University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

SplatBot
Final Report

By:
Stacey M Larese
December 5,2003

Table of Contents

Abstract……..….03
Executive Summary……..…..04
Introduction…….…….…..06
Integrated System……..….08
Mobile Platform……..…10
Actuation…….…….....11
Sensors…….....12
Behaviors…….17
Experimental Layout and Results…….18
Conclusion……20
Appendix……...21

Abstract

The SplatBot is an autonomous outdoor/indoor target seeking robot. Once activated, SplatBot roams around its environment until it moves through the targets field of view. Finding the target is easily accomplished through obstacle avoidance and direction finding SplatBot uses a modified paintball gun module to “shoot down” its target once the target is identified and in an appropriate firing range.

Executive Summary

SplatBot is an autonomous robot that seeks out it's target using obstacle avoidance. A target is equipped with PIR (passive infrared) motion sensors and light sensors to tell SplatBot it is in its area. (The target has approximately a 150 degree width and 20 feet depth area of vision)

The MCU on the robot is an Atmel Atmega128 and uses ports A,B,C,D,E and F. Port A is used as a communication bus for the LCD unit. Port B pin 7 is as an output compare pulse for the left motors. The output compare pulses are generated through pulse width modulation. Port C is used as I/O for the front bumpers. Port D's pins 0 and 1 are the SCL and SCA lines, respectively. These lines are the clock and data lines used with the Atmega128's TWI, two-wire interface, commonly known as I2C, bus. The TWI is a easy to use serial interface which enables a designer to connect up to 127 devices using just the two lines, SCL and SCA. SplatBot uses the TWI to connect two sonar units used for obstacle avoidance. Port E is used for another output compare signal for the right side motors. Port F's pins 0-3 are used for the wireless receiver units data bus.

The target uses an Atmel Atmega32 MCU for switching between different data that is sent over the wireless channel. Port's C and D are used as I/O ports. Port A uses its analog to digital converter to convert the light sensors analog voltage to a digital value. This data is used to differentiate between when there is light and no light present on the surface of the photo resistor array built on the target.

When testing SplatBot, results vary from outstanding to mediocre. Obstacle avoidance is about 90% effective. The reason for the 10% ineffectiveness is the height of the sonar units. They are mounted approximately 8 inches off the ground. This generates a bottom "blind plateau". Bump sensors were added to eliminate this problem, but the bump sensors are not very rigid. When SplatBot bumps into things, they bend in a manner that does not always trigger the bump sensors. They bend too much to add anything positive, but occasionally they work as they should. Rear bump sensors were purchased, but not installed because of there ineffectiveness.

The target works better than expected in doors. I originally purchased the PIR motion sensors with the thought that they worked with just motion of objects. I discovered that they actually detect a change in heat, specifically heat that is generated by human bodies. (Approximately temperatures around 90 – 105 degrees F.) This introduced a new problem. I needed a heat source on my robot to emit such temperatures. I purchased heat packs commonly used for skiing used a s hand/heat warmers and attached them to the front of SplatBot. This solved the motion sensor problem immediately. Once motion is sensed, SplatBot stops where it is. Light sensor data is now transmitted. Splatbot has a laser pointer on it to change the value of the light sensor

on the target. Once the laser pointer is pointing in the direction of the target, SplatBot fires its gun. This is why the entire system is not that effective outdoors. The sun causes many problems for the laser pointer, so a change in light detected on the photo resistors is not always detected.

Introduction

SplatBot is a military style robot but is used for entertainment purposes only. SplatBot is designed to search for a pre-determined target, and then once the target is found, SplatBot will release one of its rounds, a paintball in this case, at the target to symbolically shoot down an enemy piece of equipment. A target is defined as an array of motion sensors that will notify the robot when he has moved through its territory. The local environment is not always an ideal terrain, i.e. there most likely isn't a straight path from SplatBot to the target. Because of this, SplatBot must be able to navigate around objects to get to the target. This is accomplished through several sensors onboard.

For obstacle avoidance, SplatBot is equipped with two ultrasonic sonar sensors that will send signals back to SplatBot's "brain" to trigger routines to avoid the object. Two ultrasonic sensors are used for a wider field of view in front of the robot. Sonar is used for SplatBot's obstacle avoidance system because of its usefulness in an outdoor environment.

After successfully navigating through its terrain to a predetermined shooting location, SplatBot must now locate its target. It would be easy for a human to know where it is relative to the target because of our five senses. Unfortunately, robots are not equipped with any of these. As designers we must come as close as possible to recreate such environmental "antennas" and incorporate behaviors associated with stimuli encountered everyday tasks. I added a wireless system to do just that, tell the robot where it is , in some part, relative to the target. Once one of the motion sensors is triggered, the robot knows it is at least near the target. I then added a laser pointer on the robot, and photo resistors on the target, to tell the robot specifically where it was in relation to the target.

This report covers all the components of SplatBot in detail. Including its platform, movement and actuation, all sensors onboard, and the way SplatBot will successfully perform the desired task.

Integrated System

The robot consists of the following parts:

MCU : Alterra Atmega128 microprocessor

Platform:	Self Designed Tank Style in PROE
Motors:	7.2VDC Gear Head Motors
Wheels:	2.88 Inch “Foam” Wheels
Bump Sensors:	Contact Switches from Lynxmotion
Sonar Sensors:	Devantach SRF08’s
Digital Compass:	Images SI 1490 Compass (N/A)
Receiver:	Glolab RM1V
Stepper Motor:	Not Used
Paint Ball Gun:	Victor Spyder

Actuation is controlled by the MCU including movement of the robot and movement of the robot’s gun. The sonar units have PIC microcontrollers on board and so all ranging calculations are done within the unit. All data from the sonar units is sent over the TWI (two-wire interface) to the Atmega128, which is then used to control the movement of the robot.

The target consists of the following parts:

Motion Sensor:	All Electronics PIR sensor
Transmitter:	Glolab TM1V
LED Display:	Own Creation in PROTEL
Photo resistor Array:	Series of 9 1” dia. Photo resistors
MCU:	Atmega32

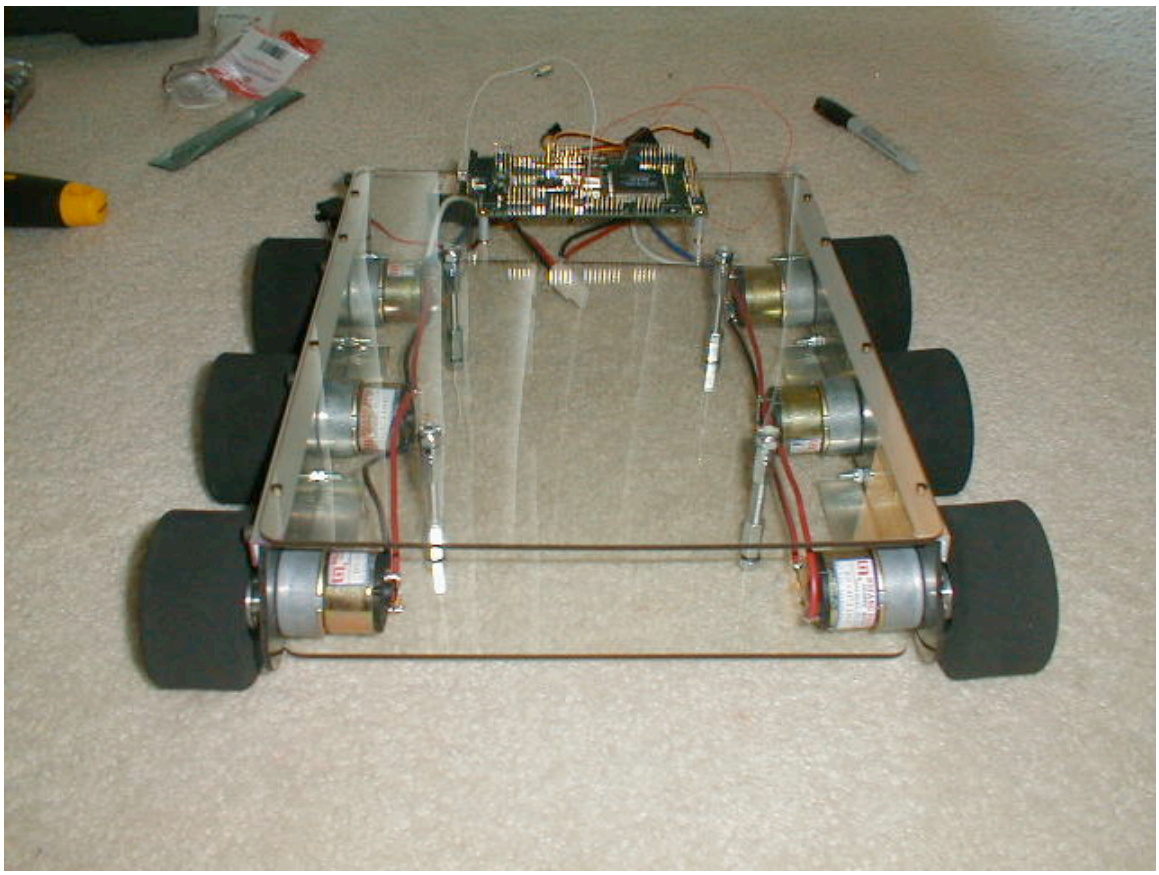
The motion sensor triggers a low true signal when something has moved through one of the sectors. The data is then sent through a wireless channel telling the robot it is in its “field of view”. The Atmega32 then waits for the motion sensors to reset. Following the PIR motion sensor reset, the MCU sends new data on the wireless channel. The first data is just a series of 0xFF’s to tell the robot to now “listen” for the photo resistor information, as compared to the motion sensor data. Once the initiation information transfer is finished being sent, the A/D sensor on

the MCU data is sent. This data is the voltage across the photo resistors. Finally, once this data has changed, because of the introduction of light into the photo resistor array, the robot is told to fire it's gun.

Mobile Platform

I have drawn the rails in CAD which give mechanical support for the wheels and motors. The following describes an idea of what the platform looks like. The rails will hold six wheels and motors, three

motors and wheels on the right and three motors and wheels on the left. I have designed the platform to be relatively wide for a stable system when SplatBot fires a round at its target. On top of the base will be a immovable turret. On top of the turret sits the paintball gun module. Rather then use a stepper motor, (because of costs), I have made the turret immovable. I will instead rotate the robot in a 360 degree pattern once it is located inside the targets range. Development of the platform is shown below.



Actuation

The motors to drive the wheels are 7.2VDC Gear Head Motors made by Hsiang Nenc. They are controlled by an SP-560 Motor Controller which uses PWM (Pulse Width Modulation) from the microcontroller to

set the direction and speed of the motor. Both parts were ordered from Lynxmotion, Inc. Pekin, IL.

A linear actuator used in power door locks is used to trigger the gun. An active relay is used to power the trigger as well as an extra 9.6 V battery. An active high signal sent from the Atmega128 is used as the input to the relay.

Gear Head Motor

Linear Actuator

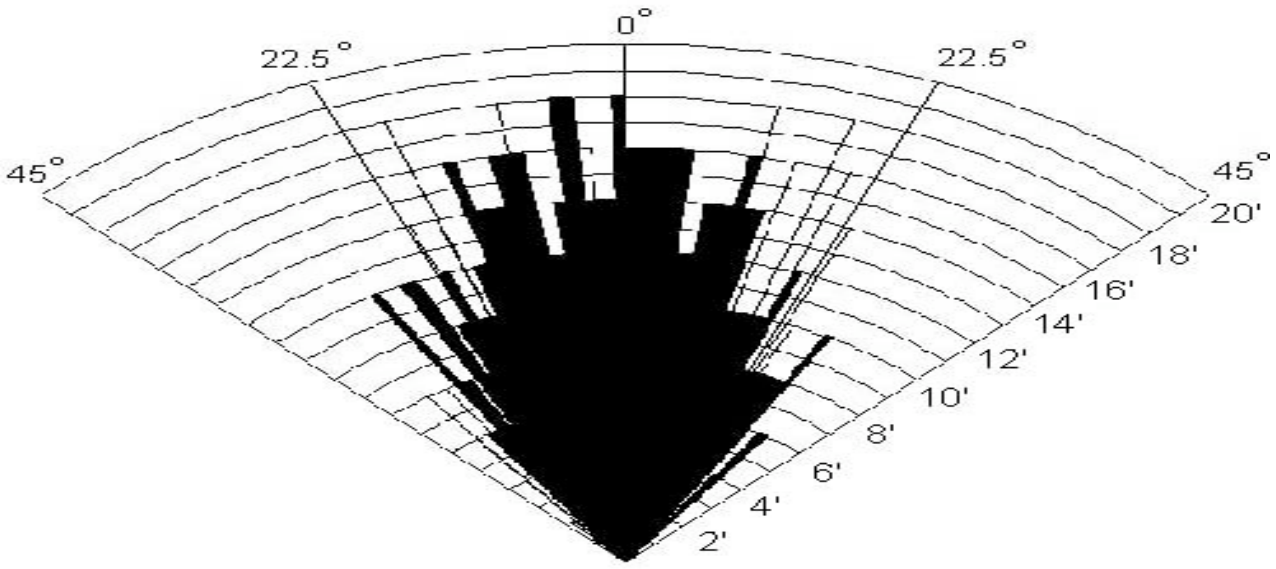
Sensors

Sensors are used to generate environment properties. Sensors are necessary to perform any type of interaction with SplatBots local “world”. The sensors that are used in SplatBot are ultrasonic rangers, RF modules, light sensors, bump sensors and motion sensors.

Sonar Sensors : Devantach SRF08's

The sonar sensors are used for obstacle avoidance. The Devantach SRF08's have the following specs:

Beam Pattern .	see graph
Voltage	5v
Current	15mA Typ. 3mA Standby
Frequency	40KHz
Maximum Range	6 m
Minimum Range	3 cm
Max Analogue Gain	Variable to 1025 in 32 steps
Connection	Standard IIC Bus
Light Sensor	Front facing light sensor
Timing	Fully timed echo, freeing host computer of task
Echo	Multiple echo - keeps looking after first echo
Units	Range reported n uS, mm or inches
Weight	0.4 oz
Size	43mm w x 20mm d x 17mm h



SRF08 Beam Width Graph

Bump Sensors: Bumper Switch Assembly Kit

The bump sensors are attached to the four corners of the robot. This gives added assurance that the robot will miss objects. In case of the switches being triggered, the robot will back up, if one of the front sensors are triggered, or move forward, if one of the rear bump switches is triggered.

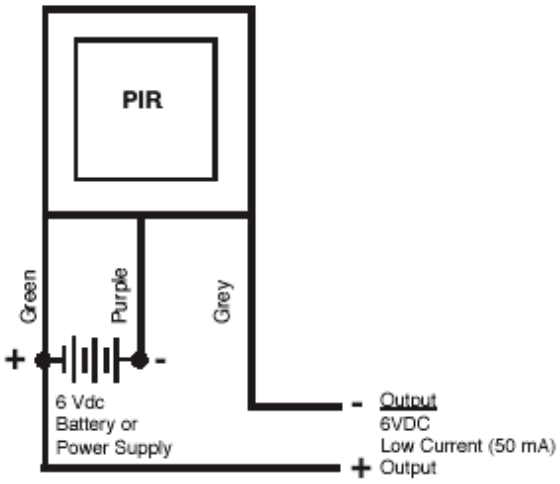
Poor Image of Bump Sensor Assembly Kit

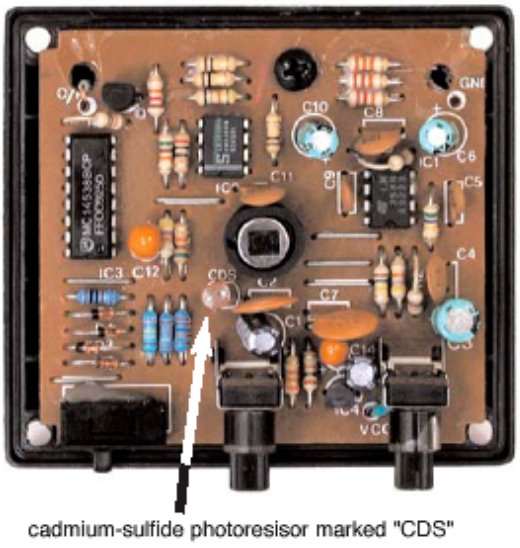
Motion Sensor: Infrared Detector, Removed from Hardware

All-Electronics description:

> "Infrared motion detectors are commonly used in alarm systems and motion activated switches. This unit was designed to activate a floodlight, after dark, when motion is present. It operates on 6Vdc and detects motion in a span of 120 degrees horizontally and 90 degrees vertically. When activated it outputs low current 6Vdc for 10, 30 or 60 seconds, depending on the switch setting. The unit was removed from a light fixture and has an adjustable ball-joint stem with wires protruding from the rear which may require alteration for mounting purposes. We supply a hook-up diagram and instructions for disabling the photo resistor so the unit can be used during daylight hours."

I will be using them during the day, so I did remove the CdS cells. This is the motion sensor schematic for hook up and removal of the CdS cell.





cadmium-sulfide photoresisor marked "CDS"

NOT USED BUT ASSEMBLED AND TESTED

Digital Compass: Images SI 1490 Digital Compass

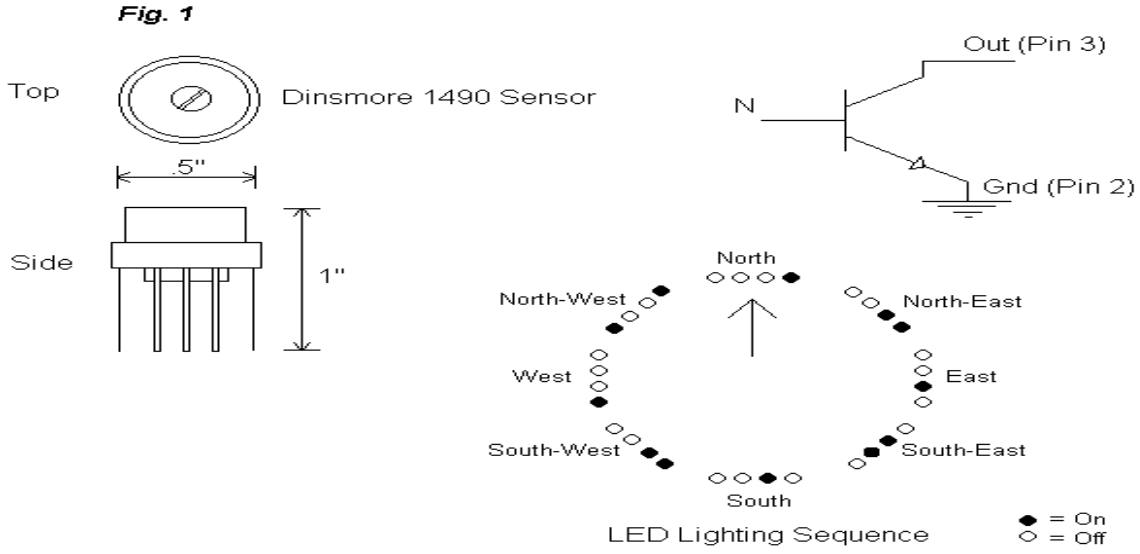
The compass is used to put something in common between the robot and the target. Without this piece of information, there is no way the robot would be able to "see" the robot.

Images description of the compass:

"The 1490 sensor is a solid state hall effect device. It is sensitive enough to detect the Earth's weak magnetic field. When rotated it can display the position of the four cardinal points on a compass, North (N), South (S), East (E) and West (W). As well as the intermediate directions: North East (NE), North West (NW), South East (SE), and South West (SW).

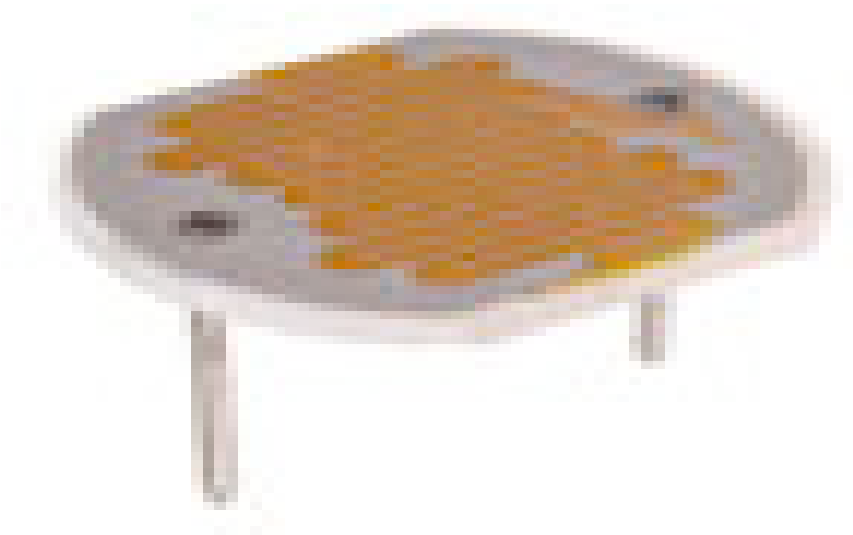
The sensor is dampened to approximate the speed of a liquid filled compass. It takes 2.5 seconds for it to respond to a 90 degree displacement. The dampening prevents over swinging the direction. In addition the built in hysterisis prevents flutter when near a switching direction.

The device is sensitive to tilt. Any tilt greater than 12 degrees will create directional errors.



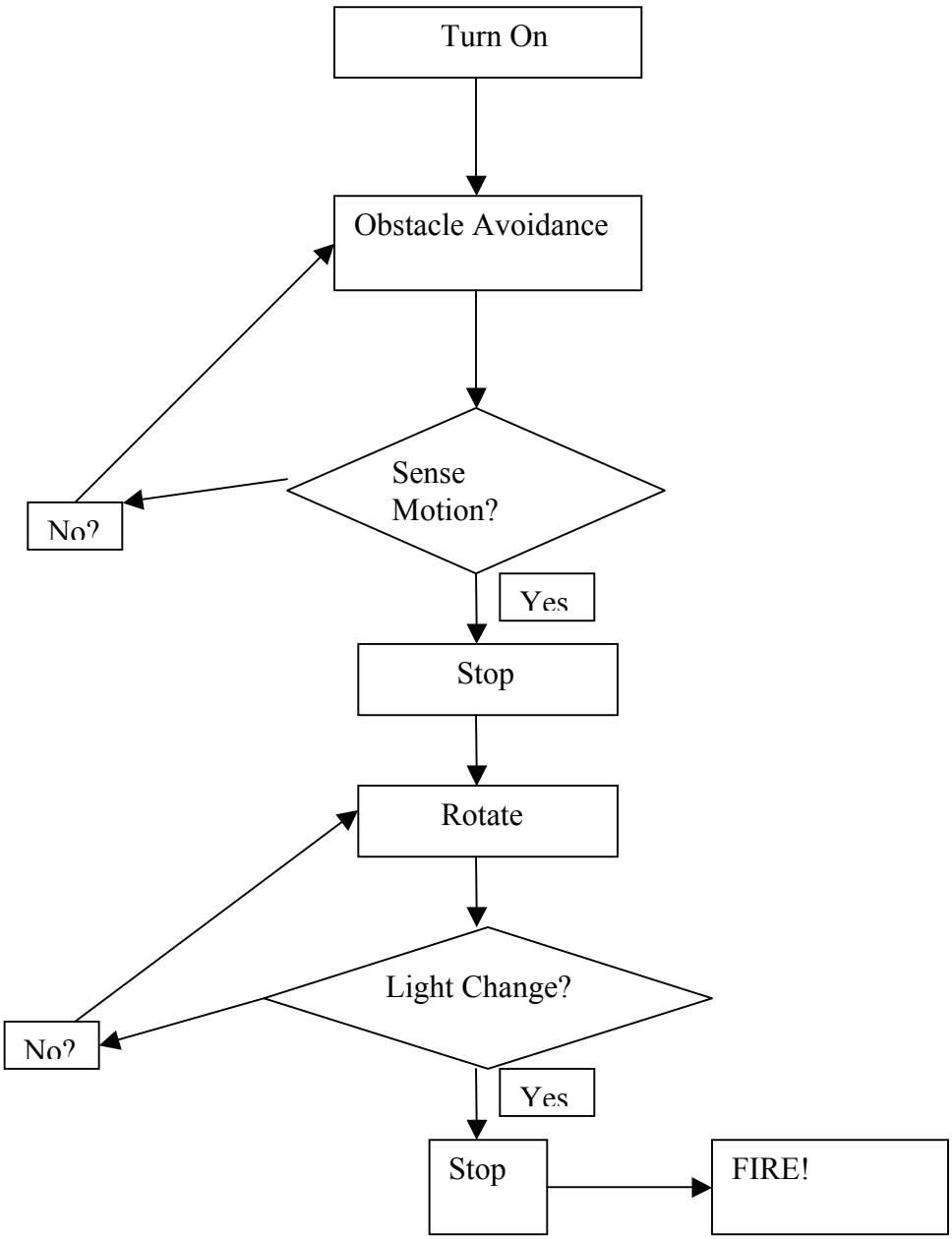
Light Sensors

The light sensors are actually 1”diamter photo resistors. Photo resistors are devices that change resistance with the amount of light present on the surface of them. Two extreme cases are absolute light, which will produce zero resistance, and the other being complete darkness which will produce infinite resistance. I have made an array of nine photo resistors on the target for them to detect when the laser pointer mounted on the robot is pointed at the target.



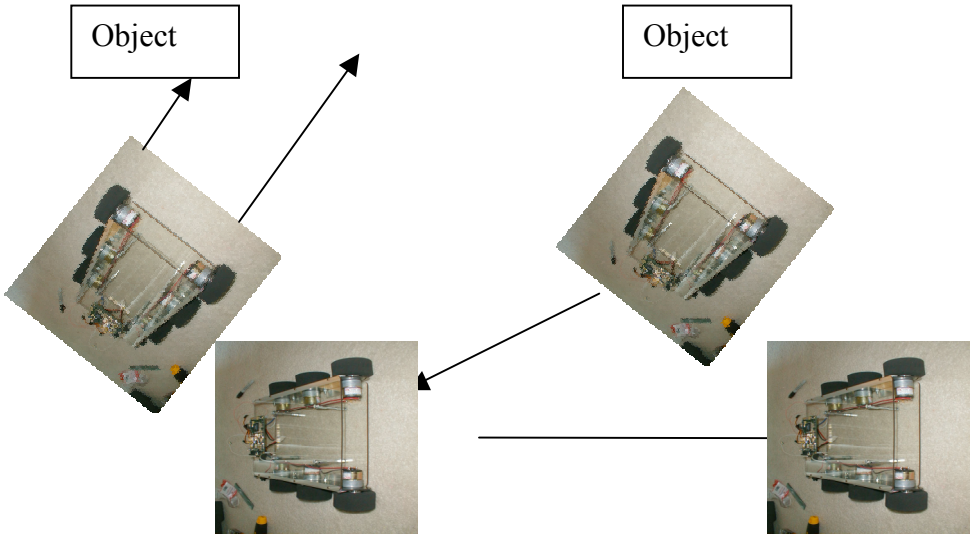
Behaviors

SplatBot will accomplish four behaviors using itself and its target. They are: obstacle avoidance, motion sensing, light sensing, and disbursement of paintballs. SplatBot follows this flow chart:



Obstacle Avoidance

SplatBot uses two sonar units and two front bump sensors for obstacle avoidance. I use two of each because this way I have the robot behave differently if something is closer, or something hits, on the left or right side of the robot. For example, if an object is closer on the right side, the robot will back up and turn to the left, and vice versa. If something is bumped on the left, it will back up to the right.



Motion Sensing

SplatBot can only tell if it is in the vicinity of the target with help from the target itself. The target is constantly sending information to the robot, telling it whether or not it has sensed its motion somewhere in front of the robot. If there is no change in information, SplatBot continues to avoid obstacles. If there is a change, Splatbot will stop where it is and wait for the data sent over the wireless channel to be light sensor information.

Light Sensing

As with motion sensing, light sensing is obtained only with the help of the target. Once motion sensing information has stopped, which is accomplished with the resetting of all motion sensors, the voltage across the photo resistors is sent wirelessly. Since my transmitter and receiver only send 4 bits of data, and I have 9 motion sensors hooked up, I added a 9 to 4 priority encoder to the transmitter unit. Once a change When photo resistor information is being sent, SplatBot makes small steps in a clockwise direction to change its direction with respect to the target, increasing the chance for the laser pointer on the robot to change the voltage across the photo resistors.

Pulling Trigger

Not really a complicated behavior, but neither are some human behaviors, but still a behavior. Once the light information is changed, robot is instructed to pull its trigger.

Conclusion

SplatBot by far has been my most exciting and rewarding project I have ever taken up. IMDL is the most practical class I have taken at the University of Florida. SplatBot consistently accomplishes its task in an indoor environment, but has trouble finishing its task outdoors. The reason? The ground is not smooth. The sonar units pick up small echos off the pavement when it is not smooth like a tile on the floor. This causes SplatBot to turn sometimes when there is nothing to turn away from. Motion sensing is accomplished 95% of the time whether indoor or outdoors. The only time it will miss is when SplatBot is moving in a direction away from the target with the heat packs not directly in front of the target. An easy solution would be to add heat packs to the backside of the robot. Light sensing works wonderful indoors and at night. (Yes, I tested at night) Outdoors during the day was not very possible to detect a change in light. (I had only been successful one time during daylight hours.) The only other problem I had was with the wireless system. Sometimes valid data was not presented to the receiver. The receiver must receive a series of three successful transmission for it to accept it as valid. SplatBot sometimes remains in its present motion, spinning, when it does not receive valid data.

When this semester is over, I plan on making the robot an RC car, this way I can control its motion, fire the gun when I want, and at obstacles I choose. I also want to add the stepper motor to the turret, the replacement came a day before presentations.

All in all, SplatBot was a success!

Appendices

ROBOT CODE

```
/* ******************************************************** */
/*                                                  INCLUDE HEADERS
          */
/* ******************************************************** */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/twi.h>
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <lcd.h>

/* ******************************************************** */
/*                                                  DEFINITION SECTION                */
/* ******************************************************** */

#define Gen_Broadcast       0x00
#define Sonar1_Addr         0xE0                                        /* Sonar 1 is Right Side Unit */
#define Sonar2_Addr                 0xE4                                        /* Sonar 2 is Left Side Unit
*/
#define Echo1_High                  0x02
#define Echo1_Low                   0x03
#define Result_Inch                 0x50
#define Result_Cent                 0x51
#define Result_Secs                 0x52
#define TWBR4                       0x10
#define TWBR1                       0x02
#define SONAR_COMMAND_REGISTER 0
#define READ                        1
#define WRITE                       0
#define ADDR_COMM_1                 0xA0
#define ADDR_COMM_2                 0xAA
#define ADDR_COMM_3                 0xA5
#define SYSCLK                      16000000UL
#define SONAR_MAX_ADDRESS 35                                    // Last byte available (36 bytes 0:35)
#define MAX_RANGE                   24                                      // Changes the maximum
range of the Sonar units to 1 meter
#define Analog_Gain                 10
#define Range_Reg                   0x02                                    // Sonar Range Register
#define Analog_Reg                  0x01

/* ******************************************************** */
/*                                                  GLOBAL VARIABLES
*/
/* ******************************************************** */

unsigned char i,j,k,l,Light, START,DATA1[18],DATA2[18],error[1],error1;
unsigned int Inch,Range,temp1,temp2;
char buffer[4];
unsigned char led, per, dutyh_1B, dutyl_1B,dutyh_3B, dutyl_3B;
unsigned char  old,bumper,receiver,receiver2,temp, temp7,sensed,stopped,
photoresistor_value,motion,counter;

SIGNAL(SIG_OUTPUT_COMPARE1A)
{
        sei();
}

SIGNAL(SIG_OUTPUT_COMPARE3A)
{
        sei();
```

```
}

INTERRUPT(SIG_OUTPUT_COMPARE3B)
{
        sei();
}
INTERRUPT(SIG_OUTPUT_COMPARE1B)
{
        sei();
        temp1 = DATA1[3];
        temp2 = DATA2[3];
        bumper = inp(PINC);
        receiver = inp(PINF);

        if ((receiver < 0x0F )&&(receiver != 0x00)&&(sensed==0x00))                                    //
Check to see if robot has entered PIR sensor area
                {
                        dutyh_1B = 0x02;
                                        // If in PIR sensor area stop robot
                        dutyl_1B = 0x43;
                        dutyh_3B = 0x02;
                        dutyl_3B = 0x43;
                        temp = 0x01;
                        sensed = 0x01;
                        goto OUT;
                }
        else if((receiver < 0x0F )&&(receiver != 0x00)&&(sensed==0x01))                 // Wait until PIR
sensors have reset
                        {
                                goto OUT;
                        }

        else if((receiver == 0x00)&&(sensed == 0x01))
        // Precaution in case data is lost wirelessly
                {
                        goto OUT;
                }

        else if (((receiver==0x0F )&&(sensed == 0x01)))
        // Target remains "off" after reset
                {

                if (temp7 == 0x00)
                                        // Get robot to rotate
                        {
                                dutyh_1B = 0x02;
                                dutyl_1B = 0x31;
                                dutyh_3B = 0x02;
                                dutyl_3B = 0x62;
                                temp7 = 0x01;
                                goto OUT;
                        }
                else
                        {
                                sensed = 0x02;
                                goto OUT;
                        }
                }

        else if ((receiver == 0x00)&&(sensed ==0x02))
        // Precaution in case data is lost wirelessly
                {
                        goto OUT;
                }
        else if ((receiver <= 0x0F)&&(receiver!=0x00)&&(sensed==0x02))                  // New light
sensor data is sent over wireless channel
                {
                        if(receiver == 0x0F)
                                {
```

```
                                        goto OUT;
                                }
                        else
                                                // Get initial A/D sensor data from
photoresistor array in target
                                {
                                        if (photoresistor_value == 0x00)
                                        {
                                                receiver2 = receiver;
                                                photoresistor_value = 0x01;
                                                goto OUT;
                                        }
                                        else if ( photoresistor_value == 0x01)
                // Routine that identifies when robot is pointing at target
                                                {
                                                        if((receiver == receiver2)&&(counter<=20))
                                                                {
                                                                        dutyh_1B = 0x02;
                                                                        dutyl_1B = 0x31;
                                                                        dutyh_3B = 0x02;
                                                                        dutyl_3B = 0x62;

                                                                        counter++;
                                                                        goto OUT;
                                                                }
                                                        else if((receiver == receiver2)&&(counter<=71))
                                                                {

                                                                        dutyh_1B = 0x02;
                                                                        dutyl_1B = 0x43;
                                                                        dutyh_3B = 0x02;
                                                                        dutyl_3B = 0x43;

                                                                        if (counter == 71)
                                                                                {
                                                                                        counter=
0;
                                                                                        goto
OUT;
                                                                                }
                                                                        else
                                                                        {
                                                                                counter++;
                                                                                goto OUT;
                                                                        }
                                                                }
                                                        else
                                                                {
                                                                        if (stopped == 0x00)
                                                                                {
                                                                                        stopped
= 0x01;

        dutyh_1B = 0x02;
                                                                                        dutyl_1B
= 0x43;

        dutyh_3B = 0x02;
                                                                                        dutyl_3B
= 0x43;
                                                                                        goto
OUT;
                                                                                }
                                                                        else
                                                                                {


        outp(0x01,PORTE);
                                                                                        temp7 =
0x00;
```

```
                                                                                temp =
0x00;
                                                                                sensed =
0x00;
                                                                                stopped
= 0x00;

        photoresistor_value = 0x00;
                                                                                motion =
0x00;
                                                                                goto
OUT;
                                                                                }
                                                                        }

                                                                }

                                                        }
                }

        else                                                            // Routine for when robot is not in PIR
sensor range
        {
        temp7 = 0x00;
        temp = 0x00;
        sensed = 0x00;
        stopped = 0x00;
        photoresistor_value = 0x00;
        outp(0x00,PORTE);

        if (bumper == 0)
                                                // Bump Sensor Routine
                {
                        dutyh_1B = 0x02;
                        dutyl_1B = 0x62;
                        dutyh_3B = 0x02;
                        dutyl_3B = 0x62;
                        goto OUT;
                }
        else if ( bumper == 1)
                {
                        dutyh_1B = 0x02;
                        dutyl_1B = 0x31;
                        dutyh_3B = 0x02;
                        dutyl_3B = 0x62;
                        goto OUT;
                }
        else if ( bumper == 2)
                {
                        dutyh_1B = 0x02;
                        dutyl_1B = 0x62;
                        dutyh_3B = 0x02;
                        dutyl_3B = 0x31;
                        goto OUT;
                }

        if ( old ==0 )
                                                        // Sonar unit Routine
        {
                if(( temp1 >= 20 ) || (temp1==0))
                        {
                                goto CHECK_SONAR_2;
                        }
                else if (temp1<20)
                        {
                                old = 1;
                                dutyh_1B = 0x02;
                                dutyl_1B = 0x62;
```

```
                                        dutyh_3B = 0x02;
                                        dutyl_3B = 0x62;
                                        goto OUT;
                                }
CHECK_SONAR_2:
                if (( temp2 >= 20)||(temp2==0))
                        {
                                dutyh_1B = 0x02;
                                dutyl_1B = 0x31;
                                dutyh_3B = 0x02;
                                dutyl_3B = 0x31;
                                goto OUT;
                        }
                else if (temp2<20)
                        {
                                old = 2;
                                dutyh_1B = 0x02;
                                dutyl_1B = 0x62;
                                dutyh_3B = 0x02;
                                dutyl_3B = 0x62;
                                goto OUT;
                        }
        }
        else if(old == 1)
        {
                if (temp1<=10)
                        {
                                dutyh_1B = 0x02;
                                dutyl_1B = 0x62;
                                dutyh_3B = 0x02;
                                dutyl_3B = 0x62;
                                goto OUT;
                        }
                else if (temp1>10)
                        {
                                dutyh_1B = 0x02;
                                dutyl_1B = 0x31;
                                dutyh_3B = 0x02;
                                dutyl_3B = 0x62;
                        if (temp1>=19)
                                old = 0;
                                goto OUT;
                        }
        }
        else if ((old==2))
        {
                if (temp2<=10)
                        {
                                dutyh_1B = 0x02;
                                dutyl_1B = 0x62;
                                dutyh_3B = 0x02;
                                dutyl_3B = 0x62;
                                goto OUT;
                        }
                else if (temp2>10)
                        {
                                dutyh_1B = 0x02;
                                dutyl_1B = 0x62;
                                dutyh_3B = 0x02;
                                dutyl_3B = 0x31;
                        if (temp2>=19)
                                old = 0;
                                goto OUT;
                        }
        }
        }
OUT:
        outp(dutyh_1B,OCR1BH);                  /* Set period of pulse with respect to OFF period */
```

```
	outp(dutyl_1B,OCR1BL);			/* 0x0052 for 1.5mS pulse 0x0040 for 1.0mS 0x0080
for 2.0 mS */
	outp(dutyh_3B,OCR3BH);			/* Shortest pulse puts robot in reverse, mid sized
pulse stops */
	outp(dutyl_3B,OCR3BL);			/* robot, and longest pulse puts robot in forward
*/


}


int main(void)
{
	old = 0;
	lcd_init(LCD_DISP_ON);
	temp = 0x00;
	sensed=0x00;
	motion= 0x00;
	counter = 0;
	photoresistor_value = 0x00;
	outp(0xFC,DDRC);
	outp(0xFF,DDRD);
	outp(0xFF,DDRE);
	outp(0xFF,DDRB);

	outp(0xF0,DDRF);

	outp(0x00,PORTE);

	TWBR = (SYSCLK / 100000UL - 16) / 2;			// Set Bit Rate to 100kHz
	TWCR =  _BV(TWEA) |_BV(TWEN) | _BV(TWINT);	//acknowledge, enable, clear IRQ
	TWAR = 0x01;                                                                    // Don't
care, won't be in Slave mode


		per = 0x00;		/* init count  */



		outp(0x18,TIMSK);			/* Enable Output Compare Register OCIE1A
and OCIE1B IN TIMSK */
		outp(0x18,ETIMSK);			/* Enable Output Compare Register OCIE3A
and OCIE3B IN ETIMSK */

		outp(0x02,OCR1AH);			/* Set compare register value 1A for 20mS
period */
		outp(0x71,OCR1AL);

		outp(0x02,OCR3AH);			/* Set compare register value 3A for 20mS
period */
		outp(0x71,OCR3AL);

		outp(0x02,OCR1BH);			/* Set period of pulse with respect to OFF
period */
		outp(0x43,OCR1BL);			/* 0x0052 for 1.5mS pulse 0x0040 for 1.0mS
0x0080 for 2.0 mS */

		outp(0x02,OCR3BH);			/* Set period of pulse with respect to OFF
period */
		outp(0x43,OCR3BL);			/* 0x0052 for 1.5mS pulse 0x0040 for 1.0mS
0x0080 for 2.0 mS */

		outp(0x31,TCCR1A);			/* Set to toggle OC1B pin on match and for
PFCPWM */
		outp(0x14,TCCR1B);	/* count with cpu clock/256 AND  PFCPWM*/

		outp(0x31,TCCR3A);			/* Set to toggle OC3B pin on match and for
PFCPWM */
		outp(0x14,TCCR3B);	/* count with cpu clock/256 AND  PFCPWM*/
```

```
                outp(per,TCNT1H);                       /* reset TCNT1 */
                outp(1,TCNT1L);

                outp(per,TCNT3H);                       /* reset TCNT1 */
                outp(1,TCNT3L);

        sei();

/* ************************************************************************** */
/*                                                      Change Maximum Range in Range Register
                                                */
/* ************************************************************************** */



loop_until_bit_is_clear(TWCR, TWSTO);                   //check no transmission in progress

        TWCR =  _BV(TWSTA) | _BV(TWEN);             //send start condition
        while(!(TWCR & _BV(TWINT))){;}          /* wait for transmission */

        if (TWSR!= TW_START)                                                    /*Check value of
TWI Status Register. Mask prescaler bits. If status different from START go to ERROR */
                {
                        lcd_home();                                                     /*
DEBUG MESSAGE FOR NOW */
                        lcd_puts("!START\n");
                        lcd_gotoxy(0,1);
                        lcd_putc(0xFF);
                         goto BEGIN;
                }



        TWDR = Sonar1_Addr | WRITE;                             //send sonar address
        TWCR = _BV(TWINT) | _BV(TWEN);                  /* clear interrupt to start
transmission */

        while(!(TWCR & _BV(TWINT))){;}                  /* wait for transmission */

        if(TWSR != TW_MT_SLA_ACK)
        {
                TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
                goto SONAR2;
                error[0] = TWSR;
                error1 = error[0];
                itoa(error1,buffer,10);
                for(;;)
                {
                        cli();
                        lcd_home();                                             /*
DEBUG MESSAGE FOR NOW */
                        lcd_puts("!Sonar1_Addr\n");
                        lcd_gotoxy(0,1);
                        lcd_puts(buffer);
                }
        }


        TWDR = Range_Reg;
                //send the address on the sonar to write to
        TWCR = _BV(TWINT) | _BV(TWEN);                                          //clear
IRQ, continue transmission
        while(!(TWCR & _BV(TWINT))){;}                                          //wait
for interrupt flag to get set
        if(TWSR != TW_MT_DATA_ACK)
        {
                TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
```

```
                        {
                                lcd_home();
                        /* DEBUG MESSAGE FOR NOW */
                                lcd_puts("!SONAR_COMMAND_REGISTER\n");
                                lcd_gotoxy(0,1);
                                lcd_putc(TW_MT_SLA_ACK);
                        }

        }

        /* ****************************************** */
        /*                      Change Max Range to 1 meter                     */
        /* ****************************************** */



        TWDR = MAX_RANGE;
                //send the sonar "start ranging" command
        TWCR = _BV(TWINT) | _BV(TWEN);
        while(!(TWCR & _BV(TWINT))){;}                                                  //wait
for interrupt flag to get set
        if(TWSR != TW_MT_DATA_ACK)
        {
                TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

                        {
                                lcd_home();
                        /* DEBUG MESSAGE FOR NOW */
                                lcd_puts("!Sonar2_Addr\n");
                                lcd_gotoxy(0,1);
                                lcd_putc(TW_MT_SLA_ACK);
                        }

        }

        TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN);                             //send stop
condition



loop_until_bit_is_clear(TWCR, TWSTO);                                                   //check
no transmission in progress

        TWCR =  _BV(TWSTA) | _BV(TWEN);
        //send start condition
        while(!(TWCR & _BV(TWINT))){;}                                                   /* wait
for transmission */

        if (TWSR!= TW_START)
        /*Check value of TWI Status Register. Mask prescaler bits. If status different from START go to
ERROR */
                {
                        lcd_home();
                        /* DEBUG MESSAGE FOR NOW */
                        lcd_puts("!START\n");
                        lcd_gotoxy(0,1);
                        lcd_putc(0xFF);
                         goto BEGIN;
                }



        TWDR = Sonar2_Addr | WRITE;
        //send sonar address
        TWCR = _BV(TWINT) | _BV(TWEN);                                                  /* clear
interrupt to start transmission */
```

```
	while(!(TWCR & _BV(TWINT))){;}                                  /* wait
for transmission */

	if(TWSR != TW_MT_SLA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
		goto SONAR2;
		error[0] = TWSR;
		error1 = error[0];
		itoa(error1,buffer,10);
		for(;;)
		{
			cli();
			lcd_home();
		/* DEBUG MESSAGE FOR NOW */
			lcd_puts("!Sonar1_Addr\n");
			lcd_gotoxy(0,1);
			lcd_puts(buffer);
		}
	}


	TWDR = Range_Reg;
	//send the address on the sonar to write to
	TWCR = _BV(TWINT) | _BV(TWEN);                                  //clear IRQ,
continue transmission
	while(!(TWCR & _BV(TWINT))){;}                                  //wait for
interrupt flag to get set
	if(TWSR != TW_MT_DATA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

			{
				lcd_home();
		/* DEBUG MESSAGE FOR NOW */
				lcd_puts("!SONAR_COMMAND_REGISTER\n");
				lcd_gotoxy(0,1);
				lcd_putc(TW_MT_SLA_ACK);
			}

	}

	/* ************************************** */
	/*                Change Max Range to 1 meter                */
	/* ************************************** */



	TWDR = MAX_RANGE;
	//send the sonar "start ranging" command
	TWCR = _BV(TWINT) | _BV(TWEN);
	while(!(TWCR & _BV(TWINT))){;}                                  //wait for
interrupt flag to get set
	if(TWSR != TW_MT_DATA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

			{
				lcd_home();
		/* DEBUG MESSAGE FOR NOW */
				lcd_puts("!Sonar2_Addr\n");
				lcd_gotoxy(0,1);
				lcd_putc(TW_MT_SLA_ACK);
			}

	}

	TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); //send stop condition
```

```
/* ************************************************************************************ */
/*                                        Change Maximum Analog Gain in Gain Register
                                    */
/* ************************************************************************************ */



loop_until_bit_is_clear(TWCR, TWSTO);                                                   //check
no transmission in progress

        TWCR =  _BV(TWSTA) | _BV(TWEN);
        //send start condition
        while(!(TWCR & _BV(TWINT))){;}                                                   /* wait
for transmission */

        if (TWSR!= TW_START)
        /*Check value of TWI Status Register. Mask prescaler bits. If status different from START go to
ERROR */
                {
                        lcd_home();
                        /* DEBUG MESSAGE FOR NOW */
                        lcd_puts("!START\n");
                        lcd_gotoxy(0,1);
                        lcd_putc(0xFF);
                         goto BEGIN;
                }



        TWDR = Sonar1_Addr | WRITE;
        //send sonar address
        TWCR = _BV(TWINT) | _BV(TWEN);                                                  /* clear
interrupt to start transmission */

        while(!(TWCR & _BV(TWINT))){;}                                                  /* wait
for transmission */

        if(TWSR != TW_MT_SLA_ACK)
        {
                TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
                goto SONAR2;
                error[0] = TWSR;
                error1 = error[0];
                itoa(error1,buffer,10);
                for(;;)
                {
                        cli();
                        lcd_home();
                        /* DEBUG MESSAGE FOR NOW */
                        lcd_puts("!Sonar1_Addr\n");
                        lcd_gotoxy(0,1);
                        lcd_puts(buffer);
                }
        }


        TWDR = Analog_Reg;
                //send the address on the sonar to write to
        TWCR = _BV(TWINT) | _BV(TWEN);                                                  //clear
IRQ, continue transmission
        while(!(TWCR & _BV(TWINT))){;}                                                  //wait
for interrupt flag to get set
        if(TWSR != TW_MT_DATA_ACK)
        {
                TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

                        {
```

```
                        lcd_home();
                /* DEBUG MESSAGE FOR NOW */
                        lcd_puts("!SONAR_COMMAND_REGISTER\n");
                        lcd_gotoxy(0,1);
                        lcd_putc(TW_MT_SLA_ACK);
                }

        }

        /* ************************************** */
        /*                Change Max Analog Gain          */
        /* ************************************** */



        TWDR = Analog_Gain;
        //send the sonar "start ranging" command
        TWCR = _BV(TWINT) | _BV(TWEN);
        while(!(TWCR & _BV(TWINT))){;}                                          //wait for
interrupt flag to get set
        if(TWSR != TW_MT_DATA_ACK)
        {
                TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

                        {
                                lcd_home();
                /* DEBUG MESSAGE FOR NOW */
                                lcd_puts("!Sonar2_Addr\n");
                                lcd_gotoxy(0,1);
                                lcd_putc(TW_MT_SLA_ACK);
                        }

        }

        TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN);                     //send stop condition


loop_until_bit_is_clear(TWCR, TWSTO);                                   //check no
transmission in progress

        TWCR =  _BV(TWSTA) | _BV(TWEN);                                         //send
start condition
        while(!(TWCR & _BV(TWINT))){;}                                  /* wait for
transmission */

        if (TWSR!= TW_START)
        /*Check value of TWI Status Register. Mask prescaler bits. If status different from START go to
ERROR */
                {
                        lcd_home();
                /* DEBUG MESSAGE FOR NOW */
                        lcd_puts("!START\n");
                        lcd_gotoxy(0,1);
                        lcd_putc(0xFF);
                         goto BEGIN;
                }



        TWDR = Sonar2_Addr | WRITE;                                             //send
sonar address
        TWCR = _BV(TWINT) | _BV(TWEN);                                  /* clear interrupt
to start transmission */

        while(!(TWCR & _BV(TWINT))){;}                                  /* wait for
transmission */

        if(TWSR != TW_MT_SLA_ACK)
```

```
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
		goto SONAR2;
		error[0] = TWSR;
		error1 = error[0];
		itoa(error1,buffer,10);
		for(;;)
		{
			cli();
			lcd_home();
	/* DEBUG MESSAGE FOR NOW */
			lcd_puts("!Sonar1_Addr\n");
			lcd_gotoxy(0,1);
			lcd_puts(buffer);
		}
	}


	TWDR = Analog_Reg;
	//send the address on the sonar to write to
	TWCR = _BV(TWINT) | _BV(TWEN);                                          //clear IRQ,
continue transmission
	while(!(TWCR & _BV(TWINT))){;}                                          //wait for
interrupt flag to get set
	if(TWSR != TW_MT_DATA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

			{
				lcd_home();
		/* DEBUG MESSAGE FOR NOW */
				lcd_puts("!SONAR_COMMAND_REGISTER\n");
				lcd_gotoxy(0,1);
				lcd_putc(TW_MT_SLA_ACK);
			}

	}

	/* ************************************** */
	/*                Change Max Analog Gain          */
	/* ************************************** */



	TWDR = Analog_Gain;                                                     //send the sonar "start
ranging" command
	TWCR = _BV(TWINT) | _BV(TWEN);
	while(!(TWCR & _BV(TWINT))){;}          //wait for interrupt flag to get set
	if(TWSR != TW_MT_DATA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

			{
				lcd_home();
	/* DEBUG MESSAGE FOR NOW */
				lcd_puts("!Sonar2_Addr\n");
				lcd_gotoxy(0,1);
				lcd_putc(TW_MT_SLA_ACK);
			}

	}

	TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); //send stop condition


/* ********************************************************************************** */
/*                                START RANGING HERE
                        */
/* ********************************************************************************** */
```

```
BEGIN:
	loop_until_bit_is_clear(TWCR, TWSTO);			//check no transmission in
progress

	TWCR =  _BV(TWSTA) | _BV(TWEN);		//send start condition
	while(!(TWCR & _BV(TWINT))){;}		/* wait for transmission */

	if (TWSR!= TW_START)							/*Check value of
TWI Status Register. Mask prescaler bits. If status different from START go to ERROR */
		{
			lcd_home();							/*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!START\n");
			lcd_gotoxy(0,1);
			lcd_putc(0xFF);
			 goto BEGIN;
		}



	TWDR = Sonar1_Addr | WRITE;					//send sonar address
	TWCR = _BV(TWINT) | _BV(TWEN);			/* clear interrupt to start
transmission */

	while(!(TWCR & _BV(TWINT))){;}			/* wait for transmission */

	if(TWSR != TW_MT_SLA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
		goto SONAR2;
		error[0] = TWSR;
		error1 = error[0];
		itoa(error1,buffer,10);
		for(;;)
		{
			cli();
			lcd_home();							/*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!Sonar1_Addr\n");
			lcd_gotoxy(0,1);
			lcd_puts(buffer);
		}
	}


	TWDR = SONAR_COMMAND_REGISTER; //send the address on the sonar to write to
	TWCR = _BV(TWINT) | _BV(TWEN); //clear IRQ, continue transmission
	while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
	if(TWSR != TW_MT_DATA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

			{
				lcd_home();
	/* DEBUG MESSAGE FOR NOW */
				lcd_puts("!SONAR_COMMAND_REGISTER\n");
				lcd_gotoxy(0,1);
				lcd_putc(TW_MT_SLA_ACK);
			}

	}

	/* ************************************** */
	/*			Command Result in Inches		*/
	/* ************************************** */
```

```
	TWDR = Result_Inch;                                    //send the sonar "start ranging" command
	TWCR = _BV(TWINT) | _BV(TWEN);
	while(!(TWCR & _BV(TWINT))){;}                  //wait for interrupt flag to get set
	if(TWSR != TW_MT_DATA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

			{
				lcd_home();
	/* DEBUG MESSAGE FOR NOW */
				lcd_puts("!Sonar2_Addr\n");
				lcd_gotoxy(0,1);
				lcd_putc(TW_MT_SLA_ACK);
			}

	}

	TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); //send stop condition


	for (i=0;i<=60;i++){                                   /* Wair for SRF08 to Finish Ping */
		for (j=0;j<=60;j++){
			for (k=0;k<=60;k++){}
		}
	}



	/* ********************************************************************** */
	/*                              READ                                      */
	/* ********************************************************************** */


	loop_until_bit_is_clear(TWCR, TWSTO);                  //check no transmission in
progress
BEGIN1:
	TWCR =  _BV(TWSTA) | _BV(TWEN);              //send start condition
	while(!(TWCR & _BV(TWINT))){;}          /* wait for transmission */

	if (TWSR!= TW_START)                                                    /*Check value of
TWI Status Register. Mask prescaler bits. If status different from START go to ERROR */
		{
			lcd_home();                                                     /*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!START\n");
			lcd_gotoxy(0,1);
			lcd_putc(0xFF);
			 goto BEGIN1;
		}



	TWDR = Sonar1_Addr | WRITE;                                    //send sonar address
	TWCR = _BV(TWINT) | _BV(TWEN);                         /* clear interrupt to start
transmission */

	while(!(TWCR & _BV(TWINT))){;}                         /* wait for transmission */
	if(TWSR != TW_MT_SLA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

		{
			lcd_home();                                                     /*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!BLAH\n");
			lcd_gotoxy(0,1);
			lcd_putc(TWCR);
		}
	}
```

```
	TWDR = SONAR_COMMAND_REGISTER; //send the address on the sonar to write to
	TWCR = _BV(TWINT) | _BV(TWEN); //clear IRQ, continue transmission
	while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
	if(TWSR != TW_MT_DATA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

			{
				lcd_home();
	/* DEBUG MESSAGE FOR NOW */
				lcd_puts("!SONAR_COMMAND_REGISTER\n");
				lcd_gotoxy(0,1);
				lcd_putc(TW_MT_SLA_ACK);
			}

	}


BEGIN2:
	TWCR =  _BV(TWINT)|_BV(TWSTA) | _BV(TWEN);	//send start condition
	while(!(TWCR & _BV(TWINT))){;}		/* wait for transmission */

	if (TWSR!= TW_REP_START)                                                    /*Check
value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR */
		{
			lcd_home();                                                 /*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!START\n");
			lcd_gotoxy(0,1);
			lcd_putc(0xFF);
			 goto BEGIN2;
		}

	TWDR = Sonar1_Addr | READ;                                          //send sonar address
	TWCR = _BV(TWEA)|_BV(TWINT) | _BV(TWEN);                                    /* clear interrupt
to start transmission */

	while(!(TWCR & _BV(TWINT))){;}                          /* wait for transmission */
	if(TWSR != TW_MR_SLA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

		{
			lcd_home();                                                 /*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!BLAH\n");
			lcd_gotoxy(0,1);
			lcd_putc(TWCR);
		}
	}

	for(i=0; i<8; i++) {
		TWCR = _BV(TWINT)|_BV(TWEA)|_BV(TWEN);
		loop_until_bit_is_set(TWCR, TWINT);
		if(TWSR != TW_MR_DATA_ACK)
		{
			TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

				{
					lcd_home();
	/* DEBUG MESSAGE FOR NOW */
					lcd_puts("!Sonar2_Addr\n");
					lcd_gotoxy(0,1);
					lcd_putc(TW_MT_SLA_ACK);
				}
		}
		DATA1[i] = TWDR;
```

```
	}

	TWCR = _BV(TWINT) | _BV(TWEN);

	while(!(TWCR & _BV(TWINT))){;}			//wait for interrupt flag to get set
		if(TWSR != TW_MR_DATA_NACK)
		{
			TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

				{
					lcd_home();
	/* DEBUG MESSAGE FOR NOW */
					lcd_puts("!data_last\n");
					lcd_gotoxy(0,1);
					lcd_putc(TW_MT_SLA_ACK);
				}
		}
		DATA1[9] = TWDR;



	TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); //send stop condition




/* ************************************************************************************** */
/*			Second Sonar Unit
			*/
/* ************************************************************************************** */


SONAR2:
	loop_until_bit_is_clear(TWCR, TWSTO);				//check no transmission in
progress

	TWCR =  _BV(TWSTA) | _BV(TWEN);		//send start condition
	while(!(TWCR & _BV(TWINT))){;}	/* wait for transmission */

	if (TWSR!= TW_START)							/*Check value of
TWI Status Register. Mask prescaler bits. If status different from START go to ERROR */
		{
			lcd_home();						/*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!START\n");
			lcd_gotoxy(0,1);
			lcd_putc(0xFF);
			 goto BEGIN;
		}



	TWDR = Sonar2_Addr | WRITE;					//send sonar address
	TWCR = _BV(TWINT) | _BV(TWEN);				/* clear interrupt to start
transmission */

	while(!(TWCR & _BV(TWINT))){;}			/* wait for transmission */

	if(TWSR != TW_MT_SLA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
		goto BEGIN;
		error[0] = TWSR;
		error1 = error[0];
		itoa(error1,buffer,10);
```

```
                for(;;)
                {
                        lcd_home();                                                     /*
DEBUG MESSAGE FOR NOW */
                        lcd_puts("!Sonar1_Addr\n");
                        lcd_gotoxy(0,1);
                        lcd_puts(buffer);
                }
        }


        TWDR = SONAR_COMMAND_REGISTER; //send the address on the sonar to write to
        TWCR = _BV(TWINT) | _BV(TWEN); //clear IRQ, continue transmission
        while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
        if(TWSR != TW_MT_DATA_ACK)
        {
                TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

                        {
                                lcd_home();
        /* DEBUG MESSAGE FOR NOW */
                                lcd_puts("!SONAR_COMMAND_REGISTER\n");
                                lcd_gotoxy(0,1);
                                lcd_putc(TW_MT_SLA_ACK);
                        }

        }

        /* ************************************** */
        /*                                Command Result in Inches                */
        /* ************************************** */



        TWDR = Result_Inch;                                     //send the sonar "start ranging" command
        TWCR = _BV(TWINT) | _BV(TWEN);
        while(!(TWCR & _BV(TWINT))){;}                  //wait for interrupt flag to get set
        if(TWSR != TW_MT_DATA_ACK)
        {
                TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

                        {
                                lcd_home();
        /* DEBUG MESSAGE FOR NOW */
                                lcd_puts("!Sonar2_Addr\n");
                                lcd_gotoxy(0,1);
                                lcd_putc(TW_MT_SLA_ACK);
                        }

        }

        TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); //send stop condition


        for (i=0;i<=60;i++){                                                    /* Wair for SRF08 to Finish Ping */
                for (j=0;j<=60;j++){
                        for (k=0;k<=60;k++){}
                }
        }



        /* ********************************************************************** */
        /*                                READ                                    */
        /* ********************************************************************** */


        loop_until_bit_is_clear(TWCR, TWSTO);                           //check no transmission in
progress
```

```
BEGIN11:
	TWCR =  _BV(TWSTA) | _BV(TWEN);		//send start condition
	while(!(TWCR & _BV(TWINT))){;}		/* wait for transmission */

	if (TWSR!= TW_START)                                                    /*Check value of
TWI Status Register. Mask prescaler bits. If status different from START go to ERROR */
		{
			lcd_home();                                                     /*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!START\n");
			lcd_gotoxy(0,1);
			lcd_putc(0xFF);
			 goto BEGIN11;
		}



	TWDR = Sonar2_Addr | WRITE;                                             //send sonar address
	TWCR = _BV(TWINT) | _BV(TWEN);                                  /* clear interrupt to start
transmission */

	while(!(TWCR & _BV(TWINT))){;}                          /* wait for transmission */
	if(TWSR != TW_MT_SLA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

		{
			lcd_home();                                                     /*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!BLAH\n");
			lcd_gotoxy(0,1);
			lcd_putc(TWCR);
		}
	}


	TWDR = SONAR_COMMAND_REGISTER; //send the address on the sonar to write to
	TWCR = _BV(TWINT) | _BV(TWEN); //clear IRQ, continue transmission
	while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
	if(TWSR != TW_MT_DATA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

			{
				lcd_home();
	/* DEBUG MESSAGE FOR NOW */
				lcd_puts("!SONAR_COMMAND_REGISTER\n");
				lcd_gotoxy(0,1);
				lcd_putc(TW_MT_SLA_ACK);
			}

	}


BEGIN22:
	TWCR =  _BV(TWINT)|_BV(TWSTA) | _BV(TWEN);	//send start condition
	while(!(TWCR & _BV(TWINT))){;}		/* wait for transmission */

	if (TWSR!= TW_REP_START)                                                /*Check
value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR */
		{
			lcd_home();                                                     /*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!START\n");
			lcd_gotoxy(0,1);
			lcd_putc(0xFF);
			 goto BEGIN22;
		}
```

```
	TWDR = Sonar2_Addr | READ;                                          //send sonar address
	TWCR = _BV(TWEA)|_BV(TWINT) | _BV(TWEN);                                  /* clear interrupt
to start transmission */

	while(!(TWCR & _BV(TWINT))){;}                          /* wait for transmission */
	if(TWSR != TW_MR_SLA_ACK)
	{
		TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

		{
			lcd_home();                                                                  /*
DEBUG MESSAGE FOR NOW */
			lcd_puts("!BLAH\n");
			lcd_gotoxy(0,1);
			lcd_putc(TWCR);
		}
	}

	for(i=0; i<8; i++) {
		TWCR = _BV(TWINT)|_BV(TWEA)|_BV(TWEN);
		loop_until_bit_is_set(TWCR, TWINT);
		if(TWSR != TW_MR_DATA_ACK)
		{
			TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

				{
					lcd_home();
	/* DEBUG MESSAGE FOR NOW */
					lcd_puts("!Sonar2_Addr\n");
					lcd_gotoxy(0,1);
					lcd_putc(TW_MT_SLA_ACK);
				}
		}
		DATA2[i] = TWDR;

	}

	TWCR = _BV(TWINT) | _BV(TWEN);

	while(!(TWCR & _BV(TWINT))){;}                  //wait for interrupt flag to get set
		if(TWSR != TW_MR_DATA_NACK)
		{
			TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);

				{
					lcd_home();
	/* DEBUG MESSAGE FOR NOW */
					lcd_puts("!data_last\n");
					lcd_gotoxy(0,1);
					lcd_putc(TW_MT_SLA_ACK);
				}
		}
		DATA2[9] = TWDR;



	TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); //send stop condition



				lcd_clrscr();
				Range = DATA1[3];
				lcd_gotoxy(0,0);
				itoa(Range,buffer,10);
				lcd_puts("Range_1: ");
				lcd_gotoxy(9,0);
				lcd_puts(buffer);
```

```
                                        Range = DATA2[3];
                                        lcd_gotoxy(0,1);
                                        itoa(Range,buffer,10);
                                        lcd_puts("Range_2: ");
                                        lcd_gotoxy(9,1);
                                        lcd_puts(buffer);



goto BEGIN;

}
```

TARGET CODE

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
unsigned char adchigh,pirsensor,photovalue,temp,i,j,k,l,m,n ;

void init_ADC(void)
{
ADMUX = 0x00; // start by selecting channel 0
ADCSRA = 0x00; // select the ADC clock frequency
ADMUX  |= (1<<ADLAR);
ADCSRA |= (1<<ADPS2);
ADCSRA |= (1<<ADPS1);
ADCSRA |= (1<<ADPS0);
ADCSRA |= (1<<ADATE);
ADCSRA |= (1<<ADEN);
ADCSRA |= (1<<ADSC);


SFIOR |= (0<<ADTS2);
SFIOR |= (0<<ADTS1);
SFIOR |= (0<<ADTS0);


}
int main(void)
{
        outp(0x00,DDRD);
        outp(0xFF,DDRB);
        outp(0xFF,DDRC);

        init_ADC();
```

```
while(1)
        {
         pirsensor = inp(PIND);
         outp(pirsensor,PORTC);

         if (pirsensor !=0xFF )
                {
                        while(pirsensor!=0xFF)
                        {
                                pirsensor = inp(PIND);
                                outp(pirsensor,PORTC);
                        }
                        outp(0xFF,PORTC);
                        for (i=0;i<=5;i++){                                     /*
Wair for SRF08 to Finish Ping */
                        for (j=0;j<=60;j++){
                        for (k=0;k<=60;k++){
                        for (l=0;l<=60;l++){

                        }
                        }
                        }
                        }
                        while (1)
                        {
                                outp(ADCH,PORTC);

                        }

                }
        }


        return 0;
}
```