Autonomous **D**riving for **A**dverse **P**erceived **T**errain

Final Report

Team Shasa Antao
Bryson Jones
Shaun Liu
Evan Schindewolf
Wesley Wang

Sponsors Dr. Dimitrios Apostolopoulos
Dr. John Dolan

Advisor Dr. David Held

Team B

Master of Science in Robotic Systems Development
Robotics Institute
School of Computer Science
Carnegie Mellon University
December 16th, 2020

Carnegie Mellon University

THE ROBOTICS INSTITUTE

Abstract

Autonomous driving is currently a large focus of research and development for many organizations in industry. However, these companies are mainly focused on fair weather and standard conditions, ultimately neglecting to address how these technologies will perform when encountering adverse road conditions. The goal of this project is to develop a system to assist and allow autonomous vehicles to safely drive in such scenarios. This type of system will be necessary before autonomous vehicles are able to become widely accepted in society. The system described in this report will detect wet and water-covered asphalt regions in front of a vehicle and adjust the vehicle speed and steering to navigate around such hazards. Additionally, the system will recognize when these hazards cannot be avoided and slow the car down accordingly to safely traverse the area. The work to accomplish these tasks is described across developing, integrating, and testing individual systems and the system as a whole against performance metrics. After discussing the current system, an outline of future work is presented to improve the system performance and increase the range of scenarios in which the system will be applicable.

Contents

1 Project Description

The autonomous vehicle industry, specifically with respect to passenger vehicle and trucking and freight applications, has quickly become a large focus in technology development. However, in general, most autonomy systems lack the ability to transition from operating in ideal road or terrain conditions to wet or ice road scenarios. Autonomous vehicles that enter these conditions could lose control and endanger passengers, nearby pedestrians, and other vehicles.

Research indicates that thousands of people are killed and hundreds of thousands are injured every year due to weather related crashes [1], and up to $3.5 billion is lost annually by the trucking and freight industry from delays due to poor environmental conditions [2]. Even in the event that a high level of autonomy is achieved for autonomous vehicles in dry and sunny conditions, it is unlikely to translate to any reduction in these numbers, since people will always need to travel and transport goods, no matter the state of the road.

The ADAPT (Autonomous Driving for Adverse Perceived Terrain) system is designed to combat this issue. The system serves to actively detect changes in the state of the road and adjust the vehicle controls to account for them, making transitions into wet or otherwise unfavorable road conditions much safer. Current systems for dealing with these scenarios are almost exclusively reactive, meaning that they only modify the vehicle's actions after an incident has occurred, whereas the ADAPT system utilizes forward looking information to maximize the vehicle's ability to safely transport passengers and goods.

2 Use Case

Alice is on her way home from a ski-resort after enjoying a vacation. Since she had too much to drink, Alice plans on taking a self-driving taxi down the mountain towards home. She is apprehensive of her choice as she recently read an article in the newspaper about autonomous vehicles being very unsafe in adverse environmental conditions. However, given the circumstances she feels she has no other choice but to take a self-driving taxi, such as the one depicted in Figure 1a, as most taxi drivers are unavailable during the holidays.

As she waits for the self-driving taxi to arrive, it starts to rain heavily, and Alice starts to feel anxious about her choice of transport. Before she can make a decision to cancel the taxi, it pulls up in front of the resort and she gets in. The car picks up speed and it reaches the winding part of the journey down the mountain. Alice starts to panic as the road is slick due to the rain. Fortunately, the car is equipped with the ADAPT system that is designed specifically for safe control of vehicles through adverse terrain.

As the car rounds a corner, it comes upon a wet spot. It identifies that this is an area of limited vehicle grip, plans a new path, and effortlessly avoids this spot, as shown in Figure 1b, in a manner that keeps Alice comfortable. She is surprised about what happened and starts to trust the ability of the vehicle. The car then rounds a second corner and identifies an unavoidable wet spot. In order

(a) Autonomous Taxi

(b) Puddle Avoidance

(c) Speed Reduction

to overcome this challenge, the car autonomously adjusts to a new maximum speed at which it can safely take the corner. Soon after, the car comes upon a third corner with another unavoidable wet spot. This time, as the vehicle encounters slippage, it slows down and counter steers to recover to the original path as shown in Figure 1c. Alice is impressed with the way the car handled each of these situations, which would have been tough even for experienced human drivers.

The self-driving taxi drops Alice off at her house that afternoon. She appreciates that the service got her home safely and that the ride was comfortable, even through inclement weather. Now, Alice has a high regard for autonomous vehicles and their ability to keep their riders safe. She will take the self-driving taxi service again, especially if it is raining outside.

3 System Requirements

The system-level requirements include performance and non-functional requirements. These requirements ensure that the system is able to meet the needs of the project. The mandatory requirements are critical for the system to function and were prioritized. The desirable requirements were less prioritized.

3.1 Mandatory Performance Requirements

The system will:

M.P.1. Achieve road segmentation IoU greater than 65% for at least 65% of frames.

M.P.2. Achieve puddle segmentation IoU greater than 65% for at least 65% of frames.

M.P.3. Have planning horizon always greater than 4m.

M.P.4. Have real-time motion controls (at least 55 Hz).

M.P.5. Stay within 0.75m of waypoints at all times.

M.P.6. The vehicle avoids or traverses 60% of puddles with speeds less than 2 m/s.

There were changes to the performance requirements of the system. The planning horizon was reduced to 4m, as it was found during testing that given the platform and test area, the operating speed needed to be lower, which led a smaller necessary planning horizon. Lastly, an extra requirement was outlined for the highest level of functionality for the vehicle, which is M.P.6.

3.2 Mandatory Non-functional Requirements

The system shall:

M.N.1. Be able to reach vehicle speeds of up to 15 km/hr.

M.N.2. Use self-contained energy sources.

M.N.3. Have IP64 or better Ingress Protection rating.

3.3 Desirable Performance Requirements

The system will:

D.P.1. Achieve road segmentation IoU greater than 85% for at least 85% of frames.

D.P.2. Achieve puddle segmentation IoU greater than 85% for at least 85% of frames.

3.4 Desirable Non-functional Requirements

The system shall:

D.N.1. Be able to function under adverse weather conditions.

D.N.2. Cost less than the average cost of a tractor-trailer incident.

D.N.3. Generate terrain map for user.

D.N.4. Provide warnings of incoming adverse terrain to user.

D.N.5. Use reinforcement learning to improve vehicle dynamics.

4 Functional Architecture

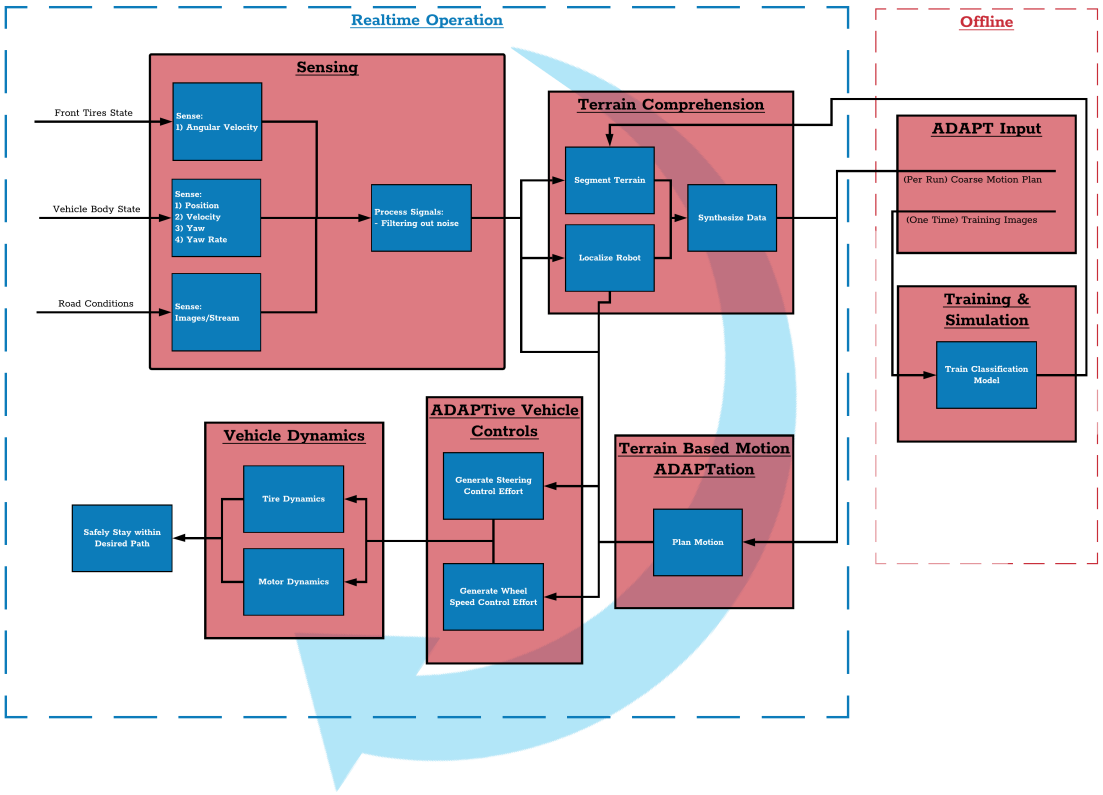

Figure 2: Functional Architecture

The Functional Architecture shown in Figure 2 details the flow of functions through the various subsystems of the ADAPT system. It also shows the relationships between the subsystems and the interdependencies therein.

The first block is Sensing which is required to perceive the oncoming road conditions and determine the state of the car. This enables the Terrain Comprehension subsystem to segment the road into traversable and non-traversable areas, and the resulting segmentation is used to generate the map. The Terrain Based Motion Adaptation block then uses the generated map to plan an optimal path through the environment and minimize the amount of time taken for the vehicle to reach the goal.

The ADAPT Input and Training & Simulation blocks are offline components of the architecture wherein data from inputs are used for training the segmentation model. The ADAPTive Vehicle Controls block generates steering effort and wheel speed control effort that enables the vehicle to safely circumvent or traverse through the area of wet road.

Overall, the various subsystems in the functional architecture work together to meet the system level requirements that are stipulated for the system.

5 System-Level Trade Studies

5.1 Platform Trade Study

Table 1: Platform Trade Study

Requirements	Weights (%)	Options					
		a	b	c	d	e	f
Maximum Speed	15	3.5	3.5	3.5	3.5	0.7	3.5
Drive	15	3.5	3.5	3.5	3.5	1.4	2.8
Steering	15	3.3	2.5	4.1	3.3	1.6	3.3
Payload Capacity	10	3.1	3.9	3.9	1.6	3.9	1.6
Cost-to-Us	10	4.0	1.0	1.0	5.0	2.0	5.0
Weight	8	3.8	3.8	3.8	1.5	3.0	2.3
Ease of Mounting On	8	3.8	3.8	3.0	2.3	3.0	2.3
Interchangeable Tire Materials	8	3.5	3.5	3.5	2.1	2.1	3.5
Structural Robustness	5	2.8	3.5	3.5	2.1	3.5	2.8
Availability of Parts	4	2.6	4.3	2.6	3.4	1.7	3.4
Aesthetics	2	2.8	3.8	3.8	1.9	0.9	4.7
Total	100	68.3	63.9	66.2	58.6	41.1	61.8

(a) Kingmotor RC Camera Car [3]

(b) Cinegears Camera Car [4]

(c) RC - Autorally GT [5]

(d) Kraken RC Car [6]

(e) Golf Cart [7]

(f) X-MAXX RC Car [8]

Figure 3: Potential Platforms

The most important traits for the platform are the maximum speed, type of drive system, and type of steering. The platform must be able to reach speeds that allow for it to perform extreme maneuvers. Four-wheel drive is important as it allows for better control of understeer and oversteer capabilities compared to just rear wheel drive. Ackermann steering is necessary to translate the results of the system to passenger vehicles.

Some other important traits are payload capacity, cost, weight, ease of mounting, and the ability to interchange tire materials. The payload capacity must be high enough to support the enclosure and items within that will be mounted on the car. The cost of the platform cannot take up a significant amount of the allotted budget. The platform must also have as large a weight as is possible to power and support, as a larger weight will result in system dynamics that scale-up more

realistically. The tires also must be interchangeable in the event that the stock tires are deemed to not be representative of passenger vehicle tires.

Other traits that were considered were the structural design of the platform chassis, the availability of replacement parts, and the aesthetics. The platform must be robust to vibrations and impacts with objects such that it does not sustain major damage during these situations. In the event that some components are damaged, there must be a readily available source of replacement parts. Having the ability to upgrade components on the vehicle was an important trait as well. Certain components on the platform ended up being upgraded after purchasing the vehicle, such as the suspension springs and tires.

Based on the trade study shown in Table 1, the platform that was chosen for the system was the Kingmotor RC Camera Car. Examples of each of the options considered can be seen in Figure 3 with their associated option label from Table 1.

5.2 On-board Computer Trade Study

Table 2: On-board Computer Trade Study

(a) Scoring Matrix

Requirements	Weights (%)	Options		
		a	b	c
Compute Power	25	2.5	4.1	2.5
Power Consumption	20	4.1	2.5	2.5
Cost-to-Us	15	3.8	3.0	2.8
Memory	10	2.1	3.5	3.5
Weight	10	4.0	3.0	2.0
Dimensions	10	3.0	3.8	2.3
Availability	5	3.8	3.8	1.5
Voltage Rating	5	3.0	3.8	2.3
Total	100	64.8	67.2	48.0

(b) Option Details

Options	Names
a	NVIDIA Jetson TX2
b	NVIDIA AGX Xavier
c	UP Xtreme

The most important traits for the on-board computer are compute power, power consumption, and cost. The GPU in the computer must have a great enough compute power to perform the real-time operations that are required, such as road segmentation. The computer cannot consume excessive power or else the system will be unable to run for extended periods of time. The voltage rating of the computer must be low enough that it can be run by the batteries the system will use. Additionally, the computer price cannot take up too much of the allotted budget.

Some other important traits are the memory, weight, and dimensions of the computer. It must have a sufficient amount of memory. A larger memory means an ability to store more data or to store items at higher resolutions, such as the localization map. The weight of the computer must not add a significant amount of weight to the platform, so as to not shift the center of gravity or

waste energy during testing trials. The computer must have dimensions to allow it to fit within a relatively small sized enclosure.

Other traits that were considered are the availability and voltage rating of the computer. Certain computers may have the capabilities necessary for the system to run quickly, but may be difficult to actually obtain on a tight schedule.

Based on the trade study shown in Table 2, the on-board computer that was used for the system was the NVIDIA AGX Xavier.

5.3 RGB Camera Trade Study

Table 3: RGB Camera Trade Study

(a) Scoring Matrix

Requirements	Weights (%)	Options			
		a	b	c	d
Frame Rate	20	3.6	1.2	6.0	1.2
Latency	20	3.4	4.3	1.7	2.6
Resolution	15	3.8	1.9	4.3	1.9
Shutter	15	1.0	1.0	5.0	5.0
Ease of Use	10	3.7	2.8	3.7	1.8
Cost-to-Us	5	3.6	4.0	0.9	3.6
Size	5	3.2	4.0	2.4	2.4
Edge Computing	5	5.5	1.1	3.3	2.2
Weight	5	3.2	3.2	2.5	3.2
Total	100	65.4	48.5	75.3	50.8

(b) Option Details

Options	Names
a	Pixy2
b	WolfWhoop WT03 Micro with Antenna
c	FLIR Grasshopper 3 GS3-U3-32S4C-C
d	RunCam Swift 2

Frame rate, latency, resolution, and shutter are the most important characteristics considered for the RGB camera trade study. The camera must have a high enough frame rate to capture details even when the vehicle is moving at a relatively high speed. The latency of the camera must be low so that the data obtained by the camera is not outdated by the time it is processed and utilized by the system. A higher resolution camera will provide finer detail images, which is important for accuracy in processes like segmentation. It is important to use a camera with a global shutter over one with a rolling shutter because global shutter will capture the entire image at once and is not as prone to artifacts as is one with a rolling shutter.

Other camera characteristics are ease of use, cost, size, edge computing, and weight. The camera should be easy to use in that it should not require a relatively long time to understand the software and perform initial setup of the camera. The camera must not cost too much of the allotted budget. The size of the camera must be small enough such that it can fit when mounted on the platform. Edge computing is beneficial because some processes such as segmentation could be offloaded from the GPU to the camera's edge computing, which would free up some space on the

GPU to perform other tasks. The camera must also be light enough weight as to not significantly contribute to the overall vehicle weight.

Based on the trade study shown in Table 3, the system ended up using the FLIR Grasshopper 3 GS3-U3-32S4C-C Camera.

5.4 Simulation Trade Study

Table 4: Simulation Trade Study

(a) Scoring Matrix

Requirements	Weights (%)	Options					
		a	b	c	d	e	f
Supports Nonlinear Dynamics	15	2.0	1.0	5.0	4.0	5.0	1.0
Path Planning, Tracking, Control	15	2.8	2.1	3.5	3.5	3.5	2.8
Road Surface Modeling	10	4.3	0.9	3.4	3.4	4.3	1.7
Ease of Use	10	3.3	4.1	1.6	3.3	2.5	3.3
Cost/License	10	4.1	4.1	0.8	4.1	0.8	4.1
Technical Support	10	2.8	1.9	3.8	3.8	2.8	2.8
Type of Suspension Models	10	2.8	1.9	4.7	2.8	3.8	1.9
Feasibility of Generating Training Data Set	10	4.7	0.9	2.8	2.8	4.7	1.9
Weather Modeling	5	5.3	1.1	1.1	3.2	5.3	2.1
Aesthetics	5	4.5	1.1	3.4	2.3	5.6	1.1
Total	100	68.2	39.0	64.3	68.3	74.2	46.0

(b) Option Details

Options	Names
a	CARLA
b	Gazebo
c	Dymola
d	MATLAB
e	NVIDIA DRIVE Constellation
f	CoppeliaSIM

The most important traits for the simulators are the ability of the simulator to model the road surface, whether the simulator supports nonlinear dynamics, the path planning, tracking, and controls, and the ease of use. Since the system will be determining the tire-road interaction during operation, the simulator must have the ability to model the road surface. The simulator must also support nonlinear dynamics, as these nonlinear models are more accurate to real-world scenarios. The path planning, tracking, and controls capabilities must be advanced enough to simulate what the system should do in the real world. The simulator must also be relatively easy to use. It is infeasible to spend a large amount of time figuring out how to use the simulator. Since there is a limited budget, the simulator must be of reasonable cost or a license must be obtainable.

Other important traits for the simulators are the amount of technical support, the types of suspension models, and the feasibility of generating training data sets. When issues occur during usage of the simulation software, there must be detailed documentation or some source of technical support from which the team members can refer to for help. Some simulators simulate the vehicle

using a point mass, but to approach more realistic behavior, a more complex vehicle suspension model is necessary. Ideally the simulator would have realistic enough aesthetics that can be used to train neural networks for segmentation of wet asphalt and puddles.

Additionally, since the system will be traversing wet asphalt and puddles in the real world, the simulator should be able to model the weather. A second aspect of having a simulator with realistic aesthetics is that the presentability of the simulation results would improve.

Based on the trade study shown in Table 4, NVIDIA Drive Constellation was the better choice, however, Drive Constellation could not be obtained during the duration of the project, so the system used MATLAB as its alternative.

6 Cyberphysical Architecture

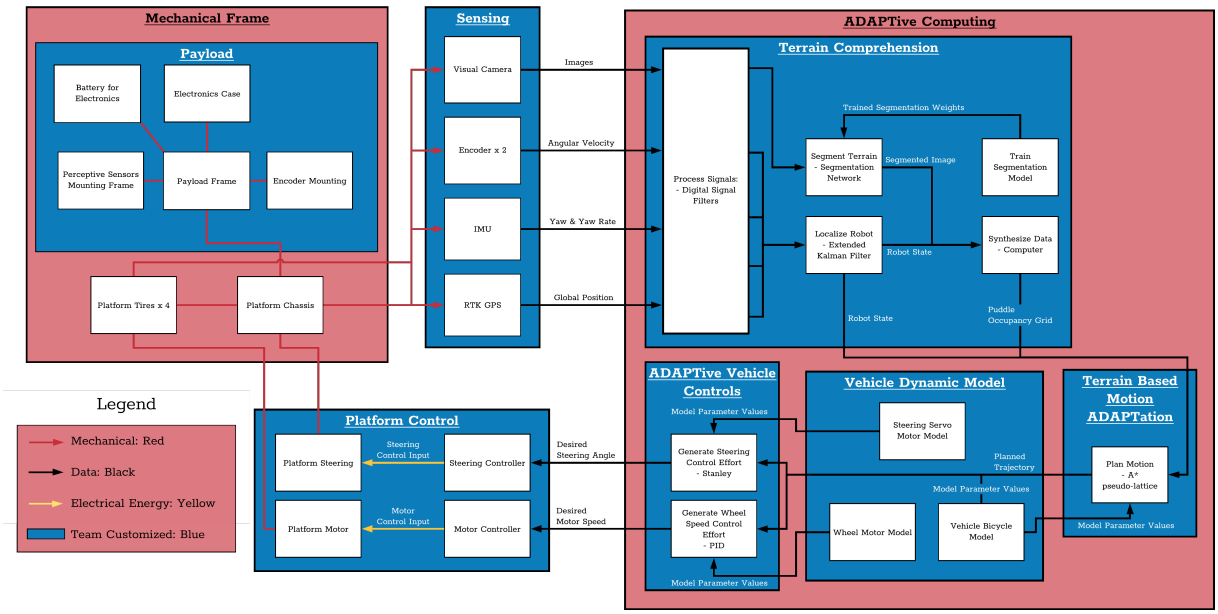

Figure 4: Cyberphysical Architecture

The first major section of the Cyberphysical Architecture, as seen in Figure 4, is the Mechanical Frame block. It consists of the entirety of the RC camera car and items that are mounted on the car such as the enclosure, sensors, and computer. The Sensing block contains the sensors that are used to obtain data necessary for the system to perform its tasks. The block includes the RGB camera, encoders, IMU, and RTK GPS. The data collected from those sensors are processed and utilized in various blocks like the Terrain Comprehension block. In this block, terrain is classified and segmented, and the robot localizes itself relative to its current environment. This information is synthesized and used to plan robot motion in the Terrain Based Motion Adaptation Block.

In the Training and Simulation block, a convolutional neural network is trained on labeled terrain data. This pre-trained network is used to segment terrain patches. The vehicle dynamic and

motor models are then utilized to generate steering and wheel speed control efforts based on the given state and trajectory in Adaptive Vehicle Controls. The control efforts lead to desired steering angle and motor speed commands that are sent to the steering servo and motor controller in the Platform Control block. The result is the vehicle being controlled to navigate adverse terrain.

7 System Description & Evaluation

7.1 Subsystem Breakdown

7.1.1 Platform & Hardware

The enclosure houses the sensors, computer, and auxiliary components. The CAD rendering is shown in the left of Figure 5, while the right of Figure 5 shows the components currently mounted within the enclosure. The frame was created from aluminum sheets that were cut using a water jet machine. Truss structures and mounts were 3D-printed with PLA, Nylon, and PVA. Major outstanding items from last semester were the lid and the window, which were completed early in the fall semester. The Lexan lid was bent to shape and attached to the enclosure via a piano hinge. The hardware necessary to achieve IP64 protection was also put into place. This involved lining the top of the enclosure with weather stripping and placing caulk along any seams of the enclosure. All external fasteners used rubber washers to prevent water from leaking through the bolt holes.

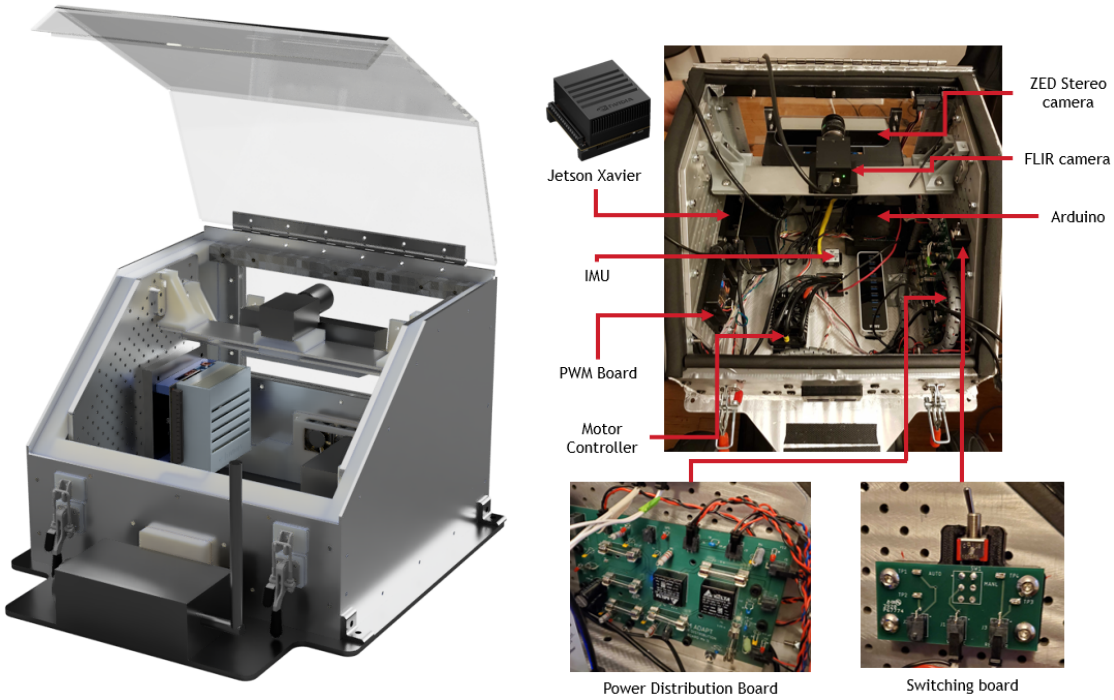

Figure 5: Enclosure CAD Rendering and Components Mounted within Enclosure

Sensors of the robot include an IMU, a GPS, a monocular camera, and encoders. The IMU is mounted near the center of mass of the vehicle to provide orientation. An RTK GPS is mounted on the back of the robot, with the GPS antenna on the top of the enclosure. A GPS base station is set up nearby to provide corrections. A FLIR Grasshopper3 camera is mounted at a small downward angle on an adjustable bridge inside the enclosure.

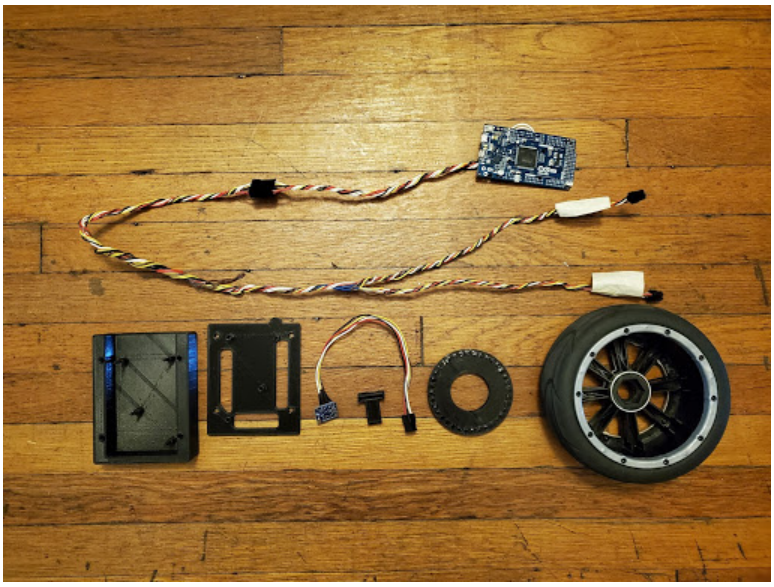

Figure 6: Encoder Assembly

Custom encoders are used because it is difficult to mount off-the-shelf encoders onto the vehicle shaft. Components for the encoder are shown in Figure 6. On the top is the Arduino and cable harness. The items on the bottom, from left to right, are the Ardunio case, Arduino case lid, magnetometer, magnetometer mount, magnet ring, and wheel. The custom encoder is comprised of a set of thirty-six cube magnets which are glued into a 3D printed ring. Adjacent magnets are oriented 90° relative to each other to create variations in magnetic field that a magnetometer can detect. The rings are attached to the insides of the front wheel hubs. Magnetometers are placed inside 3D printed mounts that are attached to uprights next to the wheel hubs and placed as close as possible to the ring of magnets. Cable harnesses are made for I^2C signals between the magnetometers and an Arduino Due. The signal cables are twisted to reduce noise and modularized using connectors. The raw magnetometer signals when the wheels are turning are shown in Figure 7. The magnetometer measures the magnetic field of three axes, and each wave corresponds to one axis, as depicted by different colors in the graph. A circular buffer is used to store the magnetic field values within a small sliding window. The Arduino processes these signals and converts them to velocities by counting the number of transitions between peaks and troughs of the signal. The two horizontal lines on the plot are the upper and lower threshold values for the blue and red curves to check whether transitions are valid. The Arduino also resets itself or power cycles the magnetometers when the they fail. The velocities are then sent to the Xavier via serial communication, where they are parsed and passed to the localization pipeline.

The onboard computer of the robot is a Jetson AGX Xavier. It is connected to a powered USB hub which connects to the GPS, IMU, Arduino, camera, and Wi-Fi dongle. The Xavier also

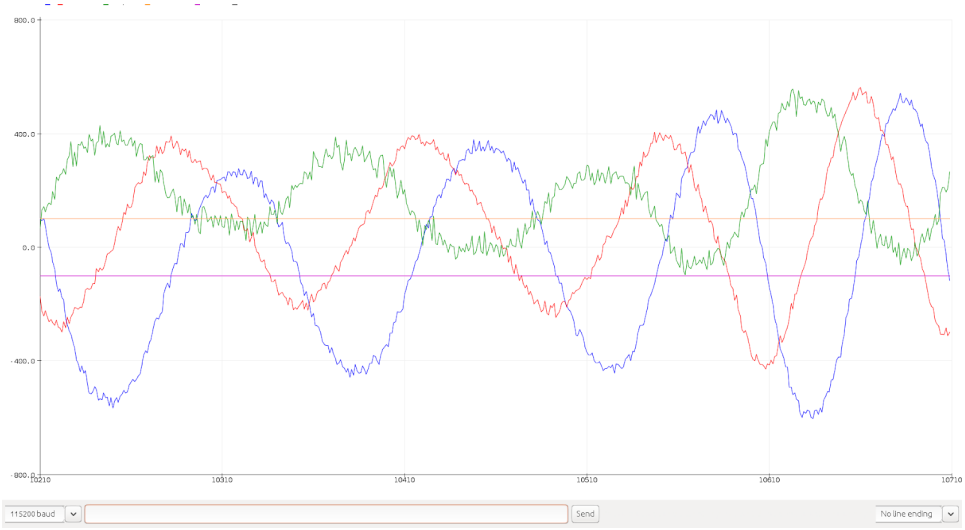

Figure 7: Magnetometer Readings

communicates with the PWM board via I^2C to send throttle commands to the motor controller and steering commands to the servo. The motor, servo, and RC receiver are powered by two 4S 14.8V LiPo batteries, and the PDB is powered by two 20V Dewalt drill batteries.

Shown in Figures 8 and 9 are the schematic and layout of the Power Distribution Board, respectively, that is used to power the system components. The board consists of components such as connectors, DC-DC converters, LEDs, and fuses. An initial connector supplies 20V power from two Dewalt batteries onboard the vehicle to the rest of the system. That is directly connected to the Jetson Xavier, GPS, and FLIR camera, all of which require 20V. Each component has its own fuse for protection and an LED to indicate whether the component is powered. A combination of a 470μF electrolytic capacitor and a 0.1μF ceramic capacitor is used as both a safety precaution and together act as a band-pass filter to remove noise. There are also two test pads near the battery to easily check the input voltage. To prevent damage to the board from reverse polarity, a Transient Voltage Suppression (TVS) diode comes after the battery connector.

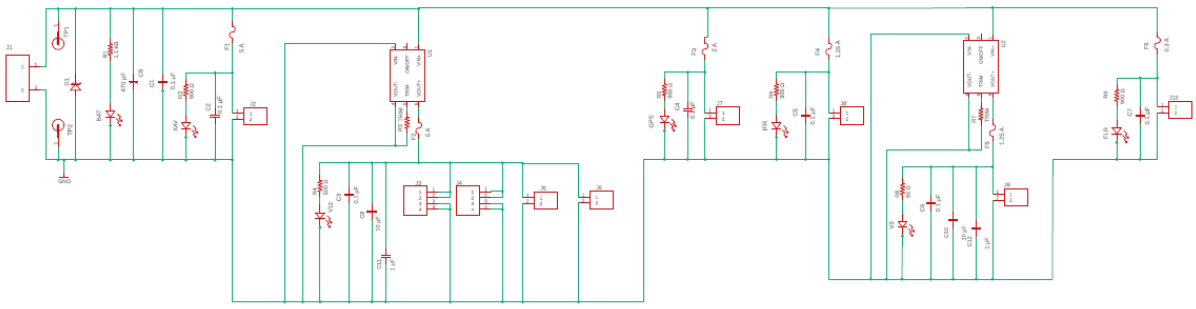

Figure 8: Schematic of the Power Distribution Board

The circuit branches out into two different parts based on the voltage requirement. There are two DC-DC converters that convert 20V into 12V for the fans and 3.3V for the PWM servo shield.

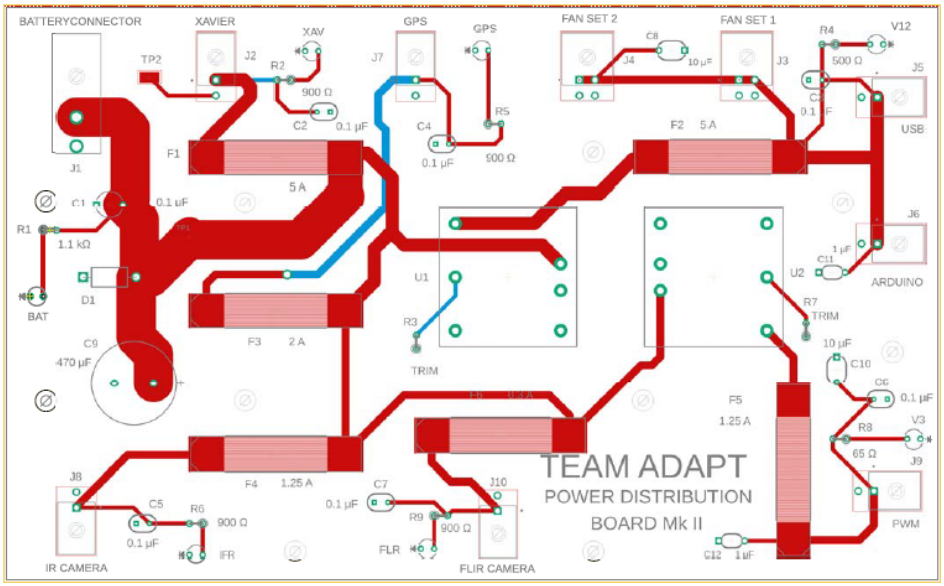

Figure 9: Layout of the Power Distribution Board

There is a fuse after each voltage regulator to protect each component. Since there are two pairs of fans and two encoders that have to be powered, individual connectors are used for each of them to increase the robustness of the board. If any one component on the board fails, it will be easy to identify the component that failed and replace it.

The fall semester iteration of the PDB has quite a few changes compared to the previous iteration. There were issues with the previous board that caused it to blow a trace. The errors in the first board were:
• The DC-DC converters were wired in the wrong configuration, where Vout- was only connected to the TRIM pin, and the built-in galvanic isolation feature was misinterpreted.
• Insufficient trace width at the beginning of the circuit. It had the necessary width for the first component, the Jetson Xavier, but at the first junction, the trace width should have allowed for a total of 10A, which is what is consumed by all the components.
• The TVS diode specification was incorrect and was also in the wrong configuration. It was later discovered that it should have a higher VCL (clamping voltage).

The above errors were rectified, and the overall design of the board was simplified to increase its robustness. Initially there were three voltage regulators with the third supplying 5V for the encoders. This was changed so that the encoders draw current from the Arduino Due board, which is connected to the USB hub, which in turn is connected to the 12V connection on the PDB.

A mode switch board was built to transfer control of the RC Car from the remote control (manual mode) to the computer (autonomous mode) and vice versa. The schematic and layout are shown in Figures 10 and 11, respectively. This functionality enables the manual driving of the car during testing and data collection.

An emergency-stop feature is also implemented to provide safety and protect the vehicle. When

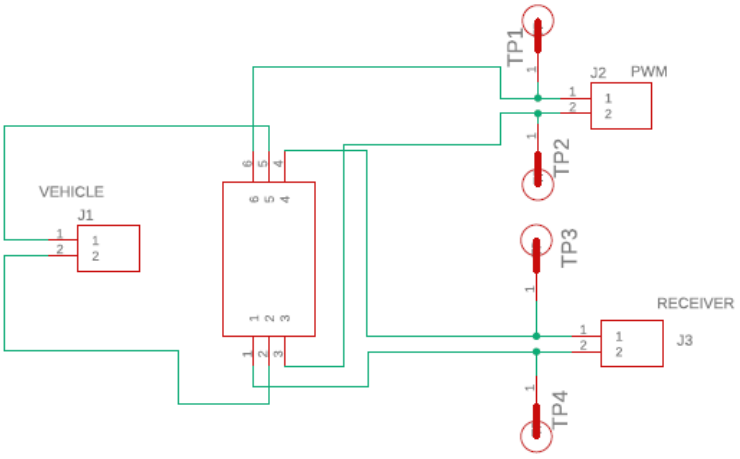

Figure 10: Schematic of the Mode Switch Board

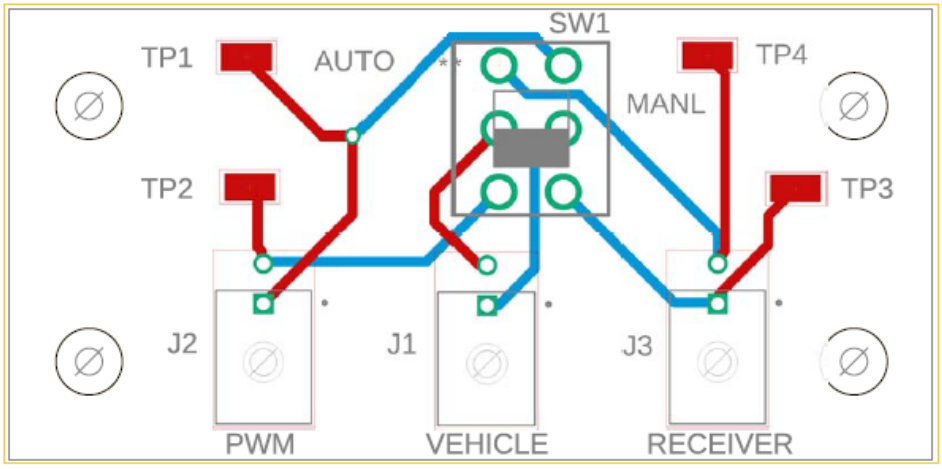

Figure 11: Layout of the Mode Switch Board

a button on the RC remote is pressed, an RC relay board is programmed to cut the throttle signal and stop the vehicle. By having the emergency-stop controlled by the RC remote instead of the computer, the vehicle is able to stop reliably even if there is no internet connection or if the computer freezes.

The switch board was also simplified from having a relay to only having an onboard toggle switch. This was done for primarily two reasons. First, this feature added redundancy to the robot, as a programmed e-stop button already existed on the car controller. The range on the controller is much better than the Wi-Fi range on which the relay would be operating, so removing the relay did not affect the overall function. Since the relay was removed, the connection to the Xavier was no longer required. As a result, only three connectors are seen in the schematic: one to the vehicle control unit, one to the receiver connector, and another one to the PWM board from which the control system operates.

Figure 12 shows the finished platform during a test session in the fall semester.

Figure 12: Vehicle Platform During FVD Testing

7.1.2 Terrain Comprehension

The Terrain Comprehension subsystem perceives adverse terrain and includes the classification, segmentation, and localization pipelines, along with a resulting map output that shows the location of the vehicle relative to the terrain.

The approach to classify and segment terrain last semester involved using a real-time instance segmentation neural network model called You-Only-Look-At-CoefficienTs (YOLACT)[9]. However, there were accuracy issues related to YOLACT being an instance segmentation network. Thus, YOLACT was replaced with FCHardNet, a PyTorch implementation of a semantic segmentation network called HarDNet [10]. YOLACT had not yet been implemented on the Xavier, whereas FCHarDNet does successfully run on the Xavier. After some optimizations, testing on the Xavier demonstrated a segmentation speed of about 18 FPS. Shown in Figure 13 is an example of a segmented frame. The blue regions represent puddles and the green regions represent road. The gray regions represent the "other" category.

Last semester, the training images were collected at Highland Park and annotated with the free LabelMe software. Since the FVD testing location was moved to the top level of the east campus parking garage, a new batch of training images was collected from there, and the old Highland Park images were no longer used for training. 516 images were manually annotated with software provided by Innotescus, rather than with LabelMe. The Innotescus software provides a greater number of features and a better user interface compared to LabelMe. These improvements enable more accurate labeling, which results in improved segmentation results, and an increase in labeling

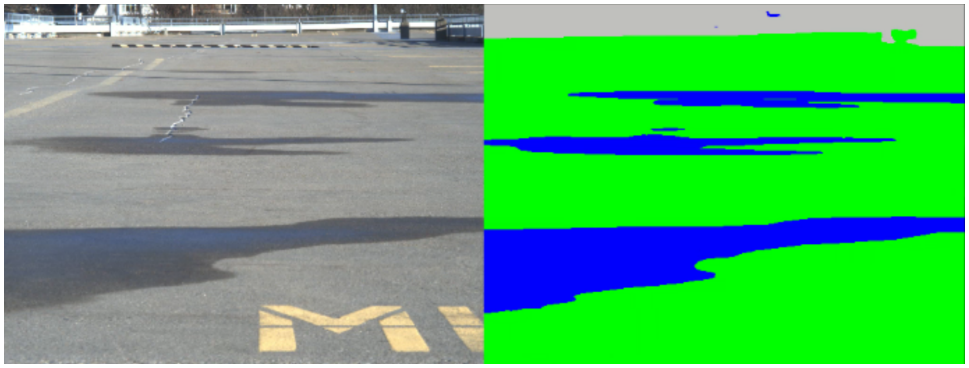

Figure 13: Side-by-side View of Original and Segmented Image

speed. In addition to the original classes of puddle and road from last semester, a third class of "other", which encompasses any non-puddle and non-road portions of the image, is used.

Usage of the ZED Stereo Camera for classical methods of detecting puddles was explored following the pipeline established in "3D Tracking of Water Hazards with Polarized Stereo Cameras" by Nguyen et al [11]. This involved placing polarized filters at different orientations over the two lenses of the stereo camera and calculating azimuth and reflection angles in the images. Ground plane estimation was then performed using RANSAC. Segmentation of puddle pixels is performed by training and evaluating a Gaussian mixture model where the features include azimuth angle, reflection angle, and saturation value of pixels. While the puddle segmentation was relatively accurate, the algorithm was unable to function in real-time. Therefore the ZED Stereo Camera was left out of the perception pipeline, and the only method of vision the system uses is the FLIR monocular camera.

As mentioned before, IoU is one of the metrics used for evaluating the performance of the network when segmenting road and puddle for a single frame. The other metric for evaluation involves looking at the percentage of frames with an IoU greater than some established threshold. Additionally, the average IoU across frames is used as a general metric to determine if segmentation performance is increasing as a whole. Simple qualitative inspections are also done to ensure the segmentation performs as expected.

The masks output by the segmentation network are used to generate an occupancy grip map, where the values in the cells of the map are the probabilities of puddles being at those locations. Shown in Figure 14 is a map generated from the segmented frame in Figure 13. The white regions represent puddles and the black regions represent road. The dimensions of the map are determined by the waypoints and road width. The origin (0, 0) of the map is set as the position of the robot when the system is turned on, and the x- and y-axes point to the right and upwards, respectively. Each cell represents a unit distance set as 0.1 meters.

The first step in this calculation is to warp the mask to a bird's eye perspective. The warp function required for this perspective transformation is obtained from the camera intrinsics and camera angle relative to the ground. Since the camera focus is adjusted based on how far ahead it views the ground, a checkerboard calibration was performed to obtain the intrinsics after settling

on a camera angle. The position and yaw information from the state is used to orient and overlay the warped mask correctly onto the map. A running average of puddle locations is weighted against the most recent mask received to calculate the final probability value that is stored in the cell. The map is then passed to the planning subsystem via ROS.

Figure 14: View of Map Frame from Segmented Video Frame

The duty of the localization system is to provide accurate and live information to the robot about its current state. The components of the state representation are $x, y, \theta, \dot{x}, \dot{y}$, and $\dot{\theta}$. It also includes acceleration information for each of x, y, and θ, but they are not used by any other subsystem. The state reference frame is defined as the pose of the robot when localization is started or reset. The frame is offset by a provided initial state, which most of the time is set as zeros. Additionally, the x-axis is set to always be parallel to the long side of the parking garage. In order to achieve live and accurate information, an Extended Kalman Filter (EKF) is utilized with three different contributing sensors: RTK GPS, IMU, and encoders. This suite of sensors was chosen so that direct measurements of each of the state values used by other systems could be obtained. The RTK GPS provides location (x and y) information. The IMU provides yaw (θ) and yaw-rate ($\dot{\theta}$) data. Finally, the encoders provide velocity information, which in conjunction with the yaw of the vehicle, is transformed into the two component velocities ($\dot{x}$ and $\dot{y}$).

7.1.3 Planning

The planning subsystem is tasked with generating a trajectory for the vehicle to follow given its current state and a map of the road. The planner also takes into consideration the limitations of the vehicle and the interactions between the vehicle and the road. Trajectory generation was accomplished in the spring using a Shooting RRT, where the planner grows a tree out from the current state. The process for Shooting RRT begins by selecting the node in the existing tree that is closest to the goal based on the distance metric. Over 50 shots are performed from that node's state with varying acceleration and steering angle controls. The shot with the best progress toward the goal based on the distance metric is selected, and all shots with the same steering angle are added as nodes to the tree. If none of the shots meet all limitations or are closer to the goal than the expanding node, then the expanding node is pruned from the tree. This process repeats until the iteration limit or goal is reached. The distance metric includes two components: the distance along the road from the point to the target and the angle of the vehicle's heading relative to the center line of the road. These two cause the planner to tend toward the goal and do so without oscillation about the center line.

This method would have been too slow to use at the speeds intended for this system, and as such, the planning system was reformulated around a simplified version of a lattice planner [12].

The planner developed in the fall semester was constructed with support from Braden Eichmeier and Shaun Ryer as part of a final project for the 16-782 course, and makes heavy use of pre-generation to speed up computation when compared to the spring implementation of Shooting RRT. Specifically, the current planning system pre-generates motion primitives based on a discretized set of start and end velocities and discretized steering angles which it propagates for a fixed duration of time to generate motion primitives. It also pre-generates the footprint of the vehicle for checking where the wheels would contact the road. Once a map is received, the planner takes the current state, selects a waypoint beyond the planning horizon, and fills out a heuristic map using backwards Dijkstra, where the heuristic value, time, is calculated using the minimum amount of time that could be spent traversing each cell. Cells containing puddles are limited to a lower speed than those without puddle. With the motion primitives and the heuristic map, weighted A* is then performed to find the best path from the current state to the selected waypoint. For safety, a phantom puddle is added around this target waypoint so that the trajectory slows down at its end.

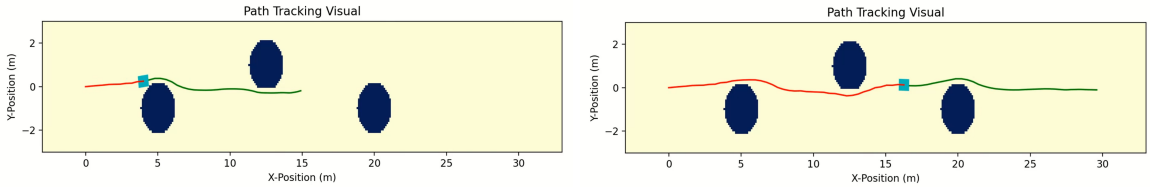

Figure 15: Online Motion Planning

Examples of two moments in the process of planning can be seen in Figure 15. Here the robot is planning online, where the red line represents the trajectory the vehicle has taken, the green line is the planned trajectory, the blue regions are puddles, and the cyan box marks the vehicle state. The left image exhibits when the system has planned a trajectory and is navigating around one puddle, planning around the next and to the intermediate waypoint at 15 meters. The right image shows the planned trajectory around the final puddle and all the way to the final waypoint at 30 meters.

7.1.4 Dynamics & Controls

The dynamics and controls subsystem houses the work of simulation development, vehicle dynamic modeling, and control algorithm design. The vehicle simulation is built within MATLAB and is used to test different dynamical system architectures, tune control gains, and evaluate robustness of motion planners. The simulation is set up with a dynamic bicycle model and linearized tire dynamics, and it utilizes a PI controller with anti-windup for longitudinal control and a Stanley controller for lateral control. The coordinate frame for this model and control algorithm implementation is based from the center of the front-axle of the vehicle, and as such, this is the reference for cross track error measurements. A visualization of the generated MATLAB simulation environment can be seen below in Figure 16.

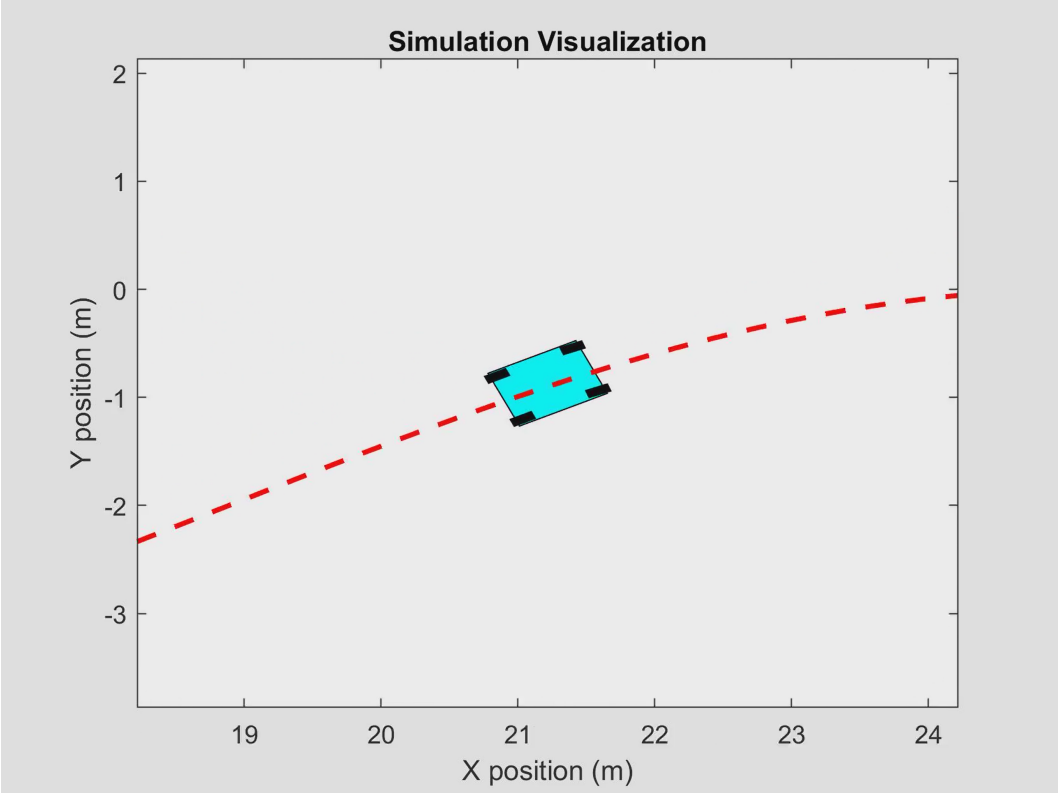

Figure 16: Simulation Running with Dynamic Bicycle Model

Where the longitudinal control law is [13]:

$$e_v(i+1) = k_{p,v}(v(i+1) - v_c(i+1)) + k_{i,v}e_{int}(i+1)$$

And e_v represents the longitudinal control effort, $k_{p,v}$ represents the proportional gain, v represents desired velocity, v_c represents current velocity, $k_{i,v}$ represents the integral gain, and e_{int} represents the integral error. The integral term is represented by:

$$e_{int}(i+1) = e_{int}(i) + (v(i+1) - v_c(i+1))$$

The lateral steering control law is represented by the equation:

$$\begin{aligned}\delta(t) = (\psi(t) - \psi_{ss}(t)) + \arctan\tfrac{k\ e(t)}{k_{\text{soft}}+v(t)} \\ +k_{d,\text{yaw}}\left(r_{\text{meas}} - r_{\text{traj}}\right) + k_{d,\text{steer}}\left(\delta_{\text{meas}}(i) - \delta_{\text{meas}}(i+1)\right)\end{aligned}$$

Where δ represents desired steering angle, Ψ represents current yaw position, Ψ_{ss} represents the calculated steady-state yaw angle based on the trajectory, k represents the lateral control gain, e is lateral error, k_{soft} is a control gain that provides numerical stability at low-speeds, v is the current vehicle speed, $k_{d,yaw}$ is a yaw-rate control gain, r_{meas} is the current measured yaw-rate, r_{traj} is the

desired yaw-rate, $k_{d,steer}$ is a damping control gain, and δ_{meas} represents the measure steering angle at different time steps.

As stated prior, this subsystem was first implemented in MATLAB for testing in the simulation environment. In the fall semester this was ported over to Python for use within the full system on the robot. After receiving the state and trajectory values from the localization and planning ROS nodes, the controls node calculates the desired steering and throttle commands, maps them to PWM signals, and sends them to the PWM board, which in turn continuously sends the signal to the electronic speed controller (ESC) and steering servo.

7.1.5 Interface

The Interface subsystem focuses on collating and relaying system information to and from users, along with inter-subsystem communications. On the back-end, this includes a variety of data collection and diagnostics tools that are passed on to the Graphical User Interface (GUI). This GUI is constructed within Python, and an example display can be seen below in Figure 17.

Figure 17: Visualization of GUI for the ADAPT System

This GUI provides the user with multiple pieces of information for operating, testing, and debugging the robot. There are two main plots located on the right side of the GUI. The top plot displays the robot's current position, the current planned trajectory, the path actually taken by the robot, and the map of the environment as generated by the perception subsystem. The bottom plot displays user selected sensor or state values that are listed, such as the vehicle yaw pose, encoder velocity, etc. The GUI is able to update with a usable frame-rate, running at about 20 FPS, but the latency between the robot and GUI was observed to be about 0.5 seconds.

The GUI was used mainly as a tool to perform testing and integration on other subsystems. The plots, sensors, and state values that are updated live during robot operation were critical in the tuning and testing of the longitudinal and lateral controllers on the robot, along with integrating the map-synthesis, motion planning, and controller together. The GUI was also used for debugging and improving the robot's localization pipeline by allowing the users to correlate the measured sensor values and the resulting estimated state values.

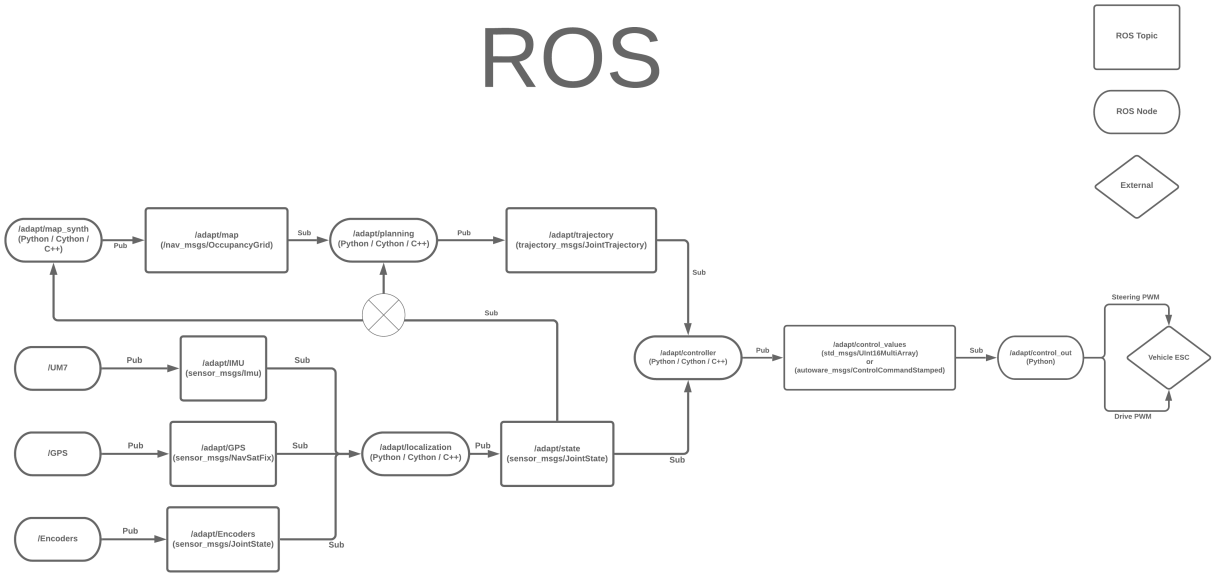

Figure 18: ROS Pipeline Constructed Used in Simulation and Onboard

A custom ROS pipeline was designed and implemented to manage data transfer between different subsystems within the on-board software. The architecture is shown in Figure 18, where the UM7 IMU, GPS, and wheel encoders are all recorded, published, and used by the localization process, which produces the current vehicle state message. That message is subscribed to by the map synthesis process, which uses the state to determine where to update the map when receiving new image frames. Based on the newly generated map and the current state, the motion planner generates a new trajectory for the vehicle to track, which then is passed to the controller as well. The resultant control efforts are then calculated and passed to a PWM control board over I^2C protocol, which end up being used by the ESC to generate motor movement.

7.1.6 Integration & Testing

Integration and Testing represents the recognition that a significant amount of work will be necessary to merge each of the different subsystems into the ADAPT system. Progress has been made in terms of integrating and testing the planning and controls subsystems in simulation as well as on the robot. Initial work has been done for combining the platform, specifically its sensors, with the planning and controls software. Testing the success of various integrated components has included developing multiple verification runs and assessing the output against expectations. An

example of this comes from the planner and controls subsystems in simulation where a road is given to the planner and the simulated car follows the resulting trajectory using the controller.

7.2 Modeling, Analysis, and Testing

7.2.1 Platform & Hardware

The platform was tasked with providing the capabilities for the other subsystems to use in order to meet the system requirements as well as housing and protecting the electrical components. This is the purpose of all the non-functional requirements. Once the enclosure was assembled on the vehicle, a variety of basic tests were performed to verify that the platform met these requirements. For M.N.1. this included driving the vehicle by remote control faster than 4 m/s and slower than 2 m/s. In order for M.N.2. to be satisfied the platform needed to be able to run without any external wired connections, which it can. Finally, with M.N.3. the ingress protection of the enclosure was tested on many occasions when the robot was run in the rain, with the interior of the enclosure remaining dry.

In addition to meeting the project requirements, other aspects of the system needed information about the platform in order to be able to work optimally. Localization calculates the state at the center of gravity of the vehicle and controls also uses this location on the robot for its calculations. To find the center of gravity on the robot, four scales were used, one under each wheel, to find the weight distribution. The track width and wheel base of the vehicle were also verified by hand measurements relative to the values provided by the system documentation. Controls also needed a mapping from steering and throttle commands to PWM signals that the ESC could use. These values were captured by using an oscilloscope connected to test points located on the switch board. When the vehicle was in manual control mode with the ESC off, the remote control trigger position was placed at the extremes for each of left steering, right steering, forward throttle, and reverse throttle signal. The associated PWM signals were captured and recorded in the control code.

Electrical testing was also a major focus in order to avoid damaging another round of PCBs. Test points built into the new board were used with a multimeter to check connections and voltages as the boards were assembled and before any external components were plugged into their connectors. The multimeter was also used to perform continuity checks across wire and soldering connections. Finally, pull tests on the connectors verified that they would not come apart unintentionally.

Beyond the electrical tests mentioned above, the custom encoders required additional analysis in order to ensure they were operating optimally. The first of these was finding the mounting location for the magnetometer where it would get the best possible signal from the magnet ring. For this, a variety of locations along and twists relative to the upright were checked. For each of these, the signal output was observed and the location with the strongest signals closest to a sinusoid were chosen. Figure 19 presents an example of a location where the signal took the shape of a series of M's and W's as compared to the signal taken from another location shown in Figure

7. In Figure 19 the blue and gold lines represent x and y axes data, which are the components used for measuring ticks. These undesirable signal features increased the chance of misreading a transition from peak to trough or vice versa. Once the magnetometer was fixed in place, these same signal readouts were used to place the thresholds to provide the most robust tick readings. Finally, all of this was verified by pushing the vehicle for one revolution of its wheels and comparing the number of ticks measured by the custom encoder with the number of expected ticks per revolution.

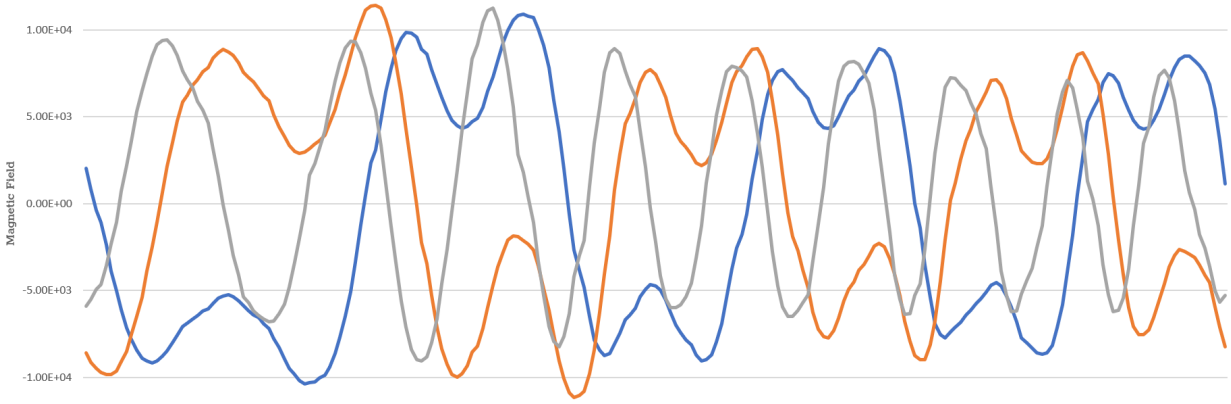

Figure 19: Magnetometer with Low Quality Magnetic Field Signal

7.2.2 Terrain Comprehension

Localization has no explicit requirements, but it needs to be sufficiently accurate to be useful for map synthesis, planning, and controls. The process to get localization working as needed started with testing each of the relevant sensors independently. To test the IMU, the vehicle was rotated by hand, and it was verified that the state yaw value matched that rotation. The test then progressed into driving the vehicle in arcs and circles to perform the same verifications. The position sensor, an RTK GPS, was tested by resetting the origin, moving the vehicle to a known location, and measuring the difference between the vehicle's actual and localized states. Latitude and Longitude coordinates obtained from Google maps were also used to verify the GPS output. It was found that the uncertainty in the GPS was less than ten centimeters, sufficiently small relative to M.P.5. for the other subsystems to use effectively. For the encoders, a video was captured of the vehicle traveling at constant speed across parking lines to check that the measured and ground truth values matched. When the encoders were not resetting, they were typically within 0.1 m/s. The localization's integration of velocity was also tested with the encoders by pushing the vehicle forward and comparing the actual measured distance traveled with the pose of the robot from localization. While less accurate than the RTK GPS, this was within 10% error.

Terrain comprehension had requirements that called for a minimum IoU for segmentation. The efficacy of the segmentation task was analyzed after every new round of data collection and network training. One condition under which segmentation initially had trouble was during sunny days when shadows were cast on the road. When the sun was lower in the sky during the late afternoons, the light poles and the small structures at the top of the parking lot cast large, high-contrast shadows. The network mis-classified those regions as puddles. To combat this issue, more

images of shadow were collected and added to the training images. The network was much more robust against these mis-classifications afterwards.

The initial implementation of FCHarDNet on the Xavier yielded a 10 FPS segmentation speed, which was too slow. Part of determining how to speed up the process was to time each individual section of code that went into the overall segmentation. The sections were image loading, image pre-processing, segmentation, post-processing, and stream display. The segmentation and image loading were the most time consuming portions, thus multi-threading was used to simultaneously segment a frame and load the next frame to be segmented. Additionally, the images were down-sampled to a fourth of the original resolution from the FLIR camera. After implementing these optimizations, a speed of 18 FPS was achieved, which is acceptable based on the desired robot speeds and planning horizon.

The first implementation of map synthesis ran at approximately 5 FPS, which was too slow. Each section of the code was timed as well, and these sections included grabbing the mask image via ROS, warping the mask, downscaling the mask, updating the map, and passing the map message to the planning node through ROS. The most time consuming operations were warping and downscaling the mask. One improvement in the code was to pre-allocate the arrays used to store the results of warping and downscaling, as dynamic array allocation is a costly operation in terms of time. This resulted in an improvement of about 2 FPS, but this was still not an acceptable speed. Next, optimizations related to warping the mask were implemented. The mask warping operation was relatively slow because the camera angle relative to the ground required warping the image through an extreme angle, typically around 80°. The top rows of the masks represented the largest regions of the resulting perspective transform because they were heavily stretched. Therefore, the top rows were cropped from the top of the masks prior to warping. Using a crop of 100 rows resulted in the overall map generation pipeline running at about 15 FPS, which was acceptable. The trade off was that the planning horizon was just long enough to meet the required speeds.

The accuracy of the map also had to be improved. Since the ZED camera was no longer a part of the pipeline, there was no standard depth information available to the perception system. To solve this issue, images were taken when the car was facing perpendicular to the parking lines. The location of the car relative to the first visible parking line is known, and the distances between each of the parking lines were measured and noted. The pixel positions of the parking lines on the image can then be mapped to a real world distance. Thus, each section of the segmented mask and the resulting warp can be mapped to a distance. To improve the accuracy of these puddle distances on the map, more parking lines were used as reference points to section the map into smaller regions. With a finer resolution of known distances, the map is able to generate more accurate placements of puddles.

The probability calculation of the puddles on the map were also improved upon. The initial implementation was based off of a running average of all the masks that had been merged up to that point in time. However, this meant that information from more recent masks would not show up on the map quickly. A weighting factor is used to fix this issue. The existing map probability is weighted at some percentage, while the information from the incoming mask is weighted at the complimentary percentage.

7.2.3 Planning

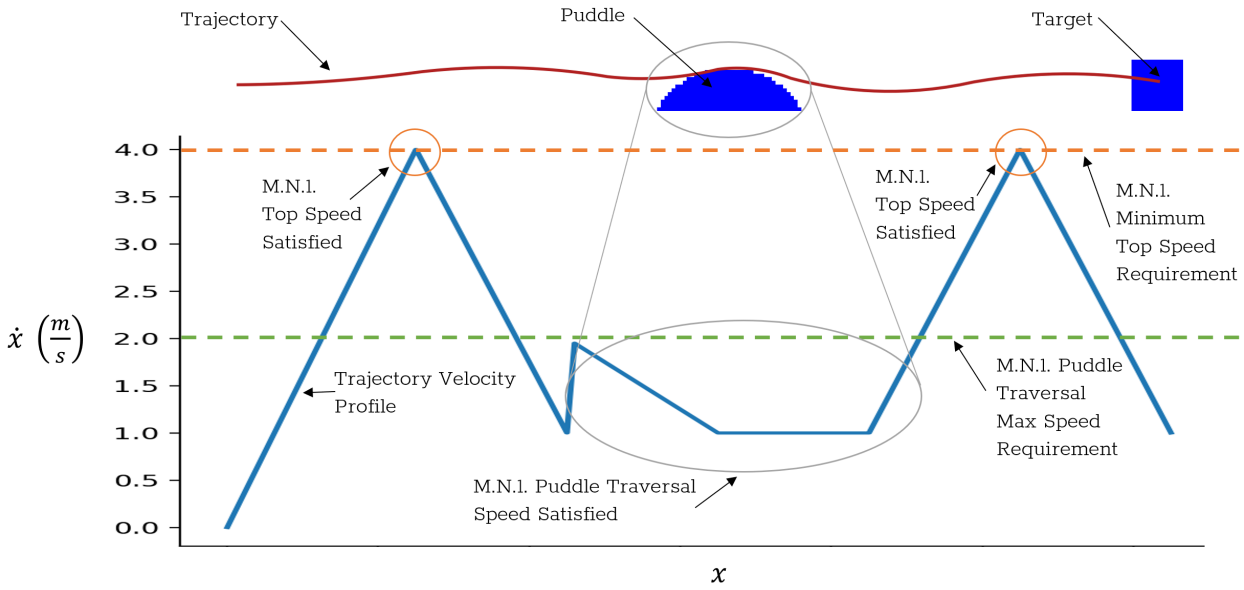

Figure 20: Planning's Speed Requirement Checks

The planning subsystem pertains to one non-functional requirement, M.N.1., and three performance requirements, M.P.3., M.P.5., and M.P.6. A brief analysis verifying that the planner meets M.N.1. can be seen in Figure 20 where the ellipses note the locations in the trajectory velocity profile that satisfy the components of the requirement. Specifically, the orange ellipses mark where the velocity of the planned trajectory reaches or exceeds 4 m/s and the gray ellipse notes where the planner is planning over a puddle traversal and proscribes a speed less than or equal to 2 m/s. M.P.3. was verified by observing the trajectory plotted from online mode or offline testing and checking that the planning subsystem either commanded the vehicle to stand still or to travel more than the planning horizon distance, which it always did. Due to the construction of the map it was not possible for the planned trajectory to cause the vehicle to exceed M.P.5. unless there was a mistake in generating the road representation in code. Cases were tested where the planner would have had no valid motion primitives other than to go off the road, and in each case the planner returned that there was no viable path to the goal. Finally, testing M.P.6. comprised of checks similar to that shown in Figure 20 as well as checks that the system will plan around puddles when the max traversal speed is dropped to zero, which it does.

Aside from simply meeting the requirements, other work was done to ensure that the planning subsystem operates as expected. As a part of this, the planner was set up such that it could be run separately from the rest of the system, in offline mode, and using the debugger in Visual Studio Code. Once the planner ran without issue in that setting, system integration testing was next. Before trying to use the map output from Terrain Comprehension, a dummy node was used that outputs a map in the same format as the map synthesis node but with phantom puddles. This allowed the testing of online planning with localization, controls, and a map of puddles with known locations. Finally, a visualization environment was developed that emulated the ROS interface and allowed for further testing of the planner when planning online.

7.2.4 Dynamics & Controls

Testing the various components of this subsystem involved multiple steps. First, simple test trajectories such as straight lines, constant radius turns, and sinusoids of varying amplitudes were generated. The vehicle then tracked these trajectories, and the dynamic responses were inspected to ensure that they made physical sense. This involved checking that the responses did not contain ballooning accelerations and that the tracking error was bounded and did not gradually increase. After ensuring the simulation had accurate physics and a stable controller design, different plots were generated to evaluate the responses to determine whether the performance metrics were met.

While on the robot, controls was tested after localization worked to a sufficiently accurate level. This was done by utilizing a dummy planning node within the ROS architecture which took in a trajectory and continuously republished it, thereby emulating the interface that the actual planning system would use to provide the trajectory for the controls system. Then, the distance traveled was compared and the speed of the vehicle over the trajectory was observed and this data was used to verify that the controls systems was properly tracking the provided trajectory. The set of trajectories started with short, slow, and straight trajectories and progressed to curvilinear and variable velocity trajectories. Controls relates to M.N.1., M.P.4., M.P.5., and M.P.6. where M.P.4. is trivially satisfied because the PWM board sends its signal at 55 Hz. As for the other metrics, all were verified by running the system and observing the live state information on the GUI and comparing with the requirement's specifications.

7.2.5 Interface

The interface had various features and communications functions built within it that required iterative testing to ensure functionality. This process began with testing the GUI construction and front end functionality verification, such as buttons, tabs, or figures being properly visualized. Then, testing when connected to the robot, while sitting immobile, was performed. In this scenario, all of the data being published that is of interest to the user is updated in the GUI and checkouts are performed to ensure that data is fully received and visualized. This was a continuous process throughout the system development to constantly improve the GUI's capabilities.

7.3 Fall Validation Demonstration Performance

7.3.1 Demonstration #1

For Demonstration #1 of the Fall Validation Demonstration (FVD), the terrain comprehension subsystem was evaluated against M.P.1 and M.P.2., which stipulate the accuracy of the segmentation of road and puddle, respectively. For both road and puddle, over 65% of frames had an IoU of at least 65%, meaning that the requirements were successfully achieved. The mean IoU across all frames of each class for the train, validation, and test datasets are shown in Table 5. The perception subsystem was also able to run in real-time at 18 FPS on the Jetson Xavier.

Table 5: Segmentation metrics of the perception system

Dataset	Mean IoU %	Background IoU %	Road IoU %	Puddle IoU %
Train	97.83	99.06	99.38	95.03
Validation	94.09	96.83	98.29	87.13
Test	95.61	97.97	98.59	90.28

7.3.2 Demonstration #2

For the second demonstration, the entire pipeline of the system from start to end was demonstrated. The performance requirements that had to be satisfied for this demonstration were M.P.3., M.P.4., M.P.5., and M.P.6. which stated that the planning horizon had to be greater than 4m, the vehicle had to exhibit real-time motion controls, the maximum distance from the waypoints had to be 0.75m, and 60% of the puddles had to be traversed at speeds less than 2m/s or avoided completely. A live feed of the real-time terrain segmentation was displayed. Sensor values, state estimates, and trajectories generated by the online planner were also shown on the GUI. The trajectories were constantly regenerated to account for deviation from the path and new terrain information. Since the GPS board was damaged on the day of the demonstration, localization accuracy suffered. This also affected the performance of planning and controls. However, the IMU and encoders still provided state estimates that system could use to avoid puddles. An example of the robot tracking around puddle can be seen in Figure 21.

Figure 21: The robot tracking a trajectory around a puddle

7.3.3 Demonstration #3

The third demonstration tested the robot enclosure's Ingress Protection (IP64) capability with a splash test to satisfy the non-functional requirement M.N.3. It involved pouring water over the

enclosure and checking if the interior was dry to touch. At FVD encore, the test was conducted with a suitable amount of water, which the enclosure withstood resulting in a dry interior.

7.4 Strengths and Weaknesses of Current System

The hardware is robust and durable. All components were still functioning after testing on rainy days. Even after a full speed crash during testing, the functionality and performance of the robot were not affected despite slight deformation to some structures. An area of improvement is serviceability. There was consistent need to reach underneath the chassis to swap batteries, and length of the cables and the tight space led to the tasks being more difficult than reasonably necessary. The enclosure was heavier than necessary, and plans to re-machine the enclosure to remove excess material fell through.

The semantic segmentation network exceeds the performance metrics and qualitatively works well. Improvements from the fall semester include running real-time on the Jetson Xavier, switching to a more accurate segmentation network in FCHarDNet, creating more and better quality annotations, adding a background class, performing data augmentation, normalizing the dataset, saving the weights with the best performance on the validation dataset, and using multi-threading. The FPS of the segmentation is also higher than the frequency the perception pipeline runs at.

One area that could use improvement is the map synthesis pipeline. The generated map is not consistently accurate in terms of puddle placement. A potential improvement is using the roll and pitch from the IMU to correct for shifts in the camera stream caused by acceleration, deceleration, turning, or vibration. Another improvement is to implement multi-threading or other optimizations to decrease time to perform image warping transformations.

Localization is one of the strengths of the system. The pipeline is able to obtain centimeter-level accuracy by passing sensor data through the EKF. The estimated velocity and orientation are also accurate and consistent.

Another strength of the system is the planning pipeline, which was re-vamped during the Fall semester, and achieves good performance in simulation and on the robot. Utilizing the ROS parameter server allowed for altering planning parameters easily, such as switching between offline and online planning modes and changing the planning horizon. Given a map, the planner is able to generate the optimal trajectory to the goal consistently, and at a high frequency when running online.

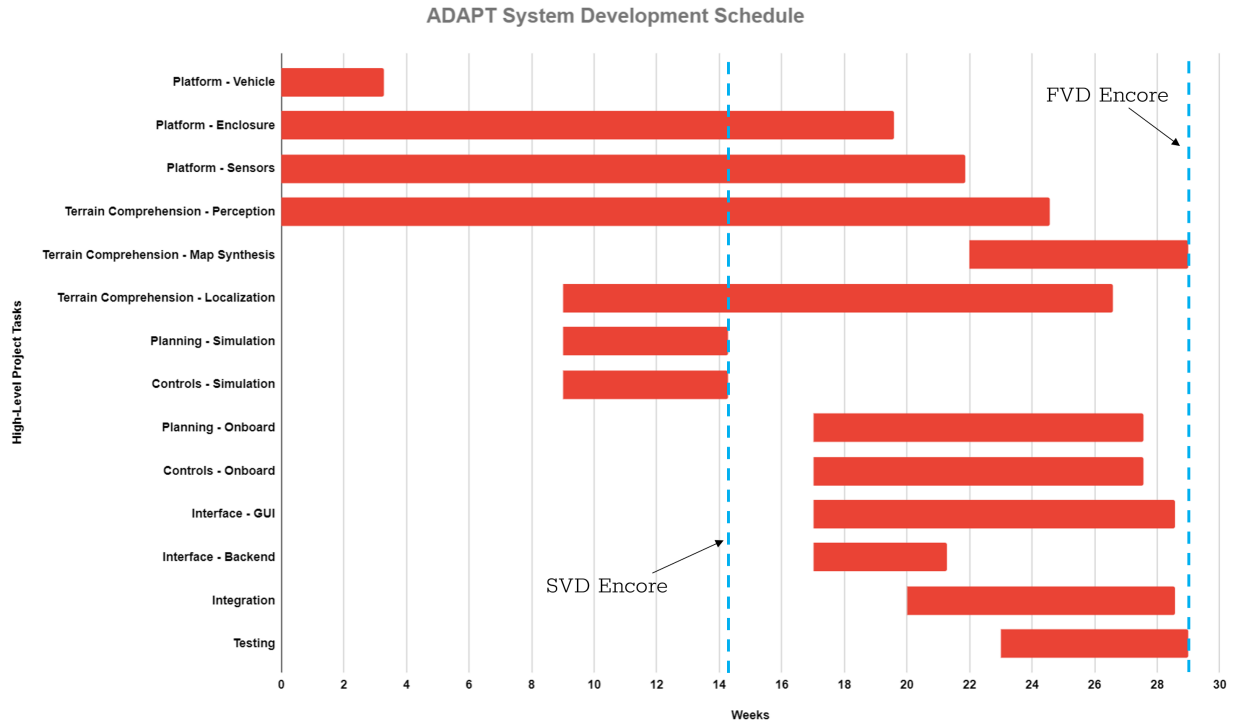

Figure 22: Project Gantt Chart

8 Project Management

8.1 Schedule

The scheduling process consisted of considering all of the work tasks that needed to be done, their dependencies, and how long each was estimated to take. With this information collated, the tasks were ordered into sequential groupings of roughly two week sprints, such that dependencies were met and each team member had something they could focus on for that time period. In terms of developing a schedule, this process was successful in aiding the understanding of how components and subsystems would need to come together and generating a sequence that would allow for the entire system to be completed.

In practice however, the time estimates for the work packages were often not aligned with the duration of work that was actually required to complete them. More often than not, the extra time required beyond what was allotted was due to realizations during the process of completing the work task. This was mostly related to debugging in that the estimated times were overly optimistic on how well the team would be able to implement the system on the first iteration. Localization presented a prime example of this. The majority of the system worked on initial implementation, however, there were significant issues with the jacobians for some sensors. At first, each sensor received the full analytical jacobian, it was then discovered that some of the terms would actually degrade performance. For example, the encoders were dwindled from a four wheel implementation with full transformations to a simple bicycle model. This testing and re-implementation process

far exceeded the expected amount of time by nearly an order of magnitude from what was initially estimated.

Although necessary changes uncovered during testing added time across many subsystems beyond expectations, the other major driver of subsystem delays were due to necessary reworks. Two notable examples of this come from perception and planning. For much of the first semester and some early weeks of the fall semester, efforts were put into getting Docker operational on the Jetson Xavier to emulate x64 chip architecture on the Xavier's arm chip architecture so that a segmentation network for which there was no implementation on arm architecture network could be used. When revisiting the idea of finding an arm implementation of a network, one was finally found that could run on the Xavier after much research. This change from Docker to direct implementation on the Xavier was necessary to explore and understand, but cost an enormous amount of time.

In the spring, the planning subsystem was implemented in simulation based on Shooting RRTs. Only near the end of the semester after significant effort was expended to get the program to meet the performance requirements was it realized that the entire planning algorithm needed to be reworked. As a result, planning largely started from scratch in the fall semester after setting out its structure over the summer. Had work on the pseudo-lattice planner been started from the outset of project, then over a months worth of working time would have been freed up for other aspects of the project that were initially planned for.

Understanding where time was actually spent as opposed to where that time was supposed to be spent shows that much of the shortfall in terms of scheduling came down to the lack of team experience in working on large projects. This lack of experience is related to not only estimating the amount of time it takes to implement a desired system, but also to debug it. The team also lacked experience in the different systems that were being implemented and the knowledge of which algorithms were most likely to provide the necessary results.

Figure 22 shows the span of weeks that different aspects of the project took to get from start to finish. The chart starts with the first week of classes in the spring semester and ends with the week of Thanksgiving, excluding the summer weeks between the end of the spring semester and the start of the fall semester.

8.2 Budget

Included is the bill of materials shown in Table 6. The Cost column is the total cost of the items, even if they were obtained for free from the MRSD inventory or from other sources like NREC. The electronic components category includes components such as connectors, resistors, capacitors, DC-DC converters, etc. as well as wire harnesses. This bill of materials is separate from the amount of money spent from the budget. Tools, spare parts, and other items were purchased and are not on the robot. The total amount spent was $4,683.74 out of $5000.00, and a breakdown of how the money was spent is shown in Figure 23.

Table 6: Bill of Materials

No.	Description	Part Name	Quantity	Unit Cost	Cost
1	RC Vehicle	Radio Control X2 Deluxe Brushless Electric 4WD RC Camera Car	1	$1,472.49	$1,472.49
2	RC Vehicle Battery	5000mAh 4S 14.8V Smart Li-Po 30C IC5	2	$79.99	$159.98
3	On-board Computer	NVIDIA AGX Xavier	1	$699.00	$699.00
4	Electronics Power Supply	Dewalt 5.0Ah 20V Battery	2	$69.25	$138.50
5	RGB Camera	FLIR Grasshopper3 USB3 GS3-U3-32S4C-C	1	$1025.00	$1025.00
6	GNSS and Base Station	SparkFun GPS-RTK2 Board - ZED-F9P	2	$219.95	$439.90
7	GPS Wi-Fi Communication	Digi XBee SX 900 RF Module	2	$199.00	$398.00
8	Encoder	Triple-Axis Magnetometer MLX90393	4	$9.95	$39.80
9	IMU	UM7 Orientation Sensor	1	$164.95	$164.95
10	Enclosure Plate 1	Front Outside Plate	1	$45.00	$45.00
11	Enclosure Plate 2	Tapered Outside Plate	2	$27.50	$55.00
12	Enclosure Plate 3	Back Exterior Plate	1	$40.00	$40.00
13	Inner Mounting Plate	Cheese Plate	2	$135.00	$270.00
14	Sensor Mounting Plate 1	Camera Mount	4	$8.75	$35.00
15	Sensor Mounting Plate 2	Camera Bridge	2	$12.50	$25.00
16	Base Plate	Base Plate	1	$195.00	$195.00
17	Window Plate	Window Plate	1	$35.00	$35.00
18	Fasteners	Bolts, Nuts, Straps	70	$0.10	$7.00
19	PCB 1	Power Distribution Board	1	$33.00	$33.00
20	PCB 2	Switching Board	1	$33.00	$33.00
21	Electronic Components	Misc. Components	N/A	N/A	$300.00
Total					$5610.62

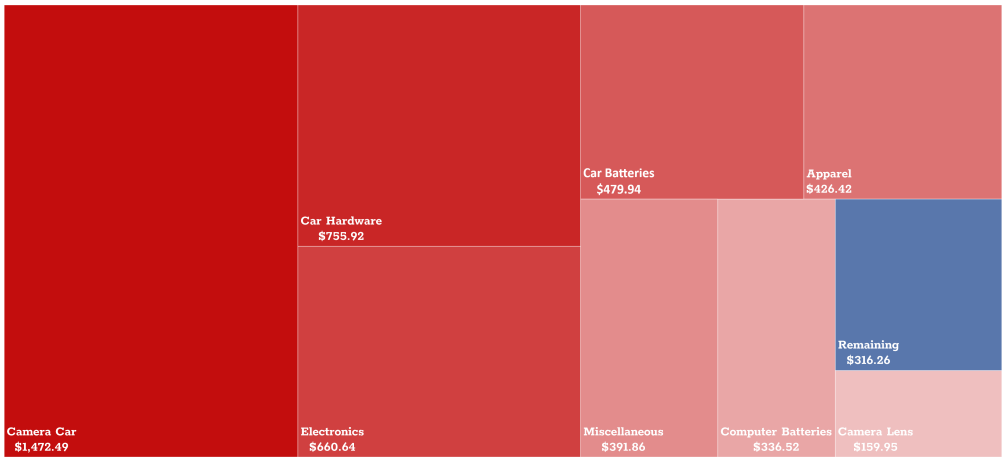

Figure 23: Budget

8.3 Risk Management

Figure 24 shows the risks that were anticipated for the fall semester along with the planned mitigation and backup actions for each risk. Figure 25 shows the plot between likelihood and consequence for the above risks and risks that fall in the red region are constantly monitored by the team.

#	Risk	Req	Type	Description	Likelihood	Consequence	Risk Reduction Plan
1	Inoperable Platform	M.N.1., M.N.2., M.N.3.	Schedule, Cost	Platform Breakdown	4	3	Stock backup components
2	Unsafe Behavior	M.P.4., M.P.5., M.N.1.	Technical	Infeasible Controls	2	4	Build in safety margin to motion planning
3	Cannot Extrapolate	M.P.5.	Technical	Vehicle Dynamics Inaccurate	4	3	Test custom model in simulation based on empirical evidence
4	Unpredictable Behavior	M.P.1-2.	Technical	Terrain Comprehension Inaccuracy	3	4	Use multiple sensors and algorithms
5	Buggy Demo	M.P.1-5., M.N.1.,M.N. 3.	Technical	Insufficient Time for Testing	5	4	Unit testing and descope if necessary
6	No Funds	M.P.1-4.	Cost	Overspend Budget	3	4	Make sure to source from inventory first
7	Water leaks	M.N.3.	Technical	Water seeps through gaps in enclosure	2	5	Test for gaps in advance and seal if required
8	Unable to Reach Speed	M.N.1.	Technical	Realtime Computation not Achieved	4	4	Track computation complexity in algorithm selection
9	Xavier crashes	M.P.1-2. , M.N.1.	Technical	Xavier needs to be reflashed just before the Demo	3	5	Extensively test software before adding it to the Xavier
10	Unable to Finish	M.P.1-5.	Program	Reduced Personnel	1	4	Consistent documentation and backup planning
11	Unable to Demo	M.P.1-5., M.N.1-3.	Program	Inhibiting Weather During Demonstration Day	3	3	Backup videos from a test
12	PDB malfunctions	M.N.2.	Technical	Trace degrades breaking the circuit	2	4	Check connections on the board regularly before supplying power

Figure 24: Risk Table

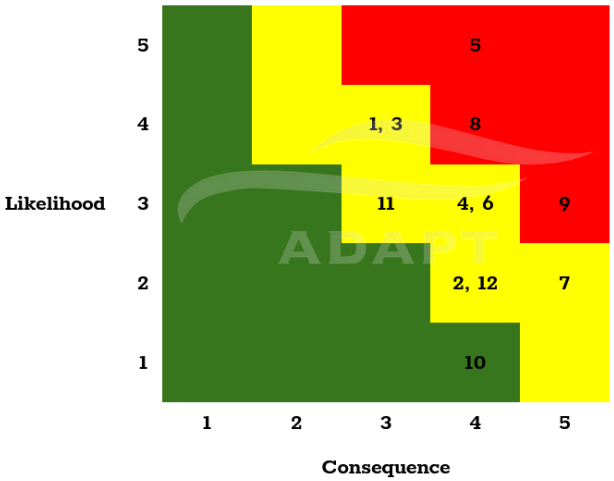

Figure 25: Risk Likelihood-Consequence graph

9 Conclusion

9.1 Key Lessons Learned

1. **Value risk mitigation:** There were preventable issues that lead to delays from the schedule that had been set. The team needed to more consistently follow through on any risk mitigation plans that had been made and been more conscious about risks in general.

2. **Learn to move on:** The team often found itself sliding down rabbit-holes. There were aspects of the system that the team was attempting to implement but were ultimately unable to. All of that time was essentially wasted, especially when the effort was not spent on a critical task. A key lesson was to make compromises and explore alternative options if one method does not seem likely to work.

3. **Focus on what is important:** There were times when members focused on minor details or desired features of the system before the baseline was completed. Some details were not important in the long run but took up valuable time nonetheless. Completing advanced features would have been meaningless if the basic system was not functional.

4. **Track individual progress:** The team used Jira and stand-up meetings to adhere to the work plan and ensure that everyone was making progress. This held members accountable to the tasks they were responsible for.

5. **Buffer in scheduling:** Starting with an unrealistic schedule and high expectation for the system resulted in constant rescheduling and de-scoping when milestones were missed. Instead, it is important to aim for a minimum viable product initially to ensure that a working system could be completed on time, and then add desirable features afterwards. The team understands now that it is better to add a buffer and overestimate the time required to complete tasks than to be optimistic about finishing all tasks on schedule.

6. **Redundancy in Safety:** Having redundant safety features is important to protect both people and the robot. Although the team had a remote emergency stop and software stop commands, when the robot throttle was commanded at full capacity, the team members were not able to react fast enough. When the software had unexpected communication errors, the robot took a sharp turn and crashed, resulting in a number of hardware components being damaged. This resulted in the team adding in map boundaries, additional stop conditions, and checks on sensors to prevent another accident from occurring.

9.2 Future Work

For future work, a few sets of items have been identified as the key areas that would provide the most value to focus on. They are outlined below:

1. The first set of tasks would relate to completing the remaining portions of the system that had their development cut short from the original development scope. This would entail getting map synthesis operational, fully integrated, and tested with planning. After thorough testing, such as driving in loops and adding puddles, and evaluating performance, the baseline functionality for the system of puddle navigation and avoidance would be complete.

2. Since autonomous vehicle companies will already have planning and controls subsystems, the venue for the ADAPT system to integrate with theirs would be through providing their planning subsystem with information focused on road conditions. This means focusing attention on improving the perception and mapping subsystems. Specific tasks would be to reduce the computation time of these processes and increase their accuracy to the vehicle. This last component includes aspects such as converting the puddle map into an occupancy grid of limitations in speed and lateral acceleration. This would then be usable by a general planner.

3. With the above progress made in software, the next would steps would be to scale up the implementation to passenger vehicles and semi-trucks in order to be able to test and demonstrate the system's capabilities on the types of vehicles that potential customers would be developing. It would also be necessary to move in the direction of being hardware agnostic, as different customers would likely use a variety of hardware suites, and showing that the system could be easily ported to their hardware would be critical.

4. Finally, aside from general product improvements, the system should cover more road conditions than simply wet and dry. It would ideally incorporate hazards such as pot holes and other weather related conditions such as snow, ice, frost heaves, sleet, hail, and slush. In addition to these common examples, there can be other dangerous scenarios that come from other adverse road conditions, such as the presence of oil, sand, or wet leaves on the road. Accurately covering all of these conditions, in addition to dry, wet, and puddle, would make the system applicable in a wide range of hazardous road scenarios and provide the best chance at reducing the number of accidents due to adverse conditions.

References

[1] "How do weather events impact roads?" [Online]. Available: https://ops.fhwa.dot.gov/weather/q1_roadimpact.htm

[2] S. Kilcarr, "Mitigating the weather's impact on trucking," November 2016. [Online]. Available: www.fleetowner.com/blog/mitigating-weather-s-impact-trucking

[3] "Camera cars." [Online]. Available: https://www.kingmotorrc.com/camera-cars/

[4] "Cine rc 4 4 all-wheel drive gimbal car." [Online]. Available: cinegears.com/product/cine-gears-all-wheel-drive-gimbal-car/

[5] "Autorally - platform specs." [Online]. Available: autorally.github.io/specs/

[6] "Kraken rc." [Online]. Available: shop.krakenrc.com/

[7] "Golf carts for sale." [Online]. Available: www.ebay.com/b/Golf-Carts/181476/bn_16581809

[8] "Traxxas." [Online]. Available: traxxas.com/products/landing/x-maxx/

[9] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: real-time instance segmentation," *CoRR*, vol. abs/1904.02689, 2019. [Online]. Available: http://arxiv.org/abs/1904.02689

[10] P. Chao, C. Kao, Y. Ruan, C. Huang, and Y. Lin, "Hardnet: A low memory traffic network," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 3551–3560.

[11] C. Nguyen, M. Milford, and R. Mahony, "3d tracking of water hazards with polarized stereo cameras," pp. 5251–5257, 01 2017.

[12] M. Pivtoraiko, "Differentially constrained motion planning with state lattice motion primitives," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, February 2012.

[13] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American Control Conference*, 2007, pp. 2296–2301.