5/15/2015

# Room Mapping

*ELE 302 Independent Project*

Yicheng Sun and Edgar Wang

# Room Mapping

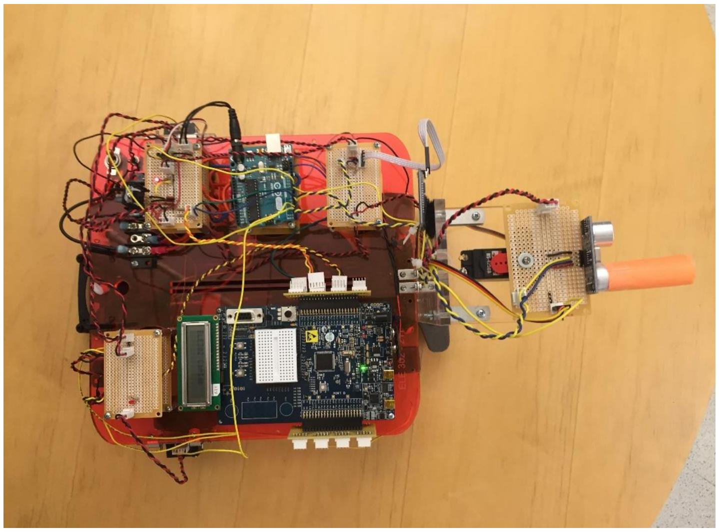
*ELE 302 Independent Project*

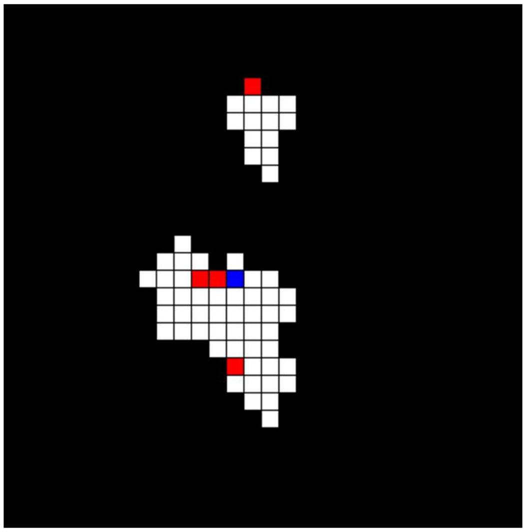
For our independent project, we built a car that constructs a two dimensional floor map as it travels. Hardware wise, this was done with the following additions:

- Ultrasonic sensor to detect obstacles
- Rotary encoders for differential odometry
- Arduino for control and calculations
- Bluetooth module for remote control and data sending
- H-bridge for backwards driving.

In software, we wrote code for the following:

- Position tracking with error correction
- Obstacle detection with error correction
- Laptop controlled remote navigation control
- Data processing with python
- Visualization with Processing





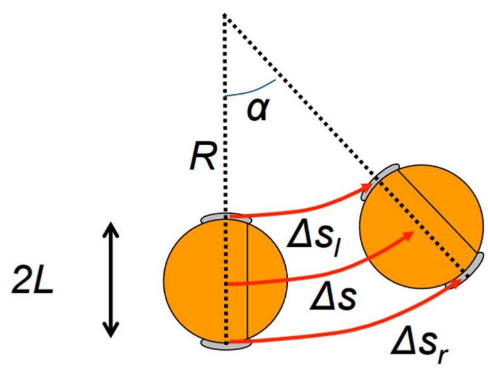
# Position Tracking

Our goal was to build a precise tracking system for the car while it maneuvered across the room. Thus we considered multiple methods:

1. Accelerometer and gyroscope combination
2. Differential odometry
3. Ultrasonic beacons for positioning

We wished to create self-contained system and so option three was not viable. In pursuing option 1, we bought an inertial measurement unit (LSM9DS0 Adafruit 9 degrees of freedom accelerometer, gyroscope, magnometer, temperature board). However, through testing, we found the positioning data to be unreliable. Because of the nature of the IMU, the positioning error accumulated from double integrations of accelerations was more than what was tolerable. We required the position tracking to be very precise as our planned grid unit was 1 foot by 1 foot. Our goal was to have position tracking to be under half of feet of discrepancy after 30 feet of driving.

Therefore, we settled on implementing differential odometry. Using differential odometry relies upon taking the difference in number of wheel rotations to find the angle the car is traveling at and the distance the car has traveled. The following diagrams represent the calculations necessary to derive the car's position and angle of travel based off the number of wheel rotations.




$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$$

$$\Delta\theta = \frac{(\Delta s_r - \Delta s_l)}{2L}$$

$$\Delta x = \Delta d \cos(\theta + \Delta\theta/2)$$
$$\Delta y = \Delta d \sin(\theta + \Delta\theta/2)$$

**Initial Hall Effect Approach**

We initially tried performing differential odometry using A1101EUA-T Hall Effect sensors to detect for 5 equidistant magnets on each wheel with interrupt triggers for the magnets passing by. We connected an Arduino to handle the interrupts. However, with only 5 triggers on each wheel, just a small difference of 1 extra magnet rotation for the left over the right wheel led to a 15 degree change in angle—leaving a lot to be desired for resolution. Furthermore, after testing the car for 10 feet straightaways, the left and right wheels never read the same number of magnet rotations due to slippage and friction. If the car was driving in a straight line and some slippage occurred between the wheels, the positioning calculations would incorrectly say the car had changed its travel direction angle.

**Error Correction**

In order to correct for some of these inconsistencies, we relied upon the angle the steering servos were positioned at. If the steering servos were positioned for 90 degrees and the differential odometry calculations stated the car had moved +10 degrees difference away, the calculations were thrown out. We also checked for right & left turns. If the steering servo was being driven at 60 degrees (left turn) and the differential odometry calculations stated travel in the opposite directions above +10 degrees (right), then the data point was thrown out. These checks improved the odometry accuracy by not by enough.

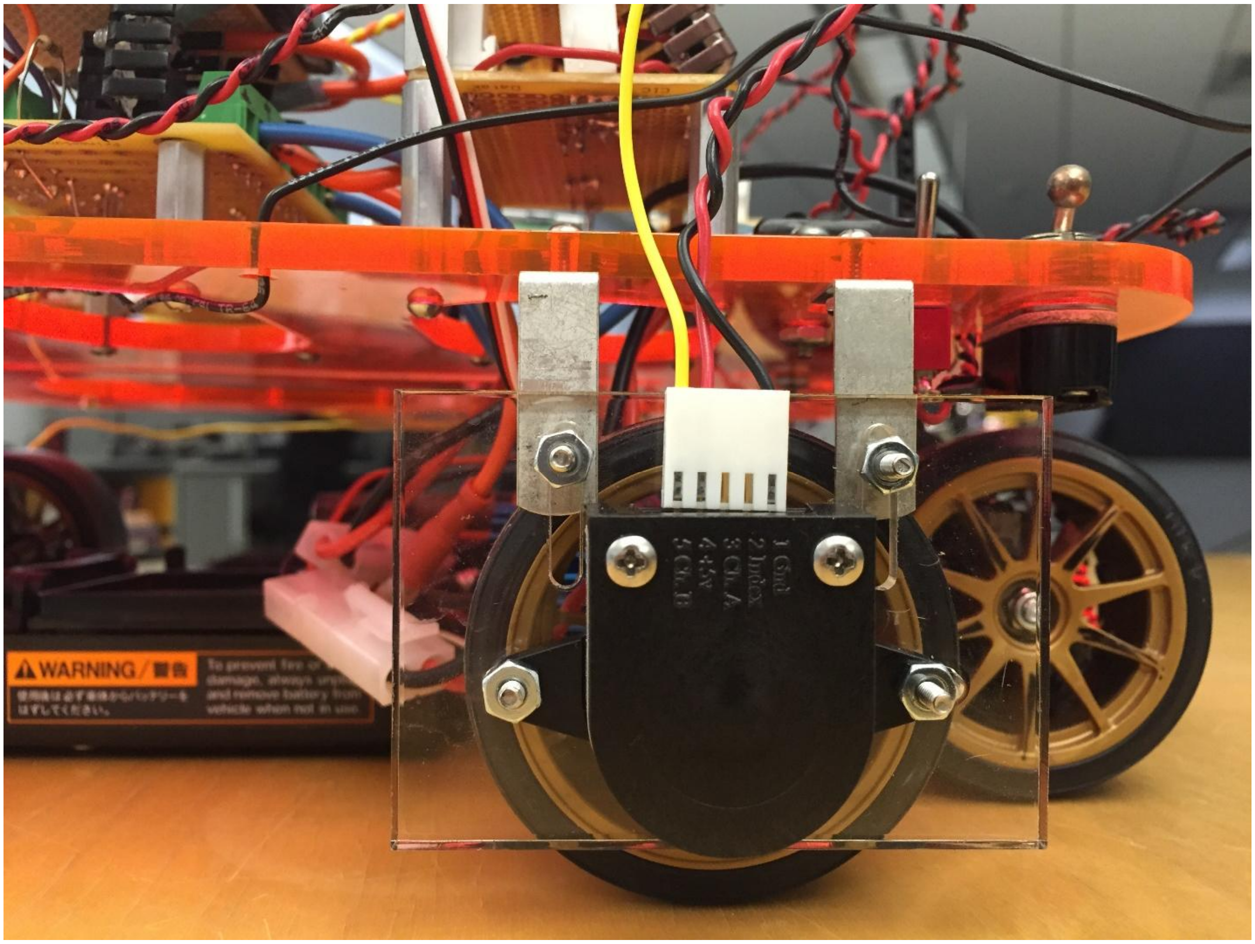
Next, we tried increasing the magnet count on each wheel from five to ten. This doubled the resolution as each magnet difference was 7.5 degrees change in car angle, but even this wasn't enough. After 10 feet of driving with turns, the position was already off by more than half a foot.

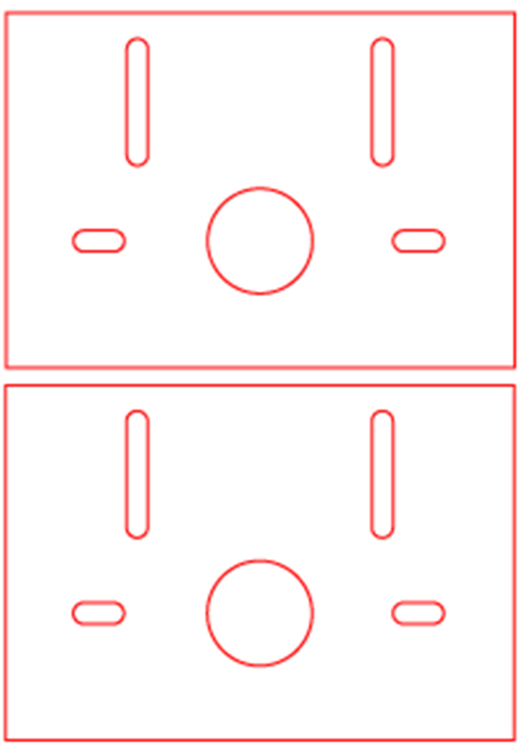
**Final Solution**

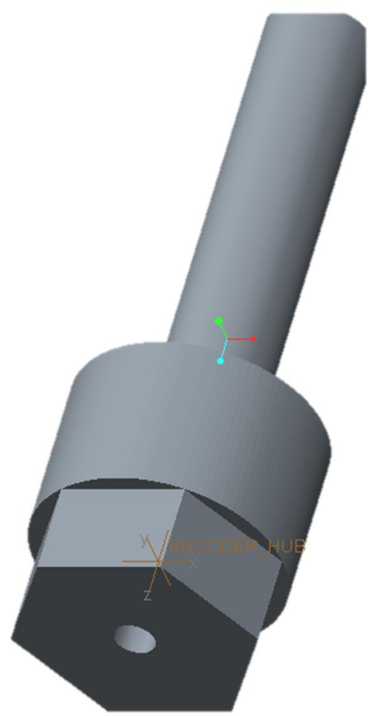
Because we were confident our calculations were correct, we came to the conclusion that the hardware lacked the resolution needed to provide accurate position tracking. Thus, we turned to purchasing a rotary encoder. We decided that an US Digital optical incremental encoder with 32 cycles per revolution (which is vastly superior to the 10 cycles per revolution our hall effect provided) would be sufficient. We had the choice of higher cycles per revolution but ultimately decided that the Arduino may not be able to process the position updating interrupt at the required frequency.

Since we wouldn't be able to attach rotary encoders to the original back wheels, we resorted to adding on an extra pair of wheels to the car and attaching the rotary encoders to these extra wheels. These wheels will act similarly to training wheel, freely rotating, and will make 1:1 rotations with the back wheels. Using laser cut overhangs, we were able to comfortably fix two wheels with rotary encoders to the edges of the car body. These wheels were attached to the encoders with a 3D printed wheel hub and shaft piece designed with PTC Creo. Pictures of the overhang, shaft, and wheel assembly can be seen on the following page.

Because the rotary encoders had 32 clicks per wheel rotation, our calculations for angle resolution improved to 2.56 degrees, substantially better than the previous 7.5 degrees. This allowed our calculations to model turns much more accurately, and we were now able to get within .5 ft accuracy after driving the car 30ft. In fact, after testing, our car was accurate up to a quarter of a foot, performing beyond expectations.


WARNING

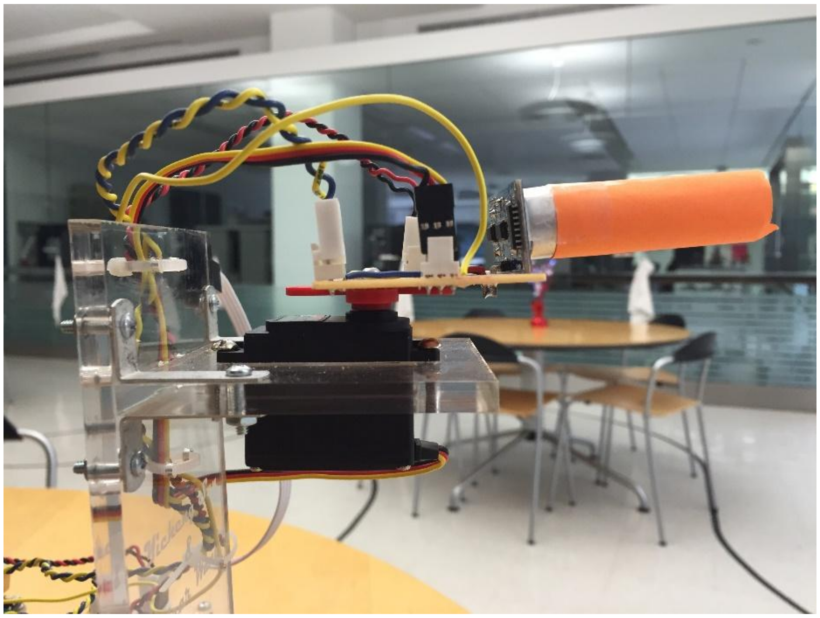


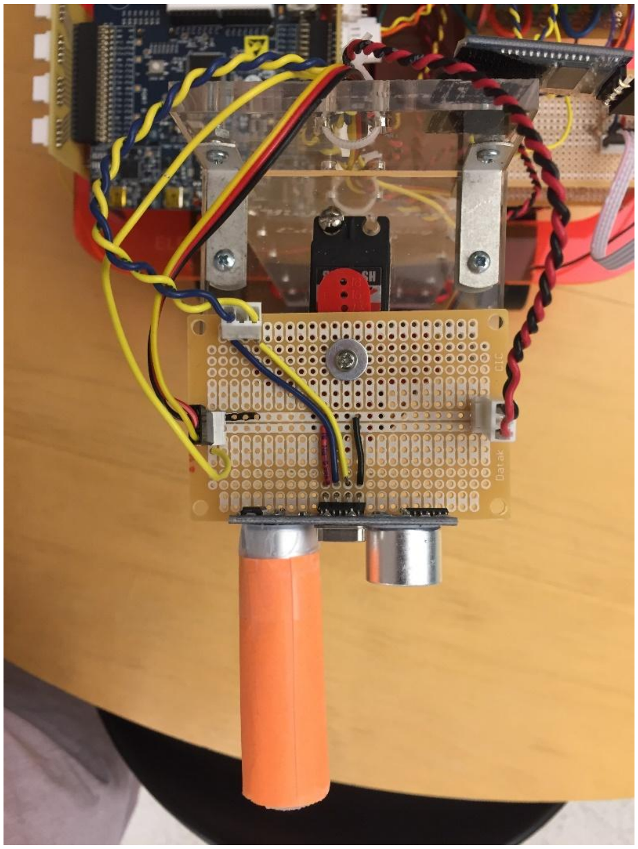


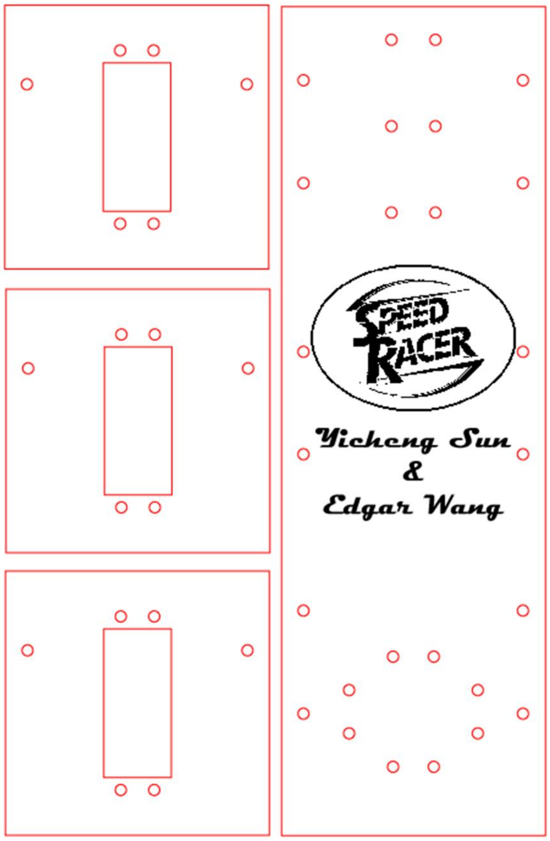
## Ultrasonic Obstacle Detection

In order to detect obstacles from around the room, we used a SunFounder HC-SR04 Ultrasonic Sensor. Mounting the sensor required creating a new mast (the original line following camera was removed) and designing servo mounts.

We allowed ourselves the possibility to mount multiple servos along the mast. Had time permitted, we would have had three ultrasonic servos mounted to create a low resolution 3D mapping of the room. Such as it is, we only had the time to do 2D mapping with one servo.





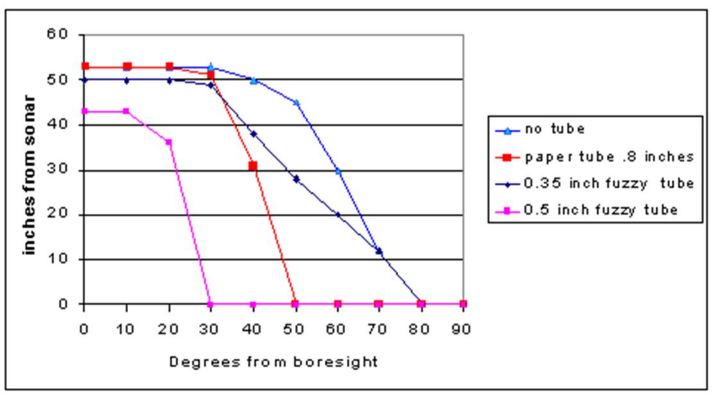


We wired up the ultrasonic sensor to a board which was mounted on a servo with 90 degrees (straight forward) being the resting position. Our code instructed the servo to oscillate from 60 degrees to 120 degrees in 15 degree increments. At every 15 degrees the ultrasonic sensor would conduct a reading to determine whether or not an obstacle was within 7-150 cm ahead, and the Arduino would send this and the servo angle, along with the car's position, to the computer. The data sent through Bluetooth was arranged as such:

X_POS    Y_POS    SERVO_READING_ANGLE    DISTANCE_OF_OBJECT_READ

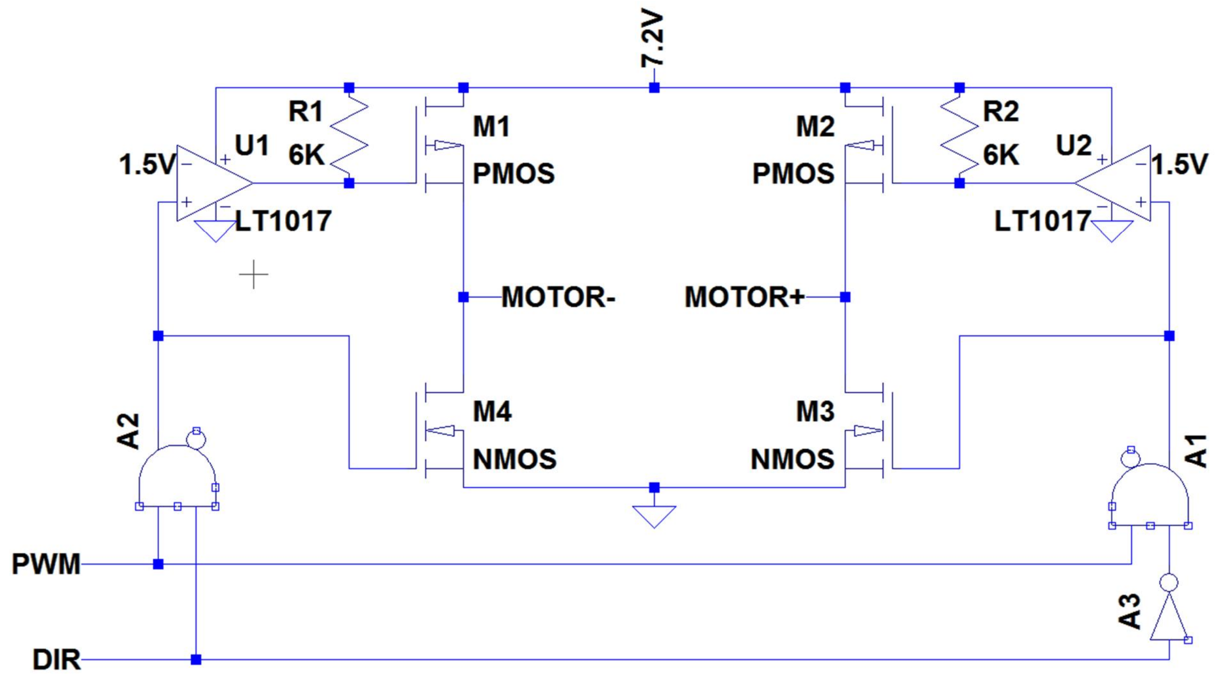
**Reducing Error**

Unfortunately, after testing, the ultrasonic sensor didn't read obstacles within the range advertised. We determined the actual range was within 7cm and 150cm, so we added a software check to validate readings within these measurements. Furthermore, the measurement cone of 30 degrees for the sensor was a too wide to provide the resolution we needed for obstacle positioning. Previous experiments have shown that adding a velvet cylinder will reduce the gain cuts down the sidelobes of the sensor.[1] However, due to a lack of velvet, we settled for adding a paper cylinder.. This led to a more precise measurement angle cone of approximately 15 degrees and allowed us to more accurately map the room, in particular we could now map gaps in between two obstacles without having the car drive up close to the gap.

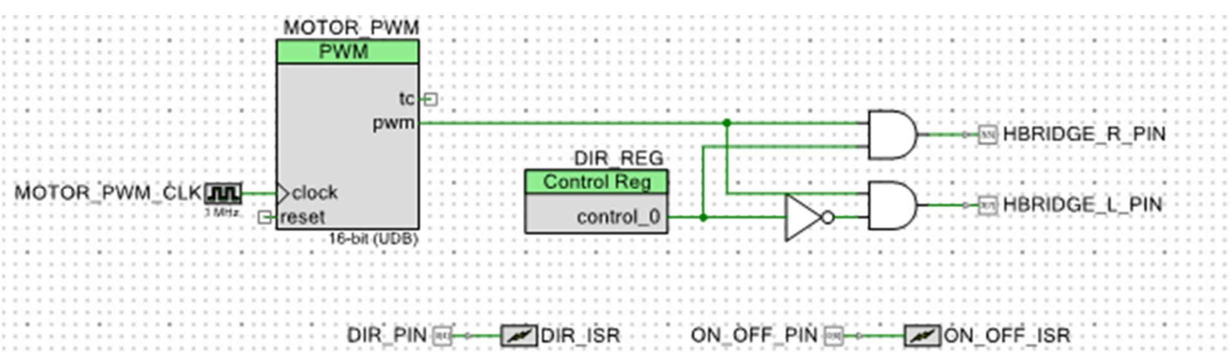



[1] http://www.robot-electronics.co.uk/htm/reducing_sidelobes_of_srf10.htm

## H-Bridge



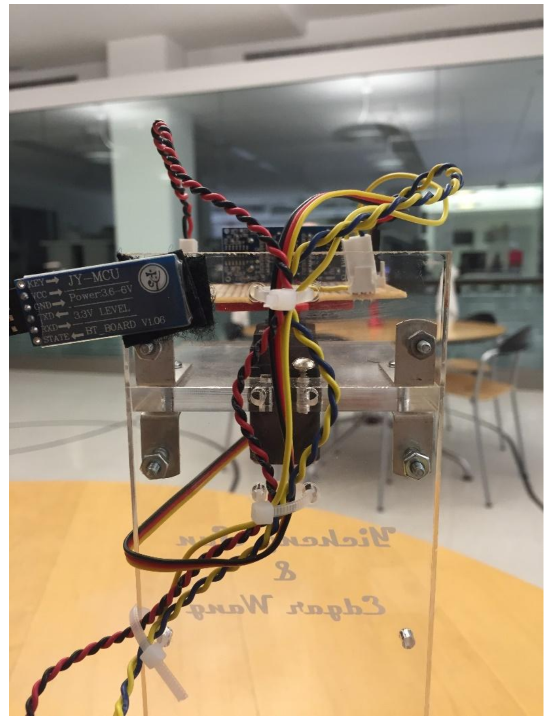

As the car will invariably need to scan areas that will be dead ends that prevent the car from going forward, we decided to build an H-bridge to allow the motor to drive the other direction. For the H-bridge we used two P-channel MOSFETS (FQP27P06) and two N-channel mosfets (MTP75NO3HDL). The four MOSFETS were hooked up in such a way such that depending on whether input DIR was high or low, input PWM voltage will be applied to the positive or negative end of the motor, with the other motor end grounded. Because Arduino is unable to output a PWM voltage high enough to power the motor sufficiently, we a dual comparator to pull up high PWM signals to the motor battery.




The circuit was also partly designed in the programmable hardware of the PSOC. The Arduino controls whether the motor is turned on/off and going forward/backward through digital output pins that are read by PSOC interrupts. The PSOC then changes the PWM and DIR accordingly. PSOC code can be found in **Appendix D.** Our position tracking code also accounted for driving backwards, see **Appendix A.**

# Arduino-Bluetooth Navigation

Our car has the ability to drive right, left, forward, and backward with Bluetooth control. We purchased a KEDSUM JY-MCU Arduino Wireless Bluetooth Transceiver Module. With laptop user input and connecting to the module with PuTTY, the Bluetooth module sends a byte to the Arduino and the Arduino handles the data accordingly.




'a': turns front wheels 12 degrees to the left

'd': turns front wheels 12 degrees to the right

'w': makes motor go forward/backward alternating

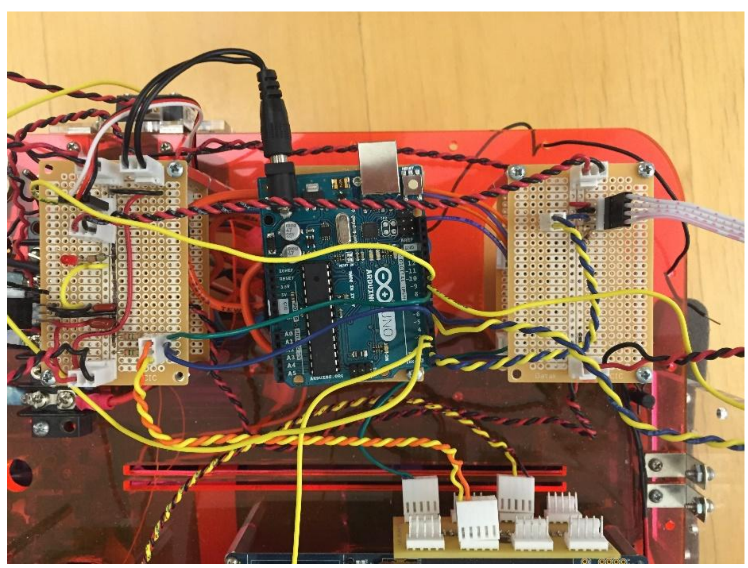
'o': turns motor on

‘s’: scan the surrounding with ultrasonic sensor servo

‘p’: prints out current location

‘s’: resets steering and ultrasonic servos to centered position and resets position tracking

We used an Arduino Uno to send and receive data through Bluetooth. In addition, the Arduino controlled the steering wheel servos, the ultrasonic servo position, and updated its position through the wheel encoders. The code for the Arduino can be found in **Appendix A.**



# Visualization

Every time the car scanned the room with the ultrasonic sensor, it would send back data to the computer in the previously described format under "Ultrasonic Obstacle Detection". We wrote up a python script to process this raw data and construct an easy to manipulate matrix. This script takes into account that driving closer to an object will change what the sensor reads, updating the matrix accordingly. The script can be found in **Appendix B.**

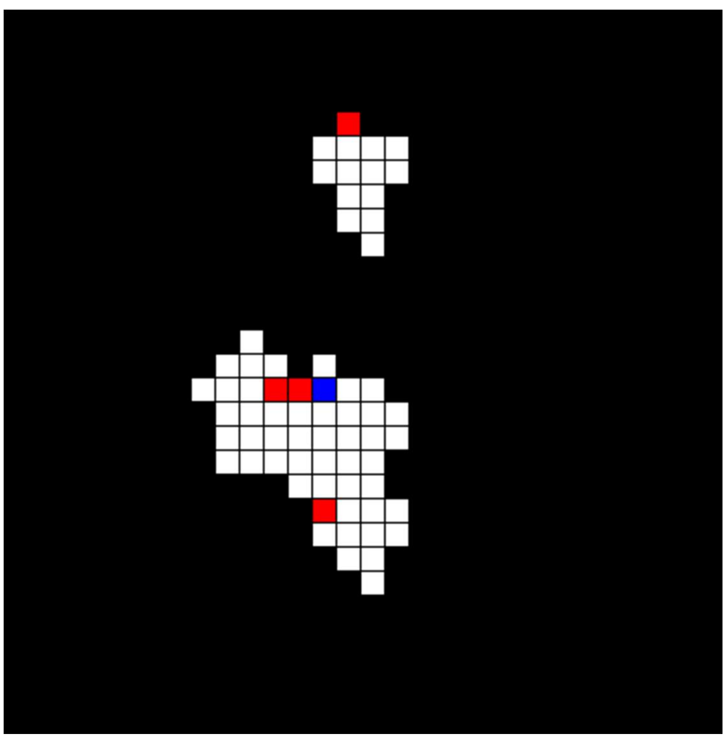
The python script reads the values in and constructs a matrix of:

- 0(black pixel): haven't explored this area of the room
- 1(white pixel): have scanned this area of the room and found empty space
- 2(red pixel): have scanned this area of the room and found an obstacle

For every ultrasonic sensor reading sent back that shows no obstacle, the script simulates the car sending a beam of 4ft from the car's current position in its current angle facing direction.

The script breaks the user-determined room size (width x height) into 1ft x 1ft squares.

Using Processing, we draw the grid with the corresponding colors filled in. An example of a room map can be seen below. The processing code can be found in **Appendix C.**



# Conclusion

After applying this visualization software on the room during the demo, it mapped Rad at a distance of 20ft from the car's starting position, which is the width of the room and exactly how many squares Radd started away from the demo.

Overall our car tracked its position and obstacles very accurately, within quarter feet accuracy, and was able to successfully map out a portion of the undergraduate lab.

Had we had additional time, we would have worked on increasing the resolution of obstacle detection to be narrower than one foot. In addition, we also had plans to add more ultrasonic servos at various height to create 3D maps and pseudo-code to implement a heuristic for the car to automatically move about and map out the room.

Go, Speed Racer!

# Appendix A: Arduino Code

```
#include <Servo.h>

#define LEFTHE_PIN 3
#define RIGHTHE_PIN 2
#define CARLENFT .82
#define WHEELLENFT .0202
#define ULTRASONICCALIBRATE 90
#define STEERINGCALIBRATE 100

/*PINS IN USE*/
#define ULTRASOUNDSERVOPIN 9
#define STEERINGSERVOPIN 10
#define FORWARDSTEERPIN 8
#define TRIGPIN 7
#define ECHOPIN 6
#define ONOFF 5

volatile double gxPos = 0;
volatile int totalReads = 0;
volatile double gyPos = 0;
volatile double ganglePos = 1.57;
volatile int glastRightCount = 0;
volatile int glastLeftCount = 0;
volatile int gleftcount = 0;
volatile int grightcount = 0;

boolean gforward = true;
boolean gon = false;
int steeringServoAngle = STEERINGCALIBRATE;
int ultrasoundServoAngle = ULTRASONICCALIBRATE;
Servo steeringServo;
Servo ultrasoundServo;

void LEFTHE_ISR() {
 gleftcount++;
 if (totalReads%10 == 0){
  updatePos();
 }
 totalReads+=1;
}

void RIGHTHE_ISR() {
 grightcount++;
}

void printSurroundings(double distance, double angleRevise){
 Serial.print(gxPos, DEC);
 Serial.print(" ");
 Serial.print(gyPos, DEC);
```

```
  Serial.print(" ");
  Serial.print((ganglePos-angleRevise), DEC);
  Serial.print(" ");
  Serial.println(distance, DEC);
}

int ultrasoundRead(){
  digitalWrite(TRIGPIN, LOW);  // Added this line
  delayMicroseconds(2); // Added this line
  digitalWrite(TRIGPIN, HIGH);
  delayMicroseconds(10); // Added this line
  digitalWrite(TRIGPIN, LOW);
  double duration = pulseIn(ECHOPIN, HIGH);
  double distance = (duration/2) / 29.1;
  return distance;
}

void scanSurroundings(){
  double duration, distance;
  int angleRotation = 45;
  boolean direction = false;
  double angleCalibrate = 0;
  for (int i = 60; i <= 120; i+=7.5){
     ultrasoundServo.write(i);
     distance = ultrasoundRead();
     if (distance < 7 || distance > 150) distance = 0;
     angleCalibrate = (i-ULTRASONICCALIBRATE)*(3.14/180);
     printSurroundings(distance, angleCalibrate);
    }
}

void updatePos(){
  double xChange = 0;
    double yChange = 0;
  double rightChange = (grightcount - glastRightCount)*WHEELLENFT;
  double leftChange = (gleftcount - glastLeftCount)*WHEELLENFT;
//  double theRightAnswer = grightcount - glastRightCount;
//  double theLeftAnswer = gleftcount - glastLeftCount;
  glastLeftCount = gleftcount;
  glastRightCount = grightcount;
  double distanceTraveled = (rightChange+leftChange)/2.0;
  double servoAngleDif = steeringServoAngle - STEERINGCALIBRATE;


  double angleChange = (rightChange - leftChange)/(CARLENFT);
  if (abs(servoAngleDif) < 10){
            angleChange = 0;
  }
  /*servoangle and actual angle are opposites*/
  else if (servoAngleDif < -10 && angleChange < 0){
    angleChange = 0;
  }
  else if (servoAngleDif > 10 && angleChange > 0){
```

```
    angleChange = 0;
  }


  if (!gforward){
    distanceTraveled = - distanceTraveled;
  }

    xChange = distanceTraveled*cos(ganglePos+angleChange/2);
    yChange = distanceTraveled*sin(ganglePos+angleChange/2);


  ganglePos += angleChange;
  gxPos += xChange;
  gyPos += yChange;
}

void printPos() {
  Serial.print("X POS IS:");
  Serial.print(gxPos, DEC);
  Serial.println();
  Serial.print("Y POS IS:");
  Serial.print(gyPos, DEC);
  Serial.println();
  Serial.print("ANGLE IS");
  Serial.println((ganglePos*180/3.14), DEC);
  Serial.print("RIGHTTURNS IS:");
  Serial.println(grightcount);
  Serial.print("LEFT TURNSIS:");
  Serial.println(gleftcount);
  Serial.println(gforward);
  Serial.println("***************************");
}

void setup() {
  attachInterrupt(1, LEFTHE_ISR, FALLING);
  attachInterrupt(0, RIGHTHE_ISR, FALLING);
  steeringServo.attach(STEERINGSERVOPIN);
  ultrasoundServo.attach(ULTRASOUNDSERVOPIN);
  steeringServo.write(steeringServoAngle);
  ultrasoundServo.write(ultrasoundServoAngle);
  sei();
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    char value = Serial.read();
    if (value == 'w'){
      if(gforward) {
        digitalWrite(FORWARDSTEERPIN, LOW);
        digitalWrite(FORWARDSTEERPIN, HIGH);
        digitalWrite(FORWARDSTEERPIN, LOW);
```

```
//      Serial.println("GO FORWARD");00
    }
    else {
      digitalWrite(FORWARDSTEERPIN, LOW);
      digitalWrite(FORWARDSTEERPIN, HIGH);
      digitalWrite(FORWARDSTEERPIN, LOW);
//      Serial.println("GO BACKWARD");
    }
    gforward = !gforward;
  }
  else if (value == 'd'){
    if (steeringServoAngle > 124) {
      steeringServoAngle = 124;
    }
    else {
      steeringServoAngle +=12;
    }
//    Serial.println("GO RIGHT");
//    Serial.println("STEERING ANGLE");
//    Serial.println(steeringServoAngle);
  }
  else if (value == 'a'){
    if (steeringServoAngle < 64) {
      steeringServoAngle = 64;
    }
    else {
      steeringServoAngle -=12;
    }
//    Serial.println("GO LEFT");
//    Serial.println("STEERING ANGLE");
//    Serial.println(steeringServoAngle);
  }
  else if (value == 'r'){
     steeringServoAngle = STEERINGCALIBRATE;
     ultrasoundServoAngle = ULTRASONICCALIBRATE;
     ultrasoundServo.write(ultrasoundServoAngle);
     gxPos = 0.0;
     gyPos = 0.0;
     ganglePos = 1.57;
  }
  else if (value == 's'){
//    Serial.println("SCAN SURROUNDINGS");
      scanSurroundings();
  }
  else if (value == 'p'){
     printPos();
  }
  else if (value == 'o'){
      gon = !gon;
      if (gon){
        digitalWrite(ONOFF, LOW);
        digitalWrite(ONOFF, HIGH);
        digitalWrite(ONOFF, LOW);
```

```
//        Serial.println("ON");
      }
      else {
        digitalWrite(ONOFF, LOW);
        digitalWrite(ONOFF, HIGH);
        digitalWrite(ONOFF, LOW);
//        Serial.println("OFF");
      }
    }
    steeringServo.write(steeringServoAngle);
  }
  delay(50);
}
```

# Appendix B: Python Script

```
import sys
import bz2
import zlib
import math


if __name__ == "__main__":
    col = sys.argv[1]
    row = sys.argv[2]
    matrix=[[2 for x in range(int(col))] for x in range(int(row))]
    closestDistances = [[200 for x in range(int(col))] for x in range(int(row))]
    lineNumber = 0;

    for line in sys.stdin:
        # print(str(lineNumber) + "GENERAL")
        if (lineNumber == 0):
            lineNumber+=1
            continue
        splits = line.split()
        xPos = float(splits[0])+20
        yPos = float(splits[1])+30
        curAngle = float(splits[2])
        obsDistance = float(splits[3])/30.48
        if (obsDistance != 0):
            xchangePos = xPos + math.cos(curAngle)*obsDistance
            ychangePos = yPos + math.sin(curAngle)*obsDistance
            if (xchangePos >= 0 and xchangePos < row
                    and ychangePos >= 0 and ychangePos < col):
                if (obsDistance < closestDistances[int(ychangePos)][int(xchangePos)]):
                    # print "obstacle here"
                    # print xchangePos
                    # print ychangePos
                    # print obsDistance
                    matrix[int(ychangePos)][int(xchangePos)] = 1
                    closestDistances[int(ychangePos)][int(xchangePos)] = obsDistance
        else:
            for i in range(0, 5):
                # print "hi"
                xChangePos = math.cos(curAngle)*i + xPos
                yChangePos = math.sin(curAngle)*i + yPos
                if (xChangePos >= 0 and xChangePos < row
                    and yChangePos >= 0 and yChangePos < col):
                    if (i < closestDistances[int(yChangePos)][int(xChangePos)]):
                        closestDistances[int(yChangePos)][int(xChangePos)] = i
                        # print "no obs dude"
                        matrix[int(yChangePos)][int(xChangePos)] = 0
        lineNumber+=1
```

```
matrix[int(yPos)][int(xPos)] = 3
for i in range(0, len(matrix)):
   for x in range(0, len(matrix[0])):
      print matrix[len(matrix)-1-i][x],
   print
# for i in range(0, len(closestDistances)):
#     print closestDistances[len(closestDistances)-1-i]
```

# Appendix C: Processing Visualization

```
import processing.serial.*;

Serial myPort;
float[][] obstacleMap;
float[][] actualMap;
float maxDistance;
int spacer;
int maxWidth = 44;
int maxHeight = 64;


void setup() {
   spacer = 15;
  obstacleMap = new float[maxHeight][maxWidth];
  String lines[] = loadStrings("roommap.txt");
  for (int i = 0; i < lines.length; i++){
   String[] isObstacle =lines[i].split(" ");
   print(i);
   if (isObstacle.length < 1) continue;
   for (int j = 0; j < isObstacle.length; j++){
     obstacleMap[i][j] = int(isObstacle[j]);
   }
   println();
  }
  for (int i = 0; i < obstacleMap.length; i++){
    for (int j = 0; j < obstacleMap[0].length; j++){
      print (obstacleMap[i][j] + " ");
    }
    println();
  }
  size(maxWidth*spacer, maxHeight*spacer);
  noLoop();  // Run once and stop
}

void draw() {
  background(0);
  // This embedded loop skips over values in the arrays based on
  // the spacer variable, so there are more values in the array
  // than are drawn here. Change the value of the spacer variable
  // to change the density of the points
  for (int y = 0; y < height; y += spacer) {
   for (int x = 0; x < width; x += spacer) {
     stroke(0);
     int mycolor = int(obstacleMap[y/spacer][x/spacer]);
     if (mycolor == 0){
       fill(255, 255, 255);
     }
     else if (mycolor == 1){
```

```
        fill(255, 0, 0);
      }
      else if (mycolor == 2){
        fill(0, 0,0);
      }
      else {
        fill(0, 0, 255);
      }
      rect(x, y, spacer, spacer);
    }
  }
}
```

# Appendix D: PSOC Code

```
#include <project.h>
#include <stdio.h>

#define INCH_PER_MAGNET 1.58
#define SEC_PER_PERIOD 357.914

#define THREE_FT_DUTY 900
#define CENTER_DUTY 4560
#define CENTER_LINE 780

#define Kp 1250
#define Ki 120
#define Kd 0

#define Kp_steer 2.25
#define Ki_steer 0
#define Kd_steer 30000

double gExpectedSpeed = 3.5;
double gTotalTraveled = 0;

double gprev_HE_count = 0;
int gfirst_HE_read = 1;
int gspeedMeasurements = 0;
double gcurSpeed = 0;
double speedCounts[5];
double gki_speederror = 0;
double gki_steererror = 0;
int glinepos = 0;
double gsteer_dutycycle = 0;

int gnum_line_reads = 0;
double gblack_pos_first_diff = 0;
double gblack_pos_second_diff = 0;
double gblack_totalpos_diff = 0;
int gCounterNReads = 0;
int gblackcount = 0;
uint32 gfirstpos = 0;
uint32 gsecondpos = 0;
uint32 gthirdpos = 0;
uint32 gfourthpos = 0;
uint32 gcaptures = 0;
int count = 0;
int gsteer_error_prev = 0;

uint8 direction = 0;

int gONOFF = 0;
```

```c
int gcurr_dir = 1;


CY_ISR(DIR_inter) {
    if (gcurr_dir == 0) {
        LCD_ClearDisplay();
        LCD_PrintString("forwards!");
        gcurr_dir = 1;
        DIR_REG_Write(gcurr_dir);
    }
    else if (gcurr_dir == 1) {
        LCD_ClearDisplay();
        LCD_PrintString("backwards!");
        gcurr_dir = 0;
        DIR_REG_Write(gcurr_dir);
    }
}

CY_ISR(ON_OFF_inter) {
    if (gONOFF == 0) {
        MOTOR_PWM_WriteCompare(2000);
        LCD_ClearDisplay();
        LCD_PrintString("motor on!");
        gONOFF = 1;
    }
    else if (gONOFF == 1) {
        MOTOR_PWM_WriteCompare(0);
        LCD_ClearDisplay();
        LCD_PrintString("motor off!");
        gONOFF = 0;
    }
}

int main()
{
    //initialize all modules
    CYGlobalIntEnable;

    DIR_ISR_Start();
    DIR_ISR_SetVector(DIR_inter);

    ON_OFF_ISR_Start();
    ON_OFF_ISR_SetVector(ON_OFF_inter);

    LCD_Start();
    LCD_Position(0,0);
    LCD_PrintString("ELE302 Carlab ");

    MOTOR_PWM_Start();
    MOTOR_PWM_CLK_Start();

    //start off driving forward
    DIR_REG_Write(1);
```

```
    for(;;)
    {
    }
}
```