Milestone 4 - Design Specification

Project : Propbot Autonomous Robot

Authors : Team- 50,

Amr Almoallim (AA) - 83386714,

Apoorv Garg (AG) - 39485545,

Cole Shanks (CS) - 54950860,

Johanan Agarwal (JA) - 29188166,

Sajjad Al-Kazzaz (SA) - 23401565

Affiliation : UBC Radio Science Lab

Table of Contents

1. Introduction **4**

2. System Architecture **4**

2.1 System Overview 4

2.2 Modes of Operation 8

2.3 Autonomy System 9

2.4 Vehicle Interface System 11

2.5 Mission Command Centre 14

2.6 Power Management 16

3. Vehicle Interface System - Component Level **18**

3.1 Overview 18

3.2 Vehicle Interface Board 18

3.3 Interfacing with Autonomy System 19

3.4 Sonar Sensors 20

3.5 Motor Controllers 21

3.6 CAN Adaptor 24

3.7 Teleoperation Controller 25

4. Autonomy System - Component Level **28**

4.1 Autonomy Package 28

4.2 Autonomy Computer 29

4.3 IMU - Inertial Measurement Unit 32

4.4 GPS - Global Positioning System 33

4.5 LiDAR 33

4.6 Cellular Modem 33

5. Mission Command Centre - Component Level **35**

5.1 Data Collection 35

5.2 Log Viewer 36

6. Power Management System **37**

6.1 Battery Selection 37

6.2 Power Distribution 40

7. Trade Studies **44**

7.1 Autonomy Computer 44

7.2 Cellular Modem 48

7.3 Battery and Power Consumption 50

8. References **52**

A. Appendix **55**

1. Introduction

This design document offers a description and justification of the key systems and components of Propbot. In our systems level view, we show the functionality of each subsystem with respect to the requirements they tackle, and how they fit into the process of a normal use case of Propbot. Within each system, we look at subsystems, referred to as either modules or packages, that interact with each other in a defined way to achieve a common goal.

In our component level view, we look at the physical devices, sensors, and communication protocols that make up each subsystem. When applicable, we present trade studies to compare the different options that were available to us.

This document is structured so as to first look into the systems level view (Section 2), where we look at Propbot's four major systems: the Vehicle Interface System (VIS), the Autonomy System (AS), Mission Command Centre (MCC), and the Power Management System (PMS), and their related subsystems. Following this, we dedicate four Sections to a component level look of each subsystem (Sections 3, 4, 5, and 6). Note that for the component level sections, we only discuss components that are relevant to our scope.

2. System Architecture

2.1 System Overview

Propbot consists of four systems that are relevant to our scope: the Vehicle Interface System (VIS), the Autonomy System (AS), Mission Command Centre (MCC), and the Power Management System (PMS). The PMS is unlike the other systems in the sense that it operates at the electrical level, and does not communicate with the other three systems, which we will refer to as the primary systems. Because of this, we consider the PMS separately in Section 2.6.

The following figure 2.1 represents a high-level description of the three primary systems. The Mission Command Centre is the user-facing system that retrieves missions, defined as a series of waypoints, which the user wants Propbot to navigate to one at a time. It is hosted on the user's laptop/desktop. The Autonomy System, mounted in Propbot's custom server rack (figure 2.2), is responsible for making decisions on where to move based on the mission it received and Propbots's surroundings. Finally, the Vehicle Interface System, mounted in Propbot's middle cavity, translates those decisions into PWM speed signals that allow Propbots motors to move in the desired direction, while also providing safety and fallback measures.

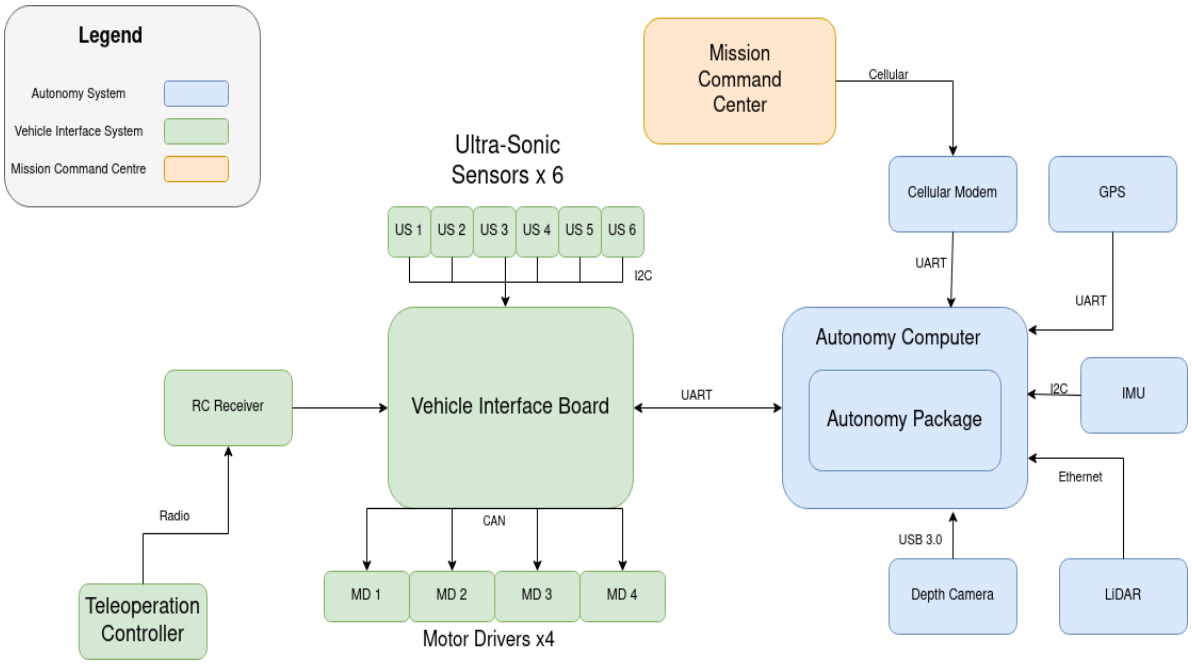

Figure 2.1 Systems and Communication Diagram

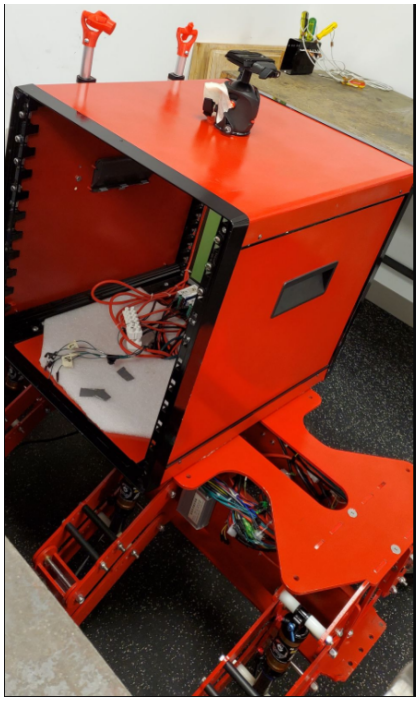

Figure 2.2 Propbot Image

The inputs and outputs to and from each system are outlined in Table 2.1 and Table 2.2, followed by a brief description of custom signals used in Propbot (Table 2.3).

	Input	Input From
Autonomy System	Depth data	Depth Camera, LiDAR
	Vision	Depth Camera, LiDAR
	Global Positioning	Global Positioning System (GPS)
	Velocity and Incline	Inertial Measurement Unit (IMU)
	Mission Command	MCC
Vehicle Interface System	Wheel speed	Remote Control, AS
	Estop Command	Remote Control
	Sonar data	Sonar Sensor
Mission Command Centre	Mission Command	User (via GUI)
	Position Data	AS

Table 2.1 : Inputs to each system and where they are communicated from

	Output	Output to
Autonomy System	Wheel speed	VIS
	Position Data	MCC
Vehicle Interface System	LHS PWM signal	LHS motor controller
	RHS PWM signal	RHS motor controller
Mission Command Centre	Mission Command	AS
	Position Data	User (via GUI)
	Mission Status	User (via GUI)

Table 2.2 : Outputs from each system and where the output is communicated to

Signal	Description
Wheel speed	A message that contains desired speeds for left/right hand side (LHS/RHS) wheels
E-Stop Command	A message that requests Propbot to stop (Brakes applied)
Mission Command	One of: ● "Start Mission" : A command to start the mission with a given set of waypoints ● "Pause Mission" : A command to pause the mission in progress ● "Stop Mission" : A command to stop the mission in progress ● "Resume Mission" : A command to resume a paused mission, or (in the case where Propbot pauses at a desired waypoint) continue to the next waypoint
Mission Status	Status of the current mission and whether it is in progress, paused, stopped, or finished.

Table 2.3 : Descriptions of custom signals communicated throughout Propbot

The Autonomy system and the Vehicle Interface System could be combined into one system. Our justification for separating them is as follows:

- It provides a clear separation of the systems needed to achieve different levels of functionalities out of Propbot. With only VIS, Propbot can still be used to conduct RSL research via teleoperation.

- It provides a clear separation of concerns: AS computes higher-level functions of autonomy whereas VIS implements the decision
- The autonomy computer will conduct computationally intensive autonomy algorithms. Adding more burden to it will likely increase the latency of the entire system, leading to an increased risk of collisions.

2.2 Modes of Operation

In order to facilitate safety requirements F3.1, F3.2, F3.4, we define Propbot to have three states: Autonomy mode, Remote Control (RC) mode, and E-Stop mode. Each mode is characterised by the kind of velocity command VIS transmits to the motors.

The flow of velocity commands through Propbot is represented in figure 2.3. In Autonomy mode, VIS transmits velocity commands based on the Autonomy System's outputted left/right wheel speeds. In RC mode the Autonomous System's output is ignored, and wheel speeds received via a remote controller are used instead. In E-Stop mode, both kinds of inputs are ignored, brakes are engaged, and Propbot comes to a complete stop. E-stop mode can be triggered either through a safety override in VIS resulting from sonar data showing an imminent obstacle collision or through a manual e-stop command received from a fallback user's remote controller. Changes in the mode of operation occur as follows according to the finite state machine diagram in figure 2.4.

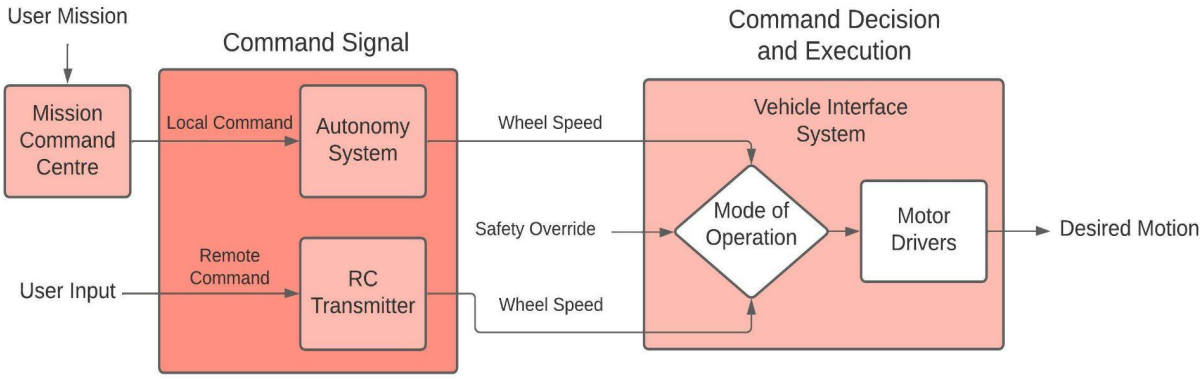

Figure 2.3 System Command Diagram

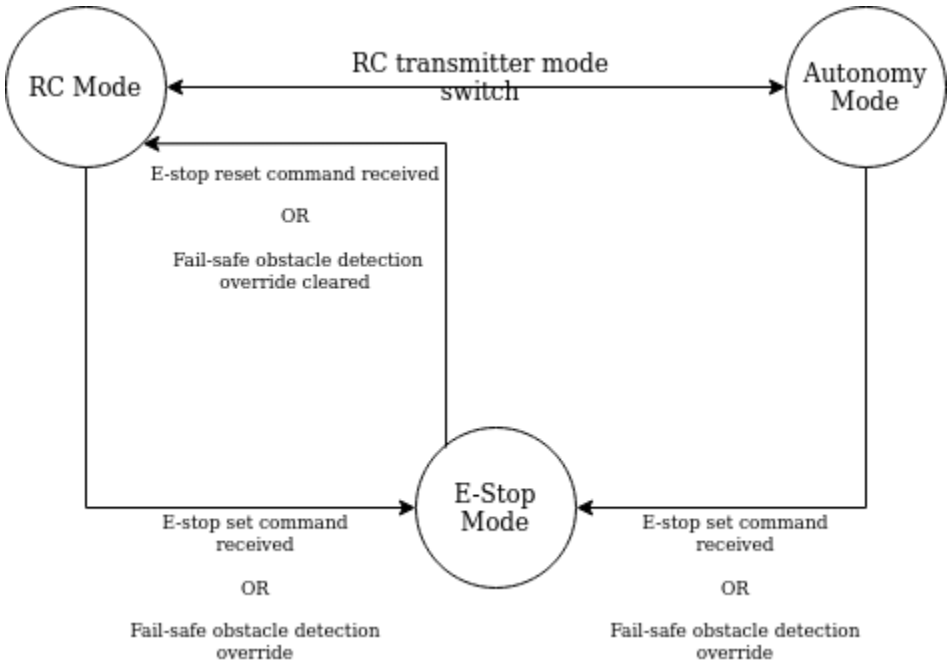

Figure 2.4 Modes Of Operation

2.3 Autonomy System

The main goal of the system is to safely guide Propbot towards its intended destination(s) with minimal aid from a fallback user. At a high level, its modules (figure 2.5) are described as follows:

Mission Command: Mission Command receives missions from the MCC and sets the target waypoint for the Motion Planning module. When the waypoint has been reached, Mission Command sets the next waypoint, if it exists, as the new target. It also receives other commands from MCC allowing Propbot to pause, resume, or stop missions.

Inputs: Mission Command (from MCC)

Outputs: Mission (to Motion Planning)

Localization: Refers to the use of sensor data to determine Propbot's location with respect to its environment. This is communicated through a 2D occupancy grid. For example, in

figure 2.5, in the localization block, the red dot represents the robot location, white represents free space, black represents obstacles, and grey represents unexplored areas.

Inputs: LiDAR, GPS, IMU (from sensors)
Outputs: Occupancy Grid (to Motion Planning)

Motion Planning: Refers to the robot generating a path that moves the robot from its starting position to a goal without colliding with any obstacles. Motion planning takes in obstacle information and location data to generate a feasible trajectory, which is converted to a velocity command.

Inputs: LiDAR, depth camera (from sensors), Occupancy Grid (from Localization), Mission (from Mission Command)
Outputs: Velocity Command (to Robot Control)

Robot Control: Refers to the conversion of a general velocity command, as output by motion planning, into specific wheel speeds based on the properties of the robot and its differential drive system.

Inputs: Velocity Command (From Motion Planning)
Outputs: Wheel Speeds

Table 2.4 shows requirements directly related to each module.

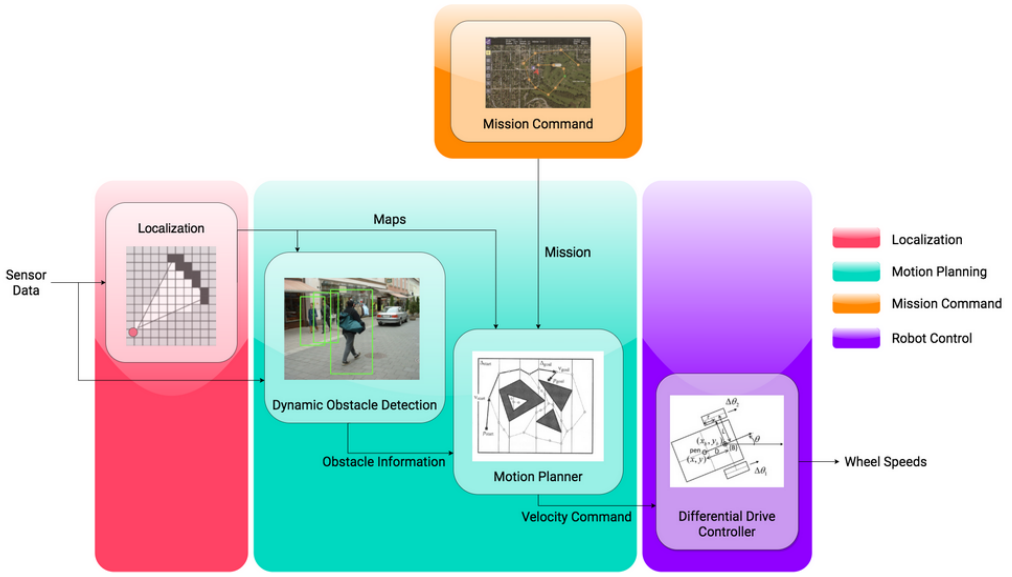

Figure 2.5 : Modularized System Diagram of Autonomy System [1]

	Navigation	Decision Making	Safety	Communication	Data Collection
Mission Command	F1.5, F1.6	F2.5	F3.5	F4.2, F4.3	
				NF4.2	
Localization	F1.4	F2.6			F5.1
		NF2.3			NF5.1
Motion Planning	F1.4	F2.2			
	NF1.2	NF2.1			
Robot Control	F1.3				
	NF1.5, NF1.6				

Table 2.4 : Requirements per Autonomy System Module

2.4 Vehicle Interface System

The main goal for VIS is the transmission of wheel speeds from the autonomy system, RC controller, and E-Stop depending on the mode of operation of Propbot. These speeds are translated into PWM signals that are sent to the motor controllers to drive Propbot. The vehicle

interface system is also responsible for collision avoidance via input from short-range sonar sensors. It consists of the following modules, which are also represented in figure 2.6:

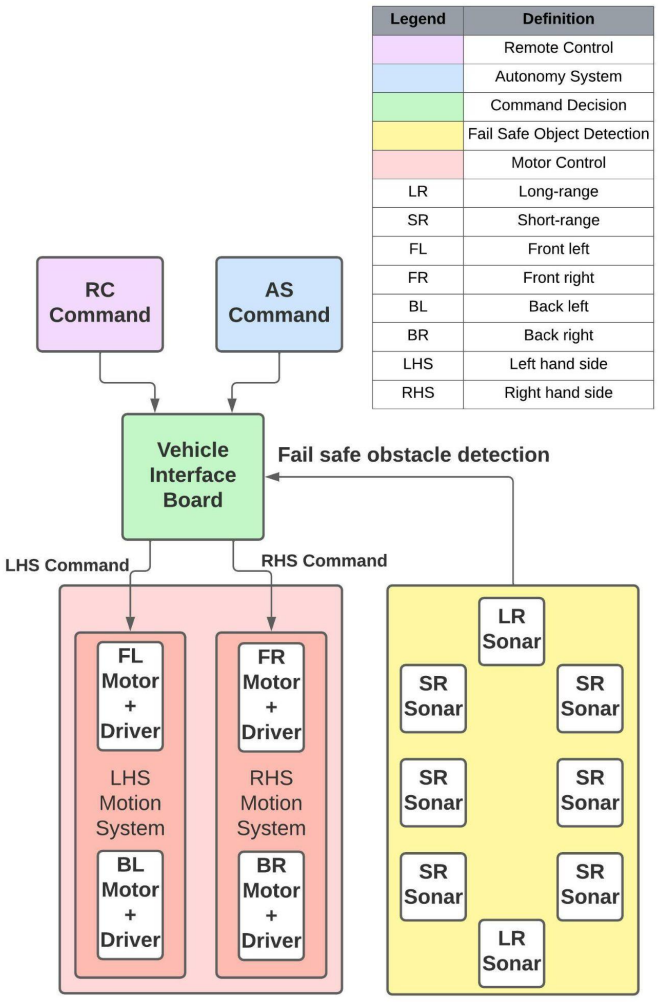

Figure 2.6 High Level Overview of VIS

Remote control (Transmitter/Receiver): This is how the user is able to control Propbot. Commands pressed on the controller are sent to the paired wireless receiver. The receiver is connected to a vehicle interface board/computer which sends the appropriate control signals to the motor drivers. Left, Right, E-Stop, and change mode commands are available on the controller.

Inputs: Wheel Speeds, E-stop command (from the remote controller)

Output: Wheel Speeds, E-stop command (to command decision)

Fail-Safe Obstacle Detection: This part of the vehicle interface system is used to detect close-by (within 50 cms) obstacles to avoid any collision. It is responsible for detecting if Propbot is at risk of colliding with an object.

Inputs: Sonar Data (from sonar sensors)

Output: Fail-safe override (to command decision)

Motor control: This layer is responsible for sending wheel speed and direction commands to the motor controller over a CAN network.

Inputs: wheel speed (from command decision)

Outputs: PWM signal (to motor controllers)

Communication Link: This subsystem is used to establish communication between the vehicle interface system and the autonomy system. It takes wheel speed inputs from the autonomy system so that they can be passed on to the motor control module described above.

Inputs: wheel speed (from Autonomy System)

Outputs: wheel speed (to command decision), status (to Autonomy system)

Command Decision: Brings together wheel speeds and sonar detections from different subsystems and decides what to output to the motors.

Inputs: Wheel Speeds (from Remote Control, Communication Link), Fail-safe override (from Fail-Safe Obstacle Detection)

Outputs: wheel speed (to Motor Control)

Table 2.5 shows requirements directly related to each module.

	Navigation	Decision Making	Safety	Communication	Data Collection
Remote Control	F1.1		F3.1, F3.2, F3.4, F3.6	F4.1	

Fail-Safe Obstacle Detection	F1.1, F1.4				
Motor Control	F1.1, F1.2, F1.3, F1.9				F5.2
	NF1.5, NF1.6, NF1.9		NF3.3, NF3.4		
Communication Link	F1.1		F3.5		
				NF4.2	
Command Decision	F1.1				

Table 2.5 : Requirements per VIS module

2.5 Mission Command Centre

The Mission Command Centre is the user-facing part of Propbot. It captures user mission commands and communicates them to the Autonomy System. It also receives information from the Autonomy System, which it communicates to the user either on a graphical user interface (GUI) , a log console, or by saving to a file.

The design of the Propbot GUI package is shown in Figure 2.8. Its components are described below:

Mapviz: ROS-based visualization tool that Powers the visualization portion of the GUI. Mapviz allows customizable plugins to be made to interact with the visualizer. For Propbot, the plugins used with Mapviz are (Figure 2.7):

- **Plan Mission Plugin:** This allows the user to set waypoints on the GUI and upload them to the robot.
- **Tile Map Plugin:** This is an existing plugin package that obtains map tiles from Bing Maps, WMTS Tile service, or Stamen Design and displays them on the GUI.
- **2D View:** This is the main GUI display.
- **NavSat Plugin**: Listens for the latest GPS location of Propbot and draws an identifiable shape on top of the GUI.

- **Custom Plugin:** A blueprint plugin that serves as an example for developers wanting to write their own plugins.

Data Handling: Responsible for highlighting logged data in realtime and saving ROS topics to files for later viewing. This component runs independently from Mapviz.

The following Table shows how the MCC is relevant to our project scope by listing requirements related to every component.

	Navigation	Decision Making	Safety	Communication	Data Collection
Plan Mission Plugin				F4.2, F4.4	
Tile Map Plugin					F4.3
2D view			F3.5	F4.3	
NavSat Plugin				F4.3	
Data Handling			F3.5	F4.3, F4.5	F5.1, F5.2, F5.3, F5.4

Table 2.6 : Requirements per MCC module

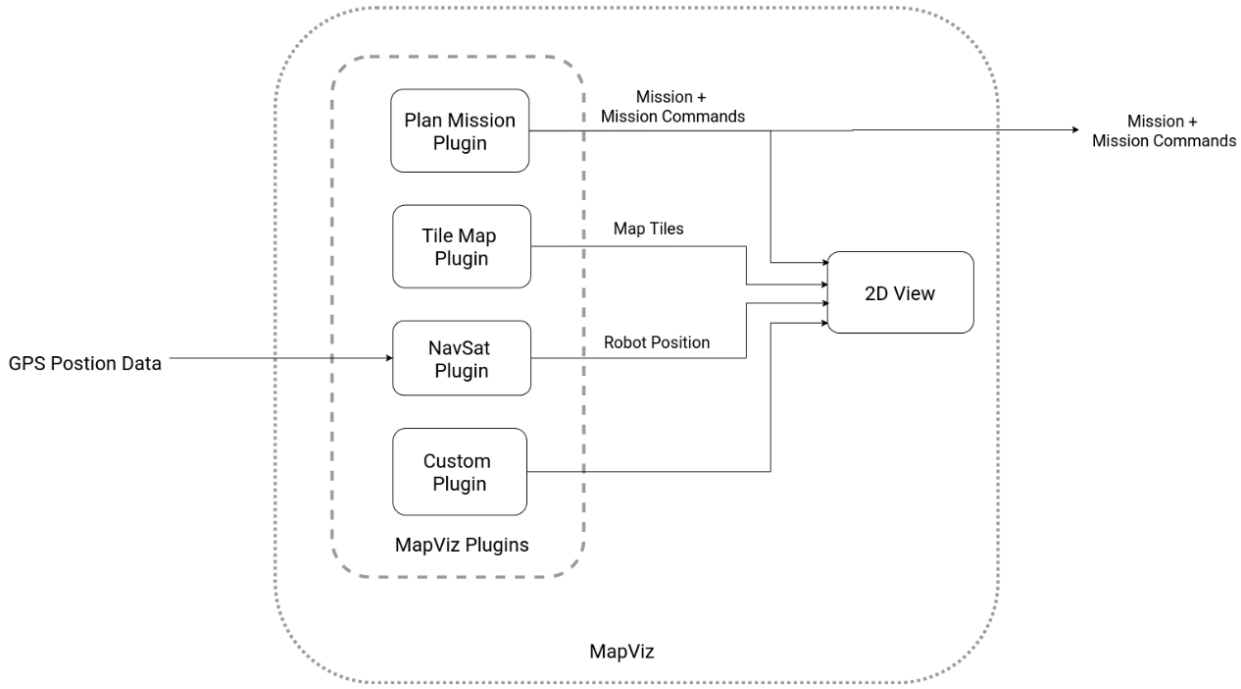

Figure 2.7 : Mapviz plugin architecture [1]

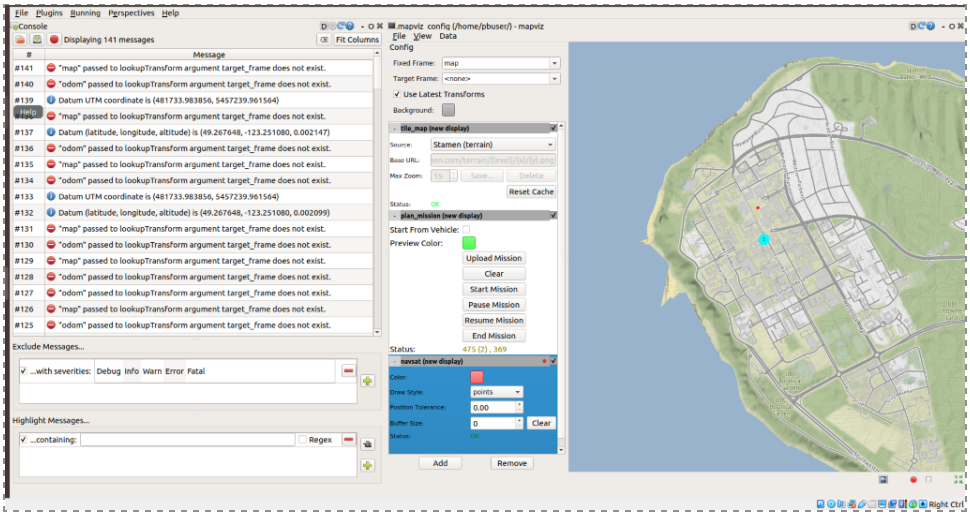

Figure 2.8: MCC GUI. Logs coming from the system are viewed on the left panel. On the right is Mapviz, where mission uploading and handling can be done.

2.6 Power Management

The purpose of this system is to power all energy consuming components that pertain to motion and autonomy. Therefore, this system is responsible for delivering power to the VIS and

the AS. According to our requirements (NF1.4 and NF2.4), all relevant components will receive their rated power. The power management system must also be capable of supplying this power for a minimum of 1.5hrs as outlined in the requirements (F1.7 and F2.7), which our chosen batteries are capable of handling (figure 5.3). Furthermore, in order to ensure safe operation the power management system is equipped with an easily accessible on-board physical E-stop device which is also in accordance with our requirements (F3.3, F3.4, and NF3.2). At a high level the main components that make up this system are described below.

Batteries: This is the main component responsible for storing electric charge and outputting it at the necessary voltage and current.

Power Distributors: The purpose of these distributors is to either unify multiple power inputs (batteries) into the main rail, or to split a single power input into multiple lines for distribution to different components.

Voltage Converters: Various components in our system do not use the high voltage rating set by the batteries, and thus require step-down voltage converters to provide an accepTable voltage to these components.

E-Stop: This is a physical switch that is in an accessible location and chosen to be easy to trigger so that power to the VIS can be cut locally.

Cabling: Cables are responsible for carrying power between the various components and are chosen to be able to handle the voltage and current that will be run between each connection.

Cable Sheathing: Metal cable sheathing is added on cabling carrying I2C signals since they are very sensitive to electro-magnetic interference from power lines and the sheathing will isolate them and provide reliable communication. This originates from the 3mA current limit for I2C communication lines which would experience interference induced by the power lines.

3. Vehicle Interface System - Component Level

3.1 Overview

The vehicle interface system includes the following components as shown in figure 3.1 below:

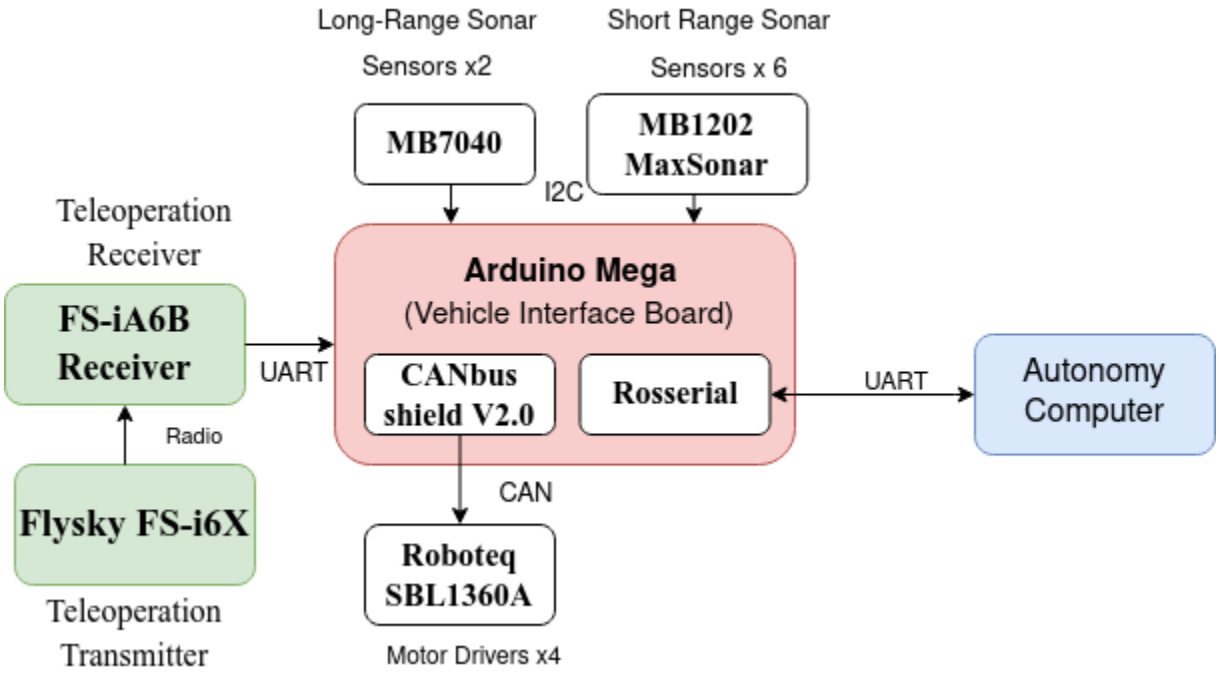

Figure 3.1 Overview of Vehicle Interface System

3.2 Vehicle Interface Board

Legacy Component: Arduino Uno

Final choice: Arduino Mega 2560 [1]

Criteria:

The vehicle interface board needs to interface with Motor Drivers, RC, range sensors, and an autonomous computer**.** It should support timers and interrupt. The required features are mentioned in Table 3.1.

Selection Justification:

The Arduino Mega provides 15 PWM outputs and 4 serial channels [7]. Arduino devices support configurable interrupts and timers and have extensive documentation online. It is the same family as previous boards which were used to control the motors (Arduino UNO), so the code is easy to port.

Related Tag(s)	Feature required	Justification
F1.3	The board should support outputting PWM signals	The primary way of controlling motor drivers
NF1.3	Should contain at least 2 serial channels	One for motor drivers (CAN), one for autonomy computer

Table 3.1 Vehicle Interface Board Required Features

3.3 Interfacing with Autonomy System

Legacy component: None

Final choice: ROSserial

Criteria:

Must be able to communicate data from the Arduino to the Autonomy board and vice versa.

Selection Justification:

ROSserial is a ROS package that was created for the purpose of adding a device to the ROS network via a UART serial connection. Being part of the ROS network means the vehicle interface board can subscribe to topics that were published by the autonomy system. Similarly, it can publish topics that the autonomy system can use to retrieve data from. Rosserial is supported both as an Arduino library and on ROS Melodic, the ROS version of the autonomy computer.

Related Tag(s)	Feature required	Justification
F1.3	Be able to retrieve autonomy computer output	The primary way of controlling motor drivers in autonomy mode
F5.2, F5.3	Be able to send autonomy computer information	To communicate motor speed, RC commands, and state

Table 3.2 Autonomy System Interface Required Features

3.4 Sonar Sensors

Legacy component: 1 x MB1202 MaxSonar, 1 x MB7040 [A.2]

Final choice: 6 x MB1202 MaxSonar, 2 x MB7040

Criteria:

To provide a fail-safe obstacle detection and avoid a collision, we need to integrate some close-range sensors. We need these sensors to help us detect obstacles 0.5m away.

Selection Justification:

These sensors have been procured and are a constraint to this project. Therefore it will be our responsibility to incorporate these close-range sensors in our proposed design. The trade study is added in the appendix which was created by the previous team [A.2]. The Matbotix MB1202 and MB7040 had the best sampling rate according to the previous team due to which these sensors where selected.

Related Tag(s)	Feature required
C18	The sonar sensors have already been scoped and purchased by the previous team
NF2.1	The sonar sensors must be able to detect an obstacle at a minimum distance of 0.5m

Table 3.3 Power Sonar Sensors Required Features

3.5 Motor Controllers

Legacy component: 1 x Roboteq SBL1360A

Final choice: 4 x Roboteq SBL1360A [8]

Criteria:

To enable safe, reliable, and accurate speed control of up to 4 motors on a connecting bus. Motors controllers should provide electronic braking, perform closed loop feedback via input from hall sensors, and have various fail-safe mechanisms in place.

Legacy:

The motor controllers that came previously installed on Propbot are generic E-bike controllers. These controllers do not satisfy the ability to control speed accurately when driven autonomously due to lack of closed-loop feedback. Additionally documentation is not available for these motor controllers and therefore its safety features cannot be validated. With that being said, basic functionality including speed, direction, and braking can be performed with these controllers and they are being used temporarily due to unavailability of the Roboteq motor controllers as a result of supply-chain issues.

Selection Justification:

Selection criteria for the Roboteq SBL1360A corresponds with design recommendation from the previous team (see Table A.1 for trade study).

Additional considerations into the selection of the Roboteq SBL1360A controllers wherewas safety, which is of utmost importance when operating an autonomous vehicle the size of Propbot. Safety features on the Roboteq controllers include under/over current and voltage protection, hardware fault protection, and command watchdog - all of which ensure safe and reliable control of the motors. Hardware fault protection is a useful feature that enables safe operation of the motor controllers in the case of power stage failure.

In addition to excellent safety features, the Roboteq SBL1360A enables closed-loop speed control using feedback from external hall sensors which is an essential feature for ensuring that the motors can accurately control speed of Propbot during autonomous operation.

Additionally, Roboteq motor controllers support electronic braking which is a crucial safety requirement for Propbot given that mechanical braking cannot be easily implemented with the current motors and is out of the scope of this project..

Another factor for the choice of Roboteq SBL1360A is its support for CANbus communication which is an industry standard communications bus that is reliable, scalable, and highly robust to external noise which is a major concern for Propbot given the close proximity of the communications bus and high voltage motors. See section A.8 for a more detailed justification for the selection of CANbus as the preferred choice of communication protocol.

Related Tag(s)	Feature required
F1.1, F1.2	The robot shall be able to move longitudinally, and be able to accelerate, decelerate, and stop
NF1.7	The robot's movement shall be fluid

Table 3.4 Motor Controllers Required Features

Implementation Design Considerations:

The datasheet for the Roboteq SBL1360A controller outlines safety considerations, implementation details, and general design guidelines for setting up the motor controllers, which are briefly referenced in this Section.

E-Stop

These controllers support a safe and reliable way to cut power to them by the user using an E-Stop which can be wired in series with the positive terminal of the battery and *Vmo*t of the controller as shown in figure 3.2. A precharge resistor is to be placed in parallel with the E-Stop to prevent arcing from occurring when E-stop is triggered. Inline fuses are placed in series with the positive battery terminal to prevent damage in the case of a high current. Relay diodes are placed in parallel with these fuses to provide a return path for current to flow to the battery in the case that a fuse is blown. Fuses, resistors, and diodes have been selected according to datasheet recommendations. The selection of these components are referenced in Section A.9.

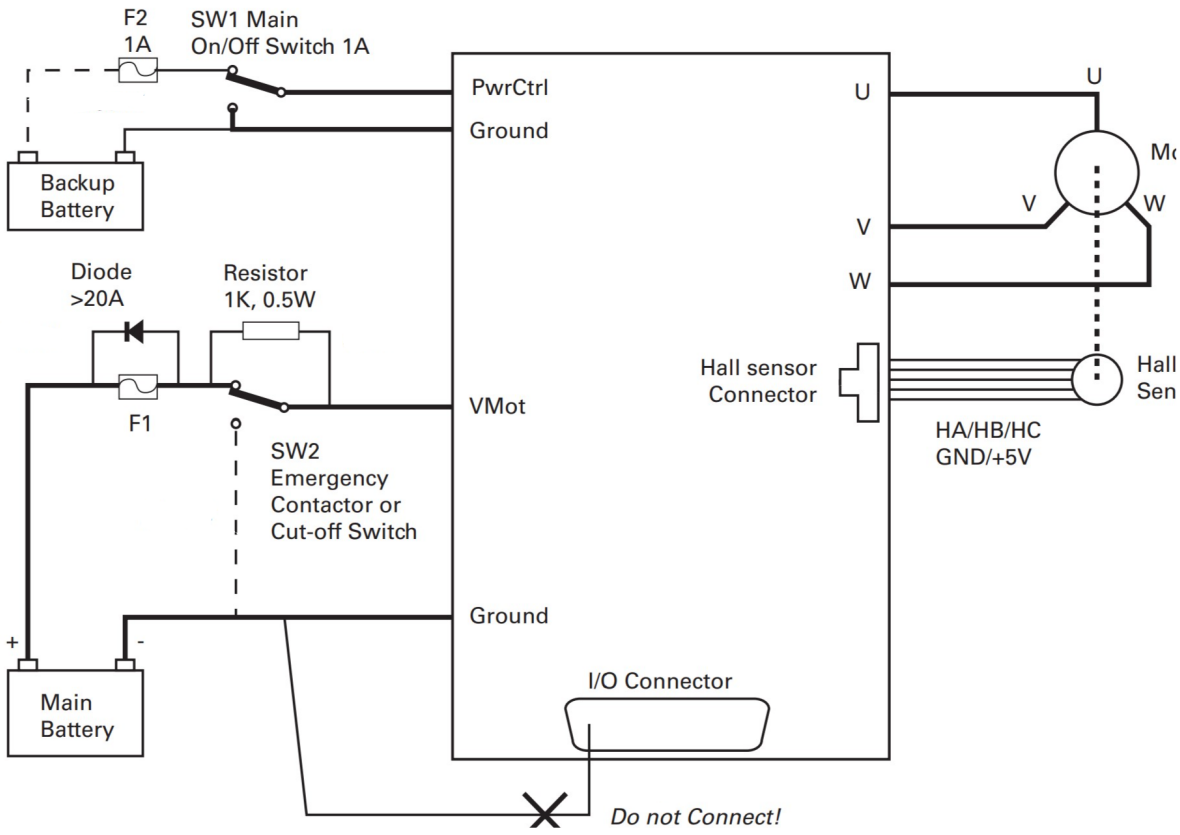

Figure 3.2 Roboteq SBL1360A Wiring Diagram

Backup Battery:

Roboteq controllers can be supplied with either a single power supply or a single plus a backup power supply. When a backup battery is used, power from it is only delivered to the motor controllers when the main battery reaches a threshold of 7V. The purpose of the backup battery is to provide enough power sufficient to power communication functionality of the controllers but not enough to power the motors.While Propbot is designed only to operate within its power delivery time limit, Roboteq controllers will naturally come to a stop when its voltage is below a certain threshold for safety purposes. However, implementation of a battery fuel gauge which can send real time state of charge information from Propbot to the mission command centre is certainly a useful feature that can be considered by the future team.

EMI interference:

Electromagnetic interference considerations are not a concern for Roboteq controllers on Propbot given the relatively small length of wiring that is used to interface with the controller.

Heat Dissipation and Mounting:

The Roboteq controllers are designed to be mounted on a metallic surface to dissipate heat. The controllers will be mounted to the metallic chassis of Propbot in order to dissipate heat.

3.6 CAN Adaptor

Legacy Component: None

Final Choice: Seeed Studio CANbus shield V2.0 [9]

Criteria:

To enable CAN communication between the vehicle interface computer (Arduino Mega 2560) and motor controllers we require a CAN adaptor. This is because the Arduino Mega does not contain a CAN transceiver/controller. The main criteria for the CAN adaptor include compatibility with Arduino Mega 2560, simple ‘plug and play’ hardware, and software implementation.

Selection Justification:

The Seeed Studio CANbus shield V2.0 is the most established and widely available CAN adaptor for Arduino devices, and is compatible with Propbot's vehicle interface computer, the Arduino Mega 2560. The software library is simple, robust, one of the default libraries present in the Arduino IDE. The hardware comprises the popular Microchip MCP2551 CAN Controller and MCP2515 CAN transceiver.

3.7 Teleoperation Controller

The Transmitter/Receiver pair is part of the vehicle interface system and serves as the primary mode of communication between the operator and Probot when issuing motion commands to the robot.

Legacy Component: FS-T6

Previous teams used an FS-T6 to control the robot remotely. Our validation tests indicated that this device was malfunctioning, and so a new device was procured.

Final Choice: Flysky FS-i6X 6CH 2.4GHz AFHDS RC Transmitter w/ FS-iA6B Receiver [10]

Criteria:

To allow for teleoperable control of Propbot, we need an RC Transmitter and Receiver pair that can satisfy our design requirements. According to requirement F1.1, the robot must be able to accelerate, decelerate, and stop when in manual control mode. Additionally, requirement F1.2 states that we are able to control the robot to move longitudinally and F1.3 says that we can control the robot's speed and safely execute turns. As a safety feature in addition to the E-Stop button connected directly to the battery, the transmitter should also be capable of sending a command to Propbot that shuts off system power in the event of an emergency.

Selection Justification:

The Flysky FS-i6X Transmitter and FS-iA6B Receiver pair satisfy our design requirements. They are very similar to the previous capstone team's design for the RC car and therefore the code can be easily transferred to work with these devices. The controller must have at least 4 available channels to execute the following functions

- Move the left-side wheels forward/backward
- Move the right-side wheels forward/backward
- Remote E-stop assertion mechanism (switch/button)
- Remote switch between manual and autonomous control modes

Related Tag(s)	Feature required
F1.2	Propbot shall facilitate longitudinal motor control of each set of wheels via a remote controller
F3.4	There should be a switch on the RC controller to send an E-stop command to Propbot when in manual control
F3.2	There should be a switch on the RC controller that changes from autonomous to manual control when asserted
F4.1	The controller must have a range up to a distance of 1km

Table 3.5 Teleoperation Controller Required Features

Figure 3.3 below illustrates how the four functions mentioned above are distributed on the transmitter:

Figure 3.3 Teleoperation Controller Functionality

4. Autonomy System - Component Level

The autonomy system includes the following components as shown in figure 4.1 below.

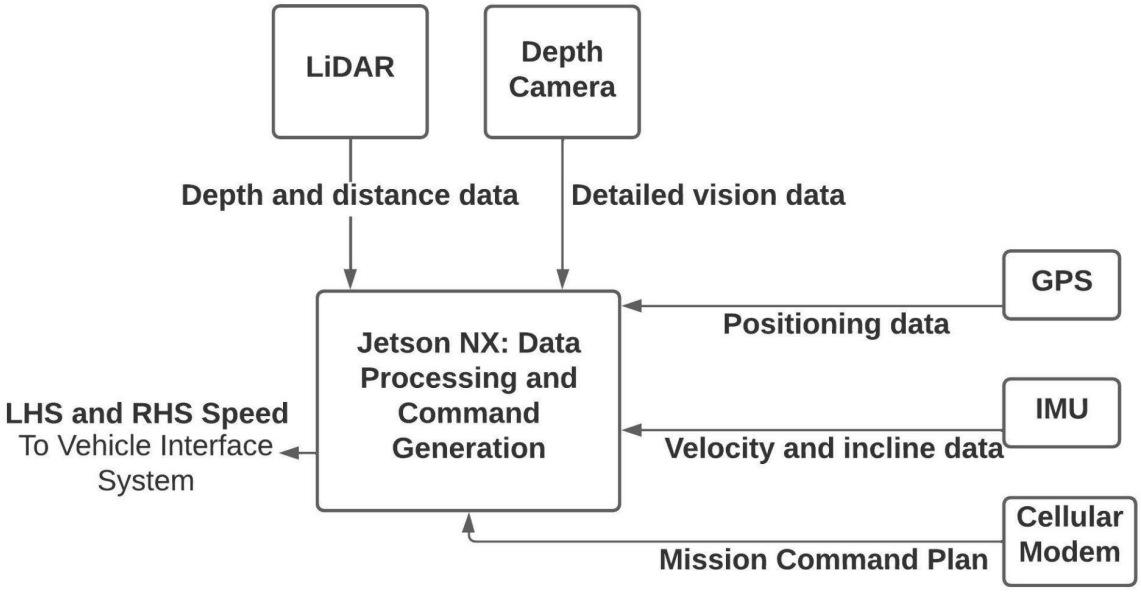

Figure 4.1 Component level view of Autonomy System

4.1 Autonomy Package

Legacy Component: Autonomy Package 1.0

There exists a standalone autonomy package, responsible for managing the modules of the autonomy system. It is developed using the ROS framework in C++. It is run on the autonomy computer, which is an NVIDIA Jetson TX1. The autonomy package is not integrated with any other component of Propbot.

For the purposes of defining our scope and design, the legacy autonomy package can be thought of as a black box, with the sensor data from LiDAR, camera, GPS, and IMU serving as input, and left side / right side wheel speeds given as output. Details about the components that make up the autonomy package can be found in the appendix.

Final Choice: Autonomy Package 2.0

Criteria: Requirements from Table 2.4

Selection Justification:

Through simulation, we have validated that the inherited autonomy package satisfies almost all requirements mentioned in Table 2.4 from the systems level. Packages will be added to accommodate the host of sensors that are being added to the Autonomy system, but the structure, dependencies, and practices used in the package will remain the same.

4.2 Autonomy Computer

Legacy Component: NVIDIA Jetson TX1

Currently, the autonomy computer used is the NVIDIA Jetson TX1. The team previously working on Propbot has expressed the challenges they faced in getting the TX1 to run all of their autonomy software. The challenges are consistent with the challenges of people using the TX1 found online [11], and were also experienced in our validation of the Autonomy Package. In our validation, several autonomy package components complained about insufficient update rate, and the simulation ran at one-tenth speed. A breakdown of the current ROS packages that are taking up the most CPU resources (conducted by the previous team):

- Move_base (navigation package) (70% total on 2 cores)
- Visual Odometry (65% total on 2 cores)
- Darknet YOLO (dynamic obstacle detection) (60% on 1 core)
- Google Cartographer (SLAM localization) (40% total on 2 cores)
- Pointcloud to Laserscan conversions (36% on 2 core)

Taking this information into account, we have decided to research and purchase a new Autonomy Computer and port over the previous team's autonomy package to run on this new computer.

Final choice: NVIDIA Jetson NX

Criteria:

Our overarching goal is to run the autonomy package at the frequency described in requirements F5.1, F5.2, and NF2.2. Due to the uniqueness of Propbot's software stack and the lack of available benchmarking data for individual components on different boards, it is hard to quantify exactly how much computing power is needed for the autonomy computer, making it hard to guarantee that simply improving the autonomy computer will solve our latency issues.

As a result, our objective becomes to ensure that the autonomy computer is not a performance bottleneck when running the autonomy package. It's cheaper to improve software, and improvements can be made across the entire stack, whereas improving hardware, especially Jetson devices, are expensive. To achieve this, we looked at current projects that make use of each device (Table 4.1) we considered in our trade study (Section 5.1), with the idea being that if enough evidence could be found that projects similar to ours used a certain device without complication, then we can be certain that the device will not be a bottleneck to our project.

Project	Project Description	Autonomy Computer Used	Similar Components	Additional Components	Comments
Chuck [12]	Autonomous vehicle for warehouse management	TX2	ROS framework, LiDAR, planning, control	8 sonar sensors	Initially tried setup on TX1 and failed, so upgraded to TX2
Notilo Plus [13]	Underwater explorer robot	TX2	Camera		
MEITUAN [14]	Autonomous delivery robot	AGX	LiDAR, planning, control		
Robottle [15]	Autonomous navigation robot	NX	Depth Camera, planning, control		
Agrobot [16]	Autonomous crop traversing vehicle	NX	ROS, Depth Camera , planning, control		

F1 Tenth [17]	Small-scale Autonomous racing vehicle	NX	ROS, LiDAR, planning, control		

Table 4.1 : Examples of similar projects and the Autonomy Computer used

Selection Justification:

In our trade study, we considered 3 options: the NVIDIA Jetson TX2, NVIDIA Jetson NX, NVIDIA Jetson AGX. Table 4.1 shows that any of these devices are capable of achieving the performance requirements of Propbot.

Overall the NX offers the best balance betw+een price and performance. From the trade study, we note that it has better AI performance and more CPU/GPU cores than the TX2, while costing the same. Having more CPU cores means future teams could explore the option of using core affinity (the ability to have processes run only on select cores) to tune the performance of certain modules. Similarly, a greater number of GPU cores means that any future improvements to the autonomy package will benefit more from CUDA optimizations. Finally, The NX includes performance upgrades specifically made for better vision processing and object detections, specifically the addition of computer vision accelerator and hardware-specific code optimizations in the latest JetPack software, which will improve the performance of Darknet YOLO.

The TX2 is becoming outdated and issues surrounding the performance of the packages we use on it, such as Google Cartographer, means it might require a greater effort to improve the software stack in the future. The AGX is an overkill, and having to support it with 30 W is a burden on our Propbot's power budget. If we use lower power modes available to the AGX, then the reported performance benefits are not so clear. Using it would also add more complexity to the porting process.

Related Tag(s)	Feature required	Justification

NF2.2	Needs to have enough computing power to not be a bottleneck to autonomy package	The main constraint to a working autonomy package, currently, is the hardware it is running on
NF2.3	Must be able to interface with LiDAR, GPS, IMU, and camera	This is the sensor array which the autonomy package depends on

Table 4.2 Autonomy Computer Required Features

4.3 IMU- Inertial Measurement Unit

The IMU (Inertial Measurement Unit) is a device that typically consists of gyroscopes and accelerometers that report angular rate and force/acceleration. In the autonomy system, IMU data is used as another sensor input to the localization module.

Legacy Component: MPU-9250

This item was researched, scoped, and purchased by a previous team (See Table A.3). However, it was not integrated into any part of Propbot's system.

Final Choice: MPU-9520

Criteria: We use the same criteria identified by the previous team [1] and found that the MPU-9520 satisfies them.

To integrate into the autonomy system, we considered building a custom ROS package based on the RTIMULib2 library [3]. However, we chose to use an existing ROS package based on an older version of the library, RTIMULib[4]. Our justification is that we want to decrease the risk of failure, and so elected to choose software that is widely used in the community.

In order to calibrate the IMU, we use RTIMULib's tool package, which contains a program for calibration, as well as instructions for calibration.

4.4 GPS - Global Position System

Legacy Component: UBLOX-NEO N8P-2

This item was researched, scoped, and purchased by a previous team. It is not integrated into any part of Propbot's system.

Final Choice: UBLOX-NEO N8P-2

Criteria: We use the same criteria that the previous team used [1], with the added criterion that there must exist a ROS package that helps us interface with the device.

Selection Justification:

The device meets the criteria set out by the previous team. There exists a ROS package that supports UBLOX and uses UART for communication [5]. It is well documented, provides access to all configurations of the UBLOX, and is regularly maintained.

4.5 LiDAR

Legacy Component: Velodyne Puck VLP-16

This item was researched and scoped, but not purchased by a previous team. Properties of the current autonomy package depend on it working with this specific model.

Final Choice: Velodyne Puck VLP-16

Criteria: We use the same criteria that the previous team used, with the added criterion that there must exist a ROS package that helps us interface with the device.

Selection Justification: The device meets all criteria. We will use the official ROS package that was made in support of Velodyne [6]. We will use Power over Ethernet to connect the device to the autonomy computer since that is what the ROS package requires.

4.6 Cellular Modem

Legacy Component: None

Final choice: Particle Boron 2G/3G OR LTE CAT M1 + BLUETOOTH

Criteria:

To allow the robot to transmit telemetry and receive missions, the hardware for cellular communication needs to be chosen. This cellular hardware will link the mission control Centre and the Propbot and fulfill requirement F4.2.

The requirements related to cellular hardware are described in Table 1.5.

Selection Justification:

In order to achieve cellular communication between Propbot and the mission command Centre we have chosen the Particle Boron as it meets all of our requirements and has easy-to-use software. The Particle Workbench is the software package that comes along with the hardware and has built-in libraries that help us communicate with the Particle Cloud services. The Particle Cloud services offer a Google Cloud platform that would allow us to save data securely and use any of the multiple Google Cloud platform products. It has a ROS library that would allow us to communicate with the autonomy computer and has 20 mixed signal GPIO (6 x Analog, 8 x PWM), UART, I2C, SPI for communicating with the Arduino Mega.

Related Tag(s)	Feature required
C1	The cellular hardware shall be integrable with ROS software
F5.1, F5.2,F5.3,F5.5, NF5.1	The cellular device should allow us to send data (of size 1-2 kb) at a minimum 1 Hz sampling rate

Table 4.3 Cellular Modem Required Features

The current limiting factor is the data plan. The Free Plan allows for 100 MB cellular data per month which equates to Propbot sending 100,000 data operations per month. Assuming that the data is sent at a 1Hz rate the total run time would be around 27.5 hours per month, which is plenty for the current use of Propbot. Particle also offers paid plans with larger cellular data options which could eliminate this limiting factor.

Note: due to this item not arriving in time, we elected to use WiFi communication. To do this, we used a D-Link DIR-605L router. It is mounted on Propbot and connects to the Jetson NX through ethernet.

5. Mission Command Centre - Component Level

5.1 Data Collection

The data collection tool is responsible for saving useful data, obtained during a mission, to a file. The file can then be inspected or analysed as the user wishes. Even though most data is generated by the Autonomy System, and it is generally advisable to save data from as close to the source as possible, we chose to add data collection as part of the MCC for two reasons:

- To decrease the CPU and memory burden of having to continuously write to disk on the Autonomy System
- So that users will not have to rebuild the Autonomy Package (a time consuming process) if they want to disable / change the configurations of the data collection component

Legacy Component: None

Final choice: Rosbag

Criteria:

We require a tool that is integrable with ROS in order to decrease the amount of data transfer between different components. Although we only require the collection of location, signal strength, and mission commands, a nice-to-have feature would be to have the option to collect any ROS topic that is published on the network. This feature will benefit development teams during testing and debugging. A summary of our criteria is provided in table 5.1.

Selection Justification:

Rosbag is a popular, well documented, tool for data collection on ROS. Customizable options can be easily based through Propbot's configuration files. It supports all message types, and automatically logs the timestamp at which each message is published.

Related Tag(s)	Feature required
F5.1, F5.2, F5.3	The data collection tool should allow collection of Location, signal strength, and mission commands
F5.1, F5.2, F5.3	The data collection tool should include a timestamp for collected data
C1	The data collection tool should be integrable with ROS

Table 5.1: Data Collection features

5.2 Log Viewer

Legacy Component: None

Final choice: rqt_console

Criteria: The criteria we require is summarised in table 5.1.

Selection Justification:

Rqt_console is a common ROS tool that is used to display logging info. It allows the user to filter all logs by level (information/warning/error), ROS topic, or ROS node. Warnings and errors are displayed in a red colour, alerting the user. Since rqt_console operates on the ROS network, it is able to display logs from the VIS given through rosserial.

Related Tag(s)	Feature required

F3.5	The log viewer should be able to display information, warnings, and error logs
F3.5	The log viewer should colour warning and error logs to make them attract the user’s attention
F3.5	The log viewer should display logs from the VIS
C1	The log viewer should be integrable with ROS

Table 5.1: Data Collection features

6. Power Management System

6.1 Battery Selection

Legacy Component:

The legacy system had two unlabeled 36V 14Ah batteries that were used to power the system. One of which has been tested as broken and cannot be used. The other is functional but has some issues. Firstly, this second battery had a lock with a lost key and therefore modifications had to be made to short the locking mechanism in order to bypass it. This means that the battery's safety and security cannot be guaranteed since the modified solution is not the best practice. Furthermore, this battery alone would not have the capacity to run Propbot, both due to its rated capacity and due to possible degradation that may have occurred.

Final Choice: 3 x B3626LiM-DT 36V 26.5Ah with custom Battery Housing Unit

Criteria:

To allow Propbot to operate remotely it must have an onboard power source that can run all the components. According to requirements F1.7 and F2.8 Propbot must have enough battery capacity to run the relevant devices for 1.5hrs minimum. A trade study was performed to

estimate the amount of power Propbots components would draw over a 1.5hr period and it was found that 2400 Wh of battery capacity was needed (section 6.3).

Selection Justification:

The B3626LiM-DT battery was chosen for a variety of great features. With 954Wh of storage per battery and a total of 2862Wh with the recommended 3 batteries, the capacity of the system will have ample storage for the system with enough headroom for unexpected surges and battery degradation. These batteries will be able to handle unexpected surges in power because their combined current delivery capability of 120A far exceeds the expected current draw from full load. Furthermore, power consumption calculations in the trade study were very conservative and assumed full-time load, meaning expected usage will be lower than 2400 Wh usage, therefore, if the batteries degrade they will still meet our requirements. Each battery is equipped with a 40A battery management system that protects the battery and the user from overdrawing current, overheating, and degradation. Furthermore, these batteries are locally sourced from a nearby E-bike shop meaning we would have no shipping expenses or delays.

In part due to the larger size of the chosen battery and in part to organize and improve power distribution a new location for housing the battery has been proposed. An enclosed rectangular unit (figure 6.1 and figure 6.2 show the unit without a roof for visibility) will store the 3 batteries and their cabling and rest on the front roof of Propbot. This design removes the batteries from the narrow main chassis. The new location provides more secure mounting as well as easier accessibility and hot-swap functionality to the batteries. Furthermore, by consolidating power storage to the front of Propbot wiring complexity is reduced.

The mounting holes are lined up and measured to fit the holes that are already on Propbot's hood. Furthermore, the length and width of the box were chosen such that they will fit within the boundaries Propbot’s frame already occupies in order to ensure no collision or interference issues. The final dimensions of the box would be 50x36x15cm.

Each of the proposed 36V 26.5Ah batteries from Grin Technologies has the following dimensions: 37x9x14cm. Three batteries are currently rendered on Propbot to show that they can comfortably fit within the volume of the unit while leaving space for wiring connections.

Related Tag(s)	Feature required
F1.7, F2.8	The battery must have enough capacity to run all components

Table 6.1 Battery Required Features

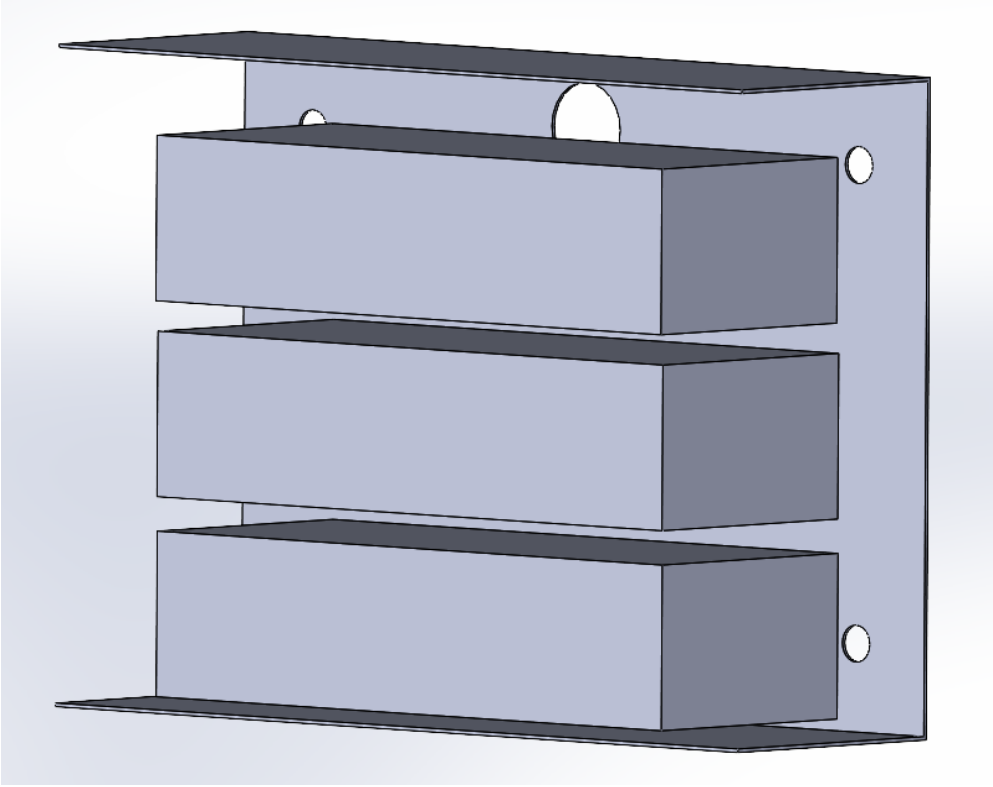

Figure 6.1 Orthogonal CAD Diagram of Batteries and Housing Unit

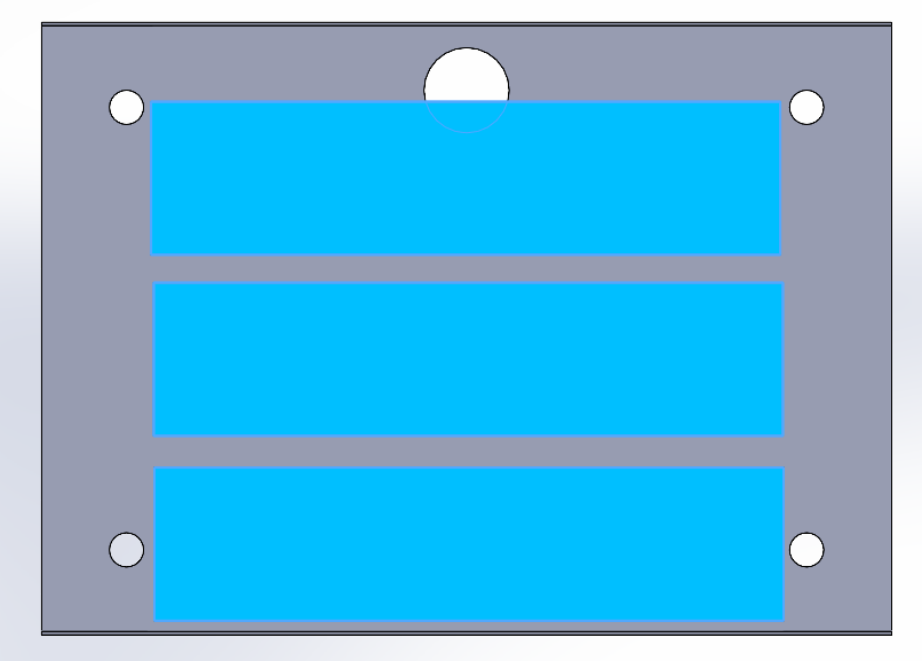

Figure 6.2 Top CAD Diagram of Batteries and Housing Unit

6.2 Power Distribution

Legacy:

Propbot's current wiring system consists of control signal cabling and power distribution. The Arduino control signals for the motor drivers are haphazardly placed and can easily come disconnected. Wires from the battery are split into the main system using choc blocks with screw-in connectors. There is some exposed wiring at these connections which could accidentally short or cause electrocution. There are no records of wiring schematics for this design or any labels on the wire harness'. In the current cabling setup, there is a power switch that controls power being delivered to the whole system. That switch was found to be defective and blocked power flow in both on and off states. This is a safety risk hence a new power delivery system is required.

Final Choice: Custom power distribution

Criteria:

All components must receive the required amount of power according to NF1.4 and NF2.4 meaning each component must receive the right voltage and current from the batteries. All power delivery must be within the specifications of each device.

Selection Justification:

Since the system is unique wiring of all components and management of voltages and currents is up to the design of this project. The power distribution system was designed in tangent with the power storage system due to the inter-reliance. The wiring system will follow the schematic shown in figure 6.3. A Powerpole bay will be placed inside the battery housing unit that allows each of the batteries to hot-swap and connect in parallel. This bay will unify the power being delivered into a single main rail. The E-stop was originally designed to be placed in series with this rail, however, upon further research, this was found to be poor practice since it could damage some of the more delicate components. Therefore, it has been moved to only be in series with the power being delivered to the motor drivers. Instead, 36V rail coming from the battery housing unit will go to a choc block distributor underneath the aforementioned housing unit so that power can easily be distributed. The power PWMs have been removed from the legacy design since the chosen Roboteq controllers can handle 36V and do not need voltage to be stepped down by the PWMs. Other components in the system, however, will need lower voltages and for that standard, high efficiency buck converters were chosen. One 36V to 12V buck converter will draw power from the choc block distributor and deliver the 12V required by the Arduino Mega, the self-leveling platform on the autonomy module, the NVIDIA Jetson NX, and the Velodyne Puck. In series with this 12V rail will be a fuse to protect the sensitive equipment and minimize the risk of component damage. One 36V to 5V buck converter will also draw power from the chock block distributor and will deliver power to the low voltage components like the sonars. The remaining autonomy components are powered through their connection to the Jetson board. The purpose of this wiring setup is to consolidate distribution, reduce clutter, reduce interdependence between compartments, and correctly power all components.

Related Tag(s)	Feature required
NF1.5, NF2.4	The wiring system should deliver the required power to each component
F3.3, NF3.2	There should be an E-stop switch that kills power to the whole system in an easily accessible location

Table 6.2 Power Distribution Required Features

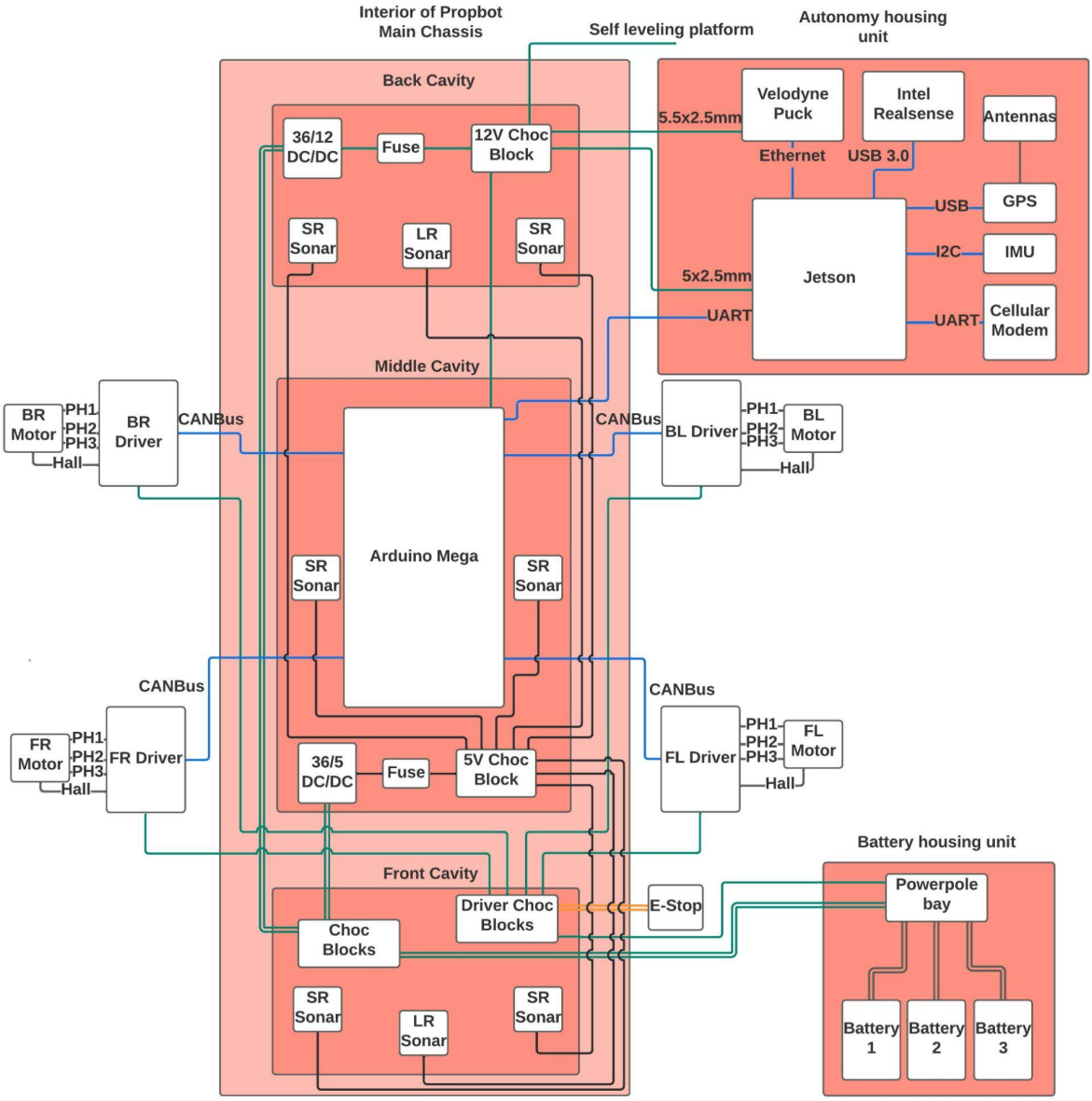

Figure 6.3 Propbot wiring diagram [2]

7. Trade Studies

7.1 Autonomy Computer

Final Choice: NVIDIA Jetson NX Developer Kit

Related Tag(s)	Feature required	Justification
NF2.2	Needs to have enough computing power to perform autonomy computations	The main constraint to a working autonomy package is the hardware it is running on
NF2.3	Must be able to interface with LiDAR, GPS, IMU, and camera	This is the sensor array which the autonomy package depends on

Table 7.1 NVIDIA Jetson NX Trade Study

Any development board that satisfies these requirements could be used. However, we have also added the requirement that the autonomy computer should come from the NVIDIA Jetson family. This is because of:

- Our team's familiarity with the NVIDIA Jetson product line
- Availability of online resources
- Environment setup is already documented for NVIDIA Jetson by the previous team

We explored three devices from the NVIDIA Jetson Family: NVIDIA Jetson TX2 Developer Kit, NVIDIA Jetson NX Developer Kit, NVIDIA Jetson AGX Xavier.

Since all these NVIDIA Jetson product kits contain support for I2C, UART, SPI, USB 3.0, and Ethernet, they fulfill NF2.3. The contribution of this trade study is to give us the necessary hardware to enable us to meet NF2.2.

NVIDIA Jetson TX2 Kit

Pros:

- Industry-standard for autonomous vehicles and AI means it is reliable
- Much more documented online resources compared to legacy TX1
- The direct successor to current TX1, meaning least amount of effort is likely required to setup
- Cost: 399 USD

Cons:

- Demonstrably weakest performance in terms of AI performance, processing power, and number of GPU cores
- Google Cartographer is reportedly very intensive and not commonly run on the TX2. To properly run, it requires effort to fine-tune the algorithm [18]
- Uses a SATA drive which is much slower than Nvme
- Not in production anymore, EOL 2025

NVIDIA Jetson NX Developer Kit

Pros:

- Official successor to TX2 and recommended by NVIDIA going forward
- NVIDIA specifically targeted better camera interfacing capabilities for this release
- Uses Nvme SSD, which is much faster than a SATA drive
- Reported 6x fps improvement on YOLO object detection compared to TX2 [19]
- Cost: 399 USD

Cons:

- Not picked up by industry or academia (because it is new) so documentation targeting it specifically might be lacking
- We will likely have to update documentation and might run into some issues since the benefits of NX come from updating to the latest NVIDIA Jetpack software and using the latest L4T kernel

NVIDIA Jetson AGX Xavier

Pros :

- Best in the edge computing business
- It beats TX2/NX in every performance test, reported 1.5x fps improvement on YOLO object detection compared to NX [19]
- Lots of academic scholarship
- Advanced features ex NVIDIA standalone profiling tools

Cons :

- Not as many online resources for non-experts
- High price ($799)
- Its customizability and abundance of features might be a burden since it is really geared for experienced professionals
- The most amount of code porting is required since no pre-built OS image is available, and NVIDIA uses a different host of tools to manage the device's environment
- Power consumption can reach 30W, higher levels than budget allows for. Although the device can be configured to remain within budget, it is unclear what performance penalties this incurs.

	Jetson TX2 Series	**Jetson Xavier NX**	**Jetson AGX Xavier Series**
AI Performance	1.33 TFLOPs	21 TOPs	32 TOPs
GPU	256-core NVIDIA Pascal™ GPU	384-core NVIDIA Volta™ GPU with 48 Tensor Cores	512-core NVIDIA Volta™ GPU with 64 Tensor Cores
CPU	Dual-Core NVIDIA Denver 2 64-Bit CPU and Quad-Core Arm®	6-core NVIDIA Carmel Arm®v8.2 64-bit CPU 6MB L2 + 4MB L3	8-core NVIDIA Carmel Arm®v8.2 64-bit CPU 8MB L2 + 4MB L3

	Cortex®-A57 MPCore processor		
DL Accelerator	—	2x NVDLA v1	2x NVDLA v1
Vision Accelerator	—	2x PVA v1	2x PVA v1
Memory	4 GB 128-bit LPDDR4 51.2GB/s	8 GB 128-bit LPDDR4 59.7GB/s	32 GB 256-bit LPDDR4x 136.5GB/s
Storage	32 GB eMMC 5.1	16 GB eMMC 5.1	32 GB eMMC 5.1
CSI Camera	Up to 6 cameras (12 via virtual channels) 12 lanes MIPI CSI-2 D-PHY 1.2 (up to 30 Gbps)	Up to 6 cameras (24 via virtual channels) 12 lanes MIPI CSI-2 D-PHY 1.2 (up to 30 Gbps)	Up to 6 cameras (36 via virtual channels) 16 lanes MIPI CSI-2 | 8 lanes SLVS-EC D-PHY 1.2 (up to 40 Gbps) C-PHY 1.1 (up to 62 Gbps)
PCIE	1 x1 + 1 x4 OR 1 x1 + 1 x1 + 1 x2 (PCIe Gen2)	1 x8 + 1 x4 + 1 x2 + 2 x1 (PCIe Gen4, Root Port & Endpoint)	1 x4 + 3 x1 (PCIe Gen4, Root Port & Endpoint)
USB	up to 3x USB 3.0 + 3x USB 2.0	1x USB 3.1 + 3x USB 2.0	3x USB 3.1 + 4x USB 2.0
Networking	10/100/1000 BASE-T Ethernet, WLAN	10/100/1000 BASE-T Ethernet	10/100/1000 BASE-T Ethernet
Power	7.5W | 15W	10W | 15W | 20W	10W | 15W | 30W

Mechanical	87 mm x 50 mm400-pin connectorIntegrated Thermal Transfer Plate	69.6 mm x 45 mm 260-pin SO-DIMM connector	100 mm x87 mm 699-pin connector Integrated Thermal Transfer Plate

Table 7.2 Technical Specifications for NVIDIA Jetson Family [20]

Overall the NX offers the best balance between price and performance. It includes performance upgrades specifically made for better vision processing and object detections, specifically the addition of computer vision accelerator and hardware-specific improvements in the latest JetPack software, which will benefit us greatly. The TX2 is outdated and issues surrounding the performance of the packages we use on it, such as Google Cartographer, cannot be ignored, even if they are avoidable. The AGX seems like overkill. Having to support it with 30 W is a burden on our system. If we use the other power modes available, then there are no performance benefits.

7.2 Cellular Modem

Final Choice: PARTICLE BORON: 2G/3G OR LTE CAT M1 + BLUETOOTH [21]

The Arduino MKR GSM 1400 and SparkFun Thing Plus - nRF9160 cellular development boards lack the ROS integration which is necessary for helping us communicate with the existing system. Both the options also do not come with dedicated cellular plans and turn out to be more expensive. The Table below compares the three options considered for the cellular modem.

	Particle Boron	**SparkFun Thing Plus - nRF9160**	**Arduino MKR GSM 1400**
Hardware Cost	$59.37 USD (EtherSIM integrated)	$129.95 USD (cellular SIM included)	$68.90 USD (Cellular SIM not included)

Cellular and Cloud Data Storage pricing	Free (100 MB cellular data per month)	None	1.99 USD per month (Entry plan)
Interface options	20 mixed signal GPIO (6 x Analog, 8 x PWM), UART, I2C, SPI	14 mixed signal GPIO (16 x Analog, 4 x PWM), UART, I2C, SPI	18 mixed signal GPIO (6 x Analog, 13 x PWM), UART, I2C, SPI
Integrated Li-po charging	Yes	No	Yes
Ros Integration	Yes	No	No
Google Cloud Platform Integration	Yes	No	Yes
Cellular network options	2G/3G (worldwide) and LTE CAT M1(North America)	LTE CAT M1 only	2G and 3G
On-board memory	1MB	4MB extended Flash memory	256KB

Table 7.3 Cellular Modem Trade Study

7.3 Battery and Power Consumption

Final Choice: 36V 26.5 Ah Downtube Battery with Panasonic GA Cells, 40A BMS

The previous capstone team had done some calculations and trade studies on how much power Propbot consumes. We went through the components listed and verified that their power draw is accurate. We also analyzed the new components that we will be proposing/using for Propbot. The combined and finalized list for these components is in Table 1.14. It was not possible to verify the power draw of the motors because we do not have a datasheet for them and have not been able to test the power draw by other means. For that reason, we have estimated the power draw based on similar components and the average scenario power draw based on speed and distance. The final power draw for Propbot is estimated to be 1599.20 W/hr. Therefore, to meet our requirement of 1.5hr minimum battery usage we would need 2398.8 Wh of battery capacity.

Hardware Component	**Power/unit (W)**	**Quantity**	**Total (W)**
RC Receiver	0.1	1	0.1
GPS	0.1155	1	0.1155
GPS Antenna	2	1	2
IMU	0.0072	1	0.0072
Ultrasonic Sensors	0.017	8	0.136
LiDAR	8	1	8
RGB-D Camera	3.5	1	3.5
NVIDIA Jetson NX	15	1	15
Vehicle Interface Board	0.34	1	0.34

Motor Driver and Motor	360	4	1440
RF Equipment	130	1	130
Total	1599.20 W		

Table 7.4 Power Consumption

Propbot currently has one battery (36V 14Ah = 604Wh) labeled working, however, we have not yet verified the true capacity of this battery since it has probably degraded. Therefore, we will be assuming that this battery is not functional and propose purchasing 3x 36V 26.5Ah (954Wh) batteries from a local electric bike vendor. This setup would give us total storage of 2862Wh. The extra storage is important since it gives some headroom for battery degradation and allows us to comfortably meet our requirements. This battery was chosen for its power density and cost. We tried to minimize the number of distinct batteries in order to reduce complexity. Higher density batteries are also cheaper per watt-hour which is beneficial since we are also trying to minimize the price. In the scenario where the chosen battery is out of stock at the time of purchase, we have also identified other batteries from the same vendor that can be purchased instead. Both configurations are summarized in Table 1.15.

Hardware Component	Price ($CAD)	Capacity (Wh)	Quantity	Total Price ($CAD)	Total Capacity (Wh)
B3626LiM-DT [22]	1247	954	3	3741	2862
B3620Li-DT [23]	1058	684	4	4232	2736

Table 7.5 Battery Trade Study

8. References

1. hannahvsawiuk, “hannavsawiuk/Propbot” *GitHub*. [Online]. Available: https://github.com/hannahvsawiuk/Propbot/wiki/Software_Design. [Accessed: 29-Nov-2021].
2. “Propbot: Lucidchart,” *Propbot | Lucidchart*. [Online]. Available: https://lucid.app/lucidchart/689448e1-0ccc-4211-909a-d3923e23cd6f/edit?invitationId=inv_6fb54af2-80ad-4a24-974c-b1ca781d7309&page=0_0#. [Accessed: 29-Nov-2021].
3. RTIMULib. (n.d.). *RTIMULib/RTIMULIB2*. GitHub. Retrieved February 14, 2022, from https://github.com/RTIMULib/RTIMULib2
4. Romainreignier, “Romainreignier/rtimulib_ros: A small package to use the rtimulib in Ros,” *GitHub*. [Online]. Available: https://github.com/romainreignier/rtimulib_ros. [Accessed: 29-Nov-2021].
5. KumarRobotics, “Kumarrobotics/ublox: A driver for ublox GPS,” *GitHub*. [Online]. Available: https://github.com/KumarRobotics/ublox. [Accessed: 29-Nov-2021].
6. “Wiki,” *ros.org*. [Online]. Available: http://wiki.ros.org/velodyne/Tutorials/Getting%20Started%20with%20the%20HDL-64E. [Accessed: 29-Nov-2021].
7. “Arduino Mega 2560 REV3,” *Arduino Official Store*. [Online]. Available: https://store.arduino.cc/products/arduino-mega-2560-rev3. [Accessed: 29-Nov-2021].
8. “SBL1360A,” *roboteq*. [Online]. Available: https://www.roboteq.com/products/products-brushless-dc-motor-controllers/sbl1360-277-detail. [Accessed: 29-Nov-2021].
9. B. Zuo, “Can-bus shield v2.0,” *seeedstudio*. [Online]. Available: https://wiki.seeedstudio.com/CAN-BUS_Shield_V2.0/. [Accessed: 14-Feb-2022].
10. “FS-I6X,” *FlyskyRC*. [Online]. Available: https://www.flysky-cn.com/fsi6x. [Accessed: 29-Nov-2021].
11. “Google cartographer running on Jetson TX1 (ROS) - YouTube.” [Online]. Available: https://www.youtube.com/watch?v=F94sJL9vgbU. [Accessed: 29-Nov-2021].

12. Wrinkler, D. (n.d.). *Winkler, D. (n.d.). How 6 River Systems Leveraged the Use of ROS and the NVIDIA Jetson Platform to Build a Fleet of Autonomous Collaborative Robots. NVIDIA*. . NVIDIA.com. Retrieved February 14, 2022, from Winkler, D. (n.d.). How 6 River Systems Leveraged the Use of ROS and the NVIDIA Jetson Platform to Build a Fleet of Autonomous Collaborative Robots. NVIDIA.
13. *Autonomous underwater drones - developer.download.nvidia.com*. NVIDIA.com. (n.d.). Retrieved February 14, 2022, from https://developer.download.nvidia.com/assets/embedded/downloads/jetson_solution_showcase-notilo.pdf
14. *An innovative challenge and holistic solution for Autonomous Delivery*. (n.d.). Retrieved February 14, 2022, from https://developer.download.nvidia.com/assets/embedded/downloads/jetson_solution_showcase-meituan.pdf
15. arthurBricq. (n.d.). *Arthurbricq/ROS_ROBOTTLE: Repo with the ros code of the Jetson Nano*. GitHub. Retrieved February 14, 2022, from https://github.com/arthurBricq/ros_robottle
16. *UBC Agrobot*. UBC AgroBot. (n.d.). Retrieved February 14, 2022, from https://ubcagrobot.com/agrobot/
17. *F1 tenth scaled autonomous vehicle installation ... - youtube*. (n.d.). Retrieved February 14, 2022, from https://www.youtube.com/watch?v=q2HhWP_m2w4
18. Peng, Tao & Zhang, Dingnan & Hettiarachchi, Don Lahiru & Loomis, John. (2020). An Evaluation of Embedded GPU Systems for Visual SLAM Algorithms. Electronic Imaging. 2020. 325-1. 10.2352/ISSN.2470-1173.2020.6.IRIACV-074.
19. "Jetson benchmarks," *NVIDIA Developer*, 14-Jun-2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-benchmarks. [Accessed: 29-Nov-2021].
20. "Jetson Modules," *NVIDIA Developer*, 14-Jun-2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules. [Accessed: 29-Nov-2021].
21. "Boron datasheet: Datasheets," *Particle*. [Online]. Available: https://docs.particle.io/datasheets/boron/boron-datasheet/. [Accessed: 29-Nov-2021].

22. “36V 19ah 21700 downtube battery,” *ebikes.ca*. [Online]. Available: https://ebikes.ca/shop/electric-bicycle-parts/batteries/b3620-li-dt.html. [Accessed: 29-Nov-2021].
23. “36V 26.5ah downtube battery,” *ebikes.ca*. [Online]. Available: https://ebikes.ca/shop/electric-bicycle-parts/batteries/36v-26-5ah-downtube-battery.html. [Accessed: 29-Nov-2021].
24. DavidW10 and Instructables, “Arduino RC Circuit: PWM to Analog DC,” *Instructables*, 13-Oct-2017. [Online]. Available: https://www.instructables.com/Arduino-RC-Circuit-PWM-to-analog-DC/. [Accessed: 04-Apr-2022].

A. Appendix

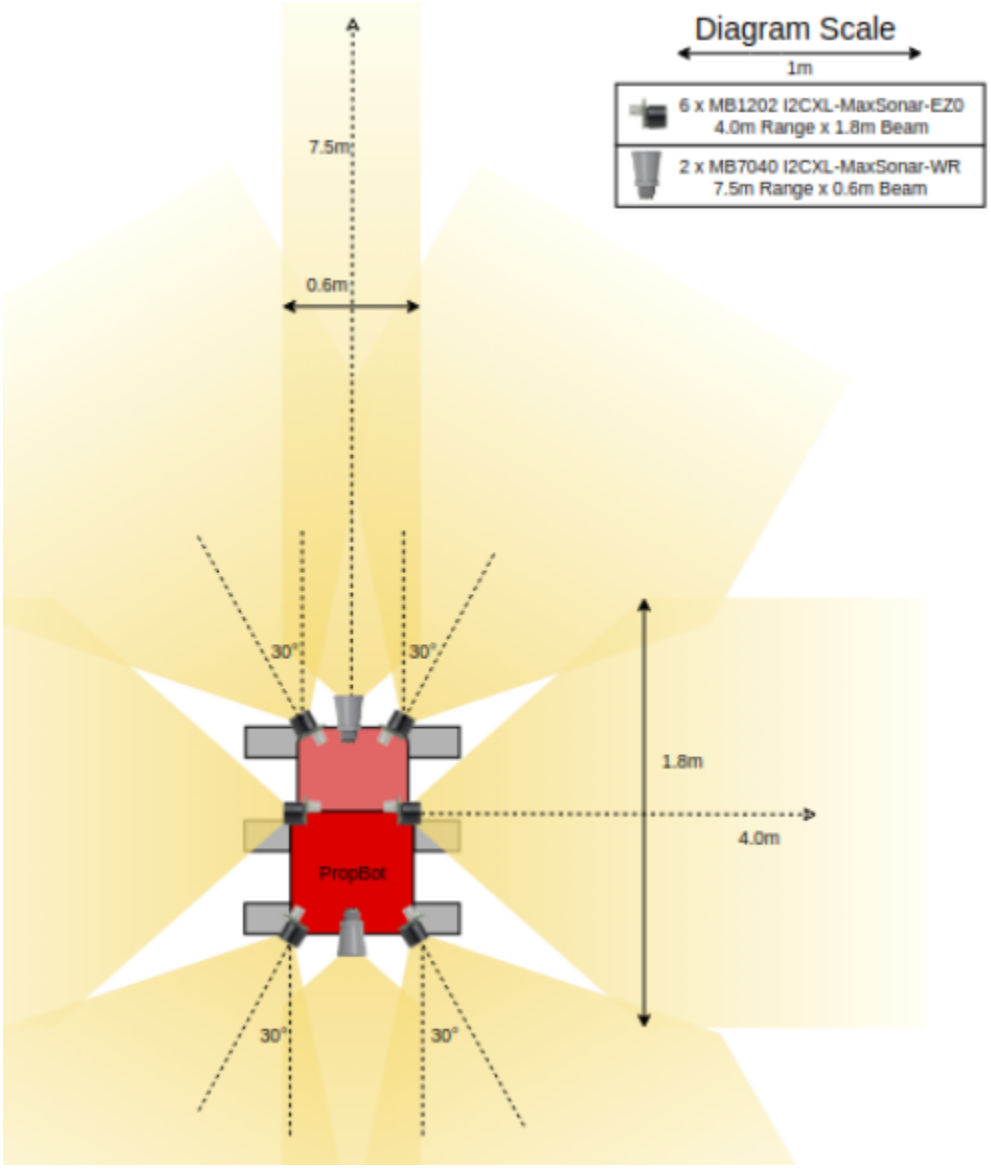

Figure A.1 Propbot Sensor Beam Mapping

A.1 Motor Controllers

Parameter	RoboteQ FBL2360	**RoboteQ SBL1360A**	KEYA DC10/50DPW15BL
Electrical Specifications			
Max Voltage	60V	60V	48V
# of Channels	2	1	1
Max current/channel	60A	30A	15A
Continuous current/channel	40A	20A	8A
ON resistance	2.5 mOhms	20 mOhms	-
Max power dissipation	3600 W	1200 W	384 W
Control loop	1000 Hz	1000 Hz	-
Regenerative braking	Yes	Yes	No
RC controllable	Yes	Yes	No
Other supported communication	Analog, RS232, USB, CANbus	Analog, RS232, USB, CANbus	Analog
Mechanical Specifications			
Weight	452g	60g	300g
Dimensions	140mm x 140mm x 25mm	70mm x 70mm x 27mm	122mm x 35mm x 72mm
Pricing and Availability			
Price for system	USD 1390 (2 units)	USD 1100 (4 units)	USD 152 (4 units)
Availability	Readily available	Readily available	Readily available

Table A.1 Motor Controller Trade Study from Previous Team

A.2 Sonar Sensors

Parameter	Sharp GP2	VL53L1X	TinyLiDAR	HC-SR04	**MaxSonar**
Sensor Type	Infrared	Laser	Laser	Ultrasonic	Ultrasonic
Cost	$18	$16	$33	$3	$100
Max Range	5.5m	4m	2m	5m	10m
Accuracy	4cm	4cm	3%	0.3cm	1cm
Sample Rate	60 Hz	50 Hz	60 Hz	20 Hz	10 Hz
Arc	Small	Point	Point	Small	Small
Casing	Not waterproof	Bare PCB	Bare PCB	Bare PCB	Some Models Waterproof
Interface	Analog	I2C	I2C	GPIO Timer	Serial or I2C

Table A.2 Sonar Sensors Trade Study from Previous Team

A.3 IMU

Parameter	**MPU-9250**	LSM9DS1
Voltage	2.4 - 3.6 V	3.3 V
Communication Protocols	I2C x 1 SPI x 1	I2C x 1 SPI x 1
Gyroscope Features	-3-axis digital output -Programmable range of $\pm 250, \pm 500, \pm 1000, and \pm 2000^{o}/sec$ -Integrated 16-bit ADCs	-3-axis digital output -Full scale of $\pm 245, \pm 500, and \pm 2000^{o}/sec$ -16-bit data output
Accelerometer Features	-3-axis digital output -Programmable range of $\pm 2g, \pm 4g, \pm 8 gand \pm 16g$ -Integrated 16-bit ADCs	-3-axis digital output -Full scale of $\pm 2g, \pm 4g, \pm 8, and \pm 16g$ -16-bit data output
Magnetometer Features	-3-axis digital output -Full scale measurement range is $\pm 4800 \mu T$ -14-bit ADC word length	-3-axis digital output -Full scale of $\pm 4, \pm 8, \pm 12, \pm 16 gauss$ -16-bit data output
Notes	Ros node driver is open-source	-
Price	USD 14.95	USD 15.95

Table A.3 IMU Trade Study from Previous Team

A.4 GPS

Parameter	**Ublox NEO-M8P-2**	Ublox SAM-M8Q	Ublox NEO-M8NL
Satellite Type	GNSS	GNSS	GNSS
# of Concurrent Satellites	2	3	3
Timepulse	Yes	Yes	Yes
RTK	Yes	No	No
Horizontal Accuracy	Without RTK: 2.5m With RTK: 2.5cm	2.5m	2.0m
Voltage	3.3V	3.3V	3.3V
Communication Ports	UART x 1 SPI x 1 I2C x 1 Micro B connector x 1	UART x 1 I2C x 1	USB X 1 UART x 1 SPI x 1 I2C x 1
Price	USD 199.95	USD 39.95	USD 22.95
Notes	This is meant to be used as a differential GPS system with a "base" and a "rover" station. However, in this case it would be used as a standalone device.	Has a built-in patch antenna on the board.	-

Table A.4 GPS Trade Study from Previous Team

A.5 LiDAR

Parameter	Slamtec RPLi-DAR	Ouster OSI-16	**Velodyne Puck**	Livox Mid-100	SICK MRS1104C-111011
Distance (m)	20	40	100	90	30
Sample Rate (points/sec)	10000 or 16000	327680	600000	300000	55000 to 165000
Evaluated Echos	1	1	2	1	3
Horizontal FOV (deg)	360	360	360	98.4	275
Horizontal Ang. Res (deg)	0.36 or 0.225	0.703125	0.1 to 0.4	0.1	0.25
Vertical FOV (deg)	N/A	16.6 to -16.6	15 to -15	38.4	7.5
Vert.Ang.Res.(deg)	N/A	0.53	2	0.1	1.875
Environmental Protection	IP5	IP68	IP67	IP67	IP67
Power (W)	3	20	8	30	13
Price (USD)	Purchased	3500	4999	1499	3862.5

Table A.5 LiDAR Trade Study from Previous Team

Slamtec RPLiDAR	-A 2D scan of the environment is not sufficient for wave propagation analysis.
Ouster OSI-16	-No software or hardware methods to mitigate interference from fog/rain/dust.
Velodyne Puck	-Velodyne Puck is widely used for mobile robot projects and thus has extensive support in many simulation software.
Livox Mid-100	-Has unique scanning pattern which increases coverage but can make it difficult to use with most SLAM packages. -Fog/rain/dust detection suppression using software algorithms which decreases sensor's ability to detect surfaces with low reflectivity. -Requires an array of at least 3 LiDARs mounted horizontally to cover a horizontal FOV >270 degrees.
SICK MRS1104C-111011	-Requires an array of at least 4 LiDARs placed vertically to cover a vertical FOV of > 30.

Table A.6 LiDAR Trade Study from Previous Team (Ctnd.)

A.6 Depth Camera

Parameter	**Intel Realsense D435 (RGB)**	Leopard Imaging Sony IMX185	D3 Engineering, Rugged Sony IMX390-953	FLIR Blackfly S GigE w/ Sony IMX430 Color
Interface	USB 3.0	CSI MIPI x2/x4	CSI MIPI/FPD	Ethernet
Cost	CAD 250	USD 379 (x1) USD 270 (x6)	USD 449	USD 545
Casing	Metal, not waterproof	None (Bare PCB)	Metal, rugged, waterproof	Metal, not waterproof
Resolution	2MP 1920 x 1080	2MP 1952 x 1241	2MP 1920 x 1200	2MP 1616 x 1240
Video Resolution	1080p	1080p	1080p	1080p
Video Frame rate	30 fps	30 fps (MIPI x2) 60 fps (MIPI x4)	60 fps	60 fps
Shutter	Global	Rolling	Rolling	Global
Pixel Size	3 micron	3.75 micron	3 micron	4.5 micron
Lens	Integrated	CS or S Mount	S Mount	Lens Not Incl.
FOV (deg)	69.4 H x 42.5 V	90 H x 50 V (CS) 92 H x 60 V (S)	76 H x 70 V	N/A
Aperture	N/A	f/2.2 (CS) f/2.8 (S)	f/2.8	N/A
Distortion	N/A	-8% (CS) -1% (S)	17% rectilinear	N/A

Table A.7 Depth Camera Trade Study from Previous Team

A.7 Autonomy Package 1.0

The Autonomy system is based on a ROS (Robot Operating System) framework, which is common for many robotics projects. ROS provides a messaging layer so that the main components, called packaged, can work independently via a defined, shared interface. Additional benefits include the availability of open-source packages and online support.

The software architecture for our ROS packages is shown in the figure below (A.3). This figure highlights all of Propbot's custom packages that are added to carry out localization, motion planning, mission command, and robot control functions. The software is modular and

complies with the general ROS architecture in mind, so that the open-source ROS SLAM, navigation, and control packages can be used.

The general functionality of the packages running on the vehicle autonomy computer in the diagram are described below.

- **Propbot Mission:** This package takes in a mission and sends goals to *Propbot Navigation*. Using the "GPS to Robot Map Transform" from , it converts GPS coordinates specified in the mission to locations on the "Robot Map". It sends these as "Goal Waypoints" on the robot map to *Propbot Navigation* consecutively as each waypoint is reached. This package also has the ability to interface with the Propbot GUI package to upload, start, and stop missions.
- **Propbot SLAM:** This package houses the robot's SLAM algorithm. The SLAM algorithm uses sensor data, specifically GPS, IMU, lidar, and the depth data from the RGB-D camera, to generate a map of the robot's surroundings. This map identifies the robot's location in the world, as well as the objects the robot sees around it. We refer to this as the "Robot Map" in figure A.3. The SLAM package outputs this map to *Propbot Navigation* to be used for motion planning. In addition, it outputs a transform between GPS coordinates to positions on the robot map to *Propbot Mission*.
- **Propbot Navigation:** This package takes in a goal waypoint on the robot map and plans a feasible route for the robot. It plans a route that is free of collisions with obstacles, and it continuously updates this route plan as it receives new information from *Propbot SLAM*. It then converts this route to velocity (speed and direction) commands and sends them to *Propbot Control*.
- **Propbot Control:** This package takes in velocity commands and translates them to left and right wheel speeds to be sent to the robot.
- **Propbot State Estimation:** This package accepts robot pose estimates from the SLAM algorithm as well as data from both situational awareness and positional sensors to compute the nonlinear state estimations for the robot moving in 3D space.
- **Propbot Object Detection:** This package detects and tracks dynamic obstacles by computing their bounding boxes with the *You Only Look Once (YOLO)* and then

projecting these bounding boxes into 3D space using laser scan data and an implementation based on a pinhole camera model.

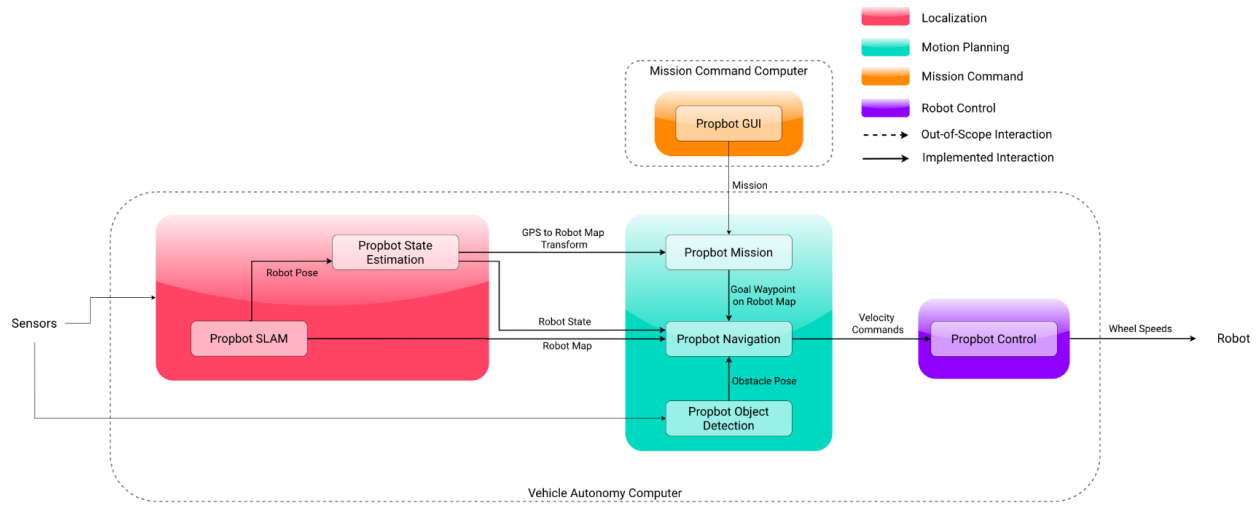

Figure A.3 Autonomy Package 1.0 Details

A.8 Vehicle Interface Communication Choice

The following justification is from the design document of the previous Capstone team:

The selected RoboteQ SBL1360A motor drivers use true serial (inverter twisted pairs, [-10V, +10V]), whereas the Arduino utilizes Transistor-Transistor Logic (TTL) serial, which is [0V, +3.3V/+5V].

To facilitate communication across the differing voltage levels, either resistors and level-shifters

can be used on the driver side to convert to TTL, or a fully integrated RS-232 to TTL converter can be purchased. However, RS-232 would be most ideal for communicating with one device,

such as a single motor driver. But, communicating over RS-232 to a single driver and then having that driver relay the message over CAN introduces latency. Instead, the vehicle interface should act as the master and the drivers as the slaves. Further, it is possible that more than one device will make up the vehicle interface as the number of input ports come to limit the integration of additional peripheral devices for improved emergency response, and driving capabilities. Therefore it is important to use a communication protocol that supports multiple master devices, such as CAN.

RS-232 vs. CAN:

RS-232 can be accomplished via two wires and can be extended to complex multi-wire protocols. However, the main drawback of RS-232 is that the transmitter and receiver configurations are single ended. This causes the system to be very susceptible to noise, especially at higher baud rates (up to 19.2Kbps [28])

CAN is a complex protocol, however it allows for multiple masters along with multiple connected communication nodes at a higher baud rate (up to 1Mbps for CANopen [29]) with more noise efficiency. Luckily, there is a lot of support on both the Arduino side and the RoboteQ side for CAN protocols (CANopen, RoboCAN, etc). In terms of hardware support, CAN adapter boards for the Arduino Mega are available so nothing custom has to be created. For these reasons, the vehicle interface will communicate with the motor drivers via CAN.

A.9 Motor Control Component Selection

The following components were selected in accordance with Roboteq SBL1360A datasheet recommendations.

Name	Specification	Power rating	Model	Quantity
Precharge resistor	1kΩ	0.5W	Stackpole Electronics RNF12FTD1K00	4

Flyback diode	30A	66W @ 2.2V forward voltage	Infineon Technologies IDV30E65D2X KSA1	4
Inline fuse	2A-30A	360W @ 30A	JOREST 60Pcs Car Fuse Kit	4
DB15 breakout	N/A	N/A	NorComp Inc. 171-015-103L00 1	4
DB9 breakout	N/A	N/A	TE Connectivity AMP Connector 747844-2	1

Table A.8 Motor Control Component Selection

A.10 Existing Motor Controllers

Due to supply chain issues surrounding the Roboteq SBL1360A motor controllers, we resorted to using the motor controllers that were existing on Propbot for this project. These motor controllers are unbranded and we were unable to find documentation for them. However these styles of motor controllers seem to be a common standard in Chinese E-bikes kits. Through various online sources we were able to identify the key pinouts for this motor controller which are power, motor phase,speed, direction, and braking. See Figure A.4.

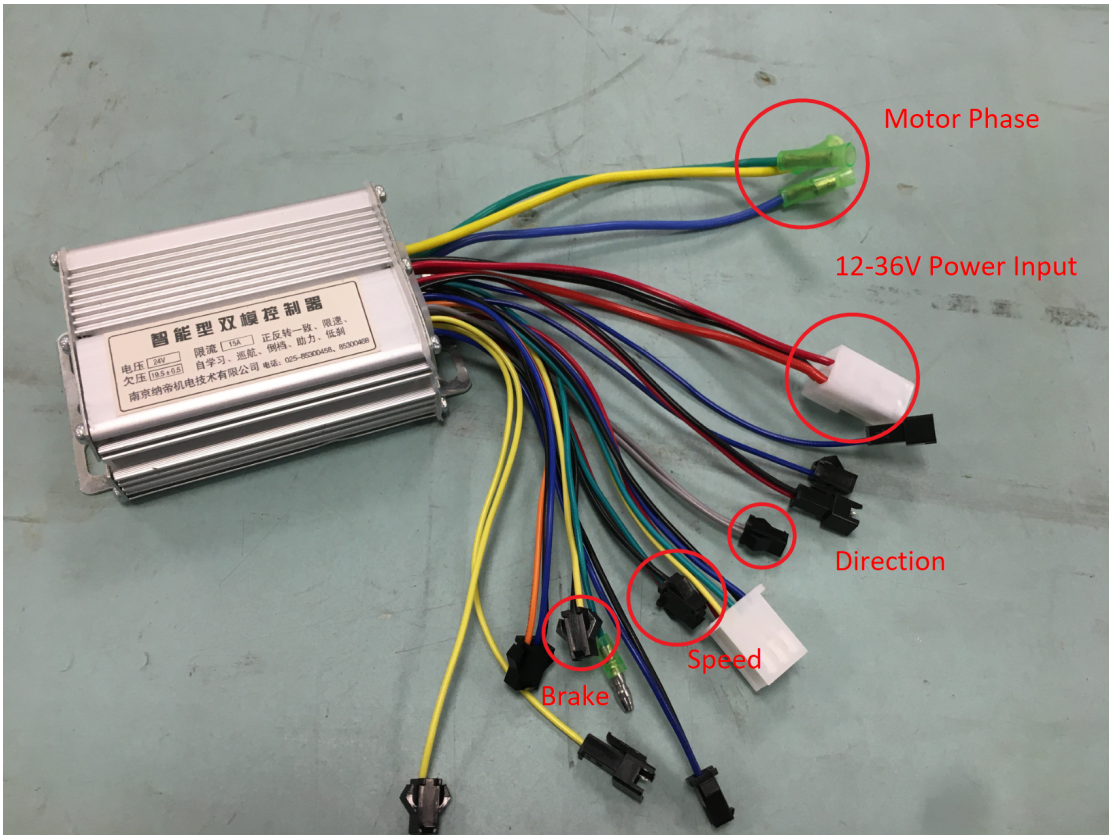

Figure A.4

As these motor controllers are designed to be used with E-bikes, the method for controlling speed is based upon a user-controlled throttle which varies an analog voltage to the speed pinout, similar to a potentiometer. This is different from most standard controllers which rely on a digital PWM input to control speed, usually driven by a microcontroller. Because of this key difference, the E-bike motor controllers can only be operated reliably using an analog voltage input. The result of using a PWM signal is discontinuous and jerky motion of the motors, likely due to the sampling method employed by the controller. To overcome this issue, a low-pass RC filter can be placed between the Arduino PWM output and motor controller input, which serves to filter out higher frequency switching harmonics, and therefore produce a more ideal analog output similar to that of the potentiometer-based throttle. See Figure X.X.

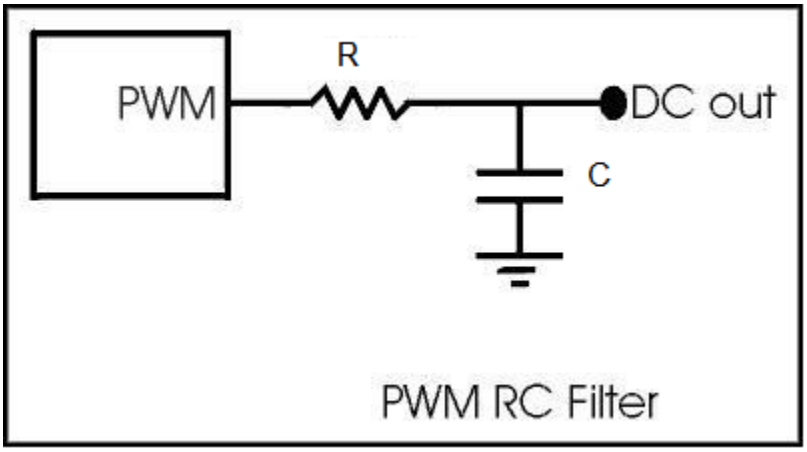

Figure A.5

This basic setup was used for all four motor controllers resulting in smooth operation of the motors. The values for the resistors and capacitor values were chosen based on the desired cutoff frequency according to the following formula:

$$f_{cutoff} = \frac{1}{2\pi RC}$$

Since the default frequency produced by Arduino Mega 2560 PWM outputs is 490Hz this cutoff frequency was chosen. The corresponding values for R and C were selected as 1kΩ and 100 µF. These values provided an output signal similar to the one shown in image A.6 [24] taken from an online guide which performed filtering for the Arduino 2560 using the same selected component values.

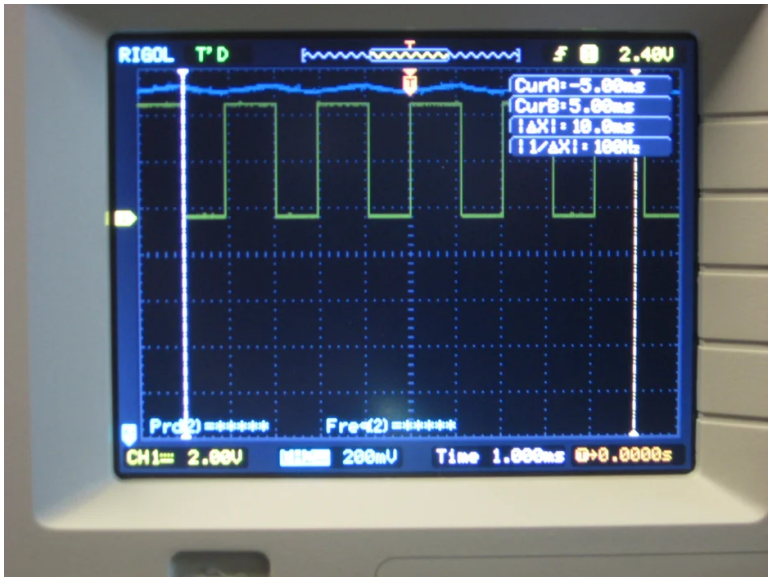

Figure A.6

While a small amount of ripple was still present, this filtered signal enabled the motor controllers to drive the motors smoothly to meet requirement NF1.7.