# RoboDutchman

Team A: Robo Dutchmen
Manoj Bhat
Oshadha Gunasekara
Calen Robinson
Kenny Sladick

May 8, 2020

# 1 Abstract

As more companies begin to invest in technology to automate simple tasks, it isn't a question of *when* automation will affect a given company, but *how* it will be implemented. Should a device be created to fulfill the tasks/services of a human operator within the current infrastructure? Or should the entire infrastructure be replaced with updated, computer controlled devices? In the naval industry, the latter approach is embraced by research company Leidos in their autonomous unmanned surface vehicle the Sea Hunter. This ship has revolutionized the way we think about sea navigation, and begged the question: Can the systems that power and maintain the Sea Hunter be translated to a large-scale cargo ship?

Leidos is at the forefront of this research, and they currently believe it will be more cost-effective to develop an autonomous robot that will help perform maintenance on the various electro-mechanical devices found on a cargo ship, rather than designing and building new cargo ships with the Sea Hunter's infrastructure. Our team was tasked with developing this robot, which is able to manipulate all valves and breakers on a mock-up testbed. With RoboDutchman, we took a minimalist approach to cut down cost and complexity, investing more time into developing a robust planning algorithm. By leveraging the Robot Operating System, we've integrated various sensors and controllers that allow for accurate localization in any environment, fast object-detection/classification, and optimized task execution. This platform allows for seamless integration of other sensors/software that can improve the future performance of the robot.

# 2 Table of Contents

# 3 Project Description

The goal of this project is to develop an autonomous mobile robot which can assist the crews of large ships in operating their vessels. The primary function of this robot is to interface human facing elements in the ship, such as valves and breakers. The robot needs to be able to autonomously 1) move to their location and 2) change the state of valves and breakers. However, this robot is not concerned in dealing with all the complexity of a ship, such as ladders, wet floors, the motion of the ship in the waves, etc.

# 4 Design Requirements

## 4.1 Explicit Requirements

The following requirements are created from the provided project specifications for ShipBot.

| Requirement ID | Requirement |
|---|---|
| 1 | Robot size must not exceed 1.5’ (width) x 1.5’ (depth) x 2.5’ (height). |
| 2 | Robot must have a dedicated power supply. |
| 3 | Robot must be robustly constructed. |
| 4 | Robot must not damage anything with which it interacts. |
| 5 | Robot must be able to parse a mission file. |
| 6 | Robot must be able to set the desired state of breakers and valves. |
| 7 | Robot must finish initial calibration procedures within a 1-minute period. |
| 8 | Robot must complete mission within target mission completion time. |

## 4.2 Implicit Requirements

The following requirements are implied from the explicit requirements. Their implications are represented by the requirement IDs.

| Requirement ID | Requirement |
|---|---|
| 3.1 | Robot must ensure all components stay attached at all times. |
| 3.2 | Robot must not use prototype kits or toys in the construction. |
| 2.1 | Robot must be able to run for at least 20 minutes. |
| 2.2 | Robot must be able to distribute power to all components. |
| 6.1 | Robot must be able to identify the state of a breaker. |
| 6.2 | Robot must be able to identify the interaction position of a breaker. |
| 6.3 | Robot must be able to identify the state of a valve. |
| 6.4 | Robot must be able to identify the interaction position of a valve. |

### 4.3 Additional Requirements

These requirements are created by our chosen coolness factors: increased mobility independence, ability to accept multiple missions in a single file, and ability to determine an efficient order to enact all missions.

| Requirement ID | Requirement |
|---|---|
| 2.3 | Robot must get power from an on-board battery. |
| 2.4 | Robot must be rechargeable. |
| 5.1 | Robot must be able to parse multiple mission files. |
| 5.2 | Robot must be able to determine efficient order to handle all missions. |

## 5 Functional Architecture

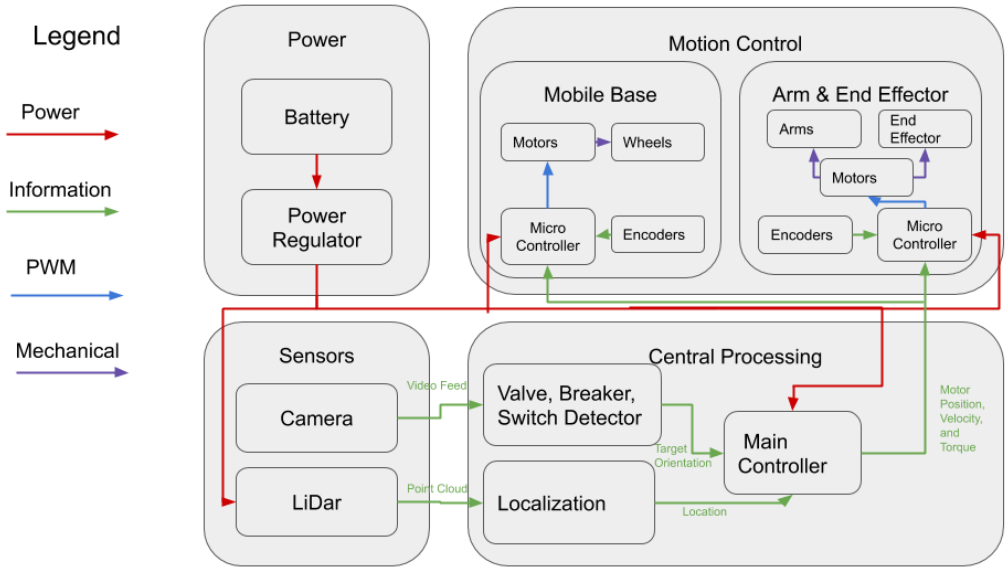



Figure 1: Functional Architecture

Above in figure 1, we see that there are four primary components to our functional architecture: Power, Sensors, Motion Control, and Central Processing. Due to the COVID-19 pandemic, we were unable to properly set up battery operation. As a result, our system is tethered to AC power. The Central Processing component is further broken down in our ROS Software Architecture diagram, shown below in figure 2.

Here, we see six custom ROS nodes: Base Localizer, Base Planner, Target Localizer, Central Planner, Arm Localizer, and Arm Planner. The Base Localizer and Arm Localizer use information from the Hebi actuators, using the hebiros ROS package, to determine the current position and orientation of both the mobile base and the robot arm end-effector. The Base Planner and Arm Planner use feedback-based methods to plan trajectories within the robot's workspace. The Target

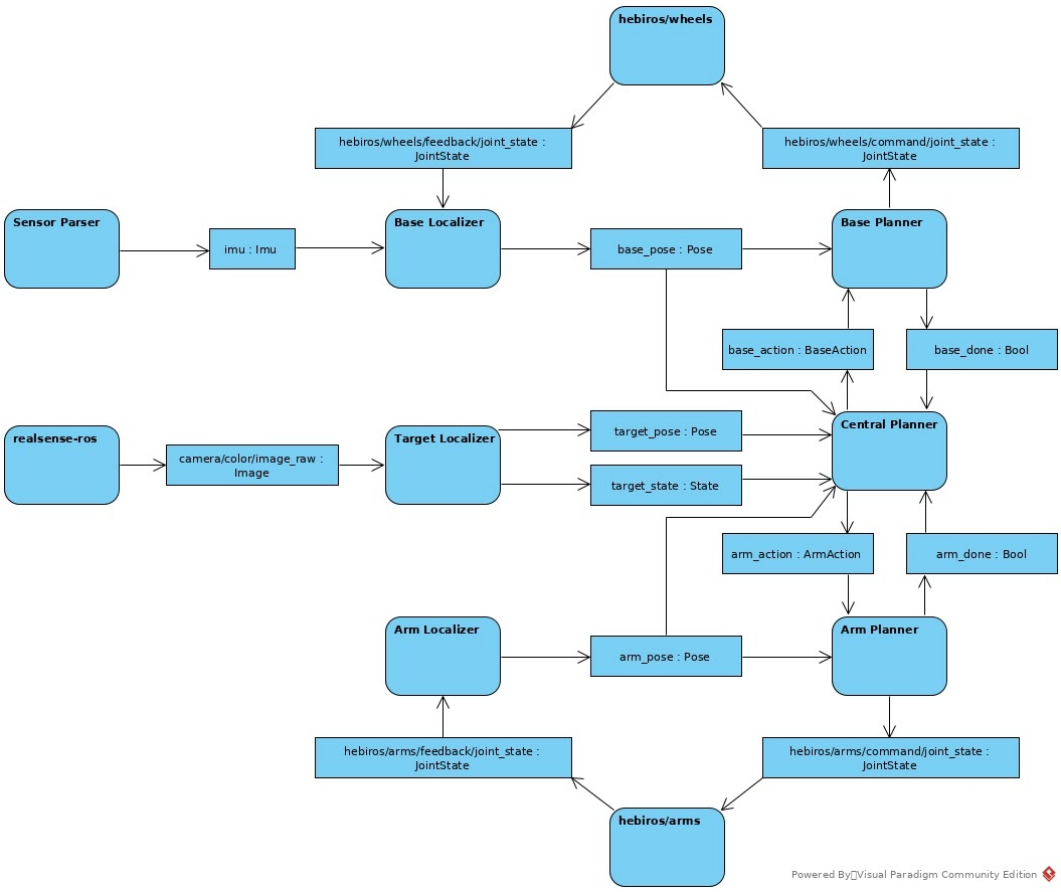



Figure 2: ROS Software Architecture

Localizer uses the Intel Realsense D435 image, using the realsense-ros ROS package, to classify and identify the state of valves and breakers. The Central Planner acts as the central logic behind the system, parsing mission files and sending commands to each of the planning nodes.

# 6 Design Concepts

## 6.1 Chassis

In many ShipBots that we studied from previous years, we saw an omnidirectional drive implemented for base locomotion. As we watched the videos of these robots perform, we noticed that this method of driving was prone to slipping while interacting with a target on the testbed. We also noted that implementing this meant more actuators which increased power consumption and debugging difficulty. For these reasons, we chose to implement a differential drive with two passive casters, shown in our initial sketch in Figure 3. Another feature for which we wanted to utilize our base was as another rotational degree of freedom for our robot arm, rather than having another actuator to power at the base of the arm. Since we chose to use the base for this purpose, we had to consider the shape of the base. As we observed the test bed, we noticed a pronounced corner in

the test bed that could pose a challenge to a differential drive robot. For this reason, we chose to make our base a circle so that we could turn in place and not get stuck in the corner.

Table 1: Base Drive Pugh Matrix

| | | Omni-Drive | | Differential Drive | |
|---|---|---|---|---|---|
| Criteria | Weight | Raw Score | Weighted Score | Raw Score | Weighted Score |
| Ease of Fabrication | 3 | -1 | -3 | 1 | 3 |
| Cost | 1 | -1 | -1 | 1 | 1 |
| Power Consumption | 2 | -1 | -2 | 1 | 2 |
| Speed | 2 | 1 | 2 | -1 | -2 |
| Ease of Control | 2 | 1 | 2 | 0 | 0 |
| Total | | | -2 | | 4 |

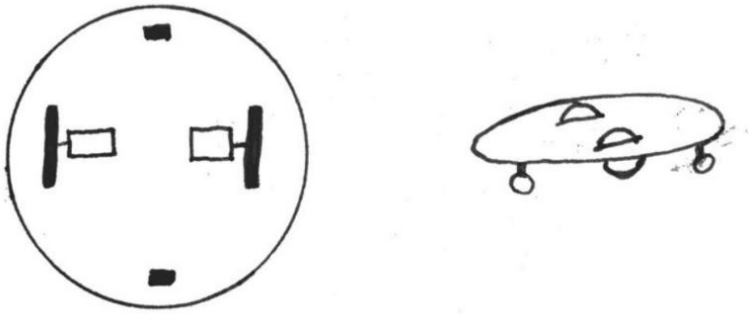


Figure 3: Differential Drive with a rounded base

## 6.2 Robot Arm

We decided to use a 4 degree of freedom robotics arm to interact with the targets. As depicted in figure 4, our arm has one shoulder joint, one elbow joint, and two wrist joints. Because the shoulder is only one joint, the arm cannot rotate about the vertical plane, which means that its work space is planar. In order to position the end of the arm at an arbitrary point in 3D space it is necessary to move the entire robot base. While this does complicate the software, we made this design decision for two reasons: fewer joints increase stability, as well as decrease complexity for planning trajectories, which makes the arm perform faster.

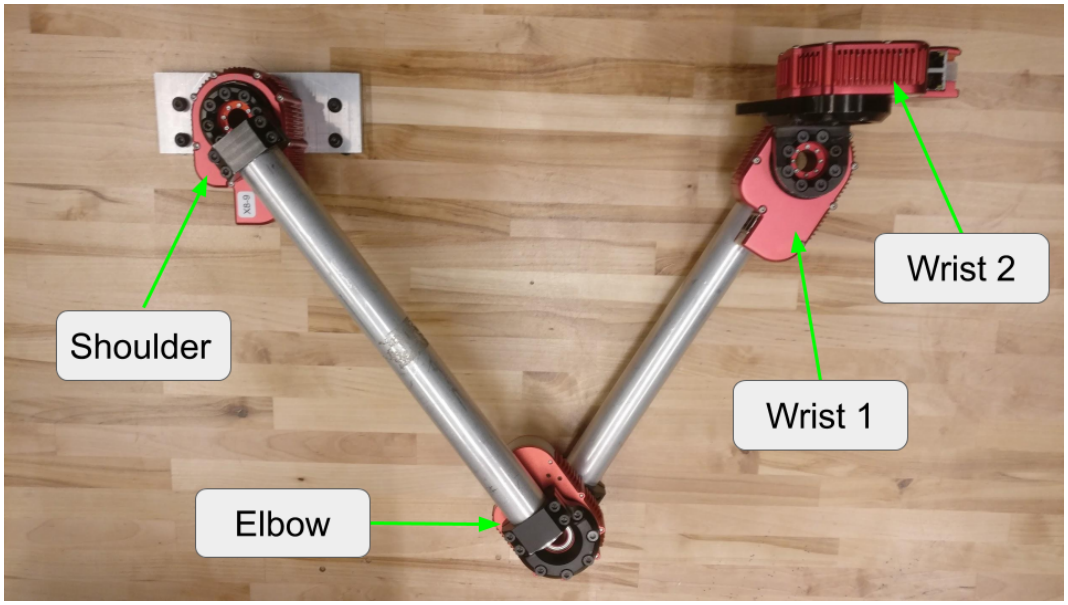



Figure 4: Image of 4 DOF robot arm

### 6.3 End-Effector

As we observed ShipBots from recent years, we found that the end-effector known as the ”Grannular Jammer” was quite popular. It seemed relatively simple to make and it allowed for a custom fit to any surface it was pressed upon. Another end-effector we found interesting was from Team E - Pirates from Spring 2019, utilizing a multi-pin spring loaded design. Upon doing a trade study on both of these (shown in Table 2, we found that the Spring-Loaded multi-pin end-effector would be easier to fabricate, give us enough dexterity to complete the course, and most importantly it was a passive element, which meant fewer components to power, debug, and integrate into the central planner. A very rudimentary sketch of the end-effector we chose is shown in Figure 5.

Table 2: End-Effector Pugh Matrix

| | | Granular Jammer | | Spring-Loaded Multi-Pin | |
|---|---|---|---|---|---|
| Criteria | Weight | Raw Score | Weighted Score | Raw Score | Weighted Score |
| Ease of Fabrication | 3 | 0 | 0 | 1 | 3 |
| Cost | 1 | 1 | 1 | 0 | 0 |
| Dexterity | 2 | 1 | 2 | 0 | 0 |
| Speed | 2 | -1 | -2 | 1 | 2 |
| Ease of Control | 2 | 1 | 2 | 0 | 0 |
| Total | | | 3 | | 5 |

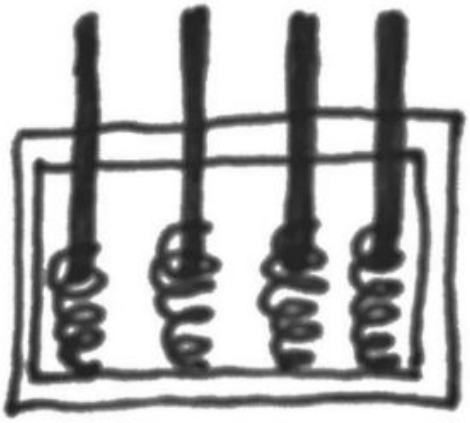


Figure 5: Cross-section sketch of the spring-loaded multi-pin end-effector.

### 6.4 Base Localization

In order to determine the position of the robot on the course, we decided to map our position relative to a generated map. To generate this map, we fused dead reckoned estimates from the Hebi Actuators along with LIDAR information from the RPLIDAR A1 in a SLAM algorithm. For our implementation, we decided to use the gmapping ROS package [2] to handle the mapping portion of localization. To localize within this map, we would fuse the dead reckoned estimates with the LIDAR data using a particle filter to determine our position on the map. For our implementation, we decided to use the AMCL ROS package [1] for the particle filter portion of the base localization.

### 6.5 Base Planning

In planning the base motion from some state A (position and orientation) to some other state B, we decided to take a three-step approach. First, the robot would pre-rotate to the position which would best allow the robot to move straight to the target. Then, the robot would move either straight forward or straight backward toward objective B. Afterwards, the robot would post-rotate until the robot's current orientation matched that of objective B. To incorporate feedback into the base planning loop, a proportional controller dependent on orientation error was used to determine to determine the angular velocity sent to the robot during the rotation components. On the straight component, Pure Pursuit was used to accurately follow a straight line.

### 6.6 Object Detection and Classification

In the vision system, we decided to use D435i sensor which has a single 1280x1024 resolution camera and a IR sensor for measuring the depth along with an IMU sensor for reporting the motion of the camera.We utilize the camera and IR sensor to measure depth of each pixel and get corresponding position. We can also generate a point-cloud for each 2D pixel and depth. But during

final stages, we ended up majorly employing the camera for Target Bounding box detection and classification.

## 7 Cyber-Physical Architecture

In figure 6, we see the Cyber-Physical architecture of RoboDutchman. Here, we see that there are two routers, a local router and a main router. The local router allows the Jetson Nano to communicate with both the Hebi modules as well as the main router. The main router is connected to the internet, which allows us to setup SSH and NX protocols for remote access of the Jetson Nano.

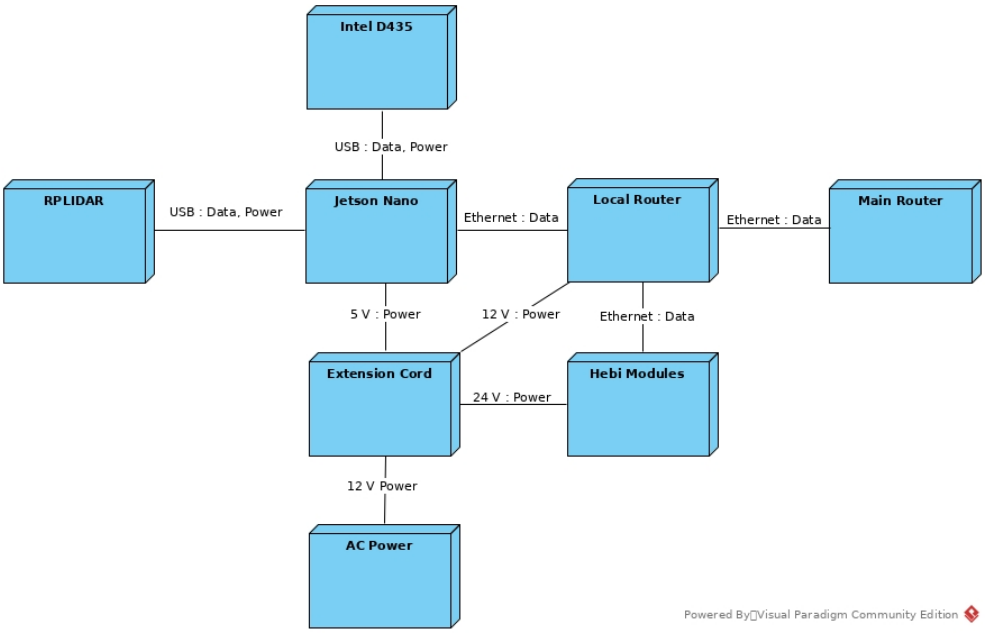



Figure 6: Cyber-Physical Architecture

## 8 System Description and Evaluation

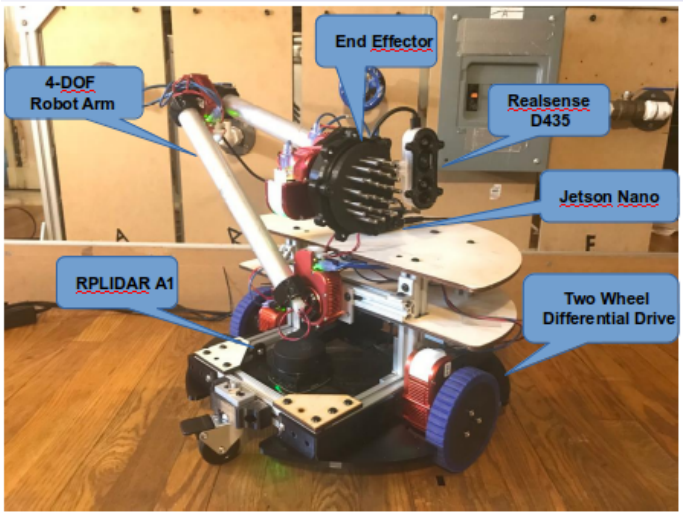



Figure 7: Labelled Picture of RoboDutchman

## 8.1 Chassis

Due to limited access to fabrication resources because of the COVID-19 pandemic, any redesigns to the base had to be printed on a personal 3D printer.

### 8.1.1 Frame

We constructed the chassis out of slotted aluminum profiles and $1/4$" laser-cut plywood, which is shown in Figure 8. Slotted aluminum was chosen for its strength-to-weight ratio, and it was used to bear the weight of the other components of our robot. Another advantage to using the slotted aluminum was its ease of assembly, as well as ease of adjustment in case of design changes. Figure 9 shows the initial location of the shoulder joint of our robot arm, while Figure 10 shows the updated location of our robot arm. This adjustment was made so that we could more easily reach the targets on the testbed, and was very easy to make because of the slotted aluminum profiles used in the frame.

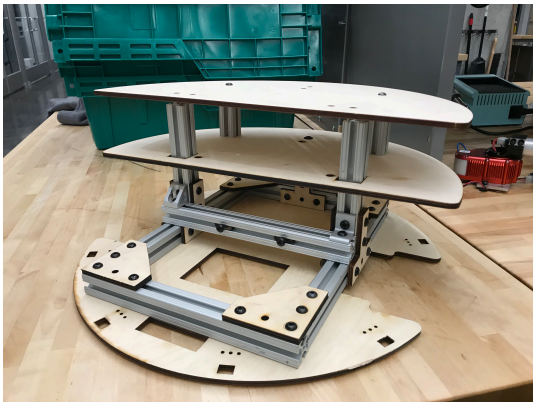


Figure 8: T-Slot aluminum was used for the frame and $^1/_4$" plywood was used to create the platforms for the electronics used by the RoboDutchman

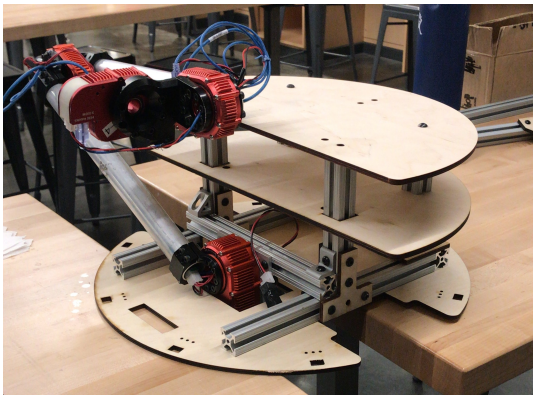


Figure 9: In our initial design, we had the robot arm located much closer to the base of the robot.

### 8.1.2 Locomotion

To drive the wheels of the robot, we chose to use Hebi X8-9 Actuators. The driving factor behind this decision was that the we were also using Hebi actuators for our robot arm, and so it was much easier to plug/power two more Hebi Actuators both from a hardware integration perspective, as well as a software integration perspective. Since we were the only ShipBot team this year, we had access to enough Hebi Actuators to do this. This differential drive is shown in Figure 11

## 8.2 Robot Arm

We decided to use Hebi actuators to actuate our arm. Hebi modules are smart actuators which are easy to integrate into a robot arm, have built in PID tuning mechanisms, can interface with ROS, and have a sufficient torque output. As a result, we decided that these actuators would be the easiest to integrate into our robotics system. One drawback of these actuators is that they have serial elasticity, so they are not stiff or precise. As a result they tend to sag when the arm is carrying a load. However, they serial elasticity allows the robot arm to be flexible when the arm accidentally collides with an obstacle, which mitigates damage to both the environment and the robot arm. The

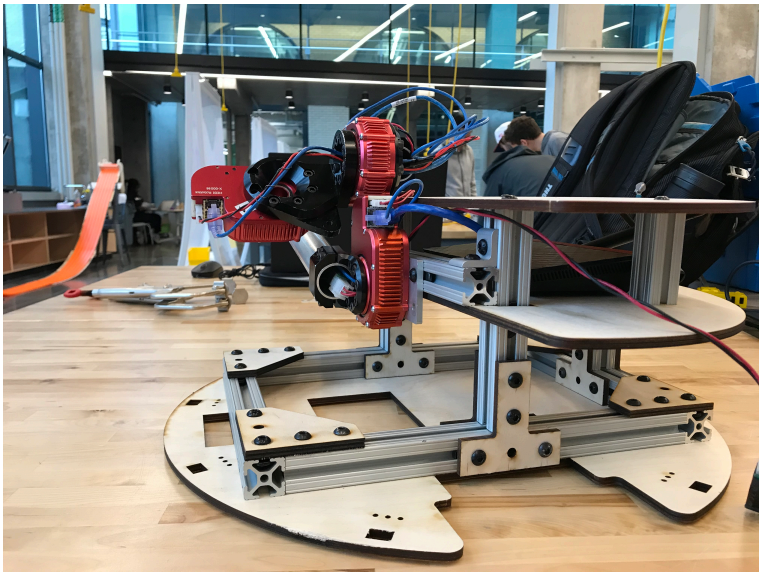


Figure 10: In our final design, we elevated the robot arm further away from the base, allowing us to access the all valves and breakers on the testbed more easily. This was an easy and fast adjustment because of the slotted aluminum profiles used to construct the frame of our robot.

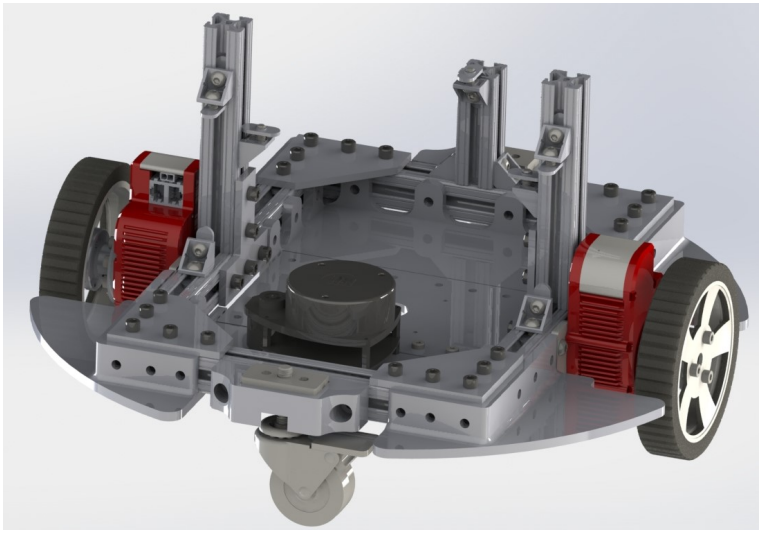


Figure 11: The two driven wheels were actuated by Hebi X8-9 Actuators, with an output bearing capacity that could withstand a robot in excess of 400lbs when coupled together.

serial elasticity is a design element of the the HEBI actuators that allows them to detect the amount of force that they are applying, but time constraints prevented us from using force feedback while controlling the arm. Force feedback was not necessary for the switches and valves. However, during testing we noticed that it was difficult to flip the breakers with only position and velocity control. Further more, some of the HEBI modules would over heat when making repeated attempts to flip the breakers.

## 8.3 End-Effector

Since we chose to use a spring-loaded multi-pin end-effector, we had to ensure that the pins would have an interface with the springs. Initially, we planned to use E-style retaining rings. However, due to COVID-19, we didn't have access to a metal shop to machine the pins with a groove to work with this type of retaining ring. Because of this, we had to choose push-on retaining rings. Unfortunately, these types of retaining rings are much larger in diameter, meaning the pins could not be as close to each other as originally intended in order to avoid hardware interference. A cross section of the final end-effector design is shown below in Figure 12. An open view of the inner spring enclosure is shown in Figure 13.

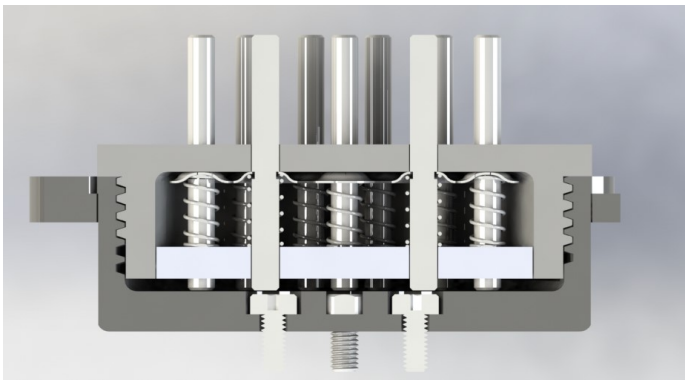


Figure 12: The end-effector contains spring-loaded pins in one enclosure, which is light gray in this image. Then, an outer enclosure (dark gray) was attached to the Hebi wrist 2, which can be seen Figure 4. Once the outer enclosure was on the wrist, the enclosure with the spring-loaded pins was screwed into the outer enclosure, and finally the two were screwed together to avoid unscrewing the end-effector while turning valves.

Figure 13: An image of the inner enclosure that housed all of the springs/pins. A cover went over this enclosure to load the springs.

### 8.4 Base Localization

As stated in our Design Concepts, the base localization utilized two components: gmapping for map creation and AMCL for particle filter based localization within the map. With each of these components came a number of challenges in assuring a desirable localization estimate. For both of these components, we needed to get an initial Dead Reckoning estimate using the Hebi actuators.

#### 8.4.1 Dead Reckoning

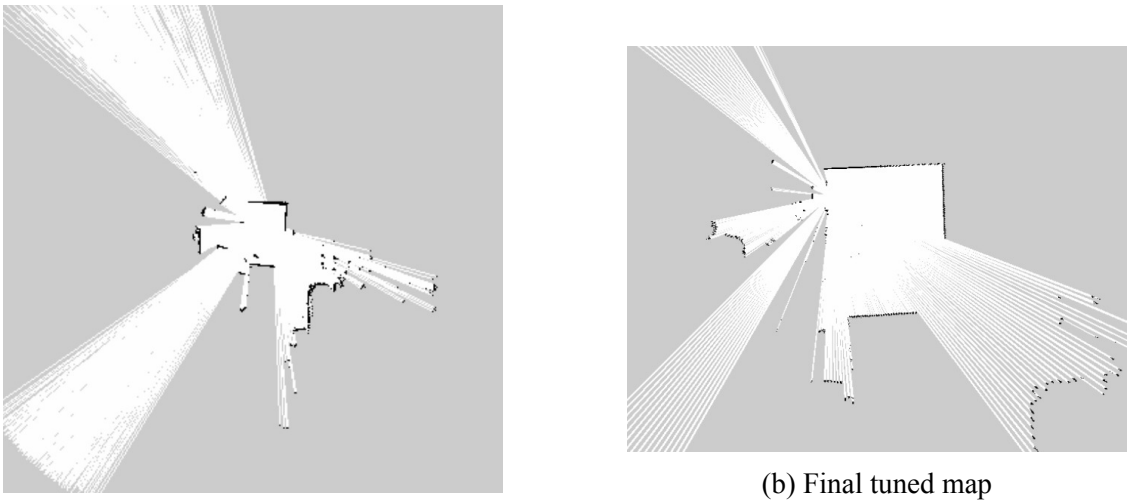
Dead Reckoning is a classic way to get a very rudimentary estimate of our state with only encoder feedback. The basic premise is to determine the linear ($v$) and angular ($w$) velocity and update the state using the following update rule:

$$
\begin{aligned}
x &= x + v\cos(\theta)\Delta t \\
y &= y + v\sin(\theta)\Delta t \\
\theta &= \theta + w\Delta t
\end{aligned}
$$

This method alone, however, does not produce an accurate enough estimate. To get a better estimate, we use a more robust integrator approximation, known as Runge-Kutta [5]. This provides a better state estimate that can be used for both the map creation and the particle filter estimate.

### 8.4.2 Map Creation

Using the Dead Reckoned state estimate, we can utilize gmapping [2] to create a map of the environment. To do this, our process required us to create a bag file where the robot remotely travels around the course, allowing us to collect both LIDAR and Hebi encoder data. By running gmapping and the bag file on a separate computer, we were able to create a map. Initially, our parameters were not ideal, resulting in the first map shown in figure 14a. After tuning the gmapping parameters, we were able to get a map with better resolution, as shown in figure 14b.



(a) Initial map

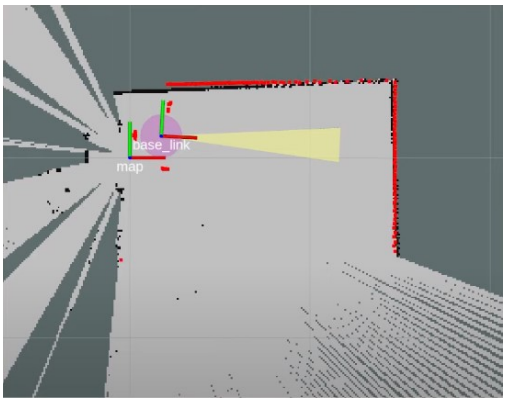
(b) Final tuned map

Figure 14: Maps Generated Using gmapping

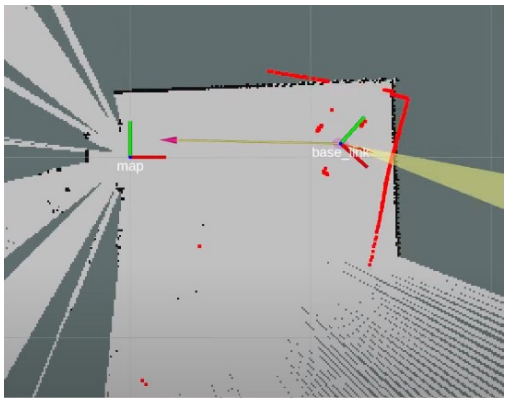
### 8.4.3 Particle Filter Localization

With the Dead Reckoned state estimate and the generated map, we can utilize AMCL [1] to accurately determine the robot's state within the map. This is desired over purely Dead Reckoning due to the likelihood of drift with Dead Reckoning as well as the potential for the wheels to slip on the surface. In order to tune the AMCL parameters, we used the real robot, commanding the robot to move in a fixed manner, initially a straight line. We would visualize the estimate rviz. If we saw that the estimated LIDAR points matched the expected position on the map, we were able to conclude that our parameters were properly tuned. Below, in figures 15a and 15b, we can see an example of a situation where the estimated LIDAR points do align with the map and a situation where the estimated points do not align with the map.

In the tuning process, we identified that the LIDAR information often lags behind the state estimate, as the LIDAR is sending information at a lower update rate than the state estimate. As a result, there is some inherent lag to the particle filter. As a result, in determining whether our parameters result in proper convergence, we wait until the LIDAR values have settled after moving.




(a) Aligned LIDAR points




(b) Unaligned LIDAR points

Figure 15: Localization Estimates Using AMCL

### 8.5 Base Planning

Our Base Planner went through two main iterations. Initially, the planner was developed to be open-loop. This meant that no feedback was involved. However, after developing and testing the planner, it was evident that due to slip in the environment, the error in the robot's movements using this open-loop planner was far too high. As a result, a closed-loop system was developed. To get from point A to point B using this system, as mentioned in the Design Concepts, there were three parts: a pre-rotation to point towards point B, move straight to point B, and a post-rotation to match the orientation of point B. This meant, there were two separate planners: a rotational planner and a linear planner.

#### 8.5.1 Rotational Planner

The rotational planner was a bounded proportional controller. This meant that we would send angular velocity commands based on the following equation:

$$w = \min(k_\theta(\theta_{des} - \theta_{cur}), \theta_{max})$$

By tuning the value of the proportionality constant, $k_\theta$, we were able to get a fast and accurate response with the rotational planner.

#### 8.5.2 Linear Planner

The linear planner used both a bounded proportional controller to determine linear velocities as well as pure pursuit [4] to correct the trajectory by also determining a desired angular velocity. Similar to the rotational planner, our velocity was calculated using the following equation:

$$v = \min(k_d\sqrt{(x_{des} - x_{cur})^2 + (y_{des} - y_{cur})^2}, v_{max})$$

With pure pursuit [4], we could tune our lookahead distance and calculate the desired curvature $\gamma$ to stay on the straight path based on feedback from the base localization. Using this gamma, we could calculate an angular velocity $w$ to be sent along with the linear velocity using the curvature property $w = v \times \gamma$.

### 8.6 Arm Planning

The arm planner has five primary functions: Interfacing with the HEBI ROS library [3], performing forward and inverse kinematics, creating work space trajectories, executing work space trajectories, and interfacing with he central planner. This arm planner is relatively simple and straight forward, but it is robust and gets the job done.

#### 8.6.1 Interfacing with HEBI ROS library

This first responsibility of the arm planner is to connect to the HEBI modules via the HEBI ROS library [3]. This is done using a series of service calls to the HEBI ROS node which initialize the HEBI module group.

#### 8.6.2 Kinematics

Even though our arm has 4 DOF, our kinematics treats it like a 3 DOF arm. The last joint, wrist 2, is controlled separately from the other three joints. This design allows us to use conventional RRR forward and inverse kinematics for our arm.

#### 8.6.3 Trajectory Generation

Trajectory generator can create arbitrary linear work space trajectories. It does this by linearly interpolating between work space way points, then performing inverse kinematics on each of the interpolated way points. This method optimizes for precision and ease of implementation, but does not create a minimum jerk trajectory. In order to create protect the HEBI modules from jerking, we make sure that the robot arm moves at sufficiently slow speeds.

#### 8.6.4 Executing Trajectories

At first, our arm planner would give our interpolated work space way points to the HEBI library, which would then create a minimum jerk trajectory based off of our way points and execute it. However, we found that the HEBI library was not accurate or predictable enough for our use case. Instead, the arm planner manually sends position and velocity commands in the configuration space to the HEBI modules at 200 MHZ. Manually sending the commands produces more accurate and reliable trajectories.

### 8.6.5 Interfacing with the central planner

In order to be a non-blocking function, we decided to wrap the functionality of the arm planner node in a ROS action server. This allows the central planner to make a non-blocking call to the arm planner node, perform other work while the arm planner executes its trajectory, then poll the arm planner to see when it has finished its work. Because the arm planner functionality is non-blocking, it allows our robot to do other tasks in parallel.

## 8.7 Vision system

The vision system interacts with the central planner and sends commands based on Target location and classified recent state.

### 8.7.1 Masking with Colorspace

To understand the shape and structure of the Target, we focus on using prior information of colors on the target objects and segment the object from image. A practical solution to image segmentation is utilizing the color spaces to match and mask. The matching is done by providing a target color and threshold-ing pixel intensity at either HSV or RGB channel space. As from figure 16 it can be observed that HSV space has a spike of blue. We threshold on HSV space for each queried image frame to segment out objects, for both blue and orange objects. We then extract a blob from this segmentation and fit a circle with center and radius, as a key-point. Although this method is time-complex, its precision and accuracy is 99% which is good for our application. In future, well trained deep learning methods can be employed for the task.

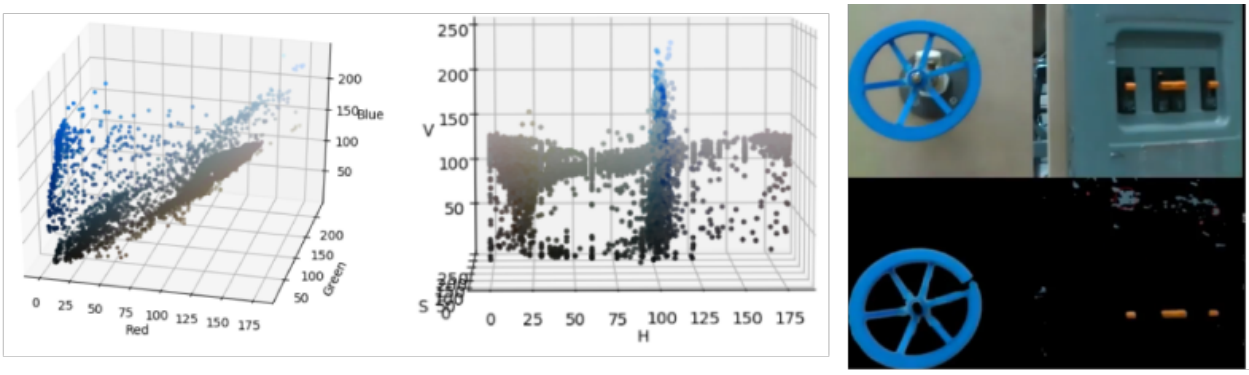



Figure 16: Image segmentation using Color-space thresholding. Left two images show RGB and HSV color spaces. Right image shows Masking performed on data recorded on D435 sensor

### 8.7.2 Object detection and classification

For the task of object detection and classification, we employ a Deep Learning model. We have about 30 frames annotated for training along with state-classes i.e On/Off for standing valves, one for shuttle and wheel valve, and only 3 for 3 breaker switches and we employ masking along with

this to determine state. We chose YoloV3 as our base model. Vanilla YoloV3 pretrained/retrained initially doesnt perform very well. We used extra skip connections and added dialated convolutions into the YoloV3-SPP model which jumped the accuracy from 70% to 98% Precision. The model is trained with Pytorch. We were unsuccessful in using ONNX for exporting custom model to TensorRT, hence we wrote model code in C++ utilizing Nvidia's NvInfer library for Jetson device. Further we also use Quantization (FP16) to improve prediction speed. Hence for any new annotated data we train on pytorch, generate txt file of weights and compile a serialized engine which runs inference in real-time. The detection and classified model is named "Modified and Quantized YoloV3-SPP" (MYDT).

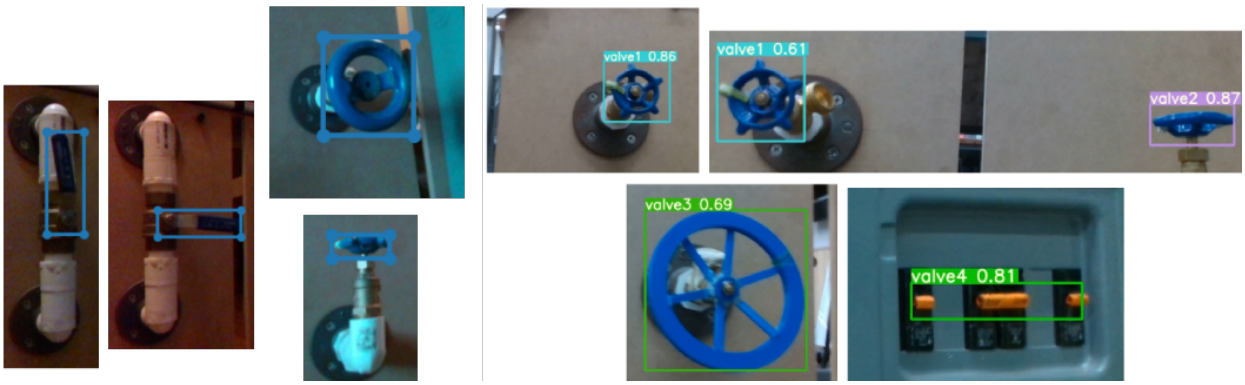



Figure 17: Training and inferred samples. The left four images show training samples along with state-classes for standing valve. On the right is the detection from the model with prediction score

### 8.7.3 Object tracking strategy

We used a distance based tracker to track the bounding boxes (BBOX) for every new detection and keep on assigning the values according to the Functions of the tracker:

1) State generation: The state is generated based on both masking data and the detection of BBOX. The advantage of using a masking procedure is that you get exact location of the breaker switch due to its peculiar color.

2) Location generation: The MYDT does the major part of the job in detection of exact location.

For example, for a new image frame input to the tracker, the *get_mask* function will generate mask which will extract a blog of orange color. This blob key-point will have a center and radius. when this center is on top of the BBOX i.e the y-axis value of the keypoint center is greater than that of the BBOX center, it means that the breaker switch is on. When lower, it means that its off. Hence avoiding another extra class for MYDT training for classification.

### 8.7.4 Interfacing with central planner

The tracker directly talks to the central planner. As central planner sends a signal by publishing a topic, the tracker receives and forwards another signal to the MYDT. The MYDT is a TensorRT

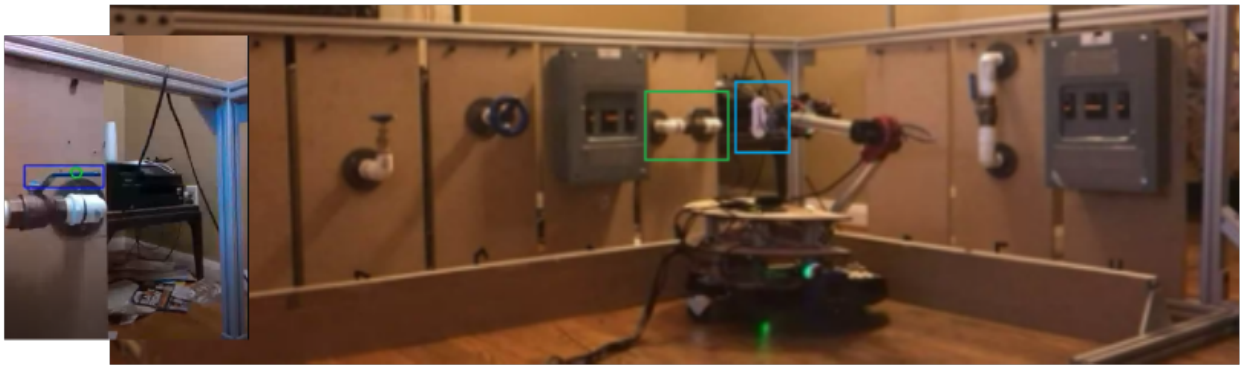


Figure 18: Real-time Query-based detection. The central planner queries the Tracker to send it detection. Here the left image is what camera sees and Blue is the detected box whereas green circle is the mask keypoint. On right image, the same instant robot state illustrated, when the robot sees the valve in green box through the camera in blue box.

inference engine, it takes about average 0.53s for predictions on each new image. While the MYDT works, the tracker starts another parallel process of Masking. As the function of masking process is to generate mask based on colors, it identifies and stores a center keypoint and radius of the blob on a global variable. As soon as detector is ready with the prediction, it publishes a message consisting of anchor point and size (x, y, w, h) of each BBOX back to the tracker. Using the stored mask and stored detection, for every signal from central, the tracker publishes back the state and location of the target. This is made such that the central planner can send messages whenever it needs, and don't need to run the detector all the time. Hence reducing the gpu and memory load, needed for other important processes like mapping and trajectory generator of HEBIs. But note that this framework can run for any fps considering the tracker system uses BBOX interpolated tracking.

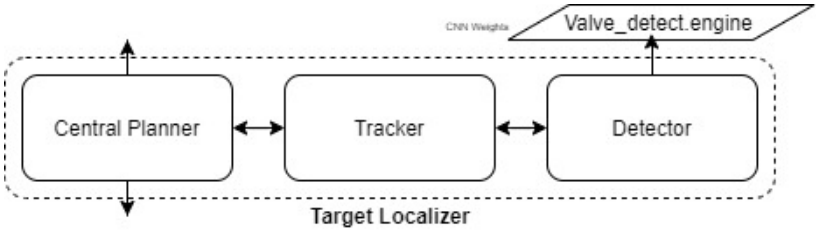



Figure 19: Target localizer connectivity

## 8.8 Integration and Evaluation

The integration of our components was split into the integration of our mechanical and hardware subsystems and the integration of our software subsystems. Although we did finish a majority of the mechanical integration prior to Spring Break, some of the mechanical integration as well as all of the software integration occurred after Spring Break. At this time, we were working remotely,

and as such our procedure for integration had to change. Kenny was the only person with physical access to the robot.

To perform our integration, we setup the SSH and NX protocols to allow us to edit code and visualize the Jetson Nano's desktop remotely. After working on our code on our local machines, we would schedule times to meet with Kenny to test the code on the physical robot. This definitely proved to be doable, albeit more time consuming than normal.

# 9 Project Management

## 9.1 Schedule

The schedule, as shown in figure 22, shows the initially planned deadlines for our robot as well as the primary class deadlines we must meet. We have planned to complete our prototype robot early to give us time to account for unexpected occurrences and further polish the robot. This also allows us to account for our team members' lowering availability as the semester progresses. However, due to the COVID-19 pandemic, our schedule has shifted and changed to account for our transition to remote development.

## 9.2 Team Responsibilities

Our responsibilities are split up as the following:

1. Manoj Mohan Bhat - Computer Vision and software development and Electronics
2. Kenny Sladick - Mechanical Design and Software-Electronics interface Management
3. Oshadha Gunasekara - Motion Planning, interfacing, Telemetry and Electronics
4. Calen Robinson - Telemetry Management, Software Package Management, Robot Arm Development

## 9.3 Budget

Currently our preliminary budget assumes the ability to use motors and motor controllers. This results in our primary reimbursable budget shown in figure 20.

## 9.4 Risk Management

Our risks and mitigation strategies are represented in the table in figure 21.

# 10 Conclusions

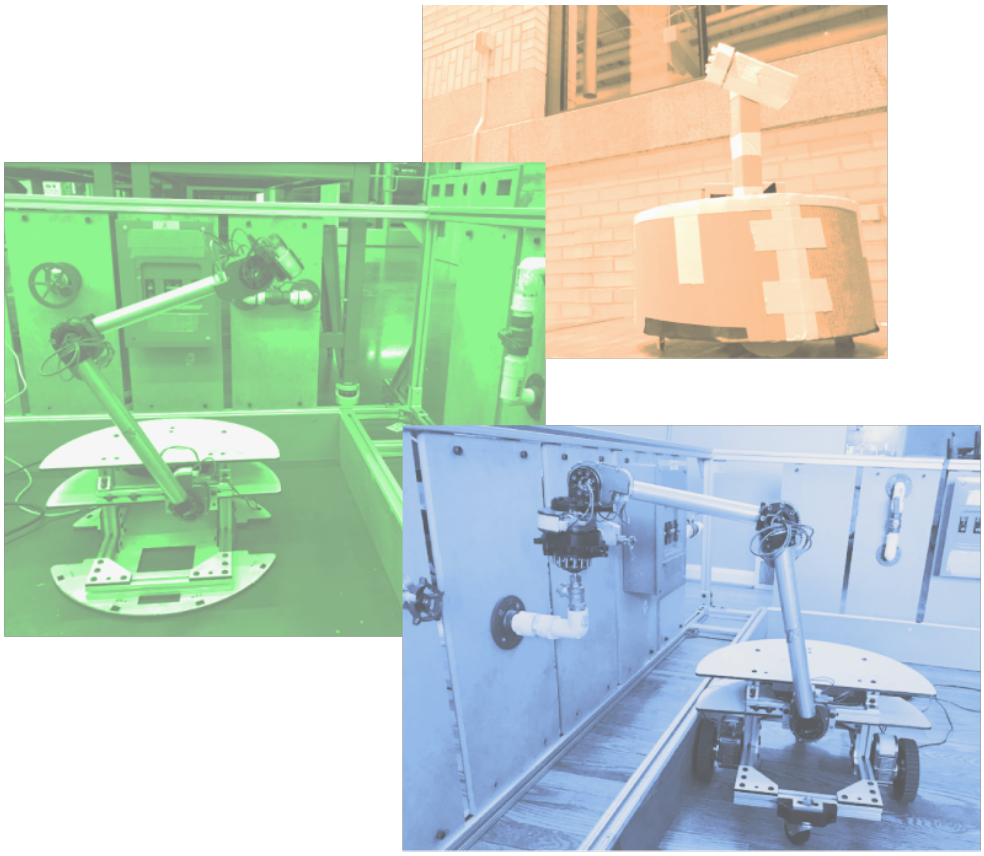
## 10.1 Lessons Learned

Throughout this process, we've learned a lot. First, we learned that tuning our subsystems is a very timely process. For example, initially tuning the base localizer was set to only take a few

days. However, in reality, the tuning of this system easily took an entire week. We also greatly underestimated the time it would take to tune the PID gains on the arm, and so we learned that if this needs to be done virtually, that either there needs to be a lot of time set aside to do this, or ideally a more systematic approach could be taken to speed up the process. Lastly, we all learned more about patience, not only with robotic system development, but with each other as we did our best to communicate our ideas and problems with each other virtually, which would've been much more efficient in person if everyone could be with and see the robot.

## 10.2 Future Work

For future work, this robotic system would benefit from having a smaller end-effector. This end effector would allow it to manipulate the breakers with better ease, minimizing the likelihood in getting stuck on the uneven surface of the breaker. Also, better tuning of the PID gains of the Hebi actuators would benefit our ability to more accurately control the robot arm to manipulate the valves and breakers. The arm would benefit from having a minimum jerk trajectory generator, as well as a force-control trajectory generator, to enable switching breakers in the more resistant direction. Something else that would benefit the robot would be to power the system remotely with an on-board battery, making it easily deployable. Further, having a backup battery in case of emergencies would be ideal, as the end-use of this robot would be on a ship with very few or no people. Along with these, vision management is also a hard topic which could be imporved. The basic requirement is fruitful feedback from the vision system into the planner. This can be done through Image detection, but is complex in time and memory. We used the vision system but not to its fullest potential. A better management/engineering of this system would make the robot solve dynamic target locations problem and make it robust in reaching desired state and performing desired action.



## 11 References


[1] *amcl - ROS WIKI*. URL: http://wiki.ros.org/amcl. (accessed: 08.05.2020).

[2] *gmapping - ROS WIKI*. URL: http://wiki.ros.org/gmapping. (accessed: 08.05.2020).

[3] *hebiros - ROS WIKI*. URL: http://wiki.ros.org/hebiros. (accessed: 08.05.2020).

[4] *Implementation of the Pure Pursuit Tracking Algorithm*. URL: https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf. (accessed: 08.05.2020).

[5] *Lab 3: Dead Reckoning*. URL: https://www.cs.cmu.edu/~16311/s07/labs/NXTLabs/Lab%203.html. (accessed: 08.05.2020).

[6] *RoboDutchman*. URL: https://sites.google.com/view/flyingdutchman. (accessed: 08.05.2020).

[7] *RoboDutchmen/RoboDutchman*. URL: https://github.com/Robo-Dutchmen/Robo-Dutchman. (accessed: 08.05.2020).

# 12 Appendices

## 12.1 Budget

| Date (mm/dd/yyyy) | Vendor | Description | Qty | Unit Price | Total Price | Cumulative Price |
|---|---|---|---|---|---|---|
| 2/11/2020 | Other | Jetson Nano | 1 | $89.00 | $89.00 | $89.00 |
| 2/13/2020 | Amazon | Samsung 64GB 100MB/s (U3) MicroSDXC EVO | 1 | $11.99 | $11.99 | $100.99 |
| 2/18/2020 | Other | Intel RealSense D435 | 1 | $179.00 | $179.00 | $279.99 |
| 2/18/2020 | Other | 5 Inches Robot Wheel Without Key Hub | 2 | $12.50 | $25.00 | $304.99 |
| 2/18/2020 | Other | Metal Key Hub - 15mm | 2 | $2.48 | $4.96 | $309.95 |
| 2/19/2020 | McMaster-Carr | Flange-Mount Ball Transfer | 2 | $3.97 | $7.94 | $317.89 |
| 2/19/2020 | Amazon | Gowoops 5PCS 150W DC-DC 10-32V to 12-3 | 1 | $19.99 | $19.99 | $337.88 |
| 2/19/2020 | Other | Emergency Stop Switch | 2 | $5.90 | $11.80 | $349.68 |
| 2/19/2020 | Amazon | MCIGICM 10 Pair XT60 Connectors, XT-60 M | 1 | $5.99 | $5.99 | $355.67 |
| 2/19/2020 | Amazon | EYESKY Fireproof Explosionproof Lipo Batter | 1 | $8.99 | $8.99 | $364.66 |
| 2/19/2020 | Amazon | Electronics-Salon Screw Terminal Block Break | 1 | $32.00 | $32.00 | $396.66 |
| 2/25/2020 | Amazon | Svance 2pcs 22mm Mounting Hole Latching B | 1 | $10.99 | $10.99 | $407.65 |
| 2/26/2020 | Amazon | PZRT 12-Pack Aluminum Profile Corner Brack | 1 | $11.69 | $11.69 | $419.34 |
| 2/26/2020 | McMaster-Carr | 18-8 Stainless Steel Button Head Hex Drive Screw | 2 | $5.65 | $11.30 | $430.64 |
| 2/26/2020 | Other | 1/4-20 Slide-in Economy T-Nut - Centered Thread | 100 | $0.21 | $21.00 | $451.64 |
| 3/4/2020 | Other | RPLidar | 1 | $99.00 | $99.00 | $550.64 |
| 3/4/2020 | McMaster-Carr | Side-Mount External Retaining Rings for 3/16" OD, | 1 | $4.15 | $4.15 | $554.79 |
| 3/4/2020 | McMaster-Carr | 18-8 Stainless Steel Dowel Pin, 3/16" Diamete | 2 | $10.82 | $21.64 | $576.43 |
| 03/04/2020 | Amazon | Adafruit 9-DOF Absolute Orientation IMU Fusi | 1 | $32.00 | $32.00 | $608.43 |
| 03/04/2020 | McMaster-Carr | Compression Spring, 0.5" Long, 0.25" OD, 0.1 | 1 | $10.84 | $10.84 | $619.27 |
| 3/5/2020 | McMaster-Carr | 70 lb. Capacity Plate Mount Swivel Caster for | 2 | $15.98 | $31.96 | $651.23 |
| 03/16/2020 | McMaster-Carr | Compression Spring, 0.5" Long, 0.25" OD, 0.1 | 2 | $10.84 | $21.68 | $672.91 |
| 03/16/2020 | McMaster-Carr | Low-Strength Steel Thin Square Nut, M3 x 0.5 | 1 | $8.00 | $8.00 | $680.91 |
| 03/17/2020 | McMaster-Carr | Alloy Steel Low-Profile Socket Head Screw, He | 1 | $9.84 | $9.84 | $690.75 |
| 03/17/2020 | McMaster-Carr | Alloy Steel Low-Profile Socket Head Screw, H | 1 | $10.35 | $10.35 | $701.10 |
| 03/17/2020 | McMaster-Carr | Push-on External Retaining Rings, for 3/16" O | 1 | $10.46 | $10.46 | $711.56 |
| 3/17/2020 | McMaster-Carr | Push-on External Retaining Rings, for 3/16" O | 1 | $5.33 | $5.33 | $716.89 |
| 3/17/2020 | McMaster-Carr | Tweezers with 0.002" Thick Slim Pointed Tips, | 1 | $6.38 | $6.38 | $723.27 |
| 3/17/2020 | Amazon | 3D Solutech Real Black 3D Printer PLA Filam | 1 | $18.99 | $18.99 | $742.26 |
| 3/18/2020 | McMaster-Carr | Alloy Steel Low-Profile Socket Head Screw, H | 1 | $10.62 | $10.62 | $752.88 |
| 3/18/2020 | McMaster-Carr | Low-Strength Steel Square Nut, Class 5, Zinc- | 1 | $8.00 | $8.00 | $760.88 |
| 3/18/2020 | McMaster-Carr | Alloy Steel Low-Profile Socket Head Screw, H | 1 | $11.90 | $11.90 | $772.78 |
| 3/19/2020 | Amazon | Stanley 84-096 5-Inch Needle Nose Plier | 1 | $7.99 | $7.99 | $780.77 |
| 2020-03-20 | Amazon | 3D Filament Clear PETG Filament 1.75mm wi | 1 | $21.99 | $21.99 | $802.76 |
| 2020-03-20 | Amazon | 7pcs 230x280mm Sandpaper 400-1200 Grit V | 1 | $0.00 | $0.00 | $802.76 |
| 2020-03-25 | McMaster-Carr | Compression Spring, 0.5" Long, 0.24" OD, 0.1 | 3 | $10.84 | $32.52 | $835.28 |
| 2020-03-29 | McMaster-Carr | 18-8 Stainless Steel Square Nut, 3/8"-16 Thre | 1 | $8.60 | $8.60 | $843.88 |
| 2020-03-29 | McMaster-Carr | Compression Spring, 0.625" Long, 0.240" OD | 3 | $5.50 | $16.50 | $860.38 |
| 2020-03-30 | McMaster-Carr | Black-Oxide Alloy Steel Socket Head Screw, M | 1 | $8.33 | $8.33 | $868.71 |
| 2020-04-02 | McMaster-Carr | Zinc-Plated Steel Hex Nut, Medium-Strength, | 1 | $2.22 | $2.22 | $870.93 |
| 2020-04-02 | Amazon | OVERTURE PETG Filament 1.75mm with 3D | 1 | $65.99 | $65.99 | $936.92 |
| 2020-04-03 | Amazon | ScotchBlue Original Multi-Surface Painter's Ta | 2 | $5.64 | $11.28 | $948.20 |
| 2020-04-06 | Amazon | SMAKN DC 5V/4A 20W Switching Power Sup | 1 | $9.99 | $9.99 | $958.19 |
| 2020-04-06 | Amazon | Jadaol Cat 7 Ethernet Cable 50 ft Shielded, S | 2 | $21.99 | $43.98 | $1,002.17 |
| 2020-04-06 | Amazon | GE 6 Outlet Power Strip, 12 Ft Long Extensio | 1 | $11.99 | $11.99 | $1,014.16 |
| 2020-04-08 | Amazon | Wire Cutter - 5" Flush Cutter Micro Shear Wire | 1 | $6.29 | $6.29 | $1,020.45 |
| 2020-04-09 | Amazon | SIQUK 22 Pieces 3D Printer Nozzles MK8 No | 2 | $7.98 | $15.96 | $1,036.41 |

Figure 20: Budget Table

## 12.2 Risk Management

## 12.3 Schedule

## 12.4 Code and Documentation

Our code for this project is on Github [7]. Media and further documentation can be found on our website [6].

| Failure | Description | Mitigation |
|---|---|---|
| Localization failure | The robot is unable to determine where it is. | Change position of sensor or ensure localization algorithm is functional. |
| Unable to grip obstacle | The robot is unable to properly grip onto the valves/breakers. | Try out different end effector designs. |
| Manipulation causes movement | The robot loses traction with the floor while manipulating valves/breakers. | Use a motor with more torque at the end effector or design component to ensure robot stability. |
| Classification failure | Robot is unable to classify objects as valves or breakers. | Re-train object classification network using more data. |
| Computer Latency | Computation time of CV algorithms not able to meet the performance requirements of the controller. | Upgrade the primary computer or utilize a less sophisticated computer vision algorithm. |
| Battery failure | Battery is not capable of powering all components. | Maintain a spare battery. |

Figure 21: Risk Management Table

| Day | Date | Deadline |
|---|---|---|
| W | 1/29 | Design Concept Proposal |
| M | 2/3 | Mock-Up Demo |
| W | 2/5 | Sensors Demo |
| W | 2/12 | Motor Control Demo |
| W | 2/19 | Initial work with robot arm finished |
| W | 2/19 | Initial version of particle filter working |
| W | 2/19 | System Demo #1 |
| F | 2/21 | Website Check #1 |
| W | 2/26 | Initial fabrication of mobile base |
| W | 2/26 | Full robot arm assembled |
| W | 2/26 | Software architecture finalized |
| W | 2/26 | System Demo #2 |
| M-W | 3/2-3/4 | Mid-semester Presentation |
| W | 3/18 | Robot arm teleoperation and FK |
| W | 3/18 | Functional localization on course |
| W | 3/18 | Initial fabrication of end-effector |
| W | 3/18 | Valve/breaker detection functional |
| W | 3/18 | System Demo #3 |
| W | 3/25 | End-effector fabrication complete |
| W | 3/25 | Target localizer functional |
| W | 3/25 | Base planner complete |
| W | 3/25 | Arm planner/localizer complete |
| W | 3/25 | System Demo #4 |
| W | 4/1 | Able to change breaker state |
| W | 4/1 | System Demo #5 |
| F | 4/3 | Website Check #2 |
| W | 4/8 | Able to change valve state |
| W | 4/8 | System Demo #6 |
| W | 4/15 | Working arm planner/localizer |
| W | 4/15 | Working target localizer |
| W | 4/15 | System Demo #7 |
| W | 4/22 | System Demo #8 |
| W | 4/29 | Final System Demo |
| W | 5/6 | Final System Demo Encore |
| F | 5/8 | Final Report |
| M | 5/11 | Website Check #3 |
| T | 5/12 | Lab Cleanup |

Figure 22: Schedule