University of Pennsylvania
Center for Sensor Technologies
Philadelphia, PA 19104

SUNFEST REU Program

BUILDING A PROTOTYPE OF A 3-D DISTRIBUTED MOBILE SENSOR NETWORK

NSF Summer Undergraduate Fellowship in Sensor Technologies
Greg Kuperman (Electrical Engineering) – University of Pennsylvania
Advisor: Dr. Daniel D. Lee

ABSTRACT

Biological organisms have the ability to gather information over a variety of senses from multiple viewpoints for accurate sensory perception. Artificial systems typically use a static array of sensors that lack mobility, or employ motion in a robotic platform in two dimensions.

The goal of this project is to design and build a prototype of an adaptive, distributed sensor network utilizing small, modular sensors and actuating components that will accurately position sensors in three-dimensional space. A working prototype was built, using a simple motorized spool design and a Motorola HC11 microcontroller. The system monitors the current position in three dimensions using encoders. Based on the change of lengths of the network cables, the motors are set to spin at speeds that will achieve motion in the desired direction. Control of the system was implemented in C to position an adaptive network in three dimensions. A user interface using serial communication and infrared remote control was designed.

Table of Contents

1. INTRODUCTION

In nature, biological organisms use multiple viewpoints to accurately perceive their environment. Designers of artificial sensory systems have used some of the same techniques that biological systems employ to gain sensory information from multiple viewpoints. For example, in auditory sensing, a large number of microphones are used in an acoustic sensing array. Employing beamforming techniques on a static microphone array can greatly amplify sound sources along certain directions while decreasing the array's sensitivity to noise in other directions, allowing a more accurate detection of the direction of the sound source [1]. Visually, multiple cameras are used to create binocular viewing systems to allow stereo depth perception.

However, a number of constraints limit the ability to gather information from multiple viewpoints. If an object is not placed exactly in the designated field that is being sensed, the information gathered can be spurious and inaccurate. Mobile sensor arrays have limitations of being constrained to two dimensions, and are usually wheeled. A wheeled sensor network, such as a robot, can easily disturb the environment it is trying to monitor. for example, a wheeled robot will disturb an animal, and the information gathered would not necessarily be accurate.

To replace static sensor arrays and wheeled robots, we propose to build an adaptive distributed sensor network that can be accurately positioned in three dimensions to gather information from multiple viewpoints. The system will consist of small, modular sensors and actuating components that will make the sensor network mobile. The sensor network can contain visual, audio, and olfactory sensors. The goal of the project is to build a prototype of an adaptive mobile network to assess the feasibility of such a system.

2. BACKGROUND

2.1 Biological Motivation

Being able to move provides a variety of viewpoints that can be used to perceive and make decisions about an area. The dependence of an observer's perception on his or her viewpoint can be illustrated with the Ames room illusion. Figure 1 shows the actual construction of the Ames room. If an observer looks through the viewing hole, limited to a single viewpoint, a pair of identical twins standing on opposite sides of the room,will be perceived as drastically different in size, as shown in Figure 2. Looking through the viewing hole removes any depth cues and makes the room appear normal and cubic, although its shape is actually trapezoidal, the floor is actually on an incline, and the walls are slanted outward. However, when the room is viewed from a different perspective, it becomes immediately clear to the observer that the room is not regularly shaped, and that the twins appeared different in size merely because they are at different distances from the observer [2]. This simple illustration demonstrates the role that an observer's location plays in sensory perception and highlights the importance of gathering information from multiple viewpoints for accurate perception.

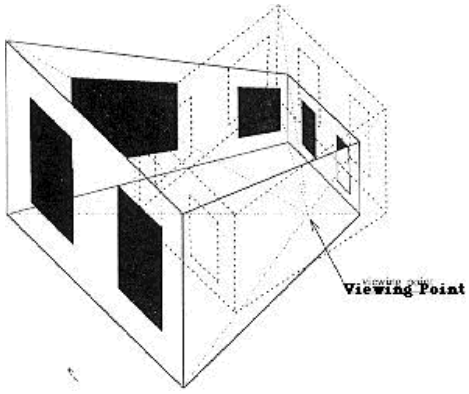


Figure 1: Actual shape and structure of an Ames room.

Figure 2: The Ames room illusion — seen through the viewing hole, the identical twins appear to be drastically different in size.

Biological organisms have developed binocular vision systems and binaural auditory systems that allow them to take advantage of multiple viewpoints in perception. In addition, most biological organisms employ movement in a process known as "active perception" to acquire new viewpoints and combine sensory information from multiple viewpoints to accurately perceive their surroundings [3]. For example, in olfactory sensing, humans actively move around an area while sniffing to detect the source of an odor. Since odors propagate slowly through wind transport and diffusion, an observer has little chance of detecting the source of an odor without obtaining observations of scents from multiple locations.

2.2 Existing Sensor Network Projects

Many different sensor network research projects are currently being conducted. The Palms Fixed/Mobile Experiment at the University of California–Berkeley is trying to deploy a sensor network via unmanned aerial vehicles. This sensor network will be able to spread quickly across an area and deliver information from numerous viewpoints [4].

The Smart Dust project, also at the University of California–Berkeley, is aimed at building sensor nodes that occupy less than 100 cubic millimeters and possess complete sensing and communication capabilities on a tiny mote. Researchers envision the sensor

nodes eventually being small and light enough to be capable of floating around in a room, communicating information to a base station for processing [5].

2.3 Description of Proposed Prototype

The inspiration for the three-dimensional positioning of the system came from theatrical performances such as *Peter Pan* that incorporate "flying" actors and props that are suspended in the air by multiple cables.

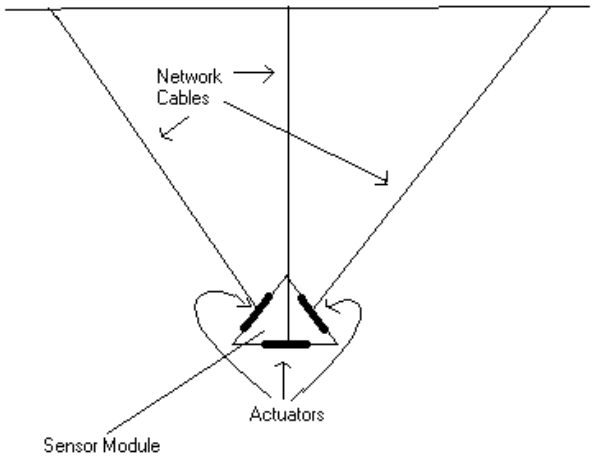


Figure 3: Sketch of Proposed Prototype.

The sensor module will have a number of thin cables connecting it to the ceiling that will provide mechanical support. Small actuators will be controlled by software to adjust the lengths of connected cables by winding or unwinding them. Changing the lengths of the supporting cables will enable the three-dimensional position of the sensor node to be quickly and freely adjusted. A diagram of the proposed system is shown in figure 3.

The sensor module will consist of some combination of a charge coupled device camera to capture video information, microphones to monitor audio signals, and gas sensors to measure local chemical concentrations. An embedded microcontroller will be used to digitize the sensor readings, perform preliminary processing on the data, and relay this information to external computers for further processing.

3. HARDWARE DESIGN

The goal for the summer was to develop an adaptive network that is able to move freely in three dimensions. By the end of the project, a system had been built consisting of three motors attached to a single platform that also acts as the sensor node. The three motors allow mobility in three dimensions constrained to a triangular area. Using the algorithm the system implements, it is straightforward to implement a fourth motor that will allow mobility across an entire room.

3.1 Hardware Description

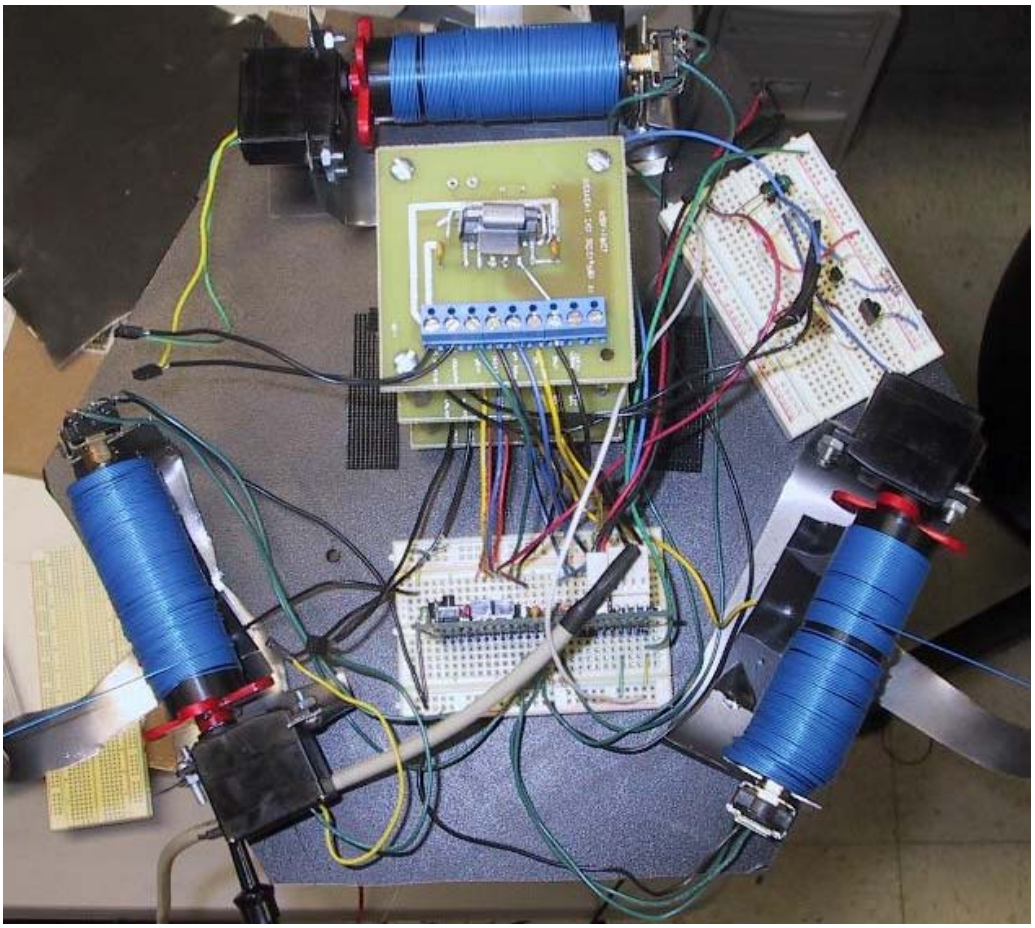


Figure 4: Fully assembled sensor node.

The spool system for controlling the lengths of the cables consists of an HS-225MG Hitec Mini Precision Servo that is modified to act as a geared DC motor, and custom-made plastic, aluminum and steel parts that were designed and machined. Connected to the ends of each spool is a CTS-288 two-bit rotary encoder that enables the location of the node to be accurately detected. To allow for the power source to be located off the sensor system, the network cables are 28-guage wire, which will trickle charge an onboard battery that powers the motors.

A Technological Arts Adapt11C24DX board fitted with a Motorola 68HC11E0 microcontroller is used for software control of the system. The microcontroller board is plugged into a solderless breadboard to enable easy connections to external devices. The microcontroller used has 24K of external RAM and 32K of EEPROM. To interface a remote control, an infrared detector circuit was designed and connected to the microcontroller board. A lead-acid battery is used to provide power to the motors and is recharged through the 28-gauge wire, which is connected to a laboratory DC voltage source. On board is a battery pack containing 4 AA batteries that is used to power the

microcontroller and infrared detector circuitry. The completed prototype is shown in figure 4.

3.2 HC11 Setup and Infrared Receiver Circuit

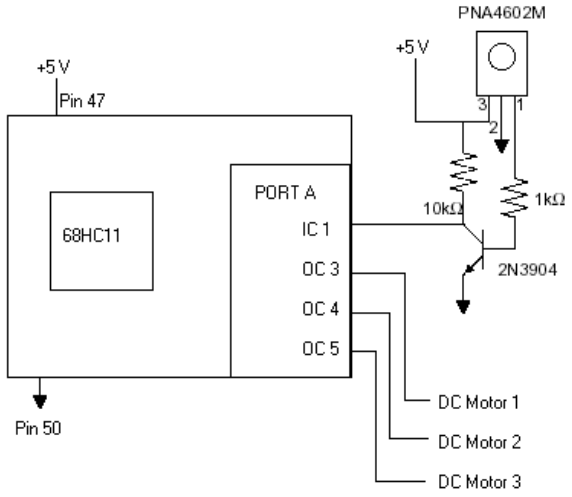


Figure 5: HC11 pin connections and infrared receiver circuit.

The Motorola 68HC11 is a multi-purpose, robust microcontroller chip used in many embedded systems for control and sensing applications, such as regulating temperature in a refrigerator and controlling the rate of combustion in a car engine. The Technological Arts Adapt11C24DX board combines a 68HC11 microcontroller, a voltage regulator, a port replacement unit, and a RS-232 serial communications interface in a single design that can be easily plugged into a solderless breadboard.

The output capture pins on the board were used to control the DC motors. The pins are connected to a National Semiconductor LMD18201 H-Bridge chip, which allows for an input signal of 5 volts and an output of 12 to 55 volts, as well as ease in changing direction and braking a DC motor. The H-bridges outputted 12 volts.

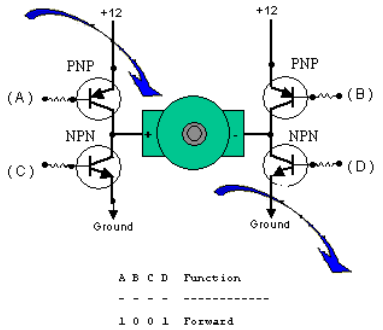
An infrared detector circuit, seen in figure 5 along with the pin connections for the 68HC11, consisting of a Panasonic PNA4602M IR detector chip and a 2N3904 transistor for signal amplification, was connected to the input capture pin of the HC11 Board to allow for detection and processing of infrared signals [6].

3.3 Modification of the HS-225MG Hitec Mini Precision Servo

A servo, an active device often used to control motion in remote control cars and planes, consists of a DC motor, a gear box, and on-board position feedback electronics that control the motor. It has a limited range of motion, usually being unable to spin more than 180 degrees.

Inside the servomotor are gears connected to the DC motor to slow down the output spin and also allow for increased torque. The HS-225MG has a rated torque of 66.7 oz-in at 6 volts. High torque is necessary in this application to hold the sensor nodes firmly in place. The servo is also small, 1.25 x 1.25 x .75 inches, and lightweight. This is necessary as well because to allow easier mobility the servos will need to carry their own weight plus the weight of the sensors on board.. To allow use of the compact size and high torque, the servo was modified to act like a DC motor, which will enable continuous rotation and speed control. Two holes were punched into the back of the servomotor. The onboard electronics were disconnected from the DC motor, and two wire leads were attached, thus enabling the DC motor to be controlled from outside the servo.

To control the DC motor in the servo, a pulse-width modulated (PWM) signal is required. To allow greater ease in controlling direction and voltage, an H-Bridge circuit is attached from the output compare pin of the HC11 chip to the DC motor. The H-bridge circuit allows for control of current through a DC motor. Turning on and off certain transistors controls the current to spin the motor in one direction or another. As is seen in figure 6 and 7, turning on and off certain transistors allows the motor to spin in opposite directions with only a single user controlled input. Putting the H-bridge in brake position turns off the current flowing through the motor and prevents the motor from spinning by maintaining the same voltage across the two nodes of a DC motor. As the motor tries to turn, it acts like a generator, producing its own voltage, but the H-bridge maintains the same voltage across the two nodes of the motor, turning the motor back to its original position and keeping it there [7].



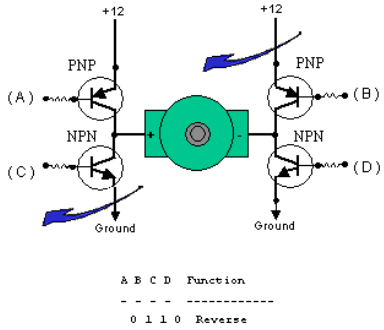


Figure 6: H-bridge flow diagram forward. Figure 7: H-bridge flow diagram backward.

The PWM signal is sent into the H-bridge at 5 volts (the output signal of the HC11) and changed to the equivalent signal at 12 volts (user-decided voltage of 12 to 55 volts). The PWM signal is high for a set amount of time, and then goes low. This repeats itself every 7.3 milliseconds. By setting the amount of time high, the user can control how fast the motor spins. The amount of time set high, which is called the percent duty cycle, is a certain percentage of the total time.A PWM signal is shown in figure 8. The motor spins faster with a high percent duty cycle, as it gets a high voltage across it for a longer time.

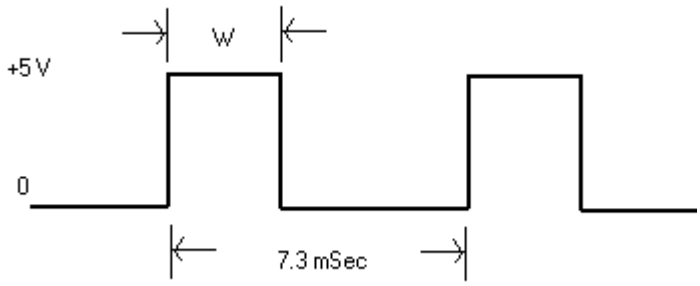


Figure 8: A Pulse-width modulated (PWM) signal.

The servo itself allows for rotation of only 180 degrees. To allow for greater rotation, the mechanical stopper must be removed using pliers. On the bottom of the output shaft, another stopper is also removed, as shown in figure 9. The rest of the circuitry is then cut from the DC motor, and two leads are soldered on, as shown in figure 10. Two holes are drilled into the back of the servo case to allow access to the leads [8].

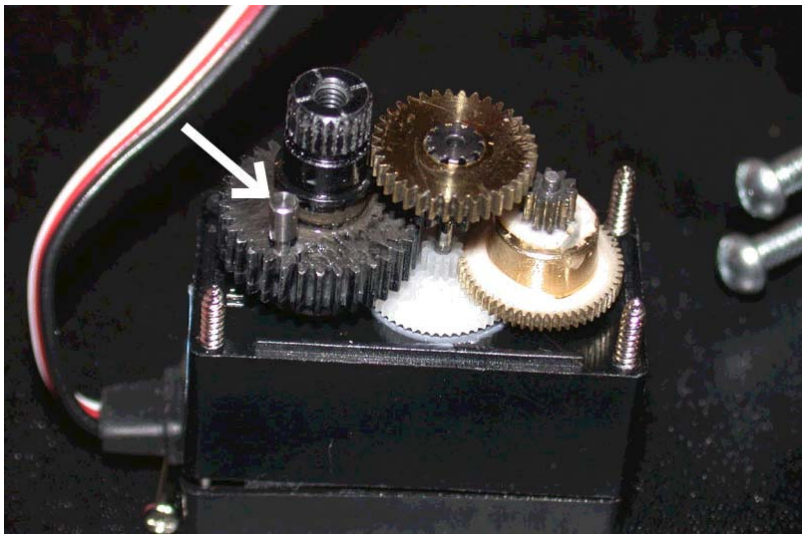


Figure 9: Modified servomotor gears.

Figure 10: DC motor leads.

3.4 Design of Mechanical Spool System

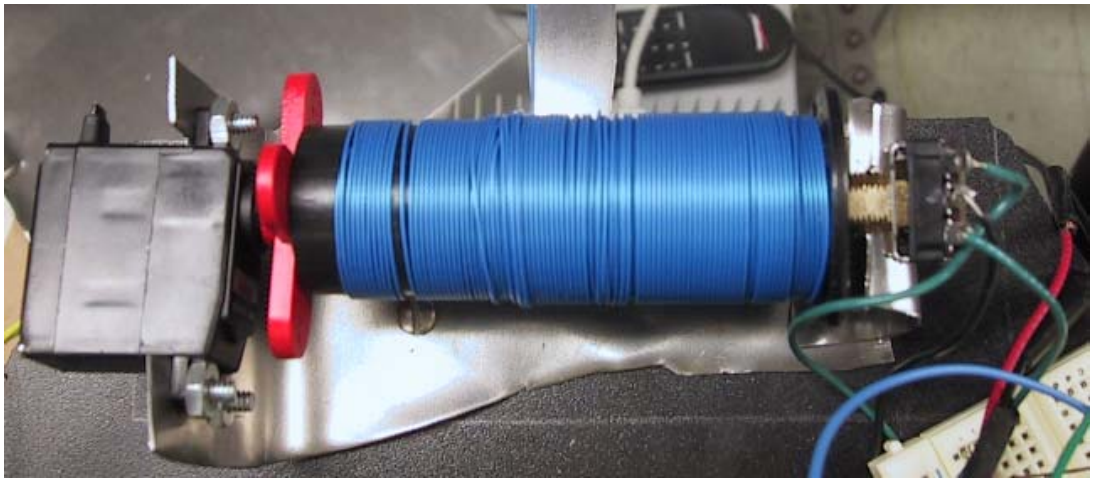


Figure 11: Mechanical spool.

The actual spool is a simple piece of cylindrical plastic, three inches long and one inch in diameter, that is screwed into the servo motor shaft. On the other end, a CTS-288 encoder is held firmly in place with epoxy, so it will spin with the motor. The encoder has two channels and a common pin. A 5-volt source is connected to the common pin. The two channels have a signal that goes high and low four times per revolution of the motor. The "A" channel leads the "B" channel so direction can be discerned. Since one channel leads the other, there are a total of 16 positions per revolution [9]. The timing diagram for the two channels are shown in figure 12. The outputs of the two channels are grounded and directed into the HC11. The encoder shaft is made of solid aluminum. A screw is drilled into the end going into the spool to enable a wire to be attached. The wire is then connected to the 28-gauge wire connected to the power source that charges the battery. The other end of the encoder is held by a piece of steel that also holds the motor for support. The steel, since it is connected to the other end of the encoder, has a connection to the voltage source and has a wire running to the battery, which trickle charges it. The entire spool with motor, encoder, wire, and steel holder are shown in figure 11.

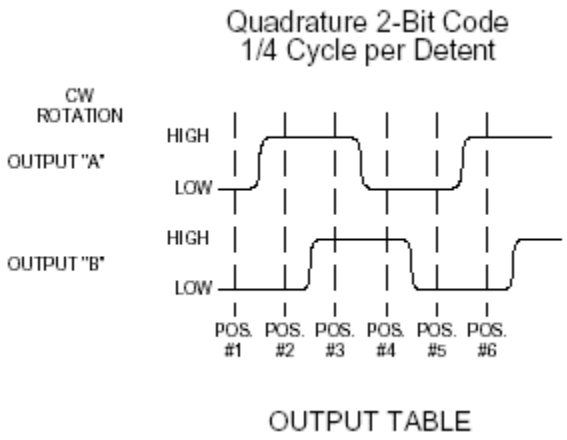


Figure 12: Encoder channel diagram.

3.5 Design of Sensor Node

The first prototype of the system, shown in figure 13, had three motors mounted on the ceiling and had a weight as the sensor node. This system allowed ease of testing because of the lightweight and simpler design.

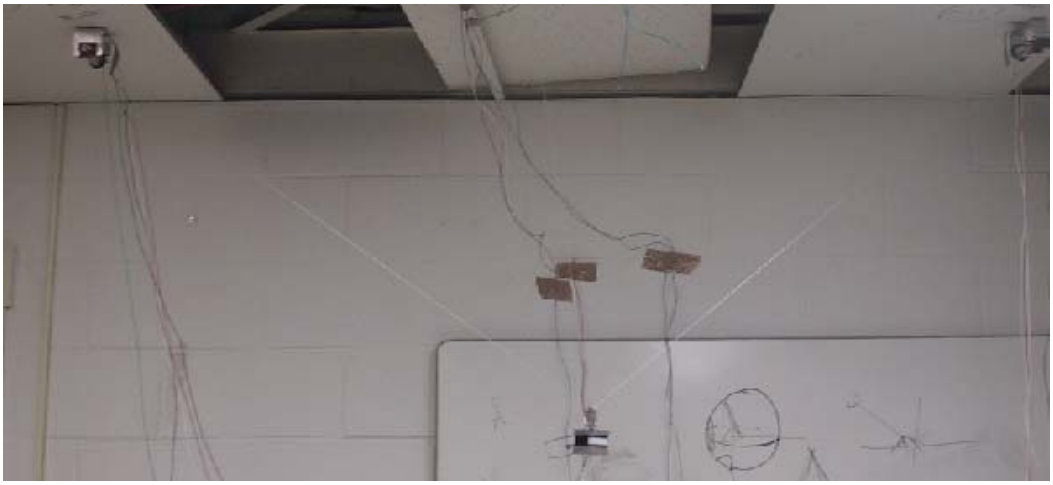


Figure 13: Original prototype.

The next system (see Figure 4) took the three motors onto a single platform that acts as the sensor node. This allows the system to adapt easily, requiring attachment to three hooks, instead of needing to mount motors on the ceiling. The node itself is a hexagon-shaped sheet of plastic board, with each side measuring six inches. The motor spools and encoder are placed on three edges to form an equilateral triangle. In the center is the 68HC11 and the three H-bridges, each H-bridge controlling a single motor. On another edge is the remote control circuitry. Each spool has 28-guage wire wound around it, which is used to deliver power to trickle charge the lead-acid battery on board. The wire is then pulled and connected to hooks in the ceiling. The individual wires are called

Side A, Side B, and Side C, respectively, throughout the remainder of this report. The battery pack for the 68HC11 and remote control circuitry is located next to the H-bridges, and the lead-acid battery is not placed on the board, because it is too heavy, but would ideally be placed on the other side of the H-bridge.

4. SOFTWARE DESIGN FOR SYSTEM CONTROL

4.1 Description of Software

Control software for the system was written in C and compiled using the Imagecraft ICC11 compiler that assembles code for the 68HC11. The Imagecraft ICC11 compiler was also used to load the program onto the 68HC11. The code consists of three major sections: code to decode user input, either via serial interface or remote control interface; code to find the position of the node; and code to control the DC motors.

4.2 User Input Decoding

There are two forms of user input: serial interface and remote control. The serial interface is used to communicate with the 68HC11 over a serial cable that allows the user to enter commands on the computer. The remote control communicates directly to the 68HC11 using infrared signals.

When the program first boots, the serial interface asks for the dimensions of the system. A base assumption is that the three points on the ceiling to which the wire connects forms an isosceles triangle, which is the most stable system to hold three sides. Each of the string lengths of the initial system must be entered, and the equal sides of the isosceles triangle following the non-equal side of the isosceles triangle are entered. Using this information, the system calculates the x, y, and z coordinates of the system. The x, y, and z coordinates are then outputted to the screen, and the user can then use either the remote control or the serial interface to move the system. With the serial interface, the user can then input the new x, y, and z coordinates he or she would like the system to go to. After the system has moved, the system calculates the new x, y and z coordinates. The coordinates are discussed further in section 3t.

For easy control of the system, remote control functionality was added. A Hauppauge remote control was used as an input device. Whenever a key is pressed, the remote control transmits two infrared pulse trains, each containing 12 pulses. The infrared receiver circuit connected to the input capture pin of the microcontroller triggers an interrupt service routine each time a pulse is received. The interrupt service routine records the value of the time counter (TCNT) on the microcontroller each time a high-to-low or low-to-high transition is detected. For each button pushed on the remote control, the 48 time counter values corresponding to the low-to-high and high-to-low transitions for the 24 transmitted pulses are recorded and stored in an array. The difference between every two time counter values is then calculated to get the width of each received pulse. Pulse widths larger than 3000 clock ticks are assigned a value of 1, while shorter pulse widths are assigned a value of 0. Pulse width assignments are shown in figure 14.

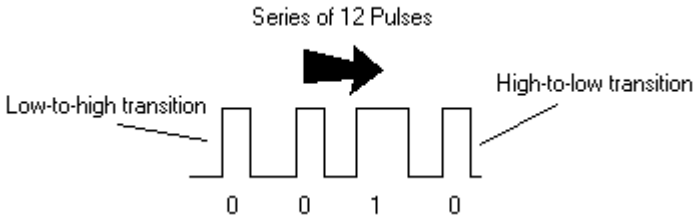


Figure 14: Decoding of pulse sequences.

Since each key of the remote control transmits a pulse sequence that has a unique binary bit pattern, the control software is capable of identifying the desired user input by matching the received bit pattern to the matching key on the remote control. Once the x, y, and z coordinates of the system are established, the user enters on the remote control the change desired in each direction. The channel plus and minus buttons tell the system to respectively add or subtract in a certain orthogonal direction. To change the position of the node, the user enters plus or minus, followed by two digits for change in x, followed by the same for y and z. An LED turns on for half a second to let the user know that the system has received a command. After z has been entered, the node is moved to the desired position, and then the new x, y, and z coordinates are calculated.

4.3 Finding the Position of the Node

As noted in section 2, when the program is first started, it asks for the dimensions of the system and uses them to calculate the x, y, and z coordinates. The x, y, z plane used has the base of Side A on the ceiling as the origin, with positive x going toward Side B, positive y going toward Side C, and positive z going toward the floor. Since the dimensions of the isosceles triangle in the x, y plane are known, the x, y, and z coordinates can be delineated. Using each length of the wire as a radius of a sphere centered on each point of the isosceles triangle, the system can calculate the x, y, and z coordinates of the node by finding the intersection of the three spheres. After the system runs and moves the node to a new position, the new x, y, and z coordinates are calculated. The system uses the number of encoder ticks seen at each of the motors to find the new lengths of the wires. From these lengths, the x, y, and z coordinates are calculated.

4.4 Code to Control the DC Motors

When the user inputs the new location of the node, either through the serial interface or remote control, the program determines the speeds at which each motor needs to move to reach the new point. The program does this by calculating the new length of each wire and subtracting the old length from the new length. If the number is negative, then the direction is set for the motor to wind the string inwards, and vise versa. The program also takes the difference between the two lengths and determines how many encoder ticks will be necessary for an individual motor to have spun the correct amount. The program monitors the number of encoder ticks, and when it sees the correct amount

and the node has reached the correct point, the brake is turned on for the motors. After the difference in the lengths of string is determined, the ratio of the other two side differences to the maximum difference is found. The ratios are used to set the motors to spin at a speed corresponding to the ratios to achieve smooth motion to the user-defined destination.

To achieve the correct speed for the motors, two arrays are created: one for unwinding and one for winding. Two arrays are necessary because there is more torque required to wind the wire than to unwind it. Each number position of the array is the number of time units between encoder ticks. Stored in that position is the correct PWM size to give the respective time unit. With the use of the ratios, the correct PWM size can be quickly assigned to each motor. Testing was done on the motors to find the correct PWM size for each time. Precisely timed PWM signals are easily generated with the output compare function of the 68HC11 microcontroller. The programmable timer on the 68HC11 is used to produce high and low voltages on the output compare pins to generate appropriate pulse sequences to control the DC motors.

5. DISCUSSION AND CONCLUSIONS

The completed system contains three motors that enable the system to be positioned anywhere in three dimensions within the boundaries formed by the isosceles triangle to which the wire is connected. The system is adaptable to any room as long as the connections to which the wires hook are in the shape of an isosceles triangle. When different sensors are connected to the node then multiple viewpoints will be accessible, allowing for a more accurate perception of the environment.

Certain difficulties were encountered. One issue was weight. The motors are not designed to handle the amount of weight that the entire node is. Therefore the node has variable performance and does not achieve desired results on every trial. A related issue is the power supply. The lead-acid battery is too heavy to be placed on board the node to power the system, and the system cannot be powered directly off the 28-gauge wire because the resistance of the long, thin wire does not allow enough current to pass.

6. RECOMMENDATIONS

As this project is still in its initial stages, there are many more interesting research problems to tackle in building a complete distributed 3-D mobile sensor network system. The system currently has trouble supporting its weight, therefore making reliable movement to a desired location difficult. When a large amount of tension is placed onto a motor, the motor begins to unwind. Using better motors can solve this problem. The motors used are servomotors with 66.7 oz-in torque. Using DC motors that can produce much more torque will allow the system to move more reliably.

The system currently uses three motors to move the node in three dimensions. This allows movement within a constrained triangular area. Since the movement of the motors is not reliable, a fourth motor that would have allowed a greater range of

movement was deemed unpractical. The use of more reliable motors, as discussed earlier, will allow a fourth motor to be interfaced reliably.

The system also does not have a great deal of feedback while in movement. The 68HC11 is not powerful enough to be able to watch for correct speeds and move the motors at the same time. A more powerful microcontroller will allow for the addition of feedback control and give more reliable movement.

Once a reliable working prototype of a single node is created, multiple nodes can be hooked together to create a network of sensors that can simultaneously provide information about a network. In a multiple node system, the position of each actuator would vary along with the position of each sensor node for any given configuration. To control such a system, it would be necessary to solve the forward and inverse kinematics problem related to the positioning of multiple nodes and actuators given different cable lengths.

If a reliable working prototype can be successfully completed, it will open up the possibility of many other interesting research fields related to distributed processing of sensor data, algorithms for subject tracking, and command recognition based on multimodal sensory input. As a student at the University of Pennsylvania, I hope to continue work on this project in the coming school year.

7. ACKNOWLEDGMENTS

I would like to thank the NSF for their support of the REU program and SUNFEST. Special thanks to Dr. Daniel Lee, who has been a great mentor, advisor, and source of encouragement throughout this project. In addition, I'd like to thank Sid Deliwala for his invaluable assistance in the Electrical Engineering lab, Lois Clearfield for taking care of the many administrative details, and James Tripp of the Math Department for his help in solving various geometrical problems. Last but not least, I would like to thank Dr. Jan Van der Spiegel for making SUNFEST possible and my fellow SUNFEST researchers for making the program an enjoyable one.

8. REFERENCES

1. B. D. Van Veen and K. M. Buckley, Beamforming: a versatile approach to spatial filtering. *ASSP Magazine, IEEE,* 5(2):4-24, 1988.

2. S. Del Prete, Ames room, Available at http://psylux.psych.tu-dresden.de/i1/kaw/diverses%20Material/www.illusionworks.com/html/ames_room.html, 6/10/03.

3. R. Bajcsy,. Active perception. *Proc. IEEE,* 76(8):996-1005, 1988.

4. 29 Palms Fixed/Mobile Experiment Tracking vehicles with UAV-delivered sensor network. University of California at Berkeley and MLB Co, http://www.eecs.berkeley.edu/~pister/29Palms0103, 6/10/03.

5. Smart Dust project website. University of California at Berkeley, http://robotics.eecs.berkeley.edu/~pister/SmartDust, 6/10/03.

6. D.D. Lee, IR asynchronous communicator, https://courseweb.library.upenn.edu/courses/1/EE400-2003A/content/_287694_1/lab05.pdf, 6/26/03.

7. J. Brown, DPRG: Brief H-Bridge Theory of Operation, http://www.dprg.org/tutorials/1998-04a/, 7/10/03.

8. Hacking Servos, http://www.geocities.com/kokop76/tips/tips.html, 7/3/03.

9. CTS 288 Series 16mm Rotary Encoder, http://rocky.digikey.com/WebLib/CTS/Web%20Data/288%20Series.pdf, 7/2/03.

APPENDIX A

```
/*
SUNFEST project
3-D Sensor Network
Greg Kuperman
August 1, 2003
*/

#include <hc11.h>

#pragma interrupt_handler OC3ISR
#pragma interrupt_handler OC4ISR
#pragma interrupt_handler OC5ISR
#pragma interrupt_handler SCIISR
#pragma interrupt_handler RTIISR
#pragma interrupt_handler IC1ISR

#define NUMBER 15000L /*Total Length of ticks on HC11 counter of duty
                                                    cycle length for motor output*/

void OC3ISR();
void OC4ISR();
void OC5ISR();
void SCIISR();
void IC1ISR();
void RTIISR();
void TOIISR();

/*uparray and downarray are what duty cycle will give that
 position of the array time between encoder ticks. anything
 below 5 are unaccesable positions. so to get 25 units of time
 between encoder clicks for bringing object up, choose duty cycle
 at uparray[25], values for this array are assigned from testing
 (write program to assign duty cycle to a motor and direction, and
 have RTI count and every encoder tick, output number of RTI's for
 that specific duty cycle*/
int uparray[105] = {04,04,04,04,04,04,04,04,04,04,04,04,04,04,04,04,04,04,04,
           95,95,95,90,85,80,78,77,75,70,68,67,65,63,62,60,59,57,56,
           55,55,54,54,53,53,52,52,51,50,48,47,45,45,44,44,43,43,42,42,
           41,40,40,40,40,40,39,39,39,39,39,39,38,38,38,38,38,38,38,37,37,
           37,37,37,37,37,37,36,36,36,36,36,36,36,36,36,35,35,35,35,35,35,
           35,35,35,34,34,34,34,34};
int downarray[105]={04,04,04,04,04,04,04,04,04,04,04,04,04,04,95,90,85,80,
                    70,65,63,60,58,55,52,50,45,44,42,41,40,39,38,37,36,35,
                    34,33,32,31,31,30,29,28,28,27,27,26,26,25,25,24,24,23,23,
                    22,22,22,21,21,21,20,20,20,19,19,19,18,18,18,18,
                    17,17,17,17,16,16,16,16,16,15,15,15,15,15,15,15,15,14,
                    14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14};

int distance;

int newx,newy,newz;
int x,y,z;
long deltaa,deltab,deltac;
```

```
int sidea,sideb,sidec,sided,sidee;
int x3,y3,range;
int a,b,c,d,da,db,dc,dx;

int acount,bcount,ccount,pulsetrack1,pulsetrack2,pulsetrack3;

int remotex,remotey,remotez,remotecount,xdir,ydir,zdir;
int sciflag;

int flagc,flagd,flage,ecount1,ecount2,ecount3;
int testarray1[10], testarray2[10],testarray3[10];
int pulsecount1,pulsecount2,pulsecount3;

int flagcA,flagcB,flagdA,flagdB,flageA,flageB,flagc1,flagd1,flage1;
int currentstateC,currentstateD,currentstateE,laststateC,laststateD,laststateE;

int i,j,k;
int ledflag,ledcount;

int ii;
unsigned int period[48];
int diff[47];
int count;

int changeflag,countflag;

int direction1=1, direction2=1,direction3=1;

unsigned int t3;

unsigned int t_high3; /*pulse width for OC3*/
unsigned int t_low3;

unsigned int t4;

unsigned int t_high4; /*pulse width for OC5*/
unsigned int t_low4;

unsigned int t5;

unsigned int t_high5; /*pulse width for OC5*/
unsigned int t_low5;

void main(){

*(unsigned char *)0xD9 = 0x7E;
*(void (**)())0xDA = OC3ISR;
*(unsigned char *)0xD3 = 0x7E;
*(void (**)())0xD4 = OC5ISR;
*(unsigned char *)0xD6 = 0x7E;
*(void (**)())0xD7 = OC4ISR;
*(unsigned char *)0xC4 = 0x7E;
*(void (**)())0xC5 = SCIISR;
*(unsigned char *)0xEB = 0x7E;
*(void (**)())0xEC = RTIISR;
*(unsigned char *)0xE8 = 0x7E;
```

```
*(void (**)())0xE9 = IC1ISR;


/*Set initial duty cycle*/
t3 = 5;
t4 = 5;
t5 = 5;

t_high3 = NUMBER*t3 / 100;
t_low3 = NUMBER - t_high3;

t_high4 = NUMBER*t4 / 100;
t_low4 = NUMBER - t_high4;

t_high5 = NUMBER*t5 / 100;
t_low5 = NUMBER - t_high5;

/*SCI settings*/

BAUD = 0x30; /*9600 BAUD*/
SCCR1 = 0x00;
SCCR2 = 0x0C; /* Enable SCI transmitter and reciever */
SCCR2 |= 0x20; //Reciever interrupt enable

/* OC5 settings */
PACTL &= 0xFB; //Set OC5 in PACTL
TOC5 = 0x7FFF;
TCTL1 |= 0x03; //set output pin high for OC5
TMSK1 |= 0x08; //Enable OC5 interrupt locally
TFLG1 &= 0x08; //Clear OC5 Flag

/* IR input settings */

TCTL2 |= 0x30; //capture high-to-low and low-to-high transition
TMSK1 |= 0x04;

/* OC3 settings */
TOC3 = 0x7FFF;
TCTL1 |= 0x30; //set output pin high for OC3
TMSK1 |= 0x20; //Enable OC3 interrupt locally
TFLG1 &= 0x20; //Clear OC3 Flag

/* OC4 settings */
TOC4 = 0x7FFF;
TCTL1 |= 0x0C; //set output pin high for OC4
TMSK1 |= 0x10; //Enable OC3 interrupt locally
TFLG1 &= 0x10; //Clear OC3 Flag

/*Real time interrupt settings*/
//Timerover flow settings
TMSK2 |= 0x40;
TFLG2 |= 0x40;
PACTL |= 0x01;

DDRC=0; //all PORTC is input
```

```
PORTB |= 0x2A; //turn brake on

/*clear testarrays*/
for(i=0,j=0;i<10;i++,j++){
        testarray1[i]=0;
        testarray2[j]=0;
        testarray3[j]=0;
        }
i=0;
j=0;
k=0;

asm("cli");

while(1){

        }
}

/*absolute value function*/
int absolute(int value){
        if(value<0) value*=-1;
        return value;
        }


/*Square root without use of floating point*/
unsigned int sqrtnewton (unsigned int num) {
        unsigned temp, div = num;
        unsigned rslt = num;
        if (num <= 0)
                return 0;
        while(1){
                        temp = num / div + div;
                        div = temp >> 1;
                        div += temp & 1;
                if (rslt > div)
                  rslt = div;
                else return rslt;
        }
}

/*Calculate x y z coordinates of object using intersection of three spheres*/
void xyz(){

        x = (sideb*sideb - sidee*sidee - sidea*sidea) / (-2*sidee);
        y = (sidec*sidec + 2*x3*x - x3*x3 - y3*y3 - sidea*sidea) / (-2*y3);
        z = sqrtnewton(sidea*sidea - x*x - y*y);

        range=(y3-y)*x3 / y3;/*width of accessable x for triangular area*/

        printf("\nx=%d y=%d z=%d",x,y,z);
        }

/*maximum value function*/
long max(long first, long second){
```

```
        if (first > second) return first;
        else return second;
        }


/*change lengths of strings to new lengths based off of encoder readings*/
void change(){

         int TICKS=1460; /*ticks * 100 per revolution on encoder,
                           adjusted by -8% for accuracy*/
         int DIAMETER=1; /*Diameter of spool in inches*/

         deltaa = (100 * (long)pulsecount1*22)/TICKS/7; //change in sidea
         pulsecount1=0;

         deltab = (100 * (long)pulsecount2*22)/TICKS/7; //change in sideb
         pulsecount2=0;

         deltac = (100 * (long)pulsecount3*22)/TICKS/7;       //change in sidec
         pulsecount3=0;

         /*change side lengths*/
         sidea += (int)deltaa;
         sideb += (int)deltab;
         sidec += (int)deltac;

        putchar('\n');
        if (deltaa<0){
                printf(" SideA IN ");
                  }
        else if (deltaa>0){
                 printf(" SideA OUT ");
                  }

        if (deltab<0){
                printf(" SideB IN ");
                  }
        else if (deltab>0){
                 printf(" SideB OUT ");
                  }

        if (deltac<0){
                printf(" SideC IN ");
                  }
        else if (deltac>0){
                 printf(" SideC OUT ");
                  }

        printf("\nCHANGE: deltaa=%d deltab=%d deltac=%d NEW: sidea=%d sideb=%d sidec=%d",
                          (int)deltaa,(int)deltab,(int)deltac,sidea,sideb,sidec);

         }


/* calculate and output duty cycle to motors to achieve desired motion*/
void dist(){
```

```
        int ratio1,ratio2,ratio3;
        int temp,temp1,temp2;

        printf("\nnewx=%d newy=%d newz=%d",newx,newy,newz); //desired x y z

        /*find change in length for each the sides*/
        /*start*/
        temp = newx*newx + newy*newy + newz*newz;
        temp1 = (sided-newx)*(sided-newx) + newy*newy + newz*newz;
        temp2 = (newx - sided/2)*(newx - sided/2) + (newy - (int)*
                (866L*(long)sided / 1000))*(newy - (int)((long)866L*sided / 1000))
                + newz*newz;

        a = sqrtnewton(temp);
        b = sqrtnewton(temp1);
        c = sqrtnewton(temp2);

        da = (sidea-a);
        db = (sideb-b);
        dc = (sidec-c);
        /*finish*/

        /*find number of encoder ticks for each motor for change in length*/
        pulsetrack1 = (long)da * 1460 * 7 / 22 / 100;
        if (pulsetrack1<0) pulsetrack1 *=-1;
        pulsetrack2 = (long)db * 1460 * 7 / 22 / 100;
        if (pulsetrack2<0) pulsetrack2 *=-1;
        pulsetrack3 = (long)dc * 1460 * 7 / 22 / 100;
        if (pulsetrack3<0) pulsetrack3 *=-1;

        /*set direction of string movement for each motor*/
        if (da<0) {direction1 = 2; PORTB &= 0xFB; putchar('A');   }
        else if(da>0) {direction1 = 1;PORTB |= 0x04; putchar('a'); }
        if (db<0) {direction2 = 2; PORTB &= 0xFE;putchar('B');  }
        else if (db>0) {direction2 = 1;PORTB |= 0x01; putchar('b');}
        if (dc<0) {direction3 = 2; PORTB &= 0xEF;putchar('C');  }
        else if (dc>0) {direction3 = 1;PORTB |= 0x10; putchar('c');}

        putchar('\n');

        printf("\nda=%d db=%d dc=%d direction1=%d direction2=%d direction3=%d",
                        da,db,dc,direction1,direction2,direction3);

        if (da<0) da *= -1; //absolute value
        if (db<0) db *= -1; //absolute value
        if (dc<0) dc *= -1; //absolute value

        ratio1=1001;
ratio2=1001;
ratio3=1001;

        /*setting the duty cycle*/
        /*algorithm: 1. find the maximum length change
         2. find the ratio of the other two side changes to the maximum change
        multiplied by 1000 to keep it an integer with 3 decimal precision
```

```
 3. use 20th position in each array as maximum speed (20 unit time between
encoder ticks)
4. assign other motor speeds by dividing 20 by ratio (scaled by 1000)
5. if motor speed too slow (ratio below 200) assign manually based on
specifications of the system
*/

if (max(max(da,db),dc) == da) {
        ratio2=(1000*db)/da;
        ratio3=(1000*dc)/da;
        printf("\ndA max, ratio2=%d ratio3=%d",ratio2,ratio3);


                if (direction1==1) t4 = uparray[20];
                else if (direction1==2) t4 = downarray[20];
                if (direction2==1) t5 = uparray[20000/ratio2];
    else if (direction2==2) t5 = downarray[20000/ratio2];
    if (direction3==1) t3 = uparray[20000/ratio3];
    else if (direction3==2) t3 = downarray[20000/ratio3];

                 if ((ratio2<200 && ratio2>100) && direction2==1) t5 = 33;
                 else if ((ratio2<200 && ratio2>100) && direction2==2) t5 = 15;
                 else if(ratio2<100 && direction2==1) t5=31;
                 else if(ratio2<100 && direction2==2) t5=13;
                 if ((ratio3<200 && ratio3>100) && direction3==1) t3 = 33;
                 else if ((ratio3<200 && ratio3>100) && direction3==2) t3 = 15;
                 else if(ratio3<100 && direction3==1) t3=31;
                 else if(ratio3<100 && direction3==2) t3=13;


}

else if (max(max(da,db),dc) == db){
        ratio1=(1000*da)/db;
        ratio3=(1000*dc)/db;
        printf("\ndB max, ratio1=%d ratio3=%d",ratio1,ratio3);


                if (direction1==1) t4 = uparray[20000/ratio1];
                else if (direction1==2) t4 = downarray[20000/ratio1];
                if (direction2==1) t5 = uparray[20];
            else if (direction2==2) t5 = downarray[20];
            if (direction3==1) t3 = uparray[20000/ratio3];
            else if (direction3==2) t3 = downarray[20000/ratio3];

                 if ((ratio1<200 && ratio1>100) && direction1==1) t4 = 33;
                 else if ((ratio1<200 && ratio1>100) && direction1==2) t4 = 15;
                 else if(ratio1<100 && direction1==1) t4=31;
                 else if(ratio1<100 && direction1==2) t4=13;
                 if ((ratio3<200 && ratio3>100) && direction3==1) t3 = 33;
                 else if ((ratio3<200 && ratio3>100) && direction3==2) t3 = 15;
                 else if(ratio3<100 && direction3==1) t3=31;
                 else if(ratio3<100 && direction3==2) t3=13;

}
```

```
        else if (max(max(da,db),dc) == dc){
                ratio1=(1000*da)/dc;
                ratio2=(1000*db)/dc;
                printf("\ndC max, ratio1=%d ratio2=%d",ratio1,ratio2);

                        if (direction1==1) t4 = uparray[20000/ratio1];
                        else if (direction1==2) t4 = downarray[20000/ratio1];
                        if (direction2==1) t5 = uparray[20000/ratio2];
                   else if (direction2==2) t5 = downarray[20000/ratio2];
                   if (direction3==1) t3 = uparray[20];
                   else if (direction3==2) t3 = downarray[20];

                         if ((ratio1<200 && ratio1>100) && direction1==1) t4 = 33;
                         else if ((ratio1<200 && ratio1>100) && direction1==2) t4 = 15;
                         else if(ratio1<100 && direction1==1) t4=31;
                         else if(ratio1<100 && direction1==2) t4=13;
                         if ((ratio2<200 && ratio2>100) && direction2==1) t5 = 33;
                         else if ((ratio2<200 && ratio2>100) && direction2==2) t5 = 15;
                         else if(ratio2<100 && direction2==1) t5=31;
                         else if(ratio2<100 && direction2==2) t5=13;

        }

        /*adjusted for this system specifically to correct for erroneous movement*/
        if (direction1==2){t4+=10;putchar('Q');
        else if (t4<30) {t4+=10;putchar('W');
        if (direction2==2) {t5+=10;putchar('E');
        else if (t5<30) {t5+=10;
        if (direction3==2) {t3+=10;
        else if (t3<30) {t3+=10;

        printf("\nt3=%d t4=%d t5=%d pulsetrack1=%d pulsetrack2=%d pulsetrack3=%d",
                t3,t4,t5,pulsetrack1,pulsetrack2,pulsetrack3);
}


/*called when user asks system to change position*/
void changetime(){

                PORTB |= 0x2A; //turn brake on

                putchar('\n');

                printf("\npulsecount1=%d pulsecount2=%d pulsecount3=%d",
                        pulsecount1,pulsecount2,pulsecount3);

                change();       //find new lengths of sides
                xyz();          //calculate new xyz
                printf("\nNew x=%d y=%d z=%d",x,y,z);

                putchar('\n');

                /*remote input*/
                if(sciflag==0){
                        newx=remotex+x;
                        printf("\nx has %d added, newx=%d",remotex,newx);
```

```
                    newy=remotey+y;
                    printf("\ny has %d added, newy=%d",remotey,newy);
                    newz=remotez+z;
                    printf("\nz has %d added, newz=%d",remotez,newz);
                    }

            /*serial input*/
            else if(sciflag==1){
                    printf("\nPlease enter new X: ");
               newx=scanchar();
               printf("\nPlease enter new y: ");
               newy=scanchar();
               printf("\nPlease enter new z: ");
               newz=scanchar();
            }

            dx = newx-x; //change in x direction
            if (dx<0) dx*=-1;

            if (y>y3) printf("\nRange Error: y too large");
            else if (x>(x3+range) || x<(range-x3))
                    printf("\nRange Error: x out of trianlge");

            dist(); /*assign duty cycles for each motor to get
                     system to move correctly*/

            acount=0;
            bcount=0;
            ccount=0;

            putchar('\n');

            printf("\ndx=%d",dx);

            putchar('\n');

            t_high3 = NUMBER*t3 / 100;
t_low3 = NUMBER - t_high3;

            t_high4 = NUMBER*t4 / 100;
t_low4 = NUMBER - t_high4;

            t_high5 = NUMBER*t5 / 100;
t_low5 = NUMBER - t_high5;

            PORTB &= 0xD5; //turn brake off

            /*if duty cycle below 5, leave brake on for that motor*/
            if(t3<5)PORTB |= 0x20;
            if(t4<5)PORTB |= 0x08;
            if(t5<5)PORTB |= 0x02;

            remotex=0;
        remotey=0;
            remotez=0;
            remotecount=0;
```

```
}

int scanchar(){
	int val=0;
	char c;
	c=getchar();
	while (isdigit(c)){
		val = 10*val + (c-'0');
		putchar(c);
		c=getchar();
		}
	return val;
}

/*if system being accessed for first time, assign side lengths
 and dimensions of system*/
void first(){
			changeflag=1;
			printf("\nPlease enter length of SideA: ");
			sidea=scanchar();
			printf("\nPlease enter length of SideB: ");
			sideb=scanchar();
			printf("\nPlease enter length of SideC: ");
			sidec=scanchar();
			printf("\nPlease enter Equal Length of
				Isosceles Traingle: ");
			sided=scanchar();
			printf("\nPlease enter Non-Equal Length of
				Isosceles Traingle: ");
			sidee=scanchar();

			x3 = sidee/2;
			y3 = sqrtnewton(sided*sided - x3*x3);
			printf("\nx3=%d y3=%d ",x3,y3);

			xyz();
			}

void SCIISR(){
	unsigned char scsrval,scdrval;
	scsrval=SCSR;
	scdrval=SCDR;
			sciflag=1;
			if(changeflag==0)first();
	else changetime();
}

void OC3ISR(){
	  countflag++;
	TFLG1 &= 0x20;
	if (TCTL1 & 0x10){
		TOC3 += t_high3;
		TCTL1 &= 0xEF;
		}
	else {
		TOC3 += t_low3;
```

```
            TCTL1 |= 0x10;
            }
}

void OC4ISR(){

      TFLG1 &= 0x10;
      if (TCTL1 & 0x04){
            TOC4 += t_high4;
            TCTL1 &= 0xFB;
            }
      else {
            TOC4 += t_low4;
            TCTL1 |= 0x04;
            }
                  }

void OC5ISR(){

      TFLG1 &= 0x08;
      if (TCTL1 & 0x01){
            TOC5 += t_high5;
            TCTL1 &= 0xFE;
            }
      else {
            TOC5 += t_low5;
            TCTL1 |= 0x01;
            }
                  }

/*poll for encoder ticks*/
void RTIISR(){
          TFLG2 |= 0x40;

          /*to count time between encoder ticks, use the ecount's below,
           reset them everytime
            a tick occurs, but print out a value before you do so*/
          ecount1++;
          ecount2++;
          ecount3++;

          ledcount++;

          /*turn on LED for remote for 1/2 second*/
          if(ledflag==1){
                  PORTB |= 0x40;
                  if(ledcount > 60){
                          PORTB &= 0xBF;
                          ledflag=0;
                  }
          }


          /*Dual channel encoders, edge A leads edge B*/

          if ((PORTC & 0x08) && flagc<3) flagc++;
```

```
if ((PORTC & 0x08) && flagc==3){
  flagcA=1;
  flagc++;
  }
if (!(PORTC & 0x08) && flagc==4) {flagc=0;flagcA=0;}

if ((PORTC & 0x01) && flagc1<3) flagc1++;
if ((PORTC & 0x01) && flagc1==3){
  flagcB=1;
  flagc1++;
  }
if (!(PORTC & 0x01) && flagc1==4) {flagc1=0; flagcB=0;}


if(flagcA==0 && flagcB==0)      currentstateC=0;
else if(flagcA==1 && flagcB==0) currentstateC=1;
else if(flagcA==1 && flagcB==1) currentstateC=2;
else if(flagcA==0 && flagcB==1) currentstateC=3;

if(laststateC != currentstateC){
            if (((laststateC < currentstateC)&&!(laststateC==0 && currentstateC==3)) ||
             (laststateC==3 && currentstateC==0)){ pulsecount1++;acount++;putchar('C');}
            else if((laststateC > currentstateC) || (laststateC==0 && currentstateC==3)){
                        pulsecount1--;acount++;putchar('c');}
            laststateC=currentstateC;
}

if ((PORTC & 0x02) && flagd<3) flagd++;
if ((PORTC & 0x02) && flagd==3){
  flagd++;
  flagdA=1;
  }
if (!(PORTC & 0x02) && flagd==4) {flagd=0;flagdA=0;}

if ((PORTC & 0x10) && flagd1<3) flagd1++;
if ((PORTC & 0x10) && flagd1==3){
  flagd1++;
  flagdB=1;
  }
if (!(PORTC & 0x10) && flagd1==4) {flagd1=0;flagdB=0;}

if(flagdA==0 && flagdB==0)      {currentstateD=0; }
else if(flagdA==1 && flagdB==0) {currentstateD=1; }
else if(flagdA==1 && flagdB==1) {currentstateD=2; }
else if(flagdA==0 && flagdB==1) {currentstateD=3; }

if(laststateD != currentstateD){
            if (((laststateD < currentstateD)&&!(laststateD==0 && currentstateD==3)) ||
             (laststateD==3 && currentstateD==0)){ pulsecount2++;bcount++;putchar('D');}
            else if((laststateD > currentstateD) || (laststateD==0 && currentstateD==3)){
             pulsecount2--;bcount++;putchar('d');}
            laststateD=currentstateD;
}

if ((PORTC & 0x20) && flage<3) flage++;
if ((PORTC & 0x20) && flage==3){
```

```
            flage++;
            flageA=1;
            }
          if (!(PORTC & 0x20) && flage==4) {flage=0;flageA=0;}

          if ((PORTC & 0x04) && flage1<3) flage1++;
          if ((PORTC & 0x04) && flage1==3){
            flage1++;
            flageB=1;
            }
          if (!(PORTC & 0x04) && flage1==4) {flage1=0;flageB=0;}

          if(flageA==0 && flageB==0)     { currentstateE=0; }
          else if(flageA==1 && flageB==0){ currentstateE=1; }
          else if(flageA==1 && flageB==1){ currentstateE=2; }
          else if(flageA==0 && flageB==1){ currentstateE=3; }

          if(laststateE != currentstateE){
                    if (((laststateE < currentstateE)&&!(laststateE==0 && currentstateE==3)) ||
                     (laststateE==3 && currentstateE==0)){pulsecount3--;ccount++;putchar('e');}
                    else if((laststateE > currentstateE) || (laststateE==0 && currentstateE==3)){
                     pulsecount3++;ccount++;putchar('E');}
                    laststateE=currentstateE;
          }

          /*wait until one of the motors goes the distance necessary and the other two are
           within 3 encoder ticks, the stop the system*/
          if ((pulsetrack1 < acount && pulsetrack2 < bcount + 3 && pulsetrack3 < ccount + 3) ||
           (pulsetrack1 < acount + 3 && pulsetrack2 < bcount && pulsetrack3 < ccount + 3) ||
           (pulsetrack1 < acount + 3 && pulsetrack2 < bcount + 3 && pulsetrack3 < ccount)){
            PORTB |= 0x2A; //turn brake on
            acount=0;
            bcount=0;
            ccount=0;
            }
}

/*Remote control*/
/*Enter ch+ or ch- to increase or decrease coordinate respetively.
currently, two digit change for each coordinate.
example: to increase x by 10, not change y, and decrease z by 3, enter the following:
CH+ 1 0 CH+ 0 0 CH- 0 3.
LED turns on for 1/2 second to confirm each time something is entered
*/

void IC1ISR(){

                    if(countflag>100){
           ii=0;
           countflag=0;
           }

      TFLG1 &= 0x04;
      period[ii] = TIC1;
      ii++;
      if (ii==48){
```

```
for(count=0;count<47;count++){
	diff[count]=period[count+1]-period[count];
	if(diff[count]>3000){diff[count]=1;}
	else{diff[count]=0;}
}
if(diff[13]==1 && diff[14]==1 && diff[19]==1 && diff[20]==1){printf("Source");}


else if (diff[13]==1 && diff[14]==1 && diff[21]==1){
	printf("CH -");
                                                    if(remotecount==0)xdir=0;
                                                    else if(remotecount==3)ydir=0;
                                                    else if(remotecount==6)zdir=0;
                                                    remotecount++;
                                                    ledflag=1;
                                                    ledcount=0;
}
else if (diff[13]==1 && diff[14]==1 && diff[15]==1){printf("Full Screen");}
else if (diff[13]==1 && diff[14]==1 && diff[17]==1){printf("Minimize");}


else if (diff[13]==1 && diff[14]==1){
	printf("CH +");
                                                    if(remotecount==0)xdir=1;
                                                    else if(remotecount==3)ydir=1;
                                                    else if(remotecount==6)zdir=1;
                                                    remotecount++;
                                                    ledflag=1;
	ledcount=0;
}


else if (diff[15]==1 && diff[16]==1 && diff[21]==1){
	printf("VOL -");

}


else if (diff[15]==1 && diff[16]==1){
	printf("VOL +");

}


else if (diff[17]==1 && diff[18]==1 && diff[21]==1){
	printf("9");
                                                    if(remotecount==1) remotex=9;
                                                    else if(remotecount==2) remotex=remotex*10+9;
                                                    else if(remotecount==4) remotey=9;
                                                    else if(remotecount==5) remotey=remotey*10+9;
                                                    else if(remotecount==7) remotez=9;
                                                    else if(remotecount==8) remotez=remotez*10+9;
                                                    remotecount++;
                                                    ledflag=1;
                                                    ledcount=0;
```

```
}


else if (diff[17]==1 && diff[18]==1){
      printf("8");
                                                if(remotecount==1) remotex=8;
                                                else if(remotecount==2) remotex=remotex*10+8;
                                                else if(remotecount==4) remotey=8;
                                                else if(remotecount==5) remotey=remotey*10+8;
                                                else if(remotecount==7) remotez=8;
                                                else if(remotecount==8) remotez=remotez*10+8;
                                                remotecount++;
                                                ledflag=1;
                                                ledcount=0;

}

else if (diff[14]==1 && diff[20]==1 && diff[21]==1){
      printf("Mute");
}


else if (diff[17]==1 && diff[20]==1){
          printf("Radio");

          }


else if (diff[17]==1){
          printf("TV");

}


else if (diff[20]==1 && diff[21]==1 && diff[19]==1){
      printf("5");
                                                if(remotecount==1) remotex=5;
                                                else if(remotecount==2) remotex=remotex*10+5;
                                                else if(remotecount==4) remotey=5;
                                                else if(remotecount==5) remotey=remotey*10+5;
                                                else if(remotecount==7) remotez=5;
                                                else if(remotecount==8) remotez=remotez*10+5;
                                                remotecount++;
                                                ledflag=1;
                                                ledcount=0;

}


else if (diff[20]==1 && diff[19]==1){
      printf("4");
                                                if(remotecount==1) remotex=4;
                                                else if(remotecount==2) remotex=remotex*10+4;
                                                else if(remotecount==4) remotey=4;
                                                else if(remotecount==5) remotey=remotey*10+4;
                                                else if(remotecount==7) remotez=4;
```

```
                                        else if(remotecount==8) remotez=remotez*10+4;
                                        remotecount++;
                                        ledflag=1;
                                        ledcount=0;

}


else if (diff[19]==1 && diff[22]==1){
        printf("6");
                                        if(remotecount==1) remotex=6;
                                        else if(remotecount==2) remotex=remotex*10+6;
                                        else if(remotecount==4) remotey=6;
                                        else if(remotecount==5) remotey=remotey*10+6;
                                        else if(remotecount==7) remotez=6;
                                        else if(remotecount==8) remotez=remotez*10+6;
                                        remotecount++;
                                        ledflag=1;
                                        ledcount=0;

}


else if(diff[19]==1){
        printf("7");
                                        if(remotecount==1) remotex=7;
                                        else if(remotecount==2) remotex=remotex*10+7;
                                        else if(remotecount==4) remotey=7;
                                        else if(remotecount==5) remotey=remotey*10+7;
                                        else if(remotecount==7) remotez=7;
                                        else if(remotecount==8) remotez=remotez*10+7;
                                        remotecount++;
        ledflag=1;
                                        ledcount=0;
}

else if (diff[15]==1 && diff[22]==1){printf("Reserve");}


else if (diff[22]==1 && diff[21]==1){
        printf("2");
                                        if(remotecount==1) remotex=2;
                                        else if(remotecount==2) remotex=remotex*10+2;
                                        else if(remotecount==4) remotey=2;
                                        else if(remotecount==5) remotey=remotey*10+2;
                                        else if(remotecount==7) remotez=2;
                                        else if(remotecount==8) remotez=remotez*10+2;
                                        remotecount++;
        ledflag=1;
                                        ledcount=0;
}


else if (diff[21]==1){
        printf("3");
                                        if(remotecount==1) remotex=3;
```

```
                                                else if(remotecount==2) remotex=remotex*10+3;
                                                else if(remotecount==4) remotey=3;
                                                else if(remotecount==5) remotey=remotey*10+3;
                                                else if(remotecount==7) remotez=3;
                                                else if(remotecount==8) remotez=remotez*10+3;
                                                remotecount++;
            ledflag=1;
                                                ledcount=0;
        }


        else if(diff[23]==1){
            printf("1");
                                                if(remotecount==1) remotex=1;
                                                else if(remotecount==2) remotex=remotex*10+1;
                                                else if(remotecount==4) remotey=1;
                                                else if(remotecount==5) remotey=remotey*10+1;
                                                else if(remotecount==7) remotez=1;
                                                else if(remotecount==8) remotez=remotez*10+1;
                                                remotecount++;
                                                ledflag=1;
                                                ledcount=0;
        }

        else {
                                        printf("0");
                                        if(remotecount==1) remotex=0;
                                        else if(remotecount==2) remotex=remotex*10+0;
                                        else if(remotecount==4) remotey=0;
                                        else if(remotecount==5) remotey=remotey*10+0;
                                        else if(remotecount==7) remotez=0;
                                        else if(remotecount==8) remotez=remotez*10+0;
                                        remotecount++;
                                        ledflag=1;
                                        ledcount=0;
                                        }

        printf("\n");

        ii=0;
        countflag = 0;

                                    if (remotecount==9){
                                      if(xdir==0)remotex*=-1;
                                      if(ydir==0)remotey*=-1;
                                      if(zdir==0)remotez*=-1;
          printf("\nRemote Entered: deltax=%d deltay=%d deltaz=%d",
                                remotex,remotey,remotez);
                                      sciflag=0;
                                      changetime();
                                      }
    }
}
```