DESIGN AND IMPLEMENTATION OF A QUADRUPEDAL ROBOT

by

Daniel John DiLorenzo

SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL ENGINEERING AND COMPUTER
SCIENCE IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1987

Signature of Author _____
Department of Electrical Engineering and Computer Science
June 1, 1987

Certified by _____
Asst. Professor William Durfee, PhD.
Thesis Supervisor

Accepted by _____
Professor Leonard A. Gould
Chairman, Undergraduate Thesis Committee

DESIGN AND IMPLEMENTATION OF A QUADRUPEDAL ROBOT

by

Daniel John DiLorenzo

Submitted to the Department of Electrical Engineering and Computer Science on June 1, 1987 in partial fulfillment of the requirements for the degree of Bachelor of Science.

Abstract

This thesis encompasses the development of a quadrupedal robot from design through successful implementation. This project spans several disciplines, each of which is focused upon seperately in this writing. This project begins with the design and construction of the mechanical component, a four-legged electrically powered machine, possessing twelve degrees of freedom. The next phase involves the development of necessary power control and delivery circuitry, including a twelve channel 8-bit pulse width modulator and twelve channel amplifier with a net power capacity of 3 kilowatts. Two levels of software are developed. The bottom software level is the implementation of a motion dynamics controller for the twelve motors. The top software level controls the robot gait, or walk sequence. The various design goals and constraints are discussed, the performance of the numerous subsystems and the overall system is evaluated, and possibilities for future improvement are explored.

Thesis Supervisor: Asst. Professor William Durfee, PhD.
Title: Professor of Mechanical Engineering

Dedication

To my parents for their unending encouragement and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of the Field

The developing field of robotics, which has matured within the last decade, is already branching into distinct fields with highly specialized applications. Industrial robotics is the most mature of these disciplines, capturing a sizable and growing market in automated manufacturing. Self navigating mobile robots are making the transition from research and development into an industry targeting the business markets of internal parcel delivery and factory automation [Waber 84]. Robots have been put to use in the medical field, ranging from such applications as automated prosthetics systems [Bak 86] to devices for guiding the hand of a neurosurgeon [Propper 86].

The latest robotics offshoot is the field of walking machines, offering the possibility of a highly adaptable all terrain-vehicle. Several research institutions, large corporations, and the military have already taken an interest in this area of robotics [Science News 85] and [Drogin 86]. Such possibilities being explored by the

military include unmanned combat vehicles [Fulsang 85]. The oil industry is involved in the development of undersea mobile robots for the maintenance of offshore oil platforms [Murphy 84]. The development of a walking all terrain vehicle has obvious military and commercial applications and may produce spinoffs in the consumer marketplace.

1.2 Present State of the Art

Several research institutions and large companies have undertaken projects in walking robotics, having successfully built legged machines, each exhibiting a particular technique of locomotion. A six-legged crawling machine built at Ohio State University represents one of the most advanced designs targeting the application of all-terrain navigation. This device is entirely automatic, requiring only supervisory user input to facilitate long range navigation [McGhee 86]. Still further from actual application are a collection of hopping machines built at Carnagie Mellon University under Marc Raibert. These include a monoped, a biped, and a quadruped [Raibert 83]. Although still in their developmental stages, these projects show promise in the production of a practical all-terrain legged vehicle.

1.3 Scope of this Project

This thesis project includes all aspects of the design and implementation of an electrically powered four legged walking robot. This project encompasses mechanical engineering, electrical engineering, and computer science as well as basic control theory and study of animal walking gaits. This project undertakes the goal of developing a successful walking machine in an effort to understand the inherent obstacles and explore possible solutions.

1.4 Breakdown into Subparts

The quadrupedal robot is divided into several subsystems. The first system to be designed and built is the mechanical componant, including the frame, twelve axes of movement, and an electrical drive system. The second system is electrical, consisting of a power regulation system and a power delivery system. The third system is the dynamic motor controller, implemented as the bottom level in the controlling software. The fourth system is the locomotion coordination implemented as the top level in the controlling software.

Chapter 2
Mechanical Design

2.1 Initial Design Goals

In designing a walking robot, I decided on a quadrupedal design as opposed to a bipedal or other design due to the statically stable gait possible with a four legged configuration. For a bipedal design, where the feet offer only two points of contact, only one dimension of stability, the vertical axis, is offered. In the process of walking, at least one leg must be able to break contact with the supporting surface and take a step. For a machine of N legs, this leaves at most N-1 legs to provide support and balance. Legged locomotion can be achieved through two basic schemes, incorporating either a statically stable or a dynamically stable gait. In a statically stable design, the robot's position is always uniquely defined by the positions of its legs. Static stability requires that three legs have a foothold at any given stage in the walking sequence. For (N-1)=3, a minimum of N=4 legs is required. This is the basis for the decision in favor of a quadrupedal design. Statically and dynamically stable gaits are further explored in Section 5.1 on page 40.

The mechanical design of the robot was divided into two main parts, a rigid supporting structure to function as the body and four reflection-symmetric legs. In the interest of preserving maximum flexibility of operation, an ambidexterous design was chosen. In this configuration, the body supports the legs such that symmetry along both horizontal axis is preserved. This nonpolarized design approach was selected to facilitate simplicity in the walking control algorithm.

2.2 Physical Implementation

Dimensions were chosen and a preliminary design for the body and legs was drawn up. In order for the robot to have the ability to walk forward and backward as well as be able to turn, legs possessing three degrees of freedom were required. These axis took the forms of a rotary hip, lateral hip and a knee, performing the functions of froward/backward rotation, abduction/adduction, and flexion/extension respectively. The rotary hip and knee have a range of movement of 180 degrees, while the lateral hip has one of 90 degrees. Several designs were drawn up and considered, and the final design is shown in figure 2-1.

To avoid the backlash inherent in a chain drive system, the gear motor is directly coupled to the rotary hip shaft. A similar approach also would have been

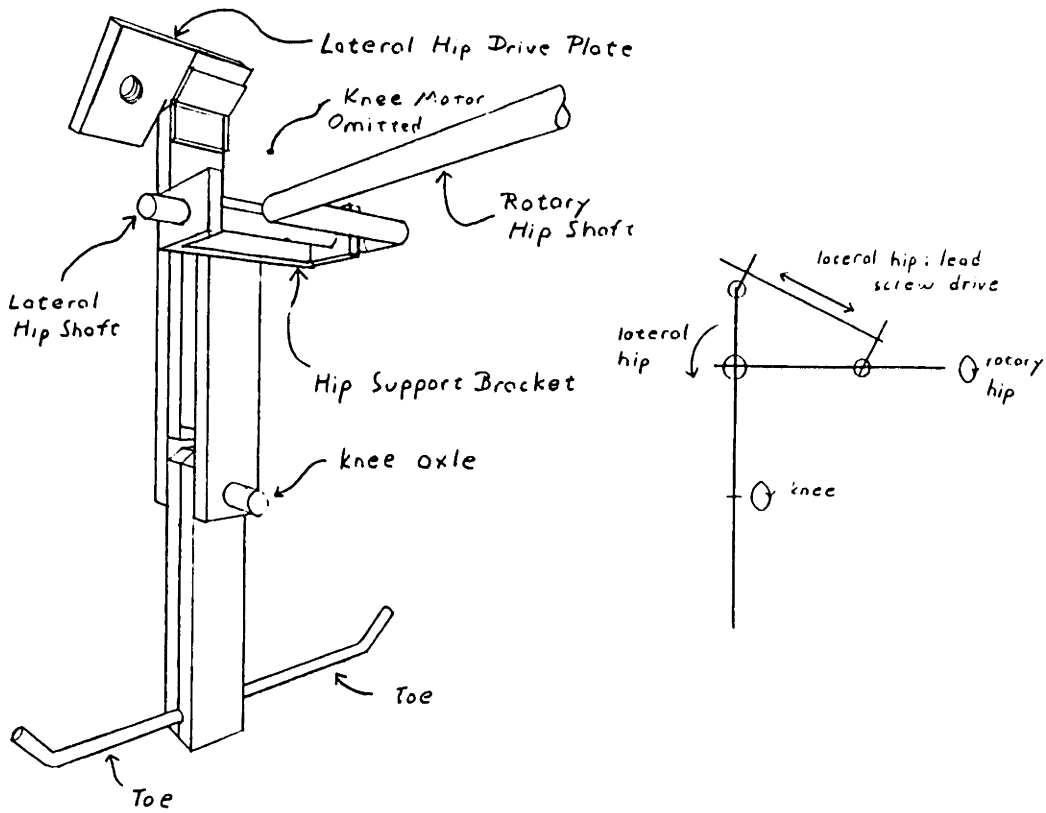


Figure 2-1: Mechanical Design for 3-axis Leg

possible for the knee joint, but this would have presented a much higher moment arm for the rotary hip to overcome. To reduce the load placed on the rotary hip, the knee drive motor is located above the rotery hip shaft, allowing the motor to function as a counterweight. Since robot motion is considerably less sensitive to knee movement than to rotary hip movement, the backlash characteristic of a chain drive is tolerable. The knee drive motor is linked to the knee axle via a .25 inch pitch chain employing a gear ratio of 13:30. The lateral hip design incorporates a lead screw to accomplish

abduction and adduction. Here again, a direct coupling was originally considered; however, this would have required a considerably higher torque motor and would have resulted in a bulkier design, possibly at the expense of the leg's range of movement.

The body of the robot takes the form of a rectangular frame with the four legs mounted symmetrically about the center. With the exception of two torsional reinforcement struts, the robot body and legs are symmetric with respect to the forward and lateral axis intersecting at the center of mass which is located at the center of the robot. Maintaining a center of mass at the intersection of the lines connecting the diagonally opposing legs is essential in a statically stable gait. If the center of mass is too far forward of the leg diagonal intersection (LDI), raising either of the front legs will result in a fall. The converse problem exists if the center of mass is too far backward of the LDI.

In the original leg design, the feet were points. In order for the robot to take a step forward, the robot would have to shift its weight backward far enough so that the center of mass was well within the triangle defined by the three legs that were to remain planted during the step. This weight shift was required to be greater than the weight shift resulting from the forward extension of the stepping leg. After preliminary testing, it was

determined that a more stable design would be achieved by placing long feet on the legs. Each foot consists of a 12 inch long section of .25 diameter stainless steel rod, the center of which is mounted to the leg bottom. The last 1 inch of each end of the foot is bent 45 degrees upward to avoid catching the foot on the ground while walking.

On the basis of the preliminary designs and specifying aluminum materials for weight minimization, the weight was determined to be safely under 60 lbs. From the specified dimensions and the calculated weight, motor torque requirements were computed. It was determined that the robot should possess sufficient torque capacity to support itself with all four legs held parallel and inclined at a 45 degree angle. Although in a normal walk sequence, the legs will never be angulated more than 45 degrees from vertical, a more stable system with better recovery from falls would have resulted from computing required torques for the legs stretched out horizontally, the most difficult collapse position from which the robot would have to recover. Reversible d.c. gear motors were selected to provide the required torques with an acceptable safety margin, and these torques are given in table 2-I on page 16.

To facilitate closed loop control, position feedback sensors are incorporated into the mechanical design. One turn linear taper potentiometers function as these

Axis of Movement	Required Torque	Available Torque
Rotary Hip	10.6 ft lbs	12.5 ft lbs
Lateral Hip	10.6 ft lbs	15.7 ft lbs
Knee	5.3 ft lbs	28.8 ft lbs

Table 2-I: Torque Requirements

sensors. The rotary hip position sensors are mounted to the robot frame and coupled to the shaft via a timing belt. The lateral hip design includes .25" of shaft protruding from the hip mounting bracket (see Figure 2-1 to which the potentiometer is mounted using a shaft coupling. Similarly, a the knee position sensors are mounted to extensions of the knee axle using shaft couplings. The potentiometers function as voltage dividers, producing an output voltage signal ranging from 0 to 5 volts.

Chapter 3
Electrical Design

3.1 Control Delegation

There are several levels of control involved in the automation of this robot. To facilitate development, the walking coordination algorithm is implemented in software. For the same reason, the motor controller is also implemented in software. The next lower level of control, motor power control, is accomplished in hardware.

3.2 On Board Motor Power Control

3.2.1 Design Goals and Constraints

There are a variety of techniques by which motor power can be controlled. The two particular schemes which were under consideration in this project were analog voltage control and pulse width modulation. Analog voltage control offered the advantage of design simplicity, involving little more than a latched digital to analog converter and a few op amps. Pulse width modulation offered more precise motor speed control at lower output speeds with less hysteresis, i.e. the motor will not begin to turn until a voltage threshold is passed

after which the motor may continue to rotate at voltages below the threshold [Kreuter 79]. Pulse width modulation could be accomplished in software but would put a severe computational burden on the computer, and might seriously impede controller performance by decreasing its response time. For this reason, in the final design, pulse width modulation is accomplished in hardware by the curcuit described in Section 3.2.2.

Analog voltage control was the first approach which was considered. The circuit in Figure 3-1 was constructed. A preliminary output stage circuit shown in Figure 3-2 was also designed and constructed. To assure reliable operation, this output stage was designed to handle 15 amps continuous duty, while the maximum individual motor requirements were 6-8 amps (at zero angular velocity) for a supply voltage of 12 volts. The circuit operated successfully, and motor speed control was as expected; however, this circuit exhibited poor power efficiency at intermediate motor speeds.

The power dissipated in the output stage power transistors was computed as a function of motor voltage (which represents the desired motor speed), and is shown in Figure 3-3. Since the motors would be driven at intermediate speeds for a substantial fraction of their operational cycle, the power inefficiency of this design presented a serious drawback. This robot incorporates

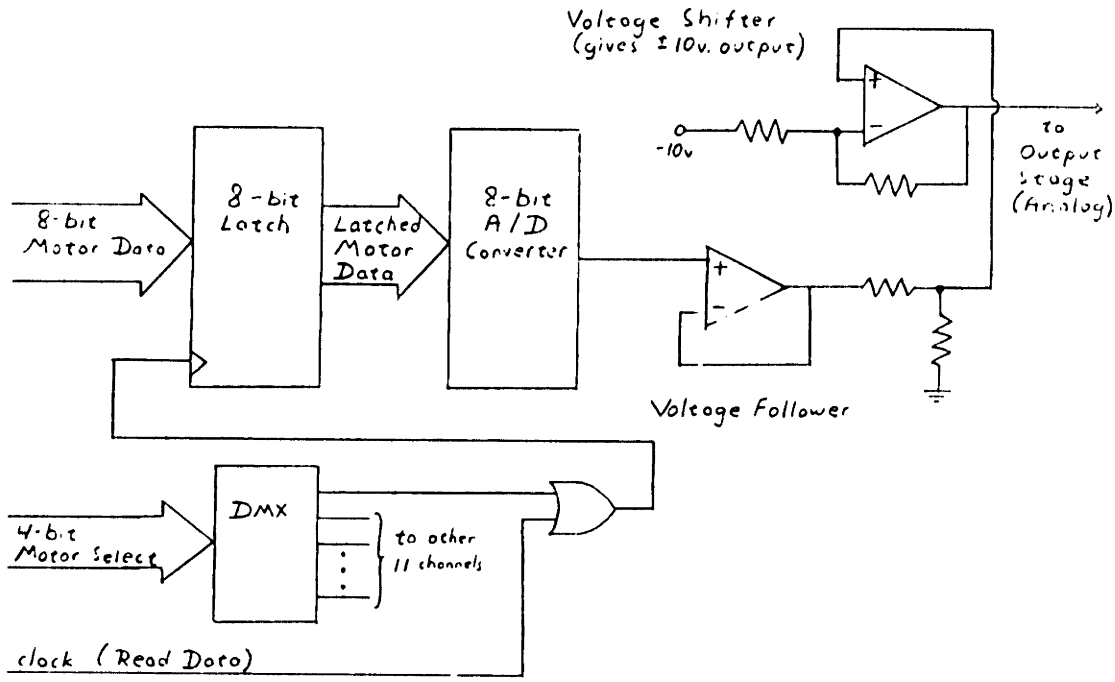


Figure 3-1: Motor Power Control: Analog Voltage Design

twelve motors, each possible of consuming approximately 70 watts, bringing the total power requirement to 1 kilowatt. An inefficient output stage would not only be in danger of overheating but would present an undesirably high load on a power supply system. For this reason, a pulse width modulator was chosen as a means of motor power regulation.

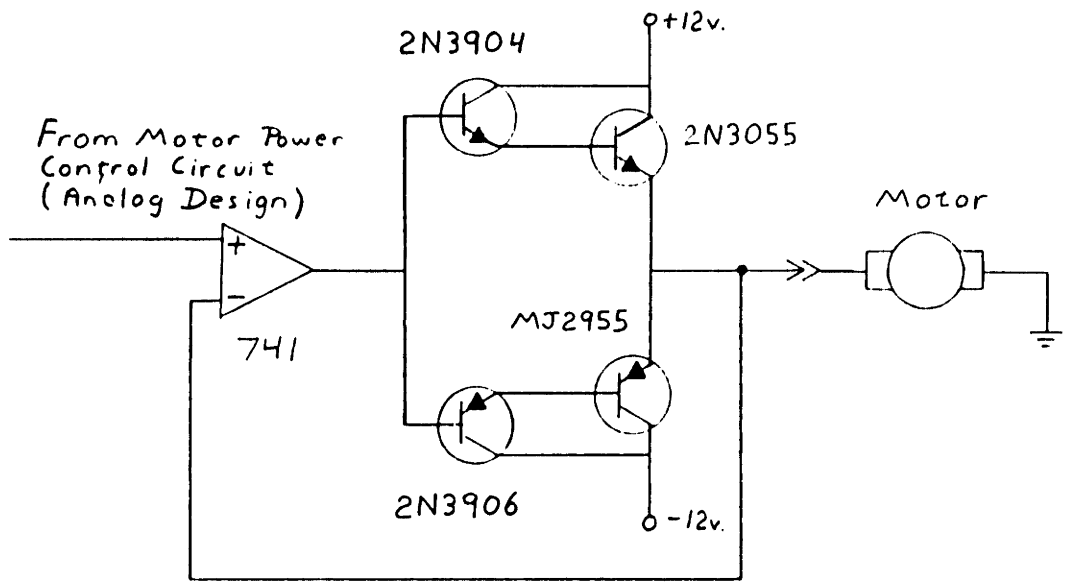


Figure 3-2: Output Stage: Analog Design

3.2.2 Pulse Width Modulator

After exploring several analog approaches, a digital pulse width modulater design was devised. A 4-bit prototype was tested and evaluated. As expected, the output stage exhibited a much greater power efficiency when driven by the pulse width modulator (PWM) than by the analog voltage design. At any given point in time, the PWM output is either fully on or off. In the on state, the power dissipated in each activated power transistor is Pd = (collector-emitter voltage) * (motor current) which is of the order Pd = (2v) * (3A) = 6 watts, considerably less than the power dissipation of the analog design.

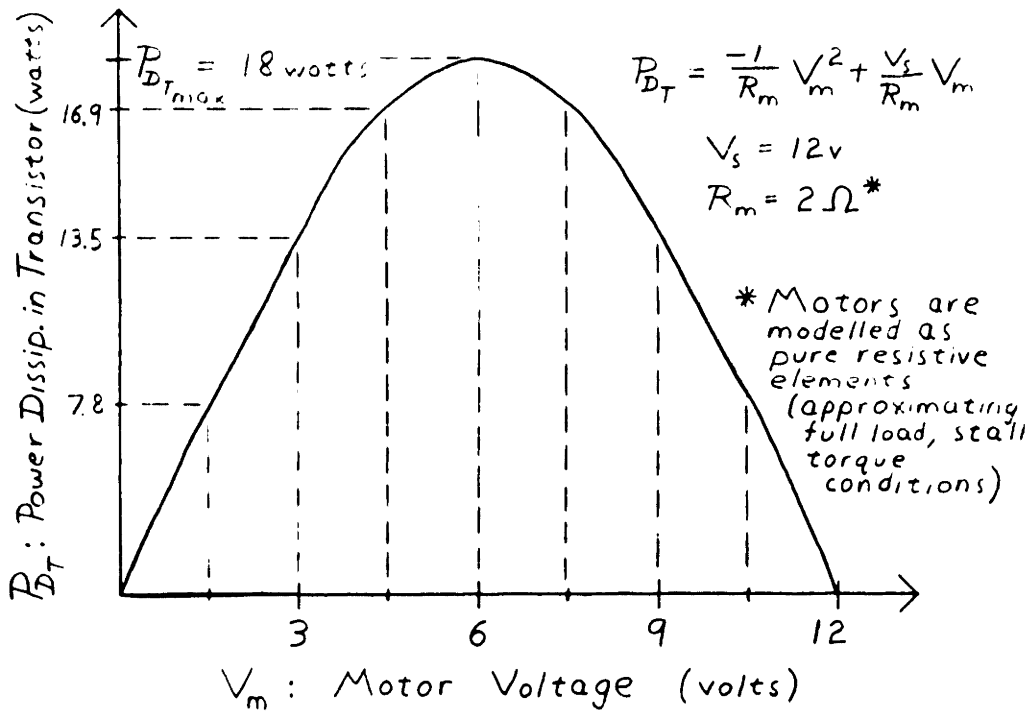


Figure 3-3: Analog Output Stage Power Efficiency

3.2.3 Circuit Implementation

The pulse width modulator circuit consists of twelve parallel channels, one for each motor. Each channel consists of a latching stage, a binary comparator stage, a counter which is common to several channels, and analog voltage conditioning circuitry. All channels are connected to a common data bus which is written to by the computer. The bus consists of 8 bits of motor speed data, two bits of motor direction data (forward, reverse, and two off states), four bits of motor addressing data, and one clock channel. The computer sequentially updates the

motor speeds, wich correspond to the fraction of the PWM time interval during which the output is on. To accomplish this, the computer outputs the address of the motor to be updated (0-11), its desired speed (0-255), a motor direction code (binary 00 to 11); and the data is read on a low to high transition of the clock line. The circuit diagram of the final pulse width modulator design is given in Figure 3-4, and is divided among three identical boards, the layout of which is given in Figure 3-5.

Although each board of the pulse width modulater possesses seperate clock circuitry, the circuit was laid out such that with minor modification, all three boards can be synchronously clocked. In case the high frequency power line spikes imposed by the PWM became a problem, the circuit design includes a feature enabling the user to switch between up and down counting modes; and an input is provided to synchronize these count cycles. It can be seen that in the up counting mode, the power consumed by the overall PWM circuit will be greater at the beginning of each count cycle than at the end. To produce a digital PWM output, the latched 8-bit motor speed value is compared to a reference value which repeatedly cycles through an 8-bit count. When the motor speed value is greater than the reference value, the PWM output is on; otherwise, the PWM output is off. For an up count cycle,

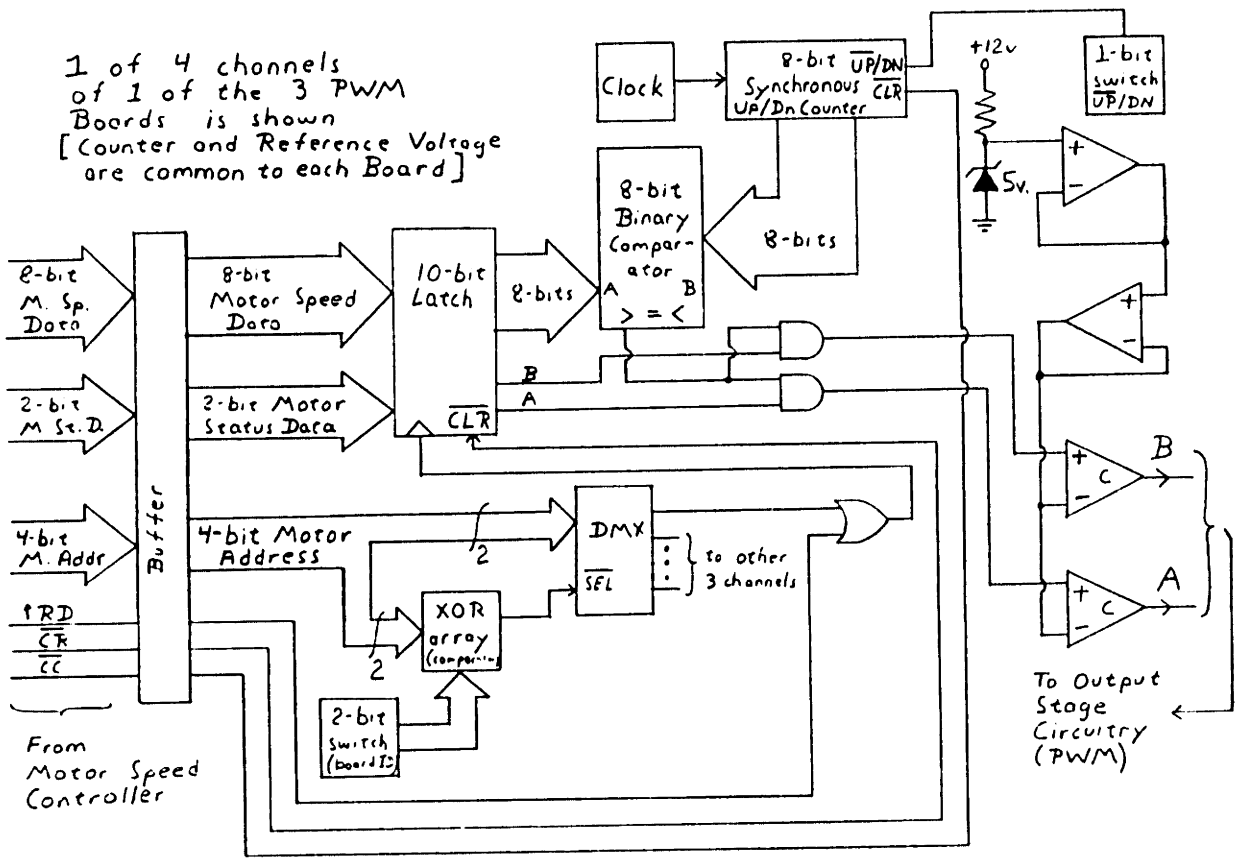


Figure 3-4: Pulse Width Modulator

this corresponds to the motor currents being on at the cycle beginning and off at the cycle end. For a down count cycle, the power consumption will be greater at the end of the count cycle. To more evenly distribute power consumption over the count cycle, the PWM board driving

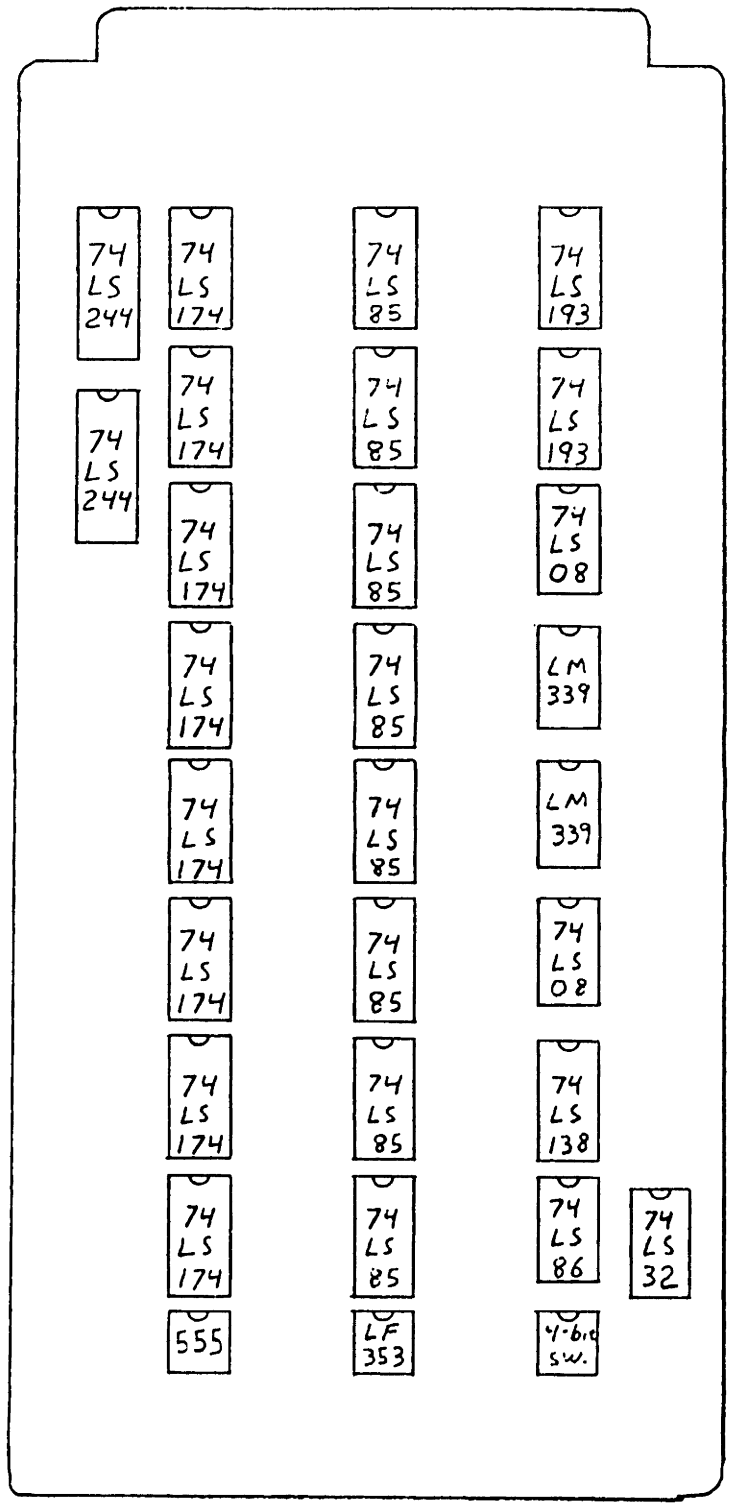
the output stage board with the highest load would count in one direction while the other two boards count in the other direction. For example, the rotary hip board can be set to count down, while the lateral hip and knee boards count up. Although this flexibility is incorporated into the PWM design, no power supply spiking problems were detected with unsynchronized counting; so the count synchronization and direction reversing features were not utilized.

3.3 Power Delivery

3.3.1 Design Goals and Constraints

Two separate power delivery circuits, hereafter referred to as output stages, were designed, corresponding to the two motor power control schemes. The first power delivery circuit was designed for analog voltage control. The most direct approach involved a "half H" configuration, using a complementary pairs of Darlington transistor pairs. An operational amplifier was used to maintain output linearity, provide high input impedance and isolate the power control circuitry from inductive motor voltage spikes. This circuit, shown in Figure 3-2, was breadboarded and then constructed on a printed circuit board, for which a foil pattern was made and etched. The op amp selected was a 741 for its ease of availability and

Figure 3-5: PWM Board Layout



relatively high current output. An additional drawback of a simple analog voltage control circuit becomes evident in the design of an output stage. A half H output stage configuration requires positive and negative supplies but can only supply one half of the difference in these supplies to the motor. A more complicated analog scheme could generate an inverted "complementary" voltage to drive another half H, resulting in a full H output stage. This design would be able to deliver twice the voltage as a half H configuration but would be more sensitive to calibration errors, especially at low motor voltages approaching zero since the complementary signal would have to meet the original signal at a common zero point. This would require precise offsets to insure that the motors are completely off for a zero input signal.

Upon selection of a pulse width modulator as a means of motor power control, a different set of constraints were presented into the design of the output stage circuitry. The PWM introduces an entirely different mode of operation for the output stage. The analog voltage control circuit operates in a linear manner, requiring a linear (ideally an identical) correlation between input and output voltages. The PWM circuit gives a binary (on or off) output. There are two significant differences between these modes. The PWM driven output stage does not require a linear correspondence between input and output

voltages since the motor is either fully on or off at any given point in time. In addition, the slew rate of the output stage becomes significant in accomplishing rapid on-off transitions.

3.3.2 Circuit Implementation

The breadboarded version of the PWM was tested with two channels of the circuit shown in Figure 3-2 in an "H" configuration. The output transition time occupied a small fraction of the total count cycle and prevented the output stage from delivering full on voltage for small motor power levels (short "on" pulses). The slew rate of the 741 op amp used in the first design was .5 volts per microsecond, requiring t = (20 volts) * (.5 volts/mse) = 10 microseconds to accomplish the 20 volt output swing required in the PWM design. The modified design, shown in Figure 3-6, incorporates an LM339 comparator with a slew rate of 13 volts/msec to achieve rapid on-off transitions. Since the input voltage from the PWM circuitry is either +10v or -10v, and no output voltage feedback is necessary, the inputs are connected to the +Vin channels and the -Vin channels are connected to ground.

Since the output of the LM339 Quad Voltage Comparator can source only 20 milliamps, as opposed to the 100 milliamps provided by the 741 op amp, an additional transistor gain stage is needed to provide the 6 to 8 amps

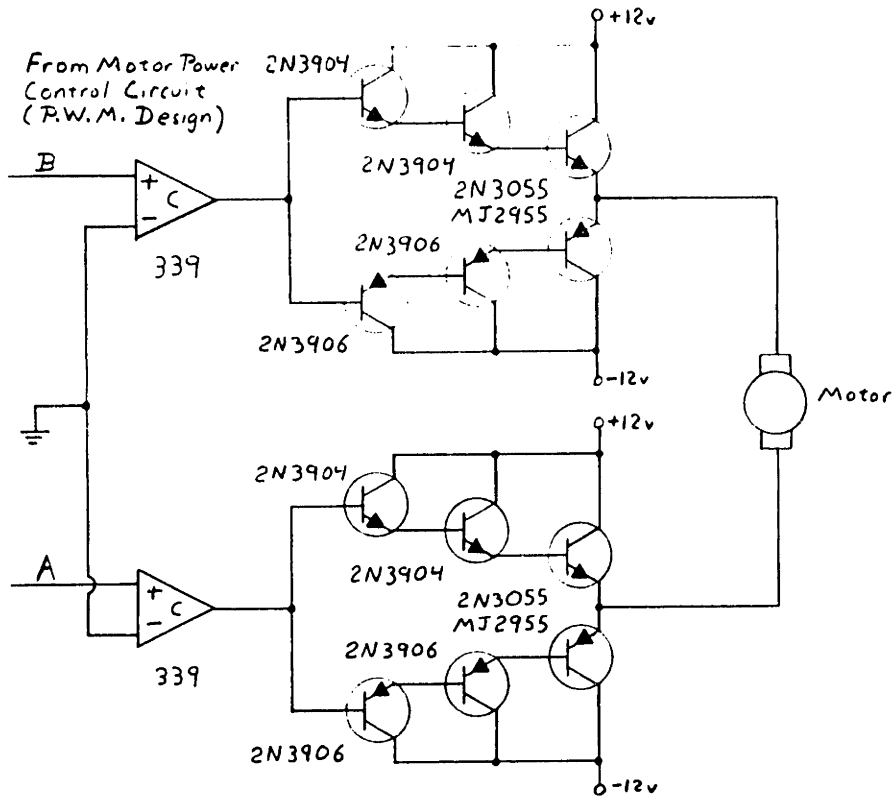


Figure 3-6: PWM Dedicated Output Stage

required per motor. The entire Pulse Width Modulater dedicated output stage occupies three 8" by 11" printed circuit boards which were drilled and etched. Each board drives four motors; PWM boards 1, 2, and 3 drive the four rotary hips, four lateral hips, and the four knees respectively. All three PWM boards were designed identically for three reasons. Identical design allowed for simultaneous drilling of the boards, considerably reducing construction time. It also allowed easy board swapping which proved useful in the debugging stage. Finally, in the event of failure of the rotary hip or knee

output stage boards, essential for forward and backward walking, replacement by the less used lateral hip board is provided for. All 6 quad comparators are socketed for easy replacement, and all 48 power transistors are mounted on heat sinks, eliminating the possibility of overheating during normal operation.

Chapter 4
Motor Control

4.1 Performance Goals and Constraints

The perormance goals in the development of this control system are not uncommon to those in most other control systems. This controller was to be of the closed-loop type, receiving angular position data in the form of analog voltages from the potentiometers coupled to each axis, described in Section 2.2. Minimization of settling time was a primary consideration as was minimization of overshoot. The walking speed is limited by the response time of the motor controller and was therefore a strong motivation in controller design. The magnitude of the overshoot and associated decaying oscillations imposes proportional stress on the mechanical linkages; these stresses can reduce the operational lifetime of the mechanical componants and require frequent maintenance, such as tightening of set screws, and should be minimized. It was realized in advance that a controller more complex than a simple proportional controller might be required to meet the performance goals. It was decided to develop a proportional

controller and evaluate its performance. From that evaluation, it would be determined how sophisticated a control algorithm was necessary.

4.2 Implementation and Evaluation

4.2.1 Proportional Controller

For the first iteration of controller development, a proportional controller was designed and implemented in C. A short stepping sequence was written with which to test the controller. The dynamic response of the robot axes were evaluated under two opposite conditions. With the robot suspended above the floor, system response was observed and overshoot and settling time were noted. The same stepping sequence was then repeated with the robot supporting itself on a level flat surface. This allowed no load and full load system response to be observed. This was necessary since each axis would experience conditions of zero to full load. The planted legs will be required to maintain balance by exerting forces necessary to support the 50 pound robot, and the stepping leg will be required to rapidly take a step under no load conditions, overcoming the limb's inertia and frictional forces.

After some experimentation with the proportional gain and its associated saturation point, values were

determined which gave satisfactory performance. Overshoot of approximately 5 degrees was achieved, and a typical response time for a 30 degree angular increment was reduced to just under 1 second. With this controller, a complete walking sequence was developed, allowing for a thorough evaluation of this controller. The robot successfully completed a 4-step walking sequence using the proportional controller supported by the QROBOT.C library in Appendix Section A.4 on page 64. The motor speed output of the controller is plotted versus the angular positional error in Figure 4-1. The maximum motor speed output is 255 "on" units, corresponding to 20 volts being applied to the motor for 255/256 of the PWM count cycle. The error corresponds to the difference between the desired motor position and the actual position determined from the digitized voltage of the corresponding analog feedback potentiometer. Both the desired position and the actual position variables have a range on the order of plus or minus 300. The actual range is dependent on the physical range of movement of the axis, 180 degrees for the rotary hip and knee and 90 degrees for the lateral hip. The analog feedback potentiometers have a range of 0 to 5 volts, and the analog to digital converter has 12 bits of resolution on a plus or minus 10 volt signal. This gives a resolution of r = (20v/4096count) = 4.9 millivolts per count. Since the feedback potentiometers

give a 5 volt range for a 270 degree angular span, an angular resolution of Ra = (270deg/5v) * (4.9mv/count) = .26 degrees for the error variable. The analog feedback voltages could easily be conditioned to fill the entire 20 volt span of the A/D module to provide greater resolution, but since a motor error colerance of 25, corresponding to an angular tolerance of Ta = (25*.26) = 6.6 degrees, is allowed in this controller implementation, a greater resolution would be entirely unnecessary.

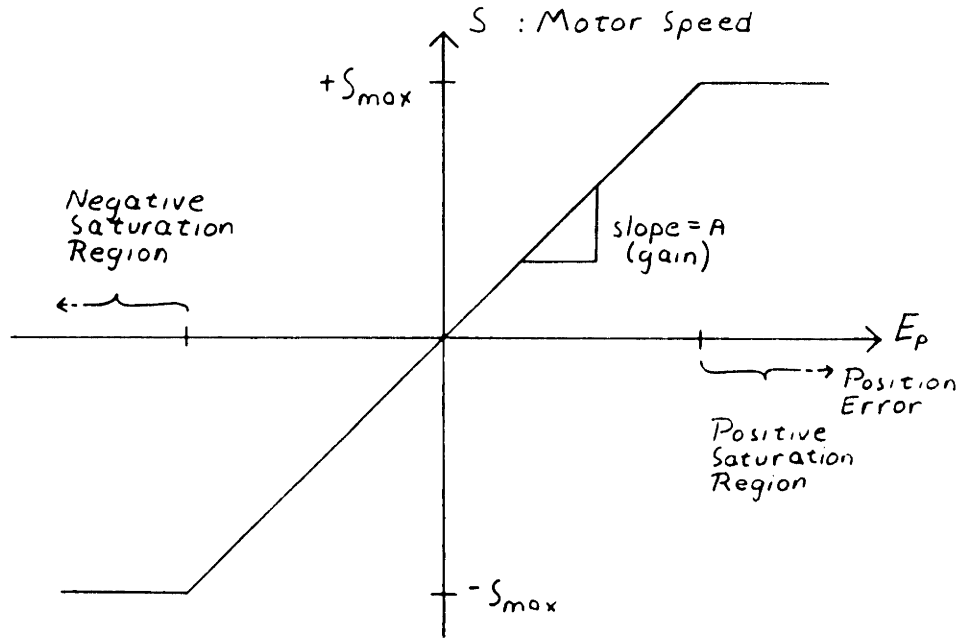


Figure 4-1: P-Controller: Motor Speed vs. Error

The robot successfully completed a walking cycle; however, there were points along the sequence where the robot would occasionally pause. This was due to insufficient torque being delivered to achieve the desired motor position configuration within the given tolerances.

Herein lies a serious drawback inherent to a proportional controller. In trying to achieve maximum response time and minimum overshoot, gain has to be compromised. As the positional error approaches zero, so does the applied torque. Under full load conditions, this can preclude the achievement of the desired motor configurations unless a sufficiently large tolerance, or positional error is allowed. From this evaluation, it became evident that a more sophisticated controller would be necessary to achieve smooth walking.

4.2.2 P-D Controller

Consideration of angular velocity would be necessary in the motor controller algorithm. For this, a Proportional-Derivative Controller (or P-D Controller) was designed and implemented. As in the evaluation of the Proportional controller, suspended and self-supported robot response was observed for a range of control parameters. The performance of the P-D controller was far superior to the Proportional Controller, exhibiting virtually undetectable overshoot and a response time under .5 second for a 30 degree positional step. The control algorithm was derived from the differential equations as follows:

In continuous time:

$$u(t) = A * (1 + T*s/(s+alpha)) * err \quad (4.1)$$

where u(t) is the effective voltage applied to the motor, A is the controller gain, T is the derivative section gain, s is the differential operator, alpha is the rolloff frequency, and err is the position error.

The steady state no load speed of a motor is linearly proportional to the applied voltage. From rest, the motor speed follows an exponential curve approaching the steady state speed. For the slow output speeds of the gear motors used, the motors approach their steady state speeds almost instantly. Under a full load condition, u(t) represents the motor torque.

Transforming equation (4.1) to discrete time:

$$u[k] = A * (1 + (a/T) * (z-1)/(z+g)) * err[k] \quad (4.2)$$

where a is the relative differential gain, T is the sampling period, z is the discrete time differential operator, and g is the inverse of the rolloff frequency. These parameters are illustated in Figure 4-2.

The proportional component of Equation becomes:

$$u[k] = A * err[k] \quad (4.3)$$

and the differential componant becomes:

$$u[k] * (z+g) = A * (a/T) * (z-1) * err[k]$$

$$u[k+1] + g*u[k] =$$

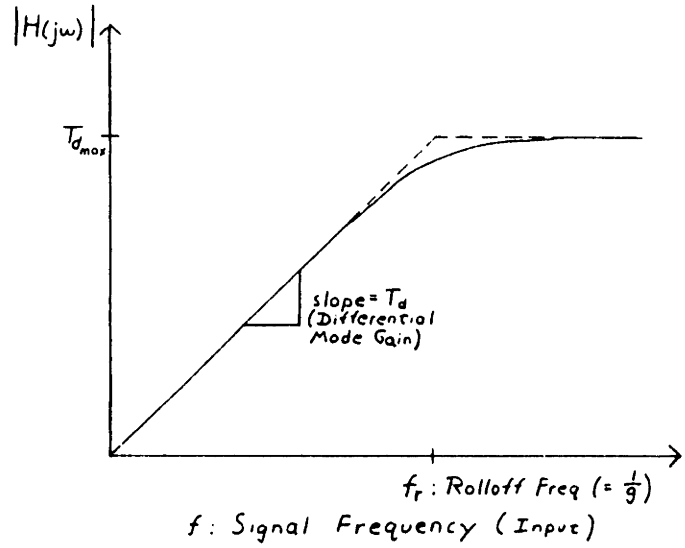


Figure 4-2: P-D Controller Frequency Response

$$A * (a/T) * (err[k+1] - err[k]) \quad (4.4)$$

Equations (4.2) and (4.3) combine to give the control law for u[k]:

$$u[k] = -g*u[k-1] + A*(a/T) * (err[k] - err[k-1]) + A*err[k] \quad (4.4)$$

The P-D controller was initially tested with g=0 and a=0, resulting in a system response identical to that of the P-Controller. A range of values were tested for the control law given in equation (4.4), and evaluation of both suspended and self-supporting robot performance was made. For the P-D Controller, a sample period of T = 3.4 ms was achieved, corresponding to a sample rate of about 300 Hz. As expected, response time increased with controller gain A. Response time also increased with relative differential gain a, and decreased as g increased.

System performance was evaluated for a range of control parameter values to gain a more solid conceptual understanding of what characteristics defined an ideal controller for this application. Qualitative characteristics of the robot's walking performance were noted and are shown in Table 4-I. In this evaluation, three variables were tested; "1/R-Frq" represents the inverse of the rolloff frequency or g, "D-Gain" represents the differential gain of A*(a/T), and "P-Gain" represents the proportional gain of A.

The dynamic response of the P-D Controller exhibited much more concise positional increments than the Proportional Controller. The walking sequence was executed much smoother and with very little oscillation of the robot body, a characteristic which was present with the Proportional Controller. Under proportional control, as the robot axes moved to assume the desired configuration, the body would oscillate at 1 to 2 Hertz and with a magnitude of 1 inch, primarily in the forward/backward direction. The P-D Controller eliminated this oscillation, which slowed the 4-step walk cycle by a factor of 2, and produced a well defined swift walk cycle occupying roughly 5 seconds. In addition, in a "static" stance (for which the controller maintained the robot in a constant axes configuration), the robot remained rigid, exhibiting no noticable bodily movement in response to

1/R-Frq	D-Gain	P-Gain	Performance Evaluation
0	0	3	Suspended: approx. 5 degrees of overshoot Walking: Walks successfully, with some bodily oscillation (~2Hz, 1" magnitude) (emulates a P-Controller)
0	30	3	Suspended: exhibits very low overshoot Walking: walks with solidity, concise axis movement
0	60	3	Suspended: no noticable overshoot, motion is very defined, almost choppy Walking: walks successfully, but with some choppiness, motion is less fluid than before
.3	30	3	Suspended: very solid/ defined motion Walking: walking is weaker, robot exhibits limited power

Table 4-I: Qualitative Controller Evaluation

sudden perturbational forces of 5 to 10 lbs. It was

1/R-Frq	D-Gain	P-Gain	Performance Evaluation
1	30	3	Suspended: virtually no power, motors oscillate without developing much torque at all Walking: incapable of delivering necessary torque
0	30	5	Suspended: quick, with some overshoot which seems to be due more to shaft springiness than to controller response. Walking: is much stronger than before, robot walks with very little hesitation
0	50	5	Suspended: very fast motion, very short response time Walking: fast, choppy walking, little hesitation

Table 4-II: Qualitative Controller Evaluation [cont'd]

expected that the P-D Controller would serve to yield a much smaller overshoot than the P-Controller; however, the improved response time, unnoticable overshoot, and walking solidity far exceeded original expectations.

Chapter 5
Walking Coordination

5.1 Gait Selection

5.1.1 Statically Balancing Gait

There are two distinct categories of gaits for legged locomotion: statically balancing gaits and dynamically balancing gaits. The former is the simplest and the easiest to implement. A statically balancing gait (SBG) involves a walking sequence which includes three planted legs (defining a plane) during all points in the sequence. To maintain balance on three legs, the center of mass must be maintained within the triangle defined by the three planted legs. To accomplish this, the robot must shift its weight in preparation for each step. The center of mass must be shifted far enough into the "support triangle" so that the movement of the stepping leg does not shift the balance out of the triangle. A weight shifting and stepping sequence which accomplishes this is shown in Figure 5-1.

The gait depicted in Figure 5-1 maintains the center of mass within the 3 or 4 sided polygon defined by the planted legs. In the 16 stage sequence, 4 stages each are

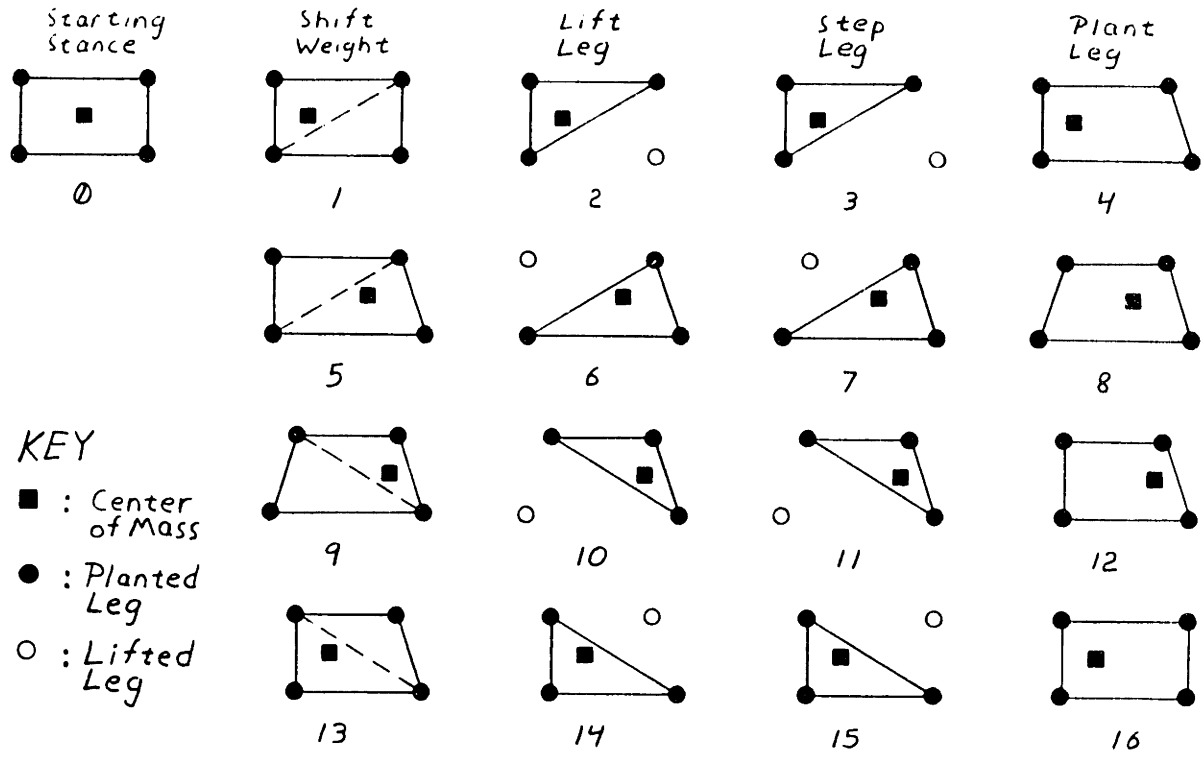


Figure 5-1: Statically Balancing Gait

required to shift weight, lift the leg, step the leg, and plant the leg. The repeated weight shifting slows the walking sequence considerably and provides for an oscillatory body motion, the body moving forward and backward as the robot shifts weight between steps of the four legs. An abbreviated version of the walking sequence of Figure 5-1 was implemented, but a very slow walk was achieved. For this reason, a dynamically balancing gait is favorable, and was explored.

5.1.2 Dynamically Balancing Gait

A dynamically balancing gait allows the robot to assume axis configurations, that if maintained statically (i.e. held rigidly), would result in a fall. A statically unstable configuration can result if either of two conditions is met: less than three legs are planted or the center of mass lies outside the supporting triangle. For a monoped, biped, or triped, a dynamically balancing gait is the only form of locomotion that is possible. A quadruped, however, is capable of a statically stable gait, a dynamically stable gait, or a hybrid of the two, allowing a shift between statically stable and unstable configurations during the walking sequence.

A gait which is statically unstable during all points in the sequence is given in Figure 5-2, in which pairs of diagonally opposite legs are synchronized. This gait is employed in a four-legged hopping machine built at Carnagie-Mellon University under Prof. Marc H. Raibert [Raibert 83]. This machine hops on pneumatically actuated telescoping legs; since the leg changes length by telescoping rather than by bending at the knee, the mode of operation is considerably different than the walking robot presented herein. The hopping machine requires machine attitude information from on board gyroscopes to maintain balance for each hop. Although it appears

possible that a walking machine (with bendable knees) could perform this walking sequence without the additional attitude information feedback, another possibility was explored.

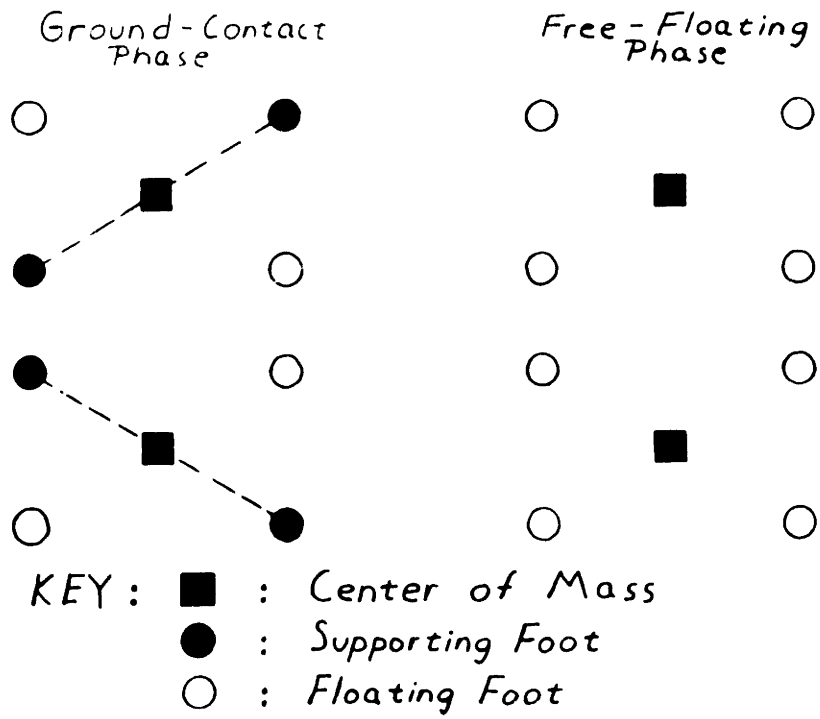


Figure 5-2: Alternating Leg DBG Sequence

The sequence of human walking can be broken down into two phases: offbalancing and stepping. From a stable vertical stance, a person prepares for a forward step by leaning forward and thus offbalancing himself. This is followed by the stepping of one of the two legs to arrest the forward directed angular velocity of the body, resulting in a forward translational movement. The SBG

proposed in Section 5.1.1 depicts a crawling movement as opposed to the true walking just described. The Dynamically Balancing Gait (DBG) developed for this machine is a hybrid between a crawl and a walk.

At the beginning of the DBG sequence, the robot shifts its weight backwards in preparation for a forward step. This is similar to the beginning of the SBG sequence. The difference is in the magnitude of the weight shift. In the DBG sequence, the weight is shifted less than in the SBG sequence. As the DBG sequence progresses, the left front leg is lifted and stepped. As the leg steps forward, the center of mass crosses out of the supporting triangle and the robot begins to dip forward. Upon planting of the left front leg, the robot arrests the forward dipping motion and continues to shifts its weight forward, developing momentum. The robot then lifts the right back leg and proceeds to step it forward. Upon planting, the robot weight is shifted forward again, resulting in a slight forward offbalancing. The right front leg quickly takes a step and is planted, preventing a forward fall. The left back leg completes the walking sequence with its step.

A weight shifting technique developed while working on a DBG sequence allowed the magnitude of the weight shift to be further reduced by taking advantage of the 6" "toes" protruding forward and backward of the leg bottoms.

By bending the knees backward, the "feet" are inclined such that the point of contact with the ground is at the tip of the forward toe. This shifts the support triangle 5" to 6" forward, depending on the angle of inclination. As a result, the weight has already effectively been shifted 6" backward and requires little or no additional weight shifting in preparation for a forward step.

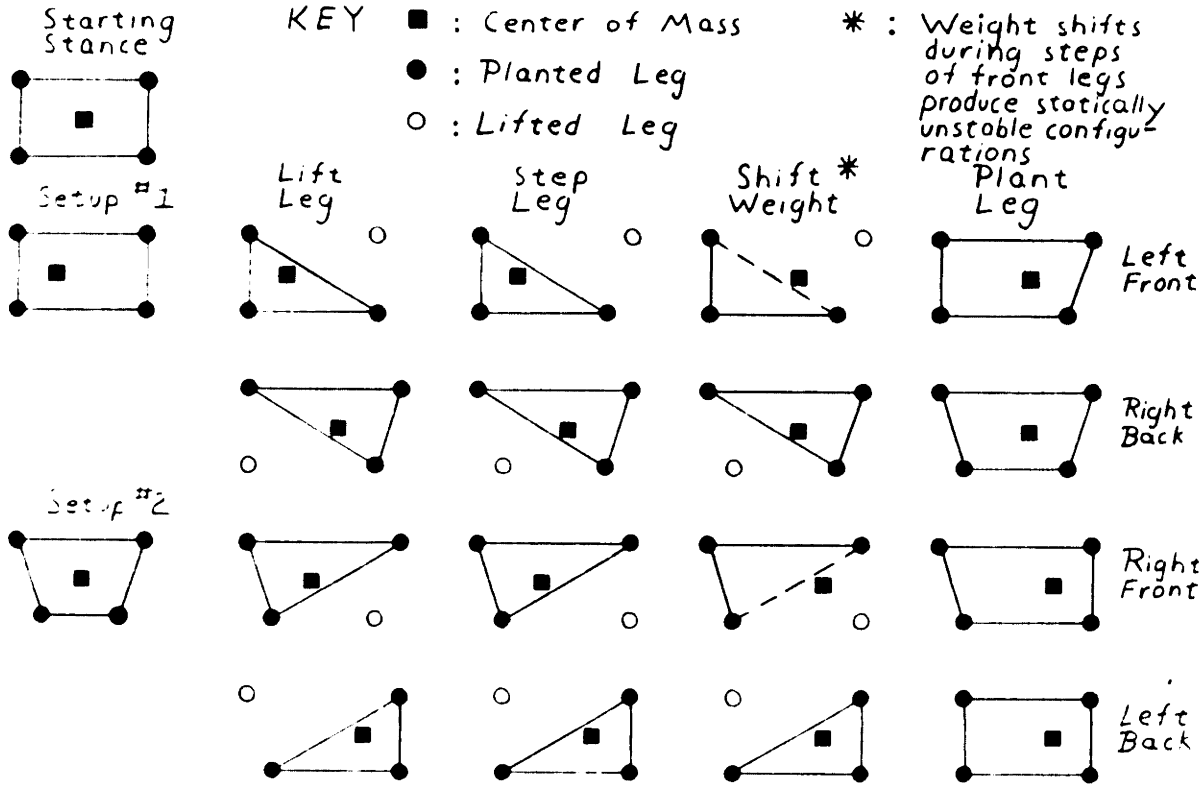


Figure 5-3: DBG Walking Sequence

This DBG sequence is shown in Figure 5-3, and is implemented in the WALKPD.C control program included in appendix Section A.2. As expected, the dynamically balancing gait resulted in a walk that is much faster than

the statically balancing gait sequence, typically by a factor of 2 to 3. A dynamically balancing gait was implemented using the P-Controller described in Section 4.2.1, and the robot completed a successful 4-step sequence; but the resulting motion included some bodily oscillation which occasionally would result in hesitation between steps. A much more steady walk was achieved when the DBG was implemented based on a P-D Controller, allowing the robot to take quick steps which are important in arresting the forward motion resulting from a forward offbalancing.

Chapter 6

Software Development

6.1 Control Heirarchy

The software component of the quadrupedal robot controller occupies three levels. The top level dictates the walking sequences to be performed. These instructions supported for the top level specify the directions and magnitudes of robot movements. Such movement commands include SHIFT_WEIGHT, SHIFT_AND_DIP, SHIFT_AND_GRIP, LIFT_LEG, STEP_LEG, and PLANT_LEG. The intermediate software level defines these robot movements. Each of the robot movement routines receives one or two magnitude parameters, and a corresponding modifications to a Motor Desired Position Array are effected. From this, control is passed to the bottom software level, which performs the actual motor speed control and the interfacing to the Pulse Width Modulator hardware. Dividing the robot control into three levels provides flexibility in developing an operational walking algorithm. Each level is distinct, and as such can endure considerable modification without requiring changes in the adjoining software levels. It was this stratification that allowed

the rapid development of a P-D Controller from a P-Controller; the walking sequence (top) and movement definition (intermediate) levels remained unchanged, as the P-D Controller was written to be software compatable with the P-Controller.

6.2 C Language Implementation

6.2.1 Motor Control and Interface

The bottom software level was the first to be developed. This entailed writing routines to interface the computer to the pulse width modulator hardware. This interface is accomplished through two I/O modules on board an AT&T PC 6300 Plus microcomputer on which the software was written and executed. An Analog Devices RTI-815 analog I/O module is used to input the analog position feedback data, and a Metra-Byte PIO-12 digital I/O module is used to write data to the pulse width modulator, accomplishing motor speed control. The analog interface card provides 32 input and 2 output channels, of which 12 of the input channels are used. The digital interface card provides three 8-bit I/O ports, all of which are configured and used in the output mode.

A collection of specific interfacing routines were developed and are combined in the QROBOT.C and QROBOTPD.C libraries listed in Appendix Sections A.4 and A.5

respectively. The routine UPDATE_MOTORS sequentially updates the status of the twelve motors according to the motor speed and direction arrays passed to it. Updating the status of a motor involves writing to the respective output ports the motor speed (an integer value of 0 to 255), motor direction (a two bit binary code of 00 to 11, specifying FWD, REV, and two OFF states), and the address (0 to 11) of the particular motor. Latching (or reading) by the PWM is accomplished by cycling a clock line low and high again. Seperate routines were written to load the present motor positions into an array (UPDATE_MRPA), to generate the required motor speeds based on the particular control law being implemented (CONTROL_SPEED), and to check that all twelve motors are within the allowed tolerance of their desired position (CHECK_TOLERANCES). These four routines are combined in the motor alignment control routine (DAM : Dynamically Align Motors). Special programs were written to test the routines in this bottom level before development proceeded to the next higher level.

6.2.2 Motion Definition

The intermediate software level supports the discrete movements to be combined into a walking sequence in the top level. Such movement routines include SHIFT_WEIGHT, SHIFT_AND_DIP, SHIFT_AND_GRIP, LIFT_LEG, STEP_LEG, and

PLANT_LEG. The shift commands allow the center of mass of the robot to be shifted forward or backward as needed in the walking sequence. Each routine performs an appropriate set of modifications to the Motor Desired Position Array (MDPA[]). The SHIFT_WEIGHT routine performs a simple forward or backward weight shift, incorporating hip rotations and knee bends necessary to maintain flat foot positions for the planted legs. The SHIFT_AND_DIP routine shifts and lowers the center of mass for a more stable stance. The SHIFT_AND_GRIP routine performs the more efficient weight shift mentioned in Section 5.1.2 and which allows the foothold point to be shifted to the tips of the toes, necessitating a smaller weght shifting magnitude and allowing for a faster walking sequence. The lift, step, and plant leg routines are self explanatory and operate in the same manner as the routines already mentioned.

6.2.3 Walking Sequence

The last and highest software level to be developed is the Walking Sequence control. This developmental stage involved extended real time experimentation with robot walking sequences and their consequences in gait speed and stability. The WALK.C program listed in Appeddix A.1 was the first such sequence and represents the first step of a 4-step statically stable gait. After determining the

superiority of a dynamically stable gait, WALKPD.C (Walk using P-D Control) listed in Appendix A.2 was developed and successfully took the robot through a 4-step DSG sequence. These walking sequence controllers are based on the gait theory described in Subsections 5.1.1 and 5.1.2 and are the product of numerous iterations of sequence evaluation and improvement.

Chapter 7

Conclusion and Performance Evaluation

7.1 System Evaluation

7.1.1 Mechanical System

The physical implementation of the robot allows successful walking. Occasionally, during a particular walking sequence, the robot will hesitate in its motion as the motors apply torques to align the robot in the specified configuration. For this reason, it is concluded, a superior mechanical system will be arrived at by incorporating more powerful actuators, such as larger gear motors or possibly hydraulic drive mechanisms.

The lateral hip design incorporates a 1/4"-20 lead screw to accomplish abduction and adduction. This lead screw has been susceptible to damage resulting from sudden violent robot falls. This weakness can be overcome by using a larger lead screw or preferrably by installing mechanical limiting mechanisms, restricting the range of leg motion to prevent the bending of the lead screw against the robot body in the event of a rapid and complete fall.

7.1.2 Electrical System

The pulse width modulator has performed completely up to expectation, and facilitates reliable 8-bit control of motor speeds. This circuit exhibits no noticable susceptibility to EMI noise from the motor and power supply lines, and no improvement in this area is necessitated.

The output stage circuitry has also performed up to expectation, with no reliability problems or overheating observed.

7.1.3 Motor Control System

As mentioned in Section 4.2.1, the proportional controller exhibited several undesirable characteristics, including noticable overshoot and mediocre response time, resulting in robot body oscillation during the walking sequence. The development of a P-D Controller appears to have fulfilled the motor control performance requirements necessary for quadrupedal locomotion. Depending on the control parameters used, overshoot can be practically eliminated; and response time appears to be limited by the power delivered by the mechanical system rather than by the motor controller.

7.1.4 Locomotion System

The development of walking sequences was the last phase of this project and could be taken further. As described in Section 5.1.1, a statically balancing gait was experimented with and rejected due to the slow walking performance that it offered. One particular dynamically balancing gait was developed, offering a much improved walking speed. The diagonally alternating leg gait mentioned in Section 5.1.2 may prove to be superior to the walking gait which was developed, and there are numerous other gaits ranging from trots to gallops which deserve attention and may require the development of higher performance mechanical and control systems in their implementation.

7.2 Further Development Directions

This project focuses on only one particular type of walking machine, an electrically powered quadrupedal robot. Section 7.1 gives several specific areas in which performance could be improved for this particular device. Walking machines include a spectrum of single and multilegged designs with a variety of power and control schemes. Undoubtedly, in the development of a practical all-terrain legged vehicle, a diversity of designs must be explored, tested, and evaluated before the field matures into a successful industry.

Appendix A

C Code Listings

A.1 WALK.C Program Listing

```
/*
Author:          Daniel J. DiLorenzo
Filename:        WALK.C
Revision Date:   4/23/1987
Function:        This program steps the robot through a fo\
rward walking
                 sequence using incremental routines.
                 This program makes use of the QROBOT.C li\
brary.
*/

#include<stdio.h>
#include<conio.h>                    /* basic functions, i.e. \
getc() */
#include<c:\djd\qrobot.c>            /* Quadrupedal ROBOT prim\
itives */
#include<c:\djd\locom.c>             /* robot LOCOMote primiti\
ves */

main()
{
int swm;          /* Shift Weight Magnitude */
int dwm;          /* Dip Weight Magnitude */
int llm;          /* Lift Leg Magnitude */
int blm;          /* Bend Leg Magnitude */
int slm;          /* Step Leg Magnitude */

int sl;
char sel, wait;
int mdpa[12], mrpa[12], mra[12], mta[12], mpa[12];


init_io_modules();
printf("I/O Modules Initialized.\n");
printf("Align robot in starting position, hit <RETURN> : \
");
wait = getch();
calibrate(mra);
```

```
init_mta(mta);
zero_array(mdpa);
printf("Motor tolerance array:\n");
print_array(mta);

printf("This program steps the quadrupedal robot through \
a forward \n");
printf(" walking sequence using a successive alignment mo\
dification \n");
printf(" technique.\n");

printf("Enter SWM (~ 50) : ");
sl = scanf("%d",&swm);
printf("Enter DWM (new)  : ");
sl = scanf("%d",&dwm);
printf("\nEnter LLM (~ 100) : ");
sl = scanf("%d",&llm);
printf("\nEnter SLM (~ 50) : ");
sl = scanf("%d",&slm);
blm = llm;
printf("\n Shift Weight mag= %d, Lift mag= %d, Step mag= \
%d\n",swm,llm,slm);

pause_for_user();

sel = 'r';
while(sel != 'q') {
        shift_weight(1,1,1,1,-swm,mdpa,mrpa,mra,mta);
        pause_for_user();
        shift_weight(1,1,1,1,-swm,mdpa,mrpa,mra,mta);
        pause_for_user();
        lift_leg(LF,llm,mdpa,mrpa,mra,mta);
        pause_for_user();
        step_leg(LF,slm,mdpa,mrpa,mra,mta);
        pause_for_user();
        shift_weight(0,1,1,1,swm,mdpa,mrpa,mra,mta);
        pause_for_user();
        shift_weight(0,1,1,1,swm,mdpa,mrpa,mra,mta);
        pause_for_user();
        plant_leg(LF,llm,mdpa,mrpa,mra,mta);

        sel = rerun_prompt();
/*      printf("Run or Quit : ");
        sl = scanf("%c",&sel);
        printf("\n"); */
}
printf("Quit acknowledged, program exited.\n");
}
```

A.2 WALKPD.C Program Listing

```
/*
Author:          Daniel J. DiLorenzo
Filename:        WALKF.C
Revision Date:   4/23/1987
Function:        This program steps the robot through a fo\
rward walking
                 sequence using incremental routines.  In \
this implementation,
                 the robot is allowed to be off-balanced d\
uring the walking
                 sequence.  (The idea is to regain balance\
 later in the
                 sequence, i.e. on the following increment\
, without falling
                 between increments ==> timing & speed are\
 essential.)
                 This program makes use of the LOCOMPD.C a\
nd QROBOTPD.C
                 libraries.
                 This implements a P-D Controller (Proport\
ional, Differential)
*/

#include<stdio.h>
#include<conio.h>                /* basic functions, i.e. \
getc() */
#include<c:\djd\qrobotpd.c>      /* Quadrupedal ROBOT prim\
itives */
#include<c:\djd\locompd.c>       /* robot LOCOMote primiti\
ves */

main()
{
int swm;         /* Shift Weight Magnitude */
int dwm;         /* Dip Weight Magnitude */
int llm;         /* Lift Leg Magnitude */
int blm;         /* Bend Leg Magnitude */
int slm;         /* Step Leg Magnitude */
int sl;
char sel, wait, pause_response;
int pause_status;
int mdpa[12], mrpa[12], mra[12], mta[12], mpa[12];


init_io_modules();
printf("I/O Modules Initialized.\n");
printf("Align robot in starting position, hit <RETURN> : \
");
```

```
wait = getch();
calibrate(mra);
init_mta(mta);
zero_array(mdpa);

printf("This program steps the quadrupedal robot through \
a forward \n");
printf(" walking sequence using a successive alignment mo\
dification \n");
printf(" technique.\n");

printf("Enter SWM (~ 50) : ");
sl = scanf("%d",&swm);
printf("Enter DWM (new)  : ");
sl = scanf("%d",&dwm);
printf("\nEnter LLM (~ 100) : ");
sl = scanf("%d",&llm);
printf("\nEnter SLM (~ 50) : ");
sl = scanf("%d",&slm);
printf("\nEnter BLM/2 (~SLM/2) : ");
sl = scanf("%d",&blm);

printf("\n Shift Weight mag= %d, Lift mag= %d, Step mag= \
%d\n",swm,llm,slm);
printf("\n Insert Pauses (yes=1, no=0): ");
pause_response = getch();
printf("%c \n",pause_response);

pause_status = 0;
if((pause_response == 'y') || (pause_response == 'Y')) { \
pause_status = 1;}

pause_for_user();



sel = 'r';
while(sel != 'q') {
        printf("LEFT FRONT LEG-----------------\n");
        shift_and_dip(1,1,1,1,-swm/4,-dwm/4,mdpa,mrpa,mra\
,mta);
        shift_and_dip(1,1,1,1,-swm/4,-dwm/4,mdpa,mrpa,mra\
,mta);
        shift_and_dip(1,1,1,1,-swm/4,-dwm/4,mdpa,mrpa,mra\
,mta);
        shift_and_dip(1,1,1,1,-swm/4,-dwm/4,mdpa,mrpa,mra\
,mta);
        lift_leg(LF,llm,2*blm,mdpa,mrpa,mra,mta);
        step_leg(LF,slm/2,mdpa,mrpa,mra,mta);
        step_leg(LF,slm/2,mdpa,mrpa,mra,mta);
        shift_and_dip(1,1,1,1,swm/2,dwm/2,mdpa,mrpa,mra,m\
```

```
ta);
        shift_and_dip(1,1,1,1,swm/2,dwm/2,mdpa,mrpa,mra,m\
ta);
        plant_leg(LF,l1m,2*b1m,mdpa,mrpa,mra,mta);

        if(pause_status) {pause_for_user();}

        printf("RIGHT BACK LEG-----------------\n");
        lift_leg(RB,l1m,2*b1m,mdpa,mrpa,mra,mta);
        step_leg(RB,s1m/2,mdpa,mrpa,mra,mta);
        step_leg(RB,s1m/2,mdpa,mrpa,mra,mta);
        shift_weight(1,1,1,0,swm/2,mdpa,mrpa,mra,mta);
        shift_weight(1,1,1,0,swm/2,mdpa,mrpa,mra,mta);
        plant_leg(RB,l1m,2*b1m,mdpa,mrpa,mra,mta);

        if(pause_status) {pause_for_user();}

        printf("RIGHT FRONT LEG----------------\n");
        lift_leg(RF,l1m,2*b1m,mdpa,mrpa,mra,mta);
        step_leg(RF,s1m,mdpa,mrpa,mra,mta);
        shift_weight(1,0,1,1,swm/2,mdpa,mrpa,mra,mta);
        shift_weight(1,0,1,1,swm/2,mdpa,mrpa,mra,mta);
        plant_leg(RF,l1m,2*b1m,mdpa,mrpa,mra,mta);

        if(pause_status) {pause_for_user();}

        printf("LEFT BACK LEG------------------\n");
        lift_leg(LB,l1m,2*b1m,mdpa,mrpa,mra,mta);
        step_leg(LB,s1m,mdpa,mrpa,mra,mta);
        shift_weight(1,1,0,1,swm/2,mdpa,mrpa,mra,mta);
        shift_weight(1,1,0,1,swm/2,mdpa,mrpa,mra,mta);
        plant_leg(LB,l1m,2*b1m,mdpa,mrpa,mra,mta);

        starting_position(mdpa,mrpa,mra,mta);

        sel = rerun_prompt();

}
printf("Quit acknowledged, program exited.\n");
}
```

A.3 LOCOMPD.C Library Listing

```
/*
Author:          Daniel J. DiLorenzo
Filename:        LOCOMPD.C      (quadrupedal robot LOCOMOT\
E primitives library)
Revision Date:   4/28/1987
Function:        This library provides the routines essen\
tial to locomote
                 the quadrupedal robot using a successive\
 alignment
                 modification technique.
                 This library makes use of the P-D Contro\
ller supported by
                 QROBOTPD.C
*/

#define LF 0     /* Leg numbers */
#define RF 1
#define LB 2
#define RB 3

#define RH 0
#define LR 4
#define KN 8

#define LFRH 0
#define RFRH 1
#define LBRH 2
#define RBRH 3
#define LFLH 4
#define RFLH 5
#define LBLH 6
#define RBLH 7
#define LFKN 8
#define RFKN 9
#define LBKN 10
#define RBKN 11

shift_weight(lfs,rfs,lbs,rbs,magn,mdpa,mrpa,mra,mta)
int lfs, rfs, lbs, rbs, magn, mdpa[12], mrpa[12], mra[12\
], mta[12];
{
printf("SHIFT_WEIGHT reached.\n");       /* ***** RUN TIM\
E TEST STMT */
if(lfs==1) { mdpa[LFRH] = mdpa[LFRH] - magn; }
if(rfs==1) { mdpa[RFRH] = mdpa[RFRH] - magn; }
if(lbs==1) { mdpa[LBRH] = mdpa[LBRH] - magn; }
if(rbs==1) { mdpa[RBRH] = mdpa[RBRH] - magn; }
```

```
dam(mdpa,mrpa,mra,mta);
}

lift_leg(ln,llm,blm,mdpa,mrpa,mra,mta)
int ln, llm, blm, mdpa[12], mrpa[12], mra[12], mta[12];
{
printf("LIFT_LEG reached \n");
mdpa[RH+ln] = mdpa[RH+ln] + llm;
mdpa[KN+ln] = mdpa[KN+ln] - blm;

dam(mdpa,mrpa,mra,mta);
}

plant_leg(ln,llm,blm,mdpa,mrpa,mra,mta)
int ln, llm, blm, mdpa[12], mrpa[12], mra[12], mta[12];
{
printf("PLANT_LEG reached \n");
mdpa[RH+ln] = mdpa[RH+ln] - llm;
mdpa[KN+ln] = mdpa[KN+ln] + blm;

dam(mdpa,mrpa,mra,mta);
}

step_leg(ln,magn,mdpa,mrpa,mra,mta)
int ln, magn, mdpa[12], mrpa[12], mra[12], mta[12];
{
printf("STEP_LEG reached \n");
mdpa[RH+ln] = mdpa[RH+ln] + magn;

dam(mdpa,mrpa,mra,mta);
}

shift_and_dip(lfs,rfs,lbs,rbs,smagn,dmagn,mdpa,mrpa,mra,\
mta)
int lfs, rfs, lbs, rbs, smagn, dmagn, mdpa[12], mrpa[12]\
, mra[12], mta[12];
{
printf("SHIFT_AND_DIP reached \n");
if (lfs==1) {
        mdpa[LFRH] = mdpa[LFRH] - smagn;
        mdpa[LFKN] = mdpa[LFKN] + dmagn;
        }
if (rfs==1) {
        mdpa[RFRH] = mdpa[RFRH] - smagn;
        mdpa[RFKN] = mdpa[RFKN] + dmagn;
        }
if (lbs==1) {
        mdpa[LBRH] = mdpa[LBRH] - smagn;
        mdpa[LBKN] = mdpa[LBKN] + dmagn;
        }
if (rbs==1) {
```

```
        mdpa[RBRH] = mdpa[RBRH] - smagn;
        mdpa[RBKN] = mdpa[RBKN] + dmagn;
        }
dam(mdpa,mrpa,mra,mta);
}

starting_position(mdpa,mrpa,mra,mta)
int mdpa[12], mrpa[12], mra[12], mta[12];
{

printf("Robot will be realigned to starting position,  "\
);
pause_for_user();
printf("CONFIRM THAT ROBOT IS SUSPENDED !!!!!!,  ");
pause_for_user();
printf("\nRealigning robot to starting position...\n");
zero_array(mdpa);
dam(mdpa,mrpa,mra,mta);
printf("Done.\n");
}




/* Useful Support Routines (not essential for robot loco\
motion) */

pause_for_user()
{
char wait;
printf("Hit a key to continue : ");
wait = getch();
printf("-->continuing...\n");
}

rerun_prompt()
{
int sl;
char sel;
sel = ' ';
while( ((sel!='r') && (sel!='R')) && ((sel != 'q') && (s\
el!='Q')) ) {
        printf("\n<R>erun or <Q>uit : ");
        sl = scanf("%c",&sel);
        printf("\n");
        }
return(sel);
}
```

```
print_table(a)
int a[12];
{
printf("                              LF \t RF \t LB \t RB \\
n");
printf("-----------------------------------------------\
-----------------\n");
printf("Rotary Hip          :      ");
printf("%d \t %d \t %d \t %d\n",a[0],a[1],a[2],a[3]);
printf("Lateral Hip         :      ");
printf("%d \t %d \t %d \t %d\n",a[4],a[5],a[6],a[7]);
printf("Knee                :      ");
printf("%d \t %d \t %d \t %d\n",a[8],a[9],a[10],a[11]);
printf("-----------------------------------------------\
-----------------\n");
}
```

A.4 QROBOT.C Library Listing

```
/*
Author:          Daniel J. DiLorenzo
Filename:        QROBOT.C        (Quadrupedal Robot LIBra\
ry)
Revision Date:   4/23/1987
Purpose:         This library contains useful primitive r\
outines to be used
                 by higher level control programs in the \
control of the
                 quadrupedal robot..
                 This Library includes the following rout\
ines:
                          CALIBRATE(mra,mnmax,board,gain)
                          UPDATE_MRPA(mrpa,mra,board,gain)
                          CHECK_TOLERANCES(mrpa,mdpa,mta)

*/

#include<stdlib.h>       /* Standard math fns, i.e. abs()\
 */

#define MNMAX 11
#define DIO_BASE_ADDRESS 896
#define PORTA (DIO_BASE_ADDRESS)
#define PORTB (DIO_BASE_ADDRESS + 1)
#define PORTC (DIO_BASE_ADDRESS + 2)
#define CONTROL_PORT (DIO_BASE_ADDRESS + 3)
#define CONTROL_BYTE 128
#define READY_LATCH 0    /* Latch clock is brought low in\
 preparation for : */
#define CLOCK_LATCH 1    /* Latch is clocked on the risin\
g edge and should  */
                         /*   be held high until the next\
 latching cycle */
#define OFF 0            /* 00 in bits D5,D4; OFF = 0000 \
0000 bin */
#define FWD 16           /* 01 in bits D5,D4; FWD = 0001 \
0000 bin */
#define REV 32           /* 10 in bits D5,D4; REV = 0010 \
0000 bin */
        /* Motor addresses A3-A0 are in Port C bits D3-D\
0 respectively */
        /* The directional specifier (fwb/rev/off) is ad\
ded to byte_c ^^ */
#define BOARD 1
#define GAIN 1

        /* Dynamic Control Parameters (1-3), used by CON\
```

```
TROL_SPEED */
#define DCP1 5              /* Minimum output speed (to over\
come friction) */
#define DCP2 75             /* Error value for which motor i\
s slowed below DCP3 */
#define DCP3 255            /* Maximum motor speed (used whe\
n maerr>DCP2) */

#define GTOL 75

init_aio()
{
int er;
char wait;
init(&er);  /* Requires the supplied library ADIMCS.LIB \
for the RTI-815 */
if(er) {
        printf("********** Analog I/O ERROR %d\n",er);
        printf("ABORT PROGRAM AND RUN CLOAD FIRST !!!!!!\
!!!!!!!!!!\n");
        scanf("%c",&wait);
        }
printf("\n\n");
}

init_dio()
{
outp(CONTROL_PORT,CONTROL_BYTE);
outp(PORTB,0);
outp(PORTC,0);
outp(PORTA,CLOCK_LATCH);
}

init_io_modules()
{
init_aio();
init_dio();
}

init_mta(mta)
int mta[12];
{
int mn;
for(mn=0; mn<=MNMAX; mn++) {
        mta[mn] = GTOL;
        }
}

zero_array(array)
int array[12];
{
```

```
int i;
for(i=0; i<=MNMAX; i++) {
        array[i] = 0;
        }
}


calibrate(mra)
int mra[12];
{
int mn, er, val;
for(mn = 0; mn <= MNMAX; mn++) {
        val = ain(BOARD, mn, GAIN, &er);
        mra[mn] = val;
        }
}


update_mrpa(mrpa,mra)
int mrpa[12],mra[12];
{
int mn, mp, er;

for(mn = 0; mn <= MNMAX; mn++) {
        mp = ain(BOARD, mn, GAIN, &er);
        mrpa[mn] = mp - mra[mn];
        }
}

check_tolerances(mrpa,mdpa,mta)
int mrpa[12], mdpa[12], mta[12];
{
int mn, tol_accum, maerr;

tol_accum = 0;
for(mn=0; mn <= MNMAX; mn++) {
        maerr = abs(mrpa[mn] - mdpa[mn]);
        if (maerr > mta[mn]) {
                tol_accum = tol_accum + 1;
                }
        }
/* printf("Motors exceeding tolerance = %d\n",tol_accum)\
; */
return(tol_accum);
}

stop_motors()
{
int i;

for(i=0; i<=MNMAX; i++) {
```

```
        outp(PORTB,0);
        outp(PORTC,i);
        outp(PORTA,READY_LATCH);
        outp(PORTA,CLOCK_LATCH);
        }
}

dam(mdpa,mrpa,mra,mta)
int mdpa[12], mrpa[12], mra[12], mta[12];
{
int msa[12], mda[12], motors_exceeding_tol;

motors_exceeding_tol = 1;
while(motors_exceeding_tol >0) {
        update_mrpa(mrpa,mra);
        control_speed(mrpa,mra,mdpa,msa,mda);
        update_motors(msa,mda);
        motors_exceeding_tol = check_tolerances(mrpa,mdp\
a,mta);
        }
stop_motors();
}


control_speed(mrpa,mra,mdpa,msa,mda)
int mrpa[12], mra[12], mdpa[12], msa[12], mda[12];
{
int mn, merr, maerr, appr_slope;

for(mn=0; mn<=MNMAX; mn++) {
        merr = mrpa[mn] - mdpa[mn];
        maerr = abs(merr);
        if (maerr>DCP1) {
                msa[mn] = DCP3; /*Max speed */
                }
        else {
                appr_slope = (DCP3 - DCP1)/DCP2;
                msa[mn] = (appr_slope * maerr) + DCP1;
                }
        if(merr > 0) {
                mda[mn] = REV;
                }
        else {
                mda[mn] = FWD;
                }
        }
}


update_motors(msa,mda)
int msa[12], mda[12];
```

```
{
int mn, byte_c;

for(mn = 0; mn <= MNMAX; mn++) {
        outp(PORTB,msa[mn]);
        byte_c = mn + mda[mn];
        outp(PORTC,byte_c);
        outp(PORTA,READY_LATCH);
        outp(PORTA,CLOCK_LATCH);
        }
}

/* For Debugging */

print_array(array)
int array[12];
{
int i;
for(i=0; i<= MNMAX; i++) {
        printf("array(%d) = %d \n",i,array[i]);
        }
}
```

A.5 QROBOTPD.C Library Listing

```
/*
Author:          Daniel J. DiLorenzo
Filename:        QROBOTPD.C      (Quadrupedal Robot LIBra\
ry)
Revision Date:   4/23/1987
Purpose:         This library contains useful primitive r\
outines to be used
                 by higher level control programs in the \
control of the
                 quadrupedal robot.
                 This supports a P-D Controller.
                 This Library includes the following rout\
ines:
                          CALIBRATE(mra,mnmax,board,gain)
                          UPDATE_MRPA(mrpa,mra,board,gain)
                          CHECK_TOLERANCES(mrpa,mdpa,mta)

*/

#include<stdlib.h>       /* Standard math fns, i.e. abs()\
 */
#include<math.h>         /* More standard math fns, i.e. \
modf(x,&n) */

#define MNMAX 11
#define DIO_BASE_ADDRESS 896
#define PORTA (DIO_BASE_ADDRESS)
#define PORTB (DIO_BASE_ADDRESS + 1)
#define PORTC (DIO_BASE_ADDRESS + 2)
#define CONTROL_PORT (DIO_BASE_ADDRESS + 3)
#define CONTROL_BYTE 128
#define READY_LATCH 0    /* Latch clock is brought low in\
 preparation for : */
#define CLOCK_LATCH 1    /* Latch is clocked on the risin\
g edge and should  */
                         /*   be held high until the next\
 latching cycle */
#define OFF 0            /* 00 in bits D5,D4; OFF = 0000 \
0000 bin */
#define FWD 16           /* 01 in bits D5,D4; FWD = 0001 \
0000 bin */
#define REV 32           /* 10 in bits D5,D4; REV = 0010 \
0000 bin */
        /* Motor addresses A3-A0 are in Port C bits D3-D\
0 respectively */
        /* The directional specifier (fwb/rev/off) is ad\
ded to byte_c ^^ */
#define BOARD 1
```

```
#define GAIN 1

        /* Dynamic Control Parameters (1-3), used by CON\
TROL_SPEED */
#define PD_GAIN 3               /* Overall Gain of  P-D \
Copntroller */
#define FREQ_SLOPE 1    /* Slope of Bode Plot of Differe\
ntial Componant */
#define ROLLOFF_FREQ 0          /* Freq above which gain\
 levels off */
#define SAMPLE_PERIOD .005      /* Sampling period of co\
ntroller */
#define MSMAX 255       /* Maximum motor speed (used whe\
n maerr>DCP2) */

#define DCP1 0          /* = ROLLOFF_FREQ */
#define DCP2 50         /* = INT(PD_GAIN*FREQ_SLOPE/SAMP\
LE_PERIOD */
#define DCP3 5          /* = PD_GAIN */

#define GTOL 75

init_aio()
{
int er;
char wait;
init(&er);  /* Requires the supplied library ADIMCS.LIB \
for the RTI-815 */
if(er) {
        printf("********** Analog I/O ERROR %d\n",er);
        printf("ABORT PROGRAM AND RUN CLOAD FIRST !!!!!!\
!!!!!!!!!!\n");
        scanf("%c",&wait);
        }
printf("\n\n");
}

init_dio()
{
outp(CONTROL_PORT,CONTROL_BYTE);
outp(PORTB,0);
outp(PORTC,0);
outp(PORTA,CLOCK_LATCH);
}

init_io_modules()
{
init_aio();
init_dio();
}
```

```
init_mta(mta)
int mta[12];
{
int mn;
for(mn=0; mn<=MNMAX; mn++) {
        mta[mn] = GTOL;
        }
}

zero_array(array)
int array[12];
{
int i;
for(i=0; i<=MNMAX; i++) {
        array[i] = 0;
        }
}


calibrate(mra)
int mra[12];
{
int mn, er, val;
for(mn = 0; mn <= MNMAX; mn++) {
        val = ain(BOARD, mn, GAIN, &er);
        mra[mn] = val;
        }
}


update_mrpa(mrpa,mra)
int mrpa[12],mra[12];
{
int mn, mp, er;

for(mn = 0; mn <= MNMAX; mn++) {
        mp = ain(BOARD, mn, GAIN, &er);
        mrpa[mn] = mp - mra[mn];
        }
}

check_tolerances(mrpa,mdpa,mta)
int mrpa[12], mdpa[12], mta[12];
{
int mn, tol_accum, maerr;

tol_accum = 0;
for(mn=0; mn <= MNMAX; mn++) {
        maerr = abs(mrpa[mn] - mdpa[mn]);
        if (maerr > mta[mn]) {
                tol_accum = tol_accum + 1;
```

```
                }
        }
/* printf("Motors exceeding tolerance = %d\n",tol_accum)\
; */
return(tol_accum);
}

stop_motors()
{
int i;

for(i=0; i<=MNMAX; i++) {
        outp(PORTB,0);
        outp(PORTC,i);
        outp(PORTA,READY_LATCH);
        outp(PORTA,CLOCK_LATCH);
        }
}

dam(mdpa,mrpa,mra,mta)
int mdpa[12], mrpa[12], mra[12], mta[12];
{
int msa[12], mda[12], motors_exceeding_tol;

motors_exceeding_tol = 1;
while(motors_exceeding_tol >0) {
        update_mrpa(mrpa,mra);
        control_speed(mrpa,mra,mdpa,msa,mda);
        update_motors(msa,mda);
        motors_exceeding_tol = check_tolerances(mrpa,mdp\
a,mta);
        }
stop_motors();
}


control_speed(mrpa,mra,mdpa,msa,mda)
int mrpa[12], mra[12], mdpa[12], msa[12], mda[12];
{
int mn, merr, maerr, appr_slope;
static int oea[12], nea[12], osa[12], nsa[12];
int ns_temp;

update_input_arrays(osa,nsa,oea,nea,mdpa,mrpa);
for(mn=0; mn<=MNMAX; mn++) {

/*      ns_temp = (-ROLLOFF_FREQ*osa[mn]) + ((PD_GAIN*FR\
EQ_SLOPE/SAMPLE_PERIOD)*(nea[mn]-oea[mn])) + (PD_GAIN*ne\
a[mn]); */

        nsa[mn] = (-DCP1*osa[mn]) + DCP2*(nea[mn]-oea[mn\
```

```
]) + DCP3*nea[mn];

        if(nsa[mn] < 0) {
                msa[mn] = -nsa[mn];
                mda[mn] = REV;
                }
        else {
                msa[mn] = nsa[mn];
                mda[mn] = FWD;
                }
        if(msa[mn] > MSMAX) { msa[mn] = MSMAX; }
        }
}

update_input_arrays(osa,nsa,oea,nea,mdpa,mrpa)
int osa[12], nsa[12], oea[12], nea[12], mdpa[12], mrpa[1\
2];
{
int mn;
for(mn=0; mn<=MNMAX; mn++) {
        osa[mn] = nsa[mn];
        oea[mn] = nea[mn];
        nea[mn] = mdpa[mn] - mrpa[mn];
        }
}


update_motors(msa,mda)
int msa[12], mda[12];
{
int mn, byte_c;

for(mn = 0; mn <= MNMAX; mn++) {
        outp(PORTB,msa[mn]);
        byte_c = mn + mda[mn];
        outp(PORTC,byte_c);
        outp(PORTA,READY_LATCH);
        outp(PORTA,CLOCK_LATCH);
        }
}

/* For Debugging */

print_array(array)
int array[12];
{
int i;
for(i=0; i<= MNMAX; i++) {
        printf("array(%d) = %d \n",i,array[i]);
        }
}
```

Appendix B

Pictures of Robot

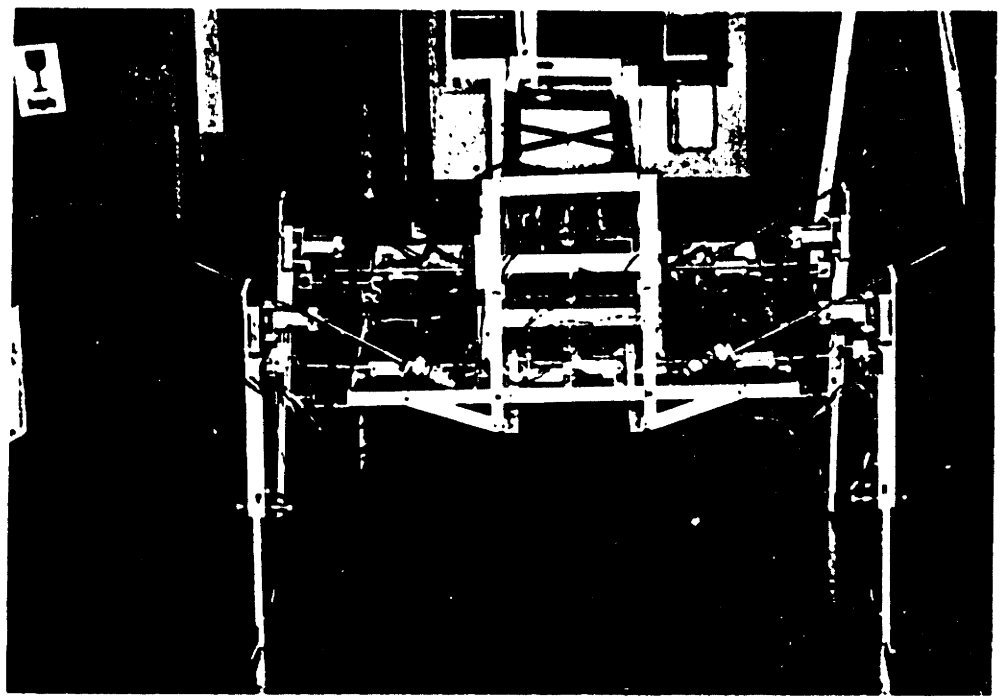


Figure B-1: Quadrupedal Robot: Front View

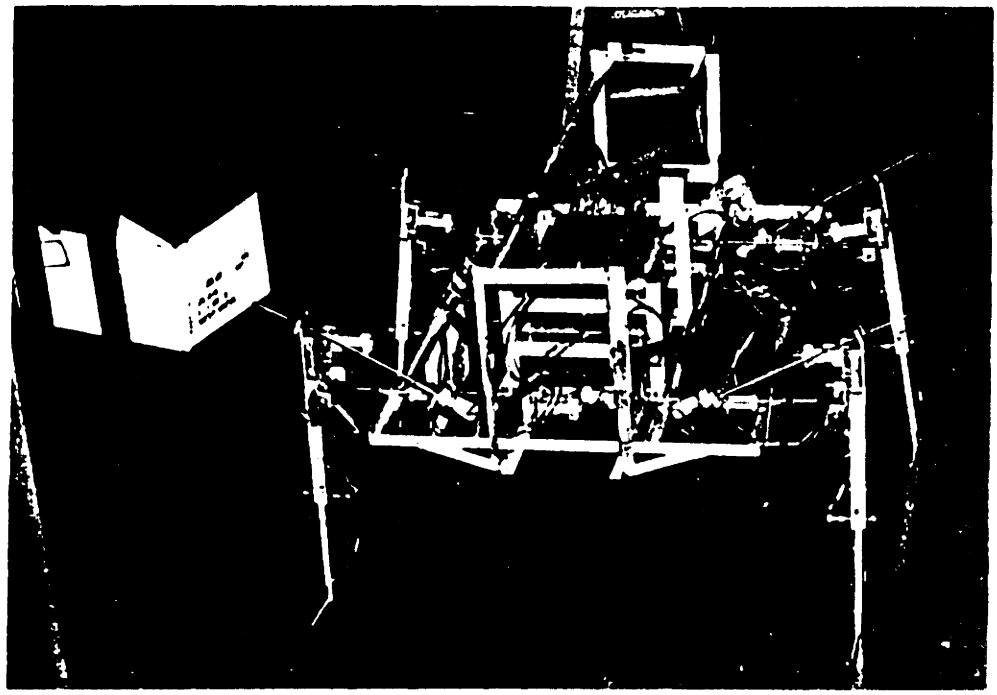


Figure B-2: Quadrupedal Robot: Diagonal View

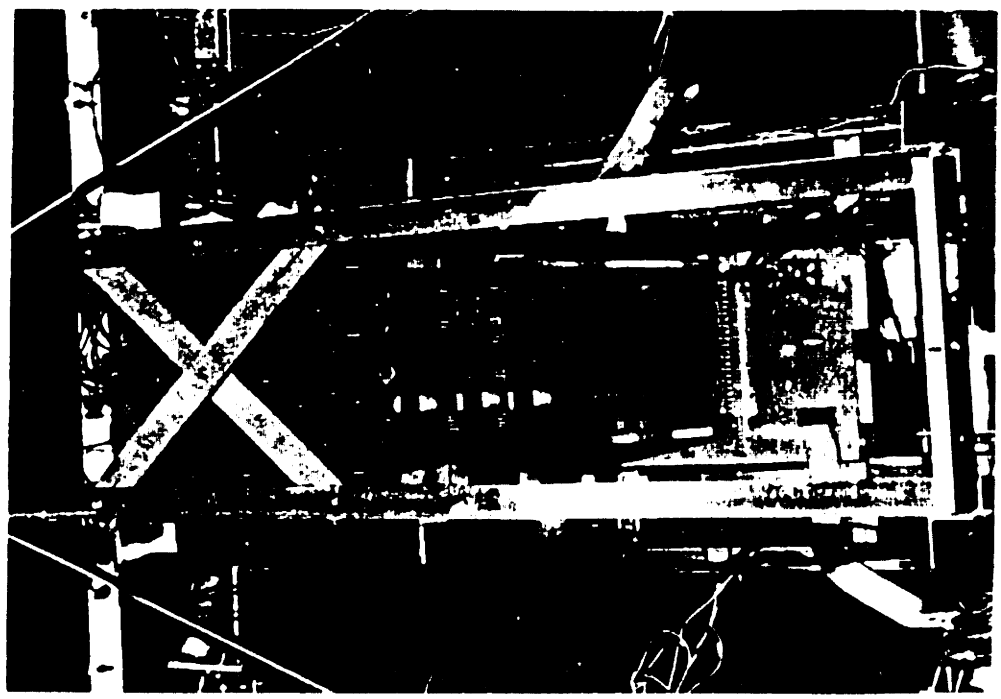


Figure B-3: Quadrupedal Robot: Electronics Modules

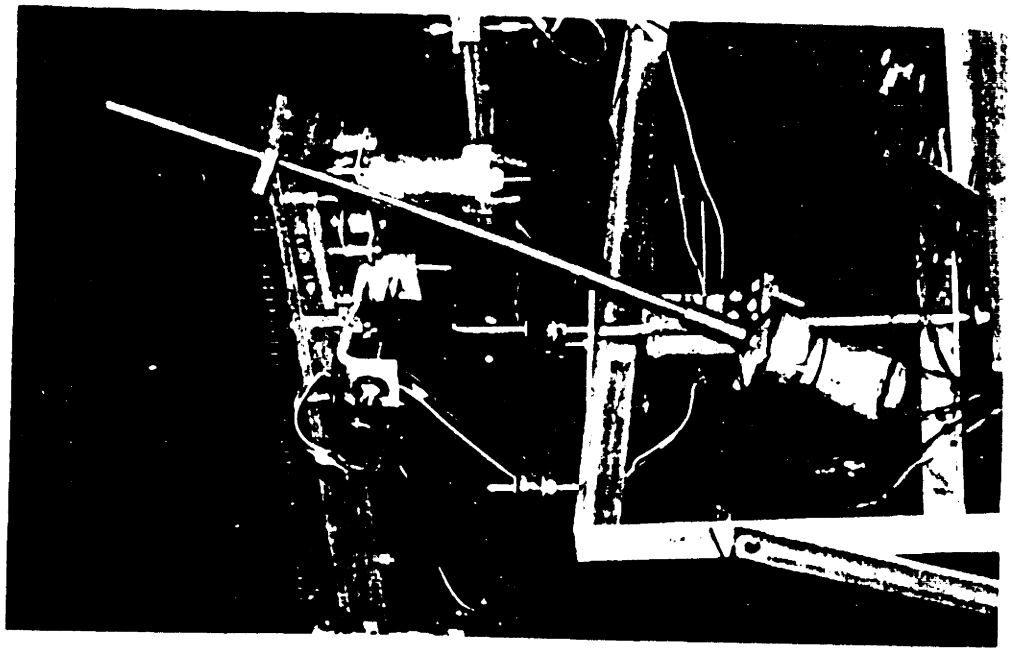


Figure B-4: Quadrupedal Robot: Hip Closeup

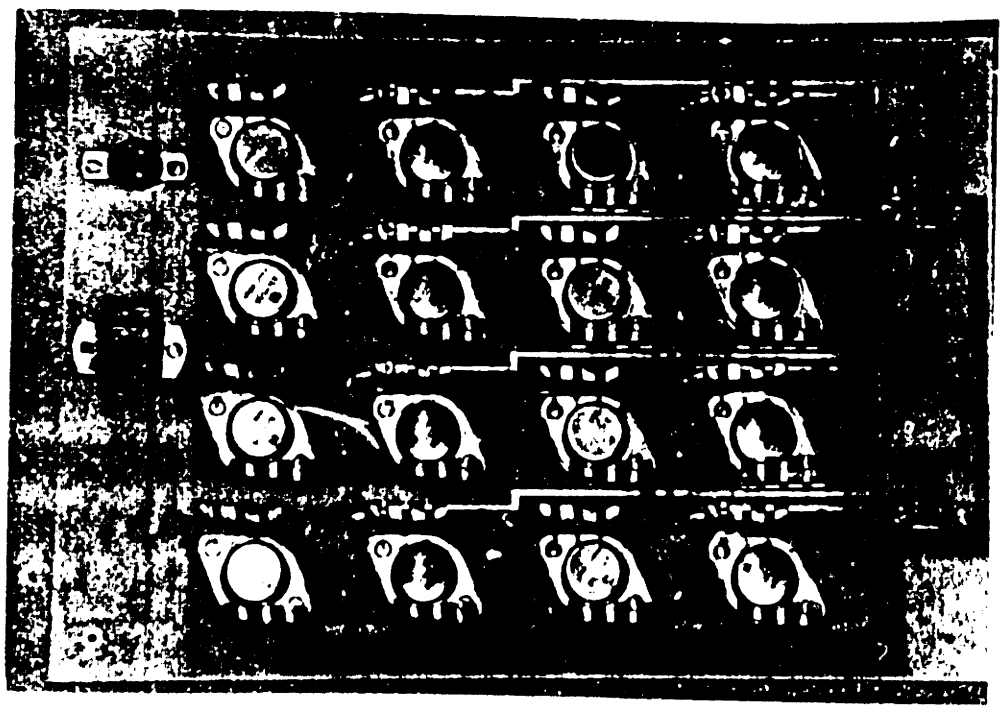


Figure B-5: Output Stage {1 of 3 boards}

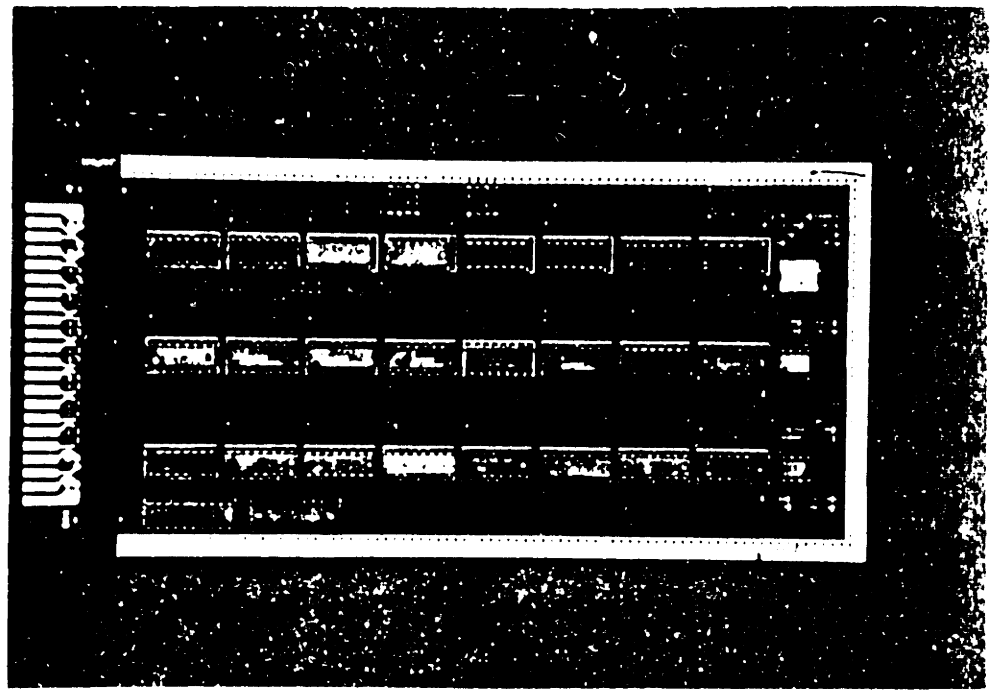
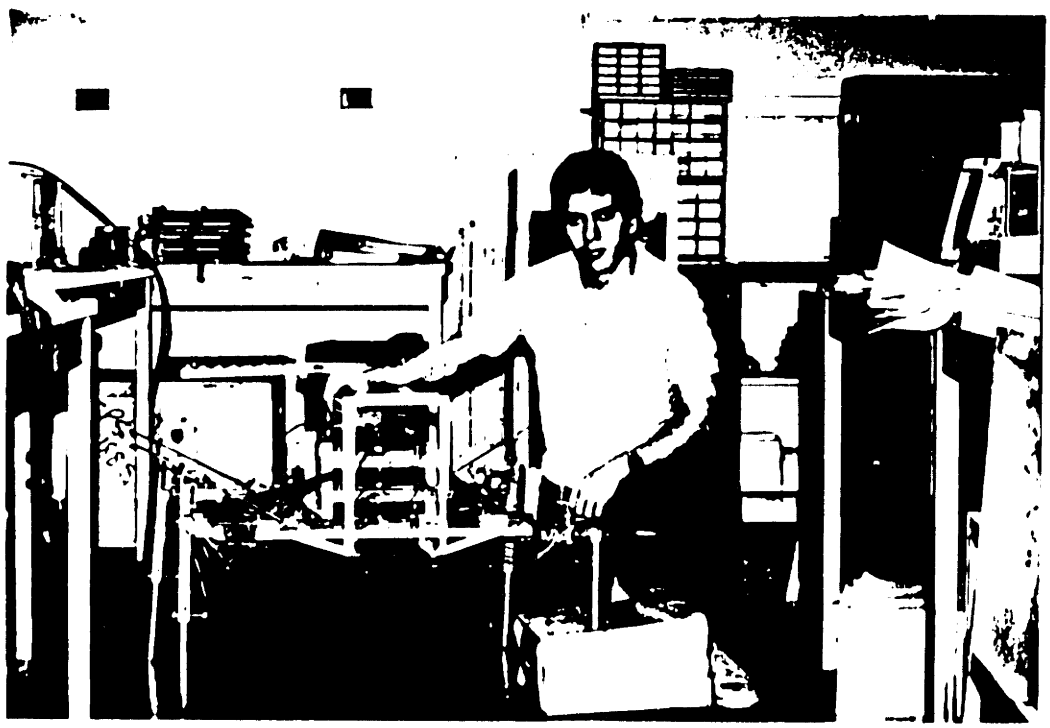


Figure B-6: PWM {1 of 3 boards}



REFERENCES

[Bak 86] Bak, David J.
Automated Care Station Frees Severely Handicapped.
Design News , October, 1986.

[Drogin 86] Drogin, Edwin M.
The Pentagon Funds a Walking Robot.
Defense Electronics , June, 1986.

[Fulsang 85] Fulsang, Ejner J. III.
Robots on the Battlefield.
Defense Electronics , October, 1985.

[Kreuter 79] Kreuter, Rodney A.
Controlling DC Power with Pulse Width Modulation.
Popular Electronics , June, 1979.

[McGhee 86] McGhee, Robert B and Waldron, Kenneth J.
The Adaptive Suspension Vehicle.
IEEE Control Systems Magazine , December, 1986.

[Murphy 84] Murphy, Marvin.
A Self Powered Undersea Robot Could Be Reality in 1 or 2 years.
Oil Daily , January, 1984.

[Propper 86] Propper, Welles.
Brainy Robot Hits the Mark: Mechanical Arm Helps Guide Neurosurgeons.
American Medical News , March, 1986.

[Raibert 83] Raibert, Marc H. and Sutherland E.
Machines That Walk.
Scientific American , January, 1983.

[Science News 85]
High Stepping Walking Machine.
Science News , July, 1985.

[Waber 84] Waber, David M.
Smart Carts Teckle In-plant Tasks.
ElectronicsWeek , August, 1984.