Lab 1

EE183DA- Design of Robotic Systems

By Victor Rios, Sokchetra Eung, and Cooper Simpson

1/22/2020

Abstract

Within this lab we aimed to construct and simulate a system which could be described as a two-wheeled car equipped with various sensors. A portion of the lab was dedicated to constructing the physical hardware for the system and then calibrating all the sensors so that they provide accurate readings. In parallel to this, a mathematical model was constructed that provided an accurate simulation of car behaviour and sensor output. This mathematical model was then used to implement a software simulation on Python. The goal of this lab was to compare the real behaviour and data output of the physical car with the emulated behaviour and output of the simulation. We were unable to reach this comparison stage as we ran out of time, but the hardware calibration and extensive development of the simulator provide a great foundation that can be improved upon as the quarter continues and we develop the system in further labs.

Background

The goal of this lab was to provide us with a solid foundation in hardware interfacing/calibration, mathematical analysis and simulation development. Hardware interfacing and calibration is an important cornerstone for the field of robotics because much of what robots aim to achieve is reliant on their ability to interact and observe the world. This necessitates calibration so that accurate statements and developments can be made. The importance of mathematical analysis is that it allows us to develop and use accurate simulations of real-world phenomenon. Although, our final products are tailored for the physical world, the development and production of the real materials can be expensive and sometimes unfeasible. Thankfully, through the use of simulations, we are able to avoid such costs and save on others by trialing decisions in a simulation before implementing them.

System Description

This system takes two inputs and produces five outputs. The inputs of the system are the two PWM signals sent from the arduino to the servos. These are used to produce a translational velocity for each wheel. As for the observed outputs of the system, these included the distance measurements of two lasers (one directly in front of the car and the other pointing to the right of the car), the magnetometer readings along the x and y axis, and the gyroscope reading of rotation around the z-axis. In order to model the behaviour of the car as well as simulate the observed output we had to choose a state vector that satisfies the Markov property. The state vector we chose contained three state variables: x-position, y-position, and orientation relative to the positive x-axis. Theses variables in combination with the inputs allow us to update our state as well as produce the outputs. Furthermore, the history of input velocities can be captured in our current position and orientation. The following figure gives a general depiction of our system. For specific details regarding mathematical model and simulation, refer to mathematical analysis and simulation section of the report.

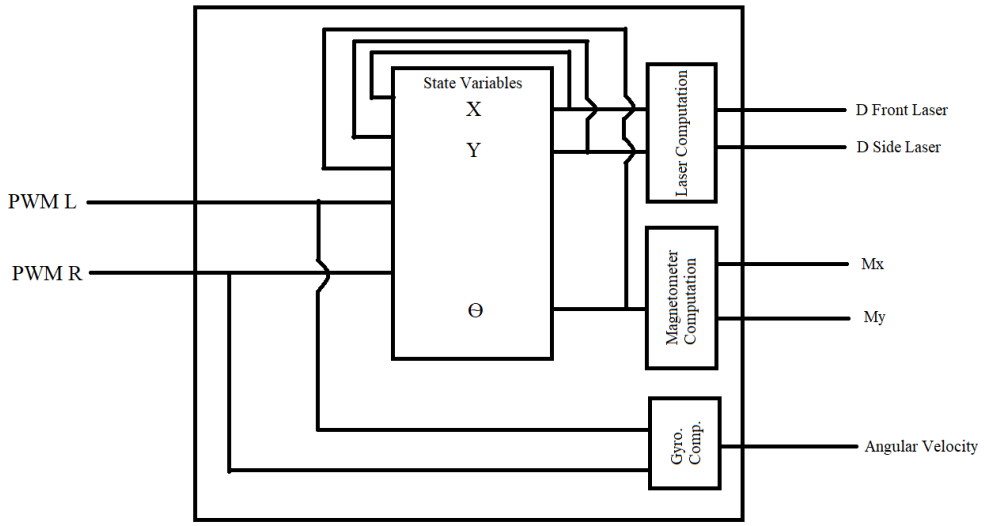

Figure 1

Hardware and Calibration

The robotic system contained several outputs, in the form of sensors, and inputs, in the form of pwm values into the actuators. This was accomplished using an ESP8266 microcontroller to produce the pwm values, a motor breakout board to route both the sensor and motor signals, and a

For sensors, the robot used a MPU-9250 inertial measurement unit to detect absolute heading and angular velocity. Additionally, it used two GYVL53L1X laser rangefinders to measure distance in the forward and right directions. For actuation, the robot used two FS90R continuous rotation servos to drive a left and right wheel.

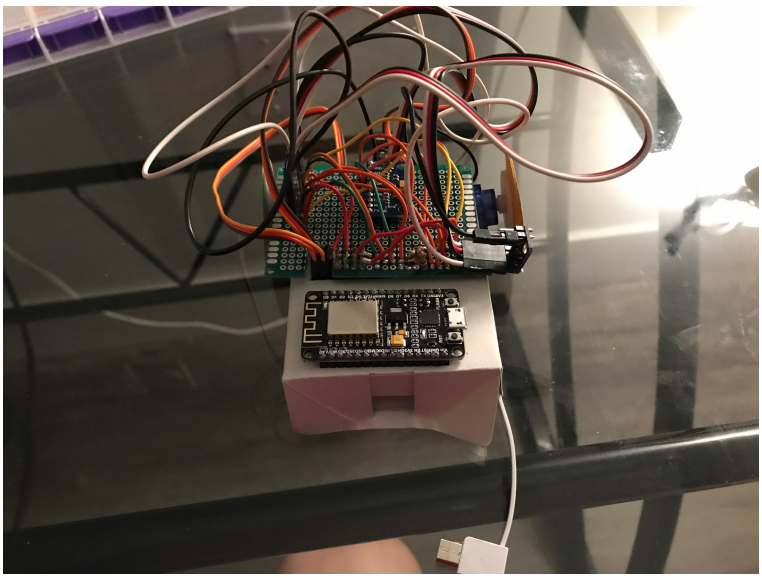

Figure 2

The MPU-9250 IMU was composed of a gyroscope, accelerometer, and magnetometer. While the accelerometer was not used in this lab, the gyroscope and magnetometer needed to be calibrated. While data from the gyroscope simply needed to be scaled depending on the maximum angular velocity requested, the magnetometer data required more detailed calibration. In order to calibrate the magnetometer, the robot was moved in a figure-eight pattern in various directions, and magnetometer readings were recorded. Of this data, minimums and maximums for each axis were recorded, and then used to calculate the bias in the magnetometer readings. While these readings can be calculated on a case by case basis, for ease of testing, the average of five trials was taken and hard-coded as a default option if calibration is not done. The raw values of these trials are shown in Table 1.

Trial	Mx bias	My bias	Mz bias	Mx Scale	My Scale	Mz Scale
1	68	-165	-4	0.97	1.05	0.98
2	64	-160	-4	1.04	1.02	0.94
3	59	-162	3	1.02	1.01	0.97
4	65	-165	0	1.02	0.96	1.02
5	64	-161	0	1.01	1.01	0.98
Average	64	-162.6	-1	1.012	1.01	0.978

Table 1

While the GYVL53L1X laser rangefinders did not need to be calibrated, limitations had to be taken into account. Namely, the minimum range of 50mm had to be considered, as well as the accuracy of +/- 5mm.

There was no calibration necessary for the two FS90R servo motors. However, it was necessary to consider the weight of the wheels, as this produced a minimum amount of servo torque required to move the robot forward or backwards, which in turn resulted in a deadzone of pwm values that did not correspond to movement. In addition to this, the wheels could slip slightly when moving, so it was important to account for the potential lack of forward or backward movement in these cases. Aside from these considerations, the behaviour of the motor was modeled is shown in Figure 3.

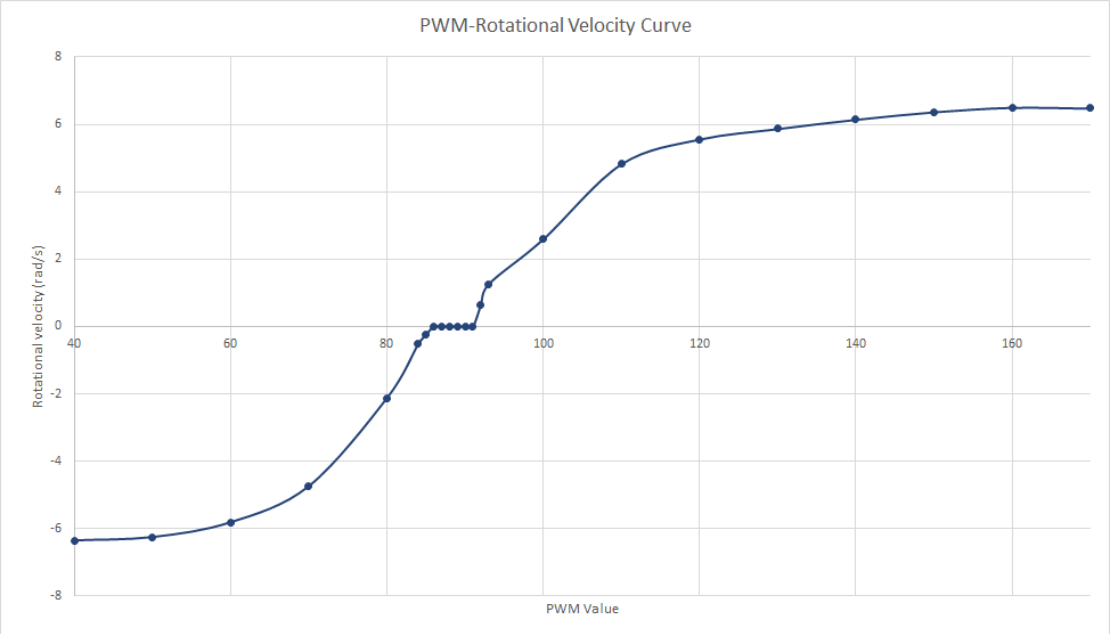

Figure 3

Mathematical Analysis and Simulation

The PWM signals were turned into angular velocities using the modeling discussed in the previous section and this model was then turned into a piecewise equation to separate the various regions within the model. The formulation for this separation is as follows:

$$w = 6.5\ rads/s \quad if\ PWM \geq 150$$
$$w = PWM * 0.0386\ rads/s + 0.582\ rads/s \quad if\ PWM \geq 110$$
$$w = PWM * 0.254\ rads/s - 23.114\ rads/s \quad if\ PWM \geq 91$$
$$w = 0\ rads/s \quad if\ PWM \geq 86$$
$$w = PWM * .297\ rads/s - 25.456\ rads/s \quad if\ PWM \geq 70$$
$$w = PWM * .0756\ rads/s - 10.038\ rads/s \quad if\ PWM \geq 60$$
$$w = -6.33\ rads/s \quad if\ PWM < 60$$

This piecewise separation was done in a ladder format because this representation is similar to how this mapping from PWM to angular velocity was coded. This mapping was used for both wheels in the simulation but this is not accurate since both wheels are not expected to be similar. This was done due to a hardware difficulty where one wheel was experiencing extreme slippage to the degree that no PWM signal could reliable get the wheel to turn. This issue will be fixed in the near future when other labs are run and each wheel will have its own mapping. For now, this model is sufficient for implementing a rudimentary simulation. We then convert the angular velocity to a translational velocity using the following equation. The value of R used was 90mm

$$v = w * R \quad where\ R\ is\ the\ radius\ of\ the\ wheel$$

After obtaining the translational velocities, we can use the car's current states and these velocities to obtain the next state. The movement of the car can be viewed as travelling around some instantaneous center of curvature at any point in time. Both translational velocities of the wheels can be seen as traveling tangent to some arc centered at this point. Each wheel is a different distance from this point, but both are perpendicular to this distance and traveling at the same angular velocity around this center of curvature. This is analogous to how two different points on a rotating disc travel with the same angular velocity. The following image provides a clear depiction of how to view this motion, and this image comes from a document written by Thomas Hellstrom, computer science department of Umea University in Sweden. A link to this document can be found in the references section and this document was the primary source for developing our state dynamics equations.

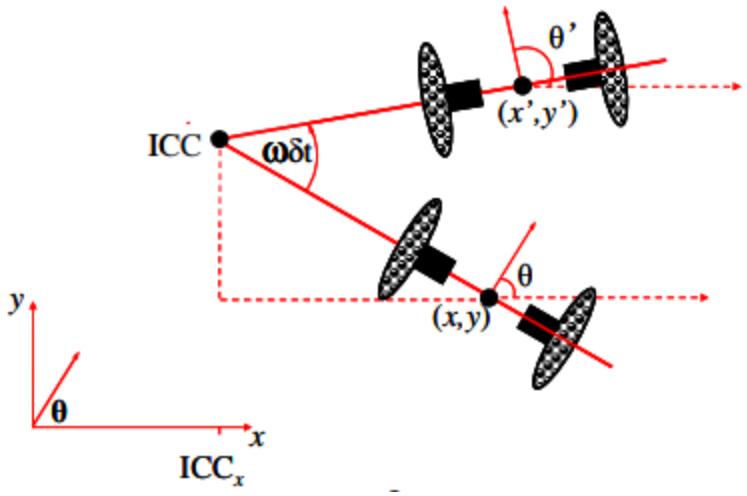

Figure 4

We started off by writing the equations relating angular velocity and to the translational velocity of each wheel. Note that R is the radius to the center of the car and W is the width of the car.

$$w(R + W/2) = vr$$
$$w(R - W/2) = vl$$

Using these two equations, we could solve for w and R, since they must be equal in both cases. We got the following equations.

$$w = (vr - vl)/W$$
$$R = W(vl + vr)/2(vr - vl)$$

After finding the radius of the curve, the position of the center of curvature can be found using our current orientation and trigonometry since the R represents the hypotenuse leg of a right triangle. The equations for finding the center are as follows

$$CCx = x - Rsin(\theta)$$
$$CCy = y + Rcos(\theta)$$

With this information we are able to update our state variables since our car's behaviour is a rotation around this center. The rotation is by an angle equal to the angular velocity calculated times the sampling period since this is how long that angular velocity holds true. The following equations make up our state dynamics description (T is the sampling period).

$$x' = (x - CCx)cos(wT) - (y - CCy)sin(wT) + CCx$$
$$y' = (x - CCx)sin(wT) + (y - CCy)cos(wT) + CCy$$
$$\theta' = \theta + wT$$

Note that when, vr is equal to vl, we travel in a straight line, which gives is an angular velocity of 0 and a radius from the center of curvature of infinity. Although this makes sense theoretically since movement along a straight line can be seen as a curve with a center infinitely far away, it will computation to fail since dividing by zero is not possible on a computer. Instead we recognize we are traveling in a straight line and use the following equations to update our state in the simulation when this special case is detected.

$$x' = vTcos(\theta) + x$$
$$y' = vTsin(\theta) + y$$
$$\theta' = \theta$$

Within our simulation, after the next state is calculated, the validity of this new state is checked, and if the state is not valid then the next state is recalculated with new information. This information usually corresponds to setting one or both of the velocities to zero and this is because the state checker checks for when the car is touching the wall and cannot move along the calculated curve. The wheel(s) that must be set to zero is determined using the car's current position and orientation since one can visualize the finite scenarios in which the car makes contact with the wall. For further details on the implementation of the state checker refer to the simulation.

The output measurements for the distance lasers were calculated by first choosing which wall the laser was pointing at and then using our orientation and immediate distance to that wall to find the expected output of the laser. To find out which wall our laser was targeting we calculate four critical angles that correspond to the angle made by the line coming from the center of the car to the corner of the arena. This could be done for all four corners since the position of all four corners never changes and our current position is known. The formula for calculating a critical angle is as follows

$$Critical\ Angle = tan^{-1}(vertical\ displacement\ from\ corner / horizontal\ displacement\ from\ corner)$$

After calculating all four critical angles, you can use them in combination with the current orientation of the car to determine which wall is being pointed to. After this, you can use your

immediate distance from the wall in combination with your orientation to determine the reading received from that laser. The process for the side laser is identical except that the orientation of the laser is equal to the orientation of the car minus 90 degrees. The following figure depicts one iteration of this process in which the laser is pointing to the top wall. To clarify, each red line has its own critical angle, but to avoid cluttering the image, only one angle was charted.

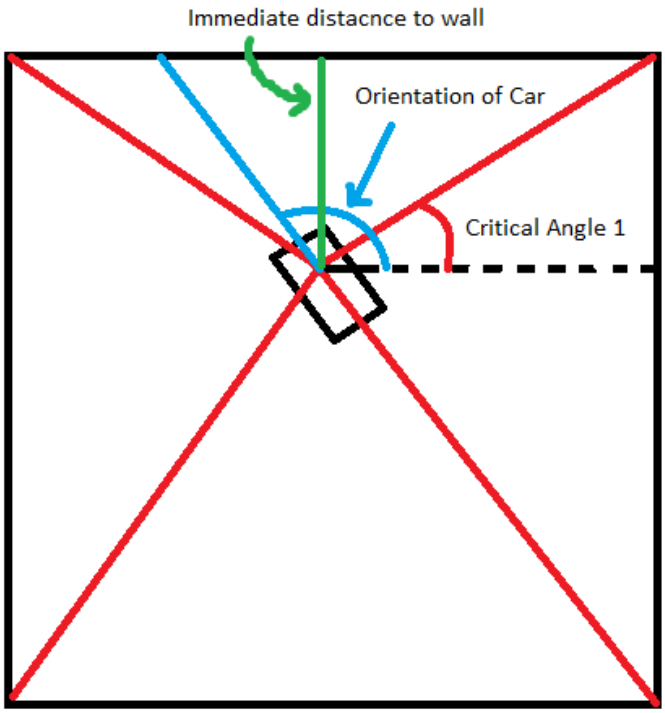

Figure 5

The magnetometer readings were directly calculated from the current orientation and a magnetic magnitude M that was gathered from experimentation. The following equations were used to calculate the magnetometer readings.

$$M_x = Msin(\theta)$$
$$M_y = Mcos(\theta)$$

Finally, the gyroscope reading was directly calculated from the velocities of the wheels. In fact, this angular velocity is equal to the angular velocity that we calculated for the motion of the car. So the following formula is used to get the gyro reading.

$$gyro = w$$

With this final equation, we have a complete mathematical description of the system, which was then used to implement in python. For further details, check the simulation code on the github repository.

Conclusion

The goal of this lab, to produce a robot and a simulation of that robot in parallel, creates an excellent testbed for future experiments. Being able to accurately simulate the inputs and outputs of this robotic system is invaluable in making testing and experimentation, especially in a group, much easier. However, our experience in this lab was not without issue. Building the sensor board was difficult, and use of a breadboard or PCB would have allowed more time to be spent developing and testing the simulation. Additionally, the magnetometer could be very difficult to calibrate; even with the given example code, many issues had to be corrected, and even the slightest issue produced unusable results. Finally, the paper frame of the robot was not very sturdy, which exacerbated the slip and deadzone issues in the modeling of the motor behavior.

References:

https://www8.cs.umu.se/kurser/5DV122/HT13/material/Hellstrom-ForwardKinematics.pdf
-Used to help model the state dynamics

https://github.com/Thequantums/ECE183DA.git
-Link to github Repository

https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration
-Used for magnetometer