Worcester Polytechnic Institute

A Major Qualifying Project

WALRUS Rover Expansion

Submitted by:

Ozan Akyıldız, Robotics Engineering

Andrew Davis, Robotics Engineering

Carson Wolf, Robotics Engineering

Advised by:

Kenneth Stafford, Associate Teaching Professor

Robotics Engineering, Mechanical Engineering

Abstract

The WALRUS rover is a capable search and discovery platform aid in disaster relief. It utilizes actuated pods, onboard cameras, and aquatic mobility to provide responders with the information they need. The goal of this project is to enhance the WALRUS rover, by improving the situational awareness of the users. We utilized 3D mapping to present the environment in a natural way. We fabricated a new water resistant mast, to provide a superior view point. Finally, we implemented obstacle avoidance to allow the user to focus on the task at hand, instead of the obstacles. This document outlines the requirements and design to implement these features.

Acknowledgements

The Team would like to thank the previous WALRUS team: Brian Eccles, Brendan McLeod, Timothy Murcko, Thomas Jeffery Watson, and Mitchell Wills. Without their hard work, and continued dedication to the project, none of this would have been possible. We would also like to thank our advisor professor Kenneth Stafford for his guidance and patience. Additionally, we would like to thank Joe St. Germain for his help. We also want to thank the machining company UMSAN located in Kocaeli, Turkey for producing the inclosure for the renewd mast.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The dangers of sending humans into disaster areas have created a growing demand for disaster response robots. Though robots have been used for disaster response for 14 years, many modern disaster response robots still have the same limitations as the first robots. The WALRUS rover was designed and created in 2015 to address these problems. It represents a new generation of disaster response robots, with advanced sensors, and a powerful on-board computer. This project utilized the WALRUS rover to create a system that can provide disaster response specialists and personnel the data they need.

2. Background Research

2.1. Disaster Response

Disaster response robots face challenges and environments that are often unknown and unique to each disaster. The most versatile robots are built to be able to handle as many scenarios as possible. The purpose of robots that serve in disaster situations is often not to replace the first responders but to provide critical data to everyone involved in the disaster relief effort.

The terrorist attack on the World trade center on September 11, 2001 was one of the first disasters in which robots were utilized. The robots were used to enter places too small for a person, as well as deep unstructured voids. This gave the first responders a new perspective, allowing them to do structural inspection and find paths through the rumble. One of the largest shortcoming of the robots used was that most of the robots did not have sensors other than cameras and had a difficult time determining their position in space. [9]

In 2011, Japan was hit with a massive tsunami caused by an earthquake off the coast. Robots were again called in. Underwater robots were used to evaluate critical infrastructure. The use of these robots allowed a fishing port to reopen in six days, opposed to the 6 months it would have taken if a diver team had to have been brought in. The use of robots in this disaster identified the need for research in vision and mapping. One of the most significant problems faced in Japan was lack of situational awareness. The tether of one of the robots became tangled in a mooring buoy and had to be manually recovered. [11]

Search and rescue is one of many tasks that robots can be utilized for during disaster relief. We believe that WALRUS can be more than a “search and rescue” robot focused solely on the search for survivors. By redefining WALRUS to be a “disaster relief” robot we are not changing its purpose but expanding it.

2.2. Disaster Response Robots

There are already various robots that are being used for disaster response. This section will cover three robots that are designed to respond to disasters. They have been selected to be representative of the current field of ground based disaster response robots. Each robot has been evaluated, and the strengths and weaknesses will be explained, and compared to the WALRUS rover.

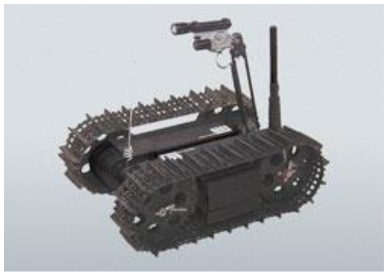


Figure 1: Foster-Miller Solem

The simplest robot is the SOLEM made by Foster-Miller. Like many disaster response robots, it was designed for mine and IED clearance. It was first used for disaster response in the 9/11 world trade center hot zone. SOLEM is a small tracked vehicle that weighs 15 kg. It is controlled over a tether, and is able to send black and white video to the operator. SOLEM is small enough to enter small openings, however, it lacks the sensors and cameras to provide critical data. There are many similar robots with similar advantages and limitations. In general, these types of robots are small and robust, and can enter smaller openings that robots such as WALRUS are too large to. However, the video they provide in often very hard to interpret and they have very basic localization. [1]

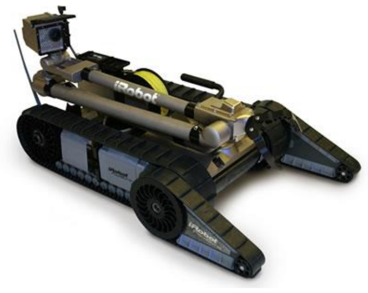


Figure 2: iRobot PackBot

One of the most abundant robots used for disaster response is the iRobot PackBot. Like the SOLEM it was originally developed for the military but is now used for anything from disaster response to bomb disposal. PackBot is a step up from SOLEM and other basic robots in a variety of ways. First and foremost, it has multiple cameras, which give the user better situational awareness. It also has treaded "flippers" that enhance maneuverability. Lastly PackBot can be equipped with the iRobot UAP (User Assist Package), which adds software based updates to minimize risk and increase situational awareness. These features include: retro-traversal, self-righting, heading hold, as well as GPS mapping and still Image capturing. [3]

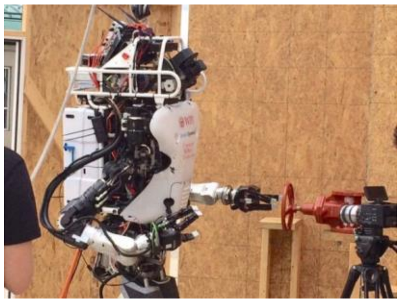


Figure 3: Boston Dynamics Atlas Robot

The final category of disaster response robots can be represented by ATLAS. These robots are extremely advanced, many are bipeds and all have 3D sensing capabilities, object recognition and shared autonomy functionality. They will be able to better interact with the environment, and accomplish tasks that up until now, could only be done by humans. They represent the future of disaster response robotics, however, they are still under development. It will be many years before any can be deployed. Even upon their deployment, there will still be a need for less advanced robots such as PackBot and Solem.

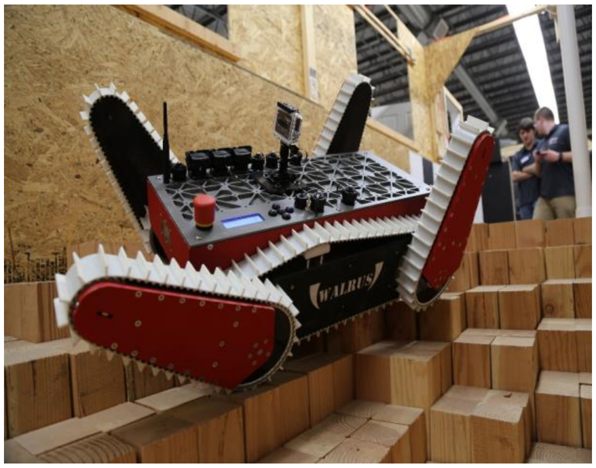


Figure 4: The WALRUS rover

The WALRUS rover was created in 2015 by a team of students at WPI. WALRUS is comparable to an advanced PackBot. They have similar designs, but WALRUS boasts more onboard processing power, as well as amphibious capabilities. WALRUS also runs ROS (Robot Operating System), and has a 3D camera, which will allow us to enhance the user's situation awareness by implementing SLAM (Simultaneous Localization and Mapping).

2.3. User interface

2.3.1. Shared autonomy

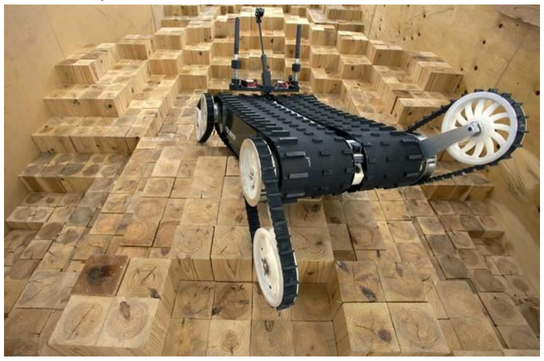


Figure 5:Quince

WALRUS, like many other disaster response robots, has multiple “flippers” or subtracks. These help with agility on uneven terrain and can be used to help climb stairs. The downside of these flippers, is that they add additional DOFs (degrees of freedom), that the operator must control. Some more advanced robots such as Quince have autonomously actuated flippers, that reduce the workload of the driver. This kind of subsystem autonomy is often referred to as shared autonomy. The system acts autonomously, but the user can direct or override the system at any point. Some shared autonomy functionality was developed on the WALRUS rover, but none has been implemented yet. The past project developed a system for assisted stair climbing and hallway following, to aid in indoor movement.

2.3.2. Data Distribution

"The biggest problem is the data, the informatics, because these people need to get the right data at the right time." - Robin Murphy [11]

One aspect of user interface that was not covered under the initial project is remote use of the data gathered. One of the primary purposes of the WALRUS rover is to gather data about potentially hazardous environments. While this data is useful for the direct user, or driver, it is also useful to users that may not be on site. Professor Robin Murphy, the director of the Center for Robot Assisted Search and Rescue (CRASAR), talks about this need in her 2015 TED talk. During the 9/11 response efforts, only a limited number of people were allowed into the "hot-zone" around the World Trade Center. This means that many experts who needed data from the robots were not allowed in with the robots. The ability to effectively distribute the data generated by a robot greatly increased the effectiveness of that robot. [9]

2.4. Situational Awareness

Disasters of the previous century as well as more recent ones showed the need for specialized unmanned equipment [18]. Beginning with the World Trade Center attacks, more robots were dispatched for disaster response and rescue operations. These experiences led experts to realize that robots' real potential in DR operations came from their potential for data acquisition and the need for it. Increased situational awareness meant more after-knowledge gathered about disasters as well as a safer operation for the rescuers, victims, and valuable equipment. In a simple comparison, while rescue robots in WTC got stuck in paper heaps because of poor vision, CRASAR UAVs quickly mapped the drastically changed area in Oso mudslides 2014, making the operation easier for all rescuers [9].

2.4.1. Simultaneous Localization and Mapping

One of the key challenges in robot navigation is creating a map of the task environment and locating the robot relative to this map concurrently. This concept, simultaneous localization and mapping (SLAM), is essential for any robot to gain autonomy. In a probabilistic robotics approach SLAM algorithms try to estimate the posterior (the map and the full path, $X_{t,}$ or location, x_t) from the priori data (sensor data and odometry) [14](Eq. 1). That is, the location of the objects and the robot is guessed by continuously updated distance measurements.

$$p(X_t\,, m|Z_T, U_T) \quad (1)$$

As for disaster robotics, SLAM has an indisputable importance. Tele-operation without sufficient situational awareness in unexplored and hazardous areas endangers the robot and the success of the operation itself. It is also impossible to assume the robot has a planar and consistent task space. This indicates a DR robot should be capable of SLAM in 3D to map and navigate clustered areas. [10]. One of the earliest tools used to process sensor data is the Extended Kalman Filter (EKF). The Kalman Filter assumes that measurements are linear functions of state variables with Gaussian distribution model (μ, σ2). In real time the KF recursively estimates the variables, then reads the next measurements and updates the weighted average for the next estimation [4]. EKF extends this method to nonlinear, but differentiable state variables. However, using normal distribution means high uncertainty in the posterior will cause linearization in the EKF to fail[12]. EKF tends to get slower in bigger sets of data but, earlier work n EKF-SLAM in three-dimensional space is present [13]

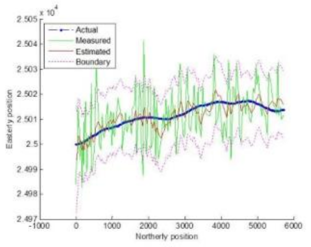


Figure 6: Estimation of sensor data by EKF

The Particle filter method became the preferred method when it was introduced to SLAM problem as FastSlam by Montemerlo. FastSlam presents the location variables as particles, whose distribution are generated based on robot's motion model and previous location relative to particle [14](2). This means, particle filter's performance depends on robot's kinematic model and particle population.

$$X_t^{[k]} \sim \; p(x_t)|x_{t-1}^{[k]}, u_t)$$

When a measurement is received, the algorithm computes the probability of the new distance measurement for each particle. Then the importance weight of these particles is calculated based on the desired probability. As the final step of the operation, a new set is formed from more important or "likely" particles [14].

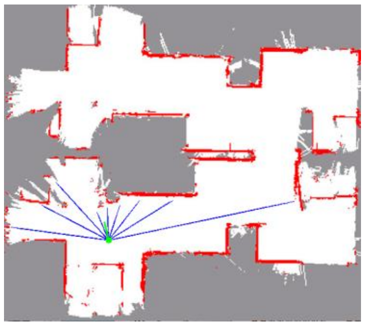


Figure 7: A map generated by particle filter .

Current SLAM algorithms provide robust solutions for their task-space, but the computational costs increase significantly as the area gets bigger. There are successful examples of SLAM, such as self-driving Google Cars, which in 2014 managed to get vehicle license from the State of Nevada [2]. In 3D SLAM, sensor data provided to estimators are usually a collection of range data, features (surfaces) or images with depth data (depth-map). So 3D mapping robots are mostly deployed with laser scanners, mono 3D cameras, stereo cameras or IR-depth sensors. Artisan is a laser-based mapping system developed for areas with radiological, chemical and fire hazards. Its LIDAR collects range data to map the area and create 3D datasets. This data is used to create a surface mesh and compared with presets to recognize dangerous or contaminated objects [13]. Pioneer, a robot specially designed for exploring Chernobyl nuclear plant, uses stereo cameras for SLAM. They perceive depth in a process similar to us humans and generate a photo-realistic 3D mesh of the environment. Missing areas in the image is a result of the robot's onboard lighting [13].

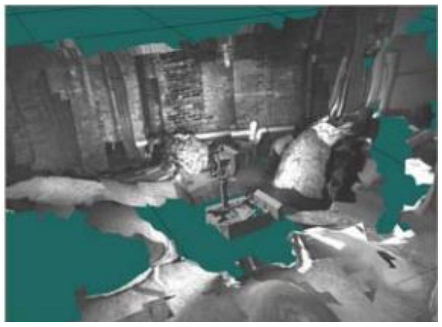


Figure 8: Pioneer's photo realistic surface mesh

One of the most versatile tools used in 3D SLAM is Microsoft's motion controller Kinect. Despite being marketed as an end user entertainment product, the Kinect has also seen wide usage in robotics research especially after 3rd party developers produced Linux drivers for the device, in 2010[5]. The device includes an RGB camera, a depth sensor consisting of an IR laser projector and a CMOS sensor, and multi-array microphone. It can stream RGB-D or IR-depth (depth-map) images up to 30 fps that can be used in 3D mapping or object scanning. It also provides invaluable situational awareness with object recognition/ tracking, voice recognition and even pulse detection [8]. Kinect is present on famous robots like PR2 and Turtlebot, as well as on WALRUS. A faster and powerful version, Kinect One, is released but is not fully utilized in Linux systems.

2.5. WALRUS Status

While the WALRUS rover was functional at the beginning of this project, there were a few issues that need to be addressed before the project can move forward. This section will cover the larger issues with the rover, as well as some of the features that were not fully implemented.

One of the largest obstacles that stood in the way of implementing any sort of processor intensive features, is heat dissipation. Since WALRUS is water tight, none of the components can be air cooled effectively. The CPU is connected directly to the top plate, which serves as a heatsink. While this serves adequately while the CPU is under low load, it is insufficient for sustained high loads, such as those seen during image processing.

Like many large robots with brushless drive motors, WALRUS has trouble stopping quickly. This is due to the high back EMF created when breaking with the motors quickly. WALRUS's long stopping distance leads to challenges in multiple areas. The most obvious is remote driver control, it is difficult for an inexperienced driver to estimate the stopping distance required, and break in time, causing the robot to hit an obstacle. The second is the limitation this places on autonomous movement. The increased stopping distance will effectively lower the maximum autonomous speed of the robot.

Originally the front pod motor drivers were not fully functional. This is due to a weakness in the USB communication used between the computer and the motor controllers, that makes them susceptible to electronically noisy environments, such as the inside of WALRUS. While this only minimally impacts the overall mobility of the rover, it drastically limits the rover's agility on stairs.

The original WALRUS driver interface is suitable for use by one user in close proximity to the rover. A user connects to the rover via Wi-Fi and opens the interface using a web-browser. The interface shows the camera views, system information, and allows the user to drive the robot. There is currently no support for multiple drivers, or for anyone off-site to view the data WALRUS gathers in real time.

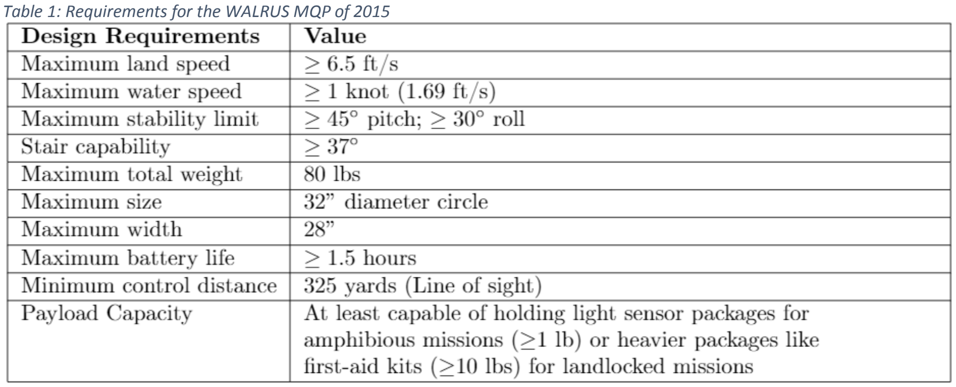
The original WALRUS team created a camera mast to support vision payloads. This mast had three actuated joints; the deploy, the pan and the tilt. The mast mounted on the top plate, and connected over USB and the power breakouts. The mast was developed to provide a better point of view to the operator. The major drawback of this mast, was that it was not water-resistant, and could not be used in inclement weather.

In the time between projects, this mast was worked on in various labs. Concurrently some of these labs moved around campus, or to a different school. During this time, multiple key components were misplaced. Many of these key components were not located until months into this project.

3. Requirements

This section will cover the technical requirements were for this project. The first requirement, was that this expansion does compromise the original requirements shown in Table 2: Design Requirements from our proposal.. The Additional requirements are outlined in Table 1, and explained in the following subsections:

Table 1: Requirements for the WALRUS MQP of 2015



Design Requirements	**Value**
Maximum land speed	$\geq$ 6.5 ft/s
Maximum water speed	$\geq$ 1 knot (1.69 ft/s)
Maximum stability limit	$\geq$ 45° pitch; $\geq$ 30° roll
Stair capability	$\geq$ 37°
Maximum total weight	80 lbs
Maximum size	32” diameter circle
Maximum width	28”
Maximum battery life	$\geq$ 1.5 hours
Minimum control distance	325 yards (Line of sight)
Payload Capacity	At least capable of holding light sensor packages for amphibious missions ($\geq$1 lb) or heavier packages like first-aid kits ($\geq$10 lbs) for landlocked missions

Table 2: Design Requirements from our proposal.

Design Requirements	**Value**
Shared Autonomy	
Stair climbing	Entry and exit of a straight flight of stairs with $\geq$ 12 stairs of width $\geq$ 3.5 ft
Obstacle Avoidance	Hallway of length $\geq$ 20ft and of width$\geq$ 5ft with no collisions
Diagnostics Readout	
Diagnostics readout	The LCD screen with all diagnostic data
Data distribution	
Data Distribution	Real-time 3D mapping and localization of the robot, available over the Internet.
Situational Awareness	
Depth camera feed	$30fps$@$640 \times 480px \sim 9fps$@$1290 \times 960$px
Visuals	3D Map with full robot model
Point of View	1st and 3rd person
3D Map Size	$\geq$ 1000 sq ft
Map update rate	$\geq$ 0.2 Hz
Odometry update rate	$\geq$ 10 Hz or $\geq 4m^{-1}$

3.1. Maintained Functionality

Any changes made to WALRUS must not impact the previous year's requirements that it has already met. The robot must maintain buoyancy in water. The robot must not overheat when we add additional required computation.

3.2. Situational Awareness

WALRUS will be able to generate a readable and accurate 3D map of an area approximately 1500 ft/sq. In this map, the robot model should be fully presented, including the orientation of the pods and mast payload, in real time to allow a third person view in GUI. The update rate should not be lower than 10 Hz, so the robot can't travel more than 0.25 m between updates at full speed. In order to maintain driving capabilities, the localization should continue even when the mapping fails to catch up with the update speed. Resolution of the video stream, should not fall under the limits defined by the stream rate of the camera. As an extension of the proposed features, if the driver support is ensured, the camera system can be upgraded to Kinect One for a faster video stream at higher resolutions.

3.3. Shared Autonomy

It is important to have modes that reduce the amount of low level control the operator must provide. This will reduce the focus and skill required to operate WALRUS. Two tasks we would like to have semi-automated would be stair climbing and obstacle avoidance. Automatic stair climbing will be considered implemented if the robot is able to enter, climb, exit 12 stairs with only forward directional input. Obstacle avoidance will be considered implemented if the robot is able to drive down a 20 ft hallway with only a forward input without colliding into the wall or obstacles

3.4. Implement Diagnostics Readout

Currently WALRUS has a LCD screen and 5 buttons that are not being utilized that was intended to be used for diagnostics purposes. It would be helpful for the operator to troubleshoot if a diagnostic readout was implemented. An operator will be able to navigate the LCD screen to see any message in the diagnostic topic.

3.5. Data Distribution

The primary objective of any reconnaissance robot it to convey data to the user. For most disaster response situations, WALRUS can be used in a reconnaissance role. To aid in this we will expand upon the WALRUS data distribution system. Our expanded system will support advanced driver controls, real-time viewing, and playback. All of these features will be available from any computer connected to the Internet.

3.5.1. Driver station

The driver station will have the same features as the current system, along with features based 3Dmapping. This interface will provide the user with a real-time map, third person driving view, as well as control over the shared autonomy functionality. The Interface will also allow the driver to take pictures with the WALRUS cameras that will be linked to the 3D map.

3.5.2. Real-time Spectating

All of the data gathered by WALRUS will be able to be viewed on multiple computers in real time over the Internet. The data will primarily be centered on the 3D map generated by WALRUS.

3.6. Mast

A mast comparable to the previous mast will be fabricated. The mast must have actuated deploy, pan, and tilt joints. The position of the mast must be able to be sensed within 1 degree.   The mast must be able to be mounted onto the rover, and draw power and control from the top plate. The mast must support the weight of a Kinect sensor. The mast must be able to withstand inclement weather, the IPx4 standard will be used for testing.

4. Methodology

Our main focus was improving the situational awareness of the users. To accomplish this, we designed and fabricated a new mast, and implemented 3D mapping and obstacle avoidance. In this section we discuss each aspect of our design, and our implementation.

4.1. Renewed Mast

The mast designed and build by the previous group functioned well, and much of our design is based on the previous design. We tried to use existing parts when available, although many of these parts where not located until after they had been replaced. The most significant improvement of our design is the water resistance, although other aspects had to be changed to accommodate this new functionality.

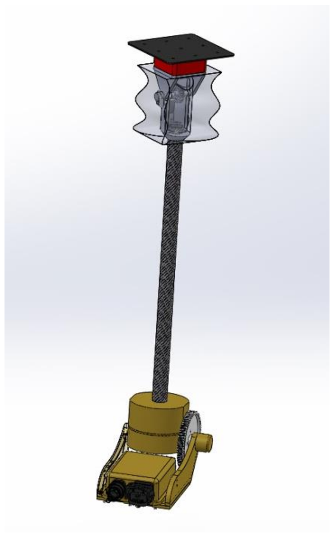


Figure 9: Initial mast design

4.1.1. Preliminary design

Our preliminary design utilized all of the components of the previous design, and protected the electronic components with covers. The goal of this design was to simply modify the existing design to meet our water resistance requirements. While this design had some issues that required it to be modified, the core concepts remained. The three areas of focus are the top gaiter, base enclosure, and the potentiometer and motor covers.

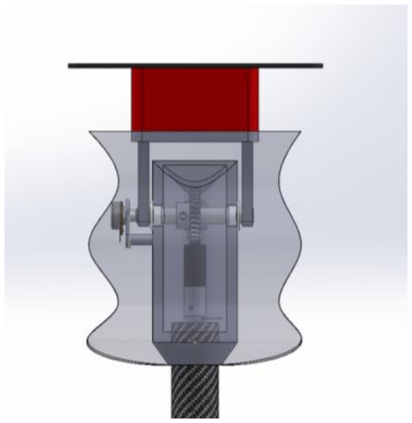


Figure 10: Preliminary top joint

The top joint contains a motor and potentiometer that need to be protected. We chose to use a gaiter design that covers the entire joint. The small shaft size makes it difficult to use a shaft seal, and the potentiometer would require a separate water protection method. Additionally, the enclosure would have to be drastically redesigned to all it to be sealed during assembly. The gaiter design protects all the components from water and is the simplest of the proposed designs.

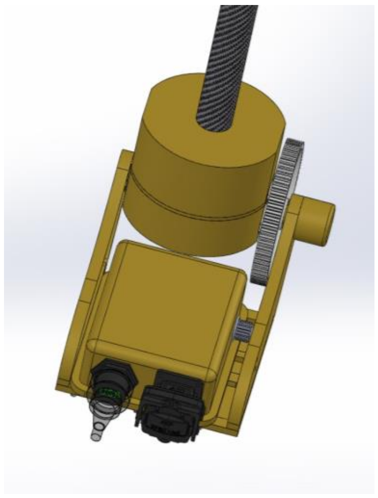


Figure 11: Preliminary base design

The base enclosure contains the deploy motor and the control board for the entire mast. Our initial design was a lid that would seal against the existing base plate between the two supports. A shaft seal would be used to allow the pulley to pass through the lid. Commercial water resistant connecters would be used to allow the mast to be connected to the rover.

The pan motor and potentiometer would be protected by a cover that extended the existing pan assembly and used a shaft seal to seal against the mast rod. The design interfered with the stowing of the mast and would later be changed, but is the basis for the final design. The deploy potentiometer was simply covered by a small 3D printed enclosure.

This preliminary design changed drastically before being implemented, but the way the problem was addressed stayed the same. The following sections document the changes made in each aspect of the design; the top, base, and pan and deploy. Design choices are discussed and the designs are analyzed.

4.1.2. Base

The second iteration of the base enclosure varies drastically from the first. Only one part from the original mast is used, the rest of the components are new. The preliminary design had multiple flaws that necessitated a redesign. Primarily it was determined that there was not sufficient space between the deploy motor and the right side support for an effective seal. Additionally, the compliance of the gasket would cause variation in the distance between the deploy pulleys, which could lead to slipping.

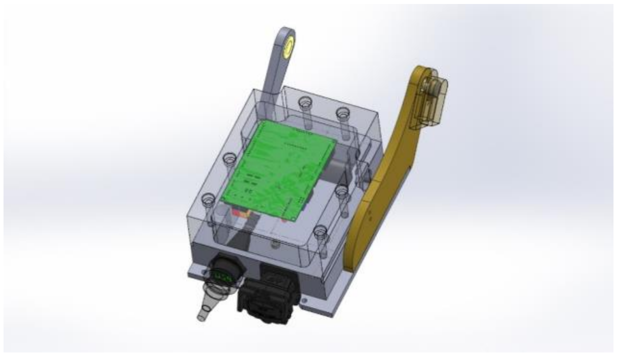


Figure 12: Mast base design.

These problems were addressed by redesigning the base plate and mounting both the motor and support to the base plate. Additionally, this allowed the seal to be raised off the deck, to prevent water from pooling next to it and leaking through. The design is comprised of a machined aluminum base that is covered by a HDPE lid. These materials were chosen for their low density and corrosion resistance, as well as machinability. The base must be strong enough to support the mast assembly so aluminum was needed. The lid is non-load-bearing, so HDPE was sufficient.

The original support was used on the left side, but lack of room on the right side necessitated a new support. This was designed to be bolted onto the side of the base, and match up with the original support. The design was refined in preparation for manufacturing. Any unnecessary small radiuses were removed, and final changes were made. Additional clearance was created around the connectors so they could be tightened. Then both the base and lid were manufactured in Turkey.

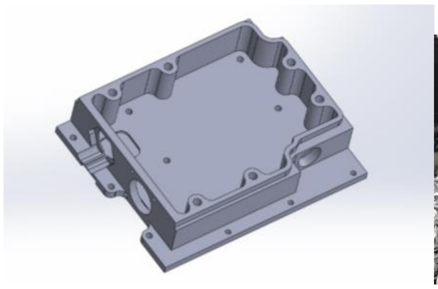


Figure 14: Base CAD

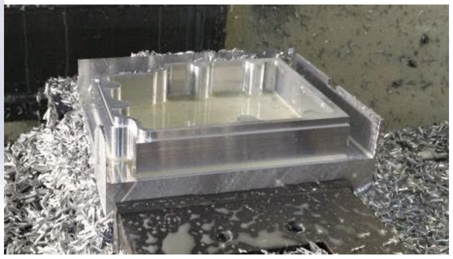


Figure 13: Machining of base

Preliminary analysis was performed on the supports using solid works simulation. An assembly was made with the supports and base. Virtual bolts and constraints were added. A static load of 100lb was applied to simulate the force applied by the mast during deceleration. The results for stress, strain and deformation were all deemed too been within tolerances.

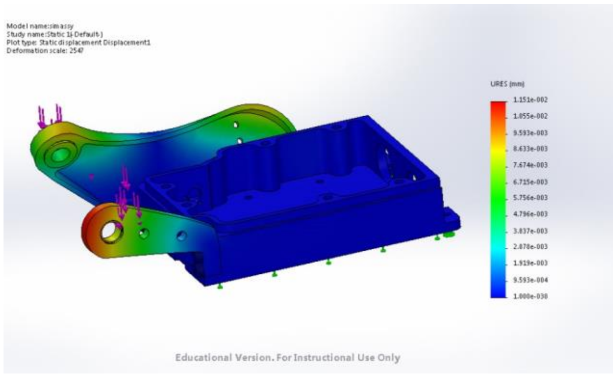


Figure 15: Deflection

The new support was machined in Washburn shop, and proved to be challenging to fixture. Because of the irregular shape of the part, a vice could not be used to fixture the part. Instead holes had to be drilled into it to bolt it to a sacrificial plate during the second operation. Due to an error in probing, the first attempt was ruined, and new one had to be made. When assembling the mast, and additional error was found. The pan assembly does not clear the base lid as shown in the figure below. New supports would have to be designed and manufactured.

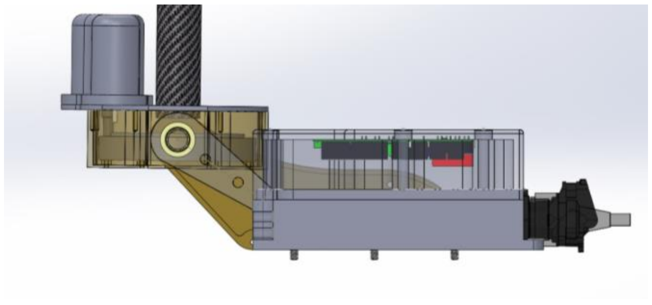


Figure 16: Pan base intersection

The redesigned supports would be subject to various constraints. First the supports must position the pan assembly in a way that it will not collide with the base lid, and will allow the mast to be stored parallel to the top plate of the rover. In order to accommodate standard sized belts, the distance between the pan assembly and the base pulley must be one of a few standard lengths. These constraints guided the design of the new supports.

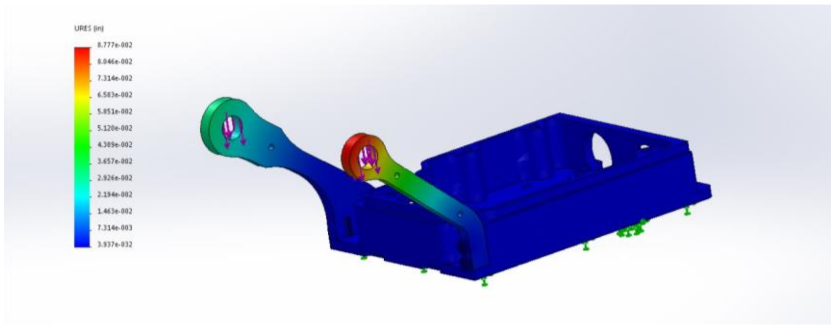
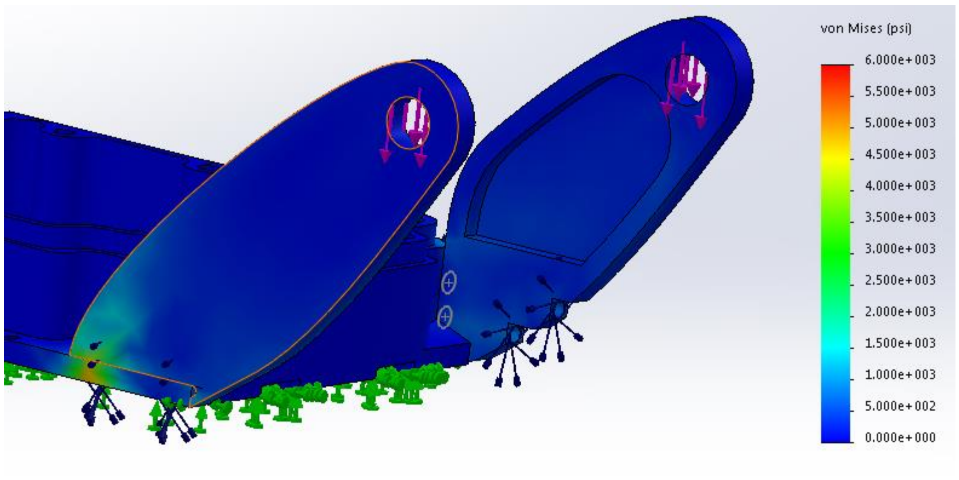


Figure 17: Deflection

The first design of these supports was a simplistic design that did not utilize pocketing like the original supports. This would make them much easier to manufacture, as they could be made on a water-jet. Once the design was complete analysis was performed on the supports. Based on this analysis and the input of our advisor, the supports would deflect too much. It was determined that a design utilizing pocketing was need.



The supports were redesigned to have a larger cross-section to add rigidity. The added weight was offset by pocketing. This gave the final design a much better strength to weight ratio then the previous design. Analysis was performed and the deflection was determined to be within tolerances. Because of the pocketing the parts had to be machined in Washburn.

4.1.3. Top gaiter

Once the gaiter design was selected to protect the top of the mast, analysis began on how to realize this design. Three feasible options were found, and evaluated; a commercial CV join gaiter, a custom made neoprene gaiter, or a 3D printed gaiter. The CV joint gaiter was initially chosen as a reliable off the shelf part. Ultimately it was rejected over concerns that it would be too stiff. The 3D printed gaiter was also rejected due to concerns about flexibility and water resistance. The neoprene gaiter was chosen, after a neoprene sample was evaluated.

The gaiter was created by placing the neoprene over the top joint and pinning it in place. The joint was actuated, and adjustments were made accordingly. The neoprene was then sewn and glued together leaving a hole at the bottom so the gaiter could be removed. This hole was sealed with acrylic plates.

4.2. Driver Assistance

We wanted to implement obstacle avoidance because it might be difficult for the user to identify obstacles because of WALRUS's limited view ability. The desired behavior of obstacle avoidance was to not have it take driving control away from the operator but to assist the driver by slightly changing the operator input. This would allow the operator to determine general direction to go while still reducing the change that WALRUS will hit an obstacle. Since WALRUS uses ROS we had our choices of two languages C++ and Python. We decided to use Python because it was the language we were all most familiar with. We broke down obstacle avoidance down into two steps, identifying obstacles and adjusting user input.

Commented [OA1]: Can you emphasize it's driving characteristics. I kind of mentioned cameras in the next one.

Our original plan for obstacle identification was not the same from what we implemented. Our original obstacle identification was to create a 2D grid in the form of a 2D array. The next step would then be taking the points from the Kinect and fit them into a grid cell of this 2D grid. At this point the data would be in the form of a 2D array of list of points. We would then take this 2D array of list of points and find the slope for each grid cell based on adjacent grid cell. It was then easy to have a slope threshold that would identify if a grid cell was considered an obstacle or not. During implementation we discovered that this design was to draining on the CPU and we had to develop a simpler design.

Our next design for obstacle identification would still create a 2D grid but would randomly sample a portion of the points from the point cloud. A point would be considered an obstacle if it occupied the same vertical space as WALRUS. This design may not have been not as accurate, but it allows the obstacle avoidance to update at rate that was about 20 time faster and used 4 times less CPU time.

For adjusting user input the obstacles in the obstacle map generated from obstacle identification were expanded to reduce WALRUS to a point. WALRUS would determine the distance to an obstacle at different angle intervals. WALRUS would then scale forward motion based on how close obstacles were.

There would also be a slight turn depending on where the angle of the furthest obstacle was. During testing we adjusted the factor of which obstacle avoidance effect user input.

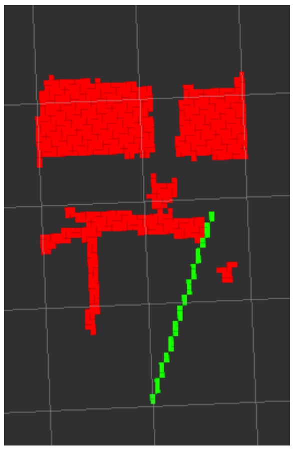


Figure 19 Obstacle map with line of furthest obstacle

4.3. Three Dimensional Simultaneous Localization and Mapping (3D-SLAM)

We established that WALRUS's remote operation capabilities should be augmented. Situational awareness and environmental knowledge is crucial for a successful remote operation. WALRUS already has three camera feeds on its base and attachable environmental sensors, making itself a capable data collector. But given the cluttered and undiscovered spaces the WALRUS is designed for, modeling the environment and the robot itself also a necessity.

In the initial design, we intended to develop our own, simpler 3D-SLAM algorithm. The mapping algorithm would simply stitch the point clouds generated from Kinect's depth image stream. WALRUS does not have a graphics processing unit (GPU), so efficient calculation of visual data is only possible at the central processing unit (CPU) level. Installing a GPU would increase the depth-registration and overall graphics performance while putting less strain on the CPU. At time of this design, WALRUS had an important CPU overhead problem, making the design decision critical. Also, additional (graphical) processing power would come at the expense of internal space, power consumption and heat management. Eventually, this led to the decision of using the close-proximity and newly scanned point cloud data as a local map available on the robot (Figure 20). A global map would be produced on the driver station computer by combining the local point cloud data at a slower rate and transforming them relative to map's coordinate frame. This also benefits the obstacle avoidance module, which had to access the local occupancy (point clouds) data faster than the global map's update rate.

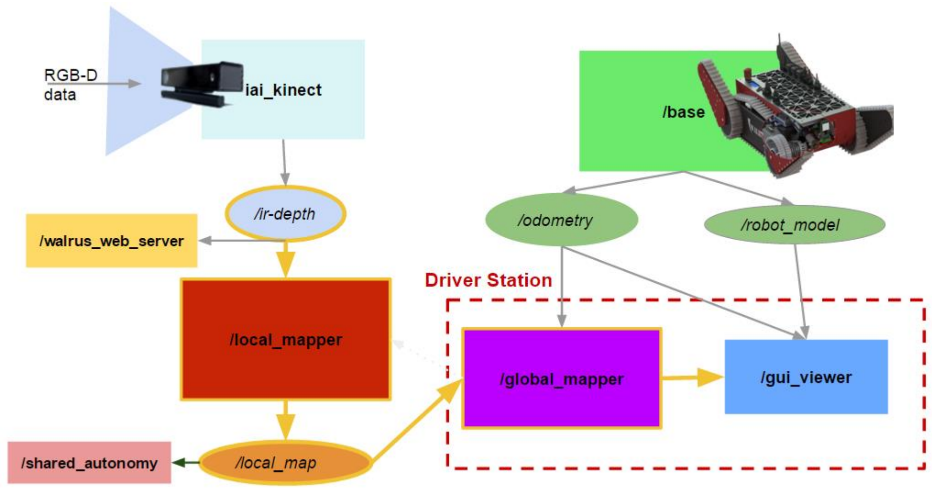


Figure 20: Initial 3D-Slam/Situational awareness module overlay.

A lightweight 3D-SLAM algorithm is beneficial in terms of resource usage, but lack of features might cause problems. The integrity of the map was not guaranteed as it assumed the navigation data of the WALRUS was accurate enough. But since WALRUS no longer possessed the IMU and GPS modules it previously had, we realized this design choice was impractical.

Considering the implementation time required for a sophisticated SLAM package, we decided to use one of the available open source packages. Our choice of 3D-SLAM implementation, RTabMap [6][7], presents fast, reliable and customizable solution for mobile and static robots. It can use IMUs, LIDARs, Encoders or fake 2D laser scan from Kinect as well as visual odometer to update location measurement. It builds the map based on a weighted trees which hold the point-cloud readings as nodes. If needed, the algorithm uses loop-closure to update the localization and combine the patches of map. Finally, the transformations between Kinect's optical frames and robot's base and map calculated by reading WALRUS's URDF model. The package also uses RViz API [12] for its visualization, which is our intended software for 3D scene representation.

We exploit the configurability of RTabMap in our launch configurations to switch between local and remote mapping, physical configuration of the robot, Kinect placement and the sensor selection for the position.

As our initial designs foresaw, bandwidth usage during a remote operation is also a concern. So, we measured the network load for different mapping configurations using the *rostopic rate* tool for an informed decision. If the mapping is done locally (robot-side) and sent to the drive station for visualization, the map data stream initially uses a few KB/s but quickly reaches tens of MB/s. Since the camera views are needed anyways, video streams also add up to the network usage, and, of course, locally mapping and visualizing the map only costs CPU power. On the other hand, directly sending odometer and Kinect data to drive station for mapping and visualization uses a constant bandwidth. In this case, consumption amount is at the discretion of the user, and it equals to the image size of the RGB and depth images times the frame rate. That being said, we throttle the video stream to a lower rate for moderated network load and relay the camera nodes on the driver station side to prevent multiple connections.

The RTabMap nodes are running independent of the GUI and RViz can optionally be used to visualize the map and the robot model.

4.4. Graphical User Interface (GUI)

Increasing overall ease of use of WALRUS and enhancing the situational awareness of the driver was one of the main goals of our project. We wanted to replace the old interface, which focused on the video streams of the cameras on the WALRUS and/or the mast.

This interface was a browser-based JavaScript applet that required WALRUS to run a web server (**Error! Reference source not found.**). The interface was partially dynamic with some incomplete features and it had displays for mast and pod control as well as diagnostics print-out. On the other hand, its design didn't make use of the whole display and wouldn't leave much space to development. Concurrently, the web server could use up to 40% of each four CPU core on WALRUS.

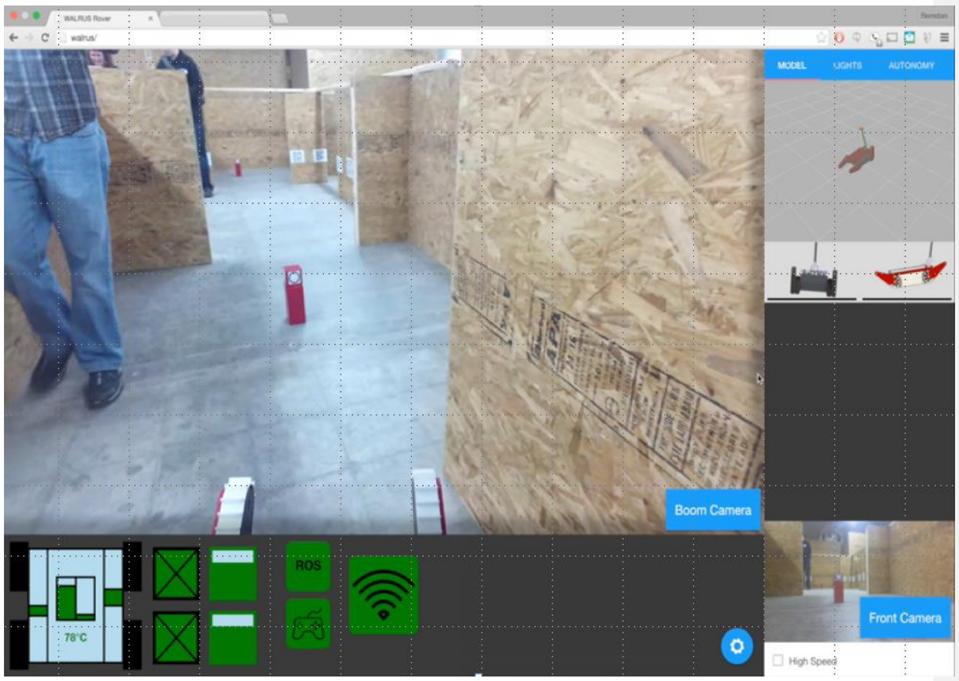


Figure 21: Old WALRUS interface reached through a browser.

The GUI software consists of the *QNode* as the model class, which holds the ros communications and system data. The remaining classes form the interface part of the application and they rely on Qt API. The interface and the *QNode* communicates by using Qt's callback system: Slots and Signals [16] . The *QNode* on the other hand creates a ros node to communicate with WALRUS's ros framework.

Commented [OA2]: Qt Signals and slots reference.

Commented [OA3]: Rqt_graph that shows the nodes /topics pf the interface and mapping

4.4.1. 3D Scene

The main feature of the GUI is a full screen view of the environment that includes the 3D model of WALRUS based on the live data and more importantly, 3D map of the environment and most recent point cloud. These elements are placed on an RViz::RenderPanel and positioned according to the transformation between their coordinate frames and the panel's reference frame. Consequently, any display class added to the panel should have this transformation (TF) data available.

4.4.2. Graphical Overlay

The graphical overlay on the 3D scene is designed to provide more knowledge to the driver. Therefore, its main function is to visualize the diagnostics and environmental data streamed by WALRUS. Some of this functionality could be provided by using Qt's existing classes. That also meant these components would come with support for keyboard and mouse control to control their behavior such as selecting, rescaling, hiding and redisplaying. Meanwhile, some components required us to extend existing display elements.

The radar, or orthonormal view is another RViz::RenderPanel with a separate panel manager. Which also menas, more displays or visual represntaitons can be added independently form the main 3D Scene. The camera feed of the WALRUS is not a video stream but a series of *sensor_msgs/Image* bring published o a topic. Therefore, it can't be displayed by a video player that internally handles video streams. The image stream is visualized by converting each incoming image to appropriate format using OpenCV libraries and updating the widget after each conversion (Figure 22).

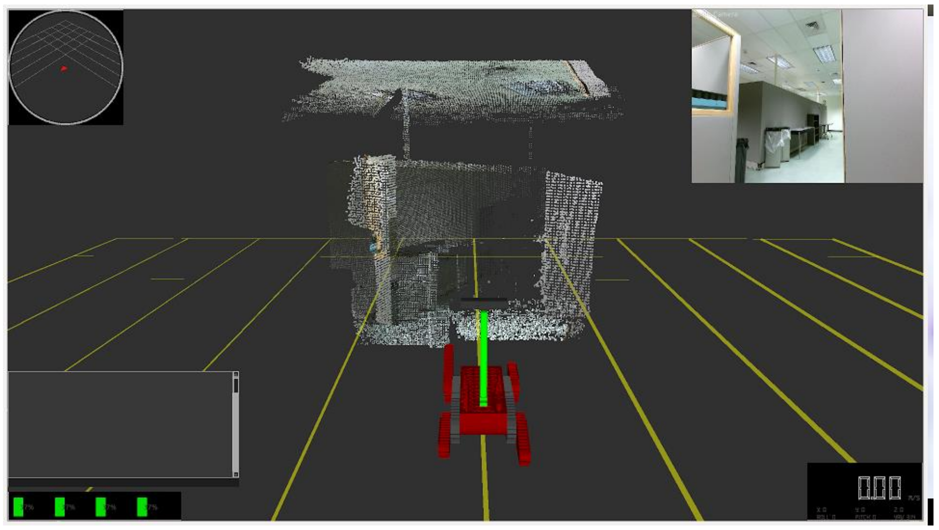


Figure 22: New GUI layout.

We designed our own extensions of QWidget class to provide specialized behaviour. They focus on an updatable image and an optional progress bar for displaying values.. The warnings have fade-in/fade-out animation. These indicators and warnings are grouped in panels that automatically handle their placement, and the orientation can be specified as horizontal or vertica(lFigure 23).

Base Indicator:

A *QWidget* for the basis of indicators. It holds an image (*QLabel*) to display and can place a red diagonal line over it to indicate error. It also defines callback funciton that calls the update method of the respective *Indicator* class. The *error_value* for all Indicators is defined as a constant by this class.

IterIndicator:

Iterative indicator class. It has multiple images added after initialization. These images are updated according to the _value attribute of the class. If _value equals to the error, it reverts to the specified default image, and displays a red diagonal line over it.

BarIndicator:

BarIndicator extends *BaseIndicator*. It holds an image and a progress bar (QProgressBar), which can be placed horizontally or vertically if specified on initialization. The progress bar will show the _value the indicator has. If _value attribute equals to *error_value*, this means the data or the associated device is faulty and a red line is shown on the image.

BatteryIndicator:

BatteryIndicator extends BarIndicator. This child class has a battery icon as the source image and the progress bar is directly beneath it to represent a battery gauge. If the charge of the associated battery doesn't exist, the red line is placed above the indicator.

Warning:

Warning extends the BaseIndicator. Warning holds one image but the image starts to fade in and out on display.

IndicatorsPanel:

An aggregation of any Indicators. Any Indicator added to the panel will be aligned vertically or horizontally according to the panel's specified direction.

WarningsPanel:

WarningsPanel is an IndicatorsPanel. Unlike IndicatorsPanel, this aggregation of Warnings doesn't necessarily show every warning it is parenting. If a warning is enabled, it will be aligned to the last visible one, depending on the direction specified by the panel, and will start its animation. If a Warning is removed, the remaining Warnings will be shifted towards the prior direction.

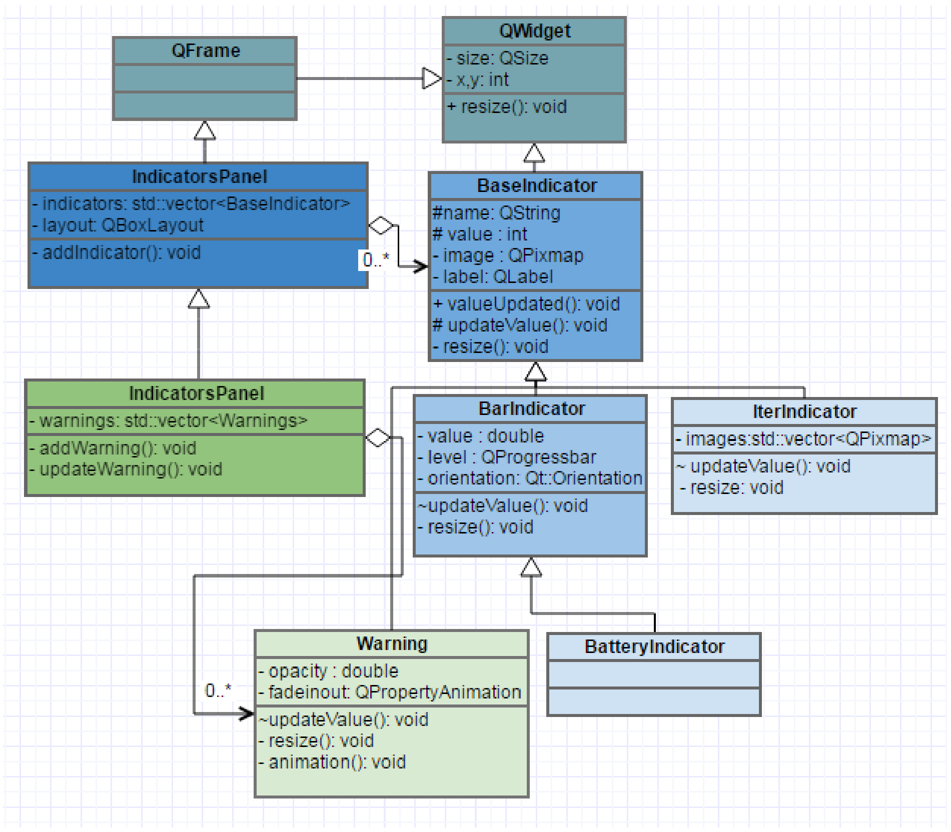


Figure 23: UML Diagram of the Interface.

Commented [OA4]: This will become an UML diagram, I can't access the software before we send it today. It will be replaced before the final submission.

4.5. Maintaining Functionality

When we started the project, WALRUS had impeded movement. The frontal pods would not move because the recently installed motor drivers wouldn't work over USB due to electromagnetic noise inside the WALRUS and couldn't be configured for serial at the same time. We managed to get the motors functioning y modifying the driver files to include a flag for USB-Serial selection. But interestingly, they would cease to work as soon as we bolted them to their place. By shielding the problematic drive with construction tape, and leaving the screws of the driver bolt looser, we managed to get both sets of pods to work.

During our runs we realized that one of the CPU cores of WALRUS was always saturated. Initially we assumed that was because of a bad multicore implementation. Further inspection showed us that the node controlling the mast was running as fast as it can without a rate, hence choking the CPU. After fixing this issue, idle CPU usage of the WALRUS was as low as 5%. This improvement also resulted in less heat production.

WALRUS was missing a camera and the remaining ones had their USB ports changed, causing the node managing them to malfunction. We restored the missing camera, as well as reconfiguring the other ones. We also added an argument to disable cameras when launching the WALRUS.

5. Results

In this section, we go over the implemented designs for the project. We elaborate the resulting functionality and shortcomings of these features and draw comparisons to the design requirements specified in the proposal and reiterated in this paper.

5.1. Mast

We were able to maintain the full functionality of the original mast. On top of that, our new design is water resistant. The mast has the three required actuated joints with potentiometers, which are protected by 3D-printed covers. Additionally, the range of motion of the mast was not limited by any design changes.

The mast was evaluated based on the IPx4 standards. Preliminary tests were done on the top gaiter, by filling it with water. The gaiter only leaked through the mounting holes as expected, indicating that the seals were intact. The system was tested as a whole by spraying the required 10 L/min of water over it and splashing additional water over it for 5 min.

The base enclosure stayed dry, which was vital as the most sensitive electronics are protected by the base enclosure. The top gaiter and other covers, experienced mild liquid intrusion, but not enough to damage the components. Overall the system survived intact and would be able to withstand moderate to severe weather.

5.2. Driver Assistance

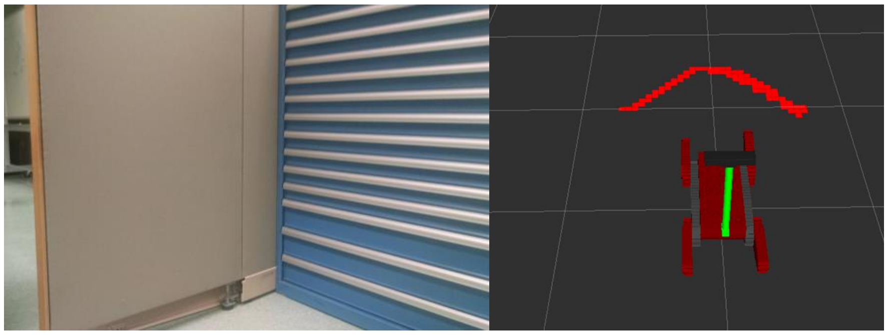


Figure 24 Image from Kinect and obstacle map

While testing obstacle avoidance we made some observation on it performance. Obstacle avoidance preformed best when driving straight into a wall. When driving straight into a wall, WALRUS would begin to slow down at 2 meters and almost come to a stop when a few centimeters away. If walrus came at a wall at an angle closes to perpendicular to the wall it would cause the robot to turn away from the wall. Obstacle avoidance sometimes bumped into a wall when the wall was near parallel. Since obstacle avoidance used the point cloud from the Kinect and not the map WALRUS had a limited viewing range. While testing, we also adjusted the percentage of point we sampled from the point cloud. We found that sampling around 4% of the point would still successfully generate an accurate obstacle map as you can see in Figure 24. One key mistake we made early on was the decision to program obstacle avoidance in Python instead of C++. C++ has a very efficient point cloud library that we failed to take advantage of. We probably could have implemented our original design for obstacle identification.

Commented [OA5]: There are Python bindings for pcphttp://www.pointclouds.org/news/2013/02/07/python-bindings-for-the-point-cloud-library/

5.3. Mapping

The 3D-SLAM related nodes run on the driver station computer, hence, long sessions of mapping do not affect the performance of the WALRUS. In other words, the map can get bigger as long as the driver station computer can support storage and graphics-wise. The map can also be carried through multi sessions.

Since the SLAM algorithms are dependent on the location measurement, map's accuracy depends on the precision of location data. As the mast or the robot gets unstable, or if the odometer data coming from WALRUS is off due to the factors like traction, the map also starts to show artifacts. This can be by overcome by attaching the Kinect to the top of the static base so the vibrations are eliminated enough to use visual odometry. This configuration is supported through launch files.

One problem with the cameras is that since that are attached to USB 2.0 ports, no two camera can be watched together. This includes using cameras while mapping through Kinect.

The update speed of the map depends on the clock rate of the mapping node which can be specified before launch. As long as the tree nodes of point clouds can be merged successfully, the map will be updated successfully. The part of the point cloud immediately in front of the WALRUS is directly displayed as the data comes in from the Kinect and computed. This operation's rate depends on the rate of the image stream we throttle before transmitting, which defaults to 5 Hz.

5.4. Graphical User Interface

Our GUI implementation is a full screen 3D application with overlaying 2D indicators. Using RViz API enables us to visualize anything using markers or through 3rd party own RViz classes. Computer vision applications which allow transition from a camera frame to world frame are such use cases.

The interface can be used to replay the data through rosbag as it will still subscribe to the same topics

The diagnostics data used by the indicators are parsed by another node and can be externalized or isolated depending on the usage. The indicators itself show the batteries, PC resources, environmental sensors, warnings, orthonormal view (radar), cameras, Wi-Fi and joystick status as well as navigation data (Figure 25).

By creating an instance of the joystick controllers in the driver station computer, the joystick can be recognized as “plug ‘n play” and can be used to control WALRUS.

Fixing the cameras onto certain coordinate frames and angles such as orthonormal or third-person following views can be set using RViz::ViewControllers. Currently there is lack of documentation on the topic and the current developers are not familiar with the solution.

While overlaying indicators of the GUI use Qt, 3D scene is an RViz::RenderPanel embedded into a QWidget. This means, the 3D Scene is using an OGRE renderer which is external to Qt. Because of that, overlaying Qt Widgets are not able to access the pixel data below and cannot implement transparency. This can be overcome by getting these two main components to work with the same renderer or by indirectly feeding the pixels below to the overlaying widgets.

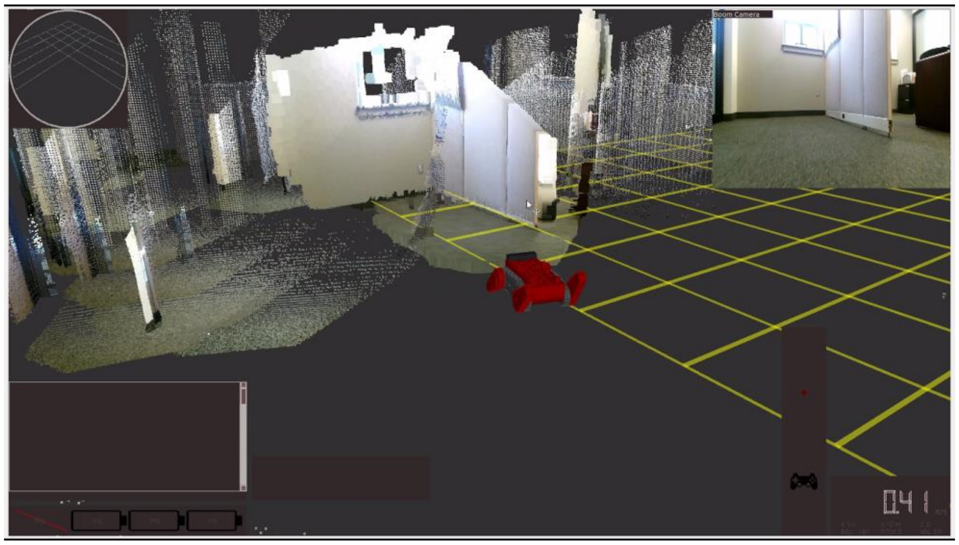


Figure 25: New Interface without data feed to indicators

Commented [OA6]: Should I number the components

6. Conclusion

In this report, we presented you the new features we implanted on WALRUS. Our proposed requirements and work focus on improving the operational functionality of WALRUS and situational awareness of the robot operator while maintaining the previous capabilities of WALRUS.

We managed to reach our goals by producing a new water-resistant mass, implementing obstacle avoidance and creating a new GUI application with 3D visualization. We also managed to overcome hardware and software issues WALRUS faced such as excessive CPU usage and non-functional pods.

Due to the limited time, we had to forgo stair climbing, a feature that was planned for since robot's initial creation. We also indirectly alleviated the heat problem by lowering the CPU usage of the system.

Future work may include augmenting the obstacle avoidance into a full driver assistance pack that is modeled after autonomous and tele operated vehicles such as rovers and unmanned automobiles. The GUI application is able to support more indicators and specialized view mode for spectating and replaying data. Face and obstacle detection feedback can also be displayed in the 3D scene of the application.

Reflecting back at our design requirements and considering the work completed, we think we made the WALRUS rover a better system for robotic search and discovery and disaster response.

Bibliography

[1] Fireengineering.com. The use of robots as a search tool, 2015.

[2] Mark Harris. How Google's autonomous car passed the first U.S. state self-driving test, 2014.

[3] Irobot.com. iRobot 510 PackBot, the multi-mission robot, 2015.

[4] Rudolph E. Kalman. A New Approach to Linear Filtering and Prediction Problems. ASME Journal of Basic Engineering, 1960.

[5] Caleb Kraft. Open source Kinect contest has been won, 2010.

[6] M. Labbe and F. Michaud, "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation", *IEEE Trans. Robot.*, vol. 29, no. 3, pp. 734-745, 2013.

[7] M. Labbe and F. Michaud, "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation", *IEEE Trans. Robot.*, vol. 29, no. 3, pp. 734-745, 2013.

[8] Msdn.microsoft.com. Kinect for windows sensor components and specifications, 2015.

[9] R.R. Murphy. Activities of the rescue robots at the world trade center from 11-21 September

2001. IEEE Robotics and Automation Magazine, 11(3):50–61, 2004.

[10] Hartmut Surmann. Slam in rescue environments, 2008.

[11] TED. These robots come to the rescue after a disaster. 2015.

[12] "RViz: Main Page", *Docs.ros.org*, 2016. [Online]. Available: http://docs.ros.org/indigo/api/rviz/html/c++/. [Accessed: 26- Apr- 2016].

[13] Scott M. Thayer. Four generations of robotic mapping and exploration in extreme environments. Pages 2–4.

[14] Sebastian Thrun. Simultaneous localization and mapping. Robotics and Cognitive Approaches to

Spatial Mapping, pages 13–41, 2008.

[15] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT Press, 2005.

[16] "Signals & Slots | Qt 4.8", Doc.qt.io, 2016. [Online]. Available: http://doc.qt.io/qt-4.8/signalsandslots.html. [Accessed: 05- Feb- 2016].

[17] J. Weingarten and R. Siegwart. Ekf-based 3D slam for structured environment reconstruction.

2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005.

[18] Zhe Zhang, Goldie Nejat, Hong Guo, and Peisen Huang. A novel 3d sensory system for robot-assisted mapping of cluttered urban search and rescue environments. Intelligent Service Robotics, 4(2):119–134, 201

[19] O. Akyildiz and C. Wolf, "OAkyildiz/walrus_expansion", GitHub, 2016. [Online]. Available: https://github.com/OAkyildiz/walrus_expansion.git. [Accessed: 26- Apr- 2016].