**ELEC 291 Section 20C**

Project 1 Progress Report

L2C

Group 2A and 2B

February 26, 2016

*Student name Student number*

Andy Ruan 36863141

Kevin Wong 32105132

Clarence Su 36387132

Timothy Leung 14189147

Kevin Qiu 14188149

Derek Tam 24630148

**Introduction**

This report outlines the progress that Group 2 has made in Project 1 of ELEC 291-20C from the 11th of February to the 26th and will be organized as follows: First all completed tasks of the principle and additional functionalities will be briefly explained. Following, all current individual member contributions will be listed. Finally documentation of our robot, circuit, and code will be shown in the appendices of this report. Note that all work, including what we consider to be completed work, is subject to change before the due date as implementations may change depending on what we intend to implement in the future.

**Progress**

As of February 26, 2016, our group currently has a functional robot based on the 2WD mobile platform and utilizing a temperature sensor, an ultrasonic sensor mounted on a servo, and three reflective optical sensors mounted to the bottom of the mobile platform. Hall sensor code has been completed however, we are still in the process of finalizing their mounted positions. Our group has the code completed to implement functional movement and line following algorithms. We also have IR remote control capabilities, a smarter movement algorithm and the ability to switch functionalities in the works.

**Principle Functionalities**

The ultrasonic and servo allows the robot to check for obstacles in its direction of movement and scan its surroundings when necessary to determine an available direction of movement. The data is utilized in our PID logic in order to slow the robot down as it approaches obstructions to avoid overshooting the stopping point and crashing. The temperature sensor is used to enhance our calculations of distance from the ultrasonic sensor.

The reflective optical sensors are used to track a piece of black electrical tape on the ground. The contrasting values read from the tape compared to the white ground allow the robot to maneuver itself along the path of the tape.

The hall sensors in combination with the multiple magnets provides a method for the robot to track the position of its wheels. This allows the robot to autonomously correct its pathing by adjusting the motor power provided to each wheel. We plan to use the ISR functionality of the Arduino in order to update this sensor when needed.

**Additional Functionality**

Our group plans to implement two additional functionalities:

1. Smarter obstacle navigation
2. IR remote control functionality

For smarter navigation, our given objective was to traverse the most area as possible; whether that be by navigating around obstacles or turning away and coming back to the obstacle at a different angle, we have not decided on yet. However, we will have the ability to map the area and its obstacles as a matrix and knowledge of our robot’s dimensions. Our current plan is to use this matrix like a chess board and move our robot around the occupied spots. We are considering using a combination of the ultrasonic sensor as well as a bumper switch to achieve this mapping.

For the IR remote control functionality, our goal is to be able to control the robot similar to how drones are controlled. The only additional sensor we will implement is the IR receiver, TSOP38238. We currently have the ability to interface with the Arduino using this receiver through a set of signals generated by the remote controller. These signals will be decoded by the receiver in order to determine which button on the remote is pressed. We plan to relate these keys to specific drive commands.

Future plans include implementing an LCD and switches so that we have the ability to switch between the different functionalities. Due to the amount of extra pins needed for this functionality, we will most likely utilize a second Arduino through I2C interface.

**Member Contributions to date**

Andy Ruan managed and organized the workflow, skeletal code, and documentation. He also worked on the main loop code to drive efficient performance.

Kevin Wong implemented the ability to interface with the Arduino using the IR sensor to control our robot’s movement.

Clarence Su has completed a large amount of research on algorithms to navigate using ultrasonic sensors (see references below) and written partial code for the implementation. [1][2][3]

Timothy Leung wrote the basic drive functions to be used by all our functionalities as well as implementing the line following algorithms for the principle functionalities.

Kevin Qiu implemented the logic for driving the robot straight with the use of the hall sensors as well as the PID logic to reduce the robot’s speed as it approaches an obstacle. He also worked on the majority of the circuit.

Derek Tam contributed to the line following algorithms as well as working on the majority of the hardware on the robot.

**Conclusion**

No major problems have been encountered as we have modularized the work done on this project immensely. Ideally this will continue as we gather our code snippets together however we suspect that this is when most of the debugging will need to happen. We believe that we are on track to finishing this project.

**References**

[1] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=88147&tag=1

[2] http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7270728

[3] http://electronics.howstuffworks.com/gadgets/home/robotic-vacuum2.htm

**Appendix I - Robot and Circuit**

Below is a picture of the current state of our robot and a Fritzing breadboard schematic of the circuit used as of February 26, 2016.

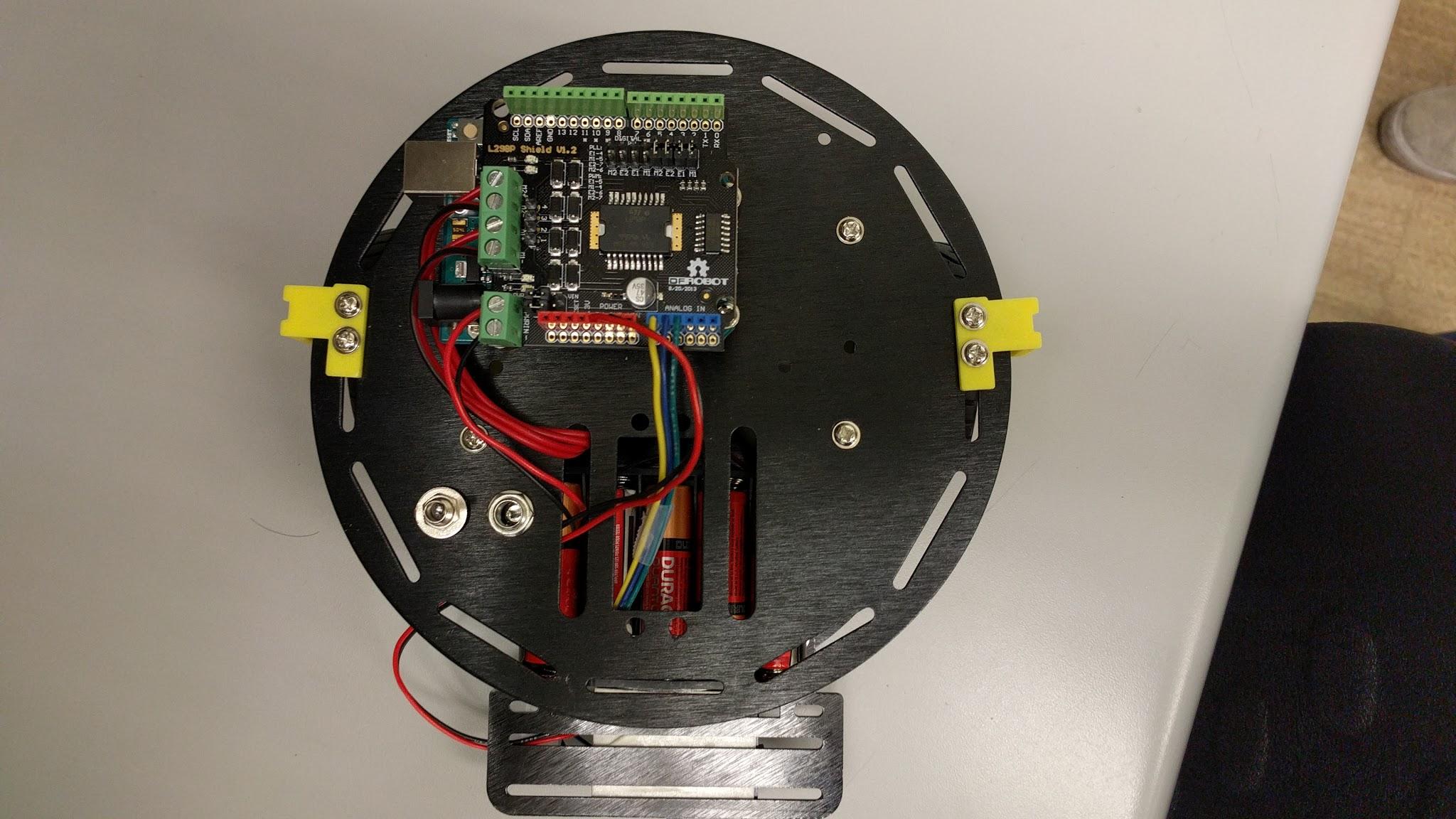


Figure 1: Top view of Group 2’s robot

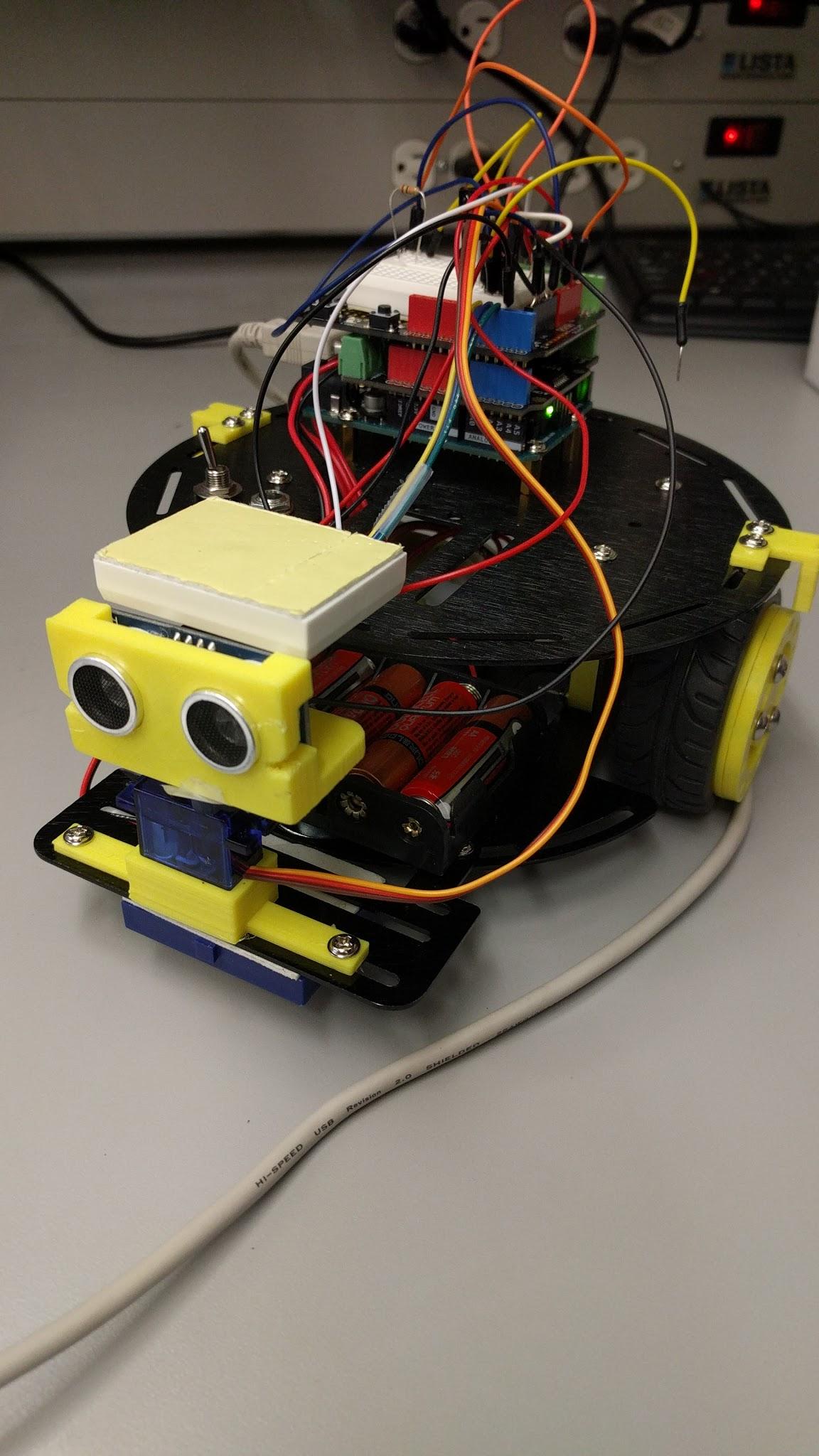


Figure 2: Side angle view

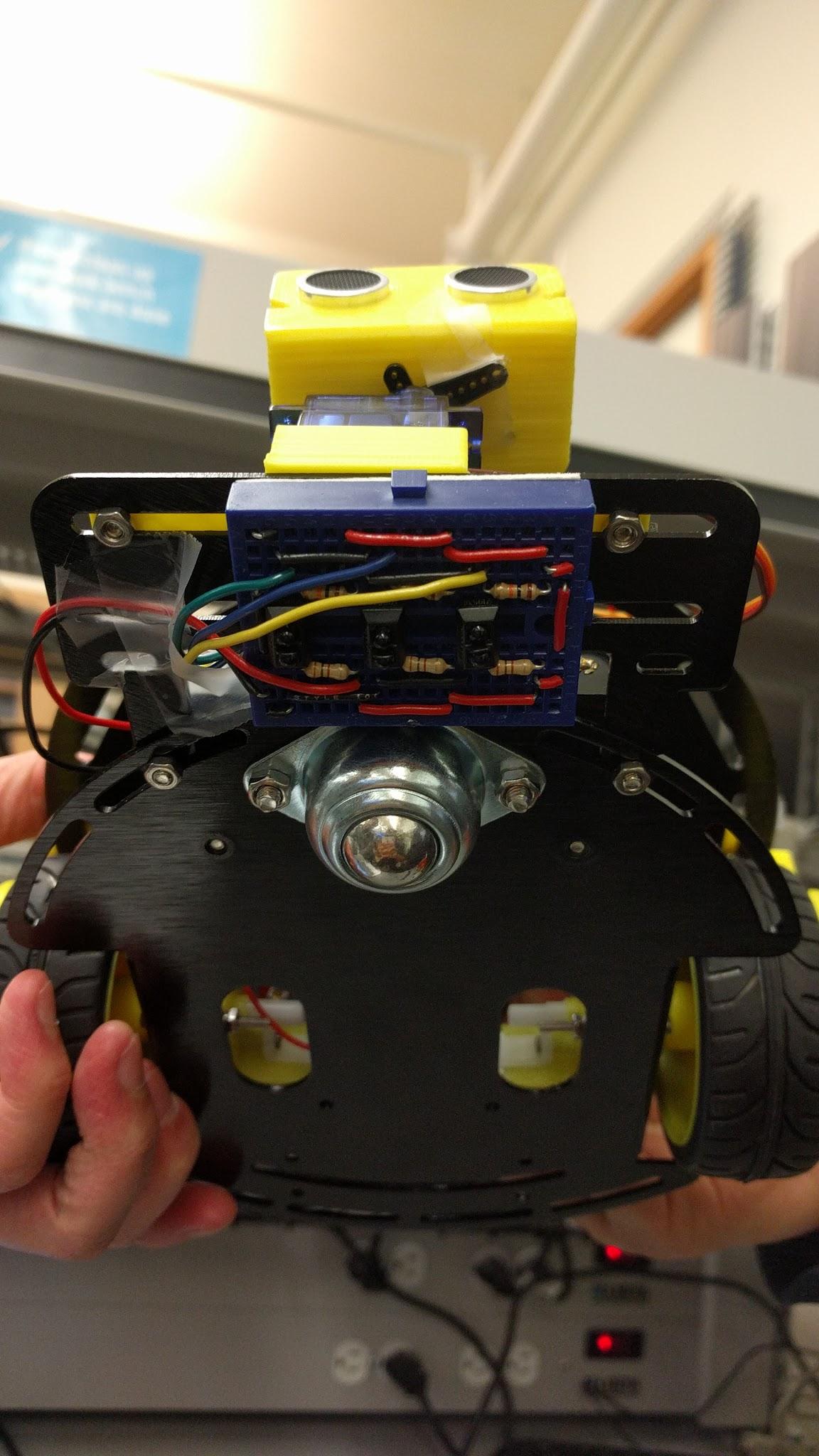


Figure 3: Underside view

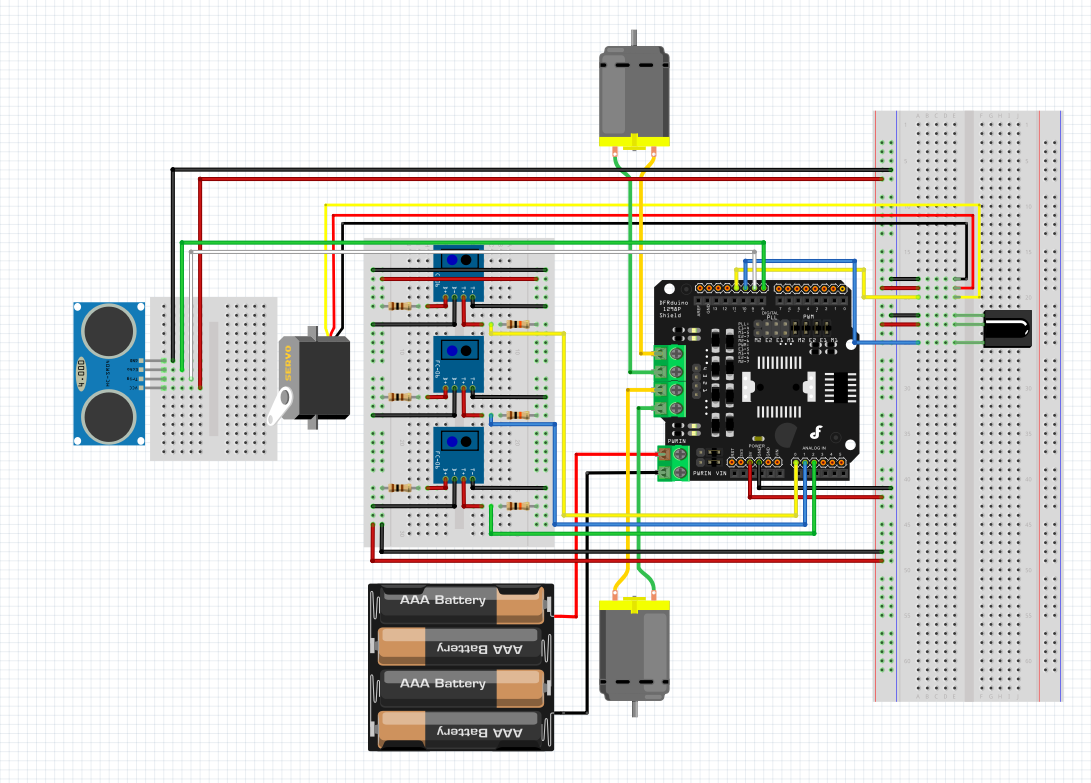


Figure 4: Fritzing schematic of the circuit used in the robot in Figure 1

**Appendix II - Code**

**Fully Functional - Includes the majority of code for the principle functionalities**

#include <Servo.h>

//////////////////////////////////////////////////////

// PIN LAYOUT //

//////////////////////////////////////////////////////

#define LEFT\_DRIVE\_SPEED 5

#define LEFT\_DRIVE 4

#define RIGHT\_DRIVE\_SPEED 6

#define RIGHT\_DRIVE 7

#define LINE\_LEFT A0

#define LINE\_MIDDLE A1

#define LINE\_RIGHT A2

#define SERVO\_PIN 11 // Servo requires PWM pin

Servo servo;

#define TRIG\_PIN 8

#define ECHO\_PIN A3

#define TEMP\_PIN A3

//////////////////////////////////////////////////////

// CONSTANTS //

//////////////////////////////////////////////////////

#define HALL\_LEFT 0

#define HALL\_RIGHT 1

#define FORWARD 0

#define BACKWARD 1

#define RIGHT 0

#define LEFT 1

#define scanLeft 180

#define scanRight 0

#define centerServo 90

/\* Conversion constant from analogRead() output to temperature in C\*

Derived from LM35 temperature sensor datasheet

Voltage vs. Temperature 0-1500mV <--> 0-150C \*/

#define CONVERT\_TO\_TEMP 1/1024.0\*500

// Define motor speed control and distance constants

#define DIST\_THRESH 7.0

#define MAP\_CONST 60

#define MIN\_MOVEMENT\_POWER 70

#define MAX\_SPEED 255

#define LINE\_THRESHOLD 100

void setup()

{

pinMode(LEFT\_DRIVE, OUTPUT);

pinMode(RIGHT\_DRIVE, OUTPUT);

pinMode(LEFT\_DRIVE\_SPEED, OUTPUT);

pinMode(RIGHT\_DRIVE\_SPEED, OUTPUT);

pinMode(TRIG\_PIN, OUTPUT);

pinMode(ECHO\_PIN, INPUT);

servo.attach(SERVO\_PIN);

Serial.begin(9600);

delay(100);

}

void loop()

{

driveAndScan();

//lineFollow();

//lineTest();

}

//////////////////////////////////////////////////////

// DRIVE FUNCTIONS //

//////////////////////////////////////////////////////

void loopDriveStraight()

{

updateEncoders();

drive\_P\_Straight(FORWARD, 200);

}

// Call to update tick value, must be called

// as soon as possible in order to not miss ticks

int L\_Encoder = 0;

bool left\_lastState = 0;

int R\_Encoder = 0;

bool right\_lastState = 0;

void updateEncoders()

{

if (digitalRead(HALL\_LEFT) != left\_lastState)

{

L\_Encoder++;

left\_lastState = !left\_lastState;

}

if (digitalRead(HALL\_RIGHT) != right\_lastState)

{

R\_Encoder++;

right\_lastState = !right\_lastState;

}

}

// Uses P to lower/increase motor speed to match the rotation speed to drive straight

// Needs to be called in a while loop, does not stop the motors

void drive\_P\_Straight(int direction, int speed)

{

digitalWrite(LEFT\_DRIVE, direction ? LOW : HIGH);

digitalWrite(RIGHT\_DRIVE, direction ? HIGH : LOW);

// Calulate error

int Kp = 1.5;

int error = R\_Encoder - L\_Encoder \* Kp;

// Set motor speed

analogWrite(LEFT\_DRIVE\_SPEED, speed + error);

analogWrite(RIGHT\_DRIVE\_SPEED, speed - error);

}

// Drive without acceleration or decleration, Speed 0-255, timed

// Sets the motors, waits and then stops the motors

void drive\_Timed(int direction, int speed, int time)

{

// Set direction

digitalWrite(LEFT\_DRIVE, direction ? LOW : HIGH);

digitalWrite(RIGHT\_DRIVE, direction ? HIGH : LOW);

// Set motor speed

analogWrite(LEFT\_DRIVE\_SPEED, speed);

analogWrite(RIGHT\_DRIVE\_SPEED, speed);

delay(time);

// Stop motors

analogWrite(LEFT\_DRIVE\_SPEED, 0);

analogWrite(RIGHT\_DRIVE\_SPEED, 0);

}

void turn\_Timed(int direction, int speed, int time)

{

// Set direction

digitalWrite(LEFT\_DRIVE, ~direction);

digitalWrite(RIGHT\_DRIVE, ~direction);

// Set motor speed

analogWrite(LEFT\_DRIVE\_SPEED, speed);

analogWrite(RIGHT\_DRIVE\_SPEED, speed);

delay(time);

7

// Stop motors

analogWrite(LEFT\_DRIVE\_SPEED, 0);

analogWrite(RIGHT\_DRIVE\_SPEED, 0);

}

//////////////////////////////////////////////////////

// PRINCIPLE FUNCTION 1 //

//////////////////////////////////////////////////////

/\*

\* Drive the robot up to an object up to a predetermined threshold.

\* Once the robot reaches the distance threshold,

\* the robot will scan its left and right side

\* and select a direction to turn to.

\*/

void driveAndScan()

{

// Get ultrasonic distance

double distance = getUltrasonic();

// Set baseline speed

double motorSpeed = MAX\_SPEED;

// Loop until robot reaches distance threshold

while(distance > DIST\_THRESH)

{

// Update distance

distance = getUltrasonic();

// Update motor value only if ultrasonic returns valid value

if(distance <= 500)

motorSpeed = getMotorSpeed(distance);

// Set to drive forward

digitalWrite(LEFT\_DRIVE, HIGH);

digitalWrite(RIGHT\_DRIVE, LOW);

// Set motor speed

analogWrite(LEFT\_DRIVE\_SPEED, motorSpeed);

analogWrite(RIGHT\_DRIVE\_SPEED, motorSpeed);

}

// Find a direction to turn to, if scanSurroundings returns 2, scan again

int turnDir;

do

{

turnDir = scanSurroundings();

} while(turnDir == 2);

delay(1000);

// Turn left or right based on clearance of each side

turn\_Timed(turnDir ? LEFT : RIGHT, 150, 600);

delay(2000);

}

/\*

\* Calculate the motor speed based on

\* the measured ultrasonic sensor value and

\* the distance an object is away from the robot.

\* Param: distance - measured distance from ultrasonic

\* Return: motorVal - updated motor speed

\*/

int getMotorSpeed(double distance)

{

// Set motor value to MAX\_SPEED

int motorVal = MAX\_SPEED;

// Check if robot has not approached object

if(distance > DIST\_THRESH)

{

// Calculate new motor value based on distance of object using map() function

motorVal = map(distance, DIST\_THRESH, MAP\_CONST, MIN\_MOVEMENT\_POWER, MAX\_SPEED);

}

else

{

motorVal = 0;

}

return motorVal;

}

/\*

\* Check the left and right side of the robot

\* using the ultrasonic sensor on a servo mount.

\* Determine which direction to turn based on

\* the further of the two distances.

\* Returns: 0 - to turn the robot left

\* 1 - to turn the robot right

\* 2 - scan again

\*/

int scanSurroundings()

{

// Set the servo to scan left side

servo.write(scanLeft);

delay(1500);

// Get distance of object on the left side

double distanceLeft = getUltrasonic();

delay(250);

// Set the servo to scan right side

servo.write(scanRight);

delay(1500);

// Get distance of object on the right side

double distanceRight = getUltrasonic();

delay(250);

// Reset servo to face forward

servo.write(centerServo);

// If left and right distances are garbage, tell main loop to scan again

if(distanceLeft > 500 && distanceRight > 500)

return 2;

// Determine which direction to turn in, 0 = right, 1 = left

return distanceLeft > distanceRight;

}

// Return distance in centimetres

double getUltrasonic()

{

//double temp = analogRead(tempPin) \* CONVERT\_TO\_TEMP;

double temp = 21; // Standard temp in celsius

double speedOfSound = (331.5 + (0.6 \* temp)) \* 0.0001; // cm/microsecond

digitalWrite(TRIG\_PIN, HIGH);

delayMicroseconds(10);

digitalWrite(TRIG\_PIN, LOW);

return (pulseIn(ECHO\_PIN, HIGH) / 2) \* speedOfSound;

}

//////////////////////////////////////////////////////

// PRINCIPLE FUNCTION 2 //

//////////////////////////////////////////////////////

/\*

\* Send the robot to trace a line. Each optical sensor's value

\* is read and the speed of each wheel is determined based on

\* which optical sensors detect the line.

\*/

void lineFollow()

{

int speed = 255;

int L\_LINE = analogRead(LINE\_LEFT);

int M\_LINE = analogRead(LINE\_MIDDLE);

int R\_LINE = analogRead(LINE\_RIGHT);

if (R\_LINE > LINE\_THRESHOLD)

{

// Set direction

digitalWrite(LEFT\_DRIVE, HIGH);

digitalWrite(RIGHT\_DRIVE, LOW);

// Set motor speed (Right)

analogWrite(LEFT\_DRIVE\_SPEED, speed);

analogWrite(RIGHT\_DRIVE\_SPEED, 0);

}

else if (L\_LINE > LINE\_THRESHOLD)

{

// Set direction

digitalWrite(LEFT\_DRIVE, HIGH);

digitalWrite(RIGHT\_DRIVE, LOW);

// Set motor speed (Right)

analogWrite(LEFT\_DRIVE\_SPEED, 0);

analogWrite(RIGHT\_DRIVE\_SPEED, speed);

}

else if (R\_LINE > LINE\_THRESHOLD && M\_LINE > LINE\_THRESHOLD)

{

// Set direction

digitalWrite(LEFT\_DRIVE, HIGH);

digitalWrite(RIGHT\_DRIVE, LOW);

// Set motor speed (Right)

analogWrite(LEFT\_DRIVE\_SPEED, speed - 75);

analogWrite(RIGHT\_DRIVE\_SPEED, 0);

}

else if (L\_LINE > LINE\_THRESHOLD && M\_LINE > LINE\_THRESHOLD)

{

// Set direction

digitalWrite(LEFT\_DRIVE, HIGH);

digitalWrite(RIGHT\_DRIVE, LOW);

// Set motor speed (Right)

analogWrite(LEFT\_DRIVE\_SPEED, 0);

analogWrite(RIGHT\_DRIVE\_SPEED, speed - 75);

}

else if(M\_LINE > LINE\_THRESHOLD)

{

// Set direction

digitalWrite(LEFT\_DRIVE, HIGH);

digitalWrite(RIGHT\_DRIVE, LOW);

// Set motor speed

analogWrite(LEFT\_DRIVE\_SPEED, speed);

analogWrite(RIGHT\_DRIVE\_SPEED, speed);

}

}

void lineTest()

{

int leftVal = analogRead(LINE\_LEFT);

int midVal = analogRead(LINE\_MIDDLE);

Serial.print(leftVal);

Serial.print(" ");

int rightVal = analogRead(LINE\_RIGHT);

// LEFT < LINE\_THRESHOLD? 1 : 0;

// MIDDLE = MIDDLE < LINE\_THRESHOLD? 1 : 0;

// RIGHT = RIGHT < LINE\_THRESHOLD? 1 : 0;

Serial.print(midVal);

Serial.print(" ");

Serial.println(rightVal);

}

**Partially Implemented - Includes some test code and code that has not been merged**

**IR Code**

#define RECV\_PIN 10

IRrecv irrecv(RECV\_PIN);

String in;

decode\_results results;

void loop()

{

detectIR();

switch(IR\_Instruction)

{

case 0: // Stop driving

drive\_P\_Straight(0,0);

Serial.println("Stopped");

break;

case 1: // Robot is driving

drive\_P\_Straight(0,200);

Serial.println("Drive Straight");

break;

}

}

void detectIR() {

if (irrecv.decode(&results)) {

in = String(int(results.value), HEX);

irrecv.resume(); // Receive the next value

if (in == "a05f") // Pressed on keypad #2

{

IR\_Instruction = 1; // Drive straight

}

if (in == "906f") // Pressed on keypad #5

{

drive\_P\_Straight(1,150); // Drive backwards

}

if (in == "10ef") //Pressed on keypad #4 {

Serial.println("4");

IR\_Instruction = 0; // Stop the movement of the robot.

}

if (in == "50af") //Pressed on keypad #6{

Serial.println("6");

turn\_Timed(RIGHT, 150, 600); // Code to turn right. Same as the code Drive Functions tab

}

}

delay(100);

}

**Mode Select and Hall ISR (I2C Master)**

#include <LiquidCrystal.h>

#include <IRremote.h>

#include <Wire.h>

// pin assignments

const int ULTRASONIC\_PIN1 = 4;

const int ULTRASONIC\_PIN2 = 3;

const int LINESENSL\_PIN = A0;

const int LINESENSR\_PIN = A1;

const int LINESENSM\_PIN = A2;

const int IRRECV\_PIN = 11;

const int HALLL\_PIN = 2;

const int HALLR\_PIN = 3;

const int MODEBTN\_PIN = 12;

const int MODEBTNUP\_PIN = 10;

const int MODEBTNDOWN\_PIN = 9;

int ultrasonicDistance;

unsigned long prevUltrasonicMilli = 0;

const long ULTRASONIC\_INTERVAL = 250;

int hallLCount = 0;

int hallRCount = 0;

IRrecv irrecv(IRRECV\_PIN);

decode\_results IRVal;

int mode = 1;

int modeBtnState;

int prevModeBtnState = LOW;

unsigned long lastMBDebounceTime = 0;

int modeBtnUpState;

int prevModeBtnUpState = LOW;

unsigned long lastMBUDebounceTime = 0;

int modeBtnDownState;

int prevModeBtnDownState = LOW;

unsigned long lastMBDDebounceTime = 0;

long debounceDelay = 50;

void setup() {

Serial.begin(9600);

irrecv.enableIRIn();

attachInterrupt(digitalPinToInterrupt(HALLL\_PIN), updateHallL, RISING);

attachInterrupt(digitalPinToInterrupt(HALLR\_PIN), updateHallR, RISING);

attachInterrupt(digitalPinToInterrupt(HALLL\_PIN), updateHallL, FALLING);

attachInterrupt(digitalPinToInterrupt(HALLR\_PIN), updateHallR, FALLING);

Wire.begin();

}

void loop() {

selectMode(); // choose mode

switch ( mode ) {

case 1:

while ( debounceModeButton() == HIGH ) {

//lineFollow();

}

break;

case 2:

while ( debounceModeButton() == HIGH ) {

int distance = getUltrasonic();

}

break;

case 3:

while ( debounceModeButton() == HIGH) {

// clarence loop

}

break;

case 4:

while ( debounceModeButton() == HIGH) {

// kwong loop

}

break;

default:

selectMode();

break;

}

}

void selectMode() {

sendToSlave(0);

byte dispCode = 0;

while( modeBtnState == LOW ) {

debounceAllButtons();

if ( modeBtnDownState = HIGH ) {

if( dispCode >= 4 )

dispCode = 1;

else

dispCode++;

}

else if( modeBtnUpState == HIGH ) {

if ( dispCode <= 1 )

dispCode = 4;

else

dispCode--;

}

sendToSlave(dispCode);

delay(250);

}

mode = dispCode;

}

void sendToSlave(byte x) {

Wire.beginTransmission(8);

Wire.write(x);

Wire.endTransmission();

}

void debounceAllButtons() {

debounceModeButton();

debounceModeDownButton();

debounceModeDownButton();

}

int debounceModeButton() {

int reading = digitalRead(MODEBTN\_PIN);

if ( reading != prevModeBtnState ) {

lastMBDebounceTime = millis();

}

if ( (millis() - lastMBDebounceTime) >= debounceDelay ) {

if ( reading != modeBtnState )

modeBtnState = reading;

}

prevModeBtnState = reading;

return modeBtnState;

}

int debounceModeUpButton() {

int reading = digitalRead(MODEBTNUP\_PIN);

if ( reading != prevModeBtnUpState ) {

lastMBUDebounceTime = millis();

}

if ( (millis() - lastMBUDebounceTime) >= debounceDelay ) {

if ( reading != modeBtnState )

modeBtnUpState = reading;

}

prevModeBtnUpState = reading;

return modeBtnUpState;

}

int debounceModeDownButton() {

int reading = digitalRead(MODEBTNDOWN\_PIN);

if ( reading != prevModeBtnDownState ) {

lastMBDDebounceTime = millis();

}

if ( (millis() - lastMBDDebounceTime) >= debounceDelay ) {

if ( reading != modeBtnDownState )

modeBtnDownState = reading;

}

prevModeBtnDownState = reading;

return modeBtnDownState;

}

int getUltrasonic() {

unsigned long currMilli = millis();

if ( currMilli - prevUltrasonicMilli >= ULTRASONIC\_INTERVAL ) {

prevUltrasonicMilli = currMilli;

//ultrasonicDistance = getUltrasonic(ULTRASONIC\_PIN1, ULTRASONIC\_PIN2);

}

return ultrasonicDistance;

}

void updateHallL() {

hallLCount++;

}

void updateHallR() {

hallRCount++;

}

**Mode Select (I2C Slave)**

#include <LiquidCrystal.h>

#include <Wire.h>

const int LCDRS\_PIN = 2;

const int LCDE\_PIN = 3;

const int LCDD0\_PIN = 4;

const int LCDD1\_PIN = 5;

const int LCDD2\_PIN = 6;

const int LCDD3\_PIN = 7;

const int LCDD4\_PIN = 8;

const int LCDD5\_PIN = 9;

const int LCDD6\_PIN = 10;

const int LCDD7\_PIN = 11;

LiquidCrystal lcd(LCDRS\_PIN, LCDE\_PIN, LCDD0\_PIN, LCDD1\_PIN, LCDD2\_PIN, LCDD3\_PIN, LCDD4\_PIN, LCDD5\_PIN, LCDD6\_PIN, LCDD7\_PIN);

byte upArrow[8] = {

0b00100,

0b01110,

0b11111,

0b00100,

0b00100,

0b00100,

0b00100,

0b00000

};

byte downArrow[8] = {

0b00100,

0b00100,

0b00100,

0b00100,

0b11111,

0b01110,

0b00100,

0b00000

};

void setup() {

Wire.begin(8);

Wire.onReceive(receiveEvent);

Serial.begin(9600);

lcd.createChar(0, upArrow);

lcd.createChar(1, downArrow);

}

void loop() {

}

void receiveEvent(int howMany) {

int x = Wire.read();

lcd.clear();

switch(x) {

case 0:

lcd.print("Select mode");

lcd.setCursor(0, 1);

lcd.print("using ");

lcd.write(byte(0));

lcd.print(" and ");

lcd.write(byte(1));

break;

case 1:

lcd.print("Principle 1:");

lcd.setCursor(0, 1);

lcd.print("Drive and Scan");

break;

case 2:

lcd.print("Principle 2:");

lcd.setCursor(0, 1);

lcd.print("Line Following");

break;

case 3:

lcd.print("Additional 1:");

lcd.setCursor(0, 1);

lcd.print("Smarter Movement");

break;

case 4:

lcd.print("Additional 2:");

lcd.setCursor(0, 1);

lcd.print("IR RC");

break;

default:

lcd.print("Select mode");

lcd.setCursor(0, 1);

lcd.print("using ");

lcd.write(byte(0));

lcd.print(" and ");

lcd.write(byte(1));

break;

}

}