Yushun He
Written Report
Project: DCR
EEL5666 FALL 2011
Intelligent Machines Design Laboratory
Dr. Arroyo and Dr. Schwartz

Table of Contents

Abstract

DCR (Debris Clearing Robot) is an autonomous debris removing agent. It is capable of traversing a designated area and removes any debris from that area, also it can be switched to a different mode, in which it follows a path and remove any debris from the path. Once the debris is removed it will continue searching for other debris. In the 'following the path' mode, once it reaches the end of the path its objective is completed. It will spin as re-traverse path looking for more debris.

Executive Summary

DCR acts as a micro prototype for later large world implementation. It performs fairly well, most of the time it follows the line except for some occasions. This is due to the limited number of sensors on the robot. DCR has five IR sensors for line detection and, sonar and two more IR sensors for object detection. A decision was made not to add more sensors as the current quantity of sensors has needed a substantial amount of power. DCR is designed to remove objects within a certain range and it does this will. It grabs the object with its grippers and then quickly pushes it out of the way.
The project is designed for the intelligent machines design laboratory, department of ECE, University of Florida: Supervised by Dr. A. Antonio Arroyo and Dr. Eric M. Schwartz.
For more information: https://sites.google.com/site/sonnyhe2002/imdl

Introduction

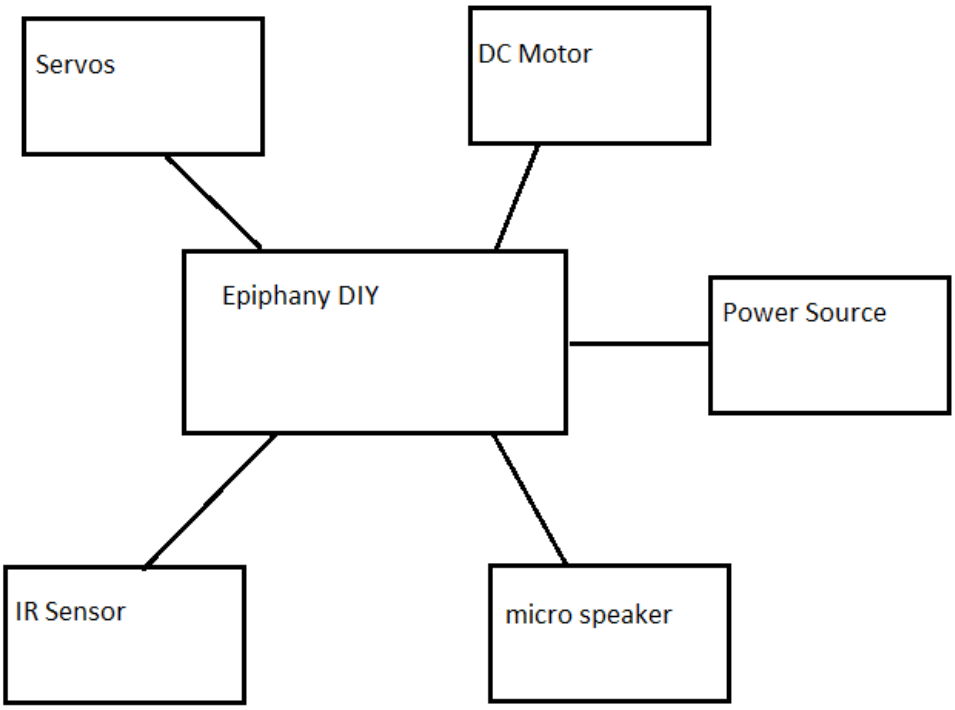
In the real world, debris is a major concern in our everyday lives. Any object that is discarded is considered debris. Debris on the road can cause major damage to vehicles and cause road accidents and debris in the vicinity is an environmental hazard. It is hard for people to remove debris as these objects may weight from a few grams to thousands of kilograms, also there are constantly new debris everyday. It is best suited for autonomous robots to deal with this problem. The autonomous robot is more powerful and works longer than a human. The robot also needs to patrol roads as this is where most problems occur.

DCR stands for Debris Clearing Robot and its main goal to clear all debris in the area.

DCR will patrol within the parameter and search for an obstructions. If it finds any debris, it will try to scoop up the debris, like a bulldozer and drop it outside of the designated area. For objects that are too big to pick up, it will try to push the object out. If the object is immovable then it will stop and emit a siren to notify the people around the area. It will stop the siren when the object is removed.
DCR is also designed to work on the road. Like an autonomous robot or vehicle that operates on the road it should follow the rules of the road such as obeying stop signs, taking detours when roads are closed and following other traffic rules. On the road it navigates through the path and remove any debris on the path so the road is cleared. DCR will stop when the road end, this shows that DCR has completed it's task and removed all debris from the road and is now safe for vehicles to use.

Integrated System

The main part is the Epiphany DIY circuit board by Tim, which features the Atmel Xmega micro-controller. All other components is built around the Epiphany DIY board. It has 5 IR sensors underneath the robot to detect the path it is following, and 2 IR sensors in the front to detect objects ahead. The total number of IR sensors on DCR is 7. It uses 2 servos to lift up its scoop arm, one servo for the left and right side of DCR. It will have a mico speaker on the top of the robot to emit sound. For movement, it is 2 wheel drive with a DC motor for each wheel. All of this is powered by 2, 8 pack AA batteries mounted at the back of the robot. The 'search within area' mode and 'follow a path' mode is toggled with a switch.



Obstacle analysis:

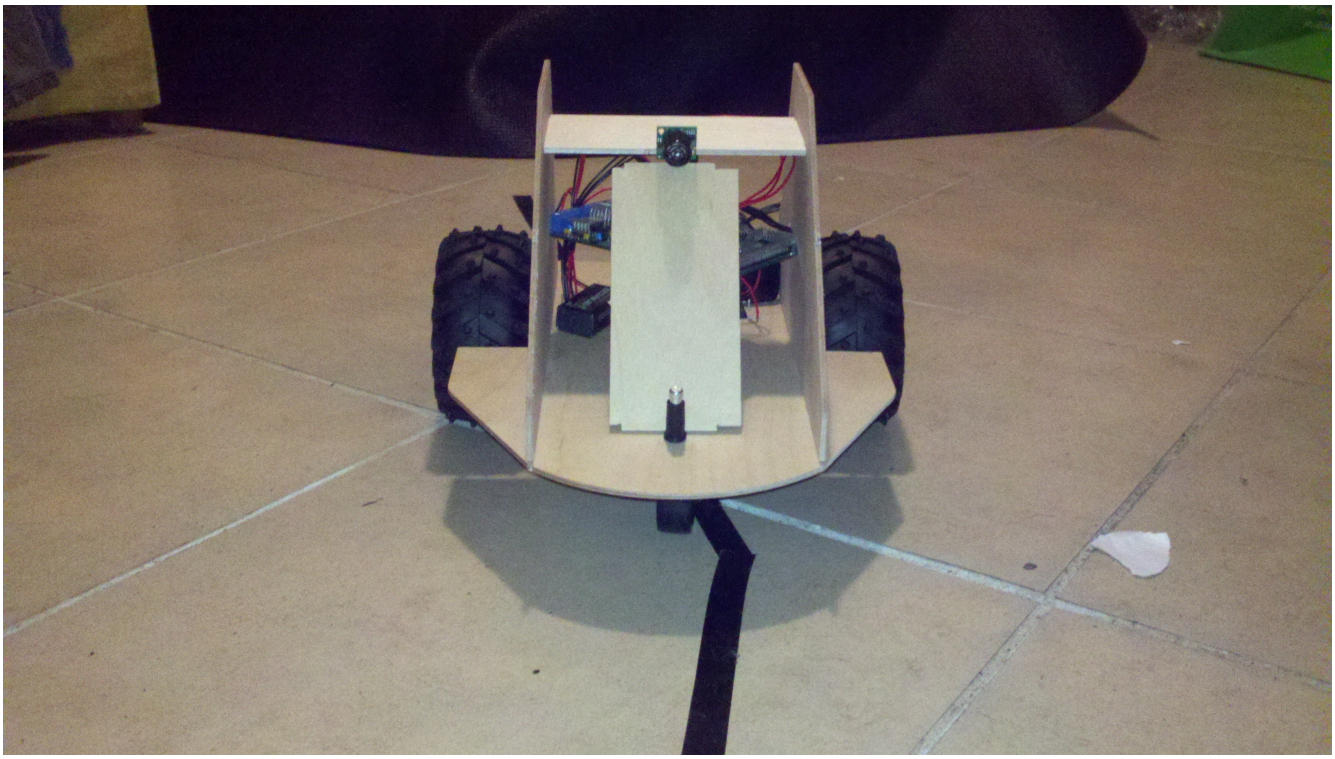
1. Sonar sensor to detect debris from a distance.
2. Move closer in distance and align with the object with sonar in conjunction with the two side IR sensors.
3. Compute size of the object using the front sensors.
4. Depending on the object size: grab, turn and move the object out of the way, or send alarm using led and sound until the object is remove
5. Return to the path and continue.

Line follow and detection system:

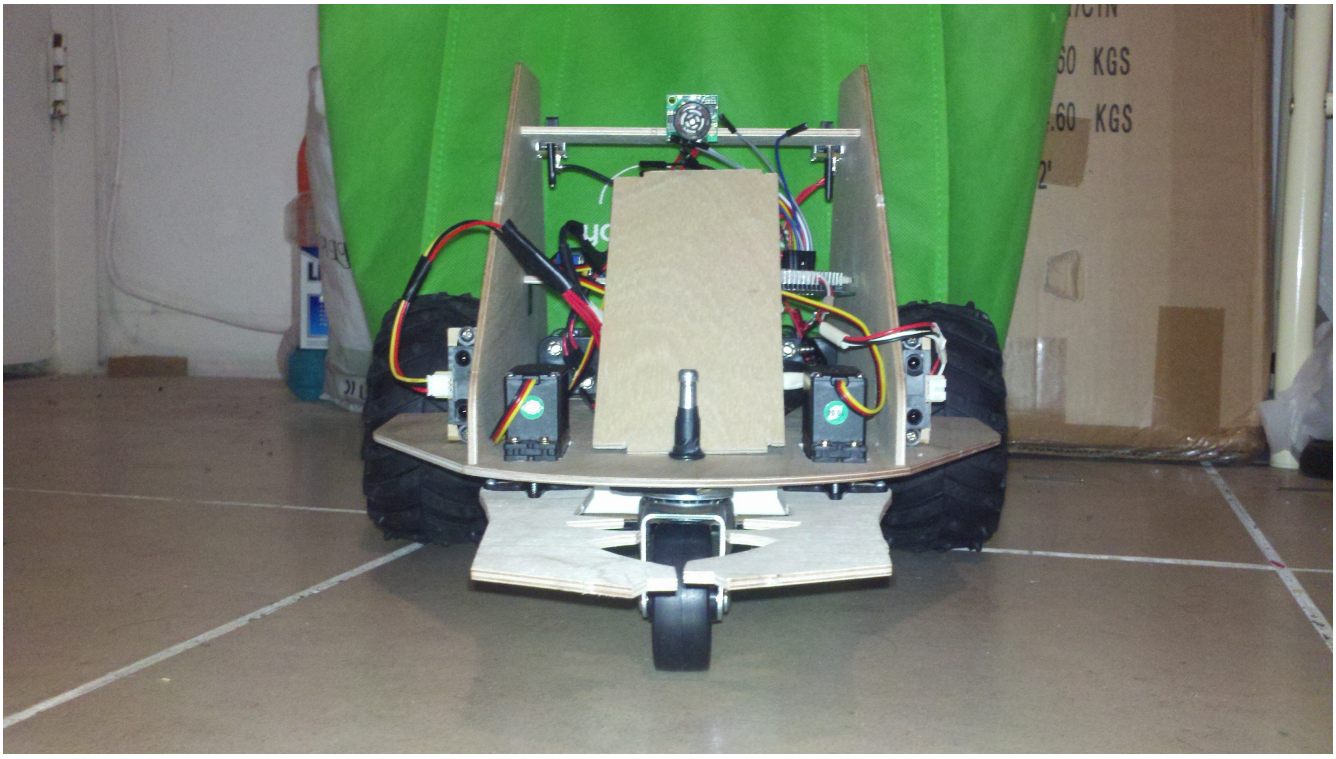
Use the IR sensors underneath the robot to detect the line.

- If there is no line detected, drive until line is found.
- If all five sensors detect the line spin until middle IR sensor detects and realign the line follow.
- If the 1 sensor detect the line turn hard left.
- If the 2 sensor detect the line turn left.
- If the 3 sensor detect the line drive.
- If the 4 sensor detect the line turn right.
- If the 5 sensor detect the line turn hard right.
- If the 3 and 4 sensor detect the sway right.
- If the 2 and 3 sensor detect the sway right.
- If line is not detected but last sensor data detects the line, continue performing last function.

Mobile Platform



The platform was designed with for precision and functionality.
The robot platform is from 1/8” thick board as it has less weight. The base platform has a curve at the front for impact absorption. It also extends out to the wheels to protect the wheels from being stuck. The two side walls are used to support the CPU board and upper level. These are shaped like a tail wing for better stability and greater center of gravity.



It also has two arms for grabbing objects. These arms are also made out of 1/8” board that are shaped like a beetle’s pincer. This was not designed from a look of a beetle but it was designed to hold things well.

Actuation

DCR travels on two wheels each running on an individual motor. There is a third wheel for support and rotational help. The DC motors are10:1 Micro Metal Gearmotor HP which runs at 6V.

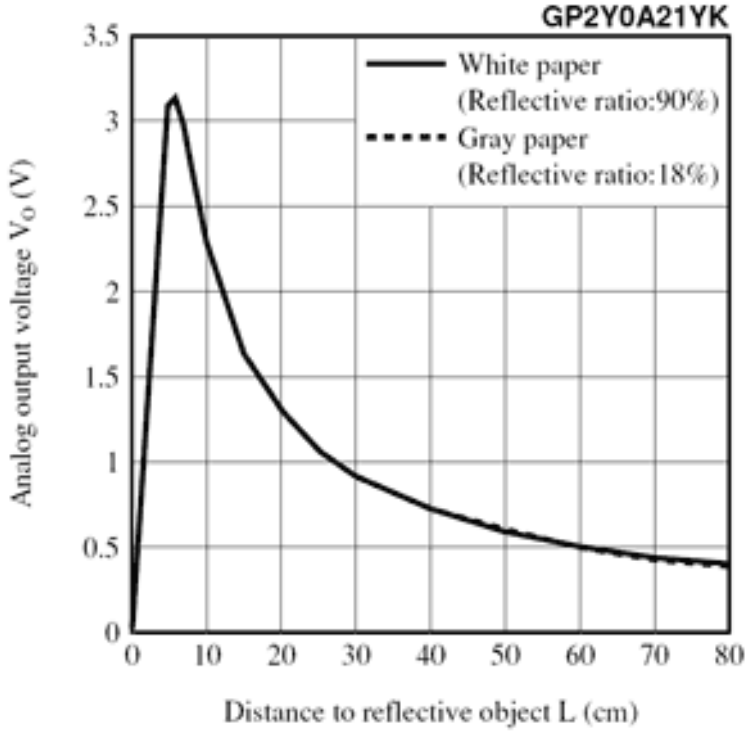
The two servos that control the scoop are ROB-09064 Large Servo, which is powerful enough of life heavy objects. The ROB-09064 runs at 4.8 V to 6V, 0.16 sec/ 60 degree movement and a stall torque of 6.5 kg*cm.

The motors and arm work together to scoop up objects. DCR will close into the object with its arms gripping it with the jagged edges in between.

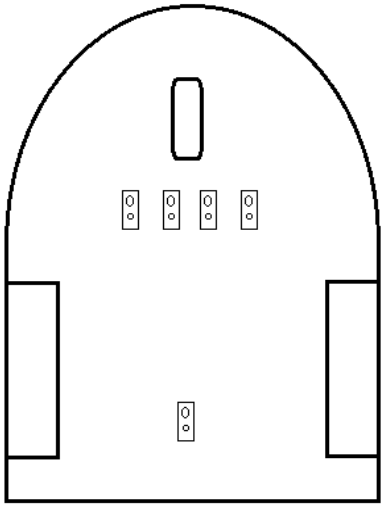
DCR send a sound alert using a hacked Hallmark card that plays sound.

Sensors

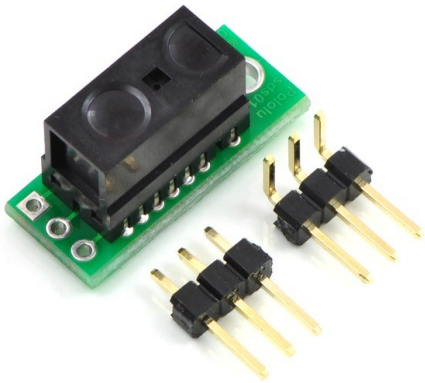
The GP2Y0A21 Sharp distance sensor is used for obstacle detection and obstacle avoidance or motion sensing to your robot or any other project. With a detection range of 4" to 32" and an analog voltage indicating the distance. This sensor is mounted on each side of the robot.



Overview Special Sensor Implementation



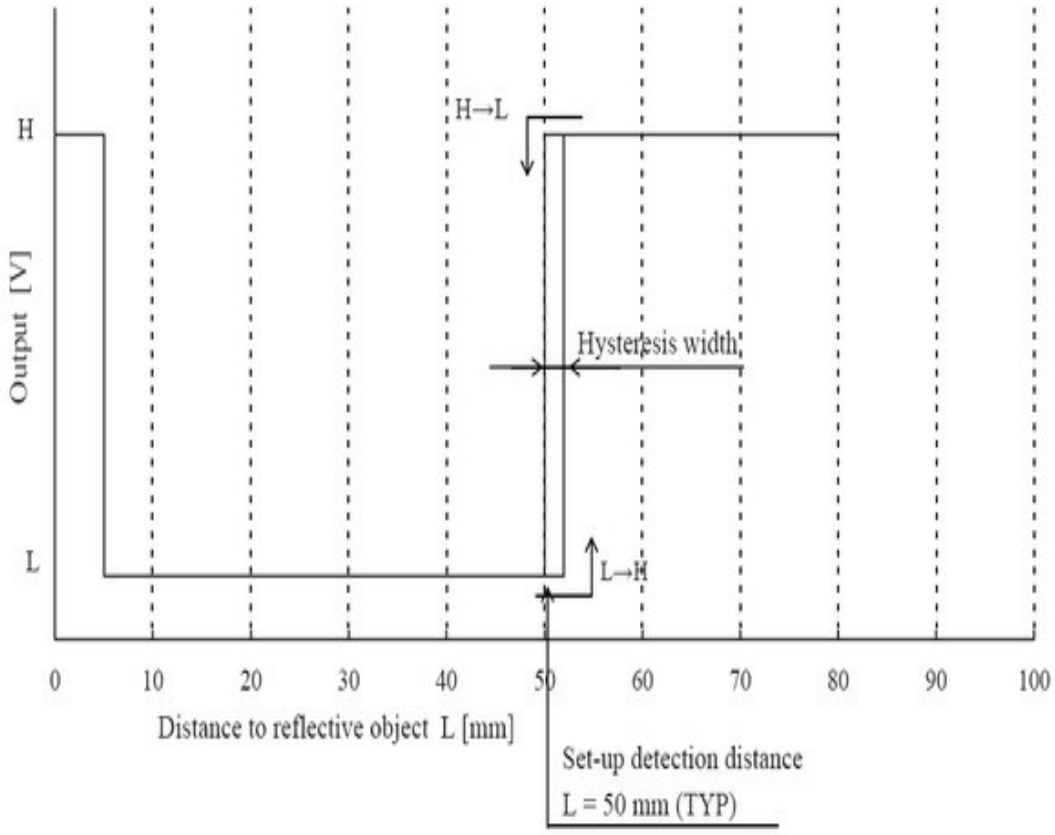
The main function of DCR is to follow a line to find obstacles in the path. The special sensor for doing this is an array of IR sensors arranged beneath the robot. The IR sensors will detect the difference in reflectivity in the object (Mainly the color of the object will determine this). The line will be made of white masking tape for the robot to follow.
The IR sensors are Sharp GP2Y0D810Z0F which is detects objects in distances of 2 to 10 cm. It is also small enough to fit 4 of these sensors underneath the robot. The sample rate of the sensors is 390 Hz which is adequate since the robot moves fairly slowly. The distance from the floor to the bottom of the robot is 2.4 inches which is perfect since the range of these IR sensors are 1 inch to 4 inches.



Although 4 sensors are good enough for the array, 5 will help verify the line and also will give a more precise measurement of which way the line is directed.

Sensor data

The IR sensor measures a high and low value. Therefore it cannot measure distance but the purpose of line follow only requires the detections to different color on the ground.



From the testing of a IR sensor, The sensor gives a signal of 2977 plus or minus 10, when nothing is detected or detection on dark color objects. The detection of an light color object gives a signal of 530 plus or minus 10. All of the IR sensors gives the same readings of around 2977 for no detection and 530 for detection.

Sample Result:

2971
2974
2973
2975
2980
2974
2971
2989
2972

2977
533
533
536
531
534
526
534
526
534

This is a sample of one of the tests. The first numbers show that the sensor has not detect anything, then a white paper is presented and the sensor display values of 530.
The experiments were done with various objects and the values are the same as long as the object is of light color: white wall paint, paper, light color clothes. If dark color objects are presented, it will show high values of 2970 the same as not being detected.

The time of sensor response is approximately 0.30 seconds.

Details of Implementation

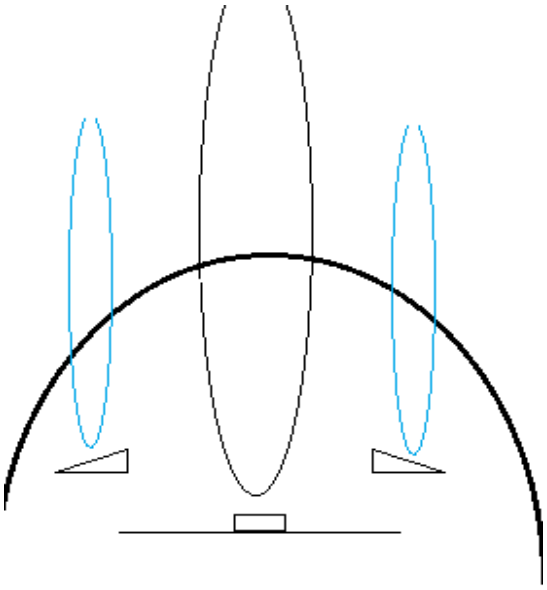
An array of integer is used to store the values for the IR sensors. The exact value is not needed, only '1' for line detected and '0' for no line detected. The extra IR sensor between the wheels is stored in a separate integer variable so it is easier to debug then being part of the array.

0	0	1	0	0

The above figure shows the array of int. It indicated the line is directly in the middle of the robot since the width of the line is the distance of the two IR sensors. If the array shows 11000 that means the line is turning left and the robot turns left. If the array shows 00011 shows the line is going right and robots turns right. If the array shows 11111 then the robot will spin until any of the previous array formation is detected. If the array is 00000 then it will go into random search mode to look for the line. If an edge case of 10001 is detected then the robot will spin around and try to find a proper configuration.

Detecting Objects

The Robot needs to detect objects in-front of the path, for this a sensor is used to detect the distance of the object and get close to it. Once close enough the 2 IR sensors mounted on the front of the robot will help detect if the object is small enough to be placed in the robots arm grippers. Since the IR sensors have a very narrow beam, the acceptable with can be determined easily. The 2 IR sensors is 8 inches apart. If any object is only detected by the sonar and not by the IR sensors then the robot will try and pick it up but if the IR sensors pick it up then it will be determined as too large.



The IR sensors are represented by the two blue beams and the sonar by the black.
If only one of the IR sensors and the sonar detect the object then robot will reverse and then drive until it is facing the center of the object. If it is still detected by any of the IR sensors then it is determined to be too large and then robot will search around for other object. If it is in follow the path mode and the large object is on the path then the robot will sound an alarm until the object is moved.

Sensor Data

The sonar used is Maxbotics LV-MaxSonar-EZ1 which detects objects 1inch to 6 inches according to the datasheet. But from testing the sonar can detect object up to 10 inches. This sensor is also used for obstacle avoidance.

From testing once the signal is 235, the object is right in front of the robot. I found that dividing the signal by 512 to get the distance is inaccurate, so the robot needs to be calibrated to account for the distance.
Also I have test three sonars each of them produces different values, so sensor 'A' may produce a signal of 2056 of no object being detect while sensor 'B' produces a signal of 4019 when no object is detected.

From the results of the three sensors, the width of the object has to be less than 4.7 inches for the 2 IR sensors not to pick it up and for the sonar to pick it up. This size is perfect for the servo arms.

Conclusion

The main functionality of the robot is complete. It successfully removes objects from the path and also with the line following mechanism; the robot can be tweaked to race on a track. The line following with difficult to implement because of the limited number of ADCA ports on the board and the angular movement of the turn. I have used filters and also tested the function:
Speed = (K/K+1)*Old_value + (1/K+1)*New_value ($K \geq 0$)
In the end I have dropped the function for a simpler turning system as it was not responsive enough.
One major concern about the robot is that it requires a lot of power. I have use 16 AA batteries to support its operation but it last less than half an hour. Some functionality in the end was not implemented; the sound for the alarm had to be dropped because the mini speaker's audio was too low quality. There were also restrictions; the robot was too large, this disallows the robot to make sharp turns and needs a large testing area. Therefore the demonstration mat I used was inadequate to see the robot's full range of motions
The DIY board had some glitches during the implementation, so development time was lost.
But overall the robot functions and performs its tasks.

The class was challenging but fun. I had a really hard time at the beginning because as a computer science student, this was all new to me. I had to learn basic things like soldering the board to connecting wires. I ended up always being behind a week of work. It was not until I demonstrated successful obstacle avoidance that then I know I could make the robot.

Appendix

// Code for DCR. \\

```
#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



int CR = 1.05;
int MAX = 820;
int MAXSTRAIGHT = 595;
int MIN = 180;
int SONAR = 3;
int RIGHT_MOTOR = 4;
int LEFT_MOTOR = 2; //
int LEFT_ARM_FOLD = 25;
int RIGHT_ARM_FOLD = 155;
int LEFT_ARM_OPEN = 180; // arm right in
int RIGHT_ARM_OPEN = 0;
int _rightSpeed = 0;
int _leftSpeed = 0;
```

```
Bool _closedArmState = false;

Bool sensorDetect(void);

Bool detectInDistance(void);

Bool outOfLineDetect(void);

Bool middleDetect(void);

Bool largeDebrisDetect(void);

Bool irLeftDetect(void);

Bool irRightDetect(void);

Bool irLEFT(void);

Bool irRIGHT(void);


int irONE(void);

int irTWO(void);

int irTHREE(void);

int irFOUR(void);

int irFIVE(void);


void start(void);

void alertUntilRemove(void);

void sound(void);

void stop(void);

void stopHarder(void);

void stop2();

void reverse();
```

```
void drive(void);

void push(int);

void grabStuff(void);

void drive2();

void stopTurn(void);

void returnPatrol(void);

void increaseRightMotor(int);

void increaseLeftMotor(int);

void increaseSpeed(int);

void decreaseLeftMotor(int);

void decreaseRightMotor(int);


void turn180();

void mostLeft();

void mostRight();

void turnLeft();

void turnRight();

void swadLeft();

void swadRight();

void turnLeft2();

void turnRight2();

void swadLeft2();

void swadRight2();
```

```
void lineFollow();

void getInDistance();

void ungrab();

void grab();


int main (void)
{
	board_init();

	servoControlInit();

	uartInit(&USARTC0,57600);

	stdout = &USB_str;

	ADCsInits();

	motorInit();

	sei();

	_delay_ms(100);

	ungrab();

	_delay_ms(400);

	start();
}
```

```
void start(){
	while(1){
		if(sensorDetect()){
			stopHarder();
			_delay_ms(120);
			getInDistance();
			stopHarder();
			_delay_ms(400);
			if (largeDebrisDetect())
			{
				alertUntilRemove();
			}else{
				grabStuff();
			}
		}else{
			lineFollow();
		}
	}
}

void grabStuff(){
	grab();
	_delay_ms(1800);
	stopTurn();
	stop();
	_delay_ms(50);
	stopHarder();
	_delay_ms(500);
```

```
	push(MAX+10);
	_delay_ms(260);
	stop();
	ungrab();
	_delay_ms(1000);
	reverse();
	_delay_ms(2080);
	stop();
	_delay_ms(500);
	while(!middleDetect()){
		drive();
	}
	turnRight();
}

void getInDistance(){
	while(!detectInDistance())
	{
		if (irLeftDetect() && irRightDetect()){
			_rightSpeed = 0.8 * _rightSpeed + (555*0.2);
			_leftSpeed = 0.8 * _leftSpeed + (555*0.2);
			setMotorDuty(RIGHT_MOTOR, _rightSpeed*CR,
MOTOR_DIR_FORWARD_gc);
			setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
		} else if(irLeftDetect()){
```

```
                    _rightSpeed = (0.5 * _rightSpeed) + (641 * 0.5);

                    _leftSpeed = (0.5 * _leftSpeed) + (555 * 0.5);

                    setMotorDuty(RIGHT_MOTOR, _rightSpeed*CR,
MOTOR_DIR_FORWARD_gc);

                    setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);

            } else if(irRightDetect()){

                    _rightSpeed = (0.5 * _rightSpeed) + (555 * 0.5);

                    _leftSpeed = (0.5 * _leftSpeed) + (632 * 0.5);

                    setMotorDuty(RIGHT_MOTOR, _rightSpeed*CR,
MOTOR_DIR_FORWARD_gc);

                    setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);

            } else{

                    _rightSpeed = 0.9 * _rightSpeed + 55;

                    _leftSpeed = 0.9 * _leftSpeed + 55;

                    setMotorDuty(RIGHT_MOTOR, _rightSpeed*CR,
MOTOR_DIR_FORWARD_gc);

                    setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);

            }
        }
}



Bool detectInDistance(){

        int vcc, BestVcc;

        Bool finding = true;

        for(int count = 0; count<5;count++){

                ADCA_request(0,SONAR);
```

```
                while(!ADCA_CH0_ConvComplete);
                vcc = ADCA_getVal(0);
                if(count == 2)
                        BestVcc = vcc;
                if (vcc < MIN){
                        finding = false;
                }
        }
        printf("BestVcc %d \r",BestVcc);
        if (BestVcc < 231)
        {
                return true;
        }else{
                return false;
        }
}

void alertUntilRemove(){
        while(largeDebrisDetect()){
                DbLedToggle();
                _delay_ms(1000);
        }
}
```

```
void grab(){

	if (getServoAngle(5) != LEFT_ARM_FOLD)

	for (int x = getServoAngle(5); x>LEFT_ARM_FOLD; x--)

	{
			setServoAngle(x,5);
	}

	if(getServoAngle(9) != RIGHT_ARM_FOLD)

	for(int j = getServoAngle(9);j<RIGHT_ARM_FOLD;j++){

			setServoAngle(j,9);
	}

	_closedArmState = true;

}


void ungrab(){

	if (getServoAngle(5) != LEFT_ARM_OPEN)

	for (int x = getServoAngle(5); x<LEFT_ARM_OPEN; x++)
	{
			setServoAngle(x,5);

			_delay_ms(10);
	}

	if(getServoAngle(9) != RIGHT_ARM_OPEN)

	for(int j = getServoAngle(9);j>RIGHT_ARM_OPEN;j--){

			setServoAngle(j,9);

			_delay_ms(10);
	}
	_closedArmState = false;

}
```

```
Bool irLeftDetect()
{
	for(int x = 10; x >0; x--){

		if(!irLEFT()){

			return false;

		}
		_delay_ms(1);
	}
	return true;
}


Bool irRightDetect()
{
	for(int x = 10; x >0; x--){

		if(!irRIGHT()){

			return false;

		}
		_delay_ms(1);

	}
	return true;

}


Bool middleDetect()
{
	int line = 0;

	for(int x = 0; x <10; x++){

		line = line + irTHREE();

		_delay_ms(1);
	}

	if (line == 10){

		return true;
```

```
        }else{

                return false;
        }
}


Bool outOfLineDetect()
{
        int line = 0;

        for(int x = 0; x <10; x++){

                line = line + irTWO()+ irTHREE() + irFOUR();

                _delay_ms(1);

        }

        if (line == 0){
                        return true;
                }else{
                        return false;
                }
}


void lineFollow()
{

        int secondLastMove = 0;

        int lastmove = 0;


        DbLedOff();


        // filter ir sensors

        int countTWO = 0, countTHREE = 0, countFOUR = 0, countFive = 0;

        for(int x = 0; x <10; x++){

                countTWO = countTWO + irTWO();
```

```
        countTHREE = countTHREE + irTHREE();
        countFOUR = countFOUR + irFOUR();
        countFive = countFive + irFIVE();
        _delay_ms(1);                       // this here for best result
}
int left =0, center = 0, right = 0, moreRight = 0;
if (countTWO==10)   {       left      = 1;}
if (countTHREE==10)         {       center  = 1;}
if (countFOUR==10)  {       right    = 1;}
if (countFive==10)     {       moreRight = 1;}

uint8_t irs = (left << 3 | center << 2 | right <<1 | moreRight);

switch(irs)
{
        case 0b0000001:                 // most right
                mostRight();
                break;
        case 0b0000010:                 // right
                turnRight();
                break;
        case 0b0000011:                 // most right
                mostRight();
                break;
        case 0b0000110:                 // swad right
```

```
                swadRight();
                break;
            case 0b0000100:                 // center
                DbLedOn();
                drive();
                break;
            case 0b0001100:                 // swad left
                swadLeft();
                lastmove=1100;
                break;
            case 0b0001000:                 //left
                turnLeft();
                lastmove=1000;
                break;
            case 0b0001111:                 // turn around and try again
                stopHarder();
                _delay_ms(450);
                returnPatrol();
                break;
        }
}
```

```
void returnPatrol(){
	Bool outOfStart = false;
	Bool foundLine = false;
	turn180();
	_delay_ms(830);
	while (!foundLine)
	{
		turn180();
		if(!outOfStart && outOfLineDetect())
			outOfStart= true;
		else if (outOfStart && middleDetect())
		{
		stop();
			_delay_ms(100);
			if (middleDetect())
				foundLine = true;
		}
	}
}

Bool largeDebrisDetect(){
	if(irLeftDetect() && irRightDetect()){
		return true;
	}else{
		return false;
	}
}
```

```
Bool sensorDetect()
{
		int vcc, BestVcc;
		Bool finding = true;
		for(int count = 0; count<4;count++){
			ADCA_request(0,SONAR);
			while(!ADCA_CH0_ConvComplete){
			}
			vcc = ADCA_getVal(0);
			if(count == 1)
				BestVcc = vcc;
			if (vcc < MIN){
				//printf("false less than min %d \r",BestVcc);
				finding = false;
			}
		}
		if(finding && BestVcc < 250){
			//printf("true %d \r",BestVcc);
			return true;
		}else{
			//printf("false %d %d\r",BestVcc, MIN);
			return false;
		}
}
```

```
Bool irLEFT(){

        ADCA_request(3,2);

        while(!ADCA_CH3_ConvComplete);

        int x = ADCA_getVal(3);

        if (x >1400)
                {

                        return true;

                }else{

                        return false;

                }

}


Bool irRIGHT(){

        ADCA_request(0,0);

        while(!ADCA_CH0_ConvComplete);

        int x = ADCA_getVal(0);

        if (x >1400)
                {

                        return true;

                }else{

                        return false;

                }

}
```

```
int irONE(){

        ADCA_request(0,3);
        while(!ADCA_CH0_ConvComplete);
        int x = ADCA_getVal(0);
        if (x <1500)
        {
            return 1;
        }else{
            return 0;
        }
}

int irTWO(){

        ADCA_request(0,4);
        while(!ADCA_CH0_ConvComplete);
        int x =ADCA_getVal(0);
        if (x <1500)
        {
            return 1;
        }else{
            return 0;
        }
}
```

```
int irTHREE(){

		ADCA_request(0,5);

		while(!ADCA_CH0_ConvComplete);

		int x = ADCA_getVal(0);

		if (x <1500)
		{
			return 1;

		}else{

			return 0;

		}

}


int irFOUR(){

		ADCA_request(1,6);

		while(!ADCA_CH1_ConvComplete);

		int x = ADCA_getVal(1);

		if (x <1500)
		{

			return 1;

		}else{

			return 0;

		}

}
```

```
int irFIVE(){


		ADCA_request(1,7);

		while(!ADCA_CH1_ConvComplete);

		int x = ADCA_getVal(1);

		if (x <1500)
		{
			return 1;

		}else{
			return 0;
		}

}

void swadLeft(){

	_leftSpeed = MAXSTRAIGHT - 45;

	_rightSpeed = MAXSTRAIGHT + 35;

	setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);

	setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);

}

void swadRight(){

	_leftSpeed = MAXSTRAIGHT + 35;

	_rightSpeed = MAXSTRAIGHT - 40;

	setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);

	setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);

}
```

```
void turnLeft(){
        _leftSpeed = MAXSTRAIGHT - (MAXSTRAIGHT *0.62);
        _rightSpeed = MAXSTRAIGHT * 1.2; // 110 is better // 100
        setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
}
void goodturnLeft(){
        _leftSpeed = MAXSTRAIGHT - 260; // 260 is better // 110
        _rightSpeed = MAXSTRAIGHT + 110; // 110 is better // 100
        setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
}
void mostRight(){
        _rightSpeed = 0;
        _leftSpeed = MAXSTRAIGHT *1.19;
        setMotorDuty(RIGHT_MOTOR, 500, MOTOR_DIR_BRAKE_gc);
        setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
}
void turnRight(){
        _rightSpeed =MAXSTRAIGHT - (MAXSTRAIGHT*0.62);
        _leftSpeed = MAXSTRAIGHT *1.19;
        setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
}
```

```
void goodturnRight(){
	_leftSpeed = MAXSTRAIGHT + 110;
	_rightSpeed =MAXSTRAIGHT - 265;
	setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
	setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
}

void mostLeft(){
	_rightSpeed = MAXSTRAIGHT*1.19;
	_leftSpeed = 0;
	setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
	setMotorDuty(LEFT_MOTOR,500, MOTOR_DIR_BRAKE_gc);
}

void stopTurn(){
	int turnSpeed = MAX-50;
	for(uint16_t j=500;j<turnSpeed;j++){
		setMotorDuty(RIGHT_MOTOR,j,MOTOR_DIR_FORWARD_gc);
		if ( j < 520){
			setMotorDuty(LEFT_MOTOR,j,MOTOR_DIR_NEUTRAL_gc);
		}else{
			setMotorDuty(LEFT_MOTOR,j,MOTOR_DIR_BACKWARD_gc);
		}
		_delay_ms(2);
	}
	_delay_ms(62);
}
```

```
void turn180(){
	_rightSpeed = MAXSTRAIGHT+50;
	_leftSpeed =  MAXSTRAIGHT;
	setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_BACKWARD_gc);
	setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
}

void stop(){
	_leftSpeed = 0;
	_rightSpeed = 0;
	setMotorDuty(RIGHT_MOTOR,500,MOTOR_DIR_BRAKE_gc);
	setMotorDuty(LEFT_MOTOR,500,MOTOR_DIR_BRAKE_gc);
}

void stopHarder(){
	_leftSpeed = 0;
	_rightSpeed =0;
	setMotorDuty(RIGHT_MOTOR,900,MOTOR_DIR_BRAKE_gc);
	setMotorDuty(LEFT_MOTOR,900,MOTOR_DIR_BRAKE_gc);
}

void stop2(){
	_rightSpeed = 0.7*_rightSpeed;
	_leftSpeed = 0.7*_leftSpeed;
```

```
        setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
}

void reverse(){
        for (int x=100; x<MAXSTRAIGHT+20;x++)
        {
                _rightSpeed = x;
                _leftSpeed = x;
                setMotorDuty(RIGHT_MOTOR, _rightSpeed*CR, MOTOR_DIR_BACKWARD_gc);
                setMotorDuty(LEFT_MOTOR, _leftSpeed, MOTOR_DIR_BACKWARD_gc);
        }
}

void drive(){

        _rightSpeed = MAXSTRAIGHT;
        _leftSpeed = MAXSTRAIGHT;
        setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
}


void push(int speed){
        setMotorDuty(RIGHT_MOTOR, speed/2, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(LEFT_MOTOR,speed/2, MOTOR_DIR_FORWARD_gc);
        for (int x =_leftSpeed; x < speed;x++)
        {
                _rightSpeed = speed;
```

```
            _leftSpeed = speed;
            setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
            setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
        }
}
void drive2(){
        // Drive using (k/(k+1))*oldvalue + (1/k)*newvalue;
        _rightSpeed = (0.7*_rightSpeed) + (0.3* MAXSTRAIGHT);
        _leftSpeed = (0.7*_leftSpeed) + (0.3 * MAXSTRAIGHT);
        setMotorDuty(RIGHT_MOTOR, _rightSpeed, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(LEFT_MOTOR,_leftSpeed, MOTOR_DIR_FORWARD_gc);
}

void increaseSpeed(int increase){
        int newLeftSpeed = _leftSpeed+increase;
        if(newLeftSpeed < MAX)
            _leftSpeed = newLeftSpeed;
        else
            _leftSpeed = MAX;
        int newRightSpeed = _rightSpeed+increase;
        if (newRightSpeed < MAX)
            _rightSpeed = newRightSpeed;
        else
            _rightSpeed = MAX;
        setMotorDuty(LEFT_MOTOR, _leftSpeed, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(RIGHT_MOTOR, _rightSpeed*CR, MOTOR_DIR_FORWARD_gc);
}
```

```
void increaseRightMotor(int increase){

	int newRightSpeed = _rightSpeed+increase;

	if (newRightSpeed < MAX)

		_rightSpeed = newRightSpeed;
	else
		_rightSpeed = MAX;

	setMotorDuty(RIGHT_MOTOR,_rightSpeed*CR,MOTOR_DIR_FORWARD_gc);

}


void decreaseRightMotor(int decrease){

	if (_rightSpeed-decrease > 0)

		_rightSpeed = _rightSpeed-decrease;
	else
		_rightSpeed = 0;

	setMotorDuty(RIGHT_MOTOR,_rightSpeed*CR,MOTOR_DIR_FORWARD_gc);
}


void increaseLeftMotor(int increase){

	if(_leftSpeed+increase < MAX)

		_leftSpeed = _leftSpeed + increase;
	else
		_leftSpeed = MAX;

	setMotorDuty(LEFT_MOTOR,_leftSpeed,MOTOR_DIR_FORWARD_gc);

}


void decreaseLeftMotor(int decrease){
	if(_leftSpeed-decrease > 0)
		_leftSpeed = _leftSpeed - decrease;
	else
		_leftSpeed = 0;
	setMotorDuty(LEFT_MOTOR,_leftSpeed,MOTOR_DIR_FORWARD_gc);
}
```

```
void sound(){
        PORTC_OUT;
        //PORTR.OUTCLR = 0x02
}
```