ECE 183DA - Design of Robotic Systems I

Lab Report - Lab 1

William Argus
Lab Group: Buffalo
January 24th, 2019

Will Argus
004610455

Contents:

1. Introduction

The goal in this lab is for the group to construct, wire, and program a simple, two wheeled robot and characterize the sensors on it. Specifically, a theoretical model of robot state dynamics will be mathematically derived and models of the robot's sensor and actuator responses will be generated based on data collected in this lab. The pictures, videos and data generated, as well as the code that is used, modified, and generated for this lab will be uploaded in the git repository for this lab so that it can be referenced at any time.

The robot uses an ESP8266 to take in data from the sensors and send command signals to the two servos attached to wheels. This ESP8266 chip can communicate with a computer using either a wired connection via micro-USB, or a wireless connection courtesy if its on-board Wi-Fi module. The set of sensors is two laser range finders, and a magnetometer. The model numbers for these two sensors are GYVL53LOX and MPU9250 IMU, respectively. The magnetometer gives the value of the angle between the vector in the direction the car is facing and the vector pointing towards magnetic north. The two laser range finders can be used in a closed environment, such as a box, by the robot, in order to find its position. In order words, because the robot moves in 2D space, it only needs two values to completely define its translational state, and the robot is able to use these laser range finders in conjunction with the magnetometer to determine its translational state relative to the origin. This is given by a vector made up of an x-value and a y-value. The magnetometer provides the angle of orientation of the robot, giving the rotational state of the robot, thereby allowing the three sensors to completely define the translational and rotational state of the robot in 2D. The actuation of the wheels is done by two continuous servos, one on each wheel, with model number FS90R. The direction and speed of each servo is controlled independently of the other, by varying the PWM signal sent to it by the

ESP8266 chip. In this way, the robot is able to move in a straight line or turn itself 360 degrees about the vertical axis. The breakout board for ESP8266 is used to connect everything together with removable jumper wires, rather than being soldered to the ESP8266. The robot is powered by a rechargeable lithium-ion battery, allowing it to move freely without a tethered power cord.

The robot is constructed around a folded paper frame that houses the battery, breakout board, and servos, and allows the sensors to sit atop the robot to collect data. The wheels of the robot are spaced 85mm apart and are 200mm in diameter. The back end of the robot is supported by a fold of a paper that drags along the ground approximately 70mm behind the centerline of the front wheels.

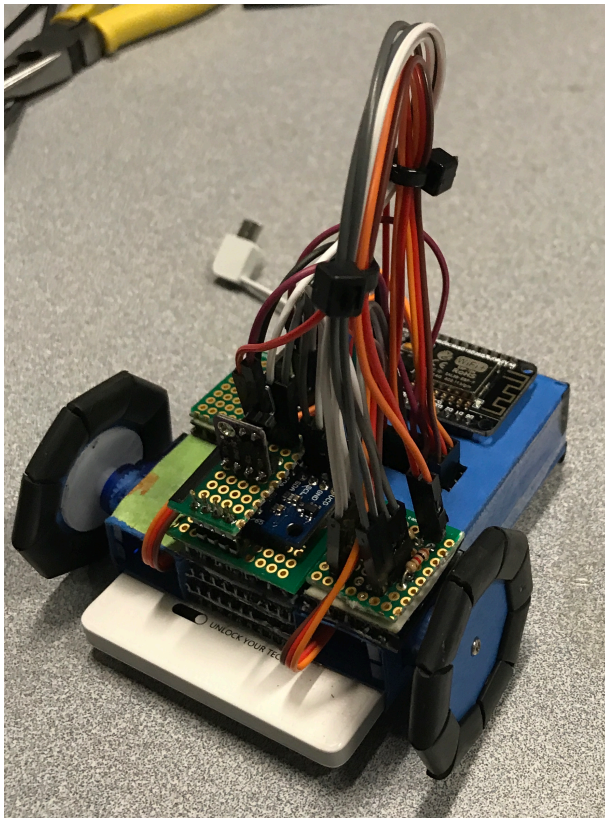

Figure 1, Constructed robot used for this lab.

2. Methods

2.1 Setup

In order to set-up the paperbot to take data with its sensors, the laser range detectors and magnetometer were soldered to protoboards with pins to plug in jumper wires to each of the pins on the respective sensors, essentially creating breakout boards that could securely attached to the robot. A protoboard was also created with pins linking the four SDA and SCL pins for I2C communication amongst the ESP8266, the two laser range finders, and magnetometer. The physical car was then constructed using the paper sheet supplied to the team and the instructions found in the git repository. The battery, breakout board, and servos were appropriately placed inside of car and tape was used to secure it all together. The wheels were glued together and screwed into the servo outputs. The ESP8266 was then plugged into the breakout board, and the servos, magnetometer, and laser range finders were all wired to the ESP8266 through the breakout board using jumper wires. The wiring was done according to the wiring found on the git repository.

2.2 Theoretical, Geometrical Model

The car's dynamics were separated into two modes: moving in a straight line and turning. In order to make this model easier to derive, the magnetometer was placed on the car such that its center was on the midpoint of the line between the two wheels. This made the math easier, as when the car rotates, its wheels rotate with the same magnitude, but opposite direction, so the car rotates about the vertical axis passing through the point where the magnetometer takes its measurements. This means that the clockwise and counter clockwise rotations can be treated as the same. The theoretical geometrical model will be presented in the results section.

2.3 Data Driven Actuator Model

The goal of finding an input-output relationship of the servos required the set-up and data collection on them. In the data-driven model that is the end goal, the input is the PWM signal sent to the servo from the ESP8266 and the output is the angular speed at which the servo rotates. As the car will be driving in both a straight line and turning, it is necessary to collect data and generate models for all cases: going forward in a straight line, going backward in a straight line, turning about a centerpoint clockwise, and turning about a centerpoint counter clockwise. Note that no modifications were made to the sample code meant for controlling the PWM signal from the ESP8266 to servos, as the code allowed the team to wirelessly control, from a phone or laptop, the servo speed and direction by altering the PWM sent to each servo. This was sufficient for collecting the data that was needed to generate the data driven actuator model.

The first approach used to collect data did not result in the collection of any meaningful data for use in creating the models. The left wheel was suspended in the air and the speed at which the wheel rotated was recorded at different PWM signals. The same was done with the right wheel (See Figure 2). The data is collected during this first attempt at data collection can be viewed in the git repository. This approach had several problems that invalidated the data that was collected. The first was that this data was only good for modeling the car moving in a straight line. As the car will also be turning when it moves, a model to predict the angular velocity of the car itself when it is turning was necessary. The second problem was made evident when the data was used to try to move the car in straight line. The idea behind collecting the speed of wheels free spinning in the air was that it could be graphed as angular velocity vs PWM signal, so PWM signals could be easily selected to cause the servo to rotate at the desired speed. When this approach was used with the data collected using the method described above,

however, the car did not move in a straight line, despite the fact that the wheels should have been moving with the same angular velocity according to the graph of the angular velocity vs PWM signal. It was then realized that spinning the wheels in the air would not be sufficient to collect data for a sensor model, as it failed to take into account noise such as wheel slippage and the increased load on the servos.

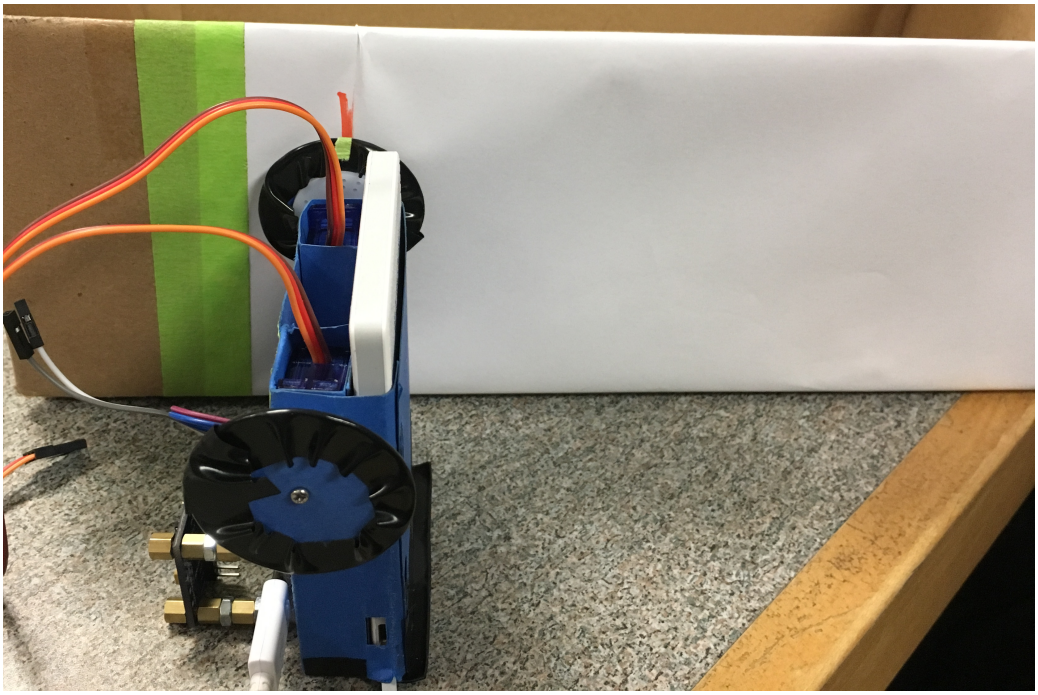

Figure 2, Setup used to measure angular velocity. This was the first setup used to try to measure angular velocity of the wheels. The data from this was faulty because it did not take into account noise that the robot's servos experience under operation.

After the first failure at collecting meaningful data, the strategy for data collection was adjusted so that the data would take into account noise such as the wheel slippage and servo load mentioned above. The approach was to place the car on a flat table, and fix the PWM signal sent to one servo, and then experimentally determine the signal that should be sent to the other servo such that the car would move in a straight line. Once these two values, right servo and left servo PWM signal, were found. The car would be placed at the start of a measured distance of 45

centimeters and the time for it to travel those 45 centimeters in a straight line was measured. This was then used to determine straight line speed of the car for that set of PWM values. These was repeated until there were 5 pairs of PWM values for moving forward in a straight line, and 5 pairs of PWM values for moving backward in a straight line. The corresponding straight line speed for each of these 10 pairs of PWM values was calculated using the approach described above; measuring the time to travel 45 centimeters in a straight line. Next, 10 pairs of PWM values, 5 for clockwise and 5 for counterclockwise rotation, were found for the rotation of the car about the centerpoint between its two wheels. Once these 10 pairs of PWM signals were found, the angular velocity of the car at each of these pairs of values was found by measuring the time it took the car to travel 360 degrees. This data was then plotted and used to create the data driven actuator model that takes the PWM signal as input and gives the angular velocity as output. It should be noted that in the case of straight line motion, the angular velocity outputted is that of the wheels, while in the case of rotational motion, the angular velocity outputted is that of the entire car. The videos taken showcasing this data collection can be found in the git repository, linked in the appendix.

Since this data was recorded by having the car actually drive, the data, and therefore the corresponding model created from it, takes into account the noise, such as wheel slippage and servo load. This means the data driven actuator model takes into account noise and uncertainty as well.

2.4 Data Driven Sensor Model

The final step of this lab was the creation of data driven models of the sensors on the robot. The data for the laser range sensors was collected by measuring the sensor output of distance from the sensor to the obstacle to the known actual distance from the car to obstacle. The uncertainty and noise in this model is the fact that there is an offset from where the sensor is mounted on the robot to the actual edge of the robot. By measuring the distance from the robot to an obstacle and comparing it to the reading of distance from the sensor to the obstacle, this noise is taken into account when creating the data driven model of this sensor.

The sample code for taking data from the laser range sensors was modified slightly in order to take data for this part of the lab. The only difference was that the sample code was changed to output only the raw values of the sensor for which data was being collected. This was done to make the data easier to import into MS Excel. This code is available in the git repository for this project. Data was taken by placing an obstacle at different known distances from the car and recording 200 sensor readings, (See Figure 3). This data was then averaged in MS Excel. This gave a relation between actual distances to obstacles and sensor measured distances to obstacles. This data was plotted and the relation was used to create a data driven model for this sensor seen in the results section of this report.

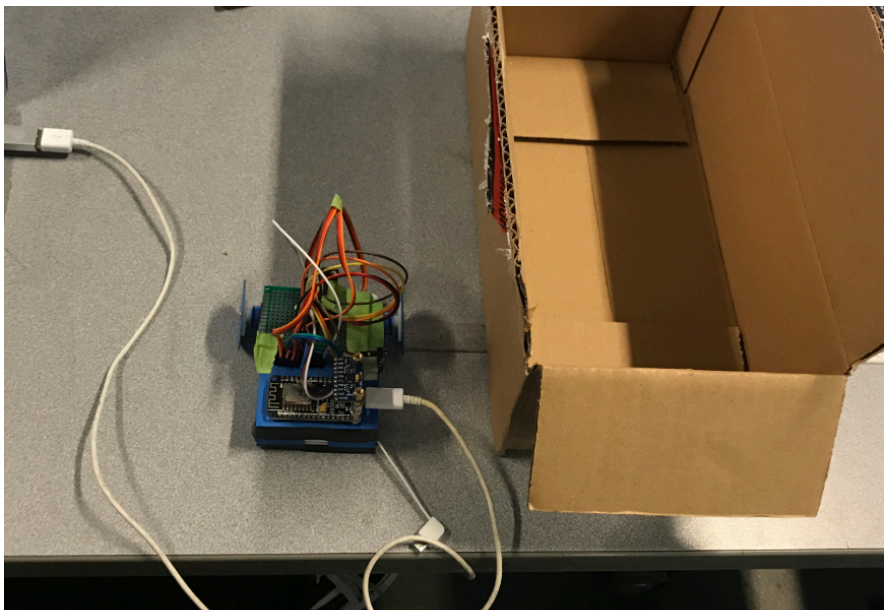

Figure 3, Data collection for right lidar sensor. This image shows how data for the right lidar sensor was collected; by placing an object at different known distances from the car and recording the output of the sensor. A similar setup was used for the front lidar sensor.

The magnetometer had to be calibrated before meaningful data could be collected from it. The first step in this was to use the website provided in the sample code, “http://www.magnetic-declination.com/”, to find the declination angle for the latitude and longitude of Los Angeles. The sample code was then edited to take this declination angle into account when outputting the sensor readings. After this was done, the magnetometer was calibrated using code created by Kris Winer and modified for the purposes of this lab. This modified code can be found in the git repository for this lab. Calibrating the magnetometer involves correcting hard iron and small iron biases. Hard iron biases are corrected in the code by taking many data points for the magnetometer and then subtracting the average of the greatest and least value from all the data along each of the axes. The purpose of this is to center the data from each axis about a common origin, as uncalibrated data appears as spheres not uniform about the origin, while data after calibration is uniform about the origin. The soft iron biases were corrected using this code by

scaling the data from each of the axes using the mean, maximum, and minimum value from each axis. The goal of this was to make all the axes equal in magnitude. After this calibration code ran, it displayed on the serial monitor the 3 bias values and the 3 scaling values used to correct the hard and soft iron biases, respectively. See the screenshot of these 6 values below.

```
Finished for loop!
Mag Calibration done!
MPU9250 Biases:
        Mx Bias: 79.000000
        My Bias: 35.000000
        Mz Bias: 2.000000
MPU9250 Scales:
        Mx Scaling Factor: 0.679012
        My Scaling Factor: 0.833333
        Mz Scaling Factor: 3.055556
```

Figure 4, Hard iron biases and soft iron scaling values from magnetometer calibration code.

Once the magnetometer was calibrated, data could be collected. First, an iPhone using a third party compass app, which told the user the strength of signal to ensure accuracy, was used to find the direction of north. Then a printed compass was taped to a table with its north aligned with the north on the phone. The phone was then taken away to avoid its compass interfering with the magnetometer and the car was placed at the center of the compass, facing north such that the +x direction on the magnetometer also faced north.

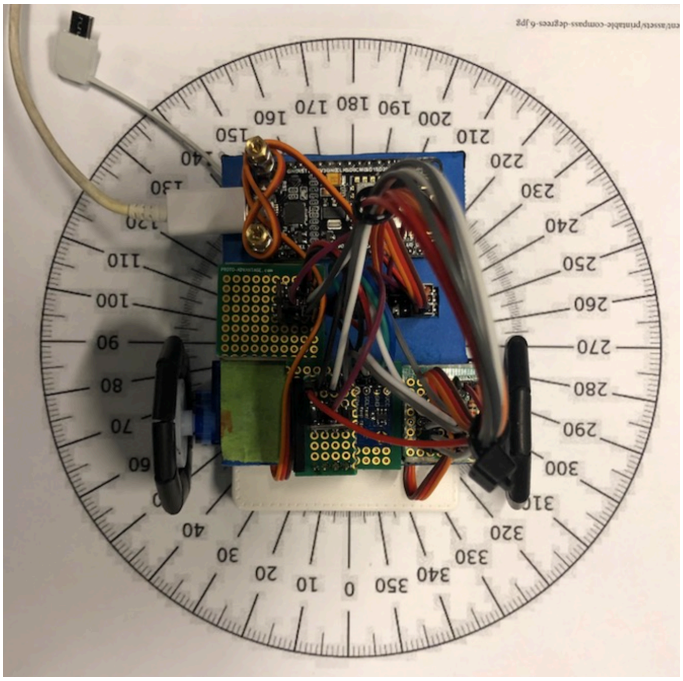

Figure 5, Robot on oriented compass. The robot is show here placed on a compass that was already oriented north using a compass. The purpose of this was to check if the calibrated magnetometer gave the correct readings, or if an offset was necessary.

The values of the magnetometer were collected to determine if the magnetometer's zero degree reading was aligned with north, or if an offset value would need to be added into the code such that the value given by the magnetometer when facing north was zero. As the magnetometer read zero degrees when facing north, no offset was needed. This is how it should be, as the declination angle that was found earlier and added to the code essentially acts as the offset when we are at the longitude and latitude of UCLA. Data from the magnetometer was then collected as it moved around the compass in increments of thirty degrees, and ensure that it matched up with the reading of the iPhone's compass, represented by the printed compass taped to the table. This data, along with the code used to gather it, can be found in the git repository. The data collected was then used to verify that the magnetometer is calibrated and working correctly. This will be shown later in this report, in the results section.

3. Results

3.1 Theoretical, Geometrical Model

Variables:

P: position of robot in 2D space

v: linear velocity of robot

N: noise associated

θ: Orietnation of the car in 2D space

ω: Angular velocity of the robot

D: distance from rotational axis to the car's reference point (the top right of the car)

Robot state update dynamics

The objective of this section is to mathematically find a theoretical, geometric model of the robot state update dynamics which will be of the form:

$$x'(t) = x_{t+1} = f(x_t, u_t, t) = Ax_t + Bu_t$$

In order to be in complete control of the car, the movements of the car are split into two cases:

Case 1: Traveling along a straight line (forward, backward)

$$p_{x,t+1} = p_{x,t} + (v_t + N_v)t * cos\theta_t$$

$$p_{y,t+1} = p_{y,t} + (v_t + N_v)t * sin\theta_t$$

$$\theta_{t+1} = \theta_t$$

Expressing in matrix form,

$$x_{t+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{x,t} \\ p_{y,t} \\ \theta_t \end{bmatrix} + \begin{bmatrix} (v_t + N_v)t * cos\theta_t \\ (v_t + N_v)t * sin\theta_t \\ 0 \end{bmatrix}$$

Which agrees with the form:

$$x_{t+1} = I_3 x_t + Bu_t = Ax_t + Bu_t$$

Case 2: Turning about its rotational axis (clockwise, counter-clockwise)

Note that $p_{x,t+1} \neq p_x$ and $p_{y,t+1} \neq p_y$ because rotational axis is different from the car reference point.

$$p_{x,t+1} = p_{x,t} - D[cos\theta_t - cos\theta_{t+1}]$$

$$p_{y,t+1} = p_{y,t} + D[sin\theta_t - sin\theta_{t+1}]$$

$$\theta_{t+1} = \theta_t + (\omega_t + N_\omega)t$$

Expressing in matrix form,

$$x_{t+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{x,t} \\ p_{y,t} \\ \theta_t \end{bmatrix} + \begin{bmatrix} -D[cos\theta_t - \cos(\theta_t + (\omega_t + N_\omega)t)] \\ D[sin\theta_t - \sin(\theta_t + (\omega_t + N_\omega)t)] \\ (\omega_t + N_\omega)t \end{bmatrix}$$

Which agrees with the form:

$$x_{t+1} = I_3 x_t + Bu_t = Ax_t + Bu_t$$

Note that the equations work for all θ_t.

3.2 Data Driven Actuator Model

Raw data used to derive the data driven actuator model can be found in the git repository. Here, the graphs of that data will be presented and examined to draw meaning from them and ultimately create the data driven actuator model.

Case 1: Forward Linear motion of robot

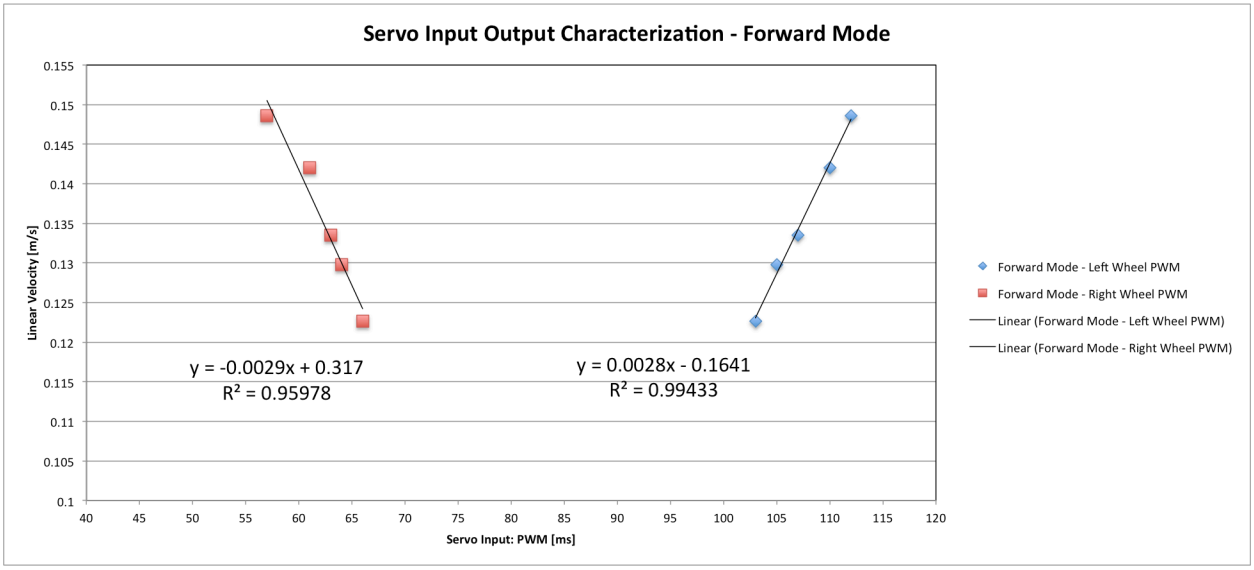

Figure 6, Forward Linear Motion. Shown is the plot of 5 pairs of PWM inputs for the right and left servos that when applied to their respective servo at the same time, result in forward motion of the car in a straight line. The models for the respective servos are given by the equations displayed on the graphs, with input being the PWM signal and the output being the linear speed of the robot corresponding to that input.

Case 2: Backward Linear motion of robot

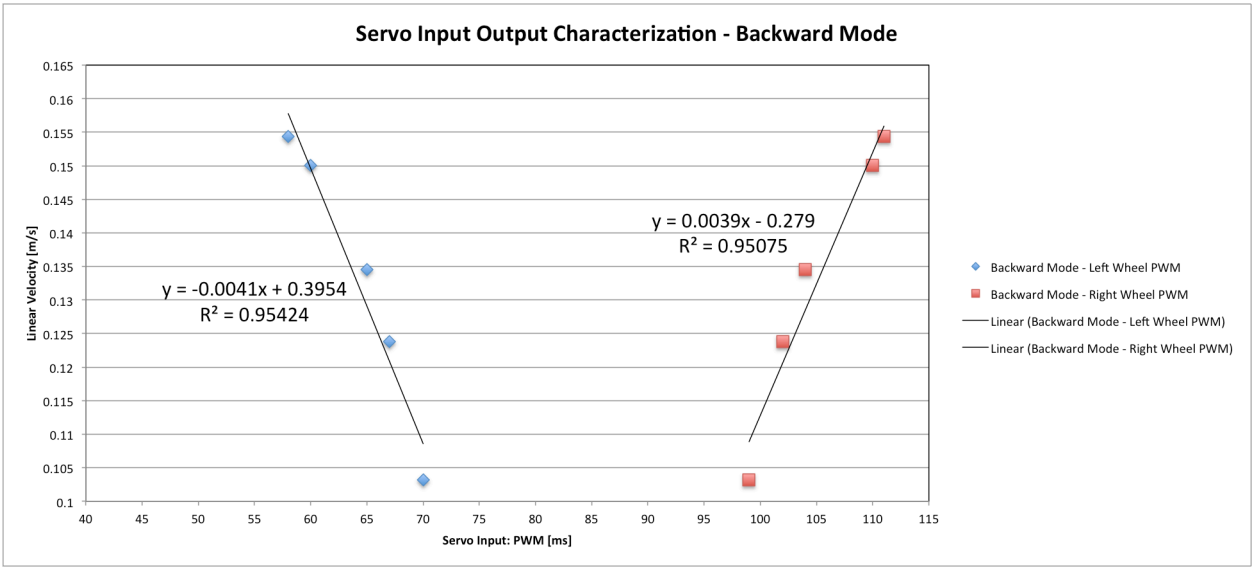

Figure 7, Backward Linear Motion. Shown is the plot of 5 pairs of PWM inputs for the right and left servos that when applied to their respective servo at the same time, result in backward motion of the car in a straight line. The models for the respective servos are given by the equations displayed on the graphs, with input being the PWM signal and the output being the linear speed of the robot corresponding to that input.

Case 3: Clockwise Rotation

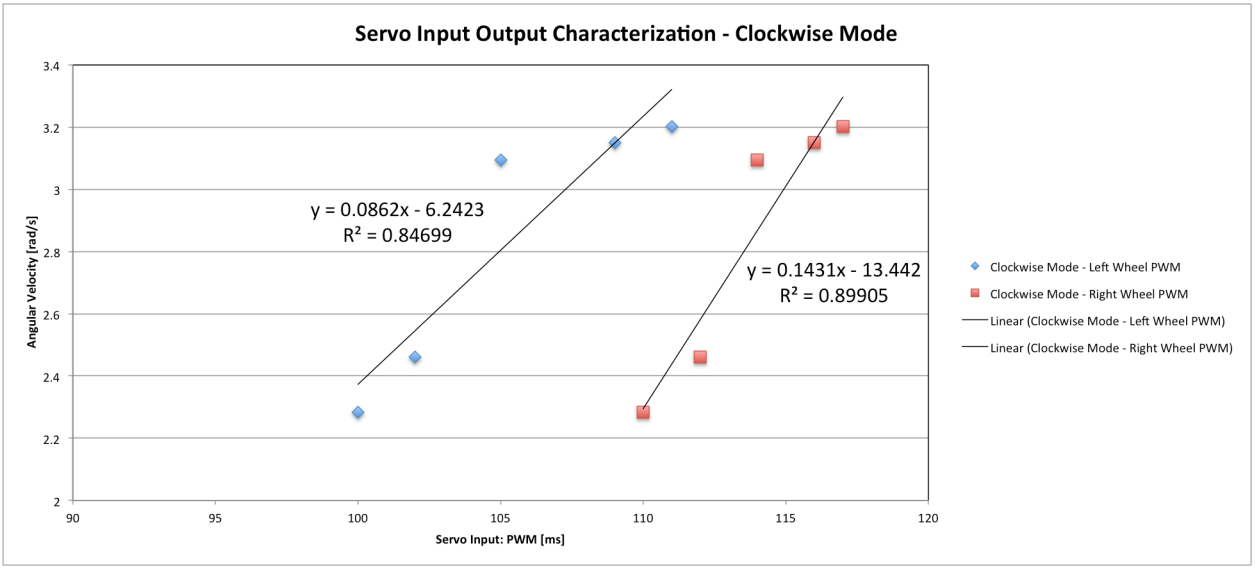

Figure 8, Clockwise rotation. Shown is the plot of 5 pairs of PWM inputs for the right and left servos that when applied to their respective servo at the same time, result in a clockwise rotation about the center point of the line between the two wheels. The models for the respective servos are given by the equations displayed on the graphs, with input being the PWM signal and the output being the angular speed of the robot corresponding to that input.

Case 4: Counter Clockwise Rotation

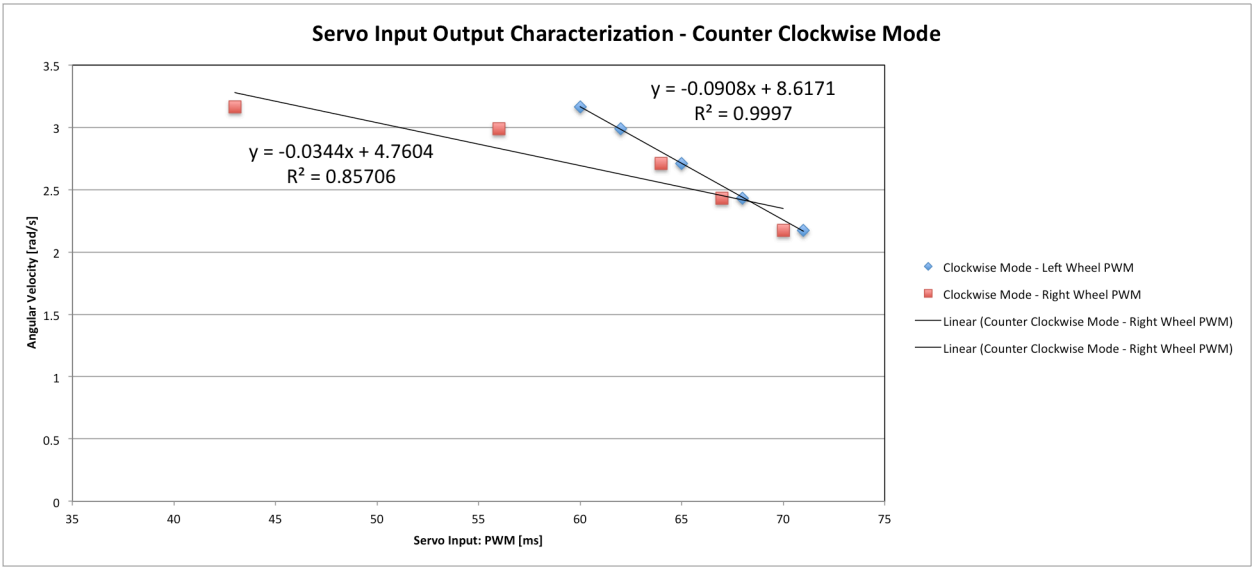

Figure 9, Counter Clockwise rotation. Shown is the plot of 5 pairs of PWM inputs for the right and left servos that when applied to their respective servo at the same time, result in a counter clockwise rotation about the center point of the line between the two wheels. The models for the respective servos are given by the equations displayed on the graphs, with input being the PWM signal and the output being the angular speed of the robot corresponding to that input.

Data Driven Actuator Model

We can convert the equations of linear speed in cases 1 and 2 to angular speed, as required by the model, by the following equation:

$$Angular\ Speed\ (\varpi)\ [rad/s]\ =\ \frac{linear\ speed\ [m/s]}{r\ [m]}, where\ wheel\ radius\ (r)\ =\ 0.02\ [m]$$

Below are the Actuator models, where x is the input of PWM [ms] signal, and y is the output of Angular speed (ϖ) [rad/s].

Case 1:

Left Wheel PWM: $y\ =\ 0.14x - 8.205$

Right Wheel PWM: $y\ =\ 0.145x - 15.85$

Case 2:

Left Wheel PWM: $y = -0.205x + 19.77$

Right Wheel PWM: $y = 0.195x - 13.95$

Case 3:

Left Wheel PWM: $y = 0.0862x - 6.2423$

Right Wheel PWM: $y = 0.1431x - 13.442$

Case 4:

Left Wheel PWM: $y = -0.0908x + 8.6171$

Right Wheel PWM: $y = -0.0344x + 4.7604$

Note that the angular velocity given in cases 1 and 2 represents the angular velocity of the wheels while the car is traveling in a straight line, while the angular velocity given in cases 3 and 4 represents the angular velocity of the car itself while turning about the center of its two front wheels.

3.3 Data Driven Sensor Model

Raw data used to derive the data driven sensor model can be found in the git repository. Here, the graphs of that data will be presented and examined to draw meaning from them and ultimately create the data driven sensor model.

Front Lidar Sensor:

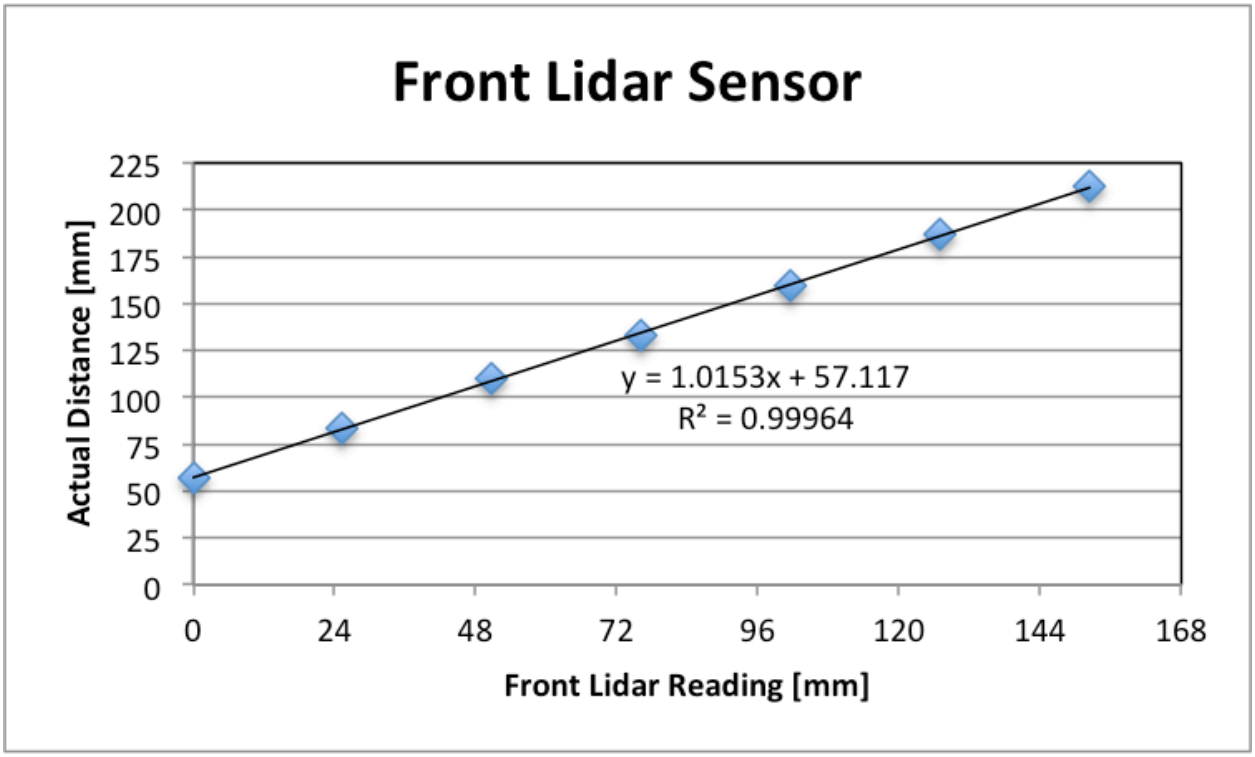

Figure 10, Front Lidar Sensor Readings at given distances. The plot displays data points that show the senor's readings of distance to an object as they correspond to the actual object distance from the car. The equation of the fit line takes the sensor reading as the input and gives the actual distance as the output.

Right Lidar Sensor:

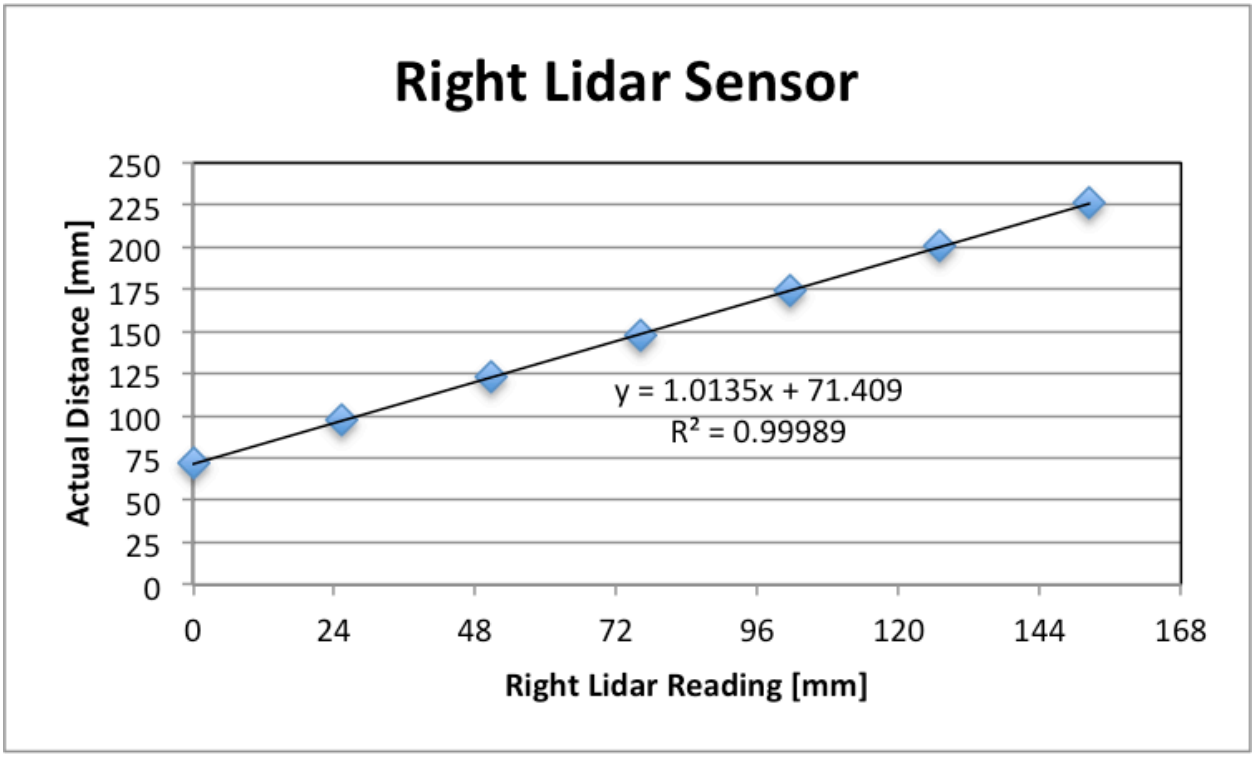

Figure 11, Right Lidar Sensor Readings at given distances. The plot displays data points that show the senor's readings of distance to an object as they correspond to the actual object distance from the car. The equation of the fit line takes the sensor reading as the input and gives the actual distance as the output.

Below are the data driven sensor models for each sensor. The models are created using the equations in the front and fight lidar sensor figures above, except they are changed to take the actual distance as the input of x and give the sensor reading as the output of y, as was the goal of this lab. Both input and output values are given in [mm].

Front Lidar Sensor: $y = 0.9849x - 56.2563$

Right Lidar Sensor: $y = 0.9867x - 70.4578$

4. Appendix

Link to Git Reposity: https://github.com/IouSC/UCLA_EE183DA_TeamBuffalo