Final Report

The Sentinel

Aleksandras Kaknevicius

Intelligent Machines Design Laboratory
EEL 4665 / EEL 5666
Dr. Eric M. Schwartz & Dr. A. Antonio Arroyo
Tim Martin & Ryan Stevens

Table of Contents

I. Abstract

The purpose of this project is to create an autonomous robot which can successfully safeguard a doorway or hallway, blocking intruders or alerting others when an intruder has tried to pass it. This robot, called the Sentinel, will sit in a hallway or doorway and wait for an approaching heat source. If the heat source is friendly, they can produce a sequence of four audible frequencies to disable the Sentinel and pass by. If the heat source is not friendly, they will not have the four note key, and the Sentinel will warn the target to stay back. Failure to comply results in the Sentinel blaring an alarm, and if the target continues to approach, the Sentinel will run away, attempting to keep the alarm running for as long as possible. Although the Sentinel can detect and alert, it does not take an active role in subduing the target.

II. Executive Summary

When enlisting in the Machine Intelligent Design Lab, the goal was to create a robot which could be used for security purposes. This evolved into the idea of safeguarding a certain room or valuable item, and so the Sentinel was born. The Sentinel is a robot which can effectively guard a hallway, detecting a person who is attempting to enter and determining whether or not the person is allowed to pass. An approved person is able to produce four audible frequencies in the correct order, and if the person tries to pass by the robot without them, or gets too close to the robot (in order to disable it), the robot drives away and alerts everyone in the vicinity with an alarm.

The initial design for detection was to use pyroelectric sensors to find and track the target once it had shown up. The Sentinel would read its PIR sensors, and the one which was active would tell the robot which direction to move in order to block the heat source. However, upon several attempts (spanning the course of several weeks), this task appeared to be near impossible. The PIR sensors were typically not very responsive, and several times the wrong one would trigger for seemingly no reason.

The altered design for detection only used the PIR sensors when they were most useful- for initial detection. After the Sentinel reads that someone has entered, it sweeps back and forth, using its distance sensor to detect any foreign objects. Once something is detected, it stops, and continues to read how close the person is. If the person leaves the Sentinel's field of view, the Sentinel cannot tell whether or not the person went left or right, so it goes to the right. This is because if the person goes to the right, then the Sentinel can track them, but if the person goes to the left, to step around the robot, the Sentinel can use its rangefinders to see whether or not someone has attempted to pass by it.

The frequency detection is done using a preamplifier and microprocessor code written to sample the audio at a certain rate and detect the crests of the signal. A recorder is used to output the required frequencies because it can produce moderately clean waveforms at a decent distance. The obstacle avoidance is done using two wide beam sonar rangefinders, which reads when an object is quickly approaching. If the difference between the two sonar readings is small, then the robot can tell that it is headed right for it and should turn around quickly. If the difference between the two sonar readings is large, then the robot knows it does not have to turn as quickly to avoid the obstacle. The alarm which the robot uses is simply a piezo buzzer which is triggered by the microprocessor.

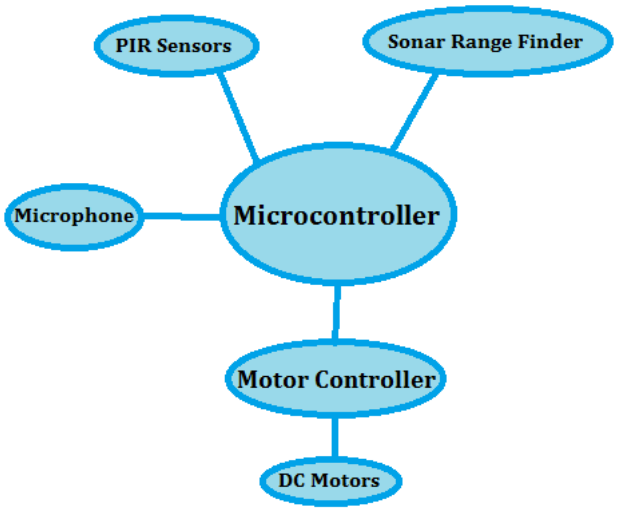
III. Introduction

According to the FBI, a burglary occurs within the United States every 15 seconds. Alarm systems can be costly, and a house without an alarm system is several times more likely to be broken into. This is where the Sentinel comes in. The Sentinel is made to detect a person who is not supposed to be within the vicinity, and if the person does not leave, then the Sentinel will produce an alarm to alert those who can hear it. This can be especially valuable for instances of home invasion, as the robot will alert the unaware occupant who is inside the home.

This report details the way the Sentinel works, and shows how it uses several sensors to make an effective detector for intruders who attempt to silently sneak into the house.

IV. Integrated System

The Sentinel is run by Tim Martin's Epiphany DIY board. All sensors are controlled by the corresponding microcontroller, including the microphone, which has the microcontroller check for a certain frequency by performing crest detection. The DC motors are controlled through the motor controller which was purchased with the Epiphany board.



V. Mobile Platform

The platform of the Sentinel is the typical circular design which accommodates a two main wheel robot. The PIR sensors and sonar sensor were mounted on the side of the robot next to the main right wheel. The purpose of this placement is to allow the Sentinel to move back and forth in a straight line to block the heat source's approach. The robot also has a lower secondary level which houses the main board, battery pack, and all other sensors (aside from the microphone). The platform was designed using Solidworks, and was milled out in the IMDL lab.

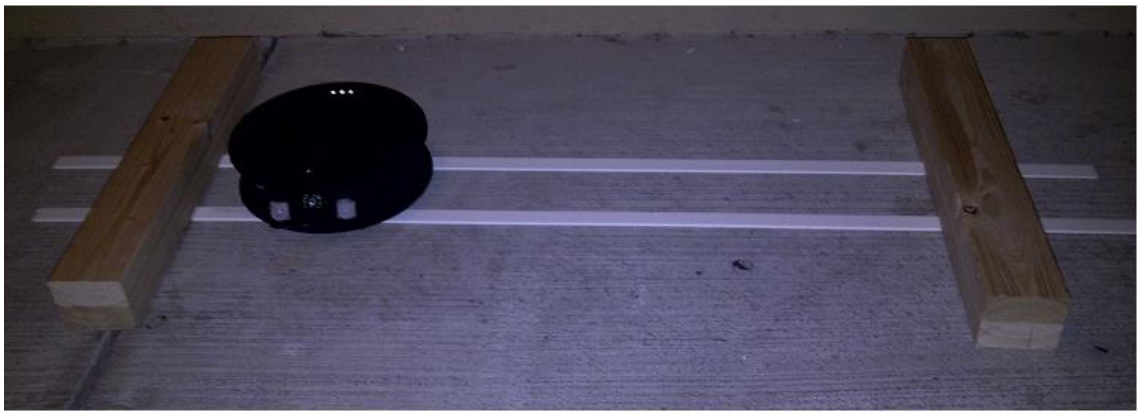
VI. Actuation

Because the robot mainly relies on its sensors for its functions, there is not a lot of complex actuation. Two DC motors drive the robot's wheels, which provide the robot with the ability to move perpendicular to the incoming heat source. DC motors were chosen because servos would not be able to move the platform fast enough to keep up with a human target. The average walking speed of a human is approximately 4mph, and to find the torque required, the equation

$$Torque = \frac{Robot\ Weight * Coefficient\ of\ Friction * Radius\ of\ Wheel}{Number\ of\ Motors}$$

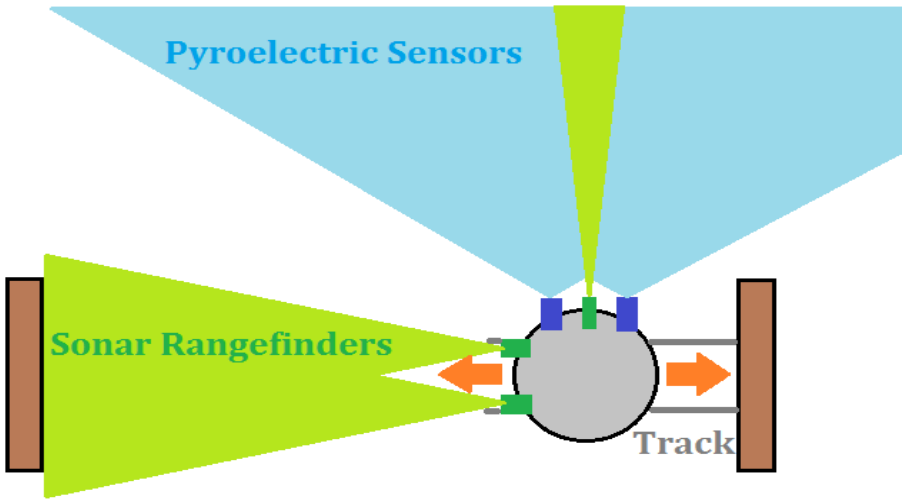
was utilized. 70mm wheels were chosen, and the weight of the robot was overestimated to be 5lbs. With these values, the torque required for the robot was 16.8 oz*in. The motors obtained were the 50:1 Micro Metal Gearmotors from Pololu. These not only provided a torque of 20 oz*in, but gave a speed of 625rpm, resulting in a max robot speed of 5.11mph.

The robot has a lot of back and forth motion, and depending on the environment it is in, it can have a tendency to drive in a curved line. To fix this, a model hallway was built with tracks to keep the robot driving perpendicular to the hallway wall.



The model hallway comprised of four pieces of 2x4 wood to make the walls and two thin slats of wood to make the tracks. The walls were chiseled on the bottom so that the tracks could slip into place and stay straight.

VII. Sensors



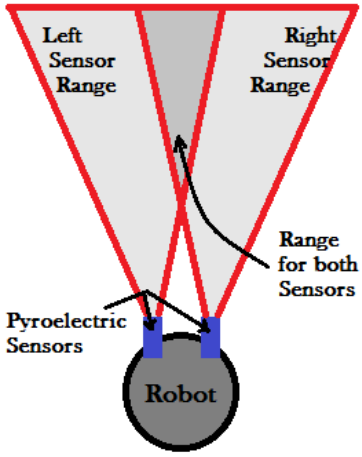
Sonar Range Finder

A Maxbotix LV-MaxSonar-EZ4 Sonar Range Finder is put between the two pyroelectric sensors so that once the robot lines up with its target, the robot will know how far away the target is. The reading from the sonar range finder is what instructs the robot to perform various actions as the heat source moves closer. After the rangefinder is given power, the analog pin is connected to the ADC, and the digital value obtained directly corresponds to the distance detected by the sensor.

PIR Sensors

The PIR sensors used to detect the presence of heat sources in front of the Sentinel are the SE-10 PIR sensors from pololu.com. By putting two of them on the robot, the robot can roughly track the location of the incoming heat source. The initial plan was to utilize both PIRs to track the incoming person and get in front of them, but these sensors proved to be unreliable for this function. These sensors were then used only for initial detection.

The data online for the PIR sensors said that the PIR sensors would produce a low voltage when movement was detected and a high voltage while no movement was detected. However, after testing the PIR sensor several times, this did not seem to be the case. The PIR sensor toggles when heat movement is detected, which can result in either a high or low voltage corresponding to movement. Also, the response time after the initial detection can be slow.



Microphone

The microphone is used to detect the frequency of the audio signal which allows the target to pass. The signal is filtered and amplified, and then sent to the microprocessor where its frequency can be identified using crest

detection. If the frequencies roughly match, then the robot will move to the side and allow the target to pass.

The amplification of the microphone output is done using the preamplifier circuit shown in Figure 1 of the Appendices. The preamplifier was designed to have a gain of about 60dB, or 1000V/V. To obtain this value, a few simple calculations had to be made. Most microphones produce an output of 1V for every Pascal detected. Conversational speech at one meter away produces a sound pressure of 0.02Pa. However, the robot should be able to detect this level of noise from about 3 meters away, and because the pressure of sound is inversely related to the square of the distance away, increasing the distance by a factor of 3 decreases the sound pressure by a factor of 9. This means that the sound should only produce 0.002444Pa at 3 meters away. To get a readable frequency, the circuit was designed to get a 2V output, which corresponds to 2Pa. To get to 2Pa, 0.002444Pa had to be amplified by a factor of 818.18V/V. 1000V/V, or 60dB, was chosen to compensate for miscellaneous losses.

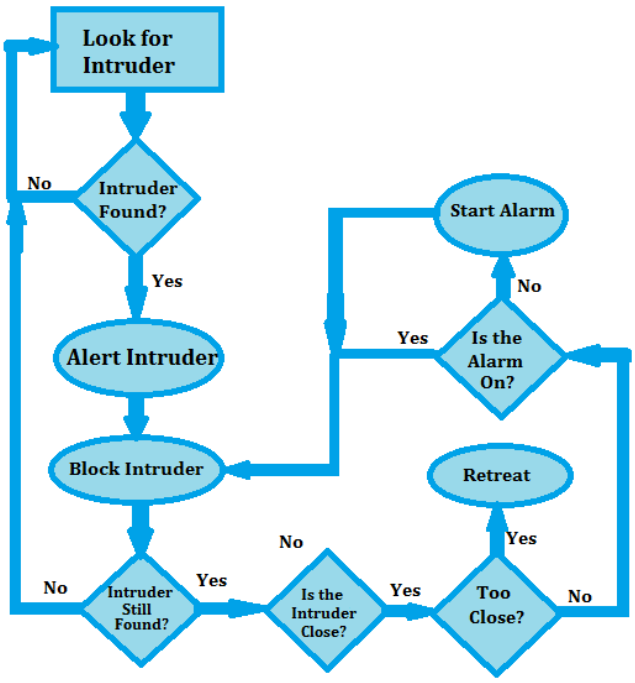
The code for the crest detection can be found in the Appendices, but it essentially samples the signal at 2600 times per second. At every sample, the microprocessor checks whether or not the signal has hit a maximum value. If a maximum value is detected, then the microprocessor increments a counter variable. After 200 samples, the counter variable is multiplied by 13, and the resulting number is the detected frequency.

Sonar Range Finders and Bump Sensors

The sonar rangefinders and bump sensors are added onto the robot to provide feedback for obstacle avoidance. When the heat source gets close enough to the robot, the robot flees. As it does, it uses obstacle avoidance to avoid objects and the target (which is rapidly approaching). The Maxbotix LV-MaxSonar-EZ0s were chosen as the sonar range finders because of their wide beam width, which aids in detecting obstacles. To implement them, they were given voltage and ground, and the output from the analog pin was inputted into the ADC. Because the robot did not seem to crash often, the bump switches, which were initially implemented, were removed to free up more ports. The finalized obstacle avoidance code can be found in the Appendices.

The sonar rangefinders have a secondary function as well. When the robot is first turned on, the robot is put in the model hallway so that the rangefinders can read the total width of the hallway. When the robot moves back and forth, it uses the sensors to figure out where in the hallway it currently is, preventing it from running into the hallway walls. Also, a sudden change in the sensor readings lets the robot know when someone is attempting to pass by on the left side.

VIII. Behaviors



Detection

When the Sentinel is in detection mode, it is looking for a heat source target, and the blue LED is on. It moves back and forth to allow the PIR sensors to detect any heat source, and if they do, it starts the target acquired behavior.

Target Acquired

At this point, the heat source has been detected by the PIR sensors, and the blue LED starts blinking. The robot sweeps to the left side of the hallway, and then back to the right end. All the while, the robot is checking the sonar range finder to see how close the target is, and if a target is detected, then the red LED turns on. Once the target is moderately close, the robot will let out an alarm, which will continue unless the target moves out of range. If the target can no longer be detected by the PIRs or sonar range finder, then the robot returns to its detection behavior. But, if the target comes too close to the Sentinel, or attempts to step around the Sentinel, then the retreat behavior is initialized. While the robot detects someone who is not too close, it is listening for the required audible frequencies. If these

frequencies are detected, then the robot moves out of the way, and when the person passes by, the robot resumes its detection behavior. This is signified by the white LED turning on.

Retreat

The Sentinel is too small to physically block its target, so it turns away and starts driving, still sounding its alarm. During this phase, obstacle avoidance is implemented, and the robot will run away from the target, evading anything which blocks its path. During all of this, all three LEDs are blinking.

IX. Experimental Layout and Results

Microphone

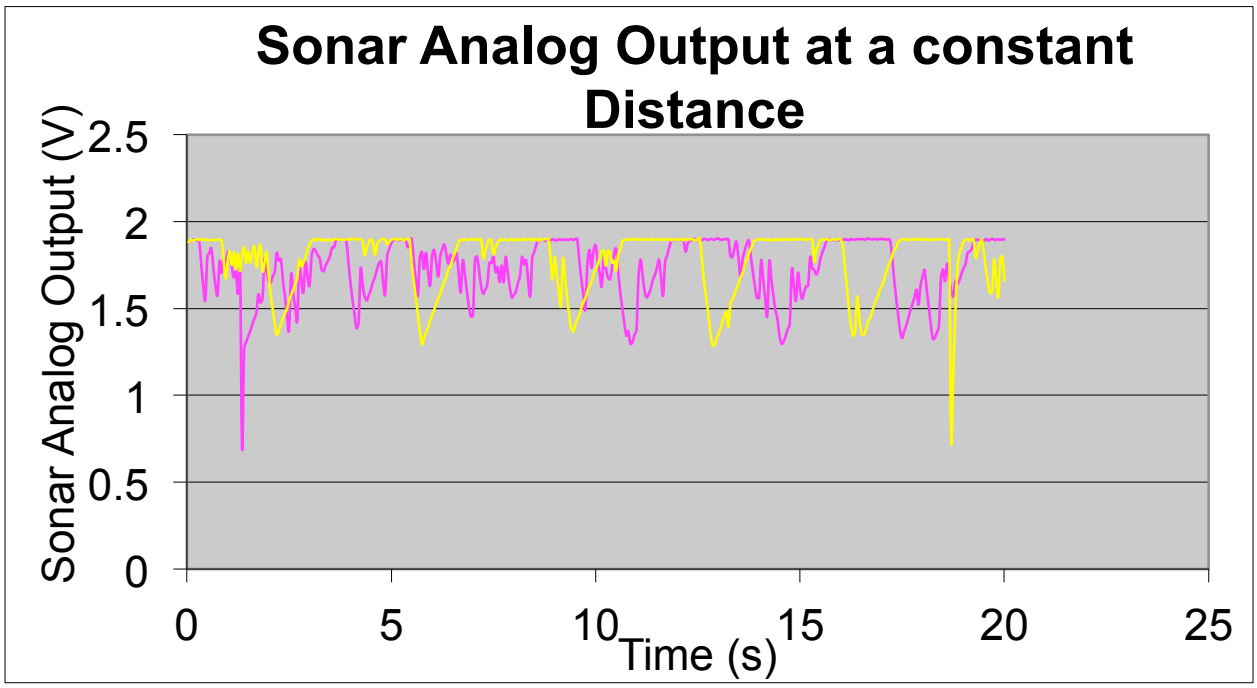
The microphone preamplifier was first simulated using LTSpice, and the design and simulation results can both be found in the Appendices (Figures 1 and 2, respectively). The design showed a gain of 60dB for all desired frequencies, which was the initial goal. The circuit was then physically built on a breadboard and the output was viewed through an oscilloscope. At about 10 feet away, the circuit could pick up a 1kHz frequency and produce a sinusoidal output of about 400mV peak to peak. The oscilloscope itself could also detect that the audio which was received had a frequency of 1kHz.

The frequency detection code proved to be reliable, especially at higher frequencies. Due to the sampling by the microprocessor, the highest frequency which could be detected was 1300Hz, but the four note recorder tune did not exceed this. When the microphone was detecting ambient noise, the frequency read tended to be around 600Hz, which did not trigger any false positives for the four frequency key.

Sonar Range Finders

All rangefinders were tested by outputting the digital voltage value of each sensor to the LCD. The Maxbotix LV-MaxSonar-EZ4 Sonar Range Finder outputted digital equivalent voltages of approximately 150-1800, 150 being extremely close to the sensor, and 1800 being the farthest possible object detection. The Maxbotix LV-MaxSonar-EZ0s on the other hand only produced digital values from approximately 210-540, where 210 is the closest range and 540 is the farthest range. Although the Maxbotix LV-MaxSonar-EZ0 does not seem to detect as far (or have as strong a resolution), it is only used for obstacle avoidance and hallway position awareness, which primarily deals with the closer ranges of the robot.

These sensors also seemed to spike false positives. When they were detecting far range objects, it would randomly spike a low distance, as shown below.



A coupling capacitor was used in an attempt to remedy this problem, but did not prove effective. An algorithm was used to make sure that the sensors actually read a closer/farther value by reading several consecutive values in the detected range to make sure that a distance change was actually happening.

X. Conclusion

Overall, the robot was a moderate success. The obstacle avoidance code was very effective, even at higher driving speeds, and the frequency detection was able to read the four frequencies at a decent distance almost every time. The only part of the robot which could have used reasonable improvement was the tracking capability of the robot. The performance of the PIR sensors was disappointing, even after supplying them with 9V for better responsiveness.

If the robot were to be built again, a few things would be done differently. PIR sensors would not be used, and another sensor would be used to detect an incoming person. Temperature sensors may have been more reliable, or maybe a camera with blob detection. Either way, it would seem more plausible to actively keep the robot in front of someone trying to enter the hallway. Also, the 5V servo regulator would be bought for the Epiphany DIY board. Servos were not necessarily required, but more voltage could be supplied to the sensors, making them more accurate and responsive. The 5V regulator also

prevents the 3.3V jumper from being soldered, which opens up 4 more ADC channels. If the negative effects of the absence of the 5V regulator had been more apparent at the beginning of the process, the 5V regulator would probably have been purchased.

As a robot sentry, the Sentinel performed well, although its responsiveness was slower than desired. Powering some of the sensors with higher voltages or getting a faster microprocessor may have made the robot more of a viable detection/security system.

XI. Documentation

[1] Atmel, "Atmel ATxmega64A1 Microprocessor", datasheet, 2009, http://www.atmel.com/dyn/resources/prod_documents/doc8077.pdf

[2] Linear Technology, "LT1632/1633 Dual/Quad Precision Op Amps", datasheet, http://www.datasheetcatalog.org/datasheet/lineartechnology/16323fs.pdf

[3] Maxbotix ,"Maxbotix LV-MaxSonar-EZ0 Sonar Range Finder MB1000", datasheet, 2007, http://www.pololu.com/file/0J68/LV-MaxSonar-EZ0-Datasheet.pdf

[4] Maxbotix, "Maxbotix LV-MaxSonar-EZ4 Sonar Range Finder MB1040", datasheet, 2007, http://www.pololu.com/file/0J149/LV-MaxSonar-EZ4-Datasheet.pdf

[5] Pololu, "Passive Infrared (PIR) Detector SE-10", datasheet, http://www.pololu.com/file/0J250/SE-10.pdf

[6] Pololu, Specifications for 50:1 Micro Metal Gearmotor HP, http://www.pololu.com/catalog/product/998/specs

XII. Appendices

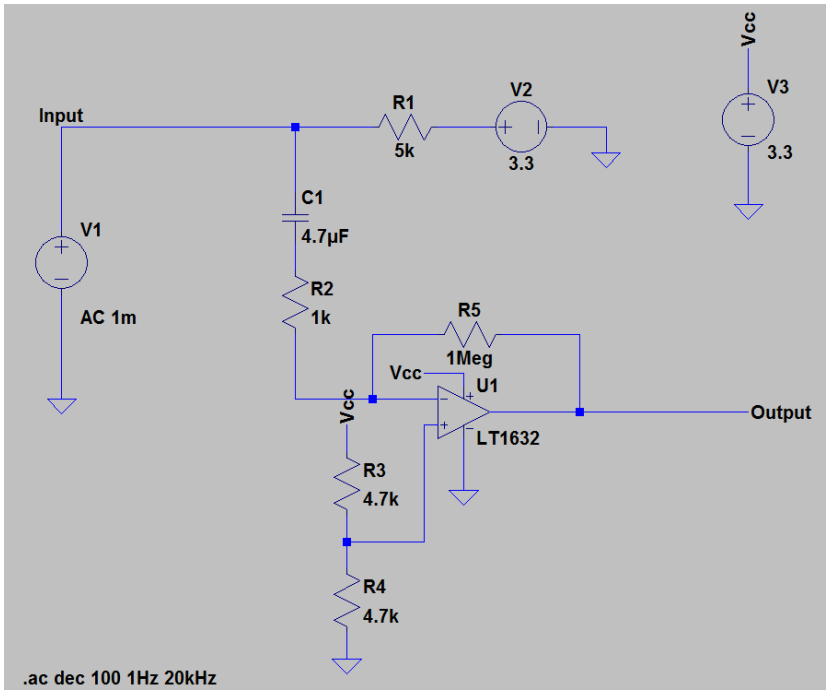


Figure 1 - The Preamplifier Circuit design for the Microphone

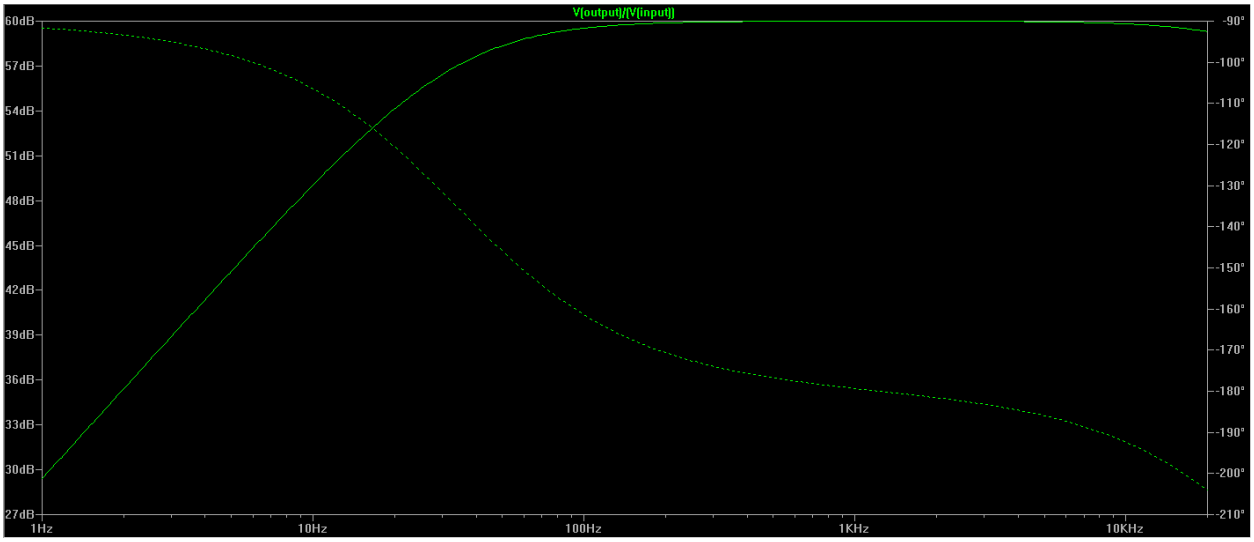


Figure 2 - A Simulation of the gain in the Microphone Preamplifier Circuit

Obstacle Avoidance Code (Loop)

```
//OBSTACLE AVOIDANCE
```

```
            PORTD_OUT=0x01;  //All LEDs on
            PORTB_OUT=0x01;
            PORTF_OUT=0x80;

            ADCA_request(ExternalChannel2,ExternalChannel2);
            uint16_t distanceright = ADCA_getVal(ExternalChannel2);
            ADCA_request(ExternalChannel3,ExternalChannel3);
            uint16_t distanceleft = ADCA_getVal(ExternalChannel3);
            x=distanceright;
            y=distanceleft;

      if(x>260 && y>260){   //If no object ahead, keep driving

            setMotorDuty(3,700,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(4,700,MOTOR_DIR_FORWARD_gc);
            }
      else if (y<x){//Left sensor reads a close object, turn right
            setMotorDuty(3,700 + 100*(1/(x-y)),MOTOR_DIR_FORWARD_gc);
            setMotorDuty(4,700 + 100*(1/(x-y)),MOTOR_DIR_BACKWARD_gc);
            }
      else if (x<y){ //Right sensor reads a close object, turn left
            setMotorDuty(3,700 + 100*(1/(y-x)),MOTOR_DIR_BACKWARD_gc);
            setMotorDuty(4,700 + 100*(1/(y-x)),MOTOR_DIR_FORWARD_gc);
            }
```

Frequency Detection Code (Loop)

```
            //FREQUENCY DETECTION CODE

            if(samples == 200){      //After 200 samples…
            ADCA_request(ExternalChannel0,ExternalChannel0);
            uint16_t distance = ADCA_getVal(ExternalChannel0);

/*Code samples at a frequency of 2600Hz.  At 200 samples, the number of crests
detected is multiplied by 13 to get the frequency*/
            frequency = maxes*13;
            printf("\r%dHz",frequency);
            maxes = 0;
            samples = 0;
            if((frequency < 800) && (frequency > 750) && (songkey == 0)){
      songkey=1;         //First note detected
            timerVar=0;
            _delay_ms(200);
            }
            if((frequency < 900) && (frequency > 850) && (songkey == 1)){
            songkey=2;         //Second note detected
            timerVar=0;
```

```
        _delay_ms(200);
        }
        if((frequency == 1170) && (songkey == 2)) {
songkey=3;          //Third note detected
        timerVar=0;
        _delay_ms(200);
        }
        if((frequency == 1248) && (songkey == 3)){
printf("\rAccess Granted");     //Last note detected
        PORTB_OUT=0x00;
        PORTD_OUT=0x00;
        PORTF_OUT=0x80;
        _delay_ms(1000);
        AccessGranted=1;   //Robot is set to move out of the way
        moveRight=1;
        songkey=0;
        objectfound = 0;
        distance = 1000;
        break;
        }
        }

//if a maximum is detected, the counter is incremented
        if(x > last_val && low == 1) maxes++; low = 0;
        if(x < last_val && low == 0) low = 1;


        if (TCC0.INTFLAGS & 0x01)//Timer set every 1/2600 of a second
        {
                last_val=x;
                ADCA_request(ExternalChannel1,ExternalChannel1);
                x = ADCA_getVal(ExternalChannel1);
                samples++;
                timerVar++;
                //Frequency key is reset after 2 seconds
                if(timerVar == 5200) songkey=0;

                TCC0.INTFLAGS &= 0x01;
        }
```

Complete Sentinel Code

```
//THE SENTINEL

#include <asf.h>
#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
```

```
#include <util/delay.h>
#include "motor.h"
#include "lcd.h"
#include "uart.h"
#include "RTC.h"
#include "picServo.h"
#include "ADC.h"
#include "sonar.h"

#define DbLedOn()       (PORTR.OUTCLR = 0x02)         //Turns the debug led
                  on.  The led is connected with inverted logic
#define DbLedOff()            (PORTR.OUTSET = 0x02)         //Turns the debug
                  led off.  The led is connected with inverted logic
#define DbLedToggle()   (PORTR.OUTTGL = 0x02)         //Toggles the debug led
                  off.  The led is connected with inverted logic



struct cir_buffer {
            uint16_t array[64];
            int arraysize,array_ind;
};

int main (void)
{
      board_init();     /*This function originates in the file init.c, and is
                        used to initialize the Epiphany DIY
                        motorInit() is declared within because by default you
                        the user should define what your
                        motor setup is to prevent hurting the Epiphany.  You
                        can do this by
                              */
      DbLedOn();  //I like to do this by default to show the board is no
                  longer suspended in the bootloader.
      servoControlInit();//initializes the servo module ***including enabling
      global interrupts*** required for the servo control module
      uartInit(&USARTC0,57600);//as can be seen in the schematic.  This uart
      is connected to the USB port.  This function initializes this uart
      stdout = &lcd_str;//This function points stdout to the USB device.  So
      this means printf will send data out the USB port.
      ADCsInits();//this function initializes the ADCs inside the Xmega
      LCDInit();
      motorInit();
      //you should add any further initializations here
      sei();


      struct cir_buffer buff;
      buff.arraysize=5;
```

```
buff.array_ind=0;



int y = 0;
int higher = 0;
int h = 0;
int check = 0;

TCC0.PER = 12;    //set the timer period to ~1/2600 second (default
                  count up mode)
TCC0.CTRLA = 7;         //set the timer prescaler to 1024 (this starts
                        the timer automatically)

TCC1.PER = 2600;  //set the timer period to ~1/2600 second (default
                  count up mode)
TCC1.CTRLA = 7;         //set the timer prescaler to 1024 (this starts
                  the timer automatically)


PORTC_DIR=0xFF;         //Read PIRs, Write to LEDS
PORTE_DIR=0x00;
PORTB_DIR=0xFF;
PORTD_DIR=0xFF;
PORTF_DIR|=0x80;

int looper=0;           //Initialize variables
uint16_t x = 0;
int randomVar2 = 0;
int low = 0;
int randomVar = 0;
int frequency = 0;
int maxes = 0;
int tester = 0;
int samples = 0;
int last_val = 0;
int begin=0;
uint16_t distanceright;
uint16_t distanceleft;
uint16_t distancerightmax;
uint16_t distanceleftmax;
uint16_t distancerightthreshL;
uint16_t distanceleftthreshL;
uint16_t distancerightthreshH;
uint16_t distanceleftthreshH;
uint16_t distanceleftmin = 220;
uint16_t distancerightmin = 220;
uint16_t distance;
int init=0;
```

```
	int songkey=0;
	int nothasty = 0;
	int nothasty2 = 0;
	int nothasty3 = 0;
	int nothasty4 = 0;
	int nothasty5 = 0;
	int nothasty6 = 0;
	int nothasty7=0;
	int objectfound = 0;
	int moveRight = 0;
	int recapture = 1;
	int i = 0;
	int noleft=0;
	int objectfound2=0;
	int timerVar;
	int AccessGranted;
	_delay_ms(500);



//PORT F=white
//PORT D=blue
//PORT B=red

	while(1){


	while(begin<10){        //Reads the initial width of the hallway
		ADCA_request(ExternalChannel2,ExternalChannel2);
		uint16_t distanceright = ADCA_getVal(ExternalChannel2);
		printf("R:%u",distanceright);
		ADCA_request(ExternalChannel3,ExternalChannel3);
		uint16_t distanceleft = ADCA_getVal(ExternalChannel3);
		printf(" L:%u",distanceleft);
		_delay_ms(100);
		printf("\r");
		begin++;          //The sonars are read several times to make
		distanceleftmax = distanceleft;     //sure the readings are
		                                    //correct
		distancerightmax = distanceright;
		distanceleftthreshL = distanceleftmin + (distanceleftmax -
distanceleftmin)/2;
		distancerightthreshL = distancerightmin + (distancerightmax -
distancerightmin)/2;
		distanceleftthreshH = distanceleftmax - (distanceleftmax -
distanceleftmin)/2;
		distancerightthreshH = distancerightmax - (distancerightmax -
distancerightmin)/2;
		}
```

```
            while(begin==10){
            printf("MR:%u ML:%u",distancerightmax,distanceleftmax);

            printf("\nWaiting for PIRs");

            _delay_ms(18000); //Waits for PIR sensors to calibrate

            printf("\r");

            begin++;
            }

      objectfound=0;    //In detection mode
      songkey=0;        //Resets frequency detection

      while(moveRight==1){                //Moves right until an object is
found or the robot reaches the other end of the hallway
            setMotorDuty(3,600,MOTOR_DIR_BACKWARD_gc);
            setMotorDuty(4,600,MOTOR_DIR_BACKWARD_gc);
            ADCA_request(ExternalChannel2,ExternalChannel2);
            uint16_t distanceright = ADCA_getVal(ExternalChannel2);
            ADCA_request(ExternalChannel3,ExternalChannel3);
            uint16_t distanceleft = ADCA_getVal(ExternalChannel3);

            if((distanceleft>distanceleftmax) ||
(distanceright>distancerightmax)){
            setMotorDuty(3,0,MOTOR_DIR_BACKWARD_gc);
            setMotorDuty(4,0,MOTOR_DIR_BACKWARD_gc);
            printf("HAVE A NICE DAY");    //The robot allows passage and
            while(AccessGranted==1){      //waits for the person to pass
            ADCA_request(ExternalChannel2,ExternalChannel2);
            uint16_t distanceright = ADCA_getVal(ExternalChannel2);
            ADCA_request(ExternalChannel3,ExternalChannel3);
            uint16_t distanceleft = ADCA_getVal(ExternalChannel3);
            if((distanceleft<(distanceleftmax-20)) &&
(distanceright<(distancerightmax-20))){ AccessGranted=0; break;}
            }
            printf("\r");
            moveRight=0;
            }

            _delay_ms(50);
            printf("\r");

      }

      while(objectfound==0){
      PORTF_OUT=0x00;         //Blue LED turns on
```

```
	PORTB_OUT= 0x00;
	PORTD_OUT = 0x01;
	if((PORTE.IN &= 0x02) == 0){
		printf("Right");  //PIR sensors are read to see if movement
	objectfound = 2;  //has been detected
		}
		else printf("     ");

		if((PORTE.IN &= 0x01) == 0){
		printf(" Left");
		objectfound = 2;
		}
		else printf("     ");
		_delay_ms(50);
		printf("\r");
	}

	PORTD.OUTTGL=0x01;
	printf("\r");            //Moving Heat detected
	printf("DETECTION!");
	PORTC_OUT=0xFF;
	_delay_ms(500);
	PORTD.OUTTGL=0x01;
	PORTC_OUT=0x00;



	ADCA_request(ExternalChannel0,ExternalChannel0);
	uint16_t distance = ADCA_getVal(ExternalChannel0);

	nothasty3=0;

	while (distance > 600){

	if (TCC1.INTFLAGS & 0x01)           //Blinks the Blue LED
		{
			PORTD.OUTTGL=0x01;
			TCC1.INTFLAGS &= 0x01;
		}

/*Moves the robot to the left until a person is detected. If no person is
detected by the time the robot reaches the end of the hallway, then the robot
continues its code*/

	ADCA_request(ExternalChannel2,ExternalChannel2);
	uint16_t distanceright = ADCA_getVal(ExternalChannel2);
	ADCA_request(ExternalChannel3,ExternalChannel3);
	uint16_t distanceleft = ADCA_getVal(ExternalChannel3);
```

```
	setMotorDuty(3,600,MOTOR_DIR_FORWARD_gc);
	setMotorDuty(4,600,MOTOR_DIR_FORWARD_gc);

	if((distanceleft<distanceleftmin) && (distanceright<distancerightmin)){
	setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
	setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
	break;
	}
	ADCA_request(ExternalChannel0,ExternalChannel0);
	uint16_t distance = ADCA_getVal(ExternalChannel0);

	if(distance < 550) nothasty3++;
	else nothasty3 = 0;
	if (nothasty3 == 5){
	printf("\n%u",distance);
	objectfound=1;                  //Object Detected!!
	setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
	setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
	break;
	}

	_delay_ms(50);
	//printf("\r");
	}

	nothasty6=0;
	noleft=0;

/*Moves the robot to the right until a person is detected. If no person is
detected by the time the robot reaches the end of the hallway, then the robot
continues its code*/

	while(objectfound != 1){

	if (TCC1.INTFLAGS & 0x01)
		{
			PORTD.OUTTGL=0x01;
			TCC1.INTFLAGS &= 0x01;
		}

	ADCA_request(ExternalChannel2,ExternalChannel2);
	uint16_t distanceright = ADCA_getVal(ExternalChannel2);
	ADCA_request(ExternalChannel3,ExternalChannel3);
	uint16_t distanceleft = ADCA_getVal(ExternalChannel3);

	setMotorDuty(3,600,MOTOR_DIR_BACKWARD_gc);
	setMotorDuty(4,600,MOTOR_DIR_BACKWARD_gc);

	if((distanceleft>distanceleftmax) || (distanceright>distancerightmax)){
```

```
        setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
        break;
        }
        ADCA_request(ExternalChannel0,ExternalChannel0);
        uint16_t distance = ADCA_getVal(ExternalChannel0);

        if(distance < 550) nothasty6++;
        else nothasty6 = 0;
        if (nothasty6 == 5){
        printf("\n%u",distance);
        objectfound = 1;        //Object Detected!!
        setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
        break;
        }

        _delay_ms(50);
        //printf("\r");
        }

        nothasty4=0;
        nothasty5=0;
        nothasty6=0;

/*Object has been detected, different functions depending on distance

        while(objectfound == 1){

                PORTF_OUT=0x00;   //Red LED turns on
                PORTD_OUT=0x00;
                PORTB_OUT=0x01;

                ADCA_request(ExternalChannel0,ExternalChannel0);
                uint16_t distance = ADCA_getVal(ExternalChannel0);

                printf("DISTANCE: %u",distance);

                if(distance < 260){
                nothasty4++;
                if(nothasty4==8){

                PORTC_OUT=0xFF;

                setMotorDuty(3,1024,MOTOR_DIR_FORWARD_gc);
                setMotorDuty(4,1024,MOTOR_DIR_BACKWARD_gc);

                printf("\rRETREAT!!!");
```

```
_delay_ms(250);

while(1){

//OBSTACLE AVOIDANCE

PORTD_OUT=0x01;   //All LEDs on
PORTB_OUT=0x01;
PORTF_OUT=0x80;

ADCA_request(ExternalChannel2,ExternalChannel2);
uint16_t distanceright = ADCA_getVal(ExternalChannel2);
ADCA_request(ExternalChannel3,ExternalChannel3);
uint16_t distanceleft = ADCA_getVal(ExternalChannel3);
x=distanceright;
y=distanceleft;

if(x>260 && y>260){     //If no object ahead, keep driving

setMotorDuty(3,700,MOTOR_DIR_FORWARD_gc);
setMotorDuty(4,700,MOTOR_DIR_FORWARD_gc);
}
else if (y<x){//Left sensor reads a close object, turn right
setMotorDuty(3,700 + 100*(1/(x-y)),MOTOR_DIR_FORWARD_gc);
setMotorDuty(4,700 + 100*(1/(x-y)),MOTOR_DIR_BACKWARD_gc);
}
else if (x<y){ //Right sensor reads a close object, turn left
setMotorDuty(3,700 + 100*(1/(y-x)),MOTOR_DIR_BACKWARD_gc);
setMotorDuty(4,700 + 100*(1/(y-x)),MOTOR_DIR_FORWARD_gc);
}

//OBSTACLE AVOIDANCE END

}



}
}

if(distance < 350){
nothasty5++;            //At a certain distance, the alarm turns
on
if(nothasty5==3){
printf("\nALARM!!!");
PORTC_OUT=0xFF;
nothasty5=2;
}
}
```

```
            if(distance>350){
                  PORTC_OUT=0x00;
                  nothasty4=0;
                  nothasty5=0;
            }

            while ((distance > 400) && (distance < 800)){

            PORTC_OUT=0x00;
            //FREQUENCY DETECTION CODE

            if(samples == 200){     //After 200 samples…
            ADCA_request(ExternalChannel0,ExternalChannel0);
            uint16_t distance = ADCA_getVal(ExternalChannel0);

            if(distance < 400){     //Person still in range?
            nothasty++;
            if(nothasty == 10) break;}
            else nothasty=0;

            if(distance > 800){     //Person still in range?
            nothasty2++;
            if(nothasty2 == 10) break;}
            else nothasty2=0;

/*Code samples at a frequency of 2600Hz.  At 200 samples, the number of
crests detected is multiplied by 13 to get the frequency*/
            frequency = maxes*13;
            printf("\r%dHz",frequency);
            maxes = 0;
            samples = 0;
            if((frequency < 800) && (frequency > 750) && (songkey == 0)){
            songkey=1;        //First note detected
            timerVar=0;
            _delay_ms(200);
            }
            if((frequency < 900) && (frequency > 850) && (songkey == 1)){
            songkey=2;        //Second note detected
            timerVar=0;
            _delay_ms(200);
            }
            if((frequency == 1170) && (songkey == 2)) {
            songkey=3;        //Third note detected
            timerVar=0;
            _delay_ms(200);
            }
            if((frequency == 1248) && (songkey == 3)){
            printf("\rAccess Granted");   //Last note detected
```

```
            PORTB_OUT=0x00;
            PORTD_OUT=0x00;
            PORTF_OUT=0x80;
            _delay_ms(1000);
            AccessGranted=1;  //Robot is set to move out of the way
            moveRight=1;
            songkey=0;
            objectfound = 0;
            distance = 1000;
            break;
            }
            }

      //if a maximum is detected, the counter is incremented
            if(x > last_val && low == 1) maxes++; low = 0;
            if(x < last_val && low == 0) low = 1;


            if (TCC0.INTFLAGS & 0x01)     //Timer set every 1/2600 of a
second
            {
                  last_val=x;
                  ADCA_request(ExternalChannel1,ExternalChannel1);
                  x = ADCA_getVal(ExternalChannel1);
                  samples++;
                  timerVar++;
                  //Frequency key is reset after 2 seconds
                  if(timerVar == 5200) songkey=0;

                  TCC0.INTFLAGS &= 0x01;
            }

            //FREQUENCY DETECTION CODE END

            }


      ADCA_request(ExternalChannel0,ExternalChannel0);
      distance = ADCA_getVal(ExternalChannel0);

      if(distance > 800){
            PORTC_OUT=0x00;
            nothasty7++;
            if(nothasty7 == 10) recapture=1;
      }
      else nothasty7=0;

      if(recapture==1){ //If the object has fled, the robot searches for it
again
```

```
            recapture=0;
            nothasty7=0;

            _delay_ms(500);

            ADCA_request(ExternalChannel2,ExternalChannel2);
            uint16_t distanceright = ADCA_getVal(ExternalChannel2);
            ADCA_request(ExternalChannel3,ExternalChannel3);
            uint16_t distanceleft = ADCA_getVal(ExternalChannel3);

            if((distanceleftthreshH < distanceleft) || (distancerightthreshH
< distanceright)){
                  printf("\rGO LEFT");

//GO LEFT UNTIL FINDING AN OBJECT. IF NO OBJECT, THEN GO RIGHT

      objectfound2=0;

      ADCA_request(ExternalChannel0,ExternalChannel0);
      distance = ADCA_getVal(ExternalChannel0);

      nothasty3=0;

      while (distance > 600){
      ADCA_request(ExternalChannel2,ExternalChannel2);
      uint16_t distanceright = ADCA_getVal(ExternalChannel2);
      ADCA_request(ExternalChannel3,ExternalChannel3);
      uint16_t distanceleft = ADCA_getVal(ExternalChannel3);

      setMotorDuty(3,612,MOTOR_DIR_FORWARD_gc);
      setMotorDuty(4,600,MOTOR_DIR_FORWARD_gc);

      if((distanceleft<distanceleftmin) && (distanceright<distancerightmin)){
      setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
      setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
      break;
      }
      ADCA_request(ExternalChannel0,ExternalChannel0);
      uint16_t distance = ADCA_getVal(ExternalChannel0);

      if(distance < 550) nothasty3++;
      else nothasty3 = 0;
      if (nothasty3 == 15){
      printf("\n%u",distance);
      objectfound2=1;         //Object found!!
      setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
      setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
      break;
      }
```

```
	_delay_ms(50);
	}


	//GOING RIGHT

objectfound2=0;
nothasty6=0;

	while(1){

	ADCA_request(ExternalChannel2,ExternalChannel2);
	uint16_t distanceright = ADCA_getVal(ExternalChannel2);
	ADCA_request(ExternalChannel3,ExternalChannel3);
	uint16_t distanceleft = ADCA_getVal(ExternalChannel3);

	setMotorDuty(3,600,MOTOR_DIR_BACKWARD_gc);
	setMotorDuty(4,612,MOTOR_DIR_BACKWARD_gc);

	if((distanceleft>distanceleftmax) || (distanceright>distancerightmax)){
	setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
	setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
	break;
	}
	ADCA_request(ExternalChannel0,ExternalChannel0);
	uint16_t distance = ADCA_getVal(ExternalChannel0);

	if(distance < 550) nothasty6++;
	else nothasty6 = 0;
	if (nothasty6 == 15){
	printf("\n%u",distance);
	objectfound2 = 1;
	setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
	setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
	break;
	}

	_delay_ms(50);
	}

	if(objectfound==0)	break;
		}



	else if((distanceleftthreshL > distanceleft) && (distancerightthreshL >
distanceright)){
		printf("\rGO RIGHT");
```

```
//GO RIGHT UNTIL FINDING AN OBJECT. IF NO OBJECT, THEN BREAK

            objectfound2=0;
            nothasty6=0;
            while(1){

ADCA_request(ExternalChannel2,ExternalChannel2);
uint16_t distanceright = ADCA_getVal(ExternalChannel2);
ADCA_request(ExternalChannel3,ExternalChannel3);
uint16_t distanceleft = ADCA_getVal(ExternalChannel3);

setMotorDuty(3,600,MOTOR_DIR_BACKWARD_gc);
setMotorDuty(4,612,MOTOR_DIR_BACKWARD_gc);

if((distanceleft>distanceleftmax) || (distanceright>distancerightmax)){
setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
break;
}
ADCA_request(ExternalChannel0,ExternalChannel0);
uint16_t distance = ADCA_getVal(ExternalChannel0);

if(distance < 550) nothasty6++;
else nothasty6 = 0;
if (nothasty6 == 15){
printf("\n%u",distance);
objectfound2 = 1;       //Object found!!
setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
setMotorDuty(4,0,MOTOR_DIR_FORWARD_gc);
break;
}

_delay_ms(50);
}

if(objectfound2 == 0) break;        //No object found
      }
}

_delay_ms(50);
printf("\r");

}
}
      }
```