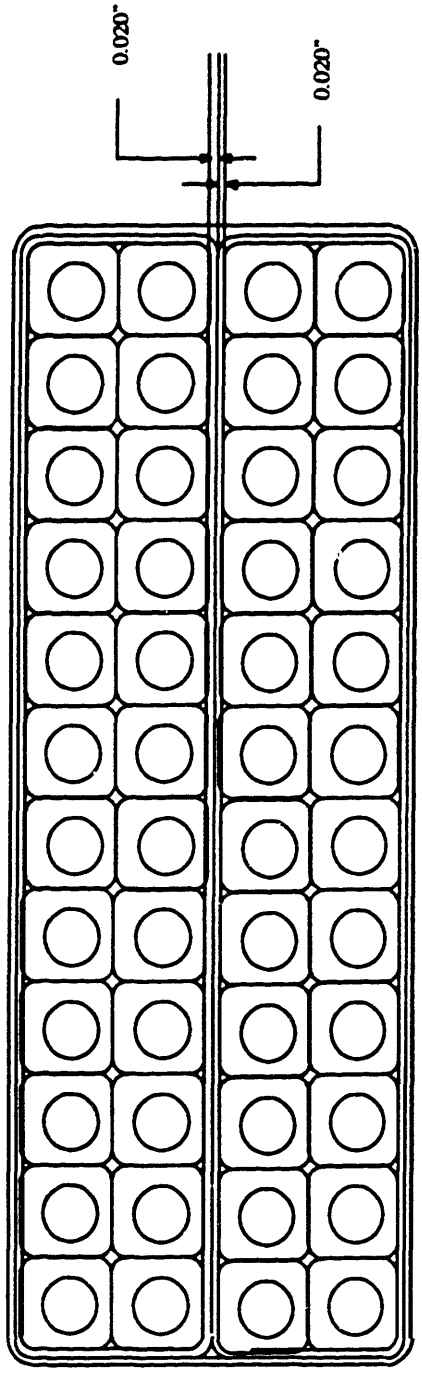
MAGNET3B.DRW

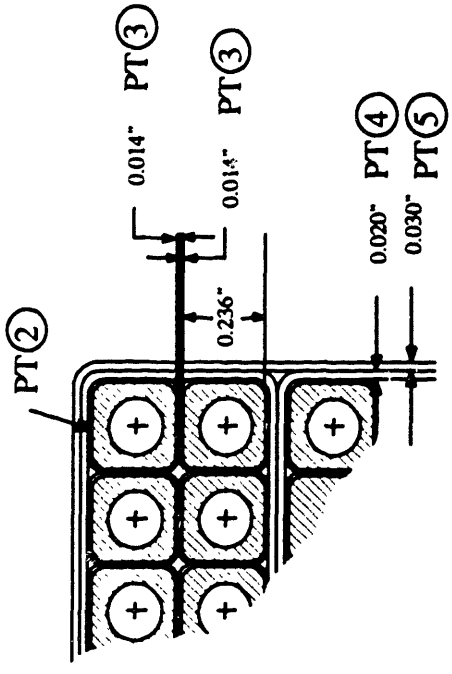


MAGNET4.DRW

(SMALL COIL - DETAIL "D")

MAGNET4B.DRW



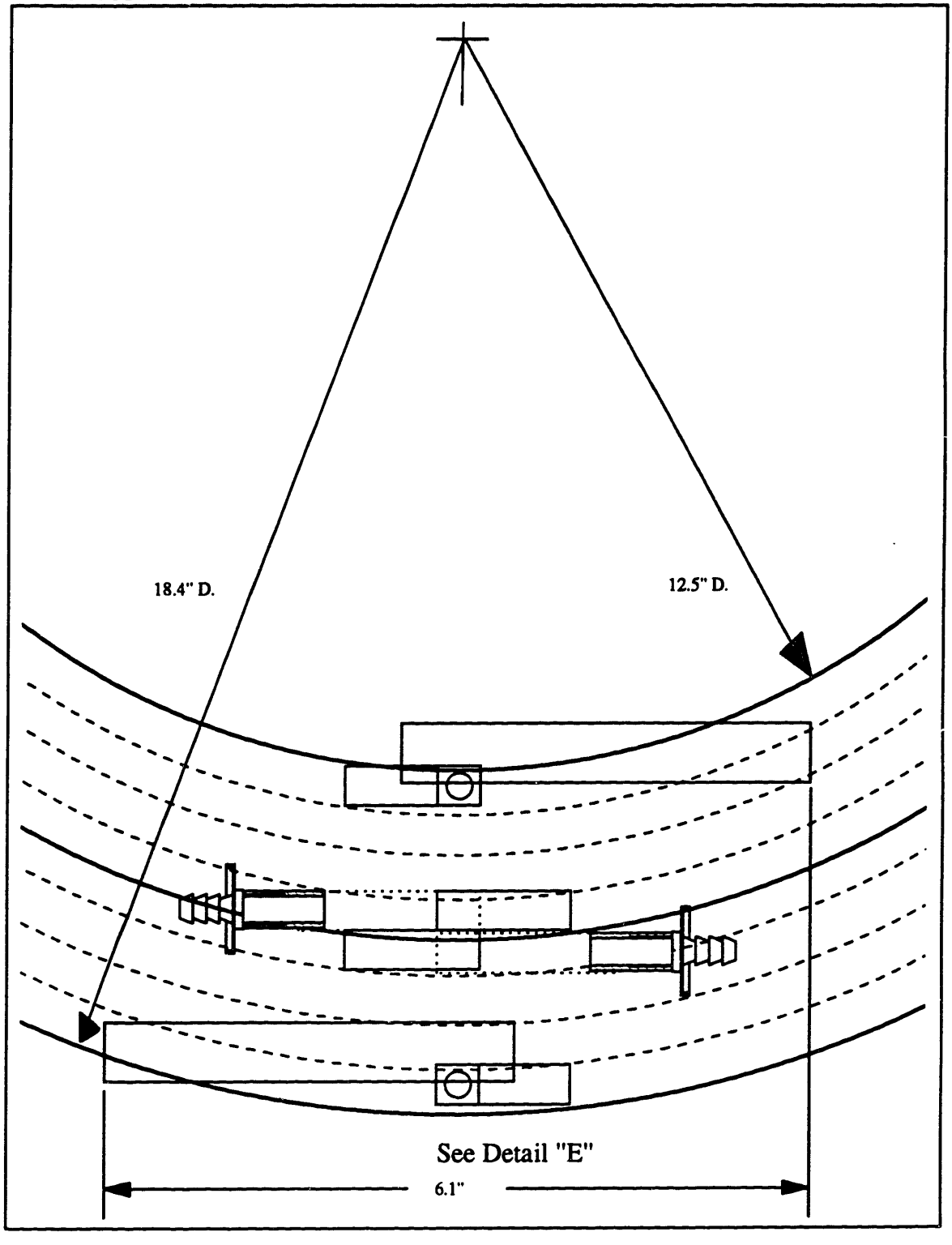


Note: Voids caused by crossovers to next layer should be filled with a filler material (e.g. G-10 or chopped glass fibers)

SOLENOIDS

(LARGE COIL)

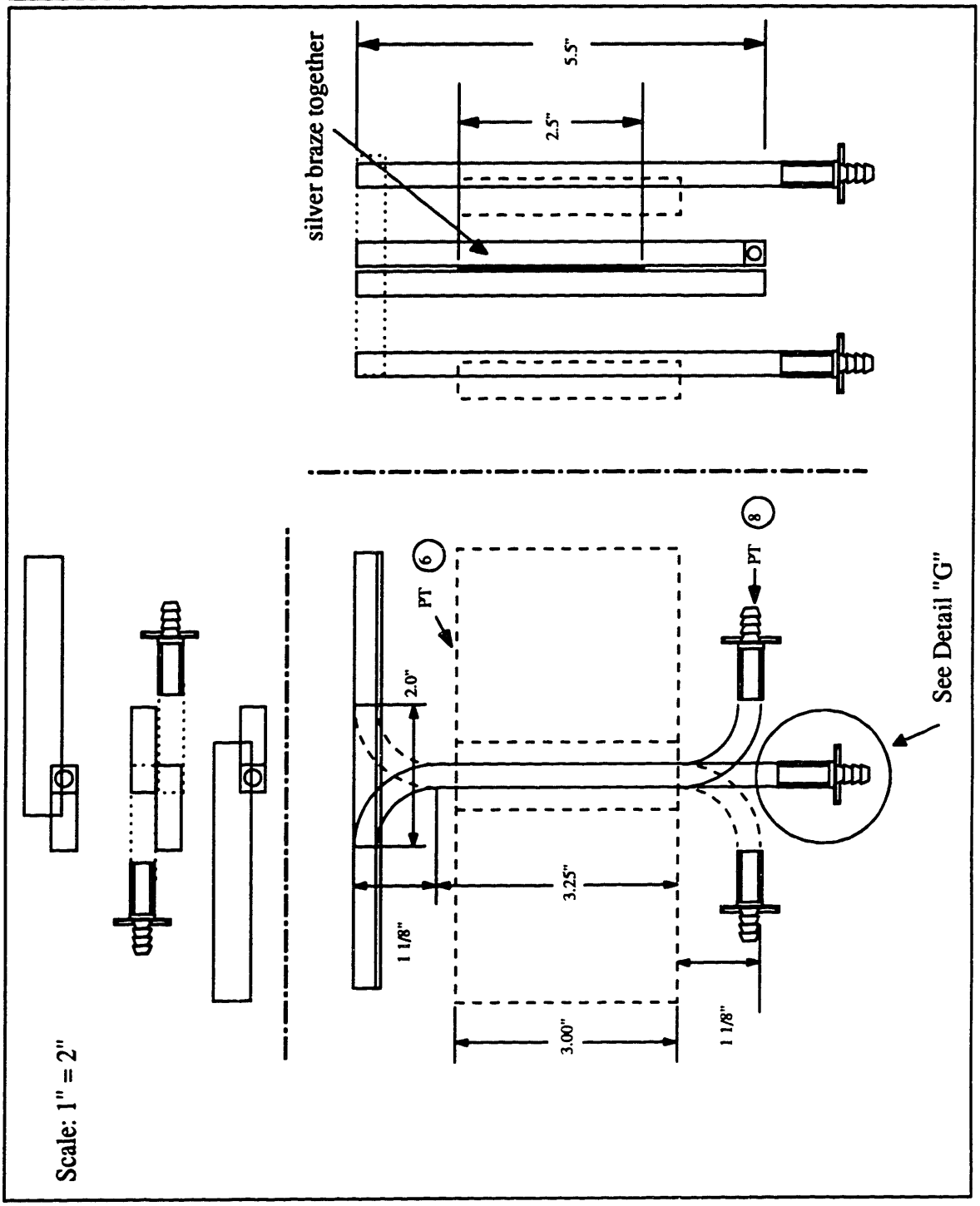
MAGNET6A.DRW



SOLENOIDS

(LARGE COIL - DETAIL "E")

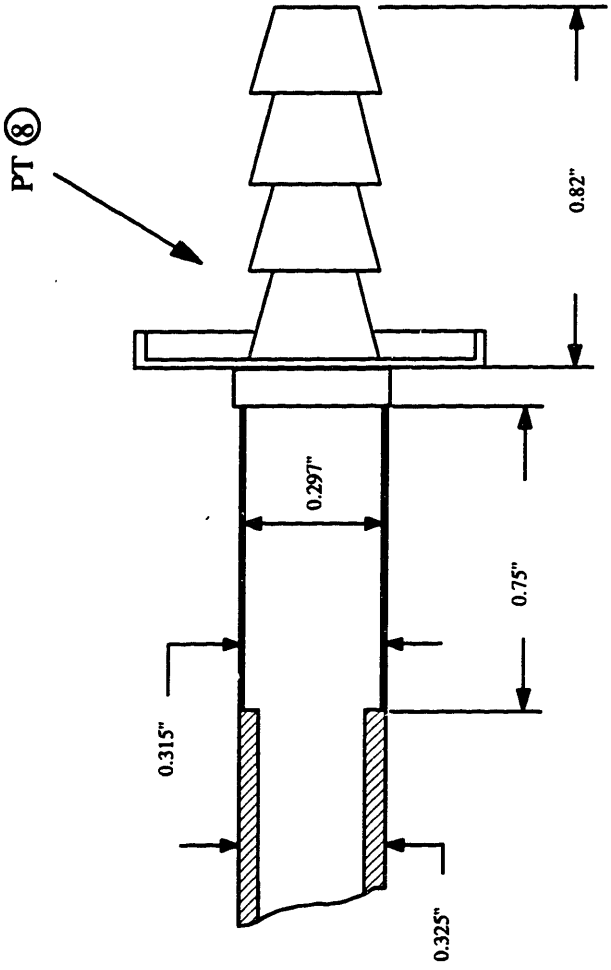
MAGNET7A.DRW



SOLENOIDS

(LARGE COIL - DETAIL "G")

MAGNET8A.DRW

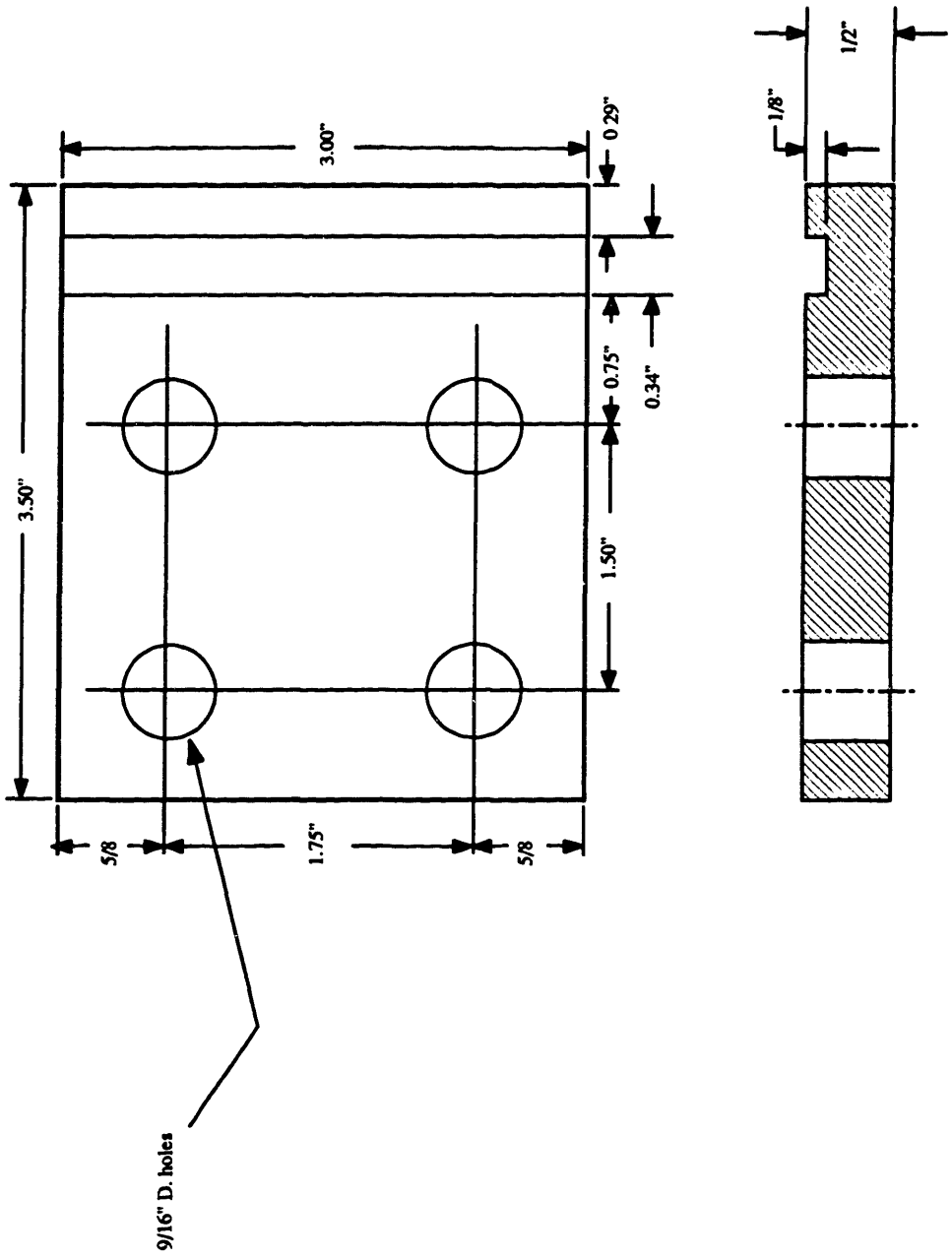


Scale: 1" = 0.4"

SOLENOIDS

(LARGE COIL - PART 6)

MAGNET5A.DRW

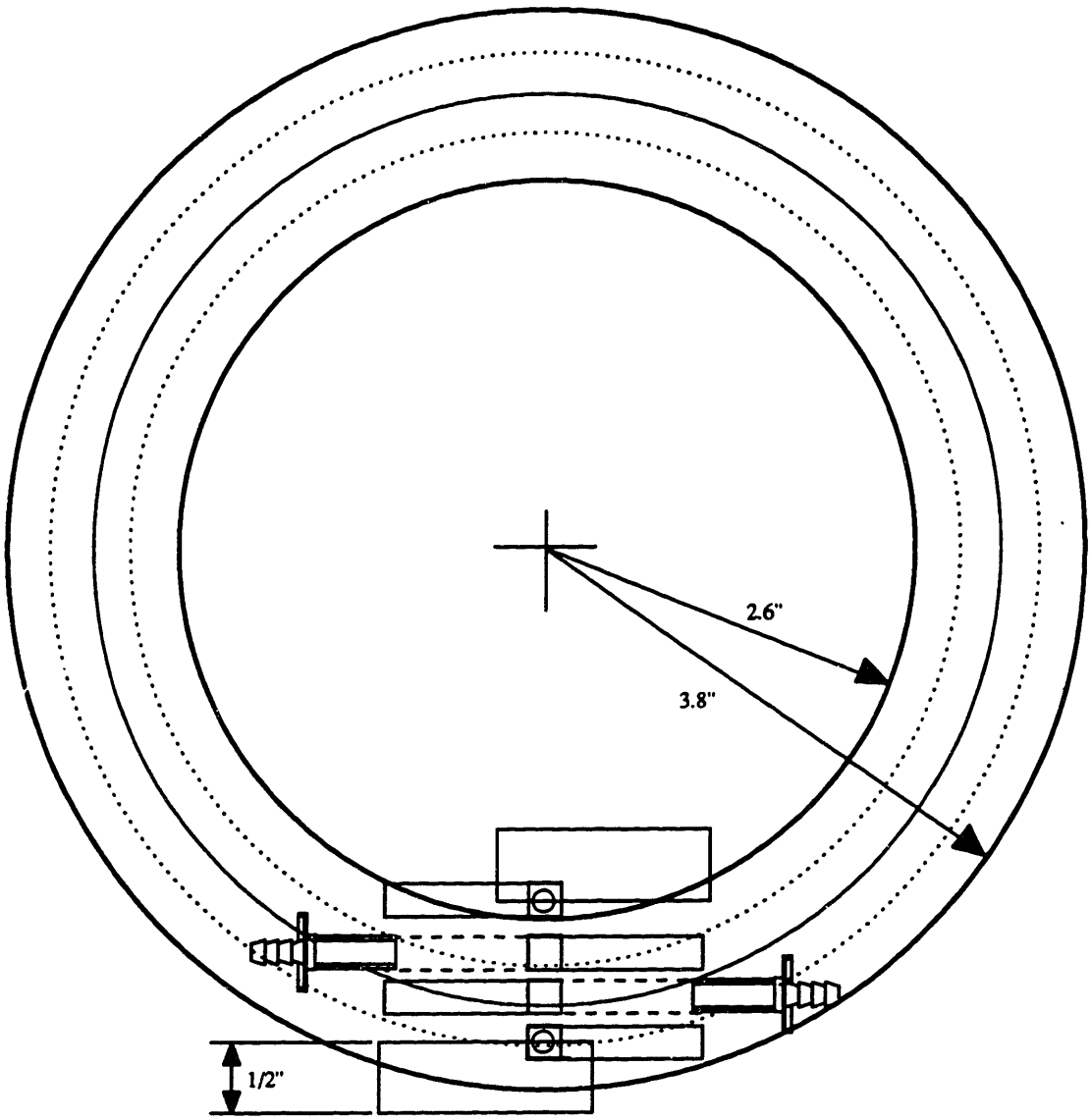


Scale: 1" = 1"
Make Four

SOLENOIDS

(SMALL COIL)

MAGNET6B.DRW

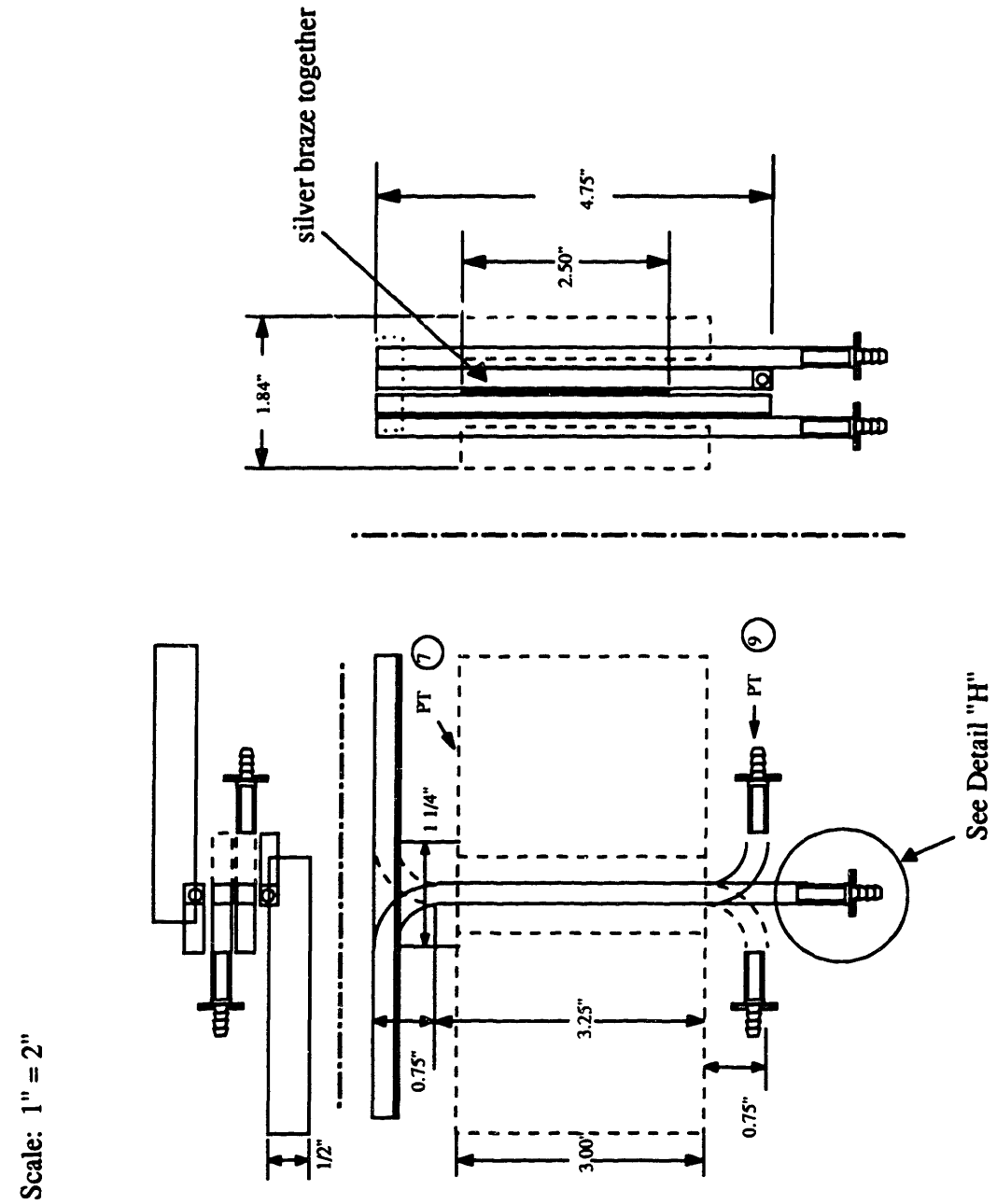


See Detail "F"

SOLENOIDS

(SMALL COIL - DETAIL "F")

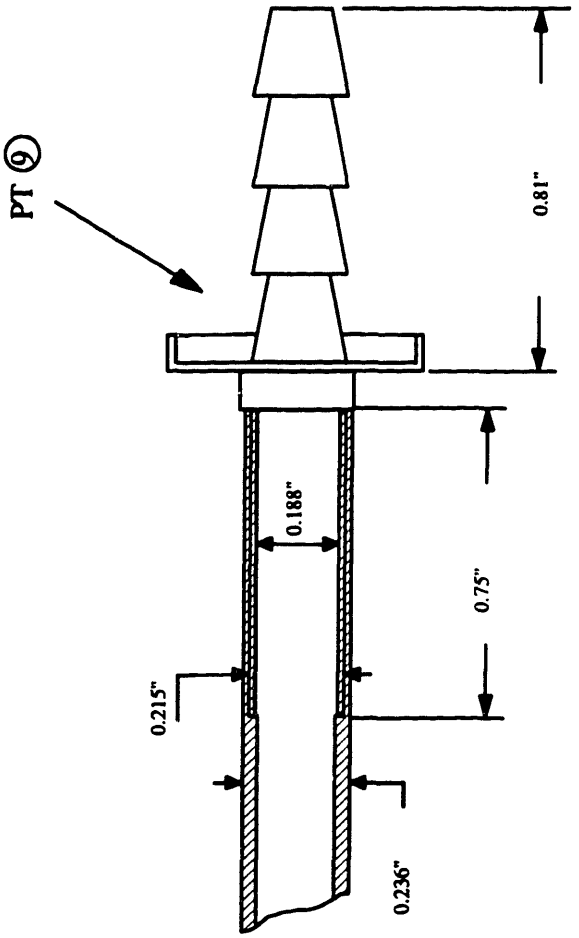
MAGNET7B.DRW



SOLENOIDS

(SMALL COIL - DETAIL "H")

MAGNET8B.DRW

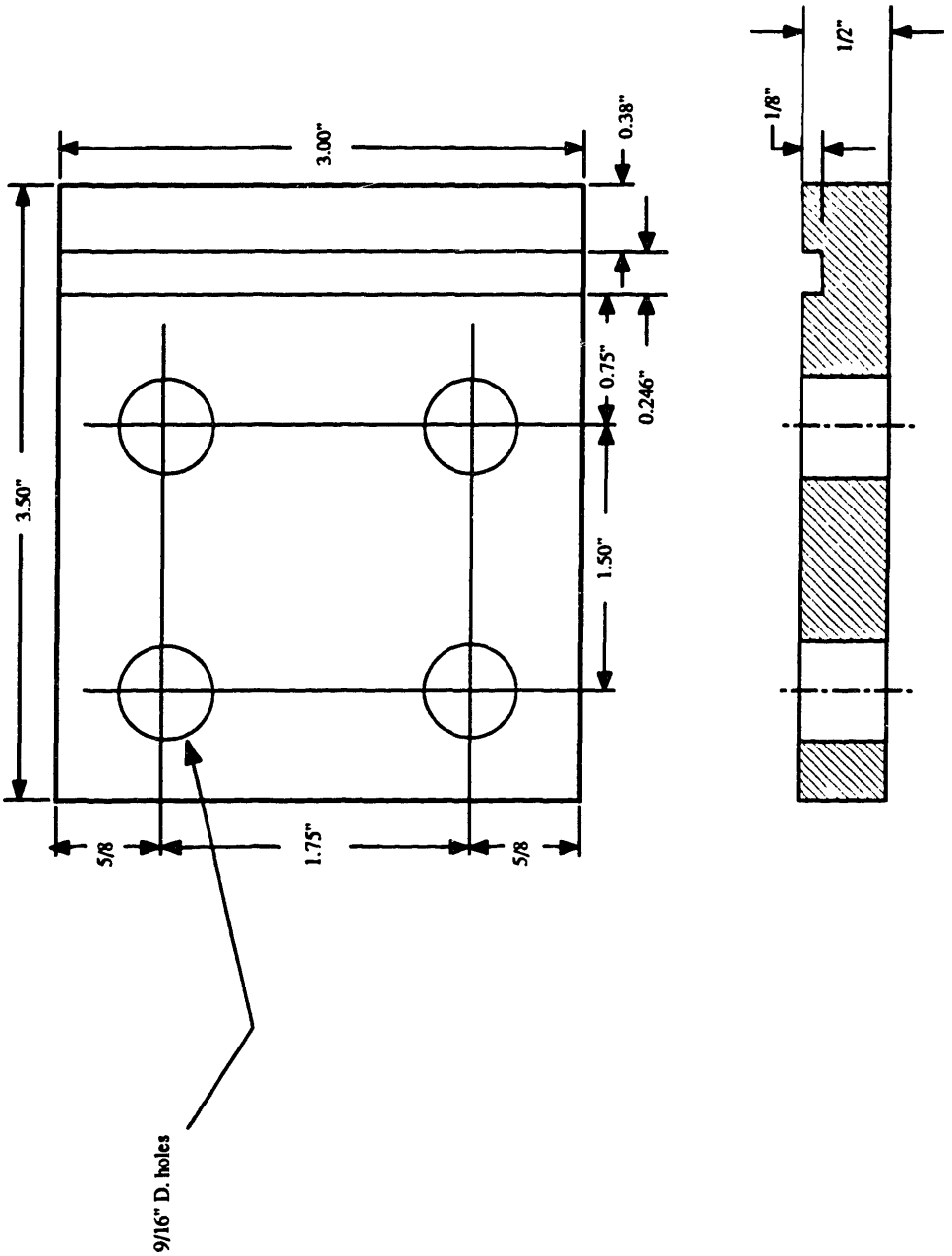


Scale: 1" = 0.4"

SOLENOIDS

(SMALL COIL - PART 7)

MAGNET5B.DRW

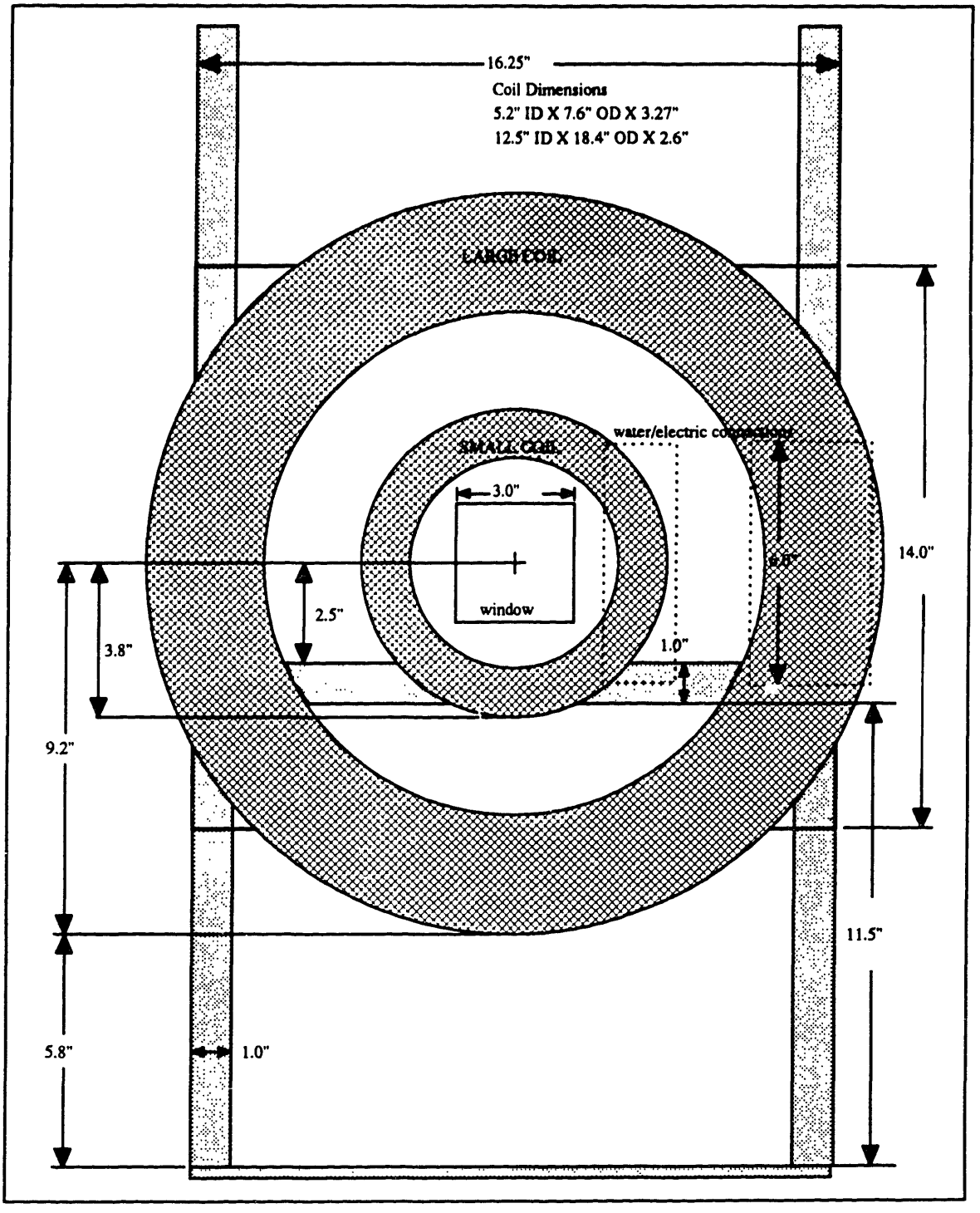


Scale: 1" = 1"
Make Four

SOLENOIDS

(SOLENOID MOUNTING CONFIGURATION)

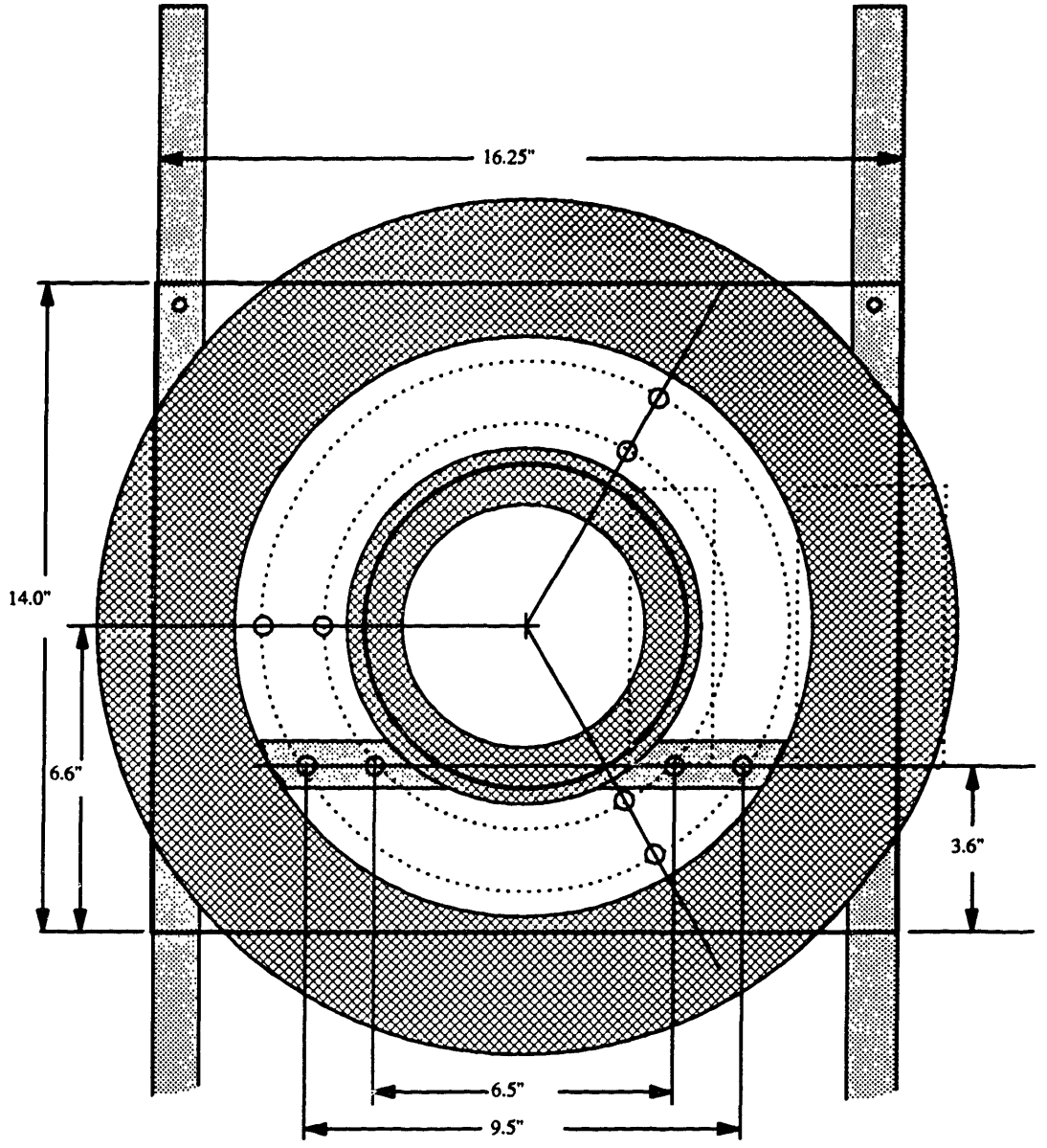
MAGNET9A.DRW



SOLENOIDS

(DIMENSIONS OF MOUNTING PLATE)

MAGNET9B.DRW



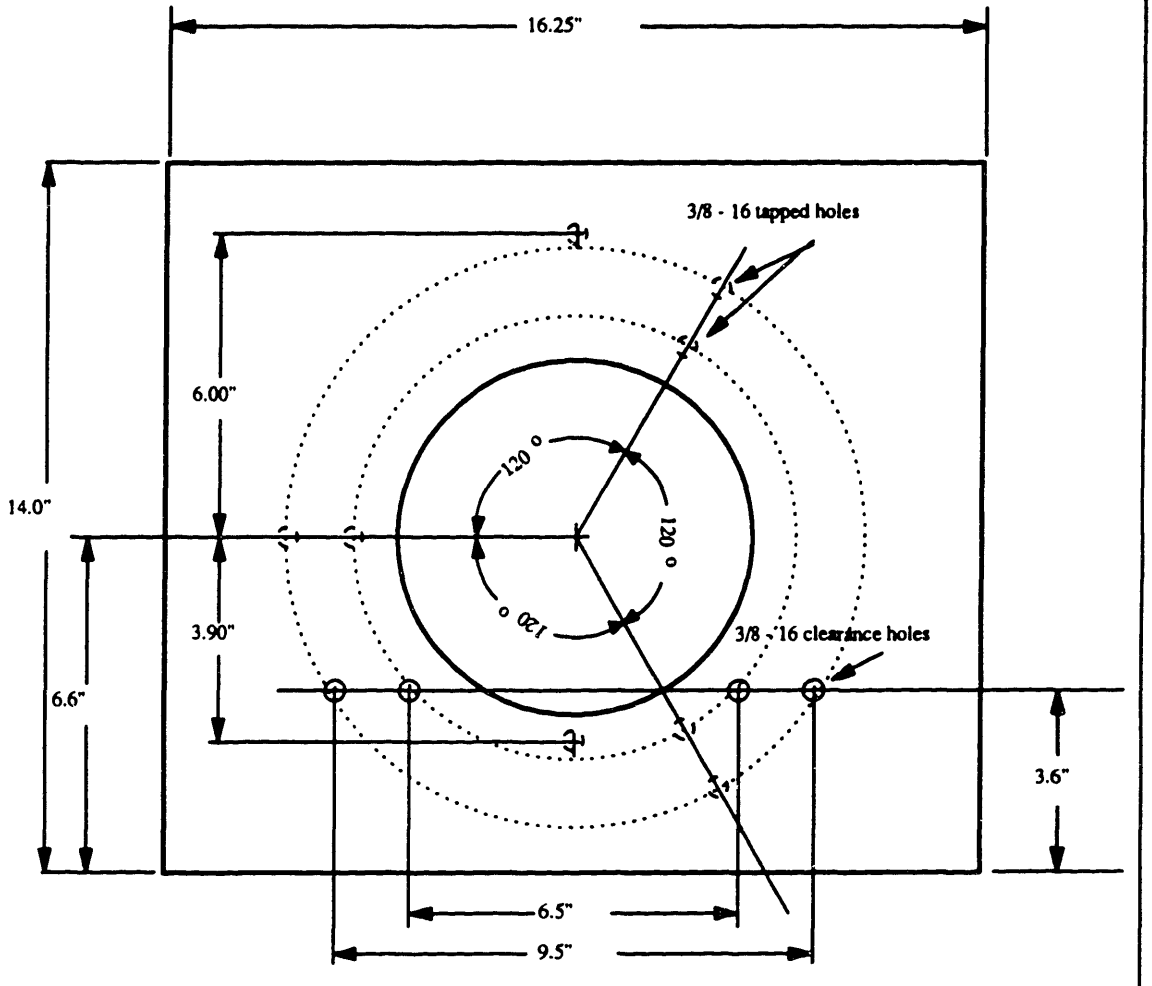
16.25" X 14.0" X 1/4" aluminum plate with
7.0" diameter hole drilled off-center.

SOLENOIDS

(SOLENOID MOUNTING PLATE)

MAGNET9C.DRW

Make Two



16.25" X 14.0" X 1/4" aluminum plate with
7.0" diameter hole drilled off-center.

three evenly spaced 3/8 - 16 tapped holes on 8.8" B.C.
three evenly spaced 3/8 - 16 tapped holes on 11.5" B.C.

SOLENOIDS

May 16, 1990

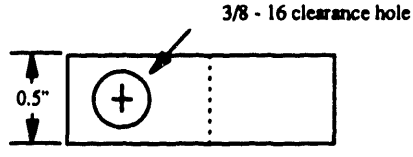
(SOLENOID MOUNTING HOLDERS)

MAGNET10.DRW

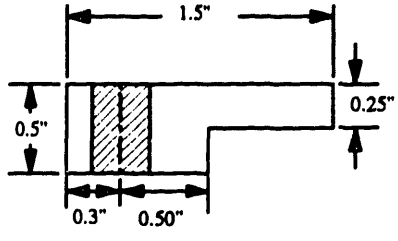
Material: Aluminum

Make six of these

top view

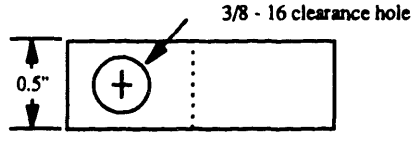


cross section

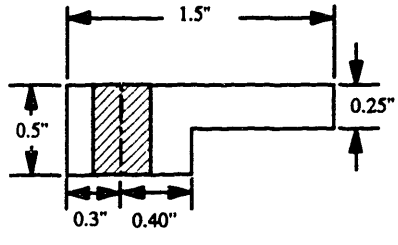


Make six of these

top view



cross section



APPENDIX C

ION ANALYZER DESIGN

This appendix contains the design drawings for fabricating the ion analyzer and schematics of the electrical connections associated with the analyzer. The grids of the analyzer are made from tungsten. The collecting plate is made from silver plated MACOR. Parts were purchased from Unique Wire Weaving, Co., Kimball Physics, and fabrication of parts were done at Signet Tool & Engineering, Inc. and Pekay Industries.

ION ANALYZER

(GRIDS)

FILE: ION1.DRW

Scale: 1" = 1"

Spacer Length: Al2O3-SP-C-###

d1 = 0.032" (teflon sheet)
d2 = 0.100"
d3 = 0.100"
d4 = 0.100"
d5 = 0.870"

Grid Radii

	collection angle
1 = 0.474"	(41.2 °)
2 = 0.651"	(42.2 °)
3 = 0.786"	(39.5 °)
4 = 0.995"	(44.7 °)

Plate Thickness:

1 = 0.025" SS-PL-C5x5-R625
2 = 0.025" SS-PL-C5x5-R875
3 = 0.025" SS-PL-C5x5-R1000
4 = 0.423"

Total collection angle = 39.5 °

Spacers, Screw, and Plates

Kimball Physics, Inc.
Wilton, NH 03086
(603) 878-1616

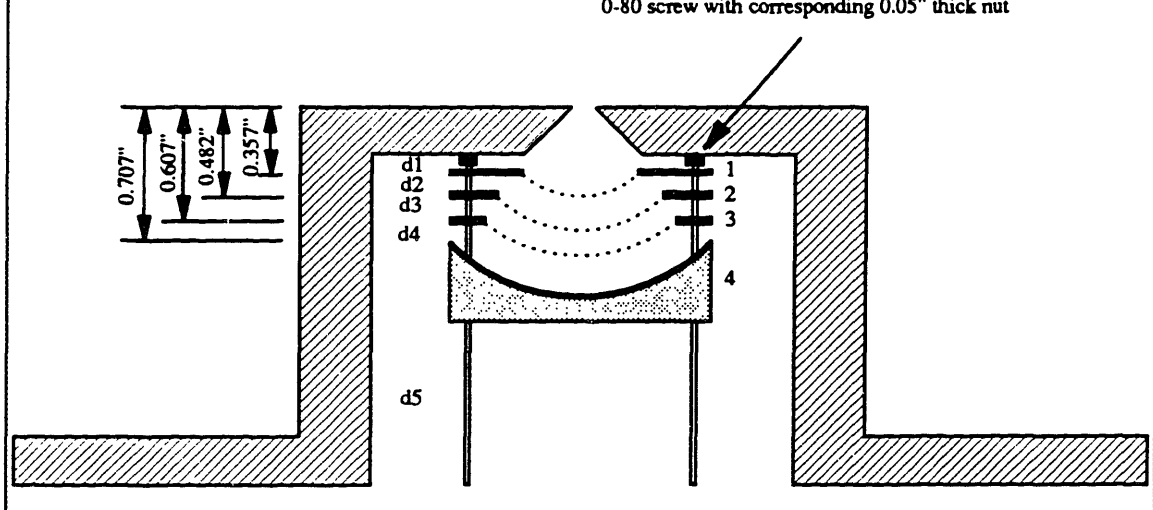
Collecting Plate = silver plated MACOR

1) machining MACOR
Signet Tool & Engineering, Inc.
205 Newbury St.
Peabody, MA 01960

2) plating MACOR with silver
Pekay Industries
Farmingdale, NJ 07727
(201) 938-2722

Tungsten Grid

50 X 50 mesh, 0.001" sire (90.25% transmission per sheet)
Unique Wire Weaving Co., Inc.
Hillside, NJ



ION ANALYZER
(COLLECTING PLATE)

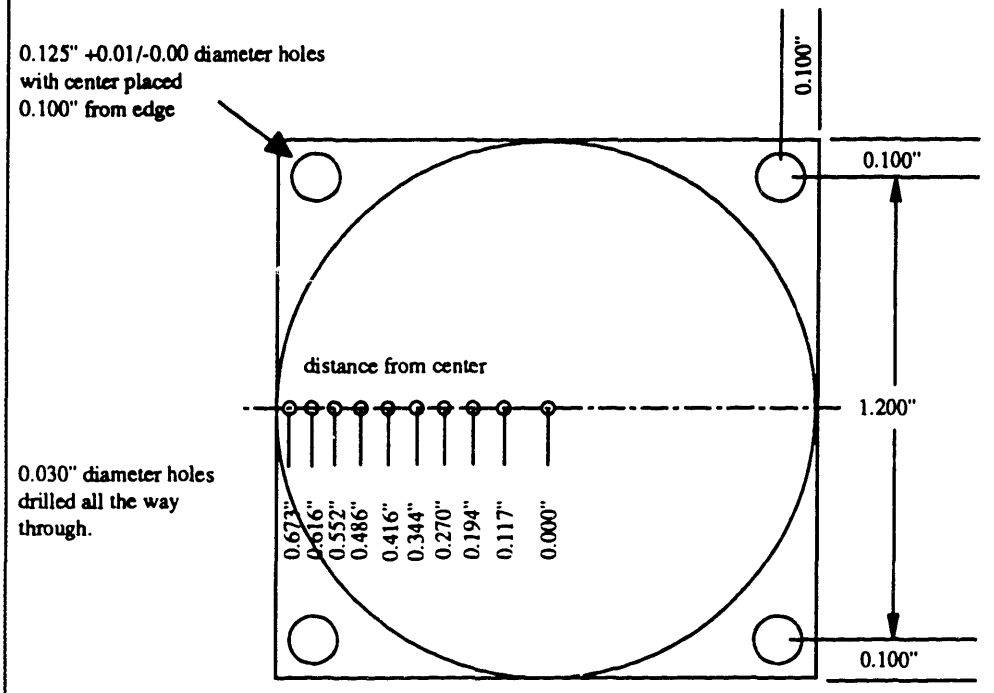
Nov. 7, 1989

FILE: ION2.DRW

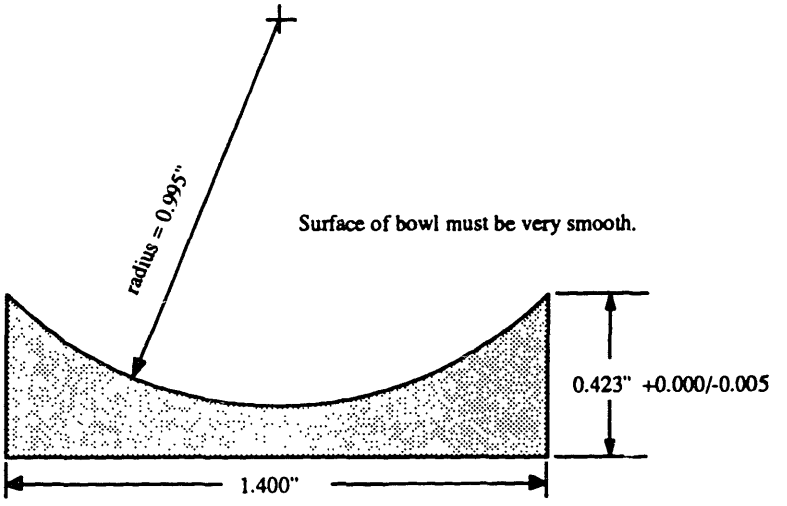
Scale: 2" = 1"
Material: MACOR
Machining by Signet Tool & Engineering, Inc.

Top View

** Note: Make sure the bowl is well centered on block because you will need to find the center of the bowl later on.



Cross Section

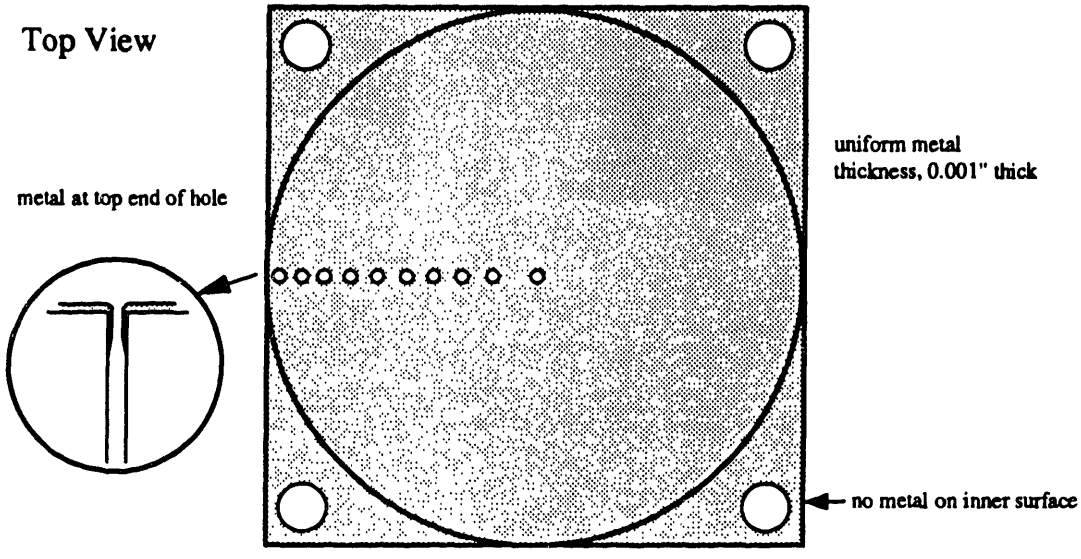


ION ANALYZER
(COLLECTING PLATE)

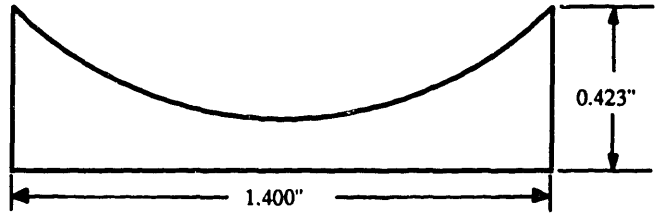
FILE: ION3.DRW

Material: MACOR
Scale: 2" = 1"
Plating by PeKay Industries

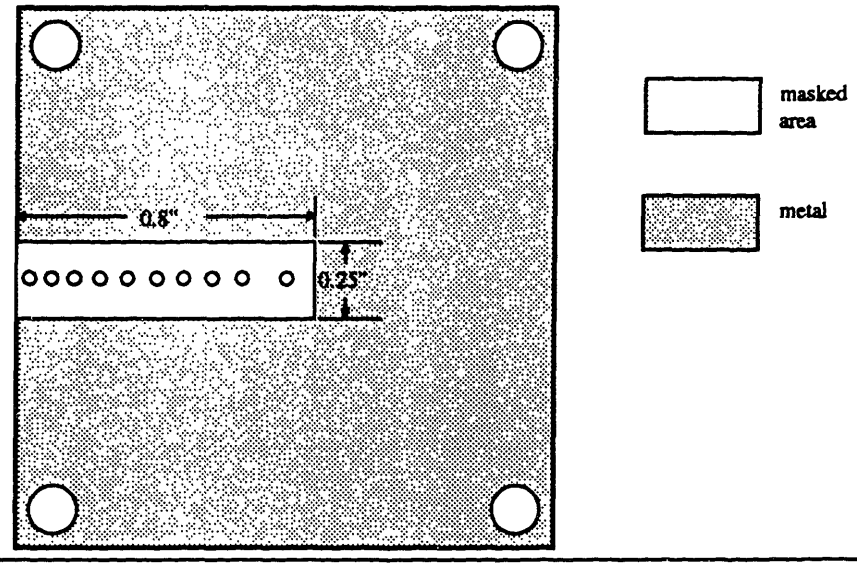
Top View



Cross Section



Bottom View



ION ANALYZER

(COLLECTING PLATE)

Dec. 4, 1989

FILE: ION4.DRW

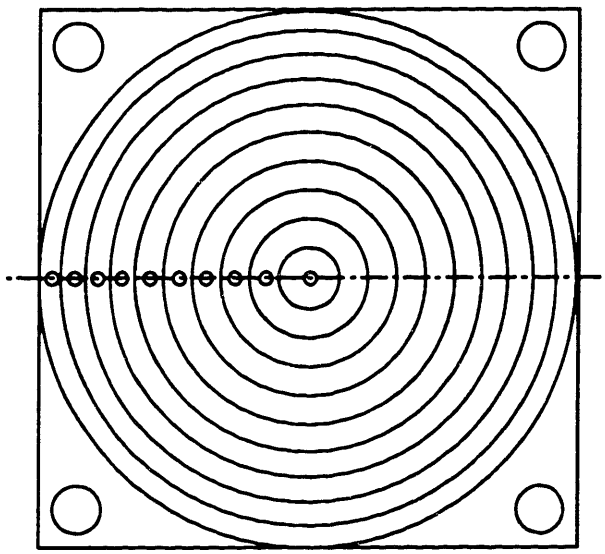
Scale: 2" = 1"

Material: MACOR (machineable ceramic by Corning)

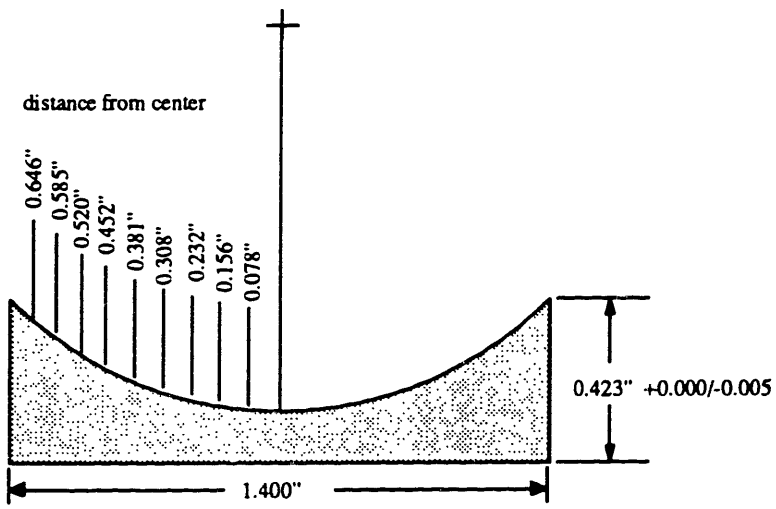
Machining by Signet Tool & Eng., Inc.

Please use very thin tool, less than 0.01" wide, to etch the metal surface. The depth of the grooves should be deep enough to separate the annular metal rings, approximately 0.002" deep. Please check for electrical isolation of annular rings before removing from machine.

Top View



Cross Section

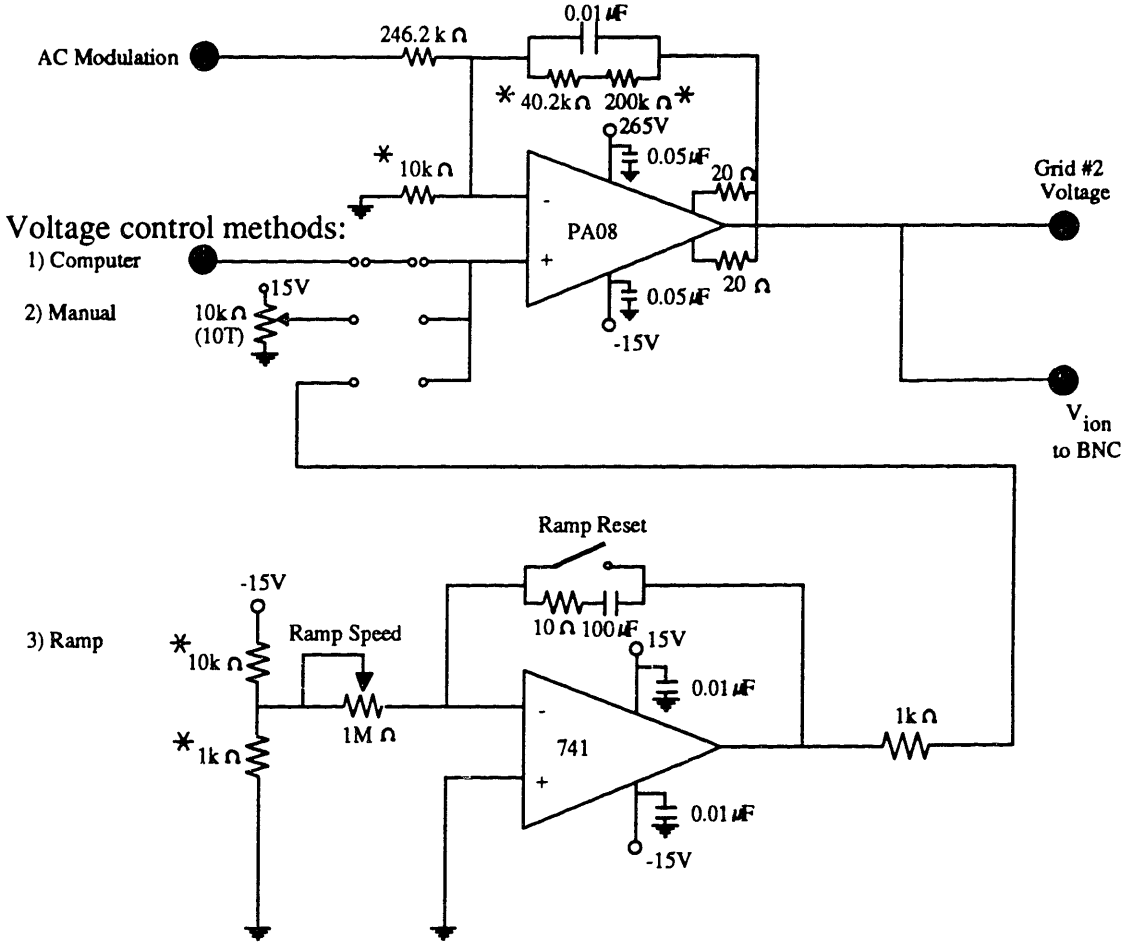


ION ANALYZER

(CIRCUIT FOR GRID # 2)

FILE: ION5.DRW

Second Grid: V_{ion}



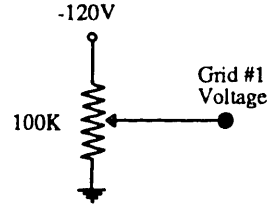
* = Metal Film Resistors, 1%

ION ANALYZER

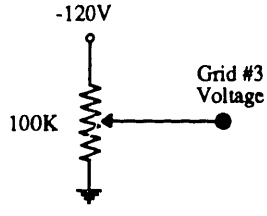
(CIRCUIT FOR GRID # 1 AND # 3)

FILE: ION6.DRW

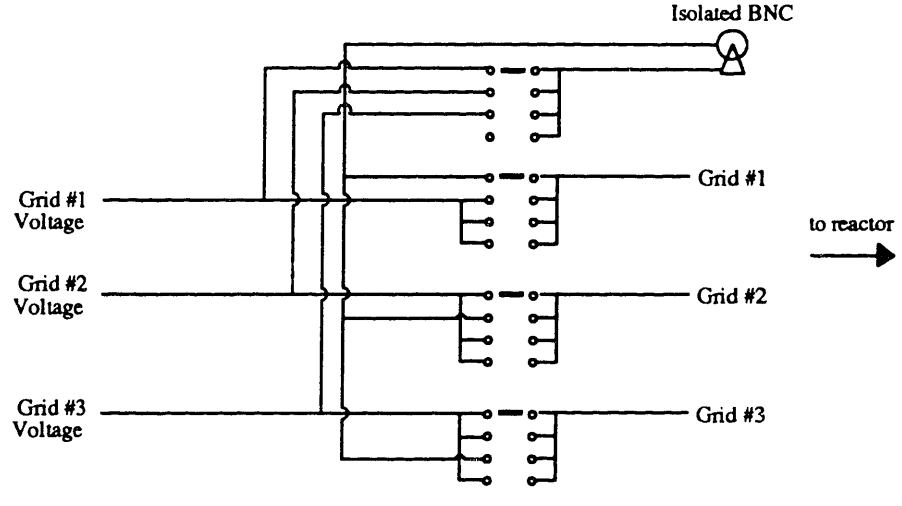
First Grid Voltage



Third Grid Voltage



Switch for Measuring Grid Current

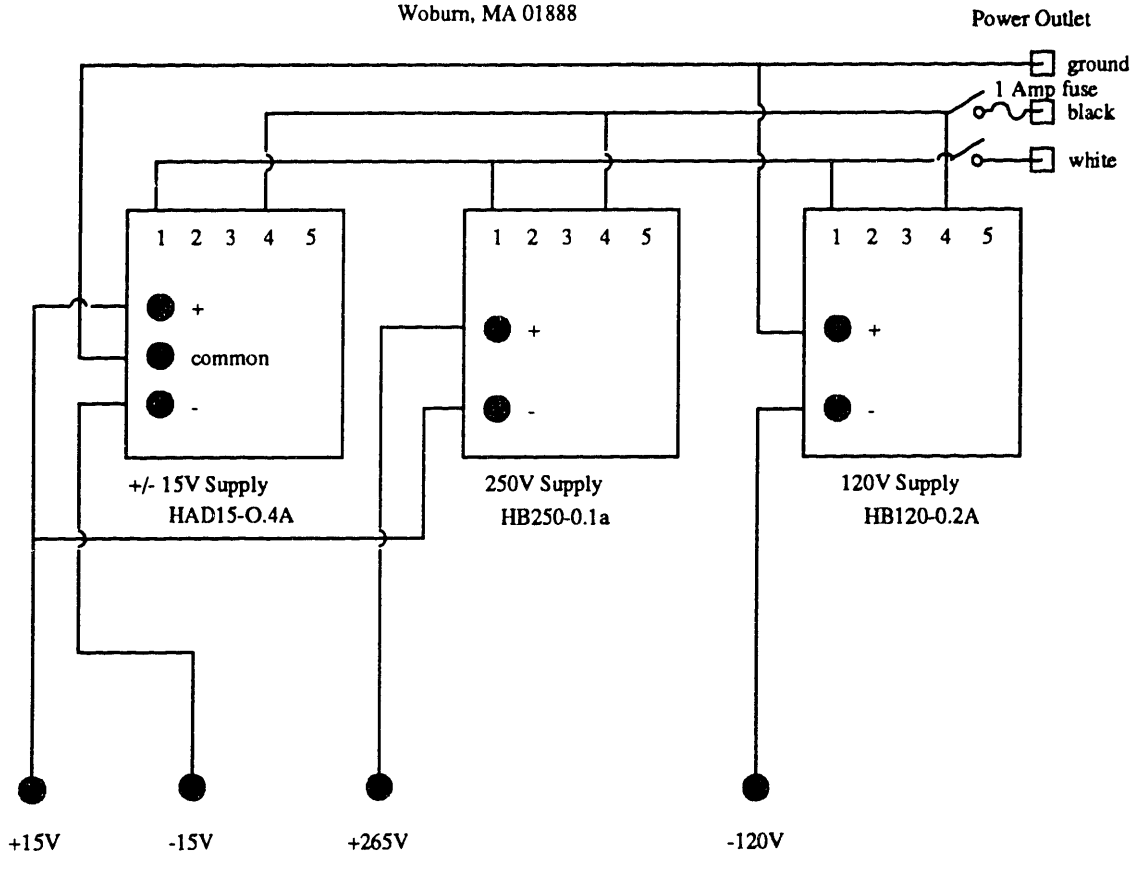


ION ANALYZER

(POWER SUPPLIES FOR GRIDS) FILE: ION7.DRW

DC Power Supplies

Sterling Electronics
15D Constitution Way
Woburn, MA 01888

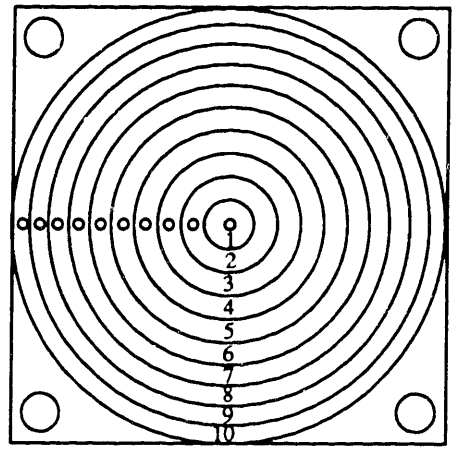


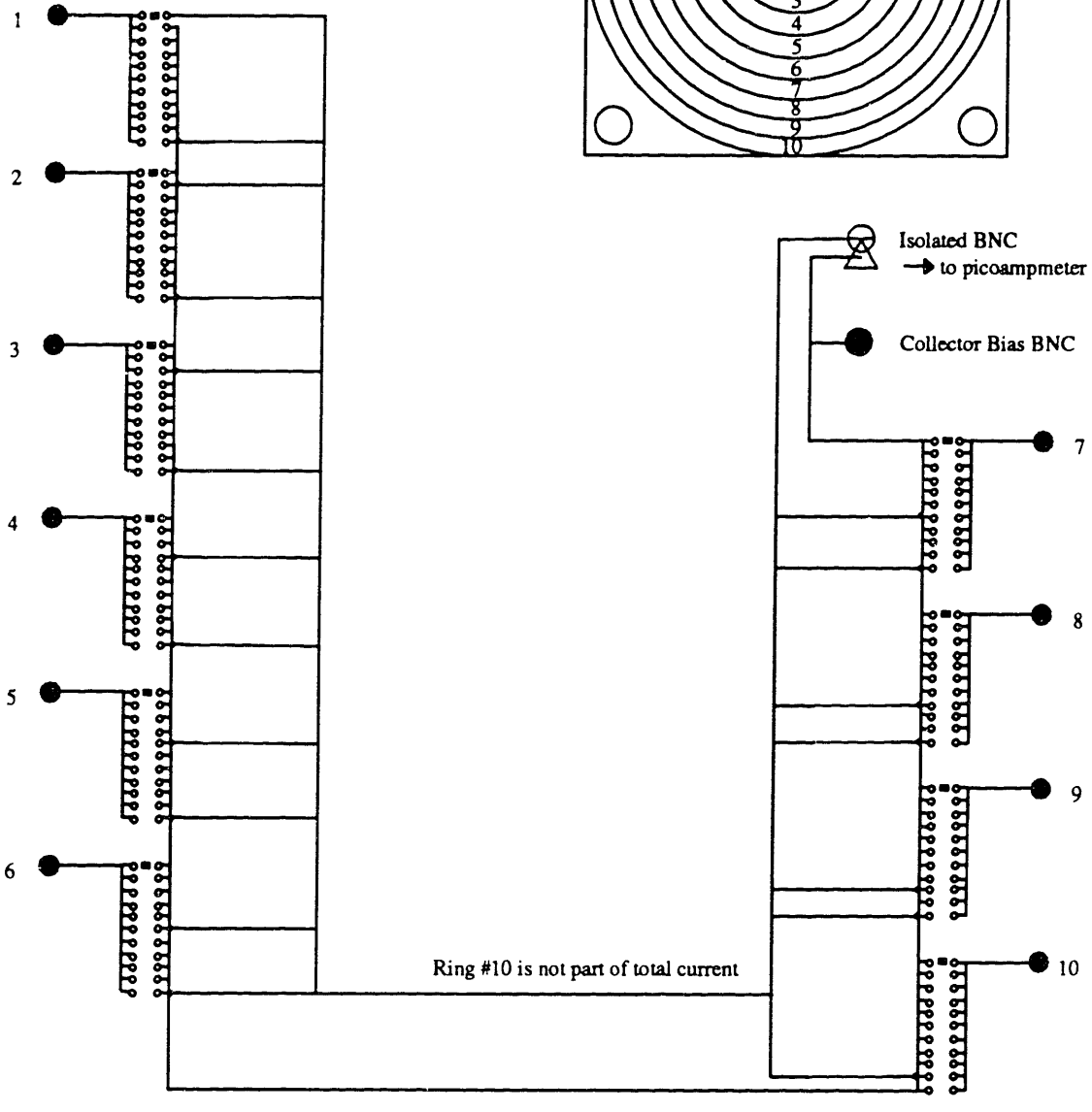
ION ANALYZER

(ION CURRENT SWITCH)

FILE: ION8.DRW

Newark
10G Roessler Rd
Woburn, Ma 01801
Part No. 22F851 & 22F651 - CENTRALAB SWITCH

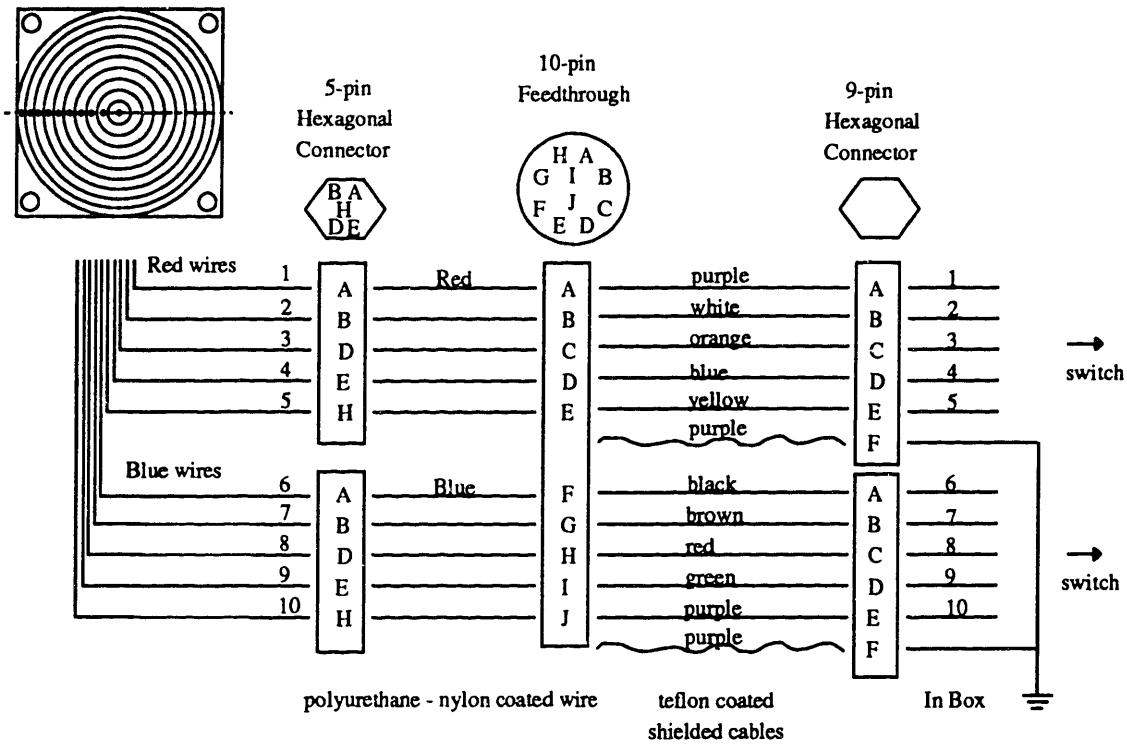




ION ANALYZER

(ION COLLECTING PLATE CONNECTIONS)

FILE: ION9.DRW

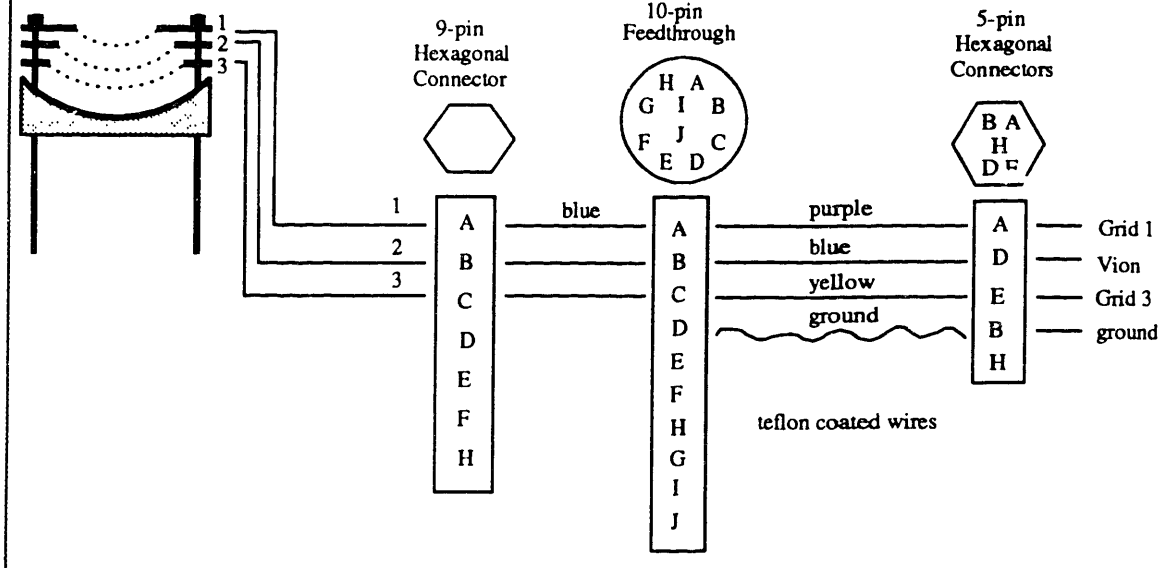


ION ANALYZER

Aug. 25, 1989

(CONNECTIONS FROM GRIDS TO BOX)

FILE: ION10.DRW

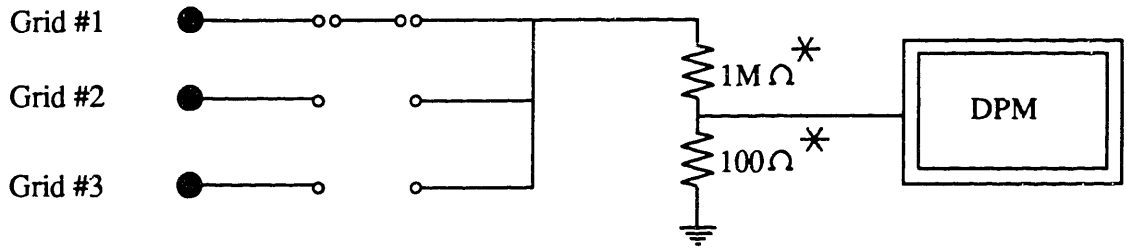


ION ANALYZER

(DIGITAL PANEL METER)

FILE: ION11.DRW

Burton-Rogers Co., Inc.
Calibron Instruments
220 Grove Street
Waltham, MA 02154
Model: AN25MOO-TP-1-X-X-01-X



APPENDIX D

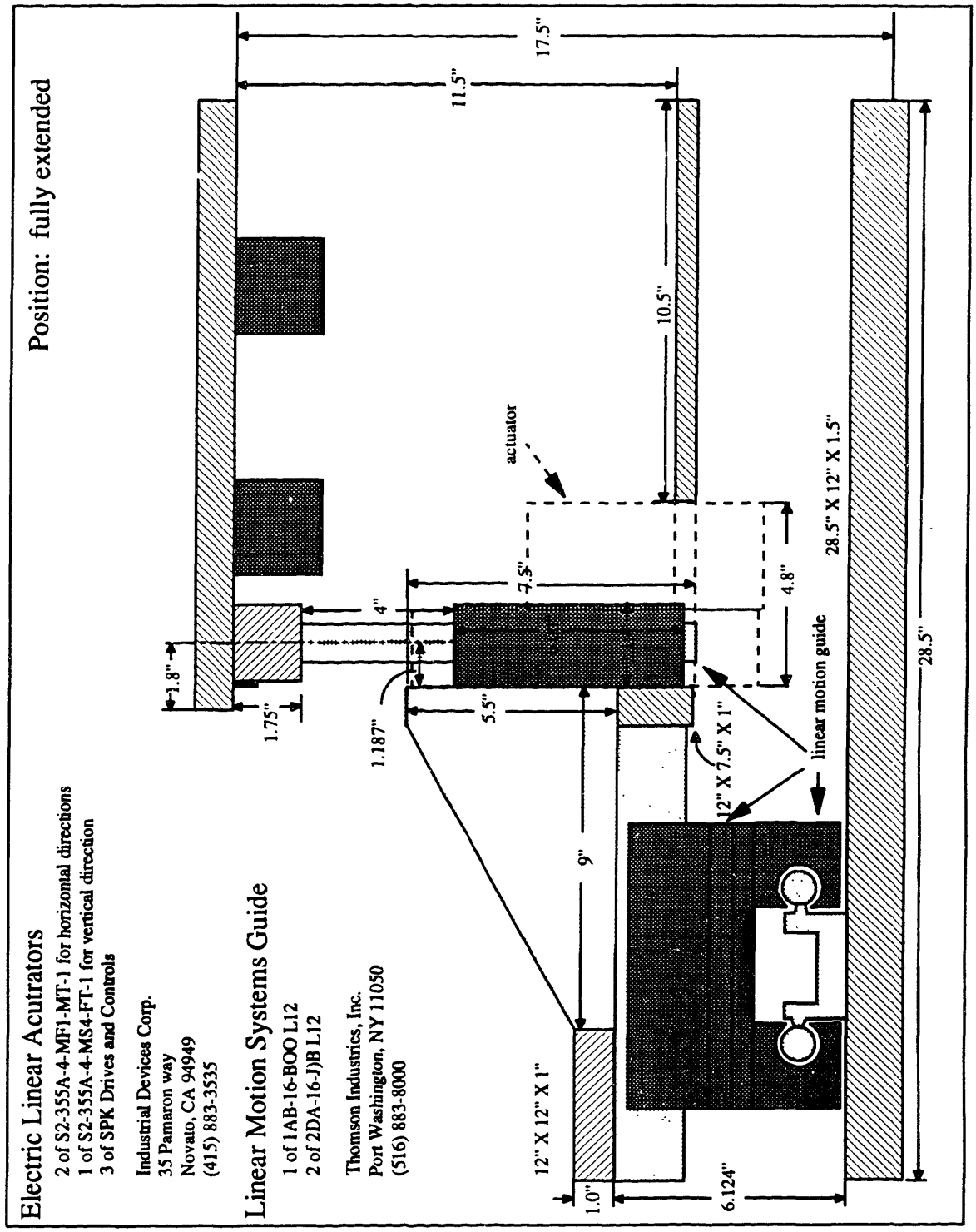
REACTOR MOVEMENT ASSEMBLY DESIGNS

This appendix contains the machine drawings for making the x-y-z translational stage for the reactor. Some of the drawings contain alterations to existing reactor pieces, and pieces purchased with the linear motion systems. Also included is the circuit diagram for connecting the computer D/A board to the stepping motor drive controller.

REACTOR MOVEMENT ASSEMBLY

(ASSEMBLED DRAWING)

MOVE1A.DRW

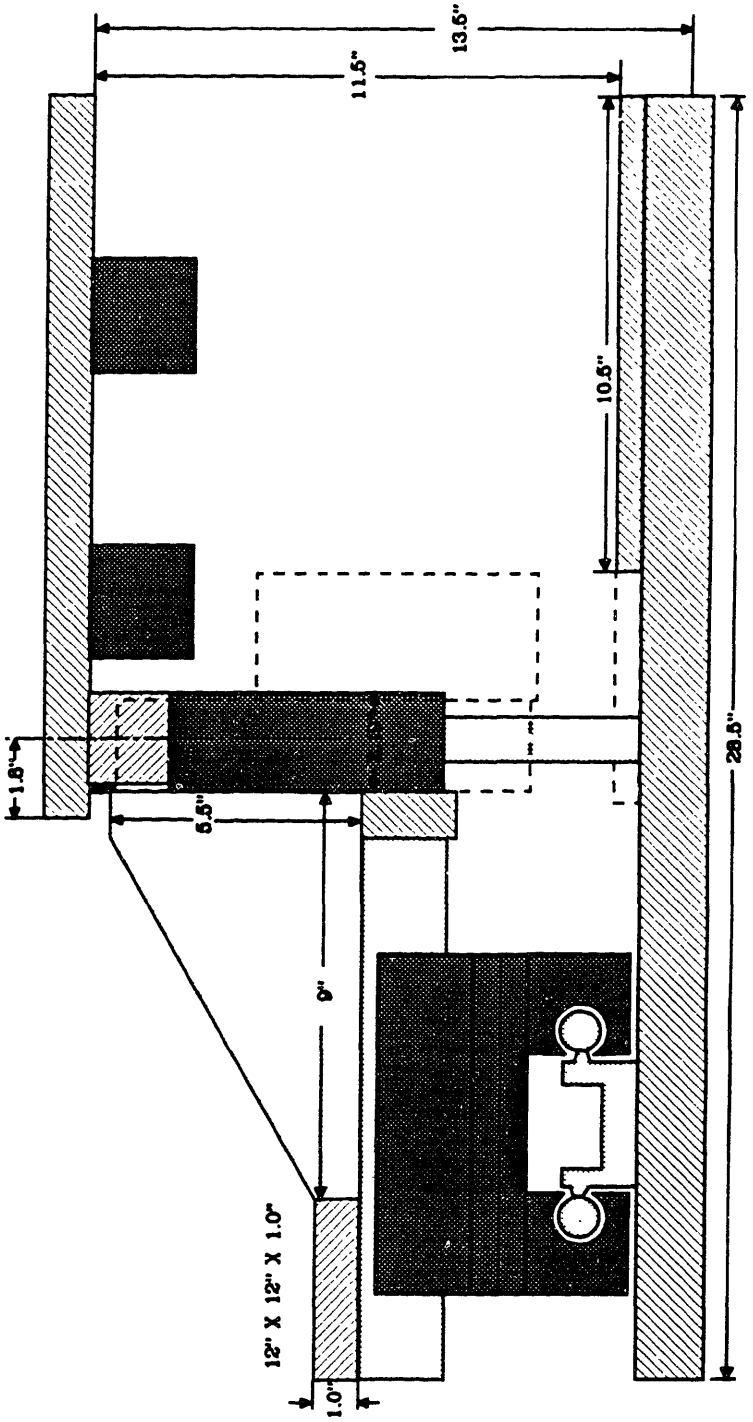


REACTOR MOVEMENT ASSEMBLY

(ASSEMBLED DRAWING)

MOVE1B.DRW

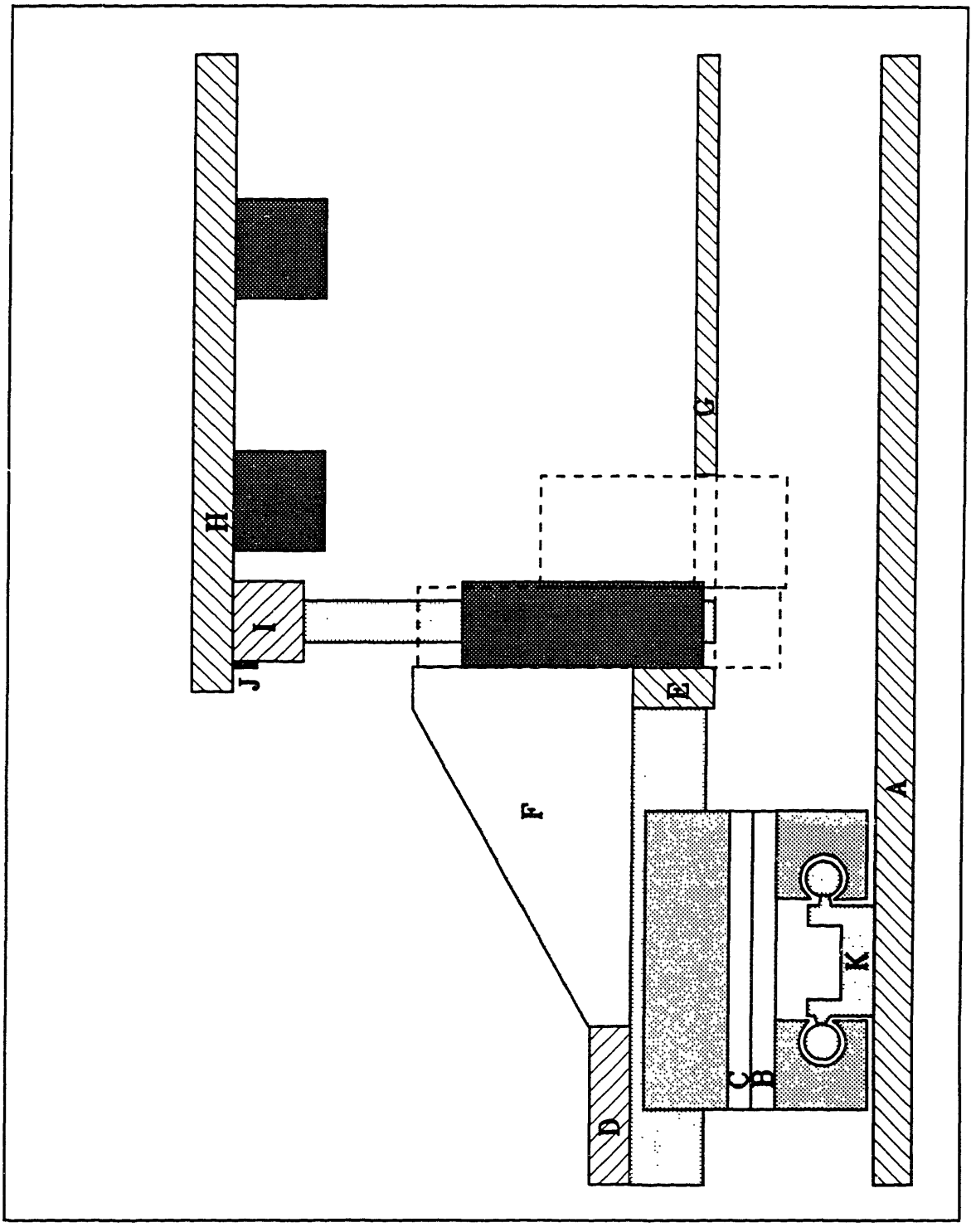
Position: fully retracted



REACTOR MOVEMENT ASSEMBLY

(PARTS IDENTIFICATION)

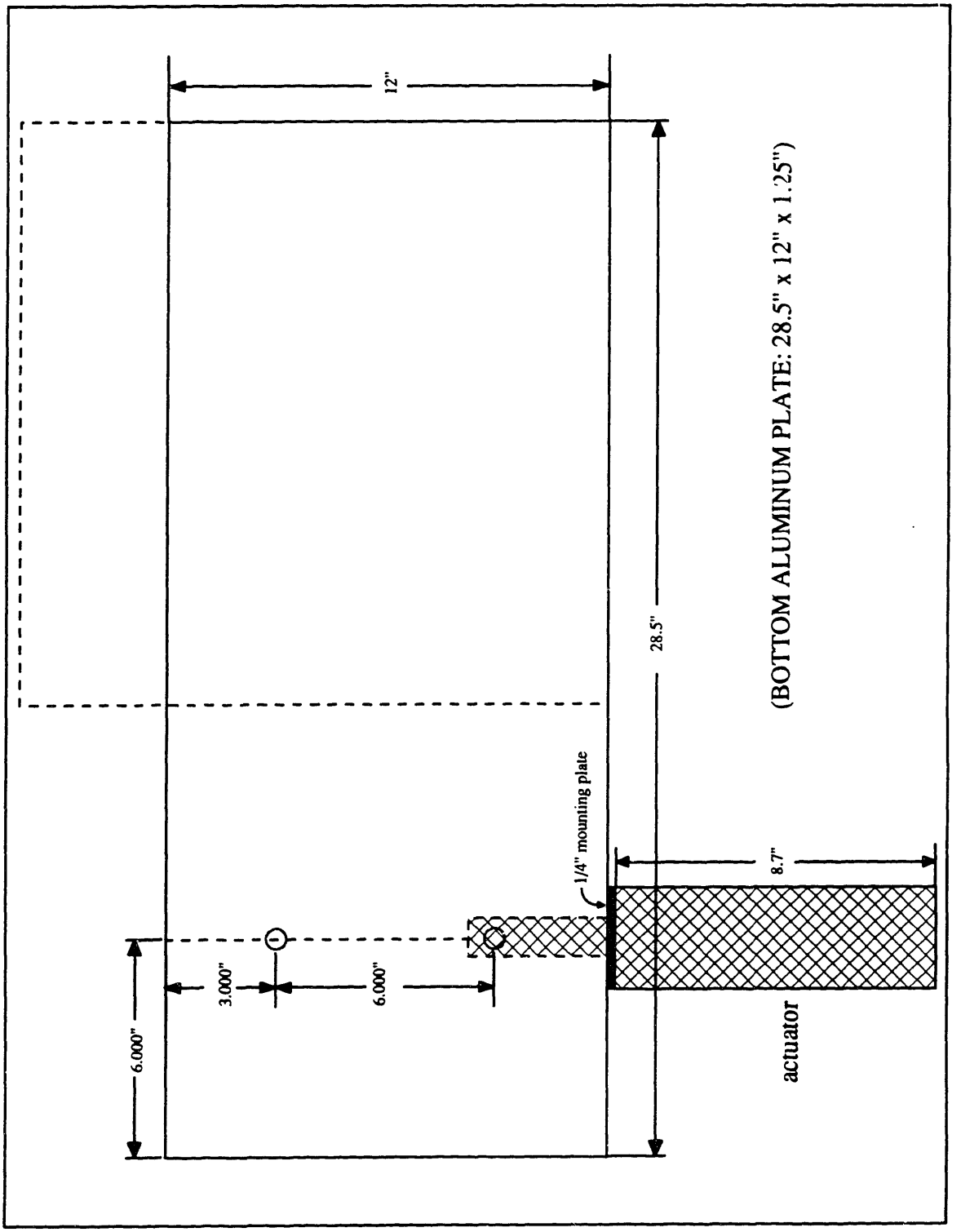
MOVE2.DRW



REACTOR MOVEMENT ASSEMBLY

(PART A)

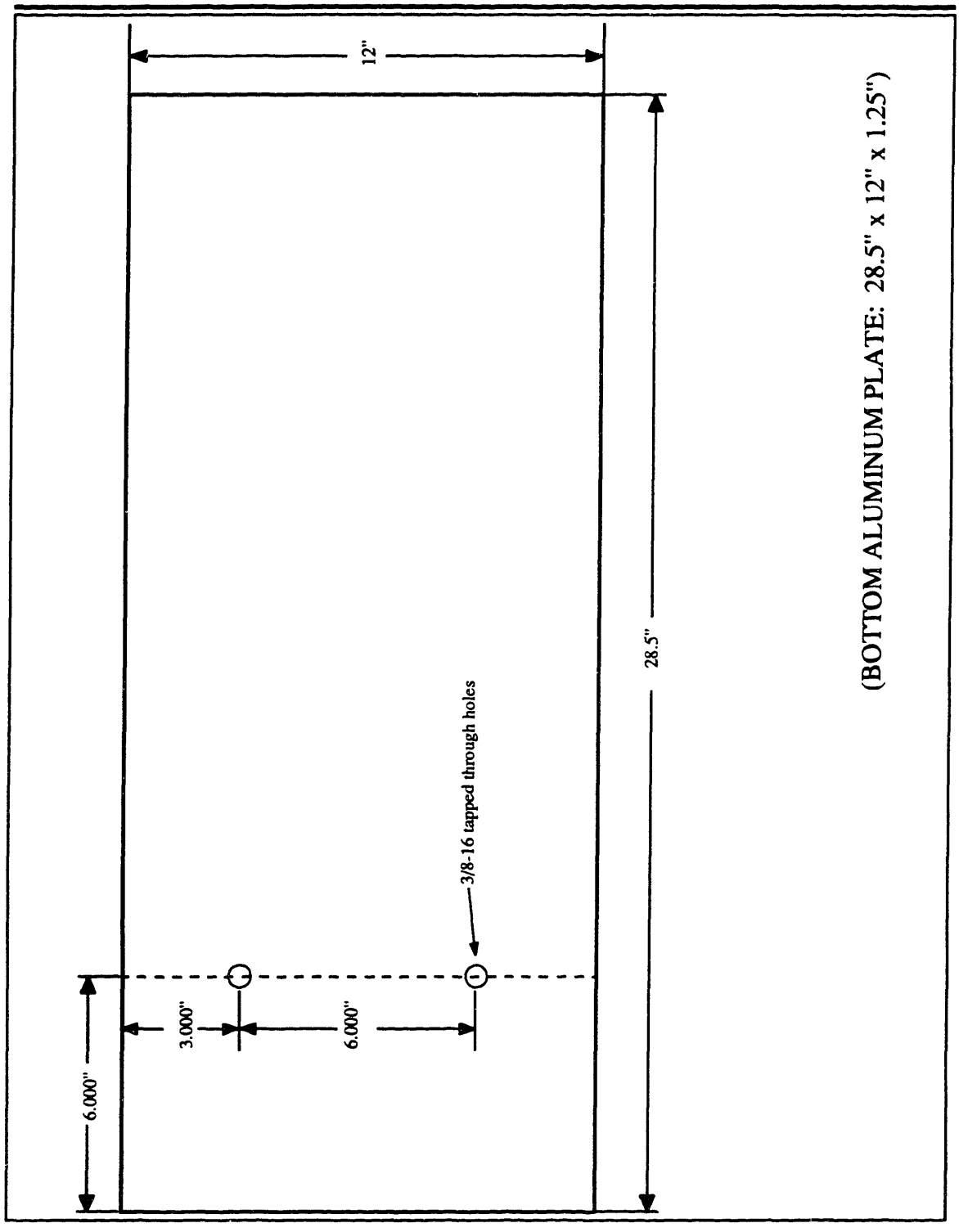
MOVE3A.DRW



REACTOR MOVEMENT ASSEMBLY

(PART A)

MOVE3B.DRW



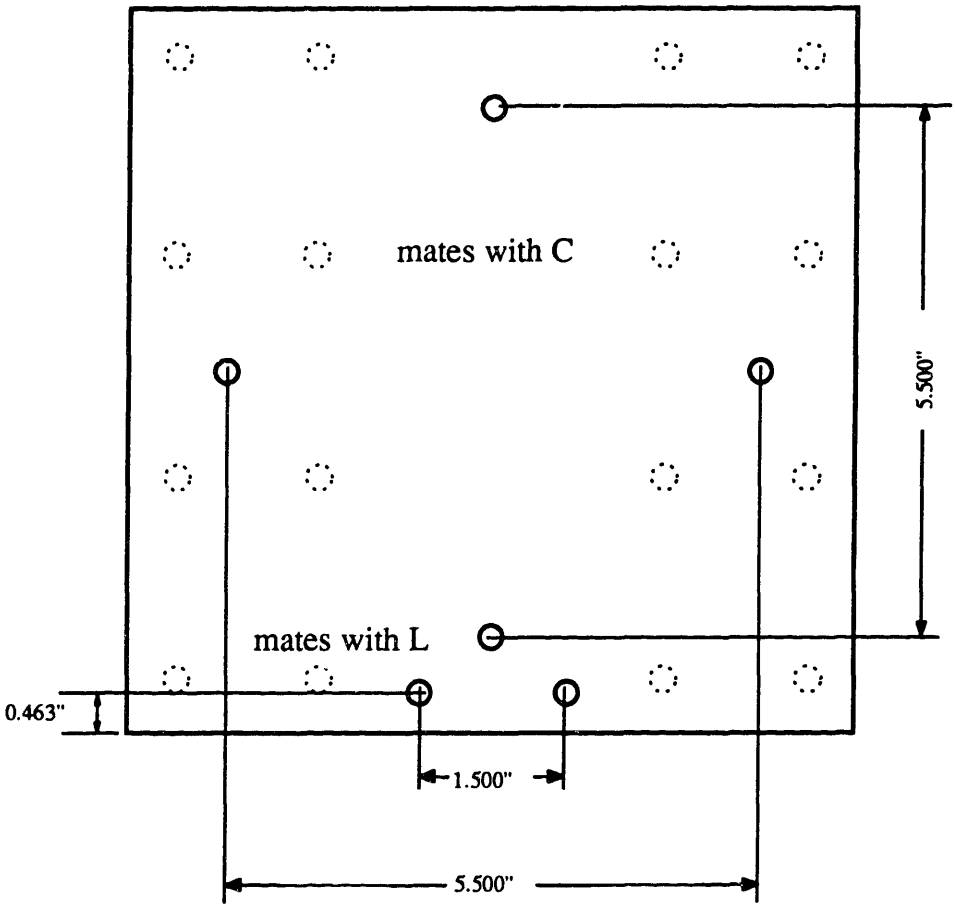
REACTOR MOVEMENT ASSEMBLY

(PART B)

MOVE10A.DRW

ALTERATIONS TO TOP PLATE OF SLIDER (#1)

Scale: 1" = 2"



O add 10-32 tapped holes

existing holes

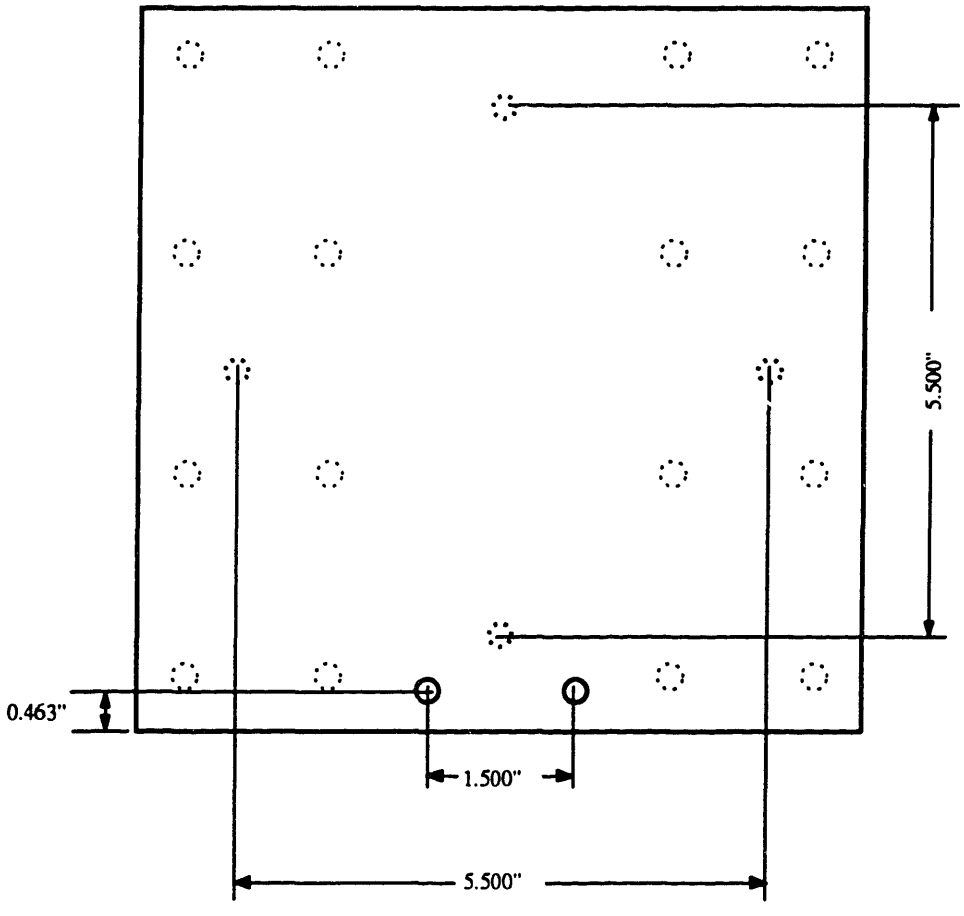
REACTOR MOVEMENT ASSEMBLY

(PART C)

MOVE10B.DRW

ALTERATIONS TO TOP PLATE OF SLIDER (#2)

Scale: 1" = 2"



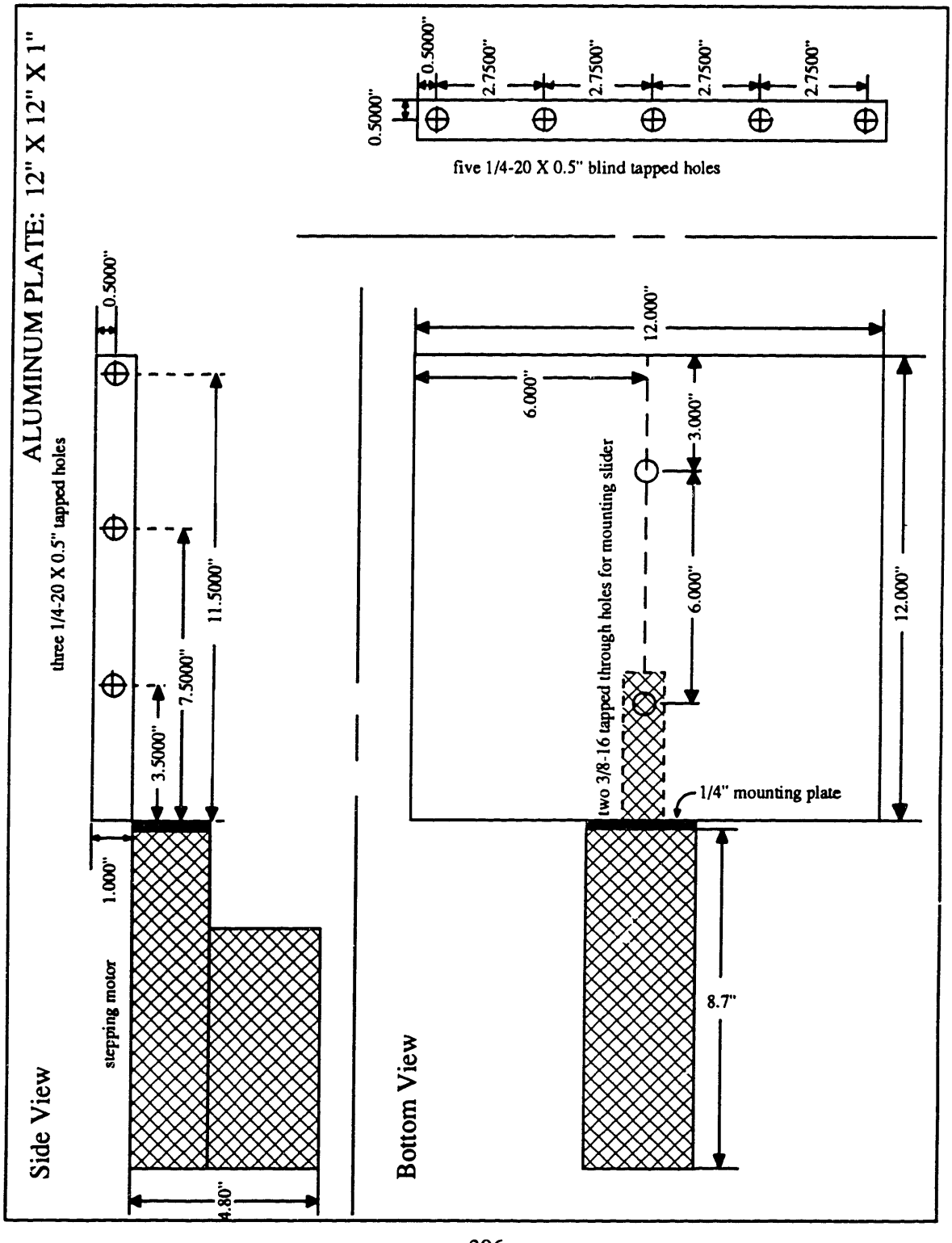
10-32 clearance holes with countersinks
for socket head cap screws
(countersinks on opposite side of existing countersinks)

10-32 tapped holes

existing holes

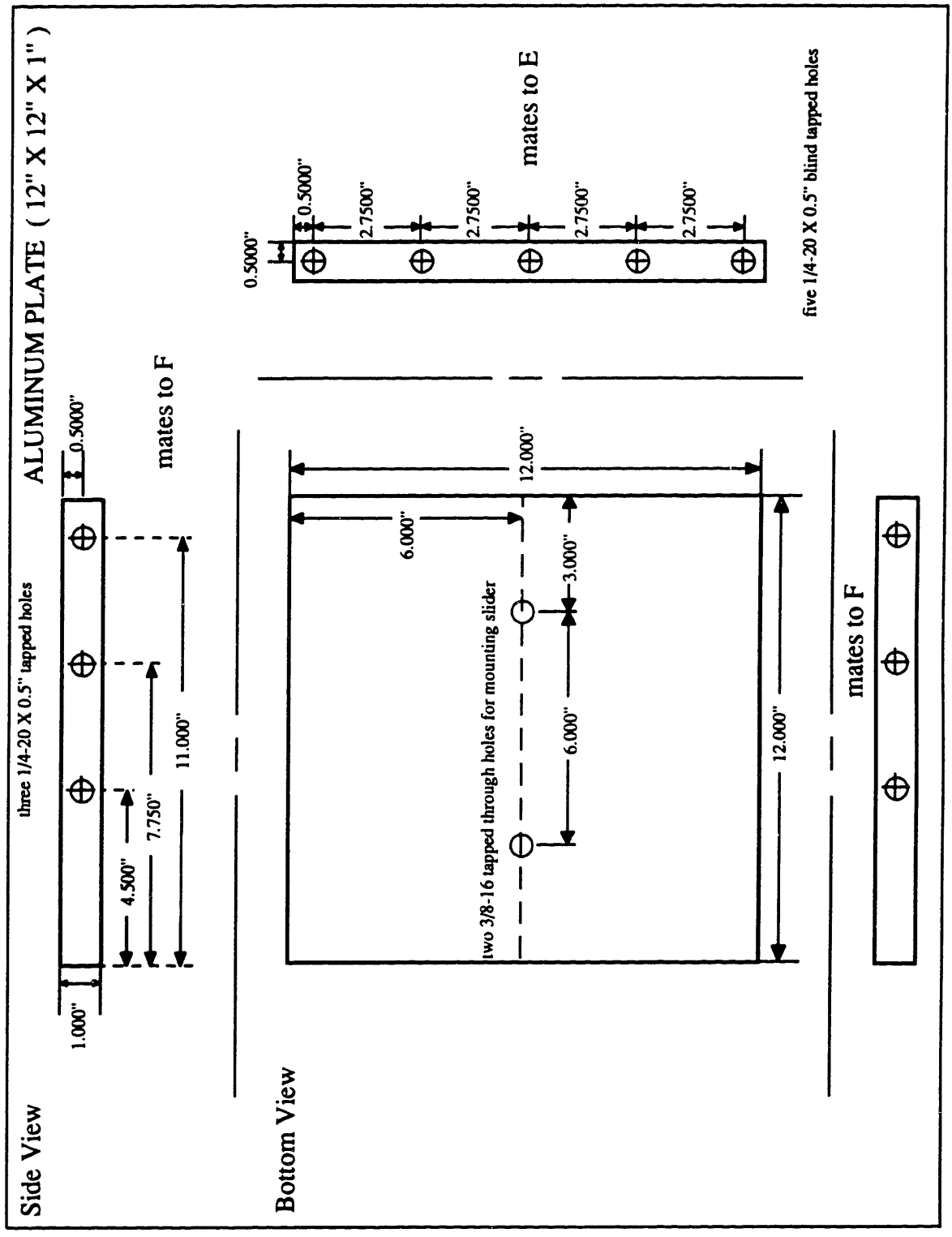
REACTOR MOVEMENT ASSEMBLY

(PART D) MOVE4A.DRW



REACTOR MOVEMENT ASSEMBLY

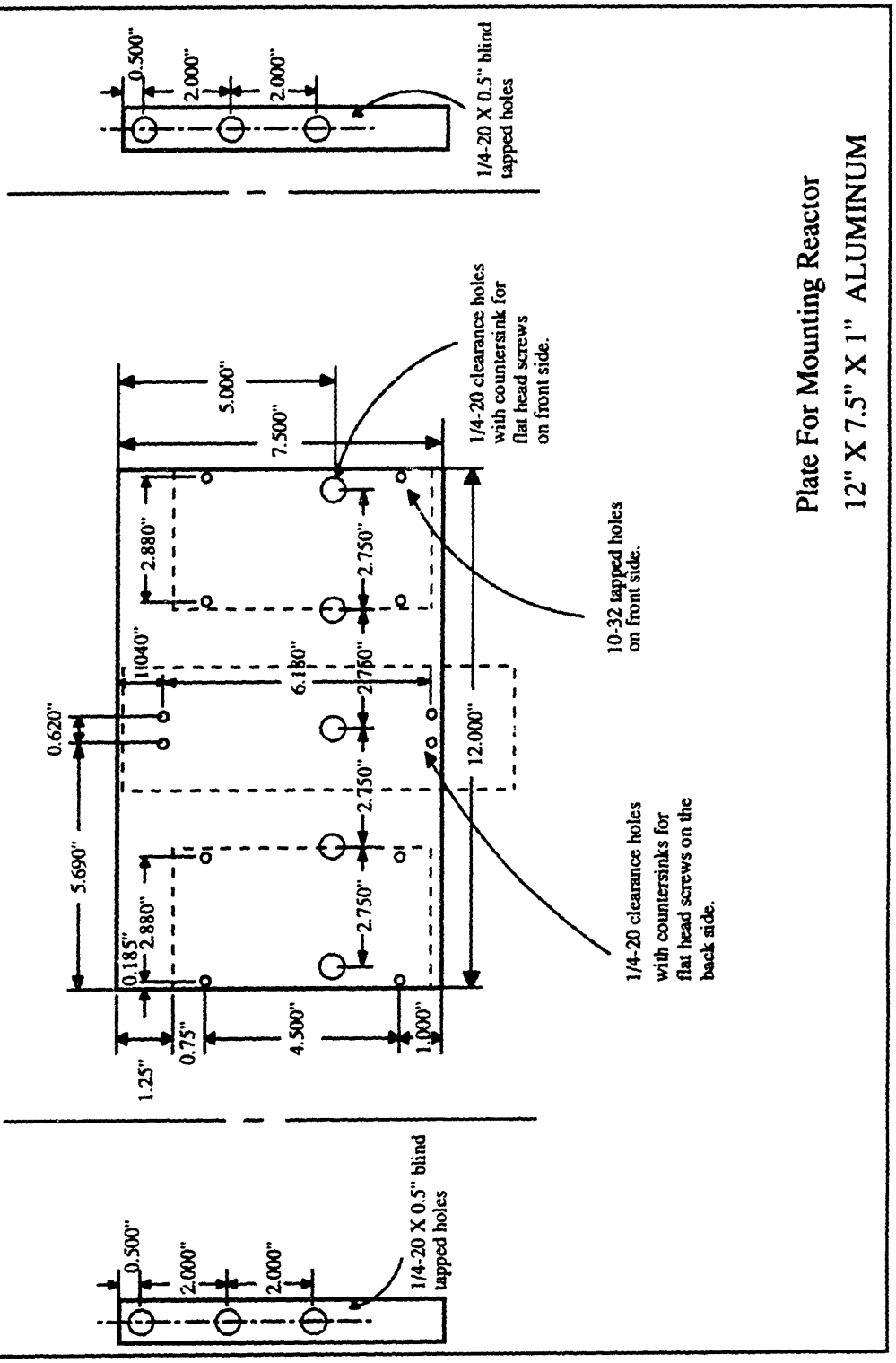
(PART D) MOVE4B.DRW



REACTOR MOVEMENT ASSEMBLY

(PART E)

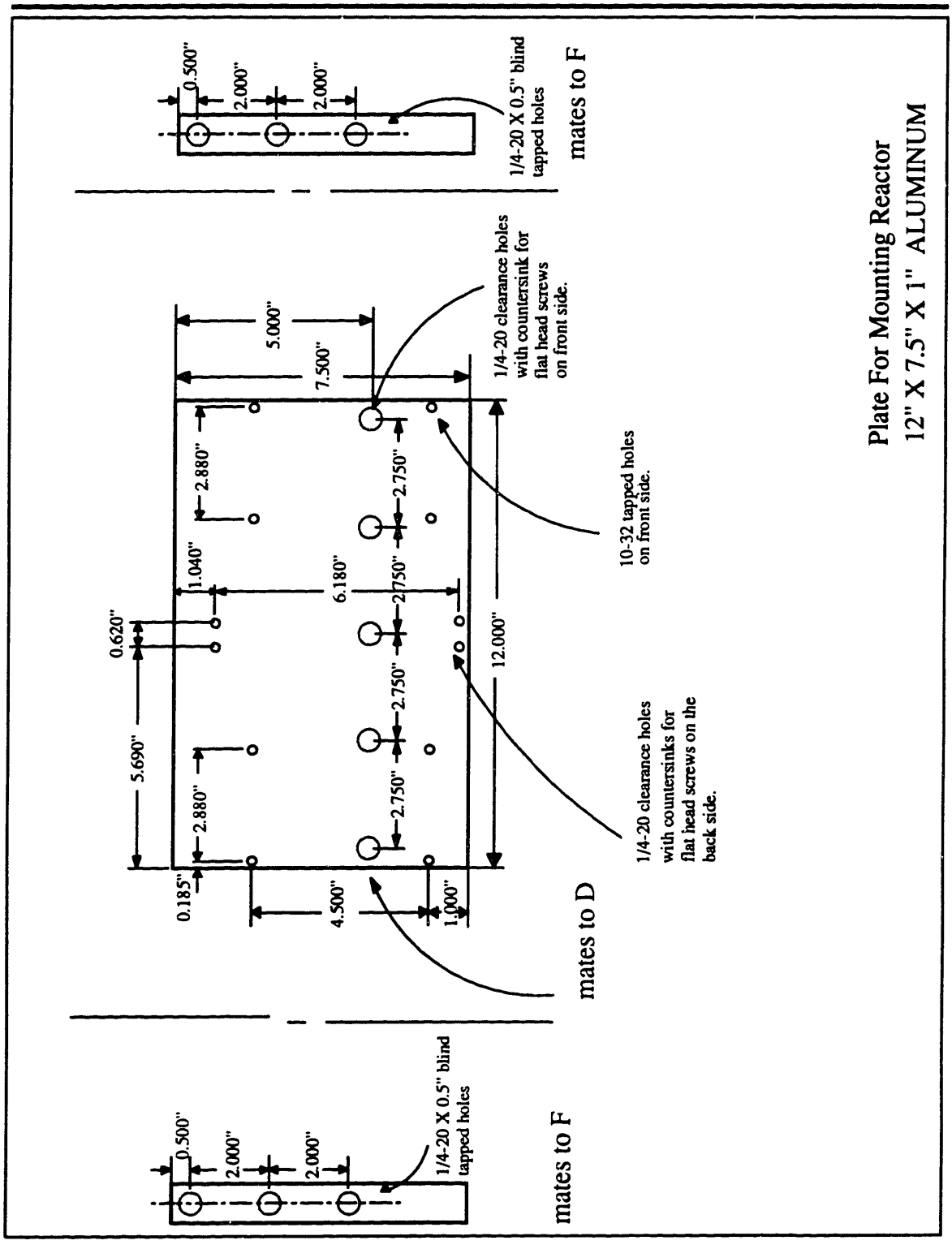
MOVE5A.DRW



REACTOR MOVEMENT ASSEMBLY

(PART E)

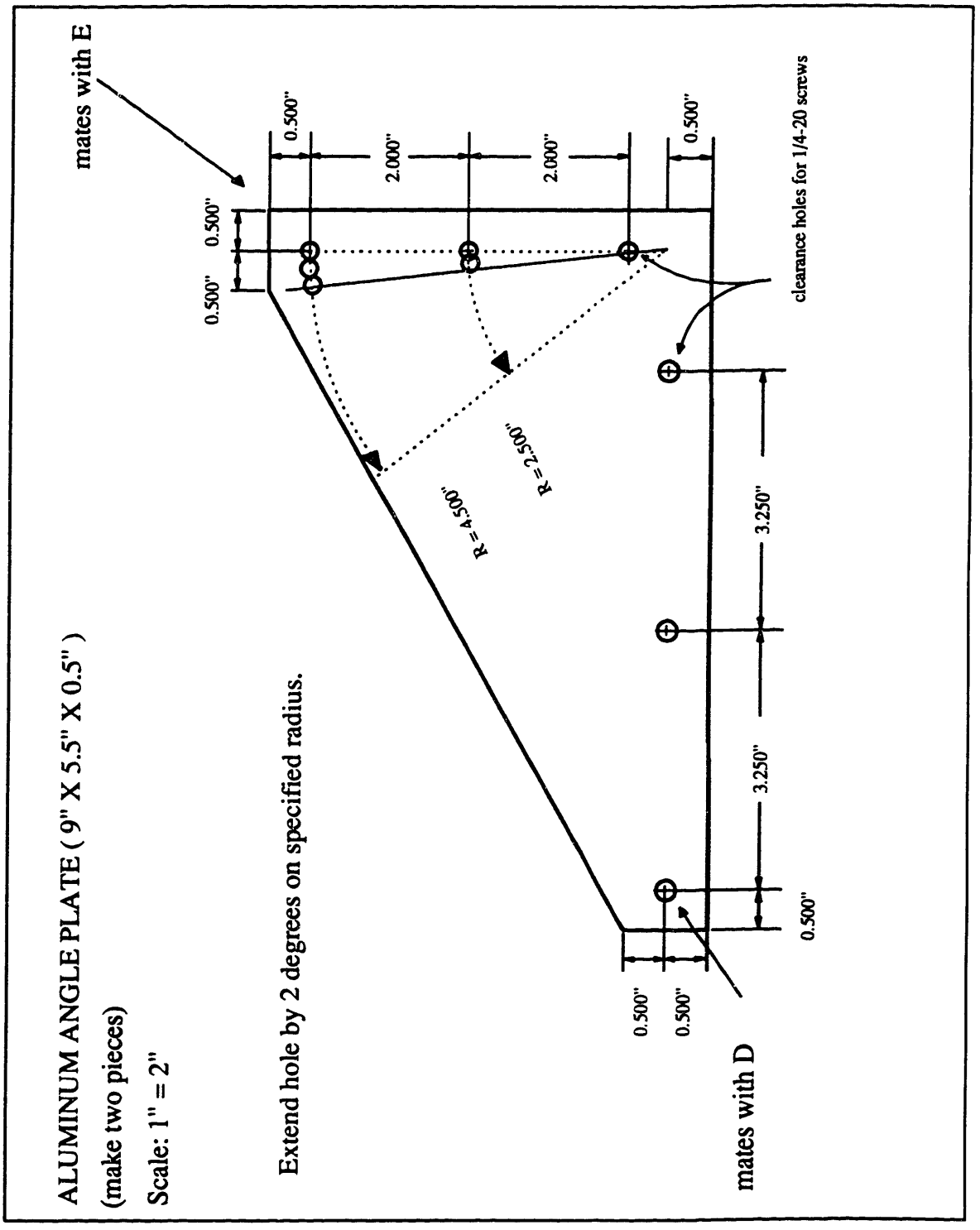
MOVE5B.DRW



REACTOR MOVEMENT ASSEMBLY

(PART F)

MOVE6.DRW

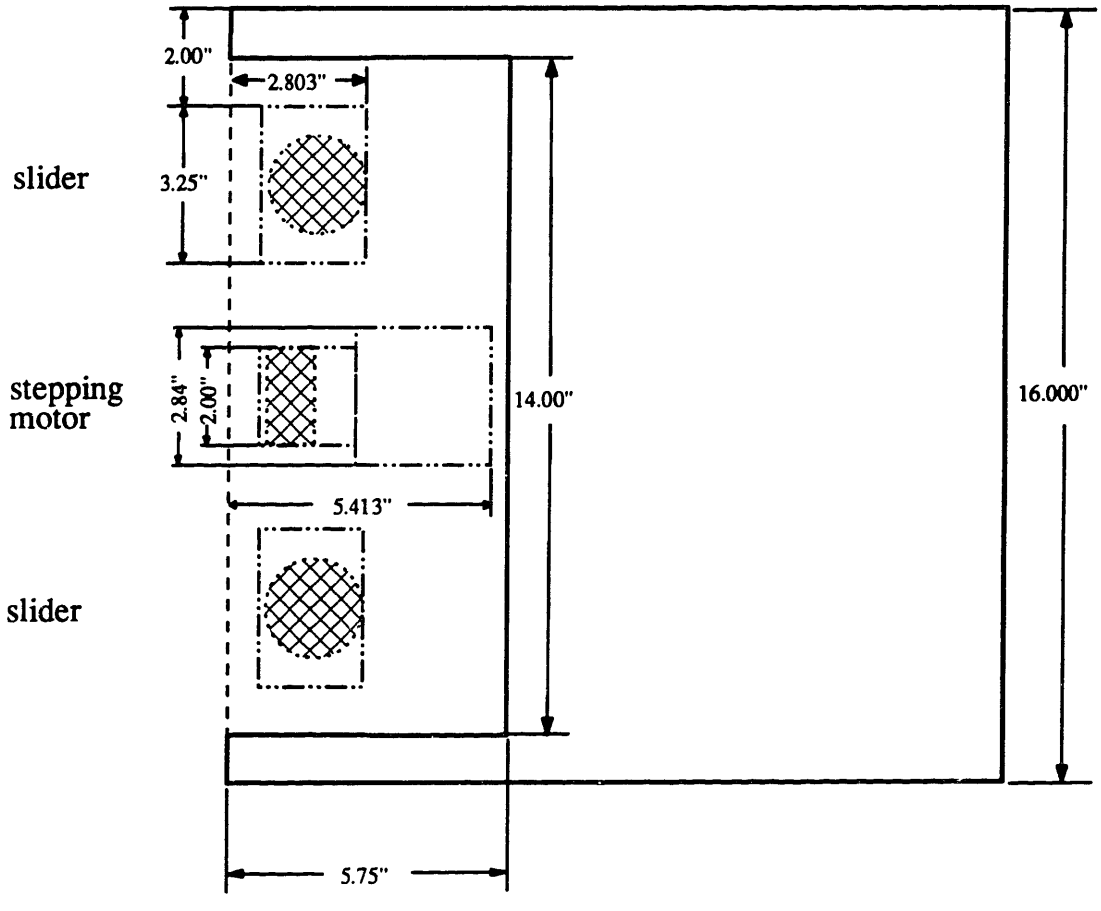


REACTOR MOVEMENT ASSEMBLY

(PART G)

MOVE13A.DRW

ALTERATIONS TO BOTTOM PLATE



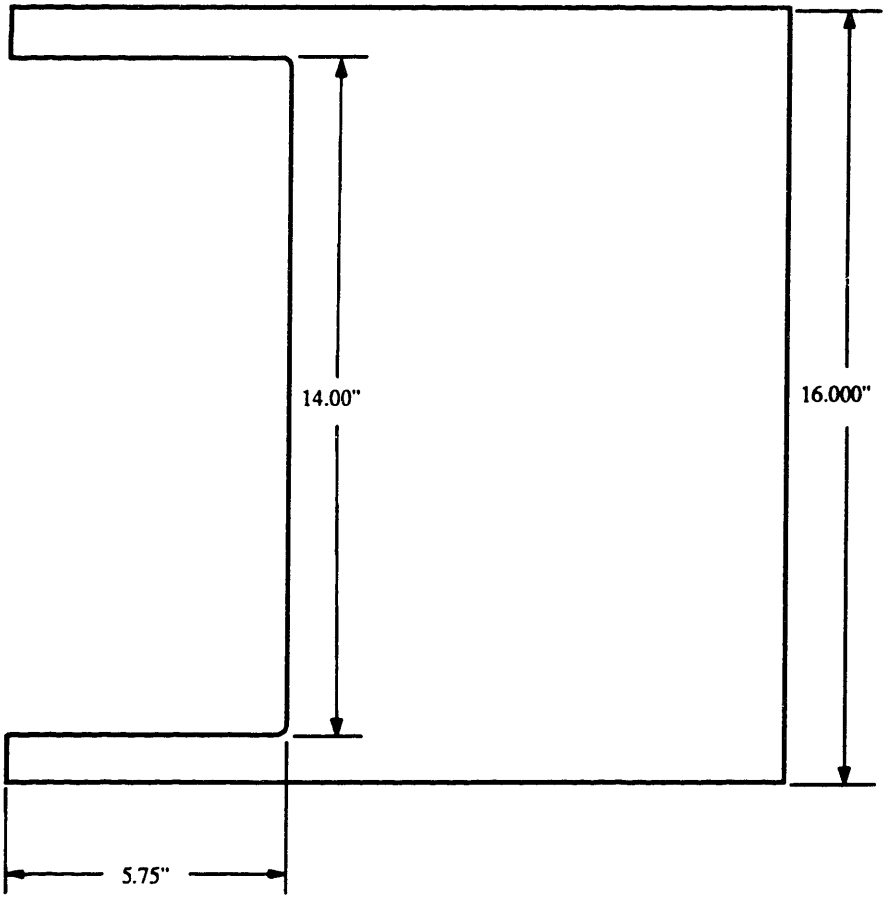
REACTOR MOVEMENT ASSEMBLY

(PART G)

MOVE13B.DRW

ALTERATIONS TO BOTTOM PLATE

1/4" thick aluminum plate

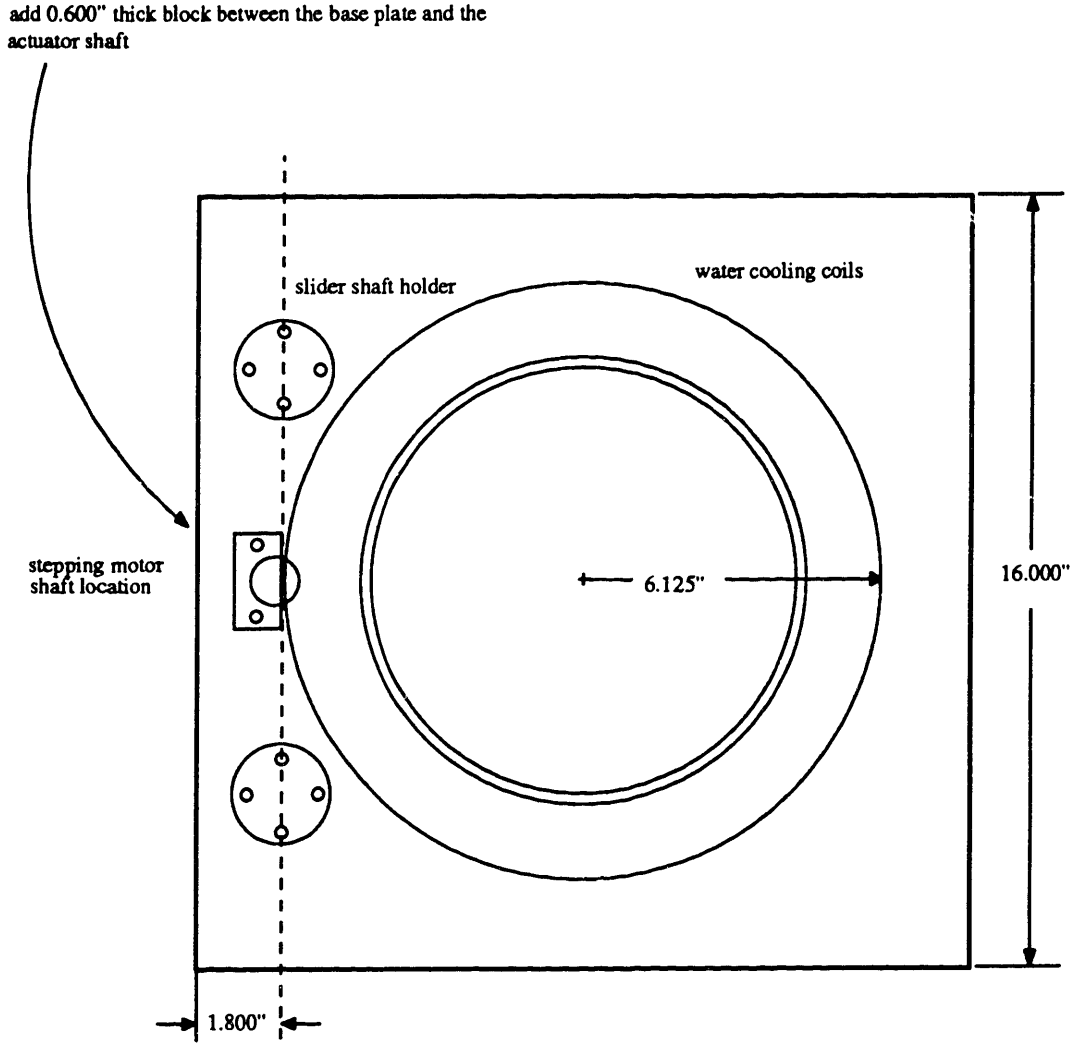


REACTOR MOVEMENT ASSEMBLY

(PART H)

MOVE7A.DRW

ALTERATIONS TO BASE PLATE

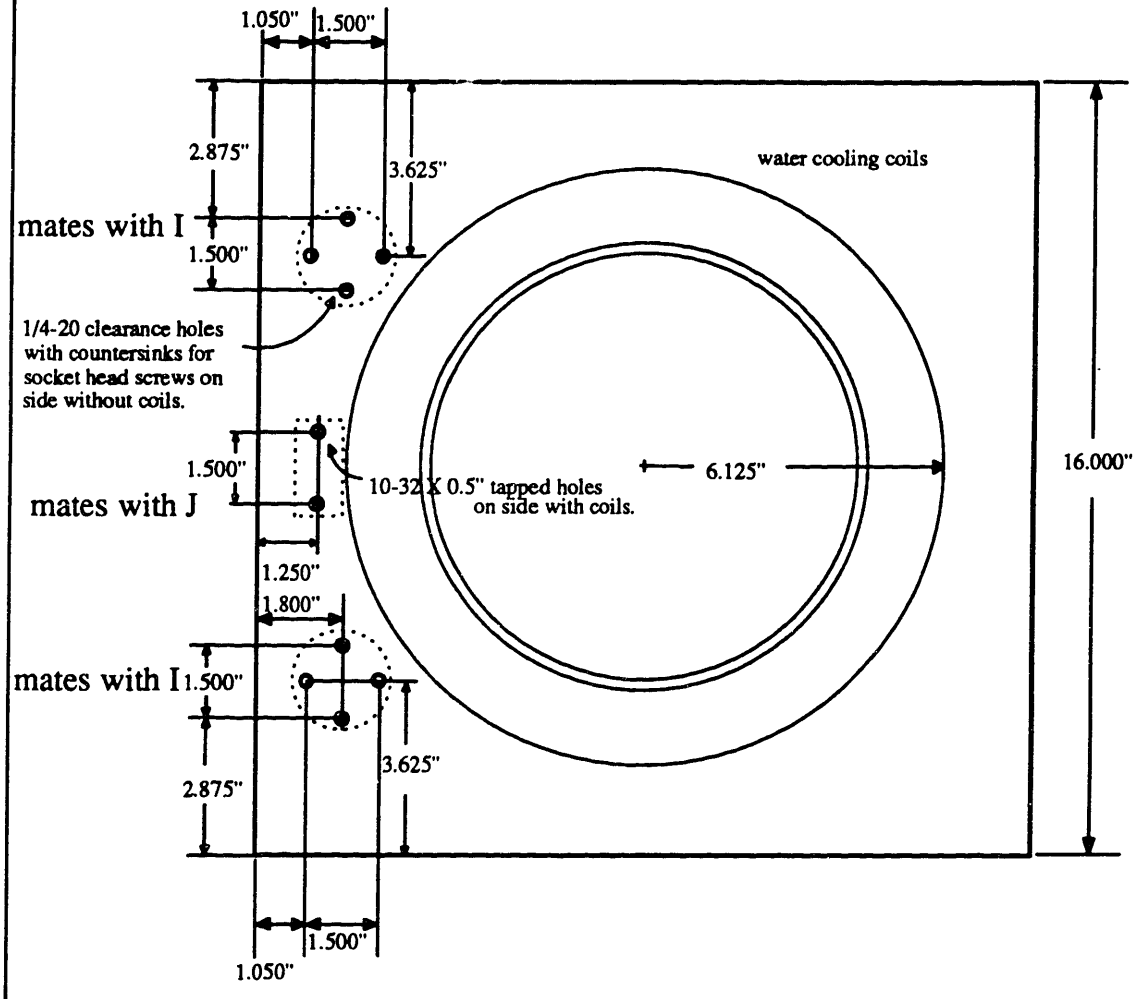


REACTOR MOVEMENT ASSEMBLY

(PART H)

MOVE7B.DRW

ALTERATIONS TO BASE PLATE

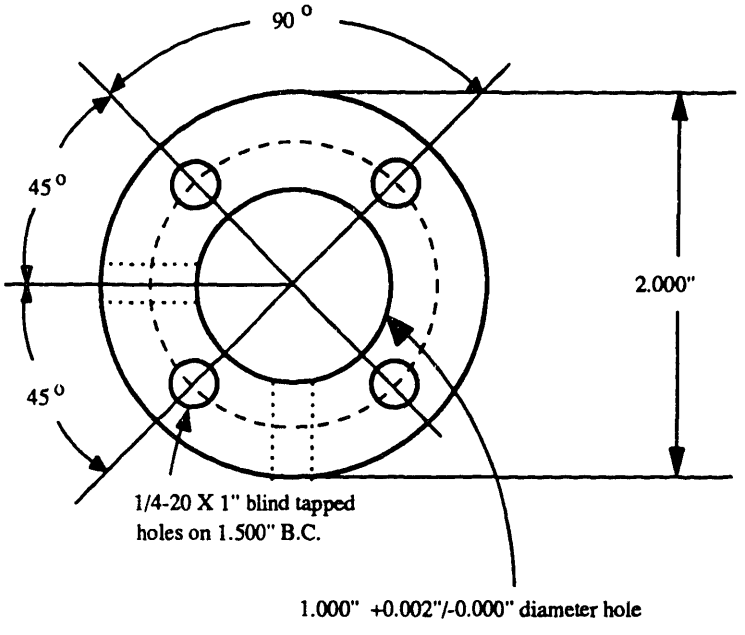


REACTOR MOVEMENT ASSEMBLY

(PART I) MOVE8.DRW

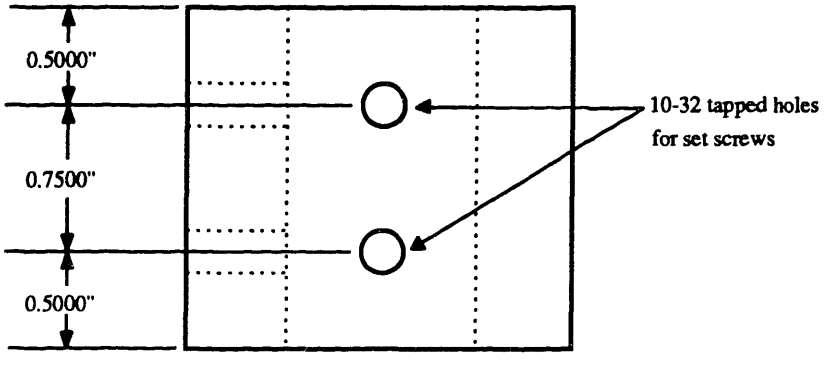
BRASS SLIDER SHAFT HOLDER (make two)

TOP VIEW



mates with H

SIDE VIEW



REACTOR MOVEMENT ASSEMBLY

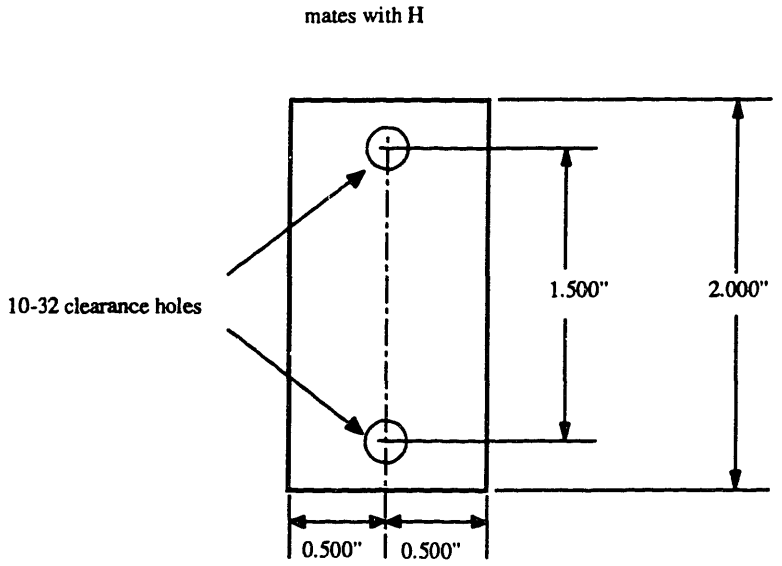
(PART J)

MOVE12.DRW

BLOCK BETWEEN ACTUATOR AND REACTOR BASE

Scale: 2" = 1"

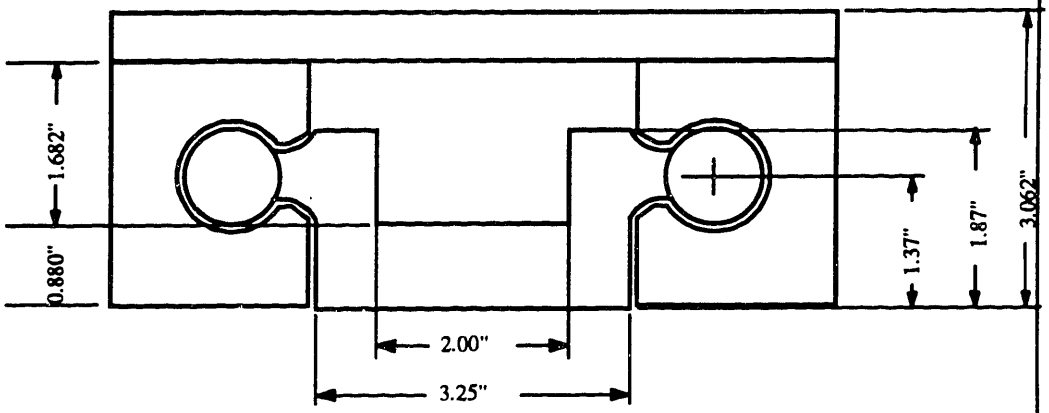
Material: 0.60" thick aluminum

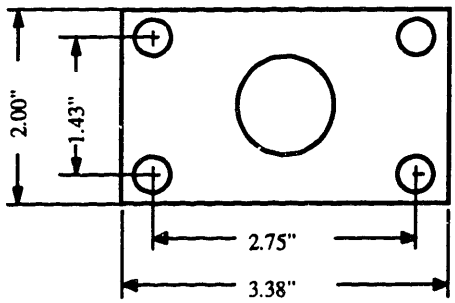


REACTOR MOVEMENT ASSEMBLY

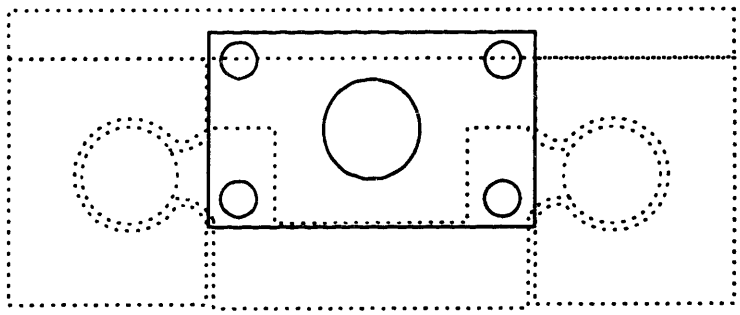
(LINEAR MOTION GUIDE AND ACTUATOR ASSEMBLY) MOVE9A.DRW

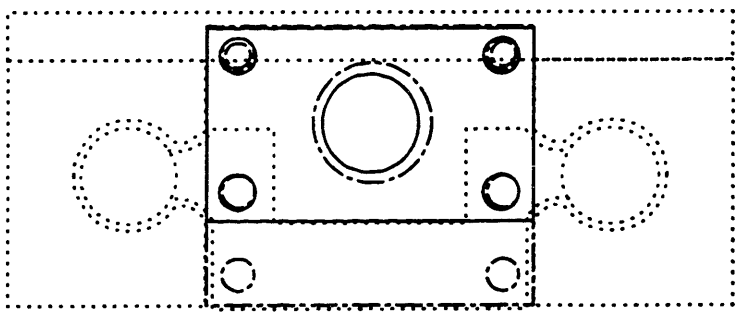
Scale: 1" = 2"





Actuator mounting plate





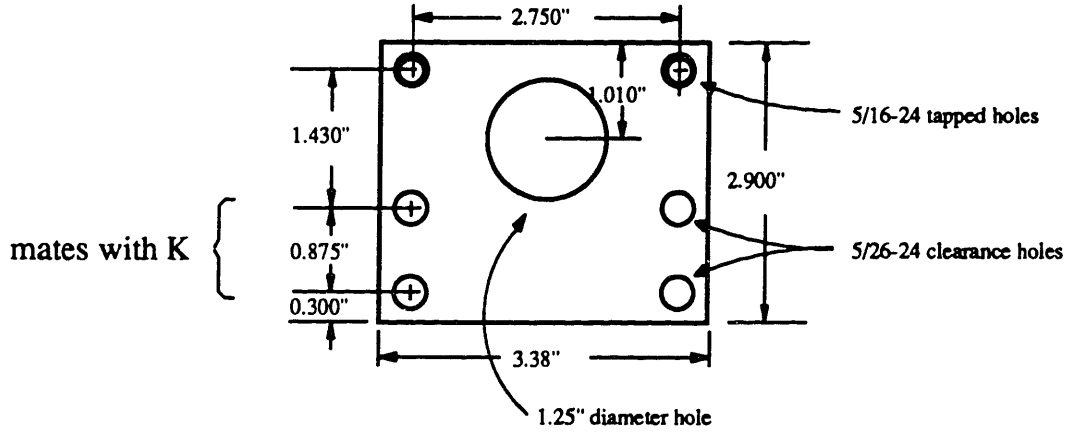
REACTOR MOVEMENT ASSEMBLY

(PART M)

MOVE9B.DRW

CONNECTOR BETWEEN SLIDER AND ACTUATOR (make two)

Material: 1/4" aluminum
Scale: 1" = 2"



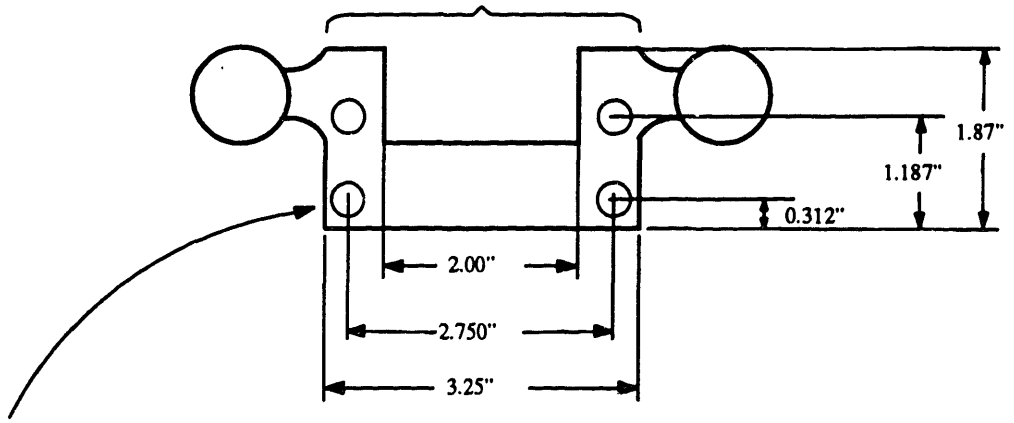
REACTOR MOVEMENT ASSEMBLY

(PART K) MOVE9C.DRW

ALTERATIONS TO SLIDER (alter two pieces)

Scale: 1" = 2"

machine finish this end of slider (the main 3.25" X 1.87" portion)
before drilling holes. Use shafts for vertical and horizontal alignment.



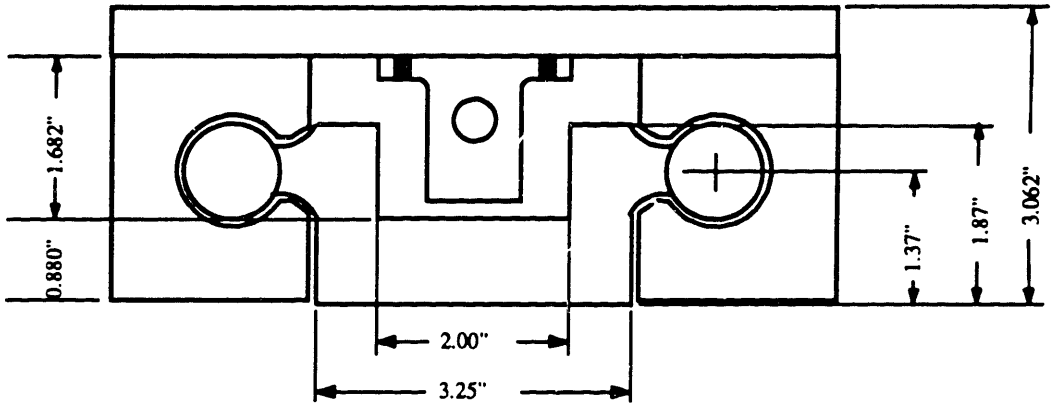
add four 5/16-24 X 1" blind tapped holes

mates with M

REACTOR MOVEMENT ASSEMBLY

(CONNECTION BETWEEN SLIDER AND ACTUATOR

MOVE11A.DRW



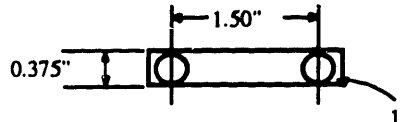
REACTOR MOVEMENT ASSEMBLY

(PART L) MOVE11B.DRW

ACTUATOR SHAFT HOLDER

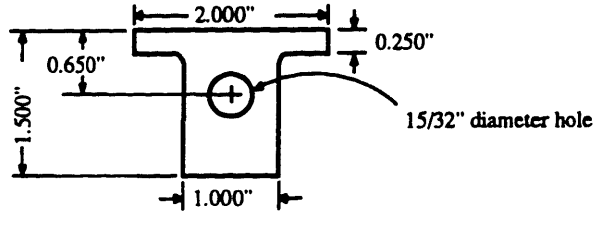
Material: Aluminum
Scale: 1" = 2"
(make two pieces)

Top View



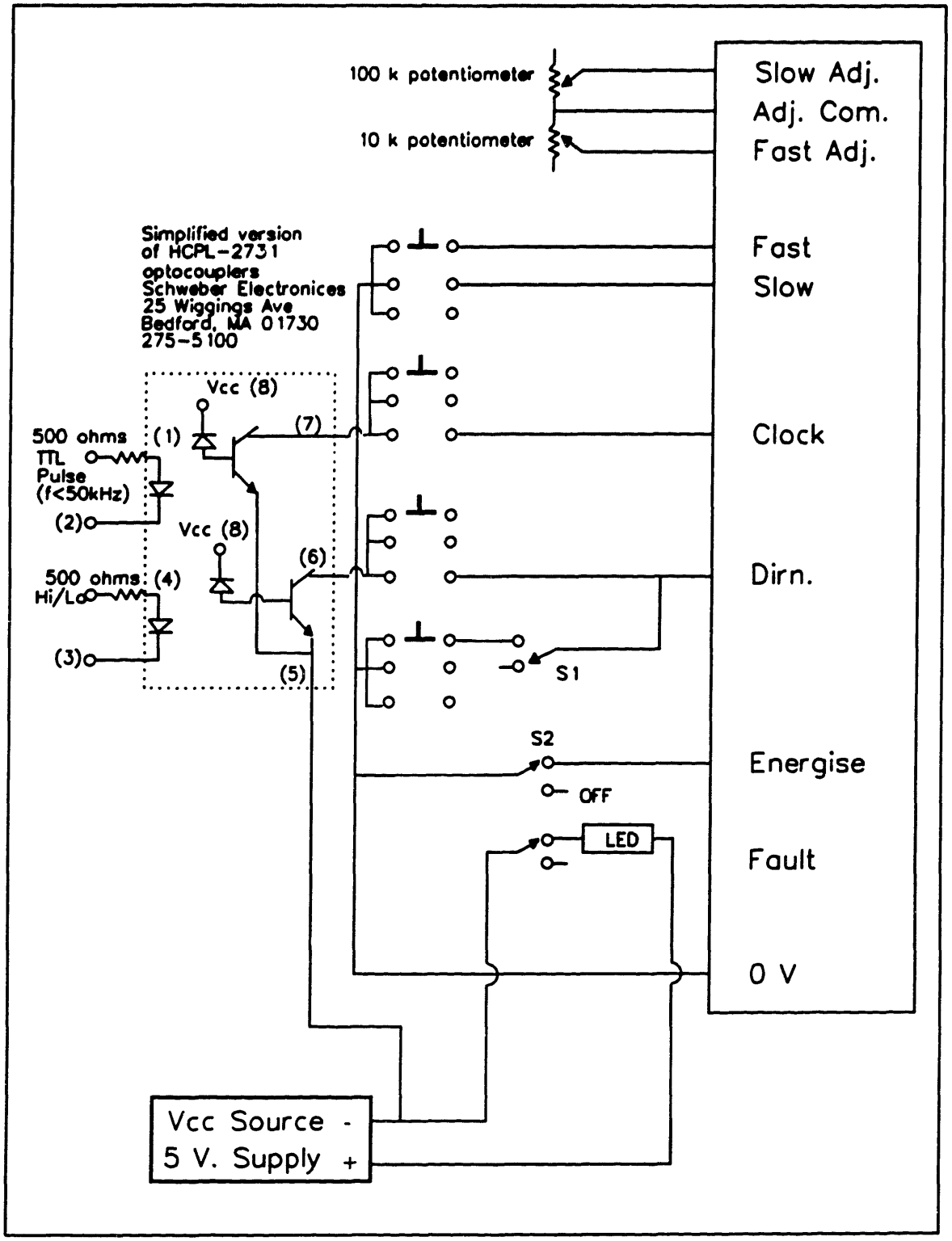
mates with B and C

Side View



ACTUATOR CONTROLS

ACTUATOR.DRW



APPENDIX E

PLASMA CONFINEMENT PARTS

In this appendix are all the parts used confine the plasma. They include teflon rings around the electrodes, teflon holders for the MgF_2 windows to put into the four reactor flanges, and the specifications for the MgF_2 windows.

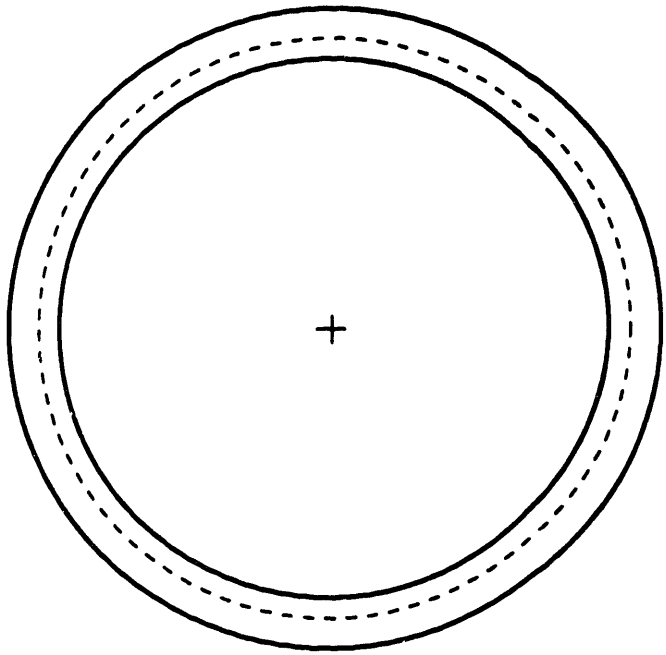
PLASMA CONFINEMENT

Sept. 19, 1990

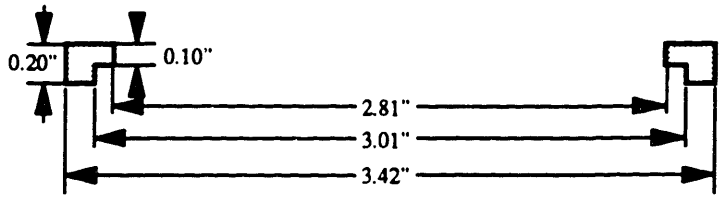
(TEFLON FOR 3"D. BOTTOM ELECTRODE)

TEFLON6.DRW

Top View



Cross Section



Note: Make jig flexible for machining rings of larger outside diameter.

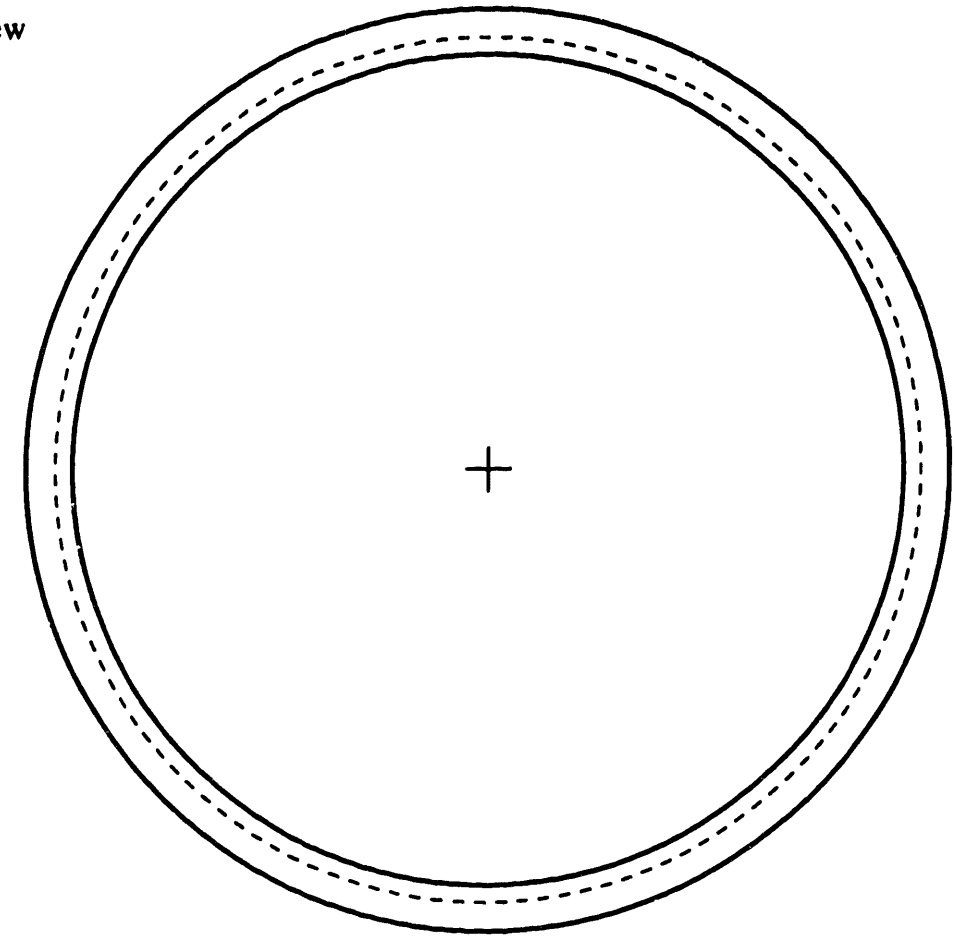
PLASMA CONFINEMENT

Sept. 19, 1990

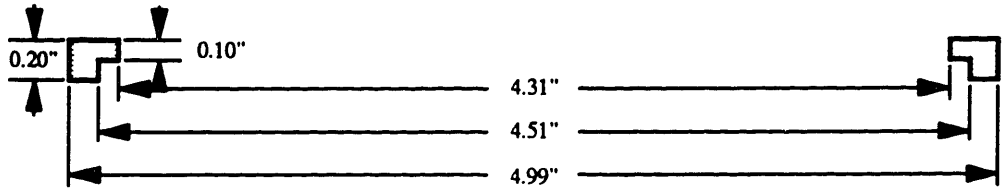
(TEFLON FOR 4.5"D. BOTTOM ELECTRODE)

TEFLON7.DRW

Top View



Cross Section



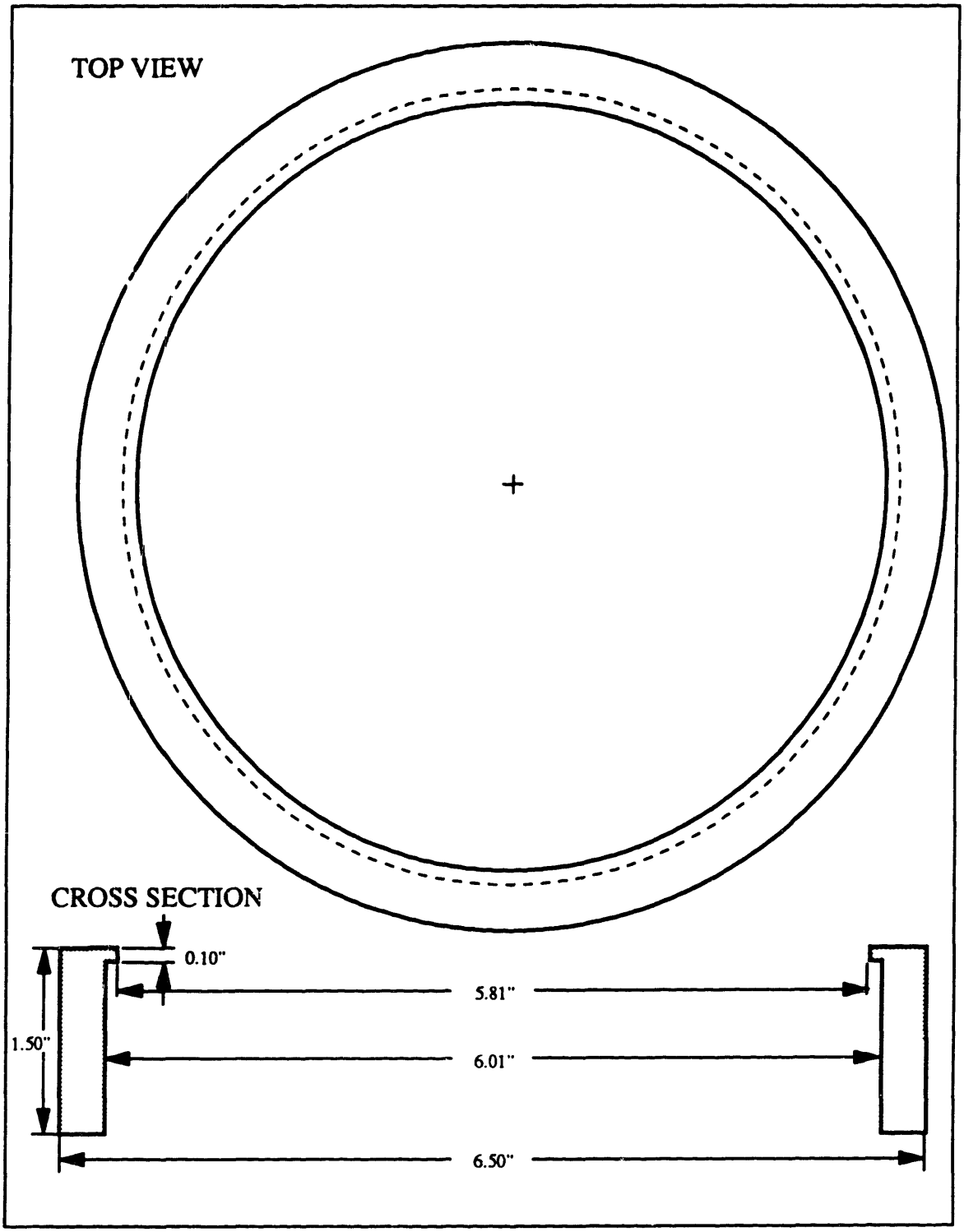
Note: Make jig flexible for machining rings of larger outside diameter.

PLASMA CONFINEMENT

July 2, 1991

(TEFLON FOR 6"D. BOTTOM ELECTRODE)

TEFLON8.DRW



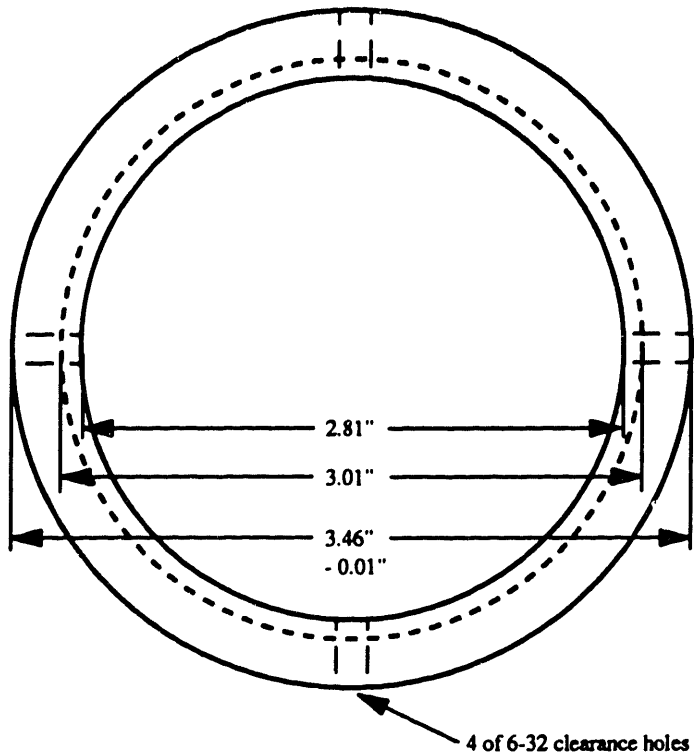
PLASMA CONFINEMENT

November 20, 1990

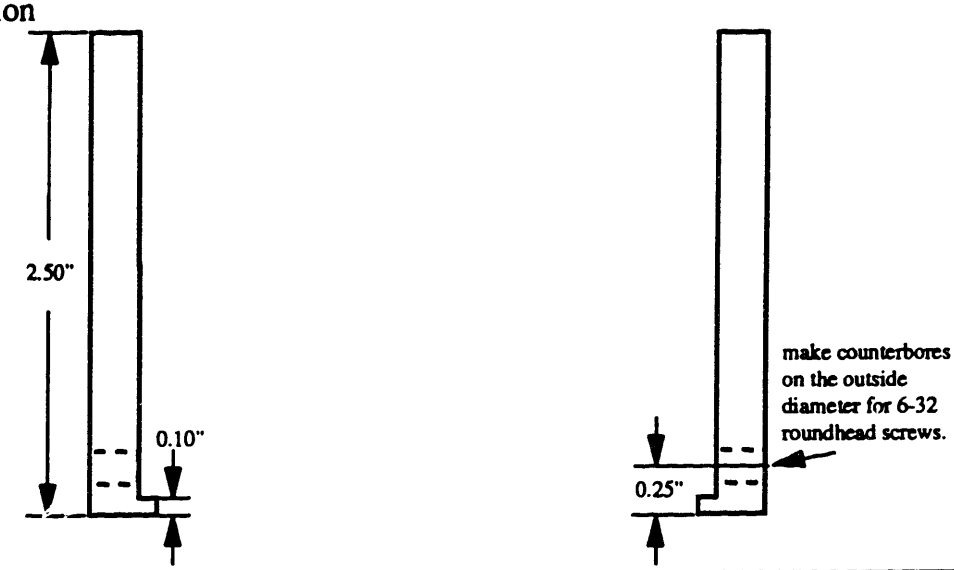
(TEFLON FOR 3"D. TOP ELECTRODE)

TEFLON9.DRW

Top View



Cross Section



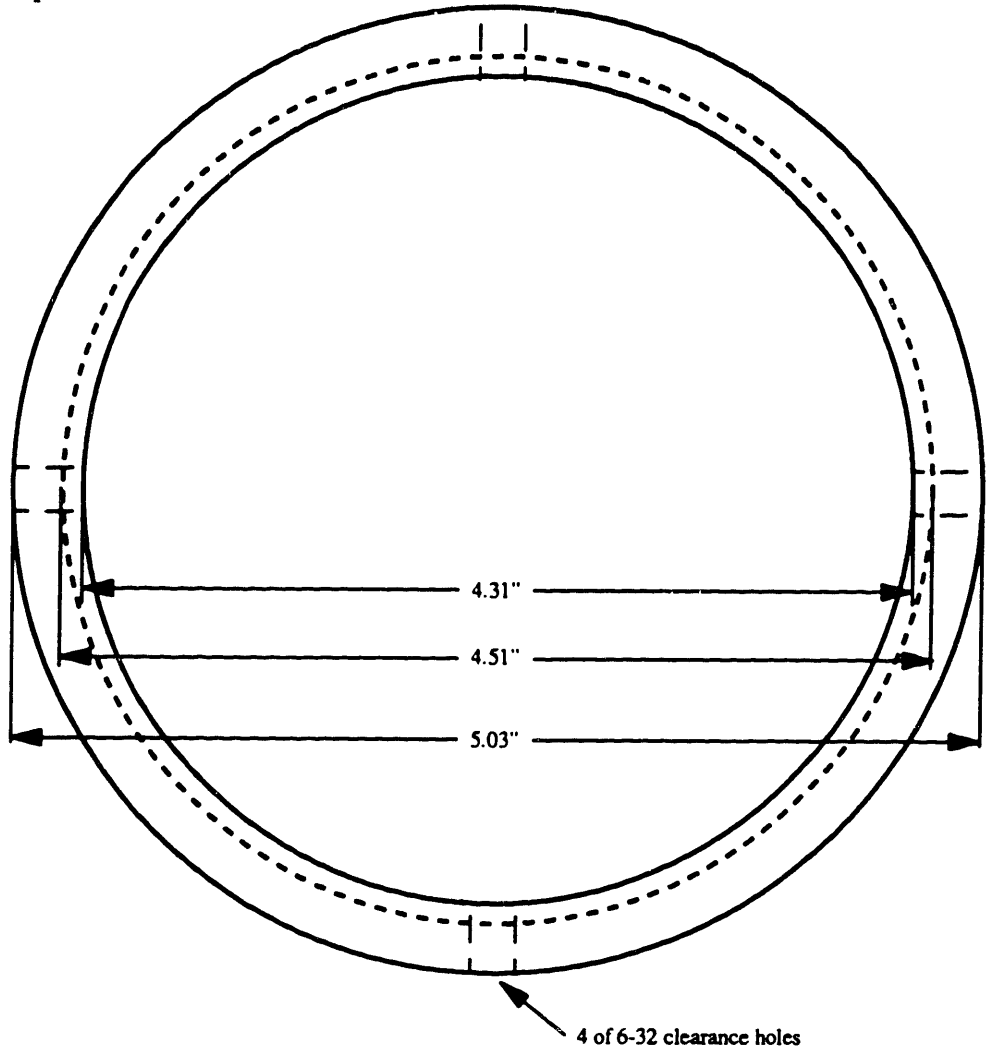
PLASMA CONFINEMENT

Sept. 19, 1990

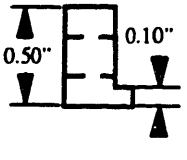
(TEFLON FOR 4.5"D. TOP ELECTRODE)

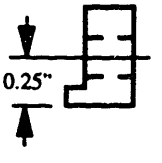
TEFLON10.DRW

Top View



Cross Section





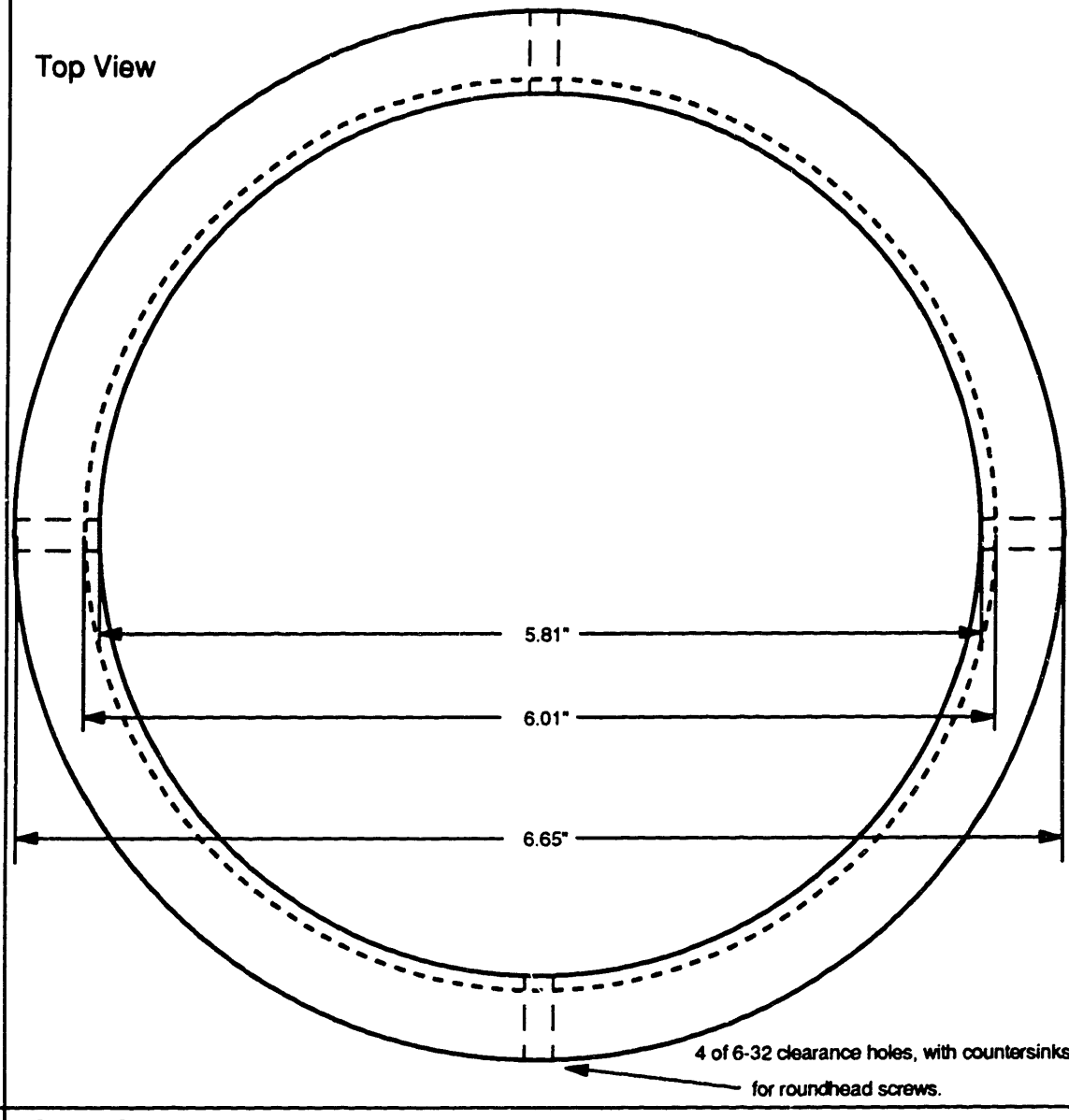
PLASMA CONFINEMENT

Sept. 19, 1990

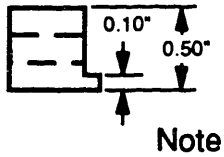
(TEFLON FOR 6"D. TOP ELECTRODE)

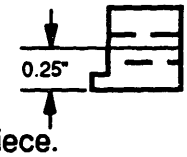
TEFLON11.DRW

Top View



Cross Section





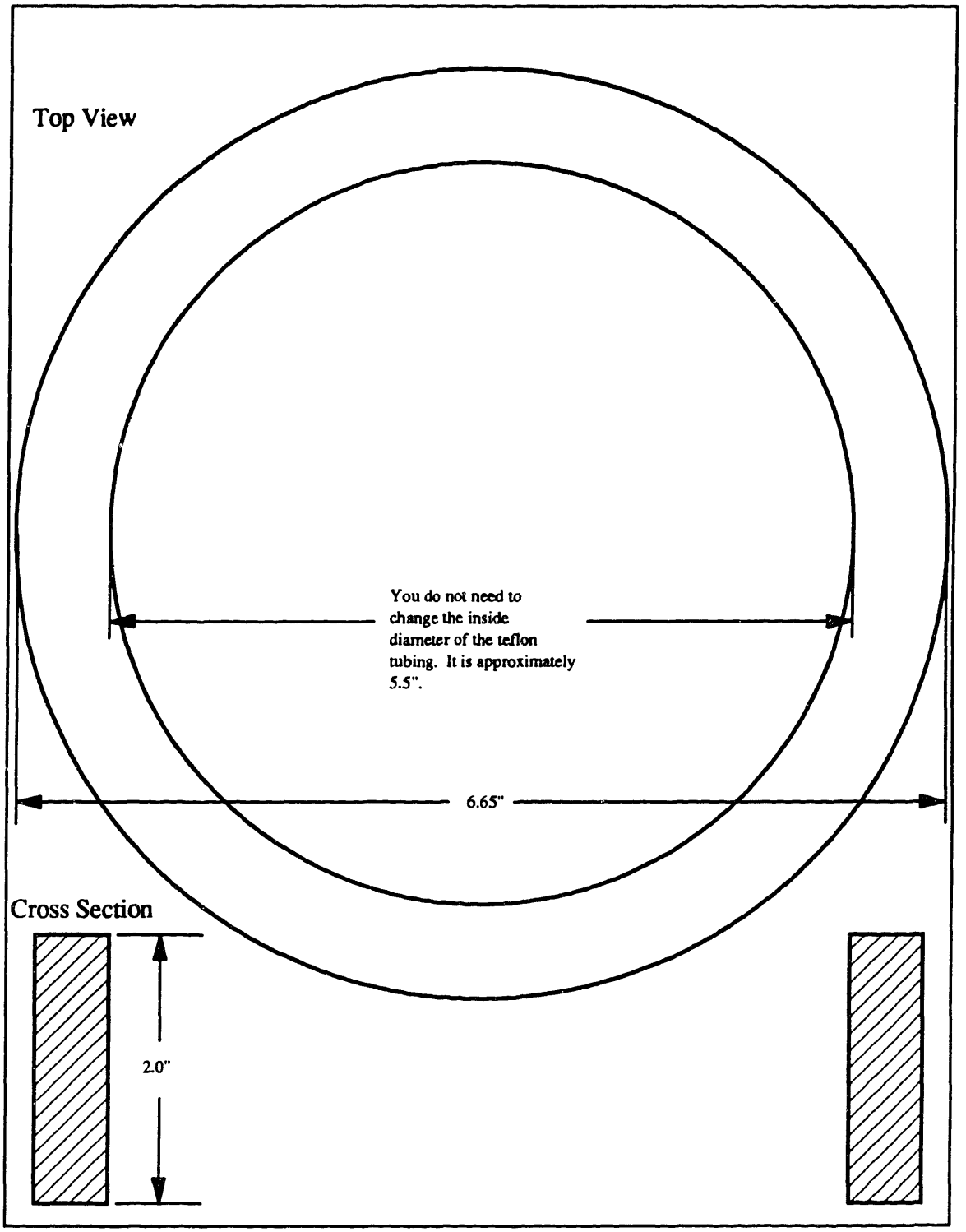
Note: I would like to keep the jig you use to make this piece.

PLASMA CONFINEMENT

November 20, 1990

(TEFLON FOR 6"D. TOP ELECTRODE)

TEFLON12.DRW



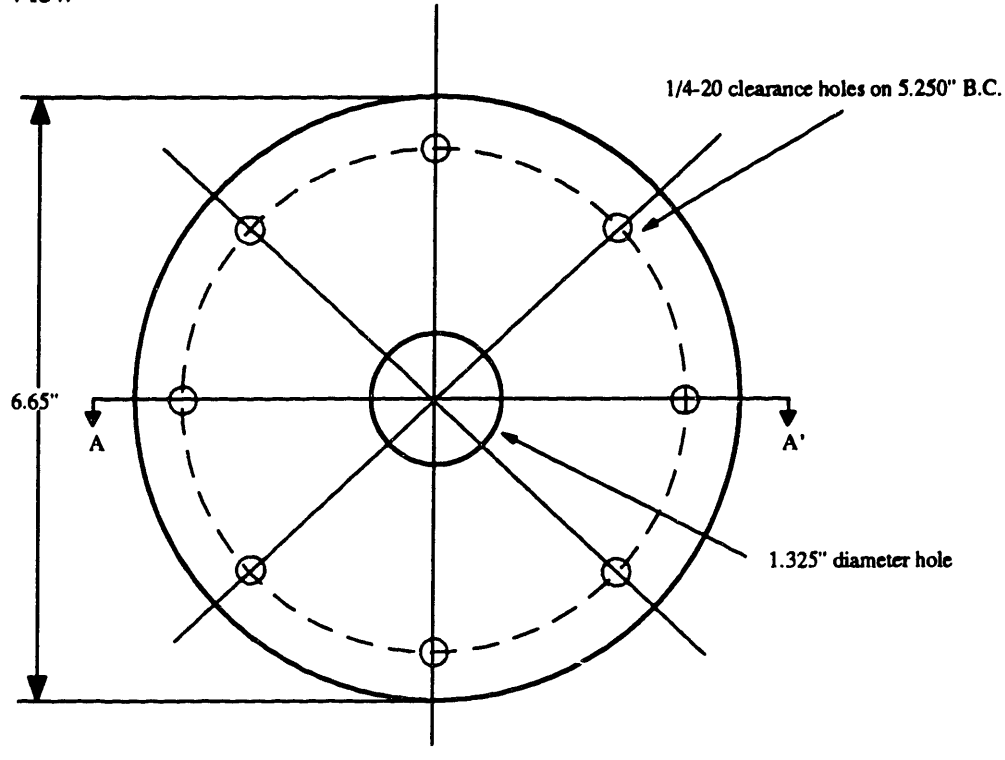
PLASMA CONFINEMENT

July 19, 1991

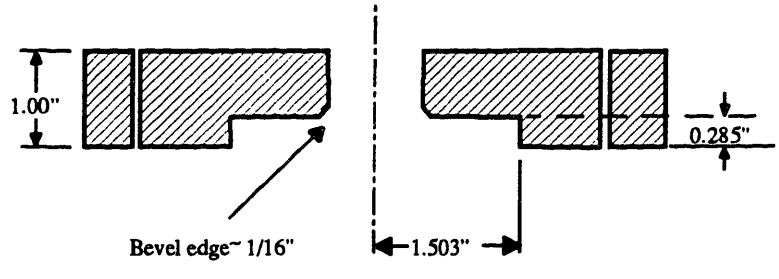
(ASYMMETRIC SYSTEM: UPPER ELECTRODE ASSEMBLY)

TEFLON13.DRW

Top View



Cross Section A - A'

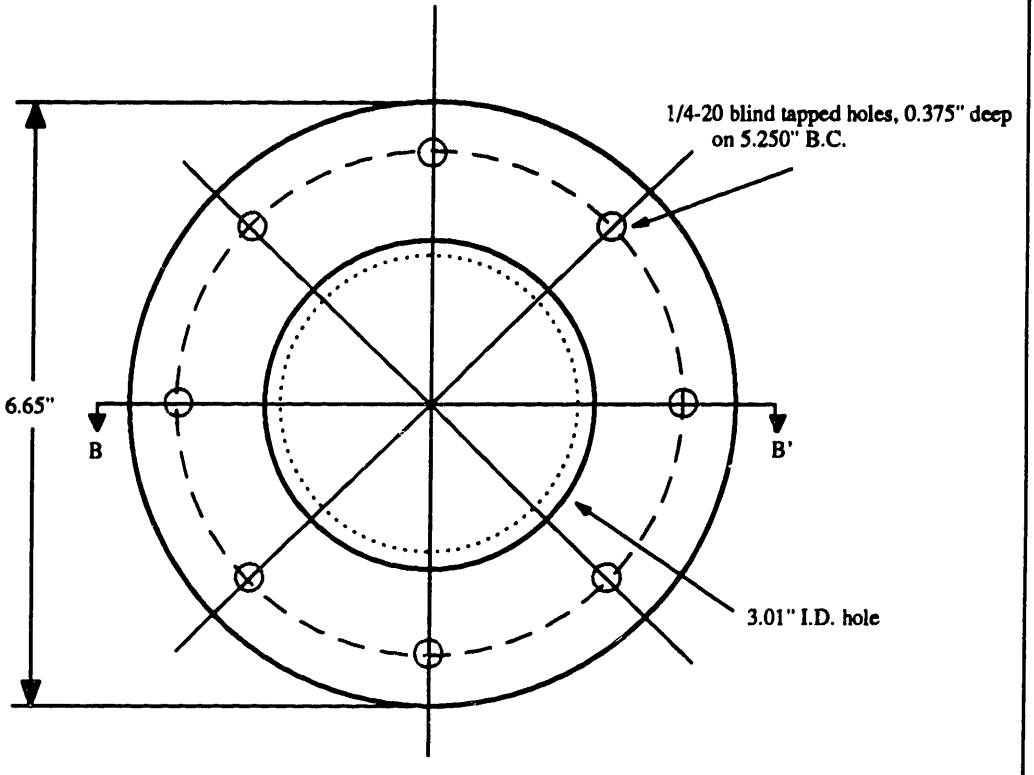


PLASMA CONFINEMENT

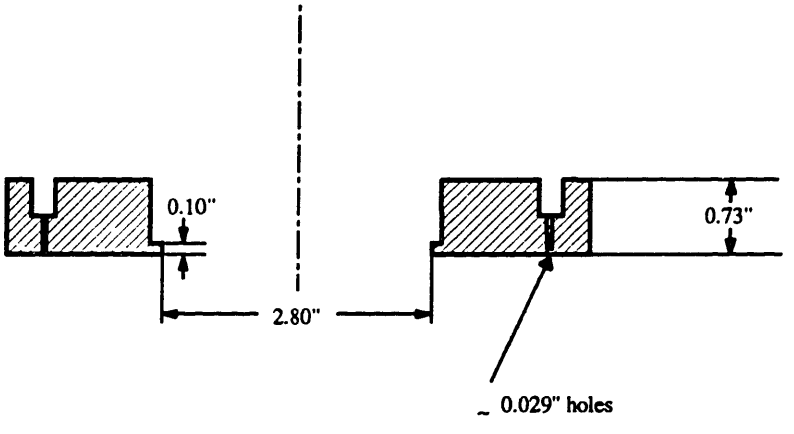
July 19, 1991

(ASYMMETRIC SYSTEM: 3"D. TOP ELECTRODE) TEFLON14.DRW

Top View



Cross Section B - B'



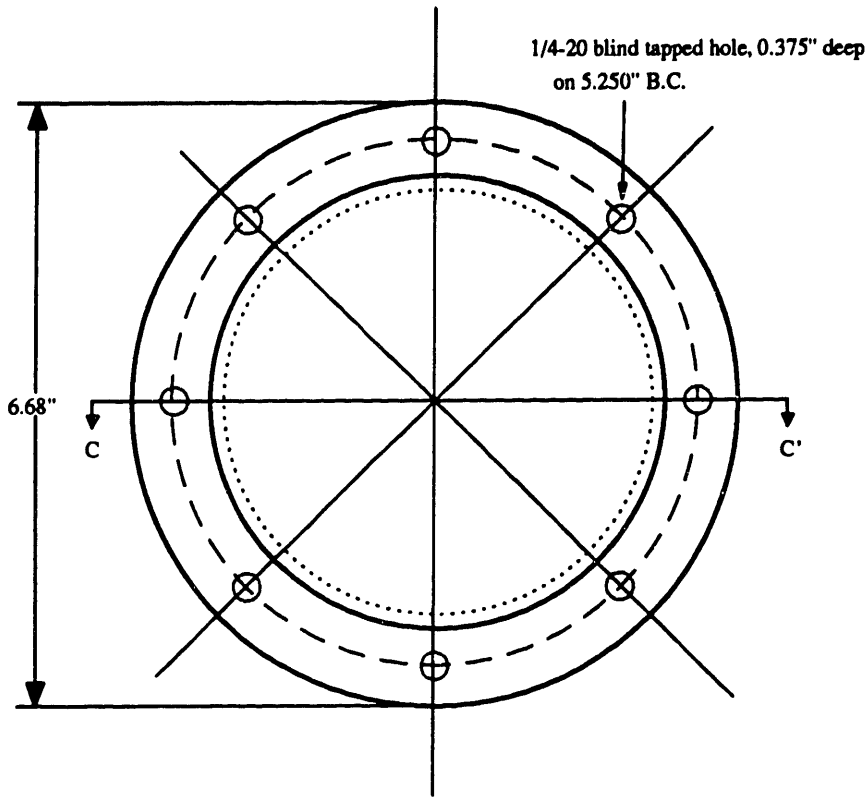
PLASMA CONFINEMENT

July 19, 1991

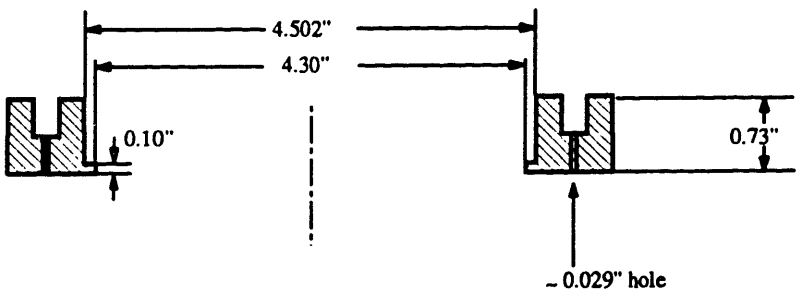
(ASYMMETRIC SYSTEM: 4.5"D. TOP ELECTRODE)

TEFLON15.DRW

Top View



Cross Section C - C'



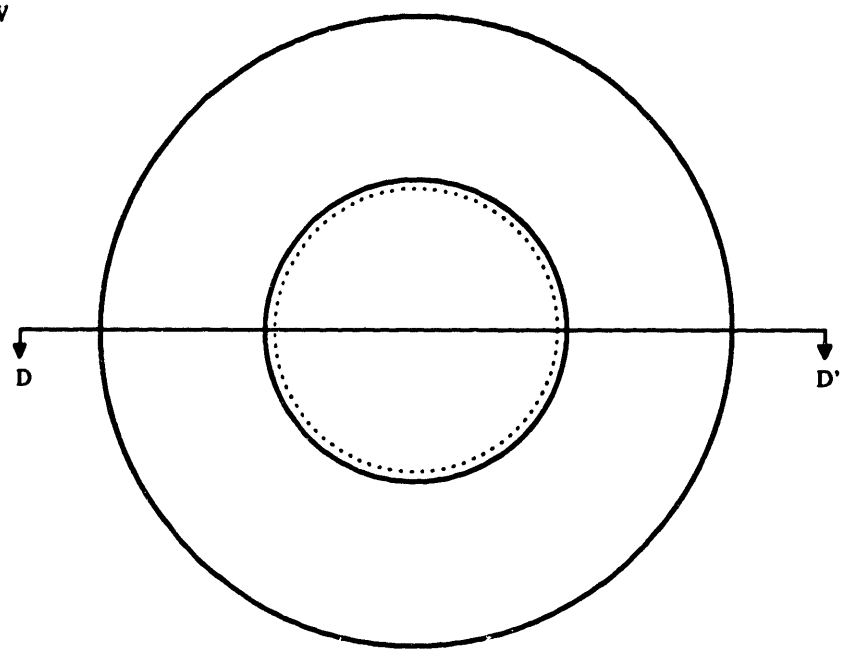
PLASMA CONFINEMENT

July 2, 1991

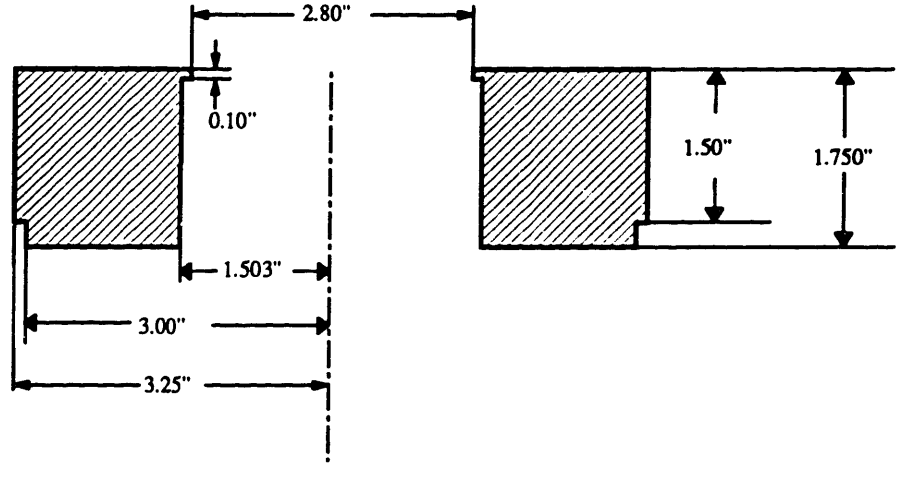
(ASYMMETRIC SYSTEM: 3"D. BOTTOM ELECTRODE)

TEFLON16.DRW

Top View



Cross Section D - D'



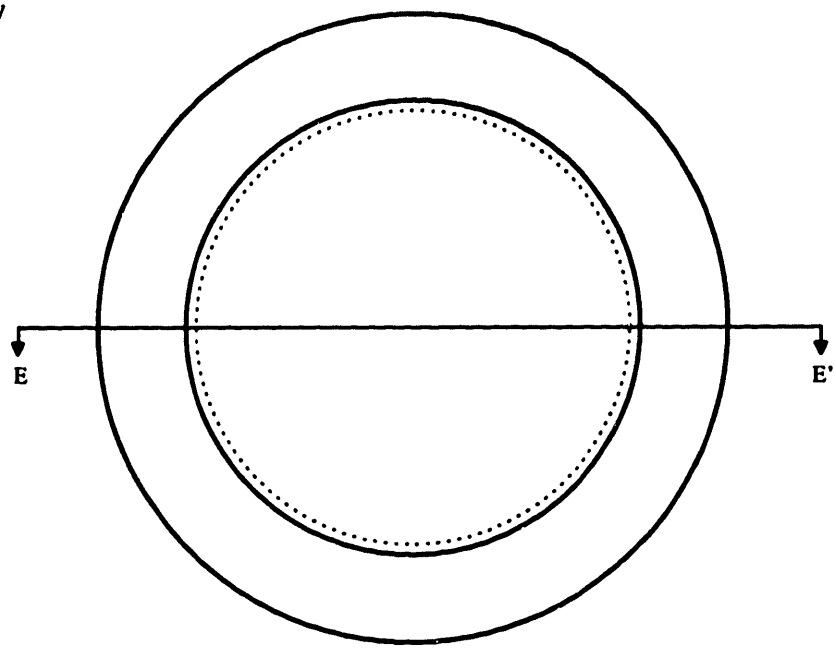
PLASMA CONFINEMENT

July 2, 1991

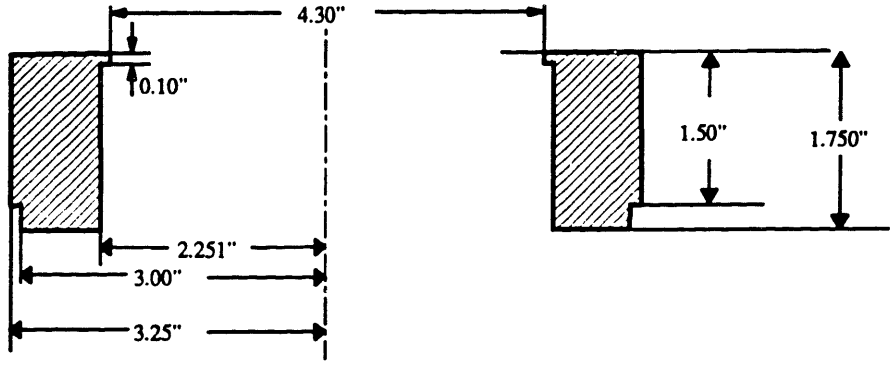
(ASYMMETRIC SYSTEM: 4.5"D. BOTTOM ELECTRODE)

TEFLON17.DRW

Top View



Cross Section E - E'



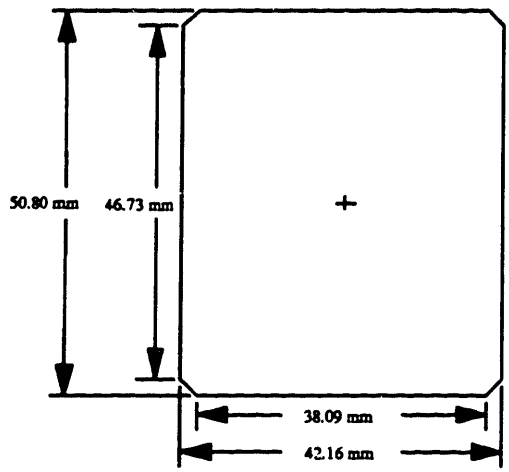
PLASMA CONFINEMENT

(MgF2 WINDOWS)

MGF2.DRW

Janos Technology Inc.
Route 35
Townshend, VT 05353
(802) 365-7714

Joanne Liu
MIT Room 66-219
Building 20, Receiving
18 Vassar St.
Cambridge, MA 02139
(617) 253-6586



Condition of windows before cutting the corners:
1) Material: VUV MgF2, 0.20" X 1.66" X 2.00" +/-0.010"
2) Surface quality: 20/10 polish
3) 3 min parallel
4) 2 wave flatness
5) 80% elliptical aperture
6) original cost per window: $515.00

I have sent you 5 MgF2 windows to have the corners cut to the dimensions specified in the drawing above. These windows were purchased from your company a few months ago. I understand that to cut the corners, you will stack all five windows together, then cut the corners. Once cut, the surface polish on these windows are not guaranteed to be 20/10. The cost of this project according to your quote on January 9, 1991 is $350. Once I receive the windows and find that I would like the windows repolished to a 20/10 surface quality, the cost will be $65 per window or $325 for all five windows.

If any of this information is incorrect or if you have any questions, please call me at 617-253-6586.

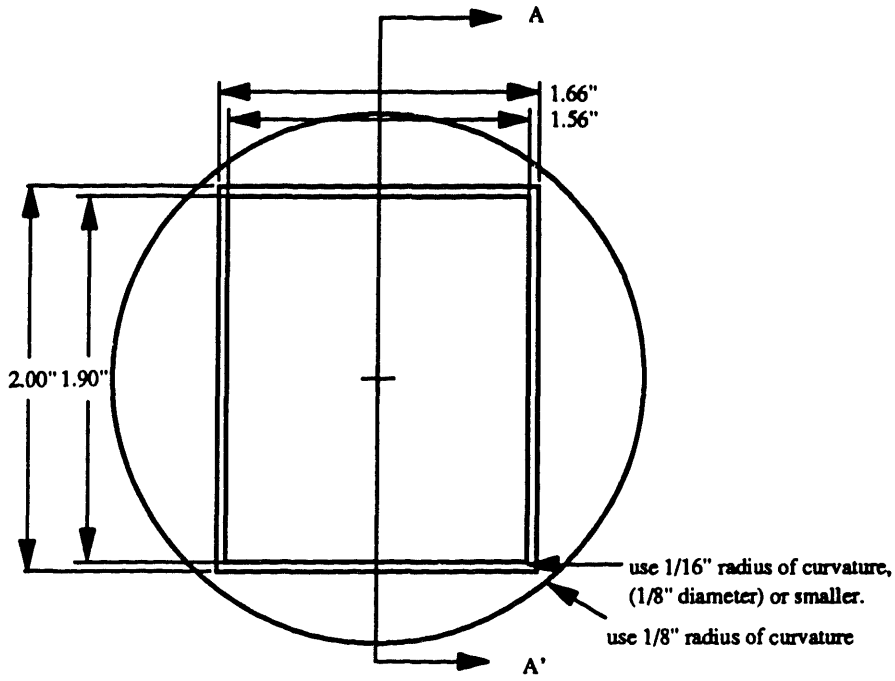
PLASMA CONFINEMENT

Jan. 29, 1991

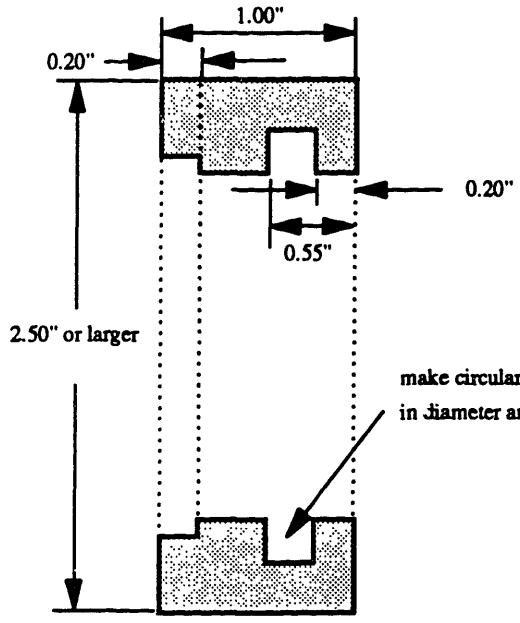
(TEFLON TO HOLD MgF2 WINDOWS)

CIRCLE1.DRW

Front View



A - A' Cross Section



This TEFLON diameter should be 0.02" smaller than the pyrex tube.

This will vary depending on which tube on the pyrex chambers you are making this for.

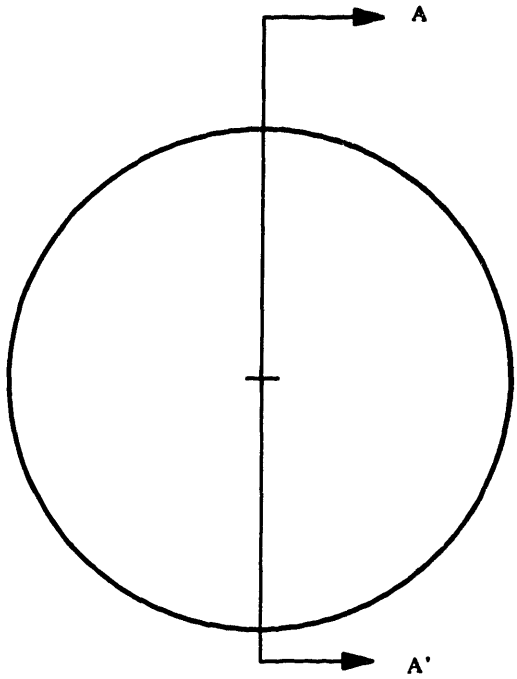
PLASMA CONFINEMENT

November 20, 1990

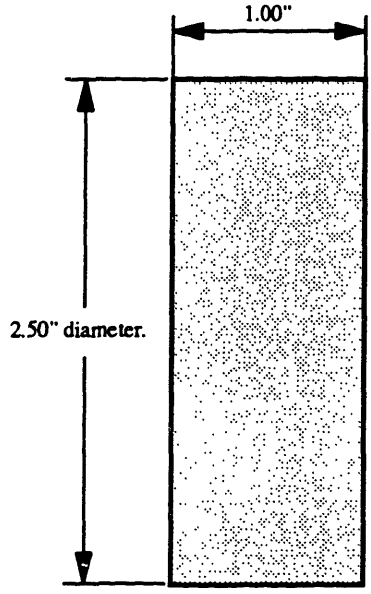
(TEFLON FOR CHAMBER PORTS WITHOUT MgF2 WINDOWS)

CIRCLE2.DRW

Front View



A - A' Cross Section



APPENDIX F

LIST OF EQUIPMENT

1. Equipment for powering the reactor
   - Hewlett Packard 3325B Synthesizer/Function Generator
   - ENI 2100L rf power amplifier
   - ENI 3200L rf power amplifier

2. Measuring power
   - Bird rf directional thruline wattmeter, model 43
   - Ion Physics CM-100-M current probe
   - Tektronix P6015 1000X voltage probe
   - LeCroy 9400 dual 125 MHz Digital Oscilloscope

3. Gas flow meters and controllers
   - Tylan FM360 50 sccm mass flow controller
   - Vacuum General 80-2 throttle valve control module
   - Vacuum General MDV-015S04 motor driven throttle valve
   - MKS 220CA capacitance manometer

4. Vacuum pumps
   - CTI cryogenics Torr 8 pump
   - Balzers TPU050 turbomolecular pump
   - Balzers TCP040 turbomolecular pump controller
   - Leybold-Heraeus TRIVAC D16AC rotary vane pump

5. Ion current measurement equipment
   - Keithley 485-3 Autoranging Picoammeter with IEEE-488 Interface

6. Solenoids and accessories
   - Magnecoil Corp. custom made solenoids (4)
   - Sorensen DCR32-310T5 power supply

7. Optics equipment
   - Instruments SA, Inc. Spectra Link monochromator controller
   - Instruments SA, Inc. HR-640 monochromator
   - Products for Research, Inc. PR-1405RF00ℓ photomultiplier
   - Hamamatsu regulated dc power supply, model C665
   - Stanford Research Systems, Inc. Model SR250 Gated Integrator and Boxcar Averager
   - Stanford Reserach Systems, Inc. Model DG535 Four Channel Digital Delay/Pulse Generator

8. Stepping Motors and other reactor movement parts
    Slo-Syn M093-FD07 stepping motor
    Slo-Syn STM101 translator module with 2 6.5 Ω resistors
    Industrial Devices Corp. linear actuator
        2 of S2-355A-4-MF1-MT1
        1 of S2-355A-4-MS4-FT1
    Industrial Devices Corp. drives and controls
        3 of SPK
    Thomson Industries, Inc. linear motion systems guide
        2 of 2DA-16-JOB L12
        1 of 1AB-16-BOO L12

APPENDIX G

IMPEDANCE CELL DESIGN

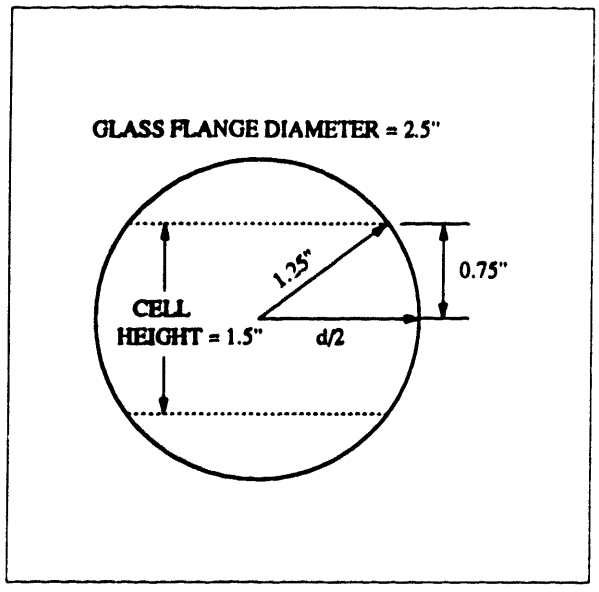
In designing the dimensions of the impedance cells, there are two size limitations to consider. First, the contact area between the impedance cell and the electrodes must be large enough that the capacitance between the contacts is negligible compared to the impedance of the cell. Second, the impedance cell must be small enough to fit through the glass flanges of the reactor. This is necessary to eliminate taking the reactor apart to put each impedance cell between the electrodes.

The minimum diameter of the impedance cell is estimated by requiring the contact impedance between the cell and the electrodes to be less than 2 Ω, or less than 1 Ω per cell-electrode interface. Since the plasma impedance is much larger than 2 Ω and the impedance cells are designed so that their values are near that of the plasma, 2 Ω should be sufficiently small. Two assumptions about the contact area are: the Al_2O_3 layer, which formed on the aluminum electrodes through a hard anodization process, is 2 μm thick; the dielectric constant of Al_2O_3, $\varepsilon_{Al2O3} \approx \varepsilon_{SiO2}$, is $3.8\varepsilon_o$ - $3.9\ \varepsilon_o$ (Wolf and Tauber 1986). For a 2 Ω contact impedance at 13.56 MHz, the minimum cell diameter is:

$$Z = \frac{1}{\omega C} = \frac{\ell}{\omega \varepsilon A}$$

$$A_{min} = \frac{\ell}{\omega \varepsilon Z_{max}} = \frac{2\mu m\ 10^{-4} cm/\mu m}{2\pi\ 13.56 X 10^6\ 3.8(8.85 X 10^{-14} F/cm)\ 1\Omega} = 7 cm^2 \qquad (G.1)$$

$$\text{or} \quad d_{min} = 3cm = 1.2" .$$



The maximum diameter of the impedance cell is limited by the size of the glass flange which has an inner diameter of 2.5". The impedance cell height is approximately 1.5".

$$d_{max} = 2\sqrt{1.25^2 - 0.75^2} = 2" \quad \text{(G.2)}$$

Therefore, the diameter of the cell is chosen to be 2". Figure G.1 shows the cell fabrication design.

Five impedance cells are made with various combinations of resistors and capacitors so that their impedance values are near those of actual plasmas. Each of the cell impedances are measured from 400 kHz to 10 MHz and represented as a resistor and a capacitor in series. These values are listed in Table G.1.

Table G.1: Impedance Cell Values

	R (Ω)	C (pF)
Cell A	10,448	5.05
Cell B	525	2.06
Cell C	∞	91.88
Cell D	4637	6.06
Cell E	5717	3.36

Figure G.2 shows the designed cell impedance values compared to some argon and SF_6 plasma impedances at 13.56 MHz. The range of impedance cell values should cover most of the plasma impedance values.

During the stray impedance calibration with the cells, the electrode area not covered by

the impedance cell also contributes to the measured impedance. This capacitive impedance can be calculated by subtracting the impedance cell area from the electrode area, as shown in Figure G.3, and using the dielectric constant for air. The electrode capacitance is added to the cell impedance values calculated from Table G.1. Therefore, during the stray impedance calibration procedure, both the cell and electrode impedances are not included in the stray impedance values. This means that the de-embedded voltage and current, calculated using the stray impedance net work, will include the electrode impedance. The electrode impedance can easily be calculated from the area and separation of the electrodes used when running the plasma. Subtracting the electrode impedance from the de-embedded impedance yields the plasma impedance.

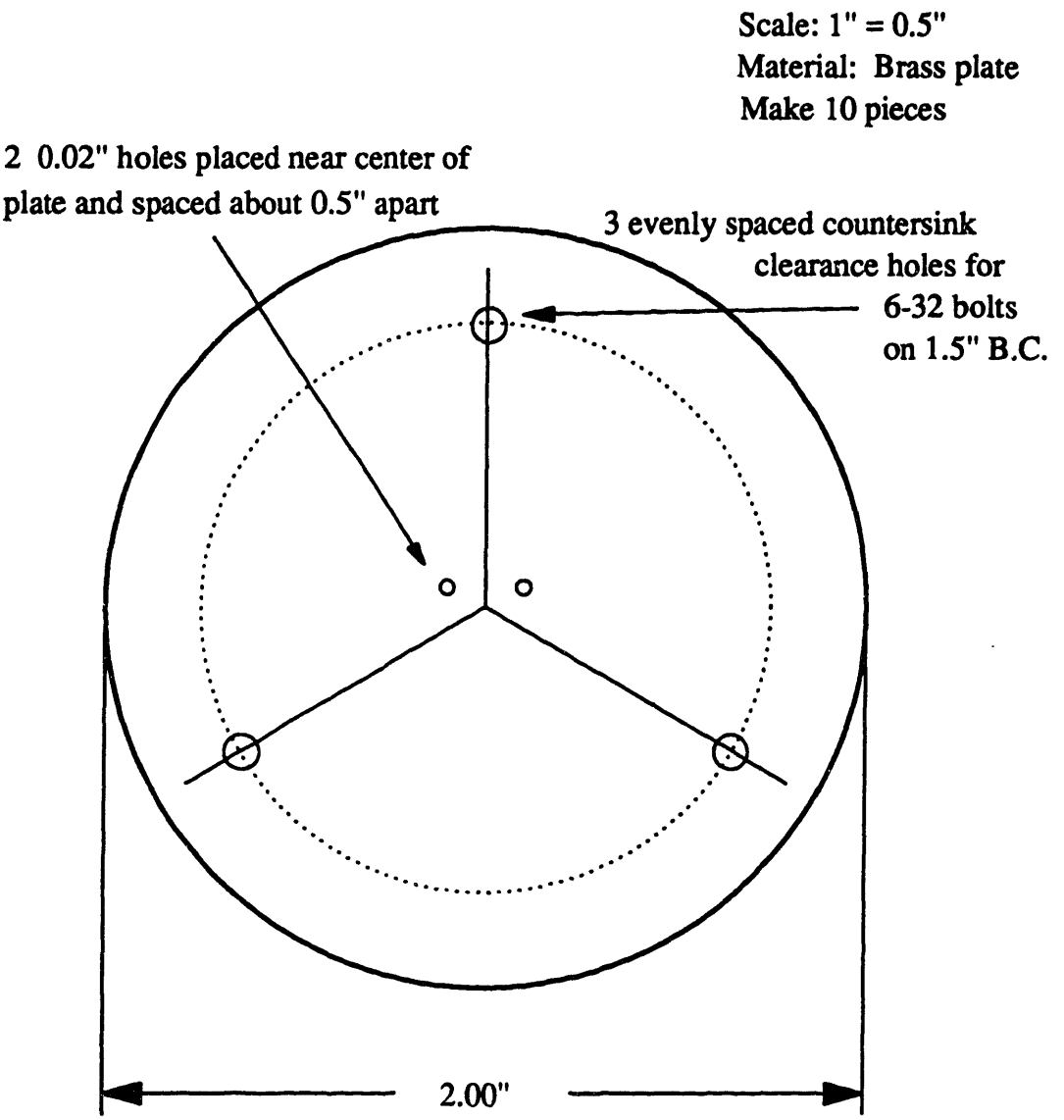


Figure G.1: Fabrication drawing for impedance cell plates. Each pair of brass plates are connected by three 0.75" long ceramic spacers. Capacitors and resistors are soldered in between the plates.

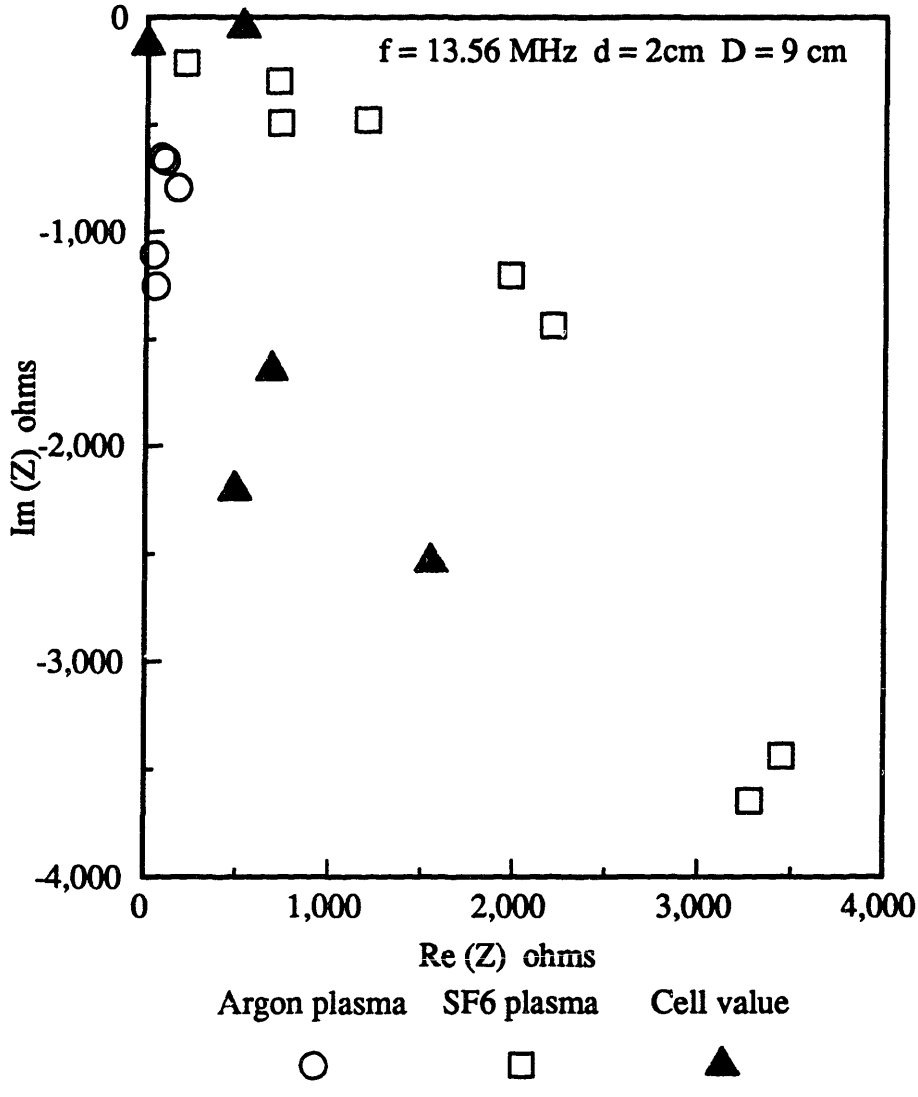
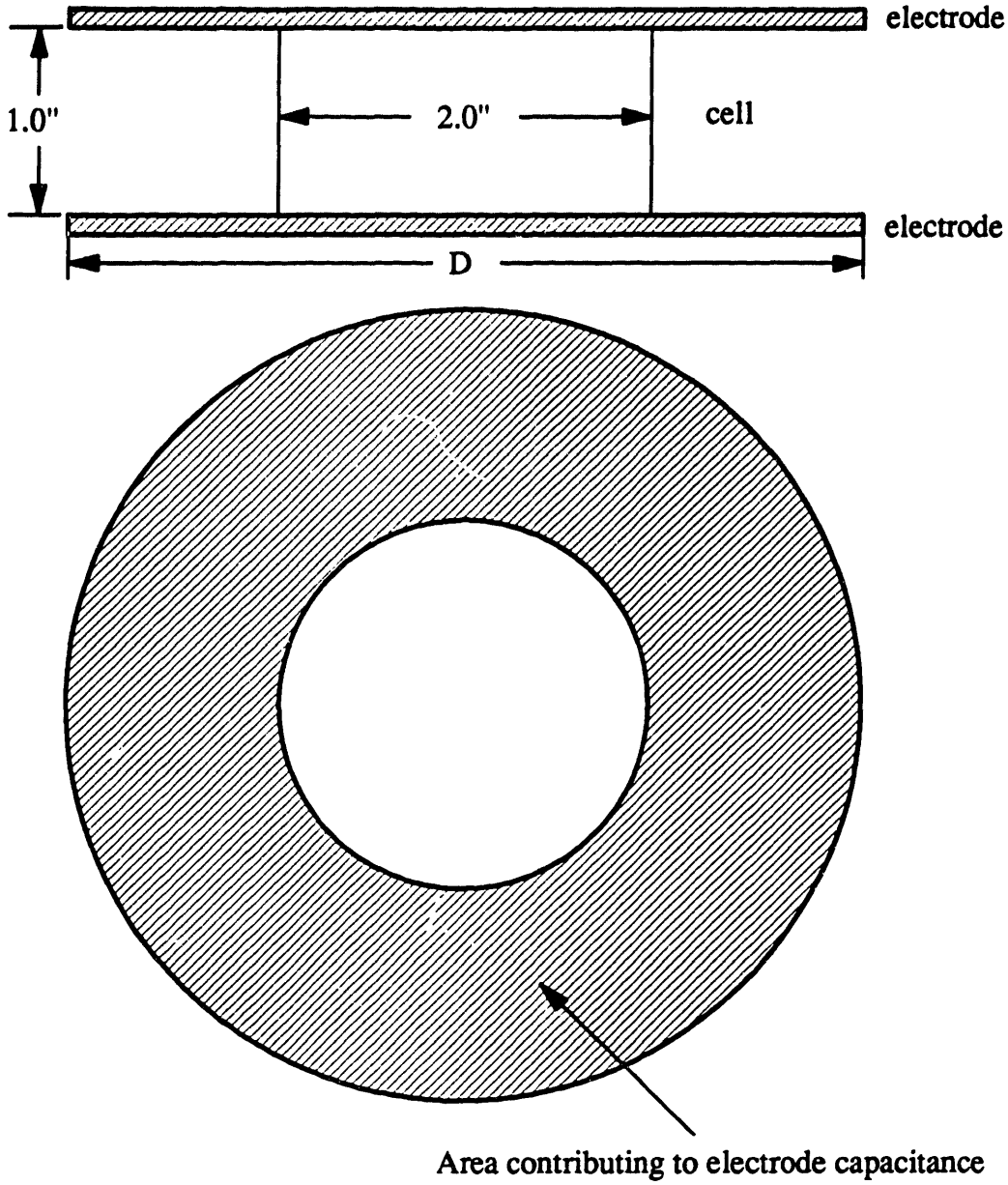


Figure G.2: Some argon and SF6 plasma impedance values plotted along with the impedance cell values. This is a check to make sure that the designed cell values are in the same range as the actual plasma values.



Impedance values for different electrodes:

D = 7.6 cm	Z = - 17313i
D = 11.4 cm	Z = - 4588i
D = 15.2 cm	Z = - 2243i

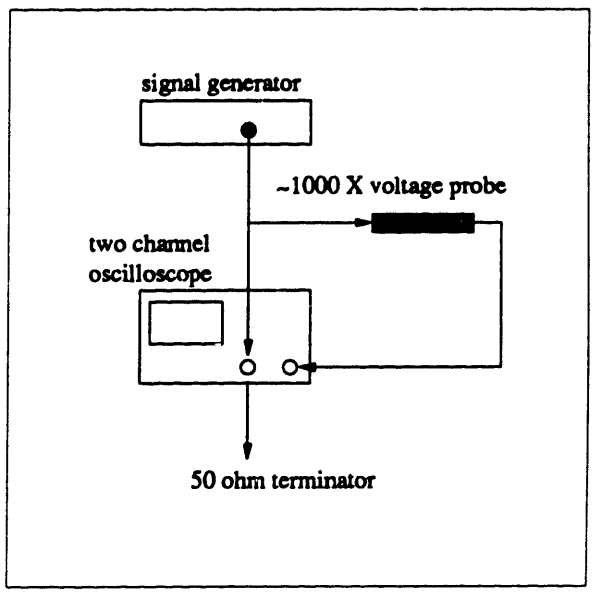
Figure G.3: Figure showing how electrode capacitance is included with the cell impedance. Since the cell and electrode impedance are in parallel, when electrode impedance is much larger than the cell impedance, then the total impedance is equal to the cell impedance.

APPENDIX H

VOLTAGE PROBE CALIBRATION

The voltage probe was first calibrated at 1 kHz using a square wave, and at higher frequencies, using both a square wave and a sine wave. In the course of the calibration, it was discovered that the probe calibration factor was not 1000 X.

To find the calibration factor for the voltage probe, a set-up schematically shown in the adjacent figure was used to compare the input signal and the probe output. Only input signal frequencies from 13.6 to 14.3 MHz were used. Comparing the magnitude of the input voltage to and output voltage from the voltage probe on a two channel oscilloscope, the following calibration factor was found as a function of frequency.



$$\text{Factor} = 48.18\,f + 447.6 \qquad \text{(H.1)}$$

where f = MHz

This equation is only valid in the range of frequencies used to calibrate the probe. At 13.56 MHz, the voltage probe calibration factor is 1105.3.

APPENDIX I

CONSIDERATIONS OF ION MEASUREMENT TECHNIQUE

This appendix contains correction factors needed for the measured ion angle distribution (IAD) and also discusses the various factors that can affect the ion trajectory as it passes through the sampling orifice to the collecting plate of the ion analyzer.

I.1 Orifice Area Correction

The effective ion collection area is smaller for ions incident at an angle θ than those incident normal to the surface because of the finite thickness, t, of the orifice. It is necessary to calculate the effective area so that the flux of ions at various incident angles in the IAD can be normalized to the same collection area. Figure I.1 (a) and (b) show the view of an orifice for an ion with normal incidence and incidence at an angle θ, respectively. The effective area, A_{eff}, is the projected orifice area, A′, minus the area blocked by the orifice wall, W′ and 2·S′.

$$A_{eff} = A' - W' - 2 \cdot S' \tag{I.1}$$

Expressions for t′ and d in Figure I.1 (c), written in terms of the orifice dimensions and ion incident angle, are needed to find the area blocked by the orifice wall. First, the projected orifice thickness, t′, viewed from an angle θ is

$$t' = t \sin \theta \ . \tag{I.2}$$

Comparing Figures I.1 (a) and (b), t′ is also the projected length of 2·ℓ,

$$t' = (2 \cdot \ell) \cos \theta \ . \tag{I.3}$$

Equating equations (I.2) and (I.3) gives

$$\ell = (t/2) \tan \theta \ . \tag{I.4}$$

From Figure I.1 (a),

$$d = (R^2 - \ell^2)^{1/2} \ . \tag{I.5}$$

Thus **d** can now be written in terms of the orifice radius and thickness by substituting equation (I.4) into (I.5), giving

$$d = [R^2 - (t/2)^2 \tan^2\theta]^{1/2} \ . \tag{I.6}$$

The wall area, W′, is equal to the product of t′ and 2d as seen in Figure I.1 (c). Using expressions for t′ and **d** in equation (I.2) and (I.6), W′ is

$$W' = (4R^2 - t^2 \tan^2\theta)^{1/2} \ t \sin\theta \ . \tag{I.7}$$

The wall area, S′, can be found by projecting the sector area, S, of the circle shown in Figure I.1 (a). The sector area is

$$S = \tfrac{1}{2} R^2(\phi - \sin\phi)$$
$$\text{where} \quad \phi \equiv 2 \cdot \sin^{-1}\left(\frac{\ell}{R}\right) = 2 \cdot \sin^{-1}\left[\frac{t \tan\theta}{2R}\right]. \tag{I.8}$$

The projected sector area or the wall area, S′, is

$$S' = S \cos\theta = [R^2 (\phi - \sin\phi) \cos\theta]/2 \ . \tag{I.9}$$

Therefore, **f**, the ratio of open area, viewed from an angle θ, to the total projected orifice area is

$$f = \frac{A_{eff}}{A'} = 1 - \frac{W' + 2 \cdot S'}{\pi R^2 \cos\theta} \ . \tag{I.10}$$

Substituting (I.8) and (I.9) into (I.10) gives

$$f = 1 - \frac{\xi \tan\theta \, [4 - \xi^2 \tan^2\theta]^{1/2}}{\pi} - \frac{(\phi - \sin\phi)}{\pi} \qquad \text{(I.11)}$$

where $\xi \equiv t/r$.

This is the fraction of ions which pass through the orifice, *i.e.* do not strike the inner walls of the orifice. For example, for an ion with normal incidence ($\theta = 0°$) or for an infinitely thin orifice ($t \rightarrow 0$), f equals unity. Figure I.2 shows how the measured IAD is changed by taking into account the finite thickness of the orifice. Correcting for the different size openings that ions with different incident angles see will increase the fraction of ions at larger angles, but the effect on the average angle is very small. The area correction factor is most important for high aspect ratio orifices.

I.2 Collisions Downstream of Orifice

The static pressure in the ion analyzer chamber and the expanding gas from the orifice can cause ion-neutral collisions, potentially corrupting the ion bombardment measurements. The calculation of the percent of ions that undergo collisions after passing through the sampling orifice is based on the work of Coburn and Kay (1971). In the apparatus used in this experiment, the pumping speed is high enough to provide a sufficiently low static pressure in the ion analyzer chamber so that the mean free path is on the order of several meters. Therefore, scattering from the static gas is insignificant. The expanding gas from the orifice forms a localized, high gas density downstream of the orifice which varies with plasma pressure and orifice radius. According to Coburn and Kay (1971), the fraction of ions that make it to the collector without collisions can be expressed by

$$\frac{I(\ell)}{I_o} = \exp\left[-\frac{(\sigma R N)}{2}\tan^{-1}\left(\frac{\ell}{R}\right)\right] \qquad (I.12)$$

where $I(\ell)$ is the ion current at a distance, ℓ ,from the orifice; I_o is the current at the orifice; σ is the ion collision cross section; N is the neutral density on the upstream side of the orifice; and R is the orifice radius. The factor of ½ in the exponent corrects for an error in this reference which was pointed by J. W. Coburn. It accounts for the lower density of gas immediately above the orifice. Substituting in ℓ = 2.5 cm (the distance from the orifice to the ion collecting plate), 2R = 75 μm, $\sigma = 6.5 \times 10^{-15}$ (Cramer 1959), and pressures of 0.05 and 0.5 Torr, the fraction of ions reaching the collector without collisions is 0.99 at 0.05 Torr and 0.84 at 0.5 Torr. Therefore, no corruption of the measured ion properties at 0.05 Torr and below is caused by downstream collision, even with the 75 μm diameter orifice. However, at the higher pressures (0.5 Torr), some corruption is expected. Use of the 50 μm orifice at 0.5 Torr reduces the scattered fraction to 0.06.

I.3 Space Charge Broadening of Ion Beam

As the ion beam passes through the orifice, the ions in the beam repel each other, and may cause the ions to spread out. This would result in ions having larger measured incident angles than the actual incident angles at the electrode surface. The effect of space charge is investigated assuming a current flux of 0.03 Amp/m^2 on the downstream side of the orifice and 5 eV ions. The following equation uses the worst case assumption that all the ions have the same incident angle, that is, they form a collimated ion beam. In a collimated beam, the ions repel each other more than a diverging ion beam from the plasma where there is a rapid reduction of space charge density as the ions pass through the orifice. Integrating the following equation (Brodie and Muray 1982) gives the radius of the ion beam, r, as a function of the distance away from the

orifice, z:

$$\frac{dR}{dZ} = A\sqrt{\ln R}$$

$$\text{where} \quad R = r/R_o \,, \quad Z = z/R_o \,, \tag{I.13}$$

$$A = \left(\frac{I}{2\pi \varepsilon_o V \sqrt{2qV/m}} \right)^{\frac{1}{2}} .$$

R_o is the initial beam radius, qV is the initial ion energy, m is the ion mass, and I is the ion current. At the highest measured ion flux, 0.03 Amp/m^2, space-charge repulsion causes the width of the ion beam to increase by a factor of 6.7, or deflect the beam by 2°. Since the angular resolution of the detector is 4.5°, the space-charge effects can be neglected.

I.4 Potential Field Perturbation Around the Orifice[3]

Electric field variations near an orifice deflects ions and can limit the accuracy of IAD measurements. Ion trajectory deflection was modeled using D. A. Dahl's (1985) SIMION program, which was developed at Idaho National Engineering Laboratory. SIMION does not include space charge effects, but these are not important for the density and length scales of the orifices considered.

In the SIMION program, the sheath electric field, $\mathcal{E}_{sh}$, is assumed to be spatially uniform. Nonlinear field simulations are not needed since the electric field is approximately uniform within a few orifice diameters above the electrode. There are three independent input parameters for SIMION: $\mathcal{E}_{sh}$, orifice diameter (d_o), and orifice aspect ratio. The electric field upstream of the orifice is set at 1000 V/cm, a worst case condition, and the potential of the orifice and a parallel

[3]All the simulations with SIMION to model ion trajectories as the ions travelled through the non-uniform field at the orifice were done by G. L. Huppert.

plane more than four orifice diameters downstream of the orifice are set to zero. Scaling of the variables requires certain cases to be equivalent. For example, a 25 μm orifice with an aspect ratio of 0.5 and a field of 1000 V/cm is equivalent to a 50 μm orifice with an aspect ratio of 0.5 and a field of 500 V/cm.

Figure I.3 shows the effect of orifice thickness on the potentials near the orifice. The potential lines are not significantly affected by changing the orifice thickness. This can be seen by noting that the 0.2 V and greater potential curves are essentially the same in all cases. The differences in the curves for lower potentials will only affect ions with comparable energies. Therefore, there are no significant variations of the potential lines with different orifice aspect ratios.

Figure I.4 shows the effect of orifice diameter on the potential curves near the orifice. The orifice thickness is 9 μm, and the diameters are 25, 50, and 75 μm. Larger diameter orifices result in larger potential line deviations, and thus, greater ion trajectory deviations. This can be seen by noting that the potential line at the front face of the orifice increases with orifice size. Since the orifice thickness does not significantly affect the potentials, the effect of doubling the orifice diameter is approximately equivalent to the effect of doubling $\mathcal{E}_{sh}$. The simulations show that potential deviation is mainly a function of $\mathcal{E}_{sh}d_o$. This group should be small to minimize potential and ion trajectory deviations.

Ions passing through the orifice are deflected from their original trajectories by the bending potential lines. Figure I.5 illustrates the trajectory deflection for low energy ions at normal incidence as they pass through a 75 μm orifice. Under the conditions presented in Figure I.5, the deflection is sufficiently severe that most ions are deflected more than two degrees. Table I.1 shows the effect of various parameters on the ion trajectory deflection. The first and second sets of data show that ions experience larger deflections passing through larger diameter orifices.

This effect suggests that the orifice diameter should be small to avoid angular deflections. The third set of data shows that deflection becomes worse for ions with large incident angles. The fourth set of data shows that more energetic ions experience less deflection.

Simulation results with various initial ion energies indicate that the important criterion to assure small angular deflections (i.e. less than 2°) at the orifice is:

$$\frac{\varepsilon_{ion}}{q\mathscr{E}_{sh}d_o} \geq 2 \ , \tag{I.14}$$

where ε_{ion} is the energy of the ion. In the experimental apparatus, ions with energies greater than 5 eV should not be deflected more than 4.5°, the angular resolution of the detector.

The orifices used to separate the plasma from the ion analyzer are constructed from aluminum, and may have thin oxide layer coating the surface. The oxide acts as an insulator, which may result in charging of the orifice walls and distortion of the local potential fields. This effect is examined by setting the potential on the orifice wall to 2.5, 1, -1, and -2.5 V to simulate charging due to excess ion or electron bombardment. Results for ion deflection are presented in the last section of Table I.1. Orifice charging has very little effect on the ion flight path of high energy ions even though the potential profile is significantly changed around the orifice. The fields at the center of the orifice is small, limiting ion scattering to the edge of the orifice. However, ions which pass near the edge of the orifice may be reflected toward the center when the orifice walls are positively charged.

Field perturbation near the orifice has insignificant effect for most ions, *i.e.*, ions with energy greater than 5 eV. Charging of the inner orifice surface also has little effect on ion flight path. Angular distributions reported in this thesis are therefore accurate within the measurement resolution of 4.5°.

Table I.1: Ion Deflection from Non-Uniform Fields at the Orifice

Ion deflection, in degrees, relative to initial incidence trajectory. The conditions set in the simulation are: $\mathcal{E}_{th}$ = 1000 V/cm upstream of electrode, 0 V at electrode and four orifice diameters downstream. Edge denotes ions traveling past the edge of the orifice, where maximum deflection typically occurs. Center denotes ions which pass through the center of the orifice, where minimum deflection typically occurs. All entries are angular deflections of the ion trajectories from that which would occur at the orifice's front plane if no field distortion were present.

1. Deflection of ions incident normal to the surface, 2.5 eV ions.

Orifice Diameter	25 μm	50 μm	75 μ
Edge	6.5	12.6	19.5
Center	0	0	0

2. Deflection for ions incident at 10.9°, 3.5 eV ions.

Orifice Diameter	25 μm	50 μm	75 μ
Edge	6.4	9.2	13.9
Center	0.3	3.0	5.4

3. 25 μm orifice, 3.5 eV ions, varying initial incident angles.

Incident Angle	1.6°	4.9°	10.9°	20.8°
Edge	6.2	7.6	6.4	12.3
Center	0.2	0	0.3	0.6

4. 25 μm orifice, 10.9° incident angle, different final energies.

Ion Impact Energy	3.5 eV	7.5 eV	12.5 eV	22.5 eV
Edge	6.4	2.6	1.9	1.2
Center	0.3	0.9	0.7	0.5

5. Orifice charging effects, 25 μm orifice, 10.9° angle, 3.5 eV.

Orifice Edge Charging	-2.5 V	-1.0 V	+1.0 V	+2.5 V
Edge	0.8	2.7	1.1	25 ± 5
Center	0.3	0.3	0.5	0.9

I.5 Magnetic Field Effect on Ion Trajectory

Since the distance the ion must travel between the orifice and the ion collecting plate is approximately 2.54 cm, this can be sufficient distance for the ion trajectory to change more than 4.5°, the resolution of the angle measurement technique, in the presence of a magnetic field. The stronger the magnet field, the more the ion trajectory deviates from its original direction. The ion analyzer grids create electric fields parallel to the ion trajectory which can lessen ion trajectory change if the field accelerates the ion and increase ion trajectory deviation from its original path if it slows down the ion. How much the ion trajectory changes also depends on the ion incident angle relative to the magnetic field lines.

It is necessary to find out which ions under what conditions are affected by the magnetic field. The following derivation uses force balances to find the ion position as a function of the time the ion spends in the ion analyzer, the ion's incident angle with respect to the magnetic field line, the electric field and the magnetic field in the ion analyzer. The xyz coordinate system is defined by the ion incident angle and the magnetic field direction, as shown in Figure I.6. The y-axis is defined as the direction of the magnetic field. The component of the initial ion trajectory perpendicular to the magnetic field defines the x-axis. Finally, the z-axis is just the cross product of the x and y axes.

With the coordinate system shown in Figure I.6, the components of the magnetic and electric fields are expressed as follows:

$$\mathbf{B} = B_y \hat{y} \qquad \mathbf{E} = E_x \hat{x} + E_y \hat{y} \; . \tag{I.15}$$

The ion incident angle is $\theta = 90° - \phi$. Another angle, ψ, defines the ion trajectory relative to the magnetic field lines. However, it is more useful to be able to specify the angle, β, which is the angle between the component of the ion trajectory projected onto the plane of the electrode surface and the magnetic field line. The angle ψ can easily be found from the specified angles,

θ and β using the following equation:

$$\cos\psi = \cos\beta \sin\theta \ . \tag{I.16}$$

Once ψ has been calculated, the components of the electric field are

$$E_x = E\sin\psi \qquad E_y = E\cos\psi \tag{I.17}$$

The force balance on the ion gives

$$F = qE + q\vec{v} \times B = qE + q\begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ \dot{x} & \dot{y} & \dot{z} \\ 0 & B & 0 \end{vmatrix} = \hat{x}(qE_x + q\dot{z}B) + \hat{y}qE_y + \hat{z}(q\dot{x}B) \tag{I.18}$$

where $\dot{x}$, $\dot{y}$, and $\dot{z}$ are the ion velocities in the x, y, and z directions. The force balances in x, y, and z directions from Equation I.18 produce:

$$\begin{aligned} \ddot{x} &= \frac{AE_x}{B} - A\dot{z} \\ \ddot{y} &= \frac{AE_y}{B} \\ \ddot{z} &= A\dot{x} \\ \text{where } A &\equiv \frac{qB}{m} \ . \end{aligned} \tag{I.19}$$

After some manipulations, and using the condition that at time, t = 0, the initial ion energy can be broken down into velocities in only the x and y directions, the following equations result for the velocity and the position of the ion as a function of time.

$$\dot{x} = C_1 \cos(At+\phi)$$

$$\dot{z} = C_1 \sin(At+\phi) + \frac{E_x}{B}$$

$$x = \frac{C_1}{A}[\sin(At+\phi) - \sin\phi] + x_o$$

$$y = \frac{AE_y}{B}\frac{t^2}{2} + \dot{y}_o t + y_o \qquad \text{(I.20)}$$

$$z = \frac{C_1}{A}[\cos\phi - \cos(At+\phi)] + \frac{E_x}{B}t + z_o$$

$$\text{where} \quad C_1 \equiv \frac{\dot{x}_o}{\cos\phi}$$

$\dot{x}$ and $\dot{y}$ are the initial ion velocities in the x and y directions. Recall that $\phi = 90° - \theta$. There are four sections in the ion analyzer, each with a different electric field, which the ion must pass through to reach the collecting plate. The time is incremented in Equation I.20, to solve for x, y, and z, until $L = (x^2+y^2+z^2)^{1/2}$, the distance from the orifice to grid #1 is reached. Then, a new electric field and initial velocities and positions are used with the same equations to calculate the ion trajectory for the next section. The ion trajectory is followed until the ion passes through grid #2 or, if the ion does not have enough energy to pass through grid #2, until the ion reverses direction toward grid #1. The calculation is repeated for the section between grid #2 and #3, and between grid #3 and the collecting plate. Newton's method is used to solve Equation I.20 for the time it takes the ion to travel through each section in the ion analyzer.

The difference between the ion incident angle at the collecting plate and the original incident angle, $\Delta\theta$, is easily found from the final position of the ion. The magnetic field only affects the ion velocity components perpendicular to the field lines. Therefore, only change in the plane formed by the x-z axes will affect which annular ring on the collecting plate the ion falls on. Referring to Figure I.7, the incident direction vectors of the ion in the x-z plane is calculated

using:

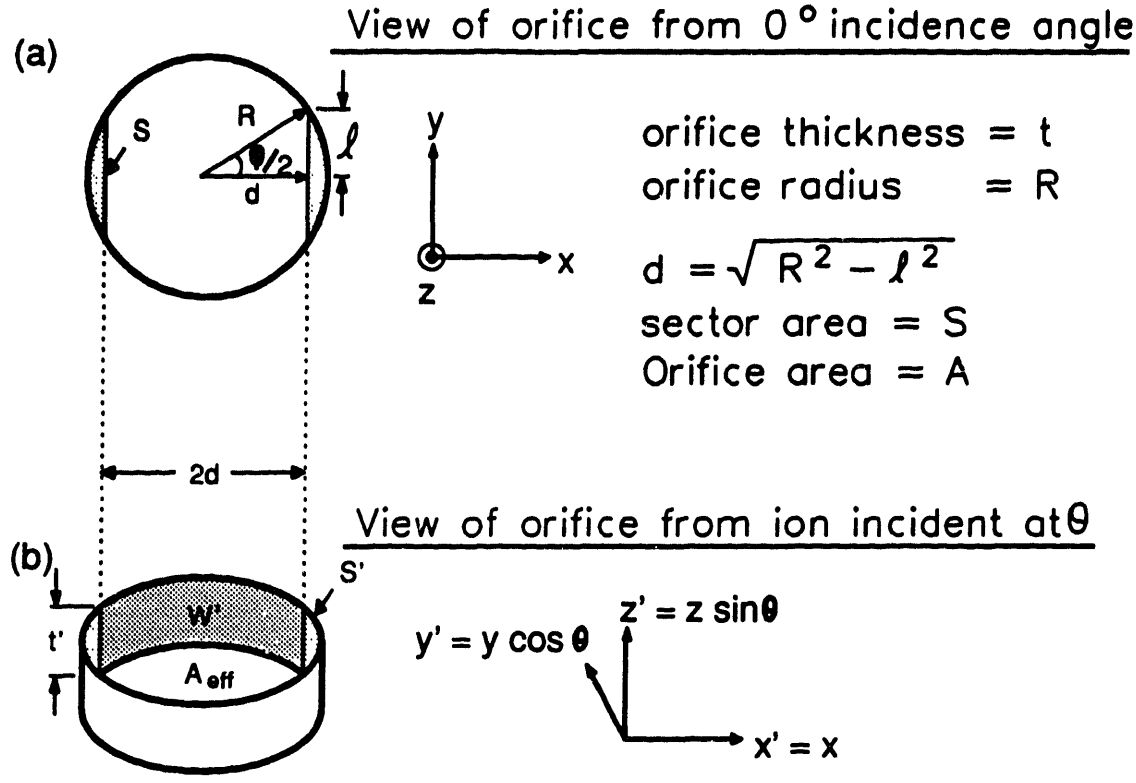
$$\tan\rho = \frac{\sin\theta \sin\beta}{\cos\theta}$$
$$p = x\sin\rho + z\cos\rho \qquad (I.21)$$
$$n = x\cos\rho - z\sin\rho$$

The altered ion incident angle, θ^*, and the change in ion incident angle, $\Delta\theta$, are:

$$\tan\theta^* = \frac{\sqrt{p^2 + y^2}}{n} \quad \text{and} \quad \Delta\theta = \theta - \theta^* . \qquad (I.22)$$

Program TRAJ.PAS, found in Appendix K, calculates the ion trajectory in the ion analyzer using the procedure outlined above with the magnetic field strength, potential on all three grids, initial ion energy, and ion incident angle relative to the magnetic field lines as inputs. The magnetic field has the greatest effect on 0° incident angle ions because $\mathbf{v} \times \mathbf{B}$ in the force balance equation has the largest magnitude at $\theta = 0°$. The magnetic field change low energy ion trajectories more than it does high energy ion trajectories.

Figure I.8 shows the effect of grid #2 bias along with the magnetic field on the ion trajectory. All the results presented here have been done assuming a 200 Gauss magnetic field to find the worst case scenario. Only when the grid #2 potential is almost equal to the ion energy does the ion trajectory get reversed by the magnetic field. For potentials less than the ion energy, the ion will reach the collecting plate. This shows that it is possible to use the ion analyzer to measure the ion energy distribution (IED) during experimental runs with the presence of magnetic fields. On the other hand, the IAD measurements will not be very useful since ions with energies around 5 eV or less will not strike the correct annular ring on the collecting plate, even with grid #2 biased at 0 V.



t' = projected thickness = t sin θ

A' = projected orifice area = A cosθ

S' = projected sector area = S cosθ

A_{eff} = area seen by ion with θ incidence angle

W' = wall of orifice

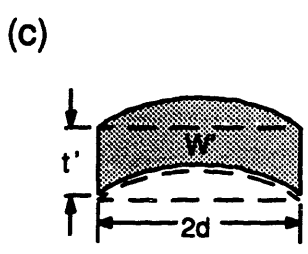


Figure I.1: Various views of the orifice. Ions with off normal incident angle do not see the entire orifice opening.

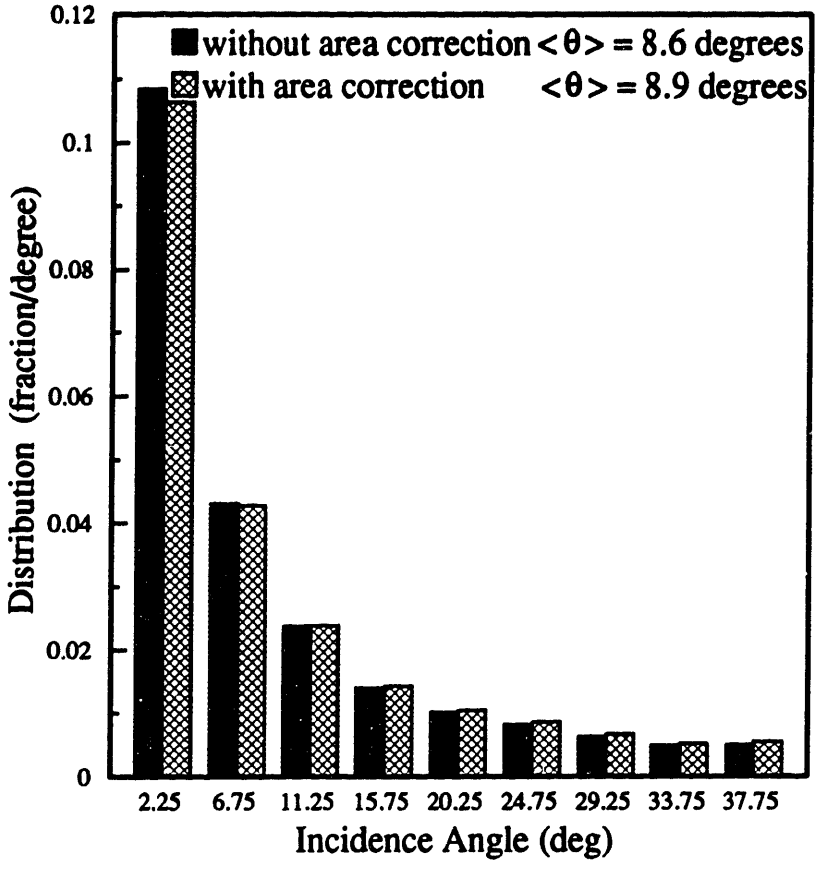


Figure I.2: Argon ion angle distributions showing the effect of using the orifice area correction factor. The data is taken at 0.010 Torr and Vo=115 with an orifice aspect ratio, t/r, of 0.12.

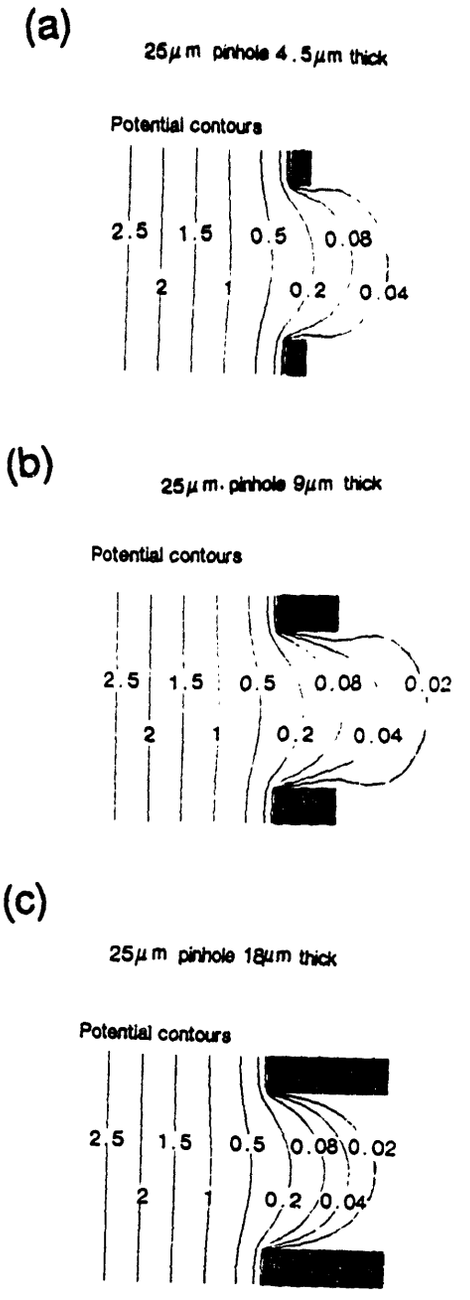


Figure I.3: Effect of orifice thickness on potentials near orifice. 1000 V/cm electric field, 25 um orifice. Potentials are shown in Volts. (a) 4.5 um thick orifice, 0.18 aspect ratio; (b) 9 um thick orifice, 0.36 aspect ratio; (c) 18 um thick orifice, 0.72 aspect ratio.

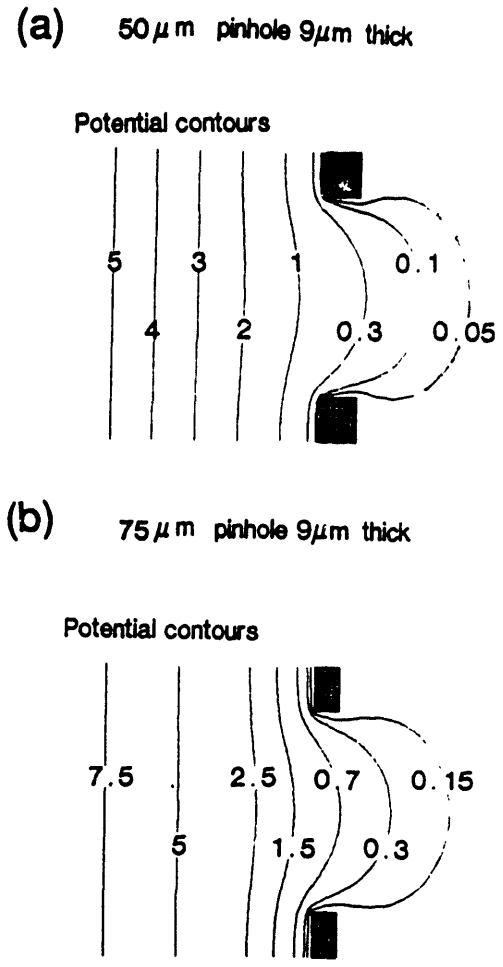


Figure I.4: Potential deviations near 9 um thick orifices with 1000 V/cm sheath electric fields. The 25 um diameter case is shown in Figure I.3 (b). (a) 50 um thick orifice. (b) 75 um thick orifice.

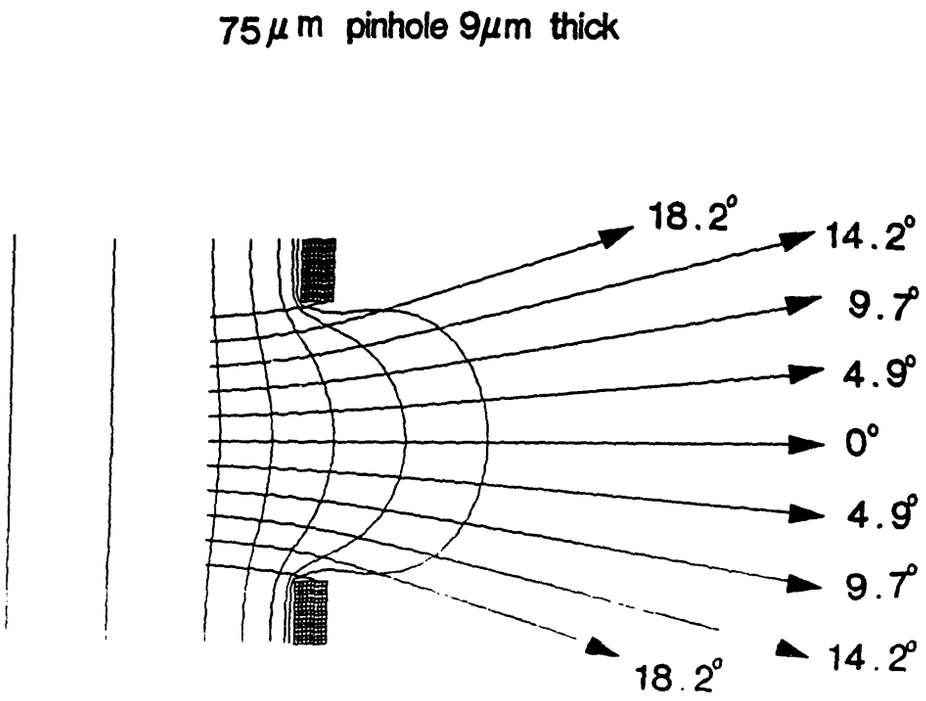


Figure I.5: Angular deviations induced by non-uniform potentials near the orifice. Potential contours are the same as that shown in Figure I.4 (b). Angles presented are with respect to electrode surface normal. Ions have 3.5 eV energy at the detector. Note that very few ions will be detected at 0 degrees for these low energy ions.

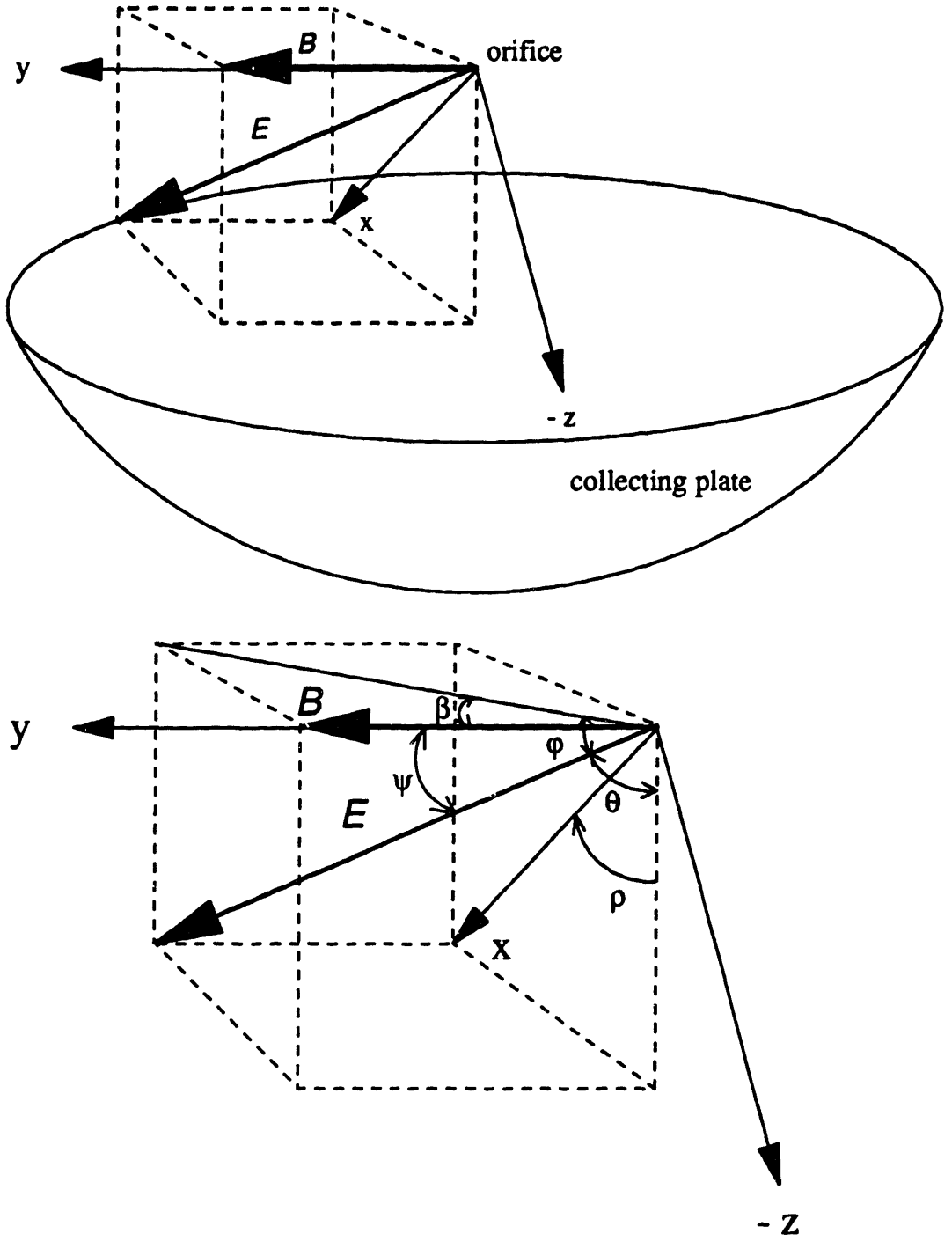


Figure I.6: Coordinate system defined by the magnetic field line and the ion trajectory. The original ion path as it passes through the sampling orifice is the same direction as the electric field.

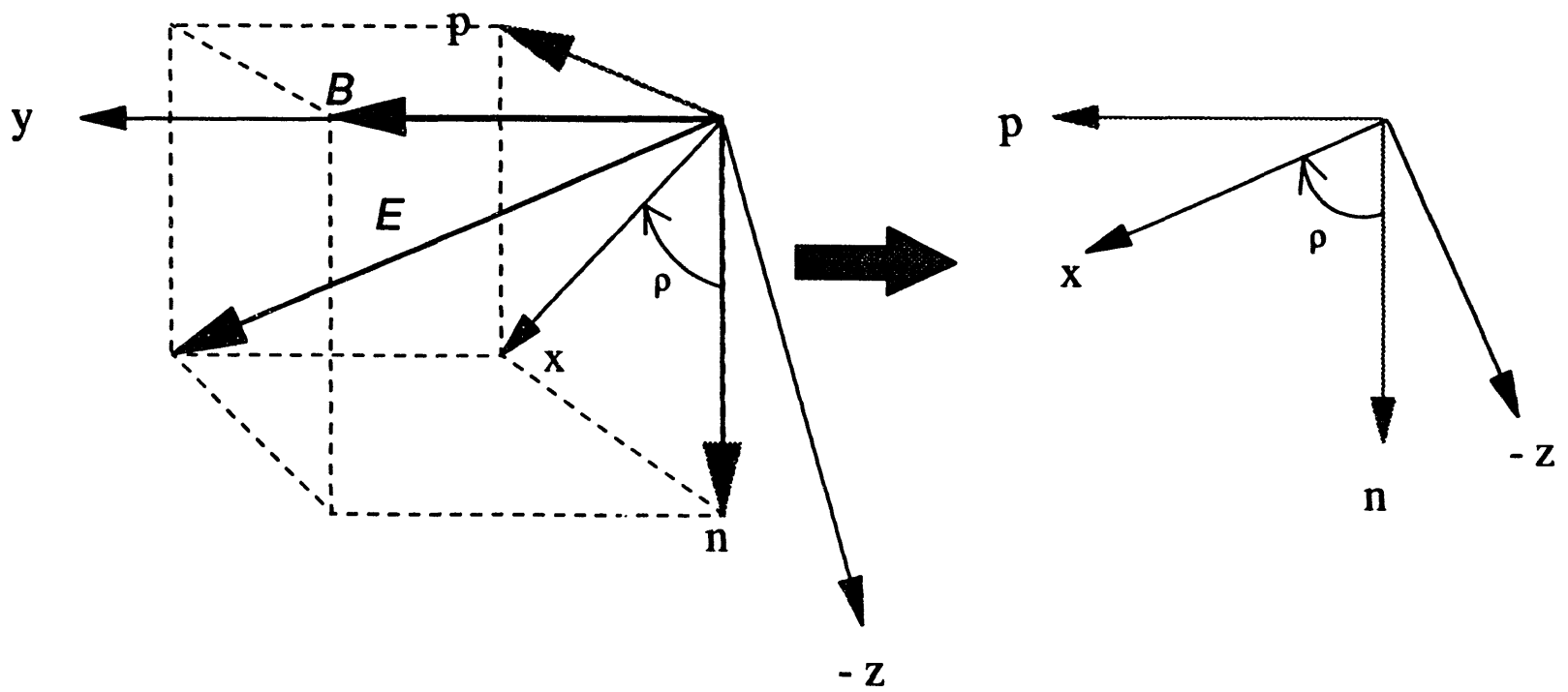


Figure I.7: Vectors p and n with are oriented relative to the electrode surface and vectors x and y which are oriented with respect to the ion incident angle and the magnetic field.

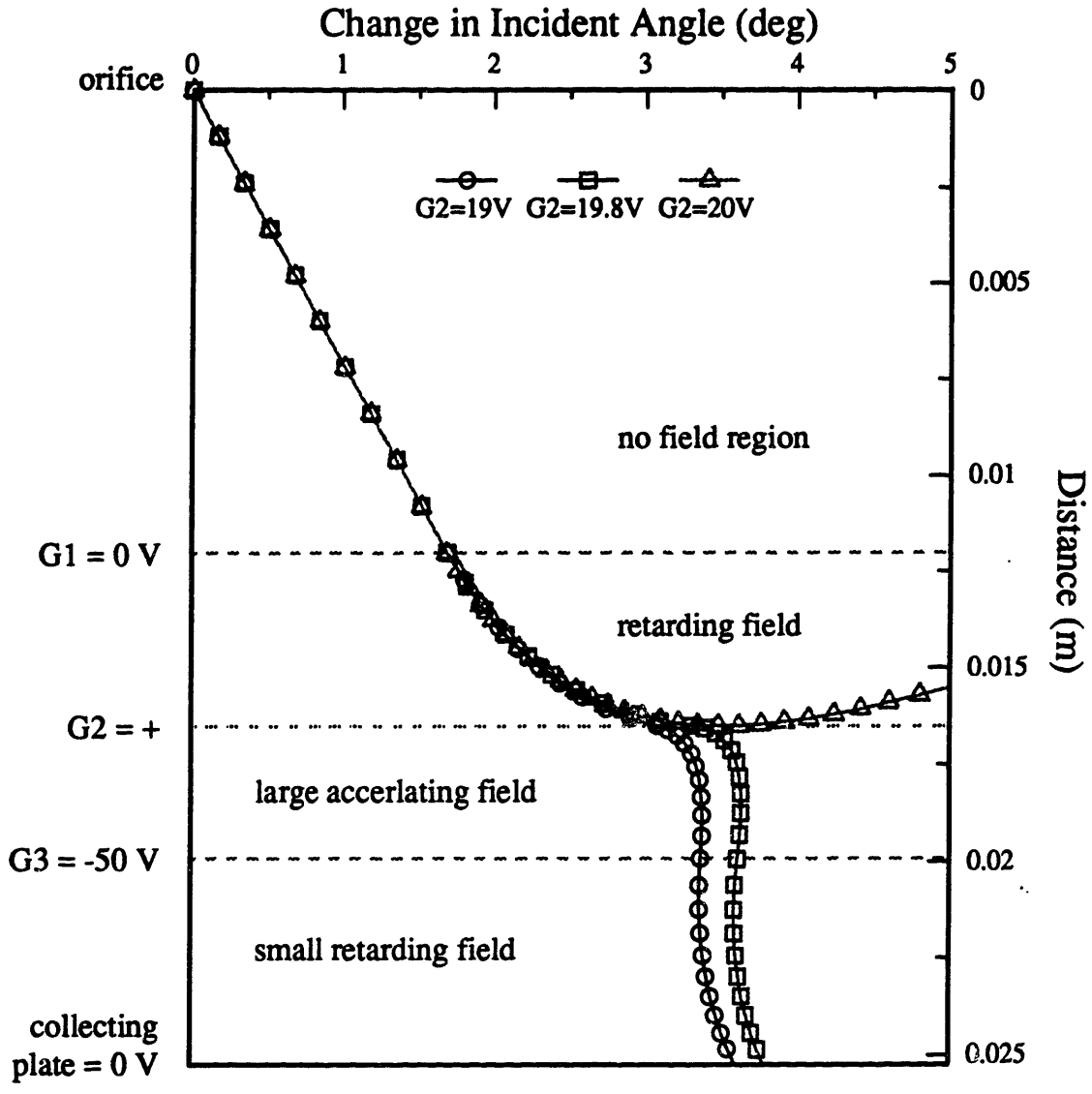


Figure I.8: Argon ion trajectory at various G2 biases. The ion energy is 20 eV with orginal incidence angle of 0 degrees. The magnetic field is 200 Gauss.

APPENDIX J

GAS FLOW CALIBRATION

Gas flows through a Tylan FM360 50 sccm mass flow controller into the region between the electrodes in the reactor. The mass flow controller gives an output reading that is proportional to the mass flow rate. This appendix shows how the proportionality constant is found. Since a teflon cylinder surrounds the electrodes and pressure is measured outside of this cylinder with a capacitance manometer, gas flowing from the plasma region out of the teflon cylinder creates a pressure drop between the plasma region and the pressure measuring point. The second section of this appendix shows how the pumping conductance between the plasma region and the rest of the reactor is calibrated. Knowing the pumping conductance, the plasma pressure can be related to the measured pressure.

J.1 Mass Flow Meter Calibrations

The gas controller gives an voltage output proportional to the mass of the gas flowing through the controller. This voltage is reduced through a voltage divider circuit and displayed on a digital panel meter. The proportionality constant, β, between the displayed voltage and the gas mass flow rate is found by flowing gas into a fixed volume at several flow rates and measuring the pressure rise with time. The fixed volume used is the 17 cm diameter pyrex chamber, without any teflon confinement rings and with the valve to the cryogenics pump closed. The total volume, V_R, which includes the gas line from the gas flow meter to the reactor, the reactor itself, and the vacuum lines from the reactor to the pumps, is 8.3 liters. The gas leak rate, $\dot{n}_L$, into the reactor is measured by shutting the valve from the gas cylinder and the valve to the pump, and measuring the pressure rise rate, $\dot{p}$. The leak rate is

$$\dot{n}_L = \dot{p}\, V_R = 9.3 \times 10^{-5}\ \frac{\text{Torr}\cdot\ell}{\text{sec}}\ . \tag{J.1}$$

This gas leak rate is so much smaller than the gas flow rate that it will not affect the calibration of the mass flow controller.

Figure J.1 shows the measured pressure rise in the reactor for several gas flow meter voltage output readings for argon gas. The intercept of the best fit line through these points is zero, corresponding to a zero capacitance manometer output with no gas flow. This indicates that the capacitance manometer is correctly zeroed. Since the gas flow rate, $\dot{n}$, is

$$\begin{gathered} \dot{n}(\text{Torr}\cdot\ell/\text{sec}) = \dot{p}(\text{Torr/sec})\ V_R(\ell) \\ \text{or} \\ \text{voltage reading(V)} \times \beta(\text{Torr}\cdot\ell/\text{sec V}) = \dot{p}\, V_R\ , \end{gathered} \tag{J.2}$$

the slope in Figure J.1 is equivalent to

$$\text{slope} = \frac{V_R}{\beta}\ . \tag{J.3}$$

Therefore, the mass flow controller calibration factor for argon gas is

$$\beta_{Ar} = \frac{V_R}{\text{slope}} = \frac{8.3}{12.6} = 0.657\ \frac{\text{Torr}\cdot\ell}{\text{sec}\cdot\text{V}} = 51.96\ \frac{\text{sccm}}{\text{V}}\ . \tag{J.4}$$

Similarly, the calibration factor for SF_6 gas is

$$\beta_{SF_6} = 0.12\ \frac{\text{Torr}\cdot\ell}{\text{sec}\cdot\text{V}} = 9.45\ \frac{\text{sccm}}{\text{V}}\ . \tag{J.5}$$

J.2 Pupming Conductance Calibrations

In the 9 cm and 13 cm diameter pyrex reactors, a teflon cylinder surrounds the plasma region so that there is a pressure drop between the electrodes and the pressure measuring point.

To calibrate the pressure drop and plasma pressure as a function of gas flow rate, a second capacitance manometer is temporarily installed where the ion analyzer usually sits. The pressure reading from this capacitance manometer is the pressure above the grounded electrode. Figure 2.6 in the main body of the thesis shows the locations of the capacitance manometers with respect to the plasma region.

Before calculating the pumping conductance in these reactor configurations, the reactor volume with all the teflon confinement rings is needed. This is found by flowing gas into the reactors, with the valve to the pumps closed, at various known mass flow rates and measuring the pressure rise in the reactor. The following equation,

$$V_R = \frac{\dot{n}_L}{\dot{p}} = \frac{\beta \cdot \text{voltage reading}}{\dot{p}}, \qquad \text{(J.6)}$$

yields 4.77 ℓ and 5.74 ℓ for the reactor volumes of the 9 cm and 13 cm diameter reactors, respectively.

The pumping conductance, U_A, between the two pressure measuring points is calculated in two steps. First, from the two measured pressures, the pumping conductance, U_B (between the first capacitance manometer and the pump) is calculated; followed by calculation of the pumping conductance, U_o (between capacitance manometer #2 or the plasma region and the pump). Then U_A is solved as a function of U_B and U_o.

Figure J.2 shows the measured pressure, using both capacitance manometers, at various gas flow rates for the 9 cm diameter reactor. The first few initial points fall on a linear line, but as flow rate increases, the points deviate from a straight line. This is because as pressure increases, the gas flow changes from rarefied to viscous flow, and thus the pumping conductance changes from being pressure independent to pressure dependent. The gas flow through the reactor can be written as

$$\dot{n} = U_i (p_i - p_o) \tag{J.7}$$

where p_i is the pressure measured with capacitance manometer #1 or #2, U_i is the pumping conductance between the measuring point and the pump, and p_o is the pressure at the pump. If the pressure at the pump is assumed to be much less than p_1 or p_2 and the pumping speed of the cryogenics pump is assumed to be much larger than the pumping conductance, then differentiating Equation J.7 gives:

$$\frac{d\dot{n}}{dp_1} = U_B \qquad \text{and} \qquad \frac{d\dot{n}}{dp_2} = U_o \,. \tag{J.8}$$

This means that the slopes in Figure J.2 are equal to U_B and U_o. U_B and U_o are found by finding the best polynomial fit equations through the points in Figure J.2, shown by the dashed lines, and differentiating the polynomial equations, resulting in:

$$\begin{aligned} U_B &= 1.92 + 65.96\, p_1 \\ U_o &= 1.54 + 35.53\, p_2 \end{aligned} \tag{J.9}$$

where the pressure is expressed in Torr and U_i is expressed in ℓ/sec. Figure J.3 shows the gas flow rate data as a function of the measured pressures, similar to Figure J.2, but for the 13 cm diameter reactor. The pumping conductance calculated from these points are:

$$\begin{aligned} U_B &= 2.98 + 43.26\, p_1 \\ U_o &= 2.59 + 27.15\, p_2 \,. \end{aligned} \tag{~~J.10~~}$$

Although the pumping conductances are found using argon gas, they should remain the same for any gas at low pressures where rarefied flow dominates. At higher pressures in the viscous flow regime, the type of gas should have only a small effect on the pumping conductance. In this regime, the pumping conductance is inversely proportional to the gas viscosity which does not vary much for different types of gases.

During the experiment, the plasma pressure, p_2, can be calculated from the measured

pressure, p_1, and the gas flow rate, $\dot{n}$. Knowing U_B and U_o, the pumping conductance between the plasma and the pressure measuring point, U_A, is

$$\frac{1}{U_A(p_1,p_2)} = \frac{1}{U_o} - \frac{1}{U_B} , \tag{J.11}$$

which depends on both p_1 and p_2. From the measured pressure, the plasma pressure, p_2, can be solved iteratively using

$$p_2 = \frac{\dot{n}}{U_A(p_2)} + p_1 . \tag{J.12}$$

PLASMA_P.PAS and MEAS_P.PAS, found in Appendix K, are programs that solve p_2 from p_1, and p_1 from p_2, respectively.

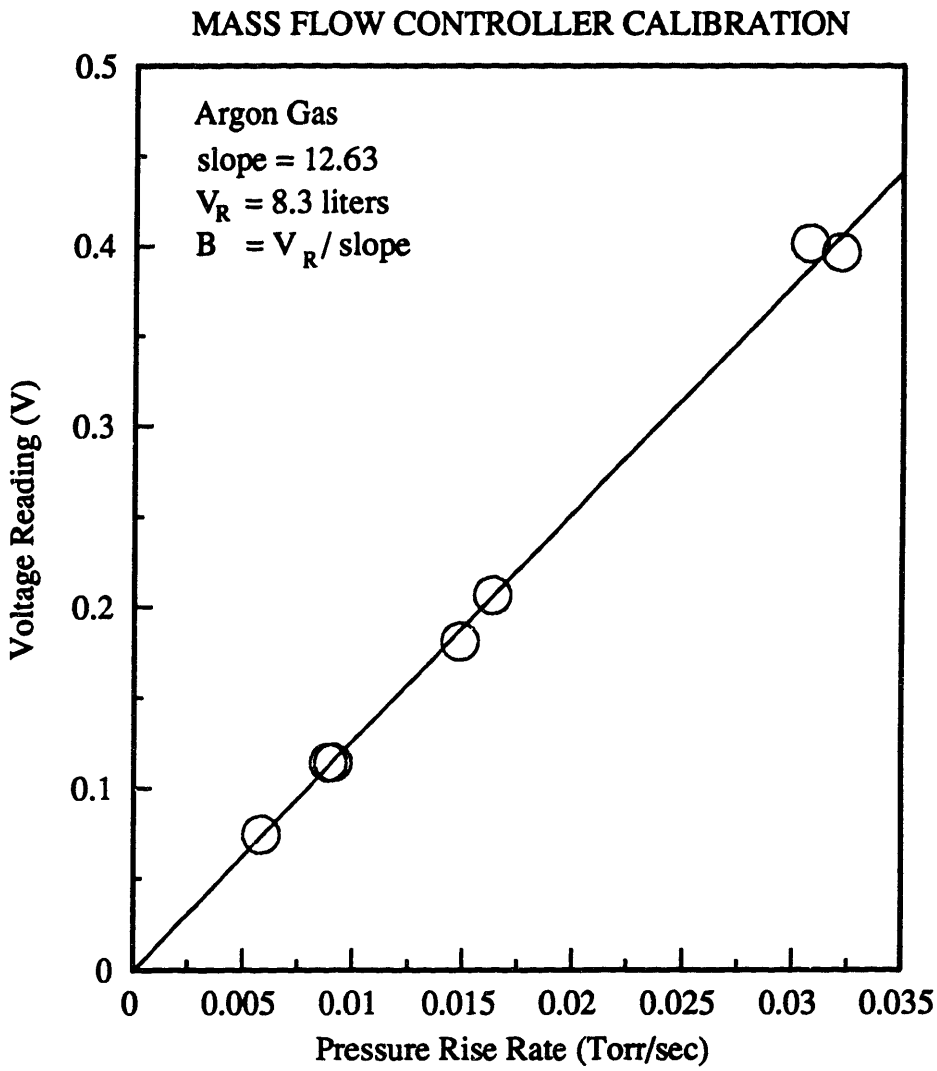


Figure J.1: Data for calibrating the mass flow controller so that the voltage output can be converted to mass flow rates. This graph shows the voltage output from the mass flow controller at several gas flow rates, plotted as a function of the measured pressure rise in the 17 cm diameter reactor with the valve to the cryogenics pump closed.

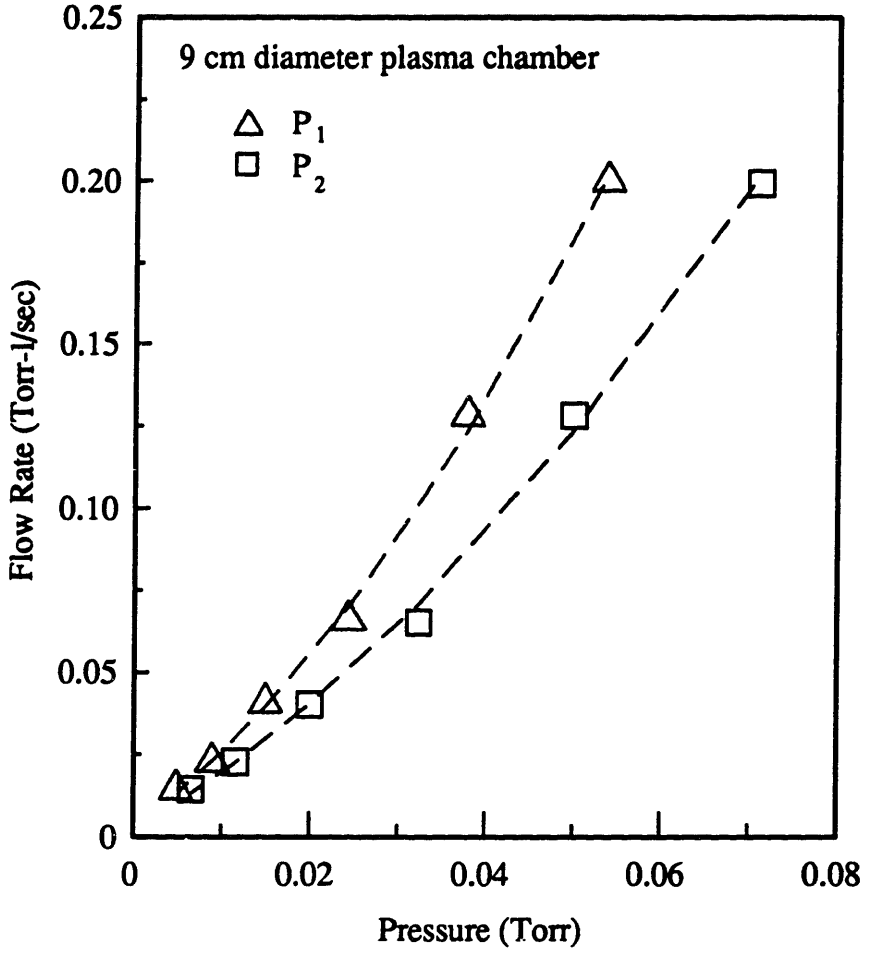


Figure J.2: Several argon gas flow rates plotted as a function of the pressure measured at two places in the reactor. p1 is the pressure outside of the teflon cylinder and p2 is the pressure between the electrodes.

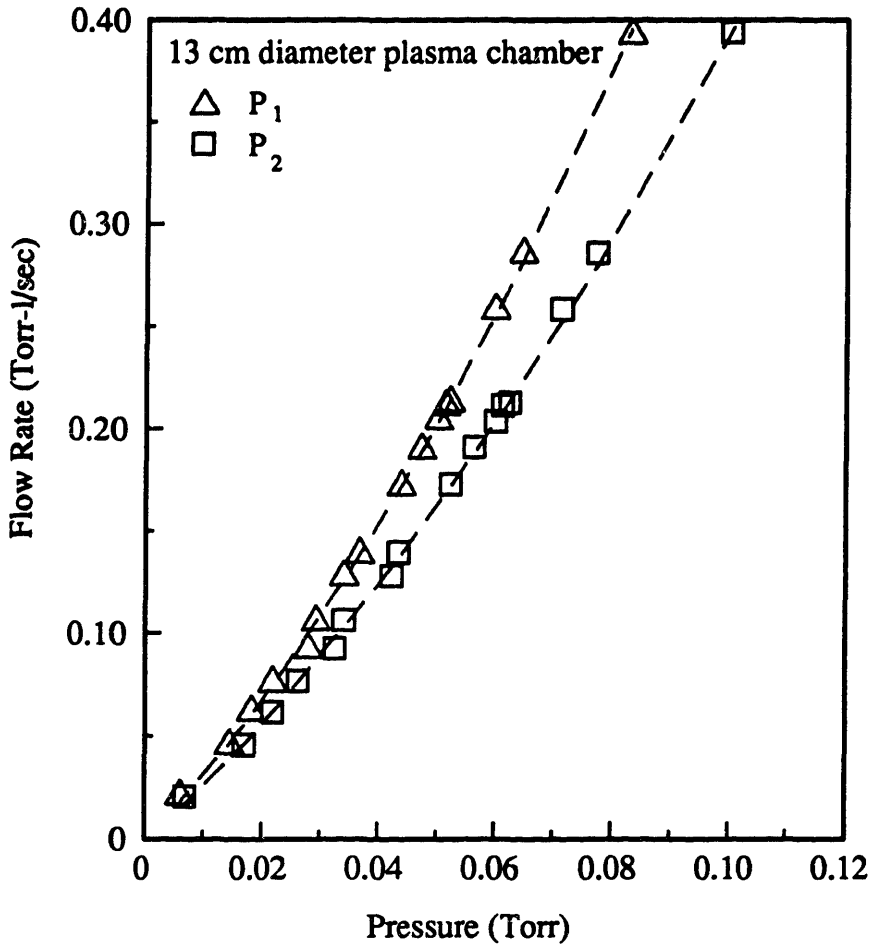


Figure J.3: Same as Figure J.2 except using the 13 cm diameter chamber.

APPENDIX K

FUNCTIONALITIES FOR $\mathscr{E}_b$, ℓ_s, L_b, AND R_s

The bulk electric field dependence on the plasma dimensions and operating pressure is derived using an electron energy balance and an ion mass balance[4]. Assuming that the density of ions equals that of electrons in the bulk, an ion balance equating the number of ions created in the bulk to the number of ions lost to the surface gives

$$\nu_i n_e d_b = 2 v_B n_i = 2 v_B n_e \quad \rightarrow \nu_i = \frac{2 v_B}{d_b} \quad , \tag{K.1}$$

where ν_i is the ionization frequency and v_B is the Bohm velocity. If the current through the bulk is carried by electrons and is in the mobility limited regime, and the plasma bulk resistance results from electron collisions, an electron energy balance for bulk power loss is

$$\nu_i n_e d_b A e_\ell = \frac{1}{2} I_o^2 R_b$$

$$\left(\frac{2 v_B}{d_B}\right) n d_b A e_\ell = \frac{1}{2} (A q n_e v_d)^2 \frac{m \nu_c}{q^2 n_e} \frac{d_b}{A} \tag{K.2}$$

$$v_d = \sqrt{\frac{4 e_\ell v_B}{m \nu_c d_b}}$$

where $v_d = \mu \mathscr{E}_b$ is the electron drift velocity and e_ℓ is the electron energy lost per ionization collision. Since the collision frequency, ν_c, is proportional to pressure, the drift velocity depends on pressure and plasma bulk length as

$$v_d \propto (p d_b)^{-1/2} \quad . \tag{K.3}$$

The voltage drop in the bulk can now be written as

[4]M. Surendra, private communication, 1992.

$$V_b = I_o R_b = (Aqn_e v_d)\left(\frac{m_e \nu_c}{q^2 n_e}\frac{d_b}{A}\right) \qquad \text{(K.4)}$$

$$\rightarrow V_b \propto v_d \nu_c d_b \propto (pd_b)^{1/4} \quad .$$

The sheath width dependence on operating parameters is estimated in a similar way, using Equation K.2. The sheath width should be proportional to the distance the electron can travel within one rf cycle or the electron oscillation amplitude at the sheath edge. An estimate for the oscillation amplitude is the product of the electron velocity and rf time period or

$$\ell_s \propto \tau_{rf} v_d \propto \frac{1}{\omega\sqrt{pd_b}} \quad . \qquad \text{(K.5)}$$

The resistors, capacitors, and inductors used to represent the plasma electrical properties in this thesis have also been used by others (Godyak *et al.* 1991b). This appendix describes how the expressions for the bulk inductance, L_b, and the sheath resistance, R_s, terms for this thesis are derived. The bulk resistance, R_b, and the sheath capacitance, C_s, expressions are found in any elementary physics text (*e.g.* Halliday & Resnick 1981).

The bulk inductance, L_b, arises from the inertial effects of charge carriers (ions or electrons) in a time varying electric field. Assuming that the electric field varies as

$$\mathscr{E} = \frac{|V|}{d} e^{i\omega t} \quad , \qquad \text{(K.6)}$$

the force balance on particle j is

$$F = m_j \frac{dv}{dt} = q\mathscr{E} = q\frac{|V|}{d} e^{i\omega t} \qquad \text{(K.7)}$$

Combining Equation K.7 with the expression for current,

$$I_j = An_jqv_j \quad \rightarrow \quad \frac{dI_j}{dt} = An_jq\frac{dv_j}{dt} \quad , \tag{K.8}$$

results in

$$\frac{dI_j}{dt} = \frac{An_jq^2|V|}{m_jd}e^{i\omega t} \quad . \tag{K.9}$$

From the definition for inductance, the current can also be written as

$$|V|e^{i\omega t} = L\frac{dI_j}{dt} \quad \rightarrow \quad \frac{dI_j}{dt} = \frac{|V|}{L}e^{i\omega t} \quad . \tag{K.10}$$

Equating Equations K.9 and K.10 gives the equation for inductance,

$$L = \frac{m_j}{n_jq^2}\frac{d}{A} \quad . \tag{K.11}$$

The sheath resistance for both sheaths in the plasma, R_s, is derived by setting the power deposited in the sheath equal to the energy lost via ions striking the electrode and then combining that with the assumption that displacement current dominates in the sheath. The results is

$$\begin{aligned} P_s = \frac{1}{2}I_o^2R_s = 2I_iV_s \quad \text{and} \quad V_s = \frac{\ell_s}{A\varepsilon_o\omega}I_o \\ \rightarrow \quad R_s = \frac{4}{A\varepsilon_o\omega}\frac{I_i}{I_o} \quad . \end{aligned} \tag{K.12}$$

APPENDIX L

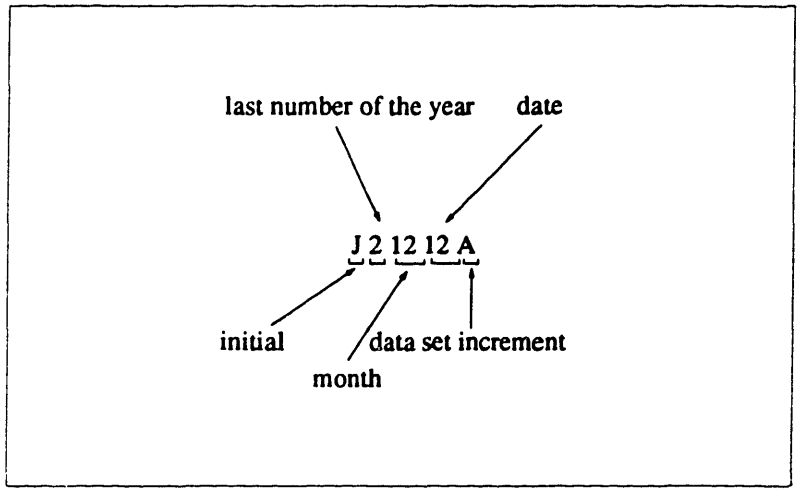
PROGRAMS

Documentation for ACQ.PAS (September 1992)

This program controls the stepping motors for moving the reactor, sends signals to the Spectral Link to move the monochromator grating, and ramps the voltage for grid # 2 of the ion analyzer. Data signals are taken using the IEEE 488 computer interface board which link the computer to the Keithley picoammeter and the LeCroy oscilloscope. Time averaged plasma emission signals and the measured ion current are taken from the picoammeter. The voltage and current waveforms are taken form the oscilloscope.

The units in ACQ.PAS are: IONE.PAS, POWER.PAS, PIE.PAS, ATOOLS.PAS, FTOOLS.PAS

Data files names are generated using the following format:



The data files are stored in the following directory with the following extension.

Data Subject	Directory	File Extension
experiment operating conditions	C:\EXPERIME\INFO\	*.SYS
ion acquisition parameters	C:\EXPERIME\IE\	*.INF
ion data	C:\EXPERIME\IE\	*.IE
voltage and current waveforms	C:\EXPERIME\WAVEFORM\	*.VWF, *.IWF
plasma acquisition parameters	C:\EXPERIME\PIE\	*.PIE
spatial emission scan data	C:\EXPERIME\PIE\	*.SPA
wavelength emission scan data	C:\EXPERIME\PIE\	*.WAV

```
PROGRAM ACQ;

        {This program uses three data acquisition units:  IONE and POWER
         and PIE.
         IONE takes ion energy data from the Keithley ammeter.
         POWER takes current and voltage waveform data from the LeCroy 9400.
         PIE records plasma emission from the Keithley ammeter, moves
             the grating via the Spectra Link, and moves the reactor
             using the stepping motors.
         Data is saved under: Initial+Year(1)+Month(2)+Data(2)+Letter.
              All data are saved in records and must use WriteDat.Pas and
              WriteWav.Pas to convert into a form usable with Symphony or
              Grapher.
         Ion energy data, both differentiated and raw, are saved in
              C:\EXPERIME\IONE\filename.IE in ASCII format.
         Ion energy acquisition parameters are saved in record format in
              C:\EXPERIME\IONE\filename.INF
         Waveform data in record format are saved in
              C:\EXPERIME\WAVEFORM\filename.IWF and .VWF
         Plasma Emission data, in ASCII, are saved in
              C:\EXPERIME\PIE\filename.SPA and .WAV
         Plasma emission data aquisition information are stored in record
              format in C:\EXPERIME\PIE\filename.PIE
         System information is saved in record format
              in C:\EXPERIME\INFO\filename.SYS

         ********IMPORTANT*********
         this program can be run only from c: drive because the IEEE will
         crash otherwise.
         *************************
         Written 9/88 --- J.Liu}


USES DOS, Crt, GDriver, GKernel, GWindow, GShell,
     ATools, FTools, IonE, Power, PIE;

VAR
   IsExitProgram, AreSysParam, IsFileSaved, IsFileReplace : boolean;
   SaveData : boolean;
   FuncKey : integer;
   Initial : str8;
   Year, Month, Day : str8;

   CurrentPoint : DataPointer;
   ExitSave : pointer;

   DirInfo : SearchRec;
   reply   : char;


{SF+}
PROCEDURE MyExit; {SF-}

          {The program always uses this procedure before exiting instead of
           its default exit.  This will make sure the program gets out of
           the Turbo Graphix and close all files before ending.}

var
   CurrentPoint : DataPointer;

begin

 {Print exit code}
   ClearScreen;
   if (ExitCode <> 0) then begin
      GotoXY (10,10);
      Write ('Exit Code is: ', ExitCode);
      Sound (1000);
      Delay (2000);
      NoSound;
   end;

 {Terminate graphics}
   LeaveGraphic;

 {Terminate comunication via IOTech card}
   Writeln (IeeeOut, 'RESET');
   Writeln (IeeeOut, 'LOCAL');
   Close (IeeeOut);
   Close (IeeeIn);

 {Erase all the pointer lists}
   EraseRunData (RunData);
   EraseData (DistributionData.DistFirstPoint, DistributionData.DistLastPoint);
   EraseData (VFirstPoint, VLastPoint);
   EraseData (IFirstPoint, ILastPoint);
   EraseData (OFirstPoint, OLastPoint);
   With OpticalData do begin
```

```
      EraseData (SpaFirstPoint,SpaLastPoint);
      EraseData (WavFirstPoint,WavLastPoint);
   end;

 {Set all DA to zero}
      Port[$21A] := Lo (2048);
      Port[$21B] := Hi (2048);
      Port[$21C] := 0;
      Port[$21D] := 0;
      Port[$21E] := 0;

 {Return to system termination routine}
   ExitProc := ExitSave;
end;


PROCEDURE InitScreen;

         {Write program entering screen}

begin
   ClearScreen;
   SelectWindow (6);
   DrawBorder;
   GotoXY (25,5);
   Write ('DATA ACQUISITION PROGRAM');
end;


PROCEDURE WriteCommands;

         {This writes the choices possible from main section}

begin
   GotoXY (5,18);
   Write ('F1=System Parameters');
   GotoXY (5,19);
   Write ('F2=Save Data');
   GoToXY (5,20);
   Write ('F3=Switch Screens');
   GotoXY (5,21);
   Write ('F10=Exit');
end;


PROCEDURE WriteSysScreen (var SysParam : SysRecord;
                          var Status   : StatusRecord);

         {This writes the current values of system parameters such
          as pressure, gas, gas flow rate, electrode spacing and diamter,
          and frequency.}

begin
   ClearScreen;

   GotoXY (30,1);  Write ('Current File:  ', Status.FileName);

   SetHeaderOn;
   DefineHeader (2,'Ion Energy');
   SelectWindow (2);
   DrawBorder;
   DefineHeader (3,'Power Waveform');
   SelectWindow (3);
   DrawBorder;
   DefineHeader (4,'Optical Spectrum');
   SelectWindow (4);
   DrawBorder;
   DefineHeader (9,'Communication Window');
   SelectWindow (9);
   DrawBorder;
   SetHeaderOff;

   WriteCommands;

   GotoXY (5,10);  Write ('Pressure:');
   GotoXY (5,11);  Write ('Frequency:');
   GotoXY (45,10);  Write ('Electrode Sep:');
   GotoXY (45,11);  Write ('Upper Elec Diam:');
   GotoXY (45,12);  Write ('Lower Elec Diam:');
   GotoXY (5,13);  Write ('Gas:');
   GotoXY (21,13);  Write ('Reading:');
   GotoXY (45,13);  Write ('Gas Flow:');

   With SysParam do begin
      GotoXY (20,10); Write (Pressure:6:1, ' (mtorr)');
      GotoXY (20,11); Write (Frequency:5:2, ' (MHz)');
      GoToXY (63,10); Write (ElecSep:4:2, ' (cm)');
```

```
      GoToXY (63,11); Write (UpElecDiam:4:1, ' (in)');
      GoToXY (63,12); Write (LowElecDiam:4:1, ' (in)');

      GoToXY (10,13); Write (GaslName);
      GoToXY (31,13); Write (GaslMeterReading:5:3);
      GoToXY (56,13); Write (GaslFlow:6:3, ' (sccm)');

      GotoXY (4,14);  Write ('Description');
      GotoXY (16,14);  Write ('-', Description1);
      GotoXY (16,15);  Write ('-', Description2);
      GotoXY (16,16);  Write ('-', Description3);
   end;
end;


PROCEDURE GetSysParam (var SysParam : SysRecord);

         {This section gets from the user the values of system paramters.}

var
   InputString : str8;
   code : integer;

begin
   ClearCommWindow (9);
   GotoXY (5,18);
   Write ('Input System Parameters');
   with SysParam do begin

      Str (Pressure:6:1, InputString);
      GotoXY (4,10); Write ('*');
      GetString (InputString, Numeric, 20,10);
      Val (InputString, Pressure, code);
      GotoXY (4,10);   Write (' ');
      GotoXY (20,10);  Write ('                    ');
      GotoXY (20,10);  Write (Pressure:6:1, ' (mtorr)');

      Str (Frequency:5:2, InputString);
      GotoXY (4,11); Write ('*');
      GetString (InputString, Numeric, 20, 11);
      Val (InputString, Frequency, code);
      GotoXY (4,11);   Write (' ');
      GotoXY (20,11);  Write ('              ');
      GotoXY (20,11);  Write (Frequency:5:2, ' (MHz)');

      Str (ElecSep:4:2, InputString);
      GotoXY (44,10); Write ('*');
      GetString (InputString, Numeric, 63,10);
      Val (InputString, ElecSep, code);
      GotoXY (44,10);  Write (' ');
      GotoXY (63,10);  Write ('              ');
      GotoXY (63,10);  Write (ElecSep:4:2, ' (cm)');

      Str (UpElecDiam:4:1, InputString);
      GotoXY (44,11); Write ('*');
      GetString (InputString, Numeric, 63, 11);
      Val (InputString, UpElecDiam, code);
      GotoXY (44,11);  Write (' ');
      GotoXY (63,11);  Write ('              ');
      GotoXY (63,11);  Write (UpElecDiam:4:1, ' (in)');

      Str (LowElecDiam:4:1, InputString);
      GotoXY (44,12); Write ('*');
      GetString (InputString, Numeric, 63, 12);
      Val (InputString, LowElecDiam, code);
      GotoXY (44,12);  Write (' ');
      GotoXY (63,12);  Write ('              ');
      GotoXY (63,12);  Write (LowElecDiam:4:1, ' (in)');

      InputString := GaslName;
      GotoXY (4,13); Write ('*');
      GetString (InputString, Mix, 10,13);
      GaslName := InputString;
      GotoXY (4,13);  Write (' ');
      GotoXY (10,13);  write ('    ');
      GotoXY (10,13);  Write (GaslName);

      Str (GaslMeterReading:5:3, InputString);
      GotoXY (20,13); Write ('*');
      GetString (InputString, Numeric, 31,13);
      Val (InputString, GaslMeterReading, code);
      GotoXY (20,13);  Write (' ');
      GotoXY (31,13);  Write ('        ');
      GotoXY (31,13);  Write (GaslMeterReading:5:3);

      Str (GaslFlow:6:3, InputString);
      GotoXY (44,13); Write ('*');
      GetString (InputString, Numeric, 56, 13);
```

```
      Val (InputString, GasFlow, code);
      GotoXY (44,13);  Write (' ');
      GotoXY (56,13);  Write ('              ');
      GotoXY (56,13);  Write (GasFlow:6:3,   '(sccm)');

      GotoXY (15,14);
      Write ('*');
      GetDescription (Description1,17,14);
      GotoXY (15,14);
      Write (' -', Description1);

      GotoXY (15,15);
      Write ('*');
      GetDescription (Description2,17,15);
      GotoXY (15,15);
      Write (' -', Description2);

      GotoXY (15,16);
      Write ('*');
      GetDescription (Description3,17,16);
      GotoXY (15,16);
      Write (' -', Description3);
      ClearCommLine (19);
   end;
end;


PROCEDURE NewFileName (var FileName : str7; var IsFileReplace : boolean);

var
   TempFile1 : str8;
   TempFile2 : str7;
   reply     : char;

begin
   TempFile1 := ' ';
   TempFile2 := ' ';

   GotoXY (5,21);  Write ('Enter Filename:                                    ');
   repeat
      GotoXY (21,21);  Write ('          ');
      GotoXY (5,22);  Write ('                                                ');
      GetString (TempFile1, mix, 21, 21);
      TempFile2 := TempFile1;
      GotoXY (5,22);  Write ('Should data be saved in ', TempFile2,'? (y/n)');
      reply := UpCase (readkey);
   until ((reply = 'Y') or (reply = chr(27)));

   If reply = chr(27) then IsFileReplace := false;

   If (IsFileReplace) then begin
      FindFirst (Path+'ie\'+TempFile2+'.*', Anyfile, DirInfo);
      If (DosError = 0) then begin
         GotoXY (5,23);  Write ('File already exist.  Replace?  (Y/N):');
         reply := UpCase (readkey);
         If (reply = 'N') then IsFileReplace := false;
      end else begin
         FindFirst (Path+'waveform\'+TempFile2+'.*', Anyfile, DirInfo);
         If (DosError = 0) then begin
            GotoXY (5,23);  Write ('File already exist.  Replace?  (Y/N):');
            reply := UpCase (readkey);
            If (reply ='N') then IsFileReplace := false;
         end;
      end;
   end;
   If (IsFileReplace) then FileName := TempFile2;
end;


BEGIN
 {Insert custom termination routine before system termination routine}
   ExitSave := ExitProc;
   ExitProc := @MyExit;

 {Initiate Turbo Pascal Graphics Toolbox Routines}

   InitGraphic;
   ClearScreen;
   SetUpWindows;

 {Initialize parameters and communication and screen}
   InitScreen;
   InitCommun;

 {Generate filename}
   GetFileName (Status, Date, 1);

 {Write the system screen}
```

```
  WriteSysScreen (SysParam, Status);

{Start the decision loop}
  IsExitProgram := false;
  AreSysParam   := false;
  IsFileSaved   := false;
  repeat

   {Obtain the desired function}
     FuncKey := 1;
     if keypressed then GetFuncKey (FuncKey);

     Case FuncKey of
       {Get system parameters}
         59:begin
              GetSysParam (SysParam);
              AreSysParam := true;
               WriteCommands;
            end;

       {Save the obtained data}
         60:begin
              ClearCommWindow (9);
              if (not(AreSysParam)) then GetSysParam (SysParam);
              AreSysParam := true;
              SaveData := false;
              Lpos := 18;
              With Status, OpticalData.PIEParam, OpticalData do begin
                 if ((IsIonESpectra and IsVWaveSpectra and IsIWaveSpectra) and
                    (IsSpatialScan or IsWavelengthScan)) then
                    SaveData := true
                 else begin
                    GotoXY (5,Lpos);
                    if not(IsIonESpectra) and not(IsVWaveSpectra) and
                       not(IsIWaveSpectra) and not(IsSpatialScan) and
                       not(IsWavelengthScan) then begin
                       Write ('NO DATA TO SAVE                                    ');
                       Sound (1000);  Delay (1000);  NoSound;
                    end
                    else begin
                       Sound (700);  Delay (700);  NoSound;
                       if not(IsIonESpectra) then begin
                          Write ('No Ion Energy Data to save.');
                          Lpos := succ(Lpos);
                       end;
                       GotoXY (5,Lpos);
                       if not(Status.IsVWaveSpectra) then begin
                          Write ('No voltage waveform to save.');
                          Lpos := succ(Lpos);
                       end;
                       GotoXY (5, Lpos);
                       if not (Status.IsIWaveSpectra) then begin
                          Write ('No current waveform to save.');
                          Lpos := succ(Lpos);
                       end;
                       GotoXY (5, Lpos);
                       if not(IsWavelengthScan) and not(IsSpatialScan) then begin
                          GotoXY (5,Lpos);  Write ('No Optical Data to save.');
                          Lpos := succ(Lpos);
                       end;
                       GotoXY (5,24);  Write ('Hit F2 to save data.');
                       FuncKey := 1;
                       repeat until keypressed;
                       GetFuncKey (Funckey);
                       if FuncKey = 60 then SaveData := true
                         else begin
                            ClearCommWindow(9);
                            WriteCommands;
                         end;
                    end;
                 end;

                 if (SaveData) then begin
                    GotoXY (5,Lpos);  Write ('Should data be saved in ',Status.Filename,'? (y/n)');
                     reply := UpCase(readkey);
                     IsFileReplace := true;
                    If (reply = chr(27)) then IsFileReplace := false
                    else if not(reply = 'Y') then
                       NewFileName (Status.FileName, IsFileReplace);
                     If (IsFileReplace) then begin
                       SaveRun (SysParam, Status, WfmDataRec.PowerParam);
                       if (Status.IsIonESpectra) then begin
                          SaveIonE (RunData, DistributionData, IonEParam, Status);
                          EraseRunData (RunData);
                          EraseData(DistributionData.DistFirstPoint,
                                    DistributionData.DistLastPoint);
                       end;
                       if (Status.IsVWaveSpectra) then begin
```

```
                    SaveVWave (VFirstPoint, Status);
                    EraseData(VFirstPoint,VLastPoint);
                 end;
                 if (Status.IsIWaveSpectra) then begin
                    SaveIWave (IFirstPoint, Status);
                    EraseData(IFirstPoint,ILastPoint);
                 end;
                 If (IsOWaveSpectra) then begin
                    SavePIEData (PIEParam,SpatialScan,
                                 WavelengthScan, SpaFirstPoint,
                                 WavFirstPoint);
                    EraseData(SpaFirstPoint,SpaLastPoint);
                    EraseData(WavFirstPoint,WavLastPoint);
                 end;
                 FileNameGen (Status.Initial, Date, 1, Status.FileName);
                 AreSysParam := false;
                 Status.IsVWaveSpectra := false;
                 Status.IsIWaveSpectra := false;
                 Status.IsOWaveSpectra := false;
                 IsSpatialScan := false;
                 IsWavelengthScan := false;
                 Status.IsIonESpectra := false;
                 IsFileSaved := true;
                  WriteSysScreen (SysParam, Status);
              end else begin
                 ClearCommwindow (9);
                  WriteCommands;
               end;
            end;  {If (SaveData)}
         end;  {With (OpticalData, Status, Optical.PIEParam)}
      end;

{Switch to one of the data acquisition sections}
  61:begin
        ClearCommWindow (9);
        GotoXY (5,18); Write ('F1=Ion Energy');
         GotoXY (5,19); Write ('F2=Waveforms');
        GotoXY (5,20); Write ('F3=Optical Spectrum');
         GotoXY (5,21); Write ('F10=Exit');
        repeat
           FuncKey := 1;
           GetFuncKey (FuncKey);
        until ((FuncKey >= 59) or (FuncKey <= 59));

        Case FuncKey of
           59: IonESection (Status, IonEParam, RunData,
                            DistributionData, DistributionParam);
           60: PowerSection (Status, WfmDataRec);
           61: OpticalSection (Status, OpticalData);
        end;

        WriteSysScreen (SysParam, Status);
        if (Status.IsIonESpectra) then begin
            DrawSpectra (RunData.IonEFirstPoint,
                         IonEParam.Max, IonEParam.Min,
                         IonEParam.StartEv, IonEParam.EndEv,
                         2);
            DrawSpectra (DistributionData.DistFirstPoint,
                         DistributionParam.Max, DistributionParam.Min,
                         IonEParam.StartEv, IonEParam.EndEv, 2);
        end;
        if (Status.IsVWaveSpectra) then
           DrawSpectra (VFirstPoint,VWorldVar.Max,VWorldVar.Min,
                        VWorldVar.StartTime,VWorldVar.EndTime,3);
        if (Status.IsIWaveSpectra) then
           DrawSpectra (IFirstPoint,IWorldVar.Max,IWorldVar.Min,
                        IWorldVar.StartTime,IWorldVar.EndTime,3);
        With OpticalData do begin
           if (PIEParam.IsSpatialScan) then with SpatialScan do
              DrawSpectra (SpaFirstPoint,Max,Min,StartLocation,
                           EndLocation,4);
           if (PIEParam.IsWavelengthScan) then with WavelengthScan do
              DrawSpectra (WavFirstPoint,Max,Min,StartWave,EndWave,4);
        end;
        IsFileSaved := false;
     end;


{Exit the program}
  68:begin
        ClearCommWindow (9);
        if not(IsFileSaved) and (Status.IsIonESpectra or
           Status.IsIWaveSpectra or Status.IsVWaveSpectra or
           OpticalData.PIEParam.IsWavelengthScan or
           OpticalData.PIEParam.IsSpatialScan) then begin
           Sound (1100); Delay (500); NoSound;
           GotoXY (5,18);
           Write ('WARNING --- Data NOT Saved');
```

```
                end;

                GotoXY (5,19);
                Write ('Hit F10 to Terminate');
                GetFuncKey (FuncKey);
                if (FuncKey = 68) then IsExitProgram := true
                 else begin
                    ClearCommWindow (9);
                    WriteCommands;
                 end;
             end;

      end;  {case}

   until (IsExitProgram);

END.
```

```
Unit IonE;

INTERFACE
uses DOS, Crt, GDriver, GKernel, GWindow, GShell,
     ATools, FTools;

PROCEDURE IonESection (var Status           : StatusRecord;
                       var IonEParam        : IonERecord;
                       var RunData          : RunDataRecord;
                       var DistributionData : DistDataRecord;
                       var DistributionParam : DistriParamRecord);


IMPLEMENTATION
PROCEDURE IonESection (var Status     : StatusRecord;
                       var IonEParam  : IonERecord;
                       var RunData    : RunDataRecord;
                       var DistributionData  : DistDataRecord;
                       var DistributionParam : DistriParamRecord);
     {This procedure contains all the procedures used solely by the
      IonE method analysis.  These procedures c  ain system and
      acquisition parameters, and take data.}

CONST
   SmallestStepIonE = 0.061035156;
   XSlopeIonE   = 1.0;
   NegIntercept = 124.0;
   PosIntercept = 125.0;

VAR
   FuncKey, code          : integer;
   IsExitIonE, IsAcqParam : boolean;
   Found                  : boolean;   {used in RetrieveIED}
   IonEMax, IoneMin       : real;
   NewFileName            : str7;
   InputString            : str8;

   Intercept              : real;
   CurrentPoint           : DataPointer;
   Terminate              : boolean;


Procedure WriteIonEAcqParam;

begin
   GotoXY (4,18);  Write ('SYSTEM PARAMETERS');  {Write parameter headings}
   GotoXY (4,19);  Write ('Grid 1:');
   GotoXY (4,20);  Write ('Grid 3:');
   GotoXY (25,18);  Write ('ACQUISITION PARAM.');
   GotoXY (25,19);  Write ('Starting eV:');
   GotoXY (25,20);  Write ('Ending eV:');
   GotoXY (25,21);  Write ('Step eV:');
   GotoXY (25,22); Write ('Number of Steps:');
   GotoXY (25,23); Write ('Data per Step:');

   With IonEParam do begin                          {Write parameters}
      GotoXY (12,19); Write (Grid1:5:0, ' V');
      GotoXY (12,20); Write (Grid3:5:0, ' V');
      GotoXY (42,19);  Write (StartEv:6:1);
      GotoXY (42,20);  Write (EndEv:6:1);
      GotoXY (42,21); Write (StepEv:6:2);
      GotoXY (42,22); Write (NumberSteps:4:0);
      GotoXY (42,23); Write (PointsPerStep:2);
   end;
end;


procedure WriteIonEScreen (var IonEParam       : IonERecord;
                           var Status          : StatusRecord);
    {This procedure sets up the Ion Energy screen and writes all the system
     and acquisition parameters.}

begin
   ClearScreen;
   DefineHeader (8,'Ion Energy Distribution');
   SetHeaderOn;
   SelectWindow (8);
   DrawBorder;

   DefineHeader (1,'Commands');
   SetHeaderOn;
   SelectWindow (1);
   DrawBorder;

   DefineHeader (9,'Data Acquisition Parameters');
   SetHeaderOn;
   SelectWindow (9);
```

```
   DrawBorder;

   {Write file name at the top of the screen}
   GotoXY (30,1);  Write ('Current File:  ', Status.FileName);

  {Print permitted options}
   GotoXY (61,3);
   Write ('F1 = Acq. Param.');
   GotoXY (61,4);
   Write ('F2 = Get Data');
   GotoXY (61,5);
   Write ('F3 = Scan Data');
   GotoXY (61,6);
   Write ('F4 = Retrieve IED');
   GotoXY (61,14);
   Write ('F10 = Exit');

   WriteIonEAcqParam;
end;


Procedure GetIonEAcqParam (var IonEParam : IonERecord; var Terminate:boolean);

     {This procedure inputs the IonE acquisition parameters.  If <return> is
      entered as the first key, previous parameter settings are saved.
      Since the AD/DA converters can only be incremented by discrete
      values, not all starting, ending, and step values are possible.
      The routine adjusts these values based on the following:
         Total possible interger values for DA/AD converter is 4096
         The range of a scan is 0 -- 125 volts
      The smalest possible steps are saved in constant
      SmallestStepIonE and in the variable IonEParam.SmallestStepIonE}

var
   InputString : str8;
   code : integer;
   Reply : char;

begin
   repeat
      GotoXY (5,24);  Write ('                                                    ');
      with IonEParam do begin

         Str (Grid1:5:1, InputString);
         GotoXY (3,19); Write ('*');
         GetString (InputString, Numeric, 12, 19);
         Val (InputString, Grid1, code);
         GotoXY (3,19);  Write (' ');
         GotoXY (12,19);  Write (Grid1:5:0, ' V');

         Str (Grid3:5:1, InputString);
         GotoXY (3,20); Write ('*');
         GetString (InputString, Numeric, 12, 20);
         Val (InputString, Grid3, code);
         GotoXY (3,20);  Write (' ');
         GotoXY (12,20);  Write (Grid3:5:0, ' V');

         Str (StartEv:6:1, InputString);
         GotoXY (24,19); Write ('*');
         GetString (InputString, Numeric, 42, 19);
         Val (InputString, StartEv, code);
        {Prevent StartEv from being greater then 124.0 eV or less then -12.0}
         if (StartEv > 123.0) then StartEv := 123.0;
         if (StartEv < -12.0) then StartFv := -12.0;
         GotoXY (42,19); Write (StartEv:6:1);
         GotoXY (24,19); Write (' ');

         Str (EndEv:6:1, InputString);
         GotoXY (24,20); Write ('*');
         GetString (InputString, Numeric, 42, 20);
         Val (InputString, EndEv, code);
        {Prevent EndEv from being smaller that StartEv or greater then 124 eV}
         if (EndEv <= StartEv) then EndEv := StartEv+1.0;
         if (EndEv > 124.0) then EndEv := 124.0;
         GotoXY (42,20);  Write (EndEv:6:1);
         GotoXY (24,20); Write (' ');

         Str (StepEv:6:2, InputString);
         GotoXY (24,21); Write ('*');
         GetString (InputString, Numeric, 42, 21);
         Val (InputString, StepEv, code);
         if (StepEv = 0.0) then StepEv := 1.0;

         StepEv  := Round(StepEv/SmallestStepIonE)*SmallestStepIonE;
          StartEv := Round(StartEv/StepEv) * StepEv;
         if (StepEv < SmallestStepIonE) then StepEv := SmallestStepIonE;
         NumberSteps := Round((EndEv-StartEv)/StepEv)+1;
         EndEv := StartEv+(NumberSteps-1)*StepEv;
```

```
          GotoXY (24,21);  Write (' ');
          GotoXY (42,19); Write (StartEv:6:1);
          GotoXY (42,20);  Write (EndEv:6:1);
          GotoXY (42,21);  Write (StepEv:6:2);
          GotoXY (42,22);  Write (NumberSteps:4:0);

          Str (PointsPerStep:2, InputString);
          GotoXY (24,23); Write ('*');
          GetString (InputString, Numeric, 42, 23);
          Val (InputString, PointsPerStep, code);
          GotoXY (24,23); Write (' ');
          GotoXY (42,23); Write (PointsPerStep:2);

       end;
       GotoXY (5,24);
       Write ('Are the parameters correct ? (y or n)   ');
       Reply := readkey;
       If Reply = chr(27) then Terminate := true;
       Reply := Upcase (Reply);

    until ((Reply = 'Y') or Terminate);

    GotoXY (5,24);
    Write ('                                                  ');

end;


procedure GetIonEData (var IonEParam : IonERecord;
                       var RunData   : RunDataRecord);

     {The procedure obtains data from devices connected to the IEEE 488
       card. The acquired data is stored in a linked list.
       For a scan, only one DA set to 0 -- 5 V range is used.}

var
   OutInt          : integer;         {variable used for DA conversion}
   OutLow, OutHigh : byte;

   CurrentEv, OldEv                  : real;
   Y, OldY, tempY                    : double;
   CurrentScaleHigh, CurrentScaleLow : double;
   PointCount, code, FuncKey, count  : integer;
   IsInterrupt                       : boolean;

begin

 {Parameter initialization}
   PointCount := 0;
   With IonEParam do begin
      If (Round(StartEv) < 0) then Intercept := NegIntercept
      else Intercept := PosIntercept;
      CurrentEv := StartEv;
      Max := -1.0;
      Min :=  1.0;
   end;
   CurrentScaleHigh := 0.0000000001;
   CurrentScaleLow  := -0.000000000001;


 {Reset IEEE interface.  Initialize IEEE communication to the Keithley ammeter}
 InitKeithley;

 {Start acquisition loop}
   GotoXY (5,24);
   Write ('Taking Ion Energy Spectra');

   FuncKey := 1;

   repeat

    {Check for early termination}
      if keypressed then GetFuncKey (FuncKey);

      {Terminate}
      if (FuncKey = 68) then begin
       {Reset devices}
         Writeln (Ieeeout, 'CLEAR 22');
         Writeln (IeeeOut, 'LOCAL 22');
         Port[$21A] := Lo (2048);
         Port[$21B] := Hi (2048);
       {Set NumberSteps to actual number of points collected, and update}
       {the ending value}
         IonEParam.NumberSteps := PointCount;
         IonEParam.EndEv := (Pointcount-1)*IonEParam.StepEv + IonEParam.StartEv;
         EXIT;
      end;
```

```
    {Set DA to X position}
      OutInt  := Round((CurrentEv*XSlopeIonE+Intercept)
                       /SmallestStepIonE);
      OutLow  := Lo (OutInt);
      OutHigh := Hi (OutInt);
      Port[$21A] := OutLow;
      Port[$21B] := OutHigh;          {Start DA conversion}


      Delay (500);     {Allow time for grid voltage to be set before
                        collecting current}

      count := 0;
      tempY := 0.0;
      repeat

      {Start data collection, averaging over steps per point}
         {First check if the Keithley ammeter is ready}
         IsKeithleyReady;

         {Get data}
         Writeln (IeeeOut, 'ENTER 22');
         Readln (IeeeIn, Response);

        {Determine the real value of the data from the response}
         Response := RemoveSpaces (Response);
         Val (Response, Y, code);
         tempY := tempY + Y;
         count := count + 1;
      until (count = IonEParam.PointsPerStep);
      Y := tempY/count;

      {Find Max and Min}
      if (Y < IonEParam.Min) then IonEParam.Min := Y;
      if (Y > IonEParam.Max) then IonEParam.Max := Y;

      {Store value in a linked list}
      with RunData do begin
         if IonEFirstPoint = nil then begin
            New (IonEFirstPoint);
             IonELastPoint := IonEFirstPoint;
         end
         else begin
            New (IonELastPoint^.NextPoint);
            IonELastPoint := IonELastPoint^.NextPoint;
          end;

         IonELastPoint^.Xvalue := CurrentEv;
         IonELastPoint^.Yvalue := Y;
         IonELastPoint^.NextPoint := nil;              {Store data}
         PointCount := PointCount + 1;
      end;

      {Plot data on the screen with correct vertical scale}
      If PointCount = 1 then begin
         CurrentScaleHigh := 1.2*Y;
         DefineWorld (1, IonEParam.StartEv, CurrentScaleHigh,
                      IonEParam.EndEv, CurrentScaleLow);
         SelectWorld (1);
         SelectWindow (8);
      end;
      if ((Y > CurrentScaleHigh) or (Y < CurrentScaleLow)) then begin
         if (Y > CurrentScaleHigh) then CurrentScaleHigh := 1.2*Y;
         if (Y < CurrentScaleLow ) then CurrentScaleLow  := 1.2*Y;
         DefineWorld (1, IonEParam.StartEv, CurrentScaleHigh,
                      IonEParam.EndEv, CurrentScaleLow);
         SelectWorld (1);
         SelectWindow (8);
      end;
      If (PointCount <> 1) then DrawLine (OldEv, OldY, CurrentEv, Y);
      GotoXY (35,24);
      Write ('X: ', RunData.IonELastPoint^.Xvalue:8:3,'    Y: ', Y:12);

      {Set to next X position}
      OldEv := CurrentEv;
      OldY := Y;
      CurrentEv := CurrentEv + IonEParam.StepEv;
      {Correct for double zero values in D/A convertor when
       switching from negative to positive values by changing
       intercept}
      If CurrentEv > 0 then Intercept := PosIntercept;

    until (CurrentEv > IonEParam.EndEv);          {Terminate data acquisition}

  {Reset devices}
    Writeln (IeeeOut, 'CLEAR 22');
    Writeln (IeeeOut, 'LOCAL 22');
    Port[$21A] := Lo (2048);
```

```
    Port[$21B] := Hi (2048);
end;


PROCEDURE Differentiate (var IonEParam : IonERecord;
                         RunData : RunDataRecord;
                         var DistributionData : DistDataRecord;
                         var DistributionParam : DistriParamRecord);

{This procedure differentiates the collected data.  The difference of Y values
 is taken; then divided by the difference in step size of the x values.
 Therefore, the resulting Y value has units of amps/eV.  If the difference
 is negative, then the Y value is set equal to zero.  The last Y value is
 set equal to zero.}

var
   CurrentPoint, NewPoint : datapointer;
   Maximum, Minimum       : double;

begin
   GotoXY (35,24);  Write ('                                    ');
   GotoXY (2,24);   Write ('Differentiating data...        ');
   with RunData, DistributionData do begin
      if (DistFirstPoint = nil) then New (DistFirstPoint);
      DistLastPoint := DistFirstPoint;
      NewPoint := IonEFirstPoint^.NextPoint;
      DistFirstPoint^.Yvalue := (IonEFirstPoint^.Yvalue - NewPoint^.Yvalue)/IonEParam.StepEv;
      DistFirstPoint^.Xvalue := IonEFirstPoint^.Xvalue;
      If DistFirstPoint^.Yvalue < 0.0 then DistFirstPoint^.Yvalue := 0.0;
      Maximum := DistFirstPoint^.Yvalue;
      Minimum := DistFirstPoint^.Yvalue;
      CurrentPoint := NewPoint;
      NewPoint := CurrentPoint^.Nextpoint;
      While (NewPoint <> nil) do begin
         New (DistLastPoint^.NextPoint);
         DistLastPoint := DistLastPoint^.NextPoint;
         If (CurrentPoint^.Yvalue > NewPoint^.Yvalue) then begin
            DistLastPoint^.Yvalue := (CurrentPoint^.Yvalue - NewPoint^.Yvalue)/IonEParam.StepEv;
            If (DistLastPoint^.Yvalue > Maximum) then Maximum := DistLastPoint^.Yvalue;
            If (DistLastPoint^.Yvalue < Minimum) then Minimum := DistLastPoint^.Yvalue;
         end else DistLastPoint^.Yvalue := 0.0;
         DistLastPoint^.Xvalue := CurrentPoint^.Xvalue;
         DistLastPoint^.NextPoint := nil;
         CurrentPoint := NewPoint;
         NewPoint := CurrentPoint^.NextPoint;
       end;   {while}
      {set the y-value of the last point equal to zero}
      if (Newpoint = nil) then CurrentPoint^.Yvalue := 0.0;
   end;  {with}
   DistributionParam.Max := Maximum;
   DistributionParam.Min := Minimum;
   GotoXY (2,24);  Write ('                                    ');
end;  {procedure differentiate}


Procedure CalcEnergy(FirstPoint : DataPointer;
                     var IonEParam : IonERecord);

var
   CurrentPoint : DataPointer;
   AveIonEnergy : Double;
   TotalCurrent : Double;

begin
   CurrentPoint := FirstPoint;

   {remove current values taken at negative second grid voltages from list}
   If (CurrentPoint^.Xvalue < 0) then begin
      repeat
         CurrentPoint := CurrentPoint^.NextPoint;
      until (CurrentPoint^.Xvalue >= 0);
   end;

   AveIonEnergy := 0.0;
   TotalCurrent := 0.0;
   While (CurrentPoint <> nil) do begin
      AveIonEnergy := AveIonEnergy + CurrentPoint^.Xvalue * CurrentPoint^.Yvalue;
      TotalCurrent := TotalCurrent + CurrentPoint^.Yvalue;
      CurrentPoint := CurrentPoint^.NextPoint;
   end;
   IonEParam.IonEnergy := AveIonEnergy/TotalCurrent;
   GotoXY (50,18);  Write ('ION PROPERTIES');
   GotoXY (50,19);  Write('Average Energy = ',IonEParam.IonEnergy:6:1,' eV');
   GotoXY (50,20);  Write('Total Current =',TotalCurrent:9,' amp');
end;
```

```
Procedure RetrieveIED (var RetrievedData  : DistDataRecord;
                       var RetrievedParam : DistriParamRecord;
                        var Found    : boolean);

type
   DataRecord     = record
      Xvalue      : real;
      Yvalue      : double;
   end;

var
   RetrievedName : str8;
   RetrievedFile : Text;
   Data          : DataRecord;
   Dummy         : double;     {the total current at that x-value - not used}
   Path          : str60;
   DirInfo       : SearchRec;

begin
   Found := false;
   ClearCommWindow (9);
   GotoXY (5,18);  Write ('Enter Directory Path of File: ');
   GetDescription (Path, 35, 18);
   GotoXY (5,19);  Write ('Enter Filename of IED: ');
   GetString(RetrievedName, mix, 28, 19);
   FindFirst (Path+RetrievedName+'.DIS', AnyFile, DirInfo);
   If DosError = 0 then begin
      Found := true;
      GotoXY (5,20);  Write ('Retrieving data...');

      {Initialize variables}
      With RetrievedData, RetrievedParam do begin
         DistFirstPoint           := nil;
         DistLastPoint            := nil;
         DistLastPoint^.NextPoint := nil;
         Max                      := -1;
         Min                      :=  1;
      end;

      Assign (RetrievedFile, Path+RetrievedName+'.IE');
      Reset (RetrievedFile);
      repeat
         With Data do Readln (RetrievedFile, XValue, Dummy, YValue);
         With RetrievedData, RetrievedParam do begin
            If DistFirstPoint = nil then begin
               New (DistFirstPoint);
               DistLastPoint := DistFirstPoint;
            end else begin
               New (DistLastPoint^.NextPoint);
               DistLastPoint := DistLastPoint^.NextPoint;
            end;
            DistLastPoint^.Xvalue := Data.Xvalue;
            DistLastPoint^.Yvalue := Data.Yvalue;
            DistLastPoint^.NextPoint := nil;

            {Find Max. and Min. Yvalue}
            If Max < Data.Yvalue then Max := Data.Yvalue;
            If Min > Data.Yvalue then Min := Data.Yvalue;
         end;
      until (EoF(RetrievedFile));
      Close(RetrievedFile);
   end;
end;


BEGIN
   WriteIonEScreen (IonEParam, Status);
   if (Status.IsIonESpectra) then begin
      DrawSpectra (RunData.IonEFirstPoint, IonEParam.Max, IonEParam.Min,
                   IonEParam.StartEv, IonEParam.EndEv, 8);
      DrawSpectra (DistributionData.DistFirstPoint, DistributionParam.Max,
                   DistributionParam.Min, IonEParam.StartEv,
                   IonEParam.EndEv, 8);
   end;

   IsExitIonE := false;
   IsAcqParam := false;
   GotoXY (5,24);  Write ('Press Function Key ...');

{Repeat Ion Energy section}
   repeat

      FuncKey := 1;
      if keypressed then GetFuncKey (FuncKey);
      Case Funckey of

         {F1 = Acq. Param.}
         59:begin
```

```
      GotoXY (5,24);  Write ('                          ');
      GetIonEAcqParam (IonEParam, Terminate);
      IsAcqParam := true;
      GotoXY (5,24);  Write ('Press Function Key ...');
   end;

{F2=Take data}
60:begin
      GotoXY (5,24);  Write ('                          ');
      Terminate := false;
      If not (IsAcqParam) then GetIonEAcqParam (IonEParam, Terminate);
      IsAcqParam := true;

      If not(Terminate) then begin

         Status.IsIonESpectra := false;
       {Remove old data}
         EraseData (RunData.IonEFirstPoint, RunData.IonELastPoint);
         EraseData (DistributionData.DistFirstPoint,
                    DistributionData.DistLastPoint);
         GetIonEData (IonEParam, RunData);

         Differentiate (IonEParam, RunData, DistributionData,
                        DistributionParam);

         WriteIonEScreen (IonEParam, Status);

         CalcEnergy (DistributionData.DistFirstPoint, IonEParam);

         DrawSpectra (RunData.IonEFirstPoint,
                      IonEParam.Max, IonEParam.Min,
                      IonEParam.StartEv, IonEParam.EndEv,
                      8);

         DrawSpectra (DistributionData.DistFirstPoint,
                      DistributionParam.Max, DistributionParam.Min,
                      IonEParam.StartEv, IonEParam.EndEv,
                      8);

         Status.IsIonESpectra := true;
      end;
      GotoXY (5,24);  Write ('Press Function Key ...');
   end;

{F3=Analyze data}
61:begin
      GotoXY (5,24);  Write ('                          ');
      if (Status.IsIonESpectra) then begin
         FuncKey := 1;
         GotoXY (5,24);
         Write ('F1 = rawdata   F2 = distribution                                  ');
         repeat until keypressed;
         GetFuncKey (FuncKey);
         ClearCommWindow(9);
         Case FuncKey of
            59:With IonEParam do ScanData (RunData.IonEFirstPoint,
                  StartEv,EndEv,Max,Min,8);
            60:With IonEParam, DistributionParam do
                  ScanData (DistributionData.DistFirstPoint,StartEv,
                  EndEv,Max,Min,8);

         end;  {case}
         WriteIonEScreen (IonEParam, Status);
         DrawSpectra (RunData.IonEFirstPoint,
                      IonEParam.Max, IonEParam.Min,
                      IonEParam.StartEv, IonEParam.EndEv,
                      8);

         DrawSpectra (DistributionData.DistFirstPoint,
                      DistributionParam.Max, DistributionParam.Min,
                      IonEParam.StartEv, IonEParam.EndEv,
                      8);

      end
      else begin
          Sound (700);
         GotoXY (5,24);  Write ('No Data to Analyze');
          Delay (1500);  NoSound;
      end;
      GotoXY (5,24);  Write ('Press Function Key ...');
   end;   {Analyze}

{F4 = Retrieve IED}
62:begin
      GotoXY (5,24);  Write ('                          ');
      RetrieveIED(RetrievedData, RetrievedParam, Found);
      If Found then begin
         DrawSpectra (RetrievedData.DistFirstPoint,
```

```
                                RetrievedParam.Max, RetrievedParam.Min,
                                RetrievedData.DistFirstPoint^.Xvalue,
                                  RetrievedData.DistLastPoint^.Xvalue,
                                8);
                        EraseData (RetrievedData.DistFirstPoint,
                                RetrievedData.DistLastPoint);
                    end else begin
                        Sound (700);
                        GotoXY (5,22);  Write ('File NOT FOUND');
                        Delay (1500);  NoSound;
                    end;
                    ClearCommWindow (9);
                    WriteIonEAcqParam;
                   GotoXY (5,24);  Write ('Press Function Key ...');
                end;


            {F10=Exit}
            68:begin
                    IsExitIonE := true;
               end;

        end;  {case}
    until (IsExitIonE);

end;  {procedure}
END. {unit}
```

```
Unit Power;
     {The procedure obtains data from the LeCroy oscilloscope connected
      to the IEEE 488 card.  The acquired data is stored in a linked list.
      Then immediately stored in c:\experime\waveform directory under the
      extension of .IWF or .VWF.  Then the data, acquired from LeCroy 9400
      in binary numbers, is converted to real numbers and stored in a linked
      list.

      The data transfers for the LeCroy 9400 are handled using an IOTech 488
      GPIB card in the buffered format for high speed transfers.  Default
      LeCroy 9400 data, descriptor, and trigger records are used in the data
      transfer.  Specific types of transfers therefore are not supported in
      this acquisition package.

      L. Baston -- Rev 0 -- 3/7/88
      J. Liu -- Revised Sept. 1988
      J. Liu -- Rev 1.1 -- May 1, 1991}


INTERFACE
uses DOS, Crt, GDriver, GKernel, GWindow, GShell, ATools, FTools;

PROCEDURE PowerSection (var Status     : StatusRecord;
                        var wfmdatarec : waveform);



IMPLEMENTATION
PROCEDURE PowerSection (var Status     : StatusRecord;
                        var wfmdatarec : waveform);
     {This procedure contains all the procedures used solely by the
      Power Unit.  These procedures obtain system and
      acquisition parameters, and take current and voltage waveform
      data.}

VAR
   FuncKey, code : integer;
   InputString   : str8;
   IsSpectra     : boolean;
   Terminate     : boolean;
   reply         : char;
   Start, Finish : integer;


procedure GetPowerSysParam (var WfmDataRec:Waveform);
     {This procedure inputs the Power system parameters.  If <return> is
      entered as the first key, previous parameter setings are saved.}

var
   InputString : str8;
   code        : integer;
   frequency   : real;

begin
   ClearCommWindow (6);
   with WfmDataRec.PowerParam do begin
      frequency := 0.133636*VoltageMultiplier - 134.75888;
      GotoXY (35,18);  Write ('Frequency (MHz)     : ', Frequency:6:2);
      GotoXY (35,19);  Write ('Voltage Multiplier : ', VoltageMultiplier:6:1);
      GotoXY (35,20);  Write ('Current Multiplier : ', CurrentMultiplier:6:1);
   end;

   GotoXY (5,23);
   Write ('Input Voltage and Current Parameters                          ');

   repeat
      with WfmDataRec.PowerParam do begin
         Str(Frequency:6:2, InputString);
         GotoXY (55,18); Write ('*');
         GetString (InputString, Numeric, 56, 18);
         Val (InputString, Frequency, code);
         GotoXY (55,18);  Write ('              ');
         GotoXY (56,18);  Write (Frequency:6:2);

         VoltageMultiplier := 7.483*frequency + 1008.4007;
         GotoXY (56,19);  Write (VoltageMultiplier:6:1);

         Str(CurrentMultiplier:6:1, InputString);
         GotoXY (55,20); Write ('*');
         GetString (InputString, Numeric, 56, 20);
         Val (InputString, CurrentMultiplier, code);
         GotoXY (55,20);  Write ('              ');
         GotoXY (56,20);  Write (CurrentMultiplier:6:1);
      end;

      GotoXY (5,23);
      Write ('Are these parameters correct ? (y or n)');
```

```
   until (readkey = 'y');

   With WfmDataRec.PowerParam do begin
      GotoXY(61,13);
      Write('Voltage X = ', VoltageMultiplier:6:1);
      GotoXY(61,14);
      Write('Current X = ', CurrentMultiplier:6:1);
   end;

   ClearCommWindow (6);

end;


Procedure IOCTL;

{This procedure issues a Break to the IOTech 488 Card and
 sends it to its quiescent state.}

   const
      break:str255='BREAK';          {IO Tech 488 Break Code}
   var
      IeeeFileHandle:integer absolute IeeeOut;{Memory Addr of IeeeOut File}

   {Code adapted from IO Tech 488 Software for Turbo Pascal 3.0}

   begin
      with regs do begin
         AX:=$4403;
         BX:=IeeeFileHandle;
         CX:=Length(Break);
         DS:=Seg(Break);
         DX:=Ofs(Break)+1
      end;
      MsDos(Regs)
   end;


Function SwapLongInt(var LongIntValue:LongInt):LongInt;

{This function Swaps the high and low bytes of a long integer (32-bit).
 This swapping is necessary because the LeCroy 9400 outputs the most
 significant bytes first and the computer memory needs the bytes stored
 with the most significant bytes in the highest memory address and therefore
 stored last.  Byte Swapping occurs after transfer from 9400 to computer and
 prior to transfer from computer to 9400.}


   type
      LongIntArray=array[1..4] of byte;{array of sequential bytes of long int}

   var
      LongIntHolder:LongInt;         {Holder of long integer value}
      LongIntAbsArr:LongIntArray absolute LongIntValue;
                                     {Memory Space of long integer value to
                                      be swapped}
      LongIntArrHolder:LongIntArray absolute LongIntHolder;
                                     {Memory Space of sequential bytes of
                                      long integer}
      ArrIndx:integer;               {Counter}

   begin

      {Reverse storage of bytes in seqential array of bytes composing
       the long integer.  Redefine the long integer with the new order.}

      For ArrIndx:=1 to 4 do
         LongIntArrHolder[ArrIndx]:=LongIntAbsArr[5-ArrIndx];
      SwapLongInt:=LongIntHolder;
   end;


Procedure IEEEReset;

{This function resets the IOTech 488 Card and sends all devices on
 the bus to local.}

begin
   IOCTL;
   Writeln(IeeeOut,'RESET');
   Writeln(IeeeOut,'LOCAL');
end;


Procedure IeeeReinit;

{This procedure reinitializes the IEEE ports and resets the bus.}
```

```
begin
   Rewrite(IeeeOut);
   Reset(IeeeIn);
   IeeeReset;
end;


Procedure CheckInt(Signal:integer);

{This procedure is an error handler written for the IOTech 488 communication
 with the LeCroy 9400 Oscilloscope.  This procedure checks to see if an
 interrupt has been reported and gives an error message accordingly.  The
 signal value of the procedure is given by the place in the program that
 the interrupt was detected.  Throughout the program, calls to this subroutine
 have been made after Input/Output with the Ieee488 have been completed and
 each call has been given a different arbitary number.  That number should
 only appear once within the program and thus hopefully a trace of where
 the error occured can be found.

 The IOTech 488 redefines the "light pen interrupt" of the computer to be
 the signal for the IEEE bus related interrupt (such as SRQ).  Since Turbo
 Pascal doesn't have the equivalent of "ON PEN" as does BASIC, a direct
 MS DOS interrupt is used to get the "light pen interrupt."}

  var
     sp,st9400:integer;              {Serial Poll Value, Status 9400}
     valcode:integer;                {Error Code returned with String Convert}
     StatusByte:byte;                {Requested 9400 Status Value}
     StrStatusByte:str255;           {String Status Value}

  begin
     LPos:=17;

     {Set Register Address for Checking Light Pen Status, Function 4}

     Regs.AX:=$0400;

     {BIOS interrupt $10}

     Intr($10,Regs);


     While Regs.AH<>0 do begin

        {A Interrupt has Occured}

        GotoXY(5,LPos);
        LPos:=Succ(LPos);
        Writeln('Interrupt detected at signal ',Signal);
        GotoXY(5,LPos);
        LPos:=Succ(Lpos);

        {Conduct Serial Poll of IOTech 488}

        Writeln(IeeeOut,'SPOLL');
        Readln(IeeeIn,sp);
        If sp=0 then begin
           writeln('Non-SRQ Interrupt!'); Halt
        end;

        {Conduct Serial Poll of LeCroy 9400}

        Writeln(IeeeOut,'SPOLL 04');
        Readln(IeeeIn, st9400);
        If (st9400 and 64)=0 then begin
           writeln('Non-9400 SRQ!'); Halt
        end;
        If (st9400 and 32)=0 then begin
           if (st9400 and 1)<>0 then writeln('overflow');
           if (st9400 and 2)<>0 then writeln('buffer full');
           if (st9400 and 4)<>0 then writeln('buffer 1/2 full');
           if (st9400 and 8)<>0 then writeln('readking done');
           if (st9400 and 16)<>0 then writeln('busy');
        end else begin
          if (st9400 and 4)<>0 then begin
             writeln(IeeeOut,'OUTPUT 04;STB 3, ?');
             writeln(IeeeOut,'ENTER 04');
              readln(IeeeIn,StrStatusByte);
              StrStatusByte:=Copy(StrStatusByte,Pos('S',StrStatusByte)+4,
                                                Pos('S',StrStatusByte)+7);
              Val(StrStatusByte,StatusByte,ValCode);
              Case StatusByte of
                0: writeln('No softkey pressed');
                1..9: writeln('softkey ',StatusByte,' pressed');
                10: writeln('Call to Host made');
             else
                writeln('SOFTKEY PRESSED detected of unknown origin');
             end;
```

```
            end;
            if (st9400 and 8)<>0 then begin
               writeln('9400 INTERNAL STATE has changed');
               writeln(IeeeOut,'OUTPUT 04;STB 4, ?');
               writeln(IeeeOut,'ENTER 04');
               readln(IeeeIn,StrStatusByte);
               StrStatusByte:=Copy(StrStatusByte,Pos('S',StrStatusByte)+4,
                                                  Pos('S',StrStatusByte)+7);
               Val(StrStatusByte,StatusByte,ValCode);
               Case StatusByte of
                  0: writeln('9400 not acquiring data');
                  1: writeln('Channel 1 is in overload');
                  2: writeln('Channel 2 is in overload');
                  3: writeln('First Sequence Sweep triggered');
              else
                  writeln('INTERNAL STATE change detected of unknown origin');
              end;
            end;
            if (st9400 and 16)<>0 then begin
               writeln(IeeeOut,'OUTPUT 04;STB 5, ?');
               writeln(IeeeOut,'ENTER 04');
               readln(IeeeIn,StrStatusByte);
               StrStatusByte:=Copy(StrStatusByte,Pos('S',StrStatusByte)+4,
                                                  Pos('S',StrStatusByte)+7);
               Val(StrStatusByte,StatusByte,ValCode);
               Case StatusByte of
                  0: writeln('9400 screen dump complete');
                  1: writeln('no waveform processing option');
                  2: writeln('calibration complete');
                  3: writeln('averageing complete');
               else
                  writeln('OPERATION COMPLETE detected of unknown origin');
               end;
            end;
            if (st9400 and 32)<>0 then begin
               writeln(IeeeOut,'OUTPUT 04;STB 6, ?');
               writeln(IeeeOut,'ENTER 04');
               readln(IeeeIn,StrStatusByte);
               StrStatusByte:=Copy(StrStatusByte,Pos('S',StrStatusByte)+4,
                                                  Pos('S',StrStatusByte)+7);
               Val(StrStatusByte,StatusByte,ValCode);
               Case StatusByte of
                  0: writeln('No error');
                  1: writeln('All responses not read, output buffer flushed');
                  10: writeln('Invalid separator or too many parameters');
                  11: writeln('Invalid header');
                  12: writeln('Invalid number format');
                  13: writeln('Invalid keyword');
                  14: writeln('Invalid block');
                  15: writeln('Two or more strings in the same command');
                  20: writeln('Command permission error');
                  30: writeln('Command for option not installed');
                  40: writeln('Semantic error: false number of parameters or false parameter command');
                  50: writeln('Environment error: 9400 is not set to the proper status for command');
                  60: writeln('Descriptor error: inconsistency with data received with WRITE command');
                  100: writeln('Command not yet implemented');
               else
                  writeln('ERROR detected with unknown origin');
               end;
            end;
         end;

         {Check to see if another Interrupt has Occured}

         Regs.AX:=$0400;
         Intr($10,Regs);
      end;
   end;

Procedure GetWaveform(var wfmdatarec:waveform);

{This procedure retrieves the complete waveform from the requested channel
 of the LeCroy 9400 oscilliscope.  The waveform is retrieved in three
 pieces:  the descriptor, the data, and the trigger times.  The waveform
 is retrieved in the default data format which may be as long as 32,000
 data points.  (Waveforms that have been processed by the 9400 in some
 way (i.e., Signal Averaged), will only by 25,000 data points long.  The
 format of the descriptor, the data, and the trigger times are given in
 the LeCroy 9400 manual.  The descriptor is read directly into a record
 that has been sized exactly as the descriptor is sent, the data is
 read directly into a sequential array, and the trigger times are
 read directly into a sequential array using direct software memory
 addressing.

 All 16-bit and 32-bit values received from the 9400 are high to low
 byte swapped because of the reverse order with which they are sent over
 the bus.}
```

```
var
   i,j:integer;                    {Counter}
   Lpos : CommLinePos;             {Line position for screen output}
   StrMinPos,StrMaxPos:str255;     {String Values of Scope Data Addresses}
   MinPos,MaxPos:integer;          {Values of Scope Data Addresses}
   ValCodeMin,ValCodeMax,ValCodeNS,ValCodeNP:integer;
                                   {String Conversion Error Codes}
   StrWfmPoints,StrNumSweeps,StrWfmReceive,StrSwpReceive: str12;
                                   {String Values of 9400 Replies}
   AcqChan,Reply:char;             {User Reply Character}
   ChannelCode:str12;              {Channel Code to be output to 9400}
   Terminate:boolean;              {Test for User Request to Terminate}

begin

   {Initialize and Clear Communication Window}

   Terminate:=False;
   ClearCommWindow (6);
   Lpos:=17;

   {Goto Communication window and start waveform acquisition}

   GotoXY(5,Lpos);
   Write('ACQUIRING WAVEFORM');
   Lpos:=succ(Lpos);
   GotoXY(5,Lpos);
   CheckInt(5000);

   {Request 9400 Acquistion Channel until valid channel requested}

   Repeat
      GotoXY(5,LPos);
      Write('Please input Channel or Memory for acquisition (1,2,C,D,E,F): ');
      repeat until keypressed;
      AcqChan:=Readkey;
      if AcqChan=chr(27) then Terminate:=True;
      if not Terminate then begin
         GotoXY(68,Lpos);
         Write(AcqChan);
         If not (AcqChan in ChannelSet) then begin
            LPos:=Succ(Lpos);
            GotoXY(5,Lpos);
            write('not a valid channel');
            Sound(400);Delay(500);NoSound;
            GotoXY(68,Lpos-1);
            Write(' ');
            GotoXY(5,Lpos);
            write('                    ');
            LPos:=Pred(Lpos);
         end;
      end;
   Until (AcqChan in ChannelSet) or (Terminate);

   {Continue if no user request for termination: <ESC>}

   if not(Terminate) then begin
      Case AcqChan of
         '1':ChannelCode:=Channel1;
         '2':ChannelCode:=Channel2;
         'c','C':ChannelCode:=MemoryC;
         'd','D':ChannelCode:=MemoryD;
         'e','E':ChannelCode:=FunctionE;
         'f','F':ChannelCode:=FunctionF;
      end;
      IeeeReset;
      Lpos:=succ(Lpos);
      With WfmDataRec do begin

         {Initialize Waveform Record}

         FillChar(Data9400,Sizeof(Data9400),0);
         FillChar(Trig9400,Sizeof(Trig9400),0);
         FillChar(Descrip9400,Sizeof(WfmDescriptor),0);
         GotoXY(5,Lpos);

      end;
   end;

   {If not user termination requested get waveform from 9400}

   If not Terminate then begin
      With WfmDataRec do begin
         LPos:=Succ(LPos);

         GotoXY(5,Lpos);

         {Get Waveform Descriptor from 9400}
```

```
    Write('Retrieving Descriptor');
    Writeln(IeeeOut,OutputCode+ReadCode+ChannelCode+DescriptorCode);
    Writeln(IeeeOut,ReceiveCode+'#154'+BufferCode,
            seg(Descrip9400),':',ofs(Descrip9400),EOICode);
    CheckInt(5050);

    {Swap high and low bytes of 16-bit and 32-bit values}

    With Descrip9400 do begin
       VertOffset:=Swap(VertOffset);
       TrigLevel:=Swap(Triglevel);
       TrigDelay:=SwapLongInt(TrigDelay);
       NumDataPtsDiv:=Swap(NumDataPtsDiv);
       FirstAddr:=Swap(FirstAddr);
       LastAddr:=Swap(LastAddr);
       Internal1:=Swap(Internal1);
       Internal2:=Swap(Internal2);
       Internal3:=Swap(Internal3);
       PowerVolts:=Swap(PowerVolts);
       PowerSec:=Swap(PowerSec);
       MaxSweeps:=SwapLongInt(MaxSweeps);
       MultFactor:=Swap(MultFactor);
       AddConst:=Swap(AddConst);
       MaxNumber:=Swap(MaxNumber);
       AcqSweeps:=SwapLongInt(AcqSweeps);
       NumOverflows:=SwapLongInt(NumOverflows);
       NumUnderflows:=SwapLongInt(NumUnderflows);
       NumReject:=SwapLongInt(NumReject);
       CompPts:=Swap(CompPts);
       RatioWfmPts:=Swap(RatioWfmPts)
    end;
end;
IeeeReset;

{Determine the Scope Address Limits of the Waveform}

Writeln(IeeeOut,OutputCode+InspectCode+ChannelCode+LimitsCode);
Writeln(IeeeOut,ReceiveCode);
Readln(IeeeIn,Response);
CheckInt(5010);
Delete(Response,1,pos('L',Response)+7);
StrMinPos:=copy(Response,1,Pos(',',Response)-1);
StrMaxPos:=copy(Response,Pos(',',Response)+1,Length(Response));
Val(StrMinPos,MinPos,ValCodeMin);
Val(StrMaxPos,MaxPos,ValCodeMax);
IeeeReInit;

{Determine the Number of Sweeps and the Number of
 Trigger Values in the Waveform}

Writeln(IeeeOut,OutputCode+InspectCode+ChannelCode+SweepsCode);
Writeln(IeeeOut,ReceiveCode);
Readln(IeeeIn,Response);
CheckInt(5020);
Delete(Response,1,pos('N',Response)+17);
StrNumSweeps:=Response;
With WfmDataRec do begin
   With Trig9400,Data9400 do begin
      Val(StrNumSweeps,NumSweeps,ValCodeNS);
      If ((ValCodeNS=0) and (ValCodeMin=0) and (ValCodeMax=0))then begin
         numpoints:=MaxPos-MinPos+1;

         {Calculate the number of trigger values.  This depends
          on the number of sweeps and the type of record.}

         If descrip9400.RecordType<>0 then begin
            if descrip9400.RecordType<0 then begin
               numtrig:=numsweeps;
               numpoints:=numpoints*numtrig
            end else begin
               numtrig:=1;
               for j:=0 to (descrip9400.RecordType-4) do
                  numtrig:=2*numtrig;
               numtrig:= 25*numtrig;
               numsweeps:=numtrig;
            end;
         end else numtrig:=1;
         Str(numpoints+4,StrWfmReceive);
         Str(numpoints,StrWfmPoints);
         Str(numtrig*2+4,StrSwpReceive);
         Str(NumSweeps,StrNumSweeps);
      end else begin
        LPos:=succ(Lpos);
        GotoXY(5,Lpos);
        writeln('error in string conversion for numsweeps/numpoints');
        Halt;
      end;
```

```
                {Get Data from LeCroy 9400 starting at first address and
                 ending with last address}

                GotoXY(5,Lpos);
                Write('Retrieving Data          ');
                IeeeReset;
                StrMinPos:=RemoveSpaces(StrMinPos);
                Writeln(IeeeOut,OutputCode+ReadCode+ChannelCode+DataCode+
                        ',1,'+StrWfmPoints+','+StrMinPos+',0');
                Writeln(IeeeOut,ReceiveCode+'#'+StrWfmReceive+BufferCode,
                        seg(DataHolder[0]),':',ofs(DataHolder[0]));
                CheckInt(5030);
                Writeln(IeeeOut,BufferedOutput);
                Readln(IeeeIn,Response);
                Response:=RemoveSpaces(Response);
                Val(Response,numpoints,ValCodeNP);
                if ValcodeNP<>0 then begin
                   LPos:=Succ(Lpos);
                   GotoXY(5,Lpos);
                   writeln('error in string conversion of buffer');
                end;
                numpoints:=numpoints-4;
                CheckInt(5035);

                {Get Trigger Times from LeCroy 9400}

                GotoXY(5,Lpos);
                Write('Retrieving Trigger Times');
                IeeeReset;
                Writeln(IeeeOut,Outputcode+ReadCode+ChannelCode+Triggercode);
                Writeln(IeeeOut,ReceiveCode+'#'+StrSwpReceive+BufferCode,
                        seg(TrigHolder[0]),':',ofs(TrigHolder[0]),EOICode);
                CheckInt(5040);
                Writeln(IeeeOut,BufferedOutput);
                Readln(IeeeIn,Response);
                Response:=RemoveSpaces(Response);
                Val(Response,numtrig,ValCodeNS);
                if ValcodeNS<>0 then begin
                   Lpos:=succ(Lpos);
                   GotoXY(5,Lpos);
                   writeln('error in string conversion of buffer');
                end;
                numtrig:=(numtrig div 2)-2;

                {Swap high and low bytes of 16-bit Trigger values}

                For i:=2 to NumTrig do
                   TrigHolder[i]:=Swap(TrigHolder[i]);
             end;
          end;
          CheckInt(5050);
          GotoXY(5,Lpos);
          Write('                                ');

          {Return 9400 to Local}

          Writeln(IeeeOut,'LOCAL');
       end;
       ClearCommWindow (6);
       If Terminate then begin
          GotoXY(5,17);
          Write('User Break');
       end;
    end;

Procedure ReturnWaveform(var wfmdatarec:waveform);


{This procedure returns a complete waveform to the LeCroy 9400 oscilliscope.
 The waveform to be sent must be in the current waveform record (computer
 memory.  The oscilliscope channels which are available to receiva a
 waveform transfer are Memories C and D.  There is no check if another
 channel is selected (i.e.,1,2,E,F) and unpredicatable results may occur.}

   var
      i:integer;                    {Counter}
      AcqChan:char;                 {User Reply Character}
      ChannelCode:str12;            {Channel Code to Output to 9400}
      StrWfmPoints,StrNumSweeps,StrWfmReturn,StrSwpReturn,StrMinPos:str12;
                                    {String Values for 9400 Codes}
      Terminate:boolean;            {Test for User Request to Terminate}

   begin

      {Initialize and Clear Communication Window}

      Terminate:=False;
      ClearCommWindow (6);
```

```
Lpos:=17;
GotoXY(5,Lpos);
Write('SENDING WAVEFORM');
CheckInt(6000);
LPos:=succ(Lpos);
GotoXY(5,Lpos);

{Request User for 9400 Reception Channel until valid Channel is
 selected.}

Repeat
   GotoXY(5,LPos);
   Write('Please input Channel or Memory for reception (C,D): ');
   repeat until keypressed;
   AcqChan:=Readkey;
   if AcqChan=chr(27) then Terminate:=True;
   If not(Terminate) then begin
      GotoXY(57,Lpos);
      Write(AcqChan);
      If not (AcqChan in ChannelSet) then begin
         LPos:=Succ(Lpos);
         GotoXY(5,Lpos);
         write('not a valid channel');
         Sound(400);Delay(500);NoSound;
         GotoXY(57,Lpos-1);
         Write(' ');
         GotoXY(5,Lpos);
         write('                    ');
         LPos:=Pred(Lpos);
      end;
   end;
Until (AcqChan in ChannelSet) or Terminate;

{Continue if no user request for termination: <ESC>}

If not(Terminate) then begin
   Lpos:=Succ(Lpos);
   Case AcqChan of
      '1':ChannelCode:=Channel1;
      '2':ChannelCode:=Channel2;
      'c','C':ChannelCode:=MemoryC;
      'd','D':ChannelCode:=MemoryD;
      'e','E':ChannelCode:=FunctionE;
      'f','F':ChannelCode:=FunctionF;
   end;
   With WfmDataRec do begin

      {Put number of Data Points, First Address, and Number
       of Sweeps in string form for use in buffered transfer
       code description.}

      With Data9400,Trig9400 do begin
         Str(NumPoints,StrWfmPoints);
         Str(Descrip9400.FirstAddr,StrMinPos);
         Str(NumPoints+4,StrWfmReturn);
         StrMinPos:=RemoveSpaces(StrMinPos);
         Str(NumTrig*2+4,StrSwpReturn);
         Str(NumSweeps,StrNumSweeps);

         {Swap high and low bytes of 16-bit Trigger Values.}

         For i:=2 to NumTrig+2 do
            Trigholder[i]:=Swap(Trigholder[i]);
      end;
      With Descrip9400 do begin

         {Swap high and low bytes of 16-bit and 32-bit Descrip. Values.}

         VertOffset:=Swap(VertOffset);
         TrigLevel:=Swap(Triglevel);
         TrigDelay:=SwapLongInt(TrigDelay);
         NumDataPtsDiv:=Swap(NumDataPtsDiv);
         FirstAddr:=Swap(FirstAddr);
         LastAddr:=Swap(LastAddr);
         Internal1:=Swap(Internal1);
         Internal2:=Swap(Internal2);
         Internal3:=Swap(Internal3);
         PowerVolts:=Swap(PowerVolts);
         PowerSec:=Swap(PowerSec);
         MaxSweeps:=SwapLongInt(MaxSweeps);
         MultFactor:=Swap(MultFactor);
         AddConst:=Swap(AddConst);
         MaxNumber:=Swap(MaxNumber);
         AcqSweeps:=SwapLongInt(AcqSweeps);
         NumOverflows:=SwapLongInt(NumOverflows);
         NumUnderflows:=SwapLongInt(NumUnderflows);
         NumReject:=SwapLongInt(NumReject);
         CompPts:=Swap(CompPts);
```

```
            RatioWfmPts:=Swap(RatioWfmPts)
          end;
          CheckInt(6020);

          {Send Waveform Descriptor to 9400}

          GotoXY(5,Lpos);
          Writeln('Sending Descriptor');
          IeeeReset;
          Writeln(IeeeOut,OutputCode+WriteCode+ChannelCode+DescriptorCode);
          Writeln(IeeeOut,OutCode+'#154'+BufferCode,seg(Descrip9400),
            ':',ofs(Descrip9400),EOICode);
          CheckInt(6010);
          With Data9400,Trig9400 do begin

            {Send Data to 9400}

            GotoXY(5,Lpos);
            Writeln('Sending Data       ');
            IeeeReset;
            Writeln(IeeeOut,OutputCode+WriteCode+ChannelCode+DataCode+
              ',1,'+StrWfmPoints+','+StrMinPos+',0');
            Writeln(IeeeOut,OutCode+'#'+StrWfmReturn+BufferCode,seg(DataHolder[0]),
              ':',ofs(DataHolder[0]));

            {Send Trigger Times to 9400}

            GotoXY(5,Lpos);
            Write('Sending Trigger Times');
            CheckInt(6030);
            IeeeReset;
            Writeln(IeeeOut,OutputCode+WriteCode+ChannelCode+TriggerCode);
            Writeln(IeeeOut,OutCode+'#'+StrSwpReturn+BufferCode,seg(TrigHolder[0]),
              ':',ofs(TrigHolder[0]),EOICode);
            CheckInt(6040);
                end;
          GotoXY(5,Lpos);
          Write('                    ');

          {Return 9400 to Local}

          Writeln(IeeeOut,'LOCAL');

          {Swap high and low Descriptor values back so that they are
           correct in computer memory.}

          With Descrip9400 do begin
            VertOffset:=Swap(VertOffset);
            TrigLevel:=Swap(TrigLevel);
            TrigDelay:=SwapLongInt(TrigDelay);
            NumDataPtsDiv:=Swap(NumDataPtsDiv);
            FirstAddr:=Swap(FirstAddr);
            LastAddr:=Swap(LastAddr);
            Internal1:=Swap(Internal1);
            Internal2:=Swap(Internal2);
            Internal3:=Swap(Internal3);
            PowerVolts:=Swap(PowerVolts);
            PowerSec:=Swap(PowerSec);
            MaxSweeps:=SwapLongInt(MaxSweeps);
            MultFactor:=Swap(MultFactor);
            AddConst:=Swap(AddConst);
            MaxNumber:=Swap(MaxNumber);
            AcqSweeps:=SwapLongInt(AcqSweeps);
            NumOverflows:=SwapLongInt(NumOverflows);
            NumUnderflows:=SwapLongInt(NumUnderflows);
            NumReject:=SwapLongInt(NumReject);
            CompPts:=Swap(CompPts);
            RatioWfmPts:=Swap(RatioWfmPts)
          end;

          {Swap high and low Trigger values back so that they are
           correct in computer memory.}

          With Trig9400 do
            For i:=2 to NumTrig+2 do
              Trigholder[i]:=Swap(Trigholder[i]);
        end;
      end;
      ClearCommWindow (6);
      If Terminate then begin
        GotoXY(5,17);
        Write('User Break');
      end;
    end;

Procedure SetGraphics;
```

```
{This procedure sets up the window displays of the acquisition program.
 The three windows are the following:  1) Command Window -- provides a
 menu of supported operations and the current waveform filename after
 it has been saved, 2) Communication Window -- gives the user information
 about the operation and prompts user replies, and 3) Graph Window --
 autoscales the waveform and plots the waveform as a series of dots.}

const
   MaxWorldX:Float=1000.0;          {Maximum World for X-dir. for Window 1}
   MaxWorldY:Float=1000.0;          {Maximum World for Y-dir. for Window 1}

begin

      ClearScreen;

      {Define Command Window}

      DefineHeader(5,'Commands');
      SetHeaderOn;
      SelectWindow(5);
      DrawBorder;
      GotoXY(61,3);
      Write('F1 = Acq. Param.');
      GotoXY(61,4);
      Write('F2 = Get Wave');
      GotoXY(61,5);
      Write('F3 = Send Wave');
      GotoXY(61,6);
      Write('F4 = Graph Wave');
      GotoXY(61,7);
      Write('F5 = Average Wave');
      GotoXY(61,8);
      Write('F6 = Ave. Power');
      GotoXY(61,9);
      Write('F7 = Analyze Wave');
      GotoXY(61,10);
      Write('F8 = Frequency');
      GotoXY(61,11);
      Write('F10 = Exit');

      With WfmDataRec.PowerParam do begin
         GotoXY(61,13);
         Write('Voltage X = ', VoltageMultiplier:6:1);
         GotoXY(61,14);
         Write('Current X = ', CurrentMultiplier:6:1);
      end;

      {Define Communication Window}

      DefineHeader(6,'Communication Window');
      SetHeaderOn;
      SelectWindow(6);
      DrawBorder;
      {Define Graph Window}

      DefineHeader(7,'Waveform Graph');
      DefineWorld(3,0,0,MaxWorldX,MaxWorldY);
      SetHeaderOn;
      SelectWorld(3);
      SelectWindow(7);
      DrawBorder;

      GotoXY (61,1);  Write ('File: ');
      GotoXY (67,1);  Write (Status.Filename);
  end;

Procedure TimeandVolts(WfmDataRec:Waveform; var Dfirstpoint:datapointer;
                       StartPos,EndPos:integer; var WorldVar:WorldParam);


{This procedure calculates the real time and voltage associated with the
 datapoint in the waveform data.  Voltages and Times are determined using
 LeCroy 9400 descriptor codes.  The waveform voltages and times are
 calculated for only N cycles in the waveform, where N is the number of
 cycles previously determined by tracking the reference signal waveform
 zero crossings.}

var
   DCurrentpoint:datapointer;           {Current Data Record}
   DLastPoint   :datapointer;
   TimeHolder,SampleIntvl,
      RealFVG:real;                  {Voltage and Time Values associated
                                      with LeCroy 9400 Codes}
   i,j:integer;                      {Counters}
   SumPower : real;
   reply : str1;

begin
```

```
GotoXY (5,23);
Write ('converting data taken in bytes into real time and volts...');

with WfmDataRec do begin
   with descrip9400,data9400,trig9400 do begin

      {Calcuate Fixed Vertical Gain from L9400 Code}

      Case FixedVertGain of
         22:  RealFVG:=5E-3;
         23:  RealFVG:=10E-3;
         24:  RealFVG:=20E-3;
         25:  RealFVG:=50E-3;
         26:  RealFVG:=0.1;
         27:  RealFVG:=0.2;
         28:  RealFVG:=0.5;
         29:  RealFVG:=1.0;
         30:  RealFVG:=2.0;
         31:  RealFVG:=5.0;
      end;

      {Determine type of Sampling for Scope Waveform Record}

      Case Samplingintvl of
         11:  sampleintvl :=0.2E-9;
         12:  sampleintvl :=0.4E-9;
         13:  sampleintvl :=0.8E-9;
         16:  sampleintvl :=10E-9;
         17:  sampleintvl :=20E-9;
         18:  sampleintvl :=40E-9;
         19:  sampleintvl :=80E-9;
         20:  sampleintvl :=0.2E-6;
         21:  sampleintvl :=0.4E-6;
         22:  sampleintvl :=0.8E-6;
         23:  sampleintvl :=2.0E-6;
         24:  sampleintvl :=4.0E-6;
         25:  sampleintvl :=8.0E-6;
         26:  sampleintvl :=20.0E-6;
         27:  sampleintvl :=40.0E-6;
         28:  sampleintvl :=80.0E-6;
         29:  sampleintvl :=0.2E-3;
         30:  sampleintvl :=0.4E-3;
         31:  sampleintvl :=0.8E-3;
         32:  sampleintvl :=2.0E-3;
         33:  sampleintvl :=4.0E-3;
         34:  sampleintvl :=8.0E-3;
         35:  sampleintvl :=20.0E-3;
         36:  sampleintvl :=40.0E-3;
      end;

      {Start at first Position Record, Initialize Counters}

      Dfirstpoint  := nil;
      TimeHolder   := 0;
      WorldVar.Max := -100.0;
      WorldVar.Min :=  100.0;

      {Cycle through all Data Values calculating Voltage and Time
       values for each.  Create Data Record to hold values}

      For i:=StartPos to EndPos do begin
         If Dfirstpoint=nil then begin
            new(Dfirstpoint);
            Dlastpoint:=Dfirstpoint;
         end else begin
            new(Dlastpoint^.nextpoint);
            Dlastpoint:=Dlastpoint^.nextpoint;
         end;
         With Dlastpoint^ do begin

            {Set time using incremented timeholder since
             sample time intervals are all equal in non-interleaved
             data sampling.}

            Xvalue:=TimeHolder;

            {Calculate Voltage using Formula given in LeCroy 9400
             Manual on page 7-49.}

            Yvalue:=RealFVG*((Dataholder[i]-128)/32-
                   (VertOffset-200)/25)*200/(VarVertGain+80);

            {Correct for attenuation set manually so that
              voltage is in kV and current is in amps}
             If ExtProbeAttn = 1 then Yvalue := Yvalue/10.0;
             If ExtProbeAttn = 2 then Yvalue := Yvalue/100.0;
             If ExtProbeAttn = 3 then Yvalue := Yvalue/1000.0;
            nextpoint:=nil;
```

```
                {Find Maxima and Minima of graph}
                With WorldVar do begin
                   If Yvalue > Max then Max := Yvalue;
                   If Yvalue < Min then Min := Yvalue;

                   If i = StartPos then Starttime := TimeHolder;
                   If i = EndPos then Endtime := TimeHolder;
                end;

                {Increment Time Holder}

                TimeHolder:=TimeHolder+SampleIntvl;
             end; {with}
          end; {for}
       end;  {with}
    end;  {with}
    ClearCommWindow(6);
end;


Procedure GetWaveStartEnd (var Start : integer; var Finish : integer;
                           WfmDataRec : waveform);

var
    tempaddr  : str8;
    code      : integer;
    reply     : char;
    terminate : boolean;
    StrStart  : str12;
    StrFinish : str12;

begin
    terminate := false;
    Lpos := 17;
    GotoXY (5,Lpos);  Write ('INPUT RANGE:');
    Lpos := succ (Lpos);
    With WfmDataRec.descrip9400, WfmDataRec.data9400 do begin
       GotoXY (5,Lpos);
       Write ('First Addr: ', FirstAddr, '    Last addr: ', LastAddr,
              '   TotalPoints: ', numpoints);
       Lpos := succ (Lpos);
       repeat
             {repeat until valid adddress selected}

          if not(terminate) then begin
             GotoXY (8,Lpos);  Write ('FirstAddr:');
             GotoXY (21,Lpos);  Write ('0');
             FillChar (StrStart, Sizeof(StrStart), ' ');
             Delete (StrStart, 1, Length (StrStart));
             repeat
                repeat until keypressed;
                reply := readkey;
                if reply = chr(27) then Terminate := true;
                if (reply <> chr(13)) and (reply <> chr(8)) and not (terminate) then begin
                   StrStart := StrStart+UpCase(reply);
                   GotoXY (21, Lpos);
                   Write (StrStart);
                end;
                if reply = chr(8) then begin
                   delete (StrStart, length (StrStart), 1);
                   GotoXY (21, Lpos);  Write (StrStart+' ');
                end;
             until (reply = chr(13)) or terminate;
             Val(StrStart, Start, Code);
             if (reply = chr(13)) and (Length (StrStart)=0) then begin
                Code := 0;
                Start := 0;
                GotoXY (21, Lpos);  Write (Start);
             end;

             if (Code <> 0) and not (Terminate) then begin
                Lpos := Succ (Lpos);
                GotoXY (5, Lpos);
                write ('not a valid address value');
                Sound (400);  Delay (500); NoSound;
                GotoXY (21, Lpos-1);
                Write ('                          ');
                GotoXY (5, Lpos);
                write ('                          ');
                Lpos := pred (Lpos);
             end;
          end;
       until (code = 0) or terminate;

       start := Start + 4 - FirstAddr;

       repeat
```

```
          {repeat until valid last address is selected}

          if not(terminate) then begin
             GotoXY (38, Lpos);
             Write ('Last Addr:');
             GotoXY (50,Lpos);  Write('1000');
             FillChar (StrFinish, Sizeof(StrFinish), ' ');
             Delete (StrFinish, 1, Length (StrFinish));
             repeat
                repeat until keypressed;
                reply := readkey;
                if reply = chr(27) then terminate := true;
                if (reply <> chr (13)) and (reply <> chr(8)) and not(terminate) then begin
                   StrFinish := StrFinish+UpCase(reply);
                   GotoXY(50, Lpos);
                   Write (StrFinish);
                end;
                if reply = chr(8) then begin
                   delete (StrFinish, length(StrFinish),1);
                   GotoXY (50, Lpos);
                   write (StrFinish+' ');
                end;
             until (reply = chr(13)) or terminate;
             Val(StrFinish, Finish, Code);
             if (reply = chr(13)) and (length(StrFinish)=0) then begin
                Code := 0;
                Finish := 1000;
                GotoXY (50,Lpos);
                Write(Finish);
             end;

             if (Code <> 0) and not(terminate) then begin
                Lpos := succ (Lpos);
                GotoXY (5, Lpos);
                Write ('not a valid address value');
                Sound (400);  Delay (500);  NoSound;
                GotoXY (50, Lpos-1);
                Write ('                              ');
                GotoXY (5,Lpos);
                Write ('                              ');
                Lpos := pred(Lpos);
             end;
          end;
       until (Code = 0) or terminate;

       Finish := Finish + 3 - Firstaddr;
    end;
end;


Procedure CalcPower (VFirstpoint : datapointer; IFirstPoint : datapointer;
                     var WfmDataRec : waveform);

{This procedure calculates the average power over the specified number of
 'rfcycles' using the current and voltage waveforms.  Current and voltage
 multiplied together at each rf cycle interval, summed together, and divided
 by the total number of points used in averaging.}

var
   InputString      : str8;
   Code             : integer;
   rfcycles         : integer;
   zerocrossing     : integer;
   NewSign, OldSign : string[3];
   count            : integer;    {number of points used in averaging power}
   VCurrentPoint    : datapointer;
   ICurrentPoint    : datapointer;

begin
   Lpos := 17;
   GotoXY(5,Lpos);  Write('Average power over how many rf cycles?: 2');
   InputString := '2';
   GetString (InputString, Numeric, 45, Lpos);
   Val (InputString, rfcycles, Code);
   GotoXY(45,Lpos);  Write(rfcycles);
   Lpos := succ (Lpos);
   zerocrossing := 0;
   wfmdatarec.PowerParam.avepower     := 0.0;
   count        := 0;

   VCurrentPoint := VFirstPoint;
   ICurrentPoint := IFirstPoint;

   If (VCurrentPoint^.Yvalue < 0.0) then OldSign := 'NEG'
   else OldSign := 'POS';

   {Check for first zero crossing to begin calculation}
   repeat
```

```
      VCurrentPoint := VCurrentPoint^.NextPoint;
      ICurrentPoint := ICurrentPoint^.NextPoint;
      If (VCurrentPoint^.Yvalue < 0.0) then NewSign := 'NEG'
      else NewSign := 'POS';
   until (OldSign <> NewSign);
   OldSign := NewSign;
   With WfmDataRec.PowerParam do begin
      repeat
         AvePower := AvePower + VCurrentPoint^.YValue * ICurrentPoint^.YValue;
         VCurrentPoint := VCurrentPoint^.NextPoint;
         ICurrentPoint := ICurrentPoint^.NextPoint;
         If (VCurrentPoint^.Yvalue < 0.0) then NewSign := 'NEG'
         else NewSign := 'POS';
         If (NewSign <> OldSign) then begin
            ZeroCrossing := ZeroCrossing + 1;
            OldSign := NewSign;
         end;
         count := count + 1;
      until (zerocrossing = 2*rfcycles);
      AvePower := AvePower/count * CurrentMultiplier * VoltageMultiplier;

      Lpos := succ(Lpos);
      GotoXY (5,Lpos);
      Write('Power averaged over ',rfcycles:2,' rf cycles =',AvePower:6:2,' Watts');
      Lpos := succ(Lpos);
      GotoXY (5,Lpos);
      Write(rfcycles:2,' rf cycles =  ',count:5,' points.');
      Lpos := succ (Lpos);
      GotoXY (5,Lpos);
      Write('Current Multiplier = ',CurrentMultiplier:5:1);
      Lpos := succ (Lpos);
      GotoXY (5,Lpos);
      Write('Voltage Multiplier = ',VoltageMultiplier:6:1);
   end;
end;


Procedure AverageWave (DfirstPoint:datapointer; VFirstPoint:datapointer);

var
   Dcurrentpoint   : datapointer;
   VCurrentPoint   : datapointer;
   AverageValue    : double;
   rfcycles        : integer;
   zerocrossing    : integer;
   count           : integer;
   code            : integer;
   NewSign,OldSign : string[3];
   Terminate       : boolean;

begin
   Terminate := false;
   GotoXY (5,18);  Write('Average over how many rf cycles?  2');
   InputString := '2';
   GetString (InputString, Numeric, 39, 18);
   Val (InputString, rfcycles, Code);
   GotoXY (39,18);  Write (rfcycles);

   zerocrossing := 0;
   AverageValue := 0.0;
   Count        := 0;
   DCurrentpoint := DFirstPoint;
   VCurrentPoint := VFirstPoint;

   If (VCurrentPoint^.Yvalue < 0.0) then OldSign := 'NEG' else OldSign := 'POS';
   {Check for first zero crossing}
   repeat
      VCurrentPoint := VCurrentPoint^.NextPoint;
      DCurrentPoint := DCurrentPoint^.NextPoint;
      If (VCurrentPoint^.Yvalue < 0.0) then NewSign := 'NEG' else NewSign := 'POS';
   until (NewSign <> OldSign);
   OldSign := NewSign;

   repeat
      AverageValue := AverageValue + DCurrentPoint^.Yvalue;
      VCurrentPoint := VCurrentPoint^.NextPoint;
      DCurrentPoint := DCurrentPoint^.NextPoint;
      Count := Count + 1;
      If (VCurrentPoint^.Yvalue < 0.0) then NewSign := 'NEG' else NewSign :='POS';
      If (NewSign <> OldSign) then begin
         ZeroCrossing := ZeroCrossing + 1;
         OldSign := NewSign;
      end;
      If (DCurrentPoint^.NextPoint = nil) then Terminate := true;
   until ((ZeroCrossing = (2*rfcycles)) or Terminate);
   AverageValue := AverageValue/Count;
   GotoXY (5,20);
   If Terminate then
```

```
      Write ('Acquired less than ', rfcycles,
            ' rf cycles of wave.  Average value of complete wave is ',
            AverageValue:10:5, ' V')
   else Write ('Average Wave Value over', rfcycles:3, ' rf cycles is ',
            AverageValue:10:5, ' V');
end;


PROCEDURE FindFrequency(DFirstPoint:DataPointer);

var
   ZeroCrossing     : integer;
   NewSign, OldSign : string[3];
   Count            : integer;
   CurrentPoint     : DataPointer;
   rfcycles         : integer;
   Terminate        : boolean;
   Time             : double;
   Frequency        : real;

begin
   GotoXY (5,18);  Write('Use how many rf cycles?  2');
   InputString := '2';
   GetString (InputString, Numeric, 30, 18);
   Val (InputString, rfcycles, Code);
   GotoXY (30,18);  Write (rfcycles);

   zerocrossing := 0;
   count        := 0;

   CurrentPoint := DFirstPoint;

   If (CurrentPoint^.Yvalue < 0.0) then OldSign := 'NEG'
   else OldSign := 'POS';

   {Check for first zero crossing}
   repeat
      CurrentPoint := CurrentPoint^.NextPoint;
      If (CurrentPoint^.Yvalue < 0.0) then NewSign := 'NEG'
      else NewSign := 'POS';
   until (OldSign <> NewSign);
   Time    := CurrentPoint^.Xvalue;

   OldSign := NewSign;
   Terminate := false;
   repeat
      CurrentPoint := CurrentPoint^.NextPoint;
      If (CurrentPoint^.Yvalue < 0.0) then NewSign := 'NEG'
      else NewSign := 'POS';
      If (NewSign <> OldSign) then begin
         ZeroCrossing := ZeroCrossing + 1;
         OldSign := NewSign;
      end;
      count := count + 1;
      If CurrentPoint^.NextPoint = nil then Terminate := true;
   until ((zerocrossing = 2*rfcycles) or (Terminate));
   If Terminate then begin
      GotoXY(5,19);  Write('Too many rf cycles specified.  Try again');
   end else begin
      Time := CurrentPoint^.Xvalue - Time;
      Frequency := 1.0/Time*rfcycles;
      GotoXY(5,20);  Write('Frequency (MHz) = ', Frequency/1.0E6:6:2);
   end;
end;


BEGIN

   {Draw Windows.}
   ClearScreen;
   IeeeReset;
   SetGraphics;

   {Wait at Communication Window until a Function Key is Pressed}
   Repeat
      GotoXY(5,23);
      Write('Press a Function Key for selection');
      repeat until keypressed;
      FuncKey := 1;
      GetFunckey (FuncKey);

      {Execute Routine associated with Function Key}
      Case Funckey of
                              {F1 = Acq. Param.}
         59:GetPowerSysParam(WfmDataRec);

         60:begin             {F2 = Get Wave from 9400}
               ClearCommWindow (6);
```

```
      GotoXY (5,17);
      Write ('Type of Waveform? V=Voltage  I=Current  O=Other');
      IsSpectra := false;
      Terminate := false;
      Repeat
         Reply := UpCase(ReadKey);
         Case Reply of
            'V':begin
                GetWaveform(WfmDataRec);
                CheckInt(1000);
                GetWaveStartEnd(Start, Finish, WfmDataRec);
                EraseData(VFirstPoint,VLastPoint);
                TimeandVolts(WfmDataRec,VFirstPoint,
                             Start, Finish, VWorldVar);
                IsSpectra := true;
                Status.IsVWaveSpectra := true;
                end;

            'I':begin
                GetWaveform(WfmDataRec);
                CheckInt(1000);
                EraseData(IFirstPoint,ILastPoint);
                GetWaveStartEnd(Start, Finish, WfmDataRec);
                TimeandVolts(WfmDataRec,IFirstPoint,
                             Start, Finish, IWorldVar);
                IsSpectra := true;
                Status.IsIWaveSpectra := true;
                end;

            'O':begin
                GetWaveform(WfmDataRec);
                CheckInt(1000);
                EraseData(OFirstPoint,OLastPoint);
                GetWaveStartEnd(Start, Finish, WfmDataRec);
                TimeandVolts(WfmDataRec,OFirstPoint,
                             Start, Finish, OWorldVar);
                IsSpectra := true;
                Status.IsOWaveSpectra := true;
                end;

            else begin
               if Reply = chr(27) then Terminate := true;
               Sound (1000);  Delay (500);  NoSound;
            end;
         end; {Case}
      until(IsSpectra or Terminate);
   end;

61:begin             {F3 = Send Wave to 9400}
      ClearCommWindow (6);
      GotoXY (5,17);
      Write ('Sending latest acquired waveform back to oscilloscope');
      ReturnWaveform(WfmDataRec);
      CheckInt(2000);
   end;

62:begin             {F4 = Graph Wave}
     ClearCommWindow (6);
     GotoXY (5,17);  Write ('Which one?  V=Voltage, I=Current, B=Both, O=Other');
     Terminate := false;
     IsSpectra := false;
     ClearGraphWindow;
     Repeat
        Reply := UpCase(ReadKey);
        Case Reply of
           'V':if (Status.IsVWaveSpectra) then begin
                  GotoXY(5,18);  Write ('Plotting voltage waveform        ');
                  DrawSpectra (VFirstPoint,VWorldVar.Max,VWorldVar.Min,
                               VWorldVar.StartTime,VWorldVar.EndTime,7);
                  IsSpectra := true;
                  GotoXY(5,18);  Write ('                                 ');
               end else begin
                  GotoXY(5,18);  Write ('NO VOLTAGE WAVEFORM TO PLOT      ');
                  Sound (1100);  Delay (100); NoSound;
                  Terminate := true;
               end;
           'I':if (Status.IsIWaveSpectra) then begin
                  GotoXY(5,18);  Write ('Plotting current waveform        ');
                  DrawSpectra (IFirstPoint,IWorldVar.Max,IWorldVar.Min,
                               IWorldVar.StartTime,IWorldVar.EndTime,7);
                  IsSpectra := true;
                  GotoXY(5,18);  Write ('                                 ');
               end else begin
                  GotoXY(5,18);  Write ('NO CURRENT WAVEFORM TO PLOT      ');
                  Sound (1100);  Delay (100); NoSound;
                  Terminate := true;
               end;
           'B':if (Status.IsVWaveSpectra and Status.IsIWaveSpectra) then begin
```

```
                  GotoXY(5,18);  Write ('Plotting V waveform              ');
                  DrawSpectra (VFirstPoint,VWorldVar.Max,VWorldVar.Min,
                               VWorldVar.StartTime,VWorldVar.EndTime,7);
                  GotoXY(5,18);  Write ('Plotting I waveform              ');
                  DrawSpectra (IFirstPoint,IWorldVar.Max,IWorldVar.Min,
                               IWorldVar.StartTime,IWorldVar.EndTime,7);
                  IsSpectra := true;
                  GotoXY(5,18);  Write ('                                 ');
               end else begin
                  GotoXY(5,18);  Write ('NO V OR I WAVEFORMS TO PLOT      ');
                  Sound (1100);  Delay (100); NoSound;
                  Terminate := true;
               end;
            'O':if (Status.IsOWaveSpectra) then begin
                  GotoXY(5,18);  Write ('Plotting waveform                ');
                  DrawSpectra (OFirstPoint,OWorldVar.Max,OWorldVar.Min,
                               OWorldVar.StartTime,OWorldVar.EndTime,7);
                  IsSpectra := true;
                  GotoXY(5,18);  Write ('                                 ');
               end else begin
                  GotoXY(5,18);  Write ('NO WAVEFORM TO PLOT        ');
                  Sound (1100);  Delay (100); NoSound;
                  Terminate := true;
               end;

            else if Reply = chr(27) then Terminate := true;
         end; {case}
      until (Terminate or IsSpectra);
      GotoXY (5,17);
      Write ('                                                    ');
   end;

63:begin             {F5 = Average Wave}
      ClearCommWindow (6);
      GotoXY(5,17);
      Write ('Average which wave? V=Voltage  I=Current  O=Other');
      Reply := UpCase(Readkey);
      Case Reply of
         'V':begin
               If (Status.IsVWaveSpectra) then
                  AverageWave(VFirstPoint,VFirstPoint)
               else begin
                  GotoXY (5,18);
                  Write ('No voltage waveform');
                  Sound (1000);  Delay (500);  NoSound;
               end;
            end;
         'I':begin
               If(Status.IsIWaveSpectra) then
                  If (Status.IsVWaveSpectra) then
                     AverageWave(IFirstPoint,VFirstPoint)
                  else begin
                     GotoXY(5,18);
                     Write ('Need Voltage waveform as reference');
                     Sound (1000);  Delay (500);  NoSound;
                  end
               else begin
                  GotoXY (5,18);
                  Write ('No current waveform');
                  Sound (1000);  Delay (500);  NoSound;
               end;
            end;
         'O':begin
               If(Status.IsOWaveSpectra) then
                  If (Status.IsVWaveSpectra) then
                     AverageWave(OFirstPoint,VFirstPoint)
                  else begin
                     GotoXY(5,18);
                     Write ('Need Voltage waveform as reference');
                     Sound (1000);  Delay (500);  NoSound;
                  end
               else begin
                  GotoXY (5,18);
                  Write ('No waveform exist');
                  Sound (1000);  Delay (500);  NoSound;
               end;
            end;
         else begin
              GotoXY (5,18);
              Write ('Invalid key pressed');
              Sound (1000);  Delay (500);  NoSound;
         end;
      end;  {case}
      CheckInt(3000);
   end;

64:begin             {F6 = Calculate Power}
      ClearCommWindow (6);
```

```
            If (Status.IsVWaveSpectra and Status.IsIWaveSpectra)then
               CalcPower(VFirstPoint,IFirstPoint,WfmDataRec)
            else begin
               GotoXY(5,17);  Write('Need current, voltage or both waveforms');
               Sound (1100);  Delay (1000);  NoSound;
            end;
         end;

      65:begin             {F7 = Analyze}
            ClearCommWindow (6);
            GotoXY(5,17);  Write ('Which waveform? V=Voltage  I=Current  O=Other: ');
            Reply := UpCase(ReadKey);
            GotoXY(52,17);  write(Reply);
            Case Reply of
               'V':begin
                     If(Status.IsVWaveSpectra) then begin
                        With VWorldVar do ScanData(VFirstPoint,
                           StartTime,EndTime,Max,Min,7);
                        ClearCommWindow(6);
                     end else begin
                        GotoXY(5,18);  Write ('No voltage waveform to analyze');
                     end;
                  end;

               'I':begin
                     If(Status.IsIWaveSpectra) then begin
                        With IWorldVar do ScanData(IFirstPoint,
                           StartTime,EndTime,Max,Min,7);
                        ClearCommWindow(6);
                     end else begin
                        GotoXY(5,18);  Write ('No current waveform to analyze');
                     end;
                  end;

               'O':begin
                     If(Status.IsOWaveSpectra) then begin
                        With OWorldVar do ScanData (OFirstPoint,
                           StartTime,EndTime,Max,Min,7);
                        ClearCommWindow(6);
                     end else begin
                        GotoXY(5,18);
                        Write ('No waveform to analyze  ');
                     end;
                  end;

               else begin
                  GotoXY(5,18);  Write ('Invalid key pressed');
                  Sound (1000);  Delay (500);  NoSound;
               end;
            end;  {Case}
         end;

         66:begin   {Frequency}
               ClearCommWindow (6);
               GotoXY(5,17);  Write ('Which waveform? V=Voltage  I=Current  O=Other');
               Reply := UpCase(ReadKey);
               Case Reply of
                  'V':begin
                        If(Status.IsVWaveSpectra) then FindFrequency(VFirstPoint)
                        else begin
                           Sound (1000);  Delay(500);  NoSound;
                           GotoXY(5,19);  Write('No Voltage Waveform');
                        end;  {if}
                     end;
                  'I':begin
                        If(Status.IsIWaveSpectra) then FindFrequency(IFirstPoint)
                        else begin
                           Sound(1000);  Delay(500);  NoSound;
                           GotoXY(5,19);  Write('No Current Waveform');
                        end;  {if}
                     end;
                  'O':begin
                        If(Status.IsOWaveSpectra) then FindFrequency(OFirstPoint)
                        else begin
                           Sound(1000);  Delay(500);  NoSound;
                           GotoXY(5,19);  Write('No Waveform');
                        end;  {if}
                     end;
                  else begin
                     Sound(1000);  Delay(500);  NoSound;
                     GotoXY(5,19);  Write('Invalid key pressed.  Try Again');
                  end;
               end;  {Case}
         end;  {F8}

   end; {Case}
until (FuncKey=68);
end; {procedure}
```

```
END. {unit}
```

```
Unit PIE;

{This unit contains procedures used to link the SpectraLink and the picoammeter
to take optical spectra
J. Liu -- March 1991}

INTERFACE
Uses Dos, Crt, GDriver, GKernel, GWindow, GShell, AuxInOut,
     Atools, Ftools;

PROCEDURE OpticalSection (var Status      : StatusRecord;
                          var OpticalData : OpticalRec);

IMPLEMENTATION
PROCEDURE OpticalSection (var Status      : StatusRecord;
                          var OpticalData : OpticalRec);

Var
   IsExitPIE       : boolean;
   ExitSave        : pointer;
   FuncKey         : integer;
   Code            : integer;
   Reply           : char;
   NumIn           : integer;
   XtraFirstPoint  : DataPointer;
   XtraLastPoint   : DataPointer;
   WorldHigh,WorldLow : real;
   Terminate       : boolean;
   ReactorMotorSpecs  : MotorRecord;


Procedure WritePIEScreen (PIEParam:PIERecord; CurrWavelength:real);

Const
   MaxWorldX:Float=1000.0;
   MaxWorldY:Float=1000.0;

Var
   meter    : real;

Begin
   ClearScreen;
   DefineHeader(7,'Optical Spectrum');
   SetHeaderOn;
   SelectWindow(7);
   DrawBorder;

   {Define Command Window Options}
   DefineHeader(5,'Commands');
   SetHeaderOn;
   SelectWindow(5);
   DrawBorder;
   GotoXY(61,3);  Write ('F1 = PIE Param.');
   GotoXY(61,4);  Write ('F2 = Move Reactor');
   GotoXY(61,5);  Write ('F3 = Spatial Scan');
   GotoXY(61,6);  Write ('F4 = Wave Scan');
   GotoXY(61,7);  Write ('F5 = Plot Scans');
   GotoXY(61,8);  Write ('F6 = Retrieve Data');
   GotoXY(61,9);  Write ('F7 = Scan Data');
   GotoXY(61,10); Write ('F10= Exit');

   {Communication Window}
   DefineHeader (6,'Communication Window');
   SetHeaderOn;
   SelectWindow (6);
   DrawBorder;

   {Write monochromator parameters}
   GotoXY(61,13);  Write ('Grating(#/mm): ');
   GotoXY(75,13);  Write (PIEParam.GratingGrooveDensity);
   GotoXY(61,14);  Write ('Meter:');
   meter := CurrWavelength*PIEParam.GratingGrooveDensity/1200.0;
   GotoXY(68,14);  Write (meter:8:1);

   GotoXY(61,1);  Write ('File: ',Status.FileName);

   GotoXY(5,24);  Write('Press Function Key....');
end;

Procedure GetPIEParam (var PIEParam:PIERecord; Var CurrWavelength:real);

Var
   Meter       : real;
   InputString : str8;
   Code        : integer;
   reply       : char;

Begin
```

```
    ClearCommWindow(6);
    Terminate := false;
    GotoXY(5,17);  Write ('Which Grating?  A=1800grooves/mm  B=2400grooves/mm');
    repeat
       Reply := UpCase(readkey);
       ClearCommLine(18);
       Case Reply of
          'A':With PIEParam do begin
                 GratingGrooveDensity := 1800;
                 StepsPerAngstrom := 150;
                 Terminate := true;
              end;

          'B':With PIEParam do begin
                 GratingGrooveDensity := 2400;
                 StepsPerAngstrom := 200;
                 Terminate := true;
              end;

          else begin
             GotoXY(5,18);  Write('INVALID CHOICE.  TRY AGAIN');
             Sound(1000);  Delay(500);  NoSound;
          end;
       end;  {Case}
       GotoXY (75,13);  Write(PIEParam.GratingGrooveDensity);
    until (Terminate);

    GotoXY(5,17);  Write('Enter meter reading on monochromater:                ');
    Meter := CurrWavelength*PIEParam.GratingGrooveDensity/1200.0;
    Str(Meter:8:1,InputString);
    GetString (InputString, numeric, 43, 17);
    Val (InputString,Meter,Code);
    CurrWavelength := meter / PIEParam.GratingGrooveDensity * 1200.0;
    ClearCommLine(17);
    GotoXY(68,14);  Write (Meter:8:1);
    GotoXY(5,24);  Write('Press Function Key....');
end;

PROCEDURE GetMotorSpecs (var MotorSpecs:MotorRecord);
    {Calculate the number of steps to move motor by first finding
    out which motor and drive is being used}

Var
    InputString : Str8;
    code        : integer;
    Done        : boolean;
    Terminate   : boolean;
    Reply       : char;

Begin
    ClearCommWindow(6);
    Terminate := false;
    With MotorSpecs do begin
       Length := 0.0;
       InputString := '0';
       GotoXY (5,17);  Write ('How many cm?  ');
       GetString (InputString, Numeric, 22,17);
       Val (InputString, length, code);
       If length = 0.0 then begin
          GotoXY(5,17);  write ('                    ');
          Terminate := true;
       end;
       If not(Terminate) then begin
          GotoXY (30,17);  Write ('Which motor? 1=Horizontal  2=Vertical : ');
          repeat
             Motor := 0;
             Reply := readkey;
             If Reply = chr (27) then Terminate := true;
             Motor := ord(Reply);
             GotoXY (70,17);  Write (Reply);
             If ((Motor = 49) or (Motor = 50)) then Terminate := true
             else begin
                Sound (1000);  Delay (500);  NoSound;
                GotoXY(10,18);  Write ('INVALID KEY. Try again');
             end;
          until (Terminate);
          ClearCommLine(18);
          {The keyboard character of '1' is 49 and '2' is 50}
          If Motor = 49 then Motor := 1 else Motor := 2;
          If Motor = 1 then TotalSteps := Round (XYMotorStepCm * Length)
          else TotalSteps := Round (ZMotorStepCm * Length);

          GotoXY (5,18);  Write ('Which drive?');
          Repeat
             GotoXY (10,19); Write ('G=Green  O=Orange  R=Red: ');
             Terminate := true;
             Drive := UpCase(readkey);
             GotoXY (19,18); Write (Drive);
```

```
            If not (Drive in DriveSet) then begin
               Sound (1000);  Delay (500);  NoSound;
               Terminate := false;
            end;
         until (Terminate);

         GotoXY (45,18);  Write ('Which direction?');
         Repeat
            Terminate := true;
            GotoXY (50,19); Write ('E=Extend  R=Retract: ');
            Direction := UpCase(readkey);
            GotoXY (62,18); Write (Direction);
            If not (Direction in DirnSet) then begin
               Sound (1000);  Delay (500);  NoSound;
               Terminate := false;
            end;
         until (Terminate);
         ClearCommLine(19);
      end; {If}
   end; {With}
end; {Procedure}

PROCEDURE GetSpaScanSpecs (var SpatialScan:SpatialScanRecord;
                           var Terminate:boolean);
var
   Reply       : char;
   InputString : str8;
   code        : integer;

begin
   Terminate := false;
   repeat
      GetMotorSpecs (SpatialScan.MotorSpecs);

      {Distance between data points in cm}
      GotoXY (5,19);  Write ('Distance between data points (cm): ');
      Str (SpatialScan.cmBetweenData, InputString);
      GetString(InputString, Numeric, 40, 19);
      Val (InputString,SpatialScan.cmBetweenData,code);

      {Number of readings at each location between the electrodes}
      GotoXY(5,20);  Write ('Average over how many readings?');
      Str (SpatialScan.ReadingsPerPt, InputString);
      GetString(InputString, Numeric, 37, 20);
      Val (InputString,SpatialScan.ReadingsPerPt,Code);

      GotoXY(5,24);  Write('Is everything correct? (ESC to QUIT) ');
      Reply := readkey;
      If Reply = chr(27) then Terminate := true
      else Reply := UpCase(Reply);
   until(Terminate or (Reply = 'Y'));
   ClearCommLine(24);
end;

Procedure InitCom1Port;
Begin
   AssignAux (Com1In,0,$E3);
   AssignAux (Com1Out,0,$E3);
   ReWrite (Com1In);
   ReSet (Com1Out);
end;

Procedure CloseCom1Port;
Begin
   Close(Com1In);
   Close(Com1Out);
end;

Procedure SendToSpectraLink(NumIn:integer; var Terminate:boolean);

Var
   NumOut     : integer;
   TextIn     : char;
   TextOut    : char;
   Funckey    : integer;

Begin
   TextIn := chr(NumIn);
   Write (Com1In, TextIn);
   Terminate := false;
   repeat
      Read (Com1Out, TextOut);
      NumOut := ord (TextOut);
      If keypressed then begin
         GetFunckey (FuncKey);
         If Funckey = 68 then Terminate := true;
      end;
   until ((NumOut = NumIn) or Terminate);
```

```
end;

Procedure WakeUpSpectraLink(var Terminate:boolean);

Begin
   {Send wake call to Spectra Link}
   SendToSpectraLink(58,Terminate);

   {Check Module Address}
   SendToSpectraLink(3,Terminate);
end;

Procedure IsPortBusy (var Terminate:boolean);

Var
   Funckey  : integer;
   Problem  : boolean;

Begin
   WakeUpSpectraLink (Terminate);

   repeat
      {Busy Test Code}
      If not(Terminate) then SendToSpectraLink(0,Terminate);

      {Number of bytes to follow}
      If not(Terminate) then SendToSpectraLink(0,Terminate);

      {find Result of Busy test = 63}
      If not(Terminate) then begin
         TextIn := chr($3F);
         write(Com1In, TextIn);
         Terminate := false;
         repeat
            Read(Com1Out, TextOut);
            NumOut := ord(TextOut);
            If keypressed then begin
               GetFunckey(Funckey);
               If Funckey = 68 then Terminate := true;
            end;
         until (Terminate or (NumOut = 98) or (NumOut = 66));
      end;
   until (Terminate or (NumOut = 98));

   {Release Spectra Link}
   SendToSpectraLink(58,Problem);
end;

Procedure ConvertToByte (Number:longint; var LSB:integer;
                         var ISB:integer; var MSB:integer);

Begin
   MSB := Trunc(Number/65536);
   ISB := Trunc((Number-MSB*65536)/256);
   LSB := Number-MSB*65536-ISB*256;
end;

Procedure ConvertByteToNumber (var Number:longint; LSB,ISB,MSB:integer);
{used to convert hexadecimals to decimals}
Begin
   Number := LSB + ISB*256 + MSB*65536;
end;

Procedure LoadAbsolutePosition (wavelength:real; StepsPerAngstrom:integer);
{The wavelength reading must be converted to steps by first multiplying
by the number of steps per unit then converted into three bytes
to send to the MDR module of the Spectra Link.
         Grating of 1800 grooves/mm --->  150 steps/A
             "    " 2400 grooves/mm --->  200 steps/A}

Var
   Number      : longint;
   Terminate   : boolean;
   LSB,ISB,MSB : integer;

Begin
   Number := Round(Wavelength*StepsPerAngstrom);
   ConvertToByte(Number,LSB,ISB,MSB);
   WakeUpSpectraLink (Terminate);

   {Load Absolute Position Code}
   If not(Terminate) then SendToSpectraLink(65,Terminate);

   {Number of Bytes to follow = 3}
   If not(terminate) then SendToSpectraLink(3,Terminate);

   {Send Position in 3 Bytes}
   If not(Terminate) then SendToSpectraLink(LSB,Terminate);
```

```
    Delay (10);

    If not(Terminate) then SendToSpectraLink(ISB,Terminate);
    Delay(10);

    If not(Terminate) then SendToSpectraLink (MSB,Terminate);
    Delay(10);

    {Reset Spectra Link before termination}
    SendToSpectraLink(58,Terminate);
    If Terminate then begin
       GotoXY(5,23);  write('Spectra Link not released after LOAD ABS. POS.  ');
    end;
end;

Procedure LoadSpeed;
{The speed parameter may be from 1 to 65535 where 1 is the fastest speed
and 65535 is the slowest.  A speed of 10 will work with all ISA monochromators.
To load this value, it is necessary to convert it to two bytes first.  This is
accomplished by determining the MSB by taking the integer result of 10/256
or 0.  The LSB is (10-MSB*256) or 10.}

Var
   Terminate  :  boolean;

Begin
   WakeUpSpectraLink (Terminate);

   {Load Speed = 10}
   If not(Terminate) then SendToSpectraLink (83,Terminate);

   {Number of bytes to follow}
   If not(Terminate) then SendToSpectraLink (2,Terminate);

   If not(Terminate) then begin
      {Send LSB of 2 in hexadecimal}
      SendToSpectraLink (2,Terminate);

      {Send MSB of 2 in hexadecimal}
      SendToSpectraLink (0,Terminate);
   end;

   SendToSpectraLink(58,Terminate);
   If Terminate then begin
      GotoXY(5,23);  write('Spectra Link not released after LOAD Speed    ');
   end;
end;

Procedure LoadTargetPosition (Wavelength:real; StepsPerAngstrom:integer);
{The ending wavelength of the monochromator, in nanometers, must be
multiplied by steps per unit and then converted into bytes}

Var
   MSB,ISB,LSB : integer;
   Number      : longint;
   Terminate   : boolean;

Begin
   Number := Round(Wavelength*StepsPerAngstrom);
   ConvertToByte (Number,LSB,ISB,MSB);

   WakeUpSpectraLink (Terminate);

   {Load target position}
   If not(Terminate) then SendToSpectraLink (84,Terminate);

   If not(Terminate) then begin
      {Number of bytes to follow}
      SendToSpectraLink (3,Terminate);

      {Send LSB}
      SendToSpectraLink (LSB,Terminate);
      Delay(10);                {Delays are added to make sure the Spectra}
                                {is ready for next number}
      {Send ISB}
      SendToSpectraLink (ISB,Terminate);
      Delay(10);

      {Send MSB}
      SendToSpectraLink (MSB,Terminate);
      Delay(10);
   end;

   SendToSpectraLink(58,Terminate);
   If Terminate then begin
      GotoXY(5,23);  write('Spectra Link not released after TARGET POSITION');
   end;
end;
```

```
Procedure MoveGrating;
{The MDR of the Spectra Link needs the Absolute and the Target Position
 before this procedure is activated}
Var
   Terminate    : boolean;

Begin
   WakeUpSpectraLink (Terminate);
   If not(Terminate) then SendToSpectraLink (71,Terminate);
   If not(Terminate) then SendToSpectraLink (0,Terminate);
   If not(Terminate) then SendToSpectraLink(58,Terminate);
end;

Procedure StopGrating;

Var
   Terminate    : boolean;

Begin
   WakeUpSpectraLink (Terminate);
   SendToSpectraLink (66,Terminate);
   SendToSpectraLink (0,Terminate);
   SendToSpectraLink (63,Terminate);
end;

Procedure UpDateWavelength (wavelength:real; GratingGrooveDensity:real);

var
   meter  : real;

begin
   meter := wavelength * GratingGrooveDensity / 1200.0;
   GotoXY(68,14);  Write (Meter:8:1);
end;


PROCEDURE AssignPortHiLo (var MotorSpecs:MotorRecord);
     {Port $21E, digital output port, see pp. 5-14 in the DT2811 manual,
      on the DT2811 need high and low signals to set the
      direction.  Also, the low to high transition of the DT2811
      digital output is converted via optocouplers to a high to
      low transition for the drive clock.  Every time the drive clock
      makes this transition, a step is taken.  The clock low needs to
      be at least 10 microseconds.  The signal to extend or retract the
      piston is different between the motor to move the reactor up and
      down and the motors to move the reactor in the horizontal directions.
      The motor for the vertical direction requires a high output at the
      power supply (a low signal from the DT2811 board) to retract the piston.
      The motors for the horizontal directions require a low output at the
      power supply to retract the piston.}

Begin
   With MotorSpecs do begin

      Case Drive of

         'G':begin
                {Check to make sure not Z-motor which has reverse
                direction from X,Y-motors for a High or Low DIRN signal}
                If (Motor = 1) then begin
                   If (Direction = 'E') then begin
                      HighMotor := Lo(16);      {00010000}
                      LowMotor  := Lo(0);
                   end else begin
                      HighMotor := Lo(48);      {00110000}
                      LowMotor  := Lo(32);      {00100000}
                   end;
                end else begin
                   If (Direction = 'R') then begin
                      HighMotor := Lo(16);      {00010000}
                      LowMotor  := Lo(0);
                   end else begin
                      HighMotor := Lo(48);      {00110000}
                      LowMotor  := Lo(32);      {00100000}
                   end;
                end;
         end; {begin of 'G'}

         'O':begin
                {Check to make sure not Z-motor which has reverse
                direction from X,Y-motors for a High or Low DIRN signal}
                If (Motor = 1) then begin
                   If (Direction = 'E') then begin
                      HighMotor := Lo(4);       {00000100}
                      LowMotor  := Lo(0);
                   end else begin
                      HighMotor := Lo(12);      {00001100}
                      LowMotor  := Lo (8);      {00001000}
```

```
                end;
             end else begin
                If (Direction = 'R') then begin
                   HighMotor := Lo(4);      {00000100}
                   LowMotor  := Lo(0);
                end else begin
                   HighMotor := Lo(12);     {00001100}
                   LowMotor  := Lo (8);     {00001000}
                end;
             end;
       end;{begin of 'O'}

       'R':begin
             {Check to make sure not Z-motor which has reverse
              direction from X,Y-motors for a High or Low Dirn signal}
             If (Motor = 1) then begin
                If (Direction = 'E') then begin
                   HighMotor := Lo (1);     {00000001}
                   LowMotor  := Lo (0);
                end else begin
                   HighMotor := Lo (3);     {00000011}
                   LowMotor  := Lo (2);     {00000010}
                end;
             end else begin
                If (Direction = 'R') then begin
                   HighMotor := Lo (1);     {00000001}
                   LowMotor  := Lo (0);
                end else begin
                   HighMotor := Lo (3);     {00000011}
                   LowMotor  := Lo (2);     {00000010}
                end;
             end;
       end; {begin of 'R'}

    end;  {Case}
  end; {With}
end; {procedure}


Procedure ResetMotorSpecs (var MotorSpecs:MotorRecord);
{reset the motorspecs to return the reactor to its original location:
reverse motor direction.}

Begin
  With MotorSpecs do
    If Direction = 'R' then Direction := 'E'
    else Direction := 'R';
  AssignPortHiLo (MotorSpecs);
end;


Procedure MoveReactor (var MotorSpecs:MotorRecord; var Terminate:boolean);

Var
  NumberStep : longint;
  Reply      : char;

Begin
  Terminate := false;
  repeat
    GetMotorSpecs (MotorSpecs);
    GotoXY(5,24);  Write('Is everything correct (Y/N)?  (ESC to QUIT)');
    Reply := readkey;
    If reply = chr(27) then Terminate := true;
    Reply := UpCase (Reply);
    ClearCommLine(24);
  until ((Reply = 'Y') or Terminate);

  If (not(Terminate) and not(MotorSpecs.length = 0.0)) then begin
    AssignPortHiLo (MotorSpecs);
    GotoXY(5,24);  Write('Moving Reactor....                    ');

    With MotorSpecs do begin
      If TotalSteps <> 0 then begin
        NumberStep := 0;
        repeat
          GotoXY (60,24);  Write (NumberStep);
          Port[$21E] := LowMotor;
          Delay (1);
          Port[$21E] := HighMotor;
          NumberStep := NumberStep + 1;
          If keypressed then begin
            Reply := readkey;
            If Reply = chr(27) then Terminate := true;
          end;
        until ((NumberStep = TotalSteps) or Terminate);
        TotalSteps := NumberStep;
      end;
```

```
        {Calculate distance moved}
        If Motor = 1 then Length := TotalSteps/XYMotorStepCm
        else Length := TotalSteps/ZMotorStepCm;
        GotoXY(5,19);  Write ('Distance moved (cm) = ',Length:10:2);
      end;
   end else MotorSpecs.Length := 0.0;

   {Reset Port that was used to move reactor}
   Port[$21E] := 0;
   ClearCommLine(24);
end;


Procedure GetSpatialScan (var SpatialScan:SpatialScanRecord;
                          var SpaFirstPoint:DataPointer;
                          var SpaLastPoint :DataPointer;
                              PIEParam:PIERecord;
                          var Currwavelength:real;
                          var Terminate:boolean);

       {This procedure moves the reactor and take data from monochrometer.
        When the computer is used to move the reactor, the drive dip
        switches must all be off except for the current switch (3)
        which is always on and the mode switch which can be on or off.
        The computer writes to the Digital Output ports in binary
        number.  This sets pins 39 - 46.  See page 4-4 in DT2811 board manual
        for pin assignments.  Only pins 39 - 44 are used to move reactor.

            Motor       Dig. Output Line       Function
             RED           0   (pin 38)          clock
              "            1   (pin 39)          direction
             ORANGE        2   (pin 40)          clock
              "            3   (pin 41)          direction
             GREEN         4   (pin 43)          clock
              "            5   (pin 44)          direction

        45 & 46 are reserved for moving upper electrode assembly.
        Optocouplers convert the computer 3.4 V (Hi) and 0 V (Lo)
        to 0 V (Hi) and 11.8 V (Lo) when connected to the actuator drives.
        A Hi to Lo step from the computer corresponds to a Lo to Hi step at
        the drive oscillator.  The actuator steps when the oscillator goes
        from  Hi to Lo, where the Lo signal should last at least 10 microsec.

        The drive needs 200 steps/rev and the gear ratio from drive to motor
        is 12:1 for the up and down motor and 3.5:1 for the other two.
        One revolution of the motor moves the piston 0.200".  Therefore,
        there are 12 (200)/0.2 = 12,000 steps/in. = 4724.41 steps/cm for one
        and 3.5 (200)/.2 =3500 steps/in = 1377.952756}

Var
   code, FuncKey       : integer;
   NumberStep          : longint;
   NumberOfDataPoints  : integer;
   InputString         : str8;
   wavelength          : real;
   Reading             : real;
   EndWavelength       : real;
   StepsPerIncrement   : integer;
   I,J                 : integer;
   Y, tempY            : real;
   OldY, OldX          : real;
   TicHeight           : double;
   TicSpace            : real;
   ValueString         : string;

begin
   ClearCommwindow(6);
   Terminate := false;

   {Initiate communication to the MDR of the Spectra Link through the
    COM1 Port}
   InitCom1Port;

   {Check to see if Spectra Link is on remote}
   IsPortBusy (Terminate);

   {Load current position of grating to Spectra Link}
   LoadAbsolutePosition (CurrWavelength,PIEParam.StepsPerAngstrom);

   {Load stepping motor speed and move grating to the target position}
   LoadSpeed;

   {Enter wavelength to collect data.  If <RETURN> is pressed, the current
   wavelength will be used to collect data.}
   GotoXY (5,17);  Write ('Enter Wavelength (A) to collect data:');
   Str (CurrWavelength:8:1,InputString);
   repeat
      GotoXY(50,17);  Write('                    ');
```

```
    GetString (InputString, Numeric, 50, 17);
    Val (InputString, Wavelength, Code);
    GotoXY(50,17);  Write(Wavelength:6:1);
    GotoXY(10,18);  Write ('Is this correct?');
    Reply := readkey;
    If Reply = chr(27) then Terminate := true;
    Reply := UpCase (Reply);
    ClearCommLine (18);
until ((Reply = 'Y') or (Terminate));

If not(Terminate) then begin
    SpatialScan.ScanWavelength := Wavelength;
    {Send target wavelength to Spectra Link}
    LoadTargetPosition (Wavelength,PIEParam.StepsPerAngstrom);

    MoveGrating;

    {Wait until stepping grating has stopped moving}
    GotoXY(5,23);  Write('Moving Grating...        ');
    IsPortBusy (Terminate);
    If Terminate then StopGrating;
    CurrWavelength := wavelength;
    UpDateWavelength(CurrWavelength,PIEParam.GratingGrooveDensity);
    ClearCommLine(23);
end;

{All done with the Spectra Link}
CloseComlPort;

If not(Terminate) then GetSpaScanSpecs (SpatialScan, Terminate);

If (not(Terminate) and (SpatialScan.MotorSpecs.TotalSteps > 0)) then begin
   AssignPortHiLo (SpatialScan.MotorSpecs);

   With SpatialScan do begin
      If MotorSpecs.Motor = 1 then
         StepsPerIncrement := Round(cmBetweenData * XYMotorStepCm)
      else StepsPerIncrement := Round(cmBetweenData * ZMotorStepCm);
      NumberOfDataPoints := Round(MotorSpecs.TotalSteps/StepsPerIncrement);
      MotorSpecs.TotalSteps := StepsPerIncrement * NumberOfDataPoints;
   end;

   {Initialize parameters}
   NumberStep := 0;
   EraseData (SpaFirstPoint,SpaLastPoint);
   With SpatialScan do begin
      StartLocation := 0.0;
      EndLocation   := MotorSpecs.Length;
      Max           := 0.0;
      Min           := 1000.0;
   end;

   {Initialize Keithley 485}
   InitKeithley;

   {Get screen ready to plot data}
   With SpatialScan do begin
      DefineWorld(1,StartLocation,WorldHigh,EndLocation,WorldLow);
      SelectWorld(1);
      SelectWindow(7);
      TicSpace := (EndLocation-StartLocation)/5.0;
      TicHeight:= (WorldHigh-WorldLow)/30.0;
      I := 0;
      repeat
         DrawLine (StartLocation+I*TicSpace,WorldLow,StartLocation+I*TicSpace,TicHeight+WorldLow);
         Str ((StartLocation+I*TicSpace):6:1,ValueString);
         DrawTextW (StartLocation+I*TicSpace,TicHeight*1.8+WorldLow,1,ValueString);
         I := I + 1;
      until (I>=5);
   end;

   GotoXY (5,22);  Write ('F1 = Pause        F10 = Quit');

   {Take first data point}
   {Read data from the PMT via the Keithley 485: averaged over time}
   J := 0;
   tempY := 0;
   repeat
      {Check if data from Keithley 485 is ready}
      IsKeithleyReady;

      {Get data}
      Writeln (IeeeOut, 'ENTER 22');
      Readln (IeeeIn, Response);

      {Determine the real value of the data from the response}
      Response := RemoveSpaces (Response);
      Val (Response,Y, code);
```

```
    Y := -Y;
    tempY := tempY + Y;
    J := J + 1;
    Delay (200);   {wait before taking another data point}
until (J = SpatialScan.ReadingsPerPt);

New (SpaFirstPoint);
SpaLastPoint := SpaFirstPoint;

With SpaLastPoint^, SpatialScan do begin
   {Store data in pointers and records}
   If MotorSpecs.Motor = 1 then XValue := NumberStep / XYMotorStepCm
   else Xvalue := NumberStep / ZMotorStepCm;
   Yvalue := tempY/J;
   NextPoint := nil;
   If Max < Yvalue then Max := Yvalue;
   If Min > Yvalue then Min := Yvalue;
   OldX := Xvalue;
   OldY := Yvalue;
end;

{Start taking data}
FuncKey := 1;
Repeat
   If KeyPressed then GetFuncKey (Funckey);
   If Funckey = 59 then begin
      GotoXY (5,22);  Write ('F2 = Continue');
      Repeat
         repeat until keypressed;
         GetFuncKey (FuncKey);
      until ((FuncKey = 60) or (Funckey = 68));
      If (FuncKey = 60) then begin
         GotoXY (5,22);  Write ('F1 = Pause    ');
         FuncKey := 1;
      end;
   end;

   If Funckey = 68 then begin
      {reset Keithley 485 and DT2811 ports before exiting}
      Writeln (IeeeOut, 'CLEAR 22');
      Writeln (IeeeOut, 'LOCAL 22');
      Port[$21E] :=0;

      {reset ending values of scan}
      With SpatialScan, SpatialScan.MotorSpecs do begin
         TotalSteps := NumberStep;
         If Motor = 1 then Length := NumberStep / XYMotorStepCm
         else Length := NumberStep / ZMotorStepCm;
         GotoXY (5,22);
         Write ('Distance moved (cm) : ', Length:10:3);
         EndLocation := StartLocation + Length;
      end;

      {Return reactor to original position}
      GotoXY(5,24);  Write('Returning reactor to original position...');
      ResetMotorSpecs (SpatialScan.MotorSpecs);
      Terminate := false;
      With SpatialScan.MotorSpecs do begin
         If TotalSteps <> 0 then begin
            NumberStep := 0;
            repeat
               GotoXY (60,24);  Write (NumberStep);
               Port[$21E] := LowMotor;
               Delay (1);
               Port[$21E] := HighMotor;
               NumberStep := NumberStep + 1;
               If keypressed then begin
                  Reply := readkey;
                  If Reply = chr(27) then Terminate := true;
               end;
            until ((NumberStep = TotalSteps) or Terminate);
            TotalSteps := NumberStep;
         end;
      end;
      EXIT;
   end;

   {Move one increment}
   With SpatialScan.MotorSpecs do begin
      J := 0;
      repeat
         GotoXY (5,24);  Write (NumberStep);
         Port[$21E] := LowMotor;
         Delay (1);
         Port[$21E] := HighMotor;
         NumberStep := NumberStep + 1;
         J := J + 1;
      until (J = StepsPerIncrement);
```

```
    end;

    {Wait 2 secs after reactor stops moving before taking data, the other
    second is in the repeat loop}
    Delay(1000);
    {Read data from the PMT via the A/D convertor: averaged over time}
    J := 0;
    tempY := 0;
    repeat
       Delay (1000);    {Wait one second between data}
       {Check to see if data from Keithley 485 is ready}
       IsKeithleyReady;

       {Get data}
       Writeln (IeeeOut, 'ENTER 22');
       Readln (IeeeIn, Response);

       {Determine the real value of the data from the response}
       Response := RemoveSpaces (Response);
       Val (Response,Y, code);
       Y := -Y;
       tempY := tempY + Y;
       J := J + 1;
    until (J = SpatialScan.ReadingsPerPt);

    {Advance pointers to the next location}
    New (SpaLastPoint^.NextPoint);
    SpaLastPoint := SpaLastPoint^.NextPoint;

    With SpaLastPoint^, SpatialScan do begin
       {Store data in pointers and records}
       If MotorSpecs.Motor = 1 then XValue := NumberStep / XYMotorStepCm
       else Xvalue := NumberStep / ZMotorStepCm;
       Yvalue := tempY/J;
       NextPoint := nil;
       If Max < Yvalue then Max := Yvalue;
       If Min > Yvalue then Min := Yvalue;

       {Plot data on screen}
       If ((Yvalue > WorldHigh) or (Yvalue < WorldLow)) then begin
          If (Yvalue > WorldHigh) then WorldHigh := 1.2 * Yvalue;
          If (Yvalue < WorldLow)  then WorldLow  := 1.2 * Yvalue;
          With SpatialScan do
             DefineWorld (1,StartLocation,WorldHigh,EndLocation,WorldLow);
          SelectWorld(1);
          SelectWindow(7);
       end;

       DrawLine (OldX, OldY, Xvalue, Yvalue);
       OldX := Xvalue;
       OldY := Yvalue;
    end;

Until (NumberStep >= SpatialScan.MotorSpecs.TotalSteps);
SpatialScan.MotorSpecs.TotalSteps := NumberStep;

With SpatialScan,SpatialScan.MotorSpecs do begin
   If Motor = 1 then Length := NumberStep / XYMotorStepCm
   else Length := NumberStep / ZMotorStepCm;
   EndLocation := StartLocation + Length;
   GotoXY (5,22);  Write ('Distance moved (cm) : ', Length:10:3);
end;

Sound(500);  Delay(500);  NoSound;

{Reset Keithley 485 and DT2811 Port to zero}
Writeln (IeeeOut, 'CLEAR 22');
Writeln (IeeeOut, 'LOCAL 22');
Port[$21E] := 0;

{Return reactor to original position}
GotoXY(5,24);  Write('Returning reactor to original position...');
ResetMotorSpecs (SpatialScan.MotorSpecs);
Terminate := false;
With SpatialScan.MotorSpecs do begin
   If TotalSteps <> 0 then begin
      NumberStep := 0;
      repeat
         GotoXY (60,24);  Write (NumberStep);
         Port[$21E] := LowMotor;
         Delay (1);
         Port[$21E] := HighMotor;
         NumberStep := NumberStep + 1;
         If keypressed then begin
            Reply := readkey;
            If Reply = chr(27) then Terminate := true;
         end;
      until ((NumberStep = TotalSteps) or Terminate);
```

```
            TotalSteps := NumberStep;
          end;
       end;
       Sound(500);  Delay(500);  NoSound;
    end;
    If Terminate then begin
       ClearCommWindow (6);
       GotoXY(5,24);  Write('Press Function Key....');
    end;
end;

PROCEDURE GetWaveScanData(var WavelengthScan:WavelengthScanRecord;
                          var WavFirstPoint,WavLastPoint:DataPointer;
                              PIEParam:PIERecord;
                          var CurrWavelength:real; var Terminate:boolean);
var
   Reply              : char;
   InputString        : str8;
   Code               : integer;
   FuncKey            : integer;
   MotorSpecs         : MotorRecord;
   reading            : real;
   Wavelength         : real;
   StepsPerIncrement  : integer;   {of grating stepping motor}
   NumberOfDataPoints : integer;
   J,I                : integer;
   Y,Yold,Xold        : real;
   tempY              : real;
   TicHeight          : double;
   TicSpace           : real;
   ValueString        : string;

Begin
   Terminate := false;
   MoveReactor(MotorSpecs,Terminate);

   {Move grating to start of wavelength scan}
   If not(Terminate) then begin
      {Initiate communication with the MDR unit of Spectra Link}
      InitComlPort;

      {Check to see if Spectra Link is on remote}
      IsPortBusy (Terminate);

      {Load current position of grating}
      If not (Terminate) then
         LoadAbsolutePosition (CurrWavelength,PIEParam.StepsPerAngstrom);

      {Load stepping motor}
      If not(terminate) then LoadSpeed;

      {Enter target wavelength}
      If not(Terminate) then begin
         GotoXY(5,20);  Write('Enter Starting Wavelength (A):');
         Str(CurrWavelength:8:1, InputString);
         repeat
            GotoXY(37,20);  Write('           ');
            GetString (InputString, Numeric, 37,20);
            Val (InputString,Wavelength,Code);
            GotoXY(37,20);  Write(Wavelength:8:1);
            GotoXY(5,21);  Write('Is this correct?');
            Reply := UpCase(readkey);
            ClearCommLine(21);
         until (Reply = 'Y');
         WavelengthScan.StartWave := wavelength;
         WavelengthScan.StartWaveMeter :=
            wavelength*PIEParam.GratingGrooveDensity/1200.0;
         {Send target wavelength to Spectra Link}
         LoadTargetPosition (Wavelength,PIEParam.StepsPerAngstrom);

         GotoXY(5,24);  Write('Moving Grating...        ');
         MoveGrating;
      end;

      {Wait until stepping motor has stopped moving before sending next command}
      If not(Terminate) then begin
         IsPortBusy (Terminate);
         If Terminate then StopGrating;
      end;

      If not(Terminate) then begin
         CurrWavelength := wavelength;
         UpDateWavelength(CurrWavelength,PIEParam.GratingGrooveDensity);
         ClearCommLine(24);
         With WavelengthScan, PIEParam do begin
            repeat
               ClearCommLine(21);
               ClearCommLine(22);
```

```
            ClearCommLine(23);
            GotoXY(5,21);  Write('Enter Final Wavelength (A)   :');
            Str(Wavelength, InputString);
            GetString (InputString, Numeric, 37,21);
            Val (InputString,Wavelength,Code);
            EndWave := wavelength;

            GotoXY(5,22);  Write('Wavelength Increments (A)    :');
            GetString (InputString, Numeric, 37,22);
            Val (InputString,WaveIncrement,Code);
            StepsPerIncrement := Round(WaveIncrement * StepsPerAngstrom);
            WaveIncrement := StepsPerIncrement/StepsPerAngstrom;

            If ((StartWave < EndWave) and (WaveIncrement <0)) then
               WaveIncrement := - WaveIncrement;
            If ((StartWave > EndWave) and (WaveIncrement >0)) then
               WaveIncrement := - WaveIncrement;
            GotoXY(37,22);  Write(WaveIncrement:8:1);

            NumberOfDataPoints := Round((EndWave - StartWave) / WaveIncrement);
            EndWave := StartWave + NumberOfDataPoints * WaveIncrement;
            EndWaveMeter := EndWave * PIEParam.GratingGrooveDensity/1200.0;
            GotoXY(37,21);  Write(EndWave:8:1);

            GotoXY(5,23);  Write('Number of readings per point :');
            ReadingsPerPt := 1;
            Str (ReadingsPerPt,InputString);
            GetString (InputString,Numeric,37,23);
            Val (InputString,ReadingsPerPt,Code);

            GotoXY(5,24);  Write('Is everytning correct? (ESC to QUIT)');
            Reply := readkey;
            If Reply = chr(27) then Terminate := true;
            Reply := UpCase(Reply);
            ClearCommLine(24);
         until (Terminate or (Reply = 'Y'));
      end;
   end;
end;

If not(Terminate) then begin
   ClearCommWindow(6);
   With MotorSpecs do begin
      GotoXY(5,17);  Write('L(cm) = ', Length:4:1);
      GotoXY(20,17);
      If Motor = 1 then Write('Motor: Horizontal')
      else Write('Motor: Vertical');
      GotoXY(40,17);  Write('Drive: ',Drive);
      GotoXY(50,17);  Write('Direction: ',Direction);
   end;
   With WavelengthScan do begin
      GotoXY(5,18);  Write('Start Wavelength (A): ',StartWave:8:1);
                     Write('     Reading: ',StartWaveMeter:8:1);
      GotoXY(5,19);  Write('End Wavelength (A)   : ',EndWave:8:1);
                     Write('     Reading: ',EndWaveMeter:8:1);
      GotoXY(5,20);  Write('Increment Size       : ',WaveIncrement:8:1);
                     Write('     Readings Per Point: ',ReadingsPerPt);
   end;
end;

If not(Terminate) then begin
   {Initialize parameters}
   EraseData(WavFirstPoint,WavLastPoint);
   With WavelengthScan do begin
      Max := 0.0;
      Min := 1000;
      wavelength := StartWave;
   end;

   {Initialize the Keithley 485}
   InitKeithley;

   {Get screen ready to plot data}
   With WavelengthScan do begin
      DefineWorld(1,StartWave,WorldHigh,EndWave,WorldLow);
      SelectWorld(1);
      SelectWindow(7);
      TicSpace := (EndWave-StartWave)/5.0;
      TicHeight:= (WorldHigh-WorldLow)/30.0;
      I := 0;
      repeat
         DrawLine (StartWave+I*TicSpace,WorldLow,StartWave+I*TicSpace,TicHeight+WorldLow);
         Str ((StartWave+I*TicSpace):6:1,ValueString);
         DrawTextW (StartWave+I*TicSpace,TicHeight*1.8+WorldLow,1,ValueString);
         I := I + 1;
      until (I>=5);
   end;
```

```
GotoXY(5,24);  Write('F1 = Pause          F10 = Quit');

{Get first point}
J := 0;
tempY := 0.0;
repeat
   {Check to see if data from Keithley 485 is ready}
   IsKeithleyReady;

   {Get data}
   Writeln (IeeeOut, 'ENTER 22');
   Readln (IeeeIn, Response);

   {Determine the real value of the data from the response}
   Response := RemoveSpaces (Response);
   Val (Response,Y, code);
   Y := -Y;
   tempY := tempY + Y;
   J := J + 1;
until (J = WavelengthScan.ReadingsPerPt);

New(WavFirstPoint);
WavLastPoint := WavFirstPoint;

With WavelengthScan,WavLastPoint^ do begin
   Yvalue := tempY/J;
   Xvalue := StartWave;
   NextPoint := nil;
   If Yvalue > Max then Max := Yvalue;
   If Yvalue < Min then Min := Yvalue;
   YOld := Yvalue;
   XOld := Xvalue;
end;

{Start data aquisition}
I := 0;
FuncKey := 1;
repeat
   If KeyPressed then GetFuncKey(FuncKey);
   If FuncKey = 59 then begin
      GotoXY(5,24);  Write('F2 = Continue  ');
      Repeat
         repeat until keypressed;
         GetFuncKey(FuncKey);
      until ((FuncKey = 60) or (FuncKey = 68));
      If FuncKey = 60 then begin
         GotoXY(5,24);  Write('F1 = Pause     ');
         FuncKey := 1;
      end;
   end;

   If FuncKey = 68 then begin
      StopGrating;
      {reset Keithley ammeter and Spectra Link}
      Writeln (Ieeeout, 'CLEAR 22');
      Writeln (IeeeOut, 'LOCAL 22');
      CloseComlPort;

      WavelengthScan.EndWave := wavelength;
      CurrWavelength         := wavelength;
      UpDateWavelength(CurrWavelength,PIEParam.GratingGrooveDensity);
      WavelengthScan.EndWaveMeter :=
         wavelength * PIEParam.GratingGrooveDensity/1200.0;
      EXIT;
   end;

   {Move grating to next wavelength increment}
   IsPortBusy (Terminate);
   LoadAbsolutePosition(CurrWavelength,PIEParam.StepsPerAngstrom);
   LoadSpeed;
   Wavelength := Wavelength + WavelengthScan.WaveIncrement;
   LoadTargetPosition(wavelength,PIEParam.StepsPerAngstrom);
   MoveGrating;
   {Wait until grating done moving}
   IsPortBusy (Terminate);
   If Terminate then StopGrating;
   CurrWavelength := wavelength;
   UpDateWavelength(CurrWavelength,PIEParam.GratingGrooveDensity);
   Delay(100);  {wait for new data to travel from PMT to Keithley Picoammeter}

   {Read data from PMT via the A/D convertor: averaged over time}
   If not(Terminate) then begin
      J := 0;
      tempY := 0.0;
      repeat
         {Check to see if data from Keithley 485 is ready}
         IsKeithleyReady;
```

```
            {Get data}
            Writeln (IeeeOut, 'ENTER 22');
            Readln (IeeeIn, Response);

            {Determine the real value of the data from the response}
            Response := RemoveSpaces (Response);
            Val (Response, Y, code);
            Y := -Y;
            tempY := tempY + Y;
            J := J + 1;
            Delay (10);     {Wait before taking next data point}
         until (J = WavelengthScan.ReadingsPerPt);

         New(WavLastPoint^.NextPoint);
         WavLastPoint := WavLastPoint^.NextPoint;

         With WavelengthScan,WavLastPoint^ do begin
            Yvalue := tempY/J;
            Xvalue := wavelength;
            NextPoint := nil;
            If Yvalue > Max then Max := Yvalue;
            If Yvalue < Min then Min := Yvalue;

            {Plot data on screen}
            If ((Yvalue > WorldHigh) or (Yvalue < WorldLow)) then begin
               If (Yvalue > WorldHigh) then WorldHigh := 1.2 * Yvalue;
               If (Yvalue < WorldLow)  then WorldLow  := 1.2 * Yvalue;
               With WavelengthScan do
                  DefineWorld (1,StartWave,WorldHigh,EndWave,WorldLow);
               SelectWorld(1);
               SelectWindow(7);
            end;
            DrawLine(Xold,Yold,Xvalue,Yvalue);
            YOld := Yvalue;
            XOld := Xvalue;
         end;
         I := I + 1;
      end;
   until ((I = NumberOfDataPoints) or Term_nate);
   WavelengthScan.EndWave := wavelength;
   CurrWavelength         := wavelength;
   UpDateWavelength(CurrWavelength,PIEParam.GratingGrooveDensity);
   WavelengthScan.EndWaveMeter :=
      wavelength * PIEParam.GratingGrooveDensity/1200.0;
   {Reset Ieee ports and Spectra Link}
   Writeln(IeeeOut,'CLEAR 22');
   Writeln(IeeeOut,'LOCAL 22');
   CloseComlPort;
end;
If Terminate then begin
   ClearCommWindow (6);
   GotoXY(5,24);  Write('Press Function Key....');
end;
end;

Procedure GetXtraData;
{This procedure retrieves PIE scan data from floppy or hard disk}

var
   Max,Min       : real;
   Terminate     : boolean;
   Reply         : char;
   InputString   : str60;
   DirInfo       : SearchRec;
   DataFile      : text;

begin
   Terminate := false;
   GotoXY(5,17);  Write('Path+Filename:');
   GetDescription (InputString, 20, 17);
   If (InputString[1] = chr(27)) then begin
      Terminate := true;
      Sound(1000);  Delay(500);  NoSound;
   end;

   If not(Terminate) then begin
      GotoXY(5,18);  Write('Type of data:');
      GotoXY(10,19); Write('(1)=Spatial Scan  (2)=Wavelength Scan');
      Reply := readkey;
      ClearCommLine (19);
      GotoXY(20,18);
      Case Reply of
         '1':begin
               InputString := InputString+'.SPA';
               Write ('Spatial Scan');
             end;
         '2':begin
               InputString := InputString+'.WAV';
```

```
            Write ('Wavelength Scan');
          end;
       else begin
          GotoXY(5,19);  Write('INVALID key pressed');
          Terminate := true;
          Sound (1000);  Delay(500);  NoSound;
       end;
    end;   {Case}
  end;

  If not(Terminate) then begin
     FindFirst (InputString, AnyFile, DirInfo);
     If DosError <> 0 then begin
        GotoXY (5,19);
        If DosError = 2 then Write ('File NOT FOUND');
        If DosError = 3 then Write ('Path NOT FOUND');
        If DosError = 8 then Write ('Not enough memory');
        If DosError =11 then Write ('Invalid format');
        GotoXY (25,19);  Write ('ERROR!!!');
        Terminate := true;
        Sound (1000);  Delay(500);  NoSound;  Delay(1000);
     end;
  end;

  If not(Terminate) then begin
     XtraFirstPoint := nil;
     XtraLastPoint  := nil;
     GotoXY (5,19);  Write ('Retrieving data....');
     Max := 0.0;
     Min := 1000;
     Assign (DataFile, InputString);
     ReSet (DataFile);
     While not Eof(DataFile) do begin
        If XtraFirstPoint = nil then begin
           New (XtraFirstPoint);
           XtraLastPoint := XtraFirstPoint;
        end else begin
           New (XtraLastPoint^.NextPoint);
           XtraLastPoint := XtraLastPoint^.NextPoint;
        end;
        With XtraLastPoint^ do Readln (DataFile,Xvalue,Yvalue);
        XtraLastPoint^.NextPoint := nil;
        With XtraLastPoint^ do begin
           If Yvalue > Max then Max := Yvalue;
           If Yvalue < Min then Min := Yvalue;
        end;
     end;
     Close (DataFile);

     ClearGraphWindow;  SelectWindow (7);  DrawBorder;
     GotoXY(5,19);  Write ('Plotting....                ');
     DrawSpectra (XtraFirstPoint,Max,Min,XtraFirstPoint^.Xvalue,
                  XtraLastPoint^.Xvalue,7);
     EraseData (XtraFirstPoint,XtraLastPoint);
  end;
  ClearCommWindow(6);
  GotoXY(5,24);  Write('Press Function Key....');
end;


BEGIN
  WorldHigh := 1.0E-6;           {High and low values for initial}
  WorldLow  := -0.00000001;      {graph window.}
  WritePIEScreen (OpticalData.PIEParam,CurrWavelength);
  GetPIEParam (OpticalData.PIEParam,CurrWavelength);

  IsExitPIE  := false;

  repeat
     FuncKey := 1;
     If keypressed then GetFuncKey (FuncKey);
     Case FuncKey of

        {F1 = PIE Param.}
        59:GetPIEParam (OpticalData.PIEParam,CurrWavelength);

        {F2 = Move Reactor}
        60:begin
             MoveReactor(ReactorMotorSpecs,Terminate);
             GotoXY(5,24);  Write ('Press Function Key ....');
           end;

        {F3 = Spatial Scan}
        61:begin
             With OpticalData do begin
                {If there is a graph in the graph window, clear window}
                If (PIEParam.IsSpatialScan) or (PIEParam.IsWavelengthScan) then begin
                   ClearGraphWindow;
```

```
            SelectWindow (7);  DrawBorder;
         end;
         GetSpatialScan(SpatialScan,SpaFirstPoint,SpaLastPoint,
                        PIEParam,CurrWavelength,Terminate);
         If not(Terminate) then begin
            WritePIEScreen (PIEParam,CurrWavelength);
            With SpatialScan do
               DrawSpectra (SpaFirstPoint, Max, Min, StartLocation,
                            EndLocation, 7);
            PIEParam.IsSpatialScan := true;
            Status.IsOWaveSpectra := true;
         end;
      end;
   end;

{F4 = Wavelength Scan}
62:begin
      With OpticalData do begin
         {If there is a graph in the graph window, clear window}
         If (PIEParam.IsSpatialScan) or (PIEParam.IsWavelengthScan) then begin
            ClearGraphWindow;
            SelectWindow (7);  DrawBorder;
         end;
         GetWaveScanData (WavelengthScan,WavFirstPoint,WavLastPoint,
                          PIEParam,CurrWavelength,Terminate);
         If not(Terminate) then begin
            WritePIEScreen (PIEParam,CurrWavelength);
            With WavelengthScan do
               DrawSpectra (WavFirstPoint, Max, Min, StartWave,
                            EndWave, 7);
            PIEParam.IsWavelengthScan := true;
            Status.IsOWaveSpectra := true;
         end;
      end;
   end;

{F5 = Plot Scan}
63:begin
      ClearCommWindow(6);
      GotoXY (5,17);
      Write ('Which Spectra? S=Spatial Scan  W=Wavelength Scan');
      Reply := UpCase(readkey);
      With OpticalData do begin
         Case Reply of
            'S':If (PIEParam.IsSpatialScan) then begin
                   ClearGraphWindow;  SelectWindow (7);  DrawBorder;
                   With SpatialScan do
                      DrawSpectra (SpaFirstPoint, Max, Min,
                                   StartLocation, EndLocation, 7);
                end else begin
                   GotoXY (5,19);  Write ('NO SPECTRA TO PLOT');
                   Sound(1000);  Delay(500);  NoSound;
                end;
             'W':If (PIEParam.IsWavelengthScan) then begin
                   ClearGraphWindow;  SelectWindow (7);  DrawBorder;
                   With WavelengthScan do
                      DrawSpectra (WavFirstPoint, Max, Min,
                                   StartWave, EndWave, 7);
                end else begin
                   GotoXY (5,19);  Write ('NO SPECTRA TO PLOT');
                   Sound(1000);  Delay(500);  NoSound;
                end;

             else begin
                GotoXY (5,19);  Write ('INVALID KEY PRESSED');
             end;
         end;   {Case}
      end;   {with of OpticalData}
      ClearCommLine(17);
      GotoXY(5,24);  Write('Press Function Key....');
end;   {F5}


{F6 = Retrieve Data}
64:begin
      ClearCommWindow(6);
      GetXtraData;
   end;

{F7 = Scan Data}
65:begin
      ClearCommWindow(6);
      With OpticalData.PIEParam do begin
         If IsWavelengthScan and IsSpatialScan then begin
            GotoXY(5,17);  Write('Which graph?  W=wavelength  S=spatial: ');
            repeat
               Reply := UpCase(readkey);
               GotoXY(44,17);  Write(Reply);
```

```
                        If not(Reply = 'W') and not(Reply = 'S') then begin
                           GotoXY(5,18);  Write('Wrong key pressed!');
                           Sound(700);  Delay(200);  NoSound;
                           GotoXy(44,17);  Write (' ');
                        end;
                     until ((Reply = 'W') or (Reply = 'S'));
                  end;
                  If IsWavelengthScan and not(IsSpatialScan) then Reply := 'W';
                  If not(IsWavelengthScan) and IsSpatialScan then Reply := 'S';
                  If not(IsWavelengthScan) and not(IsSpatialScan) then begin
                     GotoXY(5,17);  write('No data to scan!');
                     Sound(700);  Delay(200);  NoSound;
                     reply := 'C';
                  end;

                  Case Reply of
                  'W':begin
                        ClearCommWindow(6);
                        With OpticalData, OpticalData.WavelengthScan do
                        ScanData(WavFirstPoint,StartWave,EndWave,Max,Min,7);
                        ClearCommwindow(6);
                      end;

                  'S':begin
                        ClearCommWindow(6);
                        With OpticalData, OpticalData.SpatialScan do
                        ScanData(SpaFirstPoint,StartLocation,EndLocation,Max,Min,7);
                        ClearCommWindow(6);
                      end;

                  end;  {Case}
               end;   {with PIEParam}
               GotoXY(5,24);  Write('Press Function Key....');
         end;   {F7=ScanData}

         {F10 = Exit}
         68:IsExitPIE := true;

      end;   {Case}
   until (IsExitPIE);
end;
END.   {Unit}
```

Unit ATools;

```
INTERFACE
uses DOS, Crt, GDriver, GKernel, GWindow, GShell;


TYPE
   str1   = string[1];
   str7   = string[7];
   str8   = string[8];
   str12  = string[12];
   str60  = string[60];
   str255 = string[255];
   StringKind = (Alfa, Numeric, Mix);

   DataPointer = ^DataList;
   DataList = record
      Xvalue : real;
      Yvalue : double;
      NextPoint : DataPointer;
   end;

   StatusRecord = record
      IsIonESpectra  : boolean;
      IsVWaveSpectra : boolean;
      IsIWaveSpectra : boolean;
      IsOWaveSpectra : boolean;
      Initial        : str8;
      FileName       : str7;
   end;

   SysRecord = record
      Pressure      : real;
      Frequency     : real;
      ElecSep       : real;
      UpElecDiam    : real;
      LowElecDiam   : real;

      Gas1Name         : str8;
      Gas1MeterReading : real;
      Gas1Flow         : real;

      Description1      : str60;
      Description2      : str60;
      Description3      : str60;

   end;

   IonERecord = record
      Grid1 : real;
      Grid3 : real;

      StartEv       : real;
      EndEv         : real;
      StepEv        : real;
      NumberSteps   : real;
      PointsPerStep : integer;
      Max           : double;
      Min           : double;
      IonEnergy      : real;
   end;

   RunDataRecord = record
      IonEFirstPoint, IonELastPoint : DataPointer;
   end;

   DistDataRecord = record
      DistFirstPoint, DistLastPoint : DataPointer;
   end;

   DistriParamRecord = record
      Max : double;
      Min : double;
   end;

   PowerRecord = record
      CurrentMultiplier : real;
      VoltageMultiplier : real;
      AvePower          : double;
   end;

   WorldParam = record
      StartTime         : real;
      EndTime           : real;
      Max               : real;
      Min               : real;
   end;
```

```
DateRecord = record
   Year    : str8;
   Month   : str8;
   Day     : str8;
end;

CommLinePos=17..24;              {Line Positions for Screen Output}
dataarray=array[0..32004] of byte;{Array for Waveform data values}
trigarray=array[0..252] of integer;{Array for Waveform trigger values}
strdescriparr=array[1..3] of string[65];{string length for descriptors}
wfmdescriptor=record             {record of waveform descriptor}
   Preamble:integer;                {LeCroy 9400 communication preamble}
   NumPtsTrans:integer;             {Number of bytes transmitted by 9400}
   FixedVertGain:byte;              {Fixed Vertical Gain}
   VarVertGain:byte;                {Variable Vertical Gain}
   Unused1:integer;                 {UnUsed Byte}
   VertOffset:integer;              {Vertical Offset}
   ChanCoupling:byte;               {Channel Coupling}
   ExtProbeAttn:byte;               {External Probe Attenuation}
   BandWidthLimit:byte;             {Band Width Limit}
   TimeBase:byte;                   {Time Base}
   SamplingIntvl:byte;              {Time Sampling Interval}
   RecordType:ShortInt;             {Record Type, i.e. Interleaved, etc.}
   TrigCoupling:byte;               {Trigger Coupling}
   TrigMode:byte;                   {Trigger Mode}
   TrigSource:byte;                 {Trigger Source}
   TrigSlope:byte;                  {Trigger Slope}
   TrigLevel:integer;               {Trigger Level}
   TrigDelay:LongInt;               {Trigger Delay}
   NumDataPtsDiv:integer;           {Number of Data Points per Division}
   FirstAddr:integer;               {Scope Address of First Data Point}
   LastAddr:integer;                {Scope Address of Last Data Point}
   Internal1:integer;               {Internal Value}
   Internal2:integer;               {Internal Value}
   Internal3:integer;               {Internal Value}
   DataProc:byte;                   {Data Processing of Record}
   Unused2:byte;                    {UnUsed Byte}
   PowerVolts:integer;              {256*Power of Volts}
   PowerSec:integer;                {256*Power of Seconds}
   Reserved1:array[0..23] of byte;  {Reserved}
   Identity:byte;                   {Identity of Function Waveform}
   FunctionType:byte;               {Function Type, i.e., Average, etc.}
   SubFunType:byte;                 {SubFunction Type}
   PrimSource:byte;                 {Primary Source of Waveform}
   SecSource:byte;                  {Secondary Source of Waveform}
   ContAvgWt:byte;                  {Continuous Averaging Weight}
   MaxSweeps:LongInt;               {Maximum Number of Sweeps}
   MultFactor:integer;              {Multiplication Factor *100}
   AddConst:integer;                {Additive Constant *100}
   MaxNumber:integer;               {Maximum Number of Data Points}
   Reject:byte;                     {Rejection for Summed Average}
   Dithering:byte;                  {Dithering for Summed Average}
   Reserved2:array[0..15] of byte;  {Resereved}
   AcqSweeps:LongInt;               {Actually Acquired Number of Sweeps}
   NumOverflows:LongInt;            {Acquired Wfms with Overflow}
   NumUnderflows:LongInt;           {Acquired Wfms with Underflow}
   NumReject:LongInt;               {Acquired Wfms Rejected}
   CompPts:integer;                 {Data Points Used in Computing Record}
   RatioWfmPts:integer;             {Data Points in Wfm/Data Points Used}
   Reserved3:array[0..31] of byte   {Reserved}
end;
wfmtrigtime=record               {Trigger Record}
   numsweeps:integer;               {Number of Sweeps}
   numtrig:integer;                 {Number of Trigger Values}
   trigholder:trigarray;            {Array of Sequential Trigger Values}
end;
wfmdata=record                   {Data Record}
   numpoints:integer;               {Number of Points}
   dataholder:dataarray;            {Array of Sequential Data Values}
end;
waveform=record                  {Waveform Record}
   descriptor:strdescriparr;        {User Description of Waveform}
   descrip9400:wfmdescriptor;       {LeCroy 9400 Description of Waveform}
   trig9400:wfmtrigtime;            {LeCroy 9400 Trigger Times of Waveform}
   data9400:wfmdata;                {LeCroy 9400 Data Values of Waveform}
   PowerParam:PowerRecord;
end;

MotorRecord = record
   TotalSteps : Longint;
   Length     : real;
   HighMotor  : byte;
   LowMotor   : byte;
   Drive, Direction : char;   {color of drive/ extend or retract}
   Motor      : integer;      {1=XY motors or 2=Z motor}
end;

SpatialScanRecord     = record
```

```
      StartLocation      : real;
      EndLocation        : real;
      cmBetweenData      : real;
      ScanWavelength     : real;
      Max, Min           : real;
      ReadingsPerPt      : integer;
      MotorSpecs         : MotorRecord;
   end;

   WavelengthScanRecord  = record
      StartWave          : real;
      EndWave            : real;
      StartWaveMeter     : real;
      EndWaveMeter       : real;
      WaveIncrement      : real;
      DetectionLocation  : real;
      Max, Min           : real;
      ReadingsPerPt      : integer;
   end;

   PIErecord             = record
      IsSpatialScan      : boolean;
      IsWavelengthScan   : boolean;
      GratingGrooveDensity: integer;
      StepsPerAngstrom   : integer;
   end;

   OpticalRec         = record
      PIEParam        : PIERecord;
      SpatialScan     : SpatialScanRecord;
      WavelengthScan  : WavelengthScanRecord;
      SpaFirstPoint   : DataPointer; {Pointers for spatial scan data}
      SpaLastPoint    : DataPointer;
      WavFirstPoint   : DataPointer; {Pointers for wave scan data}
      WavLastPoint    : DataPointer;
   end;


CONST
   MaxWorldsGlb  = 10;
   MaxWindowsGlb = 10;
   RamScreenGlb  = true;

   Path  = 'C:\experime\';
   Path1 = 'C:\experime\INFO\';
   Path2 = 'C:\experime\PIE\';

   MixSet : Set of Char = ['A'..'Z', '0'..'9', '-'];

   Identity:str255='IDENTIFY';       {Request Code for 9400 for Identity}
   Channel1='CHANNEL_1';             {9400 Code for Channel 1}
   Channel2='CHANNEL_2';             {9400 Code for Channel 2}
   MemoryC='MEMORY_C';               {9400 Code for Memory C}
   MemoryD='MEMORY_D';               {9400 Code for Memory D}
   FunctionE='FUNCTION_E';           {9400 Code for Function E}
   FunctionF='FUNCTION_F';           {9400 Code for Function F}
   OutputCode='OUTPUT 04;';          {Bus Address and Send Code for 9400}
   ReceiveCode='ENTER 04';           {Bus Address and Retreive Code for 9400}
   InspectCode='INSPECT ';           {9400 Code for Waveform Record Inspection}
   ReadCode='READ ';                 {9400 Code for Waveform Record Read}
   DescriptorCode='.DESC';           {9400 Waveform Descriptor Code}
   DataCode='.DATA';                 {9400 Waveform Data Code}
   TriggerCode='.TIME';              {9400 Waveform Trigger Times Code}
   LimitsCode='.LIMIT';              {9400 Waveform Address Limits Code}
   SweepsCode='.NSWEEPS';            {9400 Waveform Number of Sweeps Code}
   BufferCode=' BUFFER ';            {IOTech 488 Buffered Data Transfer Code}
   WriteCode='WRITE ';               {9400 Code for Waveform Record Write}
   OutCode='OUTPUT 04';              {Send Code for Buffer 9400 Transfer}
   EOICode=' EOI';                   {End or Identifiy Terminator Code}
   BufferedOutput='BUFFERED';        {IOTech 488 Buffered Data Transfer Code}
   ChannelSet: Set of Char=['1','2','c','d','e','f','C','D','E','F'];
                                     {Characters Representing 9400 Channels}
   YesSet: Set of Char=['y','Y'];    {Characters Meaning Yes}
   NumSet: Set of Char=['-','0','1','2','3','4','5','6','7','8','9'];
                                     {Number Characters}
   ScanSet: Set of Char=['M','m','R','r','B','b'];
                                     {Waveform Characters}

   DriveSet : Set of Char=['G','O','R'];
   DirnSet  : Set of Char=['E','R'];
   XYMotorStepCm = 1377.952756;
   ZMotorStepCm  = 4724.409449;


VAR
   IeeeOut, IeeeIn : Text;           {Files for IOTech 488 communication}
   Regs:registers;                   {Registers for MS DOS Calls}
```

```
    Status            : StatusRecord;
    SysParam          : SysRecord;
    IonEParam         : IonERecord;
    DistributionParam : DistriParamRecord;
    RetrievedParam    : DistriParamRecord;

    RunData           : RunDataRecord;       {Ion Energy Record}
    DistributionData  : DistDataRecord;      {Ion Distribution (differentiated) Record}
    RetrievedData     : DistDataRecord;      {IED pointers for retrieved data}
    WfmDataRec        : waveform;            {Current Waveform}
    IFirstPoint       : DataPointer;         {Pointers for current waveform}
    ILastPoint        : DataPointer;
    VFirstPoint       : DataPointer;         {Pointers for voltage waveform}
    VLastPoint        : DataPointer;
    OFirstPoint       : DataPointer;         {Pointer for other waveforms}
    OLastPoint        : DataPointer;         {data not stored on hard disk}
    IWorldVar         : WorldParam;          {X,Y coordinates for defining world windows}
    VWorldVar         : WorldParam;
    OWorldVar         : WorldParam;

    Response:str255;                {Text Response from LeCroy 9400 or Keithley 485}
    LPos:CommLinePos;               {Line Position for Screen Output}
    FuncKey:integer;                {Ordinal Value of Key Pressed}

    Date              : DateRecord;   {parameters used to generate file name}

    Com1In, Com1Out : text; {files for writing to and reading from the COM1 Port}
    TextIn, TextOut : char; {Characters used in communication with Spectra Link}
    NumOut          : integer; {number in decimal of TextOut}
    Terminate       : boolean;

    OpticalData     : OpticalRec;
    CurrWavelength  : real;


PROCEDURE InitCommun;

PROCEDURE InitKeithley;

PROCEDURE IsKeithleyReady;

PROCEDURE SetUpWindows;

PROCEDURE ClearCommLine (Line:integer);

PROCEDURE ClearCommWindow (WindowNumber:integer);

PROCEDURE ClearGraphWindow;

PROCEDURE GetFuncKey (var Funckey : integer);

PROCEDURE GetString (var InputString  : str8;
                         KindOfString : StringKind;
                         Xpos         : integer;
                         Ypos         : integer);

PROCEDURE GetDescription (var InputString  : str60;
                              Xpos         : integer;
                              Ypos         : integer);

FUNCTION RemoveSpaces (InString : str255) : str255;

PROCEDURE DrawSpectra (FirstPoint : DataPointer;
                       Max        : double;
                       Min        : double;
                       XStart     : real;
                       XEnd       : real;
                       Window     : integer);

PROCEDURE ScanData (FirstPoint : DataPointer;
                    XStart     : real;
                    XEnd       : real;
                    Max        : double;
                    Min        : double;
                    Window     : integer);

PROCEDURE EraseRunData (var RunData : RunDataRecord);

PROCEDURE EraseData (var FirstPoint : DataPointer;
                     var LastPoint  : DataPointer);


IMPLEMENTATION

PROCEDURE InitCommun;

   {Initialize the IEEE Communications and check to make
    sure instruments connected to the IOTech card are working.}
```

```
var
   Response : str255;

begin
   Assign (IeeeOut, 'c:\IeeeOut');
   Rewrite (IeeeOut);
   Assign (IeeeIn, 'c:\IeeeIn');
   Reset (IeeeIn);

   {Initialize Status Request Line}
   Writeln(IeeeOut,'ARM SRQ');

   {Requests IOTech 488 identity}
   GotoXY (2,10);
   Write ('Testing Communication:');
   Writeln (IeeeOut, 'RESET');
   Writeln (IeeeOut, 'HELLO');
   Readln (IeeeIn, Response);
   Response := RemoveSpaces(Response);
   GotoXY (2,11);
   Write (Response);

   {Request Keithley 485 to identify}
   Writeln (IeeeOut, 'OUTPUT 22;U0X');
   Writeln (IeeeOut, 'ENTER 22');
   Readln (IeeeIn, Response);
   Response := RemoveSpaces(Response);
   GotoXY (2,12);
   Write ('   Device --- ', Response, '   OK');

   {Request LeCroy 9400 to identify}
   Writeln(IeeeOut, 'OUTPUT 04;IDENTIFY');
   Writeln(IeeeOut,'ENTER 04');
   Readln(IeeeIn,Response);
   Response:=RemoveSpaces(Response);
   GotoXY (2,13);
   Write ('   Device --- ', Response, '   OK');
   Writeln (IeeeOut, 'RESET');
   Writeln (IeeeOut, 'LOCAL');
end;


PROCEDURE InitKeithley;
{Initialize IEEE communication to the Keithley ammeter}

Begin
   Writeln (IeeeOut, 'REMOTE 22'); {Enable computer control of all devices}
   Writeln (IeeeOut, 'ARM SRQ');   {Enable SRQ to set light-pen interupt}

                  {Send the following commands to the Keithley 485
                       C0  : zero check off
                       D0  : LOG off
                       R0  : auto range control
                       Z0  : REL off
                       K0  : EOI (end or identify) enabled
                       G1  : readings without prefix (data format)
                       M9  : Reading done or reading overflow generates SRQ
                       T5  : trigger is one-shot on X}
   Writeln (IeeeOut, 'OUTPUT 22;C0D0R0Z0K0X');
   Writeln (IeeeOut, 'OUTPUT 22;G1M9X');
   Writeln (IeeeOut, 'OUTPUT 22;T5X');

   {Pause for Keithley to set SRQ mask}
   Delay (100);

   Writeln (IeeeOut, 'OUTPUT 22;X');
   Writeln (IeeeOut, 'ENTER 22');
   Readln (IeeeIn, Response);
end;

PROCEDURE IsKeithleyReady;

Var
   K485        : integer;
   IsInterrupt : boolean;

begin
   Writeln (IeeeOut, 'OUTPUT 22;X');
   K485 := 0;

   {Check for interrupt and determine the cause}
   IsInterrupt := false;
   repeat
      Writeln (IeeeOut, 'SPOLL 22');
      Readln (IeeeIn, K485);
      if (K485 > 63) and ((K485 and 32) <> 1 ) then begin
         if (K485 and 8) <> 0 then IsInterrupt := true;
         if (K485 and 0) <> 0 then begin
```

```
            GotoXY (5,24);  Write ('ERROR:  Reading Overflow');
         end;
      end;

      if (K485 and 32) <> 0 then begin
         GotoXY (5,24);
         Write ('ERROR');
         Writeln (IeeeOut, 'OUTPUT 22;U0X');
         Writeln (IeeeOut, 'ENTER 22');
         Readln (IeeeIn, Response);
         Response := RemoveSpaces(Response);
         GotoXY (10,25);  Write ('  ', Response);
      end;
   until (IsInterrupt);
end;


PROCEDURE SetUpWindows;

         {Define all windows for all sections of the program.  This
          includes the main section plus those used by Power and IonE.
          Window 1:  Full screen window.  Used in Main, IonE and Analyze
                     sections.
          Window 2:  There are three small windows in the main screen.
                     This is the left one.
          Window 3:  This is the center one.
          Window 4:  This is the right one.
          Window 5:  Command window in Power & PIE Sections (Upper right).
          Window 6:  Communication window in Power & PIE Sections (Bottom).
          Window 7:  Graph window in Power & PIE Sections (Upper left).
          Window 8:  Graph window in IonE section (Upper left).
          Window 9:  Parameters in IonE section (bottom).}

const
   MaxWorldX:Float=1000.0;          {Maximum World for X-dir. for Window 1}
   MaxWorldY:Float=1000.0;          {Maximum World for Y-dir. for Window 1}


begin

   {Main program windows}

   DefineWindow (2,TextLeft (4,0), TextUp(2,0),
                   TextRight (25,0), TextDown(8,0));  {work window}
   DefineWindow (3,TextLeft (29,0), TextUp(2,0),
                   TextRight (50,0), TextDown(8,0));  {work window}
   DefineWindow (4,TextLeft (54,0), TextUp(2,0),
                   TextRight (75,0), TextDown(8,0));  {work window}

   {Command Window in Unit Power}
   DefineWindow (5,(XMaxGlb-23),12,(XMaxGlb-2),(YMaxGlb-150));

   {Communication Window in Unit Power}
   DefineWindow (6,2,(YMaxGlb-138),(XMaxGlb-2),(YMaxGlb-12));

   {Graph Window in Unit Power}
   DefineWindow (7,2,12,(XMaxGlb-25),(YMaxGlb-150));

   {Graph Window in Unit IonE}
   DefineWindow (8,2,12,(XMaxGlb div 2)+20, (YMaxGlb div 2)+45);

   {Command Window in Unit IonE}
   DefineWindow (1,(XMaxGlb-23),12,(XMaxGlb-2),(YMaxGlb div 2)+45);

   {Parameter Window in Unit IonE and Main program}
   DefineWindow (9,2,(YMaxGlb div 2)+50,XMaxGlb-2, YMaxGlb);

   {Graph Window in Unit Analyze - defined in procedure InLargeSpectra
    in the Unit itself}

   {Global screen - currently used in Unit Analyze only}
   DefineWindow (11, 0, 0, XMaxGlb,YMaxGlb);

end;


PROCEDURE ClearCommLine (Line: integer);
    {Clears command line in main section and ion energy section}

begin
   GotoXY (3,Line);
   Write ('                                                                                ');
end;

PROCEDURE ClearCommWindow (WindowNumber:integer);
{This Procedure clears the communication window.}
```

```
var
   Lpos:CommLinePos;                {Screen Output Position}
   i:integer;                       {Counter}

begin
  {Start for top of communciation window (17) and write blanks to
   all lines in communication window}

   If WindowNumber = 6 then i:=17;
   If WindowNumber = 9 then i:=18;

   For Lpos:=i to 24 do begin
      GotoXY(4,Lpos);
      Write('                                                                    ');
   end;
end;


PROCEDURE ClearGraphWindow;
{This procedure clears the graph window in waveform section}

var
   Lpos:integer;                {Screen Output Position}
   i:integer;                        {Counter}

begin
  {Start for top of graph window (6) and write blanks to
   all lines in communication window}

   For Lpos:=3 to 14 do begin
      GotoXY(3,Lpos);
      for i:=1 to 56 do Write(' ');
   end;
end;


PROCEDURE GetFuncKey (var Funckey : integer);

var
   Key : char;

begin
   Key := ReadKey;
   if Key = #0 then begin
      Key := ReadKey;
      FuncKey := ord(Key);
   end
   else begin
      Sound (300); Delay (100); NoSound;
      FuncKey := 1;
   end;
end;


PROCEDURE GetString (var InputString  : str8;
                         KindOfString : StringKind;
                         Xpos         : integer;
                         Ypos         : integer);
                             {The procedure obtains a string from the
                              user.  The string is terminasted by a return.}
var
   CharValue, BoundLow, BoundHi, LetterCount : integer;
   CharIn : char;
   NewString : string[8];
   IsReturn : boolean;

begin
   IsReturn := false;
   LetterCount := 0;
   NewString := '';

   if (KindOfString = Alfa) then begin
      BoundLow := 64; BoundHI := 91; end;
   if (KIndOfString = Numeric) then begin
      BoundLow := 44; BoundHi := 58; end;
   if (KindOfString = Mix) then begin
      BoundLow := 44; BoundHi := 91; end;

   repeat

      repeat

       {Get a character}
         CharIn := readkey;

       {Check for special code and if true read the next character}
```

```
         if (CharIn = #0) then begin
            CharIn := readkey;
            CharValue := 0;
         end

         else begin
            CharIn := UpCase(CharIn);
            CharValue := ord(CharIn);

          {Check fo termination}
            if (CharIn = chr(13)) then IsReturn := true;
            if (NewString = '') then begin
               GotoXY (Xpos,Ypos);
               Write ('          ');
               GotoXY (Xpos,Ypos);
            end;

          {Check for backspace}
            if (CharIn = chr(8)) then begin
               Delete (NewString, LetterCount, 1);
               LetterCount := LetterCount - 1;
               CharValue := 0;
               GotoXY (Xpos,Ypos);
               Write ('      ');
               GotoXY (Xpos,Ypos);
               Write (NewString);
            end;
         end;

        {Inform of a not acceptable character}
         if not ((((CharValue > BoundLow) and (CharValue < BoundHi))
            or IsReturn) or (CharValue = 32))  then begin
             Sound (300);
             Delay (200);
             NoSound;
         end;

      until (((CharValue > BoundLow) and (CharValue < BoundHi)) or IsReturn);

      if (CharIn <> chr(13)) then begin
         if ((NewString = '') and (CharIn = '.')) then NewString := '0';
         NewString := NewString+CharIn;
         LetterCount := LetterCount + 1;
         Write (CharIn);
      end;

   until IsReturn;

   if (NewString <> '') then InputString := NewString;
end;


PROCEDURE GetDescription (var InputString  : str60;
                              Xpos         : integer;
                              Ypos         : integer);
                             {The procedure obtains a string from the
                              user.  The string is terminated by a return.}
var
   CharValue, LetterCount : integer;
   CharIn : char;
   NewString : str60;
   IsReturn : boolean;

begin
   IsReturn := false;
   LetterCount := 0;
   NewString := '';

   repeat

      repeat

       {Get a character}
         CharIn := readkey;

       {Check for special code and if true read the next character}
         if (CharIn = #0) then begin
            CharIn := readkey;
            CharValue := 0;
         end

         else begin
            CharValue := ord(CharIn);

          {Check fo termination}
            if (CharIn = chr(13)) then IsReturn := true;
            if (NewString = '') then begin
```

```
            GotoXY (Xpos,Ypos);
            Write ('                                ',
                   '                                ');
            GotoXY (Xpos,Ypos);
          end;

        {Check for backspace}
          if (CharIn = chr(8)) then begin
            Delete (NewString, LetterCount, 1);
            LetterCount := LetterCount - 1;
            CharValue := 0;
            GotoXY (Xpos,Ypos);
            Write ('                                ',
                   '                                ');
            GotoXY (Xpos,Ypos);
            Write (NewString);
          end;
        end;

       {Inform of a not acceptable character}
        if not (((CharValue > 31) and (CharValue < 123)) or IsReturn)
          then begin
           Sound (300);
           Delay (200);
           NoSound;
        end;

      until (((CharValue > 31) and (CharValue < 123)) or IsReturn);

      if not(IsReturn) then begin
        NewString := NewString+CharIn;
        LetterCount := LetterCount + 1;
        Write (CharIn);
      end;

   until IsReturn;

   if (NewString <> '') then InputString := NewString;
end;


FUNCTION RemoveSpaces (InString : str255) : str255;

begin
   GotoXY (2,5);
   while not (InString[1] in MixSet) do
      delete (InString,1,1);
   RemoveSpaces := InString;
end;


PROCEDURE DrawSpectra (FirstPoint : DataPointer;
                       Max        : double;
                       Min        : double;
                       XStart     : real;
                       XEnd       : real;
                       Window     : integer);

var
   CurrentPoint : DataPointer;
   X, XOld : real;
   Y, YOld : double;
   TicHeight : double;
   TicSpace  : real;
   I         : integer;
   ValueString : string;

begin
   SelectScreen (1);
   if (Max <= Min) then Max := Min + 1.0;
   DefineWorld (4, XStart, Max+(Max-Min)*0.05, XEnd, Min-(Max-Min)*0.05);
   SelectWorld (4);
   SelectWindow (Window);
   TicHeight := (Max-Min)/30.0;
   TicSpace  := (XEnd-XStart)/5.0;
   I := 0;
   repeat
      DrawLine (XStart+TicSpace*I,Min-(Max-Min)*0.05,XStart+TicSpace*I,
                TicHeight+Min-(Max-Min)*0.05);
      Str ((Xstart+TicSpace*I):8:3,ValueString);
      DrawTextW (XStart+TicSpace*I,TicHeight+Min,1,ValueString);
      I := I + 1;
   until (I >= 5);

   CurrentPoint := FirstPoint;
   XOld := CurrentPoint^.Xvalue;
```

```
   YOld := CurrentPoint^.Yvalue;
   while (CurrentPoint <> nil) do begin
      X := CurrentPoint^.Xvalue;
      Y := CurrentPoint^.Yvalue;
      DrawLine (XOld, YOld, X, Y);
      CurrentPoint := CurrentPoint^.NextPoint;
      XOld := X;
      YOld := Y;
   end;
end;


PROCEDURE ScanData (FirstPoint : DataPointer;
                    XStart     : real;
                    XEnd       : real;
                    Max        : double;
                    Min        : double;
                    Window     : integer);

var
   CurrentPoint     : DataPointer;
   X                : real;
   Y                : double;
   Counter          : integer;
   Positioncounter  : integer;
   FuncKey          : integer;

begin
   DefineWorld (4, XStart, Max+(Max-Min)*0.05, XEnd, Min-(Max-Min)*0.05);
   SelectWorld (4);
   SelectWindow (Window);

   GotoXY (5,23); Write ('<-- left     right -->      F10=Quit                                    ');
   GotoXY (5,24);
   Write ('Browsing');

 {Find the point at XStart position}
   CurrentPoint := FirstPoint;
   X := CurrentPoint^.Xvalue;
   Y := CurrentPoint^.Yvalue;
   while (X < Xstart) do begin
      X := CurrentPoint^.Xvalue;
      Y := CurrentPoint^.Yvalue;
      CurrentPoint := CurrentPoint^.NextPoint;
   end;
   DrawLine (X, Min-(Max-Min)*0.05, X, Y);
   GotoXY (25,24);
   Write ('X: ', X:12, '   Y: ', Y:12);

   PositionCounter := 1;

   repeat
      FuncKey := 1;
      if keypressed then GetFuncKey (FuncKey);

      if (FuncKey = 75) and (CurrentPoint <> FirstPoint) then begin
         SetColorBlack;
         DrawLine (X, Min-(Max-Min)*0.05, X, Y);
         SetColorWhite;
         DrawPoint (X, Y);

       {Go to the begining -- PositionCounter equals 1}
         CurrentPoint := FirstPoint;
         X := CurrentPoint^.Xvalue;
         Y := CurrentPoint^.Yvalue;
         while (X < Xstart) do begin
            X := CurrentPoint^.Xvalue;
            Y := CurrentPoint^.Yvalue;
            CurrentPoint := CurrentPoint^.NextPoint;
         end;

       {Advance upto one point behind the current position}
         PositionCounter := PositionCounter - 1;
         for counter := 2 to PositionCounter do
            CurrentPoint := CurrentPoint^.NextPoint;

         X := CurrentPoint^.Xvalue;
         Y := CurrentPoint^.Yvalue;
         if (X < XStart) then begin
            CurrentPoint := CurrentPoint^.NextPoint;
            X := CurrentPoint^.Xvalue;
            Y := CurrentPoint^.Yvalue;
            PositionCounter := PositionCounter + 1;
         end;

         DrawLine (X, Min-(Max-Min)*0.05, X, Y);
         GotoXY (25,24);
         Write ('X: ', X:12, '   Y: ',Y:12);
```

```
      end;

      if (((FuncKey = 77) and (CurrentPoint^.NextPoint <> nil))
       and (CurrentPoint^.NextPoint^.Xvalue <= XEnd)) then begin
         if (CurrentPoint <> FirstPoint) then begin
            SetColorBlack;
            DrawLine (X, Min-(Max-Min)*0.05, X, Y);
         end;
         SetColorWhite;
         DrawPoint (X, Y);

         CurrentPoint := CurrentPoint^.NextPoint;
         X := CurrentPoint^.Xvalue;
         Y := CurrentPoint^.Yvalue;
         PositionCounter := PositionCounter + 1;

         DrawLine (X, Min-(Max-Min)*0.05, X, Y);
         GotoXY (25,24);
         Write ('X: ', X:12, '   Y: ', Y:12);
      end;

    until (FuncKey = 68);
    SetColorBlack;
    DrawPoint(X,Y);
    DrawLine (X, Min-(Max-Min)*0.05, X, Y);
    SetColorWhite;
end;


PROCEDURE EraseRunData (var RunData : RunDataRecord);

var
   CurrentPoint : DataPointer;

begin

   while (RunData.IonEFirstPoint <> nil) do begin
      CurrentPoint := RunData.IonEFirstPoint;
      RunData.IonEFirstPoint := CurrentPoint^.NextPoint;
      dispose (CurrentPoint);
   end;
   RunData.IonELastPoint := nil;
   RunData.IonELastPoint^.NextPoint := nil;

end;


PROCEDURE EraseData (var FirstPoint : DataPointer;
                     var LastPoint  : DataPointer);

var
   CurrentPoint : DataPointer;

begin
   while (FirstPoint <> nil) do begin
      CurrentPoint := FirstPoint;
      FirstPoint := CurrentPoint^.NextPoint;
      dispose (CurrentPoint);
   end;
   LastPoint := nil;
   LastPoint^.NextPoint := nil;
end;


BEGIN
   with Status do begin
      IsIonESpectra  := false;
      IsVWaveSpectra := false;
      IsIWaveSpectra := false;
      IsOWaveSpectra := false;
      Initial        := 'J';
      FileName       := '';
   end;

   with SysParam do begin
      Pressure      := 0.0;
      Frequency     := 13.56;
      ElecSep       := 0.0;
      UpElecDiam    := 3.0;
      LowElecDiam   := 3.0;

      Gas1Name         := 'Ar';
      Gas1MeterReading := 0.0;
      Gas1Flow         := 0.0;

      Description1 := '     ';
      Description2 := '     ';
```

```
      Description3 := '     ';
   end;

   with IonEParam do begin
      Grid1 := 0.0;
      Grid3 := -50.0;

      StartEv        := 0.0;
      EndEv          := 100.0;
      StepEv         := 1.0;
      NumberSteps    := 1.0;
      PointsPerStep  := 1;
      Max            := 0.0;
      Min            := 0.0;
   end;

   with RunData do begin
      IonEFirstPoint := nil;
      IonELastPoint  := nil;
      IonELastPoint^.NextPoint := nil;
   end;

   with DistributionData do begin
      DistFirstPoint := nil;
      DistLastPoint  := nil;
      DistLastPoint^.NextPoint := nil;
   end;

   with DistributionParam do begin
      max := 0.0;
      min := 0.0;
   end;

   with WfmDataRec.PowerParam do begin
      CurrentMultiplier := 1.0;
      VoltageMultiplier := 1109.9;      {at 13.56MHz}
   end;

   IFirstPoint  := nil;
   ILastPoint   := nil;
   ILastPoint^.NextPoint := nil;
   VFirstPoint  := nil;
   VLastPoint   := nil;
   VLastPoint^.NextPoint := nil;
   OFirstPoint  := nil;
   OLastPoint   := nil;
   OLastPoint^.NextPoint := nil;

   With OpticalData do begin
      With SpatialScan do begin
         StartLocation      := 0.0;
         EndLocation        := 3.0;
         cmBetweenData      := 0.05;
         ScanWavelength     := 0.0;
         ReadingsPerPt      := 10;
      end;

      With WavelengthScan do begin
         StartWave          := 0.0;
         EndWave            := 0.0;
         StartWaveMeter     := 0.0;
         EndWaveMeter       := 0.0;
         WaveIncrement      := 0.0;
         DetectionLocation  := 0.0;
         ReadingsPerPt      := 10;
      end;

      With PIEParam do begin
         IsSpatialScan      := false;
         IsWavelengthScan   := false;
         GratingGrooveDensity := 1800;
         StepsPerAngstrom   := 150;
      end;

      SpaFirstPoint := nil;
      SpaLastPoint  := nil;
      SpaLastPoint^.NextPoint := nil;

      WavFirstPoint := nil;
      WavLastPoint  := nil;
      WavLastPoint^.NextPoint := nil;
   end;

   CurrWavelength := 7500.0;
END.
```

Unit FTools;

```
INTERFACE
uses DOS, Crt, GDriver, GKernel, GWindow, GShell,
     ATools;


PROCEDURE SaveRun (SysParam : SysRecord;
                   Status   : StatusRecord;
                    PowerParam   : PowerRecor');

PROCEDURE SaveIonE (RunData          : RunDataRecord;
                    DistributionData : DistDataRecord;
                    IonEParam        : IonERecord;
                    Status           : StatusRecord);

PROCEDURE SaveVWave (VFirstPoint : datapointer;
                     Status      : StatusRecord);

PROCEDURE SaveIWave (IFirstPoint : datapointer;
                     Status      : StatusRecord);

PROCEDURE FileNameGen (Initial :str8; Date : DateRecord; Direc:integer;
                       var FileName:str7);

PROCEDURE GetFileName (var Status:StatusRecord; var Date:DateRecord;Direc:integer);

PROCEDURE SavePIESystem (PIEParam:PIERecord;
                         SpatialScan:SpatialScanRecord;
                         WavelengthScan:WavelengthScanRecord;
                         FileName:str7);

PROCEDURE SaveSpatialScan (SpaFirstPoint:DataPointer; FileName:str7);

PROCEDURE SaveWavelengthScan (WavFirstPoint:DataPointer; FileName:str7);

PROCEDURE SavePIEData (PIEParam       : PIERecord;
                       SpatialScan    : SpatialScanRecord;
                       WavelengthScan : WavelengthScanRecord;
                       SpaFirstPoint  : DataPointer;
                       WavFirstPoint  : DataPointer);


IMPLEMENTATION
PROCEDURE SaveRun (SysParam   : SysRecord;
                   Status     : StatusRecord;
                    PowerParam : PowerRecord);

type
   ParameterRecord = record
      Sys    : SysRecord;
      Stat   : StatusRecord;
      Pow    : PowerRecord;
   end;

var
   Parameters : ParameterRecord;
   ParameterFile : File of ParameterRecord;

begin
   GotoXY (5,24);
   Write ('Saving System Parameters');

   With Parameters do begin
      Sys    := SysParam;
      Stat   := Status;
      Pow    := PowerParam;
   end;

   assign (ParameterFile, Path+'INFO\'+Status.FileName+'.SYS');
   rewrite (ParameterFile);
   write (ParameterFile, Parameters);
   close (ParameterFile);
   GotoXY (5,24);
   Write ('                              ');
end;


PROCEDURE SaveIonE (RunData          : RunDataRecord;
                    DistributionData : DistDataRecord;
                    IonEParam        : IonERecord;
                    Status           : StatusRecord);

var
   DataFile     : Text;
   IonEFile     : File of IonERecord;
```

```
begin
   GotoXY (5,24);
   Write ('Saving Ion Energy Data');

   assign (IonEFile, Path+'IE\'+Status.FileName+'.INF');
   rewrite (IonEFile);
   write (IonEFile, IonEParam);
   close (IonEFile);

   assign (DataFile, Path+'IE\'+Status.FileName+'.IE');
   rewrite (DataFile);
   with RunData, DistributionData do begin
      IonELastPoint := IonEFirstPoint;
      DistLastPoint := DistFirstPoint;

      {save ion energy data in three columns:
        Column 1 = Ion energy (grid bias) - eV
        Column 2 = Current measured at that grid bias - Amp
        Column 3 = Differentiated current - Amp}

      while (IonELastPoint <> nil) do begin
         With IonELastPoint^ do
            writeln (DataFile,Xvalue:8:2,'  ',Yvalue:13,'  ',DistLastPoint^.Yvalue:13);
         IonELastPoint := IonELastPoint^.NextPoint;
         DistLastPoint := DistLastPoint^.NextPoint;
      end;
   end;
   close (Datafile);

   GotoXY (5,24);
   Write ('                          ');
end;

PROCEDURE SaveVWave (VFirstPoint : datapointer;
                     Status      : StatusRecord);

{This procedure saves the voltage waveform.  The data has been converted
 from bytes as received from the LeCroy 9400 to voltage and time.  The
 number of points saved are the same as the current waveform.}

type
   DataRecord = record
      Xvalue : real;
      Yvalue : real;
   end;

var
   Data         : DataRecord;
   DataFile     : File of DataRecord;
   CurrentPoint : datapointer;

begin
   GotoXY(5,24);  Write ('Saving voltage waveform...');

   Assign (DataFile, Path+'WAVEFORM\'+Status.FileName+'.VWF');
   ReWrite (DataFile);

   CurrentPoint := VFirstPoint;
   While (CurrentPoint <> nil) do begin
      With Data do begin
         Xvalue := CurrentPoint^.Xvalue;
         Yvalue := CurrentPoint^.Yvalue;
      end;
      Write (DataFile, Data);
      CurrentPoint := CurrentPoint^.NextPoint;
   end;

   Close (DataFile);
end;


PROCEDURE SaveIWave (IFirstPoint : datapointer;
                     Status      : StatusRecord);

{This procedure saves the current waveform.  The data has been converted
 from bytes as received from the LeCroy 9400 to voltage and time.  The
 number of points saved are the same as the voltage waveform.}

type
   DataRecord = record
      Xvalue : real;
      Yvalue : real;
   end;

var
   Data         : DataRecord;
   DataFile     : File of DataRecord;
   CurrentPoint : datapointer;
```

```
begin
   GotoXY(5,24);  Write ('Saving current waveform...');

   Assign (DataFile, Path+'WAVEFORM\'+Status.FileName+'.IWF');
   ReWrite (DataFile);

   CurrentPoint := IFirstPoint;
   While (CurrentPoint <> nil) do begin
      With Data do begin
         Xvalue := CurrentPoint^.Xvalue;
         Yvalue := CurrentPoint^.Yvalue;
      end;
      Write (DataFile, Data);
      CurrentPoint := CurrentPoint^.NextPoint;
   end;

   Close (DataFile);
end;


PROCEDURE FileNameGen (    Initial : str8; Date : DateRecord; Direc:integer;
                       var FileName:str7);
                             {The procedure determines the first 7
                              characters of the file name following a
                              designated code: Initial (1 char.),
                              Year (1 char.), Month (2 char.), Day (2 char.),
                              and Run number (1 char).}

var
   MonthCh, DayCh, YearCh : string[2];
   counter1Ch : char;
   counter1 : integer;
   DirInfo : SearchRec;
   TempFileName : str7;

begin
   With Date do begin
      MonthCh := Month;
      if (Length(MonthCh) = 1) then MonthCh := '0'+MonthCh;
      DayCh := Day;
      if (Length(DayCh) = 1) then DayCh := '0'+DayCh;
      YearCh  := Year;
   end;
   FileName := Initial+YearCh+MonthCh+DayCh;

   counter1 := 64;                {find latest file on disk}
   repeat
      counter1 := counter1 + 1;
      if (counter1 = 91) then begin
         GotoXY (2,25);
         Write ('Run Sequence FULL -- Restart with new initial          ');
         EXIT;
      end;
      counter1Ch := chr(counter1);
      TempFileName := FileName + counter1Ch;
      If Direc = 1 then FindFirst (Path1+TempFileName+'.*', AnyFile, DirInfo)
      else FindFirst (Path2+TempFileName+'.*', AnyFile, DirInfo);
   until (DosError <> 0);

   FileName := TempFileName;      {assign latest file to FileName}

end;

Procedure GetFileName (var Status:StatusRecord; var Date:DateRecord;Direc:integer);

 {Obtain the initial for file name generation and generate the file name}

begin
   Status.Initial := 'J';
   Date.Year      := '0';
   Date.Month     := '0';
   Date.Day       := '0';
   GotoXY (5,17);
   Write ('Input last initial for File Name generation: ',Status.Initial);
   GotoXY (5,18);  Write ('What is the year, 199_? (0,1):', Date.Year);
   GotoXY (5,19);  Write ('What is the month? (1..12):', Date.Month);
   GotoXY (5,20);  Write ('What is the day? (1..31):',Date.Day);

   repeat
      ClearCommLine (22);
      ClearCommLine (24);
      GotoXY (3,17);  Write ('*');
      GetString (Status.Initial, Alfa, 50, 17);
      GotoXY (50,17);  Write (Status.Initial);  GotoXY (3,17);  Write (' ');

      With Date do begin
         GotoXY (3,18);  Write ('*');
         GetString (Year, Numeric, 35, 18);
```

```
         GotoXY (35,18);  Write (Year);  GotoXY (3,18);  Write (' ');

         GotoXY (3,19);  Write ('*');
         GetString (Month, Numeric, 32, 19);
         GotoXY (32,19);  Write (Month);  GotoXY (3,19);  Write (' ');

         GotoXY (3,20);  Write ('*');
         GetString (Day, Numeric, 30, 20);
         GotoXY (30,20);  Write (Day);  GotoXY (3,20);  Write (' ');

         FileNameGen (Status.Initial, Date, Direc, Status.FileName);
      end;

      GotoXY (5,22);  Write ('Filename is: ',Status.FileName);

      GotoXY(5,24);  Write ('* Everything correct? (y/n): ');

   until (UpCase(readkey) = 'Y');
end;


Procedure SavePIESystem (PIEParam:PIERecord;
                         SpatialScan:SpatialScanRecord;
                         WavelengthScan:WavelengthScanRecord;
                         FileName:str7);

Type
   ParameterRecord = record
      PIEPar     : PIERecord;
      Spa        : SpatialScanRecord;
      Wav        : WavelengthScanRecord;
   end;

var
   Parameters    : ParameterRecord;
   ParameterFile : File of ParameterRecord;

begin
   GotoXY(5,24);  Write ('Saving System Parameters...                ');

   With Parameters do begin
      PIEPar := PIEParam;
      Spa    := SpatialScan;
      Wav    := WavelengthScan;
   end;

   Assign (ParameterFile, Path+'PIE\'+FileName+'.PIE');
   ReWrite (ParameterFile);
   Write (ParameterFile, Parameters);
   Close (ParameterFile);
end;

Procedure SaveSpatialScan (SpaFirstPoint:DataPointer; FileName:str7);

Var
   DataFile       : Text;
   CurrentPoint   : DataPointer;

begin
   GotoXY (5,24);  Write ('Saving Spatial Scan Data...               ');
   Assign (DataFile, Path+'PIE\'+FileName+'.SPA');
   ReWrite (DataFile);

   CurrentPoint := SpaFirstPoint;
   While (CurrentPoint <> nil) do begin
      With CurrentPoint^ do Writeln (DataFile, Xvalue, '   ', Yvalue);
      CurrentPoint := CurrentPoint^.NextPoint;
   end;

   Close (DataFile);
end;

Procedure SaveWavelengthScan (WavFirstPoint:DataPointer; FileName:str7);

Var
   DataFile       : Text;
   CurrentPoint   : DataPointer;

begin
   GotoXY (5,24);  Write ('Saving Wavelength Scan Data...            ');
   Assign (DataFile, Path+'PIE\'+FileName+'.WAV');
   ReWrite (DataFile);

   CurrentPoint := WavFirstPoint;
   While (CurrentPoint <> nil) do begin
      With CurrentPoint^ do Writeln (DataFile, Xvalue:10:2, '   ', Yvalue:15);
      CurrentPoint :  CurrentPoint^.NextPoint;
   end;
```

```
   Close (DataFile);
end;


Procedure SavePIEData (PIEParam       : PIERecord;
                       SpatialScan    : SpatialScanRecord;
                       WavelengthScan : WavelengthScanRecord;
                       SpaFirstPoint  : DataPointer;
                       WavFirstPoint  : DataPointer);

var
   Reply     : char;

Begin
   SavePIESystem (PIEParam,SpatialScan,WavelengthScan,Status.FileName);
   If PIEParam.IsSpatialScan then
      SaveSpatialScan (SpaFirstPoint,Status.FileName);
   If PIEParam.IsWavelengthScan then
      SaveWavelengthScan (WavFirstPoint,Status.FileName);
end;
END.
```

Documentation for MASSAGE.PAS (April 1991)

This program takes data stored by ACQ.PAS and transform them into useful values. Results from this program are stored in *.SUM files. The experimental runs that are to be processed by MASSAGE.PAS must be listed in a file, e.g. E:\JOANNE\WAVE\WAVE.LST.

All the program files are as follows:

MASSAGE.PAS - the main program
ELECTRIC.PAS - Fourier analysis of waveforms and de-embeds the plasma voltage and current.
CMPLX.PAS - used by ELECTRIC.PAS for complex number arithmatics
DEFINE.PAS - contains the variable definition used by the program.

* current and voltage waveforms (from *.IWF and *.VWF) are Fourier analyzed to find the harmonics of the waveform. The first harmonic of each waveform is then used to calculate the actual amplitudes and phases of current and voltage by decoupling the stray impedance. The stray impedance network values are stored in ELECTRIC.PAS unit of MASSAGE.PAS. These network values are computed for each reactor using UNTRMNT.PAS and is done separately from this program. The directory and name of the network values need to be specified. The input files for the waveform are record files stored in the format specified in ACQ.PAS. This program calculates voltage and current amplitudes, power, electron density if the gas is specified as SF6 or Ar, driving frequency, harmonics of the measured waveforms, plasma capacitance and resistance and sheath voltage.

* experimental information files (*.SYS) are used to check what kind of data were also taken and used to print out experimental conditions and comments in the output from this program.

* to write the average ion energy if the data was taken during the experimental run, this program reads the value from *.INF files which contain the average ion energy calculated during data acquisition. *.SYS files are used to first see if ion energy data was taken before looking for the *.INF files.

* this program calculates the average ion angle and ion flux using *.ANG. If these files do not exist, the program will simply skip the ion angle and ion flux calculations. These files are entered separately from the experiment and should have this format:

line 1: thickness of the orifice in inches, e.g. 0.00035
line 2: diameter of the orifice in inches, e.g. 0.003
line 3: number of orifices e.g. 3
line 4-12: current (namps) at each angle starting at 0 degrees and upward.
If total current is 0, the average ion angle will not be calculated.
line 13: total current (namps) in analyzer

the program takes into account 4 grids of 90.25% transmission and does an area correction for the dimensions of the orifice.

* this program checks to see if optical scans exist using *.SYS files. If they do, information about these scans are read from files *.PIE. For wavelength scans, the start and ending wavelengths are recorded. For spatial scans, the wavelength of the scan, the distance and step interval of the scan are recorded.

The input files and the default directories for these files are as follows. The default directory for these files can be changed during the start of the program. To permanently change the default directories, DEFINE.PAS must be altered. Only files marked by an # are needed.

INPUT FILE NAMES	DEFAULT DIRECTORY	MUST EXIST FOR PROGRAM TO WORK
*.NET	E:\JOANNE\WAVE\SUMMARY\	#
WAVE.LST	E:\JOANNE\WAVE\	#
*.IWF	E:	
*.VWF	E:	
*.SYS	E:	#
*.INF	E:	
*.ANG	E:	
*.PIE	E:	
*.SUM	E:\JOANNE\PARAMS\	

```
PROGRAM MASSAGE;

USES electric, Define, dos, Crt;

TYPE
   Str7          = string[7];
   Str8          = string[8];

VAR
   FileFound     : boolean;
   Terminate     : boolean;
   ReadFile      : str8;                  {file containing waveform filenames}
   DirInfo       : SearchRec;
   Reply         : char;
   WaveFiles     : text;
   FileName      : str7;                  {current waveform data filename}
   PlotDo        : boolean;               {Create file for grapher?}
   IsWaveForms   : boolean;             {indicates waveforms exist}
   TextFile      : text;
   InfoFile      : File of ParameterRecord;
   Parameters    : ParameterRecord;
   String3       : String[3];
   Radius        : real;
   ImpedanceFile: string[8];
   BeepFreq      : integer;


PROCEDURE WriteElectResults (AvgElectResults : ElectricRec;
                             FileName        : Str7);

Var
   TextFile      : text;          {File to write electrical results to}

Begin
   Assign (TextFile, Path3+FileName+'.SUM');
   Append (TextFile);
   With AvgElectResults do begin
      Writeln (TextFile, 'Frequency (MHz) = ', Frequency/1E6:13:5);
      Writeln (TextFile, 'Number of data points used in Fourier Analysis = ', NumOfDataPoints);
      Writeln (TextFile);
      Writeln (TextFile, '                              Magnitude       Angle(rad)  ');
      Writeln (TextFile, ' V (V)                ', DeEmbedVolt.M:15, '  ', DeEmbedVolt.A:15);
      Writeln (TextFile, ' I (amp)              ', DeEmbedCurr.M:15, '  ', DeEmbedCurr.A:15);
      Writeln (TextFile, ' I (amp/m^2)          ', CurrArea:15);
      Writeln (TextFile, 'Vpp (V)               ', Vpp:15);
      Writeln (TextFile, 'Ipp (amp)             ', Ipp:15);
      Writeln (TextFile);
      Writeln (TextFile, 'Plasma [Z]            ', ZPlasma:15);
      If PhaseDelay > 3/2*Pi then PhaseDelay := PhaseDelay - 2*Pi;
      Writeln (TextFile, 'Phase Delay (deg)     ', PhaseDelay*180/PI:15);
      Writeln (TextFile, 'Plasma Power (W)      ', PlasmaPower:15);
      Writeln (TextFile, 'Plasma Power (W/m^2)', PlasmaPowerArea:15);
      Writeln (TextFile, 'Plasma Power Unc      ', PlasmaPowerRaw:15);
      Writeln (TextFile);
      Writeln (TextFile, '                                      MODEL 1          MODEL2');
      Writeln (TextFile, 'Sheath Capacitance (F) ', SheathCap1:15, '   ', SheathCap2:15);
      Writeln (TextFile, 'Sheath Voltage (V)     ', SheathVolt1:15, '   ', SheathVolt2:15);
      Writeln (TextFile, 'Plasma Resistance (ohm)', PlasmaRes1:15, '   ', PlasmaRes2:15);
      Writeln (TextFile, '   E/P (Vm/N)          ', EOverP1:15, '   ', EOverP2:15);
      Writeln (TextFile, 'Mobility (cm^2/V-sec)  ', Mobility1:15, '   ', Mobility2:15);
      Writeln (TextFile, 'Electron density (m^-3)', ElectronDen1:15, '   ',ElectronDen2:15);
   end;
   Close (TextFile);
end;


PROCEDURE WriteFourierResults (FourierResults:FourierRec; FileName:Str7);

Var
   TextFile       : Text;
   NumOfWaveForms : integer;
   I, J           : integer;

Begin
   Assign (TextFile, Path3+FileName+'.SUM');
   Append (TextFile);
   Writeln (TextFile);
   Writeln (TextFile);
   Writeln (TextFile, 'FOURIER RESULTS');
      Writeln (TextFile);
      With FourierResults do begin
         Writeln (TextFile, 'Harmonic  Volt.Amplitude  Volt. Angle      Curr.Amplitude Curr. Angle');
         For J := 0 to HarmonicMax do begin
            Write   (TextFile, '   ', J:5, '  ', VoltAmp[J]:13, '  ', VoltAng[J]:13, '  ');
            Writeln (TextFile, '  ', CurrAmp[J]:13, '  ', CurrAng[J]:13);
          end;
      end;
```

```
   Close (TextFile);
end;

PROCEDURE FindEAR (AR:real; var EAR:EARarray);

Var
   angle          : real;
   phi            : real;
   I              : integer;
   tan            : real;     {tan(angle)}

Begin
   I := 1;
   repeat
      angle  := (I*4.5 - 2.25)*Pi/180.0;
      If I = 9 then angle := angle - 0.5*Pi/180.0;  {9th ring is only 3.5 deg wide}
      tan    := sin(angle)/cos(angle);
      phi    := 2*ArcTan(AR*tan/sqrt(4-sqr(AR*tan)));
      EAR[I] := 1 - AR*tan*sqrt(4-sqr(AR*tan))/Pi - (phi-sin(phi))/Pi;
      I := I + 1;
   until (I>9);
end;


PROCEDURE WriteIAD (PathFile:str100;PathText:str100);
var
   DirInfo       : SearchRec;
   AngFile       : text;
   EAR           : EARArray; {effective area ratio at 9 incidence angles}
   Current       : real;     {ion current in namps}
   TotalCurrent  : real;     {ion current in namps}
   AverageAngle  : real;     {average ion incidence angle}
   IonFlux       : real;     {microamps/cm^2}
   AspectRatio   : real;     {thickness over radius ratio of orifice}
   Diameter      : real;     {diameter of orifice in inches}
   NumOrifice    : integer;  {number of orifices}
   I             : integer;  {counter - number of rings}
   TextFile      : text;

Begin
   {See if ion angle data file exists}
   DosError := 0;
   FindFirst (PathFile, AnyFile, DirInfo);
   If DosError = 0 then begin
      Assign(AngFile, PathFile);
      Reset (AngFile);
          {data should be in this order:
                line 1:    thickness of the orifice in inches
                line 2:    diameter of orifice in inches
                line 3:    number of orifices
                line 4-12: current at each angle starting at 0 deg in namps
                line 13:   total current in analyzer in namps}
      Readln (AngFile, AspectRatio);          {thickness}
      Readln (AngFile, Diameter);             {diameter}
      Readln (AngFile, NumOrifice);          {number of orifices}
      AspectRatio := 2*AspectRatio/diameter; {thickness/radius}
      FindEAR (AspectRatio, EAR);
      I := 1;
      TotalCurrent := 0.0;
      AverageAngle := 0.0;
      Repeat
         Readln (AngFile, Current);
         TotalCurrent :=  TotalCurrent + Current/EAR[I];
         If I <> 9 then  {correct for a 3.5 degree wide 9th ring}
            AverageAngle :=  AverageAngle + Current/EAR[I]*(4.5*I-2.25)
         else AverageAngle := AverageAngle + Current/EAR[I]*(4.5*I-2.75);
         I := I + 1;
      until (I > 9);
      Readln (AngFile, Current);
      Close (AngFile);
      IonFlux := Current*4/Pi/Sqr(Diameter*2.54)/1E3/0.66342/NumOrifice;  {microamps/cm^2}
      Assign (TextFile, PathText);                    {0.66342=grid transparency}
      Append (TextFile);
      Writeln (TextFile, 'Ion Flux (microamps/cm^2): ',IonFlux:6:2);
      If TotalCurrent <> 0.0 then begin
         AverageAngle := AverageAngle/TotalCurrent;
         Writeln (TextFile, 'Average Angle (deg)      : ', AverageAngle:4:1);
      end;
      Close (TextFile);
   end;
end;

PROCEDURE WriteParameters (FileName:Str7; Path4:str100; Path5:str100; Path6:str100);

var
   InfoFile   : File of ParameterRecord;
   Parameters : ParameterRecord;
   TextFile   : Text;
```

```
    IonEFile    : File of IonERecord;
    IonEParam   : IonERecord;
    PIEFile     : File of PIEParametersRec;
    PIEParameters    : PIEParametersRec;

begin
   Assign (TextFile, Path3+FileName+'.SUM');
   Append (TextFile);
   Assign (InfoFile, Path4+FileName+'.SYS');
   Reset (InfoFile);
   Read (InfoFile, Parameters);
   Close (InfoFile);
   Writeln (TextFile);
   Writeln (TextFile);
   Writeln (TextFile, 'EXPERIMENTAL PARAMETERS');
   Writeln (TextFile);
   With Parameters.Sys, Parameters.Pow, Parameters.Status do begin
      Writeln (TextFile, 'Gas                          = ', GasName);
      Writeln (TextFile, 'Pressure (mtorr)             = ', Pressure:6:1);
      Writeln (TextFile, 'Electrode Diameter    (in) = ', UpElecDiam:6:2);
      Writeln (TextFile, 'Electrode Separation (cm) = ', ElecSep:6:2);
      Writeln (TextFile, 'Voltage Multiplier           = ', VoltageMultiplier:6:1);
      Writeln (TextFile, 'Current Multiplier           = ', CurrentMultiplier:6:1);
      Writeln (TextFile);
      Writeln (TextFile, 'Comments:');
      Writeln (TextFile, '    - ', Description1);
      Writeln (TextFile, '    - ', Description2);
      Writeln (TextFile, '    - ', Description3);
      Writeln (TextFile);
      If (IsIonESpectra = false) then Writeln (TextFile, 'No Ion Energy Data')
      else begin
         Assign (IonEFile, Path5+FileName+'.INF');
         Reset (IonEFile);
         Read (IonEFile, IonEParam);
         Close (IonEFile);
         Writeln (TextFile, 'Average Ion Energy = ', IonEParam.IonEnergy:6:1, ' eV');
      end;
   end;
   Close (TextFile);  {close before going to IAD section}
   WriteIAD (Path7+Filename+'.ANG',Path3+FileName+'.SUM');
   Assign (TextFile, Path3+FileName+'.SUM');    {open after IAD section}
   Append (TextFile);
   If (Parameters.Status.IsOWaveSpectra = true) then begin
      Writeln (TextFile);
      Assign(PIEFile, Path6+FileName+'.PIE');
      Reset (PIEFile);
      Read (PIEFile, PIEParameters);
      Close (PIEFile);
      With PIEParameters do begin
         If (PIEParam.IsSpatialScan) then
            With SpatialScan do begin
               Writeln(TextFile,'Spatial scan taken at ', ScanWavelength:8:1, ' A');
               Writeln(TextFile,'Scan length: ',EndLocation-StartLocation:4:2,
                       ' cm    Scan interval: ',cmBetweenData:4:2,' cm');
            end;
         If (PIEParam.IsWavelengthScan) then Writeln(TextFile,
         'Wavelength scan taken from ', WavelengthScan.StartWave:8:1,
         ' to ', WavelengthScan.EndWave:8:1, ' A');
      end;
   end else begin
      Writeln (TextFile);
      Writeln (TextFile, 'No optical spectra exist.');
   end;
   Close (TextFile);
end;


FUNCTION RemoveSpaces (InString:str100) : str100;

begin
   while not(InString[1] in MixSet) do
      delete (InString,1,1);
   RemoveSpaces := InString;
end;


PROCEDURE NewPath(var Path:str100);

var
   TempPath : str100;
   Reply    : char;
   Done     : boolean;

begin
   Done := false;
   repeat
      Write('Enter Path: ');
      Readln(TempPath);
```

```
      TempPath := RemoveSpaces(TempPath);
      Writeln('Is ', TempPath, ' the correct directory? ');
      Reply := UpCase(readkey);
      If Reply = 'Y' then Done := true;
   until (Done);
   Path := TempPath;
end;


BEGIN
   FileFound   := false;
   Terminate   := false;
   PlotDo      := false;
   IsWaveForms := false;

   repeat
      ReadFile := 'WAVE.LST';
      Write ('Are waveform names listed in ', ReadFile, '? ');
      Reply := UpCase (readkey);
      Writeln (Reply);
      If Reply <> 'Y' then begin
         repeat
            Write ('Enter File Name:  ');
            Readln (ReadFile);
            Write ('Are waveform names listed in ', ReadFile, '? ');
            Reply := UpCase (readkey);
            Writeln (Reply);
         until (Reply = 'Y');
      end;

      Write ('Is DIRECTORY containing file ', ReadFile, ' ', Path1, '? ');
      Reply := UpCase (readkey);
      Writeln (Reply);
      If Reply <> 'Y' then NewPath(Path1);
      DosError :=0;
      FindFirst (Path1+ReadFile, AnyFile, DirInfo);
      If DosError <> 0 then begin
          Sound (1000);  Delay (1000);  NoSound;
          Write ('Try Again? ');
          Reply := UpCase (readkey);
         Writeln (Reply);
          If not (Reply = 'Y') then Terminate := true;
      end else FileFound := true;
   until (FileFound or Terminate);

   {Check to see if all files containing expt. info. exist before starting}
   If not(Terminate) then begin
      Write ('Is DIRECTORY containing the expt. parameter records ', Path4, '? ');
      Reply := UpCase(readkey);
      Writeln (Reply);
      If Reply <> 'Y' then NewPath(Path4);
      Writeln ('Checking to make sure all expt. parameter records exist......');
      Assign (WaveFiles, Path1+ReadFile);
      Reset (WaveFiles);
      Repeat
         Readln (WaveFiles, FileName);
          Write ('     Searching for File = ', Filename);
          DosError := 0;
         FindFirst (Path4+FileName+'.SYS', AnyFile, DirInfo);
          If (DosError <> 0) then begin
             Writeln;
             Sound (500);  Delay (500);  NoSound;
             Writeln (Path4, FileName, '.SYS  NOT FOUND!!');
             Terminate := true;
          end else Writeln ('  FOUND');
      Until (EoF(WaveFiles));
      Close (WaveFiles);
      If not(Terminate) then Writeln ('     All expt. parameter records have been found.')
      else Writeln ('Done checking expt. parameter files.  Remove missing files');
      Writeln;
   end;

   {Check to see if all waveform data files exist before starting}
   If not(Terminate) then begin
      Write ('If waveform records exist, are they in DIRECTORY ', Path2, '? ');
      Reply := UpCase(readkey);
      Writeln (Reply);
      If Reply <> 'Y' then NewPath(Path2);
      Writeln ('Checking to make sure all waveform records exist.....');
      Assign (WaveFiles, Path1+ReadFile);
      Reset (WaveFiles);
      Repeat
         Readln (WaveFiles, FileName);
         Assign (InfoFile, Path4+FileName+'.SYS');
         Reset (InfoFile);
         Read (InfoFile, Parameters);
         Close (InfoFile);
         If Parameters.Status.IsVWaveSpectra then begin
```

```
            IsWaveForms := true;
            Write ('     Searching for File = ', Filename);
            DosError := 0;
            FindFirst (Path2+FileName+'.IWF', AnyFile, DirInfo);
            If (DosError <> 0) then begin
               Writeln;
               Sound (500);  Delay (500);  NoSound;
               Writeln (Path2, FileName, '.IWF  NOT FOUND!!');
               Terminate := true;
            end;
            DosError := 0;
            FindFirst (Path2+FileName+'.VWF', AnyFile, DirInfo);
            If (DosError <> 0) then begin
               Sound (500);  Delay (500);  NoSound;
               Writeln (Path2, FileName, '.VWF  NOT FOUND!!');
               Terminate := true;
            end else Writeln ('    FOUND');
         end;
      Until (EoF(WaveFiles));
      Close (WaveFiles);
      If not(Terminate) then Writeln ('     All necessary waveform records have been found.')
      else Writeln ('Done checking waveform files.  Remove missing files');
      Writeln;
   end;

   {Find directory containing the stray impedances}
   If not(Terminate) and IsWaveForms then begin
      Write ('Are the stray impedances stored in', Path8, '? ');
      Reply := UpCase (readkey);
      Writeln (reply);
      If Reply <> 'Y' then NewPath (Path8);
      ImpedanceFile := 'AL3.NET';
      Write ('Are stray impedances stored in file ',ImpedanceFile,'? ');
      Reply := UpCase (readkey);
      Writeln (Reply);
      If Reply <> 'Y' then begin
         repeat
            Write ('Enter file name: ');
            Readln (ImpedanceFile);
            Write ('Are stray impedances stored in file ', ImpedanceFile,'? ');
            Reply := UpCase (readkey);
            Writeln (Reply);
         until (Reply = 'Y');
      end;
      FindFirst (Path8+ImpedanceFile, AnyFile, DirInfo);
      If DosError <> 0 then begin
         Sound (500);  Delay (200);  NoSound;
         Writeln ('     File containing stray impedances not found!');
         Terminate := true;
      end else begin
         Path8 := Path8+ImpedanceFile;
         {check to see if electrode capacitance should be subtracted}
         String3 := ImpedanceFile;
         If String3 = 'NEW' then ElecCorr.Correct := true;
         Writeln ('     De-embedding file found.');
      end;
   end;

   {Find directory containing ion energy data}
   If not(Terminate) then begin
      Write ('If ion energy data exists, are they in DIRECTORY ', Path5, '? ');
      Reply := UpCase (readkey);
      Writeln (reply);
      If Reply <> 'Y' then NewPath (Path5);
   end;

   {Check to see in ion energy data exist}
   If not (Terminate) then begin
      Assign (WaveFiles, Path1+ReadFile);
      Reset (WaveFiles);
      Writeln ('Checking to make sure all ion energy records exist....');
      Repeat
         Readln (WaveFiles, FileName);
         Assign (InfoFile, Path4+FileName+'.SYS');
         Reset (InfoFile);
         Read (InfoFile, Parameters);
         Close (InfoFile);
         If Parameters.Status.IsIonESpectra = true then begin
            Write ('     Searching for File = ', Filename, '.INF');
            FindFirst (Path5+FileName+'.INF', AnyFile, DirInfo);
            If DosError <> 0 then begin
               Sound (500);  Delay (500);  NoSound;
               Writeln ('   NOT FOUND!!');
               Terminate := true;
            end else Writeln ('   FOUND');
         end;
      until (EoF(WaveFiles));
      Close (WaveFiles);
```

```
      If not(Terminate) then Writeln ('     All necessary ion energy records have been found.')
      else Writeln ('     Done checking ion energy files.  Remove missing files.');
      Writeln;
   end;

   {Find directory containing ion angle data:}
   {Default directory=  E:\JOANNE\WAVE\SUMMARY\}
   If not(Terminate) then begin
      write ('If ion angle data exits, is it in DIRECTORY ', Path7, '? ');
      Reply := UpCase (readkey);
      Writeln (reply);
      if reply <> 'Y' then NewPath (Path7);
   end;

   {Find directory containing optical scan if that exists}
   If not(Terminate) then begin
      write ('If optical data exists, are they in DIRECTORY ', Path6, '? ');
      Reply := UpCase (readkey);
      Writeln (reply);
      if reply <> 'Y' then NewPath (Path6);
   end;

   If not(Terminate) then begin
      Write ('Should analysis results be stored in ', Path3, ' directory? ');
      Reply := UpCase (readkey);
      Writeln (Reply);
      If Reply <> 'Y' then NewPath (Path3);
   end;

   {Begin Fourier Analysis and calculate power}
   If not (Terminate) then begin
      Assign (WaveFiles, Path1+ReadFile);
      Reset (WaveFiles);
      Repeat
          Readln (WaveFiles, FileName);
          Writeln ('Processing ', FileName, '..... ');
         Assign (TextFile, Path3+FileName+'.SUM');
         Rewrite (TextFile);
         Writeln (TextFile, 'File = ', FileName);
         Writeln (TextFile);
         Close (TextFile);
         Assign (InfoFile,Path4+FileName+'.SYS');
         Reset (InfoFile);
         Read (InfoFile, Parameters);
         Close (InfoFile);
         If Parameters.Status.IsVWaveSpectra then begin
            If ElecCorr.Correct then With Parameters.Sys do begin
               Radius := (UpElecDiam/2.-0.1)*0.0254;
               ElecCorr.ElecImped.I :=-ElecSep/100./2./pi/13.56E6/
                  8.85E-12/pi/Radius/Radius;
            end;
            Electricalc (FileName, FourierResults, ElectricResults,
                         ElecCorr, PlotDo);
            WriteElectResults (ElectricResults, FileName);
            WriteFourierResults (FourierResults, FileName);
         end;
         WriteParameters (FileName, Path4, Path5, Path6);
         If keypressed then begin
            If readkey = chr(27) then Terminate := true;
         end;
         If (EoF(WaveFiles)) then Terminate := true;
         Writeln;
      Until (Terminate);
      Close (WaveFiles);
   end;
   Writeln;
   BeepFreq := 500;
   repeat
      Sound (BeepFreq);  Delay (500);
      BeepFreq := BeepFreq + 100;
   until (BeepFreq > 1000);
   NoSound;
   Writeln ('DONE!!!!!!!!!');

END.
```

Unit Define;

```
INTERFACE

CONST
   FourierSize    = 1023;
   HarmonicMax  = 5;

   Permittivity = 8.854188E-12;
   ElectronCharge = 1.6022E-19;

   MixSet : set of Char = ['A'..'Z', 'a'..'z', '0'..'9', '.','\',':'];


TYPE
   Str100        = string[100];
   Str60         = string[60];
   Str8          = string[8];
   Str7          = string[7];

   EARarray      = array[1..9] of real;

   Complex        = Record
      R, I        : double;
   end;

   Polar          = Record
      M, A        : Double;
   end;

   StatusRecord          = record
      IsIonESpectra      : boolean;
      IsVWaveSpectra     : boolean;
      IsIWaveSpectra     : boolean;
      IsOWaveSpectra     : boolean;
      Initial            : str8;
      FileName           : str7;
   end;

   SysRecord        = record
      Pressure      : real;
      Frequency     : real;
      ElecSep       : real;
      UpElecDiam    : real;
      LowElecDiam   : real;
      GasName       : str8;
      GasMeterReading : real;
      GasFlow         : real;
      Description1    : str60;
      Description2    : str60;
      Description3    : str60;
   end;

   PowerRecord           = record
      CurrentMultiplier : real;
      VoltageMultiplier : real;
      AvePower          : double;
   end;

   ParameterRecord       = record
      Sys                : SysRecord;
      Status             : StatusRecord;
      Pow                : PowerRecord;
   end;

   IonERecord            = record
      Grid1              : real;
      Grid3              : real;
      StartEv            : real;
      EndEv              : real;
      StepEv             : real;
      NumberSteps        : real;
      PointsPerStep      : integer;
      Max                : double;
      Min                : double;
      IonEnergy          : real;
   end;

   MotorRecord = record
      TotalSteps : Longint;
      Length     : real;
      HighMotor  : byte;
      LowMotor   : byte;
      Drive, Direction : char;   {color of drive/ extend or retract}
      Motor      : integer;      {1=XY motors or 2=Z motor}
   end;

   SpatialScanRecord     = record
```

```
      StartLocation       : real;
      EndLocation         : real;
      cmBetweenData       : real;
      ScanWavelength      : real;
      Max, Min            : real;
      ReadingsPerPt       : integer;
      MotorSpecs          : MotorRecord;
   end;

   WavelengthScanRecord   = record
      StartWave           : real;
      EndWave             : real;
      StartWaveMeter      : real;
      EndWaveMeter        : real;
      WaveIncrement       : real;
      DetectionLocation   : real;
      Max, Min            : real;
      ReadingsPerPt       : integer;
   end;

   PIErecord              = record
      IsSpatialScan       : boolean;
      IsWavelengthScan    : boolean;
      GratingGrooveDensity: integer;
      StepsPerAngstrom    : integer;
   end;

   PIEParametersRec       = record
      PIEParam            : PIERecord;
      SpatialScan         : SpatialScanRecord;
      WavelengthScan      : WavelengthScanRecord;
   end;

   TnVector       = Array[0..FourierSize] of Double;
   TnVectorPtr    = ^TnVector;

   Harmonics      = 0..HarmonicMax;
   HArray         = Array[Harmonics] of double;
   FourierRec     = Record
      VoltAmp     : HArray;
      CurrAmp     : HArray;
      VoltAng     : HArray;
      CurrAng     : HArray;
   end;

   ELECTRICREC  = Record
         NumOfDataPoints: integer;  {number of data points in cycles used}
         FREQUENCY      : DOUBLE;
          DEEMBEDVOLT    : POLAR;
          DEEMBEDCURR    : POLAR;
         CurrArea       : Double;
         Vpp            : Double;
         Ipp            : Double;
         ZPLASMA        : DOUBLE;
         PHASEDELAY     : DOUBLE;
         PLASMAPOWER    : DOUBLE;
         PlasmaPowerArea: Double;
          PLASMAPOWERRaw : DOUBLE;
         PLASMARES1     : DOUBLE;
         SHEATHCAP1     : DOUBLE;
         SHEATHVOLT1    : DOUBLE;
         EOVERP1        : DOUBLE;
         Mobility1      : Double;
         ELECTRONDEN1   : DOUBLE;
         PLASMARES2     : DOUBLE;
          ITERATIONS     : INTEGER;
         SHEATHCAP2     : DOUBLE;
         SHEATHVOLT2    : DOUBLE;
         EOVERP2        : DOUBLE;
         Mobility2      : Double;
         ELECTRONDEN2   : DOUBLE;
   END;

   ELECCORRrec    = Record
      Correct     : boolean;
      ElecImped   : complex;
   end;

VAR
   FourierResults         : FourierRec;
   ElectricResults        : ElectricRec;
   ElecCorr               : ElecCorrRec;
   Path1                  : str100;
   Path2                  : str100;
   Path3                  : str100;
   Path4                  : str100;
   Path5                  : str100;
```

```
    Path6                : str100;
    Path7                : str100;
    Path8                : str100;

IMPLEMENTATION

BEGIN
    Path1 := 'E:\Joanne\Wave\';   {Directory containing list of files}
                                        {to be processed}
    Path2 := 'E:';                {Directory to find waveform data}

    Path3 := 'E:\Joanne\params\'; {Directory for storing waveform}
                                        {analysis results}

    Path4 := 'E:';                {Directory containing information on}
                                       {experimental parameters}

    Path5 := 'E:';                {Directory containing ion energy info}

    Path6 := 'E:';                {Directory containing optical scan data}

    Path7 := 'E:';                {Directory containing ion angle data}

    Path8 := 'E:\JOANNE\WAVE\SUMMARY\';  {Directory containing the stray impedances}

    With ElecCorr do begin
       Correct := false;
       ElecImped.R := 0.0;
       ElecImped.I := 1000000000000.0;
    end;

END.
```

```
Unit Electric;
{
 This unit contains the procedure ELECTRICALC used by MASSAGE

 Revisions:
 5/20/88 - BUTTERBAUGH
 3/12/89 - BUTTERBAUGH
 6/7/89  - LIU
}
INTERFACE

USES Crt,Define,FFT87B2,CMPLX;

PROCEDURE ELECTRICALC(FILENAME            : string;
                      VAR FOURIERRESULTS  : FOURIERrec;
                      VAR ELECTRICRESULTS : ELECTRICrec;
                      ELECCORR            : ELECCORRrec;
                      PLOTDO              : BOOLEAN);

IMPLEMENTATION

PROCEDURE ELECTRICALC;
{
 This procedure will perform electrical analysis based on the waveforms
 from the Tektronix oscilloscope.  It will perform fourier analysis on the
 first four cycles of each of the current and voltage waveforms and keep
 track of the first six harmonics (zero thru five).  Impedance corrections
 are made based on stray impedance data from the equipment constants file.
 Finally plasma parameters are calculated for each waveform taken and are
 averaged over all waveforms.  Plot files for GRAPHER are made of each
 wave form if so desired.
}
CONST
   CYCLES          = 2;   { number of cycles from waveform for analysis }

TYPE
   VOLTPTR         = ^VOLTDATA;
   CURRPTR         = ^CURRDATA;
   VOLTDATA        =
      RECORD
         TIME      : DOUBLE;
         VOLTAGE   : DOUBLE;
         NEXTPOINT : VOLTPTR
      END;
   CURRDATA        =
      RECORD
         TIME      : DOUBLE;
         CURRENT   : DOUBLE;
         NEXTPOINT : CURRPTR
      END;
   SIGNS           = (POSITIVE,NEGATIVE);

   DataRecord      = record
      XValue       : real;
      YValue       : real;
   end;


VAR
   TEXTFILE        : TEXT;   { file variable used for info and constants file }
   I,J             : INTEGER; { all purpose counting variables }
   PRESSURE        : DOUBLE;   { reactor pressure during run - read from info }
   SPACING         : DOUBLE;
   NUMOSCPTS       : INTEGER; { number of points in each waveform}
   INTERVAL        : DOUBLE;   { time interval of oscilloscope data from info }
   VOLTZERO        : DOUBLE;   { voltage waveform zero of oscilloscope }
   CURRZERO        : DOUBLE;   { current waveform zero of oscilloscope }
   VOLTMULTIPLIER  : DOUBLE;   { voltage multiplier from oscilloscope }
   CURRMULTIPLIER  : DOUBLE;   { current multiplier from oscilloscope }
   MOBILITYPRESSURE: DOUBLE;   { electron mobility times pressure - gas property }
   Diameter        : Double;   { electrode diameter}
   AREA            : DOUBLE;   { electrode area - const}
   Z11             : COMPLEX; { element of embedding matrix }
   Z22             : COMPLEX; { element of embedding matrix }
   Z12             : COMPLEX; { element of embedding matrix }
   Z21             : COMPLEX; { element of embedding matrix }
   VOLTFIRSTPOINT  : VOLTPTR; { pointer to the first voltage waveform data pt }
   VOLTLASTPOINT   : VOLTPTR; { pointer to the last voltage waveform data pt }
   VOLTCURRENTPOINT: VOLTPTR; { points to current voltage data during analysis}
   CURRFIRSTPOINT  : CURRPTR; { pointer to the first current waveform data pt }
   CURRLASTPOINT   : CURRPTR; { pointer to the last current waveform data pt }
   CURRCURRENTPOINT: CURRPTR; { points to current current data during analysis}
   VOLTMIN         : DOUBLE; { minimum value of the voltage waveform }
   VOLTMAX         : DOUBLE; { maximum value of the voltage waveform }
   VOLTMID         : DOUBLE; { midpoint value between max and min current value }
   CurrMax         : Double; { maximum value of the current waveform}
   CurrMin         : Double; { minimum value of the current waveform}
   CURRSTARTPOINT  : CURRPTR; { first current data point for data expansion }
```

```
    VOLTSTARTPOINT   : VOLTPTR; { first voltage data point for data expansion }
    LASTSIGN,NEWSIGN : SIGNS;  { keeps track of sign changes rel. to currmid }
    EXPANDEDTIME     : DOUBLE; { keeps track of timescale during data expansion }
    REALDATA         : TNVECTORPTR; { data pointer for fourier analysis }
    IMAGDATA         : TNVECTORPTR; { data pointer for fourier analysis }
    ERROR            : BYTE; { error indicator for fourier analysis procedure }
    VIN              : COMPLEX; { measured voltage in complex form }
    IIN              : COMPLEX; { measured current in complex form }
    C1,C2,C3,C4,C5,C6 : COMPLEX; { holders used during de-embedding }
    ILOAD            : COMPLEX;
    VLOAD            : COMPLEX;
    COMPLEXCURRENT   : COMPLEX; { used during stray impedance corrections }
    OMEGA            : DOUBLE; { angular frequency of power input }
    REALPLASMAIMPED  : DOUBLE; { real part of plasma impedance }
    IMAGPLASMAIMPED  : DOUBLE; { imaginary part of plasma impedance }
    VACUUMCAP        : DOUBLE; { capacitance of vacuum between electrodes }
    BULKCAP          : DOUBLE; { sheath corrected electrode capacitance }
    JUMPOUT          : BOOLEAN; { test for ending a repeat loop }
    NEWBULKCAP       : DOUBLE;

    VFile, IFile     : File of DataRecord;
    Data             : DataRecord;
    InfoFile         : File of ParameterRecord;
    Parameters       : ParameterRecord;
    ImpedFile        : text;

Procedure CalculateMobility (var MobilityPressure:double; EOVERP:double;
                             GasName:str8);
const
   NOverRT = 1.930622039E13;

var
   X1,X2,Y1,Y2    : double;
   Slope          : double;

begin
   MobilityPressure := 0.0;
   {ARGON}
   {mobility*pressure data from L.G. Christophorou Vol. 2: ELECTRON-MOLECULE
    INTERACTIONS AND THEIR APPLICATIONS, page 137.
         For 1E-18 < E/N volts-cm^2 < 5E-17
         drift velocity = 202099 + 1.4331E22*(E/N) - 4.75623E38*(E/N)^2
                        + 5.34421E54 (E/N)^3.

         For 5E-17 < E/N volts cm^2 < 1.2E-16
         drift velocity = 1803030 * log [(E/N)/6E-17] + 0.5E6.

    To convert from drift velocity to mobility times pressure, take slope of
    drift velocity vs E/N and divide by Na/RT.
    Na/RT = 1.930622E13 assuming 500K and pressure in mtorr}

    If (GasName = 'Ar') then begin
       X1 := EOVERP/NOverRT*0.9;
       X2 := EOVERP/NOverRT*1.1;
       If (EOVERP/NOverRT <= 5E-17) then begin
          Y1 := 202099+1.4331E22*X1-4.75623E38*X1*X1
                + 5.34421E54*X1*X1*X1;
          Y2 := 202099+1.4331E22*X2-4.75623E38*X2*X2
                + 5.34421E54*X2*X2*X2;
       end else begin
          Y1 := 180303*ln(X1/6E-17)/2.3025851 + 0.5E6;
          Y2 := 180303*ln(X2/6E-17)/2.3025851 + 0.5E6;
       end;
       Slope := (Y1-Y2)/(X1-X2);
       MobilityPressure := Slope/NOverRT;
    end;

    {SF6}
    {Data also from Christophorou, p. 137}
    If (Gasname = 'SF6') then MobilityPressure := 3.2373E8;
end;


BEGIN { PROCEDURE ELECTRICALC }
   WRITELN('PROGRAM IN ELECTRICALC:');

   {Read from file containing the stray impedances (output from UNTRMNT.PAS)}
   Assign (ImpedFile, Path8);
   Reset (ImpedFile);
   Readln (ImpedFile, Z11.R, Z11.I);
   Readln (ImpedFile, Z22.R, Z22.I);
   Readln (ImpedFile, Z21.R, Z21.I);
   Close (ImpedFile);
   {In a reactor with a powered shield Z11 = Z22}
   Z12 := Z22;

   Assign (InfoFile, Path4+FileName+'.SYS');
   Reset (InfoFile);
```

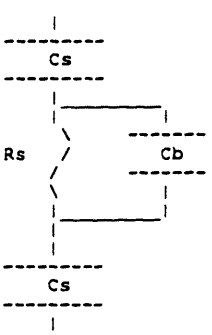
```
Read (InfoFile, Parameters);
Close (InfoFile);

With Parameters do begin
   {Electrode diameter}
   Diameter := Sys.UpElecDiam * 0.0254; {meter}
   Area     := Diameter*Diameter/4*PI;

   {Electrode spacing}
   Spacing  := Sys.ElecSep;  {3 meter}

   {Set voltage and current multipliers}
   VoltMultiplier := Pow.VoltageMultiplier;
   CurrMultiplier := Pow.CurrentMultiplier;
end;

Pressure := Parameters.Sys.Pressure;

   WRITELN('   PROCESSING WAVEFORM PAIR ', FileName,' (TAKES ABOUT 1/2 MIN)');
   WITH FOURIERRESULTS,ELECTRICRESULTS DO BEGIN
      {
       read in the voltage and current waveforms adding time and converting
       to actual values
      }
      ASSIGN(VFILE, PATH2+FILENAME+'.VWF');
      RESET(VFILE);
      VOLTFIRSTPOINT:=NIL;
      VOLTLASTPOINT:=NIL;
      WHILE NOT EOF(VFILE) DO BEGIN
         IF VOLTFIRSTPOINT=NIL THEN BEGIN
            NEW(VOLTFIRSTPOINT);
            VOLTLASTPOINT:=VOLTFIRSTPOINT;
         END
         ELSE BEGIN
            NEW(VOLTLASTPOINT^.NEXTPOINT);
            VOLTLASTPOINT:=VOLTLASTPOINT^.NEXTPOINT
         END;
         READ(VFILE,DATA);
         VOLTLASTPOINT^.TIME:=Data.Xvalue;
         VOLTLASTPOINT^.VOLTAGE:=Data.Yvalue*VOLTMULTIPLIER;
         VOLTLASTPOINT^.NEXTPOINT:=NIL;
      END;
      CLOSE(VFILE);
      ASSIGN(IFILE, Path2+FILENAME+'.IWF');
      RESET(IFILE);
      CURRFIRSTPOINT:=NIL;
      CURRLASTPOINT:=NIL;
      WHILE NOT EOF(IFILE) DO BEGIN
         IF CURRFIRSTPOINT=NIL THEN BEGIN
            NEW(CURRFIRSTPOINT);
            CURRLASTPOINT:=CURRFIRSTPOINT;
         END
         ELSE BEGIN
            NEW(CURRLASTPOINT^.NEXTPOINT);
            CURRLASTPOINT:=CURRLASTPOINT^.NEXTPOINT
         END;
         READ(IFILE,DATA);
         CURRLASTPOINT^.TIME    := Data.XValue;
         CURRLASTPOINT^.CURRENT:= DATA.YValue*CURRMULTIPLIER;
         CURRLASTPOINT^.NEXTPOINT:=NIL;
      END;
      CLOSE(IFILE);
      {
       Find the time interval between data points
      }
      Interval := VoltFirstPoint^.NextPoint^.Time - VoltFirstPoint^.Time;
      Writeln ('     Data Time Interval = ', (Interval*1E9):5:2, ' nsec');
      {
       use the voltage waveform to calculate the frequency by finding the
       voltage midpoint and then identifying midpoint crossings

       first, find max and min of voltage waveform and calculate midpoint
      }
      VOLTMIN:=VOLTFIRSTPOINT^.VOLTAGE; VOLTMAX:=VOLTMIN;
      VOLTCURRENTPOINT:=VOLTFIRSTPOINT^.NEXTPOINT;
      REPEAT
         IF VOLTCURRENTPOINT^.VOLTAGE < VOLTMIN THEN
            VOLTMIN:=VOLTCURRENTPOINT^.VOLTAGE;
         IF VOLTCURRENTPOINT^.VOLTAGE > VOLTMAX THEN
            VOLTMAX:=VOLTCURRENTPOINT^.VOLTAGE;
        VOLTCURRENTPOINT:=VOLTCURRENTPOINT^.NEXTPOINT;
      UNTIL VOLTCURRENTPOINT=NIL;
      Vpp := VoltMax-VoltMin;
      VOLTMID:=(VOLTMIN+VOLTMAX)/2;
      {
       now find the first midpoint crossing and count 2*CYCLES more to
       identify the desired number of cycles for fourier analysis
      }
```

```
IF (VOLTFIRSTPOINT^.VOLTAGE-VOLTMID) < 0 THEN LASTSIGN:=NEGATIVE
   ELSE LASTSIGN:=POSITIVE;
NEWSIGN := LASTSIGN;
J:=0;
NUMOFDATAPOINTS:=0;
VOLTCURRENTPOINT:=VOLTFIRSTPOINT;

REPEAT
   VOLTCURRENTPOINT:=VOLTCURRENTPOINT^.NEXTPOINT;

   {Make sure change in sign is not due to noise in data
    by checking to see if two consecuetive have the same sign}

   IF (((VOLTCURRENTPOINT^.VOLTAGE-VOLTMID) < 0) and
      ((VoltCurrentPoint^.NextPoint^.Voltage-VoltMid) < 0))
      THEN NEWSIGN:=NEGATIVE;
   IF (((VOLTCURRENTPOINT^.VOLTAGE-VOLTMID) > 0) and
      ((VoltCurrentPoint^.NextPoint^.Voltage-VoltMid) > 0))
      then NEWSIGN:=POSITIVE;

   IF (NEWSIGN <> LASTSIGN) THEN BEGIN
      IF J=0 THEN BEGIN
         VOLTSTARTPOINT:=VOLTCURRENTPOINT;
         NUMOFDATAPOINTS:=0
      END;
      J:=SUCC(J)
   END;
   NUMOFDATAPOINTS:=SUCC(NUMOFDATAPOINTS);
{   WRITELN('howdy from here!', nUMoFdATApOINTS); }
   LASTSIGN:=NEWSIGN;
UNTIL ((J=(2*CYCLES+1)) or (voltcurrentpoint^.nextpoint=nil));

If (VoltCurrentPoint^.NextPoint = nil) then begin
   Writeln('Insufficient data points for ',Cycles,' rf periods!!');
   Sound(700);  Delay(500);  NoSound;
end;

IF ((VOLTCURRENTPOINT^.TIME > VOLTSTARTPOINT^.TIME) and
    (voltcurrentpoint^.nextpoint<>nil)) THEN begin
   FREQUENCY:=CYCLES/(VOLTCURRENTPOINT^.TIME-VOLTSTARTPOINT^.TIME);
   Writeln('     Frequency (MHz): ', Frequency/1E6:6:2);
end ELSE BEGIN
   FREQUENCY:=0;
   WRITELN('      PROBLEM IN ANALYSIS: FREQUENCY SET TO ZERO')
END;

{
 find corresponding starting point in the current waveform and assume
 the same number of original data points as the voltage waveform
}
CURRCURRENTPOINT:=CURRFIRSTPOINT;
REPEAT
   CURRCURRENTPOINT:=CURRCURRENTPOINT^.NEXTPOINT;
UNTIL CURRCURRENTPOINT^.TIME >= VOLTSTARTPOINT^.TIME;
CURRSTARTPOINT:=CURRCURRENTPOINT;
{
 find max and min current value;  multiply waveforms
 point by point to get uncorrected power value
}
CurrMin := CurrStartPoint^.Current;
CurrMax := CurrMin;
PlasmaPowerRaw := VoltStartPoint^.Voltage * CurrStartPoint^.Current;
CurrCurrentPoint := CurrStartPoint^.NextPoint;
VoltCurrentPoint := VoltStartPoint^.NextPoint;
J := 1;
repeat
   If CurrCurrentPoint^.Current < CurrMin then
      CurrMin := CurrCurrentPoint^.Current;
   If CurrCurrentPoint^.Current > CurrMax then
      CurrMax := CurrCurrentPoint^.Current;
   PlasmaPowerRaw := PlasmaPowerRaw +
                     VoltCurrentPoint^.Voltage * CurrCurrentPoint^.Current;
   CurrCurrentPoint := CurrCurrentPoint^.NextPoint;
   VoltCurrentPoint := VoltCurrentPoint^.NextPoint;
   J := Succ (J);
until (J = NumOfDataPoints);
Ipp := CurrMax-CurrMin;
PlasmaPowerRaw := PlasmaPowerRaw/NumOfDataPoints;
{
 initialize the fourier data arrays
}
NEW(REALDATA);
NEW(IMAGDATA);
FILLCHAR(REALDATA^,SIZEOF(REALDATA^),0);
FILLCHAR(IMAGDATA^,SIZEOF(IMAGDATA^),0);
ERROR:=0;
{
 expand the data for the current waveform cycles to FOURIERSIZE+1
```

```
 points for Fourier analysis.
}
CURRCURRENTPOINT:=CURRSTARTPOINT;
FOR J:=0 TO FOURIERSIZE DO BEGIN
   EXPANDEDTIME:=(J*INTERVAL*(NUMOFDATAPOINTS-1)/FOURIERSIZE)+
                 CURRSTARTPOINT^.TIME;
   WHILE EXPANDEDTIME > CURRCURRENTPOINT^.NEXTPOINT^.TIME DO
      CURRCURRENTPOINT:=CURRCURRENTPOINT^.NEXTPOINT;
   IF INTERVAL > 0 THEN
    REALDATA^[J]:=(((EXPANDEDTIME-CURRCURRENTPOINT^.TIME)/INTERVAL)*
    (CURRCURRENTPOINT^.NEXTPOINT^.CURRENT-CURRCURRENTPOINT^.CURRENT))+
    CURRCURRENTPOINT^.CURRENT
   ELSE BEGIN
      REALDATA^[J]:=0;
      IF J=0 THEN WRITELN('      INTERVAL<=0 ? REALDATA^[J] SET TO 0')
   END
END;
{
 call the real fast fourier procedure from TURBO PASCAL NUMERICAL
 METHODS TOOL BOX
}
REALFFT((FOURIERSIZE+1),FALSE,REALDATA,IMAGDATA,ERROR);
{
 That was easy, wasn't it ?!! The REALDATA and IMAGDATA pointers now
 point to the results of the fourier analysis.  We are interested in
 the number of harmonics specified in the MAGLOBAL definitions unit.
 The results corresponding to these harmonics are located in the
 array of results depending on how many cycles are included in the
 data sent to the fourier anlaysis.  The zeroth harmonic data will
 always be at REALDATA^[0].  The first harmonic data will be at
 REALDATA^[CYCLES] and IMAGDATA^[CYCLES]; the second harmonic at
 REALDATA^[2*CYCLES], etc, etc.  Each amplitude must be divided by
 the square root of FOURIERSIZE+1 and the amplitudes for harmonics
 greater than zero must be multiplied by two to take into account
 the negative frequencies.
}
IIN.R:=2*REALDATA^[CYCLES]/SQRT(FOURIERSIZE+1);
IIN.I:=2*IMAGDATA^[CYCLES]/SQRT(FOURIERSIZE+1);
CURRAMP[0]:=REALDATA^[0]/SQRT(FOURIERSIZE+1);
CURRANG[0]:=0;
FOR J:=1 TO HARMONICMAX DO BEGIN
   CURRAMP[J]:=2*SQRT(SQR(REALDATA^[J*CYCLES])+SQR(IMAGDATA^[J*CYCLES]))
               /SQRT(FOURIERSIZE+1);
   IF REALDATA^[J*CYCLES] = 0 THEN BEGIN
      IF IMAGDATA^[J*CYCLES] >= 0 THEN CURRANG[J]:=PI/2
      ELSE CURRANG[J]:=-1*PI/2
   END
   ELSE BEGIN
      CURRANG[J]:=ARCTAN(IMAGDATA^[J*CYCLES]/REALDATA^[J*CYCLES]);
      IF REALDATA^[J*CYCLES] < 0 THEN CURRANG[J]:=CURRANG[J] + PI
   END
END;
DISPOSE(REALDATA);
DISPOSE(IMAGDATA);
{
 repeat the data expansion and fourier analysis for the voltage
 waveform
}
NEW(REALDATA);
NEW(IMAGDATA);
FILLCHAR(REALDATA^,SIZEOF(REALDATA^),0);
FILLCHAR(IMAGDATA^,SIZEOF(IMAGDATA^),0);
ERROR:=0;
VOLTCURRENTPOINT:=VOLTSTARTPOINT;
FOR J:=0 TO FOURIERSIZE DO BEGIN
   EXPANDEDTIME:=(J*INTERVAL*(NUMOFDATAPOINTS-1)/FOURIERSIZE)+
                 VOLTSTARTPOINT^.TIME;
   WHILE EXPANDEDTIME > VOLTCURRENTPOINT^.NEXTPOINT^.TIME DO
      VOLTCURRENTPOINT:=VOLTCURRENTPOINT^.NEXTPOINT;
   IF INTERVAL > 0 THEN
    REALDATA^[J]:=(((EXPANDEDTIME-VOLTCURRENTPOINT^.TIME)/INTERVAL)*
    (VOLTCURRENTPOINT^.NEXTPOINT^.VOLTAGE-VOLTCURRENTPOINT^.VOLTAGE))+
    VOLTCURRENTPOINT^.VOLTAGE
   ELSE BEGIN
      REALDATA^[J]:=0;
      IF J=0 THEN WRITELN('      INTERVAL<=0 ? REALDATA^[J] SET TO 0')
   END
END;
REALFFT((FOURIERSIZE+1),FALSE,REALDATA,IMAGDATA,ERROR);
VIN.R:=2*REALDATA^[CYCLES]/SQRT(FOURIERSIZE+1);
VIN.I:=2*IMAGDATA^[CYCLES]/SQRT(FOURIERSIZE+1);
VOLTAMP[0]:=REALDATA^[0]/SQRT(FOURIERSIZE+1);
VOLTANG[0]:=0;
FOR J:=1 TO HARMONICMAX DO BEGIN
   VOLTAMP[J]:=2*SQRT(SQR(REALDATA^[J*CYCLES])+SQR(IMAGDATA^[J*CYCLES]))/
               SQRT(FOURIERSIZE+1);
   IF REALDATA^[J*CYCLES] = 0 THEN BEGIN
      IF IMAGDATA^[J*CYCLES] >= 0 THEN VOLTANG[J]:=PI/2
```

```
      ELSE VOLTANG[J]:=-1*PI/2
   END
   ELSE BEGIN
      VOLTANG[J]:=ARCTAN(IMAGDATA^[J*CYCLES]/REALDATA^[J*CYCLES]);
      IF REALDATA^[J*CYCLES] < 0 THEN VOLTANG[J]:=VOLTANG[J] + PI
   END
END;
DISPOSE(REALDATA);
DISPOSE(IMAGDATA);
{
 Now correct the first harmonic data of the current and voltage
 waveforms for stray impedances before calculating plasma properties.
 This is done by de-embedding using a two port matrix model.

 The two port matrix model is shown schematically:

                 ------------
          O------|            |------|
        INPUT    |  Z11  Z12  |      ----
       Iin,Vin   |            |     | Zp'|
                 |  Z21  Z22  |      ----
          O------|            |------|
                 ------------

 Calculation of the current and voltage at the electrodes (Zp') from the
 measured voltage and current at the input is done using the
 following relations:

  Ip' = (Z11/Z12)*Iin - (1/Z12)*Vin

  Vp' = (Z21-(Z22*Z11/Z12)*Iin + (Z22/Z12)*Vin

 Z11,Z22,Z21 are determined experimentally by unterminating the
 network.  Since it is not possible to determine Z12 and
 Z22 uniquely, it is assumed that the matrix is reciprocal thus
 Z12=Z22.
}
WRITELN('   DE-EMBEDDING . . .');
Mult_Comp(Z22,Z11,C1);
Div_Comp(C1,Z12,C2);
Sub_Comp(Z21,C2,C3);
Mult_Comp(C3,IIN,C4);
Div_Comp(Z22,Z12,C5);
Mult_Comp(C5,VIN,C6);
Add_Comp(C6,C4,VLOAD);
Div_Comp(Z11,Z12,C1);
Mult_Comp(C1,IIN,C2);
Div_Comp(VIN,Z12,C3);
Sub_Comp(C2,C3,ILOAD);
Complex_to_Polar(VLOAD,DEEMBEDVOLT);
Complex_to_Polar(ILOAD,DEEMBEDCURR);

{subtract out the electrode capcitance}
If (ElecCorr.Correct) then begin
   Writeln ('      Subtracting out the electrode capacitance.');
   Div_Comp (VLOAD, ElecCorr.ElecImped, C1);
   Sub_Comp (ILOAD, C1, COMPLEXCURRENT);
   Complex_to_Polar (COMPLEXCURRENT, DEEMBEDCURR);
end;

{Calculate Current Flux to electrode}
CurrArea := DeEmbedCurr.M/Area;
{Calculate plasma impedance}
IF DeEmbedCurr.M <> 0 THEN ZPLASMA:=DeEmbedVolt.M/DeEmbedCurr.M
ELSE BEGIN
   ZPLASMA:=0;
   WRITELN('      ZPLASMA SET TO ZERO')
END;
PHASEDELAY:=DeEmbedVolt.A-DeEmbedCurr.A;
{
 Proceed to calculate all sorts of additional plasma parameters based
 on the impedance corrected FIRST HARMONICS.
}
PLASMAPOWER:=DeEmbedVolt.M*DeEmbedCurr.M*COS(PHASEDELAY)/2;
PlasmaPowerArea := PlasmaPower/Area;
OMEGA := 2*PI*FREQUENCY;
{
 MODEL 1 : USE ELECTRICAL ANALOG MODEL IN ITS SIMPLEST FORM
}
IF DeEmbedCurr.M <> 0 THEN
   PLASMARES1:=DeEmbedVolt.M*COS(PHASEDELAY)/DeEmbedCurr.M
ELSE BEGIN
   WRITELN('   DeEmbedCurr = 0 ??, PLASMARES1 SET TO 0');
   PLASMARES1:=0;
END;
IF ((DeEmbedVolt.M <> 0) AND (OMEGA <> 0) AND (SIN(PHASEDELAY) <> 0)) THEN
   SHEATHCAP1:=-1*2*DeEmbedCurr.M/(OMEGA*DeEmbedVolt.M*SIN(PHASEDELAY))
ELSE BEGIN
```

```
   WRITELN('   PROBLEM WITH SHEATH CALC, SHEATHCAP1 SET TO 0');
   SHEATHCAP1:=0;
END;
IF ((SHEATHCAP1 <> 0) AND (OMEGA <> 0)) THEN
   SHEATHVOLT1:=DeEmbedCurr.M/(OMEGA*SHEATHCAP1)
ELSE BEGIN
   WRITELN('   PROBLEM WITH SHEATH CALC, SHEATHVOLT1 SET TO 0');
   SHEATHVOLT1:=0;
END;
IF ((PRESSURE <> 0) AND (SPACING <> 0)) THEN
   EOVERP1:=PLASMARES1*DeEmbedCurr.M/(SQRT(2)*PRESSURE*SPACING)
ELSE BEGIN
   WRITELN('   PROBLEM WITH EOVERP CALC, EOVERP1 SET TO 0');
   EOVERP1:=0
END;

If EOVERP1 <> 0 then begin
   CalculateMobility (MobilityPressure, EOVERP1, Parameters.Sys.GasName);
   Mobility1 := MobilityPressure/Pressure;
end else begin
   MobilityPressure := 0;
   Mobility1 := 0;
end;
IF ((AREA <> 0) AND (PLASMARES1 <> 0) AND (MOBILITYPRESSURE <> 0)) THEN
   ELECTRONDEN1:=SPACING*PRESSURE/(MOBILITYPRESSURE*AREA*PLASMARES1*ELECTRONCHARGE)
ELSE BEGIN
   WRITELN('   PROB WITH ELECTRON CALC, ELECTRONDEN1 SET TO 0');
   ELECTRONDEN1:=0
END;

 {
 MODEL 2 : ACCOUNT FOR BULK CAPACTIANCE BETWEEN ELECTRODES

 calculate plasma resistance and sheath capacitance by iterating
 on the following model:
```




```
}
IF SPACING > 0 THEN VACUUMCAP := PERMITTIVITY*AREA/SPACING
ELSE BEGIN
   VACUUMCAP:=0;
   WRITELN('      SPACING <= 0 ?  VACCUMCAP SET TO ZERO')
END;
IF DeEmbedCurr.A <> 0 THEN BEGIN
   REALPLASMAIMPED := (DeEmbedVolt.M/DeEmbedCurr.M)*COS(PHASEDELAY);
   IMAGPLASMAIMPED := (DeEmbedVolt.M/DeEmbedCurr.M)*SIN(PHASEDELAY)
END
ELSE BEGIN
   REALPLASMAIMPED:=0;
   IMAGPLASMAIMPED:=0;
   WRITELN('      PLASMACURRAMP = 0 ?  PLASMAIMPED VALUES SET TO 0')
END;
WRITELN('   ACCOUNTING FOR ELECTRODE CAPACITANCE');

BULKCAP:=VACUUMCAP;
J:=0;
JUMPOUT:=FALSE;
REPEAT
   J:=SUCC(J);
   IF SQR(2*REALPLASMAIMPED*OMEGA*BULKCAP) <= 1 THEN
      PLASMARES2 := 2 * REALPLASMAIMPED /
                    (1 + SQRT(1 - SQR(2*REALPLASMAIMPED*OMEGA*BULKCAP)))
   ELSE BEGIN
      PLASMARES2:=0;
      JUMPOUT:=TRUE;
      WRITELN('      CANNOT TAKE SQRT OF NEG NUM.  PLASMARES2 SET TO 0')
   END;
   IF ((OMEGA <> 0) AND (PLASMARES2 <> 0) AND (VACUUMCAP > 0)) THEN BEGIN
      SHEATHCAP2 := -1*2 / (OMEGA * (IMAGPLASMAIMPED + OMEGA*BULKCAP/
                            (SQR(1/PLASMARES2)+SQR(OMEGA*BULKCAP))));
      SHEATHVOLT2 := DeEmbedCurr.M / (OMEGA * SHEATHCAP2)
```

```
            END
            ELSE BEGIN
               SHEATHCAP2:=0;
               JUMPOUT:=TRUE;
               WRITELN('      PROBLEMS IN ANALYSIS.  SHEATHCAP2,SHTHVOLT SET TO 0')
            END;
            IF (SHEATHCAP2 > 0) AND (VACUUMCAP > 0) AND (BULKCAP > 0) THEN BEGIN
               NEWBULKCAP:=1/((1/VACUUMCAP)-(2/SHEATHCAP2));
               IF (ABS(NEWBULKCAP-BULKCAP)/BULKCAP) < 0.01 THEN BEGIN
                  WRITELN('      CONVERGENCE !!!');
                  JUMPOUT:=TRUE
               END
            END
            ELSE BEGIN
               JUMPOUT:=TRUE;
               WRITELN('      PROBLEMS IN PLASMA IMPED LOOP')
            END;
            BULKCAP:=NEWBULKCAP;
            IF J > 100 THEN BEGIN
               WRITELN('      MORE THAN 100 ITERATIONS IN IMPED LOOP !!');
               JUMPOUT:=TRUE
            END
         UNTIL JUMPOUT=TRUE;
         ITERATIONS:=J;
         {
          end of impedance iteration loop
         }
         IF ((PRESSURE <> 0) AND (SPACING <> 0)) THEN
            EOVERP2:=PLASMARES2*DeEmbedCurr.M/(SQRT(2)*PRESSURE*SPACING)
         ELSE BEGIN
            WRITELN('   PROBLEM WITH EOVERP CALC, EOVERP2 SET TO 0');
            EOVERP2:=0
         END;
         If EOVERP2 <> 0 then begin
            CalculateMobility (MobilityPressure, EOVERP2, Parameters.Sys.GasName);
            Mobility2 := MobilityPressure/pressure;
         end else begin
            MobilityPressure := 0;
            Mobility2 := 0;
         end;
         IF ((AREA <> 0) AND (PLASMARES2 <> 0) AND (MOBILITYPRESSURE <> 0)) THEN
            ELECTRONDEN2:=SPACING*PRESSURE/(MOBILITYPRESSURE*AREA*PLASMARES2*ELECTRONCHARGE)
         ELSE BEGIN
            WRITELN('   PROB WITH ELECTRON CALC, ELECTRONDEN2 SET TO 0');
            ELECTRONDEN2:=0
         END;
         {
          Create plotting files if so desired
         }
         IF PLOTDO THEN BEGIN
            {
             Store time, voltage, then current values in textfile
            }
            ASSIGN(TEXTFILE,PATH3+'W'+FILENAME+'.DAT');
            REWRITE(TEXTFILE);
            VOLTCURRENTPOINT:=VOLTFIRSTPOINT;
            CurrCurrentPoint := CurrFirstPoint;
            REPEAT
               WRITELN(TEXTFILE,VOLTCURRENTPOINT^.TIME:11,' ',
                       VOLTCURRENTPOINT^.VOLTAGE:11, ' ',
                       CurrCurrentPoint^.Current:11);
               VOLTCURRENTPOINT:=VOLTCURRENTPOINT^.NEXTPOINT;
               CurrCurrentPoint:=CurrCurrentPoint^.NextPoint;
            UNTIL VOLTCURRENTPOINT=NIL;
            CLOSE(TEXTFILE);
         END; { of IF PLOTDO THEN . . . clause }
         {
          reclaim heap memory; dispose pointers
         }
         REPEAT
            CURRCURRENTPOINT:=CURRFIRSTPOINT;
            CURRFIRSTPOINT:=CURRCURRENTPOINT^.NEXTPOINT;
            DISPOSE(CURRCURRENTPOINT)
         UNTIL CURRFIRSTPOINT=NIL;
         REPEAT
            VOLTCURRENTPOINT:=VOLTFIRSTPOINT;
            VOLTFIRSTPOINT:=VOLTCURRENTPOINT^.NEXTPOINT;
            DISPOSE(VOLTCURRENTPOINT)
         UNTIL VOLTFIRSTPOINT=NIL;
      END; { WITH FOURIERRESULTS,ELECTRICRESULTS DO loop }
END; { PROCEDURE ELECTRICALC }

END. { UNIT ELECTRIC }
```

```
Unit Cmplx;
{
 In the following descriptions,
  Capital letters (A,B) are real numbers or real parts of complex
  numbers.  Lowercase letters (a,b) are real factors of complex
  parts.  i is the square root of -1 (Sqrt(-1-))

  written by J. Butterbaugh

  CONTENTS:  Add_Comp          (C1, C2, C_Out);
             Sub_Comp          (C1, C2, C_Out);
             Mult_Comp         (C1, C2, C_Out);
             Mult_RC           (C, R, C_Out);
             Sub_C_From_R      (R, C, C_Out);
             Div_C_By_R        (R, C, C_Out);
             Size_of_C         (C);
             Square_Size_of_C  (C);
            Conj_of_C          (A,C); actually C is the output, A the input
            Squareroot_of_C    (C,ROOTC);
             Negative          (A,C)
}

INTERFACE

USES
   Define;

PROCEDURE Add_Comp(A, B: Complex; VAR C: Complex);

PROCEDURE Sub_Comp(A, B: Complex; VAR C: Complex);

PROCEDURE Mult_Comp(A, B: Complex; VAR C: Complex);

PROCEDURE Div_Comp(A, B: Complex; VAR C: Complex);

PROCEDURE Div_R_By_C(R: Double; C: Complex; VAR C_Out: Complex);

PROCEDURE Mult_RC(C: Complex; R: Double; VAR C_Out: Complex);

PROCEDURE Sub_C_From_R(R: Double; C: Complex; VAR C_Out: Complex);

PROCEDURE Div_C_By_R(C: Complex; R: Double; VAR C_Out: Complex);

FUNCTION Size_of_C(C:Complex):Double;

FUNCTION Square_Size_of_C(C:Complex):Double;

PROCEDURE Conj_of_C(A: Complex; VAR C: Complex);

PROCEDURE Complex_to_Polar(C: Complex; VAR P: Polar);

PROCEDURE Polar_to_Complex(P: Polar; VAR C: Complex);

PROCEDURE Squareroot_of_C(C: Complex; VAR ROOTC: Complex);

PROCEDURE Negative(A: Complex; VAR C: Complex);

{*************************************************************************}

IMPLEMENTATION

PROCEDURE Add_Comp(A, B: Complex; VAR C: Complex);

{ RESULT == (A+ai)+(B+bi) == A+ai+B+bi == (A+B)+ (a+b)i }

BEGIN
   C.R:=A.R+B.R;
   C.I:=A.I+B.I;
END;

{*************************************************************************}


PROCEDURE Sub_Comp(A, B: Complex; VAR C: Complex);

{ RESULT == (A+ai)-(B+bi) == A+ai-B-bi == (A-B)+ (a-b)i }

BEGIN
   C.R:=A.R-B.R;
   C.I:=A.I-B.I;
END;

{*************************************************************************}

PROCEDURE Mult_Comp(A, B: Complex; VAR C: Complex);

{ RESULT == (A+ai)(B+bi) == AB+Abi+Bai+aibi == (AB-ab)+(Ab+aB)i }
```

```
BEGIN
   C.R:=A.R*B.R-A.I*B.I;
   C.I:=A.R*B.I+A.I*B.R;
END;

{*************************************************************************}

PROCEDURE Div_Comp(A, B: Complex; VAR C: Complex);

{ RESULT == (A+ai)/(B+bi) == (AB+ab)/(B^2+b^2)+((aB-Ab)/(B^2+BA^2))i }

VAR
   D:Double;

BEGIN
   D:=Sqr(B.R)+Sqr(B.I);
   C.R:=(A.R*B.R+A.I*B.I)/D;
   C.I:=(A.I*B.R-A.R*B.I)/D;
END;

{*************************************************************************}

PROCEDURE Div_R_By_C(R: Double; C: Complex; VAR C_Out: Complex);

VAR
  A:Complex;

BEGIN
   A.R:=R;
   A.I:=0;
   Div_Comp(A,C,C_Out);
END;

{*************************************************************************}

PROCEDURE Mult_RC(C: Complex; R: Double; VAR C_Out: Complex);

{ RESULT == (C+ci)R == CR+cRi }

BEGIN
   C_Out.R:=C.R*R;
   C_Out.I:=C.I*R;
END;

{*************************************************************************}

PROCEDURE Sub_C_From_R(R: Double; C: Complex; VAR C_Out: Complex);

{ RESULT == R-(C+ci) == R-C-ci == (R-C)-ci }

BEGIN
   C_Out.R:=R-C.R;
   C_Out.I:=-C.I;
END;

{*************************************************************************}

PROCEDURE Div_C_By_R(C: Complex; R: Double; VAR C_Out: Complex);

{ RESULT == (C+ci)/R == C/B+ci/R == (C/R)+(c/R)i  }

BEGIN
   C_Out.R:=C.R/R;
   C_Out.I:=C.I/R;
END;

{*************************************************************************}

FUNCTION Size_of_C(C:Complex):Double;

{ RESULT == Sqrt(C^2 +c^2) }

BEGIN
   Size_of_C:=Sqrt(Sqr(C.R)+Sqr(C.I));
END;

{*************************************************************************}

FUNCTION Square_Size_of_C(C:Complex):Double;

{ RESULT == C^2 +c^2 }

BEGIN
```

```
   Square_Size_Of_C:=Sqr(C.R)+Sqr(C.I);
END;

{*************************************************************************}

PROCEDURE Conj_of_C(A: Complex; VAR C: Complex);

{ RESULT = A-ai }

BEGIN
   C.R:=A.R;
   C.I:=-1*A.I
END;

{*************************************************************************}

PROCEDURE Complex_to_Polar;

BEGIN
   P.M:=Size_of_C(C);
   IF C.R = 0 THEN BEGIN
      IF C.I > 0 THEN P.A:=PI/2
      ELSE P.A:=-1*PI/2
   END
   ELSE BEGIN
      IF C.R > 0 THEN P.A:=ARCTAN(C.I/C.R)
      ELSE P.A:=ARCTAN(C.I/C.R)+PI
   END
END;

{*************************************************************************}

PROCEDURE Polar_to_Complex;

BEGIN
   C.R:=P.M*COS(P.A);
   C.I:=P.M*SIN(P.A)
END;

{*************************************************************************}

PROCEDURE Squareroot_of_C;

VAR
   P:POLAR;

BEGIN
   Complex_to_Polar(C,P);
   P.M:=SQRT(P.M);
   P.A:=P.A/2;
   Polar_to_Complex(P,ROOTC)
END;

{*************************************************************************}

PROCEDURE Negative;

BEGIN
   C.R:=-1*A.R;
   C.I:=-1*A.I;
END;

END.
```

```
Program Alter;
{       This program is used to scan through stored data (X,Y) columns,
alter old ion energy record files to ASCII files, view voltage and
current waveforms that are stored in record format, or average or smooth
noisy data files such as the temporal optical scans.}

Uses Dos, Crt, GDriver, Gkernel, GWindow, GShell;

Type
   DataPointer = ^DataList;
   DataList = record
      XValue : double;
      YValue : double;
      NextPoint : DataPointer;
   end;

   StringKind = (Alfa,Numeric,Mix);

   ParamsRecord = record
      Max,Min   : real;
      XStart,XEnd : real;
   end;

   CommLinePos=17..24;
   str60 = string[60];
   str8  = string[8];

Var
   IsExit           : boolean;  {exit program?}
   IsSpectra        : boolean;
   FuncKey          : integer;
   FirstPoint       : DataPointer;
   Params           : ParamsRecord;
   Z                : real;       {axial location in temporal PIE scan}
   Path             : str60;


PROCEDURE SetUpWindows;

         {Define all windows for all sections of the program.
          Window 5:  Command window (Upper right).
          Window 6:  Communication window (Bottom).
          Window 7:  Graph window (Upper left).}

const
   MaxWorldX:Float=1000.0;          {Maximum World for X-dir. for Window 1}
   MaxWorldY:Float=1000.0;          {Maximum World for Y-dir. for Window 1}

begin
   DefineWindow (5,(XMaxGlb-23),12,(XMaxGlb-2),(YMaxGlb-150));
   DefineWindow (6,2,(YMaxGlb-138),(XMaxGlb-2),(YMaxGlb-12));
   DefineWindow (7,2,12,(XMaxGlb-25),(YMaxGlb-150));
end;

PROCEDURE GetString (var InputString  : str8;
                         KindOfString : StringKind;
                         Xpos         : integer;
                         Ypos         : integer);
                             {The procedure obtains a string from the
                              user.  The string is terminasted by a return.}
var
   CharValue, BoundLow, BoundHi, LetterCount : integer;
   CharIn : char;
   NewString : string[8];
   IsReturn : boolean;

begin
   IsReturn := false;
   LetterCount := 0;
   NewString := '';

   if (KindOfString = Alfa) then begin
      BoundLow := 64; BoundHI := 91; end;
   if (KIndOfString = Numeric) then begin
      BoundLow := 44; BoundHi := 58; end;
   if (KindOfString = Mix) then begin
      BoundLow := 44; BoundHi := 91; end;

   repeat

      repeat

       {Get a character}
         CharIn := readkey;

       {Check for special code and if true read the next character}
         if (CharIn = #0) then begin
            CharIn := readkey;
```

```
            CharValue := 0;
         end

         else begin
            CharIn := UpCase(CharIn);
            CharValue := ord(CharIn);

          {Check fo termination}
            if (CharIn = chr(13)) then IsReturn := true;
            if (NewString = '') then begin
               GotoXY (Xpos,Ypos);
               Write ('        ');
               GotoXY (Xpos,Ypos);
            end;

          {Check for backspace}
            if (CharIn = chr(8)) then begin
               Delete (NewString, LetterCount, 1);
               LetterCount := LetterCount - 1;
               CharValue := 0;
               GotoXY (Xpos,Ypos);
               Write ('      ');
               GotoXY (Xpos,Ypos);
               Write (NewString);
            end;
         end;

        {Inform of a not acceptable character}
         if not ((((CharValue > BoundLow) and (CharValue < BoundHi))
            or IsReturn) or (CharValue = 32))  then begin
             Sound (300);
             Delay (200);
             NoSound;
         end;

      until (((CharValue > BoundLow) and (CharValue < BoundHi)) or IsReturn);

      if (CharIn <> chr(13)) then begin
         if ((NewString = '') and (CharIn = '.')) then NewString := '0';
         NewString := NewString+CharIn;
         LetterCount := LetterCount + 1;
         Write (CharIn);
      end;

   until IsReturn;

   if (NewString <> '') then InputString := NewString;
end;


PROCEDURE DrawSpectra (FirstPoint : DataPointer;
                       Max        : double;
                       Min        : double;
                       XStart     : real;
                       XEnd       : real;
                       Window     : integer);

var
   CurrentPoint : DataPointer;
   X, XOld : real;
   Y, YOld : double;
   TicHeight : double;
   TicSpace  : real;
   I         : integer;
   ValueString : string;

begin
   SelectScreen (1);
   if (Max <= Min) then Max := Min + 1.0;
   DefineWorld (4, XStart, Max+(Max-Min)*0.05, XEnd, Min-(Max-Min)*0.05);
   SelectWorld (4);
   SelectWindow (Window);
   TicHeight := (Max-Min)/30.0;
   TicSpace  := (XEnd-XStart)/5.0;
   I := 0;
   repeat
      DrawLine (XStart+TicSpace*I,Min-(Max-Min)*0.05,XStart+TicSpace*I,
                TicHeight+Min-(Max-Min)*0.05);
      Str ((Xstart+TicSpace*I):6:1,ValueString);
      DrawTextW (XStart+TicSpace*I,TicHeight+Min,1,ValueString);
      I := I + 1;
   until (I >= 5);

   CurrentPoint := FirstPoint;
   XOld := CurrentPoint^.Xvalue;
   YOld := CurrentPoint^.Yvalue;
   while (CurrentPoint <> nil) do begin
      X := CurrentPoint^.Xvalue;
```

```
      Y := CurrentPoint^.Yvalue;
      DrawLine (XOld, YOld, X, Y);
      CurrentPoint := CurrentPoint^.NextPoint;
      XOld := X;
      YOld := Y;
   end;
end;

PROCEDURE WriteScreen;

Const
   MaxWorldX:Float=1000.0;
   MaxWorldY:Float=1000.0;
Begin
   DefineHeader(7,'Spectrum');
   SetHeaderOn;
   SelectWindow(7);
   DrawBorder;

   DefineHeader(5,'Commands');
   SetHeaderOn;
   SelectWindow(5);
   DrawBorder;
   GotoXY(59,3);  Write('F1=Retrieve Data');
   GotoXY(59,4);  Write('F2=Scan Data');
   GotoXY(59,5);  Write('F3=Convert IED File');
   GotoXY(59,6);  Write('F4=Smooth Data');
   GotoXY(59,7);  Write('F5=Save PIE Data');
   GotoXY(59,8);  Write('F6=Average Data');
   GotoXY(59,9);  Write('F7=Retrieve Waveform');
   GotoXY(59,12); Write('F10=Exit');

   DefineHeader(6,'Communication Window');
   SetHeaderOn;
   SelectWindow(6);
   DrawBorder;

   GotoXY(5,24);  Write('Press Function Key ...');
end;

PROCEDURE GetFuncKey (var Funckey : integer);

var
   Key : char;

begin
   Key := ReadKey;
   if Key = #0 then begin
      Key := ReadKey;
      FuncKey := ord(Key);
   end
   else begin
      Sound (300); Delay (100); NoSound;
      FuncKey := 1;
   end;
end;


PROCEDURE ClearCommLine (Line: integer);
    {Clears command line in main section and ion energy section}

begin
   GotoXY (4,Line);
   Write ('                                                                    ');
end;

PROCEDURE ClearCommWindow (WindowNumber:integer);
{This Procedure clears the communication window.}

var
   Lpos:CommLinePos;                {Screen Output Position}
   i:integer;                       {Counter}

begin
  {Start for top of communciation window (17) and write blanks to
   all lines in communication window}

   If WindowNumber = 6 then i:=17;
   If WindowNumber = 9 then i:=18;

   For Lpos:=i to 24 do begin
      GotoXY(4,Lpos);
      Write('                                                                  ');
   end;
end;

PROCEDURE ClearGraphWindow;
{This procedure clears the graph window in waveform section}
```

```
var
   Lpos:integer;                {Screen Output Position}
   i:integer;                       {Counter}

begin
  {Start for top of graph window (6) and write blanks to
   all lines in communication window}

   For Lpos:=2 to 14 do begin
      GotoXY(3,Lpos);
      for i:=1 to 53 do Write(' ');
   end;
   DefineHeader(7,'Spectrum');
   SetHeaderOn;
   SelectWindow(7);
   DrawBorder;
end;

PROCEDURE GetDescription (var InputString  : str60;
                              Xpos         : integer;
                              Ypos         : integer);
                             {The procedure obtains a string from the
                              user.  The string is terminated by a return.}
var
   CharValue, LetterCount : integer;
   CharIn : char;
   NewString : str60;
   IsReturn : boolean;

begin
   IsReturn := false;
   LetterCount := 0;
   NewString := '';

   repeat

      repeat

       {Get a character}
         CharIn := readkey;

       {Check for special code and if true read the next character}
         if (CharIn = #0) then begin
            CharIn := readkey;
            CharValue := 0;
         end

         else begin
            CharValue := ord(CharIn);

          {Check fo termination}
            if (CharIn = chr(13)) then IsReturn := true;
            if (NewString = '') then begin
               GotoXY (Xpos,Ypos);
               Write ('                                ',
                      '                                ');
               GotoXY (Xpos,Ypos);
            end;

          {Check for backspace}
            if (CharIn = chr(8)) then begin
               Delete (NewString, LetterCount, 1);
               LetterCount := LetterCount - 1;
               {CharValue := 0;}
               GotoXY (Xpos,Ypos);
               Write ('                                ',
                      '                                ');
               GotoXY (Xpos,Ypos);
               Write (NewString);
            end;
          end;

        {Inform of a not acceptable character}
         if not (((CharValue > 31) and (CharValue < 123)) or IsReturn)
            then begin
             Sound (300);
             Delay (200);
             NoSound;
         end;

      until (((CharValue > 31) and (CharValue < 123)) or IsReturn);

      if not(IsReturn) then begin
         NewString := NewString+CharIn;
         LetterCount := LetterCount + 1;
         Write (CharIn);
      end;
```

```
   until IsReturn;

   if (NewString <> '') then InputString := NewString;
end;

PROCEDURE EraseData(var FirstPoint:DataPointer);
var
   CurrentPoint :DataPointer;
begin
   while (FirstPoint <> nil) do begin
      CurrentPoint := FirstPoint;
      FirstPoint := CurrentPoint^.NextPoint;
      dispose (CurrentPoint);
   end;
end;


PROCEDURE GetData (var FirstPoint:DataPointer; var Z:real;
                   var Params:ParamsRecord; var IsSpectra:boolean; var Path:str60);
{This retrieves data in X-Y columns from a disk or hard drive}
Var
   DirInfo    : SearchRec;
   Found      : boolean;
   LastPoint  : DataPointer;
   RetrievedFile : text;
   X,Y           : double;
   FileName      : str60;
   InputString   : str8;
   code          : integer;
   dummy         : real;

Begin
   ClearCommWindow(6);
   GotoXY(5,17);  Write('Enter Filename: ');
   GetDescription(FileName,22,17);
   GotoXy(5,18);  Write('Path: ',Path);
   GotoXy(5,19);  Write('Enter Path:');
   GetDescription(Path,17,19);
   GotoXY(17,19);  write(Path);
   FindFirst (Path+filename,AnyFile,DirInfo);
   If DosError = 0 then begin
      Found := true;
      repeat
         GotoXY (5,20);  Write ('Enter Z value:');
         GetString(InputString,Numeric,20,20);
         Val(InputString,Z,Code);
         If Code <> 0 then begin
            Sound (700);  Delay (500);  NoSound;
            GotoXY (5,21);  Write ('INVALID KEYS!!  TRY AGAIN!!');
            GotoXY (20,20);  Write ('            ');
         end;
      until (code = 0);
      ClearCommLine(21);
      GotoXY (5,21);  Write ('Retrieving data...');
      If IsSpectra then begin
         ClearGraphWindow;
         EraseData (FirstPoint);
      end;

      {Initialize variables}
      FirstPoint           := nil;
      LastPoint            := nil;
      LastPoint^.NextPoint := nil;

      Assign (RetrievedFile, Path+Filename);
      Reset (RetrievedFile);
      Readln (RetrievedFile, X,Y);
      With Params do begin
         repeat
            If FirstPoint = nil then begin
               New (FirstPoint);
               LastPoint := FirstPoint;
              XStart := X;
              Max := Y;
              Min := Y;
           end else begin
               New (LastPoint^.NextPoint);
               LastPoint := LastPoint^.NextPoint;
              {Find Max. and Min. Yvalue}
               If Max < Y then Max := Y;
               If Min > Y then Min := Y;
             XEnd := X;
           end;
           LastPoint^.Xvalue := X;
           LastPoint^.Yvalue := Y;
           LastPoint^.NextPoint := nil;

          Readln (RetrievedFile,X,Y);
```

```
        until (EoF(RetrievedFile));
      end;
      Close(RetrievedFile);
      IsSpectra := true;
      {write name of file retrieved}
      GotoXY(59,13);  write(FileName);
      ClearCommWindow(6);
   end else begin
      GotoXY(5,20);  Write('File not found.');
      ClearCommLine(17);
      Sound(700);  Delay(100);  NoSound;
   end;
   GotoXY(5,24);  Write('Press Function Key ...');
end;


PROCEDURE GetWaveform (var FirstPoint:DataPointer; var Path:Str60;
                       var Params:ParamsRecord; var IsSpectra:boolean);
{This retrieves data in waveform record format from a disk or hard drive}
Type
   DataRecord = record
      X    : real;
      Y    : real;
   end;

Var
   DirInfo    : SearchRec;
   LastPoint  : DataPointer;
   Data       : DataRecord;
   RetrievedFile : File of DataRecord;
   FileName       : str60;
   InputString    : str8;
   code           : integer;
   dummy          : real;
   J,I           : integer;

Begin
   ClearCommWindow(6);
   GotoXY(5,17);  Write('Enter Filename: ');
   GetDescription(FileName,22,17);
   GotoXy(5,18);  Write('Path: ',Path);
   GotoXy(5,19);  Write('Enter Path:');
   GetDescription(Path,17,19);
   GotoXY(17,19);  write(Path);
   FindFirst (Path+filename,AnyFile,DirInfo);
   If DosError = 0 then begin
      ClearCommLine(21);
      GotoXY (5,21);  Write ('Retrieving data...');
      If IsSpectra then EraseData(FirstPoint);

      {Initialize variables}
      FirstPoint           := nil;
      LastPoint            := nil;
      LastPoint^.NextPoint := nil;

      Assign (RetrievedFile, Path+Filename);
      Reset (RetrievedFile);
      Read (RetrievedFile, Data);
      With Params,Data do begin
         J := 23;
         I := 0;
         repeat
            If FirstPoint = nil then begin
               New (FirstPoint);
               LastPoint := FirstPoint;
              XStart := X;
              Max := Y;
              Min := Y;
            end else begin
               New (LastPoint^.NextPoint);
               LastPoint := LastPoint^.NextPoint;
               {Find Max. and Min. Yvalue}
               If Max < Y then Max := Y;
               If Min > Y then Min := Y;
              XEnd := X;
            end;
            LastPoint^.Xvalue := X;
            LastPoint^.Yvalue := Y;
            LastPoint^.NextPoint := nil;

            If I = 100 then begin
               GotoXY(J,21);  Write('.');
               J := J + 1;
               I := 0;
            end;
            I := I + 1;

            Read (RetrievedFile,Data);
```

```
         until (EoF(RetrievedFile));
      end;
      Close(RetrievedFile);
      IsSpectra := true;
      {write name of file retrieved}
      GotoXY(59,13);  write(FileName);
      ClearCommWindow(6);
   end else begin
      IsSpectra := false;
      GotoXY(5,20);  Write('File not found.');
      ClearCommLine(17);
      Sound(700);  Delay(100);  NoSound;
   end;
   GotoXY(5,24);  Write('Press Function Key ...');
end;

PROCEDURE ScanData(FirstPoint:DataPointer;Params:ParamsRecord);
{this procedure is used to browse through data retrieved from any file
in x-y column format.  This was originally in Unit Analyze.   4/91}

var
   CurrentPoint : DataPointer;
   XStart,XEnd  : real;
   Max,Min      : real;
   X,Y          : real;
   Counter      : integer;
   Positioncounter : integer;
   FuncKey         : integer;
   Scale            : real;

begin
   ClearCommWindow(6);

   Scale  := 1.0;
   XStart := Params.XStart;
   Xend   := Params.Xend;
   Max    := Params.Max;
   Min    := Params.Min;

   DefineWorld (4, XStart, Max+(Max-Min)*0.05, XEnd, Min-(Max-Min)*0.05);
   SelectWorld (4);
   SelectWindow (7);

   GotoXY (5,23); Write ('<-- left     right -->                                                    ');
   GotoXY (5,24);
   Write ('Browsing');

 {Find the point at XStart position}
   CurrentPoint := FirstPoint;
   X := CurrentPoint^.Xvalue;
   Y := CurrentPoint^.Yvalue*Scale;
   while (X < Xstart) do begin
      X := CurrentPoint^.Xvalue;
      Y := CurrentPoint^.Yvalue*Scale;
      CurrentPoint := CurrentPoint^.NextPoint;
   end;
   DrawLine (X, Min-(Max-Min)*0.05, X, Y);
   GotoXY (25,24);
   Write ('X: ', X:8, '   Y: ', Y/Scale:12);

   PositionCounter := 1;

   repeat
      FuncKey := 1;
      if keypressed then GetFuncKey (FuncKey);

      if (FuncKey = 75) and (CurrentPoint <> FirstPoint) then begin
         SetColorBlack;
         DrawLine (X, Min-(Max-Min)*0.05, X, Y);
         SetColorWnite;
         DrawPoint (X, Y);

       {Go to the begining -- PositionCounter equals 1}
         CurrentPoint := FirstPoint;
         X := CurrentPoint^.Xvalue;
         Y := CurrentPoint^.Yvalue*Scale;
         while (X < Xstart) do begin
            X := CurrentPoint^.Xvalue;
            Y := CurrentPoint^.Yvalue*Scale;
            CurrentPoint := CurrentPoint^.NextPoint;
         end;

       {Advance upto one point behind the current position}
         PositionCounter := PositionCounter - 1;
         for counter := 2 to PositionCounter do
            CurrentPoint := CurrentPoint^.NextPoint;

         X := CurrentPoint^.Xvalue;
```

```
         Y := CurrentPoint^.Yvalue*Scale;
         if (X < XStart) then begin
            CurrentPoint := CurrentPoint^.NextPoint;
            X := CurrentPoint^.Xvalue;
            Y := CurrentPoint^.Yvalue*Scale;
            PositionCounter := PositionCounter + 1;
         end;

         DrawLine (X, Min-(Max-Min)*0.05, X, Y);
         GotoXY (25,24);
         Write ('X: ', X:8, '   Y: ',Y/Scale:12);
      end;


      if (((FuncKey = 77) and (CurrentPoint^.NextPoint <> nil))
       and (CurrentPoint^.NextPoint^.Xvalue <= XEnd)) then begin
         if (CurrentPoint <> FirstPoint) then begin
            SetColorBlack;
            DrawLine (X, Min-(Max-Min)*0.05, X, Y);
         end;
         SetColorWhite;
         DrawPoint (X, Y);

         CurrentPoint := CurrentPoint^.NextPoint;
         X := CurrentPoint^.Xvalue;
         Y := CurrentPoint^.Yvalue*Scale;
         PositionCounter := PositionCounter + 1;

         DrawLine (X, Min-(Max-Min)*0.05, X, Y);
         GotoXY (25,24);
         Write ('X: ', X:8, '   Y: ', Y/Scale:12);
      end;

    until (FuncKey = 68);
    SetColorBlack;
    DrawPoint(X,Y);
    DrawLine (X, Min-(Max-Min)*0.05, X, Y);
    SetColorWhite;
    ClearCommWindow(6);
    GotoXY(5,24);  Write('Press Function Key ...');
end;


PROCEDURE ConvertIEDRecords (var DistFirstPoint:DataPointer; var Params:ParamsRecord;
                             var IsSpectra:boolean);
{This converts the old IED files from *.IE and *.DIS to ASCII files for plotting.
The new data is already saved as ASCII files under the file name *.IE}

Type
   DataRecord = record
      Xvalue  : real;
      Yvalue  : double;
   end;

Var
   DataFile    : text;
   IEFile      : File of DataRecord;
   DistFile    : File of DataRecord;
   Terminate   : boolean;
   DirInfo     : SearchRec;
   FirstPoint  : DataPointer;
   LastPoint   : DataPointer;
   DistLastPoint  : DataPointer;
   Data           : DataRecord;
   DistData       : DataRecord;
   Path           : Str60;
   N             :integer;

Begin
   ClearCommWindow(6);
   Terminate := false;
   GotoXY(5,17);  Write('Enter Path+Filename: ');
   GotoXY(10,18);  Write('(e.g. E:\JOANNE\J00000A) ');
   GetDescription(Path,26,17);
   FindFirst (Path+'.IE',AnyFile,DirInfo);
   If DosError <> 0 then Terminate := true;
   FindFirst (Path+'.DIS',AnyFile,DirInfo);
   If DosError <> 0 then Terminate := true;
   If not(Terminate) then begin
      GotoXY (5,18);  Write ('Converting data...                ');

      {Remove old data stored in datapointers and clear graph window}
      If IsSpectra then begin
         ClearGraphWindow;
         EraseData (FirstPoint);
      end;

      {Initialize variables}
```

```
      With Params do begin
         Max := 0.0;
         Min := 1000;
         FirstPoint               := nil;
          LastPoint               := nil;
          LastPoint^.NextPoint    := nil;
         DistFirstPoint           := nil;
         DistLastPoint            := nil;
         DistLastPoint^.NextPoint := nil;
      end;

      Assign (IEFile,Path+'.IE');
      Reset (IEFile);
      Assign (DistFile,Path+'.DIS');
      Reset (DistFile);

      repeat
         Read (IEFile,Data);
         Read (DistFile, DistData);
         If FirstPoint = nil then begin
            New (FirstPoint);
            LastPoint := FirstPoint;
            Params.XStart := Data.Xvalue;
            New (DistFirstPoint);
            DistLastPoint := DistFirstPoint;
         end else begin
            New (LastPoint^.NextPoint);
            LastPoint := LastPoint^.NextPoint;
            New (DistLastPoint^.NextPoint);
            DistLastPoint := DistLastPoint^.NextPoint;
         end;
         LastPoint^.Xvalue := Data.Xvalue;
         LastPoint^.Yvalue := Data.Yvalue;
         LastPoint^.NextPoint := nil;
         DistLastPoint^.Xvalue := DistData.Xvalue;
         DistLastPoint^.Yvalue := DistData.Yvalue;
         DistLastPoint^.NextPoint := nil;

         {Find the max. and min. values of the ion energy distribution}
         If Params.Max < DistData.Yvalue then Params.Max := DistData.Yvalue;
         If Params.Min > DistData.Yvalue then Params.Min := DistData.Yvalue;
         Params.XEnd := DistData.Xvalue;
      until (Eof(DistFile));
      Close (IEFile);
      Close (DistFile);

      {delete these record files from hard drive}
      Erase (IEFile);
      Erase (DistFile);

      {Store data on hard drive in 3 columns: Energy(eV),Current(Amps),Diff.(Amps)}
      Assign (DataFile, Path+'.IE');
      ReWrite (DataFile);
      LastPoint := FirstPoint;
      DistLastPoint := DistFirstPoint;
      Repeat
         Write (DataFile, LastPoint^.Xvalue:8:2,'  ',LastPoint^.Yvalue:13);
         Writeln(DataFile, '  ',DistLastPoint^.Yvalue:13);
         Lastpoint := Lastpoint^.Nextpoint;
         DistLastPoint := DistLastPoint^.NextPoint;
      until ((LastPoint = nil) or (DistLastPoint = nil));
      Close (DataFile);
      EraseData (FirstPoint);
      IsSpectra := true;
   end else begin
      GotoXY(5,20);  Write('File not found.');
      ClearCommLine (17);
      ClearCommLine (18);
      Sound(700);  Delay(100);  NoSound;
   end;
   GotoXY (5,18);  Write ('                          ');
   GotoXY(5,24);  Write('Press Function Key ...');
end;

Procedure SmoothData (var FirstPoint:DataPointer; var params:ParamsRecord);
Var
  Frac      : real;       {this is the fraction of the values adjacent to the
                           point being averaged}
  PrevPoint       : DataPointer;
  CurrentPoint    : DataPointer;
  NewFirstPoint   : DataPointer;
  NewCurrentPoint : DataPointer;

Begin
   Frac := 0.2;
   New (NewFirstPoint);
   With NewFirstPoint^ do begin
      Yvalue := FirstPoint^.Yvalue;
```

```
      Xvalue := FirstPoint^.Xvalue;
      NextPoint := nil;
   end;
   Params.Max := NewFirstPoint^.Yvalue;
   Params.Min := NewFirstPoint^.Yvalue;
   PrevPoint := FirstPoint;
   CurrentPoint := FirstPoint^.NextPoint;
   NewCurrentPoint := NewFirstPoint;
   repeat
      New (NewCurrentPoint^.NextPoint);
      NewCurrentPoint := NewCurrentPoint^.NextPoint;
      With NewCurrentPoint ^do begin
         Yvalue := Frac*PrevPoint^.Yvalue + (1-2*Frac)*CurrentPoint^.Yvalue
            + Frac*CurrentPoint^.NextPoint^.Yvalue;
         Xvalue := CurrentPoint^.Xvalue;
         NextPoint := nil;
         With Params do begin
            If Yvalue > Max then Max := Yvalue;
            If Yvalue < Min then Min := Yvalue;
         end;
      end;
      PrevPoint := CurrentPoint;
      CurrentPoint := CurrentPoint^.NextPoint;
   until (CurrentPoint^.NextPoint = nil);
   New (NewCurrentPoint^.Nextpoint);
   NewCurrentPoint := NewCurrentPoint^.NextPoint;
   With NewCurrentPoint^ do begin
      YValue := CurrentPoint^.YValue;
      XValue := CurrentPoint^.XValue;
      NextPoint := nil
   end;
   EraseData(FirstPoint);
   FirstPoint := NewFirstPoint;
   GotoXY(5,18);  Write('Max = ',Params.Max:10:6,'    Min = ',Params.Min:10:6);
end;

Procedure SavePIEData(FirstPoint:datapointer;Z:real;var Path:str60);
{saves data in 3 columns: Z,X,Y}
Var
   FileName     : Str60;
   TextFile     : Text;
   CurrentPoint : dataPointer;

Begin
   ClearCommWindow(6);
   GotoXY(5,17);  Write('ENTER Filename:');
   GetDescription(FileName,21,17);
   GotoXY(5,18);  Write(Path);
   GotoXY(5,19);  Write('Enter path:');
   GetDescription(Path,17,19);
   GotoXY(17,19);  Write(Path);
   Assign(TextFile,Path+FileName);
   ReWrite(TextFile);
   GotoXY(5,20);  Write ('Saving data...');
   CurrentPoint := FirstPoint;
   Repeat
{     With CurrentPoint^ do Writeln (TextFile,Z:5:2,'  ',XValue:12,'  ',YValue:12);}
      With CurrentPoint^ do Writeln (TextFile,XValue:12,'  ',YValue:12);
      CurrentPoint := CurrentPoint^.NextPoint;
   until (CurrentPoint = nil);
   Close(TextFile);
   ClearCommWindow(6);
   GotoXY(5,24);  Write('Press Function Key ...');
end;

Procedure AveragePIEData (var FirstPoint:datapointer;var Params:ParamsRecord);
Var
   CurrentPoint      : datapointer;
   NewPoint          : datapointer;
   FirstNewPoint     : datapointer;
   Terminate         : boolean;
   NumPoints         : integer;

Begin
   Terminate := false;
   NumPoints := 0;
   CurrentPoint := FirstPoint;
   New(Newpoint);
   With NewPoint^ do begin
      Yvalue := (CurrentPoint^.Yvalue + CurrentPoint^.NextPoint^.Yvalue)/2;
      Xvalue := CurrentPoint^.Xvalue;
      NextPoint := nil;
      With Params do begin
         Max := Yvalue;
         Min := Yvalue;
      end;
   end;
   FirstNewPoint := NewPoint;
```

```
    CurrentPoint := CurrentPoint^.NextPoint;
    CurrentPoint := CurrentPoint^.NextPoint;
    repeat
       NumPoints := NumPoints + 1;
       New(NewPoint^.NextPoint);
       NewPoint := NewPoint^.NextPoint;
       With NewPoint^ do begin
          Yvalue := (Currentpoint^.Yvalue + CurrentPoint^.NextPoint^.Yvalue)/2;
          Xvalue := CurrentPoint^.Xvalue;
          NextPoint := nil;
          With Params do begin
             If Yvalue > Max then Max := Yvalue;
             If Yvalue < Min then Min := Yvalue;
          end;
       end;
       CurrentPoint := CurrentPoint^.NextPoint;
       If CurrentPoint <> nil then begin
          If CurrentPoint^.NextPoint <> nil then
             CurrentPoint := CurrentPoint^.NextPoint
          else begin
             New(NewPoint^.NextPoint);
             NewPoint := NewPoint^.NextPoint;
             With NewPoint^ do begin
                Yvalue := CurrentPoint^.Yvalue;
                Xvalue := CurrentPoint^.Xvalue;
                NextPoint := nil;
                With Params do begin
                   If Yvalue > Max then Max := Yvalue;
                   If Yvalue < Min then Min := Yvalue;
                end;
             end;
             Numpoints := NumPoints + 1;
             Terminate := true;
          end;
       end else Terminate := true;
    until (Terminate);
    Params.XEnd := NewPoint^.Xvalue;
    EraseData (FirstPoint);
    Firstpoint := firstNewPoint;
    ClearCommLine(20);
    gotoXy(5,20);  Write ('Number of data points = ',NumPoints);
end;


BEGIN
    InitGraphic;
    ClearScreen;
    SetUpWindows;
    WriteScreen;
    IsExit := false;
    IsSpectra := false;
    Path := 'E:';

    Repeat
       FuncKey := 1;
       If keypressed then GetFuncKey (Funckey);
       Case FuncKey of

          {F1=Retrieve Data}
          59:Begin
             If IsSpectra then ClearGraphWindow;
             GetData(FirstPoint,2,Params,IsSpectra,Path);
             If IsSpectra then
                With Params do DrawSpectra(FirstPoint,Max,Min,XStart,XEnd,7);
          end;

          {F2=Scan Data}
          60:Begin
             If IsSpectra then begin
                 ScanData(FirstPoint,Params);
                WriteScreen;
                With Params do DrawSpectra(FirstPoint,Max,Min,XStart,XEnd,7);
             end else begin
                Sound(700);  Delay(200);  NoSound;
                GotoXY(5,17);  write('No Spectra!');
             end;
          end;

          {F3=Convert IED File}
          61:Begin
             ConvertIEDRecords (FirstPoint,Params,IsSpectra);
             If IsSpectra then
                With Params do DrawSpectra(FirstPoint,Max,Min,XStart,XEnd,7);
          end;

          {F4 = Smooth Data}
          62:Begin
             If IsSpectra then begin
```

```
            SmoothData(FirstPoint,Params);
            ClearGraphWindow;
            With Params do DrawSpectra(FirstPoint,Max,Min,XStart,XEnd,7);
         end else begin
            Sound(700);  Delay(200);  NoSound;
            GotoXY(5,17);  Write ('No Spectra!');
         end;
      end;

      {F5 = Save PIE data}
      63:If IsSpectra then SavePIEData (FirstPoint,2,Path);

      {F6 = Average Data}
      64:Begin
         If IsSpectra then begin
            AveragePIEData(FirstPoint,Params);
            ClearGraphWindow;
            With Params do DrawSpectra(FirstPoint,Max,Min,XStart,XEnd,7);
         end else begin
            Sound(700);  Delay(200);  NoSound;
            GotoXY(5,17);  Write ('No Spectra!');
         end;
      end;

      65:Begin
         GetWaveform (FirstPoint,Path,Params,IsSpectra);
         If IsSpectra then begin
            ClearGraphWindow;
            With Params do DrawSpectra(FirstPoint,Max,Min,XStart,XEnd,7);
         end else begin
            Sound(700);  Delay(200);  NoSound;
            GotoXY(5,17);  Write ('No Spectra!');
         end;
      end;


      {F10=Exit}
      68: IsExit:= true;

   end;  {Case}
until(IsExit);
EraseData(FirstPoint);
LeaveGraphic;
END.
```

```
Program IonBeam;

{The ion beam from the ion sampling orifice spreads radially because
 of space charge effects.  This program calculates the beam size as a
 function of distance from the orifice, z.  The beam is assumed to start
 at point z=0 with no diverging or converging component.  The equation
 for this calculation can be found in THE PHYSICS OF MICROFABRICATION by
 Brodie and Muray, 1982, p. 143, equation 2.84.  This equation is
 integrated using the trapezoid rule.}

Uses Crt;

Var
   V       : double;    {the intial energy of the ion in the z direction}
                        {given in volts}
   M       : double;    {mass of ion given in Kg}
   J       : double;    {ion flux in Amp/m^2}
   Ro      : double;    {beam diameter in meters}
   Orifice: double;     {orifice radius in meters}
   n       : integer;   {number of orifices}
   Er      : double;    {Electric field in the radial direction}
   Sum     : double;    {integral sum}
   k       : double;    {constant from equation}
   deltaR  : double;      {distance between calculated points}
   stepsize : double;     {number of steps used in the integration in}
                        {the z direction}
   R       : double;    {radial distance variable, non-dimensionalized by}
                        {Ro, the orifice radius}
   z       : double;      {axial distance from orifice}
   TotalZ    : double;   {the total distance in the z direction}
   I       : integer;   {counter used to keep track of number of steps taken}
   count   : integer;   {keep track of number of steps taken per R value}
   x       : double;    {integration variable in R direction}
   FileName  : string;  {name of file containing generated data}
   DataFile  : text;    {file for storing z,r coordinates of the ion beam}
   Reply     : char;

BEGIN
   repeat
     Write ('Ion Energy (eV)     : '); Readln (V);
     Write ('Ion Mass (kg/mole)  : '); Readln (M);
     M    := M/6.02E23;
     Write ('Ion Flux (A/m^2)    : '); Readln (J);
     Write ('Ion Beam Radius (in): '); Readln (Ro);
     Ro   := 0.0254*Ro;
     Write ('Orifice Radius (in) : '); Readln (Orifice);
     Orifice := Orifice*0.0254;
     Write ('Number of Orifices  : '); Readln (n);
     k    := 9.99369E9*sqrt(J*n/V*sqrt(M/V))*Orifice/Ro;
     TotalZ := 0.025;    {in meters}

     {Open file to store data}
     Write ('Name of file to store data : '); Readln (FileName);
     Assign (DataFile, 'E:\joanne\ares\'+FileName);
     ReWrite (DataFile);

     {the equation to integrate is

                [ 1    dR
          z   = |  ----------
                ] k sqrt(ln R)

      the limits of integration are z=0 to z=0.025 meters}

   deltaR   := 0.1;
   {From boundary condition, the first set of points are}
   R := 1;
   z := 0;
   Er:= 9.98738E19*Ro*J/R*sqrt(M/V);
   Writeln (DataFile, '    Z                    R              E');
   Writeln (DataFile, z:10:7, R:15:8, Er:15:5);
   R := 1+deltaR;
   Er:= 9.98738E19*n*Orifice*Orifice/Ro*J/R*sqrt(M/V);
   Stepsize   := DeltaR/200;  {evaluate the function at 200 pts}
                              {for each R value in integration}

   {From boundary condition, z=0 at R=1 so no need to calculate this point}
   {Start calculation at R=1+StepSize}
   x := 1 + StepSize;
   Sum := 0;
   count := 1;
   repeat
      Sum := Sum + 1/sqrt(ln(x));
      x   := x + StepSize;
      count := count + 1;
   until (count = 200);
   Sum := Sum + 1/sqrt(ln(x))/2;
   Sum := Sum/k*StepSize;
```

```
    z := Sum;
{    Writeln ('Z = ', z:10:5, '  R = ', R*Ro:15:8, '  E = ', Er:15:5);}
    Writeln (DataFile, z:10:7, R:15:8, Er:15:5);
    I := 2;
    repeat
       R := x + deltaR;
       Er:= 9.98738E19*n*Orifice*Orifice/Ro*J/R*sqrt(M/V);
       count := 1;
       Sum := 1/sqrt(ln(x))/2;
       repeat
          sum := sum + 1/sqrt(ln(x));
          x := x + stepsize;
          count := count + 1;
       until (count = 200);
       Sum := sum + 1/sqrt(ln(x))/2;
       Sum := Sum/k*stepsize;
       z := sum + z;
{       Writeln ('Z = ', z:10:5, '  R = ', R:15:8, '  Er = ', Er:15:5);}
       Writeln (DataFile, z:10:7, R:15:8, Er:15:5);
       I := I + 1;
    until ((z > TotalZ) or (I > 5000));
    Close (DataFile);
    Writeln ('z = ', z:10:5, '  r/Ro = ', R:15:5);
    Write ('Start another calculation? ');  Reply := UpCase(readkey);
    Writeln (Reply);
    Writeln;
    until (Reply <> 'Y');
END.
```

```
PROGRAM Meas_p;
{This program calculates the pressure to set at the manometer for a given
plasma pressure and flow rate for argon flow and meter #4 only.  Also
specify the reactor used and the leak rate}

Uses Crt,Dos;

Var
   Ua,Ub,Uo     : real;          {conductances between chamber and pump}
   P1,Pnew,Pold : real;          {plasma and outside teflon pressure}
   F            : real;          {flow rate in mTorr-l/sec}
   LeakRate     : real;          {leak rate in mTorr-l/sec}

   Iterations   : integer;
   delP         : real;
   Reply        : char;
   Reactor      : char;       {specify one of the three pyrex chambers}


BEGIN
Repeat
   {Enter plasma pressure required for a given flow rate}
   Write('Plasma Pressure (mTorr): ');
   Readln(P1);
   Write('Flow Rate (V): ');
   Readln(F);
   Write('Leak Rate (mTorr-l/sec): ');
   Readln(LeakRate);
   F := F*657 + LeakRate;
   Write('Which Reactor ( A=3.0"  B=4.5"  C=6.0")?: ');
   Readln (Reactor);
   Reactor := UpCase (Reactor);

   Case Reactor of

      {Calculation for 3.0"D. electrode reactor}
      'A': begin
         Pold := P1;
         Uo   := 1.53753 + 2*0.0177628*P1;
         Iterations := 0;
         repeat
            Ub   := 1.920377 + 2*0.0329804*Pold;
            Ua   := 1/(1/Uo-1/Ub);
            Pnew := P1 - F/Ua;
            delP := abs((Pnew-Pold)/Pnew);
            Pold := Pnew;
            Iterations := Iterations + 1;
         until((delP < 0.001) or (Iterations > 100));
         If Iterations > 100 then
            Writeln ('CALCULATION DOES NOT CONVERGE')
         else Writeln ('Set PRESSURE to ', Pnew:5:1, ' mTorr');
      end;

      {Calculation for 4.5"D. electrode reactor}
      'B': begin
         Pold := P1;
         Uo   := 2.592857 + 2*0.01357446*P1;
         Iterations := 0;
         repeat
            Ub   := 2.98098 + 2*0.02163014*Pold;
            Ua   := 1/(1/Uo-1/Ub);
            Pnew := P1 - F/Ua;
            delP := abs((Pnew-Pold)/Pnew);
            Pold := Pnew;
            Iterations := Iterations + 1;
         until((delP < 0.001) or (Iterations > 100));
         If Iterations > 100 then
            Writeln ('CALCULATION DOES NOT CONVERGE')
         else Writeln ('Set PRESSURE to ', Pnew:5:1, ' mTorr');
      end;

      'C': begin
         Writeln ('This reactor has not been calibrated yet.');
      end;

      else Writeln ('Wrong key pressed.');
   end;

   Writeln ('Another calculation?');
   Reply := Upcase (readkey);
   Writeln;
Until (Reply <> 'Y');

END.
```

```
Program Flow;
{This program calculates the plasma pressure from a measured flow rate and
pressure outside chamber and leak rate}

Uses Dos, Crt;

Var
   Pnew, Pold, Pb  : real;       {pressure in mTorr}
   Uo, Ua, Ub      : real;       {conductance in l/sec}
   F               : real;       {flow in mTorr-l/sec}
   LeakRate        : real;       {mTorr-l/sec}
   del             : real;       {test for convergence}
   repetitions     : integer;
   reply           : char;
   Reactor         : char;       {for 3"D or 4.5"D}

BEGIN
   REPEAT
      {specify measure plasma pressure}
      Write ('Measured Pressure (mTorr): ');
      Readln (Pb);
      {Specify flow rate and leak rate of the system}
      Write ('Flow Rate (V): ');
      Readln (F);
      Write ('Leak Rate (mTorr-l/sec): ');
      Readln (LeakRate);
      F := F*657 + LeakRate;
      Write ('Which reactor  (A=3.0"  B=4.5"  C=6.0")?: ');
      Readln (Reactor);
      Reactor := UpCase (Reactor);

      Case Reactor of
         {Calculation for 3.0"D. electrode reactor}
         'A': begin
            Repetitions := 0;
            Pnew := Pb;
            Ub   := 1.920377+2*0.0329804*Pb;
            repeat
               Pold := Pnew;
               Uo   := 1.53753+2*0.0177628*Pnew;
               Ua   := 1/(1/Uo-1/Ub);
               Pnew := F/Ua + Pb;
               del  := abs((Pnew-Pold)/Pnew);
               Repetitions := Repetitions + 1;
            until ((del <= 0.001) or (Repetitions >= 100));
            If (Repetitions >= 100) then begin
               sound(1000);  Delay (50);  NoSound;
               Writeln ('* Calculation did not converge *');
               Writeln;
            end else begin
               {Output data to the screen}
               Writeln ('Convergence: ', del*100:5:2, '%');
               Writeln ('Plasma Pressure (mTorr) : ', Pnew:6:1);
            end;
         end;

         {Calculation for 4.5"D. electrode reactor}
         'B': begin
            Repetitions := 0;
            Pnew := Pb;
            Ub   := 2.98098 +0.04326028*Pb;
            repeat
               Pold := Pnew;
               Uo   := 2.592857+0.02714892*Pnew;
               Ua   := 1/(1/Uo-1/Ub);
               Pnew := F/Ua + Pb;
               del  := abs((Pnew-Pold)/Pnew);
               Repetitions := Repetitions + 1;
            until ((del <= 0.001) or (Repetitions >= 100));
            If (Repetitions >= 100) then begin
               sound(1000);  Delay (50);  NoSound;
               Writeln ('* Calculation did not converge *');
               Writeln;
            end else begin
               {Output data to the screen}
               Writeln ('Convergence: ', del*100:5:2, '%');
               Writeln ('Plasma pressure (mTorr) : ', Pnew:6:1);
            end;
         end;

         'C': begin
            Writeln ('This reactor has not been calibrated yet.');
         end;

         else Writeln ('Wrong key pressed');
      end;  {Case}
      Writeln;
      Writeln ('Another calculation? ');
```

```
      Reply := Upcase (Readkey);
   UNTIL (Reply = 'N');
END.
```

```
Program PPlot;

{This program takes stored *.IE or *.SPA or *.WAV record files and
converts them to *.CMD files for plotting on PROPLOT.  Graph of current
versus energy or optical emission versus distance or wavelength.}

USES Dos,Crt,Atools;

CONST
   Path2 = 'E:\JOANNE\PARAMS\PPLOT\';

TYPE
   ParameterRecord = record
      System       : SysRecord;
      Status       : StatusRecord;
      Power        : PowerRecord;
   end;

   OpticsParamsRecord = record
      PIEPar       : PIERecord;
      Spa          : SpatialScanRecord;
      Wav          : WavelengthScanRecord;
   end;

VAR
   Terminate    : boolean;
   Reply        : char;
   CommandFile  : text;       {Output file for proplot}
   Parameters   : ParameterRecord;
   ParameterFile: File of ParameterRecord;
   FName        : String[6]; {Name of *.sys file}
   FileFound    : boolean;
   Counter      : integer;  {number representation of Letter}
   Letter       : char;     {character of filename ending}
   DirInfo      : SearchRec;
   Count        : integer;  {# of graphs on a page}
   A            : char;     {apostrophe variable}
   Ext           : string[3];
   Ext1          : char;
   Path1         : string[40];
   OpticsParams : OpticsParamsRecord;
   OpticsFile    : File of OpticsParamsRecord;
   IonFile       : text;
   X,XX,Y        : real;

BEGIN
   A        := chr(39);    {defining an apostrophe}
   Path1 := 'E:\JOANNE\PARAMS\SF6\';
   ClrScr;
REPEAT
   Terminate := false;
   FileFound := false;
   Writeln('Storing command files for PROPLOT in ', Path2);
   Writeln;
   Repeat
      Writeln ('Data directory = ', Path1,' ? (Y/N) ');
      Reply := UpCase(Readkey);
      If (Reply <> 'Y') then begin
         Repeat
            Write('ENTER directory: ');
            Readln (Path1);
            Writeln ('Data directory = ', Path1,' ? (Y/N) ');
            Reply := UpCase(Readkey);
         Until (Reply='Y');
      end;
      Write ('Enter filename (e.g. J20427)  : ');
      Readln (FName);
      Write ('Enter number 1=SPA or 2=WAV or 3=IE for data file extension: ');
      Readln(Reply);
      Case Reply of
         '1':Ext:='SPA';
         '2':Ext:='WAV';
         '3':Ext:='IE';
         else Terminate := true;
      end;

      If not(Terminate) then begin
         Counter := 65;
         Letter  := Chr(Counter);
         Write ('Start with ', FName, Letter, '? ');
         Reply := UpCase(readkey);
         Writeln (Reply);
         If not (Reply = 'Y') then begin
            Write ('Start with which letter? (B,C,...) ');
            Readln (Letter);
            Letter := Upcase (Letter);
            Counter := ord(Letter);
         end;
```

```
      FindFirst (Path1+FName+Letter+'.'+Ext, AnyFile, DirInfo);
      If DosError <> 0 then begin
         Writeln ('File NOT FOUND!!!');
         Writeln ('   - to continue press any key.');
         Writeln ('   - to quit, press ESC');
         if readkey = chr(27) then Terminate := true;
      end else FileFound := true;
   end;
until (Terminate or FileFound);

If FileFound then begin
   Repeat
      Count := 1;
      Ext1 := Ext[1];
      Case Ext1 of
         'S':begin
               Assign (CommandFile,Path2+FName+Letter+'S.PP');
               Rewrite (CommandFile);
               Write (CommandFile,'Legend 5.5 7.5 Justify Center Font T Size 2 ');
               Writeln (CommandFile,'"SPATIAL EMISSION SCANS"');
               end;
         'W':begin
               Assign (CommandFile,Path2+FName+Letter+'W.PP');
               Rewrite (CommandFile);
               Write (CommandFile,'Legend 5.5 7.5 Justify Center Font T Size 2 ');
               Writeln (CommandFile, '"WAVELENGTH EMISSION SCANS"');
               end;
         'I':begin
               Assign (CommandFile,Path2+FName+Letter+'I.PP');
               Rewrite (CommandFile);
               Write (CommandFile,'Legend 5.5 7.5 Justify Center Font T Size 2 ');
               Writeln (CommandFile,'"ION ENERGY DISTRIBUTIONS"');
               end;
      end;

      Repeat
         Case Count of
            1:Writeln (CommandFile,'Set Window X 1.75 to 5.25, Y 4.5 to 6.75');
            2:Writeln (CommandFile,'Set Window X 6.5 to 10, Y 4.5 to 6.75');
            3:Writeln (CommandFile,'Set Window X 1.75 to 5.25, Y 1 to 3.25');
            4:Writeln (CommandFile,'Set Window X 6.5 to 10, Y 1 to 3.25');
         end;
         If Ext <> 'IE' then begin
            Assign (OpticsFile, Path1+FName+Letter+'.PIE');
            Reset (OpticsFile);
            Read (OpticsFile,OpticsParams);
            Close (OpticsFile);
         end;
         Assign (ParameterFile,Path1+FName+Letter+'.SYS');
         Reset (ParameterFile);
         Read (ParameterFile, Parameters);
         If (Ext <> 'IE') then begin
            Writeln (CommandFile,'Set Grid X Style Dots');
            Writeln (CommandFile,'Set Grid X On');
         end;
         Writeln (CommandFile,'Set Ticks Top Off');
         With Parameters.System do begin
            Write (CommandFile,'Title Top Font T Size 1.5 ');
            Write (CommandFile,A,GasName,' ');
            Write (CommandFile,Pressure:6:1,'mTorr ');
            Write (CommandFile,'d=',ElecSep:3:1,'cm D=');
            Writeln (CommandFile,UpElecDiam:3:1,'in. f=13.56MHz',A);
            Case Count of
               1:Write (CommandFile,'Legend 3.5 6');
               2:Write (CommandFile,'Legend 8.25 6');
               3:Write (CommandFile,'Legend 3.5 2.5');
               4:Write (CommandFile,'Legend 8.25 2.5');
            end;
            Writeln (CommandFile,' Justify Center Font TBI ',A,FName,Letter,A);

            If (Ext <> 'IE') then begin
               If (Ext='SPA') then begin
               Case Count of
                  1:Write (CommandFile,'Legend 3.5 6.85');
                  2:Write (CommandFile,'Legend 8.25 6.85');
                  3:Write (CommandFile,'Legend 3.5 3.35');
                  4:Write (CommandFile,'Legend 8.25 3.35');
               end;
               Write (CommandFile,' Justify Center Font T Size 1.5 ');
               Write (CommandFile, A, 'Wavelength = ');
               Writeln (CommandFile,OpticsParams.Spa.ScanWavelength:7:1,' A',A);
               Write (CommandFile, 'Case ', A, '              ');
               Writeln (CommandFile,'        M',A);
               end;
            end;
         end;
         Close (ParameterFile);
         Case Ext1 of
```

```
            'S':begin
                  Writeln (CommandFile,'Title Bottom Font T Size 1.5 ',A,'Distance (cm)',A);
                  Writeln (CommandFile,'Title Left Font T Size 1.5 ',A,'Emission Intensity',A);
                end;
            'W':begin
                  Writeln (CommandFile,'Title Bottom Font T Size 1.5 ',A,'Wavelength (A)',A);
                  Writeln (CommandFile,'Title Left Font T Size 1.5 ',A,'Emission Intensity',A);
                end;
            'I':begin
                  Writeln (CommandFile,'Title Bottom Font T Size 1.5 ',A,'Energy (eV)',A);
                  Writeln (CommandFile,'Title Left Font T Size 1.5 ',A,'Fraction',A);
                end;
          end;
          Writeln (CommandFile,'Set Label Left On Format Exp Font T Size 1 Places 1');
          Writeln (CommandFile,'Set Limits YMIN 0');
          Case Ext1 of
             'S':Writeln (CommandFile,'Insert ',Path1,FName,Letter,'.SPA');
             'W':Writeln (CommandFile,'Insert ',Path1,FName,Letter,'.WAV');
             'I':begin
                   Assign (IonFile,Path1+FName+Letter+'.IE');
                   ReSet (IonFile);
                   repeat
                      Readln(IonFile, X,XX,Y);
                      Writeln (CommandFile, X:5:1,'  ',Y:14);
                   until (Eof(IonFile));
                   close (IonFile);
                 end;
          end;
          Writeln (CommandFile,'Join');
          If ((Ext='SPA') or (Ext='IE')) then Writeln (CommandFile,'Plot');
          Writeln (CommandFile);

          Count := Count + 1;
          Repeat
             Counter := Counter + 1;
             Letter := chr(Counter);
             FindFirst (Path1+FName+Letter+'.'+Ext, AnyFile, DirInfo);
             {When pass Z file, end program}
             If Letter >= chr(91) then Terminate := true;
          Until ((DosError=0) or Terminate);
        until ((Count > 4) or Terminate);
        Close (CommandFile);
        Write ('.');
      Until (Terminate);
   end;
   Writeln;
   Writeln ('Done!!!');
   Sound (700);  Delay(200);  NoSound;
   Writeln;
   Writeln ('Anymore files to process? (y/n) ');
   Reply := UpCase (Readkey);
UNTIL (Reply <> 'Y');
END.
```

```
Program Traj;

{This program calculates the ion trajectories in the ion analyzer in
 a magnetic field, using Cartesian coordinates.  The y-axis is parallel to
 the magnetic field and the x-axis is in the direction of the electric field
 vector the perpendicular to the y-axis.  z-axis is defined perpendicular to
 both the x and y-axis.  The output is change in the incidence angle as a
 function of distance from the origin of the spherical grids.}


USES Dos, Crt;

Type
   geometryrec = record
      beta     : real;              {original incidence angle}
      theta    : real;              {angle between B lines and projected E}
      xi       : real;              {angle between B and E}
      m        : double;            {ion mass in kilograms}
      l        : real;              {distance between grids}
      B        : real;
      V        : real;              {Second Grid Bias}
      Ex,Ey    : real;              {x and y component of Electric Field}
      Eo       : real;              {Initial ion energy}
   end;

   positionrec = record
      x,y,z,t  : double;
      xo,yo,zo : double;
   end;

   velocityrec = record
      vx,vy,vz : real;       {ion velocity}
   end;

   constantrec = record
      Cl       : real;              {integration constant}
      phi      : real;              {integration constant}
      vxo      : real;
      vzo      : real;
      vyo      : real;              {integration constant}
      A        : real;
   end;


Var
   geometry : geometryrec;
   position : positionrec;
   velocity : velocityrec;
   constant : constantrec;
   ElectricField1 : real;
   ElectricField2 : real;
   ElectricField3 : real;
   yt,zt,xt: double;

   Outfile         : text;
   FileName        : string;

   Reply           : char;
   Terminate       : boolean;
   deltat          : double;
   I               : integer;
   value           : real;
   p,n,newbeta,rho: real;
   distance        : double;
   tt : double;

Procedure FindTimeTravelled (constant:constantrec; geometry:geometryrec;
                             var velocity:velocityrec; var position:positionrec;
                             var Terminate:boolean);
{given l, the distance the ion needs to cross, find the time it takes.}

Var
   Delta     : double;
   TNTargetF : double;
   TNDerivF  : double;
   J         : integer;
   Guess     : double;
   Tolerance  : double;
   Iter       : integer;
   MaxIter    : integer;
   ax,ay,az   : double;
   p,n       : real;
   rho       : real;

Begin
   {Use Newton's method to find time needed to travel between grids}
   Guess := 1E-7;
   Tolerance := 1E-10;
```

```
    MaxIter   := 100;
    J := 1;
    With position,velocity,geometry,constant do begin
       repeat
          If B <> 0.0 then begin
             x  := C1/A*(sin(A*guess+phi)-sin(phi)) + xo;
             vx := C1*cos(A*guess+phi);
             y  := A*Ey/B*guess*guess/2 + vyo*guess + yo;
             vy := A*Ey/B*guess+vyo;
             z  := C1/A*(cos(phi)-cos(A*guess+phi)) + Ex/B*guess + zo;
             vz := C1*sin(A*guess+phi) + Ex/B;
          end else begin
             vx := 1.6E-19*Ex/m*guess + vxo;
             x  := 1.6E-19*Ex/m/2*guess*guess + vxo*guess + xo;
             vy := 1.6E-19*Ey/m*guess + vyo;
             y  := 1.6E-19*Ey/m/2*guess*guess + vyo*guess + yo;
             vz := vzo;
             z  := vzo*guess + zo;
          end;
          TNDerivF  := 2*(x*vx + y*vy + z*vz);
          TNTargetF := x*x + y*y + z*z - 1*1;
          delta     := -TNTargetF/TNDerivF;
          guess := guess + delta;
          J := J + 1;
       until ((J > MaxIter) or (abs(delta) < Tolerance));
       Iter := J;
       If ((J > MaxIter) or (guess < 0.0)) then Terminate := true
       else begin

       {If the calculated time converges, check to make sure it is not for
        a time when the ion moves in the opposite direction from analyzer,
        i.e. in the negative n-direction or direction away from the surface.}
          t := guess;
          rho := arctan (sin(beta)*sin(theta)/cos(beta));
          x := C1/A*(sin(A*t+phi)-sin(phi)) + xo;
          z := C1/A*(cos(phi)-cos(A*t+phi)) + Ex/B*t + zo;
          n   := x*cos(rho)-z*sin(rho);
          If n < 0.0 then Terminate := true;
       end;

       {Update new position and velocity}
       If not(Terminate) then begin
          If B <> 0.0 then begin
             x  := C1/A*(sin(A*t+phi)-sin(phi)) + xo;
             vx := C1*cos(A*t+phi);
             y  := A*Ey/B*t*t/2 + vyo*t + yo;
             vy := A*Ey/B*t+vyo;
             z  := C1/A*(cos(phi)-cos(A*t+phi)) + Ex/B*t + zo;
             vz := C1*sin(A*t+phi) + Ex/B;
          end else begin
             vx := 1.6E-19*Ex/m*t + vxo;
             x  := 1.6E-19*Ex/m/2*t*t + vxo*t + xo;
             vy := 1.6E-19*Ey/m*t + vyo;
             y  := 1.6E-19*Ey/m/2*t*t + vyo*t + yo;
             vz := vzo;
             z  := vzo*t + zo;
          end;
       end;
    end;

    Writeln('Iteration = ', Iter);
    Writeln ('t = ', position.t);
end;

Procedure NewTrajectory (var velocity:velocityrec; var constant:constantrec;
                         geometry:geometryrec; var position:positionrec;
                         var Terminate:boolean);
Var
    distance : double;
    p,n      : real;
    rho      : real;
    newbeta  : real;
    deltabeta: real;
    I        : integer;
    deltat   : double;


Begin
    With position, velocity, geometry, constant do begin
       xo := x;
       yo := y;
       zo := z;
       If B <> 0.0 then begin
          phi := arctan((vz-Ex/B)/vx);
          C1  := vx/cos(phi);
          vyo := vy;
       end else begin
          vxo := vx;
```

```
            vyo := vy;
            vzo := vz;
         end;
      end;
      FindTimeTravelled (constant,geometry,velocity,position,Terminate);
      With geometry do rho := arctan (sin(beta)*sin(theta)/cos(beta));
      If not(Terminate) then begin
      With position,geometry,constant do begin
         {Write data in output file in (x,y,z) coordinates}
         deltat := t/10.0;
         I := 1;
         If B <> 0.0 then begin
            repeat
               t := I*deltat;
               x := C1/A*(sin(A*t+phi)-sin(phi)) + xo;
               y := A*Ey/B*t*t/2 + vyo*t + yo;
               z := C1/A*(cos(phi)-cos(A*t+phi)) + Ex/B*t + zo;
               distance := sqrt(x*x + y*y + z*z);
               p    := x*sin(rho)+z*cos(rho);
               n    := x*cos(rho)-z*sin(rho);
               newbeta := arctan(sqrt(p*p+y*y)/n);
               deltabeta := newbeta - beta;
               Writeln (OutFile, distance, '  ', deltabeta*180/Pi:10:6);
               I := I + 1;
            until (I > 10);
         end else begin
            repeat
               t:= I*deltat;
               x := 1.6E-19*Ex/m/2*t*t + vxo*t + xo;
               y := 1.6E-19*Ey/m/2*t*t + vyo*t + yo;
               z := vzo*t + zo;
               distance := sqrt(x*x + y*y + z*z);
               p := x*sin(rho)+z*cos(rho);
               n := x*cos(rho)-z*sin(rho);
               newbeta := arctan(sqrt(p*p+y*y)/n);
               deltabeta := newbeta - beta;
               Writeln (OutFile, distance, '  ', deltabeta*180/Pi:10:4);
               I := I + 1;
            until (I > 10);
         end;
      end;
      end;
End;

BEGIN
   ClrScr;
   geometry.m  := 0.04/6.02E23;                    {ion mass}

Repeat
   Terminate := false;

   With geometry do begin
      Write ('Name of output file: ');
      Readln (FileName);
      Write ('Ion Energy (eV): ');
      Readln (Eo);
      Write ('Second Grid Bias (V): ');
      Readln (V);
      ElectricField1 := -V/0.0254/0.177;
      ElectricField2 := (V+50)/0.0254/0.135;
      ElectricField3 := -50/0.0254/0.209;
      Write ('Magnetic Field (Tesla): ');
      Readln (B);
      Write ('Incident Angle (deg): ');
      Readln (beta);
      beta := beta*Pi/180.0;
      Write ('Angle between B-field and projected ion direction (deg): ');
      Readln (theta);
      theta := theta*Pi/180.0;
      value := sin(beta)*cos(theta);
      If value = 0.0 then Xi := Pi/2.0
         else Xi := arctan (sqrt(1/value/value-1));
      constant.A  := 1.6E-19*B/m;
   end;

   Assign (OutFile, 'E:\JOANNE\ARES\MAGNET\'+FileName);
   ReWrite (OutFile);

   {Region 0:  no electric field}
   With position,geometry,velocity do begin
      x   := 0.0;
      y   := 0.0;
      z   := 0.0;
      vx  := sqrt(2*Eo*1.6E-19/m) * sin(Xi);
      vy  := sqrt(2*Eo*1.6E-19/m) * cos(Xi);
      vz  := 0.0;
      Ex  := 0.0;
      Ey  := 0.0;
```

```
      l   := 0.474*0.0254;
      Writeln (OutFile, 0.0, '  ', 0.0);
      t := 0.0;
   end;
   Writeln('Region 0');
   NewTrajectory (velocity,constant,geometry,position,Terminate);
   With position do Writeln ('distance = ', sqrt(x*x+y*y+z*z));

   {Region I: between Grid#1 and Grid#2}
   With geometry,velocity,constant do begin
      Ex  := ElectricField1*sin(Xi);
      Ey  := ElectricField1*cos(Xi);
      l   := 0.651*0.0254;
   end;
   Writeln;
   Writeln ('Region I');
   NewTrajectory (velocity,constant,geometry,position,Terminate);
   With position do Writeln ('distance = ', sqrt(x*x+y*y+z*z));
   If not (Terminate) then begin
      {Region II: between Grid#2 and Grid#3}
      with geometry do begin
         Ex  := ElectricField2*sin(Xi);
         Ey  := ElectricField2*cos(Xi);
         l   := 0.786*0.0254;
      end;
      NewTrajectory (velocity,constant,geometry,position,Terminate);
   With position do Writeln ('distance = ', sqrt(x*x+y*y+z*z));

      {Region III: between Grid#3 and Collecting Plate}
      With geometry do begin
         Ex  := ElectricField3*sin(Xi);
         Ey  := ElectricField3*cos(Xi);
         l   := 0.995*0.0254;
      end;
      NewTrajectory (velocity,constant,geometry,position,Terminate);
      With position do Writeln ('distance = ', sqrt(x*x+y*y+z*z));

   end else begin

      {for ions that do not make it past grid #2}
      Sound(500);  Delay (200);  NoSound;
      {Write data in output file in (x,y,z) coordinates}
      deltat := 5E-8;
      I := 1;
      With position,geometry,constant do begin
         rho := arctan (sin(beta)*sin(theta)/cos(beta));
         repeat
            t := I*deltat;
            If B <> 0.0 then begin
               x := C1/A*(sin(A*t+phi)-sin(phi)) + xo;
               y := A*Ey/B*t*t/2 + vyo*t + yo;
               z := C1/A*(cos(phi)-cos(A*t+phi)) + Ex/B*t + zo;
            end else begin
               x := 1.6E-19*Ex/m/2*t*t + vxo*t + xo;
               y := 1.6E-19*Ey/m/2*t*t + vyo*t + yo;
               z := vzo*t + zo;
            end;
            distance := sqrt(x*x + y*y + z*z);
            p := x*sin(rho)+z*cos(rho);
            n := x*cos(rho)-z*sin(rho);
            newbeta := arctan(sqrt(p*p+y*y)/n);
            Writeln (OutFile, distance,'  ',(newbeta-beta)*180/Pi:10:6);
            I := I + 1;
         until (I > 40);
      end;
   end;

   Close (OutFile);
   Writeln ('Another calculation? (y/n)');  Reply := UpCase(ReadKey);
   Writeln;
until (Reply <> 'Y');
END.
```