DSC 180B Final Report

Pranav Deshmane, Sally Poon

February 5, 2021

When building systems for autonomous vehicle racing and driving, it is critical for the robot to localize itself within the environment it is navigating. Robot localization comprises of the robot being able to derive its current path and its heading for future motion estimation. Popular approaches often involve using GPS data solely and computer vision sensing. However, relying heavily on GPS coordinates or LiDar in open outdoor environments can lead to several issues. GPS is prone to lag and may be infeasible in harsher and unfamiliar terrain, resulting in loss of accuracy in tracking by failing to produce necessary positional information. Computer Vision approaches often depend heavily on training data and cannot always provide continouos and accurate orientation. Our problem investigates using IMU and Odometry sensors to aid in this mission by providing relevant data, position estimates, and vehicle heading in cases when GPS and other mapping are not reliable, or to supplement these approaches. IMU (Inertial Measurement Unit) provides linear acceleration, angular velocity, and magnetic force sensing ability through the use of accelorometers, gyroscopes, and occasionally magnetometers. Wheel Odometry also provide useful measurements to estimate the position of the car through the use of the wheel's circumference and rotations per second. Together, these sensors provide relevant and invaluable data that can be fused to obtain a primary heading and position estimate for the robot. Furthermore, these sensors can be fused with navigation and obstacle avoidance systems already in place to build more robust and accurate autonomous navigation models [4].

1 Goals/Purpose

We aim to achieve the following through our work:

- Understanding of IMU and Odometry Sensor to help with reliable navigation and place within robot ecosystem.
- Guides for OLA Artemis IMU setup + calibration and Odometry tuning/analysis
- Calibration procedure and Analysis of IMU sensor to ensure reliable measurements
- Tuning procedure of and Analysis Odometry to ensure accurate measurements
- Odometry derived Position Estimate
- IMU derived Primary Heading Estimate using fusion of accelerometer, gyroscope, and magnetometer readings
- IMU and Odometry data ready to be easily ingested by other subteams through ROS using package standard and custom topics
- IMU and Odometry data ready for fusion with GPS subteam within Kalman Filter if necessary for future advancement of robot localization
- Noise Reduction Strategies
- Integrating Oak camera to our robot

2 Odometry

2.1 Introduction

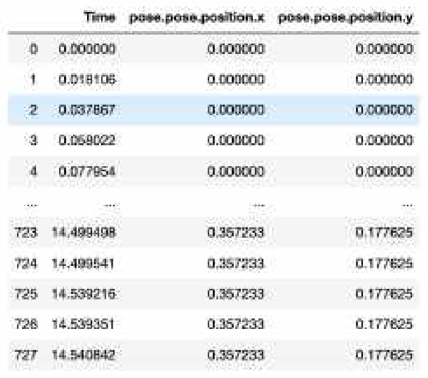
Odometry is the use of motion sensors to estimate change over time [1]. To do this, odometry requires the time, rotation per minute and steering angle. After this, we can calculate odometry by doing:

$$Position = OldPosition + Velocity * Time + 1/2 * Acceleration * Time^2 \tag{1}$$

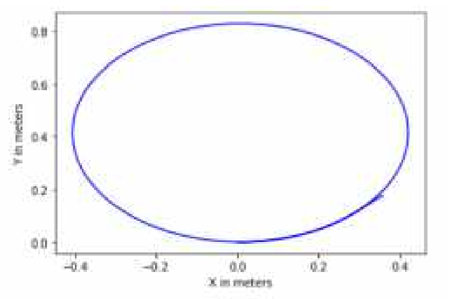
Using this equation, the robot car can predict where it's location respective to it's last position. The goal of odometry is to get a estimate of where the robot has driven to respect to its starting point. This can be problematic alone as over time it may accumulate errors since it is an estimate of the car's position [6].

2.2 Data Gathering

For odometry, the baseline framework we used was F110th Ackermann Steering [5]. When using this framework, we gathered data from our robot car and cleaned it so we only got time, estimated x and estimated y. Using these predicted positions, we were able to localize the robot and see what was its pathing.



	Time	pose.pose.position.x	pose.pose.position.y
0	0.000000	0.000000	0.000000
1	0.018106	0.000000	0.000000
2	0.037867	0.000000	0.000000
3	0.059022	0.000000	0.000000
4	0.077954	0.000000	0.000000
...	...	...	...
723	14.499498	0.357233	0.177625
724	14.499541	0.357233	0.177625
725	14.539216	0.357233	0.177625
726	14.539351	0.357233	0.177625
727	14.540842	0.357233	0.177625



2.3 Calibrating Odometry

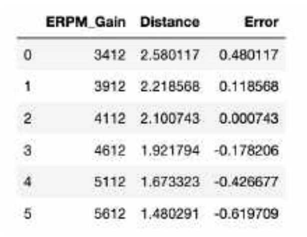
To increase the accuracy of our odometry readings, tuning has to be done on the VESC.yaml file to account both the steering and angle gain used in our equations. The equations being used are:

erpm (electrical rpm) = speed to erpm gain * speed (meters / second) + speed to erpm offset

servo value (0 to 1) = steering angle to servo gain * steering angle (radians) + steering angle to servo offset

2.3.1 ERPM Calibration

For the ERPM, we tried to find the best value for speed to erpm gain, which was only obtainable by constantly tuning and testing the value. To do this tuning, we took a tape measure and extended it by around two feet, put our car's back wheels at zero meters and drive straight. After driving straight, we can grab the distance by doing rostopic echo /vesc/odom/pose/pose/position/x. After this, if we got a distance that overshot, we decreased the speed to ERPM gain and if it undershot, we would increase the speed to ERPM gain. After testing the values of 4412, 4912, 5412, 3912, 3412, and 4012, we found that 4112 was the most accurate value with around a 0.0007 error from the actual position versus a 0.480117 ,0.118568, -0.178206, -042677, and -0.619709 error.



	ERPM_Gain	Distance	Error
0	3412	2.580117	0.480117
1	3912	2.218568	0.118568
2	4112	2.100743	0.000743
3	4612	1.921794	-0.178206
4	5112	1.673323	-0.426677
5	5612	1.480291	-0.619709

Figure 1: end result

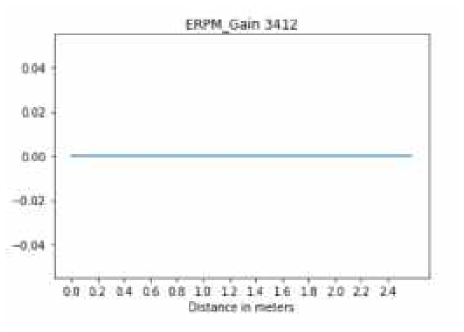


Figure 2: start

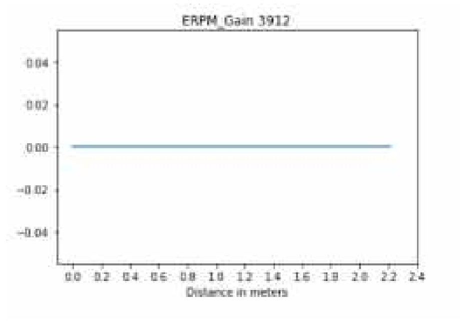


Figure 3: mid

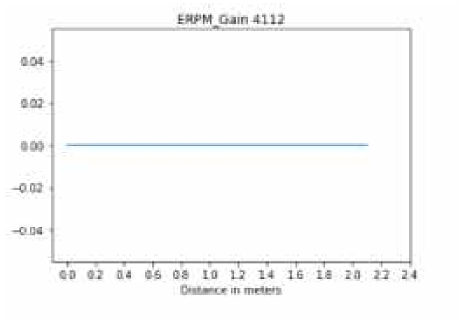


Figure 4: start

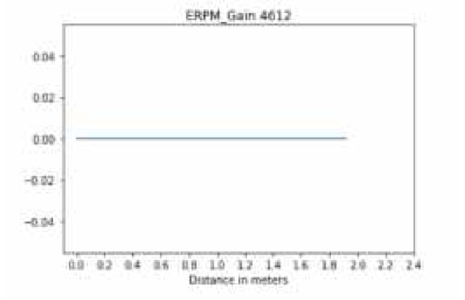


Figure 5: mid

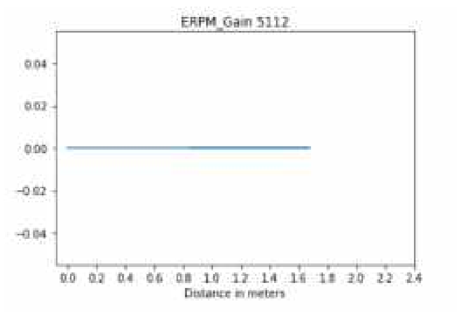


Figure 6: start

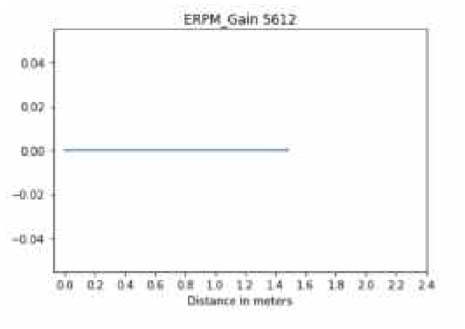


Figure 7: mid

2.3.2 MushR Steering Angle Calibration

For this part, we followed a guide from MushR [10].To test the steering angle to servo gain, we had to also have a tape measure and see the best value by running the car over and over. We had a tape measure go out to around 2.5m, then had set the back wheels to the beginning of the tape measure in a direction that makes it a T shape. To calculate what we needed as our arc, we needed to do

$$2 * R = L/2sin(beta) \tag{2}$$

and

$$beta = arctan(1/2 * tan(delta)) \tag{3}$$

given that our cars length is 0.475 and the maximum steering is 0.34. This ends up being around 2.44m. To do this test, we had to change the steering to erpm gain variable. The values had to be negative because if we set a positive steering to erpm gain, it would invert the turn. During our first tuning, we tried to do 0 however, we learned that if we set zero, it would not turn at all. During the test, our original value was 0.67 however we had to retune. This was because even though during our test it hit 2.44m, we realized that when graphing it with a constant speed, it would never hit that amount. Because of this, we decided to make our our tuning test.

Figure 8: start

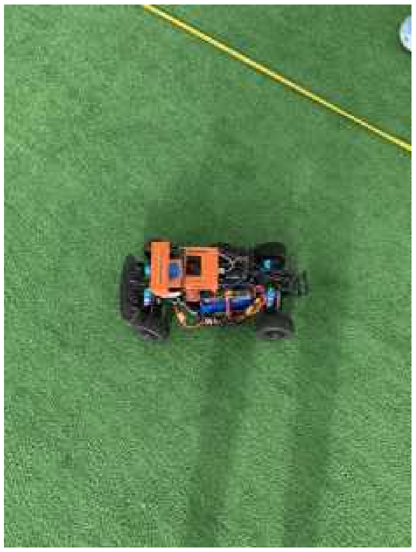
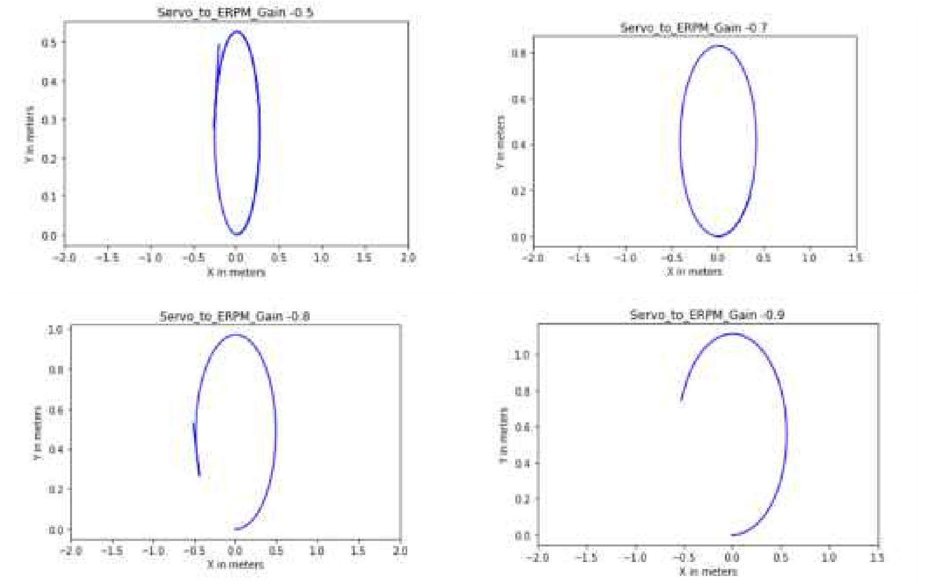


Figure 9: mid

Figure 10: end result

2.3.3 Improved Steering Angle Calibration

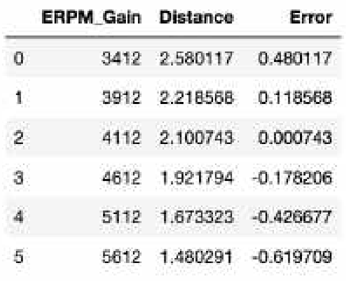
After finding that the MushR tuning test was not accurate for our needs, we decided to make our own test that uses our maximum distance value of 1.8m. For this test, we would turn the car each time and see how close it plots to 1.8m. To do this, we tried a steering to erpm gain of -0.5, which made over a full circle when we only did a half circle, showing that it was too much of a steering angle. After this, we through to increase the steering angle to -0.7 which preformed a little better but still made a full circle. We tried increasing it again to 0.8 servo to erpm gain to see how the change affects the circle because at the time, we did not know what caused it to run a full circle when we only ran a half circle. Our final try before researching more was 0.9 servo to erpm gain. Our final try before researching more was 0.9 servo to erpm gain.



After seeing that it was not giving the right predicted values, we decided to look into the file that predicts the odometry values. By going into the vesc to odom topic, we were able to find that the equation to calculate the odometry value is:
(data - steering to servo offset) / steering to servo gain current angular velocity = currentspeed * tan(current steering angle) / wheelbase

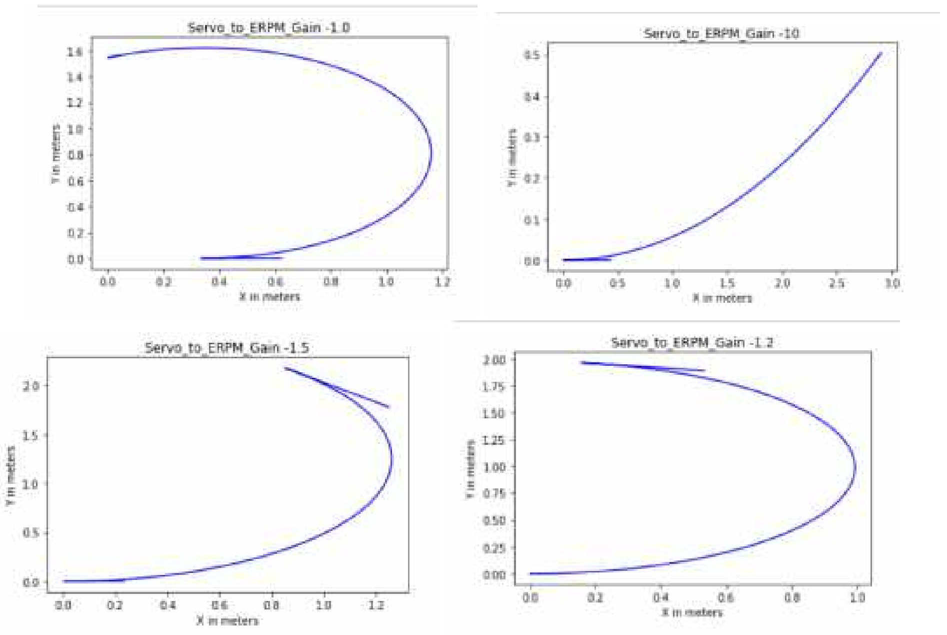
After reading the equation and testing values, we realized that the steering to servo gain directly changes the odometry prediction we would get. This means that if we increased the gain, it would start predicting a larger value and if we decreased the gain it would decrease the predicted value. Knowing this, we started changing our values again to try to perfect our odometry prediction.

For our second batch of tests, we tried starting with a -1.0 steering to erpm gain. When we did this we saw that it was close to half a circle however we wanted to ensure that by changing this servo to erpm gain to over -1, we would be getting closer to our intended half circle. Next, we tried a more extreme value of -10 and saw it only did a small arc, proving that by decreasing the servo to erpm gain we were decreasing the predicted arc value. Since we have proven the extremes of the erpm to servo gain values, we started honing into the right servo to erpm gain value. We tried -1.5 next however it still made too small of an arc. After this, we went changed the value to an erpm to servo gain of -1.2 which was very close however just shy of the amount we wanted the arc to be. The final value we tested was -1.1, which after testing gave an arc with only a 0.05m error.



	ERPM_Gain	Distance	Error
0	3412	2.580117	0.480117
1	3912	2.218568	0.118568
2	4112	2.100743	0.000743
3	4612	1.921794	-0.178206
4	5112	1.673323	-0.426677
5	5612	1.480291	-0.619709

Figure 11:



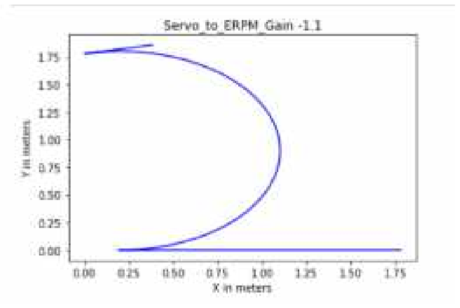


Figure 12:

2.4 Odometry Discussion/Results

The biggest issue we started with is that there was a problem with the directories of how we set our vesc file. During that time, we set the erpm to servo gain however it never changed the value when checking the erpm to servo gain parameter. To fix this issue, we had to go into forums such as the F110th and Mushr Slack and forums and ask for assistance. After discussing with them further, we found that how we set our directories was incorrect and it was not reading the vesc.yaml properly. We fixed how the VESC was reading the vesc.yaml and after that, we were able to test the erpm to servo gain.

Our first issue we started encountered was doing the steering angle tuning test and completing a 2.44m turn with a -0.67 servo to erpm gain gain. We considered this to be our final value and to validate, we started recording the predicted x and y values and graphing them to see if our tuning test was accurate. However, when we graphed the robot's pathing using the ERPM gain, we found that the odometry values overpredicted by double, showing a full circle pathing when we only drove half a circle. Because of this, we tried doing other values similar to out however we saw the same result mostly, showing that it drove half a circle. To try to understand this issue better, we went directly into the VESC node and looked at the equation used to calculate the odometry values and we found that based on the servo to erpm gain, if you decrease it even more, it will predict a smaller value, giving us a smaller arc.

Because of this, we had to retune the steering angle however since we made previous human errors, we used a constant speed of 1m/s rather than an unbounded speed. The problem however was that since our speed was bounded at 1m/s, we would never hit 2.44ms turning so we decided to tweak the test and make our own. For this new test, we wanted to make it so the distance we are trying to predict is 1.8m instead of using the kinematic model equation and tune based on that value. After this, we had to retest multiple new values such as -1.0 and -10 to prove our theory of the arc and half circle increasing and decreasing based on the servo to erpm gain. After proving our theory through testing, we decided to start honing in on the most accurate servo to erpm gain. Next we tried -1.5 which under predicted and then -1.2 which was only -0.1 off. Our final value was -1.1 with a -0.05 error.

3 Artemis IMU

3.1 Introduction

IMU or Inertial Measurement Unit allows for the sensing of acceleration, angular velocity, and magnetic fields in all X,Y,Z axes. This is accomplished through accelorementer, gyroscope, and magnetometer sensors built in within the IMU. Using these measurements Roll, Pitch, Yaw can be derived by a combination approach of all 3 sensors onboard the IMU, which allows us to gain an understanding of the robot's relative position. This is accomplished through its acceleration and angular velocity measurements, and its relative orientation by the yaw compass heading. Thus, it will be important to calibrate and thoroughly verify our IMU produces accurate readings and derived estimates [2].

Initially, due to our previous tests and seamless integration, we had decided to utilize the Sparkfun 9DoF Razor IMU for the robot. However, the Razor became an obsolete model and was no longer available to purchase. So, starting in January 2021 we decided to switch to using the Sparkfun OpenLog Artemis IMU (Dev-16832)[9], the most recent IMU from Sparkfun lineup. This IMU boasted an enhanced sensing capabilities and a simple plug and play system, but this was not the case initially. Driver installation problems with Linux, Arduino IDE issues, and extremely difficult to find ROS package were a few obstacles faced when interfacing with this specific IMU. While the Razor had a strong community supported ROS package, the Artemis did not. After a lot of digging and contacting those in the field, we were able to contact Fabrice Le Bars, who is an Assistant professor in the Robotics Topic Group at ENSTA Bretagne, Lab-STICC/CID/PRASYS. Through his help, we discovered his personally written ROS package, where he recently added support for the Artemis IMU. We also went through the a tedious back and forth process in debugging his personal visual library that was necessary to visualize the heading derived by the IMU. We then helped to point out bugs and recommendations for his official github repository to better the experience for future users. The installation and setup guide we have written is attached to this report and can also help future members of this capstone, Triton AI, and Artemis IMU users in this process. Thus, we were successfully able to calibrate and acquire the heading of the IMU which can aid in the navigation and localization of the robot. Thankfully, we were still able to attend in person lab during the COVID pandemic and have contact with the robot, which was crucial for debugging many of these steps.

It is also imperative to determine the placement and design of the IMU mount for both 1/10 and 1/5 car. Further analysis on basic mounts is included below. To design this mount we must take into account the magnetic distortions previously discussed and security of the mount itself. Then, we must find a suitable location on the car that is safe and ideally furthest from the most magnetized metals of the car. We also had major issues when trying to replicate integration steps directly on the JNX. This struggles came from the Arduino IDE unable to properly handle arm64 architecture tools needed to flash the AHRS software to the Artemis IMU. Ideally we can have all installation steps done using the JNX. Currently we are using external machine (best case running Ubuntu 20.04) to flash the software and calibrate the IMU, per the recommendation of Dusty Franklin from Nvidia and various Arduino and Jetson forums. To further mitigate the noise of the IMU, calibration procedure should be done even with the slightest changes in IMU positioning and robot modifications.

3.2 Calibration

Calibrating the Artemis IMU had to completed methodically and with care to establish reliability of its measurements. This process was similar to the Razor IMU we had worked with before, so we knew where to avoid common pitfalls. To calibrate the Artemis IMU, we made sure to follow a defined procedure by the Fabrice Le Bars ROS package [3][4]. This allowed us to configure a calibration file to calculate the final offsets and calibration parameters and easily integrate with our IMU with ROS. For the calibration of the linear acceleration in the x,y,z axes the Artemis IMU, we needed to obtain the maximum and minimum acceptable values of each axis by pointing the IMU upwards and downwards in each axis to a position expected when the IMU was to be mounted on the robot. It was imperative to reset the measurement if there were mishaps as the sensor proved to be quite sensitive, which could greatly affect the readings in future. To calibrate the gyroscope, the IMU was kept still on a flat surface for 10-15 seconds to account for the noise at rest. Calibrating the magnetometer proved to be trickier to handle due to external magnetic influences in our workspace. The magnetometer can be influenced by ”Hard” and ”Soft” iron offsets which can distort the magnetic forces around the IMU. We experienced a hard iron offset initially which can be seen in the figure below. However, after clearing the workspace of large metal objects and potentially magnetized metals, we were able to achieve satisfactory calibration.

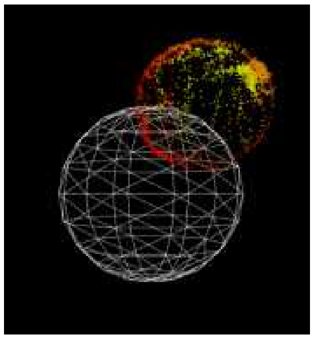


Figure 13: ”Hard” Iron Offset

In order to test whether the reliability of the measurements, we devised to collect data under the following set of conditions (which followed in the Heading validation as well). The IMU was kept flat in the x, y plane and rotated about the z axis 3 times in 30 seconds. 500 data points were collected during this process. Under these conditions we expect the linear acceleration in the x and y axis to be negligible (around 0). The acceleration in the z axis will be around $9.8m/s^2$ due to force of gravity, but will be removed from further consideration because our vehicle will be traveling in the x,y plane on the ground.

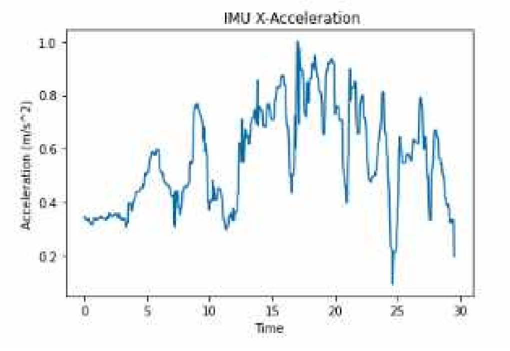


Figure 14:

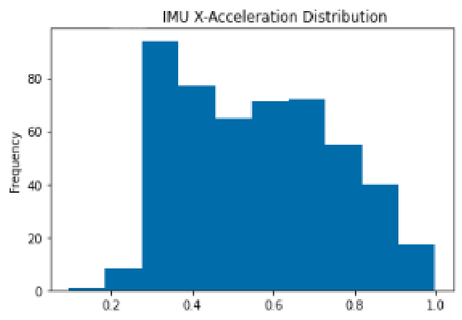


Figure 15:

Examining the the x-acceleration values from the test over time reveals that they are fairly noisy with many fluctuations, but this is expected from the accelerometer. Additionally, they lie close to 0 with a max of 0.998 , min of 0.095 , and variance of 0.034. The distribution plot also confirm that the values hover near 0 and do not vary greatly. These findings satisfies our expectations and indicates that the calibration process was successful in the linear x-axis acceleration.

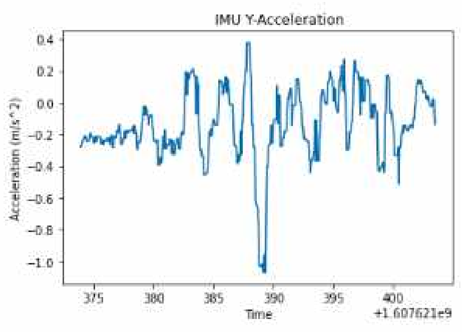


Figure 16:

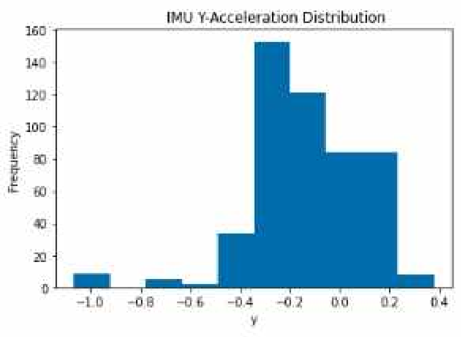


Figure 17:

The y-acceleration values exhibit similar properties to the x-acceleration values. The readings over time remain near 0 for the entire duration of the trial. The values are noisier than the x-acceleration, but not to an alarming level. The distribution of the y-acceleration demonstrates values that are mainly centered around 0 with a few negative outliers. Overall, both the X and Y acceleration values indicate positive results that the calibration for the linear acceleration was successful and provides reliable readings.

3.3 Initial Heading within ROS

Deriving the heading of the robot is critical in our robot's navigation system, especially in environments when GPS localization may be error prone and unreliable. The IMU sensor is cornerstone for obtaining the heading of our autonomous vehicle as its measurements allow for greatest accuracy in finding orientation. The following equations define the derivation for the Roll, Pitch, and Yaw. Yaw is the most important in providing us with a heading.

$$pitch = 180 * atan2(accelX, sqrt(accelY * accelY + accelZ * accelZ))/PI \tag{4}$$

$$roll = 180 * atan2(accelY, sqrt(accelX * accelX + accelZ * accelZ))/PI; \tag{5}$$

$$mag_x = magReadX*cos(pitch)+magReadY*sin(roll)*sin(pitch)+magReadZ*cos(roll)*sin(pitch) \tag{6}$$

$$mag_y = magReadY * cos(roll) - magReadZ * sin(roll) \tag{7}$$

$$yaw = 180 * atan2(-mag_y, mag_x)/M_PI; \tag{8}$$

The benefit of the using the Artemis ROS package is the inclusion of the Attitude Heading Reference System that is at the core of library. This allows us to derive orientation for the robot using the acceleration, gyroscope, and magnetometer readings in unison. This is unique compared to other firmwares and orientation systems that lack in ability to correct heading estimations using the compass (magnetometer) readings. The orientation is given in quaternions, which are four dimensional [x,w,y,z] coordinates, defining a vector and rotational transformation. The heading we aim to use for the robot is the Yaw, or rotation about the z-axis (measured in degrees). Thus it is important to convert the quaternion values to Yaw. Fortunately, ROS provides a method for converting within its transformations library (tf.euler_from_quaternion). We have also written a custom publisher for the yaw value to make this data easily accessible to potential consumers and publishes on the /yaw topic. The yaw data analyzed was collected from our custom publisher. Continuing under the previously noted conditions to test reliability of measurements (x, y plane constant and 3 rotation about z-axis in 30 second time-frame), we can determine whether our heading reading is reliable and accurate.

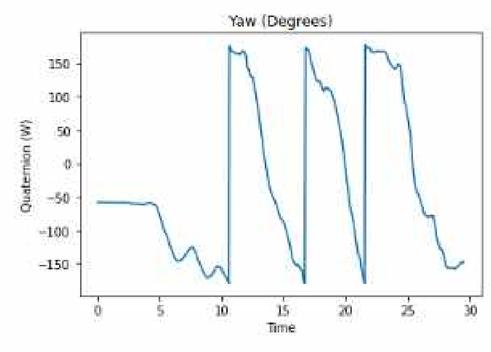


Figure 18: accel+gyro

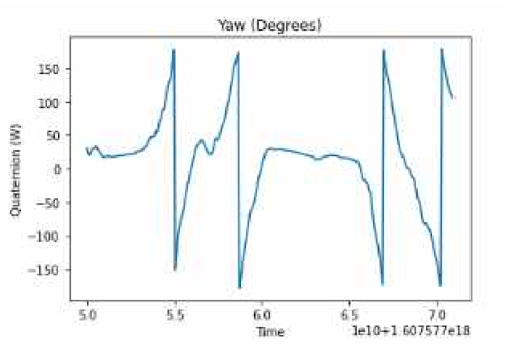
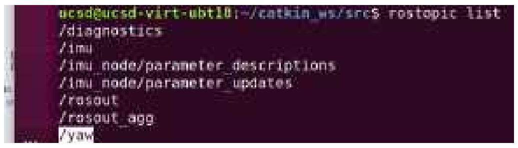


Figure 19: accel+gyro+mag

Examining the Yaw readings over the full 30 second duration shows promising results. There are three peaks at 180 degrees and three corresponding troughs at -180 degrees. The sharp jumps are due to the degree system used to measure Yaw, which goes from 0 to 180 degrees, and then from -180 degrees back to 0 (where North is indicated at 90 degrees). This is exactly what we hoped to expect as these values match exactly to the three full rotations we completed about the z-axis. The Yaw heading captured this activity accurately and robustly!

We then added an additional test under the same conditions, but instead of 3 rotations about the z-axis, we did 2 rotations clockwise, paused, and then 2 rotations counter-clockwise. As the above plot shows, this activity is accurately captured as well as there are two peaks and falls (at 180 and -180 degrees), a plateau corresponding to the pause, and then two peaks and falls rising and falling in the opposite direction (at -180 and 180 degrees). This further proves the ability of our heading estimation!

In summary to obtain the heading (Yaw), users can first extract quaternions from geometry_msgs/Quaternion from sensor_msgs/Imu published by the /imu topic, and then convert to Yaw. Or they can directly acquire the Yaw heading by subscribing to our custom written /yaw topic. Furthermore, our heading estimation is reliable, accurate, and ready to be used in our robot localization and navigation system.



```
ucsd@ucsd-virt-ubt18:~/catkin_ws/src$ rostopic list
/diagnostics
/imu
/imu_node/parameter_descriptions
/imu_node/parameter_updates
/rosout
/rosout_agg
/yaw
```

Figure 20: custom /yaw topic

4 ROS JNX Integration Workaround

To ease the process of setting up the IMU to our Jetson NX, we had hoped to have all modules and installation completely functional on the Jetson itself. The AHRS software that is integral to the OLA Artemis working must be flashed through the Arduino IDE. This is because it is written in an Arduino file format. However downloading the Arduino IDE, necessary board definitions, and build tools on the Jetson's ARM64-architecture proved to be difficult. We first tried to circumvent the usual approach in downloading the Arduino IDE itself and utilized the specific download from Jetson Hacks. While we were able to establish a connection to the IMU through this Arduino IDE installation, we were not able to successfully install the necessary build tools and Sparkfun Redboard Artemis board definition. After searching on many Arduino and Jetson forums, we were eventually able to install the required board definition by the help at the following link: https://github.com/sparkfun/ArduinoBoards/pull/62. By including the specified toolchain in the build json file, we were able to successfully compile the sketch. However, we were not able to successfully upload the software to the IMU and are still looking into how to fix this error. Our current workaround is to first flash the AHRS software to the OLA Artemis IMU using an external machine (tested running Ubuntu 20.04). This allowed us to successfully upload the appropriate software and establish ROS connection to the IMU. The ultimate result is that the software must be flashed from an external machine and therefore, the calibration must also be done while connected to an external machine. As long as the IMU is as close as possible to its mounted position or running environment, this process has not yielded any major errors in our verification analysis of heading. The guide developed in installing and setting up the OLA Artemis IMU with ROS covers this process further.

5 Heading Real Environment Analysis

Integrating the IMU to the 1/10 car allowed us to verify and adjust our calibration and helped us discover new insights regarding Yaw Heading error in a real environment. When setting up our experiment, we utilized re-calibrated settings in the new environment based on our previous best accelerometer, gyroscope, and magnetometer Yaw calibration parameters. We then tested these under the following situations and conditions we controlled during our Odometry positioning tests on the robot.

5.1 Straight Line Heading Test

We first tested the reliability of the heading in the following case, by driving the robot strictly straight 2 meters at 90 degrees (West) for 10 seconds. The IMU was mounted flat, traveling only in the x,y plane. By testing three different parameter settings for calibration, we were able to determine the most accurate heading performance. The three calibrations consisted of: accelerometer + gyroscope calibration, accelerometer + gyroscope + mangetometer calibration, and accelerometer + gyroscope + extended magnetometer (for better handling iron offsets) calibration. The specific calibration parameters per each setting are available in the attached documents. After converting the orientation quaternions to Yaw (degrees) as before within ROS, we were able to analyze the heading in a real time setting under the three described conditions.

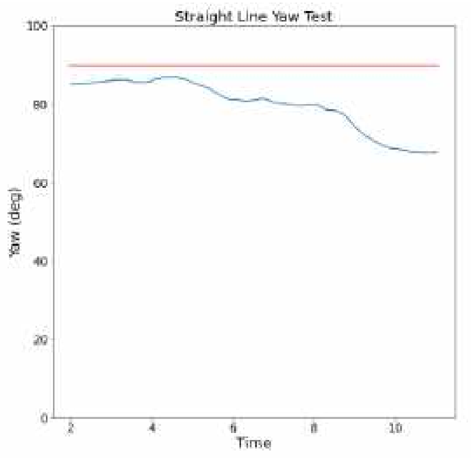


Figure 21: accel+gyro

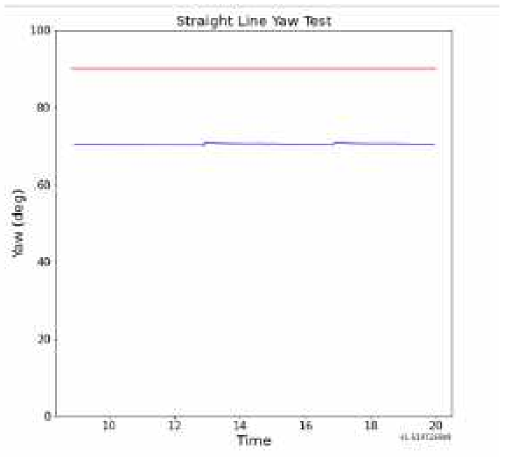


Figure 22: accel+gyro+mag

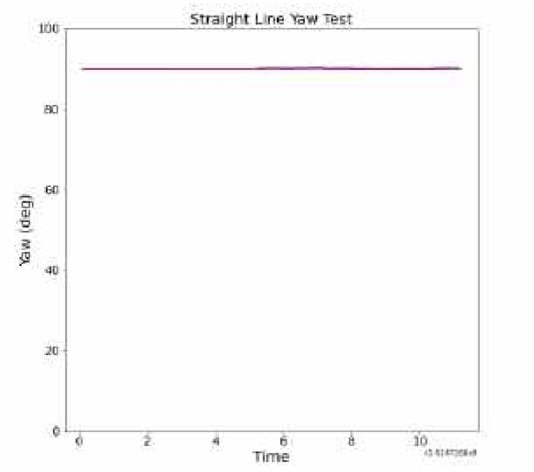


Figure 23: accel+gyro+extended mag

Analyzing the Yaw readings against each other from all three test calibration conditions over the the full 10 second interval of strictly driving 90 degrees (West) revealed interesting findings. For comparison, readings (blue) were placed against a theoretical "perfect" heading of 90 degrees (red) throughout the 10 interval.

The accelerometer and gyroscope calibration condition produced the highest average error, 90.98, in the straight line yaw test overall. It also had the most striking behavior as the result of the drift apparent in the readings. This can be seen as the values slowly decrease from 85 degrees to 70 degrees in only 10 seconds, and in the high variance. Since the car traveled on a strict straight path, this can be attributed magnetic forces present in the real time integration that drew influences from the nearby motors, processors, and hardware.

The accelerometer, gyroscope, and magnetometer calibration condition performed the second best in this test with an average error 19.29. Here, the drift apparent in the previous readings was minimized, which can be seen as the readings are fairly stable throughout the 10 seconds with a mean of 70.34 degrees and a low variance. The error was primarily due to a consistent offset which was most likely produced by a hard iron offset present in the environment that was not accounted for by the standard magnetometer calibration.

Finally, the accelerometer, gyroscope, and extended magnetometer calibration condition performed the best with minimal average error of 0.14. This was expected as this wholly utilizes the calibration capabilities available to us. The average of the readings was 90.56 degrees and the behavior stable throughout the 10 second interval. This gives us confidence that we are able to account for the hard iron offsets and stronger magnetic influences produced by the motors, processor, and other hardware in our real time environment using the extended magnetometer calibration technique.

5.2 Half Arc Heading Test

Next, we tested the reliability of the heading in the following case, by driving the robot in a 180 degree arc at a constant speed and turning angle for 10 seconds. The IMU was mounted flat in the x,y plan akin to the "second" mounting procedure in the Mounting section. The same three sets of calibration conditions were utilized from the previous straight line test. After converting the quaternion values through ROS, we were able to analyze the heading.

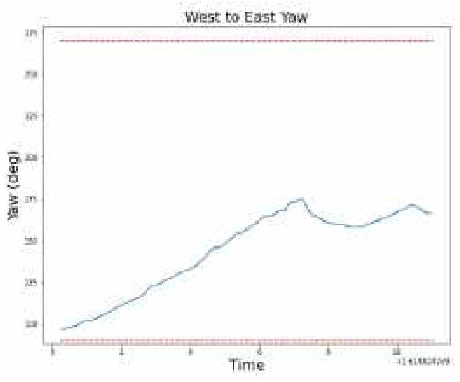


Figure 24: accel+gyro

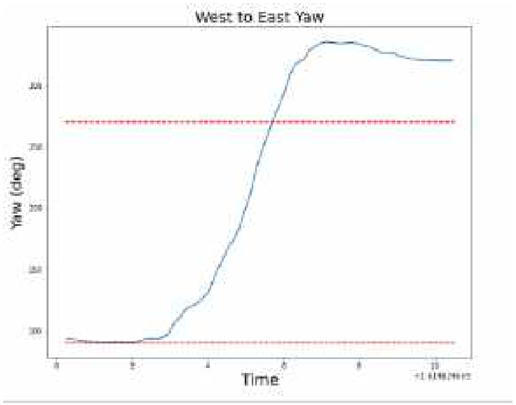


Figure 25: accel+gyro+mag

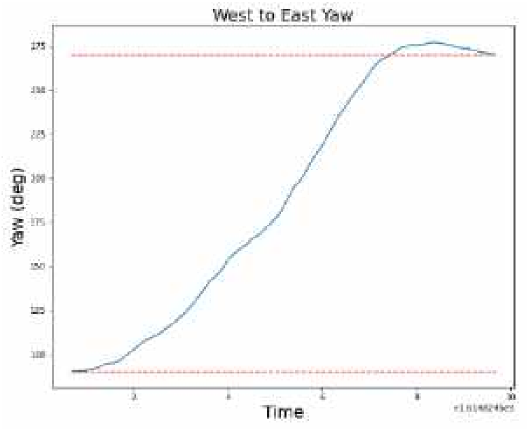


Figure 26: accel+gyro+extended mag

Examining the Yaw readings against all three calibration values in the Half Arc Test revealed to us insights into our calibration performance during turning. For comparison the robot was driven 180 degree half arc turn (West to East, 90 to 270 degrees) in 10 seconds at a constant speed and turning angle.

Similar to the straight line test, the accel+gyro calibration performed the poorest and allowed for largest variance in the readings, with an error 94.89 when compared to a theoretical increase from 90 to 270 degrees in 10 seconds. There is also random noise and drift that begins to occur towards the end of the test and the heading never crosses 170 degrees, even though the robot ends its test at 270 degrees. This demonstrates that the magnetometer calibration is crucial.

The accel+gyro+mag calibration condition performed the next best in the Half Arc Test with an error of 42.5 when compared to a theoretical increase from 90 to 270 degrees. There was less drift and noise in the readings which can be accounted by the magnetometer calibration. However, the readings indicate that the heading readings overshot the actual half arc by about 180 degrees and settled around 350 degrees. This is unacceptable level of error to utilize this calibration condition in our integration.

Lastly, the accel+gyro+extended magnetometer calibration performed the best with an error 15.07 when compared to a theoretical increase from 90 to 270 degrees in 10 seconds. The heading was also tracked in a smooth fashion with less noise and drift than the previous conditions. This was what we had hoped for and demonstrated how the magnetic presence has a strong effect when not calibrated for using the full capabilities of the calibration procedure. The extended magnetometer calibration technique allows us to confidently handle the noise and magnetic distortions with a reliable mounting setup. It is cornerstone to note that the noise is heavily dependent on the mounting position. We used the second mounting setup described in the following section for these tests, yet the noise can further be mitigated with an advanced mounting setup.

6 IMU Mounting and Analysis

The mount and mounting procedure to be used for the IMU is crucial in achieving the highest accuracy readings from the IMU. Insecure or unstable mounting positions can cause uneven external pressures to be applied to the IMU board. This can cause large errors in the IMU readings as it is extremely sensitive to even slight fluctuations in its placed environment. The propagation of these errors can greatly deviate or ruin the derived heading and position estimates. The initial mount we attempted to use was a temporary fix but revealed to us the severity of the errors that can directly be produced by insufficient mounting procedure. We naively used a ziptie and double sided electrical tape as our first mount to simply test movement with the IMU onboard the robot.

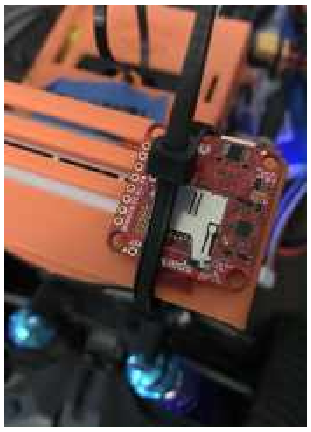


Figure 27: Simple Mount

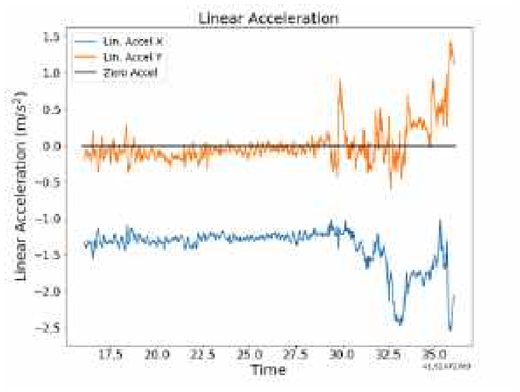


Figure 28: Simple Mount Lin. Accel

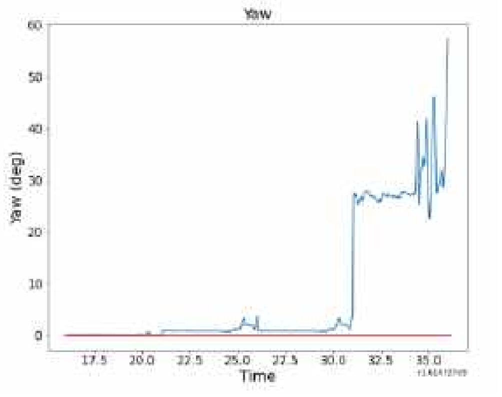


Figure 29: Simple Mount Yaw

This caused the following drift and error in the linear acceleration and the Yaw heading readings. This can be seen in the average -1.24 m/s^2 bias in the linear x acceleration and the small bias present in the y acceleration. The large variation at the ending of was due to IMU shifting during the run due to this mount being not secure enough. The Yaw heading readings slowly began to increase and drift over time in this mounting position as is apparent in the first up until 30 seconds. This was most likely due to the constant applied pressure and tilt being applied to the Artemis IMU in this mounting position. The shifting during the drive caused a massive change in the Yaw heading and demonstrated how strongly the secureness of the mount can change the readings. Overall, this mounting position was not secure, allowing the IMU to move and sway as the robot traveled its course, corrupting the measurements.

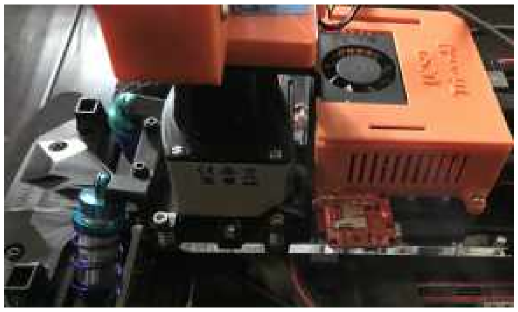


Figure 30: Second Mount

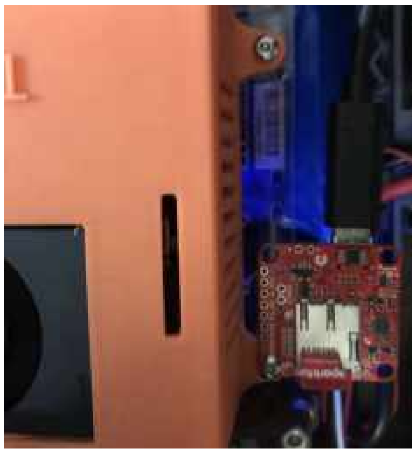


Figure 31: Second Mount

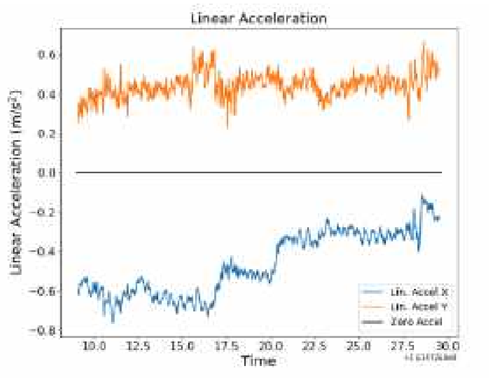


Figure 32: Second Mount Lin. Accel

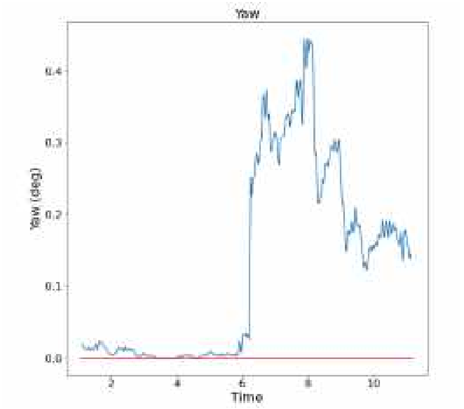


Figure 33: Second Mount Yaw

Next, we more securely fastened the IMU to the Jetson NX mount through its upper right mounting hole. This placement was not the ideal case, however it.allowed us to get more stable data readings that we used during our calibration process. In this position, the IMU was securely kept in the x, y plane. This removed the previous large bias we witnessed in the linear y acceleration. There is still small bias however, but it is less than 0.8 in both acceleration axes, which is normal. The Yaw values are also extremely close to 0 (North) throughout the run, with small acceptable fluctuations 0.5 degrees. It is still important to note, the mount still allowed some sway around the z axis, causing some fluctuations and noise in Yaw heading readings in a few runs over time. Furthermore, the magnetic influences by being directly next to the Jetson could amplify the accumulation of error.

The ideal mount would keep the IMU secure in the x,y plane, not apply uneven pressure, not allow for any sway or movement, and be furthest from magnetic influences such as the Jetson, motors, and VESC. With the help of the TAs, a mount was designed to meet these needs for the 1/5 robot. This mount will ensure the IMU is evenly secured and will be placed away from other magnetized hardware. We believe this mount will allow us to provide the most accurate and reliable IMU data, and will be integrating/testing soon. Testing different mounting strategies allowed us to gain insight in how critical the mount was in influencing the IMU readings and key factors to mitigate these corruptions.

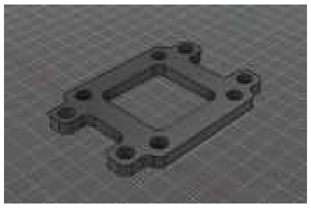


Figure 34: 1/5 Car Mount

7 Kalman Filter

To fuse our Odometry and IMU sensor reading to provide a better position estimate, a common and accepted approach from research and literature is to apply Kalman Filtering [3]. Kalman Filters are used to obtain the best estimate of states (position in our case) through the combination of measurements from various sensors in order to mitigate noise [7]. The robot-localization package in ROS provides an implementation of an Extended Kalman Filter that has popular support and can be integrated into our navigation system.

For implementation, the Kalman Filter requires a covariance matrix based on the known or estimated variances in the sensors to be used. Furthermore, the Kalman Filter requires the setting of a configuration matrix per sensor that determines which readings to input. These readings are given in the following order of: X, Y, Z, Roll, Pitch, Yaw, X vel, Y vel , Z vel, ang vel. Roll, ang vel. Pitch, ang vel. Yaw, accel $\ddot{X}$, accel $\ddot{Y}$, and accel $\ddot{Z}$. We utilized the default settings that were recommended by the community, setting the Odometry to input default covariance values 0.025 and readings from the pose.x and pose.y estimates. The IMU was also set to input default covariance values 0.050 and readings from the linear acceleration x, linear acceleration y, yaw, and yaw velocity.

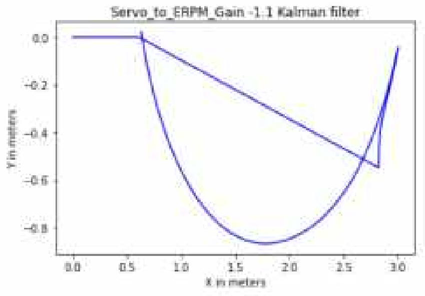


Figure 35: 1/5 Car Mount

When running our half arc test of Odometry, we found that the Extended Kalman Filter suprisingly performed slightly worse than our pure Odometry position estimate (ignoring the error at the end due to moving the car beyond the set time interval). This was seen as an offset in the x and y position due to initial drift. We believe this could be due to noise apparent in the IMU linear acceleration which retains a slight bias no matter how meticulous the calibration process due to the nature of the sensor's extreme sensitivity. Additionally, the lack of a global ground truth position that would be provided by the GPS. When the GPS team fuses the sensors, their implementation would allow them to correct for the noise with greater accuracy due to the presence of the GPS readings. The current solution is to simply disregard the linear accleration in the EKF as they result in high accumulation of drift error and solely utilizing the IMU for the Yaw and Yaw velocity inputs into the EKF. Further work is to explore options in compensating for this noise and potentially the use of multiple IMUs to mitigate these errors stemming from slight bias in the readings.

8 IMU Signal Noise Reduction Strategies

In order to further compensate and mitigate the noise in the IMU readings to provide more accurate data, we utilized a few signal processing techniques. Through our research, we found that popular approaches for handling noise in a signal were by employing frequency filters to the data. One such filter is the Low Pass Filter, which reduces high frequency noise in a signal by attenuating frequencies above a certain cutoff frequency. We implemented a Low Pass Filter in Python at a cutoff frequency of 75 Hz.

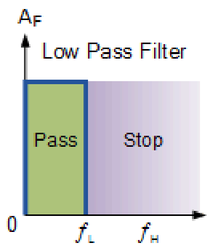


Figure 36: Low-Pass Filter

Another such filter is the Median Filter. This filter is applied to smoothen a signal by converting data points to the median of its neighbors across a "sliding window" of determined size. We then implemented a Median Filter of kernel window size of 5 data points to convolve our signal with.

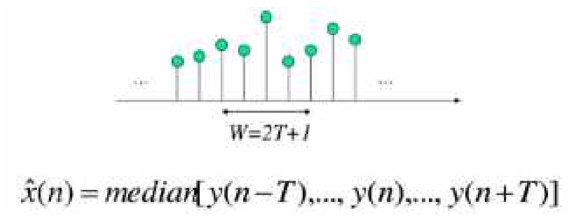


Figure 37: Median Filter

The third approach to process the noise was to use Haar Wavelets. The Haar Wavelet system is a sequence of "square" shaped functions that in union form an orthogonal basis. By decomposing our signal with functions of higher levels (higher frequencies) we can then remove the high frequency noise from the original signal and reconstruct our smoother signal. We also implemented this in Python and set removed the final 3 levels of the orthogonal basis as a default to attempt to reduce noise.

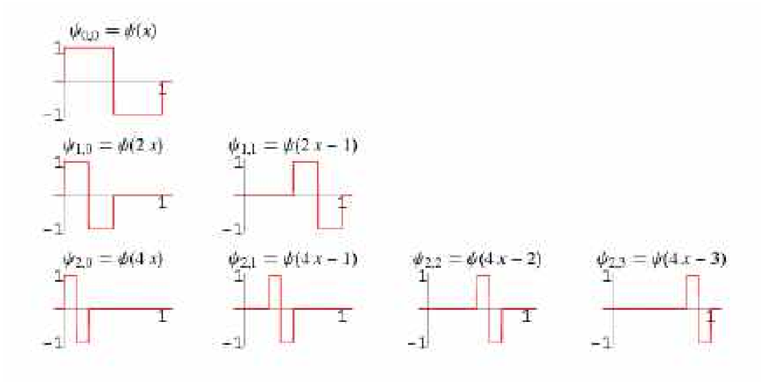

$\psi_{0,0} = \psi(x)$
$\psi_{1,0} = \psi(2x)$
$\psi_{1,1} = \psi(2x-1)$
$\psi_{2,0} = \psi(4x)$
$\psi_{2,1} = \psi(4x-1)$
$\psi_{2,2} = \psi(4x-2)$
$\psi_{2,3} = \psi(4x-3)$

Figure 38: Haar Wavelets

The experiment run on the linear acceleration signal of the IMU allowed us to further analyze the findings. Although our experiment mainly used default parameters recommended by research and various community sources, we could see that the Low Pass Filter was able to smoothen the noise to the highest degree. The Haar Wavelet decomposition produced discrete time step approximations of the signal where the Median filter also produced a smoother signal overall. The issue however is that we were still unable to account for the slight bias in the acceleration even with these approaches. Future work will be to further explore Allan Variance testing which can help us characterize the noise we are receiving and the level of bias instability. Additional exploration will also consist of methods to directly address this slight bias in with greater precision.

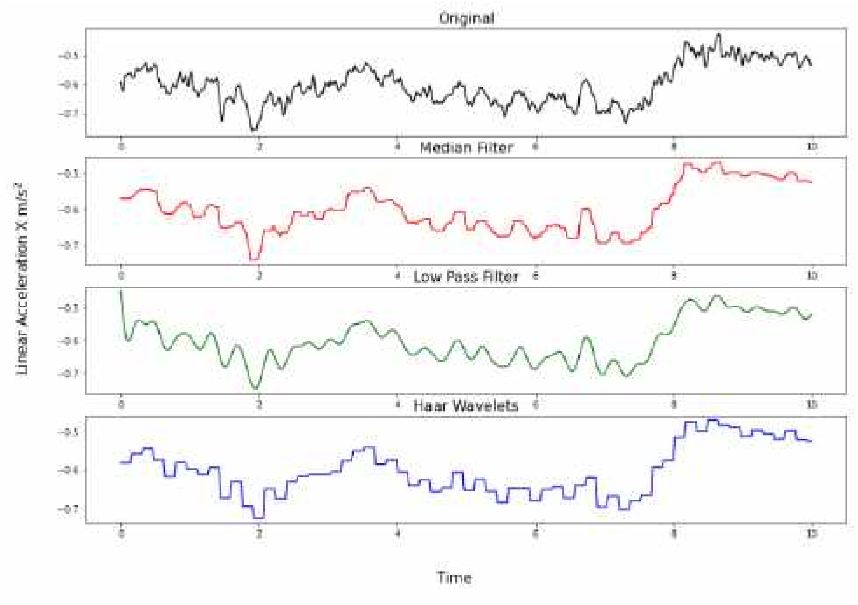


Figure 39: Signal Noise Reduction Strategies

9 OAK-1 Camera

To further expose ourselves and improve our skills in robotics, data science, and gain exposure in computer vision, we were lucky enough to begin experimentation with the new OpenCV AI Kit OAK-1 Smart Camera. This is a brand new single-camera hardware module that boasts camera capture of 4K video data at 60 fps or H.265 encoded at 30fps [8]. The camera is only the size of a Raspberry Pi, but allows for the potential to run advanced neural network models for object tracking, detection, semantic segmentation, and corner detection on the camera itself. Thus, we can utilize this camera to offload perception computation processes from the Jetson NX and run them directly on the OAK-1. The Goal is to mount the OAK-1 camera on the rear of our robot, similar to the efforts of Tesla which utilize multiple cameras on their autonomus vehicles []. This will allow our robot to gain rear perception sensing capability to complement the Intel Realsense mounted in the front. Also, the OAK-1 only has a 70 degree field of view and cannot provide stereo depth due to it being a monocular camera, so it would be better suited as a rear mounted camera. During the autonomous race we hope to enter, we want to give our robot the ability to detect fast approaching opposition racer robots on the racetrack, cones, and lanes from the rear to enhance our decision and navigation logic in order to win the race. This way we can potentially gain the ability to ”cut” off opponent racers and better improve our own ideal ”racing” line to achieve faster race times and top rankings.

To interface with the OAK-1 we began by establishing connection to the Jetson. This proved to be more difficult than on the personal macbook pro that was previously used. A USB-C cable was required for this (which had to be switched), and a special driver had to be downloaded for the Ubuntu 18.04 system. Furthermore, a /etc/udev/rules.d/80-movidius.rules file needed to add json configuration parameters found through personal discussions on the Luxonis Support Discord Channel. We then were able to establish a secure connection to the JNX. We then moved to integrate with ROS so we could incorporate the OAK to our robot system. The ROS package provided by the Luxonis Support team was is a recently published software that was warned to be error prone. Thus, through another process of trial, error, debugging, and discussion we were able to successfully launch the example MobileNet object detection neural network using ROS. However, when attempting this process on a second JNX, we ran into opencv-python errors that are still in the debugging phase. We then began collecting preliminary data to identify an incoming opposition race car from the rear annotating these images appropriately.

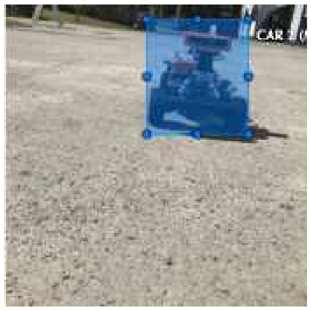


Figure 40:

Figure 41:

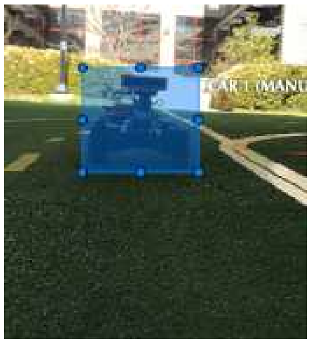


Figure 42: OAK training data

Our future work will be to collect more training images of opposition robots from a rear perspective and then train/finetune a Mobilenet Object Detection SSD model (which already includes 70 classes) to detect these robots. To train this model, we plan to use a Stochastic Gradient Descent training strategy with an Adam Optimizer for 200 epochs. Based on previous discussions and research, this was a good benchmark to begin our training. Additionally, we will employ Early Stopping strategy to stop model training at the optimal weights. If possible, we would also utilize the UCSD GPU Cluster to train our model to greatly improve training time and performance. We hope to also extend our work to detecting the cones, lines, and road signs to best improve our perception ability.

10 Conclusion

As the IMU and Odometry subteam, we have achieved most of our target goals we defined at the beginning of this report and are ready to integrate our work with the final robot. We gained a strong understanding of IMU and Odometry sensing and its ability to provide valuable data for accurate robot localization. We derived position estimates using Odometry, along with an analysis on future work to make this estimation more accurate and robust. By devising tests for straight and turning paths we were able to analyze our findings and converge to an optimal parameter for the steering gain ERPM and steering angle. We then successfully calibrated the IMU sensor and ran thorough tests to ensure reliable measurements. Using a similar set of tests, we measured the accuracy of the Yaw heading on a straight and turning path. This gave us insight into the strength of the magnetic distortions which is critical to compenstate for onboard the robot. We also investigated different IMU mounting approaches which highlighted the immense effect the mount can have on the accuracy of the extremely sensitive IMU readings, allowing us to better our mount for the 1/5 car. The Primary Heading estimate was derived from not only the accelerometer and gyroscope, but also the magnetometer readings from the Artemis IMU. This heading can now be easily consumed by a custom /yaw topic or through default ROS messaging. We succeeded in integrating the Artemis IMU and Odometry with ROS so that our data is easily digestable by other subteams, such as the obstacle avoidance team, who wish to consume this data in order to enhance their navigation models. Furthermore, the IMU and Odometry data is ready for fusion with the GPS subteam within a Kalman Filter for future advancement of robot localization methods. Finally, our work will ensure that we can hone the full potential of IMU and Odometry sensors to improve the autonomous navigation of our final robot.

11 Future Direction

Next, we hope to work on integrating the IMU and Odometry work into the 1/5 scale car and testing the new IMU mount that was designed. Calibrating, tuning, and analyzing the data on the 1/5 car will give us insight into the unique environment our sensors will reside in and allow us to best prepare the car for success during the autonmous vehicle race competition at the Thunderhill track in Northern California. This will also provide us with invaluable data to analyze in order to further improve our IMU and Odometry processing and integration into the navigation stack.

There are several future direction which we began to explore in order to further the accuracy of our position estimate, heading, and data readings from IMU and Odometry. One is to delve into addressing the inescapable slight bias in IMU data readings post calibration due to the extreme sensitivity of the sensor. Here, we aim to further research the application of the Kalman Filter and potentially the Unscented Kalman Filter to enhance our estimate. We also hope to conduct future work on researching signal processing techniques and tests to reduce the IMU noise. Allan Variance testing provides a potential method to characterize the noise and bias instability, so wecan more precisely process this noise. We hope to conduct this analysis and apply new strategies for noise compensation. Another approach would be to utilize multiple IMUs aboard the robot to better compensate for the noise in readings. This would require a higher level of calibration and synchronization, yet can be a promising approach and is one that is often used in industry.

We would also like to continue our most recent work on the OAK Smart Camera. To successfully complete the training of a Mobilenet Object Detection model for rear sensing capability, we will have to finish collecting and annotating enough training image data for the model. Based on previous exploration, we plan to use a SGD training strategy with an Adam Optimizer for 200 epochs. Additionally, we will employ Early Stopping strategy to stop model training at the optimal weights. If possible, we would also utilize the UCSD GPU Cluster to train our model to greatly improve training time and performance. Then, we would expand to detecting the cones, lines, and road signs from the rear to best improve our perception ability.

12 Acknowledgements

Overall, this project has allowed us to not only experiment, learn, and apply data science skills to future oriented hardware systems, but also grow our skill sets vastly. We would like to thank our mentor Jack Silberman, the TAs, and administration for giving us the opportunity to continue safely learning about autonomous vehicles and navigation in a hands-on manner, even during the COVID-19 pandemic. Thank you!

13 Appendix

Project Proposal
Odometry Calibration
IMU Calibration

References

[1] Mordechai Ben-Ari and Francesco Mondada. "Robotic Motion and Odometry". In: *Elements of Robotics*. Cham: Springer International Publishing, 2018, pp. 63–93. ISBN: 978-3-319-62533-1. DOI: 10.1007/978-3-319-62533-1_5. URL: https://doi.org/10.1007/978-3-319-62533-1_5.

[2] M.A. Brodie, A. Walmsley, and W. Page. "The static accuracy and calibration of inertial measurement units for 3D orientation". In: *Computer Methods in Biomechanics and Biomedical Engineering* 11.6 (2008). PMID: 18688763, pp. 641–648. DOI: 10.1080/10255840802326736. eprint: https://doi.org/10.1080/10255840802326736. URL: https://doi.org/10.1080/10255840802326736.

[3] M. Brossard, A. Barrau, and S. Bonnabel. "AI-IMU Dead-Reckoning". In: *IEEE Transactions on Intelligent Vehicles* 5.4 (2020), pp. 585–595. DOI: 10.1109/TIV.2020.2980758.

[4] Martin Brossard and Silvere Bonnabel. "Learning Wheel Odometry and IMU Errors for Localization". In: *2019 International Conference on Robotics and Automation (ICRA)* (2019). DOI: 10.1109/icra.2019.8794237.

[5] f1tenth. *f1tenth/f1tenth$_s$ystem*. URL: https://github.com/f1tenth/f1tenth_system.

[6] Máté Fazekas, Péter Gáspár, and Balázs Németh. "Calibration and Improvement of an Odometry Model with Dynamic Wheel and Lateral Dynamics Integration". In: *Sensors* 21.2 (2021), p. 337. DOI: 10.3390/s21020337.

[7] Xiaoji Niu, Yibin Wu, and Jian Kuang. *Wheel-INS: A Wheel-mounted MEMS IMU-based Dead Reckoning System*. 2020. arXiv: 1912.07805 [cs.RO].

[8] *OpenCV AI Kit: OAK-D/1 Camera Buy and Customize*. URL: https://www.arducam.com/oak-opencv-ai-kit-camera/.

[9] Sparkfun. *Sparkfun Razor 9DoF IMU*. URL: https://www.sparkfun.com/products/16832. (accessed: 01.09.2021).

[10] Siddhartha S. Srinivasa et al. "MuSHR: A Low-Cost, Open-Source Robotic Racecar for Education and Research". In: *CoRR* abs/1908.08031 (2019).