ME103: Experimentation & Measurement
Final Lab Report

Roll Car Team: Tuesday 8 AM

Aariz Hazari, Alonzo Vang, Fayyad Azhari, Richard Vu, Young Ri Son

1. Abstract

In this project we aimed to build a motorized car that would be able to track and mimic the position and speed of a vehicle travelling on a parallel lane. The ultimate goal was to implement a payload transfer, which due to several challenges, could not be achieved. However, we were able to design and develop a functional autonomous vehicle prototype that was able to detect, track, catch up to and maintain position with another vehicle incorporating a control system and ultrasonic proximity sensors.

2. Introduction

The technology as well as the market for autonomous vehicles is in a rapid state of expansion with no foreseeable end in sight. From safety features to self-driving, the possibilities seem endless. The impact of this project is to explore the idea of autonomous transportation of payloads, such as transporting a passenger between two trains, through the construction of a vehicle that can autonomously detect, track, catch up to and maintain position with another vehicle.

2.1 Background

Some highly publicized benefits of autonomous vehicle position and speed tracking on parallel lanes includes the implementation of payload transfer while in motion. This could conserve energy and time by not having to stop, load and reaccelerate. In addition the removal of human interaction prevents endangerment of human life and removes the unpredictability of human error in operations.

2.2 General Goals

The first goal that was achieved early on in the semester was to generate a well rounded model of the track. This was achieved by running the roll car over the track several times from different positions. The preexisting sensors on the track which helped us get the position of the roll car as it travelled through the track. The position information was differentiated with respect to time to get information about the velocity and acceleration for different launch positions.
The next objective was to detect when a car passes the tracking vehicle and understand in which direction the car was moving. The car was detected with the use of the two ultrasonic sensors which detected changes in the environment. The heading of the neighboring vehicle was found by knowing the order in which the sensors were triggered. The other big target to reach was the need to match the neighboring vehicle's position and velocity. Lastly, if there is time we would implement a transfer mechanism in a really safe manner.

2.3 Theory

To achieve our goal of tracking and following a neighboring vehicle we came up with two ideas. The first one involved building a robust model of the track. This model would be made using numerous trials of the lab's provided roll car, such that we would know the position, speed of the roll car depending on the position of release. Once a model would be made we would have tried to tried to make the tracking vehicle mimic the model. However, in a real world environment this model would never be achieved and hence this approach would not be very useful.

The alternative was to use a real time tracking system. In order to achieve our goal of real time tracking and mimicking the activity of a neighboring vehicle we required sensors that would be able to detect objects around the tracking vehicle. Our initial idea was to mount sensors on the tracking vehicle and detectable tags on the target vehicle. However, as we decided to use the roll car provided to us in the lab, we could not go forward with this approach as it was difficult to attach anything to it. Instead, we decided to use two HR-SR04 ultrasonic sensors on the tracking vehicle, which send sound waves through a trigger which bounces off an object and return to a receiver. The time taken by the sound wave to travel to the receiver is recorded and used to calculate the distance to the nearest object.

2.4 Assumptions

The situation we decided on was two vehicles moving parallel to each other. While we decided to use a lab provided roll car as the tracked vehicle, a chassis needed to be acquired for the tracking vehicle. We made sure that this chassis was wide enough to fit the track, but not so wide as to touch the walls of the track or the neighboring vehicle. Although our vehicle left excess space and did not hug the track, we assumed the vehicle would be able to stay true as long as the DC motors all spun at the same speed. In addition, it was assumed that slippage occurring when traveling along hills would not pose a significant problem. Knowing that our DC motors had a low maximum speed incapable of pursuing the target vehicle in the beginning of a trial, we opted to use dual ultrasonic sensors. The idea of using only one sensor was considered, but we assumed it would be impossible to tell the difference between the target vehicle slowing down and the vehicle changing direction, making a control strategy difficult.

2.5 Hypothesis

The ideal situation for our vehicle is to wait for the target vehicle to pass by, and then begin tracking it. As shown in Figure 2.1 below, one of the two sensors, keyed in blue, would be dedicated to tracking the target vehicle's position, changing velocity as needed in order to stay adjacent to the target. The secondary sensor, keyed in red, would be used to detect when the target vehicle changed direction and when our vehicle had somehow passed the target accidentally. Once our vehicle detected a switch in direction, the roles of the sensors would also be switched, as the front side sensor would be dedicated to position tracking and the rear side sensor dedicated to direction switching detection.

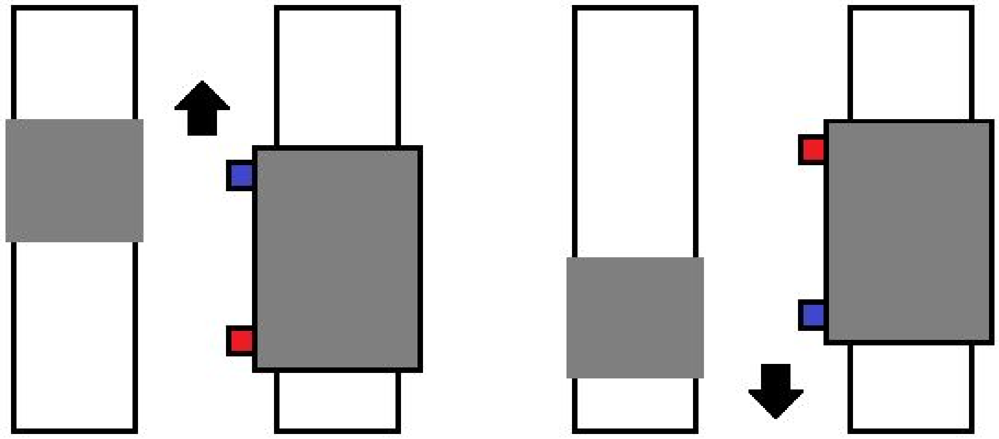
Figure 2.1: Diagram of Active Side Tracking with Dual Ultrasonic Sensors

3. Methods

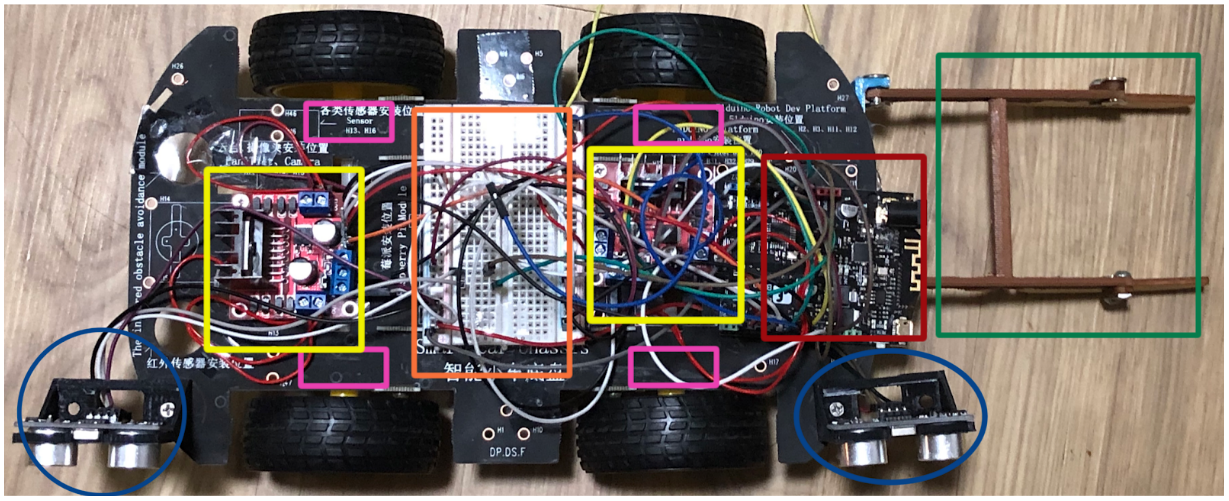
Figure 3.1: top view of the vehicle with labels

- L298N Motor Driver (Yellow) x2: Motor driver consists of two pairs of transistors to control the amount and direction in which current flows through the motor. It allows us to manipulate the speed and direction of the vehicle. One is placed at the front and another at the back to be connected with the front pair and the rear pair of motors.
- HC-SRO4 Ultrasonic Sensor (Blue) x2: This ultrasonic sensor detects the distance from the sensor to the object by measuring the time it takes for the ultrasound to come back after being reflected by the target object. Since the distance between the sensor and the roll car is constant in our experiment, the sensor is used to detect the presence of the roll car across the rail. The sensor is set perpendicular to the rail.
- Arduino Uno (Red) x1: The Arduino board is a microcontroller board which controls our vehicle by analyzing the incoming sensor data and controlling the output to the DC motors according to the created control strategy.
- Breadboard (Orange) x1: The breadboard is used instead of soldering circuit in order for us to change the wire setting freely. It is placed in the middle to connect all the electronic components such as the Arduino, motor drivers, and ultrasonic sensors.

- Brushed Motor (Pink) x4: The brushed motor is chosen over the brushless motor due to the cheaper price. These came included with the purchase of the car chassis.
- Guide Sled (Green) x1: The guide sled was specially designed to constraint the vehicle from going off the rail. The sleds and the arm are connected by pin joint to make sure the sleds always contact with the rail in parallel.

The control strategy implemented to track the target vehicle was simple and straightforward. To prevent false positives with sensor activity, a distance threshold was set for the sensors within the program, which was measured experimentally using data gathered from the Arduino Serial Monitor. Due to the assumed DC motors' low maximum speed, anytime our vehicle was unable to detect anything with the front side sensor simply meant the target was ahead, and would increase velocity in pursuit. If the front side sensor did detect anything, our vehicle would slow down as to not overtake the target vehicle. If the rear side sensor detected anything, that would signify a switch in direction, and our vehicle would act accordingly. However, we ran into a problem where the target vehicle would bounce between the two sensors once the target was moving slowly enough. This was due to the DC motors' inability to go under a minimum speed without fully stopping.

The chassis of the car was made out of fiberglass and the distance between the width of the wheels on the chassis was 9 cm. This was a problem because the width of the track was 6.4 cm so our roll car had difficulty staying straight on the track. Therefore, we had to come up with a solution in order to keep the roll car from going off the track. We decided that instead of designing a new chassis, we would just design an additional extending piece to connect to the roll car. We first decided that we would build guide rails for the roll car, however, the roll car was still running off the track. We believed this was because the guide rails did not have enough surface area touching the track from the guide rails, so we designed sleds to add to the guide rails to increase the amount of surface area touching the track. This design had the better results so we decided to keep it.

After we figured out the best design to keep our roll car on track, we decided that we wanted to know the limitations to our project. In order to do that, we choose to use the heaviest example roll car because it would run the longest. This was because it had the highest momentum compared to the lighter ones and that would lead to collecting a larger data sample. The heaviest roll car was about 2458.8 grams. For the experiment, we dropped the example roll car at a certain height, specifically the fourth line starting from the bottom on the middle hill of the track, and had our roll car at ground level waiting for it to pass by. Once the example roll car triggered the sensors on our roll car, our roll car began following the example car until it was able to match the velocity of the example roll car. As the experiment went on, the example roll car began to lose velocity, eventually slowing down to the point where our roll car would began to bounce, constantly switching direction while containing the example car between the two ultrasonic sensors. During the length of these experiments, the position sensors built into the track was also taking data on where the example car was at during specific times on the track. We used this position data and differentiated it to get the maximum and minimum velocities our roll car was able to track. The maximum velocity was when our roll car caught up to the example and the minimum was the moment our roll car began to bounce.

4. ***Results***

Since the tracking vehicle has a speed limitation, we needed to figure out the the maximum speed of the target car for our car to keep up. Moreover, we also have to calculate the minimum speed of the roll car when the tracking vehicle starts bouncing forward and backwards due to the blind spot that exists between the two sensors on the tracking vehicle.

By having the roll car starting at a position that will produce a launch velocity that is larger than the maximum velocity of the tracking vehicle, we are able to figure out the maximum speed that the vehicle can start tracking. Through using three timers by three different people, we collected the average time in Table 4.1. Then, the average time for tracking to begin and stops for four trials was calculated.

Trial	Average Time for Tracking to Begin (seconds)	Average Time for Bouncing to Begin (seconds)
1	17.32	32.94
2	18.67	33.91
3	19.66	41.61
4	18.64	39.13
Average	**18.57**	**36.90**

Table 4.1: The average Time for Tracking and Bouncing to Begin for Four Trials

Using the existing position sensors on the roll car track, we can obtain the position of the roll car track at any given time. The position data was fitted using a 'smoothing spline' on MATLAB and the data is plotted in Figure 4.1. The sensor on position nine of the track was set to be the initial position to get the optimal launch velocity in order to find the maximum speed that our vehicle can track.

By differentiating the displacement data in MATLAB, the velocity of the roll car can be obtained. The velocity of the roll car for the four trials is plotted in Figure 4.2. From the average tracking and bouncing time recorded for every trials, the maximum velocity and the minimum velocity that the vehicle can track were obtained and recorded on Table 4.2. The total duration that our tracking vehicle can maintain with the roll car can be seen in between the two vertical line both on Figure 4.1 and Figure 4.2.

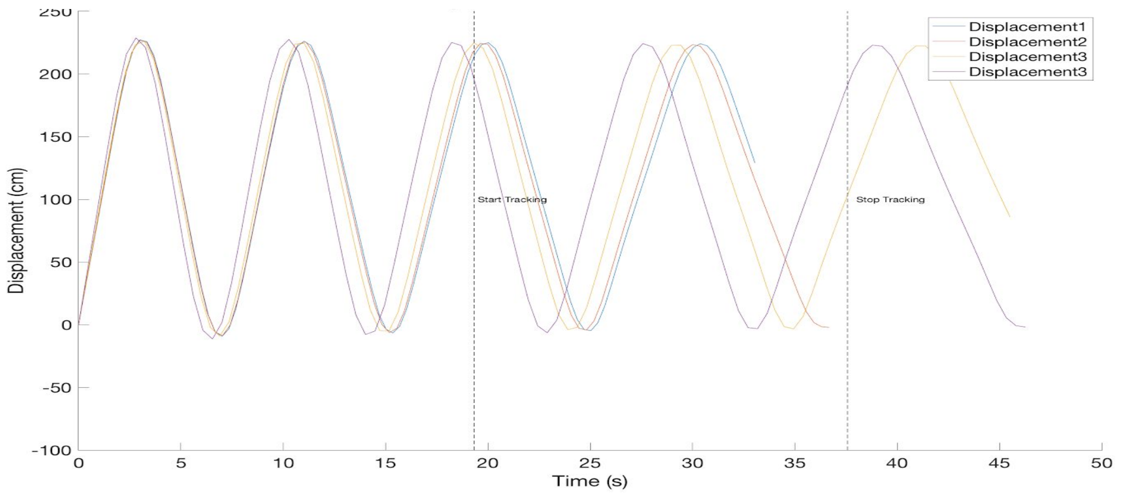

Figure 4.1: The displacement of the roll car vs time with 95% confidence level

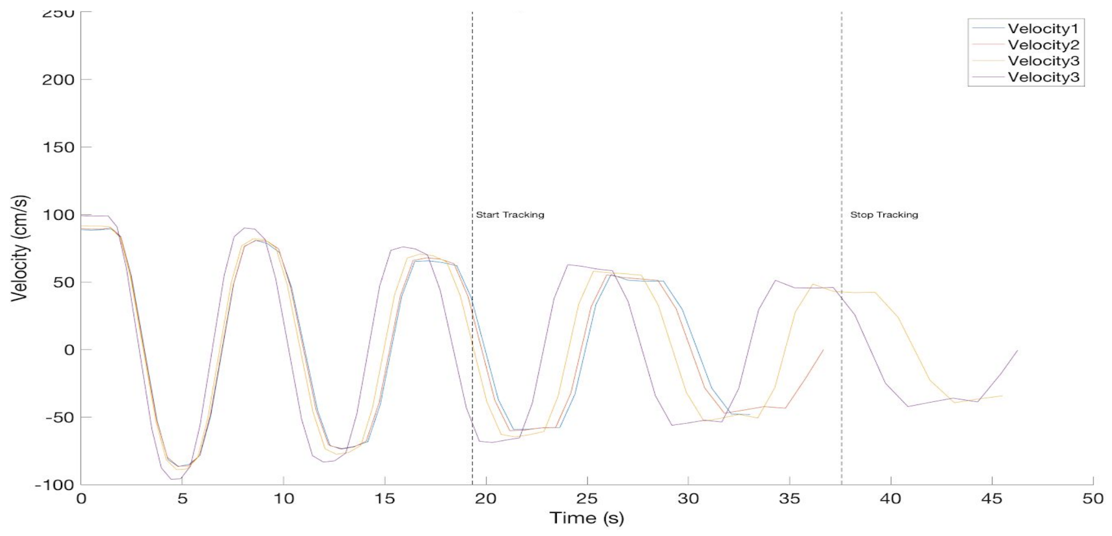

Figure 4.2: The velocity of the roll car vs time with 95% confidence level

Trial	Maximum Velocity (cm/s)	Minimum Velocity (cm/s)
1	64.35	48.17
2	63.57	42.32
3	63.17	39.46
4	65.44	42.29
Average	**64.13**	**43.06**

Table 4.2: The maximum and minimum velocity that the tracking vehicle can maintained with the roll car

5. Discussion

By the end of the semester we were able to build a model robust enough to track and mimic the speed and position of a neighboring vehicle. The model implemented was successful in tracking the vehicle going forward or in reverse. However, over the course of the semester we ran into several problems which prevented us from reaching our ultimate goal of implementing a payload transfer. The car frame along with its wheels were too big leading to the car going off track. While, this was something that we fixed by making a sled, it was a drain on the already limited time that we had. Furthermore, the car's motors were not fast enough to start tracking at a faster speed and needed the lab's roll car to slow down before tracking could be fully initiated. In the tracking setup, there exists a blind spot between the two sensors. When the vehicle to be tracked falls in this blindspot, the tracking vehicle can not tell the difference between overtaking the target car and the target car switching direction, and ends up constantly switching directions itself in order to keep the target car within the blindspot. Furthermore, the performance of the entire setup also depends on how much the battery is charged. This can be seen in the reported tracking performance in Table 4.1 and Table 4.2 where the tracking performance decreases in trials 2 and 3 after the battery has been used without being charged. The time taken to begin tracking as reported in Table 4.1 increases from 17.32 seconds to 18.67 seconds in trial 2 and eventually to 19.66 seconds. The maximum velocity at which tracking can start also decreases from 64.35 cm/s in trial 1 to 63.37 cm/s and 63.17 in trials 2 and 3.

6. Conclusion

The main goal for our roll car was to be able to track and match the speed of another car in a parallel lane. Our second goal, which was our bonus goal, was to be able to transfer a payload autonomously from our roll car to the example roll car being tracked. We were successfully able to complete our main goal on a small scale, meaning the maximum speed and minimum speed that our roll car was able to track did not have a wide range. The max speed was about 64.13 cm/s and the minimum speed was about 43.06 cm/s. However, with the proper equipment, funding, and time, we believe we would be able to successfully widen our range of a maximum and minimum speed and achieve our second goal of transferring a payload.

7. Author Contributions

Young Ri Son - Helped brainstorm idea for the project and for obstacles throughout the semester. Did CAD work for the sled and 3D printed the sled. Wrote parts of the discussion and background sections in the report.

Alonzo Vang - Helped brainstorm idea for the project and for obstacles throughout the semester. Played a part in data collection for the various experiments. Helped assemble the robot and add sled to the robot. Helped to find and acquire parts for the project. Wrote parts of the discussion and background sections in the report.

Fayyad Azhari - Helped brainstorm idea for the project and for obstacles throughout the semester. Worked on the data acquisition and analysis aspect of the experiments. Used hand tools to add the sled to the robot. Helped to find and acquire parts for the project. Wrote parts of the results, theory and discussion sections in the report.

Aariz Hazari - Helped brainstorm idea for the project and for obstacles throughout the semester. Worked on data acquisition and analysis of the experiments. Used hand tools to add sled to the robot. Helped to find and acquire parts for the project. Wrote parts of the discussion, abstract, introduction and background sections in the report.

Richard Vu - Helped brainstorm idea for the project and for obstacles throughout the semester. Wrote the entire codebase for tracking the target vehicle, and the skeleton code used for data analysis of the track’s position sensors. Described the introductory assumptions and usage of dual ultrasonic sensors, as well as problems regarding the DC motors. Wrote parts of the theory and background sections in the report, as well as the assumptions and hypothesis.

Project Reflection on ABET Goals

As a team we agree that our project was very successful in helping us apply various engineering topics. We had to use design skills, programming skills, problem solving skills, along with the need to experiment and analyze data. For example, designing and creating the guide rails for our vehicle took from our design and problem solving skills. In addition, choosing different components for the lab provided roll car and different initial conditions for its launch used our experiment design skills and data analysis and interpretation abilities. The project required us to combine all these skills to succeed and to be able to move forward.

Furthermore, our project required us to understand complex engineering problems and come up with ideas to solve them, as well as use technique, skills, and tools necessary for engineering practice. For example, analyzing the built-in track position data from LabVIEW was not as simple as differentiating position with respect to time using MATLAB; there were issues of noise and discontinuities that needed to be addressed before further analysis. Another complex problem was trying to quantify the ability of our vehicle in tracking the target vehicle, as there was only one set of position sensors available on one side of the track. Through the use of well-designed experiments, we were able to successfully analyze the position sensor data with significance to the results. Once the various engineering skills were applied for the project, we had to document the progress, our thought process and our results in a professional and formal manner, using our abilities to communicate in order to share the information we gathered with each other and anybody interested in the project.

The project also highlighted our ability to work in multi-disciplinary teams, as each member was given a specific topic of task to work on, whether that be data analysis, programming, manufacturing, etc. Seeing the imperfections of our project also helps emphasize the concept of lifelong learning. Even though our progress on the project was significant and did meet our expectations, fixing problems such as bouncing, the inability to stay true to the track, and implementing our second milestone of payload transfer would require us to research more ideas and concepts.

Appendix A: Program to Control Our Vehicle (Arduino)

```
/*
   This code is to define how our DC motors work for the car.
   From these motor drivers, we will write PWM signals to
   the pins ENA and ENB, and use IN1, IN2 to control direction
   for Motor A, and pins IN3, IN4 for Motor B. It is important that
   ENA and ENB are connected to digital outputs that support PWM!
   (Marked with a ~ sign on the Arduino board). Pins for IN1 - IN4
   do not require PWM, as they work by constant high/low voltages.

   Definitions:
   FL - Front Left
   FR - Front Right
   RL - Rear Left
   RR - Rear Right

   FS - Front Sensor
   RS - Rear Sensor

   The front motors will have input pins designated as inF1 through inF4,
   while the rear motors input pins will be designated as inR1 through inR4.

   The motor PWM pins are defined as 3, 9, 10, and 11 as digital pins 5
   and 6 give a PWM signal with a different frequency (980 Hz) compared
   to the others (490 Hz).

   As of April 16, 2019, this version of the code works!
   Features:
     Able to wait for an object to pass before beginning movement
     Able to slow down and speed up to follow the object closely
     Able to switch directions when the object passes the car
*/

const int motorFL = 3;
const int motorFR = 9;
const int motorRL = 10;
const int motorRR = 11;

const int inF1 = 2;
const int inF2 = 4;
const int inF3 = 5;
const int inF4 = 6;

const int inR1 = 7;
const int inR2 = 8;
const int inR3 = 12;
const int inR4 = 13;

const int triggerFS = A0;
const int readFS = A1;
const int triggerRS = A2;
const int readRS = A3;

int currentSpeed = 0;

long frontDuration;
long rearDuration;

/*
   The approximate time needed for a ultrasound pulse to travel

   timeThreshold represents the distance between our car and an
   object on the track

   timeTrackWidth represents the distance between our car and the
   edge of the track. Any time above this limit is rounded down to timeTrackWidth to
   represent an artificial saturation.

   Depending on where the car is placed, the timeThreshold should be changed
   accordingly.

   Closer to Roll Car - 1500
   Center - 1550?
   Farther from Roll Car - ?
*/

int timeThreshold = 1475;
int timeTrackWidth = 1600;

bool goingForward;

bool rearRead, frontRead;
bool started; // Used for initial starting

void setup() {
  pinMode(motorFL, OUTPUT);
  pinMode(motorFR, OUTPUT);
  pinMode(motorRL, OUTPUT);
  pinMode(motorRR, OUTPUT);

  pinMode(inF1, OUTPUT);
  pinMode(inF2, OUTPUT);
  pinMode(inF3, OUTPUT);
  pinMode(inF4, OUTPUT);

  pinMode(inR1, OUTPUT);
  pinMode(inR2, OUTPUT);
  pinMode(inR3, OUTPUT);
  pinMode(inR4, OUTPUT);
  pinMode(triggerFS, OUTPUT);
  pinMode(readFS, INPUT);
  pinMode(triggerRS, OUTPUT);
  pinMode(readRS, INPUT);

  goNeutral();
  started = false;
}

void loop() {
  echo();

  // Before any initial movement, this if loop should be triggered.
  // The car will not move until the rear sensor detects something.
  // loop() will continuously run here until then
  if (!started) {
    if (rearDuration < timeThreshold) {
      started = true;
      goForward();
      changeSpeed(255);
      // This delay is so we do not accidentally
      // switch direction because the object may still
      // be in range of the rear sensor even if we start
      // moving
      delay(2000);
    }
    return;
  }
  if (goingForward) {
    // Conditions to switch direction is that the object will
    // pass the rear sensor while the car is moving forward
    if (rearDuration < timeThreshold) {
      // Roll car is now rolling the other way
      stopMoving();
      goBackward();
      changeSpeed(currentSpeed);
      return;
    }
    if (frontDuration > timeThreshold) {
      // keep going forward
      changeSpeed(round(currentSpeed * 1.05));
    } else if (frontDuration < timeThreshold) {
      // If both car and object are moving forward, then once
      // the car catches up it will start to slow down
      changeSpeed(round(currentSpeed / 1.05));
    }
  }
  if (!goingForward) {
    if (frontDuration < timeThreshold) {
      // Roll car is now rolling the other way
      stopMoving();
      goForward();
      changeSpeed(currentSpeed);
      return;
    }
    if (rearDuration > timeThreshold) {
      // keep going forward
      changeSpeed(round(currentSpeed * 1.05));
    } else if (rearDuration < timeThreshold) {
      changeSpeed(round(currentSpeed / 1.05));
    }
  }
}

void goForward() {
  goingForward = true;
  digitalWrite(inF1, LOW);
  digitalWrite(inF2, HIGH);
  digitalWrite(inF3, LOW);
  digitalWrite(inF4, HIGH);
  digitalWrite(inR1, LOW);
  digitalWrite(inR2, HIGH);
  digitalWrite(inR3, LOW);
  digitalWrite(inR4, HIGH);
}

void goBackward() {
  goingForward = false;
  digitalWrite(inF1, HIGH);
  digitalWrite(inF2, LOW);
  digitalWrite(inF3, HIGH);
  digitalWrite(inF4, LOW);
  digitalWrite(inR1, HIGH);
  digitalWrite(inR2, LOW);
  digitalWrite(inR3, HIGH);
  digitalWrite(inR4, LOW);
}

void goNeutral() {
  digitalWrite(inF1, LOW);
  digitalWrite(inF2, LOW);
  digitalWrite(inF3, LOW);
  digitalWrite(inF4, LOW);
  digitalWrite(inR1, LOW);
  digitalWrite(inR2, LOW);
  digitalWrite(inR3, LOW);
  digitalWrite(inR4, LOW);
}
```

```
void changeSpeed(int motorSpeed) {
  if (motorSpeed < 100) {
    // Prevent undefined motor behavior at low voltages
    motorSpeed = 100;
  } else if (motorSpeed > 255) {
    // Anti-saturation
    motorSpeed = 255;
  }
  analogWrite(motorFL, motorSpeed);
  analogWrite(motorFR, motorSpeed);
  analogWrite(motorRL, motorSpeed);
  analogWrite(motorRR, motorSpeed);
  currentSpeed = motorSpeed;
}

void stopMoving() {
  analogWrite(motorFL, 0);
  analogWrite(motorFR, 0);
  analogWrite(motorRL, 0);
  analogWrite(motorRR, 0);
}

void echo() {
  digitalWrite(triggerFS, LOW);
  delayMicroseconds(500);
  digitalWrite(triggerFS, HIGH);
  delayMicroseconds(1000);
  digitalWrite(triggerFS, LOW);

  long timeout = 20000; // microseconds

  frontDuration = pulseIn(readFS, HIGH, timeout);

  digitalWrite(triggerRS, LOW);
  delayMicroseconds(500);
  digitalWrite(triggerRS, HIGH);
  delayMicroseconds(1000);
  digitalWrite(triggerRS, LOW);

  rearDuration = pulseIn(readRS, HIGH, timeout);

  if (frontDuration > timeTrackWidth) {
    frontDuration = timeTrackWidth;
  }

  if (rearDuration > timeTrackWidth) {
    rearDuration = timeTrackWidth;
  }
}
```

Appendix B: Program for Data Analysis (MATLAB)

```
clc
clear all
load("line5Speed1.mat")
time1= line5findmaxspeed1(:,1);
v1= line5findmaxspeed1(:,2:end);
p_vs_t1 = [0, 0];

for i = 2:1:17
     % For some reason sensor 16 seems broken? There is no voltage above
     % the usual noise threshold. Ended up using a minimum voltage of 4 to
     % signify positive, as the offset was giving way too many false
     % positives to clean up.
    [~, time1a] = findpeaks(v1{:,i-1}, 250, 'MinPeakHeight', 2, 'MinPeakDistance', 0.5);
    position1 = (i - 1).*ones(size(time1a, 1) , 1);
    p_vs_t1 = [p_vs_t1; [time1a, position1]];
    % Could be optimized by preallocation
end
p_vs_t1 = sortrows(p_vs_t1);
time1a = p_vs_t1((3:end), 1) - p_vs_t1(3,1);
position1 = p_vs_t1((3:end), 2); % sensor 9 is initial position = 0
position1 = (position1-9)*43;
actual_time1 = linspace(0, time1a(end), 100);
smooth_position1 = smoothdata(p_vs_t1, 'lowess');
position_fit1 = fit (time1a, position1, 'smoothingspline');
velocity_fit1 = differentiate(position_fit1, time1a);
figure(1);
position_fit_y1 = feval(position_fit1, actual_time1);
plot(actual_time1, position_fit_y1, time1a, velocity_fit1);
ylabel('cm/s and cm')
xlabel('time (s)');
legend('Postition', 'Velocity', 'location', 'best');
disp("Trail 1")
maxlaunchSpeed1 = max(velocity_fit1)
trackTime1 = 17.32;
timeTrack1 = find(time1a>17.3 & time1a<18);
trackSpeed1 =abs(velocity_fit1(timeTrack1(1)))
timeBounce1 = 32.94;
BounceTime1 = find(time1a>32.9 & time1a<34);
BounceSpeed1 = abs(velocity_fit1(BounceTime1(1)))

%% Trial 2
% line5_findmaxspeed2 has no useful data, so used line5_findmaxspeed3
%%clear all
load("line5Speed3.mat")
time2= line5findmaxspeed3(:,1);
v2= line5findmaxspeed3(:,2:end);
p_vs_t2 = [0, 0];
```

```matlab
for i = 2:1:17
     % For some reason sensor 16 seems broken? There is no voltage above
     % the usual noise threshold. Ended up using a minimum voltage of 4 to
     % signify positive, as the offset was giving way too many false
     % positives to clean up.
    [~, time2a] = findpeaks(v2{:,i-1}, 250, 'MinPeakHeight', 2, 'MinPeakDistance', 0.5);
    position2 = (i - 1).*ones(size(time2a, 1) , 1);
    p_vs_t2 = [p_vs_t2; [time2a, position2]];
    % Could be optimized by preallocation
end
p_vs_t2 = sortrows(p_vs_t2);
time2a = p_vs_t2((2:end), 1)- p_vs_t2(2,1);
position2 = p_vs_t2((2:end), 2);
position2 = (position2-9)*43;
actual_time2 = linspace(0, time2a(end), 100);
smooth_position2 = smoothdata(p_vs_t2, 'lowess');
position_fit2 = fit (time2a, position2, 'smoothingspline');
velocity_fit2 = differentiate(position_fit2, time2a);
figure(1);
position_fit_y2 = feval(position_fit2, actual_time2);
plot(actual_time2, position_fit_y2, time2a, velocity_fit2);
ylabel('cm/s and cm')
xlabel('time (s)');
legend('Postition', 'Velocity', 'location', 'best');
disp("Trail 2")
maxlaunchSpeed2= max(velocity_fit2)
trackTime2 = 18.67;
timeTrack2 = find(time2a>18.4 & time2a<19.2);
trackSpeed2 =abs(velocity_fit2(timeTrack2(1)))
timeBounce2 = 33.5;
BounceTime2 = find(time2a>33.2 & time2a<33.9);
BounceSpeed2= abs(velocity_fit2(BounceTime2(1)))


%% Trial 3
% line5_findmaxspeed6
%clear all
load("line5Speed6.mat")
time3= line5findmaxspeed6(:,1);
v3= line5findmaxspeed6(:,2:end);
p_vs_t3 = [0, 0];

for i = 2:1:17
     % For some reason sensor 16 seems broken? There is no voltage above
     % the usual noise threshold. Ended up using a minimum voltage of 4 to
     % signify positive, as the offset was giving way too many false
     % positives to clean up.
    [~, time3a] = findpeaks(v3{:,i-1}, 250, 'MinPeakHeight', 2, 'MinPeakDistance', 0.5);
    position3 = (i - 1).*ones(size(time3a, 1) , 1);
    p_vs_t3 = [p_vs_t3; [time3a, position3]];
    % Could be optimized by preallocation
end
p_vs_t3 = sortrows(p_vs_t3);
time3a = p_vs_t3((2:end), 1)- p_vs_t3(2,1);
position3 = p_vs_t3((2:end), 2);
position3 = (position3-9)*43;
actual_time3 = linspace(0, time3a(end), 100);
smooth_position3 = smoothdata(p_vs_t3, 'lowess');
position_fit3 = fit (time3a, position3, 'smoothingspline');
velocity_fit3 = differentiate(position_fit3, time3a);
figure(1);
position_fit_y3 = feval(position_fit3, actual_time3);
plot(actual_time3, position_fit_y3, time3a, velocity_fit3);
ylabel('cm/s and cm')
xlabel('time (s)');
legend('Postition', 'Velocity', 'location', 'best');
disp("Trail 3")
maxlaunchSpeed3 = max(velocity_fit3)
trackTime3 = 21.67;
timeTrack3 = find(time3a>21.6 & time3a<23);
trackSpeed3 = abs(velocity_fit3(timeTrack3(1)))
timeBounce3 = 42.57;
BounceTime3 = find(time3a>42.1 & time3a<44);
BounceSpeed3 = abs(velocity_fit3(BounceTime3(1)))

%% Trial 4
% line5_findmaxspeed5
%clear all
load("line5Speed5.mat")
time4= line5findmaxspeed5(:,1);
v4= line5findmaxspeed5(:,2:end);
p_vs_t4 = [0, 0];

for i = 2:1:17
     % For some reason sensor 16 seems broken? There is no voltage above
     % the usual noise threshold. Ended up using a minimum voltage of 4 to
     % signify positive, as the offset was giving way too many false
     % positives to clean up.
    [~, time4a] = findpeaks(v4{:,i-1}, 250, 'MinPeakHeight', 2, 'MinPeakDistance', 0.5);
    position4 = (i - 1).*ones(size(time4a, 1) , 1);
    p_vs_t4 = [p_vs_t4; [time4a, position4]];
    % Could be optimized by preallocation
end
p_vs_t4 = sortrows(p_vs_t4);
time4a = p_vs_t4((7:end), 1)- p_vs_t4(7,1);
position4 = p_vs_t4((7:end), 2);
position4 = (position4-9)*43;
actual_time4 = linspace(0, time4a(end), 100);
smooth_position4 = smoothdata(p_vs_t4, 'lowess');
position_fit4 = fit (time4a, position4, 'smoothingspline');
```

```
velocity_fit4 = differentiate(position_fit4, time4a);
figure(1);
position_fit_y4 = feval(position_fit4, actual_time4);
plot(actual_time4, position_fit_y4, time4a, velocity_fit4);
ylabel('cm/s and cm')
xlabel('time (s)');
legend('Postition', 'Velocity', 'location', 'best');
disp("Trail 4")
maxlaunchSpeed4 = max(velocity_fit4)
trackTime4 = 19.67;
timeTrack4 = find(time4a>21.6 & time4a<23);
trackSpeed4 = abs(velocity_fit4(timeTrack4(1)))
timeBounce4 = 41.27;
BounceTime4 = find(time4a>40.0 & time4a<42);
BounceSpeed4 = abs(velocity_fit4(BounceTime4(1)))

%%
set(0,'defaultAxesFontSize',18)
ave_maxlaunchSpeed = (maxlaunchSpeed1+maxlaunchSpeed2+maxlaunchSpeed3+maxlaunchSpeed4)/4
ave_trackSpeed = (trackSpeed1+trackSpeed2+trackSpeed3+trackSpeed4)/4
ave_BounceSpeed = (BounceSpeed1+BounceSpeed2+BounceSpeed3+BounceSpeed4)/4

% Using the 2nd trial

ave_trackTime = (trackTime1+trackTime2+trackTime3+trackTime4)/4;
ave_timeBounce = (timeBounce1+timeBounce2+timeBounce3+timeBounce4)/4;

figure(1)
hold on
plot(actual_time1, position_fit_y1) %%time1a, velocity_fit1);
plot(actual_time2, position_fit_y2) %% time2a, velocity_fit2);
plot(actual_time3, position_fit_y3) %%time3a, velocity_fit3);
plot(actual_time4, position_fit_y4) %%time4a, velocity_fit4)
hold on
plot([ave_trackTime ave_trackTime],[-100 250],'--k');
text(19.5,100,'Start Tracking')
text(38,100,'Stop Tracking')
hold on
plot([ave_timeBounce ave_timeBounce],[-100 250],'--k');
set(gcf, 'Position',  [20, 20, 1200, 800])
title('Displacement of Roll Car vs Time')
ylabel('Displacement (cm)')
xlabel('Time (s)');
legend('Displacement1','Displacement2',...,
'Displacement3','Displacement3','location',...,
'northeast');
saveas(gcf,'figure1.png')


figure(2)
```

```
hold on
plot(time1a, velocity_fit1);
plot(time2a, velocity_fit2);
plot(time3a, velocity_fit3);
plot(time4a, velocity_fit4);
plot([ave_trackTime ave_trackTime],[-100 250],'--k');
text(19.5,100,'Start Tracking')
text(38,100,'Stop Tracking')
hold on
plot([ave_timeBounce ave_timeBounce],[-100 250],'--k');
set(gcf, 'Position',  [20, 20, 1200, 800])
title('Velocity of Roll Car vs Time')
ylabel('Velocity (cm/s)')
xlabel('Time (s)');
legend('Velocity1','Velocity2',...,
'Velocity3','Velocity3','location',...,
'northeast');
saveas(gcf,'figure2.png')
```