GrubTub

Authors: Sebastian Montiel, Michael Li, Advaith Sethuraman
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— We intend to provide a convenient solution to on-campus food delivery by creating a mobile robot capable of autonomously delivering multiple orders of food on CMU's campus while avoiding pedestrians on its journey.**

***Index Terms*—Autonomous Food Delivery, Social Navigation, Path Planning, Localization and Mapping, ROS, Robotics, Embedded Systems, Sensors, Controls, Optimization**

1 INTRODUCTION

On-campus food delivery is a niche problem that traditional food delivery services cannot fully satisfy. Food delivery in general is important because it saves users time and is convenient for busy schedules. Ideally, a user could walk to the door of their campus building and pick up their food, but with the long distance between certain buildings and open roads, using traditional food delivery can be as inconvenient as going to an on-campus restaurant.

Therefore, we propose a last-mile autonomous food delivery service that can better satisfy users looking to get convenient food while they are located on-campus. There are some competing services that are attempting to solve this problem as well. Some competitors use a human operator that places way-points frequently for the robot. Others are completely remote-controlled. Finally, certain competitors provide a fully autonomous solution to the problem.

We intend for our solution, GrubTub, to be an autonomous on-campus delivery robot that travels on sidewalks on campus, but at a tiny fraction of the budget and team size of our competitors.

To be a useful and easily adopted food delivery system, the autonomous system must meet a comprehensive set of metrics regarding delivery time, payload capacity, pedestrian avoidance, and delivery accuracy.

In our case, we've quantified the metrics for this system as:

1. The robot must be able to traverse all roads and successfully deliver orders following the other requirements in a predefined region of campus on flat ground 100% of the time.
2. The robot must be stay on the sidewalk at least 60% of the time.
3. The battery life of the robot must be at least 30 minutes.
4. The robot must delivery to a certain degree of accuracy- that is, within 3 meters of the drop-off point 100% of the time and within 1 meter of the drop-off point 68% of the time.
5. The robot must hold and transport at least $2kg$ of payload at a volume of $1ft^3$.
6. The robot must deliver its payloads intact 100% of the time.
7. The robot must stay at least 0.25 meters away from all other objects and people in transit.
8. The robot must have at most 1 human intervention per 50 meters across all of its deliveries.
9. The tail latency of the robot-ground station connection must be at most 605 ms.
10. The robot's delivery time for all deliveries must be less than the Human Round Trip Time (HRTT), measured in seconds.

2 DESIGN REQUIREMENTS

Requirement 1: We ensure that our robot can reach as many users as possible so that they can order from a more convenient location than if they had to use traditional food delivery or takeout. We must make sure that the robot can drive and deliver on all flat sidewalk roads on campus. This region is outlined in Fig. 1.The sidewalks and roads contained within the red bounded region are entirely on flat ground, and must be traversable by the robot. To test this, we will run the robot on every single one of the roads in the region with max load to make sure it can traverse them.

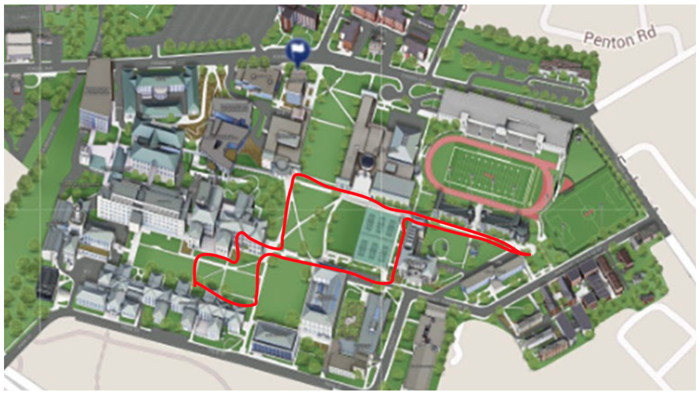


Figure 1: This Fig. shows the region of operation for our solution, outlined in red.

Requirement 2: Customers expect a level of speed and consistency from delivery services. Our robot should not wander off its path and slow down on its journey to the customer. As a result, we want the robot to use the sidewalk as much as possible. This way, we know that the terrain is flat, free of arbitrary obstacles, and relatively smooth. If we allowed the robot to go "off road", we would not have any guarantees on the type of terrain and the robot's speed over it. To obtain the requirement of at least 60% of the time on sidewalk, we used the following formulas We assume a desired speed of 0.7m/s, given *Requirement 10*, and average human walking speed of 1.4m/s. We assume the robot's maximum speed will be 1.0m/s. We chose this because it is near the average walking speed of humans. Any faster and it would be unnatural and dangerous. We assume the robot will travel twice as slowly on grass than on the sidewalk. In other words

$$\begin{aligned} \bar{s}_{robot} &= p * s_{sidewalk} + (1-p) * s_{grass} \\ 0.7m/s &= p * s_{sidewalk} + (1-p) * 0.5 * s_{sidewalk} \\ 0.7m/s &= p * 1.0m/s + (1-p) * 0.5 * 1.0m/s \\ p &= 0.6 \end{aligned} \tag{1}$$

Requirement 3: We want to ensure that our users do not have to wait for our robot to recharge its battery in order to get their delivery- this is a matter of timeliness. In this case, we must ensure that our battery can last through multiple orders. The longest delivery time between any two points within the region defined on the map in Fig. 1 is from Resnik Hall to the entrance between Porter and Baker Hall. On Google Maps, this route takes 8 min by walking, one-way. We must make sure to support multiple deliveries- at the bare minimum this means the battery must last through 2 deliveries. Assuming the average human walks at the same rate the robot travels, and that the robot starts from Resnik, the robot must travel from Resnik to the delivery point, back to Resnik, and back to the delivery point without running out of battery - 24 minutes. For slack, we've rounded this to an even 30 minutes of minimum battery life for our requirements. To test this, we will run the robot at max load until it runs out of battery to find out the battery's runtime.

Requirement 4: We must make sure that the user can have the utmost convenience when picking up their order. Therefore, they must be able to easily locate and pick up their order at a location close to their specified dropoff location. Thus, the robot must arrive close to the dropoff point. For this, we've devised a requirement that the robot must arrive within 3 meters 100% the time, and 1 meter at least 68% of the time. 3 meters is a reasonable distance for the user to conveniently grab their food, since it's the distance between a front door and the edge of the sidewalk- a common type of location in which food is delivered to. 1 meter is ideal: at that distance, a user can grab the food from arm's length. We have to be convenient all the time, so the robot must arrive within 3 meters 100% the time. For the 1 meter metric, we require that the robot must be equal to or better than a human delivery person when delivering within one meter of the dropoff point. Assuming that a human delivery person drops the order at a location specified by a bivariate Gaussian distribution across two random variables with a mean at the true delivery location, and a variance of one meter on each axis (due to human imperfection), the chance that the delivery person leaves the food within one meter of the actual delivery point is 68% (68–95–99.7 rule). Thus, to parallel a human in delivery accuracy, the robot must be able to arrive within 1m of the drop off location at least 68% of the time. We will test this with > 30 trials as suggested by the Central Limit Theorem [2].

Fig. 2 shows an example of a bivariate gaussian centered at a dropoff location of (1,1): $X \sim \mathbf{N}(\mu, \Sigma)$ with $\mu = [1,1]$ and $\Sigma = I_{2x2} * (1\text{meter})$. Each point (x,y,z) shows the probability of the dropoff, z, being at coordinate (x,y).

Requirement 5: The user must be able to order enough food to eat for this service to be satisfactory. For our solution, we must ensure that the robot can hold enough food for the user. We empirically measured the volume and dimensions of a single takeout container and found that it was significantly less than a volume of $0.25ft^3$ and a weight of $0.5kg$. The specific dimensions were 9" ∗ 9" ∗ 3". We want our solution to be able to handle multiple deliveries, so we decided to give our robot a max payload volume of $1ft^3$ and a weight of $2kg$, as this is enough for 3 deliveries' worth of payload. This will lead to a substantial delivery time speedup (since the robot can pickup and transport multiple orders) but also is small enough such that the payload can be fit onto the rest of our robot without being too large, heavy or expensive.

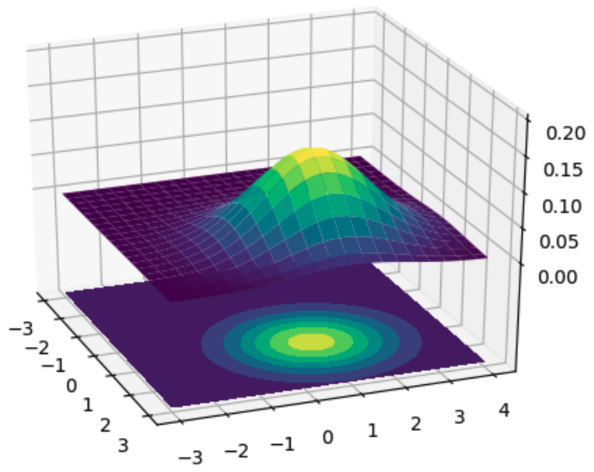


Figure 2: A bivariate Gaussian distribution

Requirement 6: The user will want to get their food delivered in an intact form, so we have to ensure that the food is intact upon every delivery. This requirement is one of quality- we assume that the vendor will pack their food sufficiently securely for delivery (as in the packaging can withstand basic movement and tipping) and require that our solution can deliver food intact with these assumptions for every delivery.

Requirement 7: The user would not want their robot hindered by bumping into random people and objects on its transit, as that would not only decrease the timeliness of the robot but also raise the chances of the food not being intact upon delivery. Additionally, people that the robot encounters during its transit will want to feel safe and not have to worry about the robot colliding with them.

In our case, we require our robot to stay at least 0.25 meters away from all other objects and pedestrians in transit at all times: given that the average human is 0.41 meters wide (we will round up to 0.50m) [6], half the width of a person is a reasonable distance for the robot to stop from a pedestrian and avoid collision.

Requirement 8: We must guarantee that the delivery will succeed all the time, per Requirement 1. Therefore we must account for situations in which the robot is incapacitated (i.e. it's stuck) and intervene to help the robot manually in order to let the user get their food 100% of the time.

For the human intervention requirement we've required that the robot must have at most 1 human intervention per 50 meters of distance travelled on average across all deliveries.

The reasoning behind this is based of the human intervention rate set by a competitor company, Kiwi. They've stated that their robot operators will set a manual waypoint once every 5 seconds CITE HERE, so assuming Kiwi's robots travel at 1 m/s, Kiwi's solution will have one intervention per 5 meters. We want to be 10 times better than this, and therefore specify that our maximum intervention rate is 1 per 50 meters.

Requirement 9: We must guarantee that the delivery will succeed all the time, per Requirement 1. Also, in case of dangerous situations, there should be a way to safely operate the robot. Therefore, we must have an emergency manual operation in case the autonomous operation fails for some reason. For this, we have a ground station that gives orders and emergency controls. If the latency between robot and ground-station is too large, the ground-station will not be able to update the robot or transfer emergency control to an operator before the robot runs off-path or damages itself. To prevent this, we require that the tail-latency, i.e. the duration of the longest request, is lower than 605ms. Given the speed that the robot travels, this would allow the ground-station to notify the robot before it travels off of a sidewalk. We assume the robot is travelling at a max speed of 1.0m/s, the sidewalk is 1.21m wide [3]. We assume the robot is travelling in the middle of the sidewalk. We use the following equation to find the intervention time until the robot drives off the sidewalk.

$$\begin{aligned} t_{intervention} &= \frac{w_{sidewalk}}{s_{max}} \\ t_{intervention} &= \frac{0.605m}{1m/s} \\ t_{intervention} &= 605ms \end{aligned} \tag{2}$$

Requirement 10: To generate value for the customer, the product should save them time. Ideally, the robot should delivery food faster than it takes the human to walk from their location on campus to the restaurant, and back. This provides justification for the user to use our solution, as it's more convenient than manually getting the food.

In our case, we upper bound our delivery time by this expectation: for each delivery, the robot must deliver its food from a restaurant to a delivery point within the Human Round Trip Time (HRTT) of the delivery point and the restaurant.

(The HRTT from point A to point B is defined as the time it takes a human to walk from A to B, and back to A- it acts as a quantitative measurement to the time a user takes to manually pick up food.)

This metric provides a soft requirement on the average speed of the robot in terms of the average human speed. Given that *Requirement 10* specifies the relation between robot travel time and human travel time, and that the relationship between distances are known, the relation between average speeds are also known

$$\begin{aligned} t_{robot} &= \frac{t_{human}}{2} \\ d_{robot} &= \frac{d_{human}}{2} \\ \bar{s}_{robot} &= \frac{\bar{s}_{human}}{2} \\ \bar{s}_{robot} &= \frac{1.4m/s}{2} \\ \bar{s}_{robot} &= 0.7m/s \end{aligned} \tag{3}$$

This quantity is used above in (1).

3 ARCHITECTURE OVERVIEW

3.1 High-level Overview

For our solution, we decided on developing both a robot and a ground station to implement autonomous order delivery. The robot and ground station operate together to successfully deliver orders to the user in a fashion that meets our requirements. The user will place orders with the ground station, the ground station comes up with the minimum-distance series of waypoints that the robot should follow to deliver any placed orders, and the robot plans its path to drive through that series of waypoints while avoiding any pedestrians. Fig. 4 and Fig. 5 contain our system diagram detailing the information flow between

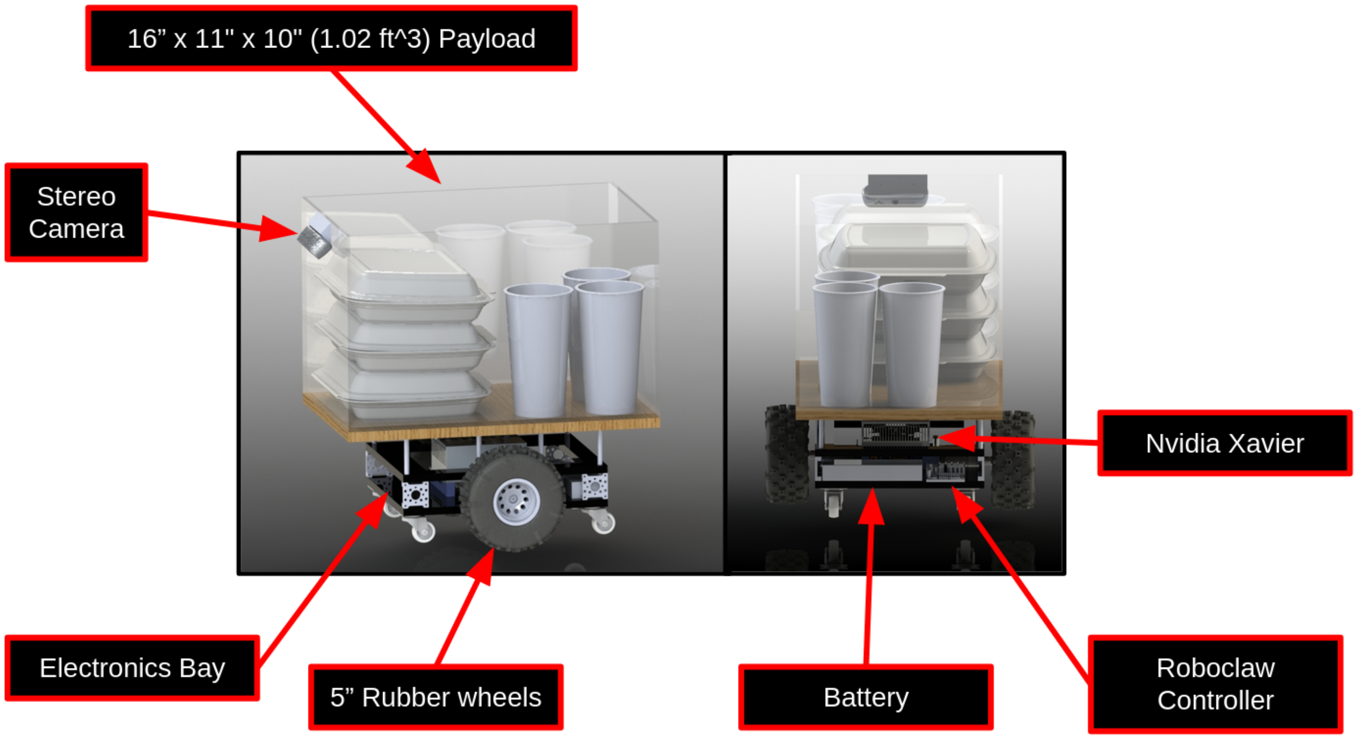


Figure 3: GrubTub's mechanical design

the robot and ground station, as well as the flow between the components running on each.

Robot: The robot's flow of information is to take in incoming information from the ground station, such as waypoints to drive to and possible emergency controller inputs, and output heartbeats back to the ground station that give telemetry such as its location, status, and logs.

Using its incoming information to set its destinations, the robot is responsible for planning its path and driving from waypoint to waypoint while meeting our requirements, sending heartbeats back to the ground station regularly, and overriding its autonomous operation with any emergency controls if there are any.

Ground station: The ground station's flow of information is to take in heartbeats from the robot and keep track of the robot's location and status, and based on those inputs send it information like waypoints to go to and possible emergency controls if the robot is in a dangerous state.

3.2 Quick Introduction to ROS

For this project, we've decided to use ROS (the Robot Operating System) as a framework for our software. ROS is an open-source robotic application framework which provides a modular paradigm for developing our robot as well as a robust selection of tools and libraries that supplement development.

The ROS runtime acts as a publish-subscribe distributed system: each sensor, actuator, and high-level element of the robot can be encapsulated within a ROS node. These nodes can subscribe to and receive messages from ROS topics, and publish messages on topics.

This offers us a very clean way of developing our software: each sensor can have a node that contains its driver and publishes its data, each actuator can have a node that contains its driver and subscribes to commands, and each high-level element can have a node that reads any sensor data and drives any actuators of its choice.

With ROS, we can divide our work cleanly across components and rigorously test each node separately in a simulated environment, which not only makes the division of labor clear but also allows us to develop and test our software in true parallel without many blockers until integration.

3.3 Mechanical Overview

The mechanical design in Fig. (3) for GrubTub uses a minimalist design language and meets all the requirements. The payload container is large enough to meet the volumetric requirements and can also hold multiple deliveries as motivated by *Requirement 5*. The design has an area to house the electronics safely, including the Xavier, motor controller, sensors, and batteries. The wheels were chosen for their grip and ability to traverse small bumps on the flat sidewalk. The design only has two drive wheels because of the requirement from the software systems. The robot will use a differential drive paradigm, so only one wheel is needed on either side. When an order is placed to the groundstation, the restaurant will place the orders within the payload container. After the robot has driven

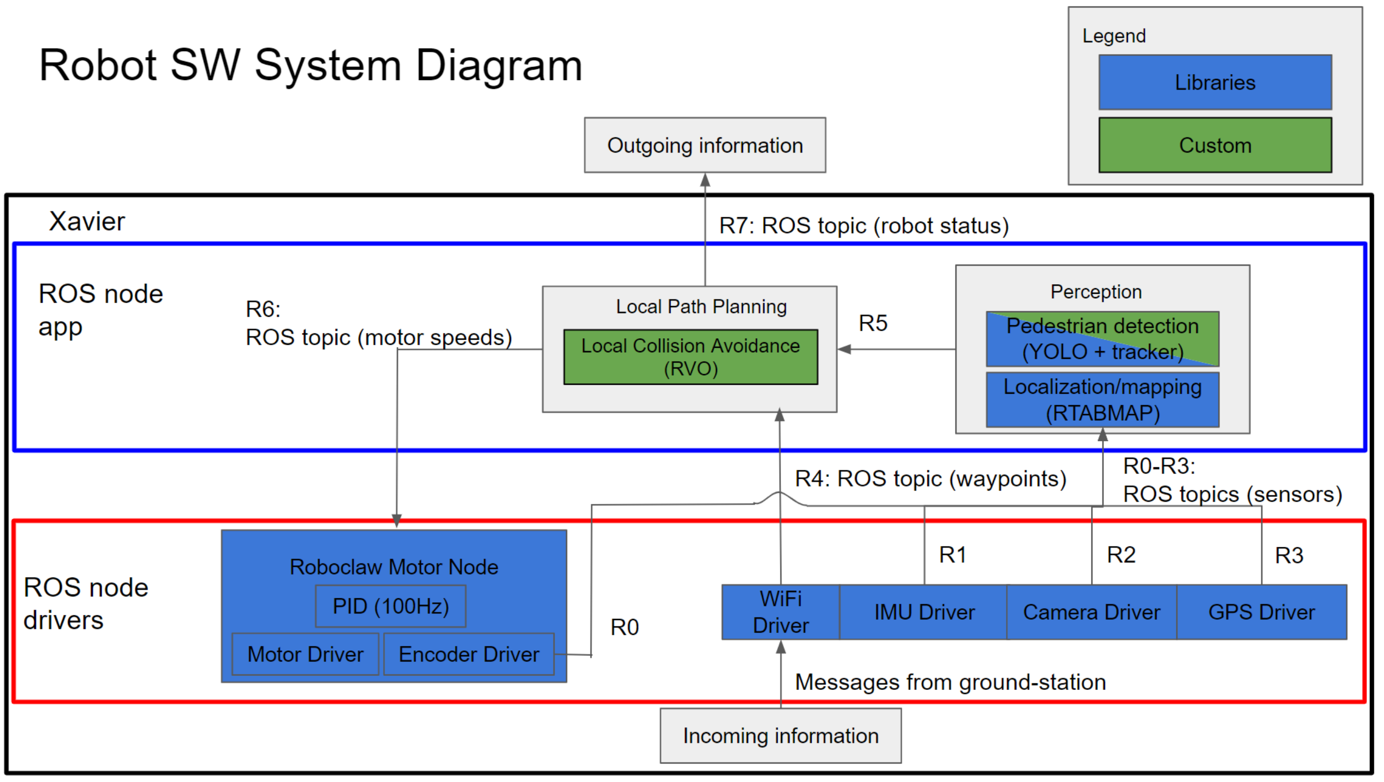


Figure 4: Diagram for the software running on the robot

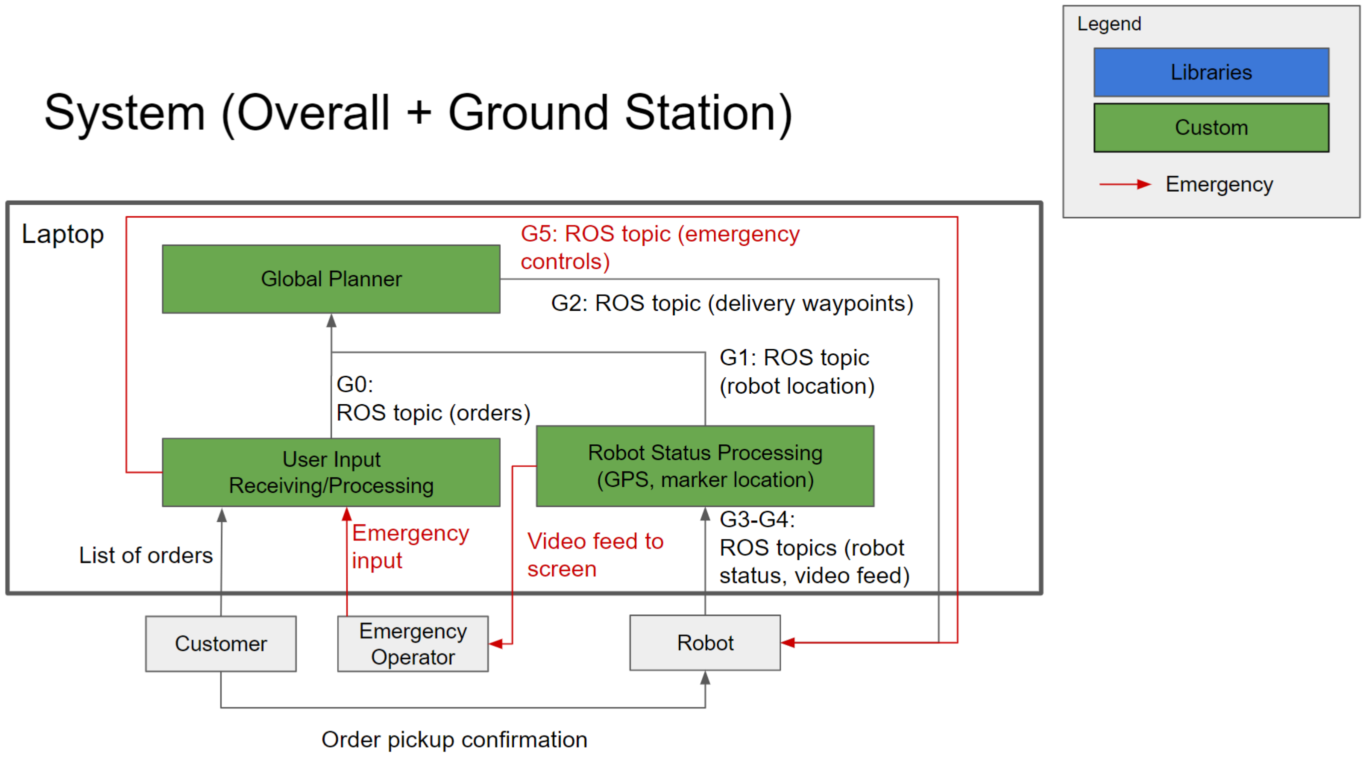

Legend
Libraries
Custom
Emergency
System (Overall + Ground Station)
Laptop
G5: ROS topic (emergency controls)
Global Planner
G2: ROS topic (delivery waypoints)
G1: ROS topic (robot location)
G0: ROS topic (orders)
User Input Receiving/Processing
Robot Status Processing (GPS, marker location)
List of orders
Emergency input
Video feed to screen
G3-G4: ROS topics (robot status, video feed)
Customer
Emergency Operator
Robot
Order pickup confirmation

Figure 5: Diagram for the groundstation and overall system

to its dropoff location, the user will take their order from the payload container.

3.4 Architectural Changes since the Proposal Presentation

We have made only one small change to our software architecture, and it is due to a hardware upgrade. Prior to our choice of the Roboclaw motor control, we would have had to write our own PID, encoder, and motor drivers. The Roboclaw, however, has an onboard PID controller and a library ROS node.

Since our proposal presentation, we have also made a number of changes on the mechanical and hardware side, which we will explain in Sections 4 and 5.

4 DESIGN TRADE STUDIES

The choice of our components in inherently coupled. The terrain determines the requirements on our motors. The motors' max current draw determines the requirements on our motor controller and the total capacity of our batteries. Furthermore, the mechanical design also changes the parameters for all of the choices listed above. We decided to first fix the mechanical design, then determine the motors, motor controller, and battery. Our priority was to keep the mechanical/electrical design simple so that we could focus our energies into the Software.

4.1 Motor Selection

Our choice of motor is limited by our budget and determined by *Requirement 1*, *Requirement 3*, and *Requirement 10*. The motor must be able to produce enough torque to traverse the area we desire to delivery to. The motor and battery combination must be efficient enough to allow for at least 30 minutes of battery life, and the motor must be able to meet the speed requirement of at least 0.7m/s. Furthermore, we would prefer motors that have encoders attached to them. This saves us the hassle and setup of discrete encoders. The encoders are necessary for controlling wheel velocities and also for the localization algorithm, as mentioned in Fig. 4, signals R0, R6.

First, let us consider the torque requirement. We measured the steepest slope on campus (between Doherty and Wean), and found that $\theta_{slope} = 21°$. We made the following assumptions: robot mass $M_{robot} = 2kg$, payload mass $M_{payload} = 3kg$, wheel radius $r = 0.0685m$. To find the necessary motor torque τ to traverse this, we used the following Equation (4):

$$\begin{aligned}\tau &= 0.5Fr \\ \tau &= 0.5[mg\tan(\theta_{slope})]r \\ \tau &= 6.585\ kgf \cdot cm\end{aligned} \tag{4}$$

We require a motor that can produce at least $6.585 kgf \cdot cm$ of torque. We were able to narrow down a series of DC motors that were compatible with this requirement. We chose the Pololu 37D series of motors for their high torque capabilities and high variety of speed. Naturally, we wanted to maximize the amount of torque the motor could output, so we chose the model with the highest torque. This motor had a 150:1 gearbox ratio, and produced $49 kgf \cdot cm$ of torque. We were confident that this would be able to traverse the slopes. However, we had to change our part when considering the motor speed requirement. Equation (5) describes the motor's angular speed requirement in terms of the robot's linear max speed requirement determined in *Requirement 2* We require a max speed $s_{max} = 1.0 m/s$, and assume the same wheel radius as before.

$$\begin{aligned}\omega &= \frac{(60 s_{max})}{2\pi r} \\ \omega &= 139.2\ rpm\end{aligned} \tag{5}$$

The 150:1 ratio 37D motor's max speed was 67rpm. Unfortunately this was not fast enough to provide the required speed. In order to choose a motor that was both fast enough and had enough torque, we had to reduce the gearbox ratio. To maximize torque, we simply found the highest ratio motor that met our speed requirement. This was determined to be the 50:1 37D motor. This motor produces 6 $kgf \cdot cm$ of torque at 139rpm, with a stall torque of 21 $kgf \cdot cm$ and while getting over the hill, runs at 125 rpm. This means that it produces more than enough torque to operate on flat ground at $1.0 m/s$ and also enough torque to traverse up the slopes.

We considered other motors as well. A class of DC motors with gearboxes called hub motors are commonly found in children's miniature electric vehicles. We were able to find spares on Amazon and considered using them, as they produced enough torque to transport a small child up slopes. This fact was qualitatively determined by examining videos of reviews on Amazon. However, we decided against this approach because it was difficult to mount encoders on the motor hubs. The importance of the encoders to both the velocity control algorithms and localization algorithms outweighed the torque capabilities of these new motors.

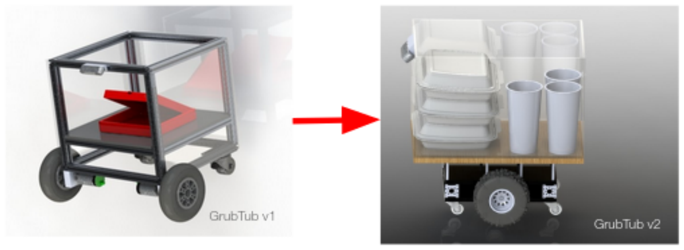


Figure 6: An iteration of our design from our proposal presentation to the present.

4.2 Motor Controller Selection

After finalizing the motors, we moved to the motor controllers. The Pololu 37D 50:1 gearmotors have a stall current draw of 15.5A each. Our design uses 2 motors, so our max current draw from the motors will be 11A. We considered several options for motor controllers. The Capstone Inventory had a L298N based motor controller. We also considered the Roboclaw class of motor controllers because it was available from a previous project. We eliminated the L298N motor controller because its max current rating was 2A per motor [8]. Since our motors operating point was around 1.5A each, we decided the motors would fry the motor controller if they experienced load. We decided the RoboClaw 2x15A motor controller was perfect for our uses. It provides enough current per motor.

4.3 Battery Selection

After finalizing the motors and compute, we were able to estimate the nominal current draw for the system. The Xavier draws $I_X = 3.38A$, while the motors draw around $I_M = 4.8A$ total under normal operation. Equation (6) shows the relation between our battery capacity C_{bat} and the rest of our finalized components. The requirement for operating time $T = 0.5h$ is given by *Requirement 3* A scaling factor of $\rho = \frac{1}{0.5}$ is used to express inefficiencies in the power system that will increase the total capacity needed.

$$\begin{aligned} C_{bat} &\geq \rho T(I_X + I_M) \\ C_{bat} &\geq 8.18Ah \end{aligned} \tag{6}$$

We rounded up to the next highest class of batteries, 10Ah. We chose Lithium Polymer batteries because of their high energy density. Our motors can draw up to 11A total at a time, so we needed a LiPo battery with a discharge rating (C) of greater than 11. We found a suitable battery, the Turnigy 10000mAh 4S 12C. However, with our simple LiPo charger, this would have taken 10 hours to charge. In order to reduce the time for charging during testing, we decided to use parallel batteries. We split 10000mAh into two 5000mAh batteries. This way, when testing, we could hot-swap the two batteries, and during full trials, we could use both of them. We finally decided on the Turnigy 5000mAh 3S 20C LiPo battery. Our motors run at a nominal voltage of 12V, so we chose our battery to be 11.1V.

4.4 Robot Chassis Tradeoffs

We designed our chassis in Solidworks in order to make sure we met the volumetric requirements. Our initial design in Fig. (6) was built using aluminum 8020 extrusions, and involved several custom parts. The design using 8020 would allow us freedom in construction, since you can create any shape you want out of it. However, the drawback was that it would take several parts to fasten at the corners, and a lot of hardware (nuts and bolts) to construct.

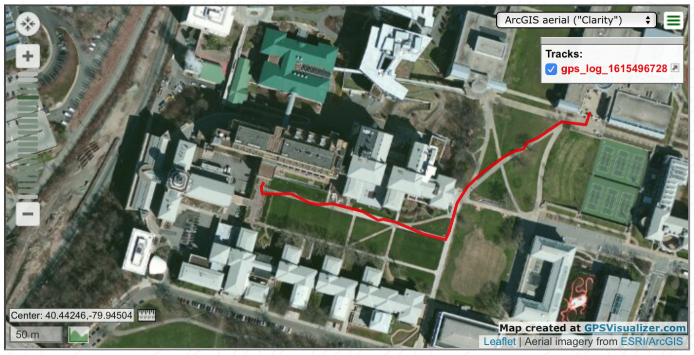

ArcGIS aerial ("Clarity")
Tracks:
gps_log_1615496728
Center: 40.44246,-79.94504
50 m
Map created at GPSVisualizer.com
Leaflet | Aerial imagery from ESRI/ArcGIS

Figure 7: The raw NMEA data from our GPS sensor superimposed onto campus satellite view.

Therefore, we decided to simplify this design by removing the custom parts and replacing them with a pre-made chassis and tub. We expect this to reduce the amount of time we spend in assembly. Design v1 costs \$211.38, whereas design v2 costs \$180.24. Design v1's estimated build time is 3 days (have to laser cut acrylic, and cut extrusions), whereas design v2's estimated build time is 1 day (only have to fasten nuts and bolts). Although we have a slight reduction in cost, we are primarily optimizing for convenience here. Given enough time to design and build a robust robot, we would have opted for a custom option. However, we plan on using our simple design as a showcase for our software algorithms.

5 SYSTEM DESCRIPTION

5.1 Hardware Architecture

Our robot hardware is designed to be modular and easy to assemble. We use off the shelf components that are compatible with each other and meet the requirements.

The *Nvidia AGX Xavier* is the most powerful embedded Linux platform we could find for our project (running JetPack). We needed a Linux platform so we could use ROS, which makes integrating and coordinating our hardware much simpler. The Xavier has more than enough compute for the algorithms we intend to run, and has enough interface for all of our sensors and actuators.

BNO055 IMU is the IMU we chose because of its driver availability and the fact that Advaith already had one. It uses the I2C protocol to communicate with a master and provides angular velocities, linear accelerations, and magnetometer readings. The magnetometer readings are needed because the GPS localization system only provides location, not a heading. The absolute earth-referenced heading is in the ENU frame (East-North-Up), and is a requirement from the robot_localization node.

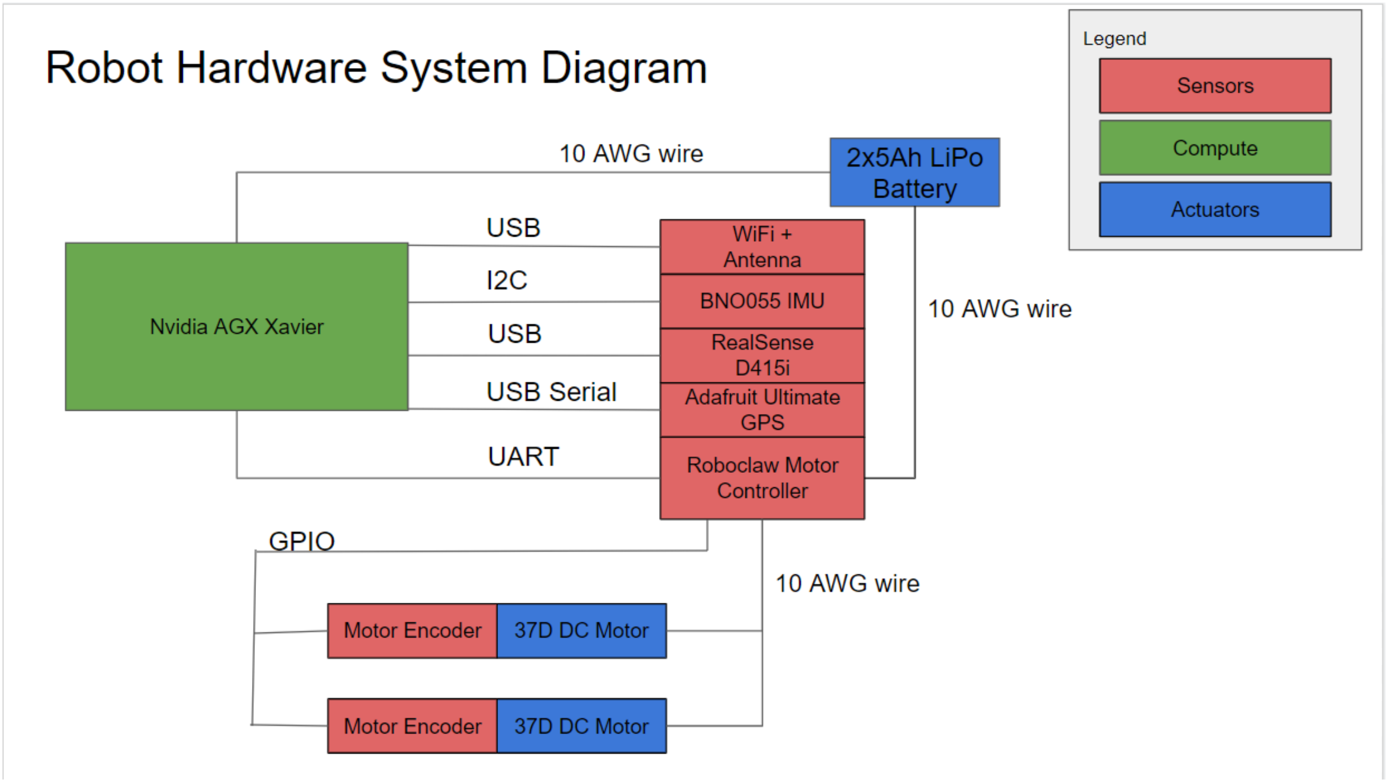


Figure 8: Specific hardware and interfaces for the GrubTub robot. All hardware was purchased; we did not design any individual component.

Intel RealSense D435i is a stereo camera that RTABMAP will use to create and localize itself within a map. We chose this camera because we had easy access to it from CMU's Roboclub, and Advaith had a personal copy. It is also useful for the perception system, since we will need a distance to pedestrians for the local planner algorithm. Furthermore, it will help us validate our *Requirement 7.*

Adafruit Ultimate GPS is a low-cost GPS sensor. We chose the sensor with a built in USB-Serial converter. This means that the sensor outputs NMEA 0183 sentences directly into the serial output. The frequency is around 1Hz, and with empirical testing, the accuracy seems reasonable for our application. The results are shown in Fig. 7

Roboclaw Motor Controller has a lot of built-in features that makes it a good choice to drive our motors. It has an on-board PID controller, supports the currents and voltages our motors need, and has a library driver and ROS node. [1]

Pololu 37D 50:1 12V Encoded Motors: the choice for these motors is elaborated upon in the Trade Studies section. These motors have a stall current draw of 5.5A, and meet our requirements for torque and speed. They have built-in encoders that operate over GPIO.

5Ah 3S 20C LiPo Batteries were chosen to power the robot because by having two batteries, we can always have one battery charging while using the other in testing. Additionally, we can attach the batteries in parallel to reach our minimum battery life requirement. As described in the trade studies, the batteries meet the requirements from the DC motors in terms of current draw and voltage.

Geekworm WiFi Adapter was chosen to add wireless communication to the robot since it was relatively cheap, had an antenna, and came with a driver intended for embedded Linux platforms.

5.2 Software Architecture of the Robot

For the robot to be able to drive through a series of waypoints autonomously, it needs high-level perception and planning, supplemented by sensor inputs from a variety of sensors, as well as an output to a motor controller.

Each of the software components for the robot shown in Fig. 4 represents a ROS node with a specific function. Every ROS topic within the signals $R*$ referenced in the Software System Diagrams (4 and 5) and these descriptions is explicitly defined and documented in Appendix A.

Local Path Planning is a custom node which drives the robot from global waypoint A to global waypoint B. It receives the next waypoint to drive to and pedestrian perception data as input from signals R4 and R5, respectively, and outputs motor velocities to the Roboclaw Motor Node such that the robot drives while avoiding collisions through R6. The node is abstracted such that a specific class of collision avoidance algorithms called social navigation algorithms can be swapped out. We are using a simple one called Reciprocal Velocity Objects [19]. If we find the time to, we can replace this with a Reinforcement Learning algorithm. The input to these algorithms are pedestrian locations/velocities and robot goal. The local planner will output the desired robot frame linear and angular velocities for that time step. It additionally outputs a heartbeat with telemetry for the ground station through topics within signal R7.

Perception provides the Local Path Planning node with detailed information about where the robot is on the map and what's around the robot, providing the information through topics in signal R5. There's two nodes which supply perception data: Localization and Mapping, for localizing the robot within a static map of our delivery region, and Pedestrian Detection, for detecting and tracking where nearby pedestrians are. Pedestrian detection will use the popular YOLO Convolutional Neural Network [9]. The pedestrians will be localized in 3D using the stereo camera's disparity map. They will be tracked with velocity using a Kalman Filter tracker. The Localization and Mapping node is a library node and the Pedestrian Detection node is a custom node which utilizes existing algorithms for its purposes.

We're using the RTABMAP ROS Library Node for SLAM [15].

Roboclaw Motor Node is a library ROS node which acts as both PID motor controls for the robot and a driver for the physical motors. It receives velocities from the Local Path Planning node through topics in signal R6 and passes them to its drivers, using the Roboclaw Motor Controller to control the motors. It also contains an Encoder Driver that outputs raw encoder data through topics in the R0 signal. We're using the ROS node documented and implemented at [1].

WiFi Driver enables ROS message sending between nodes on the ground station and nodes on the robot. It's abstracted out by the ROS runtime so that our code and the ROS library nodes never have to directly interact with it, but it facilitates all incoming information and outgoing information between the robot and ground station.

IMU Driver is a ROS library node which outputs raw sensor data through topics in the R1 signal. It communicates with the IMU through I2C, wired at the Xavier GPIO pins. The library ROS Node is documented at [11].

Camera Driver is a library ROS node for our RGBD camera that outputs raw camera data across a variety of ROS topics for the Perception node through topics through signal R2. The library ROS node is documented at [4].

GPS Driver is a library ROS node that outputs raw NMEA sentences from our GPS sensor through topic R3. The node is documented at [13].

Data flow from the Encoder Driver/IMU Driver/GPS Driver to Perception: For the encoder, IMU and GPS, we need to filter the noise, raw data. We'll use a library ROS node to take in and filter all the raw sensor data and output smoothed, processed data that goes into the Perception node, as shown in Fig. 9. This involves using the *robot_localization* library, which has every node we need for combining these raw sensor inputs and processing them. The process is documented in detail at [12].

5.3 Software Architecture of the Ground Station

The ground station has two roles to fulfill: it must send the robot batches of waypoints to drive through to fulfill user orders and also send it any emergency inputs that an emergency operator may manually input.

Each of the software components for the robot shown in Fig. 4 represents a ROS node with a specific function. Every ROS topic within the signals denoted by $G*$ referenced in the Software System Diagrams (4 and 5) and these descriptions is explicitly defined and documented in Appendix A.

Global Planner is a ROS node that calculates and sends out batches of waypoints for the robot to fulfill orders through a topic in signal G2. It takes in robot heartbeats/status updates from signal G1 and any user orders from signal G0 and batches the orders, calculating the optimal series of waypoints to fulfill a batch of orders in a way that meets requirement 10 while minimizing the robot's total distance travelled from its current location. It outputs batches of delivery waypoints for the robot.

The actual Global Planner is a ROS Node that wraps around a novel optimization algorithm which outputs a satisfactory series of waypoints for the robot to travel through and minimize total distance travelled given a set batch of orders. The ROS Node itself will batch incoming orders and only send new batches of waypoints when the robot is done with the current batch so that it never calculates waypoints on an undefined, partially complete robot state.

It uses dynamic programming over all possible robot states to minimize the total distance travelled by the robot over a set graph of the delivery region.

Robot Status Processing is a ROS Node but that receives raw heartbeats and video feed from the robot from signals G3 and G4. It processes the heartbeats and sends clean status info like the robot's node location to the Global

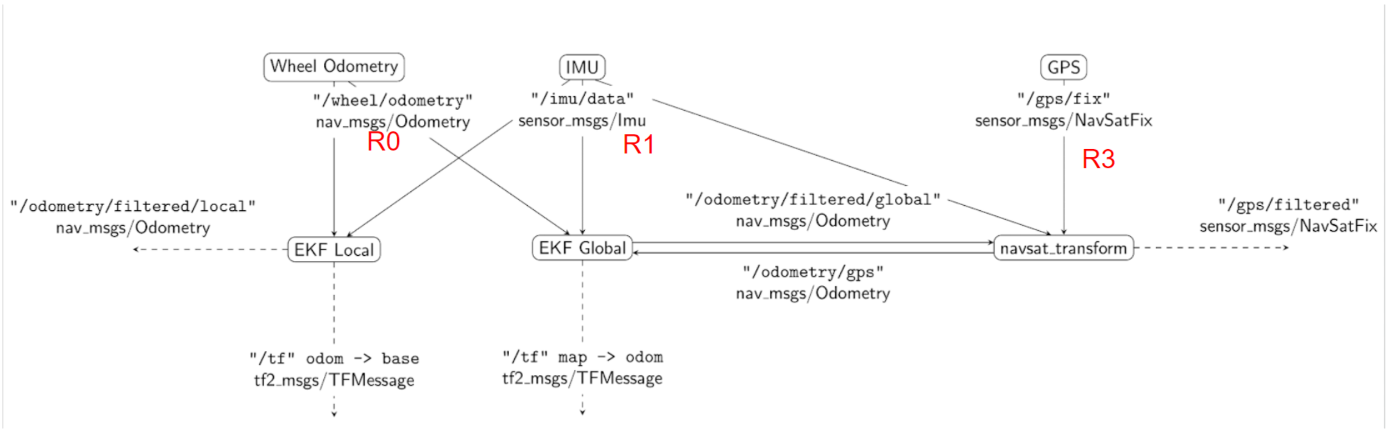


Figure 9: This figure, drawn from a ROS tutorial, shows the details of how we will filter our sensor data (signals R0,R1,R3) by passing it to the robot_localization node.

Planner node through signal G1, and passes the raw video feed to the screen for the emergency operator to see.

User Input Receiving/Processing is a ROS Node. The user can send the laptop orders, either on the laptop or through some communication, and this node sends to the Global Planner node locally on the laptop through signal G0.

An Emergency Operator may send the laptop some emergency controls through a joystick, in which case the laptop can send velocity commands directly to the robot through the ROS runtime through topics on signal G5.

6 PROJECT MANAGEMENT

6.1 Schedule

See Fig. 10 for our schedule breakdown.

6.2 Team Member Responsibilities

Sebastian is primarily responsible for network design, configuration and testing; and hardware driver integration. He is secondarily responsible for the pedestrian avoidance algorithm. Our hardware has many helpful libraries so he might take on additional work in system integration or wherever needed.

Michael is primarily responsible for the multi-order planning algorithm, sensor data scrubbing as described in Fig. 9, and system integration. He is secondarily responsible for networking and the localization algorithm.

Advaith is primarily responsible for the localization algorithm and the pedestrian avoidance algorithm. He is secondarily responsible for the low-level controls, which has reduced in scope since the Roboclaw has an onboard PID that needs to be properly configured.

6.3 Budget

See Fig. 11 at the end of this document for our budget and parts list. It does not include the spare motor that we have already ordered.

6.4 Tools and Assembly

We will be requiring several tools to assemble our robot. For our chassis, we will be using screwdrivers, drills, and wrenches. For our electronics, all our components are off the shelf, so we will be needing soldering irons, as well as breadboards. We will require power supplies for testing the motors and motor controllers before integration with the battery. For developing our software, we will be using the ROS environment that uses both Python and C++. This can be developed in any text editor and runs on a Linux system. Simulation work will be done in MATLAB/Gazebo. For testing, we will be using a measuring tape to measure the accuracy of the robot in *Requirement 4*, and a stopwatch to measure *Requirement 1, 2, 10.*

6.5 Risk Management

There are many risks associated with the development of an autonomous robot. As elaborated upon in our Design Trade Studies, we had to carefully choose our motors and batteries to meet our requirements while meeting our budget cap, and also weigh the risks of buying parts for and building our own chassis from scratch versus buying and modifying a premade chassis. Ultimately we settled on a premade chassis in order to save both money and time.

Additionally, perception can be difficult in an outdoor environment so we had to mitigate the risk of having our visual odometry fail by having a backup plan of using fiducial markers around campus and hard-coding the map for the robot.

Advaith mitigated potential problems with outdoor lighting variance messing with our visual odometry by attaching a halo light ring around our stereo camera in order to minimize lighting differences throughout the day.

7 RELATED WORK

- Kiwi: A company developing a semi-autonomous on-campus delivery robot. Kiwi uses operators to periodically set waypoints for their delivery bots. Our system differs in that we strive for minimal human intervention. [5]
- Starship: A company developing an autonomous on-campus delivery robot. Their system uses a combination of lidar and depth cameras for vision, while ours only uses depth cameras. Their system is also restricted to a 4 mile radius. [18]
- Nuro: An autonomous delivery robot that travels on open roads. Our system differs in that it is smaller and intended to drive on sidewalks, allowing it to drop off closer to customers. [7]
- refraction.ai: An autonomous last-mile delivery robot that travels in car and bike lanes. Again, our system differs in that it is intended to drive on sidewalks to allow delivery close to doors. [10]

8 SUMMARY

We hope to build a personable robot that meets the customer requirements. Currently we are testing our high-level algorithms and beginning assembly of the robot. We've already started integration of our sensors into the Jetson Xavier and we hope to finish on-time and with a functional robot that can deliver food efficiently and successfully on CMU's campus.

References

[1] Brad Bazemore. *Roboclaw ROS Node.* URL: `https://github.com/sonyccd/roboclaw_ros`.

[2] Boston University Medical Campus. *Central Limit Theorem Sample Sizes.* URL: `https://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Probability/BS704_Probability12.html`.

[3] dimensions.com. *Sidewalk Walkway Layouts.* URL: `https://www.dimensions.com/collection/sidewalk-walkway-layouts`.

[4] Intel. *Realsense ROS Node.* URL: `https://github.com/IntelRealSense/realsense-ros`.

[5] Kiwi. *Kiwi.* URL: `https://www.kiwibot.com/`.

[6] Ogden CL. McDowell MA Fryar CD. *Anthropometric reference data for children and adults: United States, 1988–1994.* Library of Congress, 2009.

[7] Nuro. *Nuro.* URL: `https://nuro.ai/`.

[8] Pololu. *L298N Specification.* URL: `https://www.pololu.com/product/924/specs`.

[9] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *arXiv* (2018).

[10] refraction.ai. *refraction.ai.* URL: `https://www.refraction.ai/`.

[11] ROS. *BNO055 I2C IMU Driver.* URL: `https://github.com/dheera/ros-imu-bno055`.

[12] ROS. *Integrating Sensors.* URL: `http://docs.ros.org/en/melodic/api/robot_localization/html/integrating_gps.html`.

[13] ROS. *NMEA Navsat ROS Node.* URL: `http://wiki.ros.org/nmea_navsat_driver`.

[14] ROS. *Odometry Message Documentation.* URL: `http://docs.ros.org/en/api/nav_msgs/html/msg/Odometry.html`.

[15] ROS. *RTABMAP ROS Node.* URL: `http://wiki.ros.org/rtabmap_ros`.

[16] ROS. *TF.Message Documentation.* URL: `http://docs.ros.org/en/jade/api/tf2_msgs/html/msg/TFMessage.html`.

[17] ROS. *Twist Message Documentation.* URL: `http://docs.ros.org/en/api/geometry_msgs/html/msg/Twist.html`.

[18] Starship Technologies. *Starship Technologies.* URL: `https://www.starship.xyz/`.

[19] J. van den Berg, Ming Lin, and D. Manocha. "Reciprocal Velocity Obstacles for real-time multi-agent navigation". In: *2008 IEEE International Conference on Robotics and Automation.* 2008, pp. 1928–1935. DOI: `10.1109/ROBOT.2008.4543489`.

Appendix A: ROS Topics

Ground Station

G0: Orders sent to the ground station

Messages on this topic are type *order*. This type contains *int id*, *int node* and $float\ weight$.

G1: Robot status information sent to the ground station

This signal is identical to R7. Messages on this topic are type *robotLocation*. This type contains an *int location*.

G2: Waypoints sent to the robot

Messages on this topic are type *batch*. This type has an array of *waypoint* messages. The *waypoint* type is comprised of $int32$ *node* and *string action*. This signal is also R4 in the robot's perspective.

G3: Robot heartbeats/status updates

The robot will send status updates over topics on this signal periodically such as its raw location as a coordinate frame transformation of type $tf2.msgs/TFMessage$, documented at [16].

Robot

R0/R1/R3: IMU/Encoder/GPS Driver Filtered Readings

Raw sensor readings from the IMU, Encoder and GPS are combined as shown in Fig. 9 to become a filtered odometry message for the Perception node of type *nav_msgs/Odometry*, documented at [14].

R4: Waypoints from the ground station

R4 is G2, except it's now in the robot's frame of reference. Messages on this topic are type *batch*. This type has an array of *waypoint* messages. The *waypoint* type is comprised of $int32$ *node* and *stringaction*.

R5: Perception readings to the Local Planner Node

The Local Planner Node takes a variety of topics across this signal from Perception. It takes in a tf coordinate transformation (*map* $\Rightarrow$ *base_link*) from the Localization/mapping part of Perception that tells it where the robot is in the map of type $tf2.msgs/TFMessage$, documented at [16], as well as filtered odometry of type *nav_msgs/Odometry*, documented at [14]. It takes in pedestrian information as well, which is an array of *PedestrianState* structs. Each *PedestrianState* struct contains information about where a pedestrian is relative to the robot. Specifically, it contains the pedestrian positions and velocities $\begin{bmatrix} x\ y\ \dot{x}\ \dot{y} \end{bmatrix}^T$.

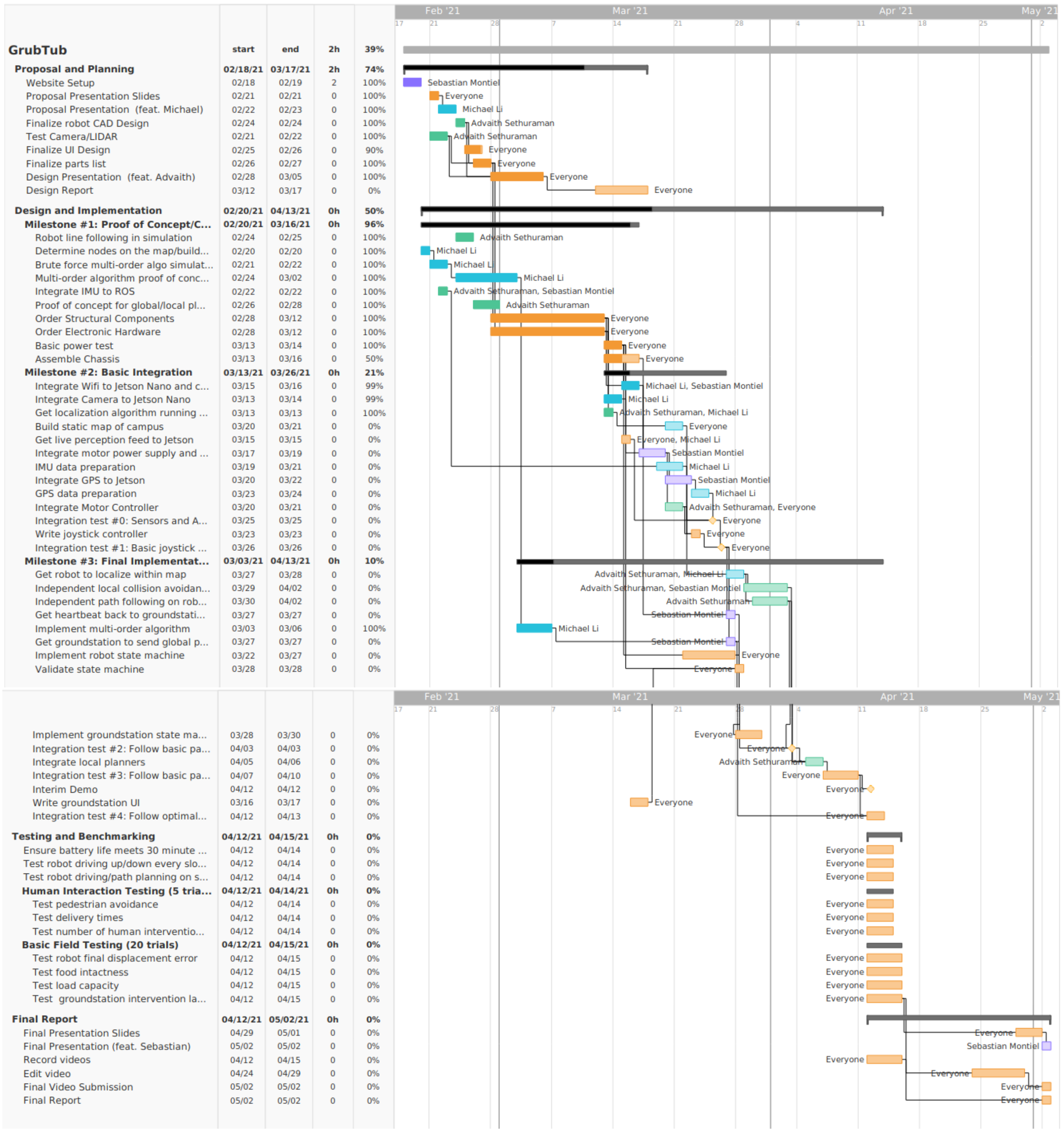
The node also needs access to the robot's velocity in the map frame, so it will require Odometry messages from the encoder and IMU of type *nav_msgs/Odometry*, documented at [14]

R6: Motor speeds to the RoboClaw Motor Node

This topic sends messages of type *geometry_msgs/Twist*, which specify linear and angular velocity. The Roboclaw Motor Node listens to this topic and controls the motors accordingly. The type is documented at [17].

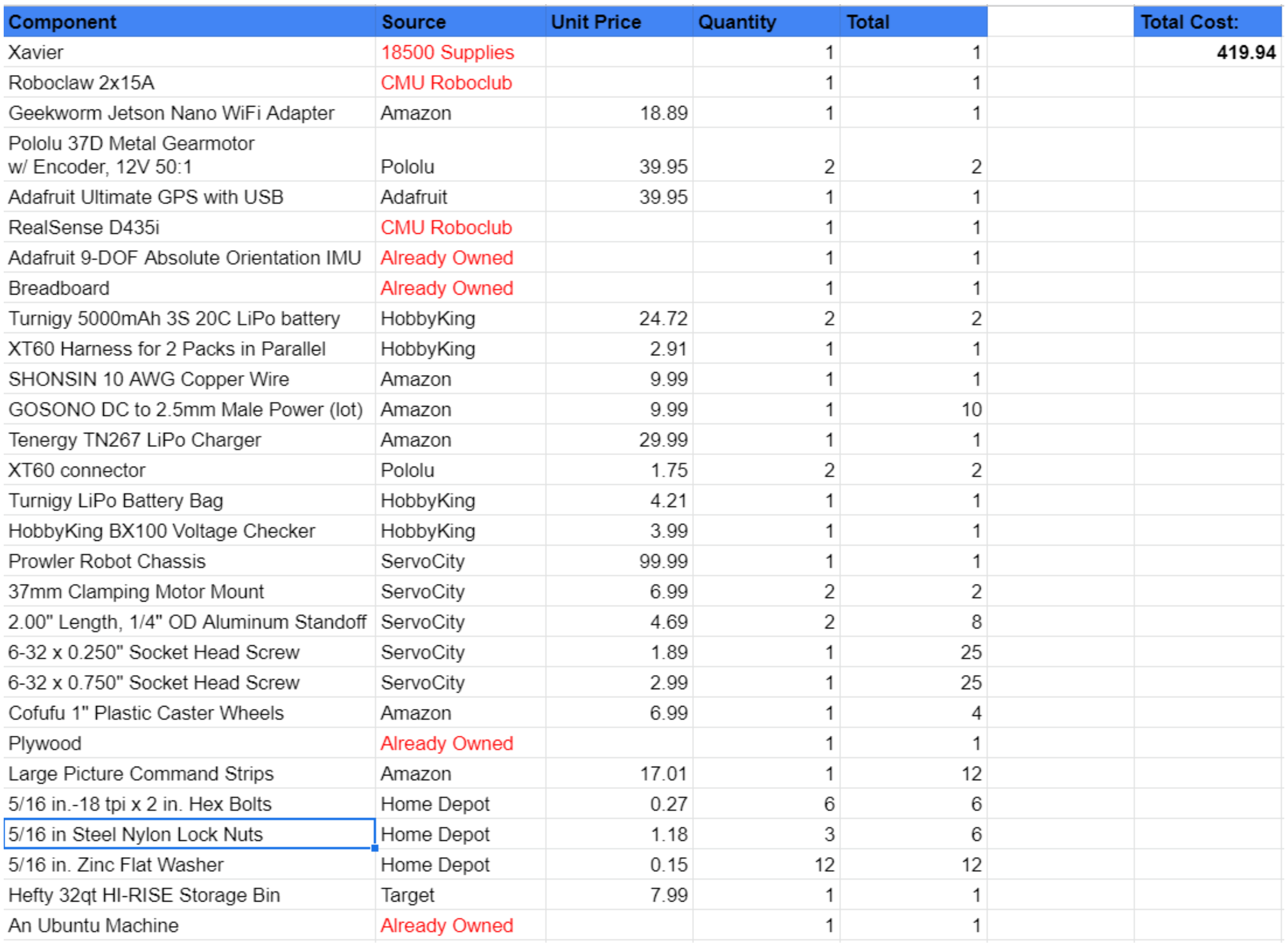
R7: Robot Heartbeat to the Ground Station

This signal is identical to G3, except from the robot's perspective. The robot sends the ground station status information like its raw location as a tf coordinate frame transform through this topic periodically, as documented in signal G3.



GrubTub	start	end	2h	39%
Proposal and Planning	02/18/21	03/17/21	2h	74%
Website Setup	02/18	02/19	2	100%
Proposal Presentation Slides	02/21	02/21	0	100%
Proposal Presentation (feat. Michael)	02/22	02/23	0	100%
Finalize robot CAD Design	02/24	02/24	0	100%
Test Camera/LIDAR	02/21	02/22	0	100%
Finalize UI Design	02/25	02/26	0	90%
Finalize parts list	02/26	02/27	0	100%
Design Presentation (feat. Advaith)	02/28	03/05	0	100%
Design Report	03/12	03/17	0	0%
Design and Implementation	02/20/21	04/13/21	0h	50%
Milestone #1: Proof of Concept/C...	02/20/21	03/16/21	0h	96%
Robot line following in simulation	02/24	02/25	0	100%
Determine nodes on the map/build...	02/20	02/20	0	100%
Brute force multi-order algo simulat...	02/21	02/22	0	100%
Multi-order algorithm proof of conc...	02/24	03/02	0	100%
Integrate IMU to ROS	02/22	02/22	0	100%
Proof of concept for global/local pl...	02/26	02/28	0	100%
Order Structural Components	02/28	03/12	0	100%
Order Electronic Hardware	02/28	03/12	0	100%
Basic power test	03/13	03/14	0	100%
Assemble Chassis	03/13	03/16	0	50%
Milestone #2: Basic Integration	03/13/21	03/26/21	0h	21%
Integrate Wifi to Jetson Nano and c...	03/15	03/16	0	99%
Integrate Camera to Jetson Nano	03/13	03/14	0	99%
Get localization algorithm running ...	03/13	03/13	0	100%
Build static map of campus	03/20	03/21	0	0%
Get live perception feed to Jetson	03/15	03/15	0	0%
Integrate motor power supply and ...	03/17	03/19	0	0%
IMU data preparation	03/19	03/21	0	0%
Integrate GPS to Jetson	03/20	03/22	0	0%
GPS data preparation	03/23	03/24	0	0%
Integrate Motor Controller	03/20	03/21	0	0%
Integration test #0: Sensors and A...	03/25	03/25	0	0%
Write joystick controller	03/23	03/23	0	0%
Integration test #1: Basic joystick ...	03/26	03/26	0	0%
Milestone #3: Final Implementat...	03/03/21	04/13/21	0h	10%
Get robot to localize within map	03/27	03/28	0	0%
Independent local collision avoidan...	03/29	04/02	0	0%
Independent path following on rob...	03/30	04/02	0	0%
Get heartbeat back to groundstati...	03/27	03/27	0	0%
Implement multi-order algorithm	03/03	03/06	0	100%
Get groundstation to send global p...	03/27	03/27	0	0%
Implement robot state machine	03/22	03/27	0	0%
Validate state machine	03/28	03/28	0	0%
Implement groundstation state ma...	03/28	03/30	0	0%
Integration test #2: Follow basic pa...	04/03	04/03	0	0%
Integrate local planners	04/05	04/06	0	0%
Integration test #3: Follow basic pa...	04/07	04/10	0	0%
Interim Demo	04/12	04/12	0	0%
Write groundstation UI	03/16	03/17	0	0%
Integration test #4: Follow optimal...	04/12	04/13	0	0%
Testing and Benchmarking	04/12/21	04/15/21	0h	0%
Ensure battery life meets 30 minute ...	04/12	04/14	0	0%
Test robot driving up/down every slo...	04/12	04/14	0	0%
Test robot driving/path planning on s...	04/12	04/14	0	0%
Human Interaction Testing (5 tria...	04/12/21	04/14/21	0h	0%
Test pedestrian avoidance	04/12	04/14	0	0%
Test delivery times	04/12	04/14	0	0%
Test number of human interventio...	04/12	04/14	0	0%
Basic Field Testing (20 trials)	04/12/21	04/15/21	0h	0%
Test robot final displacement error	04/12	04/15	0	0%
Test food intactness	04/12	04/15	0	0%
Test load capacity	04/12	04/15	0	0%
Test groundstation intervention la...	04/12	04/15	0	0%
Final Report	04/12/21	05/02/21	0h	0%
Final Presentation Slides	04/29	05/01	0	0%
Final Presentation (feat. Sebastian)	05/02	05/02	0	0%
Record videos	04/12	04/15	0	0%
Edit video	04/24	04/29	0	0%
Final Video Submission	05/02	05/02	0	0%
Final Report	05/02	05/02	0	0%

Figure 10: Gantt Chart. We have three major milestones and testing.



Component	Source	Unit Price	Quantity	Total		Total Cost:
Xavier	18500 Supplies		1	1		**419.94**
Roboclaw 2x15A	CMU Roboclub		1	1		
Geekworm Jetson Nano WiFi Adapter	Amazon	18.89	1	1		
Pololu 37D Metal Gearmotor w/ Encoder, 12V 50:1	Pololu	39.95	2	2		
Adafruit Ultimate GPS with USB	Adafruit	39.95	1	1		
RealSense D435i	CMU Roboclub		1	1		
Adafruit 9-DOF Absolute Orientation IMU	Already Owned		1	1		
Breadboard	Already Owned		1	1		
Turnigy 5000mAh 3S 20C LiPo battery	HobbyKing	24.72	2	2		
XT60 Harness for 2 Packs in Parallel	HobbyKing	2.91	1	1		
SHONSIN 10 AWG Copper Wire	Amazon	9.99	1	1		
GOSONO DC to 2.5mm Male Power (lot)	Amazon	9.99	1	10		
Tenergy TN267 LiPo Charger	Amazon	29.99	1	1		
XT60 connector	Pololu	1.75	2	2		
Turnigy LiPo Battery Bag	HobbyKing	4.21	1	1		
HobbyKing BX100 Voltage Checker	HobbyKing	3.99	1	1		
Prowler Robot Chassis	ServoCity	99.99	1	1		
37mm Clamping Motor Mount	ServoCity	6.99	2	2		
2.00" Length, 1/4" OD Aluminum Standoff	ServoCity	4.69	2	8		
6-32 x 0.250" Socket Head Screw	ServoCity	1.89	1	25		
6-32 x 0.750" Socket Head Screw	ServoCity	2.99	1	25		
Cofufu 1" Plastic Caster Wheels	Amazon	6.99	1	4		
Plywood	Already Owned		1	1		
Large Picture Command Strips	Amazon	17.01	1	12		
5/16 in.-18 tpi x 2 in. Hex Bolts	Home Depot	0.27	6	6		
5/16 in Steel Nylon Lock Nuts	Home Depot	1.18	3	6		
5/16 in. Zinc Flat Washer	Home Depot	0.15	12	12		
Hefty 32qt HI-RISE Storage Bin	Target	7.99	1	1		
An Ubuntu Machine	Already Owned		1	1		

Figure 11: The parts used to construct GrubTub. Sources in red indicate where we did not purchase components. Total refers to the total number of units, since some components come in multi-packs and we ordered multiple packs.