A Heterogeneous Swarm Solution for Disaster Reconnaissance

Feasibility Study

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree in Bachelor of Science
in
Robotics Engineering, Mechanical Engineering, and Computer Science

by

Carolyn M. Lowe

Turner J. Robbins

Jarrett A. Sarnell

Mathew C. Schwartzman

Lucy G. Stuehrmann

Abstract

The disaster mitigation process is dangerous to search-and-rescue (S&R) personnel. Organizations work to develop robotic technologies for disaster response. However, disaster robotics is often characterized by large, bulky, and expensive systems. The goal of this MQP is to analyze the feasibility of a simple, cost-effective heterogeneous robot swarm to map a disaster location for on-site S&R personnel. We designed a ground robot prototype that can be dropped by a drone. We tested collaborative algorithms to perform mapping of cluttered environments by both the ground robot and the drone. Our study also involves tests on durability and disposability of the droppable robot and scalability tests of the mapping algorithms.

1.0 Introduction

As disaster scenarios become increasingly more common, the threat of injuries posed to search and rescue personnel by unstable or unmapped areas also increases (Garrett, 2018). In order to reduce the danger to human life caused by hazardous areas, known as hot zones, disaster robotics systems are being developed to give on-site experts situational awareness (Murphy, 2014). Unfortunately most of these systems are slow, expensive, or heavy. In order to address these issues, we propose the development of a heterogeneous swarm system to map an unknown room consisting of three types of robots: an unmanned aerial vehicle (UAV), an unmanned ground vehicle (UGV), and a remote computational unit (RCU). This system provides an autonomous, scalable swarm that reduces the need for expert training and human interaction as well as an alternative low-cost option using inexpensive parts that could be bought in bulk to reduce the overhead of tooling and manufacturing costs.

Through our feasibility study, we determined how best to implement a heterogeneous robotic swarm for use in disaster scenarios. The goals we set for this study are divided into two different research branches: a simulation branch and a physical component branch. For the simulation branch, we implemented the UAV and UGV in a simulation environment that would allow us to test code and the capabilities of the swarm in a dynamic setting without having the physical robots ready to use. For the second branch of our research, we focused on physical implementation of UAVs and UGVs. For the UAV, we specifically wanted to test the capabilities of the sensors, particularly the camera. For the UGV, we created a robot that would be able to withstand the hazards of operating in a disaster area, namely the ability to be dropped and still be able to operate and sense correctly, while maintaining a low-cost and low-weight.

We created simulations for the UGV showing the integration of the major components of the full scalable system. Additionally, we created a UGV prototype that shows the durability of its components and usability after drop testing. Finally, we integrated image processing software and a monocular simultaneous localization and mapping (SLAM) framework to have the UAV survey and be able to locate the UGV on the ground.

2.0 Background

2.1 Increasing Disasters

Over the past two decades, climate-based disasters have increased in severity. As global urbanization steadily increases, more and more of these climate-based disasters occur in urban areas where larger structures are being built with the potential to become large-scale hazardous disaster areas.

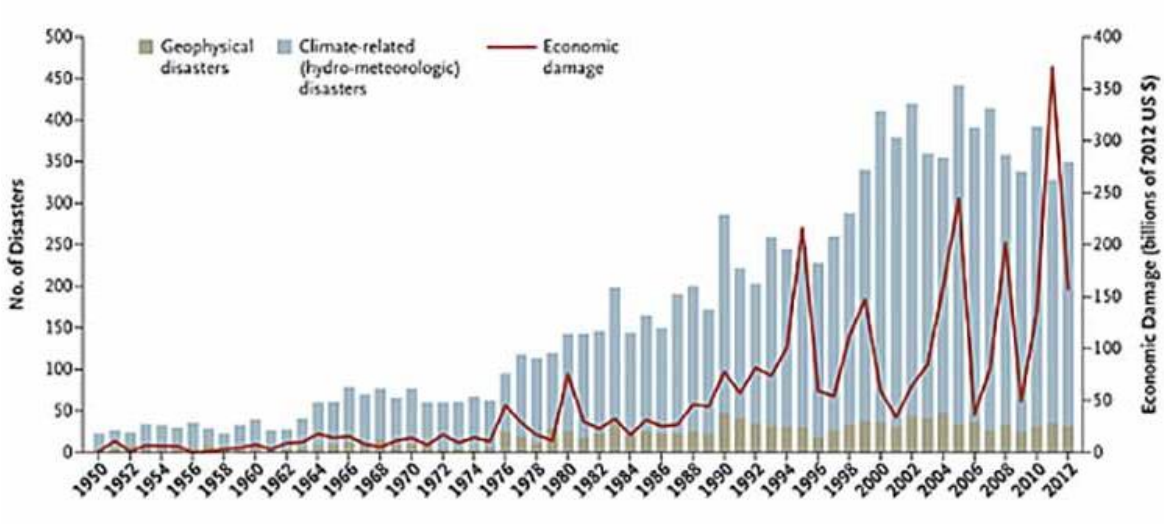

Figure 1: Disasters and Economic Damage per Year

A major contributing factor to the increased disaster rate has been an increase in global warming. As seen in Figure 1 (above), there is a direct upward trend of climate-related disasters correlating with high economic damage. The year 2011, which had the least disasters in over a decade, still had at least fifty more disasters than any year predating 1998. The year 2011 additionally saw the highest economic damage of any year, at over 350 billion dollars. One of the major contributing factors was the 2011 Japan earthquake and tsunami. The disaster resulted in the destruction or damage of 128 thousand houses (Ijba et al, February 2013). While this was an

uncommonly severe disaster, as we can see from the graph above, catastrophes in general are happening more frequently, which means that the cumulative costs of disasters will increase accordingly.

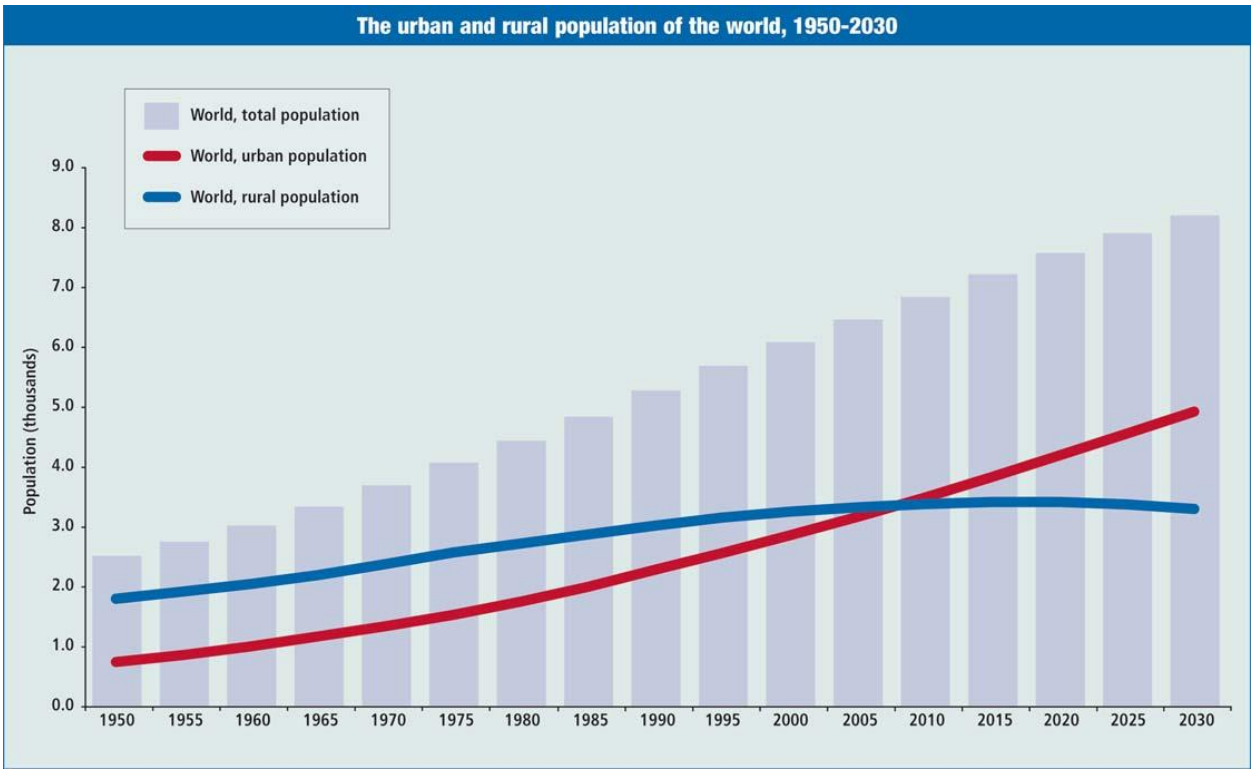

Figure 2: World Population

As we can see from Figure 2, more people are living in urban environments. As more disasters occur, and as the world population living in urban environments increases, more disasters occur in an urban environment. Due to the growth of urban infrastructure necessary to accommodate increasing population, the total cost of disasters will also increase dramatically.

2.2 Disaster Robotic Systems

In a disaster scenario, disaster robotic systems are deployed to assess the situation of the areas and work in SAR operations. Most of these units are single robots that are designed to be

all-in-one systems. They come fully equipped with sophisticated sensor suites and large, rugged chassis designed to traverse rubble and unstable terrain. These robotic systems have many complex subsystems and sensors, which causes these units to be expensive, some costing up to $200,000. Some common disaster robotic systems used are the Wolverine V2, the iRobot PackBot, and the Inuktun VGTV Xtreme. These systems are commonly used in mines for mine recovery and in various natural disasters to aid in cleanup and search and rescue (Murphy, 2014).

These complex single-unit systems are highly useful for SAR efforts. Their complex and sophisticated sensor systems alone can generate a variety of information for on-site experts to use, such as images of areas where personnel cannot traverse. They can also move rubble that might obscure rescue workers' paths. In a disaster area, the environment frequently has uneven ground and small obstacles the robots must traverse, so these robots are built to be powerful and rugged enough to travel through these areas. These robots therefore have powerful motors and all-terrain tires or tank treads. Many of these robots also have an arm apparatus that is used to move objects or retrieve samples in the field.

Just as these robots have many uses in SAR situations for surveying and exploring disaster areas, they also have many drawbacks. As these machines carry complex sensors and need powerful motors to move easily through rubble, they need to be large enough to house all the components they use. This also means that these robot can weight over 1000 lbs, which makes them unsuitable for areas that cannot support their weight, such as unstable higher floors in a damaged area.

3.0 Methodology

The large-scale operations involved in urban search-and-rescue call for multi-robot systems capable of autonomous exploration and mapping. To realize this vision, we focused on the integration of two types of robots - a UGV and a UAV, which we simulated in ARGoS.

For the UGV, we decided to focus on three main aspects: the weight, the cost, and the durability. First, since we intend for the UGV to work in hazardous environments, the UGV must survive moving in hazardous terrain, so it needs to be durable. Next, since the UGVs are meant to work in swarms, and a high production cost would make it difficult to deploy a large multitude of robots, we need to UGV to be cost-effective. Finally, since the UAV must carry the UGV, the UGV must be light. We focused on these areas above others because implementing all three aspects simultaneously is challenging.

To best generate a 3-dimensional representation of the environment efficiently and with minimal uncertainty, our system uses an UAV equipped with a single camera to generate a point cloud through a monocular SLAM algorithm. This algorithm is run on the RCU that simultaneously merges the UAV output point cloud with one generated by the UGV. This merging process is characterized by a computer-vision-based subsystem that helps localize the UGV in a global frame of reference.

Our proposed solution for exploring an unknown room in a safe and distributed manner involves two phases. In the first phase, any number of UAVs will be deployed into an unknown room. The UAVs begin to explore the unknown room by relaying scan data to the RCU. The RCU uses these scans to construct a 3D occupancy grid. Then, the RCU determines the locations of all frontiers, and assigns frontiers to each of the UAVs. The UAVs then navigate to the frontiers while continuously transmitting sensor data to the RCU. This repeats until no more

frontiers remain. At this point the RCU generates a 2.5D occupancy grid from the 3D occupancy grid, and the first phase ends. In the second phase the UAV drones carry the UGV drones into an open space, deploy them on the floor, and then begin exploring frontiers that were unreachable from the air. One particularly applicable use case is that of Once no more frontiers remain, the exploration phase is complete, and a final map is displayed for SAR personnel.

3.1 System Simulation

Due to the exploratory nature of our feasibility study, we determined that it would not be possible, in the allotted time of this project, to develop a full simulation of the system, create libraries for mapping, implement algorithms for exploration, and decentralize computation across the swarm. In order to create a framework for future work, we instead focused on building all of the individual components that will be required to create the full system in future iterations, so that more time can be spent working on interesting research opportunities such as decentralizing the system. However, in order to clarify the objectives of our project and a good milestone for the system, we define a first full implementation of the system in simulation. For our simulation, it is assumed that the UAVs can effectively navigate into the space without encountering any obstacles. In addition, we assume that the UAVs position (with reference to the room) is known. The UAV we consider has several other notable characteristics, as follows:

- It is able to hover and move around a 2D plane.
- It has a sensor which projects rays in a pyramid at a 45-degree angle below the plane that the robot moves along. This sensor functions by relaying the distance the ray travels before intersecting with an object. These rays have a maximum range which can be

changed prior to compilation. The goal of this sensor is to mimic how the monocular camera on the parrot AR drone is used in a real-life scenario.

- It has a sensor which will determine the relative range and bearing of a UGV, if said UGV is within the field of view of the sensor. This mimics another use of the monocular camera, to locate the UGV.

The room that the UAV explores also has the following characteristics:

- It is composed of mostly flat surfaces with additional obstacles scattered along the floor.
- It does not contain objects that would impede the UAVs motion, besides bounding surfaces (walls). This project does not focus on obstacle avoidance implementations.
- The UAV's plane of movement is not obstructed (with the exception of the walls that bound the room).

The UGV is designed to have minimal computational capabilities; the UGV is not intended to maintain a map and in simulation is functionally a mobile sensor. Due to this limitation, the UGV's movement while navigating to frontiers is aided by the UAV. The UAV maintains the current position and orientation of the UGV with respect to its local coordinate system and periodically course-corrects the UGV. The UGV has basic object-avoidance capabilities (since these are not computationally expensive) and the UAV will update the UGV with a goal vector (e.g. turn left 30 degrees and move 2 meters).

3.2 Building the UGV

For the Mechanical portion of the MQP, the work breaks down into three major phases: the initial design phase, the intermediate design phase, and the testing phase.

During the Initial Design Phase, the team started by creating a list of all of the necessary parts, generally speaking, for the UGV to run. For example, the UGV needs a PCB, motors, a motor controller, a distance sensor, batteries, wheels. From there, using the most available parts to the team, the group took the maximal measurements of the parts and constructed 3D models of each part in as little space as possible (See Figure 3).

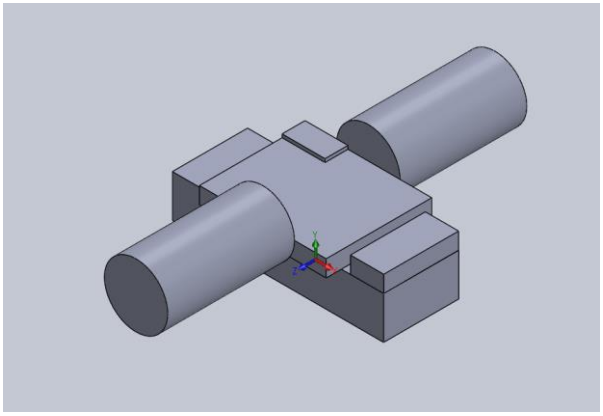

Figure 3: Internal Component Design

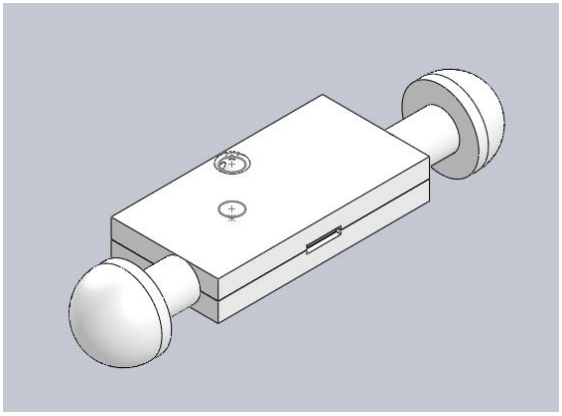

Figure 4: Initial UGV Casing Design

Then, using this internal cube design, the team designed a body to fit around the robot (See Figure 4). This allowed the team to determine the rough size of the robot, which assisted in deciding the material for the casing and acted as a basis for new designs.

The intermediate design phase began after the group decided to work on making the robot droppable and decided to use another robot for map making and SLAM. The first part of the intermediate design phase started with establishing how to add more shock absorption into the design. After a meeting with Prof. Stabile, who offered useful insight into how to add specific design features to make the robot more resilient, we brainstormed and chose several specific design features to implement in the final design.

First, the team decided on making the robot symmetrical in two planes. Since forcing the robot to fall in a specific orientation would be difficult to implement, the team decided to design the robot such that, if it were to fall, it would not matter how it fell.

The second design choice was the hemispherical wheels and a back omniwheel. This wheel design would allow the robot to be unstable on its side and thus less likely to be stuck on one of its sides. The wheels would also be far forward on the UGV so that, if it fell on front face, the robot would roll on to its top or bottom. Additionally, the back omniwheel would allow the UGV to be more maneuverable and lighter while only being slightly less stable compared to a four-wheeled design.

For the third part of the design (See Figure 5), we decided to have sectional parts: two motors sections and a main body. Both motor parts would be attached to the body section by a single shaft located in the center of the motor sections. This design choice benefitted the robot in many ways. Firstly, this allowed the motor sections to rotate with respect to the body. Additionally, the team decided to add a compressive material between the two motors such that, when the motor sections did rotate after hitting the ground, they would compress the material between them, which would act like shock absorption in a car and put less stress on the components. A second benefit to the design is that, if any of the individual components did break, we could then redesign and manufacture those individual components without having the reconstruct the whole UGV, which would also hold true for any damaged UGVs which were deployed in the field.

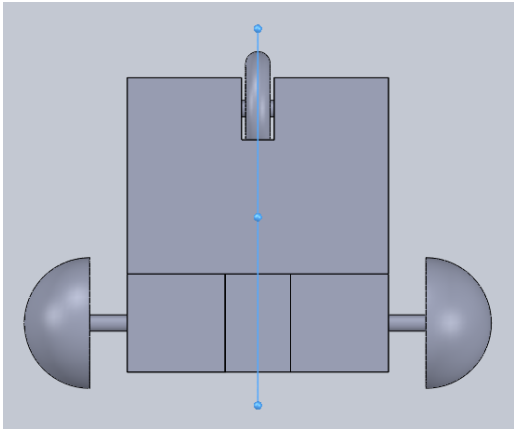

Figure 5: UGV Final Concept Top Plane Orientation

After the design phases, we opted to use rapid prototyping as the manufacturing technique, since it would keep the UGV cheap and easy to manufacture. For the first print, we opted to make the UGV entirely out of ABS because it would be much faster and cheaper than the other options, namely nylon. After the initial components finished printing, we needed to properly weigh the robot. The battery in the robot weights about 275g and each motor weights roughly 200g, however we did not want to risk damaging any of the components, so we used weight-analogous components, since we mainly wanted to test the casing to see how durable it is and how strong the bond between separate components were. For the adhesion between individual components, we used acetone to chemically weld the ABS together, since acetone melts ABS.

After assembling the robot, we performed initial drop tests. The drop tests consisted of several parts. The first variable was the drop height. As a reasonable metric, we used one meter since one floor is typically three meters and the UGV would easily survive any accidental falls from uneven surfaces if it could survive a meter drop. Next, we decided on a reasonable surface to drop onto. We chose a carpet because it is a common material in most buildings and would be a good benchmark for testing. As an analogy to carpet the team used a blanket. The team created several standard testing conditions. Several folds of the blanket would simulate a padded surface, much like the thick mats that gymnasts use to cushion their falls, two or three folds would simulate a thick carpet, and one fold would be a regular carpet. This allowed us to steadily decrease impulse and increase shock onto different orientations of the robot.

After the initial drop tests, we redesigned the UGV to fix several issues. The first fix was to increase the diameter of the shafts for the various axles. Second, we added more distance for the back omniwheel to rotate, since it would occasionally hit the body. Finally, we expanded the

space for the components. After printing, we used a dye to recolor the nylon before assembling the final UGV and performing the final tests.

In order to determine how effective the UGV design was, namely material choice, sectional design, and use of buffer material, we attached an accelerometer to the UGV and plotted the data. In conjunction with the data, we also did a visual assessment of the UGV to determine whether there was any physical damage to the printed parts or the internals.

3.3 Sensing and Electrical

The custom-made ground robot is powered by a Raspberry Pi Version 3 Model B with an RRB3 shield for controlling the two 6 volt motors. This shield also allows the Pi to be powered by the 7.2V battery pack included in the robot casing. This single-board computer has commonly been used for simple beginner's robotics projects using python and can even be equipped with ROS. For the purposes of our feasibility study, we equipped the Raspberry Pi with a standard of the shelf Ultrasonic Rangefinder and two high-voltage DC motors equipped with magnetic encoders. This simple robot is programmed with basic obstacle avoidance capabilities for demonstration, and will use a socket to relay rangefinder values to the RCU server.

The AR-Drone 2.0 is equipped with a 1GB board that runs a version of Red Hat along with a proprietary software that uses mavlinks to control messages and flight data to and from a smartphone controller. It uses a 720p HD camera and 3-axis magnetometers, gyroscopes and accelerometers for navigation and pose estimation, as well as a downward facing camera that is used to measure airspeed. This drone is a cost-effective alternative to some of the more complex and heavy-duty systems currently popular in the operation of SLAM frameworks. Additionally, with rotor guards, it is well slated for indoor use. By adding a simple 1-DOF gimbal to the neck

of the drone, we modified it for multiple angles in SLAM use. While this choice of drone is not capable of handling payloads as heavy as the prototype UGV, it remains an effective alternative for the SLAM feasibility test.

In order to test the practical benefits of our proposed framework, we extensively researched packages and libraries already implemented for monocular SLAM systems. Such a Drone-compatible package would be configured to output a point cloud to be processed by the OctoMap algorithm and merged with the point cloud output by the UGV. In 2007, the well-known publication of Parallel Tracking and Mapping (PTAM) generalized SLAM to ordinary environments around single-camera sensor suites, paving the way for systems capable of AR overlay, pose estimation, etc (Klein & Murray). Among the most popular frameworks we researched was ORM-SLAM, and its successor, ORB-SLAM2. The ORB-SLAM family of visual SLAM systems is feature-based, recognizing edges for reliable closed-loop maps. It addresses issues with the original PTAM implementation such as occlusion handling and overall scale.

3.4 Computer Vision for UGV Recognition

As part of the feasibility study, it was important to determine if we could use the UAV's camera to identify UGVs on the ground for use in SLAM and navigation. In order to do this, we decided on using a combination of a neural net for UGV identification, OpenCV, and some custom Python scripts. We began by testing out several frameworks for custom-designing a Convolutional Neural Network, or CNN. We tested Keras, pure TensorFlow, and SciKitLearn, but we determined that it was too complex to try to create a neural net architecture from scratch when we could train an open-source implementation to do this with a higher level of accuracy.

For this purpose we used a modified implementation of the Darknet framework with the YOLOv2 architecture created by Joseph Redmon, whose website can be found here[1]. We used this framework because it had a high level of associated documentation, which would assist in implementation, it was open-source, and it had a reasonable level of accuracy for our project. For training this neural net, we collated approximately 1800 images to use for training data, and tagged them by hand to create a label library. In order to increase our library size, we then used resizing functions to generate over 1000 new images for use in the neural net. We then trained the YOLOv2 architecture on our dataset using a GTX 1070 graphics card.

After training the neural net to recognize and put a bounding box around the UGV, we took the bounding box results and used them to find the x and y coordinates of the UGV. We assumed that the UGV would have a constant z, or height, of half of the UGV's total height. In order to get the x-coordinate, we obtained the x-distance between the UAV and the UGV by using the following formula:

$$\text{Distance} = (\text{Actual Width} * \text{Camera Focal Length})/\text{Pixel Width}$$

Focal length of the UAV's camera was determined to be 962.82 mm. After this, in order to get the y-coordinate of the UGV, we measured the distance from the center of the bounding box to the center axis of the image in pixels, then converted that pixel distance into feet using the following formula:

$$\text{Actual Width} = (\text{Distance} * \text{Pixel Width})/\text{Camera Focal Length}$$

From here we then began calculating the orientation of the UGV. In order to get the orientation of the UGV with respect to the UAV, we experimented with several differing approaches. The first we tried involved use of the Khepera IV and the trained neural net. We

[1] Full Link: https://pjreddie.com/darknet/

attached colored targets to the Khepera's LIDAR sensor. Unfortunately we were unable to consistently identify them due to the limitations of the AR Drone's camera, as the targets were frequently not bright enough to contrast with the Khepera. In addition, the targets were often obscured or difficult to see from the AR drone due to their small size, which made this initial approach unviable. Our second and final approach used the large hemispherical wheels on the UGV as targets to be detected by the neural net. Once they were detected, a vector was drawn between the centroids of the bounding boxes around the two wheels and the perpendicular of that vector was constructed to give the orientation of the UGV, as seen in Figure 6.

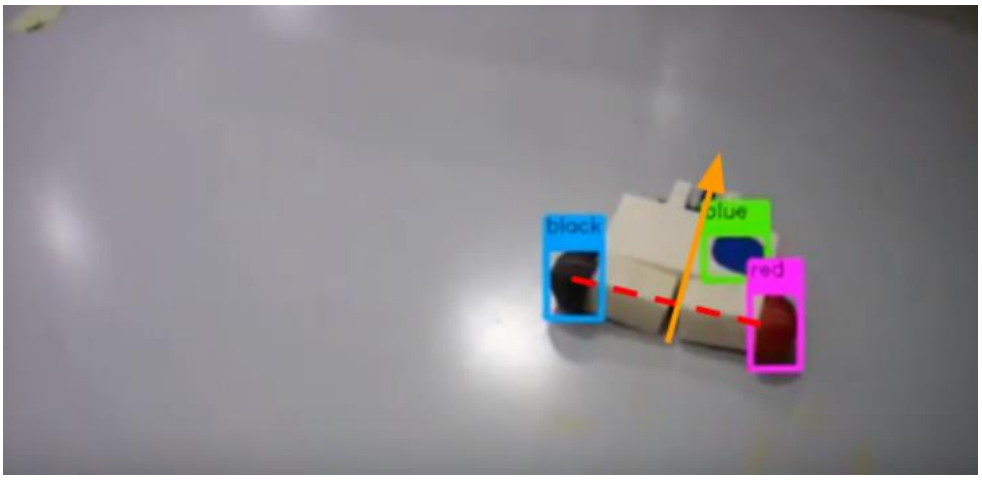

Figure 6: Visualization of Orientation Calculations

Here, the dashed line represents the line between the centroids of the black and red targets, and the orange arrow represents the orientation vector of the robot as calculated using the perpendicular bisector of the dashed line. The angle between this vector and the UAV's front was then calculated using the following code:

```
diffx = pxfront - midpoint[0]
diffy = pyfront - midpoint[1]
theta = np.tan(diffy / diffx)
```

Where `pxfront` and `pyfront` are the x and y coordinates for a point along the perpendicular of the wheel axis vector, `midpoint[0]` is the x coordinate for the midpoint of the wheel axis vector, and `midpoint[1]` is the y coordinate for the midpoint of the wheel axis vector. This approach allowed us to calculate the UGV's pose with respect to the UAV, which is integral for navigation and path planning.

4.0 Results

4.1 Computer Vision for UGV Recognition

The final iteration of the neural net was trained on a hand-tagged library of 1,800 images for 15,000 iterations. This resulted in a loss of 0.09, which is a measurement representing total error in the predictions, and a Region Average Intersection over Union (IOU) of approximately 76%. Region Average IOU was the metric used to determine accuracy of the neural net's predictions, and a value of 76% was expected given the use of three classes and a relatively small data library. A loss value of 0.09 was determined to be a result of overfitting of the model, so we used weights generated at the 9,000th iteration instead, which performed marginally better at not showing false positives or negatives upon testing. We tested the neural net on a video we gathered of the UGV using the UAV's camera, the results of which can be seen here.[2] Figure 7 shows a screenshot from the video, showing the neural net's accuracy in boxing targets on the robot.

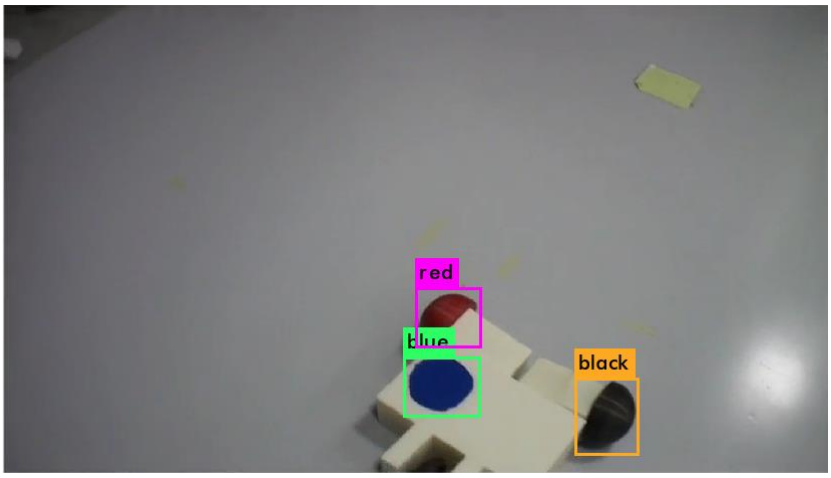

Figure 7: UGV Recognition Example 2

[2] Full Link: https://www.youtube.com/watch?v=jOwq4lS50_I

Although this system was robust when identifying the colored targets on the UGV, it did not always locate targets with the high degree of accuracy necessary for distance calculations, as seen in the following figure. This could be a result of several factors, including poor image tagging when creating the training library, overfitting of the network, or unavoidable difficulties inherent in identifying partially obscured targets. These difficulties can be seen in Figure 8. When a wheel was obscured, as with the red wheel in this image, the bounding box used for calculation was frequently distorted and did not account for the part of the wheel hidden by the rest of the robot. This introduced a percentage of error into subsequent orientation calculations.

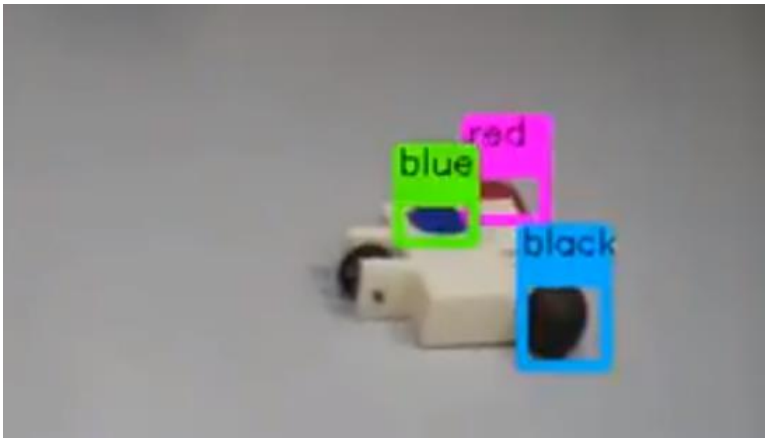

Figure 8: UGV Recognition Example 2

X, Y, and Z distance approximation worked fairly well. Approximated X-distances were usually representative of the distance from the camera to the UGV, and the y-distance approximation was also close to the actual y-offset of the UGV. These calculations could be improved through improved measurement of the camera's focal length, as well as through improvements to the precision of blue-target tagging for the neural net. The z-height was always assumed to be constant, as the robot was on flat ground, however, this will not be the case for future work. The z-distance calculations will need to be refined with a better mathematical model in order to have a more accurate approximation on uneven terrain. Overall, approximating position of the UGV using the camera output from the UAV was determined to be feasible, but

better mathematical models for position as well as generation of better data for the neural net would improve performance for this section of the project.

Orientation calculations had several issues due to aforementioned problems with the neural net. Often, the orientation calculations would produce incorrect orientation results due to the fact that one or both wheels were partially obscured. The obscuration of these targets led to incorrect determination of the wheel axis vector, which led to imprecise calculation of the perpendicular and compounded error. Some of this error could be mitigated using data smoothing to reduce impact from poorly identified targets. Despite this error, the vector mathematics and orientation code was straightforward and reliable when targets were identified well, as seen in Figure 9.

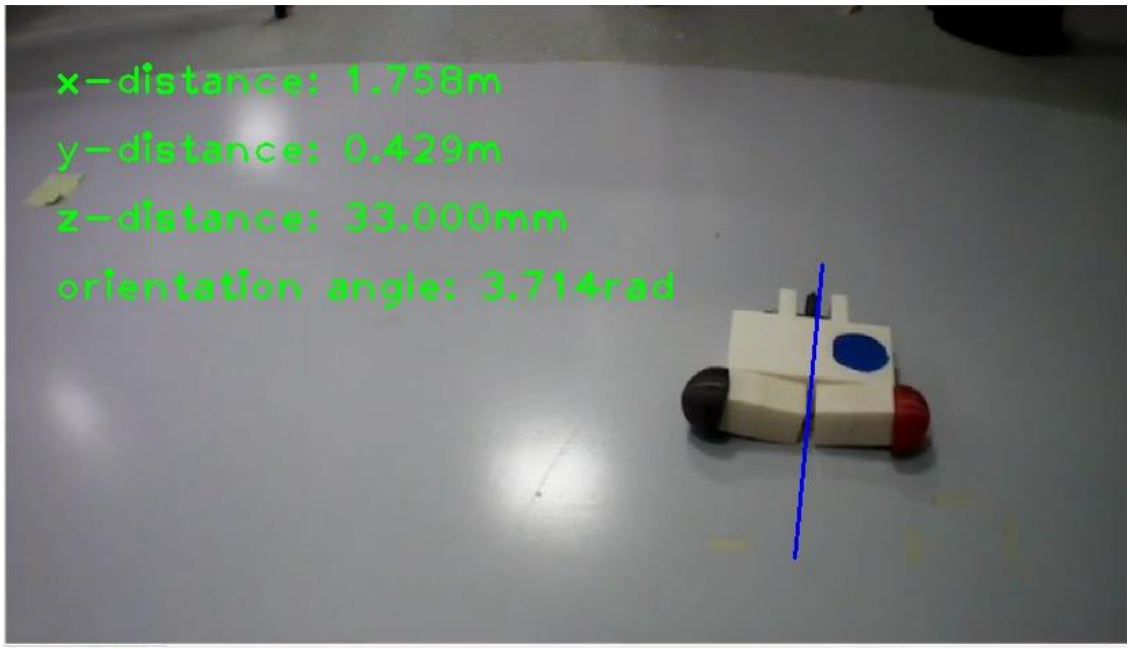

Figure 9: UGV Distance and Orientation

A video of the completed pose calculations being displayed as well as the vector showing the orientation of the robot can be seen here[3]. The code for this system can be found here.[4]

Overall, the computer vision system had several important limitations. It was too complex to run on the AR drone alone, as it required a high level of computational power to perform in real-time. Instead, the identification system had to be run on a robust remote computer. Furthermore, if the remote computer was not itself capable of high levels of computational power as could be achieved with a GTX 1070 graphics card, it ran at far less than real-time. To process one frame on a Surface Pro 3 with a quad-core Intel i7 CPU and Intel integrated graphics, the computer took 4.4 seconds, which is an unacceptable level of latency. Therefore it is necessary to increase the computational power of the drone significantly or redesign the neural network to be more efficient at the expense of accuracy. In addition, this system relied on Python, C++, and OpenCV to work, which made integration somewhat challenging. The system also needed to be retrained every time targets changed or the robot's architecture changed, which was time-consuming. To run 1000 iterations of training on a GTX 1070 took approximately 1 hour, which meant it took 6 hours to train for the recommended minimum 6,000 iterations. This could be improved using a dedicated computational cluster to train. Orientation was also somewhat imprecise, so targets need to be carefully chosen and designed to ensure accurate orientation data, and images need to be precisely tagged to reduce error in calculations.

Despite these limitations, the system did work reasonably well for determining UGV position. It ran in real-time with low latency on a commercial desktop with a GTX 1070 and did not require advanced computational clusters or parallel computing to run. It was capable of being

[3] Full Link: https://www.youtube.com/watch?v=8aXHaqqSFmY
[4] Full Link: https://github.com/lgstuehrmann/Darknet-modified

expanded and modified to account for changing mechanical design, and the innate robustness of a well-trained neural net meant that the results obtained were generally good approximations of pose. Position was straightforward to calculate using this method, and the Python scripts used to generate pose data could be easily ported onto other UAV cameras if the focal width of the camera is measured.

4.2 SLAM

The first step in running the SLAM system on the drone was to run the ARdrone-Autonomy package, a ROS package that runs on a laptop and acts as an indirect controller for the drone. The laptop connects to the drone's automatically-generated wifi network, and the user can then publish and subscribe to UAV data. LSD-SLAM Core and LSD-SLAM Viewer are other packages within the LSD-SLAM repository created by Kevin George. Typically, when starting up the program through ROS, the Core system draws a depth map generated from a raw point cloud. The Viewer allows the user to see this raw point cloud. Unfortunately, the depth map created by the Core used an older version of Qt, causing runtime errors that prevented the entire Core from working properly. We disabled the Depth map and instead relied on the Viewer to visualize our robot's viewpoint. An example is shown below (See Figure 10). Here, LSD-SLAM operates well in a medium-light environment, using intensities and direct image alignment to generate the point cloud on the right.

In practice, this point cloud would be passed through to the octomap algorithm in order to merge with the 2D point cloud generated by the UGV. The resulting output would be our finished 2.5D map of the environment for use by onsite SAR personnel.

Figure 10: UAV Point Cloud Example

4.3 UGV

The final UGV design (See Figure 11) had varying levels of success for all three benchmarks—a low-cost for manufacturing, when compared to other SAR robots, a low weight so that future UAVs could carry and deploy it, and finally durable up to a meter drop onto a carpeted surface.

The UGV reached two of its benchmark with unmitigated success. In terms of cost, the final UGV cost was under $500, with around $230 going toward the 3D printed material and under $100 for the internal components. Since typical SAR robots cost on the order of $1,000 to $100,000, we could conceivably manufacture two to two hundred UGVs for the same cost (Mahmud, 2010).

The second benchmark, weight, also succeeded quite well. At WPI, the Robotics department owns an octocopter, which has been used in several MQPs. The carrying capacity for the drone is roughly 8 lbs, which we used as our weight benchmark. The current weight of the UGV is approximately 3.4 lbs, not including the 0.75

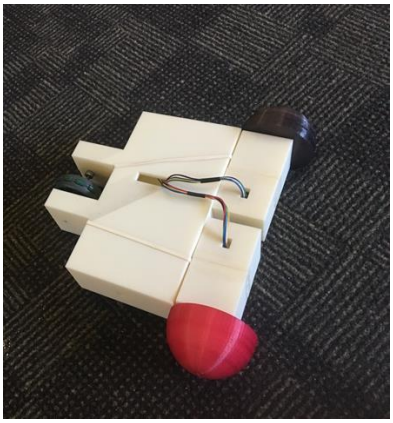

Figure 11: UGV Final Prototype

lb battery, which brings to total to around 4.1 lbs total, nearly half of the maximum weight. In terms of durability of the UGV, the team performed a series of drop tests in various orientations.

Figure 12 (below) shows the baseline result that we used for the drop test. For this baseline, the team took the accelerometer and dropped it without any buffering material. The two main important factors are the acceleration numbers and the associated slopes on the graph. From above, the accelerometer fell from t = 0 to t = 2.2, then it hit the ground and took a small bounce. Thus we have a success parameter for the system. Since the relation to voltage and acceleration is a linear transformation, the information presented by the voltage is indicative of the acceleration, and thus the information presented using voltage to see the change in acceleration is the same as the acceleration data.

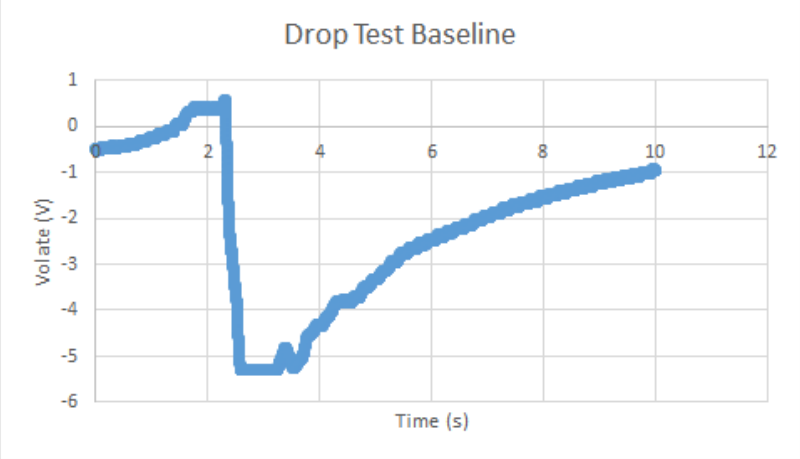

Figure 12: UGV Drop Test Baseline Graph

Figure 13 (right) has the results of the UGV dropped on our initial surface, which was heavily padded. As we can see from the graph, the UGV fell and slowed down at a much slower rate compared to the baseline test. It then took a large bounce and fell again, then slipped on the unstable

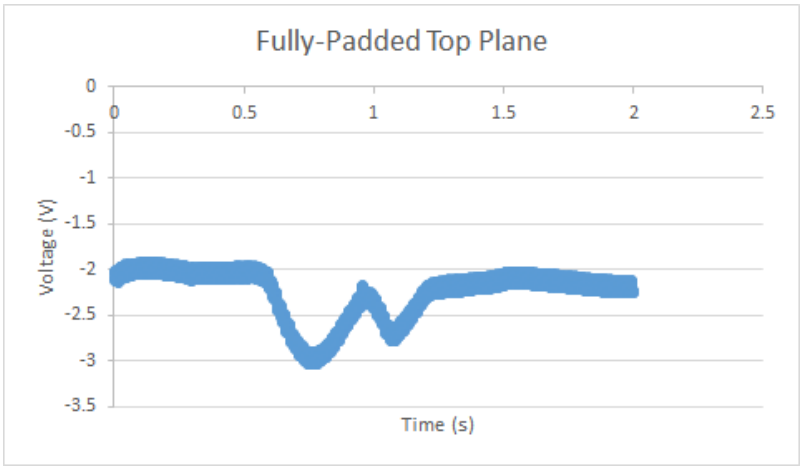

Figure 13: UGV Initial Surface Drop Test Data

surface to its resting point. Upon inspection, the UGV hadn't sustained any damage and the gentler slope on the graph indicates that the UGV took less impact from the fall over a longer period of time than the baseline. This indicates that the design effectively increased impulse and the UGV would have sustained less damage overall. This data is reaffirmed by later drop tests. As we can see from a drop test onto a thinner surface (see Figure 14), the same results occur, to a similar degree.

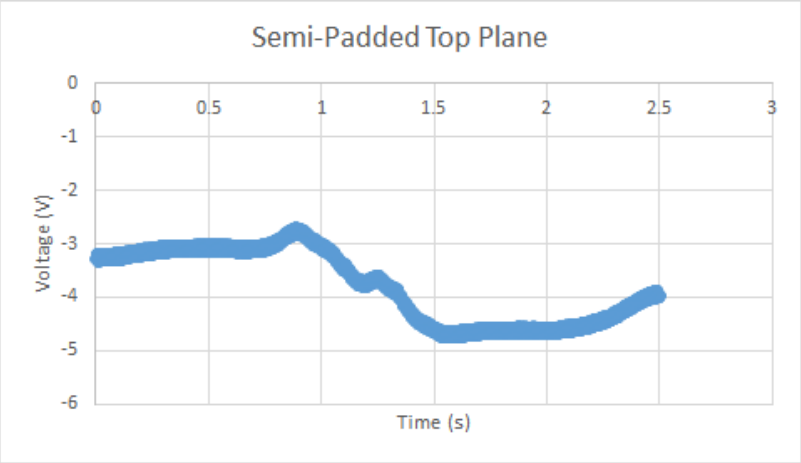

Figure 14: UGV Secondary Surface Drop Test Data

Unfortunately, the UGV and accelerometer had issues with further drop tests. After the accelerometer position was changed to accumulate data for a different drop orientation, the accelerometer stopped outputting sensible data. Then, after dropping onto thinner surfaces, the accelerometer stopped outputting any varying data at all and a wheel fell off. After the first series of 3D prints for the wheels using ABS, the shaft for the wheel fit perfectly around the axle and the motor rotated the wheel easily. However, after changing the wheel material to nylon for strength purposes, the tolerances on the nylon rapid prototyping machine were slightly different and resulted in wheels with shafts too large to have the motor turn the wheel. Due to time constraints, the team could not order the wheels to be reprinted and instead opted to use an epoxy to adhere the wheels on the motors. While this had various success, it also manage to jam one of

the motors, which had already been sealed into its casing. After performing the side face drop tests onto the thinner carpet, the epoxy adhesion failed and one wheel fell off. Coupled with the failure of the accelerometer, the team could no longer acquire accelerometer data from the drops tests. The team then proceeded with purely visual tests for the remaining drops.

After all of the drops had occurred on all orientations, the UGV had sustained no visual damage and upon inspection of the internal components, the internals were in perfect working order as well. However, since the team was unable to replace the motor and reattach the wheels, they were not able to get the robot driving, but all sensors and outputs from the Raspberry Pi indicate that everything was functioning correctly and would have driven under normal circumstances. Thus the team concluded that the UGV did succeed for the durability benchmark with the caveat that a final UGV would need a slight modification to correct for the shaft size.

4.1 Simulating the System

In order to simulate the phases of the experiment, we chose to use ARGoS (Autonomous Robots Go Swarming), an open source application that specializes in simulating highly parallelized experiments involving many robots (Pinciroli et al., 2011). ARGoS is easily extended by generating shared object files which can be registered with the simulator to create controllers, sensors, actuators, and loop functions to aid in simulation. For the purpose of our study, we chose to use a readily-available Khepera IV robot controller instead of creating the bespoke robot controller for the UGV.

During the first phase of the experiment the UAVs generate a 3D occupancy grid by exploring the previously unexplored room. Although future researchers may choose to develop an occupancy grid implementation that is more appropriate for distributed mapping, we chose to

use a well-known open-source probabilistic occupancy grid implementation called OctoMap that uses octrees to store occupancy information. OctoMap is more lightweight than other 3D occupancy grid implementations, and comes packaged with a visualization tool to display the occupancy information, an example of which can be seen in Figure 15 below (Hornung et al, 2013).

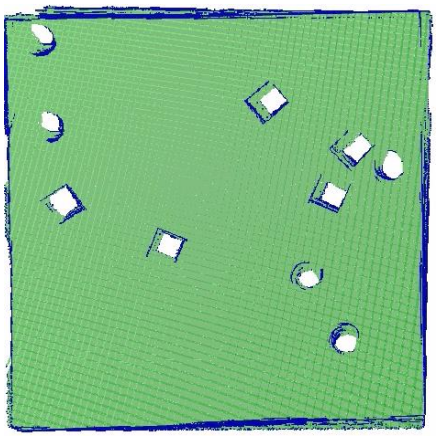

Figure 15: Octomap Example

During the second phase of the experiment the UGV must navigate the previously explored room and scan locations that the UAV could not scan from the air. The UGV will maintain a map of local scans which will be merged with other maps in order to update the full 3D occupancy grid. We chose to use an open-source grid map implementation developed by P. Fankhauser[5]. It has a suite of tools to manipulate grid maps, and comes with tools to convert from an octomap to a grid map.

In order to explore the floor-level of the room in the second phase of the experiment, we require a method to guide robots to the edge between free space and unexplored space. To do this we generate a 2D occupancy grid from the previously generated octomap, and then update the 2D occupancy grid so that all areas occluded by surfaces above the floor are marked as

[5] Full link: https://github.com/ethz-asl/grid_map

unexplored. With this new 2D occupancy grid, we can easily find all frontiers by searching for cells with neighboring unexplored cells and marking all clusters larger than the robot as frontiers. The frontiers must then be allocated to all available UGV/UAV pairs. As robotic exploration is a large and active field of study, there exist a myriad of algorithms for frontier allocation. In order to determine which was best for our use-case, we consulted a 2014 study by Faigl, which explores the effectiveness of five different frontier allocation algorithms for multi-robot exploration (Faigl, Simonin, & Charpillet, 2014). Two algorithms stuck out: Multiple Traveling Salesman Assignment (MA) and MinPos. The MA algorithms first clusters frontiers and then assigns those clusters to robots by distance. The MinPos algorithm ranks all robots for a frontier by position (not distance) and then assigns robots to frontiers based on rank. Both of these algorithms performed well, with the MA algorithm performing better across almost all conditions. However, the MinPos algorithm is computationally simpler and easier to distribute; thus we chose to use the MinPos algorithm.

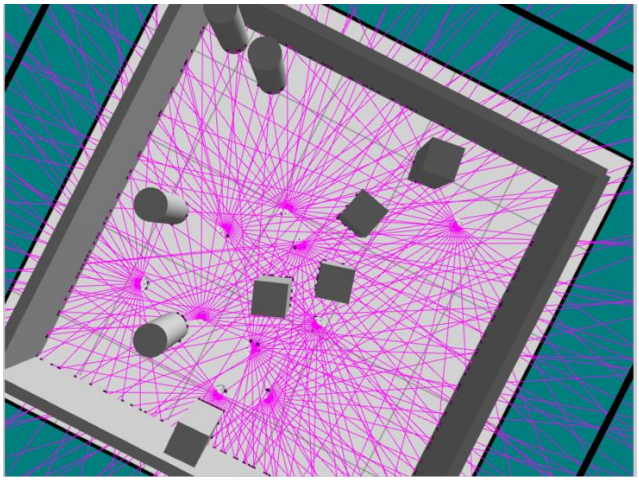

Figure 16: ARGoS Experiment

Unfortunately, due to development time constraints the system could not be built in its entirety in ARGoS. We were, however, able to create a simple experiment to demonstrate how octomap could be used in conjunction with ARGoS to map out an area (See Figure 16).[6]

[6] Future development (and additional challenges) are discussed in the conclusion.

5.0 Conclusion

Overall, the design of the UGV is a feasible way to create a swarm robot. The multi-sectioned design coupled with the buffer material drastically improved the shock absorption to the UGV. Additionally, it succeeded in its weight and cost criteria quite well. For future work, there are numerous steps to improving the final design including the materials used, the motor mechanisms, the durability, and the overall structure. As the current UGV design was intended for easily accessible, designable, and manufacturable material, the team used ABS and nylon as the materials for the final design, but plastics have many drawbacks including low heat resistance, they are easily chipped, and do not have good traction. Future works could redesign the UGV so that it can be manufactured out of a less breakable material such as aluminum or more durable plastics. A further issue with the design does not allow for easy access to its internals once the UGV is fully constructed. Creating a fixture-based design or one with more openings would allow for easier modifications to the internal components. Another issue with the design includes the motor placement. Currently the motors are directly attached to the wheels, which means that more stress is put on the motor than intended. Future students could add a gear train or treads to decrease the forces applied on the motors as well as ball bearing for decreased friction. Finally, the overall design of the current robot is only intended for use on relatively flat surfaces, which is not typical for disaster scenarios. The body could be redesigned to be more maneuverable in environments with uneven ground.

The simulated system, as previously delineated, is entirely feasible. Several outstanding challenges include creating robots and sensors in ARGoS and getting the UAV and UGV to work as a pair. Creating robots and sensors in ARGoS, while no small task, should be easy enough given a sufficient knowledge of ARGoS and linux development. Getting the UAV and

UGV to work as a pair should be very manageable, but there are quite a few ways to handle the issue of heterogeneous cooperation. For example, you could have UAVs directly manage UGVs, or UAVs could drop small beacons near frontier areas, which the UGVs would randomly search the area for. Alternatively, a many-to-one UGV to UAV structure could work, with one UAV managing the localization of multiple UGVs.

In order to make the system properly decentralized, several steps need to be taken. We can make these steps clear by hypothetically removing the RCU from the system and analyzing what changes need to be made to keep the system functional. When the RCU is removed from the system, the first problem that arises is the missing map. Once there is no centralized server to maintain the 3D and 2.5D occupancy grid, a method for maintaining a distributed map must be found. The second issue is that of updating the maps. Not only is there the issue of merging local sensor scans across a distributed system, but additionally there is the question of where the computation should take place to merge the sensor scans. Finally, the RCU can no longer perform the clustering and distribution of frontiers to UAV/UGV pairs, so there must be a distributed method to determine where the frontiers are and to allocate them to UAV/UGV pairs.

Additional future work could include creating an attachment mechanism for the UAV. Part of our research did include researching how the UAV could interact with the UGV and there are several ways of accomplishing this task, but they also have their drawbacks. Some designs we looked at include using two magnets as outlined in *The Hand-bot, a Robot Design for Simultaneous Climbing and Manipulation*, or using simple actuators and hooks (Bonani et al, 2009).

Work Cited

- Bonani, M., Magnenat, S., Rétornaz, P., & Mondada, F. (2009, December). The Hand-bot, a Robot Design for Simultaneous Climbing and Manipulation. In *International Conference on Intelligent Robotics and Applications* (pp. 11-22). Springer, Berlin, Heidelberg.

- Faigl, J., Simonin, O., & Charpillet, F. (2014). Comparison of Task-Allocation Algorithms in Frontier-Based Multi-robot Exploration. In *Multi-Agent Systems* (pp. 101–110). Springer, Cham. https://doi.org/10.1007/978-3-319-17130-2_7

- P. Fankhauser and M. Hutter, "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation", in Robot Operating System (ROS) – The Complete Reference (Volume 1), A. Koubaa (Ed.), Springer, 2016.

- Garrett, S. (2018, January 11). Are Natural Disasters Increasing? Retrieved January 26, 2018, from https://borgenproject.org/natural-disasters-increasing/

- A. Hornung,. K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees" in *Autonomous Robots*, 2013; DOI: 10.1007/s10514-012-9321-0.

- Iiba, M., Nishiyama, I., Fukuyama, H., Okawa, I., & Okuda, Y. (February 2013). *Brief Review of Building Damage by The 2011 Tohoku Japan Earthquake and Following Activities for Disaster Mitigation* (United States Japan Cooperative Program in Natural Resources, United States Japan, National Earthquake Hazard Reduction Program). Retrieved March 2, 2018.

- Mahmud, F., Hossain, S. G. M., & Bin, J. (2010). Low-Cost Rescue Robot for Disaster Management in a Developing Country: Development of a Prototype Using Locally Available Technology.

- Murphy, R. (2014). *Disaster Robotics*. The MIT Press.

- NOAA National Centers for Environmental Information (NCEI) U.S. Billion-Dollar Weather and Climate Disasters (2018). https://www.ncdc.noaa.gov/billions/

- Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., … Dorigo, M. (2011). ARGoS: A modular, multi-engine simulator for heterogeneous swarm

robotics. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 5027–5034). https://doi.org/10.1109/IROS.2011.6094829

- The United Nations. (2006, July). World Urbanization Prospects: The 2005 Revision. Retrieved March 14, 2018, from http://www.un.org/esa/population/publications/WUP2005/2005wup.htm