University of Florida

Department of Electrical Engineering

EEL5666

Intelligent Machine Design Lab

Submergency: An Autonomous Submarine

Eric Anderson

Instructor: Dr Arroyo

9 December 8, 1998

TABLE OF CONTENTS

ABSTRACT

Following is a discussion of the design of an autonomous submarine. The submarine employs two types of sensors pressure and heading. The sensors control the depth and forward motion of the sub through the ICC11 programming language. The microprocessor uses the pressure sensor to manipulate the vertical motion motors to maintain a specific depth and an electronic compass to maneuver the horizontal motors for following a course. The closing discussion explains further continuation of the development of a more fully autonomous underwater agent.

EXECUTIVE SUMMARY

Submergency is an autonomous underwater agent that was designed to navigate a body of water freely and unaided. The submarine causes challenges due to the capability of three-axis motion. The platform is a large tube that is carefully weighted to neutral buoyancy. This allows the small motors to maneuver the sub in all directions.

Submergency's principal navigation system is the electronic compass. The compass outputs a relative heading with use of the earth's magnetic field. The heading is given by a value between 0 and 359 from a relative north. Through which Submergency is capable of following any laid out course. Also the pressure sensor allows the sub to exist at any depth with accuracy up to one inch in shallow waters.

The entire system is managed by a singe microprocessor that is able to control every motion by slight increments. ICC11 programmed functions allow Submergency to dive, surface, speed up, slow down, turn slightly or large turns by degrees. The main program continuously has the system checking for proper heading and depth that adjusts quickly to any deviation of the set parameters.

INTRODUCTION

My primary goal of the IMDL class was to design a fully autonomous submarine own as Submergency. The physical platform was crucial factor in the complexity of the design. The platform had the ability of removing or adding a large difficulty factor in the design of the robot. To increase the simplicity of the platform aerodynamics was not a consideration in the development of the body. To remove the need of ballast tanks a neutrally buoyant platform was chosen. Therefore the structure had to have the ability to adjust its weight by small increments. Submergency's central behavior is to follow a course in a three dimensional space. This is accomplished with the use of a pressure sensor and an electronic compass.

The following discussion will describe Submergency in nine separate sections. *Integrated System* describes the how each piece is connected into a whole. *Mobile Platform* characterizes the physical structure of the system. *Actuation* covers Submergency's motion control system. *Sensors* specifies the type of sensor and its' integration into the motion control system. *Behavior* denotes the few behaviors that Submergnecy exhibits. *Experimental* Layout and Results examines the development of components into the integrated system. *Conclusion* summarizes the completion and incompletion of goals and the further developments that may help finalized all of the goals. *Documentation* provides a complete list of references of vendors test code and a final demonstration code.

INTEGRATED SYSTEM

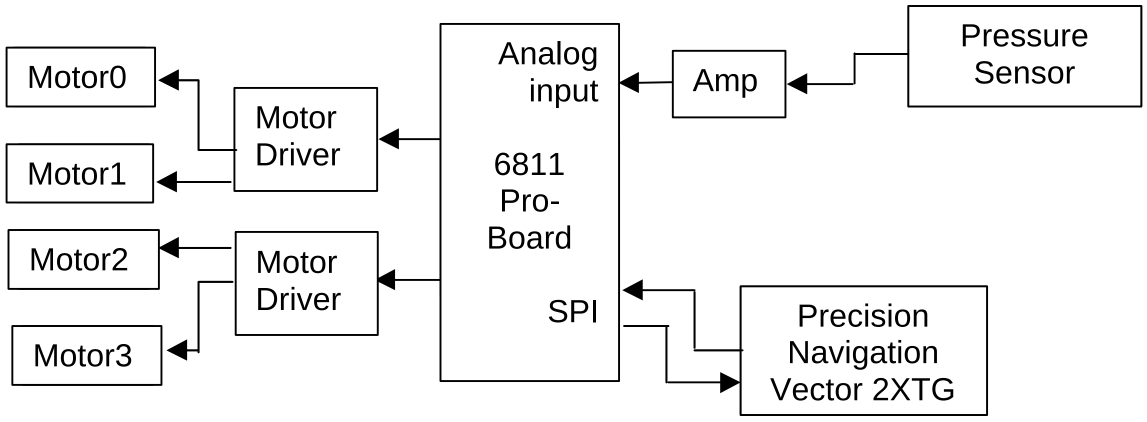

Figure 1 Integrated System

Submergency is comprised of two input systems and four independently controlled motor driver circuits. Each motor driver board, as seen in figure 1, contained two isolated motor driver circuits. The system was controlled by the 68HC11 Pro-board a product of the Mektonix Company. The Pro-board was used because of its small size and 32K of ram. Also the Pro-board has most of the same built-in functions as the Mektronix ME11 expansion board. All of the electronic components are mounted on a metal plate that slides in and out of the main tube on a rack system. This allows for a clean routing of all the wires and keeps the electronic components far away from any water that may seep in.

The 68HC11 interfaces with the two inputs with the use two separate systems. The pressure sensor readings amplified and then sent into the analog port. The compass module interfaces with the SPI system. The compass is set to master and the 68HC11 are the slave. The microprocessor just polls compass and the two bytes are taken in the SPI system. The motors are controlled by ICC11 functions that accept the sensor reading and adjust the motors accordingly.

MOBILE PLATFORM

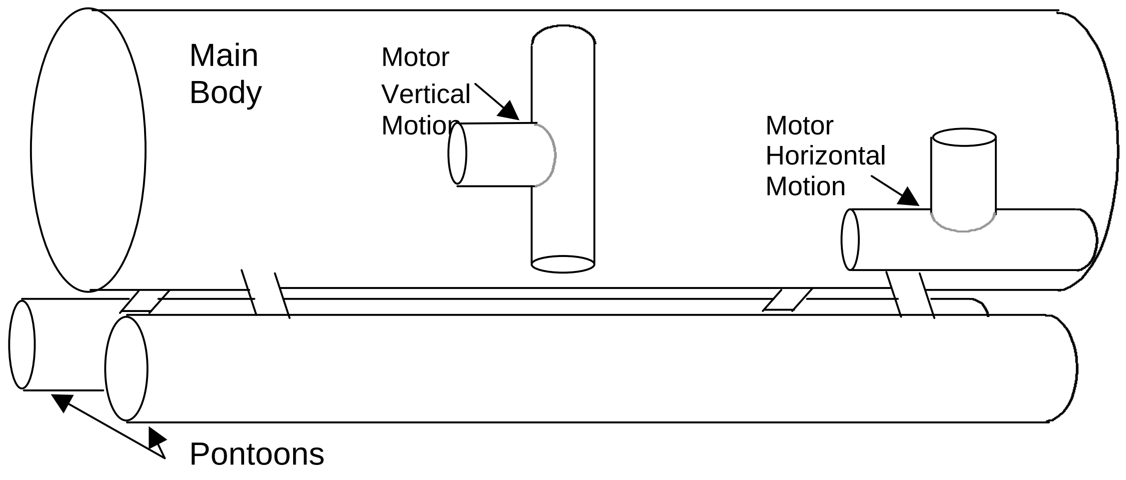

Figure 2

The platform for the sub is shown in Figure 2. The main body of the sub is made of six-inch clear acrylic tubing. The clear tubing allowed debugging to become easier. The pontoons serve many purposes. They allow the sub to land on the bottom of a body of water without damaging any of the systems. The pontoons are be used to weight down and balance the sub so that the overall system will be neutrally buoyant and horizontally stable. The pontoons are made up of 1½ '' schedule 40 steel rods. A ¼ inch threaded rod with washers is placed inside of each pontoon. This allowed for weight to be added, in the form of washers, in small increments and the weight could be moved along the threaded rod to balance the system. The weight of the sub is placed in the pontoons and the bottom of the main body to remove the need for roll and pitch sensors. The main body has one permanently sealed end and one removable end cap. The removable end cap is sealed by a gasket and locking clamps.

ACTUATION

Battery Power

Submergency has a total of seven batteries for normal operation. A 12volt lead acid battery is used to power the motor driver coils and is regulated to 5volt and 8volts for the 68HC11 and other circuits. Also there is three 7.2volt 1200ma/h battery packs to run the motors. Two power the forward motors and the other is for the vertical motors. The other three batteries are used to power the pressure sensor's circuits.

Motor drivers

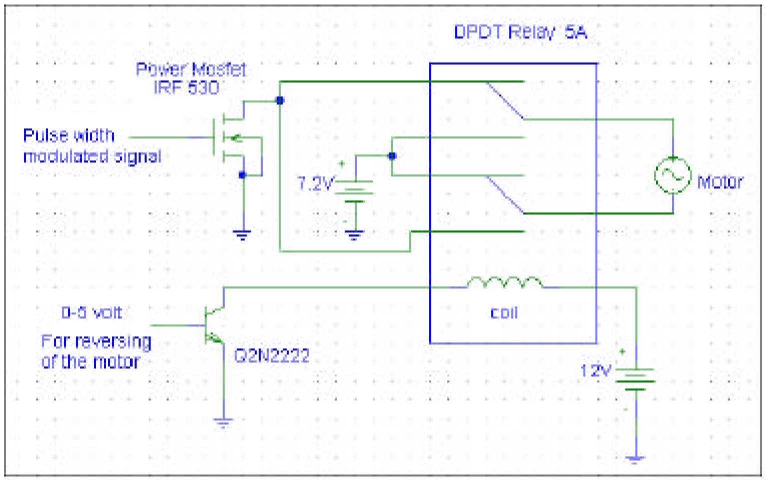

Figure 3 Motor Driver circuit

Submergency has four of the motor driver circuits seen in Figure 3. The circuit uses a Power MOSFET IRF 530 to switch the on and off the motor at a high rate. The switching controls the speed at which the motor runs. The pulse is generated from the output compare function of the 68HC11 board. The 0-5volt input to the circuit if from a memory mapped output port from the 68HC11 board. A 0-volt input was used for forward running of the motors (chosen to conserve the 12volt-battery power) and a 5-volt input would reverse the motor.

Motors

The motion of the sub was controlled by 4 motors. The Graupner Company makes the motors as a bow thruster for large model boats. Graupner motor can be seen in Figure 3. The advantages of this motor are that the propeller is not exposed to the environment and there isn't a mechanical connection such as linkages and gears reducing the complexity of the design.

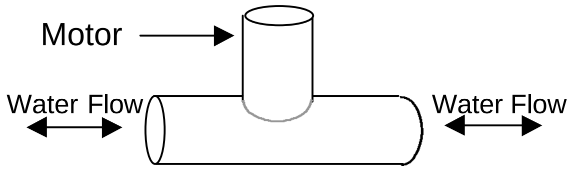

Figure 4

Two of the motors are placed toward the back of the main body with the thrust in a horizontal direction. The motors are placed near the bulk of the weight to keep from causing an upward are downward drift when engaged. The other two motors will be placed in the middle of the body in a vertical direction. These motors control the vertical motion of the sub. See Figure 2 for placement of motors.

SENSORS

Pressure Sensor

Submergency used a 0-100psi, 0-5volt pressure sensor made by Measurement Specialties Inc. For every 33 feet of water the pressure sensor would increase by 14.7psi. Since the sub was not going to be used in water no more than 10 feet in depth the output signal had to be amplified. The sensor will only change by 4.5psi in this range, which is only an change of .225 volts from the top of the water to the bottom. Also since the pressure sensor is at 1 atmosphere on land it had an output voltage of .9volts that had to be remove before the .225volt signal could be amplified. This was accomplished using a summing amplifier. The circuit is seen in figure 5. Again of about 8V/V was used. This gave the pressure sensor accuracy to 1 inch.

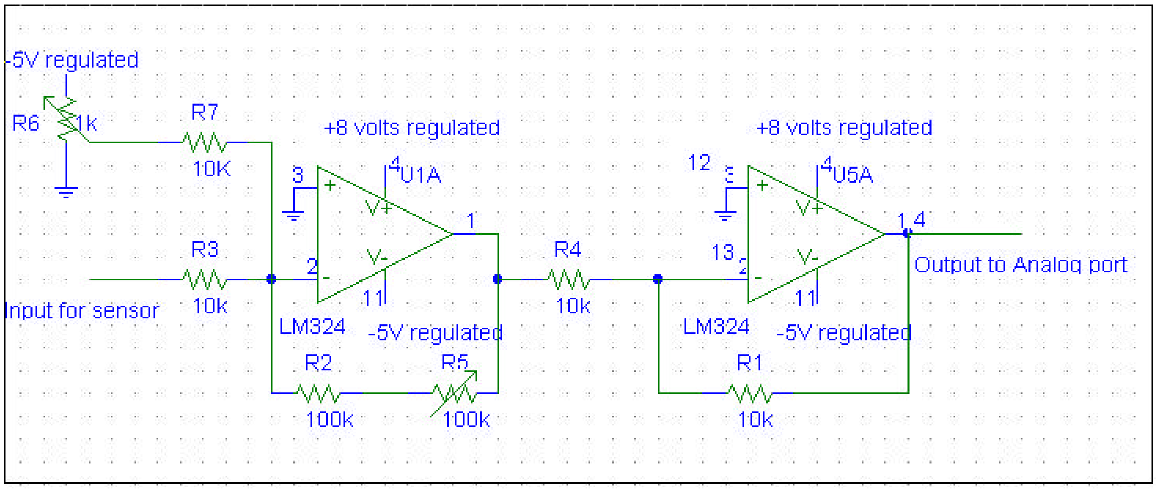

Figure 5 Pressure Sensor Amplifier

Compass

The Precision Navigation Vector 2XTG module controls Submergency's lateral motion. This module was easily interfaced with the 68HC11. The compass module was wired into the SPI system as the Master and the 68HC11 as the slave. The slave line on the microprocessor is held low at all times. A transfer of data begins when the microprocessor sends a low pulse to the Pull line of the module. The processor begins accepting data on the first rising edge of the clock pulse from the compass. The processor reads in two bytes of data. The compass module can send the data in two formats BCD or binary.

```
Binary      0000000 1   |  0   1    1   0   0   0   1   1
                    256   128  64   32  16  8    4    2   1

                  359= 256 +64+32+4+2+1
```

The binary format seemed easier to implement. The compass module has +- 2 degree accuracy. The compass module also has the ability to compensate for external magnetic field distortion, but this function was not needed for just getting a relative heading.

<u>BEHAVIORS</u>

Submergency had two basic behaviors. The first behavior is to take a given depth and maintain the depth while in motion. The depth routine takes the amount that the system is off of its depth and uses that value to determine the speed of the motors. This keeps the system from overcompensating. The second behavior was to maintain a heading that is initialized on power up and maintain that heading until a new heading if given to the system. The routine to maintain the heading also uses the amount the system is off its course to control the motors.

CONCLUSION

Submergency is a partially autonomous robot. The system has the capability to maintain a course and a depth by reading in the values from the electronic compass and the pressure sensor. The sub lacks the ability to avoid objects, which can be accomplished with the use of sonar. Until a working sonar can be made to work with the system, Submergency can only follow a predetermined course. The sonar is an area that I will continue to explore.

VENDORS

Company	**Part**
Hobby lobby	Motors, 7.2volt batteries
Mekatronic	68HC11 Pro-board
Digikey	Pressure Sensor, 12volt battery
Faulkners Plastic	Acrylic tube
Precision Navigation	Vector 2XTG compass module

APPENDIX A

```
/* test code for the pressure sensor.  Pulls the analog port and
puts the value an array that prints the value out too the screen*/

/*works just fine don't modify*/

/******************INCLUDES***********************/
#include <mil.h>
#include <subvect.c>
#include <hc11.h>
#include <submotor.c>
#include <analog.c>
#include <serial.c>

/*******************Constants*******************/
#define DIRECTION *(unsigned char*)(0x4000)


void delay ();
void up();

/********************Variables*****************/
int w=0;
int max_depth=0;
int q=0;
int first;
int data[200];
int next=0;
int motion = 0;
int count =0;

/**********************Main********************/
void main()
{
int p=0;

init_motors();
init_analog();
init_serial();

delay (1,200);
up(-100);

first = analog(3);
while(p < 200)
```

```
{

   data[q] = analog(3);
   next =data[q];

   if ((next - max_depth) < 3 && (next - max_depth) > -3 && count < 10)
   {
   count++;
   if (count == 9)
       {w++;}
   }

else if (next > max_depth)
 {
   max_depth = next;
   count =0;
 }

q++;
p++;

delay(13, 5000);

}
up(100);
delay(1500, 5000);
up(10);

while(1)
{
if (get_char() == 's')
 {
  q=0;
  while( q <200)
   {
    write_int(data[q]);
    q++;
   }
  write("max depth\n");
  write_int(max_depth);
  write_int(first);
  write_int(((max_depth-first)/2));
  write("count\n");
  write_int(count);
  write_int(w);
}

}


}




void delay (int loops, int lenght)
/* Function: delays the routine for a set amount of time
 * Inputs:   the # of loops that will be performed and the lenght of every loop
 * Outputs:  none
 *
 * Notes:    If long loops are needed then the function will needed to looped
 */
{
 int i;
 int a;
  for (i=0; i< loops; i++)
    {
      for (a=0; a<lenght; a++);
    }

}


void up(int u_rate)
/* Function:
 * Inputs:   none
 * Outputs:  none
 *
 * Notes:
 */
{
if (u_rate >= 0)
  {
   motion = motion | 0x0c;
  }

else
  {
   u_rate = abs(u_rate);
   motion = motion & 0xf3;
  }
```

```
DIRECTION = motion;
motor(2, u_rate);
motor(3, u_rate);
}
```

APPENDIX B

Submergency demo code

```
/******************INCLUDES***********************/
#include <mil.h>
#include <subvect.c>
#include <hc11.h>
#include <submotor.c>
#include <analog.c>
#include <serial.c>

/*****************Functions********************/

void delay();
void up();
void forward();
void reverse();
void init_spi();
int get_spi();
void maintain();
int init_depth();
void depth();


/*******************Constants******************/
#define DIRECTION *(unsigned char*)(0x4000)




/*******************Variables****************/


int motion=0x0;
int helm=0;
int deep;


void main()
{

int p= 0;

init_motors();
```

```
init_spi();
init_analog();
init_serial();

deep = 67;
/*init_depth();*/

depth(deep);

helm = get_spi();
write_int(helm);
forward(90, 90);

while (1)
{
while(p < 50)
{

maintain(helm);
depth(deep);
delay(1, 1000);
p++;

}
p=0;
helm = helm + 90;

if (helm > 360)
 { helm = helm - 359;}

}

}


void delay(int loops, int lenght)
/* Function: delays the routine for a set amount of time
 * Inputs:   the # of loops that will be performed and the lenght of every loop
 * Outputs:  none
 *
 * Notes:    If long loops are needed then the function will needed to looped
 */
{
 int i;
 int a;
  for (i=0; i< loops; i++)
```

```
    {
      for (a=0; a<lenght; a++);

    }

}



void up(int u_rate)
/* Function:
 * Inputs:   none
 * Outputs:  none
 *
 * Notes:
 */
{
if (u_rate >= 0)
  {
   motion = motion | 0x0c;
  }

else
  {
   u_rate = abs(u_rate);
   motion = motion & 0xf3;
  }


DIRECTION = motion;
motor(2, u_rate);
motor(3, u_rate);
}




void forward(int r_rate, int l_rate)
/* Function:
 * Inputs:   none
 * Outputs:  none
 *
 * Notes:
 */
{
```

```
if (r_rate >= 0)
  {
   motion = motion & 0xfe;
  }

if (l_rate >= 0)
  {
    motion = motion & 0xfd;
  }

if (r_rate <0)
  {
    motion = motion | 0x01;
    r_rate = abs(r_rate);
  }

if (l_rate <0)
  {
    motion = motion | 0x02;
    l_rate = abs(l_rate);
  }



DIRECTION= motion;
motor(0, r_rate);
motor(1, l_rate);
}



void reverse(int r_rate, int l_rate)
/* Function:
 * Inputs:   none
 * Outputs:  none
 *
 * Notes:
 */
{
motion = motion | 0x03;
DIRECTION= motion;
motor(0, r_rate);
motor(1, l_rate);
}
```

```
void init_spi ()
/* Function: initializes the spi system for slave mode
 * Inputs:  none
 * Outputs:  none
 *
 * Notes: HC11 is in slave mode
 */
{
DDRD= DDRD | 0x04;
SET_BIT(SPCR,0x4C);
motion= 0;
DIRECTION= motion;
delay(50, 5000);

motion= motion | 0x60;
DIRECTION= motion;
}



int get_spi()
/* Function: gets two byte from  the compass via the spi system
                  and returns a integer value in degrees
 * Inputs:  none
 * Outputs:  the heading in degrees
 *
 * Notes:   self contained
 */
{

int test=0;
int value1=0;
int value2=0;
int degree=0;

motion = motion |0x80;
DIRECTION = motion;
while (test == 0)
{
  test = SPSR & 0x80;
 }
value1= SPDR;
motion = motion & 0x7f;
DIRECTION = motion;

test=0;
```

```
while (test == 0)
{
  test = SPSR & 0x80;
}
value2= SPDR;

if (value1 > 1)
  {
  init_spi();
  degree = 0;
  }

else if (value1 > 0 && value2 < 103)
 {
   degree = value2+256;
 }
  else
  {
   degree= value2;
  }

return(degree);
}



void maintain(int heading)
/* Function: maintains the given heading
 * Inputs:  heading
 * Outputs: none
 *
 * Notes:
 */
{

int run=0;
int var_speedr=0;
int var_speedl=0;
int test1=0;

while(run == 0)
{

test1= get_spi();
test1= test1-heading;
```

```
if (abs(test1) > 180)
 {
  if (test1 > 0)
  {
   test1 = abs(test1) - 360;
  }

  else
  {
   test1 = 360 - abs(test1);
  }

}


if (test1 > 5)
{


   if (var_speedr + test1 <= 100)
          {
            var_speedr = var_speedr + test1;
            var_speedl = var_speedl - test1;
          }
   else if (abs(var_speedl) < 100)
          {
            var_speedl = var_speedl - (2 * test1);
            var_speedr = 100;
          }
   else
          {
            var_speedl = -100;
            var_speedr = 100;
          }


}


else if (test1 < -5)
{

  test1= abs(test1);

   if (var_speedl + test1 <= 100)
          {
            var_speedl = var_speedl + test1;
            var_speedr = var_speedr - test1;
          }

   else if (abs(var_speedr) < 100)
          {
            var_speedr = var_speedr - (2 * test1);
            var_speedl = 100;
          }
   else
          {
            var_speedr = -100;
            var_speedl = 100;
          }


}

else if (test1 <= 5 && test1>= -5)
{

  var_speedl = 90;
  var_speedr = 90;
  run = 1;
}
forward(var_speedr, var_speedl);
delay(1, 1000);
}

}


int init_depth()
/* Function: takes the sub to the bottom and gets the max depth
 *          reading. Then divides the max depth by two and returns
 *          that value so a mid depth can be maintained
 * Inputs:  heading
 * Outputs: none
 *
 * Notes:
 */
{
int first=0;
int max =0;
int next =0;
int count=0;
```

```
up(-100);

max = analog(3);
first= max;
while (count <= 10)
{
delay (13, 5000);

 next = analog(3);

if ((next - max) <= 2 && (next - max) >= -2)
 {
  count++;
 }

else if (next > max)
 {
   next = max;
   count =0;
 }

}

next = max - ((max - first) / 2);
up(0);
return(next);
}


void depth(int press)
/* Function:
 *
 *
 * Inputs:  heading
 * Outputs: none
 *
 * Notes:
 */
{

int current;
int d=0;
int speed=0;
```

```
while (d == 0)
{

current = analog(3);
current = current - press;
write_int(current);

if (current > 4)
  {
   speed = speed + (current * 2) ;
   if ( speed > 100)
     {
      speed = 100;
     }

}

else if (current < -4)
  {
  speed = speed -abs(current * 2);

  if ( speed < -100)
    {
     speed = -100;
    }

  }

else
  {
  speed = 0;
  d=1;
  }
write_int(speed);
up(speed);
delay(1, 2000);
}

}
```