Design Specifications

Capstone PL-116

April 8th, 2020

Hannah Sawiuk (HS), 51196160

Divya Budihal (DB), 24512155

Yan Hong (YH), 31510150

Jack Guo (JG), 31373153

Zhaosheng Li (ZL), 43032168

Contents

Glossary

BLDC (brushless DC) Synchronous motors powered by direct current electricity via an inverter or switching power converter.

COCO Mean Average Precision (COCO mAP) A metric used to evaluate object detection algorithms.

Differential Drive A two-wheeled drive system with independent actuators for each wheel. The name refers to the fact that the motion vector of the robot is sum of the independent wheel motions, something that is also true of the mechanical differential (however, this drive system does not use a mechanical differential).

Dynamic Driving Task (DDT) This term encompasses all of the real-time operational and tactical functions required to operate a vehicle.

E-Stop (Emergency Stop) A safety mechanism used to shut off power in an emergency, when it cannot be shut down in the usual way. We use it in the context of both the physical switch on the robot as well as the remote halt command.

Extended Functional/Non-functional Requirement (EFR/ENFR) Given the broad scope of the PropBot initiative and the relevance of the requirements of the initiative to the narrowed scope of our project, functional and their non-functional requirement counterparts that are out of the scope of our contributions have been renamed to "extended" functional and non-functional requirements (abbr: EFR and ENFR).

FOV (Field of View) The field of view is the extent of the observable world that is seen at any given moment.

IMU (Inertial Measurement Unit) An electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers [1].

National Highway Traffic Safety Administration (NHSTA) An agency of the U.S. federal government, part of the Department of Transportation.

Object and Event Detection and Response (OEDR) The action of detecting an object or event near the robot and responding accordingly. This specifically refers to the subtasks of the DDT that include monitoring the driving environment [2].

Operational Design Domain (ODD) Describes the operating domains in which the system is designed to function. ODD examples include speed range, lighting conditions, weather conditions, roadway types, etc. [2].

Operational Scenario The environment in which a feature is designed to function, and it is described by a set of ODDs [2].

Plugins Software components that add specific functionalities to a program..

Point Of Interest (POI) A specific location of interest. Within the context of path-planning, a POI is a waypoint that the path must include. Within the context of propagation data collection, a POI is the point where measurements are to occur.

PWM (Pulse Width Modulation) A method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts..

RC (Remote Control) An electronic device used to operate another device from a distance, typically wirelessly.

Robot Pose The position and orientation of a robot in the 3-D space..

Robot State The position, velocity, orientation and angular velocity of the robot in 3-D space..

SLAM (Simultaneous Localization and Mapping) A computational algorithm of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.

Transistor Transistor Logic (TTL) A class of digital circuit built from transistors and resistors. It is called transistor–transistor logic because the logic function (e.g., AND) and amplification is performed by transistors. Many integrated have TTL technology. They are used in applications such as computers, industrial controls, test equipment and instrumentation, synthesizers, etc. TTL gates define a voltage below 0.5V as 0, and a voltage of 4–5V as 1. [3].

World Geodetic System (wgs84) The standard coordinate system referenced by Global Positioning System (GPS). [4].

1 System Overview

1.1 Autonomy System Overview

1.1.1 Components of Autonomy

The autonomy of a system is a largely dependent on its "brain". That is why the system at a high level is most easily described by the basic software components of an autonomous mobile robot. Although hardware components of this system are very important, they are moreso a "means to an end" that give the system's software access to the right information to make meaningful decisions in a safe way. This is why our entire system is designed around the basic software components that make up an autonomous ground vehicle. These components and their interactions are illustrated in Figure 1.1. This architecture is derived from various previous autonomous mobile robot projects (ex. *TIGRE* [5] and *CaRINA* [6]) and papers on autonomy: [7, 8, 9].

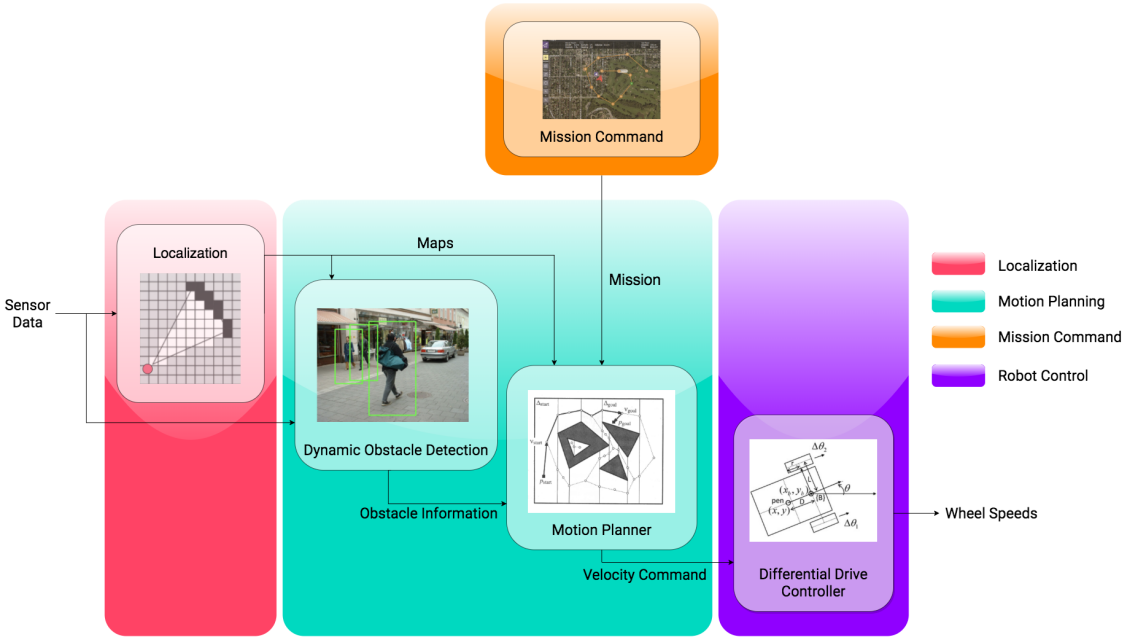


Figure 1.1: Proposed system software components

The diagram is color coded with the following categories:

- **Localization:** This refers to the robot using sensor data to determine its location with respect to its environment [10]. This also involves building a map of the environment it traverses through. This information is often represented in a 2D occupancy grid. For example, in Figure 1.1, in the localization block, the red dot represents the robot location, the white represents free space, black represents an obstacle, and grey represents unexplored areas.
- **Motion Planning:** This refers to the robot generating a path that moves the robot from its starting location to a goal without colliding with any obstacles [11]. This achieved by using maps and obstacle information to generate a feasible trajectory and output velocity commands for the robot. For the sake of simplicity, we have grouped obstacle detection with motion planning, as this will drive the considerations we make throughout the document.
- **Mission Command:** This refers to the user-facing software component that will allow for a

mission to be created and transmitted to the robot. A typical mission will consist of waypoints that are points of interest for data collection. This mission is sent to the robot motion planner so the robot can create its goal locations accordingly.

- **Robot Control:** This refers to taking a velocity command and generating the wheel speeds for the robot that will make the robot move in the desired direction at the desired speed.

These components are used in the rest of the document to categorize the different aspects of our proposed system. The cited projects [5, 6] use similar categories as us to describe their software. This is because each of the components have well-defined inputs and outputs that do not change regardless of the specific algorithms or packages that are used to implement the components. For example, localization will always output a map with the robot's position, reglardless of the specific type of localization package used.

Dividing our project into these subsystems have helped us modularize our software and hardware as well as helped us evaluate our priorities. For example, it is useful to understand that the mission command must interface with the motion planning, but it is unconcerned with what localization is doing. Using these distinct categories we are able to define the selection criteria for their relevant hardware and software.

1.1.2 Summary of our Contributions

Our project has focused on the systems level design of the hardware and software components for Propbot's autonomy package. In addition our contributions include the development of the key software components of the autonomy package, as well as the build up of a hollistic simulation environment to test software with.

More specifically our contributions are as follows:

Hardware

- Evaluation of the legacy electro-mechanical system and proposing new electrical system components.
- Evaluation the legacy sensor suite and selecting a new autonomy sensor suite.
- Design of a comprehensive connection diagram outlining system integration, as well as sensor mounting guidelines for the robot.
- Developing a vehicle interface firmware stack for interfacing with the drive system, the autonomy computer, and peripheral sensors.
- Demonstration of the of tele-operation, close-range obstacle avoidance, and vehicle firmware on a surrogate robot (small amateur remote control car).

Software

- Development of an autonomy package so a robot will safely follow a programmed route throughout campus.
- The integration of sensors for situational awareness and position estimation that improve the robot's safety and navigation features
- Development of a mission command centre graphical user interface (GUI) package so that the user can configure a mission and monitor its progress remotely

- Development of a simulation package which integrates the sensor suite, the Robot's 3D model, and a custom simulation environment which mimics the robot's operational zones in the real world.

Software

1.1.3 Safety

It is important to note that with all autonomous systems, safety is of the utmost priority. The reason that "Safety" is not one of the categories illustrated in the above section is that safety is an important consideration across *all* subsystems. Throughout our design document, there are statements labelled "**Safety Feature**" that specifically highlight the safety considerations in certain design decisions. These decisions are explained in blue text.

The following is a list of safety features and the page numbers on which they are mentioned:

List of Safety Features

1.2 Legacy System

Propbot is a six-wheel autonomous robot developed by UBC Radio Science Lab (RSL). The RSL conducts a lot of research on wireless systems, which requires a lot of wireless data measurement. Researchers can utilize the robot to deliver the data measurement equipment and avoid recording location data manually.

Figure 1.2 illustrates the current formation of Propbot.

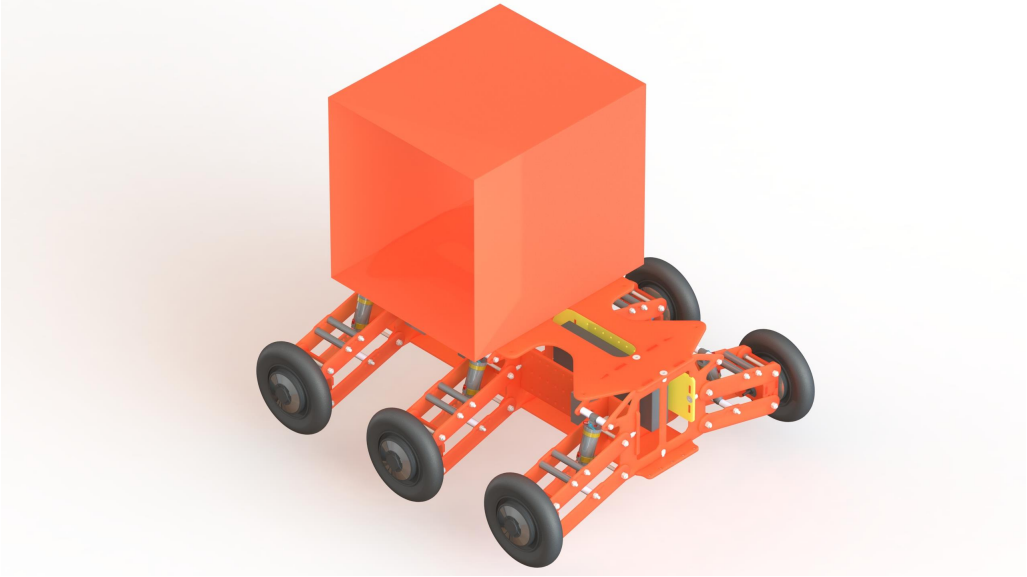


Figure 1.2: Rendering of the Propbot CAD

Currently, the robot is powered by two 36 V lithium ion batteries in parallel. The power of the batteries is sent to four DC/DC step-down converters and an emergency stop (E-stop) button. The six wheels are controlled by six independent brushless DC motors but only four of them receive power from the DC/DC converters.

The user can send commands to the vehicle interface, through Ethernet. After receiving those commands, the vehicle interface sends Pulse Width Modulation (PWM) signal to the wheel's motors.

However, the electrical system is not currently safe or operational as several components including the DC/DC and brushless motors are not performing as expected and the wires connecting the system components can easily be loosened. These faults have led to the malfunction of the E-stop button and the vehicle drive module. **For more information on the existing robot, please refer to Appendix A**.

1.3 Proposed System

Figure 1.3 shows our proposed system, at a high level and each component is described below.

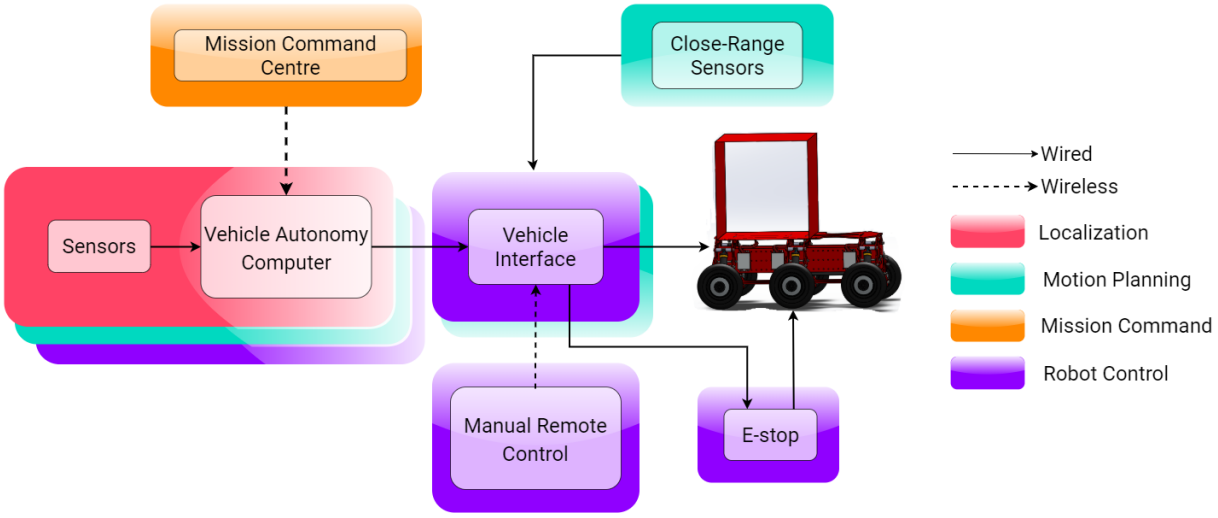


Figure 1.3: High-level system architecture

Vehicle Interface

The vehicle comprises the mechanical system along with the motors, the motor drivers, vehicle interface/control module (which may be one or more device), RC link, e-stop, and the power system. The choice to separate the vehicle interface from autonomy computer is due to the following factors:

1. The autonomy computer will be processing all the situational awareness sensors and running the autonomy algorithms, which requires a lot of processing power. If the autonomy computer was responsible for the emergency stop and motor driving, the latency caused by the other processes would decrease the reaction time of the vehicle could cause collisions.
2. The vehicle interface will allow the remote controller to interface on the lowest level. This ensures that manual control is decoupled from the autonomy computer, which is not a Real Time Operating System, and thus does not have real time guarantees.
3. The close-range sensors should be able to halt the robot's movement as quick as possible. Feeding the sensor network into the vehicle interface ensures that the triggering of the sensors will be prioritized. This also satisfies requirement EF5.1 for manual control.
4. The autonomy computer is not suitable for interfaces via GPIO.

Safety Feature 1 (Dedicated vehicle interface processor)**.** *The choice to use a dedicated vehicle interface ensures that the system that interfaces with the motors is real-time safe. Since the vehicle autonomy computer uses a Linux OS, it is not real-time safe and its processes could potentially fail during operation. Having a separate vehicle interface removes issues caused by software processes dying, latency, and low frequency updates from the autonomy software.*

Close Range Sensors

As mentioned above, the close-range sensors are active even in manual mode because they are connected directly to the vehicle interface, thus bypassing the autonomy computer. This satisfies requirement EF5.1.

Safety Feature 2 (Close-range sensors on the lowest level)**.** *The choice to connect close-range sensors directly to the vehicle interface ensures that there is "last-minute" collision avoidance in the event of miscalculations by the autonomy stack or erroneous remote commands by the user. This ensures that on the lowest level, there is some form of automatic collision avoidance for small distances.*

Sensors

These include the situational awareness and position estimation sensors which will provide the robot with an understanding of its surroundings and its current state. These sensors will be used for localization, motion planning, and robot control.

E-Stop

Note: this is not a power-off switch and does NOT cut power to any systems.

The E-Stop serves as an emergency stop for the vehicle. As shown, the emergency stopping capabilities can be activated by the vehicle controller. Once the E-stop button is pressed, the vehicle controller stops sending any PWM signal to the wheels' motors and activates the brake inside each motor. But there is still a potential hazard. When the E-stop is pressed and the robot is on a downhill, the brakes in the motors can be engaged but the robot may keep slipping. Due to our time constraint, a mechanical brake system is necessary for the future capstone teams to implement on the robot.

Safety Feature 3 (On-board E-stop)**.** *Although the current E-stop is not guaranteed to fully stop the robot, especially when the robot is on a downhill surface, it is still useful on many surfaces, given that the batteries are able to supply enough current to the motors. An on-board E-stop allows road users (an important stakeholder for our project) to press the button on the robot itself if they notice unwanted behaviour. This concern is taken into account by project constraint C1.5 which limits the operation of the robot to flat terrain.*

Power-Off Switch

For very extreme situations in which power to the system must be completely cut (power lockout), there is an existing switch that can sever the entire system's connection to the battery bank.

Safety Feature 4 (On-board Power-Off Switch)**.** *This switch allows road users as well as the robot use to cut power to the robot in emergency cases if necessary.*

Mission Command Centre

The robot reports its position and status to the mission command center via wireless link. The mission command center can be either a desktop at the Radio Science Lab or a laptop that a tester uses to monitor the robot, as long as the desktop or the laptop has *Linux* operating system. For our capstone team, the mission command center refers to the latter, but we still develop a software package that is compatible with the desktop at RSL. According to the mission command centre requirements in our *Requirements Document*, the wireless link between the robot and the mission command center allows for a limited set of commands: mission start, mission end, and mission upload.

Vehicle Autonomy Computer

The autonomy computer brings together the localization, motion planning, and control of the robot. The algorithms running on the computer take in the sensor information to generate its "world" and then makes motion decisions based off of the surroundings and the user-generated path. It is important to note, as listed in C1.3, the autonomy computer has already been chosen and acquired by the client. Thus, the autonomy computer is the NVIDIA Jetson TX1, and it is referenced in the following sections.

2 Hardware Architecture

2.1 Overview

Figure 2.1 provides an overview of the entire hardware architecture and shows the categories that each piece of hardware belong to. Figure 2.2 shows the communication protocol for each wired communication in the hardware architecture. The decision to use the sensors in the dotted box in Figure 2.1 and the specific hardware selected is described in the following sections.

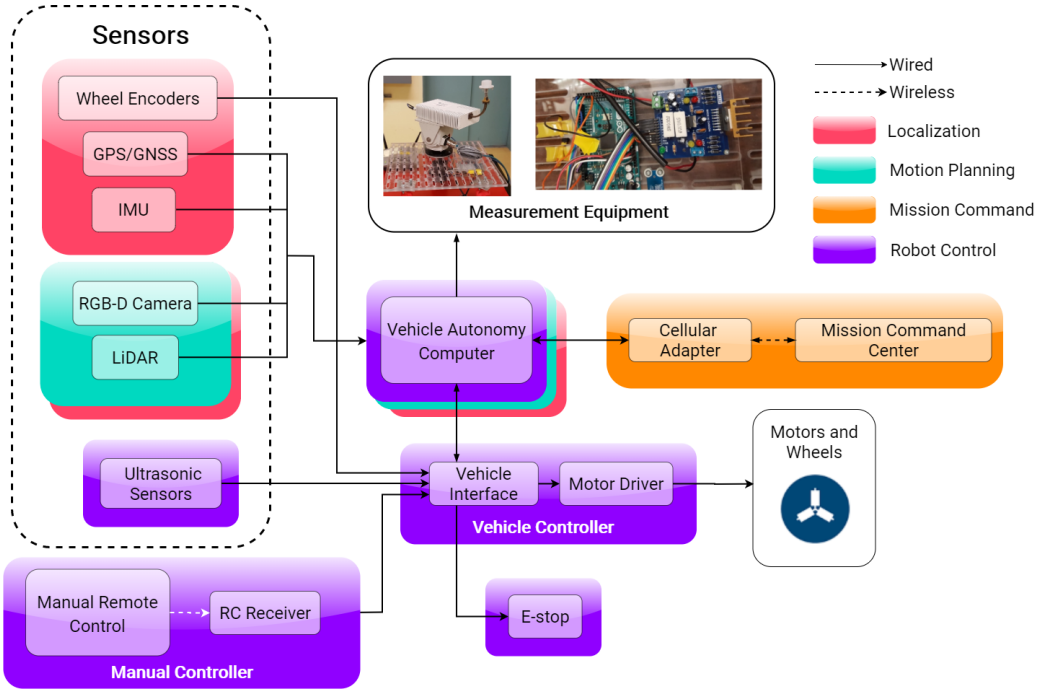


Figure 2.1: Proposed hardware architecture

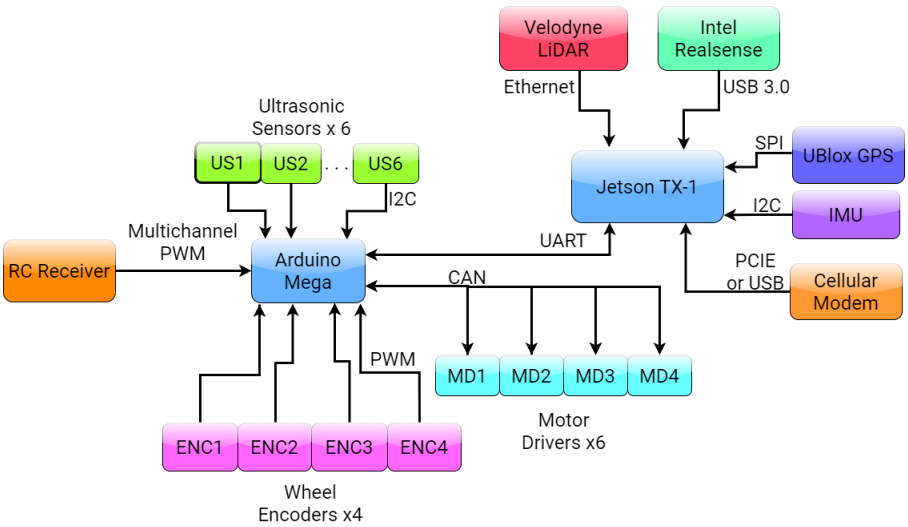


Figure 2.2: Communication protocol for wired communication

2.2 Component Decisions

For all decided components, descriptions of the sensors and reasons for the decisions will be provided in the proceeding sections. Some components have still not been decided upon. These components include a mechanical brake, the autonomy hardware mounting system, wire harnesses, and an encoder mounting system, which all belong to the category of general mechanical components. All the undecided components are not critical to the early system design as they mostly pertain to field testing which will occur later in the project. Please refer to Appendix C for more information on the undecided components.

Based on all the decided hardware components, a power consumption budget is provided in Table 2.1. Each battery on the robot supplies 540 Watts per hour. According to the power budget, four batteries will be needed to satisfy the operation duration requirement of 1.5 hours (NF1.2). Currently only two batteries are installed on the robot so it is recommended that two more be added in the future.

Hardware Components	**Power/Unit (W)**	**Quantity**	**Total (W)**	**Reference**
RC Receiver	0.1	1	0.1	[12]
GPS	0.1155	1	0.1155	[13]
GPS Antenna	2	1	2	[14]
IMU	0.0072	1	0.0072	[15]
Wheel Encoder	0.245	4	0.98	[16]
Ultrasonic Sensor	0.017	8	0.136	[17]
LiDAR	8	1	8	[18]
RGB-D Camera	3.5	1	3.5	[19]
NVIDIA Jetson TX1	15	1	15	[20]
Vehicle Interface	0.34	1	0.34	[21]
Motor Driver	480	4	1920	[22]
RF Equipment	130	1	130	[23]
Total	2080.18 W			

Table 2.1: Power consumption budget

2.3 Localization and Motion Planning

To provide accurate position information, a combination of a GPS (Global Positioning System) module, wheel encoders, IMU (Inertial Measurement Unit), LiDAR (Light Detection and Ranging), and RGB-D (Red Green Blue-Depth) camera data will be used. Fusing this data increases the accuracy of the robot's position estimate. The combination of these sensors will ensure that we can meet the requirement of sub-meter position accuracy (with the use of a SLAM (Simultaneous Localization and Mapping) algorithm. In addition, the use of all of these sensors provides for a better map output and obstacle detection that can be used for motion planning (than using just one sensor). The hardware selection of these sensors is described in the following sections.

Sensor Suite Justification

Lidar

A LiDAR sensor allows the robot to map its surroundings with reliable range data, and in varying

ambient light conditions. In combination with a GPS, and an IMU, a high resolution LiDAR scanner allows the autonomy computer to perform SLAM and obstacle detection with less processing power on the compared to systems with low resolution lidar which will require more computation to achieve similar range detection accuracy. It also provides an alternative to using solely cameras for object detection, as deterministic methods that do not involve any machine learning can be employed to detect obstacles.

GPS and IMU

Initially, off-the-shelf integrated GPS/GNSS (Global Navigation Satellite System) and IMU options were researched to provide accurate robot localization in an effort to meet our time constraints. Devices such as the VectorNav VN-200, Novatel SPAN CPT7, and Duro Inertial Ruggedized Receiver were compared, all of which cost between 3000-8000USD.

One of the features that these systems often implement is dynamic alignment. This occurs when sufficient motion allows the system to estimate heading based on the correlation of IMU acceleration measurement and change in velocity measured by the GPS receiver [24].

However, further research showed that slow moving platforms, such as the PropBot under the set ODDs (Operational Design Domain), do not experience sufficient acceleration to perform dynamic alignment. Due to this, dynamic alignment is not a required or useful feature for the PropBot system.

An alternative to this is using a magnetometer for heading measurement, but this is only feasible when the magnetic environment can be controlled by the user. In the case of PropBot, the robot will be hosting different types of lab equipment, and the robot will need to drive through different types of environments.

Thus a method to infer heading without dynamic alignment and magnetometers will need to be used. This can be done with SLAM algorithms with the help of other sensors such as LiDAR, RGB-D, and wheel encoders. These SLAM algorithms use different sensors as inputs to a filter. Due to this, it is important that the sensor data is unfiltered, as the filter used in SLAM algorithms will not perform as well if the inputs are already filtered. Due to this line of reasoning, off–the-shelf integrated GPS/GNSS and IMU options are not feasible because they provide filtered position outputs and are significantly more expensive than buying GPS and IMU units separately. In addition, using just a GPS/IMU solution for positioning is not ideal under GPS loss conditions because IMU sensors on their own can build up a lot of errors over time.

RGB-D Camera

Cameras are highly effective in classification and object tracking. As deep-learning fueled vision algorithms improve, the robustness of dynamic obstacle detection with cameras will continue to improve, and thus will be very beneficial to the robot's autonomy system. However, it does have the weakness of requiring non-deterministic algorithms to identify these obstacles, so it is beneficial to pair this sensor with others for motion planning.

Safety Feature 5 (Multiple localization and motion planning sensors)**.** *The use of multiple localization and motion planning sensors ensures that there is no single mode of failure when the robot is detecting its surroundings. The use of multiple sensors of situational awareness decreases the likelihood of robot collisions with the environment and allows for the use of advanced algorithms in the future.*

2.3.1 LiDAR

Final Choice: `Velodyne Puck`

Selection Criteria

The required features for the LiDAR are described in Table 2.2.

Selection Justification

For a detailed trade study, please refer to Section 4.1.4

The sensor configuration which meets the requirements stated in Table 2.2 while still minimizing cost and maximizing ease of integration is the **Velodyne Puck**. Mounting a single Puck at the front of the robot reduces the complexity of the SLAM and object detection/response algorithms and has a lower processing power requirements compared to more complex LiDAR setups. The Puck is widely used in mobile robotics projects so its specifications have been validated by many research groups who have also provided support in simulation software such as Airsim, CARLA, and other Gazebo . The Puck also supports multi-echo-technology which mitigates the interference from rain/fog/dust without loss in performance.

A drawback of this LiDAR configuration is that it does not provide a 360 degree view around the robot. This can be problematic when the robot makes sharp turns which increases the risk of colliding with moving objects approaching the robot's side zone (i.e. cyclists). However, this can be mitigated in the future by installing safety blinkers to alert nearby road users at least 2 seconds before the robot is about to make a turn.

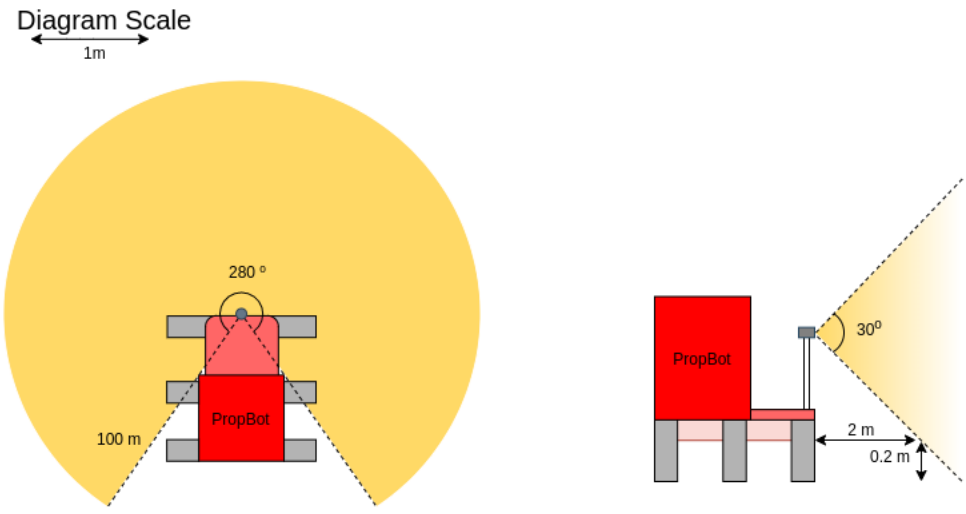


Figure 2.3: Horizontal and vertical field of view of LiDAR configuration

2.3.2 GPS

Final choice: `Ublox NEO-M8P-2`

Selection Criteria

A potential solution to achieving high precision position accuracy is using a custom differential GPS system. Differential GPS adjusts real time GPS signals using a fixed known position [25].

Related Tag(s)	Required Feature	Justification
O4.2	LiDAR shall have high sunlight resistance.	Intense sunlight can prevent scanner from being able to read its own returning light pulses.
O4.1	LiDAR shall support hardware or software methods to mitigate interference from rain/fog/dust	Precipitation can interfere with LiDARs ability to detect obstacles by reducing the intensity of the signal reflected from the target or by causing false positive detections.
O4.1, O4.2	LiDAR shall have built-in temperature control system and operating range between -20°C to 50°C	Varying the temperature will vary the Wavelength of semiconductor laser. This will result in inaccurate detection of the returning laser beams
O4.1	LiDAR shall have environmental rating of at least IP67 (protection from dust and water)	Robot will be in operating outdoors in various weather conditions.
F1.2	Sample rate 200000 points/sec	Capturing higher amounts of datapoints results in more reliable mapping, localization, and moving object detection.
F1.2, O2.1	LiDAR shall have distance range $>$ 40m	Robot travels at a maximum speed of 10km/h (4m/s).
F2.1, F2.2, F2.3, NF2.1	LiDAR shall have Horizontal FOV of $\geqslant$ 270 degrees	Robot should be able to track objects in its frontal and side zone.
F2.1, NF2.1	LiDAR shall have Vertical FOV of $\geqslant$ 30 degrees	LiDAR will be mounted on the robot at least 0.75m above ground in order to create a comprehensive scan of its environment for wave propagation data analysis. The robot should be able to detect an object of 0.2 m height (height of small dog) from 2 m away.
NF1.2	Power consumption should be $<$ 20W	Minimize power consumption to achieve desired operation time.

Table 2.2: Required features of LiDAR

Some GPS chips (Ublox NEO-M8P) even allow the base station to move and calculate the relative position of the moving "rover" to a sub-cm accuracy. The following are the pros and cons of using a differential GPS system for PropBot.

Custom Differential GPS Station

Pros

- Provides up to cm level accuracy.

Cons

- Will not work when robot loses GPS signal.
- Will not provide too much added benefit in comparison to Real-Time Kinematic positioning systems [26].
- Base station GPS will require power.
- Base station GPS will need to be mounted in open sky area and will need casing to shield against the elements.

This type of system would add complexity to PropBot and make it reliant on base stations for its navigation. Since PropBot needs other sensors such as LiDAR and camera for autonomous decision making, it makes sense to use these sensors to aid in localization with normal GPS inputs. The localization algorithm that is to be used is described in a later section and boasts up to a 5cm position accuracy level.

Due to this, differential GPS systems will not be considered, but Real-Time Kinematic (RTK) GPS systems will be considered.

The required features of the GPS are described in Table 2.3.

Selection Justification

For a detailed trade study, please refer to Section 4.1.2

The **Ublox NEO-M8P-2** has by-far the best position accuracy performance and is the only system that has an option to use RTK.

2.3.3 IMU (Inertial Measurement Unit)

Final choice: `MPU-9250`

Selection Criteria

The required features of the IMU are described in Table 2.4.

Selection Justification

For a detailed trade study, please refer to Section 4.1.3

The **MPU-9250** is the best option because its accelerometer's range is 2g, which is around $20 m/s^2$ Since the robot doesn't move that fast or increase its speed in a sudden due to the maximum speed limit of 10 km/h (around 2.78 m/s), that range is more than enough for the PropBot. More importantly, in comparison with the other IMU option, **MPU-9250** has an open-source ROS driver.

Related Tag(s)	Required Feature	Justification
NF3.2	GPS system should have metre level accuracy and us GNSS.	According to the related tags, position data must be accurate to 0.5m (NF3.2), and at least metre level GPS accuracy will be required for the robot's localization algorithms.
NF3.2	GPS system should be a standalone device (no IMU) that does not do internal filtering on the position.	Due to the use of a localization algorithm that uses multiple inputs, the position estimate must be unfiltered for optimal results.
O4.1	GPS system should have casing that shields it from light rain, or allow for a plastic case to be easily implemented for an existing circuit board.	Due to time constraints and environmental conditions, a prepackaged GPS is needed that at most requires additional plastic casing to be made.
NF3.2	GPS system should have a time-pulse (PPS) signal.	A time-pulse will mitigate the computer's clock drift and ensure that the collected data is time stamped accurately.

Table 2.3: Required features for GPS

Related Tag(s)	Required Feature	Justification
NF3.2	IMU system should be a standalone device that does not do internal filtering on the position.	Due to the use of a localization algorithm that uses multiple inputs, the position estimate must be unfiltered for optimal results.
O4.1	IMU system should have casing that shields it from light rain, or allow for a plastic case to be easily implemented for an existing circuit board.	Due to time constraints and environmental conditions, a prepackaged IMU is needed that at most requires additional plastic casing to be made.

Table 2.4: Required features for IMU

This will satisfy the above mentioned drivers and facilitate the integration between the IMU and the autonomy computer.

2.3.4 RGB-D Camera

Final choice: `Intel Realsense`

Selection Criteria

In order to select an appropriate camera for forward vision many different vendors were considered. As well the existing cameras available such as the Intel Realsense were considered to see if the currently available resources are already sufficient.

The main considerations for selecting the camera are the camera's connection interface and its imaging/sensor characteristics. The Jetson TX-1 is fairly limited in IO, particularly in USB 3.0 ports, in which only one exists. As a result cameras with other connectivity options like CSI-MIPI or Ethernet are preferred. The required features of the camera are described in Table 2.5.

Related Tag(s)	Required Feature	Justification
O2.1, O3.2, O3.4, F1.2	Global shutter is preferred over rolling shutter.	Global shutter reduces imaging artifacts for fast moving objects. Less artifacting improves object detection.
F2.2, NF2.1	Resolution should be 720p or 1080p	Low resolution makes it hard to resolve detail at long distance.High resolution uses too much processing power. Low resolution sensors usually have bigger pixel sizes, which provides better low light imaging.
F2.3, EF3.3	Wider FOV (but not too wide) is desired. Minimum of 60 ° HFOV.	Larger field of view allows the robot to see objects coming from the sides sooner. Very high FOV lenses (fisheye) distort image, which is undesired for object detection.
O4.1	Waterproof and/or rugged casing is desired	Pre-made waterproofing and rugged cases would allow for more flexible mounting positions. Removes the need to design custom waterproof casing which can be complex for cameras [27]

Table 2.5: Required features of camera

Selection Justification

For a detailed trade study, please refer to Section 4.1.5

It is important to note that our client has already provided us with a Intel Realsense camera. The Intel Realsense over has the second best sensor characteristics mainly due to the inclusion of global

shutter. The depth data can also be used for localization. Due to its high performance and ability to be used for localization, the **Intel Realsense** is the best option.

A con of the Intel Realsense is the poor horizontal FOV of 70 degrees especially since the Leopard Imaging has a 92 degree horizontal FOV lens with 1% distortion. The low horizontal FOV means PropBot needs to rely more on LiDAR and other sensors to detect objects coming on its sides.

2.4 Robot Control

Figure 2.4 shows all the hardware elements that are involved in the robot control subsystem. This section will elaborate on all these hardware elements.

Robot control involves receiving motion information from a source (RC or autonomy computer), and converting the information into a motion intention (PWM signals, CAN commands, RS-232 messages, etc.) that will be sent to the motor drivers. All of this must occur while considering inputs from the close-range sensor network and the E-stop (which is again a hardware zero speed or brake command assertion mechanism, not a power-off switch). All the information is ingested by the vehicle interface (one or more control devices) and then sent out to the various peripheral devices. The interface acts more like a command relay system, with a bit of extra functionality.

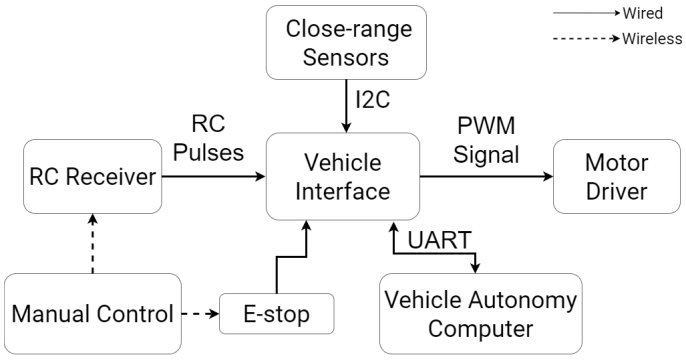


Figure 2.4: Simplified Robot Control System Diagram

2.4.1 Vehicle Interface

Final Choice: `Arduino Mega`

Selection Justification

For a detailed trade study, please refer to section 4.2.3

The need for either drivers with integrated processors or a separate device for vehicle interfacing is motivated by the ability to have the autonomy side be hardware agnostic. Although the RoboteQ has integrated control electronics and can be daisy-chained to create a connected system, writing custom code for the RoboteQ drivers eliminates firmware portability. To ensure hardware decoupling and firmware portability and scalability, the decision was made to add in a separate controller: the "Vehicle Interface". The simplest option is an Arduino device due to its affordability, availability, and ease of use. Further, switching between various Arduino platforms requires little to no changes.

Bi-directional interfaces

Table 2.6 is an overview of what the vehicle interface communicates with and how

Item	Interface	Details
Autonomy computer	USB (UART)	USB 2.0 or 3.0
Remote controller	RC pulse	2.4GHz, 6 Channel
Motor drivers	CAN	requires a CAN shield

Table 2.6: Overview of Vehicle Interface Communication

Vehicle Interface and Motor Driver Communication

The current motor drivers operate purely via PWM commands. The newly selected motor drivers - RoboteQ SBL1360A (see section 2.4.2) - 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 mostly 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.

2.4.2 Motor Drivers

Final Choice: `RoboteQ SBL1360A`

Selection Criteria

Currently, the system utilizes 24V/15A motor driver that is untraceable and has no datasheet online. We are assuming those motor drivers are BLDC (Brushless Direct Current) based on the similar motor drivers we find online [30].

Given that the complexity of motor control will affect our approach to the rest of the self-driving system, it is necessary to move to the motor drivers that have established support and possess

extensive functionality: tachometer, encoder integration, commutation sensor feedback, support for multiple communication protocols, daisy-chaining support, etc. . Although implementing four such optimal motor drivers may cost over 1000 CAD, we believe it would significantly decrease the time to achieve safe non-autonomous driving capabilities and allow for more focus on the self-driving platform.

Related Tag(s)	Required Feature	Justification
O2.1	Accurate speed control	As this robot is in initial stages, the speed is limited to 10km/h to ensure the safety of robot and other road users.This speed will allow for the robot user to achieve a small stopping distance during manual control, and easily hit the onboard E-stop if needed.
F2.1	Robot shall perform complex maneuvers (e.g. turn, decelerate, stop)	The robot must be able to perform the required maneuvers in order to appropriately navigate and avoid obstacles.
F5.3	Robot shall facilitate longitudinal motor set control via remote communication	The robot must be able to take commands from a remote controller. A user must be able to have control over the motion, so the longitudinal motor sets must be able to move independently.

Table 2.7: Required features of motor driver

Given these requirements, we are now looking for either standalone motor drivers or a system of motor drivers that can be daisy-chained together to facilitate complete control of the robot. Further, cheaper motor drivers could be integrated into the system for the non-motion motors to improve braking by either de-energizing or energizing the braking system. The requirements of the motor drivers are listed in Table 2.7.

Selection Justification

For a detailed trade study, please refer to Section 4.2.2

Among all the options, the **RoboteQ SBL1360A** best fits all the requirements outlined for the motor driver.

In addition, it requires no additional controller like a standalone motor driver would given that the motor driver can be daisy chained together:

> The SBL1360A accepts commands received from an RC radio, Analog Joystick, wireless modem, or microcomputer. Using CAN bus, up to 127 controllers can be networked on a single twisted pair cable. Numerous safety features are incorporated into the driver to ensure reliable and safe operation. [31]

Further, it is an intelligent system, and so the functionality and visibility of the system can be

customized: (Note: the "controller" in the quotation below refers to the motor driver).

> The controller's operation can be extensively automated and customized using Basic Language scripts. The driver can be configured, monitored and tuned in real time using a RoboteQ's free PC utility. The controller can also be reprogrammed in the field with the latest features by downloading new operating software from RoboteQ. [32]

2.4.3 Power-Off Switch

Currently, a hardware switch connects the entire electrical system to the battery bank. In the unlikely event that power must be cut to the entire, the switch can be flipped to cut power to entire system, which acts as a **power lockout assertion mechanism**.

Figure 2.5: Existing power-off switch

Related Tag(s)	Required Feature	Justification
F4.1	The fallback ready user shall be able to kill power to the system via power-off switch.	In an extreme case, the switch can be flipped to cut power to the entire system.

Table 2.8: Required features of power-off switch

Safety Feature 6 (Power-off switch)**.** *In an electrically powered system, it is necessary to have a power-off switch for extreme situations where power lockout is required, e.g. battery malfunction, major electrical short, fire*

2.4.4 E-stop

Figure 2.6: Selected E-stop button

To halt speed commands to the motor drivers, an emergency stop button will be used. The vehicle interface will be interrupted and shall put the system into "coast" in which no speed commands are sent. Provided the legacy motor drivers are swapped for ones that are fully functional, a "brake" command could be integrated (braking via inverting the wheel motor electromotive force (EMF)) and would replace "coast" in the E-stop context.

Additionally, a remote halt option is available via a switch on the remote controller. Again, this will not cut power to the motors, but will put the wheel motors into either "coast" ("brake" desired).

Related Tag(s)	Required Feature	Justification
F4.2	The fallback ready user shall be able halt motion commands to the robot via e-stop.	In the event of an emergency, the button can be pressed to stop sending motion commands to the system.

Table 2.9: Required features of e-stop

Safety Feature 7 (Highly visible on-board E-Stop)**.** *In order for the E-Stop to be easily accessible for road users, a very large and visible button was chosen, as shown in Figure 2.6.*

2.4.5 Close-Range Sensors

Final choice: `6 x MB1202 MaxSonar, 2 x MB7040`

Selection Criteria

The main purpose of the close range sensors is to provide a final fallback collision avoidance mechanism for the robot. The requirements for the close-range sensor are described in Table 2.10. The minimum requirements are fairly low due to the large variety of sensors with different characteristics.

Related Tag(s)	Required Feature	Justification
F2.1, NF2.1	Detects long distance. Minimum 2 meters.	Longer range gives the robot more distance to react and stop.
F2.1, F2.3	Detects Wide Arc. Minimum 30°.	A wider arc reduces the blind spots of the robot.
NF2.1	Good Resolution/Accuracy. At most 10 cm.	Should be accurate enough to be useful. But the main goal is not distance estimation.
O4.1	Weather proof. Minimum IPX3.	Allows the sensor to be placed in more optimal positions. No need to design a case.

Table 2.10: Required features of close range sensor

Selection Justification

For a detailed trade study, please refer to Section 4.2.1

Overall most sensors go far beyond our requirements for accuracy as the close-range sensors are used to detect whether an object is within a certain threshold distance rather than using its exact range values. Additionally the high sample rates of 50-60 Hz are not necessary, as PropBot moves slowly (max speed is 10km/h).

The sensor that is the best option is the MaxSonar. While it has high $100 price tag, no other sensor comes close in terms of detection range. It also has a good variety of digital interface options for easy integration and completely waterproof casing. The main disadvantage is the low 10Hz sample rate, but given the long 10m range and Propbot's slow speeds, it is likely to be sufficient: given that the robot drives at a max speed of 10km/hr (2.8m/s) and stops 1 second after the stop command is given (as defined by requirement NF4.1), the sensors should trigger when the robot is at least 5m away from a obstacle (accounting for some latency in the braking system). This indicates an update rate which is larger than (2.8)/(10-5) = 0.56Hz is adequate.

From the comparison of sensors it shows that the MaxSonar line of ultrasonic sensors have the best performance characteristics. There are many models of the MaxSonar with different use cases. To choose the appropriate sensors we analyzed their usage on PropBot which is for a fallback to avoid collision.

The most desired characteristic for this use case is a wide coverage area. This is where the **MB1202 MaxSonar** provides the best performance with its wide beam. These wide beam sensors can be placed around the robot to provide 360 degree coverage. Additionally, two **MB7040** are added to do more accurate range prediction for the front and back of the robot. This is necessary as the robot primarily moves in these directions. Due to the wide beam of the MB1202, it may produce more false alarms, thus having a narrow beam sensor lets the robot more accurately see the free space in front.

Figure 2.7 shows the optimal sensor mounting for full coverage around the vehicle. It is important to note that the beam width and range values here are fairly conservative as the datasheet values for detection of very small objects (1 inch diameter dowel). Larger objects such as humans or vehicles can be detected at greater distances.

Model	Range	Beam Width	Connection
MB1202	4.0m	1.8m	I2C
MB7040	7.5m	0.6m	I2C

Table 2.11: MB1202 and MB7040

Safety Feature 8 (Close-range sensor coverage dependent on direction of motion)**.** *The use of multiple different close-range sensor with different ranges allows for ensures that there is full coverage and there is a larger range in the direction of motion (forwards and backwards).*

2.4.6 Manual Control

Final Choice: `FS-T6`

Selection Criteria

The selection criteria for the remote controller is described in Table 2.12.

Selection Justification

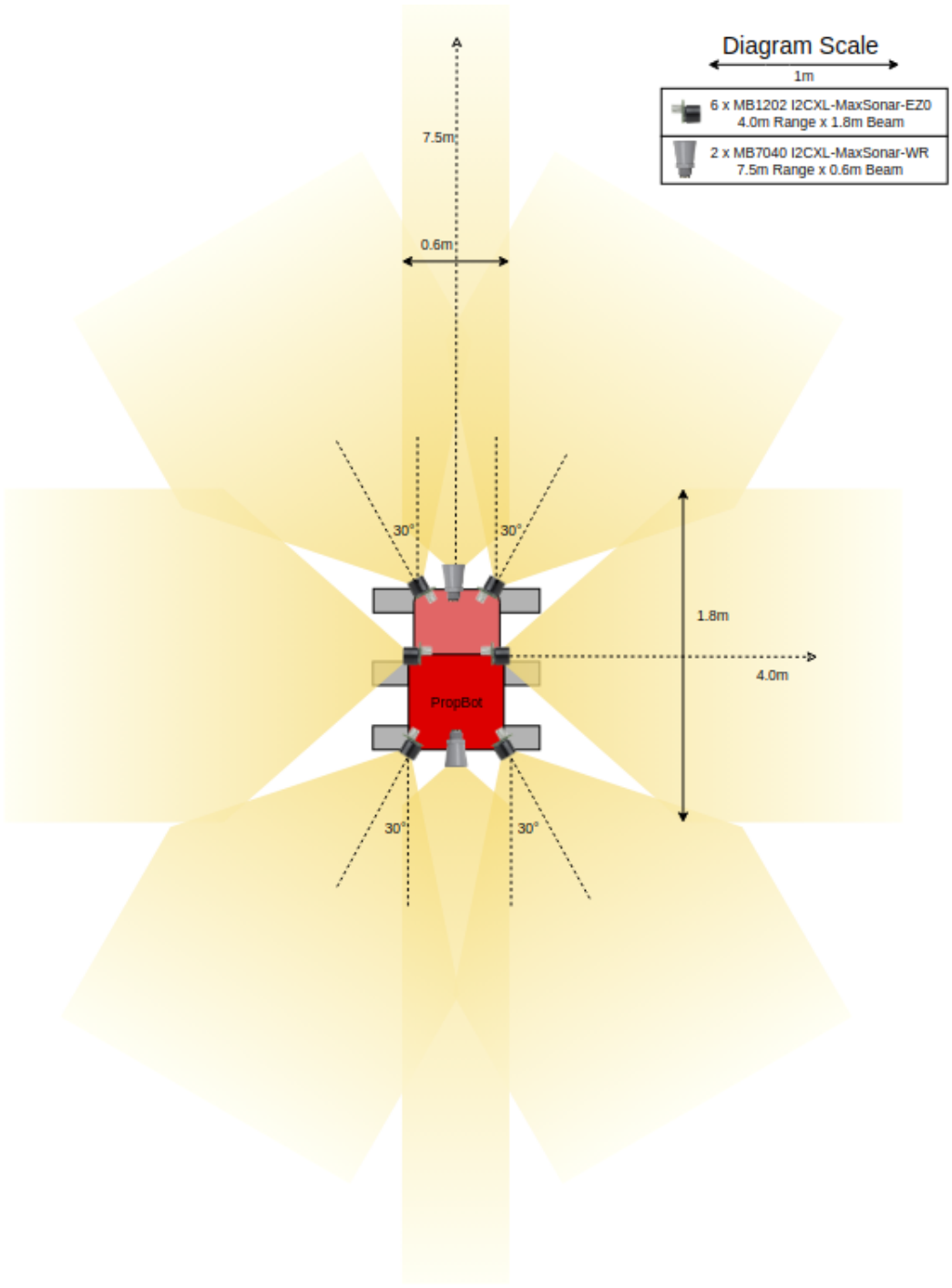


Figure 2.7: Ultrasonic sensor mounting and coverage

Related Tag(s)	Required Feature	Justification	Specification
F5.1, F5.4	The E-stop button on the PropBot is remotely linked	Users do not need to get to the PropBot and press the E-stop button in emergency cases.	The manual control device shall have some programmable buttons or switches.
F5.3	Robot shall facilitate longitudinal motor set control via remote communication	The two sets of wheels move differently when the PropBot is making a turn.	The manual control device shall have two joysticks.

Table 2.12: Required features of manual controller

There are three types of remote controllers: infrared, voice, and radio frequency (RF). The infrared remote controller is unreliable in the sunlight and the voice remote controller can be easily affected by other non-users near the robot.

The RF remote controller is the best option. We want to use the remote controller to operate four basic functions, which are:

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

Hence, the remote controller should have at least four independent channels. The **FS-T6** is a six-channel transmitter with two joysticks and four programmable switches. This provides a joystick for each set of wheels and programmable switches for a remote kill switch and a autonomy to manual mode switch. Figure 2.8 demonstrates how the four functions mentioned above are distributed on the transmitter. The remote controller receiver is **FS-R6B**, which is a six-channel receiver.

Safety Feature 9 (Remote E-Stop on remote controller)**.** *The remote E-stop (Figure 2.8 on the RC transmitter removes the need to engage the E-stop on the robot. This allows for the robot user to quickly stop the robot from a distance in the case of an unexpected obstacle or scenario.*

Figure 2.8: FS-T6 transmitter

3 Software Architecture

3.1 Overview

Our proposed software architecture heavily relies on our decision to use Robot Operating System (ROS) as our software framework.

The benefits of ROS are highlighted as follows [6]:

- Allows for modular software design
- Allows for easy integration of existing open-source packages.
- Allows for collaboration with the international community
- Allows software to be executed in a distributed system, thus balancing the computational requirements at different processing elements and PCs. This is especially relevant to the processing power limited by the Jetson TX1 in C1.4.

Safety Feature 10 (ROS distributed system)**.** *The use of ROS ensures that if one "node" or software process dies, the other systems can still continue functioning. This ensures that the robot can continue operating safely and in a predictable manner in the event of software process failures. In addition, this means that all information (such as maps) are retained even if other processes die.*

The alternative to this would integrating an existing message passing layer into a custom software stack. This would add considerable complexity and take more time to implement.

The proposed software architecture for our ROS packages is shown in Figure 3.1. This figure highlights all of the custom Propbot packages that are added to carry out localization, motion planning, mission command, and robot control functions. It is important to note that unlike Figure 1.1, this figure does not include any software related to dynamic object detection. The integration of this feature is not completed, and was chosen to be completed in the future because it is difficult to integrate into a simulation environment. However, a brief overview of our standalone dynamic obstacle detection prototype is provided in Section 3.4.3.

The packages were split up in this manner to modularize our software. This allows for future developers to easily switch out specific algorithms and libraries within the "Propbot" packages to improve the capabilities of the autonomy system. The software is modular and complies with the general ROS architecture (Figure B.1 in Appendix B in mind, so that the open source ROS SLAM, navigation, and control packages can be used.

The general functionality of the packages running on each computer (Mission Command and Vehicle Autonomy) are in the diagram are described below and their specific implementation details are described in the proceeding sections:

Mission Command Computer:

- **Propbot GUI:** This package has the GUI that a user can use to create a mission and monitor the robot's mission progress. This GUI will allow a user to set waypoints on a map, and send them to Propbot Mission. In addition, it reads in "Robot Pose" (the robot's position and orientation) from *Propbot SLAM* and displays it.

Vehicle Autonomy Computer:

- **Propbot Mission:** This package takes in a mission and sends goals to *Propbot Navigation*. Using the "GPS to Robot Map Transform" from Propbot SLAM, 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 *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 the figure. 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 received 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.

In addition to the packages above, we have software for the vehicle interface as mentioned in Section and Section (described in Section) and software used to for simulation of the robot (described in Section). Our *Propbot* repository is split into `setup`, `vehicle_autonomy`, `mission_command`, `simulation`, `vehicle_interface`, directories to compartmentalize the software used on different processors.

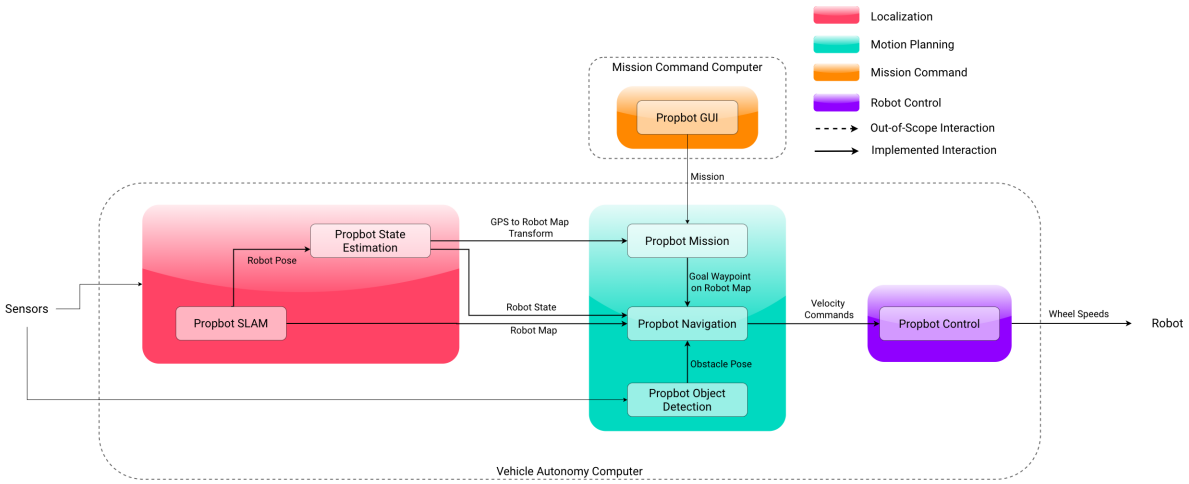


Figure 3.1: Propbot ROS package software architecture

3.2 Software Decisions

For all decided software (except drivers), descriptions of the software package and the reasons for choosing it will be provided in the proceeding sections. Some components have still not been decided upon as they depend on hardware components or do not depend on the design of the rest of the system. Please refer to Appendix C for more information on the undecided components.

3.3 Localization

3.3.1 SLAM Package

Final Choice: `Google Cartographer`

Selection Criteria

In order for PropBot to be useful to researchers, it needs to be able to record high accuracy location data. We have defined this accuracy to be 0.5m as the wireless technologies tested by the Radio Science Lab are sensitive to obstruction from the environment. This allows for the measurements made by the research payload to have accurate data about where measurements were taken. Table 3.1 outlines the required features of the localization software module and Figure 3.2 shows an overview of our final choice.

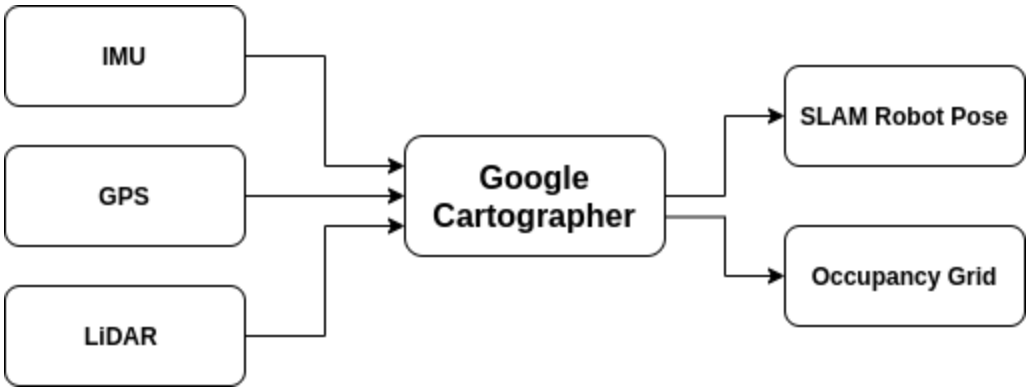


Figure 3.2: Overview of SLAM with Google Cartographer

Related Tag(s)	Required Feature	Justification
F3.1, NF3.1, O6.5	Position estimate based multiple input sensors, rather than just GPS.	According to the related tags, position data must be accurate to 0.5m (NF3.1) even in an interference zone (O6.5) where a GPS signal is not present. This means additional sensors such as wheel encoders, LiDAR, camera, and IMU should be inputs to the position estimate when GPS is not present or if GPS position is not accurate to 0.5m on its own.
F3.1, NF3.2, C1.3	Position estimate can be generated at least 1Hz.	Position estimate loop needs to be able to run at least 1Hz to ensure NF4.2 is satisfied. The localization module needs to be quick enough and able to handle the various sensor inputs on the Jetson TX1 (C1.3).

Table 3.1: Required features of localization

3.3.2 State Estimation Package

Final Choice: `robot_localization`

Design

The `propbot_state_estimation` package utilizes components from the ROS `robot_localization` package to fuse a number of input sensor data and compute the nonlinear state estimations for robots moving in 3D space. Currently `robot_localization` is the only ROS package to provide an implementation of the Extended Kalman Filter(EKF). An overview of `propbot_state_estimation` package is provided in Figure 3.3. The package's components are further described below:

- **EKF Localization (Odom):** This state estimator node accepts a variety of continuous

pose-related data and outputs a continuous robot state estimate that is locally accurate but is prone to drift over time. While this estimate cannot be used to determine the absolute state of the robot for global path planning, it is useful to the robot controller which uses this frequently updated pose information, to adjust the kinematic trajectory of the robot.

- **EKF Localization (Map):** This state estimator node accepts a variety of discrete pose-related data, as well as the state estimate produced by the Odom instance of this node to compute a continuous globally accurate robot state estimate.
- **NavSat Transform:** This node NavSat data transform node accepts GPS position data and a continuous robot state estimate from the map frame (output of the Map instance of the EKF localization node) to calculate a transform between the GPS data in the WSG84 frame and the robot state in the map frame in order to appropriately incorporate the GPS data into the robot state estimate.
- **Visual Odometry:** Visual odometry uses RGB and Depth data to build maps that estimate the robot's local position and velocity in locally accurate frame (odom). Since the SLAM package Cartographer does not use RGB data to output the 'SLAM Odom Pose' as show in Figure 3.3, we use visual odometry to use RGB data to improve the robot's state estimate in the odom frame. This estimate is fed into the Odom EKF filter for state estimation.
  - **Final Choice:** `rtab_map`
  - *Selection Criteria:* The selection of this package is based on compatibility with our chosen vision sensor which is the Realsense RGBD camera. This package provides an improvment to our state estimation methods, but there are no specific performance requirements that are pertinent to the project. The visual odometry package must provide state estimations in the odom frame, and shouln't do full SLAM (involving loop closure, mapping) because this would expend an unnecessary amount processing power.
  - *Selection Justification:* We specifically look at packages with existing ROS integrations that do purely localization (or have the option to do so) and support RGBD cameras. The available remaining options are `rpg_svo`, `dso_ros`, `ros_mono_vo`, and `viso2_ros`. Out of all of these packages, the only package that has thorough documentation, is maintained, and compatible with the newest version of ROS (melodic) is `rtab_map`. We choose `rtab_map` because it allows for easy integration, low risk due to regular maintenance, and has a dedicated pure localization node.

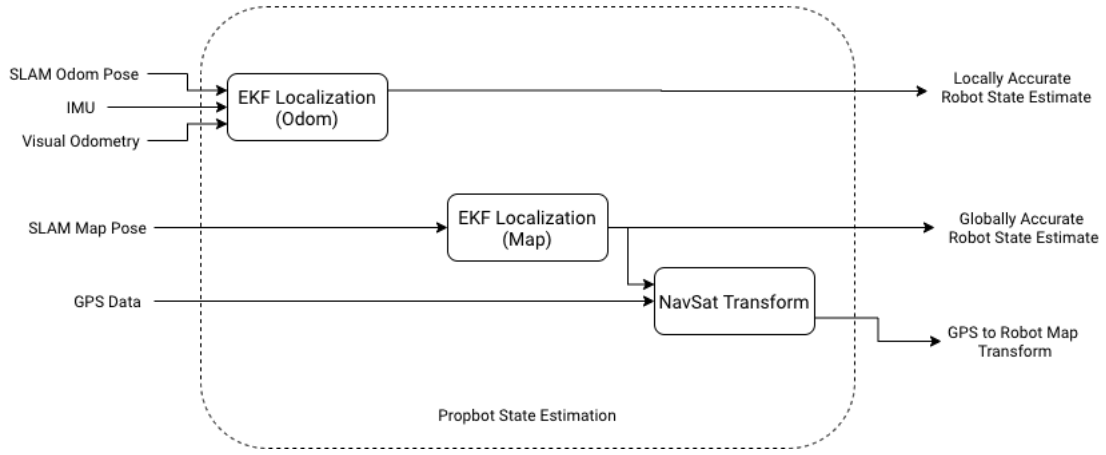


Figure 3.3: Overview of Propbot State Estimation Package

Selection Justification

For a detailed trade study, please refer to Section 4.1.1.

Many of the SLAM libraries that were found online only made use of an RGB-D camera or a monocular camera. We eliminate these as options for SLAM as we would like to achieve the best position estimate by using all of the sensors on-board.

All of the explored packages satisfy our required features, but **Google Cartographer** is chosen due to its usage of the most different input sensors, highest accuracy, and continued support with ROS. Figure 3.2 shows an overview of the Google Cartographer package. It takes in sensor data and outputs Robot Localization (Robot Pose) for *Propbot Mission* and a Robot Map for *Propbot Navigation*.

3.4 Motion Planning

3.4.1 Propbot Mission

Final Choice: `Custom Library: propbot_mission`

Selection Criteria

This package must be able to execute a mission that is created by a user. The specific requirements for this package are listed in Table 3.2.

Selection Justification

The only publicly available open-source package that had the GPS waypoint capability mentioned in Table 3.2 is the package `waypoint_nav` [33]. This package is able to accept a text file full of GPS coordinates and synchronously send them to the robot's navigation stack. This package sends a waypoint, waits for it to be reached, and then sends the next one. This is a blocking process, and the only way to stop the mission is to kill all the processes on the robot. So, this package does not allow for missions to be started, stopped, and paused at any time. In addition, this code had poor styling, was not modular, was difficult to read, and did not allow for expansion of features.

Related Tag(s)	Required Feature	Justification
F1.1, NF1.3, F4.3	Package converts GPS coordinates of a mission to robot map waypoints and sends them to the navigation stack.	The user will set GPS waypoints using a GUI, so this package must be able to take in GPS waypoints and convert them to goals that the robot can interpret. .
F1.3, NF1.3, F4.1	Package monitors mission progress (which waypoint the robot is travelling towards).	The user should be able to monitor the robot's mission progress, so this package should be able to keep track of the robot's progress. .
F1.3, F4.2, NF4.1	Package allows the mission to be started, paused, and stopped at any time.	For safety purposes, the robot should be able to stop its mission at any time if commanded. In addition, to improve the user experience and safety, the mission should also be able to "pause", such that it can be restarted after an interruption (related to the user or an external factor).
F1.1, F2.5, NF1.5	Package should be able to receive a mission from *Propbot GUI*.	The package must be able to interface with the GUI, so a user can create a mission.

Table 3.2: Required features of Propbot Mission package

This is why we have implemented our own `propbot_mission` library to satisfy the requirements for this package. The design is described below.

Design

The propbot library architecture and functionality is shown in Figure 3.4. The classes present in this library are *Waypoint, Mission, and MissionHandler*. These classes were made with future developers at RSL in mind, as they have generic interfaces between each other that still allow for further improvement or added functionality in the future. The propbot mission library uses the Simple Action Client library by ROS to communicate with *Propbot Navigation*. It uses asynchronous communication to submit and cancel goals, and receive goal statuses. We chose to use asynchronous communication because of the reasons mentioned in the following Safety Feature.

Safety Feature 11 (Asynchronous mission execution)**.** *The class that handles mission execution communicates with Propot Navigation, thus allowing for the the mission to be started and stopped at any given time. The use of this type of communication allows for a safe, clean, and predictable end to a robot's mission.*

If synchronous communication was used, killing all of the robot processes would be the only way to end the mission, and the exit behaviour would not be predictable. The exact functionality of the classes are further described below:

- **Waypoint:** This is a container class that holds different waypoint types and handles the conversion between them. This class is constructed with a GPS coordinate, and it stores this information. This class uses the "GPS to Robot Map Transform" to convert its GPS

coordinate to a waypoint on the Robot Map coordinate. This is used within the Mission Handler class. This satisfies the requirement in Table 3.2 of being able to take in GPS waypoints and convert them into waypoints on the robot map. Since the `Waypoint` class is a container class, it can be easily expanded upon in the future to include other types of coordinates.

- **Mission:** This is a container class that holds a vector of Waypoint objects and has an accessor to this vector. This structure was created to provide a standard data type for missions. Its contents and functionalities can easily be expanded upon in the future without affecting existing functions. This standard data type can easily be integrated with `Mapviz` (Section 3.6.1), our *Propbot GUI*, thus satisfying the requirement in Table 3.2 of being able to receive a mission from the GUI.

- **Mission Handler:** This class performs the majority of the mission execution function. It holds an instance of the `SimpleActionClient`, and is configured to communicate with the Propbot Navigation (as seen in Figure 3.4 with different asynchronous arrows). This `SimpleActionClient` is used to submit goals, cancel goals, and receive goal status from *Propbot Navigation*. `SimpleActionClient` is an asynchronous library, so goals can be submitted or cancelled at any time. However, to receive goal status notification, the class has `WaypointCallback` and this sends the next waypoint goal upon success, or ends the mission with a failure flag set. This class has a `Start()` function that starts the 'SimpleActionClient' and sends the first waypoint. The `Pause()` function cancels all goals, but does not end the mission and retains the current waypoint index of the mission so it can be later resumed. The `Stop()` function cancels all goals and sets a mission finished flag to true. This satisifes the start/stop/pause requirements mentioned in Table 3.2. This class also outputs "Mission Progress" in the form of the current waypoint number, whether the mission has ended, and whether the mission has failed, thus also fulfilling the mission progress requirement.

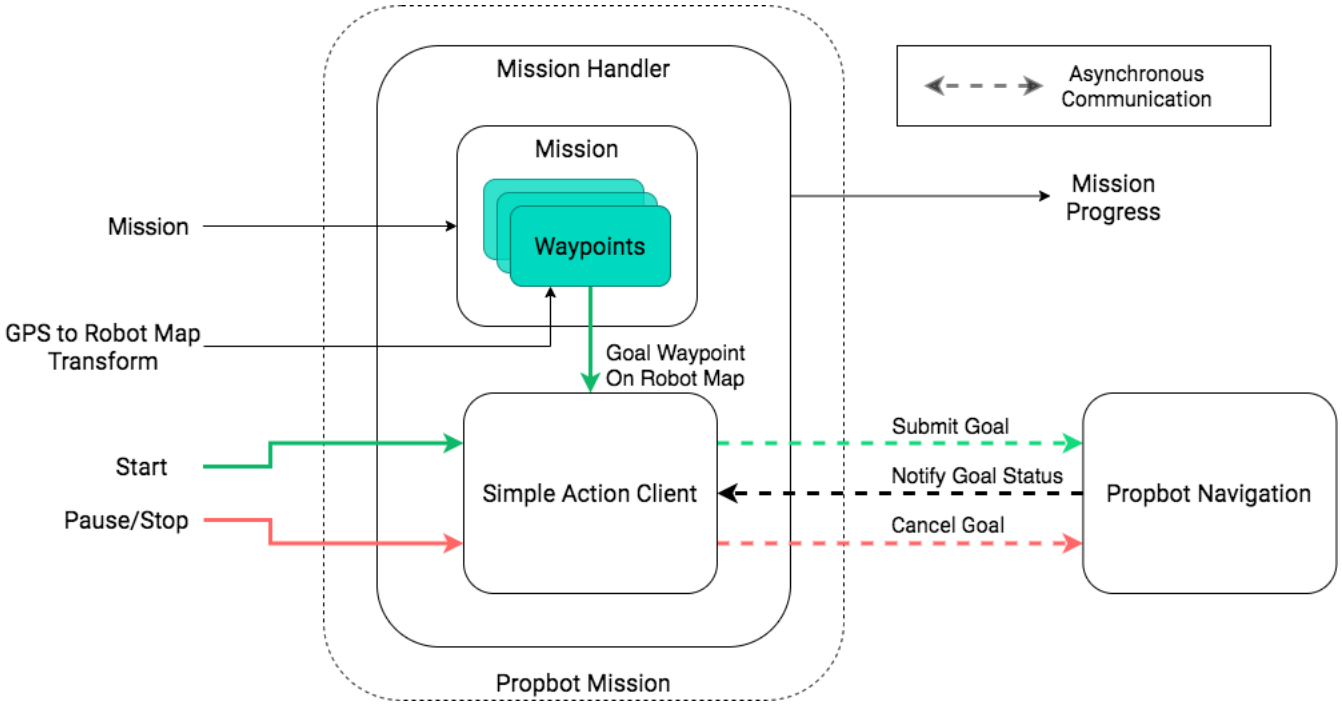


Figure 3.4: Propbot mission package software architecture

3.4.2 Propbot Navigation

The ROS `move_base` package provides a framework for robot navigation stack by linking various plugins to move a robot from its current position to a desired goal position. An overview of the `move_base` package is provided in Figure 3.5. The package's components are further described below:

- **Global Costmap:** This node accepts an occupancy grid produced by the SLAM algorithm and "inflates" the cells of the grids that are marked "occupied" as to increase the perceived size of the obstacles. The "inflation radius" is adjusted to account for the size of the robot and the robot to obstacle distance requirement. The resulting occupancy grid is referred to as the Global Costmap. This is implemented using the standard ROS `2d_costmap` package.
- **Local Costmap:** This node accepts depth sensor data such as point clouds or laser scans produced by a RGB-Depth camera and Lidar to detect the obstacles that are within close range of the robot. The detected obstacles are then "inflated" with a "inflation radius" and the resulting occupancy grid is referred to as the Local Costmap. This is implemented using the standard ROS `2d_costmap` package.
- **Global Planner:** This node takes in a 2D costmap as an input and outputs a path between the robot's current position and its goal position based on the given obstacle information and robot turn radius.
  - **Final Choice:** `A* algorithm with global_planner`
  - *Selection Criteria:* The global planner must generate a route to each waypoint that does not collide with any obstacle's in the robot's costmap. This is the only required feature of the global planner and it satisfies F1.1, F1.2, and F1.3, which state that the robot shall drive through user-generated waypoints, not collide with obstacles, and will stop at predefined points of interest.
  - *Selection Justification:* Only algorithms that are integrated in the ROS environment were considered for this trade study to ensure time and complexity of the project was not drastically increased. The algorithms available are `A*` and `Djikstra` in the ROS `global_planner` package, and the simple euclidean distance measurement method in the ROS 'carrot_planner' package. The 'carrot_planner' package is not a reasonable option because it plans a path that is the shortest vector between the robot's position and the goal waypoint, and does not plan around obstacles as required. `Djikstra` is a graph search algorithm that finds the shortest path between nodes. In general, `A*` is known to be better than `Djikstra`, as `A*` is the same as `Djikstra`, but with an additional heuristic to search closest paths in distance first, thus resulting in faster search times [34]. This is suitable for our application as quick planning and re-planning will be required as the robot maps out its environment.
- **Local Planner:** This node takes in a 2D costmap and the plan produced by the global planner as inputs and generates velocity commands. These velocity commands allow the robot to follow the planned route as close as possible while taking into the given obstacle information and kinematics of the robot.
  - **Final Choice:** `teb_local_planner`
  - *Selection Criteria:* The local planner must provide velocity commands that are feasible

for our differential drive robot and send velocity commands that traverse the global plan without encountering any obstacles. This is related to F1.2 and NF2.1, which state that the robot must avoid obstacles and stay 1m away from all obstacles at all times.

- Selection Justification: We look local planners that are already integrated in the ROS ecosytem to to ensure time and complexity of the project was not drastically increased. We also only look at local planners that are suitable for differential drive robots. There are 5 available packages that meet these conditions: `asr_ftc_local_planner`, `teb_local_planner`, `dwa_local_planner`, `eband_local_planer`, and `adaptive_local_planner`. We can immediately eliminate `asr_ftc_local_planner` and `eband_local_planer` because they do not do any obstacle avoidance. We can also eliminate `adaptive_local_planner` because it is a package that is not maintained or documented, thus it is unreliable for long-term use, and a high risk option for implementation. The remaining options are `teb_local_planner` and `dwa_local_planner`. `teb_local_planner` refers to a "Time Elastic Band" planner, and it models the local path as an elastic band that optimizes for reaching the goal in the shortest amount of time while avoiding obstacles [35]. The `dwa_local_planner` package implements a "dynamic window" of trajectories to evaluate and scores the trajectories based on distance to obstacles and proximity to the global path [36]. Both of these packages meet the bare requirements of creating suitable velocity commands of differential drive robots and avoiding obstacles. However, `teb_local_planner` is known to have better compute performance [37], along with faster replanning and smoother trajectories [38]. In addition, it has a feature that allows for integration of dynamic obstacle avoidance (a feature that no other local planner compared above has), which will be useful for future tight integration of the `propbot_perception` outputs of dynamic obstacle locations and local planning. Since performance is important as we are constrained by the TX1 processing power (C1.3), and dynamic obstacle avoidance will be important for Propbot's operation in dense campus environments, we choose `teb_local_planner` as our local planner.

- **Recovery Behaviours Plugins:** A list of recovery behaviours plugins which will be run if the navigation stack fails to generate a valid plan to the goal position. Currently our navigation stack uses the default plugins `conservative_reset` and `aggressive_reset` which will first force the robot to clear all obstacles in its map that are more than 3m away, and then perform an in-place rotation. If the robot still fails to generate a plan, the map is cleared.

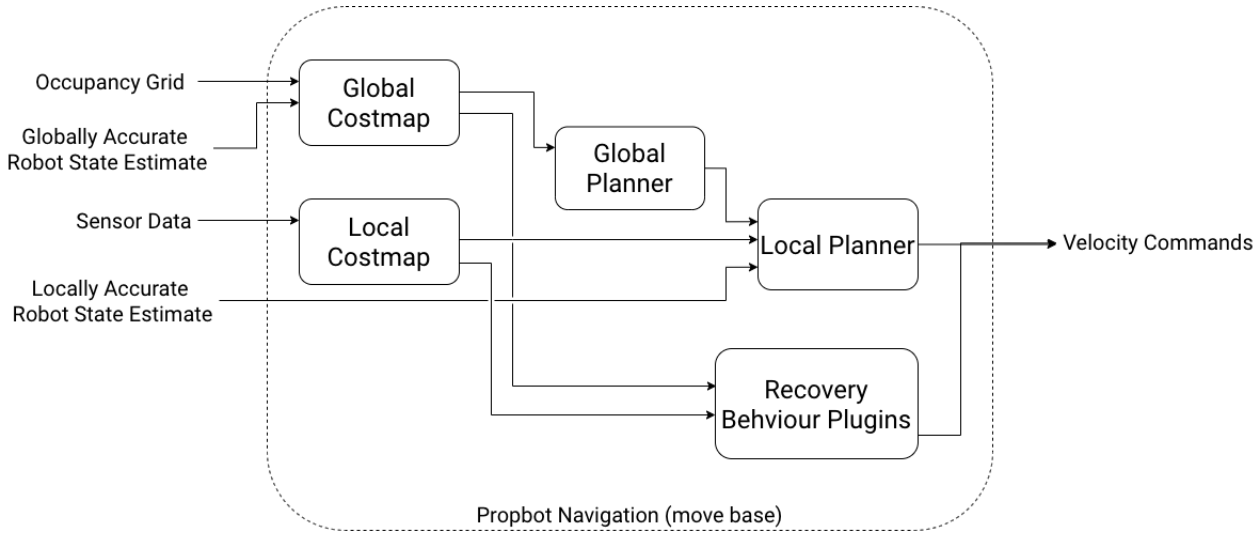


Figure 3.5: Propbot Navigation package software architecture

3.4.3 Propbot Perception

Final Choice: `YOLOv3 (Tiny)`

In order for the robot to effectively plan its route, it must intelligently identify obstacles in its path. Although the LiDAR will detect any obstacles the camera can, it is unable to distinguish between different types of obstacles. For example if there is a person on the side of a path, the LiDAR will detect the person like any other obstacle. Since the obstacle is not in the path of the robot's motion no action is taken. However, this may result in a collision if the person suddenly moves into the robot's path. In this scenario if the camera detects that the obstacle is a person then the robot can take the proper precautions to avoid them in case they move into the robot's path.

To do the object detection and classification we choose to use YOLOv3. YOLOv3 is the latest iteration on the YOLO (You Only Look Once) algorithm. YOLOv3 stands out as having decently accurate predictions while taking less resources than competing algorithms like R-CNN or SSD [39]. Compared to the competing algorithms, YOLOv3 only loses a small amount of accuracy in exchange for around double the speed of these algorithms. For example YOLOv3 scores a COCO mAP of 31 while taking 29 ms while RetinaNet-50 only gains 1 point at 32 mAP while taking 73 ms on the same hardware. (mAP = Mean Average Precision, a metric used for evaluating object detection algorithms). This is very important due the project constraints of using the Jetson TX-1 which means we have fairly limited computing resources. We evaluated YOLOv3 on the Jetson TX-1 using a demo application and found that it could reach at least 20 FPS in "Tiny" mode. Tiny mode reduces the mAP of YOLO down to 28 mAP while reducing inference time by about 30%. In practical use we found that "Tiny" mode still produced similar results to the regular YOLO on our data, so using this mode is preferred to reduce the load on the Jetson TX-1.

Another advantage of YOLOv3 is the ease of software integration. Most machine learning algorithms use frameworks like TensorFlow or Caffe. YOLOv3 is built as a single, small C library where the only major dependencies are OpenCV and CUDA both of which come pre-installed on the Jetson TX-1. This reduces the number of large dependencies we need to manage, thus reducing the project's overall complexity.

Integration with Autonomy

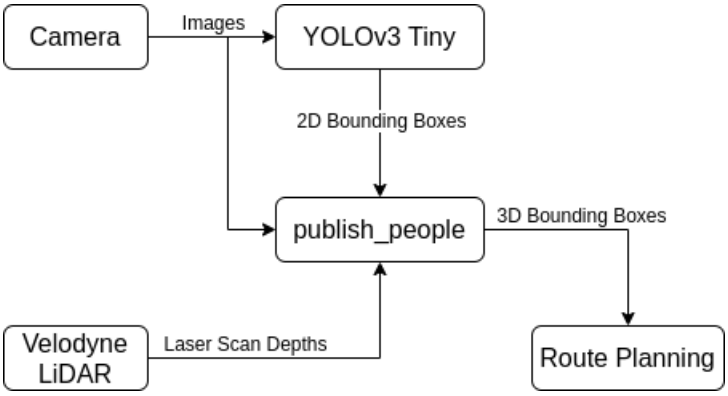


Figure 3.6: Propbot Object Detection System Overview

A challenge with integrating YOLO into our autonomy stack is that YOLO was developed to work on 2D images or video. As a result YOLO only outputs bounding boxes in pixel coordinates on the image. For our purposes we would like the 3D coordinates of a detected object like a person in order to avoid them. To solve this issue we created a custom ROS node (publish_people) which converts the bounding boxes into 3D by fusing data from the LiDAR and YOLO together.

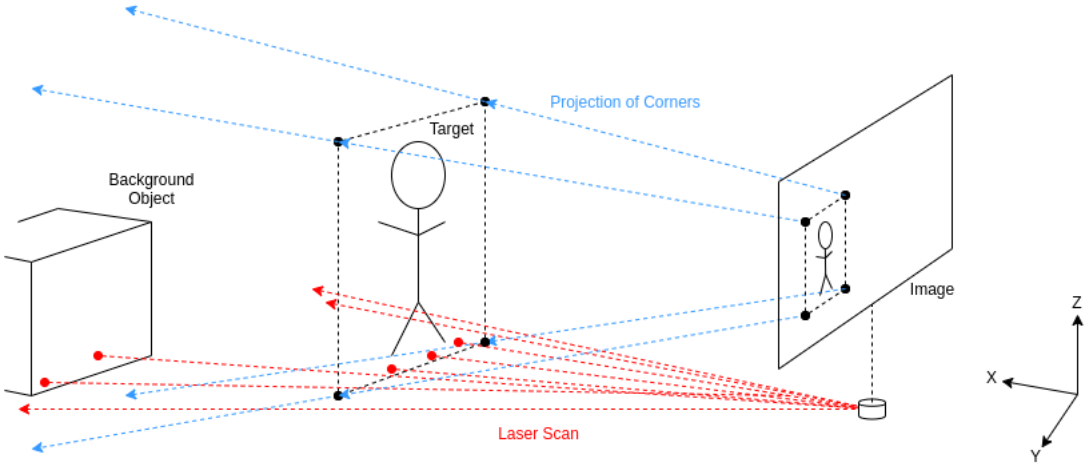


Figure 3.7: Projection of the bounding box from the image to 3D

In order to find out the location of the detected object in 3D the robot must know its direction and its distance from it. The 2D bounding box provides the direction, the robot knows the direction its camera is pointing as well as the direction corresponding to each pixel using projection. Thus by projecting the corners of the bounding box it can figure out the direction of the detected object. This can then be used to find the corresponding depth values of the object from the LiDAR scan. These two pieces of information are enough to determine the location of the object in 3D space.

Since the robot knows the locations of dynamic obstacles around it and can track them over time, it can make simple predictions on where they will move next based on their speed. This information is used in the route planning to avoid moving into the path of these obstacles such as cars or

pedestrians.

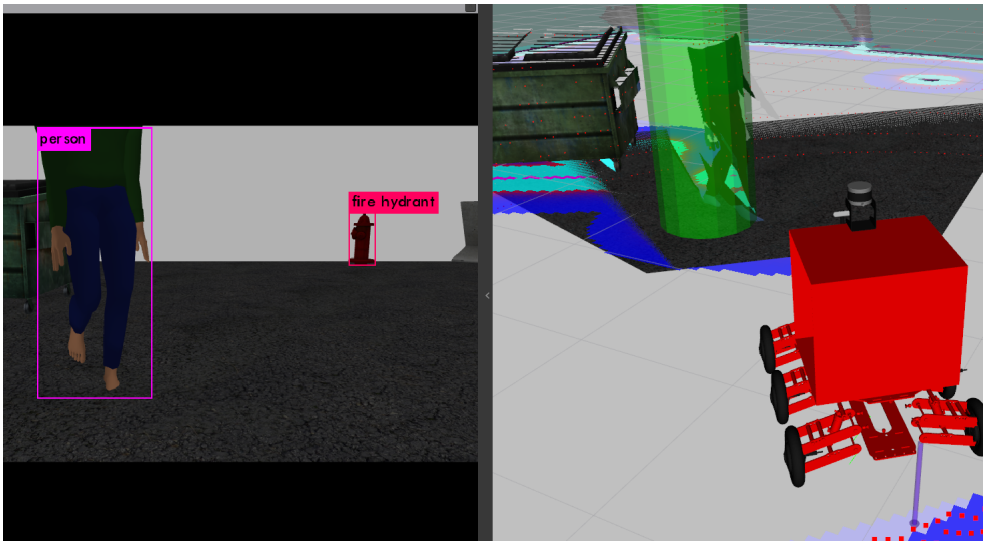


Figure 3.8: A person is detected on camera, they are then marked in 3D by a green cylinder

3.5 Robot Control

3.5.1 Propbot Control

Final Choice: `diff_drive_controller`

Selection Criteria

PropBot is a six-wheeled differential drive system that has 4 motor-drive wheels. The controller for the system needs to take in a trajectory (velocity vector) and translate it to the required left and right wheel outputs.

Selection Justification

There are only two differential drive ROS packages (`differential_drive` and `diff_drive_controller`) with similar features, and both meet the selection criteria. However, only `diff_drive_controller` is currently maintained by the ROS community, is used by many other mobile robot projects, and is real-time suitable. Forums do suggest that `diff_drive_controller` is more difficult than `differential_drive_controller`. But since `diff_drive_controller` is real-time suitable and has more support available online, it is the best choice for our uses.

3.5.2 Vehicle Interface Firmware Stack

Final Choice: `Custom firmware stack`

Selection Criteria

The stack must be scalable to allow for more than four motors as the intention to add drivers to the middle wheels has been expressed, and portable to allow for various platforms to be used mostly interchangeably (e.g. Arduino platforms, ARM devices). In short, the stack must, to an extent, be hardware agnostic

Selection Justification

Although legacy code existed for the system, it would have to be rewritten for the integration of the RC interface. And, there are non-standard Arduino libraries that can be useful for motor control. However, the use of these Arduino-specific libraries locks the functionality to Arduino platforms.

Further, the firmware must be dynamic to allow for the addition of interfaces with multiple devices: motor drivers, peripheral sensors, autonomy computer, e-stop, remote controller.

Layered Vehicle Interface Firmware Stack

The figure shown in figure 3.9 illustrates the layers of abstraction that exist within the firmware.

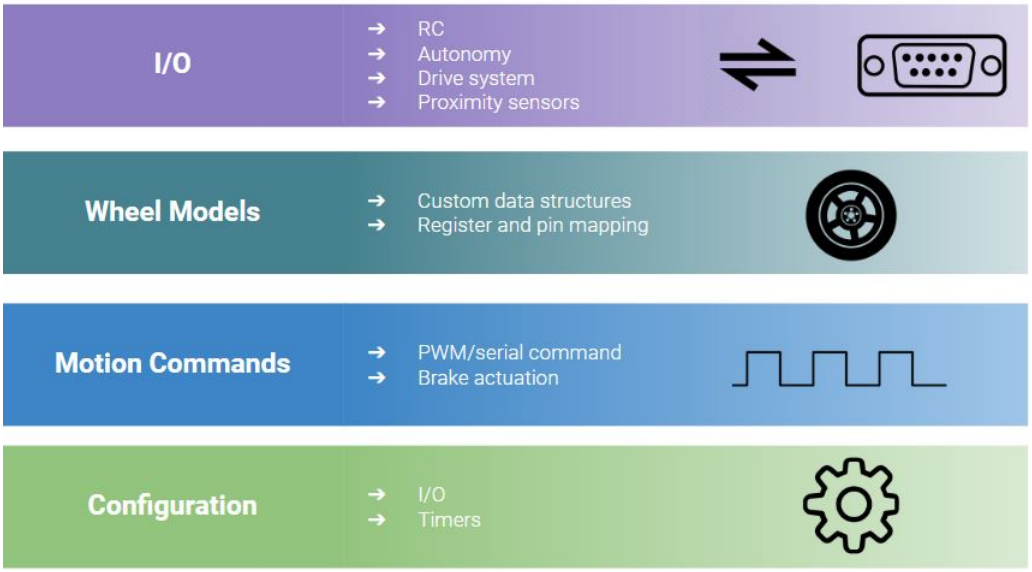


Figure 3.9: Layer structure of the vehicle interface firmware

The layered vehicle interface firmware stack illustrated in 3.9 is further explained below:

- **I/O** At the top level, there is the I/O layer. This layer establishes the interface specifications of all peripheral devices: remote controller, autonomy computer, motor drivers, and e-stop. This can be expanded to included the ultrasonic sensor network or other peripheral devices in the future.
- **Wheel Models** Next is the wheel models. In this layer, custom data structures for wheel motion commands are defined along with several default wheel modes. Further, pins and registers associated with the wheel motors are pulled from the I/O and mapped to the wheel "indices". This ensures that the model layer is agnostic to the pin and register values. This layer is completely reusable and portable across architectures.
- **Motion Commands** The motion command layer deals with the translation of the generalized motor commands to motor driver commands. The setup allows easy integration of new motor drivers, as the interface for the device only needs to be defined and then the commands can be appropriately translated. To increase separation from the hardware, custom wrappers for

interrupt enabling/disabling and pin mode setting will be integrated into the I/O which the command layer will pull from. This will render the command layer fully hardware agnostic.

- **Configuration** The configuration layer is the most intertwined with hardware. It would have to be modified when porting to a different architecture, but can remain relatively the same for porting across platforms of the same architecture (in this case, Arduino platforms). The interfaces are initialized and peripherals are configured (GPIO, timers, interrupts, etc.) so the layer is hardware dependent.

Logarithmically Scaled Speed Commands

The manual controller, shown in figure 2.8 communicates over RC pulses. Wheel speed values are determined by the left or right joystick position with respect to the calibrated maximum and minimum. Initially, the RC value was mapped linearly to a duty cycle or speed value. However, a user will most likely not drive the robot at maximal speed. Further, increasing the resolution of lower speed commands allows for higher maneuverability at medium to low speeds. To achieve this, the mapping between the RC command values and the speed values is logarithmically scaled with user-selectable parameters. The user can change the maximum duty value and logarithmic scale factor to change the mapping. The figure below shows an example of how the mapping works with the user parameters.

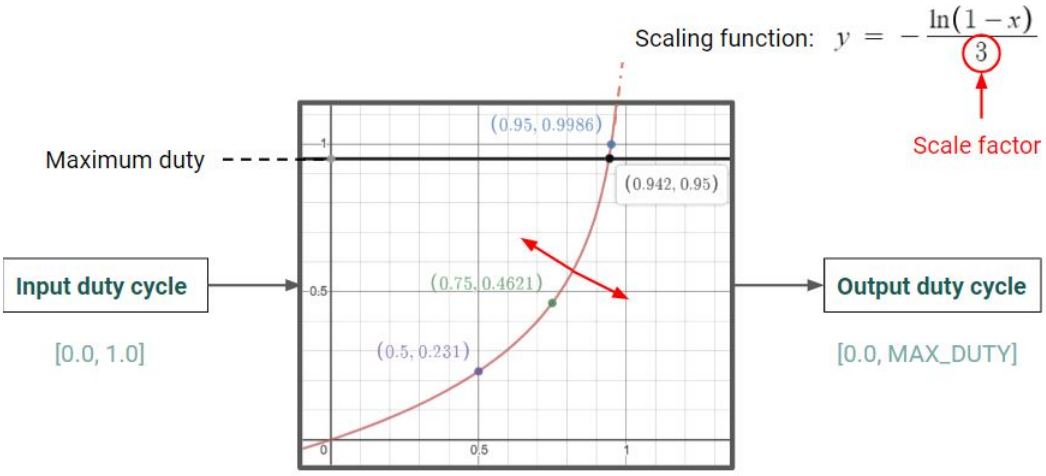


Figure 3.10: Configurable logarithmic mapping of speed commands

Safety Feature 12 (Logarithmic mapping of remote control joysticks)**.** *The RC outputs are logarithmically mapped such that the majority of the joystick movement range is centred around fairly low speeds. This ensures that it is difficult for the user to accidentally increase the speed of the robot.*

Front Ultrasonic Sensor Logic

Assembling an ultrasonic sensor at the front of the robot can help the robot detect the obstacle ahead. In the vehicle interface firmware stack, the ultrasonic sensor keeps running, measuring the distance to the obstacle ahead, and sending the distance to the vehicle interface via I2C (Inter-integrated Circuit) protocol. If the distance to the obstacle is less than or equal to twenty-five centimeters, the vehicle interface sends 'all-brake' command to the wheels, which means shutting down the car immediately. The reason for setting twenty-five centimeters as the threshold value is

because such as distance is the minimum distance that the ultrasonic sensor can detect [40].

3.5.3 Vehicle Interface to Autonomy Computer Communications

Final choice: `rosserial`

Selection Criteria

We are using Arduino UNO as the vehicle interface, the package for building this communication channel shall be compatible with Arduino boards. Also, we are running *Linux* operating system in the autonomy computer, the package shall be able to operate in *Linux* system as well.

Selection Justification

There is only one ROS package that satisfies the criteria above, which is the `rosserial`. This ROS package is not only compatible with ROS Melodic in *Linux* system, but also supports Arduino boards including Arduino UNO [41].

Further, the `rosserial` package limits the number of ROS topics' publishers and subscribers at twenty-five [41], which is much more than what we need in this communication channel. In this communication channel, we only need three publishers and three subscribers. Figure 3.11 illustrates the architecture of this channel.

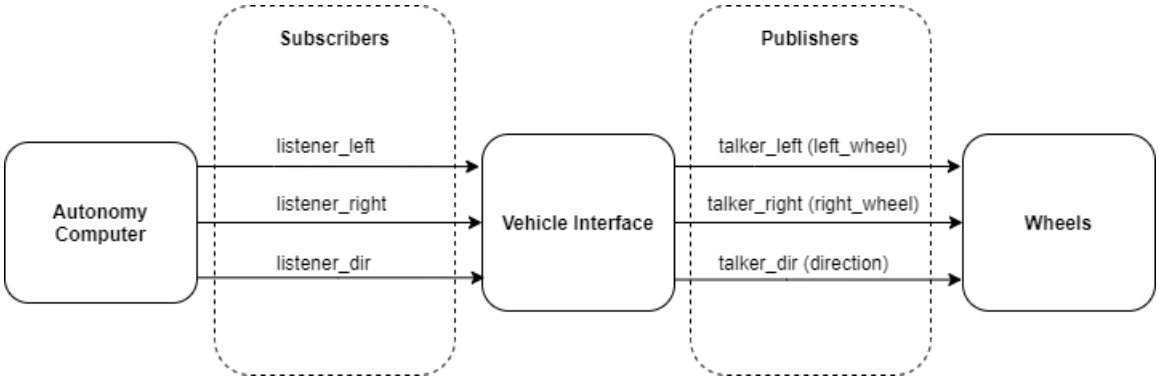


Figure 3.11: Communication channel's architecture

With respect to the autonomy computer, the vehicle interface is a listener or ROS topics' subscriber. The user sends the left wheels' speed, right wheels' speed, and wheels' direction in the three subscribers' ROS topics respectively. With respect to the wheels, the vehicle interface is a talker or ROS topics' publisher. After receiving messages from the subscriber's topics, the vehicle interface publishes those messages to the wheels directly.

3.6 Mission Command

3.6.1 Graphical User Interface (GUI)

Final choice: `Mapviz`

The GUI is pictured in Figure 3.12.

Selection Criteria

The selection criteria for the Propbot GUI are shown in Table 3.3.

Related Tag(s)	Required Feature	Justification
F4.2	GUI shows all configurable features on a single screen	GUI should be intuitive and require little to no configuration in order to improve the efficiency of the data collection process .
F4.2, F4.3	GUI displays location and route progress of the robot	Robot location is parameter that affects the wireless propagation tests conducted by the RSL lab, and the visibility of the route progress improves ease of use and in turn efficiency of the collection process.
N/A	GUI allows for the integration of custom plugins	This is a long term project which is planned to be completed after several capstone terms, so future teams should have the ability to integrate new autonomy features and new tests for the RSL
C1.3	GUI is device agnostic	This is a long term project which is planned to be completed after several capstone terms, so future teams should have the ability to replace the existing hardware such as the autonomy computer when more autonomy features are implemented and the current processing power is found to be limiting

Table 3.3: Required features of GUI

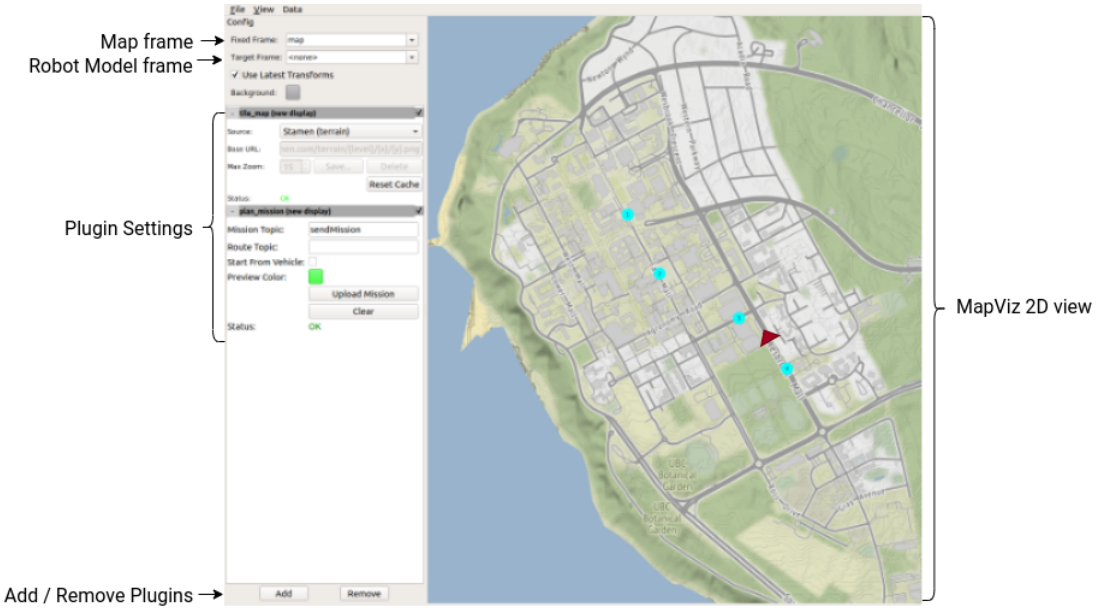


Figure 3.12: MapViz Mission Command GUI

Selection Justification

For a detailed trade study, please refer to Section 4.3.1.

MapViz is a ROS 2D visualization tool which loads plugins dynamically at runtime. This allows the developer to choose which features are included in the GUI and for future capstone teams or RSL members to extend the functionality of the GUI without modifying the MapViz source code. The other GUI option that was explored had been created specifically for drone autopilots which mean that it includes features and configuration fields that are not applicable to a ground robot. Moreover, this option was not device agnostic whereas MapViz can communicate with any computer that has ROS installed.

Design

The design of the Propbot GUI package is shown in Figure 3.13. The components are described below:

- **Plan Mission Plugin:** This is a plugin we created to allow the user to set waypoints on the GUI and upload them to the robot.
- **Tile Map Plugin:** This is an existing plugin package which obtains map tiles from Bing Maps, WMTS Tile service, or Stamen Design and displays them on the GUI.
- **Custom Plugin:** This is a place holder for addition plugins which may be added by future teams.
- **2D View:** This is the main GUI display.

3.6.2 Mission Command Communication Protocol

Final choice: `ROS Network`

Selection Criteria

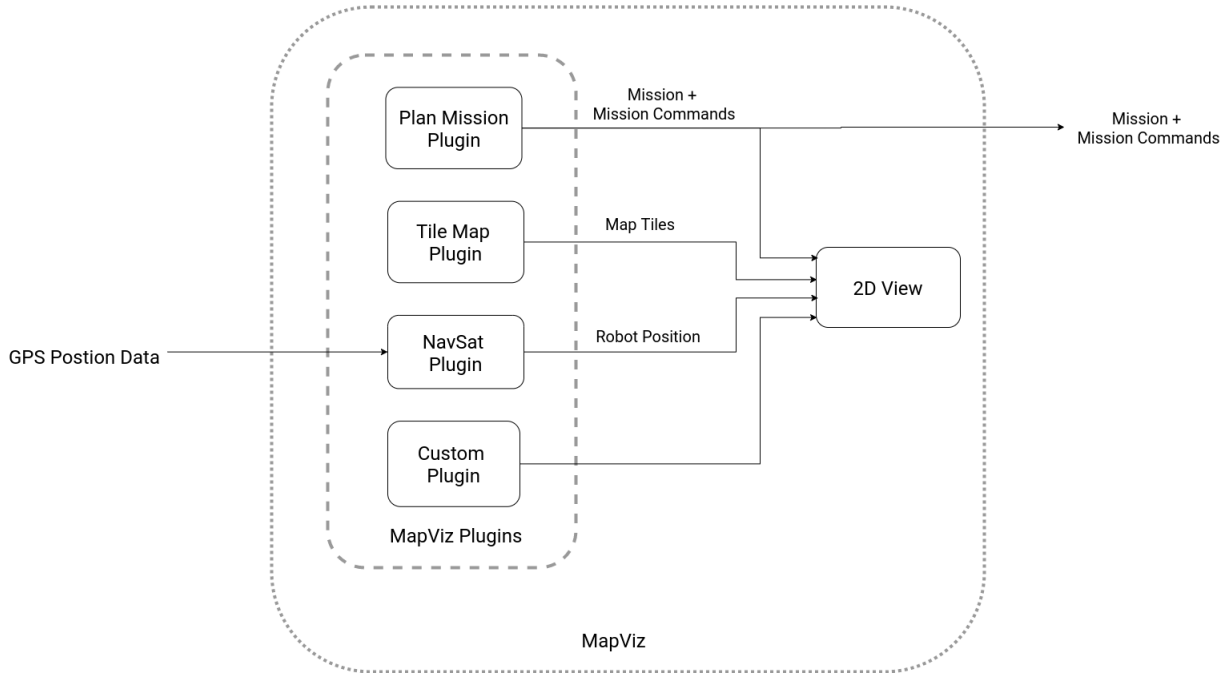


Figure 3.13: Design overview of Propbot GUI

Related Tag(s)	Required Feature	Justification
F4.4	Protocol should allow the robot to communicate with the mission command centre over a local network	Short range communication is adequate for our project scope as the robot operator will be less than 5m away from the robot at all times.

Table 3.4: Required features of Mission Command Communication Protocol

The selection criteria for the protocol are shown in Table 3.4.

Selection Justification

A pair of ROS enabled requires minimal Network setup to configure bi-directional connectivity with basic ROS Network protocols. This protocol requires configuration on both the master and the slave however, this is adequate for our scope as only one robot will be connected.

3.7 Simulation Software

In order to remove hardware factors from our software testing we will use simulation. Fortunately, ROS provides a standard set of high quality simulation/introspection tools. These include the Gazebo simulator and Rviz data visualizer which both integrate seamlessly into the ROS environment. The integration of these tools are handled by our own software packages: *Propbot Gazebo, Propbot Description*, and *Propbot Rviz*.

Figure 3.14 shows a high level diagram of how the simulation is used.

In simulation mode, Gazebo provides the following: simulated world, simulated robot, simulated sensors. The simulated world is the sandbox world the robot is inside, it can be changed to be similar to the expected real world environment. Simulated worlds can range from a simple field to

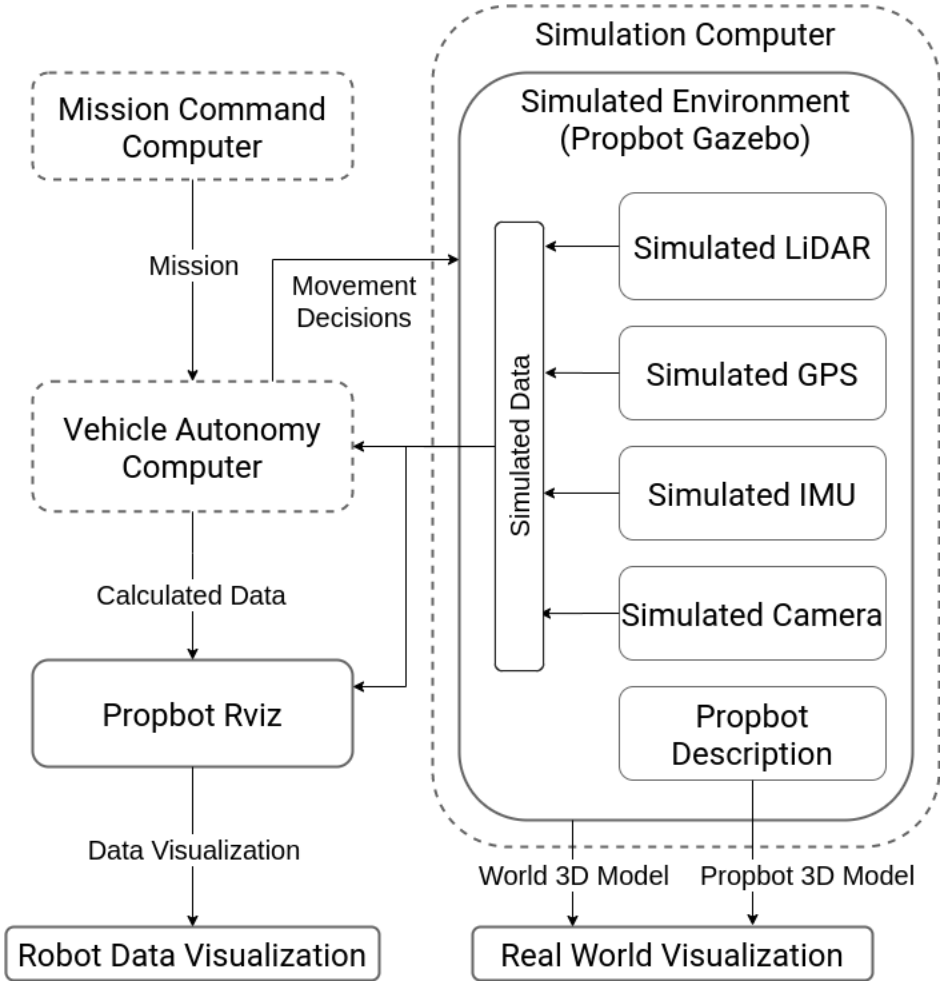


Figure 3.14: Overview of the Simulation Design

detailed 3D scans of museums or other real world locations. The simulated robot uses a 3D model of Propbot equipped with the majority of our sensor suite which inclues 3D lidar, RGB-Depth camera, GPS and IMU. This is part of our Propbot Description package.

3.7.1 Simulation Robot Model

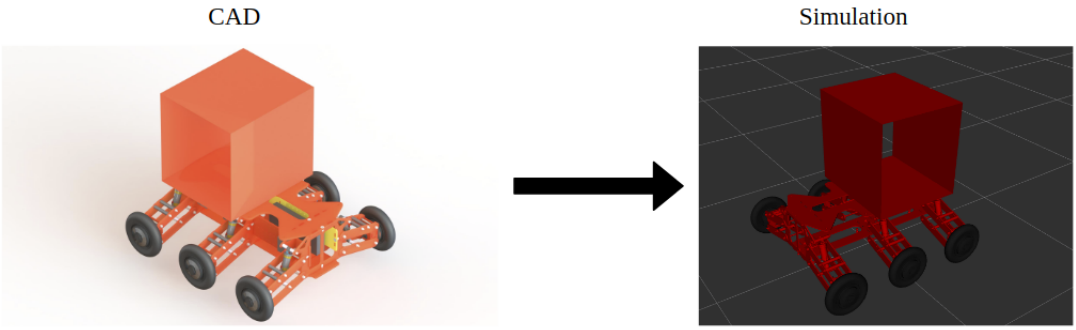


Figure 3.15: Propbot CAD model and the corresponding simulated model in Rviz

For both visual and validation purposes, the 3D model of Propbot was integrated into the simulations. This involved utilizing the existing SolidWorks CAD to generate essentially a parsable description of the robot's structure, dynamics, and appearance.

Robot Structure

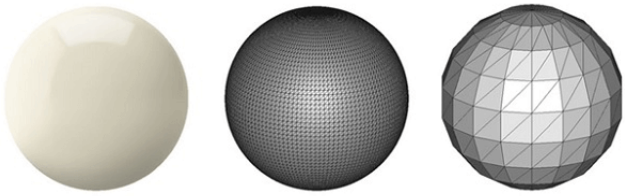


Figure 3.16: Example of a stereolithography (STL) mesh of a CAD model

Figure 3.16 shows an example of how a CAD model is converted into a mesh of triangular planes. The data size of a mesh is significantly less than that of a complete CAD as it only characterizes geometry. The lightweight nature of the meshes allow models to be integrated into simulations with ease. Then, colours, textures, and dynamics can be integrated as a layer on top of the geometry. So, to integrate the robot's structure, the CAD was converted into a series of STL meshes. Then, the meshes were symbolically linked together to form the geometry of the entire robot.

Robot Dynamics

Although integrating a visual representation of the robot into the simulations is trivial, the description of a body's dynamics is not as so. For this reason, the CAD was split into pieces so that their unique dynamics could be computed and the various joint types could be defined (e.g. rotation for wheels vs. fixed for different parts of the chassis). Using a SolidWorks plugin to export designs to a format compatible with ROS [42], the CAD was converted to links to meshes with hard-coded numerical values for the inertia and collision characteristics.

Unfortunately, shortcomings in the CAD, e.g. unconventional axes and lack of material integration, affected the performance of the simulated model. Nonetheless, we worked with the values we could extract from the exported model, and integrated them into a custom model. From there, the dynamics performance was improved by modifying and manually tuning orientations, origin placement, and scaling of the mass and moments of inertia. It is recommended that this be remedied in the future via improvements to the CAD to ensure that it appropriately reflect the robot's mass and inertial characteristics.

Wheel Dynamics

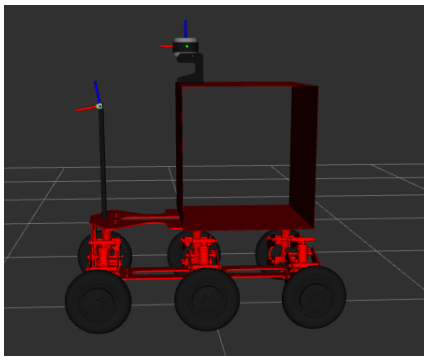
The various parts of the robot's chassis were integrated as a set of static joints, as it is one large piece. The wheels, however, need to be fixed except for on their axle. Further, the real robot has only four motors to drive six wheels, with the central wheels acting as passive stabilizers. To characterize the mass, inertia, and driving characteristics of the wheels and robot, both passive (free-wheeling, not driven) and actuated wheel model descriptions were created an utilized.

The actuated wheels were then linked to the controller so that motion commands from the autonomy stack could be sent, allowing the robot's operation to be simulated.

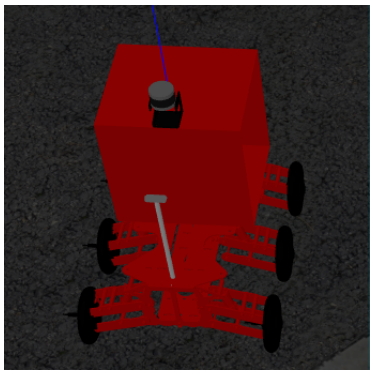
Accounting for Friction

The affects of friction vary greatly depending on the load, driving surface, incline, weather conditions, etc. To account for some conditions, ballpark values were defined for the wheel models so that sliding was mitigated.

Sensor Mounting



(a) Placement orientation of the RGB-D camera and the Velodyne Puck in Rviz



(b) Gazebo Simulated Environment

Figure 3.17: Sensor placement in the various simulation environments

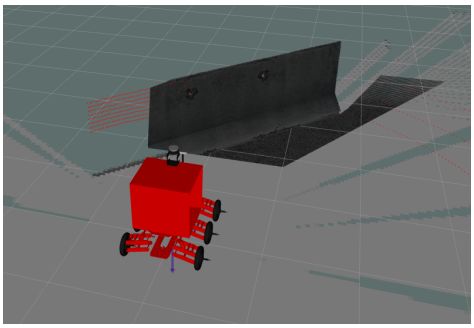
Not only can the robot's geometry be modeled, but so can the sensors. Both the Velodyne Puck and the Realsense RGB-D camera were integrated into the simulated robot model. Figure 3.17 shows the orientation specifications and placement of the sensors in both the Rviz and Gazebo simulation environments. The models used reflect the dimensions and mass of the real sensors.

3.7.2 Simulation Sensors

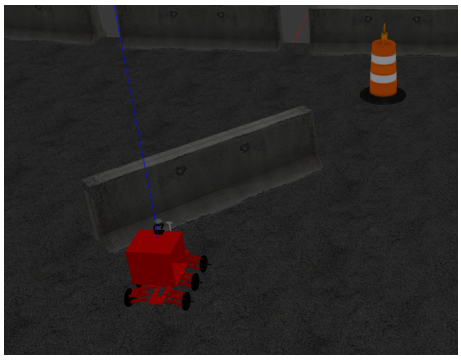
Each sensor is simulated with a Gazebo Sensor Plugins which enables the robot to read data from the simulated world and react accordingly. For example if a garbage can is in front of the robot in

the simulated world, then the simulated LiDAR would show a cylindrical object. Moreover, each sensor plugin is selected and tuned to mimic the properties and specifications of our selected sensor models.

The simulated sensors provide data to our software which performs navigation and routing as if it was running on the real robot. Our software's movement decisions are fed back into the simulator which controls the simulated robot's movement. The simulated real world can be viewed through Gazebo's GUI, which shows a bird's eye view of the robot and the simulated world. The data from both the simulated sensors and calculations output from our software can be visualized by Rviz, this makes it easy to understand what the robot is seeing. The robot interaction betwen the robot and the simulated world is illustrated in figure 3.18.



(a) Point cloud visualization in Rviz: 3D Lidar (red), RGB-Depth camera (black)



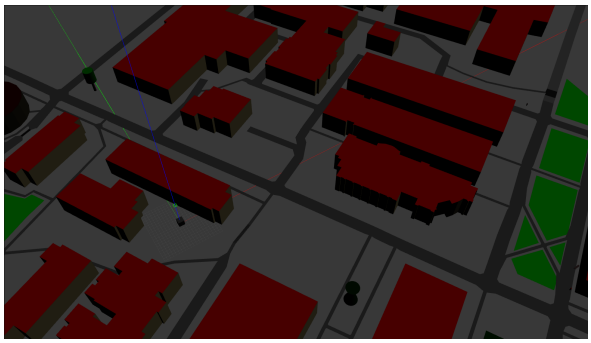
(b) Gazebo Simulated Environment

Figure 3.18: Sensor simulation for Velodyne Puck and Intel Realsense d435

3.7.3 Simulation Environment

The Gazebo simulation environment can be populated with obstacles from the Gazebo model database or custom obstacles we construct from 3-D mesh files. Our custom environment is created by downloading geographic data from OpenStreetMaps (OSM) and converting this data into a 3-D model using the OSM2World converter. The generated 3-D model is populated with building structures, roads, and trees just as they appear on the real map and can be added to the Gazebo "world" where additional dynamic and static obstacles are inserted to create a more realistic simulation.

(a) Google Earth



(b) Gazebo Simulated Environment

Figure 3.19: Building Structures along UBC West Mall

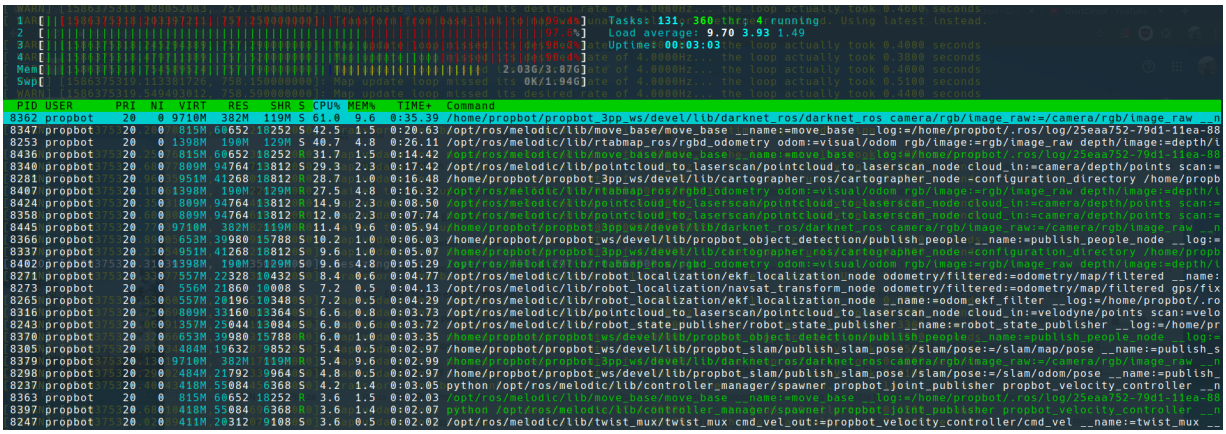


Figure 3.20: Process Monitor on the Jetson TX1

3.7.4 Processing Power Constraints

The robot contains a large number of different software components, many of which are very computationally intensive. One concerns with this is that our Jetson TX1 autonomy computer would not be able to handle the load of all of these software components. After implementing all of our features we tested the whole system on the Jetson TX1. To do the test we ran the simulation software on a separate computer connected via LAN to the Jetson TX1. The Jetson TX1 ran the autonomy software, just like in the actual robot. This is done since the simulation on its own is as intensive to run as the full autonomy software.

From the test found that the Jetson TX1's CPU was pinned at 100% the entire time on all 4 cores. These are the top 5 components that took up the most processor time:

1. Move Base for robot control (70% total on 2 cores)
2. Visual Odometry (65% total on 2 cores)
3. Darknet ROS/YOLO for the dynamic obstacle detection (60% on 1 core)
4. Google Cartographer (40% total on 2 cores)
5. Pointcloud to Laserscan conversion (36% on 2 core)

In addition to this we saw that the simulation was not running around half speed and multiple software components were complaining about insufficient update rate. Despite this, the simulation was able to run slowly and the robot could move a bit. Therefore we can conclude that the Jetson TX1 is just barely insufficient to run the autonomy software. It is possible that optimizing the software a bit more would allow the TX1 to run it. However in a safety critical system it is better to over-provision to ensure reliable operation. Therefore our recommendation would be to upgrade to the Jetson TX2 which has much greater computational power.

4 Trade Studies

Final choices are in boldface.

4.1 Localization and Motion Planning

4.1.1 SLAM Package

The following table (Table 4.1) compares different SLAM and position estimate libraries that are publicly available that have the required features mentioned above. These libraries were selected to optimize for position accuracy and usage of as many of our sensors as possible. In addition, to reduce cost and maintain the ability to share this project publicly, only open-source packages were looked into.

Software Library Name	**Google Cartographer** [43]	RTAB Map [44]
Input Sensors	GPS IMU 3D LiDAR RGB-D Odometry (Wheel encoder data)	GPS 3D LiDAR RGB-D Odometry
Output Frequency	Configurable ($\geqslant$1Hz available)	Configurable ($\geqslant$1Hz available)
ROS Integration	Ros integration exists and is supported	Ros integration exists and is supported
Notes	Showed best accuracy when compared to GMapping and Hector SLAM [45] Lots of online support and maintenance	Showed better accuracy when compared with Gmapping [46]

Table 4.1: Comparison of SLAM algorithms

4.1.2 GPS

Although there are better GPS chips out there (ex. Ublox NEO-M9), they do not come with breakout boards. Some other GPS chips (ex. Ublox LEA06) that do come with breakout boards have already reached their end of life, so it wouldn't make sense to integrate them into a new system. This resulted to the limited number of options that are compared above.

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 4.2: Comparison of GPS options

4.1.3 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 $\pm 250, \pm 500, \pm 1000, and \pm 2000°/sec$ -Integrated 16-bit ADCs	-3-axis digital output -Full scale of $\pm 245, \pm 500, and \pm 2000°/sec$ -16-bit data output
Accelerometer Features	-3-axis digital output -Programmable range of $\pm 2g, \pm 4g, \pm 8g and \pm 16g$ -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 4.3: Comparison of IMU options

4.1.4 LiDAR

Parameter	Slamtec RPLiDAR	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 4.4: Comparison of LiDAR options

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 4.5: Summary of LiDAR Comparison

4.1.5 Camera

To narrow down the cameras in the comparison, we compare Nvidia's officially recommended vendors as well as Jetson TX-1 user tested cameras. More options were considered but were overall inferior in specifications to the presented options so they were omitted from this table.

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 4.6: Comparison of camera options

4.2 Robot Control

4.2.1 Close-Range Sensors

There is a large number of different sensor types and options. Most sensors have variants based on their performance, such as max range. For simplicity this table will list the highest performance variant of each sensor to evaluate them more based on best performance rather than performance relative to cost.

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 4.7: Comparison of close range sensor options

4.2.2 Motor Drivers

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 4.8: Comparison of motion driver options

4.2.3 Vehicle Interface

Name	**Arduino Uno**	Arduino Mega
Digital I/O	14 (of which 6 provide PWM output)	54 (of which 15 provide PWM outputs)
Serial Channels	1	4
Cost	33.59 CAD	49.95 CAD

Table 4.9: Comparison of vehicle interface options

Given that more than one serial channel is required (one for autonomy computer, one for CAN), the Uno would be insufficient. Further the abundance of GPIO and serial channels on the Mega allows for future growth.

4.3 Mission Command

4.3.1 GUI

According to the research done, two options for the mission command centre GUI were found – QGroundControl and RViz. The following table demonstrates the comparison between these two options.

Parameter	**MapViz**	QGroundControl
Type	3D graphical interface	Full flight control interface
Complexity	A bit difficult to install	Easy and straight usage for beginners
Compatible with radio link option	To be determined	Yes

Table 4.10: Comparison of GUI options

Appendix A Current State

The current robot electrical system is illustrate as below:

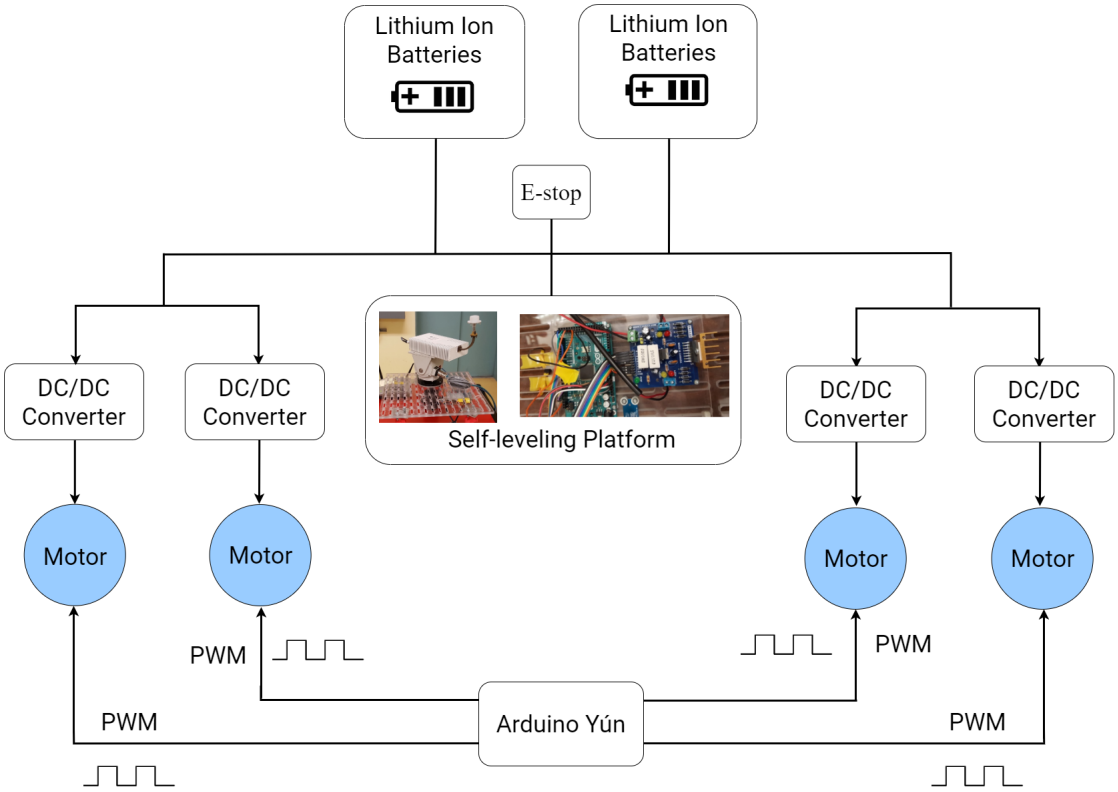


Figure A.1: Current system diagram

A.1 BLDC Motor Specifications

The current motors installed are untraceable. None of the first generation grad students, second generation grad students, and our capstone group could find documentation on this motor. The motors are however installed, and have been proven to be functional in the past.

The specification below is the only information we get from the tag on the current motors.

Parameter	Value
Power rating	250W
Nominal voltage	24VDC
Maximum continuous current	10A

Table A.1: Current BLDC motor specifications

A.2 DC/DC step-down converters

Since the output voltage of the battery is greater than 36 V and the nominal voltage of the BLDC motor is only 24 V, the usage of the DC/DC step-down converters aims to decrease the voltage to the BLDC motors. But now, the output voltage of the DC/DC converters can only be lowered to

around 35 V, which is still higher than the nominal voltage of the motor. This prevents our progress proceeding with the current motors and converters.

A.3 Speed Control

The current method of speed control is to generate PWM signals which are then fed into the motor drivers. The system is an open-loop and therefore has no feedback mechanism.

A.4 Cable Management

Currently, wires from the motor drivers and Arduino (controller) are plugged loosely into an unmounted breadboard to facilitate the connection between the drivers and the Arduino. In addition, some wires are directly connected to the Arduino headers. Regarding the DC/DC converter connections, the splitter has numerous sections that are exposed. This is unsafe and may cause faults.

Appendix B Additional Information about ROS

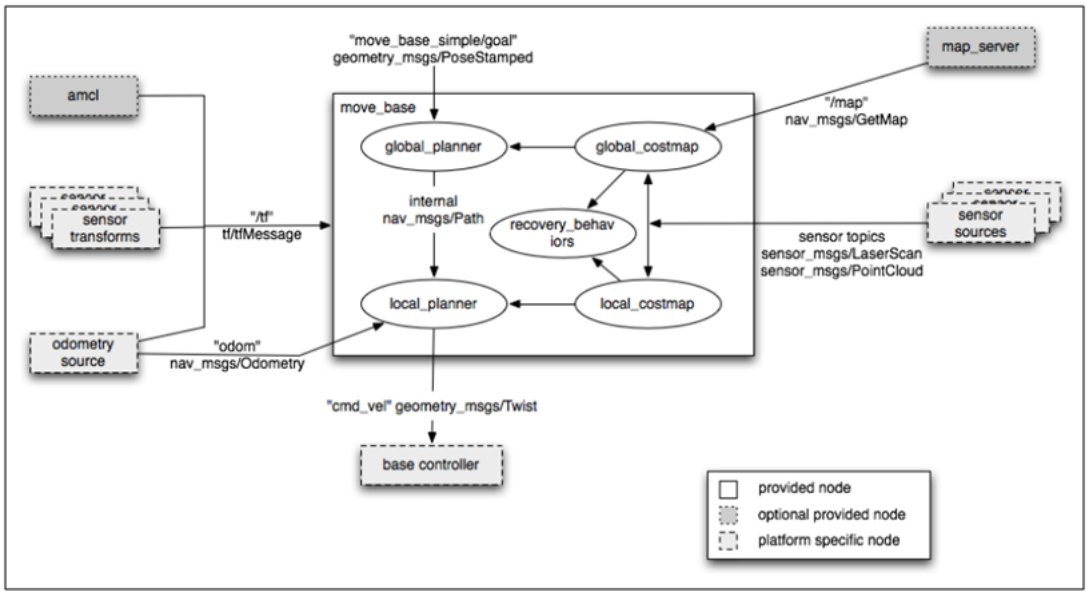


Figure B.1: Typical ROS package architecture for a robot. acml: SLAM package.

Appendix C Future Work

C.1 Hardware Recommendation

This section includes the recommendations for several hardware components which were out of scope this this project. The recommendations for a wheel encoder system and mission command centre telemetry link can be found in sections C.1.1 and C.1.2 respectively. The hardware components which are still undecided include the mechanical brake, autonomy hardware mounting system, wire harnesses, encoder mounting system.

C.1.1 Wheel Encoders

The encoder system has been selected, but will not be added to the system due to the level of difficulty of their integration into the mechanical system. However, it can be integrated in the future and so the autonomy system has been configured to allow for encoder inputs.

Final choice: `RLS LM13`

Figure C.1: LM13 rotary encoder system: readhead and magnetic ring

Selection Criteria

We need to use encoder to get the position and rotational speed of the motors to perform motion estimation and run localization algorithm as well. Initially, an optical encoder sensor along with a custom waterjet cut slotted disk was selected. However, after considering the performance of optical sensors given possible weather conditions (rain, dirt, mud, dust, low light, bright light, etc.) and the effects of the robot's movement (vibrations and shock, jerk), it was concluded that non-optical encoding system would be necessary. Although a covered optical encoder could address most of the environmental concerns, optical encoders are very sensitive to physical vibrations.

Due to the mechanical constraints, this proved to be a difficult solution to determine as most off-the-shelf magnetic or capacitive encoders require a more with than what is possible given the system (measured to be less than 14mm). The ideal encoder must be resistant to vibrations, must provide adequate resolution (above 1000 points per rotation), and must be rated for the defined maximum speed of 10km/h.

Selection Justification

For a detailed trade study, please refer to section C.2.1 The best option so far would be to use the LM13IC10B (readhead) + MR100F085A160E00 (ring) rotary incremental magnetic encoder system, see Figure C.2 for functional information. The ring encoder features a compact sealed readhead and a magnetised ring. The actuator is a periodically magnetised ring with a pole length of 2 mm. Also, the readhead can ride at up to 0.4 mm from the ring's surface. This system is ideal as a custom mounting system can be designed to fit the slim ring onto the wheel shafts. Further, the system has water-proof sealing to IP68 and is highly resistant to shock, vibrations and pressure.

The mounting system design is still in progress, and so the size of the magnetised ring may differ than the one selected (100mm OD). But the same readhead will be used.

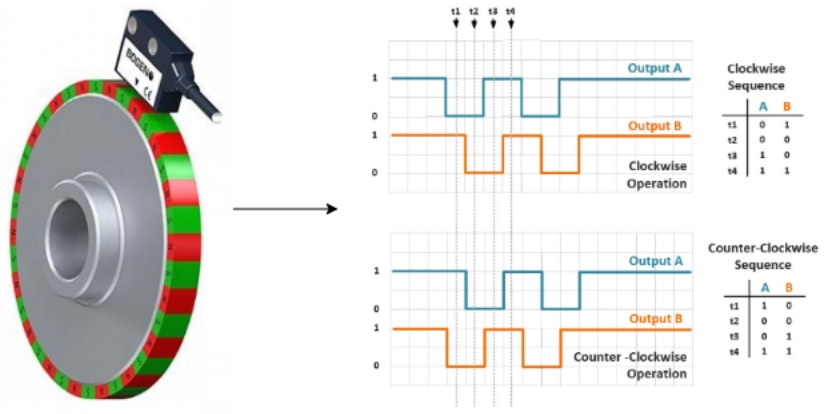


Figure C.2: How a magnetic ring incremental encoder functions

C.1.2 Mission Command Centre Link

Final choice: `Cellular link`

Selection Criteria

To allow the robot to transmit telemetry and receive missions, even if it is several kilometers away from the mission command centre, a mission command link option needs to be chosen. This link connects the mission command centre and the PropBot over a wireless connection. The required features for this wireless link are described in Table C.1.

Related Tag(s)	Required Feature	Justification	Specification
F4.3, NF4.1	The distance range is long enough. Minimum of 2 km.	Since the PropBot will be moving throughout the campus, a long-distance range is necessary.	The link shall still work even if the PropBot is several kilometers away from the mission command centre.
NF4.1	The robot shall report operational status at 1 Hz	PropBot needs to move around campus while continuously sending telemetry back to the mission command center.	The link needs high up time.
NF4.1	Sufficient bandwidth for telemetry at 1 Hz Minimum of 16 Kbit/s	Should have enough bandwidth to transport basic telemetry.	Telemetry reporting should not be constrained by bandwidth.

Table C.1: Required features of mission command link

Selection Justification

For a detailed trade study, please refer to Section C.2.2

After comparing the radio link, WiFi link, and cellular link, the telemetry option we choose is the **cellular** link. The main driving factor for the decision is the up time/availability. Since the robot

drives around campus it is highly likely that buildings will obstruct a radio link that is on the ground. By contrast a cellular connection will work anywhere on UBC campus. WiFi is infeasible due to the poor connection availability outside of buildings. Bandwidth is not really a limiting factor for any of them as we only need to send a very small amount of telemetry data every second (less than 1KB).

C.2 Trade Studies for Hardware Recommendations

C.2.1 Encoders

Parameter	PM-L45	OPB625	**LM13IC10B + MR100F085A160E00**
Sensor Type	Optical	Optical	Magnetic
Voltage	5 - 24V	4.5 - 16V	4.7-7V
Output	NPN - Open Collector	Pull-Up	Incremental, pulse
IP Rating	IP 64	-	IP68
Resolution	Dependent on disk	Dependent on disk	1024 counts per turn
Price	USD 16.61	USD 3.03	USD 85.16

Table C.2: Comparison of encoder options

Although the optical encoders are significantly less costly than the magnetic encoder, they are barebones sensors that require a custom slotted disk to be made. Further, readings could be compromised due to weather conditions such as rain, dust, mud, etc., all of which are possible operational conditions. Given this, a non-optical sensor is ideal given the encoders will be placed directly by the wheels. Although a custom encoder disk is not necessary for the LM13 as RLS (the manufacturer) provides a great deal of selection when it comes to the magnetised rings. A custom mounting system must designed to mount the rings onto the wheel shafts as well as to secure the readhead. However, this allows the constraint of the wheel shaft space (less than 14mm) to be somewhat avoided.

C.2.2 Mission Command Link

The main options for telemetry are Wi-Fi, Radio, and Cellular.

Link Type	Wi-Fi	Radio	**Cellular**
Uptime	Low - Close to buildings	Medium - Only within a certain range, and not obstructed	Very High - Available unless heavily obstructed
Data Rate	Very High (50+ Mbit/s)	Low (2+ Kbit/s)	Medium to High (1-50 Mbit/s)
Data Limit	Unlimited	Unlimited	Limited by Plan

Table C.3: Comparison of telemetry link options

References

[1] "Inertial measurement unit," Nov 2019. [Online]. Available: https://en.wikipedia.org/wiki/Inertial_measurement_unit

[2] *A Framework for Automated Driving System Testable Cases and Scenarios.* [Online]. Available: www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13882-automateddrivingsystems_092618_v1a_tag.pdf

[3] "Transistor–transistor logic." [Online]. Available: https://simple.wikipedia.org/wiki/Transistor-transistor_logic

[4] "World geodetic system (wgs84)." [Online]. Available: https://gisgeography.com/wgs84-world-geodetic-system/

[5] A. Martins, G. Amaral, A. Dias, C. Almeida, J. Almeida, and E. Silva, *TIGRE — An autonomous ground robot for outdoor exploration*, 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6623529

[6] Fernandes, *Intelligent Robotic Car for Autonomous Navigation: Platform and System Architecture*, 2012.

[7] N. Gobillot, C. Lesire, and D. Doose, "A modeling framework for software architecture specification and validation," *Simulation, Modeling, and Programming for Autonomous Robots Lecture Notes in Computer Science*, p. 303–314, 2014.

[8] Z. Littlefield, A. Krontiris, A. Kimmel, A. Dobson, R. Shome, and K. E. Bekris, "An extensible software architecture for composing motion and task planners," *Simulation, Modeling, and Programming for Autonomous Robots Lecture Notes in Computer Science*, p. 327–339, 2014.

[9] J. Becker, M. Helmle, and O. Pink, "System architecture and safety requirements for automated driving," *Automated Driving*, p. 265–283, 2016.

[10] S. Huang and G. Dissanayake, "Robot localization: An introduction," Aug 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/047134608X.W8318

[11] W. Burgard, C. Stachniss, M. Bennewitz, and K. Arras. [Online]. Available: http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf

[12] "Rc receiver power consumption." [Online]. Available: https://p11.secure.hostingprod.com/@site.hobbypartz.com/ssl/webfolder/hobbypartz/FlySkyT6Manual.pdf

[13] "Gps power consumption." [Online]. Available: https://cdn.sparkfun.com/assets/2/f/6/8/3/NEO-M8P_DataSheet__UBX-15016656_.pdf

[14] "Gps antenna power consumption." [Online]. Available: https://cdn.sparkfun.com/assets/d/f/f/0/7/2065600050-PS.pdf

[15] "Imu power consumption." [Online]. Available: https://cdn.sparkfun.com/assets/learn_tutorials/5/5/0/MPU9250REV1.0.pdf

[16] "Encoder power consumption." [Online]. Available: https://www.rls.si/media/wysiwyg/data_sheets/LM13D02_05__EN_data_sheet_.pdf

[17] "Ultrasonic sensor power consumption." [Online]. Available: https://www.maxbotix.com/documents/LV-MaxSonar-EZ_Datasheet.pdf

[18] "Lidar power consumption." [Online]. Available: http://www.mapix.com/wp-content/uploads/2018/07/63-9229_Rev-H_Puck-_Datasheet_Web-1.pdf

[19] "Rgb-d camera power consumption." [Online]. Available: https://www.mouser.com/pdfdocs/Intel_D400_Series_Datasheet.pdf

[20] "Nvidia jetson tx1 power consumption." [Online]. Available: https://devblogs.nvidia.com/nvidia-jetson-tx1-supercomputer-on-module-drives-next-wave-of-autonomous-machines/

[21] "Vehicle interface power consumption." [Online]. Available: http://www.gammon.com.au/power

[22] "Motor driver power consumption." [Online]. Available: https://www.amazon.com/DROK-DC-DC30-60V-Non-isolated-Synchronous-Transformer/dp/B018TGK57E

[23] "Rf equipment power consumption." [Online]. Available: https://docs.google.com/spreadsheets/d/1xM-NMH-D3dIKB2mW6McfNKuyJLcrWvKT/edit#gid=559707812

[24] "Vn-300 dual antenna gnss/ins." [Online]. Available: https://www.vectornav.com/products/vn-300/applications

[25] Racelogic, "How does dgps (differential gps) work?" Aug 2018. [Online]. Available: https://racelogic.support/01VBOX_Automotive/01General_Information/Knowledge_Base/How_Does_DGPS_(Differential_GPS)_Work?

[26] "Real-time kinematic and differential gps." [Online]. Available: https://www.e-education.psu.edu/geog862/node/1828

[27] "Compare intel realsense depth cameras (tech specs and review)." [Online]. Available: https://www.intelrealsense.com/compare-depth-cameras/

[28] "How rs232 works." [Online]. Available: https://www.best-microcontroller-projects.com/how-rs232-works.html

[29] "Canopen: An application layer protocol for industrial automation." [Online]. Available: https://www.rtautomation.com/technologies/canopen/

[30] "24v 250w brushless motor controller electric bicycle speed control for e-bike and scooter," Nov 2019. [Online]. Available: https://www.amazon.ca/s?k=Brushless-Controller-Electric-Bicycle-Control

[31] "Roboteq sbl1360." [Online]. Available: https://www.roboteq.com/index.php/roboteq-products-and-services/brushless-dc-motor-controllers/276/sbl1360-detail

[32] "Roboteq sbl1360a." [Online]. Available: https://www.roboteq.com/index.php/roboteq-products-and-services/brushless-dc-motor-controllers/429/sbl1360-277-detail

[33] N. Charron, "waypoint_nav," Dec 2018. [Online]. Available: https://github.com/nickcharron/waypoint_nav

[34] [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[35] "teb_local_planner." [Online]. Available: http://wiki.ros.org/teb_local_planner

[36] "dwa_local_planner." [Online]. Available: http://wiki.ros.org/dwa_local_planner

[37] J. F. Sanchez, "Implementation and comparison in local planners for ackermann vehicles," *POLITECNICO DI MILANO*, 2018. [Online]. Available: https://www.politesi.polimi.it/bitstream/10589/141744/3/Tesina_JordiFerrer.pdf

[38] B. Cybulski, A. Wegierska, and G. Granosik, "Accuracy comparison of navigation local planners on ros-based mobile robot," *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*, 2019.

[39] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[40] "Mb1202 datasheet." [Online]. Available: https://www.maxbotix.com/documents/I2CXL-MaxSonar-EZ_Datasheet.pdf

[41] "Rosserial package summary." [Online]. Available: http://wiki.ros.org/rosserial

[42] "Solidworks to urdf exporter." [Online]. Available: http://wiki.ros.org/sw_urdf_exporter

[43] "Cartographer." [Online]. Available: https://google-cartographer.readthedocs.io/en/latest/

[44] "Wiki." [Online]. Available: http://wiki.ros.org/rtabmap

[45] Filipenko, *Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment*, 2018.

[46] B. M. F. d. Silva, R. S. Xavier, and L. M. G. Gonçalves, "Mapping and navigation for indoor robots under ros: An experimental analysis." [Online]. Available: https://www.preprints.org/manuscript/201907.0035/v1