REMOTE DEPTH SURVEY OF THE CHARLES RIVER BASIN

BY

EVAN A. KARLIK

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2007

Signature of Author:_____
Department of Mechanical Engineering
11 May 2007

Certified by:_____
Franz S. Hover
Principal Research Engineer, Center for Ocean Engineering
Thesis Supervisor

Accepted by:_____
John H. Lienhard V
Professor of Mechanical Engineering
Chairman, Undergraduate Thesis Committee

REMOTE DEPTH SURVEY OF THE CHARLES RIVER BASIN

by

EVAN A. KARLIK

Submitted to the Department of Mechanical Engineering
on May 11, 2007 in partial fulfillment of the
requirements for the Degree of Bachelor of Science in
Mechanical and Ocean Engineering

ABSTRACT

Unmanned vehicles may provide more time- and cost-effective methods of gathering hydrographic survey data when compared to traditional, manned survey vessels. A remote-controlled unmanned surface vehicle (USV) was outfitted with a depth transducer for the purpose of conducting a depth survey of the Charles River Basin. Two windsurfer fins were added to the stern of the USV kayak for directional stability without significant drag, permitting a maximum vessel speed of 4.4 knots. A total of 1485 latitude-longitude GPS points with corresponding depth measurements were taken. Charles Basin data was plotted with ArcGIS software and used to create depth contours and three-dimensional surface plots of the river bottom. This prototype survey USV displays promise and could become readily feasible with further development and autonomy.

Thesis Supervisor: Franz S. Hover
Title: Principal Research Engineer, Center for Ocean Engineering

I. INTRODUCTION

Charts documenting the features of navigable waterways have been created and used by mariners for thousands of years, though the use of acoustic methods to measure water depth began after the sinking of the *Titanic* in 1912.[1] A hydrographic survey may be conducted to support a variety of goals: nautical charting, dredging and harbor maintenance, environmental studies of beaches and erosion, and bottom construction of piers, platforms, cable networks, and pipelines.[2] While details such as sediment types and thicknesses or bottom vegetation coverage can be included in a hydrographic survey, the one data type common to all surveys is depth.

Remote sensing is a prevalent, efficient method for collecting information on a subject without physical or intimate contact. While imaging satellites and seismographs passively capture electromagnetic radiation or motions from seismic activity, respectively, acoustic depth sounding is considered active remote sensing, in which a type of energy is directed at an object and the reflection received by a passive sensor. A depth sounder emits an acoustic pulse into the water; the time elapsed before the pulse returns can be used to find the range to the bottom and hence water depth.

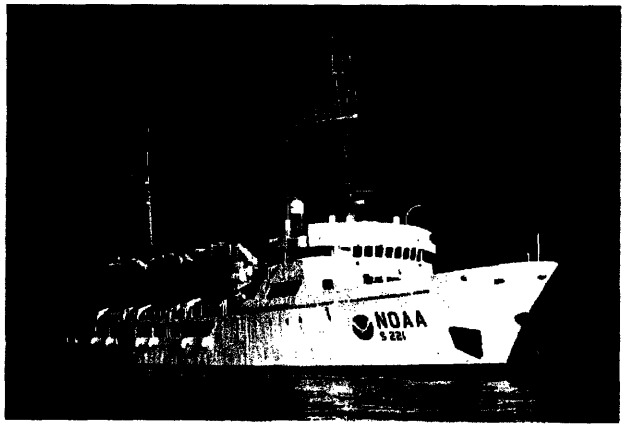

Figure 1 – The NOAA hydrographic survey ship *Rainier*, equipped with a 55-person crew.

Hydrographic survey ships, such as the NOAA vessel *Rainier*, are specially equipped to measure depth and bottom characteristics. Unmanned vehicles may provide more time-

and cost-effective methods of gathering similar data. Remote operated vehicles have been used for decades in the marine industry to explore shipwrecks, lay transocean cables, and inspect offshore oil platforms. A fleet of small, unmanned surface vehicles, controlled and tended by a larger, manned vessel, could cover a larger area in a shorter time at a fraction of the cost when compared to a traditional survey vessel.

This research aims to investigate the performance of an unmanned surface vehicle (USV), designed to carry out depth surveys while its human operator provides instructions from a remote location (Figure 2). The conclusion of this paper will evaluate the feasibility of using unmanned vessels similar to the prototype for large-scale hydrographic surveys.

Figure 2 – The unmanned surface vehicle (USV) on the Charles River Basin. The vessel was equipped with a depth transducer for conducting hydrographic surveys. The Boston skyline is in the background.

II. VESSEL OVERVIEW

The USV hull is a plastic, 3.7 meter long Wilderness Systems *Pungo 120* kayak; Table I gives its specifications as listed by the manufacturer. The kayak features a multi-chine hull with a full-length keel line for both maneuverability and stability.

Table 1 – *Pungo* 120 Specifications

Length	3.7 m
Beam	0.74 m
Draft	0.15 m
Max Loading	147 kg

The kayak was first augmented by the students in the 13.017/13.018 ocean engineering capstone design course in 2005, who outfitted the kayak with a trolling motor, servo motor, and two emergency stop buttons. The following year, the students in 2.017/2.109 added their own powering system, electronic compass, GPS, wireless modem, and Tattletale Model 8 computer. At the conclusion of the Fall 2006 term, the kayak was capable of following thrust and steering instructions sent via a 900 MHz wireless link, logging GPS positions and compass headings, and steering a line of specified heading through implementation of a feedback control routine. A complete description of this system is contained in the 2.019 Design of Ocean Systems final paper.[3]

III. DEPTH SOUNDER

A commercial off-the-shelf depth sounder was used in this project. Specifications of the Airmar DT800 P17 depth and temperature transducer are listed in Table 2.[4,5]

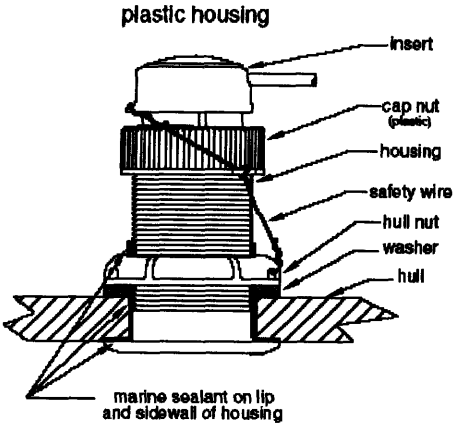

Figure 3 – Profile view of DT800 following installation.

The DT800 was mounted through the kayak hull approximately 2 cm from the keel line, for the lowest possible deadrise, and approximately two-thirds of the

total length from the bow, to distance it from any turbulence or bubbles produced by the motor. The installation procedure is thoroughly described in instructions from the manufacturer: after drilling a 1/8" pilot hole a 2" hole saw was used to create an opening in the hull.[6] The inside of the opening was sanded and cleaned before applying a layer of marine sealant around the rim of the hole as well as on the interior and exterior hull surfaces surrounding the opening. The transducer housing was inserted into the hole; a rubber washer was slid onto the housing from inside the hull after which the hull nut was tightened. After the sealant cured, the insert containing the transducer was placed into the housing, the cap nut tightened, and the safety wire attached. Figure 3 diagrams the DT800 after complete installation.

Table 2 – DT800 Specifications

Height	12.4 cm
Diameter	7.5 cm
Frequency	235 kHz
Beam Width at -3dB	14°
Rated RMS Power	200 W
Weight	0.6 kg
Output	NMEA 0183 4800 baud
Hull Deadrise	0-12° optimum 20° maximum
Depth Range	0.5 to 100 m

The depth transducer was powered by the vessel's 12V bus while ground and shield were connected to the boat power ground. The transducer NMEA+ output has a quiescent state of 0V with signals at approximately 4.5V, as observed on an oscilloscope. The Tattletale digital inputs require a 5V quiescent state with signals jumping to 0V; accordingly, the DT800 output was inverted with an LM324 op-amp (Figure 4).

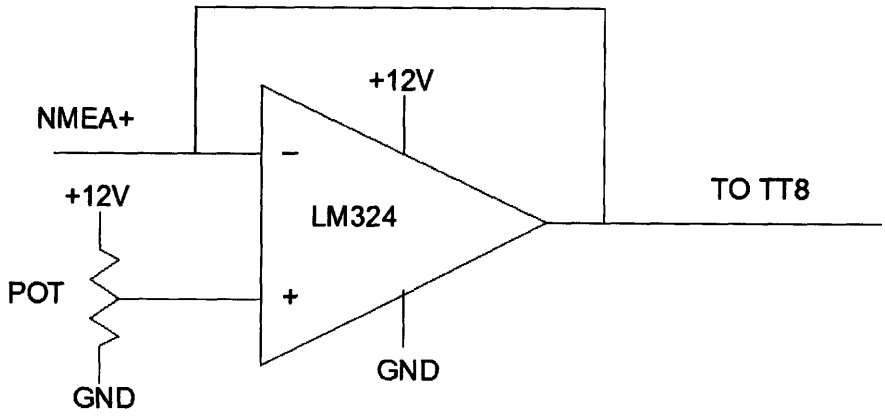

Figure 4 – Electrical schematic of the op-amp used to invert the depth transducer output.

The depth transducer NMEA+ output was used as the op-amp inverting input, while a potentiometer provided an adjustable non-inverting input in order to position the output to the Tattletale in the correct voltage range.

The DT800 outputs the following three NMEA strings once per second at 4800 baud:

`$SDDPT,2.7,*7C`	Depth (meters)
`$SDDBT,9.3,f,2.7,M,1.5,F*0D`	Depth (feet, meters, fathoms)
`$YXMTW,26.3,C*15`	Temperature (°C)

The Tattletale C++ function designed to read DT800 NMEA strings checks for the character 'P' as the fourth character after the '$.' It reads the following value as the depth in meters; the other two NMEA lines are ignored (see the Software Appendix for the full code). Figure 5 illustrates how the depth transducer is integrated with the host of other electronic sensors and actuators on board the kayak.

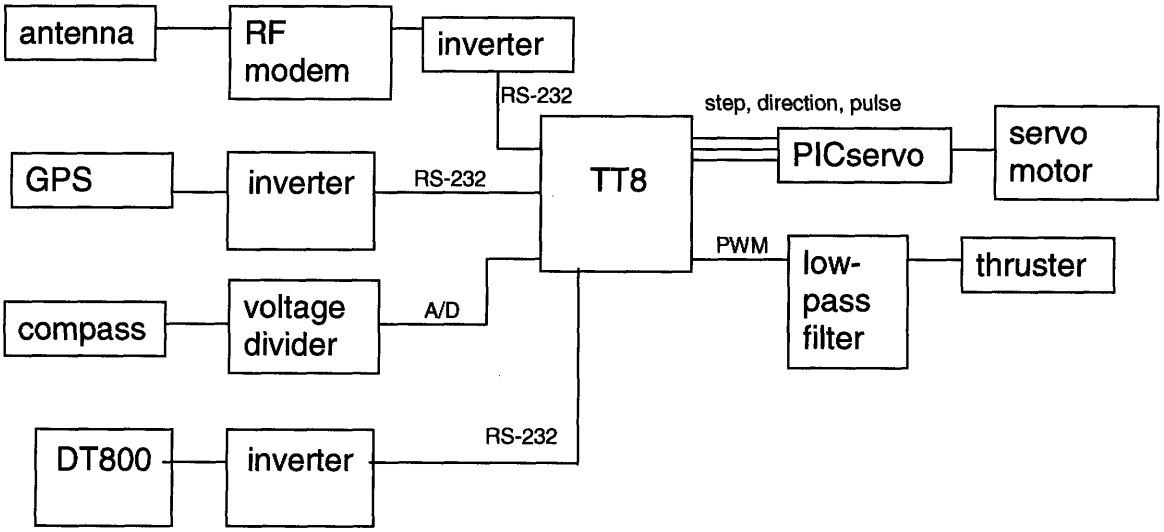

Figure 5 – Signal flow paths, illustrating how the navigation instruments, wireless hardware, and motors interface with the TT8 control system.

IV. DIRECTIONAL STABILITY

During previous field testing, a 5 kg metal elbow joint was towed behind the stern of the kayak on the end of an approximately 1.7 m line. The added drag provided a pivot point that enabled the boat to maintain directional stability when running into the wind as well as across and downwind. In an effort to improve the vessel's maximum speed, the elbow joint was removed and two Mistral windsurfer fins were bolted to the stern of the kayak (Figure 6). A generous amount of marine sealant was used to treat the exterior of the bolts, washers, and nuts, followed by butyl rubber tape. The inside of the kayak where the bolts penetrated was covered in butyl rubber tape and spray foam insulation to prevent flooding into the interior of the kayak. With a significant source of drag removed, the vessel was predicted to exhibit a higher maximum speed and thus be able to cover a larger survey area on limited battery capacity. GPS logging indicated a maximum vessel speed of 2.28 m/s, compared to a maximum speed of 1.49 m/s using the metal elbow joint – a significant improvement.

Figure 6 – Two windsurfer fins were bolted to the stern of the kayak to provide directional stability without significant drag.

V. OPERATION AND FIELD TESTING

Vessel launching and initialization greatly parallels the procedure outlined in the 2.019 final report. The 12V marine batteries would be removed to improve ease of launching.

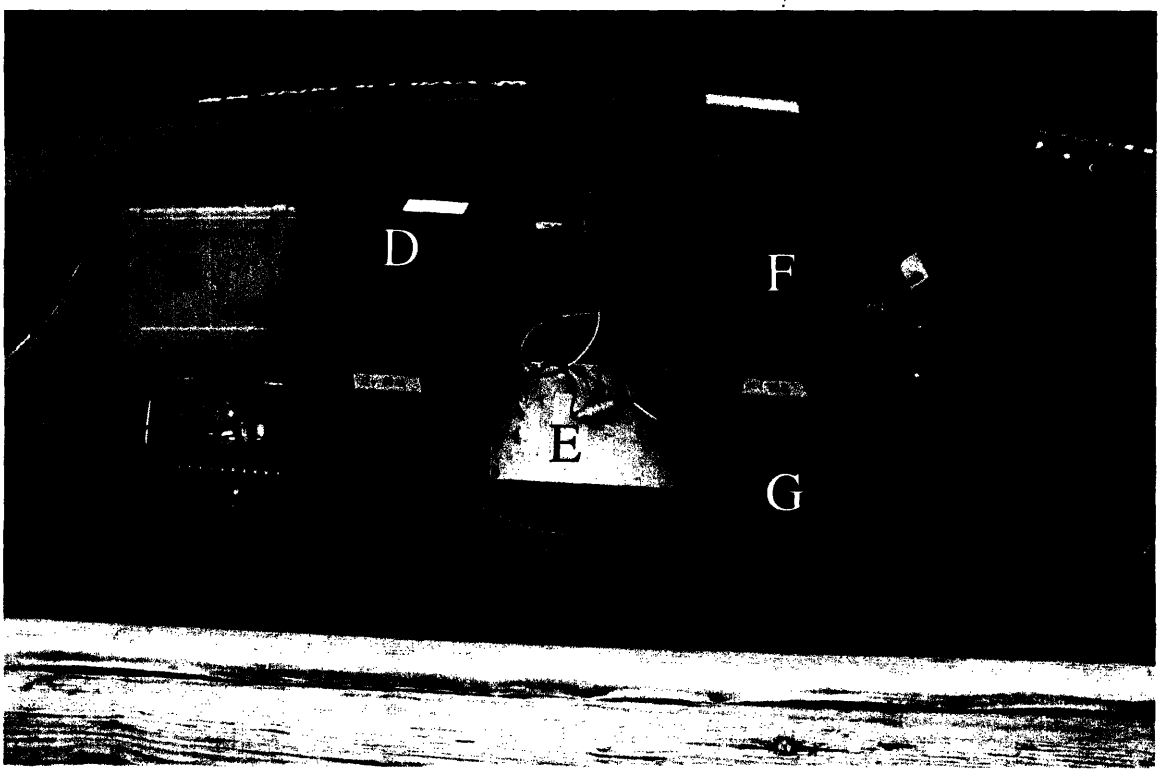

Figure 7 – The kayak interior before the spray skirt is spread over the cockpit. Key: A) bilge pump; B) Pelican box housing Tattletale, GPS, and inverter/voltage divider circuits; C) PICservo board; D) 12V battery; E) wireless modem and 9V voltage divider; F) second 12V battery, for providing 24V to the servo motor; G) 12V battery for running lights and bilge pump; H) main power switch; I) DT800 depth transducer; J) emergency stop button.

After the kayak had been placed in the water and secured to the Sailing Pavilion dock, the two batteries for boat power were connected and covered. A smaller, independent 12V battery for powering the lights and bilge pump and the 9V battery for GPS were also connected. The boat power switch was turned on and the Tattletale, connected by a serial cable to a laptop running the CrossCut serial interface program, was restarted. The thruster propeller would gradually accelerate, but typing the name of the remote control routine at the `PicoDOS>` prompt stopped the motor. At this stage, the TT8 would wait for a command line from the wireless modem.

The laptop's serial cable was then connected to the PICservo output and the operator started the NMCtest program to initialize the servo motor. The operator should select Position mode, press "GO," and clear the position error. Under I/O Mode, gains entered were $K_p = 10$, $K_d = 1000$, $K_i = 5$, and IL = 1000. Under servo parameters, deactivate switch action, set output mode to PWM and direction, and enable step and direction mode. Then press "Stop" and disconnect from the PICservo board. Applying a slight force to turn the servo motor gear by hand should be met with resistance from the motor, indicating the servo motor is functional. It is critical to ensure that the thruster motor is pointed directly towards the bow before initializing the servo motor.

The kayak's spray skirt was spread across the cockpit to protect the electronic components from chop and "green water" that might splash up onto the hull. The boat was then untied and pointed away from the dock to ensure it proceeded out into the open river upon power-up of its thruster.

MaxStream's X-CTU wireless terminal was used to command the vessel. Each motor command line is preceded by the character 'r,' to clear the buffer in the event it had been partially filled by erroneous characters. Six characters were then entered, corresponding to percentage of motor thrust (0-100%) and motor angle. The character 'e' approves the command for execution.

For example:

`r070-30e`	Set motor thrust to 70%, motor angle to -30°.
`r100+70e`	Set motor thrust to 100%, motor angle to +70°.

In the event of a typo, the user can press 'r' and reenter the correct command line. While typing 'r' will inform the program to wait for the complete command line, pressing any other keyboard character, such as another letter or the spacebar, will log the time, latitude, and longitude from GPS, the heading from the electronic compass, and the water depth from the DT800 depth transducer. The Tattletale keeps two separate data files, `data.txt` for hydrographic data, and `motor.txt` for a record of motor commands.

Field testing was conducted over two days out of the MIT Sailing Pavilion on the Charles River for a total of 1485 data points. The kayak was commanded from the dock by a laptop, XStream-PKG wireless modem connected to a USB port, and 1.7-meter Antenex long-range antenna, as well as from an outboard motorboat with the addition of a 12V battery and inverter to provide power to the laptop. Wireless commands were successfully executed even with the kayak on the opposite side of the Charles Basin, at a distance of approximately 500 meters.

The vessel's two marine 12V batteries experienced a voltage decrease of 0.29V over a 1.72 hour run period, corresponding to a voltage decay of 0.16V/hour. The calculated average power consumption ranges from 600 to 840 W, assuming the thruster is running at full power (50 A) and the servo motor runs between zero and full power (10 A).

VI. HYDROGRAPHIC DATA

The latitude-longitude data from GPS and corresponding depths were imported into ArcMap GIS software and overlaid onto images of the Charles River, local roads, and the footprints of Cambridge buildings (Figure 8). Despite efforts to keep the kayak proceeding on straight courses using corrective commands, it proved difficult to create a vessel track suggestive of the "lawnmower" pattern, used in surveys for its efficiency at covering a specified area in an organized manner without significant overlap. Wind, a slight offset in motor angle (on the order of several degrees) stemming from servo motor initialization, and yaw resulting from unbalanced windsurfer fins at the stern are liable to contribute to the difficulties of maintaining a straight-line course. A heading controller was successfully demonstrated during November 2006 testing (see feedback loop diagram and accompanying analysis in Section 6.5 of the 2.019 final report). The on-board electronic compass would require relocation, recalibration, or other refinement in order to run a pattern that requires 180-degree turns; it was observed that while the compass reports a 160° heading when facing out from the Sailing Pavilion, turning it gradually in a 180° arc causes the heading to decrease as appropriate until it erroneously returns to approximately 160° heading while facing northwest.

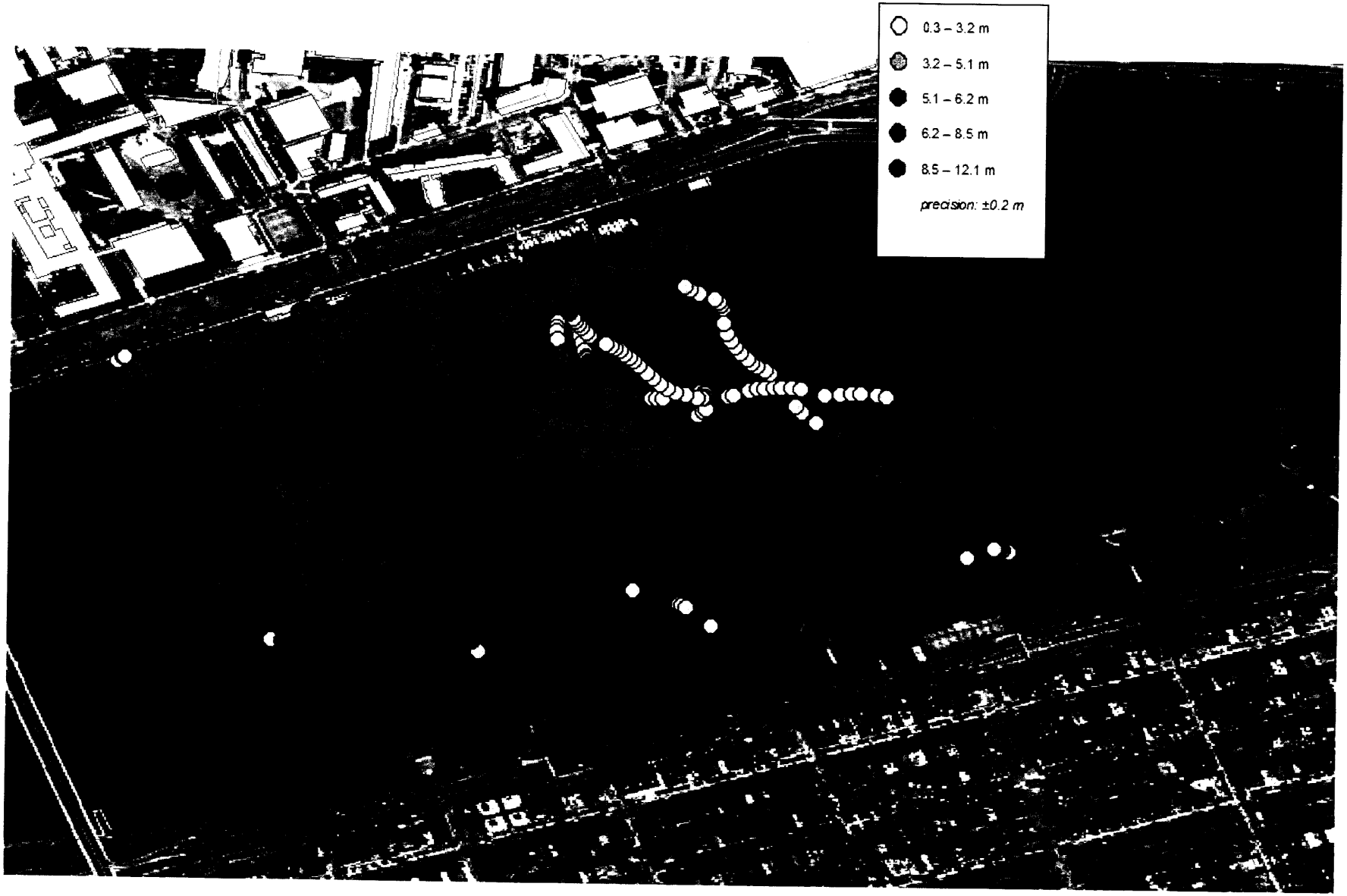

Figure 8 – Compilation of data points from field testing in the Charles River Basin. The location of each data point is fixed with a latitude-longitude position from GPS; depth in meters is indicated by the shade of blue (see key). Precision of the depth transducer is ±0.2 m, found by measuring depth repeatedly with the kayak secured to a dock.

To create depth contours, inverse distance weighted interpolation (IDW, Figure 9a) and Kriging interpolation (Figure 9b) were applied to the collected data.

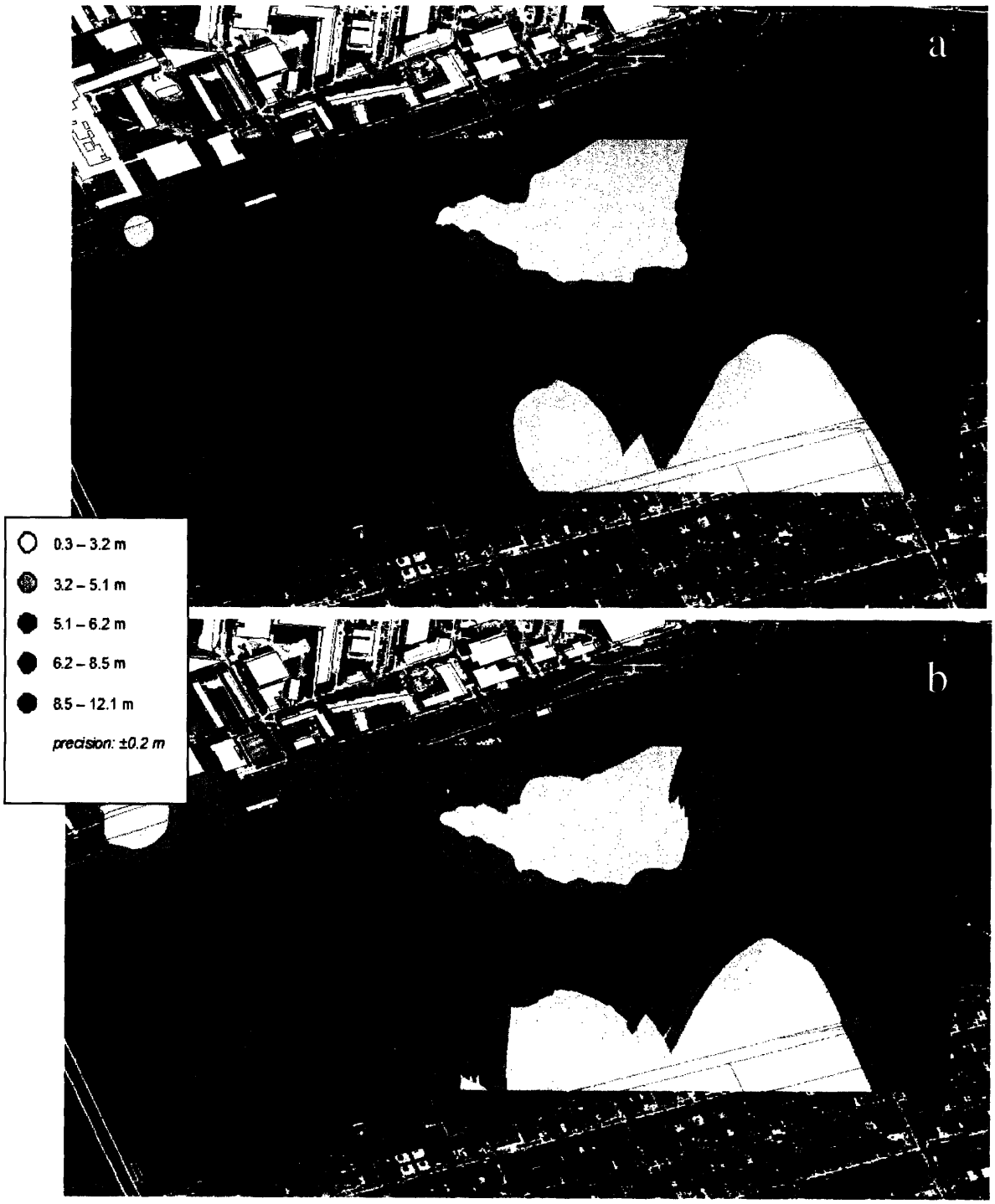

Figure 9 – Depth contours from interpolated data. **a.** IDW interpolation. **b.** Kriging interpolation. Darker shades of blue indicate deeper water; see key. Note these ArcGIS algorithms make no distinction of where the Charles River boundaries are situated. "Kriging is a geostatistical interpolation technique that considers both the distance and the degree of variation between known data points when estimating values in unknown areas."[7] IDW considers only distance; farther points have less influence. Both contour sets are similar, though IDW includes more small, isolated patches where depth differs from the surrounding region.

Lastly, ArcGIS software was used to create a triangulated integrated network (.TIN) file for three-dimensional plotting; data points are linked by triangles to create an approximate representation of the Charles Basin's bottom surface (Figure 10a-b).

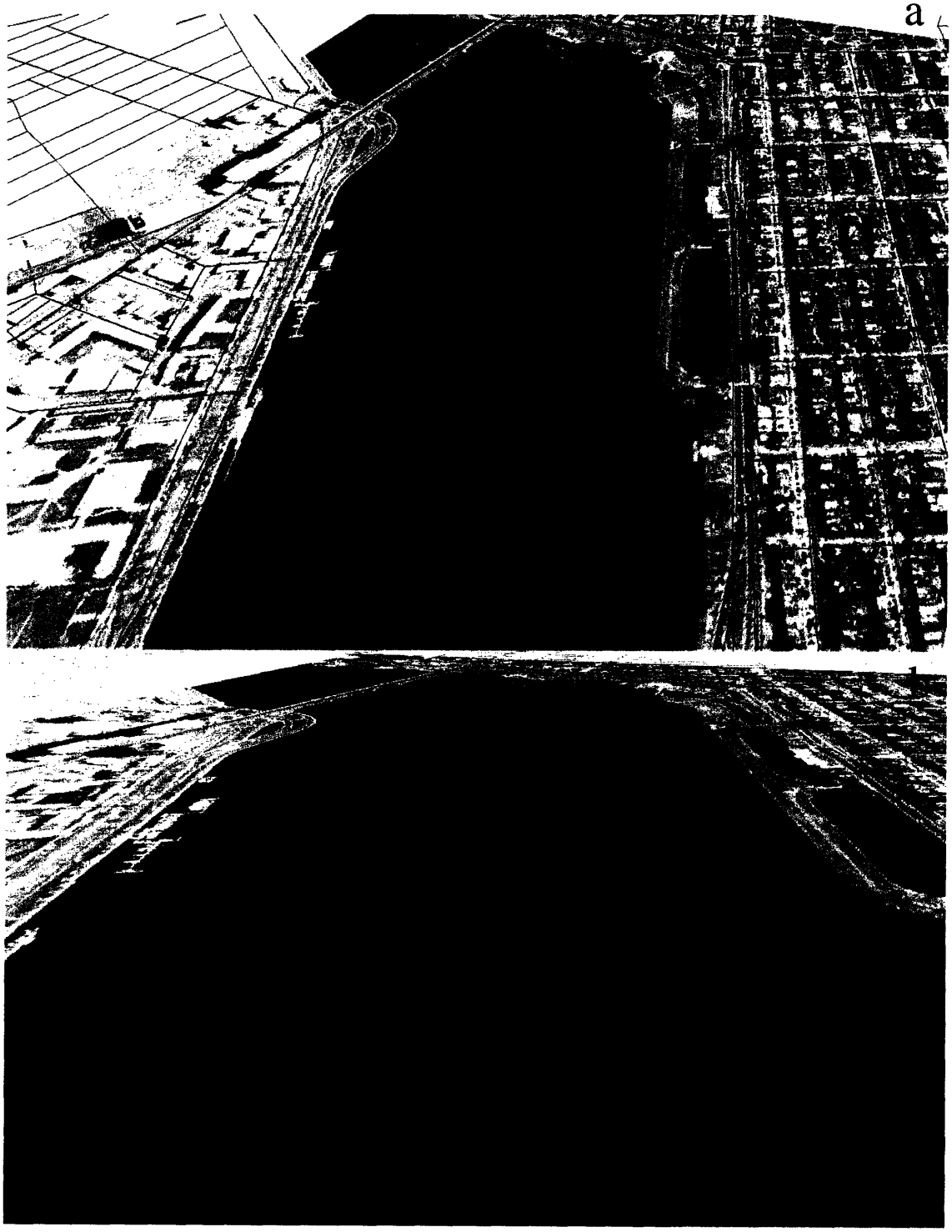

Figure 10 – Exaggerated three-dimensional surface plots from the triangulated integrated network. The TIN file is used to create an elevation mesh for the surface plot. Notice the deeper channel that appears to wind its way along the southern (Boston) side of the Charles Basin.

VII. CONCLUSIONS

The purpose of this project was to complete a depth survey of the Charles River Basin using a prototype USV to evaluate whether such technology could be used successfully in large-scale hydrographic surveys conducted by government or industry. Using a laptop and simple wireless communications equipment, a human operator, from both a dock by the shore and on an embarked motorboat, was successfully able to direct the kayak to make thrust and course adjustments and to collect depth data. The collected data and three-dimensional surface plots suggest a basin located approximately halfway between the MIT Sailing Pavilion and the opposite bank, and a deeper channel running along the Boston shore. Such information is in general agreement with data collected by the *Odyssey* series of autonomous underwater vehicles (AUVs) in the 1990s and by prior depth surveys by motorboat (see the ocean engineering PhD work of A. Bennett, 1995-8). USVs can certainly be incorporated into the hydrographic survey missions, as they can collect reliable data while offering significant cost and time savings over large, manned survey vessels.

Manual, remote direction of the kayak to hold a steady course was challenging, most likely because of environmental factors and the difficulty of attaining and keeping proper directional trim. Field tests carried out in November 2006 and reported in the 2.019 final paper demonstrate the ability of the kayak to maintain a constant compass heading using feedback control of the servo motor. Such feedback control should permit more uniform collection of data points, improving the accuracy of depth contours and surface plots. An even higher degree of autonomy could be implemented, to direct the vessel in a complete "lawnmower" pattern that includes the straight-line runs as well as turns. A marketable USV for surveys should also be capable of conducting longer missions through use of higher-capacity batteries, such as lithium-polymer which delivers four times the capacity per unit weight, or solar recharging. With the present batteries rated at 80 amp-hours, a mission can last only about 1.5 hours before the batteries require recharging.

Table 3 summarizes important vessel characteristics, and Figure 11 depicts the kayak leaving dock to conduct a survey mission.

Table 3 – General Vessel Specifications

Maximum Speed	2.28 m/s; 4.4 knots
Average Power Consumption	600 – 840 W
Calculated Maximum Mission Time	1.3 – 1.6 hours
Calculated Maximum Mission Distance	10.6 – 13.1 km
Precision of Depth Measurements	± 0.2 m
Depth Range	0.5 – 100 m

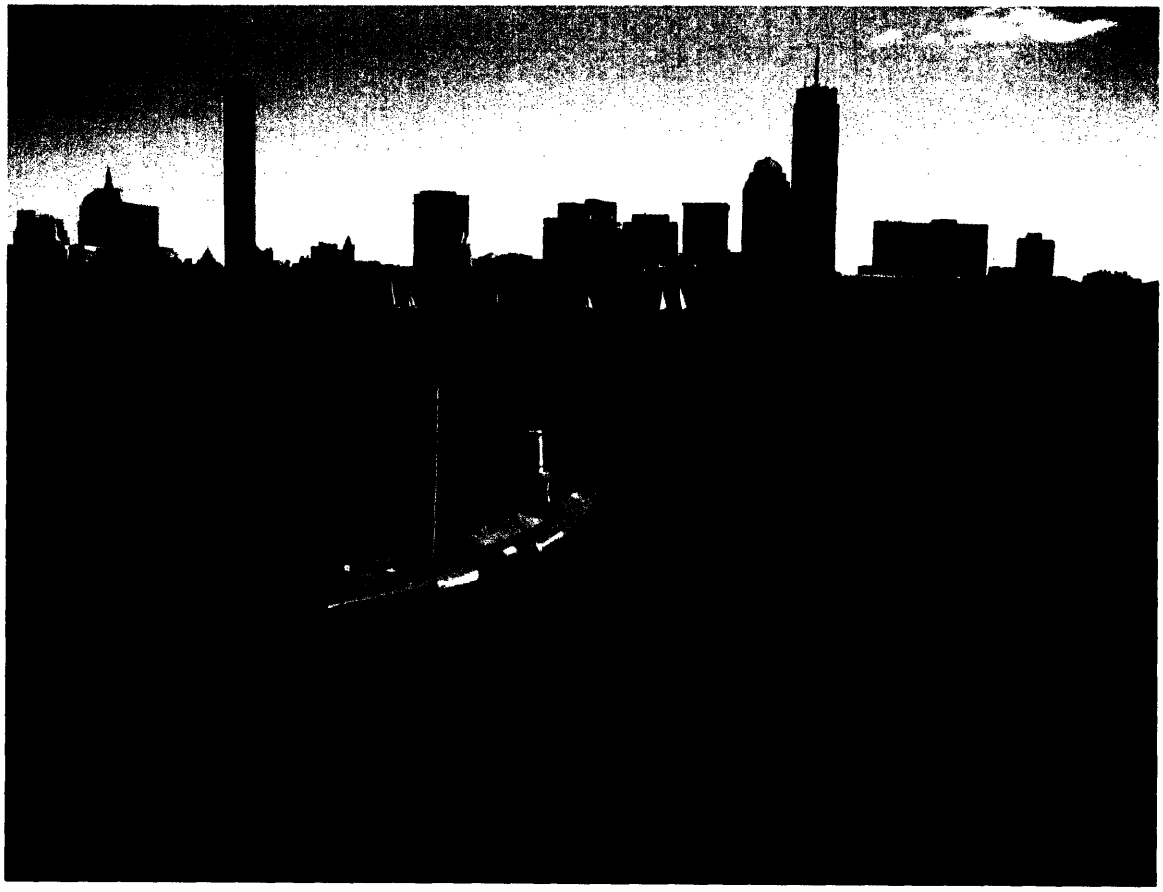

Figure 11 – Kayak departing the MIT Sailing Pavilion dock for a depth survey mission. The Boston skyline is seen in the distance, across the Charles River.

VIII. REFERENCES

1. National Oceanographic and Atmospheric Administration. "Hydrographic Survey Techniques." 12 December 2006. http://celebrating200years.noaa.gov/breakthroughs/hydro_survey/welcome.html#methods

2. National Ocean Service. "Hydrography and Hydrographic Surveys." 17 April 2007. http://chartmaker.ncd.noaa.gov/hsd/hydrog.htm

3. Chan, N., Clark, S., Gomez, C., Karlik, E., Sannino, J., and Young, N. "RoBoat: Design and Field Testing of an Autonomous Surface Vessel and Acoustic Tracking System." 2.109 Design of Ocean Systems. 13 December 2006.

4. Airmar. "Technical Data Catalog: 235 kHz – G." http://www.airmartechnology.com/uploads/CeramicDesignation/235G.pdf

5. Airmar. "D800 DT800 Series." http://www.airmartechnology.com/CatalogInter.asp?PageNo=11

6. Airmar. "Owner's Guide and Installation Instructions." 2003. http://www.airmartechnology.com/uploads/installguide/17-395-01%20r01%20ps.pdf

7. "Kriging Interpolation." http://lazarus.elte.hu/hun/digkonyv/havas/mellekl/vm25/vma07.pdf

IX. ACKNOWLEDGEMENTS

The author would like to express his gratitude to the following individuals and organizations:

- Dr. Franz Hover, who served as thesis advisor and also helped troubleshoot especially perplexing electronics issues.
- Christiaan Adams, who offered assistance numerous times for field testing, Tattletale maintenance, and GIS help.
- Victor Polidoro and James Morash of MIT Sea Grant.
- MIT Sea Grant, the MIT Center for Ocean Engineering, and the MIT Department of Mechanical Engineering.
- Fran Charles and the staff of the MIT Sailing Pavilion for storage space, motorboat access, and cheerful help in launching the kayak.
- Christina Gomez for help with electronics and James Sannino for assistance with launching and field testing.
- The students and staff of 13.017/13.018 (2005) and 2.017/2.019 (2006) for developing the kayak up to the stage at the start of this project.

X. APPENDIX – SOFTWARE

```
/*****************************************************************************
**      pd8main.c          Tattletale Model 8 Starter C File with PicoDOS Support
**
**      Copyright 1994-2002 ONSET Computer Corp.  All rights reserved.
**
**      Evan Karlik, senior thesis   Spring 2007
**      Remote Control Depth Survey  rxxx+xxe or rxxx-xxe for thrust and motor angle
**                                   spacebar or other key for GPS and depth logging
*****************************************************************************/

#include <TT8.h>                    // Tattletale Model 8 Definitions
#include <tat332.h>                 // 68332 Tattletale (7,8) Hardware Definitions
#include <sim332.h>                 // 68332 System Integration Module Definitions
#include <qsm332.h>                 // 68332 Queued Serial Module Definitions
#include <tpu332.h>                 // 68332 Time Processing Unit Definitions
#include <dio332.h>                 // 68332 Digital I/O Port Pin Definitions
#include <tt8pic.h>                 // Model 8 PIC Parallel Slave Port Definitions

#include <tt8lib.h>                 // definitions and prototypes for Model 8 library
#include <userio.h>                 // common convenient user I/O routines

#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <float.h>
#include <limits.h>
#include <locale.h>
#include <math.h>
#include <setjmp.h>
#include <sgtty.h>
#include <signal.h>
#include <stat.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "pwm.h"

#include <PicoDOS8.h>       // PicoDOS Interface definitions for Persistor CF8

/*****************************************************************************
**      main
*****************************************************************************/

#define GPS_CHAN            4                   //GPS channel on TT8
#define GPSTSBUFSIZ         128

#define CUE_CHAR            '$'         //start of data string
#define TARGET_OFFSET_TIME  7           //location of time data
#define  TARGET_OFFSET_LAT  16          //location of latitude data
#define TARGET_OFFSET_LONG  28          //location of longitude data
#define TARGET_OFFSET_SOG   45          //location of speed over ground
#define TARGET_OFFSET_COG   52          //location of course over ground
#define TARGET_LENGTH_TIME  6           //length of time data
#define TARGET_LENGTH_LAT   9           //length of latitude data
#define TARGET_LENGTH_LONG  10          //length of longitude data
#define TARGET_LENGTH_SOG   6           //length of speed over ground
#define TARGET_LENGTH_COG   5           //length of course over ground

#define DEPTH_CHAN 5                            //depth sounder channel on TT8
#define DEPTH_TSBUFSIZ      64
#define  TARGET_OFFSET_DEPTH_HEADER  4          //location of header SDDPT
#define  TARGET_LENGTH_DEPTH_HEADER  1
#define TARGET_OFFSET_DEPTH 7                   //location of depth data
```

```
#define TARGET_LENGTH_DEPTH 5          //length of depth data

#define             PWM_CHAN           1
#define             DIR_CHAN           2
#define             PULSE_CHAN         3

#define SERCHAN              0
#define TSBUFSIZ             8
#define TARGET_OFFSET        0
#define TARGET_LENGTH        7

//function declarations
void gps(void);
void depth(void);

void main()
{

int j = 0;
int q = 0;
char c[TSBUFSIZ];
char holder;
bool error = 0;
int thrust = 0;
int angle = 0;
int current_angle = 0;
int move = 0;

ulong tcr1 ;
int percent ;
int steps;
int i;
float xperiod, xtimehi ;
float Pwm_period ;
float Pwm_timehi ;

FILE *fout ;

// clears the files
fout = fopen("data.txt","w");          //opens data file (GPS and depth)
fprintf(fout," ");                     //prints space
fclose(fout) ;                         //closes

fout = fopen("motor.txt","w");         //opens data file (motor commands)
fprintf(fout," ");                     //prints space
fclose(fout) ;                         //closes

//
//        STANDARD TATTLETALE MODEL 8 LIBRARY & HARDWARE INITIALIZATION
//
InitTT8(NO_WATCHDOG, TT8_TPU);         // setup Model 8 for running C programs

//
//        STANDARD PICODOS KERNEL INITIALIZATION
//
InitCF8(CF8StdCS, CF8StdAddr);         // always follows the InitTT8() call
if (errno != 0)                        // look out for No Hardware (100) or No Media (101)
        printf("\n\n!!! InitCF8 failed, error %d !!!\n\n", errno);
if (errno == -1 || errno == PiDosNoHardware)    // no PicoDOS/Persistor !
        Reset();                       // any future PicoDOS calls would crash the TT8
TSerOpen(                    SERCHAN,
                                        MiddlePrior,
                                        INP,
                                        malloc(TSBUFSIZ+TSER_MIN_MEM),
                                        TSBUFSIZ,
                                        9600,'N',8,1);

thrust = 0;
percent=(int)(.60*(float)thrust);      //The output of the TPU channel will be 5V, but
                                       //because the motor cannot be given more than
                                       //3V, the multiplier is .60
```

```
if (percent>60)                         //precationary line to make sure TT8 does not
        percent=60;                     //output more than 60%, which would be 3V.
if (percent<0)                          //also precautionary code
        percent=0;

Pwm_period = 4000.*1;                   /* 1 ms period */
                                        /* (factor of 4000 is for 4MHZ clock) */
Pwm_timehi = 4000.*1*((float)percent*.01) ;

tcr1 = TPUGetTCR1();
xperiod = (float) tcr1 * (float) Pwm_period / 1e6;
xtimehi = (float) tcr1 * (float) Pwm_timehi / 1e6;

TPUSetPin(PWM_CHAN, 0);
TPUSetupPWM(PWM_CHAN, xtimehi, xperiod, MiddlePrior);

printf("Set TPU Channel %d to %d duty cycle.\n", PWM_CHAN, percent) ;

while(1)
        {
        holder = 'q';
        while(holder != 'r')
                {
                if(holder != '\n')
                        {
                        depth();
                        gps();
                        }

                holder = TSerGetByte(SERCHAN);
                printf("\nHolder is: %c         ",holder);
                }
        for (j=0;j<8;j++)
                {
                holder = TSerGetByte(SERCHAN);
                printf("\nHolder is: %c         ",holder);
                if(holder == 'r')
                        {
                        j = -1;
                        }
                else
                        {
                        if (holder == 'e')
                                {
                                printf("\nGot execute character.\n");
                                error = 0;
                                thrust = (c[0]-'0')*100+(c[1]-'0')*10+(c[2]-'0');
                                angle = (c[4]-'0')*10+(c[5]-'0');
                                if(c[3]=='-')
                                        {
                                        angle *= -1;
                                        }
                                //check for errors
                                if(thrust>100 || thrust<0)
                                        {
                                        error = 1;
                                        printf("\n Thrust error!");
                                        }
                                if(angle>90 || angle<-90)
                                        {
                                        error = 1;
                                        printf("\n Angle error!");
                                        }
        if(!error)
                {
                move = angle - current_angle;

                //INSERT THRUST COMMAND HERE arg(int move)

                percent=(int)(.60*(float)thrust);
```

```
            if (percent>60)     //precationary line
                    percent=60;
            if (percent<0)      //also precautionary code
                    percent=0;

            Pwm_period = 4000.*1;                   /* 1 ms period */
                                                /* (factor of 4000 is for 4MHZ clock) */
            Pwm_timehi = 4000.*1*((float)percent*.01) ;

            tcr1 = TPUGetTCR1();
            xperiod = (float) tcr1 * (float) Pwm_period / 1e6;
            xtimehi = (float) tcr1 * (float) Pwm_timehi / 1e6;

            TPUSetPin(PWM_CHAN, 0);
            TPUSetupPWM(PWM_CHAN, xtimehi, xperiod, MiddlePrior);

            printf("Set TPU Channel %d to %d duty cycle.\n", PWM_CHAN, percent) ;

            current_angle = angle;

            //INSERT SERVO COMMAND HERE arg(thrust 0-100)

            // setting the direction to go forward or backward

            if (move>0)
                    TPUSetPin(DIR_CHAN, 0 );
            else
                    TPUSetPin(DIR_CHAN, 1);
            // number of steps to move
            steps = 60*abs(move); //Measured steps per degree from 017
            for (i=0; i<steps; i++)
                    {
                    TPUSetPin(PULSE_CHAN, 0);
                    //DelayMilliSecs (40000);
                    TPUSetPin(PULSE_CHAN, 1);
                    // DelayMilliSecs (40000);
                    }

            //END OF SERVO COMMAND

            printf("\n\n Thrust: %d  Angle: %d  Moved: %d\n",thrust,angle,move);
            fout = fopen("motor.txt","a");          //open data file
            fprintf(fout,"\n %d  %d  %d  ",thrust,angle,move);
            fclose(fout);
            //depth();
            //gps();
                                        }

                                }


else
            {
            c[j] = holder;
            printf("Got char: %c",holder);

                                        }
                                }
                    }


//set all in buffer to zero
            for(q = 0; q<TSBUFSIZ; q++)
                    {
                    c[q] = '0';
                    }

//printf("\nSet to zero.\n\n");
```

```
        }

        TSerClose(SERCHAN);
        printf("\n\nSerial channel successfully closed.");

}
//
//      STANDARD TATTLETALE MODEL 8 PROGRAM EXIT
//


                //____ main() ____//


// DEPTH SOUNDER

void depth(){

long int i ;
long int imark=0;
char data;
char p[TARGET_LENGTH_DEPTH_HEADER];
char q[TARGET_LENGTH_DEPTH];
FILE *fout ;
double depth;
int got_depth;
long int depth_whole;                   //depth hundreds
long int depth_dec;//depth tenths
char c[DEPTH_TSBUFSIZ];                 //data string

printf("Depth sounder input port opened: %d (check: should be %d)\n",
                TSerOpen( DEPTH_CHAN,
                                        MiddlePrior,
                                        INP,
                                        malloc(TSBUFSIZ+TSER_MIN_MEM),
                                        TSBUFSIZ,
                                        4800,'N',8,1        ),
                                        tsOK);

//read from depth sounder
                got_depth = 0;
                while(got_depth != 1)
                {
                for (i=0;i<DEPTH_TSBUFSIZ && got_depth != 1;i++)
                {
                        data = TSerGetByte(DEPTH_CHAN);
                        printf(" %c ",data);

                        c[i] = data;        //reads in data string to c

                        if (c[i] == '$')//finds start of string, marked by $
                                {
                                imark = i;
                                }

if (i>=imark+TARGET_OFFSET_DEPTH_HEADER && i<imark+TARGET_OFFSET_DEPTH_HEADER +
TARGET_LENGTH_DEPTH_HEADER)
        {
        p[i-imark-TARGET_OFFSET_DEPTH_HEADER]=c[i];     //depth header
        }
if (i>=imark+TARGET_OFFSET_DEPTH && i<imark+TARGET_OFFSET_DEPTH+TARGET_LENGTH_DEPTH)
        {
        q[i-imark-TARGET_OFFSET_DEPTH]=c[i];  //depth
        }


if(i == imark+TARGET_OFFSET_DEPTH+TARGET_LENGTH_DEPTH)
        {
```

```
        if(p[0] == 'P')
                {

                got_depth = 1;
                fout = fopen("data.txt","a");          //open data file

                depth_whole = 0; depth_dec = 0;

                if(q[2] == '.') //depth < 99.9
                        {
                        depth_whole += (long int)(q[0]-'0')*10;
                        depth_whole += (long int)(q[1]-'0');
                        //skip decimal
                        depth_dec += (long int)(q[3]-'0');
                        }
                if(q[3] == '.') //depth < 999.9
                        {
                        depth_whole += (long int)(q[0]-'0')*100;
                        depth_whole += (long int)(q[1]-'0')*10;
                        depth_whole += (long int)(q[2]-'0');
                        //skip decimal
                        depth_dec += (long int)(q[4]-'0');
                        }
                else                            //depth < 9.9
                        {
                        depth_whole += (long int)(q[0]-'0');
                        //skip decimal
                        depth_dec += (long int)(q[2]-'0');
                        }
                depth = (float)depth_whole+((float)depth_dec/10.);
                fprintf(fout,"\n %lf  ",depth);        //write to file
                printf("The depth is: %lf meters ",depth);//write to screen
                fclose(fout) ;                 //close data file
                                                        }
                                                }


                }
                }

TSerClose(DEPTH_CHAN);
}


// GPS LOGGING

void gps(){

long int i ;
int j;
long int imark=0;
char data;

long int ilat_deg;          //latitude degrees
long int ilat_min;          //latitude minutes
long int ilat_mindec;       //decimal minutes

long int ilong_deg;         //longitude degrees
long int ilong_min;         //longitude minutes
long int ilong_mindec;      //decimal minutes

long int ic;
long int ic_dec;
char c[GPSTSBUFSIZ];        //data string
char d[TARGET_LENGTH_TIME]; //time string
char e[TARGET_LENGTH_LAT];  //latitude string
char f[TARGET_LENGTH_LONG]; //longitude string
char g[TARGET_LENGTH_SOG];  //speed string
char h[TARGET_LENGTH_COG];  //course string
double lat_min;
double long_min;
```

```
double time;

double R = 6371000;          //mean Earth radius
double boat_lat;
double boat_long;
//coordinate in middle of Charles River:
double goal_lat = 42.356633;
double goal_long = 71.087611;
double range;                //range (m) to GPS waypoint
double delta_lat;
double delta_long;
double delta_x;
double delta_y;
double theta;       //bearing to GPS waypoint

float voltage; float heading;

FILE *fout ;

// clear the file
fout = fopen("data.txt","a");          //opens data file
                                       //closes

printf("Serial input port opened: %d (check: should be %d)\n",
                    TSerOpen( GPS_CHAN,
                                                MiddlePrior,
                                                INP,
                                                malloc(GPSTSBUFSIZ+TSER_MIN_MEM),
                                                GPSTSBUFSIZ,
                                                9600,'N',8,1        ),
                                                tsOK);
printf("TIME            LAT               LONG           RANGE              BEARING
COMP RAW     COMP HEADING\n");

          for(j=0;j<1;j++)
          {

                    for (i=0;i<GPSTSBUFSIZ;i++)
                    {
                              data = TSerGetByte(GPS_CHAN);
                              //printf("\nGot char: %c",data);

                              c[i] = data;        //reads in data string to c

                              if (c[i] == '$')//finds start of string, marked by $
                                        {
                                        imark = i;
                                        }
if (i>=imark+TARGET_OFFSET_TIME && i<imark+TARGET_OFFSET_TIME+TARGET_LENGTH_TIME)
          {
          d[i-imark-TARGET_OFFSET_TIME]=c[i];    //time
          }
if (i>=imark+TARGET_OFFSET_LAT && i<imark+TARGET_OFFSET_LAT+TARGET_LENGTH_LAT)
          {
          e[i-imark-TARGET_OFFSET_LAT]=c[i];     //latitude
          }
if (i>=imark+TARGET_OFFSET_LONG && i<imark+TARGET_OFFSET_LONG+TARGET_LENGTH_LONG)
          {
          f[i-imark-TARGET_OFFSET_LONG]=c[i];    //longtitude
          }
if (i>=imark+TARGET_OFFSET_SOG && i<imark+TARGET_OFFSET_SOG+TARGET_LENGTH_SOG)
          {
          g[i-imark-TARGET_OFFSET_SOG]=c[i];     //speed over ground
          }
if (i>=imark+TARGET_OFFSET_COG && i<imark+TARGET_OFFSET_COG+TARGET_LENGTH_COG)
          {
          h[i-imark-TARGET_OFFSET_COG]=c[i];     //course over ground
          }


if(i == imark+TARGET_OFFSET_COG+TARGET_LENGTH_COG)
```

```
{

//calculates time (sec)
ic = 0; ic_dec = 0;
ic += (long int) (d[0]-'0')*3600*10;
ic += (long int)(d[1]-'0')*3600;
ic += (long int)(d[2]-'0')*60*10;
ic += (long int)(d[3]-'0')*60;
ic += (long int)(d[4]-'0')*10;
ic += (long int)(d[5]-'0');
//skip decimal
ic_dec = 0;
ic_dec += (long int)(d[7]-'0');

//printf("Time: %ld seconds.\n  ",ic);
time = (float)ic+((float)ic_dec/10.);
fprintf(fout,"%lf  ",time);           //write to file
printf("%lf  ",time);                 //write to screen


//calculates latitude
ilat_deg = 0;
ilat_deg += (long int) (e[0]-'0')*10;
ilat_deg += (long int)(e[1]-'0')*1;
ilat_min = 0;
ilat_min += (long int) (e[2]-'0')*10;
ilat_min += (long int)(e[3]-'0')*1;
ilat_mindec = 0;
ilat_mindec += (long int) (e[5]-'0')*10000;
ilat_mindec += (long int)(e[6]-'0')*1000;
ilat_mindec += (long int)(e[7]-'0')*100;
ilat_mindec += (long int)(e[8]-'0')*10;
ilat_mindec += (long int)(e[9]-'0')*1;
lat_min = (float)ilat_min+((float)ilat_mindec/100000.);
fprintf(fout, "%ld  %lf  ", ilat_deg, lat_min); //write to file
printf("%ld o %lf' N  ", ilat_deg, lat_min);    //write to screen


//longitude
ilong_deg = 0; ilong_min = 0; ilong_mindec = 0;
ilong_deg += (long int) (f[0]-'0')*100;
ilong_deg += (long int)(f[1]-'0')*10;
ilong_deg += (long int)(f[2]-'0')*1;
ilong_min += (long int) (f[3]-'0')*10;
ilong_min += (long int)(f[4]-'0')*1;
//skip 5, decimal
ilong_mindec += (long int) (f[6]-'0')*10000;
ilong_mindec += (long int)(f[7]-'0')*1000;
ilong_mindec += (long int)(f[8]-'0')*100;
ilong_mindec += (long int)(f[9]-'0')*10;
ilong_mindec += (long int)(f[9]-'0')*1;
long_min = (float)ilong_min+((float)ilong_mindec/100000.);
fprintf(fout, "%ld  %lf  ",ilong_deg,long_min); //write to file
printf("%ld o %lf' W  ",ilong_deg,long_min);    //write to screen

boat_lat = (float)ilat_deg+lat_min/60.;
boat_long = (float)ilong_deg+long_min/60.;

delta_lat = goal_lat - boat_lat;
delta_long = goal_long - boat_long;
delta_y = delta_lat*60*1852;
delta_x = delta_long*60*1852*cos(boat_lat);

range = sqrt(delta_x*delta_x+delta_y*delta_y);
theta = atan2(delta_y,delta_x);
theta = theta*180/3.14159265;         //convert to degrees
theta = 90 - theta;                   //convert to compass bearing



fprintf(fout, "%lf  %lf  ",range,theta);
```

```
        printf("%lf  %lf  ",range,theta);

        voltage = AtoDReadMilliVolts(7);
        heading = voltage/1000.*92.3;
        printf("%f    %f\n\n",voltage,heading);
        fprintf(fout,"%f    %f",voltage,heading);
        fclose(fout) ;              //close data file
                                                    }

                }

        //printf("\n");
        }
TSerClose(GPS_CHAN);


}
//
//      STANDARD TATTLETALE MODEL 8 PROGRAM EXIT
//


                //____ main() ____//
```

```
#include "pwm.h"

/*****************************************************************************
**      TPUSetupPWM                         Setup TPU channel for Pulse-Width
Modulation
**
**      Notes:
**              pwmper = period in tcr1 cycles [call TPUGetTCR1() for current value]
**              pwmhi = time high in tcr1 cycles
**              priority = LowPrior, MiddlePrior, or HighPrior [defined in tpu.h]
**              read about value limitations in the TPU Reference Manual
*****************************************************************************/
void TPUSetupPWM(short chan, short pwmhi, short pwmper, short priority)
        {

        CHANPRIOR(chan, Disabled);                  /* stop what its doing */
        if (priority == Disabled)                   /* just shutting it down */
                return;
        *CIER &= ~(1 << chan);              /* don't want interrupts enabled */
        FUNSEL(chan, PWM);                  /* channel function select */
        PRAM[chan][0] = OutputChan | NoChangePAC | (pwmhi ? ForceHigh : ForceLow);
        PRAM[chan][2] = pwmhi;                                  /* time hi */
        PRAM[chan][3] = pwmper;                                 /* period */

        HOSTSERVREQ(chan, 2);                                   /* issue request */
        CHANPRIOR(chan, priority);                  /* set channel priority */
        while (HOSTSERVSTAT(chan) & 3)              /* await reply */
                ;

        }       /* TPUSetupPWM() */
```

```
#ifndef PRECOMPHDRS
#include <TT8.h>                    /* Tattletale Model 8 Definitions */
#include <tat332.h>                 /* 68332 Tattletale (7,8) Hardware Definitions */
#include <sim332.h>                 /* 68332 System Integration Module Definitions */
#include <tpu332.h>                 /* 68332 Time Processing Unit Definitions */

#include <tt8lib.h>                 /* definitions and prototypes for Model 8 library
*/

#include <stdio.h>
#include <stdlib.h>

#include        <TT8.h>               /* Tattletale Model 8 Definitions */
#include        <tat332.h>            /* 68332 Tattletale (7,8) Hardware */
#include        <sim332.h>            /* 68332 System Integration Module */
#include        <qsm332.h>            /* 68332 Queued Serial Module */
#include        <tpu332.h>            /* 68332 Time Processing Unit  */
#include        <dio332.h>            /* 68332 Digital I/O Port Pin */
#include        <tt8pic.h>            /* Model 8 PIC Parallel Slave Port */
#include        <tt8lib.h>            /* Model 8 library */
#include        <userio.h>            /* Others ... */
#include        <assert.h>
#include        <ctype.h>
#include        <errno.h>
#include        <fcntl.h>
#include        <float.h>
#include        <limits.h>
#include        <locale.h>
#include        <math.h>
#include        <setjmp.h>
#include        <sgtty.h>
#include        <signal.h>
#include        <stat.h>
#include        <stdarg.h>
#include        <stddef.h>
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <time.h>
/*#include        <pdos8.h>          /* if this fails, try PicoDCF8.h */
#include        <PicoDCF8.h>         /* if this fails, try pdos8.h */

#include <userio.h>
#endif

#define         PWM_CHAN              1

#define                     ForceHigh          0x01
#define                     ForceLow           0x02
#define                     NoChangePAC                0x10
#define                     OutputChan                 0x80

void TPUSetupPWM(short chan, short pwmhi, short pwmper, short priority);
```