Final Report: Motion Tracker

Amelia Peterson

11/20/12

Introduction

The final project for the class was decided upon earlier in the semester when the idea of making a motion sensing robot was brought up. We later changed it to the idea of the Sentry Turret from the game Portal where the turret would attack the main character when it sensed him. This project was feasible in our eyes because the necessary equipment was readily available (i.e. ranger sensors and K'Nex pieces) and coding didn't seem to be a difficult task at the time. Later there were complications that will be explained below. The whole point of the turret was for it to take an initial scan of its environment and placing those values into a memory chip on the C8051 and then continuously scan until it saw an anomaly. The turret would then in turn follow the anomaly until it stopped and then fire its "weapon" which in this case was a green laser. Unfortunately, the green laser was too heavy for the mount, and so it was not used. Overall, the project was worthwhile and a new learning experience and helped us understand the errors that could happen in a relatively simple project.

Procedure

Overview

The tracking device consists of a moveable mount, three ultrasonic range finders, a 10k potentiometer, an AM9128 external memory chip, an H-bridge, and a motor. The mount held the three ultrasonic range finders. The H-bridge was used to control the direction and speed of the motor, which moved both the mount and the potentiometer. The potentiometer defines the position of the mount, and the position defines which address is being written to in the AM9128. The AM9128 was used to store the data collected by the ultrasonic range finders. These components allowed the ultrasonic range finders to collect distance information about its surroundings and make movements based on that data.

Construction

The final project was based off the game Portal where there are "Sentry Turrets" in each level that attack the main character upon detection. The turrets look like oblong tripods that use lasers as their guidance system and are equipped with a machine gun of sorts. The intention of the project was to make a structure that had the same functionality of the Sentry Turret, but without the actual bullets. The choice of materials turned out to be K'Nex due to the availability and durability of the plastic toy. The initial planning for the structure involved making sure that the top disc would be able swivel at least 270 degrees with the sensors being attached to the top of the disc. The top disc would then be attached to a gear on the same axis (K'Nex stick) which would turn another gear. The other gear would then turn a screw that was attached to a potentiometer. Using the potentiometer as the method of knowing the orientation of the sensors, there was no need to control the motor in a detailed manner. This meant that the motor just had to be powerful enough to turn the top disc and gears. The motor used was a metal gearmotor made by Pololu which has a torque of about 130 oz-in. A rubber bandage material was then wrapped to the top of the motor so there would be enough friction to turn the gears on the main axis.

Once specifications of the gears and motor were in place, the structure was built around the size of the motor as well as the overall weight. Using blue K'Nex connectors, the turret's base was split into 4 quadrants to ensure stability. One of the quadrants held the motor while the opposite quadrant held a small circuit board along with the potentiometer. The circuit board connected all the wires from the rangers (motion sensors) to the C8051. Finally, the weapon of choice was decided to be a green laser that would fire upon the sensors seeing a difference in their environment. The laser was supposed to be mounted on the top disc along with the rangers so there wouldn't be a need for a separate motor. However, after the laser was mounted, it was discovered that it was too heavy for the motor to turn the mount. In the end, the structure was stable enough to turn the disc without tilt, but light enough that the motor could turn the center axis.

Position Sensor

In order to track moving objects, the relative position of the mount and sensors had to be determined. This was done by using a potentiometer, a bit piece to fit in the potentiometer, and two gears (see Appendix B, Figure 3). One gear was attached to the axis of rotation of the mount, and the other gear was attached to the axis of rotations of the bit piece and potentiometer. The two gears were attached to that when the mount moved, so did the potentiometer.

By using the ADC0, the value of the potentiometer could be read in. The ADC0 conversion value gave the position of the mount. This means the position of the device can range between 0x000 and 0xFFF (the number of bits the ADC0 conversion stores in 2^{12}). The position data was used as an address for storing the ranger data in the AM9128 memory chip.

Gathering and Storing Data

In order to get information about the surrounding, three ultrasonic range finders were used. The ultrasonic range finders return an unsigned int that holds data from 0 to 8 meters. The three rangers were placed facing the same direction and spaced apart slightly (see Appendix B, Figure 2). This allowed for a moving object to be tracked as it moved to the left or right.

Only one byte of the two available bytes from the rangers were used. This was because more positions could be looked at if less data was collected from the rangers. The AM9128 has 2^{11} words, which means either a maximum of 2^{10} 2 bytes values can be stored, or 2^{11} 1 byte values can be stored. Only the high byte of return by the ranger was used because accuracy in distance data was not needed.

Connecting the rangers to the C8051 consisted of a 4-wire connection to the I2C. As shown in Appendix A, Figure 1, each ranger required +5V power, GND, SCL (smbus clock), and SDA (smbus data line). To ensure that the rangers did not interfere with each others' signals to the I2C, the addresses of each had to be different. The address of each ranger initially was 0xE0. The three rangers were changed to have the addresses 0xE0, 0xEE, and 0xFC. These values were chosen because they allow for simple and efficient calculations in the program, and because address values that are too close together will not work if each component is on the same serial line.

The AM9128 was connected to the C8051 using the C8051's external memory interface. Additional glue logic was needed to decoded the C8051's address pins A11-A15 since the AM9128 only had 10 address pins. The EMI interface and glue logic connections to the AM9128 can be seen in Appendix A, Figure 4.

Tracking Algorithm

The function of the tracking algorithm is to detect when something in the environment has changed and to follow that movement. The algorithm compares the previously stored values in the environment. If any changes are detected in the environment are found, the motor will be moved left or right.

The algorithm begins initializing the environment array. Each address of the array is initialized to 0xFF, which will never be returned by the rangers. The program then moves the mount to the "home" position. This is defined by the MINPOS variable, and is the rightmost position (shown in Appendix B, Figure 2) in the range of the device. One the mount has moved this position, it begins moving left and collecting data from the ultrasonic rangers. The device moves, stops, gets the position from the potentiometer, pings each ranger separately, and stores the ranger data in the appropriate address of the AM9128. The address of each ranger is determined by the equations below.

Address of ranger 0xE0 = ADC0H
Address of ranger 0xEE = ADC0H+ARRAY_SIZE
Address of ranger 0xFC = ADC0H +2*ARRAY_SIZE

The program collects initial data until it reaches MAXPOS, which is the leftmost position. Once the environment has been initialized, the program begins moving the mount left and right between MINPOS and MAXPOS. To ensure the position has been initialized, the program checks if the value 0xFF is stored in the environment array at the current position. If 0xFF is sores in that address, then that position has not been initialized. The program initializes the array, and then moves on to a new position.

Once a properly initialized position has been found, the program checks the current ranger data with the previously stored ranger data. If there is a difference between the values, the program tracks the anomaly. The equations to calculate the motor movement based on the difference between the stored environment data and the current ranger data is shown below.

MEN =! (environment[position] && ping(1));
M2A = (environment[position+ARRAY_SIZE] && ping(0));
M1A = (environment[position+2*ARRAY_SIZE] && ping(2));

, where MEN is the motor enable bit, M2A and M1A are the digital outputs which control the direction of the motor (see Appendix A, Figure 5), and ping(x) pings ranger at address 0xE0+x*14. The equations show that the motor stops if the middle sensor detects an anomaly or if both the left and right sensors detect an anomaly. Otherwise, the motor will move the mount left if an anomaly is detected on the left, and the motor will move the mount right if an anomaly is detected on the right. The program will resume scanning the area for anomalies only when the movement is lost (that is when the ranger data matches the stored data).

Settings

The program has several variables which define the range and accuracy of the device. The list below shows each variable and their definitions.

Table 1: Definitions of settings.

Variable	Function	Range
MAXPOS	Leftmost position for the mount	0x000-0xfff
MINPOS	Rightmost position for the mount	0x000-0xfff
ACCURACY	Defines how many bits are used from the ADC0H as positioning data	0-7
ARRAY_SIZE	Defines how much space each ranger is allocated for storing data	0-256
RANGE	Defines the range of the range finders (e.g., 1 ft, 1m, etc.)	0x00-0xff
MOTOR_TIME	The number of timer 0 overflows for which the motor	-

	can be enabled	
ERROR_MARGIN	Defines how close the stored value in the environment must be to the current value of the of the rangers for the two values to be considered different.	0x00-0xff

The MAXPOS and MINPOS allow for the range of the mount to be adjusted. The maximum range of the mount is 270 degrees.ACCURACY defines how many positions there are. If ACCURACY is equal to 0, then all 8 bits of ADC0H are used. This means that there are $2^8 = 256$ different positions. The RANGE variable allows for the distance which the ranger detect (i.e., the distance at which a ping will time out) to be adjusted. The MOTOR_TIME variable adjusts how far the mount moves each time the move() function is called. The ERROR_MARGIN

Analysis

Goals Accomplished

The device was successfully able to track movement in its surroundings. The algorithm allowed for an object to be detected, followed, and centered with the middle ranger.

Issues Encountered

The two major issues with the device were the motor and the wires. The motor had a frictional tape around it to move the gear attached to the mount. However, this was an unreliable and inefficient method of turning the mount. The motor could not always successfully turn the mount. The other issue encounter were the wires leading from the rangers to the breadboard attached to the mount (see Appendix A, figure 2). These wires were very stiff and made it difficult for the motor to move the mount. The stiffness of the wires also caused them to move in and out of the pins of the breadboard, leading to the rangers to turn on and on randomly. This caused issues especially in the ranging because the range of the rangers is reset to 8 meters when they are restarted.

Another issue encountered was rapid pinging of the three rangers caused them to stop functioning. While this was easy to fix by inserting small delays, it greatly decreased the speed of the program. Using different rangers might possibly allow for a fast, more efficient device.

The function of the device would be improved greatly if rather than tape, a gear were attached to the motor, and if more flexible wires were used. The main issue with the device was smooth movement, and both the motor and the wires prevented this.

Optimal Settings

Table 2: 8 different settings for device and their optimal range values.

Setting	MAXPOS	MINPOS	ACCURACY	ARRAY_SIZE	Max RANGE
1	0xD5	0x15	0	256	0x17
2	0x70	0x0d	1	128	0x11
3	0x38	0x03	2	64	0x0B
4	0x1A	0x02	3	32	0x05

5	0x0D	0x01	4	16	0x03
6	0x06	0x00	5	8	0x01
7	0x03	0x00	6	4	0x01
8	0x01	0x00	7	2	0x01

The table above shows the different settings tested for varying range settings. The ACCURACY column defines how many bits of ADC0H are stored into the position variable, as defined by the equation below.

position = ADC0H>>ACCURACY ()

The purpose of having the number of bits from the ADC0H change was to adjust how large each position was. This was useful when the range of the rangers was adjusted. As the range of the rangers increases, the number of positions should increase so as to detect nuances in the environment. However, as the accuracy value increased, the initialize_environment function did not initialize each position, as it would pass over them. In the program, a check_position function was created to ensure that any position that had not been initialized in the initalize_environment function would be initialized. However, this could lead to errors if the data stored in check_position stored a change in the environment.

In order to find the optimal settings, the response of each setting was examined qualitatively. Setting 4 was found to be the most reliable. Setting 4 had the largest ARRAY_SIZE while still having each array position initialized in the initialize_environment function. This meant that this setting did not have some of the erratic behavior displayed by settings 1, 2, and 3. Although the maximum range of setting 4 was only about a foot, setting 4 was still the most accurate setting.

The optimal range of each setting was then examined. The rightmost column in table 2 shows the maximum range for the setting for which the program is still functional.

Conclusion

Although the movement of the device was not perfect, the algorithm did successfully track motion in its environment. The components preventing smooth movement could be easily fixed by finding a gear and ordering more flexible jumper wires. An improvement to the device would be made if the motor could more easily turn the mount. Even with the jerky movements from the motor, the algorithm was still shown to work in practice.

Appendix A: Schematics

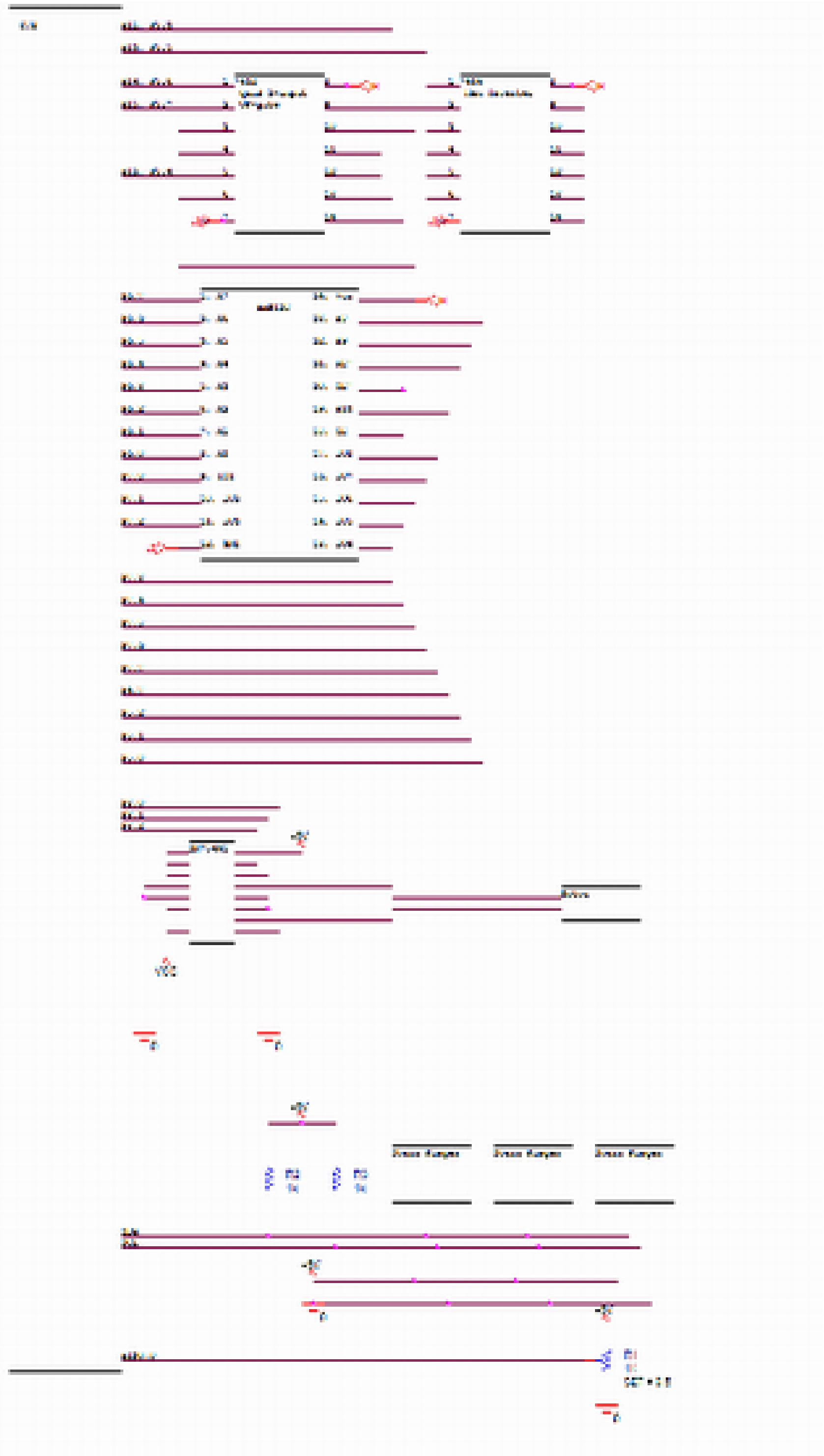

Figure 1: Schematic for device. Schematic shows the three rangers, the motor and the H-bridge which controls it, and the AM9128 external memory chip with glue logic.

Rangers

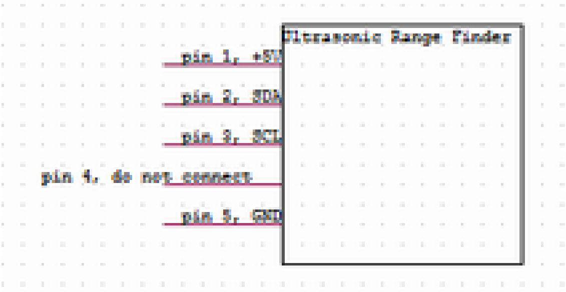

Figure 2: SRF08 Ultrasonic Range Finder pin-out.
Potentiometer

Figure 3: The 10k potentiometer [1.] *used. The pot can rotate 270 degrees and has a slot for placing a bit piece.*
AM9128

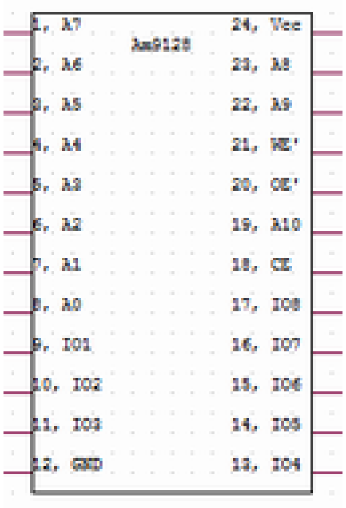

Figure 4: Pin-out for the AM9128 memory chip.
SN75441

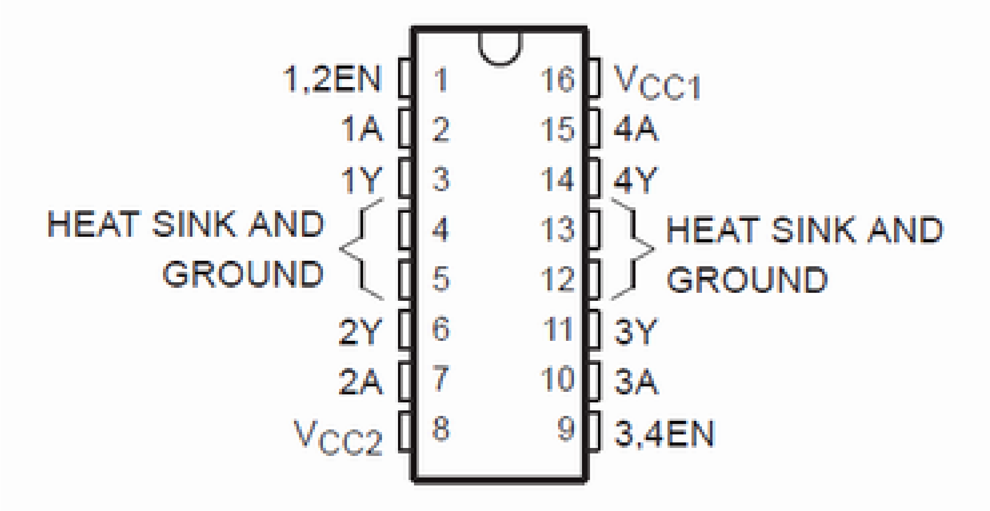

Figure 5: Pin-out for the SN75441 H-bridge [2].

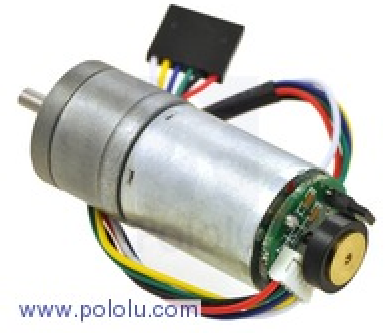

Figure 6: Motor used for moving the mount. [3]

Appendix B: Diagrams

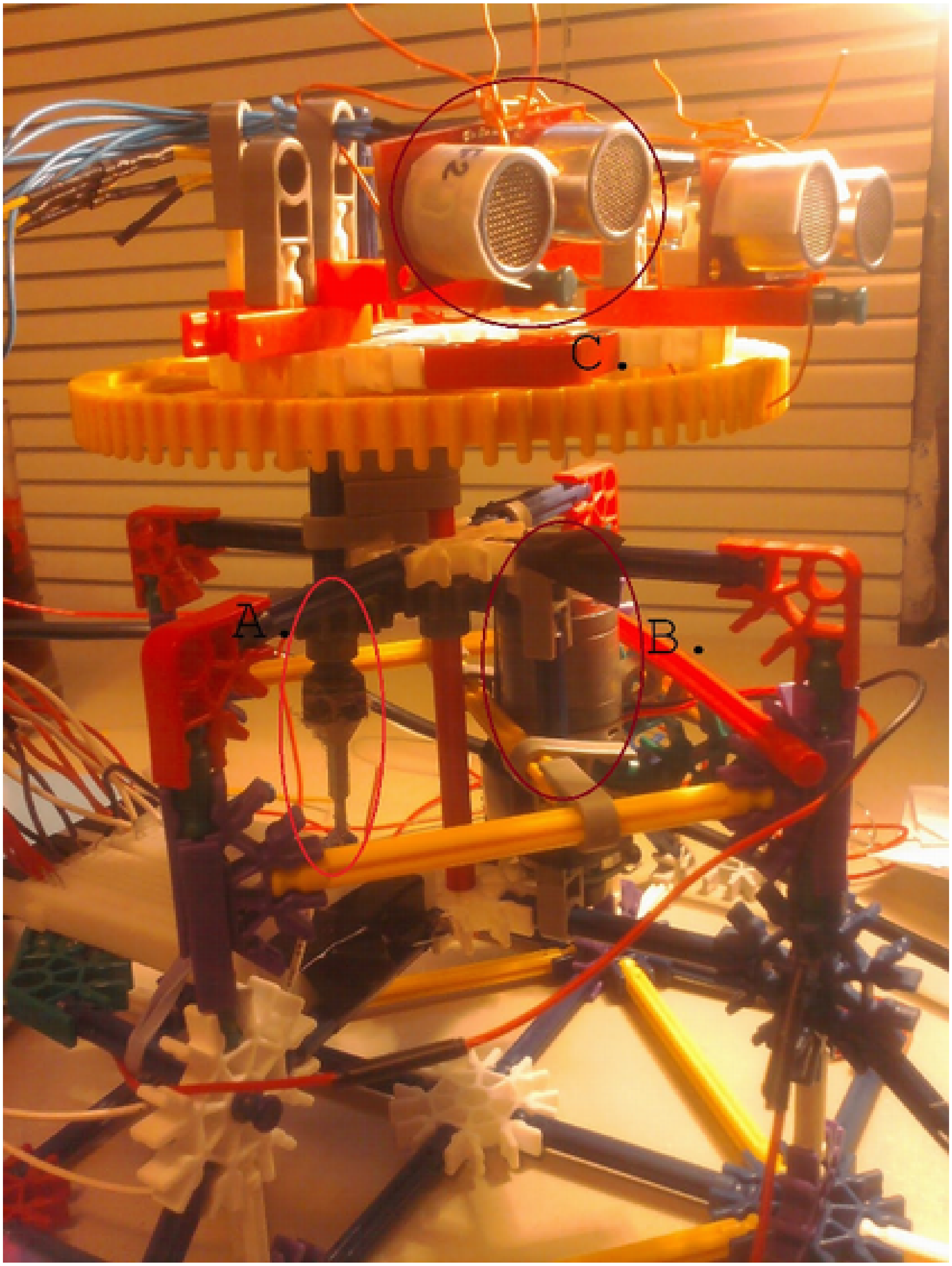

Figure 1: Diagram of device. A. : Potentiometer connected to bit piece and gear; B. : Motor; C. : Rightmost Ranger attached to mount.

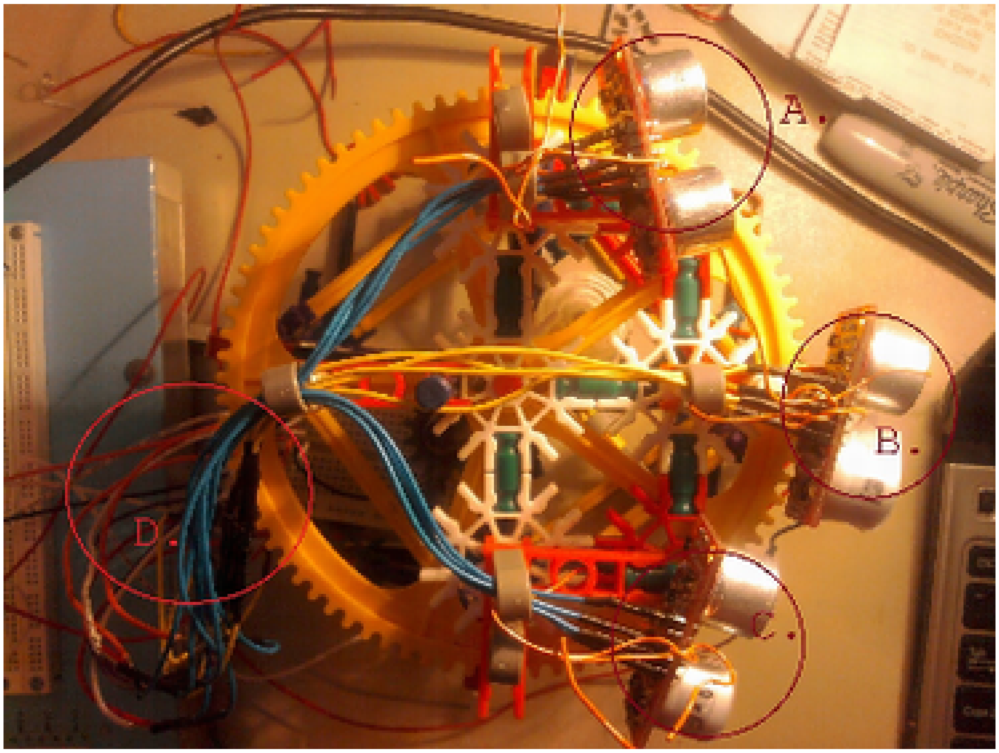

Figure 2: Top view of device. A. : Left Ranger; B. : Center Ranger; C. : Right Ranger; D. : Wires from Rangers to breadboard (+5V, GND, SDA, and SCL).

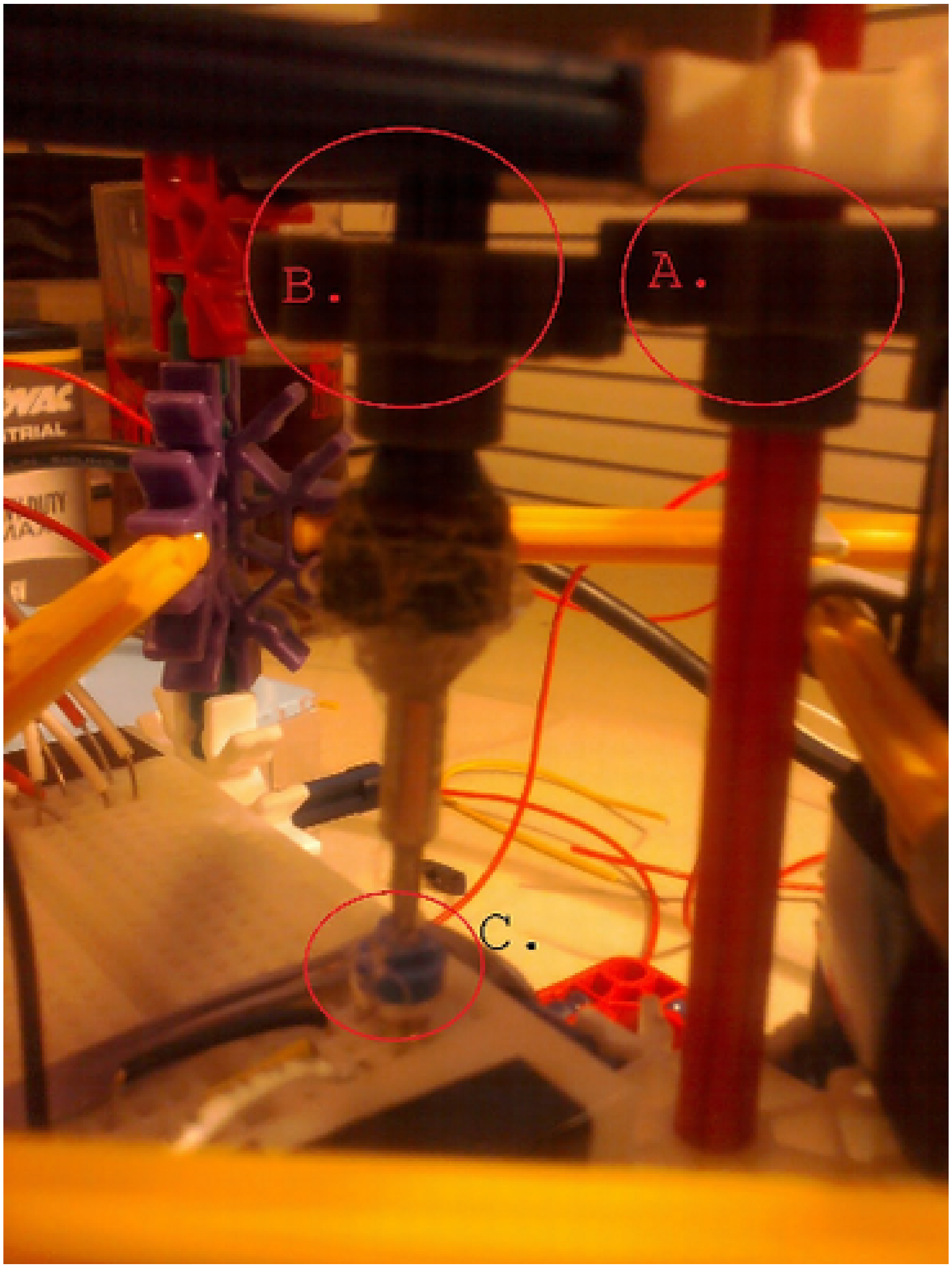

Figure 3: Potentiometer attached to mount's rotation axis. A. : Gear for mount; B. : Gear for potentiometer; C. : Potentiometer.

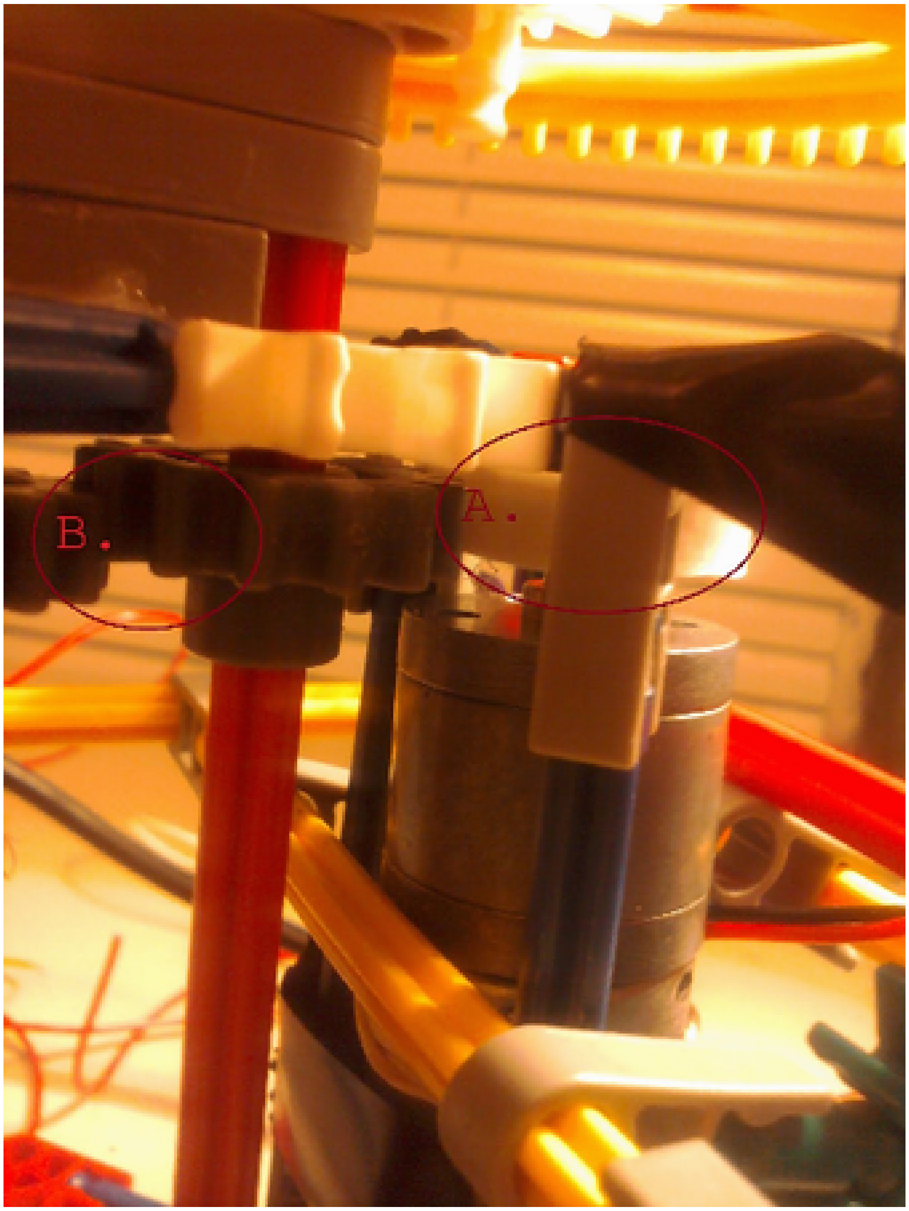

Figure 4: Movement control for device. A. : Motor and frictional tape (hockey tape) use to turn gear; B. : Gears for mount and potentiometer together.

Appendix C: Pseudocode

```
Initialize Sysclk, Uart0, ADC0, ports, EMI, Timer 0, and SMBus
Adjust Range of rangers
Initialize Environment
    initialize each element of environment array to 255
    move to MINPOS
    collect data until MAXPOS is reached
While(1)
    Move clockwise or counterclockwise within MINPOS and MAXPOS
    Stop
    while position not initialized
        store current distance
        Move to new position
        Stop
    Compare stored value with new value
    while movement detected
        if movement centered with center ranger
            stop
        else if movement on left and right
            stop
        else if movement on left
            move left
        else if movement on right
            move right
    end
end
```

Appendix D: Code

Tracking.c

```
/*
Amelia Peterson - 12/05/12
Microprocessor Systems Final Project Code
Tracking device and Turret gun

This program controls the movement of and collects data from a three
ultrasonic rangers. The program determines the position of the sensor mount
and grabs the distance from the sensors to detect motion in the environment.

The position of the ranger is defined by the voltage read from a potentiometer
attached to the motor. The voltage from the potentiometer ranges from
0V to 5V in a 270 degree radius. The potentiometer is attached to the
motor in such a way that it is adjusted with the movement of the mount
so a voltage reading from the pot corresponds directly to the relative
position of the mount. If 0V is defined as 0 degrees, then the position
of the mount is defined by equation 1.
    position = 270 (Vadc/5) [deg]    (1)

For this program, only a limited amount of data is recorded from the
environment since the external memory has not yet been interfaced.
Data from only 256 locations are stored, which means only the highest
two bits of the ADC0 conversion are used for position data.

The adjustable settings for the program are defined by the following variables:
const unsigned char MAXPOS - Leftmost position for the turret

const unsigned char MINPOS - Rightmost position for the turret

const unsigned char ACCURACY - number of bit shifts in ADC0H for positioning data

const unsigned int ARRAY_SIZE - ARRAY_SIZE = 2^(8-ACCURACY), determines how large each
array for sonic ranger distance data will be
```

```
unsigned char RANGE - Adjust range to certain distance. Shorter range will result in
faster pinging.

unsigned char MOTOR_TIME - Number of timer 0 overflows for which the motor is enabled

const unsigned char ERROR_MARGIN - Determines how close the stored value and current
ranger
value need to be for it to be considered an anomaly

Port 3 - Motion control for Motor 1, connected to SN75441
       Motor 1
       P3.0 - 1A
       P3.1 - 2A
       P3.2 - 1,2EN
Port 2 - Sonic Ranger Interface, connected to transistors
       S1            S2            S3
       P2.0 - D1     P2.1 - D1     P2.2 - D1
Port 0 - ADC0
       AIN0.0 (pin 47)- Potentiometer
Port 5 - EMI Hi-Address (A8-A15), connected to AM9128 Addressing pins and glue logic
       P5.0 - p23    P5.3 -              P5.6 -
       P5.1 - p22    P5.4 -              P5.7 -
       P5.2 - p19    P5.5 -
LOOK AT ALL THIS GODDAMN SPACE I HAVE FOR ACTIVITIES
Port 6 - EMI Lo-Address (A0-A7), connected to AM9128 Addressing pins
       P6.0 - p8     P6.3 - p5     P6.6 - p2
       P6.1 - p7     P6.4 - p4     P6.7 - p1
       P6.2 - p6     P6.5 - p3
Port 7 - EMI Data Bus (D0-D7), connected to AM9128 IO pins
       P7.0 - p9     P7.3 - p13    P7.6 - p16
       P7.1 - p10    P7.4 - p14    P7.7 - p17
       P7.2 - p11    P7.5 - p15
Port 4
       P4.6 - /RD
       P4.7 - /WR


Notes
       All positioning terms (left, right, front, back, etc.) are defined relative to
standing behind the device, with sonic rangers facing outward.
*/
#include <c8051f120.h>                                  // SFR declarations.
#include "motion.h"                                     // ADC0 intialization and
functions for motion control

void SYSCLK_init(void);
void UART0_init(void);                                  //UART0 is used for testing only
void Port_init(void);
void EMI_init(void);
void SMB_init(void);

void calibrate_EM(void);                         //Detect corrupted memory
void adjust_position(void);                             //Adjust position address to
avoid corrupted memory
void initialize_env(void);                       //Get initial data of environment
void detect_anomaly(void);                     //detect movement in environment

void initialize_env(void);                     //Get initial data of environment
void store_distance(void);                     //store current distance readings in
environment[]
void fire(void);
void detect_anomaly(void);                     //detect Movement

unsigned char corrupted_memory[10];            //Stores the addresses of the AM9128
which are corrupted
/*
unsigned char ranger_addr[1];                  //defined in ranger.h
volatile xdata at 0x2000 unsigned char* environment;//External memory starts at 0x2000
// External mamory ranges from 0x2000 to 0x2800 (2^11 bytes)
// Defined in motion.h
*/
void main(void){
       SYSCLK_init();
       UART0_init();
       ADC0_init();
       Port_init();
       EMI_init();
       TR0_init();
       SMB_init();

       WDTCN = 0xDE;                                 // Disable the watchdog timer
       WDTCN = 0xAD;                                 // Note: = "DEAD"!

       SFRPAGE = UART0_PAGE;
       printf("\033[2J");
       printf("Adjusting Range...\r\n");
       AdjustRange();                                      //Adjust the maximum range
of the rangers
       //calibrate_EM()                              //Save all corrupted addresses
       ping(0);
       ping(1);
       ping(2);
       getchar();
       printf("Initializing Environment...\r\n");
       initialize_env();                             //get initial environment
information
       while(1){
              //scan environment and continuously compare input data to stored
              // environment data
              move();                                              //Move slightly
clockwise or counter clockwise
              while(valid){
                     get_position();                       //Get position value
                     check_position();               //check if environment[position]
was initialized
                                                                   // If not, store
distance initialize distance information for
                                                                   // That position
and move slightly
```

```
             }
             valid = 1;                                     //reset valid bit
             track();                          //detect change in environment
             //if a difference in input data and stored environment is detected,
             // move turret to face change in environment
       }
}
/*
The purpose of calibrate_EM(void) is to find all corrupted addresses within the AM9128
chip and store them so as to avoid them
*/
void calibrate_EM(void){
       unsigned int addr;
       int i = 0;
       for(addr=0;addr<1024;addr++){                       //For all 1024 address
locations in the AM9128
             environment[addr] = 0xAA;                      //Write value 0xAA to
AM9128 at address 0xAA
             if(environment[addr]!=0xAA){                   //Check if correct data is
stored
                    corrupted_memory[i] = addr;                    //If correct data
is not stored, then memory is corrupted
                                                                                //
Save corrupted address
             }
       }
}
/*
adjust_position() adjusts the position addressing value to ensure
corrupted data is not accessed
*/
void adjust_position(void){
       char i = 0;
       while(position>corrupted_memory[i] || position == corrupted_memory[i] || i>10){
             position++;
             i++;
       }
}
/*
initialize_env() does a 270 degree sweep of environment and grabs distance
data for each position the mount moves to; stores data in environment[]
*/
void initialize_env(void){
       unsigned int addr;                                   //Value to store current
address being initialized

       printf("Inititalizing Array...\r\n");

       for(addr=0;addr<3*(ARRAY_SIZE);addr++){
             environment[addr] = 0xFF;               //Initalize each location to
0xFF (sensor data will never)
                                                                         // give 0xFF
- That value is sent by the ranger
                                                                         //
specifically when a ping has not completed
```

```
       }

       printf("Move home...\r\n");
       home();                                                                //Move to
far left position

       printf("Perform sweep for initial data...\r\n");
       get_position();                                                 //Get current
position
       while(position<MAXPOS){                                  //While sweep is not
complete; MAXPOS refers to the rightmost postion
             move();                                                          //Move
turret slightly
             store_distance();                                    //Store distance data for
current position
       }
       store_distance();                                          //Store final distance for
MAXPOS
       printf("Environment Initialized\r\n");
}

void fire(void){
       //Trigger Motor 2 to fire turret
       //motor_control(1,PWM,1);
}
/*
detect_anomaly() compares new ranger reading to stored environment.
If the new reading and stored reading do not match, then the system is
told to begin tracking the detected movement
*/
void detect_anomaly(void){
       unsigned char ping_1 = ping(1);
       unsigned char ping_0 = ping(0);
       unsigned char ping_2 = ping(2);
       printf("Detecting anomaly...\r\n");
       printf("Environment[position] - %x, ping(1) -
%x\r\n",environment[position],ping_1);
       printf("Environment[position+ARRAY_SIZE] - %x, ping(0) -
%x\r\n",environment[position+ARRAY_SIZE],ping_0);
       printf("Environment[position+2*ARRAY_SIZE] - %x, ping(2) -
%x\r\n",environment[position+2*ARRAY_SIZE],ping_2);
       printf("Detecting Anomalies...");
       if(ping_1 != environment[position]){
             printf("Anomaly detected\r\n");
             track();
       }
       else if(ping_0 != environment[(position+ARRAY_SIZE)]){
             printf("Anomaly detected\r\n");
             track();
       }
       else if(ping_2 != environment[(position+2*ARRAY_SIZE)]){
             printf("Anomaly detected\r\n");
             track();
       }
```

```
}

void SYSCLK_init(void){
       int i;
       char SFRPAGE_SAVE;

       SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page SFRPAGE = CONFIG_PAGE;
       SFRPAGE = CONFIG_PAGE;

       OSCXCN = 0x67;              // Start ext osc with 22.1184MHz crystal
       for(i=0; i < 3000; i++);    // Wait for the oscillator to start up
       while(!(OSCXCN & 0x80));
       CLKSEL = 0x01;              // Switch to the external crystal oscillator
       OSCICN = 0x00;                  // Disable the internal oscillator

       SFRPAGE = SFRPAGE_SAVE;     // Restore SFR page
}

void UART0_init(void){
       char SFRPAGE_SAVE;

       SFRPAGE_SAVE = SFRPAGE;                         // Save Current SFR page
       SFRPAGE = TIMER01_PAGE;

       TCON    = 0x40;
       TMOD   &= 0x0F;
       TMOD   |= 0x20;                                 // Timer1, Mode 2, 8-bit reload
       CKCON  |= 0x10;                                 // Timer1 uses SYSCLK as time
base
//     TH1           = 256 - SYSCLK/(BAUDRATE*32);//  Set Timer1 reload baudrate value
T1 Hi Byte
       TH1           = 0xEE;                                 // 0xE8 = 232
       TR1           = 1;                                    // Start Timer1

       SFRPAGE = UART0_PAGE;
       SCON0   = 0x50;                                 // Mode 1, 8-bit UART, enable RX
       SSTA0   = 0x00;                                 // SMOD0 = 0, in this mode
                                                                    // TH1 = 256 -
SYSCLK/(baud rate * 32)

       TI0 = 1;                                              // Indicate TX0 ready

       SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}

void Port_init(void){
       char SFRPAGE_SAVE = SFRPAGE;
       SFRPAGE = CONFIG_PAGE;
       //Initialize crossbar, UART0, doesn't need crossbar for ACDC0
       //unless using external trigger to start conversion
       XBR0 |= 0x05;                    //Enable UART0, TX on P0.0, RX on P0.1;
UART0EN = 1
                                                     //SDA = P0.2, SCL = P0.3
       XBR2 |= 0x40;                    //Enable Crossbar
       P0MDOUT &= ~0x0C;                //Set SDA and SCL (P0.3 and P0.2) to push-pull
       P0 |= 0xFF;

       P3MDOUT &= ~0x03;                //Set P3.0-3.3 to output for motor control

       P3 |= 0x03;                      //Disable motor
       //H-Bridge input - 2 digital output pins and 1 PWM for each of the two motors
[Port 3]
       //Potentiometer input on ADC0
       EA = 1;                                 //Enable global interrupts
       ET0 = 1;                         //Enable Timer 0 Interrupts
       SFRPAGE = SFRPAGE_SAVE;
}
void EMI_init(void){
       char SFRPAGE_SAVE = SFRPAGE;
       SFRPAGE = CONFIG_PAGE;
       P0MDOUT |= 0x01;    // Set TX0 pin to push-pull
       P4MDOUT = 0xFF;     // Output configuration for P4 all pushpull
       P5MDOUT = 0xFF;     // Output configuration for P5 pushpull EM addr
       P6MDOUT = 0xFF;     // Output configuration for P6 pushpull EM addr
       P7MDOUT = 0xFF;     // Output configuration for P7 pushpull EM data

       P5 = 0xFF;
       P6 = 0xFF;
       P7 = 0xFF;

       // EMI_Init, split mode with no banking

       SFRPAGE = EMI0_PAGE;
       EMI0CF = 0x3b;              //34
       EMI0TC = 0xFF;
       SFRPAGE = SFRPAGE_SAVE;
}

void SMB_init(void){
       SMB0CR = 0x93;
       ENSMB = 1;
}
```

motion.h

```
/*
Amelia Peterson - 11/25/12
Microprocessor Systems Final Project Code
Motion Control
*/
#include <c8051f120.h>                  // SFR declarations.
#include "timing.h"                     //Timer 0 initializations and interrupt routine
#include "ranger_control.h"

void ADC0_init(void);                   //The ADC is used to read in the voltage
```

```
                                                                            // from a potentiometer which defines
the position of the sensor/turret mount
void get_position(void);                        //Read voltage from ADC0
void move(void);                                        //Move mount slightly
void home(void);                                        //Move mount to 0 degrees (0V from ADC0)
void check_position(void);                      //Ensure environment data was intitialized for this position
void store_distance(void);                      //store current distance readings in environment[]
void motor_control(char position);      //Control motor motion
void track(void);


__sbit __at 0xB0 M_1A;                                  //H-bridge 1A pin connected to P3.0
__sbit __at 0xB1 M_2A;                                  //H-bridge 2A pin connected to P3.1
__sbit __at 0xB2 M_EN;                                  //H-bridge 1,2EN pin connected to P3.2

volatile xdata at 0x2000 unsigned char* environment;    //External memory starts at 0x2000

        // External mamory ranges from 0x2000 to 0x2800 (2^11 bytes)
unsigned char position;                                                                 //Stores ADC0H
conversion; Position is defined by the input
        // voltage from the potentiometer
bit valid = 1;
#define MAXPOS 0x0d                                     //Rightmost position for the turret
#define MINPOS 0x03                                                     //Leftmost position for the turret
#define ACCURACY 4                                              //number of bit shifts in ADC0H for positioning data
#define ARRAY_SIZE 16                           //ARRAY_SIZE = 2^(8-ACCURACY), determines how large each array
        // for sonic ranger distance data will be
//const unsigned char ERROR_MARGIN = 0xFF;                      //Determines how close the stored value and
current ranger
        // value need to be for it to be considered an anomaly

void ADC0_init(void){
        char SFRPAGE_SAVE = SFRPAGE;
        SFRPAGE = ADC0_PAGE;

        ADC0CN |= 0x80;    //enable AD0
        ADC0CN &= ~0x4D;        //Set to continuously track input, Initiate sampling manually,
        ADC0CN |= 0x01;         //Left-justify ADC0 data registers
        AMX0CF &= ~0x01;        //AIN0.0, AIN0.1 are independent single-ended inputs
        AMX0SL &= ~0x0F; //AIN0.0 channel select
        REF0CN &= ~0x11; //Voltage reference from VREF0
        REF0CN|= 0x02;          //Internal Bias generator on, Internal reference voltage is driven on Vref (+2.4V)
        ADC0CF &= ~0x06; //set PGA, gain = 1
        ADC0CF |= 0x70;    //determine SAR, must be less than 2.5MHz AD0SC = (SYSCLK)/(2*CLK) ~= 14
        ADC0CF &= ~0x01;

        AD0INT = 0;

        SFRPAGE = SFRPAGE_SAVE;
}
/*
```

```
get_position() performs an ADC conversion on ADC0. The ADC0 conversion
represents the position of the mount.
*/
void get_position(void){
        //read voltage from ADC0, store in position variable
        char SFRPAGE_SAVE = SFRPAGE;
        SFRPAGE = ADC0_PAGE;

        AMX0SL = 0;
        AD0INT = 0;
        AD0BUSY = 1;                            //Start analog conversion

        SFRPAGE = ADC0_PAGE;

        while(!AD0INT){}                //Wait for conversion to complete
        AD0BUSY = 0;                            //Clear conversion complete flag

        position = ADC0H>>ACCURACY;                     //Store the high byte of the conversion
        printf("Get Position: %x\r\n", position);//print position in hex

        SFRPAGE = SFRPAGE_SAVE;
}

/*
move() moves quickly turns on and off the motor by enabling Timer 0
and keeping track of a certain number of overflows (defined by the MOTOR_TIME variable)
during which the motor will be enabled. The function also ensures that once the leftmost
or rightmost position is reached, the motor will reverse direction.
*/
void move(void){
        //determine whether to move left or right
        get_position();
        if(position==MAXPOS){                           //if mount is as rightmost position
                P3 &= ~0x01;                                    //Start moving the motor left
                P3 |= 0x02;
        }
        else if(position==MINPOS){              //if mount is in leftmost position
                P3 |= 0x01;                                             //Start moving motor right
                P3 &= ~0x02;
        }
        //otherwise, motor is moving in correct direction
        TL0 = 0;                                                //reset Timer 0 low byte
        TH0 = 0;                                                //reset Timer 0 high byte
        overflow_counter = 0;                           //reset overflow counter
        motor_flag = 0;                                         //reset motor flag
        TR0 = 1;                                                //Start timer
        while(!motor_flag);                             //wait for Timer 0 to finish counting
                                                                                // motor will move until then
        //Timer 0 was disabled in ISR 0
}
```

```
/*
home() moves the turret to the leftmost position
*/
void home(void){
        //start moving
        P3 &= ~0x01;                                            //Start moving the motor left
        P3 |= 0x02;
        get_position();                                         //Get initial position
        while(position>MINPOS){                         //While mount is not in leftmost position
                get_position();                                 //Update position
                move();
        }
        P3 |= 0x01;                                             //Start moving the motor left
        P3 &= ~0x02;
        //stop moving
}
/*
check_position() checks to see if the position had been previously initialized.
initialize_environment might not initialize every position, and so the position
must be checked.
*/
void check_position(void){
///     printf("Checking if position %x was initialized...\r\n",position);
        if(environment[position]==0xFF){                //If the address holds the array initialization value of 255...
                printf("Position not initialized: environment[%x]: %x, initializing...\r\n", position,environment[position]);
                store_distance();                               //Store the current distance
                move();                                                         //move to next position
                valid = 1;                                              //return TRUE - get next position data
                return;
        }
        printf("Position initialized, returning to main function\r\n");
        valid = 0;                                                      //return FALSE - position was already
initialized
}
/*
store_distance() stores the the ranger values for the current position
in the environment[] array.
*/
void store_distance(){
        //get_position();
                //get current position
        environment[position] = ping(1);                                                        //store
distance from left ranger
        environment[position+ARRAY_SIZE] = ping(0);                                     //store distance
from center ranger
        environment[position+2*ARRAY_SIZE] = ping(2);                           //store distance from right
ranger

        printf("\rStoring distance, environment[%x]: %x\r\n", position, environment[position]);
        printf("\rStoring distance, environment[%x]: %x\r\n", position+ARRAY_SIZE, environment[position+ARRAY_SIZE]);
        printf("\rStoring distance, environment[%x]: %x\r\n", position+2*ARRAY_SIZE, environment[position+2*ARRAY_SIZE]);
```

```
}

void track(void){
    //char fire_counter = 0;
    bit t_MEN, t_M1A, t_M2A;
    bit M1A, M2A, MEN;                          //bits to determine motor direction
    t_MEN = M_EN;
    t_M1A = M_1A;
    t_M2A = M_2A;
    valid=1;
    M1A = 1;                                    //initialize each to ensure entering while loop
    M2A = 1;
    MEN = 1;
    printf("TRACKING\r\n");
    while(!(M1A==0 && M2A==0 && MEN==0)){       //The loop breaks ONLY if the target is lost completely
        while(valid){
            get_position();
            check_position();
        }
        valid = 1;
        MEN = (environment[position] && ping(1));                   //Will be either 0 or 1
        M2A = (environment[position+ARRAY_SIZE] && ping(0));
        M1A = (environment[position+2*ARRAY_SIZE] && ping(2));
/*
        MEN = ((environment[position]&&MARGIN_ERROR) && (ping(1)&&MARGIN_ERROR));
    //Will be either 0 or 1
        M2A = ((environment[position+ARRAY_SIZE]&&MARGIN_ERROR) && (ping(0)&&MARGIN_ERROR));
        M1A = ((environment[position+2*ARRAY_SIZE]&&MARGIN_ERROR) && (ping(2)&&MARGIN_ERROR));
*/


        /*
        if(dif[0]){
            M_EN = dif[0];
            fire();
            fire_counter++;
            if(fire_counter>3){
                //wait for 3 seconds
                store_distance();
                return;
            }
        }
        else{
            M_EN = dif[0];
            M_1A = dif[1];
            M_2A = dif[2];
            move();
        }
        */
        printf("M2A: %x     ",M2A);
        printf("MEN: %x     ",MEN);
        printf("M1A: %x \r\n",M1A);
```

```
		//If the object is detected in front of both the right and left sensors, the device will stop moving
		// because on the SN75441 chip both inputs pulled high corresponds to a 'fast stop'
		M_EN = !MEN;				// If the object is detected in front of the ranger, disable the motor
		M_1A = M2A;
		M_2A = M1A;
	//	move();
	}
	printf("stop tracking\r\n");
	M_EN = t_MEN;
	M_2A = t_M2A;
	M_1A = t_M1A;
}
```

timing.h

```
/*
Amelia Peterson - 11/23/12
Microprocessor Systems Final Project Code
Timer 0 initializations and ISR 1
*/
#include <c8051f120.h>                                      // SFR declarations.

#define MOTOR_TIME 2                           //Number of  timer 0 overflows for
which the motor is enabled

void TR0_init(void);                                        //Timer 0 is used for PWM
void TR0_Overflow(void) interrupt 1;           //Timer 0 Overflow interrupt for PWM

bit motor_flag = 0;                                         //Flag for moving motor
char overflow_counter;                                      //counts the number of
overflows of Timer 0

void TR0_init(void){
       CKCON |= ~0x0C;                                      //Use SYSCLK/12 as source
       TMOD &= ~0x0E;                                              //Timer enabled
when TR0=1 irrespective
       TMOD |= 0x01;                                               // of /INT0 logic,
Timer 0 incremented
                                                                          //by SYSCLK,
Mode 1: 16-bit counter/timer
       TR0=0;
       TL0=0;
       TH0=0;
}

void TR0_Overflow(void) interrupt 1{
       //TF0 overflow flag is automatically cleared when interrupt is entered
       TR0 = 0;                                                    //stop timer 0
       TH0 = 0;                                                    //reset TH0
       TL0 = 0;                                                    //reset TL0
       if(overflow_counter==0)
             P3 |= 0x04;                                           //Set P3.3 (Motor 1
enable) to high
```

```
       if(overflow_counter==MOTOR_TIME){
              P3 &= ~0x04;                                  //Set P3.3 (Motor 1 enable)
to low
              motor_flag = 1;                                      //set motor flag
              TR0 = 0;                                             //stop the timer
              return;
       }
       overflow_counter++;                                  //increment overflow
counter
       TR0 =1;                                                            //restart
the timer
}
```

ranger_control.h

```
/*
Amelia Peterson - 12/03/12
Microprocessor Systems Final Project Code
Sonic Ranger Control
*/
#include <c8051f120.h>           // SFR declarations.
#include <stdio.h>                      // Necessary for printf (only necessary for
testing)
#include "putget.h"                     // Necessary for printf (only necessary for
testing)

void AdjustRange(void);                 //Change Range of Rangers; 43mm - 11m in
increments of 43mm
unsigned char ping(char ranger); //Ping 1 of the 3 rangers
void PingRanger(void);
unsigned char ReadRanger(void);
void i2c_write_and_stop(unsigned char output_data);
void i2c_write(unsigned char output_data);
void i2c_start(void);
void i2c_write_data(unsigned char addr, unsigned char start_reg, unsigned char
*buffer, unsigned char num_bytes);
unsigned char i2c_read(void);
unsigned char i2c_read_and_stop(void);
void i2c_read_data(unsigned char addr, unsigned char start_reg, unsigned char *buffer,
unsigned char num_bytes);

#define RANGE 0x01                                                 //Adjust range to 1
ft

unsigned char ranger_addr[1];                               //Address of Sonic Ranger

//Range is reset at power up
void AdjustRange(void){
       unsigned char Data[1];
       Data[0] = RANGE;
       printf("Adjusting range of ranger 0xE0...\r\n");
       i2c_write_data(0xE0, 2, Data, 1);
       printf("Adjusting range of ranger 0xEE...\r\n");
       i2c_write_data(0xEE, 2, Data, 1);
       printf("Adjusting range of ranger 0xFC...\r\n");
       i2c_write_data(0xFC, 2, Data, 1);
```

```
}
//char ranger is defined as 0, 1, or 2 -
// 0 - 0xE0 (left ranger)
// 1 - 0xE1 (center ranger)
// 2 - 0xE2 (right ranger)
unsigned char ping(char ranger){
        unsigned char distance;
        ranger_addr[0] = 0xE0+(ranger*14);
//      printf("Pinging ranger %x\r\n",ranger_addr[0]);
        PingRanger();                           //delay to wait for ping to complete handled
in PingRanger() function
        distance = ReadRanger();
        return distance;
}

void PingRanger(void){
        unsigned char wait=0;
        unsigned int i;
        unsigned char Data[1];
        Data[0] = 0x51;
//      printf("Pinging...\r\n");
        i2c_write_data(ranger_addr[0], 0, Data, 1);
        while(wait<2){
                for(i=0;i<65530;i++);
                wait++;
        }
}
unsigned char ReadRanger(void){
        unsigned int distance;
        unsigned char Data[2];
        Data[1] = 255;
        while(Data[1] == 255){          //Wait until ranger is done pinging
//              printf("pinging\r\n");
                i2c_read_data(ranger_addr[0], 2, Data, 2); //read two bytes, starting at
reg 2
                distance = (((unsigned int) Data[0]<<8)|Data[1]);
//              printf("Data[1]: %x\r\n", Data[1]);
        }
        return Data[1];                                 //return low byte
}
void i2c_start(void){
        while(BUSY);                            //Wait until SMBus0 is free
        STA  = 1;                                       //Set Start Bit
        while(!SI);                                     //Wait until start sent
        STA = 0;                                        //clear start bit
        SI = 0;                                         //Clear SI
}
void i2c_write(unsigned char output_data){
        SMB0DAT = output_data;                  //Data to be written put into register
        while(!SI);                                     //Wait until send is complete
        SI = 0;                                         //Clear SI
}
void i2c_write_and_stop(unsigned char output_data){
        SMB0DAT = output_data;                  //Data to be written put into register
```

```
       STO = 1;                                //Set Stop bit
       while(!SI);                             //Wait until send is complete
       SI = 0;                                 //clear SI
}
void i2c_write_data(unsigned char addr, unsigned char start_reg, unsigned char
*buffer, unsigned char num_bytes){
       unsigned char i;                               //counter variable

       i2c_start();                            //initiate i2c transfer
       i2c_write(addr & ~ 0x01);               //write the desired address to the bus
       i2c_write(start_reg);                          //write the start address to the
bus
       for(i=0; i<num_bytes-1;i++)                    //write the data to the regusters
              i2c_write(buffer[i]);
       i2c_write_and_stop(buffer[num_bytes-1]);//Stop transfer
}
unsigned char i2c_read(void){
       unsigned char input_data;
       while(!SI);                             //Wait until data is available to read
       input_data = SMB0DAT;            //Read the data
       SI = 0;                                 //Clear SI
       return input_data;               //Return the data
}
unsigned char i2c_read_and_stop(void){
       unsigned char input_data;
       while(!SI);                             //Wait until data is available to read
       input_data = SMB0DAT;            //Read the data
       SI = 0;                                 //Clear SI
       STO = 1;                                //Set stop bit
       while(!SI);                             //wait for stop bit
       SI = 0;
       return input_data;               //Return the read data
}
void i2c_read_data(unsigned char addr, unsigned char start_reg, unsigned char *buffer,
unsigned char num_bytes){
       unsigned char j;                               //counter variable
       i2c_start();                            //Start i2c transfer
       i2c_write(addr & ~0x01);                //write address of device that will be
written to, send 0
       i2c_write_and_stop(start_reg);          //Write and stop the first register to
be read

       i2c_start();                            //Start i2c transfer
       i2c_write(addr | 0x01);                        //indicating a read operation

       for(j=0;j<num_bytes-1;++j){
              AA = 1;                                 //Set acknowledge bit
              buffer[j] = i2c_read();                 //Read data, save it in buffer
       }
       AA = 0;
       buffer[num_bytes-1] = i2c_read_and_stop();     //Read in last byte and stop,
save it in the buffer, end function
}
```

Change_Address.c

```
/*
Amelia Peterson - 12/03/12
Microprocessor Systems Final Project Code
Change of Address on Sonic Ranger
*/
#include <c8051f120.h>                  // SFR declarations.
#include <stdio.h>                      // Necessary for printf (only necessary for
testing)
#include "putget.h"                     // Necessary for printf (only necessary for
testing)

void SMB_init(void);
void SYSCLK_init(void);
void UART0_init(void);
void Port_init(void);
void change_address(void);
unsigned int ReadRanger(void);
void PingRanger(void);

//SMBUS FUNCTIONS
void i2c_write_and_stop(unsigned char output_data);
void i2c_write(unsigned char output_data);
void i2c_start(void);
void i2c_write_data(unsigned char addr, unsigned char start_reg, unsigned char
*buffer, unsigned char num_bytes);
unsigned char i2c_read(void);
unsigned char i2c_read_and_stop(void);
void i2c_read_data(unsigned char addr, unsigned char start_reg, unsigned char *buffer,
unsigned char num_bytes);

unsigned char addr[1];

void main(void){
       unsigned int distance;
       SYSCLK_init();
       UART0_init();
       Port_init();
       SMB_init();

       WDTCN = 0xDE;                    // Disable the watchdog timer
       WDTCN = 0xAD;                    // Note: = "DEAD"!

       SFRPAGE = UART0_PAGE;

       printf("\033[2J");               //clear the screen
       printf("Address Change Program - Sonic Ranger address will ");
       printf("be changed from 0xE0 to addr\r\n");

       addr[0] = 0xFC;                         //set address
       change_address();                //change address
       printf("Address changed, now pinging ranger...\r\n");
//     printf("\033[3,20r");            //Enable scrolling from line 3 to 20
       while(1){
              distance = ReadRanger();
              PingRanger();
              printf("\033[5C");        //indent
              printf("%d\r\n",distance); //print distance reading
       }
}
void SMB_init(void){
       SMB0CR = 0x93;
       ENSMB = 1;
}
void SYSCLK_init(void){
       int i;
       char SFRPAGE_SAVE;

       SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page SFRPAGE = CONFIG_PAGE;
       SFRPAGE = CONFIG_PAGE;

       OSCXCN = 0x67;              // Start ext osc with 22.1184MHz crystal
       for(i=0; i < 3000; i++);    // Wait for the oscillator to start up
       while(!(OSCXCN & 0x80));
       CLKSEL = 0x01;              // Switch to the external crystal oscillator
       OSCICN = 0x00 ;             // Disable the internal oscillator

       SFRPAGE = SFRPAGE_SAVE;     // Restore SFR page
}

void UART0_init(void){
       char SFRPAGE_SAVE;

       SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page
       SFRPAGE = TIMER01_PAGE;

       TCON = 0x40;
       TMOD &= 0x0F;
       TMOD |= 0x20;               // Timer1, Mode 2, 8-bit reload
       CKCON |= 0x10;              // Timer1 uses SYSCLK as time base
       // TH1 = 256 - SYSCLK/(BAUDRATE*32)  Set Timer1 reload baudrate value T1 Hi Byte
       TH1 = 0xE8;                 // 0xE8 = 232
       TR1 = 1;                    // Start Timer1

       SFRPAGE = UART0_PAGE;
       SCON0 = 0x50;            // Mode 1, 8-bit UART, enable RX
       SSTA0 = 0x00;            // SMOD0 = 0, in this mode
                                  // TH1 = 256 - SYSCLK/(baud rate * 32)

       TI0 = 1;                    // Indicate TX0 ready

       SFRPAGE = SFRPAGE_SAVE;     // Restore SFR page
}

void Port_init(void){
       char SFRPAGE_SAVE = SFRPAGE;
       SFRPAGE = CONFIG_PAGE;
       //Initialize crossbar, UART0, doesn't need crossbar for ACDC0
       //unless using external trigger to start conversion
       XBR0 |= 0x05;                    //Enable UART0, TX on P0.0, RX on P0.1;
UART0EN = 1
                                                     //SDA = P0.2, SCL = P0.3
       XBR2 |= 0x40;                    //Enable Crossbar
       P0MDOUT &= ~0x0C;                //Set SDA and SCL (P0.3 and P0.2) to push-pull
       P0 |= 0x0C;

       EA = 1;                                 //Enable global interrupts
       SFRPAGE = SFRPAGE_SAVE;
}
void change_address(void){
       unsigned char cmd_1[1], cmd_2[1], cmd_3[1];
       cmd_1[0] = 0xA0;
       cmd_2[0] = 0xAA;
       cmd_3[0] = 0xA5;
       i2c_write_data(0xFE,0,cmd_1,1);  //first command for address change
       i2c_write_data(0xFE,0,cmd_2,1);  //second command for address change
       i2c_write_data(0xFE,0,cmd_3,1);  //third command for address change
       i2c_write_data(0xFE,0,addr,1);   //Address will be changed from 0xE0 to addr
}
////////////////////////PING RANGER////////////////////
void PingRanger(void){
       unsigned char Data[1];
       Data[0] = 0x51;
       i2c_write_data(addr[0], 0, Data, 1);
}

////////////////////////READ RANGER////////////////////
unsigned int ReadRanger(void){
       unsigned int distance;
       unsigned char Data[2];
       i2c_read_data(addr[0], 2, Data, 2); //read two bytes, starting at reg 2
       distance = (((unsigned int) Data[0]<<8)|Data[1]);
       return distance;
}
void i2c_start(void){
       while(BUSY);                     //Wait until SMBus0 is free
       STA  = 1;                               //Set Start Bit
       while(!SI);                             //Wait until start sent
       STA = 0;                                //clear start bit
       SI = 0;                                 //Clear SI
}
void i2c_write(unsigned char output_data){
       SMB0DAT = output_data;           //Data to be written put into register
       while(!SI);                             //Wait until send is complete
       SI = 0;                                 //Clear SI
}
void i2c_write_and_stop(unsigned char output_data){
       SMB0DAT = output_data;           //Data to be written put into register
       STO = 1;                                //Set Stop bit
       while(!SI);                             //Wait until send is complete
       SI = 0;                                 //clear SI
}
void i2c_write_data(unsigned char addr, unsigned char start_reg, unsigned char
*buffer, unsigned char num_bytes){
       unsigned char i;                               //counter variable

       i2c_start();                            //initiate i2c transfer
       i2c_write(addr & ~ 0x01);               //write the desired address to the bus
       i2c_write(start_reg);                          //write the start address to the
bus
       for(i=0; i<num_bytes-1;i++)                    //write the data to the regusters
             i2c_write(buffer[i]);
       i2c_write_and_stop(buffer[num_bytes-1]);//Stop transfer
}
unsigned char i2c_read(void){
       unsigned char input_data;
       while(!SI);                             //Wait until data is available to read
       input_data = SMB0DAT;            //Read the data
       SI = 0;                                 //Clear SI
       return input_data;               //Return the data
}
unsigned char i2c_read_and_stop(void){
       unsigned char input_data;
       while(!SI);                             //Wait until data is available to read
       input_data = SMB0DAT;            //Read the data
       SI = 0;                                 //Clear SI
       STO = 1;                                //Set stop bit
       while(!SI);                             //wait for stop bit
       SI = 0;
       return input_data;               //Return the read data
}
void i2c_read_data(unsigned char addr, unsigned char start_reg, unsigned char *buffer,
unsigned char num_bytes){
       unsigned char j;                               //counter variable
       i2c_start();                            //Start i2c transfer
       i2c_write(addr & ~0x01);                //write address of device that will be
written to, send 0
       i2c_write_and_stop(start_reg);          //Write and stop the first register to
be read

       i2c_start();                            //Start i2c transfer
       i2c_write(addr | 0x01);                        //indicating a read operation

       for(j=0;j<num_bytes-1;++j){
             AA = 1;                                  //Set acknowledge bit
             buffer[j] = i2c_read();                  //Read data, save it in buffer
       }
       AA = 0;
       buffer[num_bytes-1] = i2c_read_and_stop();     //Read in last byte and stop,
save it in the buffer, end function
}
```

References

1. https://solarbotics.com/product/rt10k/
2. http://pdf1.alldatasheet.com/datasheet-pdf/view/28615/TI/SN754410.html
3. http://www.pololu.com/catalog/product/2275