8-2020

Beacon Aided Robotics for Martian Cave Mapping

Ryan Joseph Capozzi

Beacon Aided Robotics for Martian Cave Mapping

by

Ryan Joseph Capozzi

A thesis submitted to the College of Engineering and Science of
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Aerospace Engineering

Melbourne, Florida
August, 2020

We the undersigned committee hereby approve the attached thesis,
"Beacon Aided Robotics for Martian Cave Mapping."
by
Ryan Joseph Capozzi

_____

Markus Wilde, Ph.D.
Associate Professor
Aerospace Physics, and Space Sciences
Major Advisor

_____

Hector Gutierrez, Ph.D.
Professor
Mechanical and Civil Engineering

_____

Brian Kish, Ph.D.
Associate Professor
Aerospace, Physics, and Space Sciences

_____

David Fleming, Ph.D.
Associate Professor and Department Head
Aerospace, Physics, and Space Sciences

Abstract

Title: “Beacon Aided Robotics for Martian Cave Mapping”

Author: Ryan Joseph Capozzi

Advisor: Markus Wilde, Ph.D.

At present, lava tubes on the Moon and Mars have been left wholly unexplored. There are many aspects of subterranean exploration that pose additional challenges over surface exploration, and one of these issues is in localization. This project seeks to expand upon the traditional communication beacon that is dropped behind a robotic operator in subterranean environments and make low cost and low power modifications to increase their functionality by making robot localization from these beacons possible. This has been tested with the use of LED beacons that could be easily added to current solutions and used for localization of the robot in areas that otherwise offer poor options for dead reckoning or odometry. This thesis presents a low-cost camera array using modern webcams and explores whether such a system can built with commercial off-the-shelf parts. The webcams are compared with 10 mm and 18 mm lenses on a Canon T6, and the system is used to calculate the global position and orientation of the array exclusively using the visual targets. Using the Robot Operating System (ROS) and OpenCV machine vision libraries, this project was able to detect the markers at ranges up to 5 m and angles of 30°. Two versions of the visual beacon system are described and tested. This research shows that while webcams still do not possess the required sensitivity and resolution for determining the color of LEDs at range, low cost DSLRs are capable of discerning not only location, but also the color of markers on such a system at range.

Table of Contents

List of Figures

List of Tables

Acknowledgement

I would like to thank my advisor Dr. Markus Wilde for his guidance and assistance throughout the production of this work.

I would also like to thank my family and friends; it is their never-ending support and love that helped me see through the stressful times and helped me to continue to push forward when things seemed impossible.

For their assistance in this thesis, I would specifically like to thank my brother and sister, for their assistance in double checking my wiring, and lending me their photography knowledge and equipment.

Finally, and significantly, I'd like to thank my parents. Their support from day one has helped me through countless challenges and taught me that hard work and good people can make all the difference.

Dedication

I would like to dedicate this thesis to my grandmother, Pat Tassia. Your support and love have been immeasurable and helped set me on the path I'm on today. I'm forever grateful for the "Discover Mars" book you bought me, one of the first space books I've ever owned and for all the articles that helped kindle my fascination with space and engineering.

Chapter 1
Background and Objectives

Objectives

The project presented in this thesis modifies visual navigation methods developed for rendezvous and docking of spacecraft to enable a robot to maintain navigation and communication within cave systems or lava tubes. By using a five-point beacon similar to those used on the International Space Station, a robot can find its position and orientation relative to a beacon defining the origin and axes of a global coordinate frame. This enables the robot to avoid odometry drift and localizes the robot in the global frame during its travel through the cave system.

In subterranean robotic exploration, systems which choose to have a base station outside the environment they are exploring use mesh networks to transmit information from the robot out of the cave. These networks can consist of many nodes and are required to maintain communication through dead zones and around features. This project seeks to augment these systems with a low cost, low draw visual target. The inclusion of such a target increases the functionality of the communication beacons by allowing robots to accurately localize without using wheel odometry or simultaneous localization and mapping.
The objectives of this project are to:

1. Test the capability of low-cost commercial off the shelf imaging for use in beacon tracking as compared against a Digital Single-Lens Reflex camera (DSLR).
2. Determine the accuracy of a camera system using off-the-shelf image processing libraries and the established QUEST relative navigation algorithm in tracking a small self-lit beacon at distances between 1 m and 7 m, at viewing angles between 0° and 30°.

Motivation

The exploration of lava tubes on the Moon and Mars has the potential to both advance scientific understanding of planets and moons as well as to serve as relatively safe environments for future explorers to build habitats protected from planetary weather, meteorites, and radiation [1]. In addition to providing protection from the elements, caves and lava tubes tend to remain at more stable temperature and may provide a good source of water ice [2]. These characteristics may permit the simplification of habitats and provide protection for external support systems. These same characteristics make caves a potential place to find evidence of extraterrestrial microbiology [2]. The slow weathering rate of caves on Mars also means that it may be possible that mineral deposits could survive tens of millions of years [2]. This would allow scientists to look back to when the planet was both warmer and wetter, more suitable for harboring life.

Due to skylights being the most common means of entry to Martian lava tubes, so that a technically challenging vertical entry is required, Martian lava tubes have been left unexplored. The future exploration of lava tubes faces numerous challenges, most critically in the areas of navigation and communication for robots. Without any knowledge of the exact characteristics of extraterrestrial lava tubes,

Earth analogues are commonly used to gain insight into what the first robots to explore these lava tubes will experience. Current Simultaneous Localization and Mapping (SLAM) technology can be confused by a lack of fine-scale features for dead-reckoning. Terrestrial lava tubes are challenging for robotic mapping systems because they may lack these dead-reckoning features, and frequently have floors covered in fine sand [3]. This fine sand makes breaking traction with the floor easier and causes low accuracy and increased drift of odometry measurements [3]. These two coupled issues make accurate mapping in lava tubes challenging for robotic systems.

Many developments in robotics for use in subterranean environments have been made with funding from mining companies, which have invested significantly into underground communication and safety systems [4]. The frequent occurrence of military operations in cave systems and tunnel systems in recent years have resulted in additional government interest. DARPA's Subterranean Challenge aims to create robotic systems that will enable warfighters and first responders to map and effectively search a variety of underground environments [5].

The problem of communicating underground has been explored by an extensive body of research, and multi-hop communication networks have been tested both in cave and mining environments [6]. This project seeks to take advantage of the communication breadcrumb strategy by augmenting dropped communication beacons with LEDs target patterns for daisy-chained navigation. The use of low power LEDs causes power requirement of each communication beacon to increase marginally, while offering far greater functionality and improving the ability of a robot to localize itself.

Background

Terrestrial Vs Martian Caves

Lava tubes can form one of two ways. The first way occurs when low viscosity lava flows close to the surface, developing a hard crust that will thicken to create a roof above the moving lava below. At the end of the eruption, the lava can drain out, leaving a cavity. The second way for lava tubes to form is by lava being injected into existing fissures in rock, or into a cavity from previous flows. In this case, lava expands and leaves a large network of connected galleries as it moves toward the surface [7]. The creation of lava tubes on both Earth and Mars operates in the same manner, with gravity having a significant effect on the size of lava tubes [7]. On Earth, lava tubes can reach nearly 30 m (100ft) across, but on Mars, due to its lower gravity, there is evidence of lava tubes that reach 250 m (820ft) in width [8]. Lava tubes on Earth are already being used by ESA astronauts for training, and their even larger size on other planets makes them strong candidates for both scientific study and astronaut housing. Figure 1 shows the training of ESA astronauts during cave training. Training future astronauts in caves exposes them to geological research they may someday complete on the Moon or Mars.

Figure 1: ESA Astronauts training inside a Lava Tube in Lanzarote during PANGEA 2016 Course. Credit ESA/L.Ricci

For all types of lava tubes, entry will occur through a variety of openings. Some will be quite large, while others will be small and blocked by debris, including boulders and rock falls. Most candidate entrances are located at the site of a collapse, so any human or robotic exploration will need to be able to move over rough terrain to gain entry to the cave itself. Many of the entrances to lava tubes we see are skylights, as shown in Figure 2 below [9].

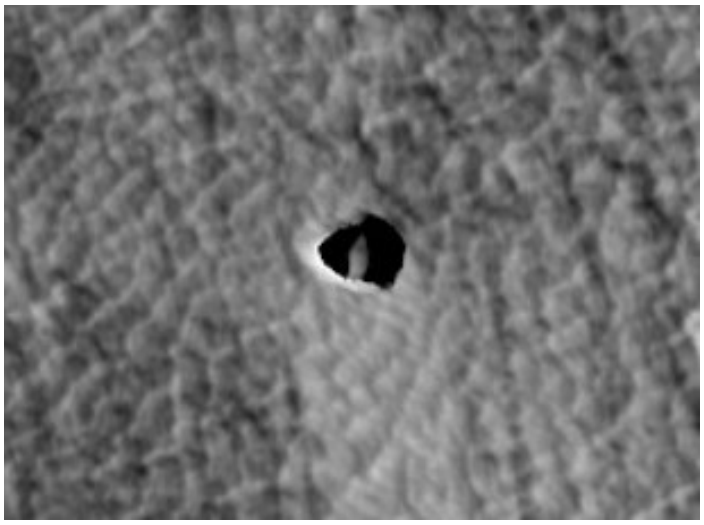

Figure 2: Skylight Entrance into Lava tube: HIRISE: ESP_041380_1775

Within the lava tubes, one must expect to find fine sand at the bottom, as this has been observed in Earth's lava tubes [3]. The terrain will be largely rough and uneven, gently sloping downhill, though flowing lava does have the ability to leave smooth floors behind. However, these are all generalizations, and in some locations on Earth, such as the blocky basaltic lava flows of the Columbia River Basin, steep slopes and terracing occurs [10].

Subterranean Mapping Robots

The development of robots capable of mapping mines and cave systems is an active field of research. Carnegie Mellon University has developed a robot that uses incremental scan matching with a Markov field to create local maps, which later is combined with lazy data association to better represent the global map. This system was tested at two mines in Pennsylvania [11]. Currently, research into subsurface communication has been largely associated with the needs of the mining

industry [4]. This, combined with the more controlled structure of mines, have allowed a body of research to grow in this area. However, there are some differences between mines and caves, largely in the regularity of distinguishing features.

For cave mapping, multiple methods have been used, ranging from teleoperation for areas with controlled lighting and direct supervision, as seen in SmartCaveDrone, to autonomous systems using multiple robots [12]. In the case of SmartCaveDrone, the goal was to provide a more efficient method of creating both 3D and 2D maps for use by researchers. It relies on receiving a qualitative map as initialization, as well as direct human oversight for determining missed branches and ending the scanning process. Another system proposed by researchers at Carnegie Mellon University specifically for mapping planetary caves uses a multi-robot framework to increase redundancy and extend mapping operations. One suggested method is the use of a "parent" robot that moves many smaller robots, known as children, to an area that they then map [13].

These multi robot approaches remain reliant on existing SLAM and odometry methods of localization, and while some relative localization can be done, for example of the children in relation to the parent robot, it still relies on the parent robot's SLAM and odometry measurements to be accurate before deploying additional robots. The inclusion of a beacon allows such systems to maintain accuracy regardless of the condition of the surface being traversed or the availability of suitable dead-reckoning points for SLAM techniques.

Caves and mines also pose challenges for communication systems. Because caves are non-uniform in density and radio absorption, it is difficult to predict signal

propagation and the communication range of a transceiver. The irregularly shaped walls only add to this challenge and can cause dead zones and other surprising effects. A group at University of Nevada was able to combat some of these challenges by using multiple frequencies and breaking up essential and nonessential communication [6]. A 5.8GHz Wi-Fi mesh network was used alongside a 915MHz mesh network. Network nodes were deployed as breadcrumbs to form a trail that could route signals to the base station outside the cave. The use of breadcrumb style systems is common in subterranean exploration and varies from simple communication spheres to fully autonomous robots that act as mobile beacons. In one example, the lead robot "pulls" two other robots, whose purpose is acting as mobile beacons, using simulated spring damper systems [14]. Experience shows that RF communications in caves face challenges such as sometimes having better connection from the lead robot directly to the ground station rather than through the physically nearer nodes of the mesh network [14].

DARPA's SubT Challenge

In August 2019, DARPA hosted the first stage of the Subterranean Challenge (SubT), one of the Grand Challenge competitions designed to further technology that will assist the U.S. military's strategic and tactical abilities [15]. To succeed in the SubT Challenge, teams must demonstrate rapidly mapping, navigating and searching three distinct underground environments. These include tunnel systems, urban underground, and cave networks. The motivation is to develop robots for use during time sensitive missions such as for combat or for disaster relief [16].

After the completion of the first challenge in the Systems Track, many teams discussed the importance of maintaining communication with the robots, as the

mines can cause dead spots and multipath effects. As a result, mesh networks were common. Because there is no pre-existing communication infrastructure in the test circuit, teams create their own by dropping additional robots or beacons to act as communication nodes. Team NCTV used pokeball-type "anchorballs" every 100 m, while team Cretsie used small tankbots to develop their communication network. [16] Communication was a major key to success, and MARBLE, another team competing in the challenge, said that over the course of a 60 minute run, these beacon networks may save 10-15 minutes, by eliminating the need for robots to exit the cave to update the ground station. [17]

European Space Agency (ESA) Cave Exploration

Similarly to NASA, the European Space Agency (ESA) has shown interest in using robots to map caves on the Moon and Mars. During the CAVES-X1 mission, the ESA tested a drone that deliberately bumps into the walls of tight sections in the cave to build a map of the system [18]. This enclosed quadcopter, designed by Flyability, was equipped with a thermal camera in conjunction with bumping into the walls to find water in the cave system that was otherwise inaccessible to researchers. ESA hopes that testing like this will help determine which technologies will be suitable for mapping of Martian lava tubes. An image of the Flyability drone in flight can be seen in Figure 3.

Figure 3: ESA Launching Flyability drone ESA Launching Flyability drone [18]

The Robot Operating System (ROS)

Due to the necessity of network communication in cave exploration, the Robot Operating System (ROS) was chosen for this project. ROS is a middleware that allows individual processes to be created as nodes. A node can be as simple or complex as is necessary to complete its task and communicates with other nodes through messages. These can be setup as either publisher/subscriber pairs or as services. A publisher sends a message over the network, where many subscribers can receive that information and operate on their own version. There is no requirement for a node to be exclusively a publisher or subscriber, and for most systems, a node will be both. In addition, due to its architecture, multiple computers can all run on the same ROS network, allowing peer to peer communication between robots and storage of data.

ROS also provides package management and currently works on the catkin platform. Each package can contain multiple nodes, libraries, and datasets. When creating a project, one or more packages will be created, and nodes will be written within the new package. Additionally, the ROS framework has extensive support and prewritten packages to assist developers in applications ranging from imaging to robot control. The ROS distribution used in this project is ROS Indigo.

OpenCV

OpenCV is an open source computer vision and machine learning library. This package provides algorithms and infrastructure to assist in computer vision applications. As such, it can be used to take video, create masks, and stitch photos, in addition it contains a significant number of computer vision algorithms for object recognition. OpenCV is a fully developed system, and is used by many corporations for facial recognition, self-driving vehicles, and robotics [19]. It caters mostly to real-time vision applications and interfaces with C++, C, Python, and Java. It works on Windows, Android, and Linux systems and can interface with ROS using ROS' vision_opencv toolbox.

Pantherbot

Florida Tech's Pantherbot is an Adept Powerbot. Initially, this robot was to have a camera array affixed to the top and driven through a closed course, localizing itself using this project's visual beacons. Therefore, its operating system, initially running Ubuntu 12.04, was updated to Ubuntu 14.04 it to be compatible with ROS Indigo. After updating the operating system, the most recent version of the ARIA package was installed, as it allows control of the robot, in conjunction with

ROSARIA by commands sent through ROS. The ROSARIA package communicates with Pantherbot using the ARIA framework through a server housed within the robot. It allows control of the robot's primary functions, including the drivetrain and sensors. Pantherbot has been equipped with a SICK lidar unit, in addition to bumpers and ultrasonic rangefinders, as well as a pan tilt forward facing camera. For this project, a webcam array has been developed to provide a full 360° viewing angle as a means of determining robot pose and orientation regardless of its current heading, as presented in Chapter 3, System Architecture. To remove obstructions from the camera view, and because the camera assembly has been designed to sit on the top plate of the robot, the robotic arm previously mounted and seen in Figure 4 has been removed.

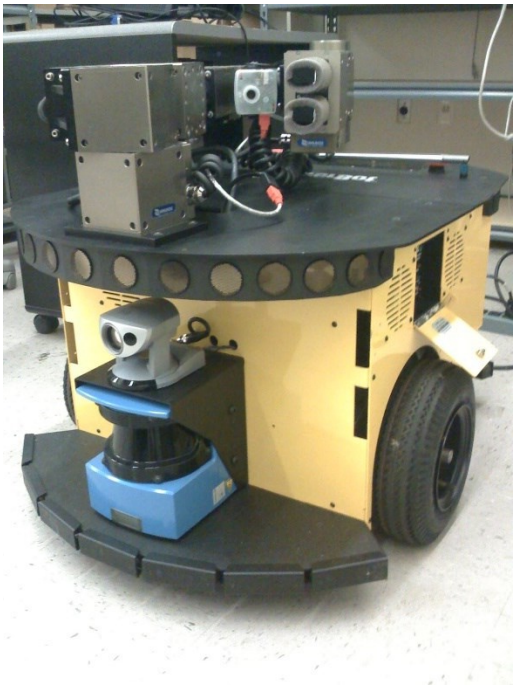

Figure 4: Pantherbot

Photogrammetry

Photogrammetry is the process of finding information about physical objects in the real world through images or video. There are many photogrammetry algorithms that are used for the determination of attitude. For this project, the QUEST algorithm was chosen as the algorithm calculating the attitude and range of the camera relative to the target due to its lower computation cost and high-speed solution [20]. Robots operating in caves will be operating for significant portions of time without access to sunlight or other means of charging, and based on this assumption, a low computation cost algorithm was chosen. The secondary advantage of the QUEST algorithm over algorithms such as the TRIAD algorithm is the lack of limitation on the upper limit of points used on a target. The TRIAD algorithm limits the beacon to three markers, two in plane and one out of plane.

Limitations

Because this project is using visual beacons and cameras, there are a few limitations. Due to the size of the visual beacon itself, there is a fixed maximum range, initially planned for 10 m. Because this is a visual system, occlusion and scattering due to dust or fog will require an increase in beacon size and intensity of LEDs used. In addition, a design limitation is that it must avoid being heavily draining on batteries for the communication system, which sets a theoretical limit on the available brightness of the design.

Chapter 2
Theory

Ad-Hoc Networks

For robotic cave exploration, the size of a cave will almost certainly exceed the transmission range of the robot-mounted antenna. Therefore, to reliably transmit a signal from the robot to the base station from any point within the cave requires an ad-hoc network protocol. An ad-hoc network is one that does not require pre-existing infrastructure to run. As a result, it must not require a modem or router. The lack of these two systems means that the network can be more easily deployed in unknown environments and is robust against failure of individual communication nodes due to the system's ability to detect and operate on new paths.

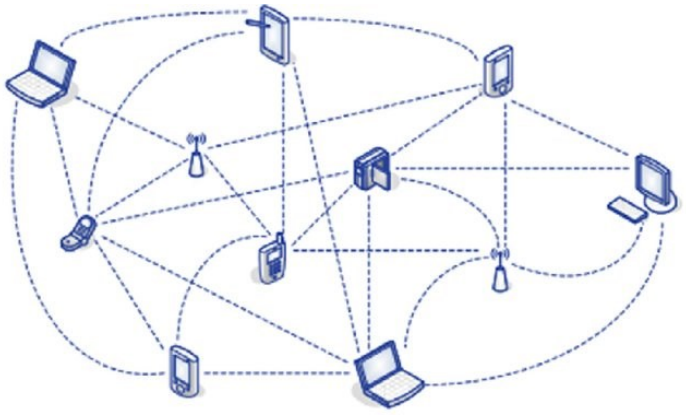

Figure 5: An example MANET network

In subterranean robotics, mesh networks are the primary method of extending communication out of a cave. DARPA's SubT challenge has many of the same challenges that a robotic system will find when mapping Martian Lava Tubes. In

this project, a Mobile Ad-hoc NETwork (MANET) system is used. Teams have reported that MANET networks are the primary way to communicate with their robots from outside the cave during a challenge run. These networks are created by dropping communication beacons as the robot progresses through the cave, and it is these beacons that this project seeks to augment. MANET networks consist of only equal peers and do not require a router to control data flow. In addition to using ad-hoc protocols, these networks are characterized as having limited battery life and bandwidth, as well as highly mobile nodes, leads to network maps such as in Figure 5. This additional characterization is important for this project, as the robot will be communicating across many different nodes while exploring a cave system, and the visual beacons designed in this project are not limited to use by a single robot at a time. While a significant body of the research published on ad-hoc networking has been accomplished using simulations, some researchers have established and tested these networks using low cost computers, primarily Raspberry Pi's [21]. In addition, because each node is a fully functioning communication device, these beacons can be equipped with expanded capabilities through sensors, allowing them to become a scientific tool themselves. As there is no theoretical limit to the size of a MANET network, this network could be expanded to allow for multiple robots and base stations to communicate simultaneously.

There many protocols designed for MANET networks, including BABEL, AODV, Optimized Link State Routing Protocol (OLSR), BATMAN, and others [21, 22]. These networks can be broken down into proactive, reactive, and hybrid types. In proactive type, or distance vector type of networking, each node collects routing information for all destinations in the network and keeps this table current through exchanging route updates with other nodes. Reactive networks only find routes when a connection is needed. This has the advantage of less network overhead, but

it has higher delay than active methods. Finally, hybrid type networks combine methods of both proactive and reactive networks. The BATMAN network used in this project is an example of a proactive network and was chosen because although active networks have an increased cost of network maintenance, the network itself has a higher throughput.

Network Implementation

For this project, a BATMAN-adv network is implemented to facilitate communication between the base station and robot. This is an extension of the BATMAN protocol, and allows all nodes to appear link-local, so that higher operating protocols won't be affected by network changes. The beacons communicate using a Raspberry Pi 3b+ running Raspbian Buster and are configured as shown in Appendix A. The base station and robot are equipped with TP-link USB wireless antennas and configured by running a .sh file on each node. Every device has been given a unique static IP-address, following the convention of 192.168.1.-/16 for nodes, and 192.168.2.-/16 for robots. This enables a Wireless Local Area Network (WLAN) to be established between all nodes, even if the robot moves out of range of the base station itself. BATMAN-adv was chosen in part because of ease of installation, as it is supported by all modern Linux kernels, and requires minimal setup. However, some studies have shown that in cases with highly mobile nodes, other network architectures may outperform the BATMAN architecture [23].

Determining Azimuth and Elevation Angles from Pixel Values

To determine the angle of the light sources on each target, OpenCV is used. The blob detection algorithm from OpenCV returns keypoints, from which y and z pixel values are extracted. These y and z coordinates are then used to calculate the azimuth (α) and elevation (β) angles with respect to the camera.

The process of determining the azimuth and elevation angles in the camera frame are included below, with the variables used for depicted in Table 1.

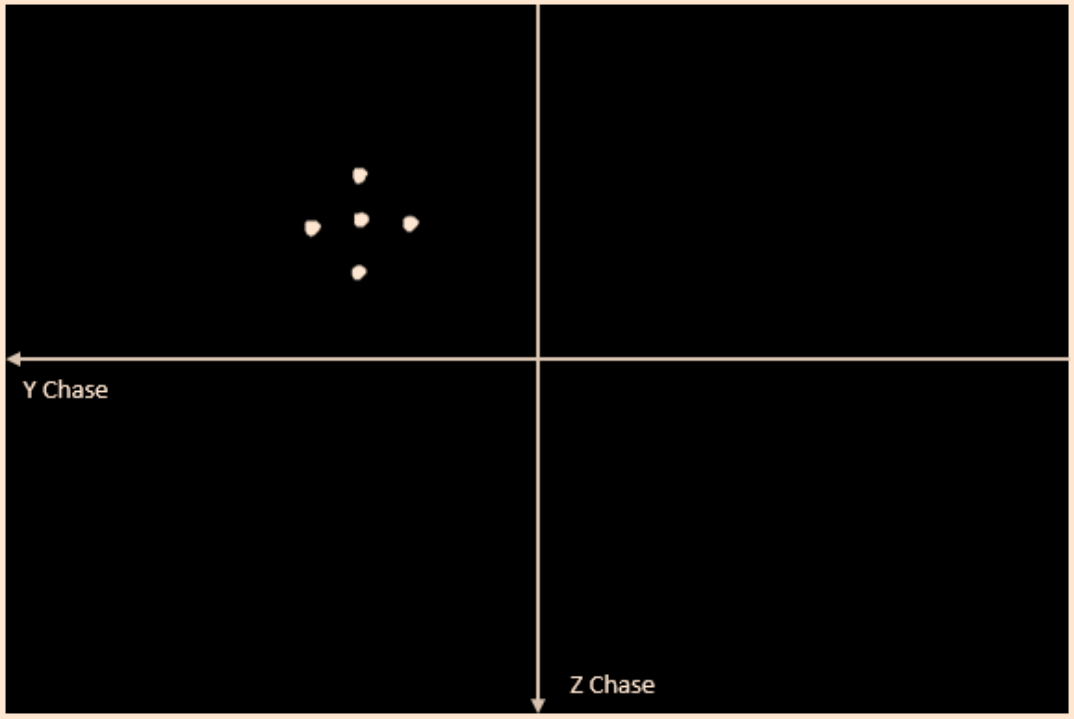

Figure 6: Image Coordinates

Table 1: Camera Nomenclature

Variable	Correlation
a	Pixel diameter
f	Focal length
N	Number of image pixels in x axis
M	Number of image pixels in y axis
ψ_{max}	Field of view in x axis
θ_{max}	Field of view in y axis
X_c	Y axis pixel coordinate measured from image center
Y_c	Z axis pixel coordinate measured from image center
X_{norm}	Normalized Y axis pixel coordinate measured from image center
Y_{norm}	Normalized Z axis pixel coordinate measured from image center

The first step in determining each marker's position relative to the camera is to determine the camera's field of view. The DSLR used in this project is a Canon T6, and many of the specifications needed for the equations below are available online. The equations for calculating the camera's field of view are as follows. [24]

$$tan\alpha_{max} = \frac{1}{2}\frac{(N\ a)}{f} \tag{1}$$

$$tan\beta_{max} = \frac{1}{2}\frac{(M\ a)}{f} \tag{2}$$

However, OpenCV's reference frame is from the top left of an image, and the angles calculated in this project are relative to the center of the camera image. To get an accurate angle relative to the camera, it is necessary to shift each point into a

central axis. The positive y axis increases as a point moves to the right across the image which must be flipped to follow the convention used later for the target. However, the z axis initially points down as intended. This behavior is what causes equations (3) and (4) to differ.

$$Y_c = X - \frac{N}{2} \tag{3}$$

$$Z_c = \frac{M}{2} - Y \tag{4}$$

These points, now in the desired frame can be normalized before being converted to the final azimuth and elevation angles.

$$Y_{norm} = Y - \frac{N}{2} \tag{5}$$

$$Z_{norm} = \frac{M}{2} - Z \tag{6}$$

$$\alpha = \frac{2Y_{norm}}{N} \tag{7}$$

$$\beta = \frac{2Z_{norm}}{M} \tag{8}$$

The QUEST Algorithm

The QUaternion ESTimator (QUEST) Algorithm is commonly used for determining the orientation of a chaser spacecraft with respect to a five-point target. Traditionally the chaser is the spacecraft actively docking, and the target is placed on the passive spacecraft. An example of this is docking with the International Space Station. The ISS docking adapter has markings that allow for visual attitude determination, similarly to this project, and the docking spacecraft is denoted as the chaser. For this project, the target is depicted in Figure 7 below, and the robot

plays the role of the chaser. Because the QUEST Algorithm calculates the orientation of the chaser in quaternions, it bypasses some of the limitations of using Euler angles, namely the singularity that occurs when the second Euler angle aligns the first and third rotation axes. The first step in calculating the position of the robot relative to the target is to find the range to the target. This is done using the Inverse Perspective Method [25, 26, 27].

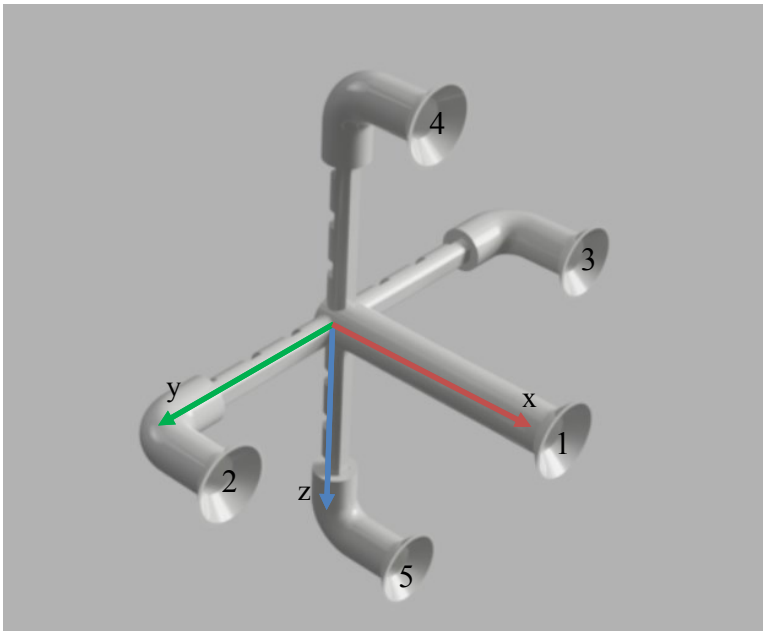

Figure 7: 5 Point visual target numbering convention

A five-point target has a set of three markers along the vertical axis, and a set of three markers along the horizontal axis. Both sets can be used in the Inverse Perspective Method to derive a range value. In the algorithm developed for this project, the range value generated is the average of the vertical and horizontal values. Following the QUEST algorithm numbering in Figure 7, the horizontal marker set is comprised of 2, 1, and 3, and the vertical set of points 4, 1, and 5. For the QUEST algorithm, the subscript *i* ranges from 1 to 5. The following equations

(9) - (13) make up the unit vector $\hat{\boldsymbol{r}}_{\boldsymbol{i}}$, the direction vector from the camera to point *i* on the target.

$$\hat{\boldsymbol{r}}_i(1) = -\cos(\alpha_i)\cos(\beta_i) \tag{9}$$

$$\hat{\boldsymbol{r}}_i(2) = -\sin(\alpha_i) \tag{10}$$

$$\hat{\boldsymbol{r}}_{\boldsymbol{i}}(3) = -\cos(\alpha_i)\sin(\beta_i) \tag{11}$$

In the above equations, α_i is the azimuth angle of the ith source from the chaser's camera along the -Y direction, and β_i is the elevation angle of each source from the camera along the – Z direction.

After finding the unit vectors to each of the targets, the dot product is used to find the cosine of the angle between each pair of vectors in the set.

$$cos\theta_{12} = \hat{\boldsymbol{r}}_{\boldsymbol{1}} \cdot \hat{\boldsymbol{r}}_{\boldsymbol{2}} \tag{12}$$

$$cos\theta_{23} = \hat{\boldsymbol{r}}_{\boldsymbol{2}} \cdot \hat{\boldsymbol{r}}_{\boldsymbol{3}} \tag{13}$$

$$cos\theta_{13} = \hat{\boldsymbol{r}}_{\boldsymbol{1}} \cdot \hat{\boldsymbol{r}}_{\boldsymbol{3}} \tag{14}$$

Given known l_{ij}, the distance from marker i to marker j in target space, the coupled non-linear equations below can be used to determine the range from the camera to each source. The range from the camera to source i is denoted by R_i, and θ_{ij} is the angle between unit vectors $\hat{\boldsymbol{r}}_{\boldsymbol{i}}$ and $\hat{\boldsymbol{r}}_{\boldsymbol{j}}$. Equations (15) through (17) below are solved together using the Newton Raphson method.

$$l_{12}^2 = R_1^2 + R_2^2 - 2R_1R_2cos\theta_{12} \tag{15}$$

$$l_{23}^2 = R_2^2 + R_3^2 - 2R_2R_3cos\theta_{23} \tag{16}$$

$$l_{13}^2 = R_1^2 + R_3^2 - 2R_1R_3cos\theta_{13} \quad (17)$$

The vectors $\boldsymbol{w_i}$ are the vectors from the lateral markers i to the center marker, expressed in the camera frame. Therefore, $\boldsymbol{w_i}$ is calculated by as in equation (18). The vector $\mathbf{v_i}$is the vector between marker 1 and i. This vector is in the target frame and is known from the geometry of the target itself. The vectors $\boldsymbol{w_i}$ and $\mathbf{v_i}$ represent the same vectors, with $\mathbf{v_i}$ representing them in the target frame and $\boldsymbol{w_i}$ representing them in the camera frame.

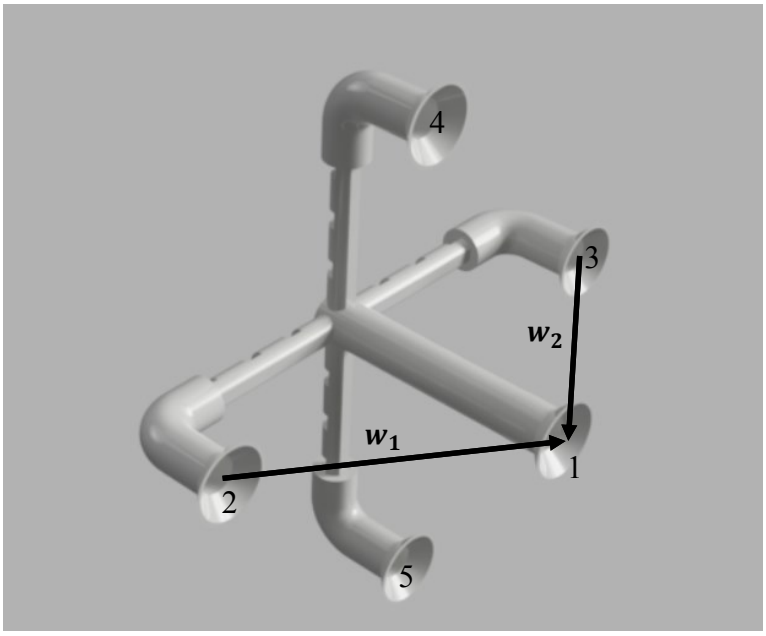

Figure 8: W_i Vectors

$$\bar{\boldsymbol{w}}_{\boldsymbol{i}} = R_1 * \hat{\boldsymbol{r}}_{\mathbf{1}} - R_{i+1} * \hat{\boldsymbol{r}}_{\boldsymbol{i+1}} \quad (18)$$

After calculating the $\boldsymbol{w_i}$ vectors, it is possible to begin solving the QUEST algorithm. Using a set of n vector measurements made in the spacecraft frame, denoted as $\boldsymbol{w_i}$ and $\mathbf{v_i}$ vector measurements made in the target frame, a least squares

estimate of the robot's attitude can be determined by finding the direction cosine matrix A which minimizes the quadratic cost function J(A).

$$J(A) = \frac{1}{2}\sum_{i=1}^{n} a_i \|\boldsymbol{w_i} - \boldsymbol{A}\mathbf{v_i}\| \tag{19}$$

The loss function can be scaled without affecting the determination of the optimal A matrix. In the above function, a_i is a positive weight assigned to each measurement. Because scaling the loss function does not affect the determination of the direction cosine matrix, a_i can be constrained by the following.

$$\sum_{i=1}^{n} a_i = 1 \tag{20}$$

Because the error corresponding to each element is unknown, and there were 4 measurements taken for each calculation of the QUEST algorithm, a_i was tuned to account for accuracy difference between horizontal and vertical measurements, as discussed in Chapter 5. The cost function $J(\boldsymbol{A})$ is next transformed to be expressed as a function $g(\boldsymbol{A})$ to be maximized, as seen in equation (21) below.

$$g(\boldsymbol{A}) = 1 - J(\boldsymbol{A}) \tag{21}$$

This is then expressed in quaternion form, denoted by μ:

$$\mu(\boldsymbol{q}) = g\big(\boldsymbol{A}(\boldsymbol{q})\big) \tag{22}$$

The goal now is to maximize the μ function. It can be shown that the quaternion that maximizes this function follows the below form:

$$\mu(\bar{\boldsymbol{q}}) = \boldsymbol{q}^T \boldsymbol{K} \boldsymbol{q} \tag{23}$$

The solution of **K** has the following parts:

$$\sigma = \sum_{i=1}^{n} a_i \boldsymbol{w}_i^T \boldsymbol{v}_i \tag{24}$$

$$S = \sum_{i=1}^{n} a_i (\boldsymbol{w}_i \boldsymbol{v}_i^T - \boldsymbol{v}_i \boldsymbol{w}_i^T) \tag{25}$$

$$\bar{\boldsymbol{z}} = \sum_{i=1}^{n} a_i (\boldsymbol{w}_i \times \boldsymbol{v}_i) \tag{26}$$

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{S} - \sigma * \boldsymbol{I}_{3x3} & \boldsymbol{z} \\ \boldsymbol{z}^T & \sigma \end{bmatrix} \tag{27}$$

Using the Gibbs vector corresponding to the quaternion can avoid using more computationally complex methods and leads to a solution accurate to the second order of the measurement error. These vectors have a singularity at 180°, and because this is outside the viewing angle of the camera, such a singularity is not a concern in this project. As noted by Schuster, λ_{max}, is close to 1, which permits the simplification of the Gibbs vector to:

$$\boldsymbol{g} = [(1 + \sigma)\boldsymbol{I}_{3x3} - \boldsymbol{S}]^{-1} \boldsymbol{z} \tag{28}$$

Which expressed as a quaternion is

$$\boldsymbol{q} = \frac{1}{\sqrt{1 + \|\boldsymbol{g}\|^2}} \begin{bmatrix} \boldsymbol{g} \\ 1 \end{bmatrix} \tag{29}$$

Converting from Local to Global Coordinates

After receiving range and attitude data of the camera with respect to a target from the QUEST algorithm, the target's LED color and the camera number are incorporated to convert the camera's attitude and position in the target frame to the robot's position in the global frame. This is achieved by using a sequence of three homogeneous transformation matrices. Each homogeneous transformation matrix has the form shown in (30).

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R}_{3x3} & \boldsymbol{l}_{1x3} \\ 0 \cdots & 1 \end{bmatrix} \tag{30}$$

Each homogeneous transformation matrix is made up of two parts, a rotation matrix **R** and a translational offset vector ***l***. The matrix **R** rotates the current coordinate system such that its axes align with the coordinate system of the next frame, and ***l*** describes the vector between the origins of the two systems in the current frame. In this project, the first matrix, $\mathbf{T}_1$, rotates and translates from the global frame into the target frame. Each visual beacon consists of 4 targets, denoted by 4 different color LEDs.

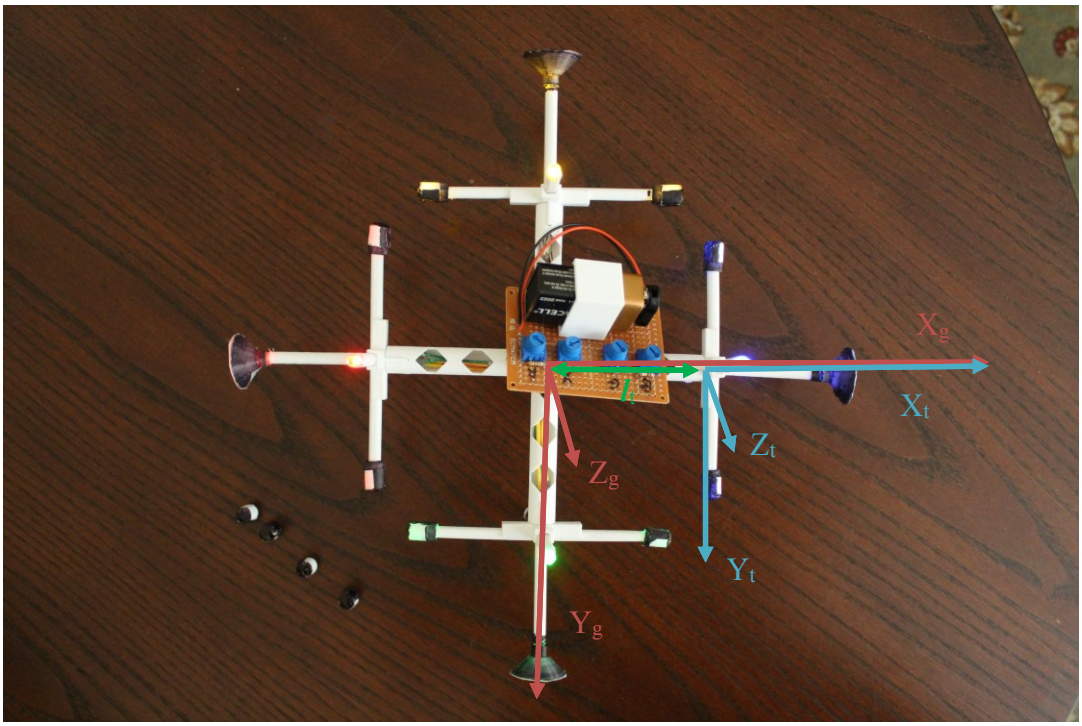

Figure 9: Target and Global Coordinate Frames

For the purposes of this project, the blue target has been defined as the X axis of the global frame. As shown in Figure 9, the beacon maintains the convention of having the Z axis point downward. Rotating clockwise, the green, red, and yellow targets are offset by $\frac{\pi}{2}$, π, and $\frac{3\pi}{2}$ radians respectively. In addition, each target has a translational offset denoted by the vector $\boldsymbol{l}_t$. The distance from the front of marker 1, the out of plane marker, of each target and the beacon is 140 mm. This distance is denoted as d_t. In calculating $\boldsymbol{l}_t$, θ is the same angle used in the rotation matrix, and is calculated as shown by equation (31). The rotation between target and beacon coordinates occurs about the Z axis, and its rotation matrix is as follows in (32).

$$\boldsymbol{l_t}(\theta) = \begin{bmatrix} d_t * \cos(\theta) \\ d_t * \sin(\theta) \\ 0 \end{bmatrix} \tag{31}$$

$$\boldsymbol{R_z}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{32}$$

$$\boldsymbol{T_1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & \mathrm{d_t} * \cos(\theta) \\ \sin(\theta) & \cos(\theta) & 0 & \mathrm{d_t} * \sin(\theta) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{33}$$

The second transformation matrix uses the calculated quaternion orientation of the camera to generate a direction cosine matrix. The calculated range from the target is also received and denoted d_c. Given that the quaternion follows the convention in (34), the direction cosine matrix can be calculated for any quaternion by using equation (35). The quaternion has two components, a vector consisting of the X, Y, and Z axis components, and a scalar. The X, Y, and Z values form the vector component, and the w component is a scalar that defines the amount of rotation about the vector part. [28]

$$\boldsymbol{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{34}$$

$$\boldsymbol{A}(\boldsymbol{q}) = \begin{bmatrix} \left(q_w^2 + q_x^2 - q_y^2 - q_z^2\right) & 2(q_x q_y + q_w q_z) & 2\left(q_x q_z - q_w q_y\right) \\ 2(q_x q_y - q_w q_z) & \left(q_w^2 - q_x^2 + q_y^2 - q_z^2\right) & 2\left(q_y q_z + q_w q_x\right) \\ 2\left(q_x q_z + q_w q_y\right) & 2\left(q_y q_z - q_w q_x\right) & \left(q_w^2 - q_x^2 - q_y^2 + q_z^2\right) \end{bmatrix} \tag{35}$$

Calculating the offset vector is more involved than in $\mathbf{T_1}$ and is completed using the following transformation. The direction cosine matrix $\mathbf{A(q)}$ transforms the vector between the target to the camera (hence along $-\hat{\boldsymbol{r}}_1$) from the camera coordinates into target coordinates.

$$\boldsymbol{l_c} = \boldsymbol{A}(q) * (-R_1 * \hat{\boldsymbol{r}}_{\boldsymbol{1}}) \tag{36}$$

This information allows for the construction of the second transformation matrix as in (37).

$$\boldsymbol{T_2} = \begin{bmatrix} \boldsymbol{A(q)} & \boldsymbol{l}_c \\ 0 \quad \cdots & 1 \end{bmatrix} \tag{37}$$

The final homogeneous transformation matrix is between the camera and the robot. It takes into account the number of cameras in the sensor, for the purposes of this project the sensor consists of 6 cameras. Following the same convention as the camera, the x axis of the robot points forward, and the z axis points down. As a result, when calculating the rotation between the camera, whose X axis points directly backward, and the robot, whose X axis faces directly forward, the angle θ must be rotated by π. By following the same convention as when calculating T_1, the calculation of the final matrix is trivial. Using a hexagonal camera array, θ increments from 0 to 2π by the following equation. The camera number, C, has a range from 0 to N-1, where N is the number of cameras in the array – for this project N is equal to 6, but can be changed as needed.

$$\theta_c = C * \frac{(N-2) * \frac{\pi}{2}}{N} + \pi \tag{38}$$

$$\boldsymbol{R}_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{39}$$

As $\boldsymbol{l_r}$ is the vector between the origin of the camera frame and the robot frame, and the cameras are assembled such that they are facing directly out from the center of the robot, $\boldsymbol{l_x}$ is only in the positive X direction. This is due to the camera's reference frame having its X axis pointing directly out the back of the camera.

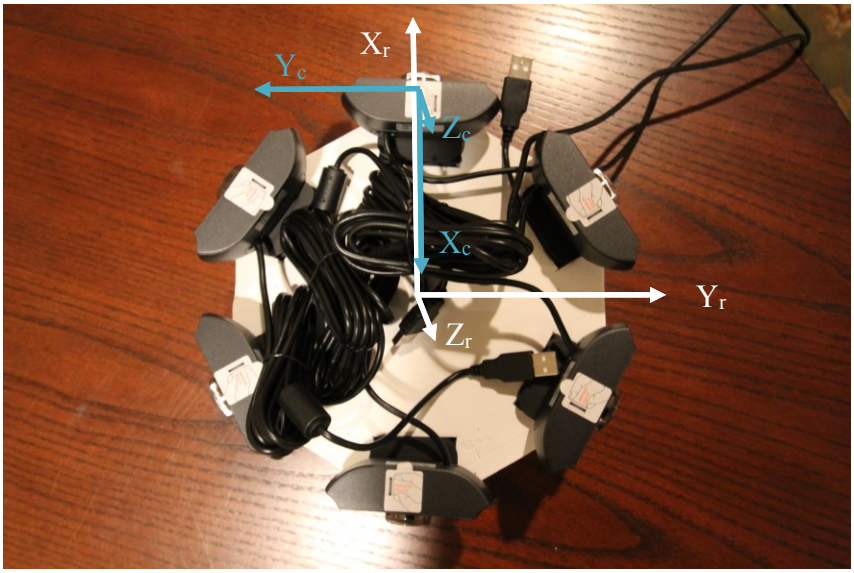

Figure 10: Camera and Robot Coordinate Frames

In this case, the distance from the front of the camera to the center of the array is 200 mm.

$$\boldsymbol{T}_3 = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & \mathrm{d}_{\mathrm{array}} \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{40}$$

It is assumed that the center of the camera array is located at the center of the robot, however, if this were not the case, a modification of the $\mathbf{T}_3$ matrix to accommodate additional transformations would be necessary.

Finally, to convert the robot's location to the global frame, the three homogeneous transformation matrices must be multiplied together. To retrieve the position of the robot, the $\boldsymbol{l}$ vector can be retrieved from $\boldsymbol{T_1^3}$, and the robot's heading can be returned using the rotation matrix. This matrix can then be converted to yaw, pitch, roll, or back into a quaternion definition.

$$\boldsymbol{T_3^1 = T_1 T_2 T_3} \tag{41}$$

Chapter 3
System Architecture

The physical system consists primarily of two main subsystems: the visual navigation targets and the camera system. Each target consists of 5 LEDs in a standard pattern, shown in Figure 16. The LEDs used in each target area single color, and 4 targets are arranged with a 90° rotation to create a full 360° beacon. The sides of the beacons are differentiated by using 4 different color LEDs. The second system is a set of webcams that are arranged such that they form a 360° field of view around the robot itself.

Visual Beacon Design

The five point visual target adapts a design commonly used for visual docking targets for spacecraft rendezvous systems. A similar system is used on the International Space Station's International Docking System Standard (IDSS). In this standard, the peripheral docking targets are located in the ring around the tunnel and are of a 4 point design depicted in Figure 11.

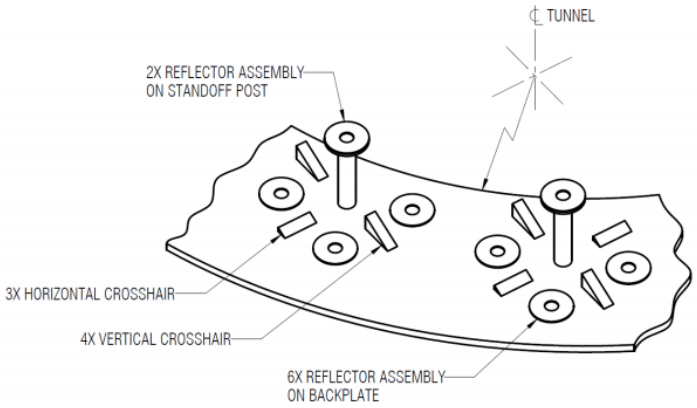

Figure 11: IDSS Peripheral Docking Target [29]

Each target is 3d printed along with a housing for each LED. Four LEDs are arranged in a plane with a fifth out of plane LED. In the V1 design, all LEDs are affixed to the end of 5 equal length 45mm arms. In the V2 version, to allow better vision at higher rotation angles, four in plane markers are equally offset from the center of the beacon, as shown in Figure 13 below. The out of plane marker is 46mm forward of the in-plane LEDs.

Figure 12: V1 Visual Target

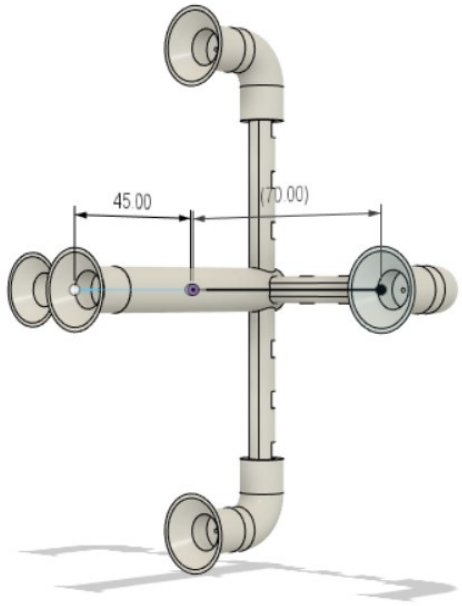

Figure 13: V2 beacon symmetry, all measurements are in mm

There were two versions tested following the same convention, one with LEDs facing parallel to their respective arms and one such that all LEDs faced out toward the camera. This was done to test the impact of the orientation of the lights as well as different housing methods. In the first version, with the lights parallel to each arm, two different housings were created, one for each of the in-plane markers and one for the out of plane marker.

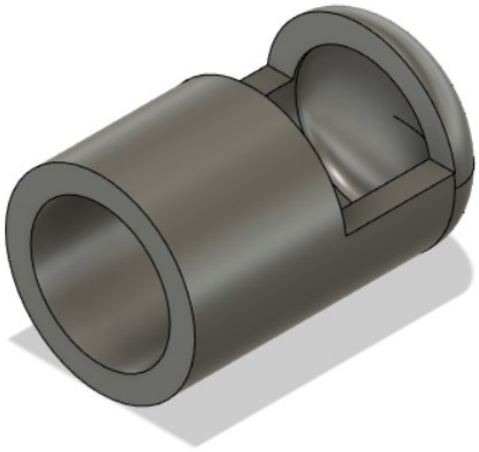

Figure 14: V1 in plane LED housing

Figure 15: V1 out of plane housing

In the second version, each LED was rotated to face the camera using an elbow attached to another standard base. An elbow was used for ease of assembly and

was friction fit to both the base and LED housing. For this design, due to the uniform orientation, the V1 out of plane housing seen in Figure 15 was modified for easier assembly as well as space for a thicker piece of foam to diffuse the light. The length of each arm on this base was increased from 45 mm per side to 60 mm, with an additional 10mm per side being added by the elbow. This was done as a result of testing the V1 housing and finding that despite being large enough for the webcam to detect, the intensity of the LEDs caused the center LED to blind the camera enough that side LEDs became nearly invisible. In addition, this second version's larger side length and marker gives higher accuracy and visual acuity at range.

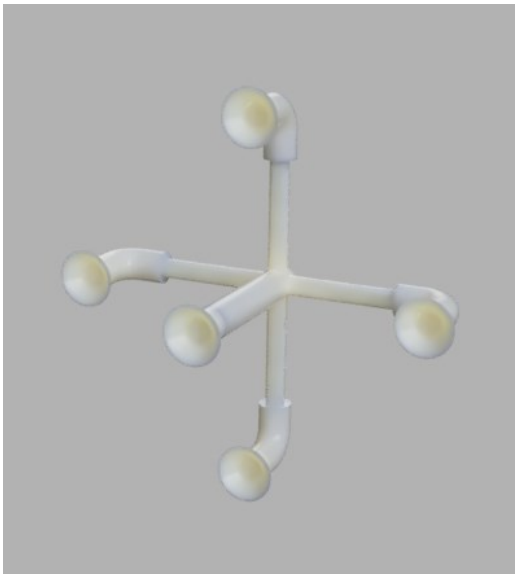

Figure 16: Assembled V2 Visual Target

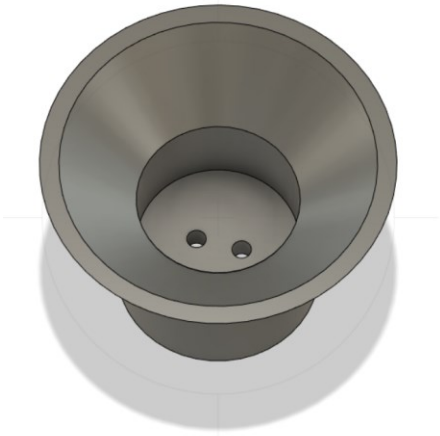

Figure 17: V2 LED housing

Camera Array

Two webcams were tested for the array and weighed against one another for cost. The Spedal 920 and the Logitech C270. Both stream in HD, however the Spedal is more expensive, but comes with a 120° diagonal field of view as opposed to the Logitech's 60° field of view. As a result a completed sensor array using the Spedal 920 webcam was more compact. Due to the geometry of the cameras a blind spot is unavoidable, as is some visual overlap. Using a hexagon, as shown in Figure 19 of the Spedal camera below, the blind spot between cameras was reduced to 507 mm from the center of the array to the overlap between visual frames. The Spedal 920's dimensions are highlighted in Figure 18.

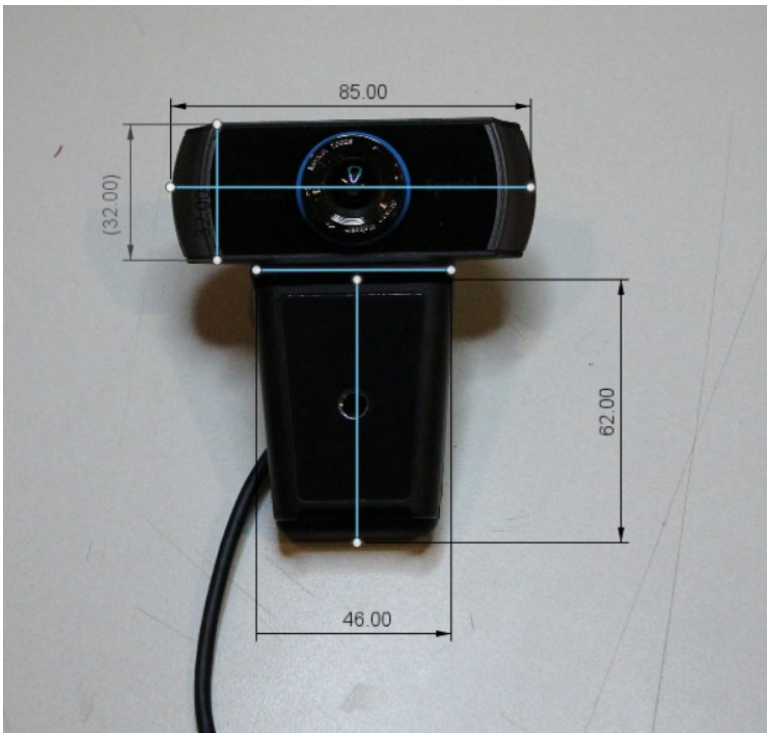

Figure 18: Spedal 920 dimensions

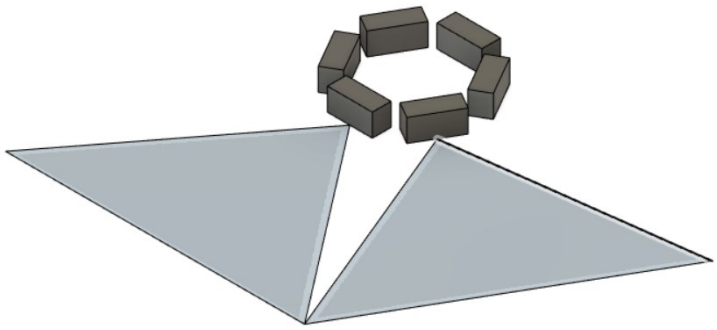

Figure 19: Spedal 920 hexagonal camera array

Due to the Logitech's smaller field of view it required nearly twice as many cameras to create a full view and to take advantage of the lower unit cost an actuation system would be required. In practice, the array was built using closed cell foam, and each camera affixed by cutting a hole in the foam just large enough to fit the tongue generally used for attaching the camera to the top of a screen.

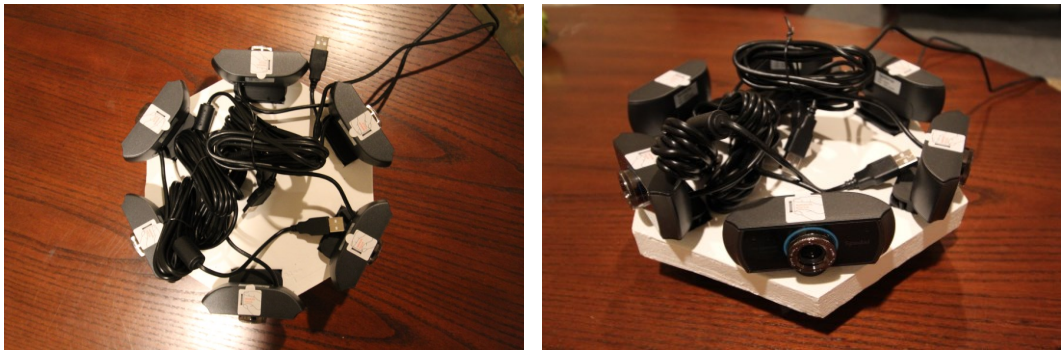

Figure 20: Completed Camera Array

Programming Architecture

Three separate ROS nodes were written for this project, each consisting of an individual program. These nodes each handle a specific aspect of localizing the robot, and are broken up as follows, one node handles the QUEST algorithm, a second runs the target tracking algorithm, and a third converts from the target's reference frame to the global reference frame. Three custom message types accompany these nodes to send pertinent information between each aspect of the system. The LEDPoints message consists of information coming from the target tracking node, including the camera's number in the array, LED positions, and LED color. The LEDPoints message follows the QUEST algorithm numbering convention, previously depicted in Figure 7. The LED color identifies the side of the beacon that the robot is currently on, and the camera number allows for post-

QUEST rotation of the direction of heading of the robot. The second message, BotPositionQuat, transmits the unit vector of the range from the camera to the center marker, R1 – the range from the center marker to the camera, and the quaternion that defines the orientation of the camera frame in the target frame. The final message, GlobalPos, defines the robot's position in the global frame. This includes the roll, pitch, and yaw of the robot and its global Cartesian coordinates.

Target Tracking Node

The target tracking node takes in the camera number and camera input, and outputs the LEDPoints message. This node follows the logic in Figure 21 determine the location of each marker as azimuth and elevation angles in the camera frame.

Figure 21: Target Tracking Node

Once implemented, the logic translates to the set of images in Figure 22.

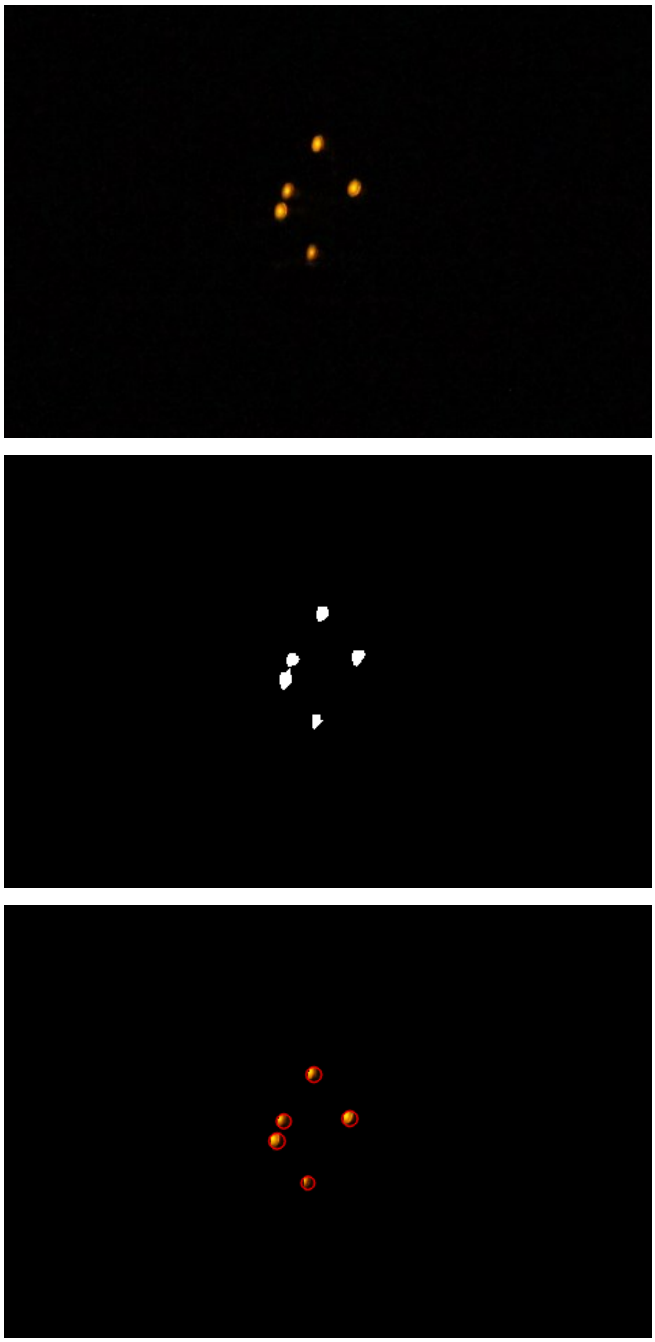

Figure 22: Image Analysis Workflow - (Top) input image, (Middle) masking desired color, (Bottom) Image with mask overlay and blob detection

Between the blob detection and message publishing, a few operations need to occur. First, the program must check if there are 5 blobs within the expected pixel range. If no target is in view, the camera is not presently facing a beacon. For example, in the red frame, there may be 2 blobs detected from background or light from that target reflecting toward the camera, but as the robot is facing the green target, 5 blobs will be visible. Because only two blobs are detected on the red frame, these will not be published to the LEDPoints message, and are ignored. Only the coordinates of the 5 markers detected in the green frame will be published.

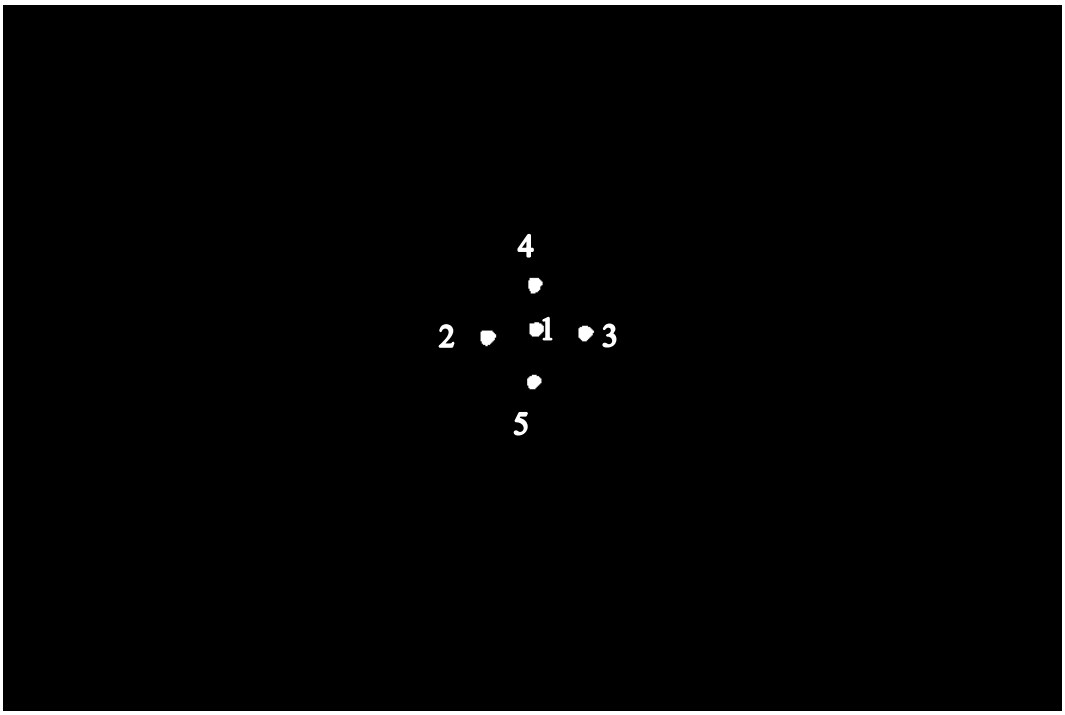

Figure 23: Filtered mask of straight on beacon, Numbered in QUEST system

After determining which side of the beacon the camera is on, the markers must be sorted to follow the QUEST Convention. Sorting occurs as follows: the marker with the greatest elevation is marker 4, and the lowest elevation is marker 5. The

three remaining markers, from least to greatest azimuth are 2,1, and 3. This is illustrated in Figure 23. The simpleblobdetect algorithm returns the keypoint type, which contains information not required for this project such as blob size and angle. Therefore, before sorting the points, x and y pixel values are extracted the returned keypoints. These pixel values are sorted, normalized, and converted using field of view and focal length to azimuth and elevation angles. Both azimuth and elevation angles have been calculated in radians to maintain a consistent unit base across ROS nodes.

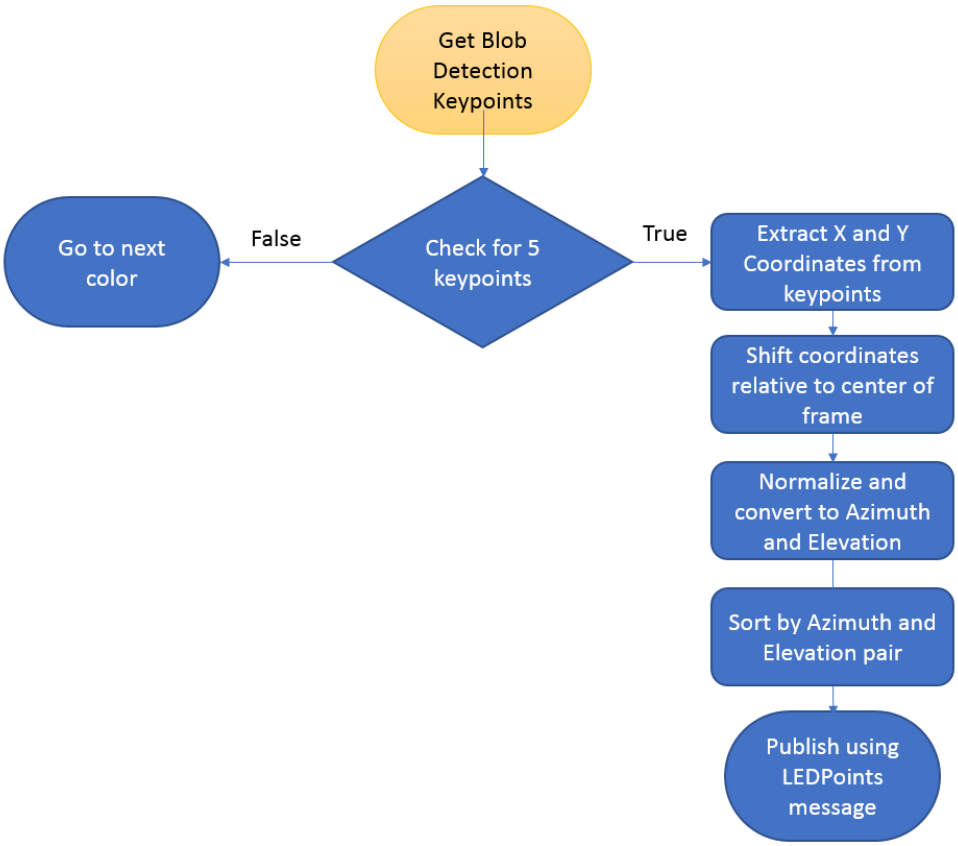

Figure 24:Logic for determining color and location of marker

Chapter 4
Experiment Setup

Preliminary testing

A number of different tests were conducted to determine the ability of a webcam to pick up on the visual beacon. The test cases were designed to be of increasing difficulty. The first test utilized a webcam in a controlled environment to detect not just the brightness of the LED but to actually sense its color. In this test the beacon was placed 150 mm from the webcam and images were taken of each color for use in determining threshold values, intensity requirements, and filtering needs. This was done in both ambient light as well as in darkness.

Figure 25: Closeup of Visual Beacon V1

The second experiment positions the webcam at 2 m, 5 m and 10 m from the target and takes images with both the lights on and off to determine whether a low-cost webcam can be used for beacons at this range. This was done twice, once for each beacon and was replicated with a DSLR camera with a 18 mm lens, 6400 ISO and

5.4 mm aperture. Shutter time was 1/60 of a second and remained unchanged between light and dark rooms.

Figure 26: (Top Left) V1 Beacon - Dark (Top Right) V2 Beacon - dark, (Bottom Left) V1 beacon bright (Bottom Right) V2 Beacon – Bright, captured with webcam

Primary Experiment

The primary experiment is conducted to determine the accuracy of the visual-beacon system at various ranges. When conducting the experiment, a reference measurement is first taken of both range from LED 1 on the target as well as the angle of the beacon with respect to the camera. Then the lights are turned off and

an image taken. Measurements were taken from 0° to 30° in 5° increments and range was varied from 1 m to 5 m in 1 m increments. This allows for an accurate representation of the accuracy of the project in various configurations. All images were taken on the V2 beacon with a Canon T6 DSLR camera with a wide-angle lens. The images were taken with the same specifications as in preliminary testing, 1/60 sec shutter speed, 6400 ISO, and 18 mm lens.

Chapter 5
Experimental Results and Data Analysis

From the tests done in initial experiment, it was quickly discovered that though using OpenCV makes it trivial to detect the brightest points in an image and similarly simple to detect the color of an object given controlled lighting, the low cost webcam has difficulty in not becoming overexposed by the LEDs, and therefore losing all color information. In addition, these webcams are incompatible with OpenCV's and other camera utilities' controls, including exposure control, and as a result, cannot be forced to underexpose the rest of the picture to retain the color of the LEDs. Therefore, tests were run with varying level of intentional occlusion of the LED, including using semi-opaque foam and colored printer paper to reduce the harshness of the LEDs themselves. The effects of covering the front of the housing with various thicknesses of paper is highlighted in Figure 27.

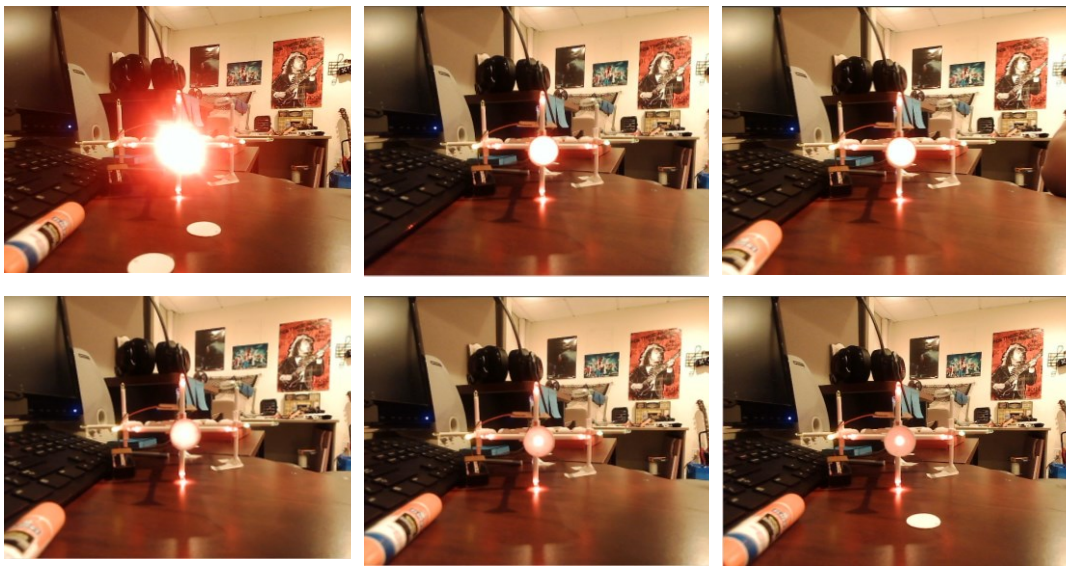

Figure 27: Varying thickness paper for occlusion – (Top Left) housing only, (Top Center) 1 sheet covering, (Top Right) 2 sheets covering, (Bottom Left) 3 sheets covering, (Bottom Center) 4 sheets covering, (Bottom Right) 5 sheets covering

Because the paper alone leaves a very bright spot over the LED itself, the semi-opaque foam was added to better disperse the light coming from the LEDs before reaching the paper. The effect of the foam is displayed in Figure 28. The foam helped reduce the sharpness of the LED, making the marker larger. These tests were run only in the V1 front housing, and lessons learned from V1 were applied to the V2 housing. With the later inclusion of a dimmer, a single piece of colored paper was used in V2 for occlusion in conjunction with the semi-opaque foam.

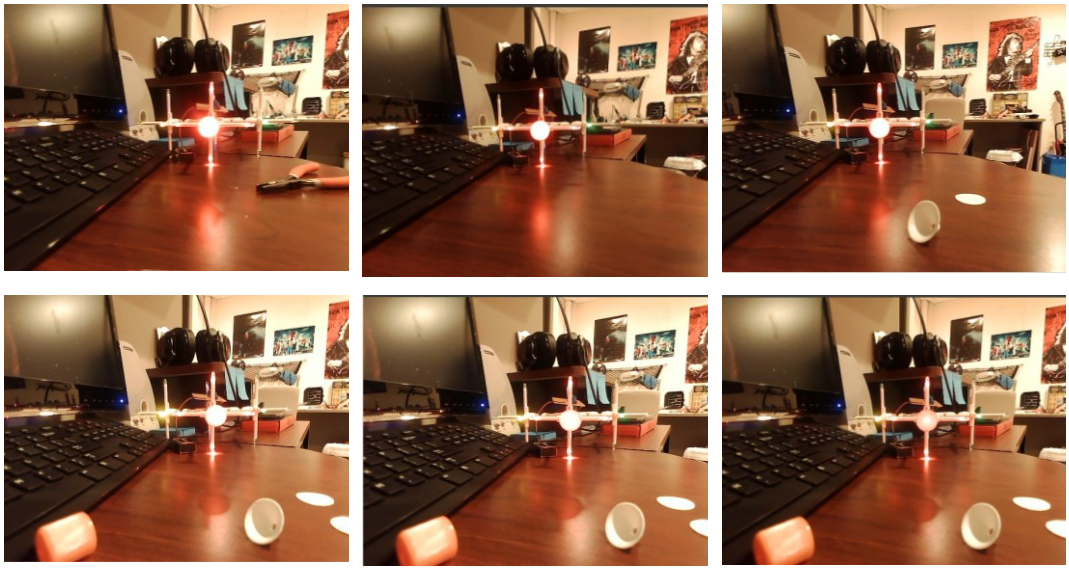

Figure 28: Foam with varying thickness paper for occlusion - (Top Left) foam only, (Top Center) foam and 1 sheet covering, (Top Right) foam and 2 sheets covering, (Bottom Left) foam and 3 sheets covering, (Bottom Center) foam and 4 sheets covering, (Bottom Right) foam and 5 sheets covering

In the second experiment, it became apparent that the V1 target was inferior to the V2 when comparing side marker visibility. At only 5 m, the webcam had difficulty seeing the beacons, and at 7 m they became indiscernible from the background. Furthermore, despite being tuned for visibility at very close range, the LEDs still became overexposed by the webcam, and determining LED color became impossible. While it was possible to determine the locations of LEDs at 2 m,

distinguishing color for the in-plane LEDs was challenging, and upon increasing the range to 5 m, the webcam could no longer distinguish between colors at all. At 7 m the webcam's resolution was too low to successfully differentiate between the different markers on the target.

Figure 29: V1 Red taken at 2 m, 5 m, and 7 m respectively, taken with webcam

When testing the V2 target in the second experiment, it became clear that the greater size and equal brightness across all LEDs in a target provides drastically increased acuity. This is the case for both the webcam and the DSLR, though the DSLR's greater sensitivity was able to pick up the V1 beacon as well. This difference becomes apparent when viewing the V2 beacon. During this test, the webcam is more than capable of detecting the target, however, it is unable to distinguish color accurately, and as distance increases, it quickly becomes unable to differentiate between the markers themselves.

Figure 30: V2 Beacon - Red. Images taken at 2 m, 5 m and 7 m respectively, taken with webcam

Camera rectification was tested with varying numbers of references images, of which an example is included in Figure 31. These image sets, taken with an 18mm lens, varied from 46 input images to 299 images. Not every image was a successful

match, and as a result the number of images interpreted in this step was somewhat lower.

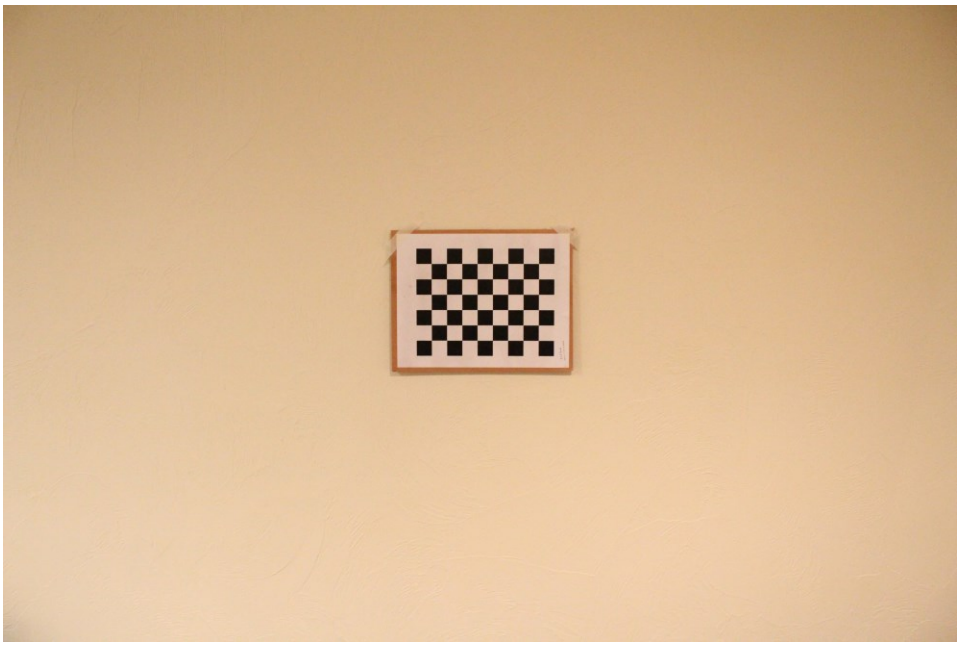

Figure 31: Example Camera Rectification Photo

The results of different camera rectification steps on an equally weighted QUEST system (a = 0.25,0.25,0.25,0.25) at 1 m can be seen in Figure 32 below. The equally weighted system causes the measured angle to be significantly less than the actual angle, and the camera rectification does little to assist this step. Setting alpha equal to 0 in the calibration function causes the removal of some pixels near the corners of the image.

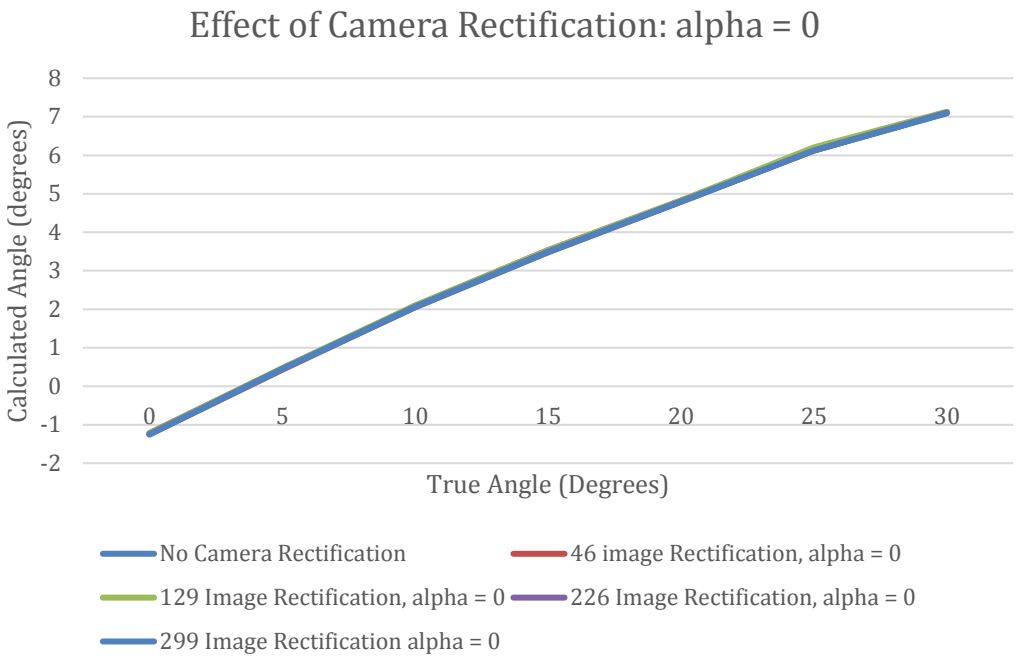

Figure 32: Effect of Camera Rectification on equally weighted system at 1 m: alpha = 0, 18mm lens

Modifying the weights of the horizontal and vertical measurements in the QUEST algorithm was significantly more successful in reducing the error between the actual and measured angle of the target. Figure 33 shows the effect of different weights without camera rectification at 1 m.

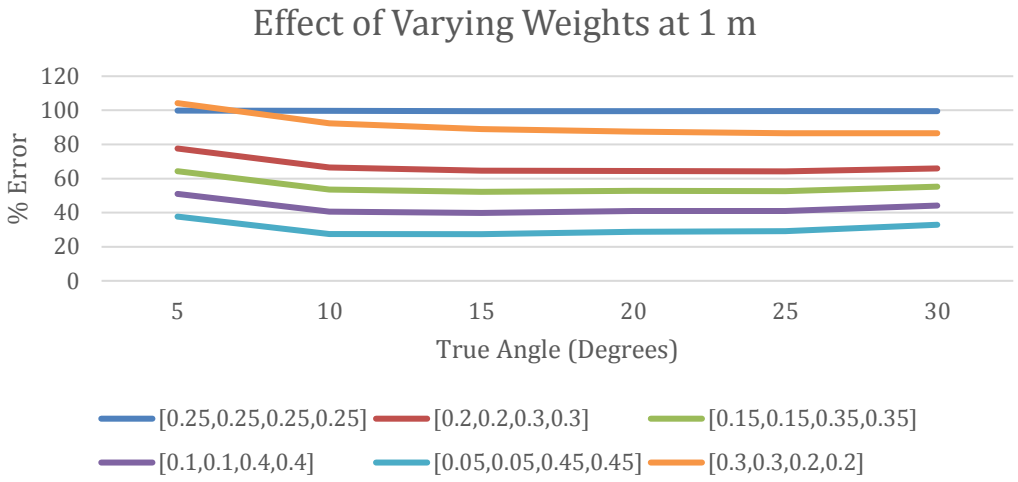

Figure 33: Effect of varying weights on error at 1 m without camera rectification, 18 mm lens

Percent error was calculated using equation (42).

$$\%Error = \frac{|True\ Angle - Measured\ Angle|}{True\ Angle} * 100 \tag{42}$$

The lowest error was achieved by lowering the weights of the horizontal measurements to 0.05, and increasing the vertical measurement weights to 0.45. After finding the weights that gave the lowest measurement error, the camera rectification was again attempted, with much better results than for the equally weighted system. Using the 226 image camera rectification with an alpha of one, which avoids the deletion of pixels near the edge of the image for rectification purposes further increased accuracy. The inclusion of the image rectification on the weighted system lowered the measured error at 1 m from 30% to 14%. Figure 34 shows the effect of adding the camera rectification from a 226 image rectification set on the accuracy of the system.

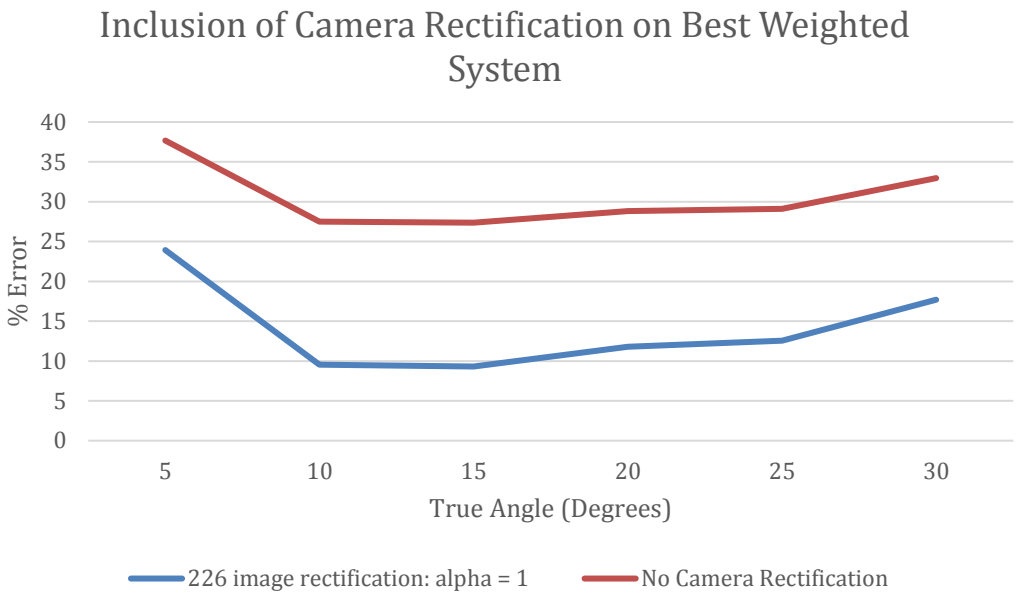

Figure 34: Camera Rectification on Best Weighted System at 1m, 18 mm lens

The error profile remains the same as what is experienced by the weighted system, with 5° true angle having a higher error than all other angles. However, the rectification on an already weighted system significantly lowers the error in the system at all tested angles.

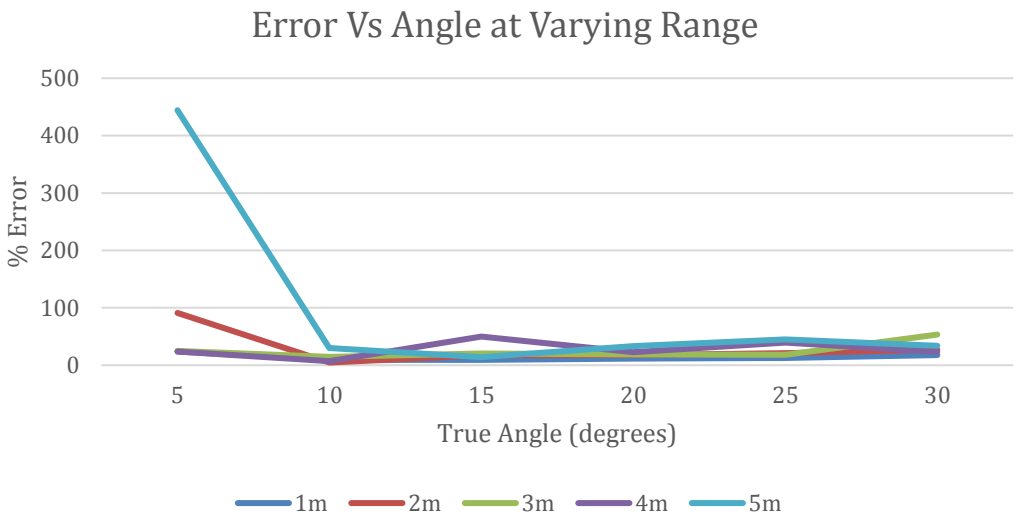

Figure 35: Error vs Angle at ranges 1m to 5 m, 18mm lens

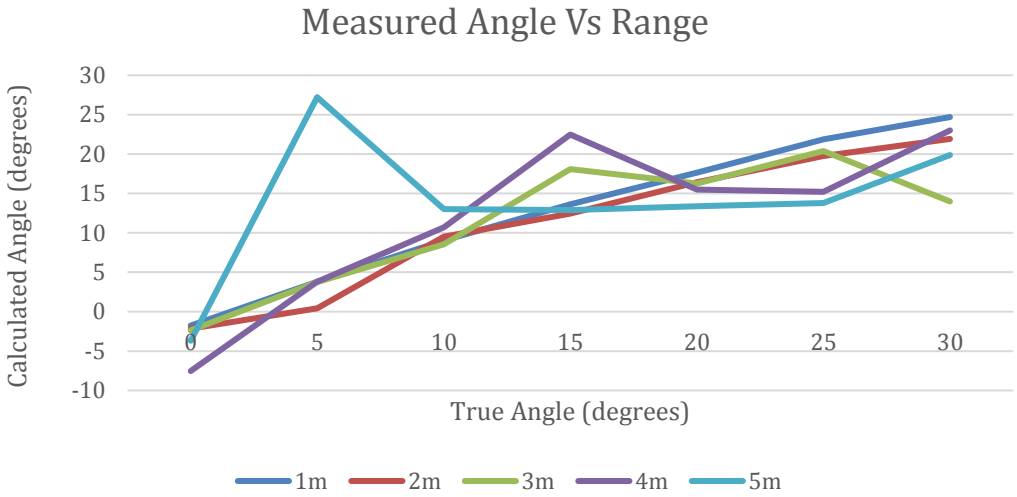

Figure 36: Measured Angle at Varying Range 18mm lens

As highlighted in Figure 35, the highest error occurred at 5°, an effect that was consistent for all ranges tested. As expected from an image-based approach, increasing the distance between the camera and target also increased the average error of the system. However, at 2 m and 5 m, the measured angle at 5°

experienced an exceptional amount of error. At 2 m, the measured angle was 0.445°, an error of 91%, and at 5 m the system measured an angle of 27°, an error of 444% off the actual value.

At 5m the system experiences large error at 5° and is largely incapable of determining angle relative to the target. The system measures an angle of between 12.88° and 13.8° for actual angles from 10° to 25°, displaying a lack of ability to find its orientation. However, range calculations at 5m average to 4.885 m, an error of only 2.3%. The Newton Raphson method used to calculate range is still accurate for this system at 5 m. Other QUEST methods such as REQUEST, a recursive QUEST algorithm that considers past elements *[26]*, or a Newton Raphson method can be used to further refine the calculated angle and further testing with these methods is needed to improve on measured angles and overcome the errors that enter the system at 5 m.

When testing the 10 mm lens, similar results were obtained when modifying the weighting factors of each measurement. As is true for the 18 mm lens case, reducing the weight of the horizontal measurements reduced the error of the system considerably. In addition, this also caused the 10 mm lens to have a flatter error curve than the 18 mm lens when tested at 1 m. The effect lowering the weights of the horizontal measurements has on the error of the system can be seen in Figure 37.

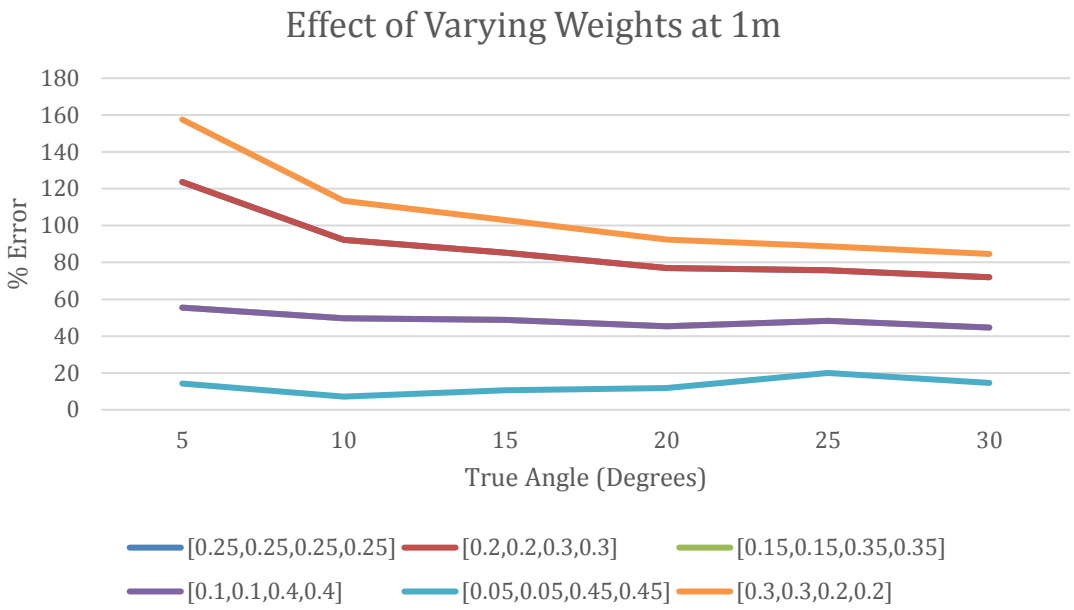

Figure 37: Effect of Varying Weights at 1 m, 10 mm lens

Though the error profile of the 10 mm lens differs slightly from the 18 mm lens, it still benefits from the same lower weighting factor for the horizontal measurements. The use of this factor in the 10 mm lens flattens the error curve at an average of 13.05% at 1 m, however this drops off more quickly as range increases, leading to multiple markers combining into a single point or having points being removed from the mesh. Point combination occurred at 4 m after turning down the erode function to receive data, and to get any data at 5 m, the close function, used to reduce background noise, was turned off. Even so, the data, though marginally better than the 18mm lens at 1 m quickly deteriorated, and at 3m the algorithm calculated its relative rotation at between 10° and 13° for all true rotation angles greater than 10°. The deterioration in estimation quality is apparent in Figure 39.

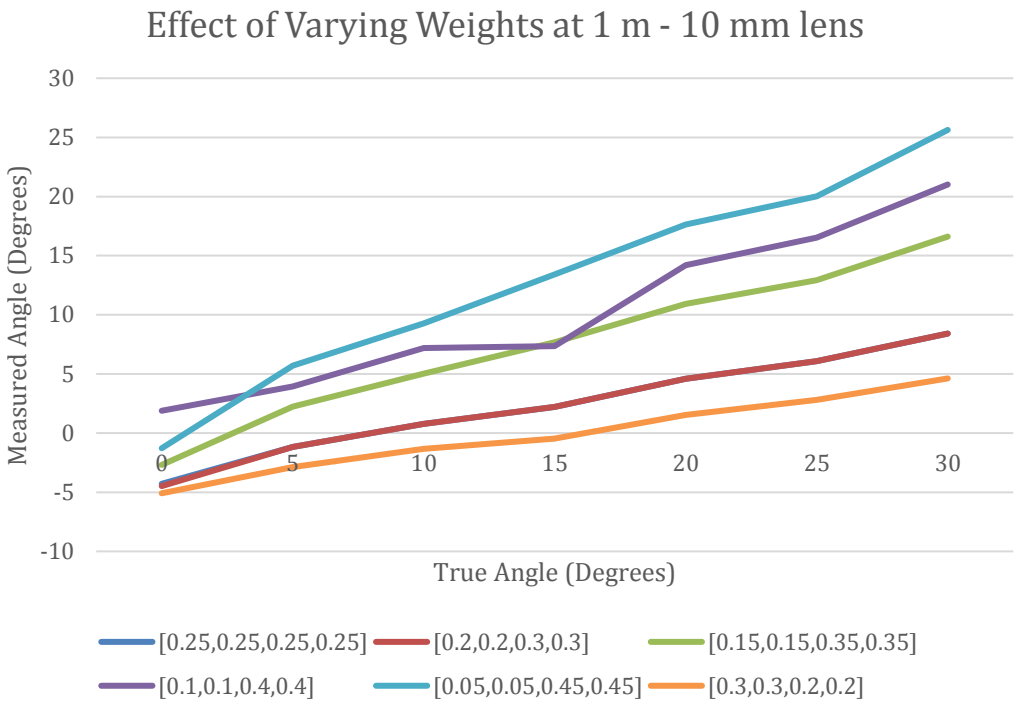

Figure 38: Effect of Varying Weights at 1 m, 10 mm lens

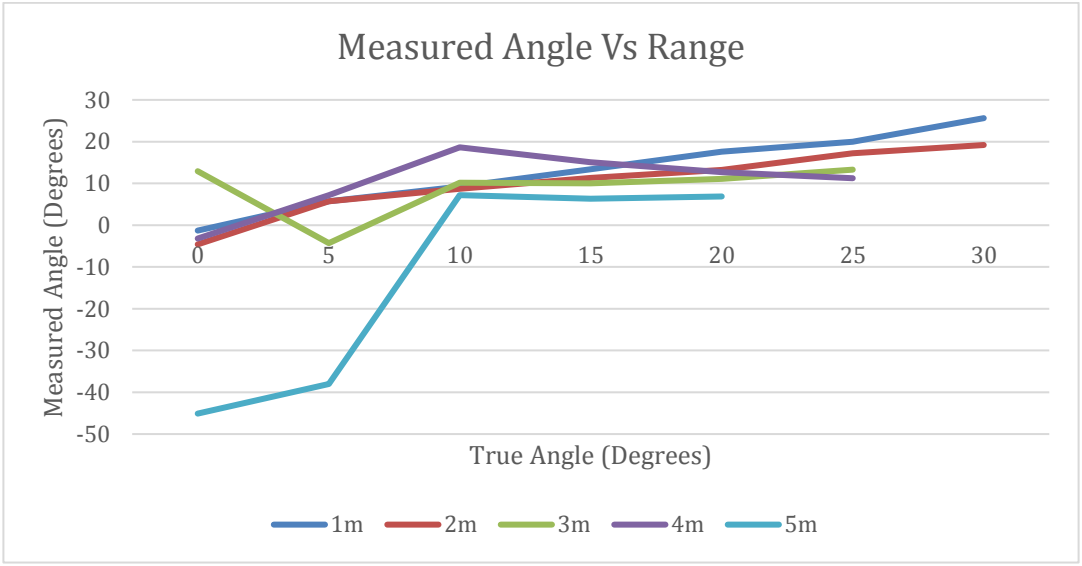

Figure 39: Measured at Varying Range 10 mm lens

Combined points occur when the created mesh merges two markers. This causes the simpleblobdetect algorithm to only detect four markers instead of five and thus causes the system to not transmit the azimuth and elevation angles for the markers. This effect occurs earlier in the 10 mm lens than the 18 mm, where the issue did not occur during testing up to 5 m. In the 10 mm lens, this began at 3 m, and an example of the combination is shown in Figure 40. This occurred at 30° in both the 3 m and 4 m image sets, and at 5 meters point loss took place in both 25° and 30° photos. The points that were removed or not detected belonged to the bottom and right markers.

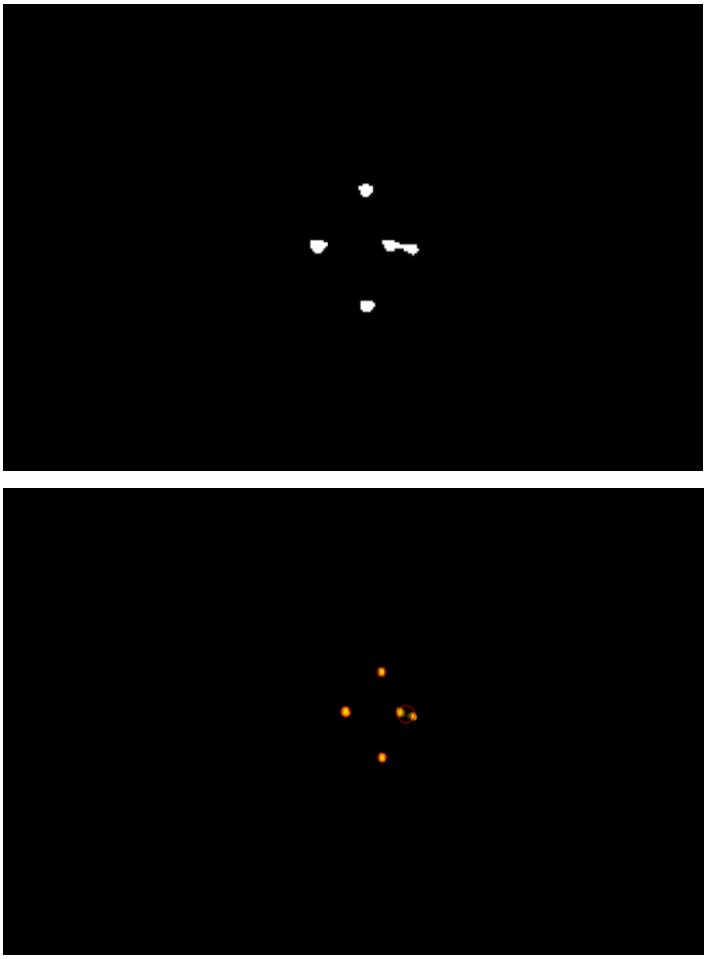

Figure 40: Point Combination in 10 mm lens

All four colors could be tracked by the beacon tracking node. Images showing the recognition of blue, green and red LEDs are below for completeness.

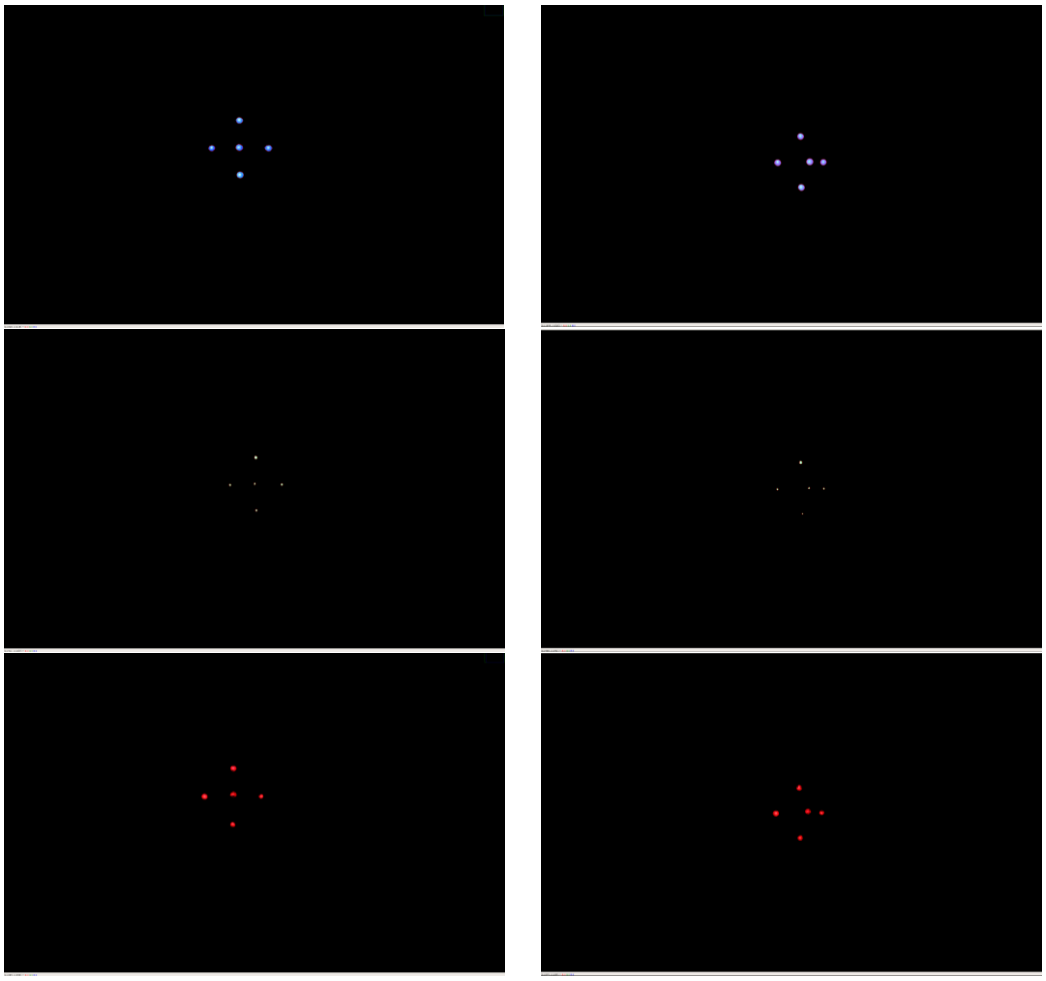

Figure 41: Detection of Blue, Green, and Red Beacons respectively at 1 m, images on the left are taken at 0°, images on the right are taken at 30°.

Communication was tested using two Raspberry Pi's runing the .sh file outlined in Appendix A. However, due to limitations in the test environment, the Pi's could not be powered far enough away to cause signal dropping in a residential environment. These tests were run in an apartment, and the signal was able to maintain a connection from one end of the apartment to the other, including traveling through a residential wall. As a result, further tests were not performed, but the test run highlights that in a line of sight situation such as is experienced in this project, the limiting factor to a combined beacon's range is the LED targets or

camera resolution. To increase the effective range of the localization system, either the side length of the target or the focal length of the lens can be increased. In one test, a Raspberry Pi lidar unit was attached and its data sent through the network to "base station" PC using ROS. This further shows that a low cost beacon can be augmented using a set of LED targets to provide both localization and communication. Increased accuracy of localization can then be achieved without removing the beacons, and instead improved through the deployment of better cameras and processing techniques.

Chapter 6
Lessons Learned

OpenCV provided many functions essential to this project, including the InRange function, used to eliminate objects of the wrong color, and simpleblobdetect, both of which were easily implemented in Python. Tuning the InRange function was difficult for the webcam, especially between the yellow and green targets due to the webcam's sensor. This process was much easier and significantly more effective when using the Canon DSLR. In addition, the 18 megapixels of the camera was larger than the available resolution of the screen, and as a result the images needed to be scaled by 50 percent to make them manageable which still caused the images to be beyond screen dimensions. Having such large images led to long calculation time for camera rectification steps, and some of the resolution was lost in scaling the images.

Despite camera rectification, the error in the horizontal measurements was significantly higher than the vertical counterparts. This persisted regardless of the number of images used in rectification and appears to be in part from the dilation and erosion of the mask. The most significant effect of this is the joining of points in Figure 40, but other images show the stretching of the blobs after the dilation and erosion steps. One tested solution was using erosion inside a for loop as long as the blob size was outside a predefined range. However, this had no effect on reducing the error of the system.

In the physical system, it was discovered that the LEDs were far too bright on their own and needed to be dimmed. Potentiometers were used for this purpose due to availability, but electronic LED drivers (buckpucks) would be more effective for

tuning the LED brightness as well as regulating current to each target. Furthermore, the LEDs used do not emit the same amount of light from their sides as through the top. This was a major fault of the V1 design, as without additional dimming on the out-of-plane LED, no amount of occlusion was able to equalize the brightness between in-plane and out-of-plane markers. The shape of housing for in-plane markers used in the V1 design also caused significant shedding of light out the top and back, which was able to be picked up by the cameras, as well as causing the target to appear to have square lights. These housings were difficult to size properly and were difficult to seat such that the in-plane markers were equidistant from the center of the target.

Chapter 7
Conclusion and Future Work

Conclusion

Though low-cost webcams have increased dramatically in resolution, and function well in well-lit environments, they are not sensitive enough for this application. Their resolution limits their effective range. In addition, though they can pick out light spots and would be sufficient for sub 1 m filtering by color, this adds little to a robot's own capabilities. As range increases beyond 1 m the sensor is easily overexposed. Also, because the webcams used had limited controllability from OpenCV and other Linux utilities, their exposure could not be manually limited. These webcams do have the potential to be used at the ranges tested in some applications if the only necessary filtering criteria is brightness.

The Canon T6 has an 18-megapixel sensor, and by using manual mode, consistent camera settings can be achieved. This sensor had no issues in capturing the markers at range, both with an 18mm and 10mm lenses. As the angle exceeded 30° there were events where the accepted values for the mask would connect, leading to the blob detect algorithm connecting two markers into a single blob. This is the limiting factor to the range, and more advanced masking and mask cleaning logic would allow a drastic increase in the effective range of the system. The maximum effective angle for this system may be increased in further iterations by increasing the offset of in-plane markers or shortening the arm on which the out-of-plane marker sits. Increasing the length of arm for the in-plane markers would have the further advantage of increasing the maximum range and decreasing the error of the system.

Because the Newton Raphson method was effective in calculating the range of the system at 5 m to approximately 2%, the system shows promise for use in longer ranges. In its current configuration, the 18mm lens still gives plenty of resolution to separate the markers at greater ranges than 5 m. However, the QUEST algorithm stumbles in calculating the attitude of the robot at this range, and further improvements are needed to increase the effective range of this project. Additionally, improvements and additional logic used in creating the mesh from which blob detect determines the location of the markers would increase maximum range, which would especially affect the 10 mm lens, and allow the 18 mm lens to be used at much greater distance. Furthermore, more precise mesh making, would expect to see some improvement in both attitude and range determination.

Future Work

Due to the use of ROS, any node can be swapped out easily for another system without modification to the whole. As a result, further testing into any subsystem of this project is possible. However, the most logical next step is in more accurately detecting the angles to each target. Though the method used above was successful in detecting and calculating the range and angle of such a target, as well as passing such information to a localization node, additional tuning to the QUEST algorithm through a Newton Raphson solver or other QUEST method would increase the attitude accuracy. An additional option would be the use of another 3-axis attitude determination algorithm, such as the Fast Optimal Attitude Matrix (FOAM) method. The limiting factor for such a system is the amount of resources available to the robot itself, and the tradeoff between accuracy, speed, and power should be considered as well. The current method of determining the global coordinates of the robot from the quaternion given by the QUEST algorithm

utilizes homogeneous transformation matrices, and in the future, this system could be extended to additional beacons by way of additional transformation matrices.

References

[1] S. Clark, "No place like lava tubes for Martian "cavenauts"," *New scientist,* vol. 206, no. 2757, p. 12, 2010.

[2] R. J. Léveillé and S. Datta, "Lava tubes and basaltic caves as astrobiological targets on Earth and Mars: A review," *Planetary and Space Science,* vol. 58, no. 4, pp. 592-598, 2010.

[3] X. Huaang, J. Yang, M. Storrie-Lombardi, G. Lyzenga and C. M. Clark, "Multi-robot Mapping of Lava Tubes," in *Field and Service Robotics*, Springer International Publishing Switzerland, 2016, pp. 471-486.

[4] M. D. Bedford and G. A. Kennedy, "Modeling Natural Microwave Propagation in Natural Caves Passages," *IEEE Transactions on Antennas and Propagation,* vol. 62, no. 12, pp. 6463-6471, 2014.

[5] DARPA, "DARPA Seeks Tools to Capture Underground Worlds in 3D," DARPA, 7 March 2019. [Online]. Available: https://www.darpa.mil/news-events/2019-03-07. [Accessed 4 May 2020].

[6] F. Mascarich, H. Nguyen, T. Dang, S. Khattak, C. Papachristos and K. Alexis, "A Self-Deployed Multi-Channel Wireless Communications System for Subterranean Robots," in *IEEE Aerospace conference 2020*, Big Sky, Montana, USA, 2020.

[7] A. Heward, "Lava tubes as hidden sites for future human habitats on the Moon and Mars," 29 September 2017. [Online]. Available: phys.org.

[8] R. Pozzobon, "Lava tubes: the hidden sites for future human habitats on the Moon and Mars," 24 September 2017. [Online]. Available: https://www.europlanet-society.org/lava-tubes-the-hidden-sites-for-future-human-habitats-on-the-moon-and-mars/. [Accessed 13 January 2020].

[9] "Possible Skylight on a Lava Tube Northeast of Arsia Mons," 21 August 2009. [Online]. Available: https://www.uahirise.org/ESP_014380_1775. [Accessed 20 July 2020].

[10] J. J. Banfield, C. S. Edwards, D. R. Montgomery and B. D. Brand, "The dual nature of the martian crust: Young lavas and old clastic materials," *Icarus,* vol. 222, no. 1, pp. 188-199, 2013.

[11] D. Ferguson, A. Morris, D. Hahnel, C. Baker, Z. Omohundro, C. Reverte, S. Thayer, W. Whittaker, W. Whittaker, W. Burgard and S. Thrun, "An

Autonomous Robotic System for Mapping Abandoned Mines," *IEEE Robotics & Automation Magazine,* vol. 11, no. 4, pp. 79-91, 2004.

[12] G. Zhang, B. Shang, Y. Chen and H. Moyes, "SmartCaveDrone: 3D Mapping Using UAV's as Robotic Co-Archaeologiests," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, Miami, Fl, USA, 2017.

[13] A. Husain, H. Jones, B. Kannan, U. Wong, T. Pimentel, S. Tang, S. Daftry, S. Huber and W. L. Whittaker, "Mapping Planetary Caves with an Autonomous Heterogeneous Robot Team," in *2013 IEEE Aerospace Conference*, Big Sky, MT, USA, 2013.

[14] D. Tardoli, L. Riazuelo, D. Sicignano, C. Rizzo, F. Lera and J. L. Villarroel, "Ground robotics in tunnels: Keys and lessons learned after 10 years of research and experiments," *Journal of Field Robotics,* vol. 36, no. 6, pp. 1074-1101, 2019.

[15] "About DARPA," DARPA, n.d.. [Online]. Available: https://www.darpa.mil/about-us/about-darpa. [Accessed 20 January 2020].

[16] C. Linder, "A Cave Is No Place for Humans, So DARPA Is Sending In the Robots," Popular Mechanics, 23 August 2019. [Online]. Available: https://www.popularmechanics.com/military/research/a28771417/darpa-subterranean-challenge/. [Accessed 14 November 2019].

[17] R. Linsenmayer, "Field Report: Lessons From First Leg of DARPA Subterranean Challenge," Robotics Business Review, 29 August 2019. [Online]. Available: https://www.roboticsbusinessreview.com/events/field-report-lessons-from-first-leg-of-darpa-subterranean-challenge/. [Accessed 15 January 2020].

[18] European Space Agency, "Exploring underground with a colliding drone," ESA, 22 5 2017. [Online]. Available: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/CAVES_and_Pangaea/Exploring_underground_with_a_colliding_drone. [Accessed 4 9 2019].

[19] OpenCV team, "About," OpenCV, 2020. [Online]. Available: https://opencv.org/about/. [Accessed 28 5 2020].

[20] S. Sati and A. El-bareg, "MANET Testbed using Raspberry PIs," *I.J. Wireless and Microwave Technologies,* vol. 8, no. 2, pp. 52-63, 2018.

[21] D. Mahajan, "Routing in Mobile Robots," Rochester Institute of Technology, Rochester, New York, 2016.

[22] F. Zeiger, N. Kraemer and K. Schilling, "Commanding Mobile Robots via Wireless Ad-Hoc Networks - A Comparison of Four Ad-Hoc Routing

Protocol Implementations," in *2008 IEEE International Conference on Robotics and Automation*, Pasadena, California, USA, 2008.

[23] W. Fehse, Automated Rendezvous and Docking of Spacecraft, New York: Cambridge University Press, 2003.

[24] C. Phillip and R. Dabney, "Solution to the problem of determining the relative 6 DOF state for spacecraft automated rendezvous and docking," in *SPIE's 1995 Symposium on OE/Aerospace Sensing and Dual Use Photonics*, Orlando, Florida, United States, 1995.

[25] I. Y. Bar-Itzhack, "REQUEST - A recursive QUEST algorithm for sequential attitude determination," *Journal of Guidance, Control, and Dynamics,* vol. 19, no. 5, pp. 1034-1038, 1996.

[26] M. D. Shuster and S. D. Oh, "Three-Axis Attitude Determination from Vector Observations," *Journal of Guidance and Control,* vol. 4, no. 1, pp. 70-77, 1981.

[27] S. M. Kelley and S. P. Cryan, "Navigation and Alignment Aids Concept of Operations and Supplemental Design Information. Revision A," 2016.

[28] T. Chung, "DARPA Subterranean (SubT) Challenge," DARPA, [Online]. Available: https://www.darpa.mil/program/darpa-subterranean-challenge. [Accessed 27 September 2019].

[29] DARPA, "Teams CoSTAR and BARCS Take Top Spost in DARPA Subterranean Challenge Urban Circuit," 27 2 2020. [Online]. Available: https://www.darpa.mil/news-events/2020-02-27. [Accessed 8 6 2020].

[30] R. A. Kerr, "Rainbow of Martian Minerals Paints Picture of Degradation," *Science,* vol. 305, pp. 770,771, 2004.

Appendix A:
Setting Up Raspberry Pi Communication Beacons

The hardware requirements for setting the communication nodes are a raspberry pi version 3b+ and a sd card. In this case, a 32gb card was used.

1. Getting Raspbian

Raspbian Buster lite was downloaded from the official source at https://www.raspberrypi.org/downloads/raspbian/.
Install the .ISO onto the sd card using a second pc. While any .iso writing utility can be used, balenaEtcher was recommended, and was able to install the iso without unzipping the download above. https://www.balena.io/etcher/

2. Configuring Raspbian

The wlan0 network will be disabled by default, as well as using the British keyboard by default. To enable wlan0, it is necessary to first configure the raspberry pi's wireless location. While completing, it makes sense to change the keyboard and language as well.

First, sign into the pi, by default the username is pi, and the password is raspberry. You may wish to change this later. Then enter the configuration menu using the command

```
$sudo raspi-config
```

Choose the second option, Localization, and set your language in locale, using space to select and unselect options, choosing the -UT8 version of the language of choice. Next, change the keyboard layout. For English(US) select other, English(US), the top option, and finally finish with any options desired in the keyboard layout.
Finally, to enable wlan0, select the final option "Change Wi-Fi Country." After selecting a country, the wlan0 network will automatically be enabled.

3. Updating the card and getting B.A.T.M.A.N-adv

All commands will be run in root, for ease the following command was used:

```
$ Sudo su
```

Following this, update the raspberry pi and install B.A.T.M.A.N.

```
apt-get update -y
apt-get install batctl
```

Now, create the script for starting the network

```
nano /root/BATMAN-mesh.sh
```

The following code will configure the wlan0 network for BATMAN protocol.

```
sudo modprobe batman-adv
killall wpa_supplicant
sudo ip link set wlan0 down
sudo iwconfig wlan0 mode ad-hoc
sudo iwconfig wln0 essid <NETWORK NAME>
sudo iwconfig wlan0 ap any
sudo iwconfig wlan0 channel 8
sleep 1s
sudo ip link set wlan0 up
sleep 1s
sudo batctl if add wlan0
sleep 1s
sudo ifconfig bat0 up
$sleep 5s
$sudo ifconfig bat0 192.168.1.1/16
```

Replace <NETWORK NAME> with the desired network name, in the case of this project, MarsNet. When configuring additional nodes, the ip address must be changed to another, unused address. In this project, most of the ip address remained the same following the convention192.168.1.-/16 for nodes, and 192.168.2.-/16 for robots.

Finally, the script must be given execute commands using chmod.

```
#Chmod 755 mesh.sh
```

To run the script, cd to root and run mesh.sh

```
# cd root
# ./mesh.sh
```

Check that all nodes are connected to the desired network using

```
$sudo iwconifg
```

Or by checking the connected nodes through

```
$sudo batctl n
```

Appendix B:
ROS Custom Messages

LEDPoints.msg

```
## message to send the position of one set of LED
##data from OpenCV as well as camera number and led color

float64 LED_1_alpha
float64 LED_1_beta

float64 LED_2_alpha
float64 LED_2_beta

float64 LED_3_alpha
float64 LED_3_beta

float64 LED_4_alpha
float64 LED_4_beta

float64 LED_5_alpha
float64 LED_5_beta


int8 Camera_number

string LED_color
```

GlobalPos.msg

```
float64 Yaw
float64 Pitch
float64 Roll

float64 Xpos
float64 Ypos
float64 Zpos
```

BotPositionQuat.msg

```
# Send both Position and orientation of a Robot based on the
Hexagonal Sensor Array

float64 Range

float64 QuatX
float64 QuatY
float64 QuatZ
float64 QuatW

#Unit vector of quest point 1
float64 Unit_Vec1x
float64 Unit_Vec1y
float64 Unit_Vec1z
```

Appendix C:
QUEST Algorithm – C++

```
/*
A QUEST algorithm for a 5 led beacon array
By Ryan Capozzi
March 20, 2020
*/

//include ROS dependencies
#include <ros/ros.h>
#include <geometry_msgs/Transform.h>
#include <message_filters/subscriber.h>


//include matrix dependencies
#include </usr/include/eigen3/Eigen/Dense>
#include </usr/include/eigen3/Eigen/Eigenvalues>
#include </usr/include/eigen3/Eigen/QR>
#include </usr/include/eigen3/Eigen/SVD>
#include </usr/include/eigen3/Eigen/Geometry>

#include <armadillo>


//include custom Messages
#include "thesis_messages/BotPositionQuat.h"
#include "thesis_messages/LEDPoints.h"



using namespace arma;
using namespace std;
using namespace message_filters;
using namespace std::chrono;
using namespace Eigen;

class QuestALG
{
public:
QuestALG();

void UpdateLocation(const thesis_messages::LEDPoints
&LEDPositions);
```

```
void GetRange();

void GetQuat();

private:
//PRIVATE ROS VARIABLES
ros::NodeHandle nh;
ros::Publisher pub_;
ros::Subscriber sub_;

// distance between each set of leds along unit vectors
double l12;
double l23 = 140;//138.18;//140;
double l13;
// array for ranges to each LED in Newton Solution (3 leds at a
time)
double NS[3];
//unit vectors to all points (x,y,z)
Vector3d r1;
Vector3d r2;
Vector3d r3;
Vector3d r4;
Vector3d r5;
//ranges to all 5 LEDs
double R1;
double R2;
double R3;
double R4;
double R5;
double Range_to_base;

//returned quaternion
Eigen::Quaterniond q;
//functions required for finding solution to QUEST algorithm
//newton-Raphson solver
void NewtonSolver(double Cr12, double Cr13, double Cr23);

void GetUnitVectors(double LED_1_a, double LED_1_b, double
LED_2_a, double LED_2_b, double LED_3_a, double LED_3_b, double
LED_4_a,
double LED_4_b, double LED_5_a, double LED_5_b);
};


QuestALG::QuestALG():
r1{0,0,0}, r2{0,0,0}, r3{0,0,0}, r4{0,0,0}, r5{0,0,0}
```

```
{
    l12 = l13 = 83.762;//80.75;//83.762;



    pub_ = nh.advertise
<thesis_messages::BotPositionQuat>("/BotRelLoc",10);

    sub_ = nh.subscribe("/LEDPositions",1,
&QuestALG::UpdateLocation, this);
}
void QuestALG::UpdateLocation(const thesis_messages::LEDPoints
&LEDPositions){
    //create publish messages
    thesis_messages::BotPositionQuat msg;

QuestALG::GetUnitVectors(LEDPositions.LED_1_alpha,LEDPositions.
LED_1_beta,LEDPositions.LED_2_alpha,LEDPositions.LED_2_beta,

LEDPositions.LED_3_alpha,LEDPositions.LED_3_beta,LEDPositions.L
ED_4_alpha,LEDPositions.LED_4_beta,LEDPositions.LED_5_alpha,LED
Positions.LED_5_beta);
    QuestALG::GetRange();
    QuestALG::GetQuat();

    //publish
    double quat[] = {q.w(),q.vec()[0],q.vec()[1],q.vec()[2]};
    msg.Range = R1;
    msg.QuatW = quat[0];
    msg.QuatX = quat[1];
    msg.QuatY = quat[2];
    msg.QuatZ = quat[3];
    msg.Unit_Vec1x = r1[0];
    msg.Unit_Vec1y = r1[1];
    msg.Unit_Vec1z = r1[2];
    pub_.publish(msg);
}
void QuestALG::GetUnitVectors(double LED_1_a, double LED_1_b,
double LED_2_a, double LED_2_b, double LED_3_a, double LED_3_b,
double LED_4_a,
double LED_4_b, double LED_5_a, double LED_5_b)
{
    // calcuate unit vectors needed for both range and
quaternion position functions
    r1 << -cos(LED_1_a)*cos(LED_1_b),
    -sin(LED_1_a),
    -cos(LED_1_a)*sin(LED_1_b);
```

```
    r2 << -cos(LED_2_a)*cos(LED_2_b),
    -sin(LED_2_a),
    -cos(LED_2_a)*sin(LED_2_b);

    r3 << -cos(LED_3_a)*cos(LED_3_b),
    -sin(LED_3_a),
    -cos(LED_3_a)*sin(LED_3_b);

    r4 << -cos(LED_4_a)*cos(LED_4_b),
    -sin(LED_4_a),
    -cos(LED_4_a)*sin(LED_4_b);

    r5 << -cos(LED_5_a)*cos(LED_5_b),
    -sin(LED_5_a),
    -cos(LED_5_a)*sin(LED_5_b);

    //ROS_INFO_STREAM("Found Unit Vectors");
}
// Get the range from the camera to the beacon
void QuestALG::GetRange(){

    //For first range measurement
    double Cr12 = r1.dot(r2);
    double Cr23 = r2.dot(r3);
    double Cr13 = r1.dot(r3);
    //For second range measurement
    double Cr14 = r1.dot(r4);
    double Cr45 = r4.dot(r5);
    double Cr15 = r1.dot(r5);
    // ROS_INFO_STREAM("Calculated Cosines");

    //call newton solver for fist set of LED's
    NewtonSolver(Cr12,Cr13,Cr23);
    double R1a = NS[0];
    R2 = NS[1];
    R3 = NS[2];
    //call newton solver for second set of LED's
    NewtonSolver(Cr14,Cr15,Cr45);
    // ROS_INFO_STREAM("NewtonSolverComplete");
    double R1b = NS[0];
    R4 = NS[1];
    R5 = NS[2];
    // get better accuracy on distance to out-of-plane led
    R1 = (R1a + R1b)/2;
    ROS_INFO_STREAM("Ranges R1a, R1b");
    ROS_INFO_STREAM(R1a);
```

```
    ROS_INFO_STREAM(R1b);
    // ROS_INFO_STREAM("GotRange");


}
void QuestALG::NewtonSolver(double Cr12, double Cr13, double
Cr23)
{
    int max_iterations = 10000;
    // initialize (max) error value to check against
    double errorM = -1;
    // error array of R1, R2, and R3
    double earray[] = {-1, -1,-1};
    // Set Maximum error value to be accepted
    double NR_TOLERANCE = 0.000001;
    // set initial guess equal to half desired effective range
(mm)
    Vector3d Xo;
    Xo << 100,500,1000;
    // initialize X
    Vector3d X;
    X=Xo;
    MatrixXd Jac(3,3);

    // initialize non-changing jacobian cells
    Jac(0,2)=Jac(1,0)=Jac(2,1)=0;

    //create jacobian inverse
    MatrixXd Jac_inverse(3,3);
    //define Newton Raphson Function
    Vector3d NRfun(3);

    int iterations = 1;
    MatrixXd Mult_ans(3,3); //initialize output for
multiplication function
    do
    {
        errorM = -1; //reset errorM to -1 -- protect against
false assignment

        //Iterate Newton-Raphson Method
        R1=X(0);
        R2=X(1);
        R3=X(2);
        //calculate Jacobimatrix
        Jac(0,0)=2*R1-2*R2*Cr12;
        Jac(0,1)=2*R2-2*R1*Cr12;
```

```
        Jac(1,1)=2*R2-2*R3*Cr23;
        Jac(1,2)=2*R3-2*R2*Cr23;

        Jac(2,0)=2*R1-2*R3*Cr13;
        Jac(2,2)=2*R3-2*R1*Cr13;

        //ROS_INFO_STREAM("Jacobian Calculated");
        //calculate Function Vector

        NRfun << (pow(R1,2)+pow(R2,2)-2*R1*R2*Cr12 -
pow(l12,2)),
        (pow(R2,2)+pow(R3,2)-2*R2*R3*Cr23 - pow(l23,2)),
        (pow(R1,2)+pow(R3,2)-2*R1*R3*Cr13 - pow(l13,2));

        //ROS_INFO_STREAM("Calcualte Function Vector");
        // ROS_INFO_STREAM("Jac");
        // ROS_INFO_STREAM(Jac);
        Jac_inverse = Jac.inverse();
        X = X - (Jac_inverse * NRfun);


        //calculate maximum error
        for (int i=0;i<=2;i++)
        {


            earray[i] = abs(X(i)-Xo(i));


            if(earray[i] > errorM)
            {
                errorM = earray[i];
            }


        }

        Xo = X;
        iterations++;
    }while(errorM >= NR_TOLERANCE && iterations <=
max_iterations);
    //set NS to final solved ranges
    // ROS_INFO_STREAM("NS Range Solutions");
    for (int i = 0; i <= 2; i++){
    NS[i] = Xo[i];
    // ROS_INFO_STREAM(NS[i]);
```

```
    }
    // ROS_INFO_STREAM("");
}

void QuestALG::GetQuat(){


    //w - i'th measurement in robot frame (cartesian)
    Vector3d w;
    //v - i'th measurement in reference(beacon) frame
(cartesian)
    MatrixXd B_MV(4,3);

    // B_MV.row(0) << 46,-70,0;    //12
    // B_MV.row(1) << 46,70,0;     //13
    // B_MV.row(2) << 46,0,70;     //14
    // B_MV.row(3) << 46,0,-70;    //15

    B_MV.row(0) << 46,-70,0;    //12
    B_MV.row(1) << 46,70,0;     //13
    B_MV.row(2) << 46,0,70;     //14
    B_MV.row(3) << 46,0,-70;    //15


    Vector3d v;
    //define sigma
    double sigma = 0;
    //define S
    Matrix3d S = Matrix3d::Zero();
    //define z (vector)
    Vector3d z = Vector3d::Zero();
    //define K
    Matrix4d K = Matrix4d::Zero();
    // define a -- assumed to remain 1
    double a[] = {0.05,0.05,0.45,0.45};

    //calculate summations
    for(int i=0;i<4;i++){
    //calculate v and w vectors
    v = B_MV.row(i);
    v = v/v.norm();

    //set W to r1-r5 unit vectors previously calculated
    switch(i){
        case 0:
        w = R1*r1-R2*r2;  //r12
```

```
    break;

    case 1:
    w= R1*r1-R3*r3;    //r13
    break;

    case 2:
    w= R1*r1-R4*r4;    //r14
    break;

    case 3:
    w= R1*r1-R5*r5;    //r15
    break;

    default:
    ROS_DEBUG_ONCE("GetQuat Error, for(i) out of bounds");
    break;
}
w = w/w.norm();
// ROS_INFO_STREAM("v");
// ROS_INFO_STREAM(v);
// ROS_INFO_STREAM("w");
// ROS_INFO_STREAM(w);

//calculate sigma
    sigma +=a[i]*w.dot(v);
//calculate S
    S += a[i]*(w*v.transpose()+ v*w.transpose());
//calculate z
    z += a[i]*w.cross(v);
}
ROS_INFO_STREAM("r1");
ROS_INFO_STREAM(R1);
ROS_INFO_STREAM("r2");
ROS_INFO_STREAM(R2);
ROS_INFO_STREAM("r3");
ROS_INFO_STREAM(R3);
ROS_INFO_STREAM("r4");
ROS_INFO_STREAM(R4);
ROS_INFO_STREAM("r5");
ROS_INFO_STREAM(R5);
K << (S-sigma*Matrix3d::Identity()), z,
        z.transpose(), sigma;
ROS_INFO_STREAM("k Matrix");
ROS_INFO_STREAM(K);

ROS_INFO_STREAM(K.eigenvalues());
```

```
    //Gibb's Vector Solution
    MatrixXd p;  //intermediate matrix
    Vector3d g;
    p = ((1+sigma)*Matrix3d::Identity()-S);
    g = p.inverse()*z;
    // ROS_INFO_STREAM("P and invP");
    // ROS_INFO_STREAM(p);
    // ROS_INFO_STREAM("");
    // ROS_INFO_STREAM(p.inverse());
    q.vec() = 1/(sqrt(1+g.squaredNorm()))*g;
    q.w() = 1/(sqrt(1+g.squaredNorm()));

}

int main(int argc, char *argv[]){

    //initiate ros node
    ros::init(argc,argv, "QuestAlgorithm");
    //create quest object to run everything
    QuestALG Quest_obj;

    ros::spin();
}
```

Appendix D:
Beacon Tracking Node – Python

```
#!/usr/bin/env python

import rospy

#import opencv
import cv2

#some python dependencies for this node
import glob, time, argparse
import numpy as np
from operator import itemgetter
from math import atan, tan
import pandas as pd
import csv
#import the LEDPoints message
from thesis_messages.msg import LEDPoints
global testcounter
testcounter = 0

if __name__ == '__main__':
    #CAMERA PARAMETERS
    VisionMod = 1#.28 #corrects for static innacurate range
from camera
    #parameters in mm
    focal_len = 10
    sensorsizex = 22.3
    sensorsizey = 14.9
    pixelpitch = 0.0043

    sensorpixx = 5184
    sensorpixy = 3456
    #read camera settings
    CameraMatrix =
pd.read_csv("/home/ryan/catkin_ws/src/beacon_localization/Camer
aMatrix.csv", header = None)
    mtx = CameraMatrix.to_numpy()

    distarray =
pd.read_csv("/home/ryan/catkin_ws/src/beacon_localization/dista
rray.csv", header = None)
    dist = distarray.to_numpy()
```

```
    rvecsdf =
pd.read_csv("/home/ryan/catkin_ws/src/beacon_localization/rvecs
.csv", header = None)
    rvecs = rvecsdf.to_numpy()

    tvecsdf =
pd.read_csv("/home/ryan/catkin_ws/src/beacon_localization/tvecs
.csv", header = None)
    tvecs = tvecsdf.to_numpy()
    # print('camera matrix')
    # for row in mtx:
    #     print(row)
    # print('dist')
    # for row in dist:
    #     print(row)
    # print('rvecs')
    # for row in rvecs:
    #     print(row)
    # print('tvecs')
    # for row in tvecs:
    #     print(row)


    #ROS Publisher Setup
    rospy.init_node('VisualTargetTracker')
    pub =
rospy.Publisher('/LEDPositions',LEDPoints,queue_size=15)


    # Get the filename from the command line
    files =
glob.glob('/home/ryan/catkin_ws/src/beacon_localization/scripts
/images/5 degree change 1-5m v1/5m/*.JPG')
    files.sort()
    # load the image
    rawimage = cv2.imread(files[0])

    #Resize the image
    scale_percent = 50
    width = int(rawimage.shape[1]*scale_percent/100)
    height = int(rawimage.shape[0]*scale_percent/100)
    dim = (width,height)
    rawimage = cv2.resize(rawimage,dim)
    #parameters in pixels
    height, width, channels = rawimage.shape
```

```
    #get new optimal camera matrix
    # newcammtx, roi =
cv2.getOptimalNewCameraMatrix(mtx,dist,(width,height),1,(width,
height))
    #crop the image
    # x,y,w,h = roi

    #create windows
    cv2.namedWindow('P-> Previous, N->
Next',cv2.WINDOW_AUTOSIZE)
    cv2.namedWindow('Red',cv2.WINDOW_AUTOSIZE)
    cv2.namedWindow('Green',cv2.WINDOW_AUTOSIZE)
    cv2.namedWindow('Blue',cv2.WINDOW_AUTOSIZE)
    cv2.namedWindow('Yellow',cv2.WINDOW_AUTOSIZE)


    #show all images
    cv2.imshow('Red',rawimage)
    cv2.imshow('Green',rawimage)
    cv2.imshow('Blue',rawimage)
    cv2.imshow('Yellow',rawimage)

    #create thershold values for each color
    minRed = np.array([81,0,96]) #red may need new thresholding
    maxRed = np.array([180,35,255])

    minGreen = np.array([70,64,50])
    maxGreen = np.array([85,143,255])

    minBlue = np.array([84,40,152])
    maxBlue = np.array([180,255,255])

    minYellow = np.array([12,190,45])
    maxYellow = np.array([26,255,255])
    i=0
    while not rospy.is_shutdown():
        cv2.imshow('P-> Previous, N-> Next',rawimage)
        k = cv2.waitKey(1) & 0xFF

        # check next image in folder
        if k == ord('n'):
            i += 1
            rawimage = cv2.imread(files[i%len(files)])
            rawimage = cv2.resize(rawimage,dim)#interpolation =
cv2.INTER_AREA)

            # # undistort image
```

```
            # rawimage = cv2.undistort(rawimage, mtx, dist,
None)
            # rawimage = rawimage[y:y+h,x:x+w]

        # check previous image in folder
        elif k == ord('p'):
            i -= 1
            rawimage = cv2.imread(files[i%len(files)])
            rawimage = cv2.resize(rawimage,dim)#,interpolation
= cv2.INTER_AREA)

            # undistort image
            # rawimage = cv2.undistort(rawimage, mtx, dist,
None)
            # rawimage = rawimage[y:y+h,x:x+w]



        #convert image to HSV
        verpixels, horpixels, channels = rawimage.shape
        hsvimg = cv2.cvtColor(rawimage,cv2.COLOR_BGR2HSV)

        #create masks for images
        maskR = cv2.inRange(hsvimg,minRed,maxRed)
        maskG = cv2.inRange(hsvimg,minGreen,maxGreen)
        maskB = cv2.inRange(hsvimg,minBlue,maxBlue)
        maskY = cv2.inRange(hsvimg,minYellow,maxYellow)

        #Mask Modification
        kernel =
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
        #   use closing to remove holes in mask
        maskR = cv2.morphologyEx(maskR,cv2.MORPH_CLOSE,
kernel,iterations=5)
        maskG = cv2.morphologyEx(maskG,cv2.MORPH_CLOSE,
kernel,iterations=5)
        maskB = cv2.morphologyEx(maskB,cv2.MORPH_CLOSE,
kernel,iterations=5)
        maskY = cv2.morphologyEx(maskY,cv2.MORPH_CLOSE,
kernel,iterations=5)

        kernelE =
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(2,4))
        maskR = cv2.erode(maskR, kernelE, iterations = 4)
        maskG = cv2.erode(maskG, kernelE, iterations = 4)
        maskB = cv2.erode(maskB, kernelE, iterations = 4)
        maskY = cv2.erode(maskY, kernelE, iterations = 4)
```

```
        # maskY = cv2.morphologyEx(maskY,cv2.MORPH_CLOSE,
kernelE,iterations=2)

        # create composite of masks and original image
        resultR = cv2.bitwise_and(rawimage, rawimage, mask =
maskR)
        resultG = cv2.bitwise_and(rawimage, rawimage, mask =
maskG)
        resultB = cv2.bitwise_and(rawimage, rawimage, mask =
maskB)
        resultY = cv2.bitwise_and(rawimage, rawimage, mask =
maskY)


        #show the result of masking
        # cv2.imshow('Red',resultR)
        # cv2.imshow('Green',resultG)
        # cv2.imshow('Blue',maskB)
        cv2.imshow('MaskTest',maskY)

        #Blob Detector Parameters
        params = cv2.SimpleBlobDetector_Params()
        #filter by blob distance
        params.minDistBetweenBlobs = 2.5
        #filter by min/max area
        params.filterByArea = True
        params.minArea = 2
        params.maxArea = 100000

        #filter by convexity
        params.filterByConvexity = True
        params.minConvexity = 0.0

        #filter by circularity
        params.filterByCircularity = True
        params.minCircularity = 0.0

        #filter by Intertia
        params.filterByInertia = True
        params.minInertiaRatio = 0.0

        #Use the masks for blob detection
        detector = cv2.SimpleBlobDetector_create(params)

        #detection
        Rkeypoints = detector.detect(255-maskR)
        Gkeypoints = detector.detect(255-maskG)
```

```
        Bkeypoints = detector.detect(255-maskB)
        Ykeypoints = detector.detect(255-maskY)

        #show detected keypoints
        RwithKeypoints =
cv2.drawKeypoints(resultR,Rkeypoints,np.array([]),(0,0,255),cv2
.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
        GwithKeypoints =
cv2.drawKeypoints(resultG,Gkeypoints,np.array([]),(0,0,255),cv2
.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
        BwithKeypoints =
cv2.drawKeypoints(resultB,Bkeypoints,np.array([]),(0,0,255),cv2
.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
        YwithKeypoints =
cv2.drawKeypoints(resultY,Ykeypoints,np.array([]),(0,0,255),cv2
.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

        cv2.imshow('Red',RwithKeypoints)
        cv2.imshow('Green',GwithKeypoints)
        cv2.imshow('Blue',BwithKeypoints)
        cv2.imshow('Yellow',YwithKeypoints)

        #Determine if a beacon is found, and which side
        j=0
        sourcePosition= [[]]
        cartkeypoint = [0,0]
        kp_size = 0
        color = ''
        if len(Rkeypoints) ==5:
            sourcePosition = cv2.KeyPoint_convert(Rkeypoints)
            color = 'R'
        elif len(Gkeypoints) ==5:
            sourcePosition = cv2.KeyPoint_convert(Gkeypoints)
            color = 'G'
        elif len(Bkeypoints) ==5:
            sourcePosition = cv2.KeyPoint_convert(Bkeypoints)
            color = 'B'
        elif len(Ykeypoints) ==5:
            sourcePosition = cv2.KeyPoint_convert(Ykeypoints)
            color = 'Y'


        if len(sourcePosition)==5:

            #--------------Camera Math--------------

            #calculate horizontal and vertical fields of view
```

```
            ymaxcam = sensorpixy/2
            xmaxcam = sensorpixx/2

            ymax = verpixels/2  #scaled image
            xmax = horpixels/2

            horFOV = atan(xmaxcam*pixelpitch/focal_len)
            verFOV = atan(ymaxcam*pixelpitch/focal_len)
            # print(horFOV)
            # print(verFOV)


            #shift points to center of camera frame
            shiftedkeypoints = [[0,0],[0,0],[0,0],[0,0],[0,0,]]
            for j in range(5):
                shiftedkeypoints[j][0] = xmax -
sourcePosition[j][0]
                shiftedkeypoints[j][1] = sourcePosition[j][1]-
ymax

             #Normalize Keypoints
            normalizedkeypoints =
[[0,0],[0,0],[0,0],[0,0],[0,0,]]

            for j in range(5):
                normalizedkeypoints[j][0] =
shiftedkeypoints[j][0]/xmax
                normalizedkeypoints[j][1] =
shiftedkeypoints[j][1]/ymax

            #convert to azimuth and elevation (rad)
            viewangles = [[0,0],[0,0],[0,0],[0,0],[0,0,]]
            for j in range(5):
                viewangles[j][0]=
atan(normalizedkeypoints[j][0]*xmaxcam*pixelpitch/focal_len)
                viewangles[j][1]=
atan(normalizedkeypoints[j][1]*ymaxcam*pixelpitch/focal_len)

            #Sort Keypoints
            sortedkeypoints = [[0,0],[0,0],[0,0],[0,0],[0,0,]]
            sortingkeypoints = sorted(viewangles, key=lambda x:
x[1])
            #find points 4 and 5
            sortedkeypoints[3] = sortingkeypoints[-1]  #final
element in list
            sortedkeypoints[4] = sortingkeypoints[0]
            del sortingkeypoints[-1]
```

```
            del sortingkeypoints[0]
            #find points 1, 2, and 3
            sortingkeypoints = sorted(sortingkeypoints,
key=lambda x: x[0])
            sortedkeypoints[2] = sortingkeypoints[0]
            sortedkeypoints[1] = sortingkeypoints[-1]
            sortedkeypoints[0] = sortingkeypoints[1]



            if sortedkeypoints[0]!=[0,0]:
                rosmessage = LEDPoints()
                rosmessage.Camera_number = 0
                rosmessage.LED_color = color
                rosmessage.LED_1_alpha = sortedkeypoints[0][0]
                rosmessage.LED_1_beta = sortedkeypoints[0][1]

                rosmessage.LED_2_alpha = sortedkeypoints[1][0]
                rosmessage.LED_2_beta = sortedkeypoints[1][1]

                rosmessage.LED_3_alpha = sortedkeypoints[2][0]
                rosmessage.LED_3_beta = sortedkeypoints[2][1]

                rosmessage.LED_4_alpha = sortedkeypoints[3][0]
                rosmessage.LED_4_beta = sortedkeypoints[3][1]

                rosmessage.LED_5_alpha = sortedkeypoints[4][0]
                rosmessage.LED_5_beta = sortedkeypoints[4][1]
                pub.publish(rosmessage)

            # if testcounter%500 ==0:
            #     print(sourcePosition)
            #     print(shiftedkeypoints, color)
            #     print()
                # print("sorting keypoints")
                # print(sortingkeypoints)
                # print()
                # print("sorted keypoints")

                # print(sortedkeypoints)
                # print()
                # print("Horizontal FOV Is {}".format(horFOV))
                # print("Vertical FOV Is {}".format(verFOV))
                # print("Horizontal scaling factor Is
{}".format(pixscalex))
                # print("Vertical scaling factor Is
{}".format(pixscaley))
```

```
testcounter =testcounter +1
```

Appendix E:
Local To Global Coordinate Transformation – Python

```
#!/usr/bin/env python

import rospy
import message_filters
import math
import numpy as np
#get necessary messages
from thesis_messages.msg import LEDPoints
from thesis_messages.msg import BotPositionQuat
from thesis_messages.msg import GlobalPos

LEDMESSAGE = LEDPoints()
BOTMESSAGE = BotPositionQuat()
def TransformMatrixFromQuat(quatx,quaty,quatz,quatw,lx,ly,lz):
    """Creates a Homogeneous transformation matrix from
quaternion and linear transformation
        inputs, using a direction cosine matrix"""

    #generate direction cosine matrix from quaternion
    q0 = quatw
    q1 = quatx
    q2 = quaty
    q3 = quatz
    cosmatrix = [[(q0**2+q1**2-q2**2-q3**2), 2*(q1*q2-q0*q3),
2*(q1*q3-q0*q2)], [2*(q1*q2-q0*q3), (q0**2-q1**2+q2**2-q3**2),
2*(q2*q3+q0*q1)], [2*(q1*q3+q0*q2), 2*(q2*q3-q0*q1),(q0**2-
q1**2-q2**2+q3**2)]]
    # cosmatrix=[[1,2,3],[4,5,6],[7,8,9]] #for testing only

    HomTransform = [[cosmatrix[0][0], cosmatrix[0][1],
cosmatrix[0][2], lx], [cosmatrix[1][0], cosmatrix[1][1],
cosmatrix[1][2],ly], [cosmatrix[2][0], cosmatrix[2][1],
cosmatrix[2][2],lz],[0,0,0,1]]
    return(HomTransform)

def
TransformMatrixFromRotation(theta_x,theta_y,theta_z,lx,ly,lz):
    """Returns Homogeneous Transformation matrix given rotation
angles(given in Radians) in the x,y,z axes
    and linear transformations"""
```

```
    RotX = [[1, 0, 0], [0, math.cos(theta_x), -
math.sin(theta_x)], [0, math.sin(theta_x), math.cos(theta_x)]]
#X rotation matrix
    RotY = [[math.cos(theta_y), 0, math.sin(theta_y)], [0, 1,
0], [-math.sin(theta_y), 0, math.cos(theta_y)]] #Y rotation
matrix
    RotZ = [[math.cos(theta_z), -math.sin(theta_z), 0],
[math.sin(theta_z), math.cos(theta_z), 0], [0, 0, 1]] #Z
rotation matrix

    RotMat = RotZ
rotation
    HomTransform = [[RotMat[0][0], RotMat[0][1], RotMat[0][2],
lx], [RotMat[1][0], RotMat[1][1], RotMat[1][2],ly],
[RotMat[2][0], RotMat[2][1], RotMat[2][2],lz],[0,0,0,1]]
    return(HomTransform)

def GetTargetRotation(Color):
    '''takes in LED color and returns z axis rotation from
beacon plane'''
    if Color == 'R':
        return math.pi
    if Color == 'G':
        return math.pi/2
    if Color == 'B':
        return(0)
    if Color == 'Y':
        return 3*math.pi/2

def CameraRotation(Cam_numb,sensor_cameras):
    '''convert camera number to angle around sensor'''
    '''Cam_numb: camera number clockwise around sensor'''
    '''sensor_cameras: number of cameras in sensor'''
    interior_angle = ((sensor_cameras-
2)*math.pi/2)/sensor_cameras + math.pi

    camera_angle = interior_angle*Cam_numb
    return camera_angle

def localtoglobal(LED_data, botquat_data):
    """uses the LEDPoints and BotPositionQuat messages to
convert from local position to global position
    """
    #get useful information from LEDPoints message:
    Camera_number = LED_data.Camera_number
    LED_color = LED_data.LED_color
    #get useful information from BotPositionQuat message:
```

```
    TargetRange = botquat_data.Range
    #quat is in [w,x,y,z] format
    Quat = [botquat_data.QuatW, botquat_data.QuatX,
botquat_data.QuatY, botquat_data.QuatZ]
    unitVec = [botquat_data.Unit_Vec1x,
botquat_data.Unit_Vec1y, botquat_data. Unit_Vec1z]

    #go from beacon frame to target frame
    lt = 140 #distance from target to center of beacon in mm
    Ct = GetTargetRotation (LED_color)
    T1 =
TransformMatrixFromRotation(0,0,Ct,lt*math.cos(Ct),lt*math.sin(
Ct),0)

    T1np = np.array(T1)
    # print('T1')
    # print(T1np)
    # print('')
    #go from target to camera frame
    #generate direction cosine matrix from quaternion
    q0 = Quat[0]
    q1 = Quat[1]
    q2 = Quat[2]
    q3 = Quat[3]
    cosmatrix = [[(q0**2+q1**2-q2**2-q3**2), 2*(q1*q2-q0*q3),
2*(q1*q3-q0*q2)],
                 [2*(q1*q2-q0*q3), (q0**2-q1**2+q2**2-q3**2),
2*(q2*q3+q0*q1)],
                 [2*(q1*q3+q0*q2), 2*(q2*q3-q0*q1),(q0**2-q1**2-
q2**2+q3**2)]]

    #find offset
    l2 = TargetRange*(np.array(unitVec))
    print(unitVec)
    print (l2)
    T2 = TransformMatrixFromQuat(q0, q1, q2, q3, l2[0], l2[1],
l2[2])

    T2np = np.array(T2)
    print('T2')
    print(T2np)
    print('')
    #go from camera to robot frame
    l3 = 200 #distance from camera to center of camera array
    camera_angle = CameraRotation(Camera_number,6)
    print(camera_angle)
```

```
    T3 = TransformMatrixFromRotation(0,0,camera_angle, l3, 0,
0)
    T3np = np.array(T3)
    # print('T3')
    # print(T3np)
    BotinGlobal = np.matmul(np.matmul(T1np,T2np),T3np)
    #calculate orientation as yaw/pitch/roll
    GlobalRotation = BotinGlobal[0:3,0:3] # get rotation matrix
for system
    yaw = math.atan(GlobalRotation[1][0]/GlobalRotation[0][0])
    pitch = math.atan(-
GlobalRotation[2][0]/(math.sqrt(GlobalRotation[2][1]+GlobalRota
tion[2][2])))
    roll = math.atan(GlobalRotation[2][1]/GlobalRotation[2][2])

    #Update Data and Publish - Quaternions not implemented
    # print(BotinGlobal)
    # print('')
    # print('globalrotation')
    # print(GlobalRotation)
    GlobalMessage = GlobalPos()
    GlobalMessage.Xpos = BotinGlobal[0][3]
    GlobalMessage.Ypos = BotinGlobal[1][3]
    GlobalMessage.Zpos = BotinGlobal[2][3]

    GlobalMessage.Yaw = yaw
    GlobalMessage.Pitch = pitch
    GlobalMessage.Roll = roll
    #not implemented
    GlobalMessage.QuatW = 0
    GlobalMessage.QuatX = 0
    GlobalMessage.QuatY = 0
    GlobalMessage.QuatZ = 0

    pub.publish(GlobalMessage)

def testcall(data):
    print(3)

if __name__ == '__main__':
    #initialize ROS node
    rospy.init_node('BotGlobalPosition')
    pub =
rospy.Publisher('/GlobalPosQuat',GlobalPos,queue_size=5)

    #while rosnode is active, do stuff
    while not rospy.is_shutdown():
```

```
        #subscribe to 2 needed topics
        camera_sub =
message_filters.Subscriber('/LEDPositions',LEDPoints)
        botquat_sub =
message_filters.Subscriber('/BotRelLoc',BotPositionQuat)

        ts =
message_filters.ApproximateTimeSynchronizer([camera_sub,
botquat_sub], 10, 0.5, allow_headerless = True)
        ts.registerCallback(localtoglobal)
        # rospy.Subscriber("LEDPositions",LEDPoints,testcall)
        rospy.spin()
```

Appendix F:
Camera Rectification – Python

```
#!/usr/bin/env python

import cv2
import numpy as np
import os
import glob
import csv

# Defining the dimensions of checkerboard
CHECKERBOARD = (6,8)
square_size = 24.4 #mm
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
30, 0.001)

# Creating vector to store vectors of 3D points for each
checkerboard image
objpoints = []
# Creating vector to store vectors of 2D points for each
checkerboard image
imgpoints = []


# Defining the world coordinates for 3D points
objp = np.zeros((1, CHECKERBOARD[0]*CHECKERBOARD[1], 3),
np.float32)
objp[0,:,:2] = np.mgrid[0:CHECKERBOARD[0],
0:CHECKERBOARD[1]].T.reshape(-1, 2)
objp = objp * square_size
prev_img_shape = None

# Extracting path of individual image stored in a given
directory
images =
glob.glob('/home/ryan/catkin_ws/src/beacon_localization/scripts
/images/CameraRect/*.JPG')
numimages = len(images)
print("found {} images".format(numimages))
for image in images:
    #scale imge to something more manageable
    img = cv2.imread(image)
    if numimages == len(images):
```

```
        scale_percent = 50
        width = int(img.shape[1]*scale_percent/100)
        height = int(img.shape[0]*scale_percent/100)
        dim = (width,height)
    img = cv2.resize(img,dim,interpolation = cv2.INTER_AREA)



    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # Find the chess board corners
    # If desired number of corners are found in the image then
ret = true
    ret, corners = cv2.findChessboardCorners(gray,
CHECKERBOARD, None)

    if ret == True:
        objpoints.append(objp)
        # refining pixel coordinates for given 2d points.
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-
1),criteria)

        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, CHECKERBOARD,
corners2,ret)

    cv2.imshow('img',img)
    cv2.waitKey(5)
    print("remaining images : {}".format(numimages))
    numimages = numimages - 1
print('image capture complete')
cv2.destroyAllWindows()

h,w = img.shape[:2]

# Performing camera calibration

print('working:calibrateCamera')
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
imgpoints, gray.shape[::-1],None,None)

print("Camera matrix : \n")
print(mtx)
print("dist : \n")
print(dist)
print("rvecs : \n")
```

```
print(rvecs)
print("tvecs : \n")
print(tvecs)

#print to files

with open("CameraMatrix.csv","w+") as camera_csv:
    csvWriter = csv.writer(camera_csv,delimiter=',')
    csvWriter.writerows(mtx)

with open("distarray.csv","w+") as dist_csv:
    csvWriter = csv.writer(dist_csv,delimiter=',')
    csvWriter.writerows(dist)

with open("rvecs.csv","w+") as rvecs_csv:
    csvWriter = csv.writer(rvecs_csv,delimiter=',')
    csvWriter.writerows(rvecs)

with open("tvecs.csv","w+") as tvecs_csv:
    csvWriter = csv.writer(tvecs_csv,delimiter=',')
    csvWriter.writerows(tvecs)
```