5-2017

Bird's Eye View: Cooperative Exploration by UGV and UAV

Shannon Hood
University of South Carolina

Bird's Eye View: Cooperative Exploration by UGV and UAV

by

Shannon Hood

Bachelor of Science
University of South Carolina 2016

Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2017

Accepted by:

Ioannis Rekleitis, Director of Thesis

Marco Valtorta, Reader

Jason O'Kane, Reader

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

Acknowledgments

To begin with, I would like to thank all the people who helped and supported me in writing this thesis. First, I would like to thank my major advisor, Dr. Ioannis Rekleitis, for his help and advice. I am very grateful to my boyfriend, Jacob Cox, and to my family for their never-ending support. I would also like to thank the University of South Carolina, and acknowledge support for this work from a Google Faculty Research Award and the National Science Foundation grants (NSF 1513203, 1637876, 1526862).

Abstract

This paper proposes a solution to the problem of cooperative exploration using an Unmanned Ground Vehicle (UGV) and an Unmanned Aerial Vehicle (UAV). More specifically, the UGV navigates through the free space, and the UAV provides enhanced situational awareness via its higher vantage point. The motivating application is search and rescue in a damaged building. A camera atop the UGV is used to track a fiducial tag on the underside of the UAV, allowing the UAV to maintain a fixed pose relative to the UGV. Furthermore, the UAV uses its front facing camera to provide a birds-eye-view to the remote operator, allowing for observation beyond obstacles that obscure the UGV's sensors. The proposed approach has been tested using a TurtleBot 2 equipped with a Hokuyo laser ranger finder and a Parrot Bebop 2. Experimental results demonstrate the feasibility of this approach. This work is based on several open source packages and the generated code will be available online.

Table of Contents

List of Figures

Chapter 1

Introduction

With the proliferation of unmanned air vehicles (UAVs) a new viewpoint, a birds-eye-view, has become available in many domains, allowing structure inspection, environmental monitoring, virtual tourism, and search and rescue. The majority of aerial vehicles are severely limited in the sensing payload they can carry due to weight constraints. As such, simple tasks such as obstacle avoidance and localization in an unknown environment become rather challenging. On the other hand, ground robots, while extremely capable in navigating safely inside an unknown environment, are restricted to a limited field of view. Deploying a robot team comprised of a ground and an aerial vehicle enables an enhanced field of view while maintaining safe navigation and localization. Recent advances in robotics have made collaboration among a heterogeneous team of robots possible [39]. At the same time, many interesting problems associated with multi-robot collaborative exploration are being discussed, such as coordination, communication protocols, bandwidth constraints, etc. [24, 42]. One motivating application is search and rescue in a damaged building, where the UGV navigates on the ground while the UAV maintains a fixed pose above the UGV employing a robot-tracker sensor consisting of a camera and fiducial marker. The robot team could search and map dangerous areas, monitored by a remote operator at a safe distance. [1]

The goal of this thesis is to present a collaborative exploration approach utiliz-

[1] Early parts of this work have been submitted jointly with Kelly Benson, Daniel Madison, Patrick Hamod, Jason O'Kane, and Ioannis Rekleitis to the 2017 International Conference on Unmanned Aircraft Systems.

ing a UGV and a UAV. The UGV, a Kobuki TurtleBot 2 equipped with a Hokuyo laser range finder and an upwards-facing camera, is responsible for guiding the UAV through the corridors of a building; see Fig. 1.1. The UAV, a Parrot Bebop 2 equipped with a forward facing camera, provides enhanced situational awareness to a remote expert; see Fig. 1.2. The UAV does not process all data and commands on-board. Instead, utilizing a WiFi connection, the sensor data are transmitted to a ground station where all the processing takes place. Currently we utilize two computers: one on the UGV and a separate one as a ground control station (GCS). A unique fiducial tag placed on the bottom of the UAV is tracked by a wide angle camera on top of the UGV, and the relative pose between the two vehicles is estimated.

Using cooperative localization, the pose of the UAV is calculated using the TurtleBot's pose and the relative pose of the two robots. This calculated pose is compared to the pose provided by ORB-SLAM2 [29], a monocular SLAM package run using data provided by the UAV's camera. These two localization techniques improve the accuracy of UAV's estimated pose, allowing for easier path planning and tracking of the tag.

Related work is discussed next. Chapter 3 outlines the proposed methodology. Experimental results and factors affecting the performance are discussed in Chapter 4. The thesis concludes with a discussion of lessons learned and future directions.

Figure 1.1 The Kobuki TurtleBot 2 used in this thesis. Also seen are the Hokuyo laser, TurtleBot camera, and router used.

Figure 1.2 The Parrot Bebop 2 used in this thesis.

Figure 1.3 A fiducial tag attached to the bottom of a Parrot Bebop 2.

Chapter 2

Related Works

Cooperative robotics has been explored as early as the 90's, such as when Cao et al. [4] discussed possible future applications and the open problem of collaboration between autonomous robots. Yuta and Premvuti [43] discussed how to organize multiple autonomous robots. Several previous experiments have examined the best software for tag tracking, autonomous communication between robots, or have used similar setups involving both a ground and an aerial based robot. Collaboration between UGVs and UAVs has been proposed for a variety of fields. For example, in the detection and fighting of wildfires [32], in measuring the amount of nitrogen in the soil for agricultural applications [41], for surveillance [17], and for detecting and disposing of mines [10]. UAVs have also been used to monitor traffic [20].

2.1 Robot Collaboration

Chaimowicz and Kumar [5] discussed ground robot and aerial robot localization in their 2007 paper. Their research focused on environments with limited GPS access, and on using GPS in conjunction with other sensors. Similar concerns also drove the work of Frietsch, Meister, Schlaile, and Trommer [15]. However, unlike these two previous works, the approach proposed by this thesis assumes no GPS access, instead relying on cooperative localization [36] between the two vehicles as well as pose estimates calculated from the laser scan and cameras.

Another area of interest is collaboration between sea-based robots and aerial robots; see Murphy et al. [30] and Shkurti et. al. [39]. In their paper titled "Multi-

domain monitoring of marine environments using a heterogeneous robot team," Shkurti et. al. used underwater vehicles, airboats, and aerial vehicles to autonomously collect footage of underwater scenes. In Murphy's experiment, the robots were used to survey storm damage from Hurricane Wilma. It was also "the first known use of unmanned sea surface vehicles (USVs) for emergency response" [30].

MacArthur and Crane [25] investigated using a UAV in the state estimation of a UGV. In their paper, they discuss camera calibration and extracting pose from camera data, as well as the coordinate transforms needed. Unlike with this project, GPS was integrated with pose information. MacArthur and Crane found that localization of a UGV using a passive UAV was successful in both simulated and experimental runs.

Cooperation between UAVs and UGVs for target detection and formation holding was proposed by Tanner [40]. Duan and Liu [9] discuss an overview of multiple vehicles collaborating with a focus on military applications. Papachristos and Tzes [31] proposed a tethered solution to address the limited battery time for longer operations. Autonomous landing on a UGV charging station was proposed by Rezelj and Skocaj [37].

2.2 Tag Tracking

Other projects on similar topics have focused more on landing the drone on a target rather than following the target for cooperative exploration with a ground vehicle. One example is the research by Minhua and Jiangtao, which involved landing an AR.Drone 2.0 on an augmented reality tag that designated a landing pad [28]. Minhua and Jiangtao tried two different methods of tracking the landing pad. The first was a QR code, which they found gave the UAV more information about the landing area, but at the cost of more processing time [28]. This caused delays and inaccuracy while tracking the QR code. The second method they tried was tracking augmented reality tags through a package called `ar_track_alvar`, which they found processed the tag

faster. This faster computation is possible because `ar_track_alvar` is initialized with the size of the tag to help it better determine the position of the tag relative to the frame given. In this experiment, Minhua and Jiantao modified the front camera of the drone to point towards the ground for better tag tracking because the front camera has a higher resolution [28]. They also experimented with using `ar_track_alvar` in conjunction with `tum_ardrone`, a wrapper for PTAM (Parallel Tracking and Mapping) and PID controller for the drone, when they were landing the AR.Drone on a tag. They found that using `ar_track_alvar` to find the tag and `tum_ardrone` for the PID controller to move the UAV into position of the tag landing zone worked fairly well [28].

Figure 2.1 An example of a QR code is shown to the left, while two fiducial tags generated by ar_track_alvar are shown on the right.

Another example of previous work related to autonomous flight is the experiment by Rezelj and Skocaj focused on landing a modified AR.Drone on a TurtleBot with an augmented reality tag, so that the drone could charge and then take off again by making use of the ground vehicle's longer battery life [37].

Although a fiducial tag will be used as a marker for the UGV in this project, it is important to point out that other methods of tracking are available, such as tracking a colored object or a 3-D moving target, such as in Chakrabarty's experiment with an AR.Drone [6].

Lawrence and Turchina [23] evaluated different approaches to fiducial tag tracking. In their experiments on pose estimating software packages, they found that `ar_track_alvar` was the best package for estimating the pose of a tag from the frame of a camera. Packages they tried include: `ar_sys`, `ar_pose`, `visp_auto_tracker`, and `ar_track_alvar`. However, not all of them work with ROS and some require a wrapper. Due to time constraints, and the fact that `ar_pose` was not available in ROS, they did not test it extensively. It is also important to note that the camera they used to test the packages had a resolution of 640x480, while our tracking camera has a resolution of 1920x1080. The higher resolution of the Bird's Eye View camera allows for better tag tracking, since the tag can move farther distances and still remain within view of the camera. The next tested package, `ar_sys`, gave a range of 30cm to 120cm at 30hz. Unfortunately, with our camera's resolution, this would require the UAV to fly too close to the UGV. The next package, `visp_auto_tracker`, had a range of 60cm to 300cm at 30hz. One major drawback of this package was that if the UAV lost sight of the tag, it needed to re-establish at the minimum distance to continue tracking. Thus with this package, any time the tag was lost the UAV would have to maneuver to a precise location to reinitialize the tracking. Lastly, Lawrence and Turchina tested `ar_track_alvar`, which ran from 30cm to 300cm at 10hz. This package had the best range for establishing tracking and is what this project will use. One problem of tag tracking software, in addition to range limitations and reinitializing the package when the tag is lost, is blurring of the tag due to motion. This makes detecting the tag more difficult, and is discussed in detail in "A motion blur resilient fiducial for quad-copter imaging" by Meghshyam Prasad et al. [33].

One problem with `ar_track_alvar` is determining what to do if the tag moves out of view of the camera. If the tag is not visible, the package has no way of tracking the UGV. Huang et al. proposed a solution to this in their paper "An Object Following Method Based on Computational Geometry and PTAM for UAV

in Unknown Environments" [18]. Their solution was to use a Kalman filter in order to predict the velocity of the tag, so the tag can be found and tracking reinitialized.

2.3 Monocular SLAM

Monocular Simultaneous Localization and Mapping relies on computer vision techniques to calculate the pose of the robot and map the environment. In this thesis, monocular SLAM is performed on the UAV's front camera for state estimation. Real-time monocular SLAM was once thought impossible due to high computational costs and was first implemented in the Parallel Tracking and Mapping (PTAM) package. Now, there exists a great variety of monocular SLAM packages, including LSD-SLAM [11], SVO [13], and RatSLAM [27]. Many open-source vision-based SLAM packages were compared in the 2016 paper by Li et. al. [12]. ORB-SLAM was found to be one of the better performing packages, and is used in this project.

PTAM was originally made by Klein and Murray in 2007 [21]. Using the information of the UAV's camera combined with the IMU, it is able to find "features," or the edges of detected obstacles, in order to estimate distance traveled as well as scale of distances of objects from the UAV. PTAM has also been used in the collaboration of multiple UAVs. Bazen et al.'s 2016 experiment [2] involved coordinating a fleet of quad-rotors using `ardrone_autonomy` and `tum_ardrone`. They found that, while the completed missions were more accurate when PTAM was used, the missions were successful less frequently. They theorized this was due to delay "between acquisition of information by the drones and their actual displacement based on this information" [2].

Another later monocular SLAM package, ORB-SLAM [29], improves upon the work by Klein and Murray. ORB-SLAM uses ORB features, which are binary features that are invariant to rotation and scale. Unlike PTAM, ORB-SLAM is able to initialize with no user input, which is necessary for autonomous robot systems. ORB-

SLAM allows for more accurate mapping by performing loop closure, where loops are detected and the ends merged to remove drift over time. Additionally, the number of keyframes used by ORB-SLAM when matching features increases depending on the complexity of the image, unlike in PTAM where keyframes increase over time. This is done by "culling" keyframes that are no longer used, and allows for ORB-SLAM to run for longer periods of time.

2.4 Cooperative Localization

Cooperative localization (CL) deals with estimating the pose of teams of mobile robots using sensor data. This is done in order to provide enhanced localization compared to the robots' localization capabilities without cooperation. In order for this to be done, a transformation matrix between two robots must be calculated. Cooperative localization was first introduced by Kurazume et. al. [22] in 1994. At the time it was known as "cooperative positioning." In this experiment, robots were separated into two different teams: one team moving, while the other team remains a stationary landmark for the other team. The teams then switch roles until all robots reach their target position.

The term "cooperative localization" first appeared in 1998 in works by Bison et. al. and Rekleitis et. al. [3, 36]. Since its conception, CL has been applied to many different problem types in both 2- and 3-D, including vision-based [26, 14], sonar-based [38, 34], and range-based [35, 16]. Two types of cooperative localization techniques especially relevant to this thesis are those dealing with vision-based and 3-D problems. In 2009, Bahr et. al. used cooperative localization for the localization of autonomous underwater vehicles. The goal of this work was to "create a fully mobile network of AUVs [Autonomous Underwater Vehicles] that perform acoustic ranging and data exchange with one another to achieve cooperative positioning for extended duration missions over large areas" [1].

Early work was done on vision-based CL by Jennings, Murray, and Little [19]. In this work, two stereo vision-based mobile robots explore and map their environment. One robot finds landmarks in the environment that the second robot then uses to localize itself relative to the first robot's reference frame. This allows the robots to collaborate on tasks using a common local reference frame without a complete map of the environment. In 2011, Chang et. al. [7] worked with humanoid robots in the RoboCup environment to localize and track moving objects using vision-based cooperative localization. First, they estimated the robot pose by modeling the uncertainty of motion commands and measurements. Then when other robots were within the field of view, state estimates were refined using the estimated pose and distances calculated based on the image.

Chapter 3

Approach

This chapter describes the necessary parts of the Bird's Eye View system and how they must act together in order to achieve the goals set out above. The robots must be able to communicate with each other and with a Ground Control Station. One of the robots must detect the fiducial tag on the other and extract the coordinate transformation between the UAV and the UGV. The UAV has to be able to hover above the UGV while tag-tracking is maintained. The UGV must autonomously explore a hallway, and finally, the UAV must be able to detect the pose of the UGV and move accordingly while maintaining tag tracking.

In the development of the UAV/UGV system, several open source ROS packages were used: `bebop_autonomy`, as a driver for the Bebop and joystick; `usb_cam`, a driver for the Logitech HD Pro Webcam C910; and ORB-SLAM 2, a monocular SLAM package. Package documentation such as the setup and use of these packages is described in more detail in Appendix A.

3.1 Robot Team Network

Communication between the UAV, UGV, and a Ground Control Station is enabled by a network connecting the 3 systems. The Bird's Eye View project requires two PCs: a Ground Control Station (GCS) and a TurtleBot PC. The TurtleBot PC is connected to the TurtleBot and the Hokuyo laser scanner via USB. A portable router and power source (for the router) are also placed on the TurtleBot. The specific router used in this project is the TP-Link 150Mbps Wireless-N Nano Pocket Router. The TurtleBot

PC was connected to this router using an Ethernet cable. The Ground Control Station was then connected wirelessly via WiFi to the router. These two connections allow the TurtleBot PC and GCS to communicate with each other. Since the TurtleBot PC can be controlled wirelessly using the GCS, the GCS can now be used to control both the Bebop and the TurtleBot. The GCS must be kept within range of the portable router in order for both robots to operate correctly. However, the TurtleBot PC will always be within range of the router. All components of the Bird's Eye View network are shown in the diagram below; see Fig 3.1.

Figure 3.1 Shows the network connections between the ground Control Station (GCS), the TurtleBot, and the Bebop. Connections are shown in blue.

3.2 Movement of the UGV

Two different motion strategies were used for the UGV. First, in order to challenge the tracking capabilities of the UAV, the UGV is placed inside an enclosed area and a random walk is executed. The UGV moves forward until it reaches near an obstacle, then performs an in-place rotation of 180 degrees plus a small random value, then moves again in a straight line. An example of random walk motion can be seen in Figure 3.2. The UAV was able to follow the UGV throughout these movements, even though in place rotations are more challenging.

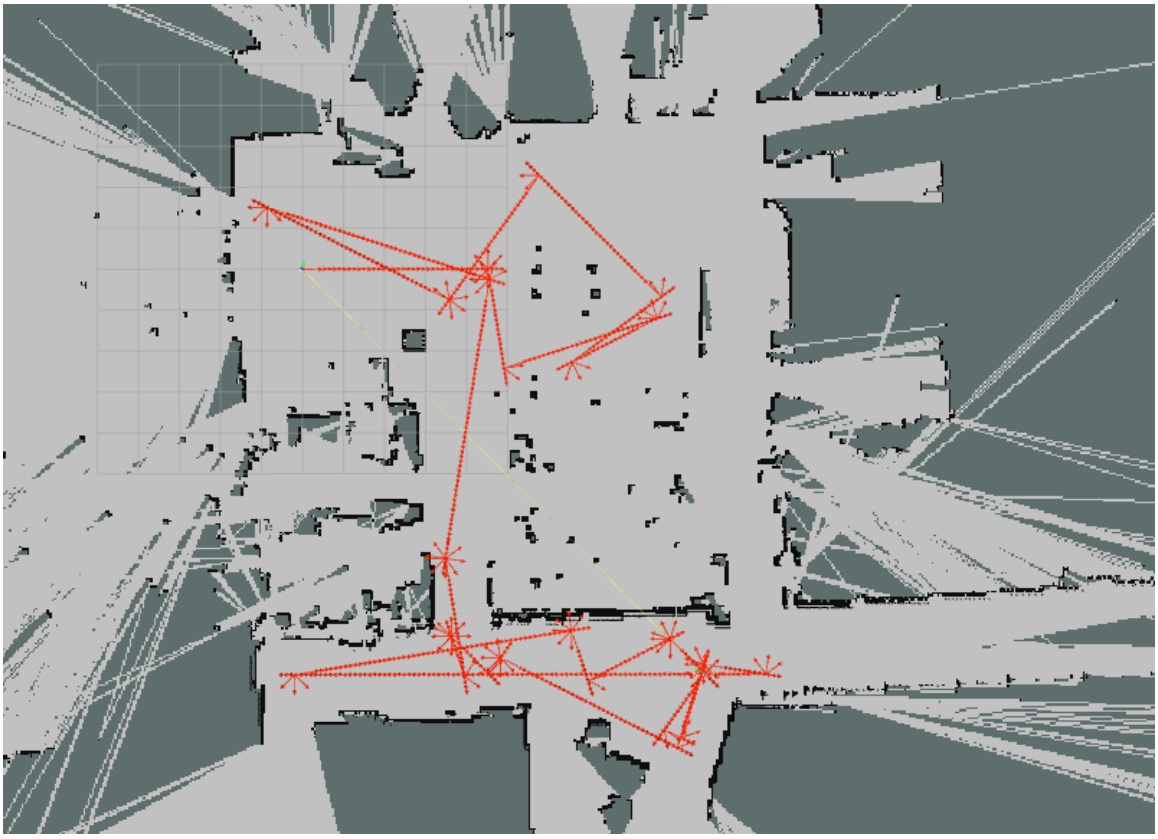

Figure 3.2 An example of a random walk motion algorithm showing laser scans and odometry data in rviz.

Second, the UGV uses a center-of-the corridor following algorithm employing a PID controller to navigate hallways. This algorithm compares the distances read by

the laser scanner 60 degrees off the center of the UGV. As the UGV moves forward, it also turns slightly favoring the side with the longer distance. This is the core behavior of the generalized Voronoi graph algorithm [8]. Under the GVG algorithm, the UGV begins by moving away from the closest obstacle, and then moves following the equidistant from two obstacles loci. Once the UGV is equidistant to more than two obstacles, its location is defined as a meetpoint. The GVG is defined as the graph where edges represent equidistant to two obstacles points, and vertices represent points where three or more edges meet. Every new meetpoint has at least three edges. If all the edges of a meetpoint are explored then the meetpoint is marked as explored. The algorithm follows edges until all meetpoints are marked as explored.

Communication between the two robots improves the UGV's movement algorithm. The UGV does not start moving until the UAV is able to see the fiducial tag. This allows for a much simpler initialization process. Furthermore, should the UAV lose sight of the tag for more than two seconds, the UGV halts its motion and waits for the UGV to find the tag again. Should the UGV be unable to find the tag again, it is possible for an operator to step in and move the UAV back into position using the joystick controller.

3.3 Tag Detection & Movement of the UAV

When first launched by a trained operator, the UAV must be moved into position so the fiducial tag is above the TurtleBot's camera using a Logitech F710 joystick and the `bebop_autonomy` package. The joystick allows for precise control in moving the UAV to a position where it is able to recognize the tag. The joystick also provides a safety feature to control the UAV should loss of control occur during the flight. Once in position, the UAV then autonomously follows the UGV utilizing the fiducial tag. Early work employed the Parrot AR.Drone 2.0 and the tag was mounted on the UGV; see Fig. 3.3. This previous approach is discussed further in Section 3.6.

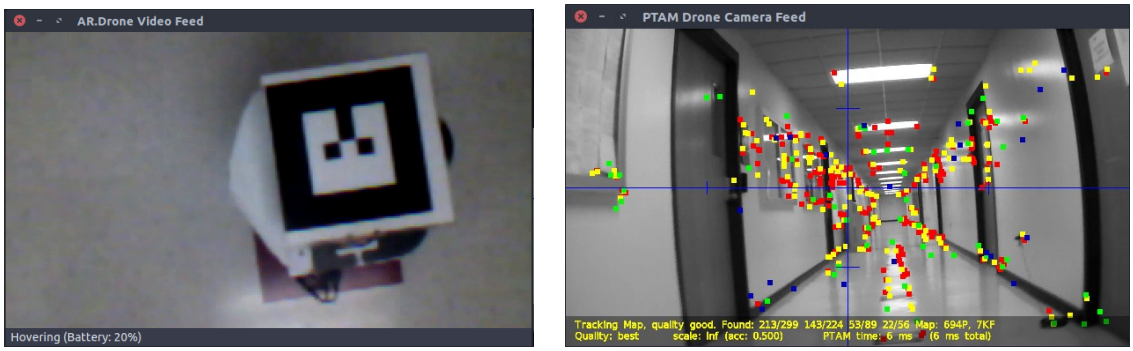

Figure 3.3 Early UAV camera feeds: In the left image, the bottom camera observes the tag placed on top of the UGV. The right image shows the front camera view with PTAM detected features overlayed.

The Bird's Eye View system uses tags generated by the `ar_track_alvar` ROS package. When a tag is within view of the UGV's upward-facing camera, the estimated pose of the tag is published to the `ar_pose_marker` ROS topic; see Fig. 3.4.

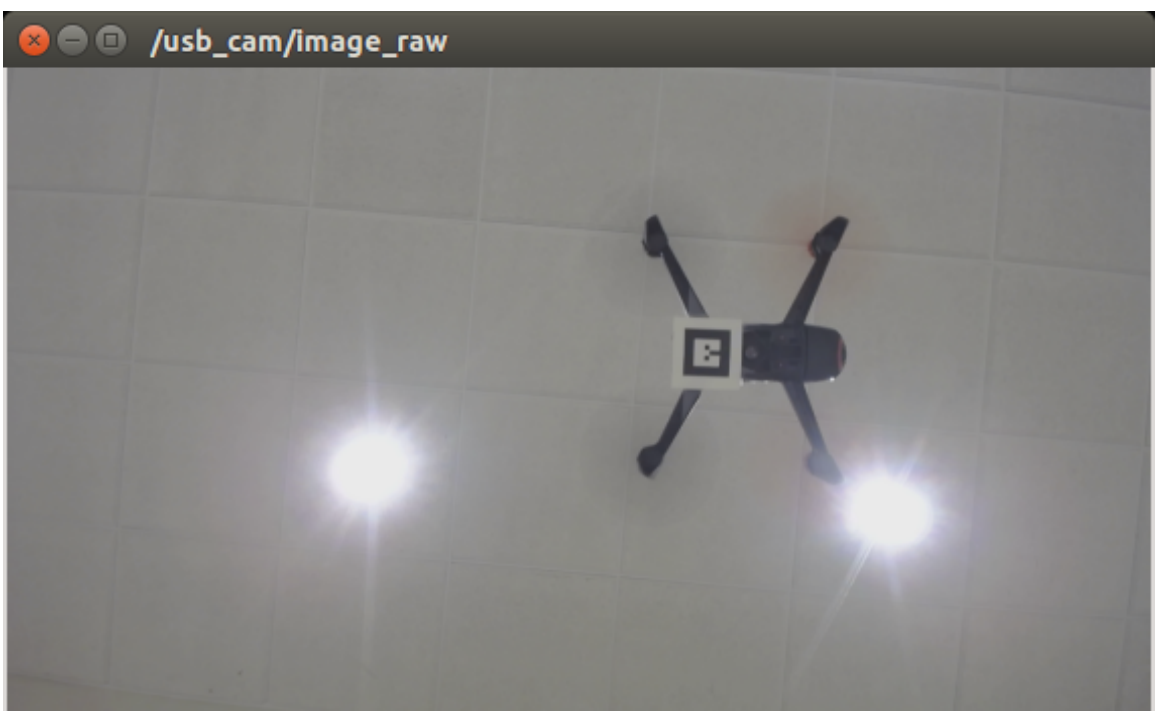

Figure 3.4 Bebop 2 flying over the UGV with a tag.

The drone then moves so that the tag remains directly over the camera. This is done by transforming the tag's pose to x, y, z, and yaw values relative to the UAV's frame of reference. To maintain tracking, the tag must remain within the camera's field of view. A 4.6 cm tag was found to be most visible from a height of 0.65 to 0.95 meters. As a result, a constant height value of 0.8 is added to the negative z dimension. Next, the x- and y-velocities of the tag are calculated using the current state of the tag, the previous state, and the time difference between the two states. The current pose of the tag and the tag velocities are used to calculate the speed that UAV should fly in order to maintain sight of the tag. The UAV's velocities are calculated as follows:

$$UAV.angular.z = tag.yaw * c$$

$$UAV.linear.z = tag.z * c$$

$$UAV.linear.x = tag.x * d + tag.vx$$

$$UAV.linear.y = tag.y * d + tag.vy$$

In these equations, c and d are constant scaling factors. These factors are necessary because the `bebop_autonomy` driver expects velocities between -1 and 1 inclusive. Best results were obtained using $c = 0.4$ and $d = 0.07$. The velocities do not take into account any information generated by ORB-SLAM, as it is much less reliable than cooperative localization.

3.4 Cooperative Localization

The Bird's Eye View package performs the cooperative localization of the UAV using the pose of the UGV provided by the ROS topic `odom` and pose of the tag provided by `ar_pose_marker`. First, the odometry data is transformed from the base_link frame to the map frame. This transform converts the local pose at the rotational center of the robot to the fixed world frame. The map frame should not drift over

time and is generated by the `gmapping` package using odometry data. This transform is necessary to ensure that cooperative localization done with respect to the UGV's pose can be shown in the world map frame. The relationship between the camera's pose and the robot's base_link is always fixed. The camera is located approximately 0.45 meters above the TurtleBot's base. This distance is taken into account when calculating the UAV's pose. Next, the relationship between the fiducial tag's position

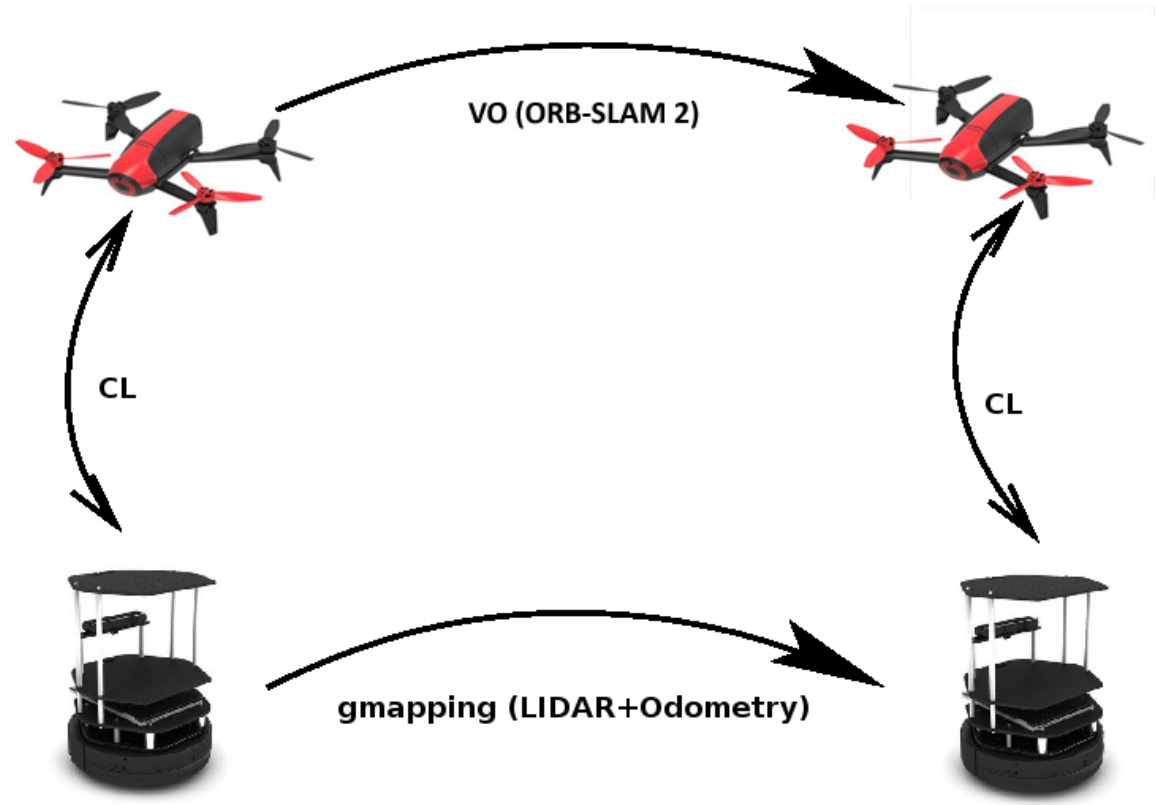

Figure 3.5 Shows the relationships between different odometry and localization methods used.

and the TurtleBot's camera is obtained using the data provided by `ar_track_alvar`. This pose along with the difference between the height of the camera and height of the base_link is added to the UGV's pose. The resulting pose is the UAV's current position in the world frame as estimated using cooperative localization. Since the UAV's and UGV's poses are both with respect to the world frame, they are easily

plotted in rviz alongside the laser scan data. The two principal methods used in the cooperative localization package are available in Appendix B. Figure 3.5 shows how the different odometry and SLAM data relate to one another.

3.5 UAV Localization Using ORB-SLAM

Figure 3.6 Shows the Bebop's front camera view with ORB-SLAM features overlayed in green.

In addition to being calculated through cooperative localization, the UAV's pose is also computed by ORB-SLAM 2. ORB-SLAM detects features in the UAV's camera feed and uses the change in location of these features to determine the pose of the camera. These features can be seen in Fig. 3.6; new features are added every time a new keyframe is taken. Because monocular ORB-SLAM does not use odometry data from the UAV or have a baseline for the size of its surroundings, it has no way of knowing how to scale the poses it calculates. As a result, it uses an arbitrary

scale. In order to compare ORB-SLAM poses to the poses calculated by `gmapping` and cooperative localization, the scale of the ORB-SLAM data must be adjusted to match. The data is translated to the start position in the world frame and then multiplied by a scaling factor.

In order for ORB-SLAM to be used during the experimental runs, ORB-SLAM 2 must be modified to publish the calculated camera pose to a rostopic. This is done by getting the matrix returned by the ORB-SLAM method `TrackMonocular`, converting it to an odometry message by separating out the rotation and translation values, and then publishing this odometry message.

3.6 Rejected Approach

Originally this project was attempted using an AR.Drone 2.0 rather than a Bebop. The ROS package `tum_ardrone` was used as a PID controller and monocular SLAM (Simultaneous Localization and Mapping) package. In this iteration, the tag was placed on the UGV while the UAV used its bottom-facing camera to track the tag's motion. However, this UAV was unable to use the front and bottom cameras simultaneously. To deal with this issue, a workaround using a camera switching node to toggle between the front and bottom cameras approximately once a second was implemented. If the cameras swapped too fast, the topics would become corrupted; too slow and the UAV would be unable to track the UGV. The ROS package `joy` was used for joystick control.

However, this approach did not succeed. PTAM was not able to accurately estimate the UAV's pose with such limited camera frames, resulting in large amounts of drift. Additionally, the bottom-facing camera on the AR.Drone had a lower resolution, meaning if the UGV moved at a rate greater than 0.2 meters per second, the UGV could move out of view of the UAV in the time it took the cameras to switch. This would lead to a loss of tracking. Furthermore, the switching delay oc-

casionally caused corrupted images. The Parrot AR.Drone 2.0 was very unstable in position keeping, and it drifted easily away from the target due to a less advanced ROS driver. In particular, the AR.Drone had difficulty rotating while maintaining its position. This drift was compounded with a loss of feature tracking by PTAM, which does not perform well during rotations because all known features are lost if rotation occurs too quickly.

Another problem encountered was that the PID controller used by `tum_ardrone` was not intended to be used with a moving target such as a TurtleBot. The controller would not advance to new target destinations until the previous destination had been reached. When combined with the constant drift of the AR.Drone, this meant that the target destination could not be updated quickly enough. By the time the AR.Drone accounted for it's drift and made it to the first target, the TurtleBot would have moved out of the field of view of the AR.Drone's downward-facing camera.

Chapter 4

Results

In order to evaluate the system's performance, several criteria must be taken into account: how well the UGV navigates the environment; how well the UAV tracks and follows the UGV; and whether the above tasks are able to be completed autonomously. Experimental data was collected by having the robots explore the hallways of the Swearingen Engineering Center and a bounded arena in multiple trials. After the UAV is positioned within the camera's view, no other assistance is provided by the operator except in the case of an emergency.

The Bird's Eye View project can be evaluated by comparing the poses of the UAV and UGV to determine how well the UAV succeeds in following the UGV. The pose of both the UAV and UGV are visualized in rviz, and the pose provided by cooperative localization is compared to the pose calculated by ORB-SLAM. Finally, the video from the TurtleBot's camera is be examined to determine if any issues in tag detection or following are present.

Figure 4.1 presents the paths of the two robots for an early experiment using the Parrot Ar.Drone 2.0 UAV. While the UGV moves smoothly along the corridor, the UAV has a much less direct trajectory as it drifts away repeatedly, and the controller brings it back above the UGV. Additionally, the ending position of the UAV was several meters before the end position of the UGV. This is because PTAM would lose feature detection frequently because of the low frame rate. Whenever feature detection and tracking was lost, the pose of the UAV could not be updated. When feature detection resumed, all of the motion that occurred while tracking was down

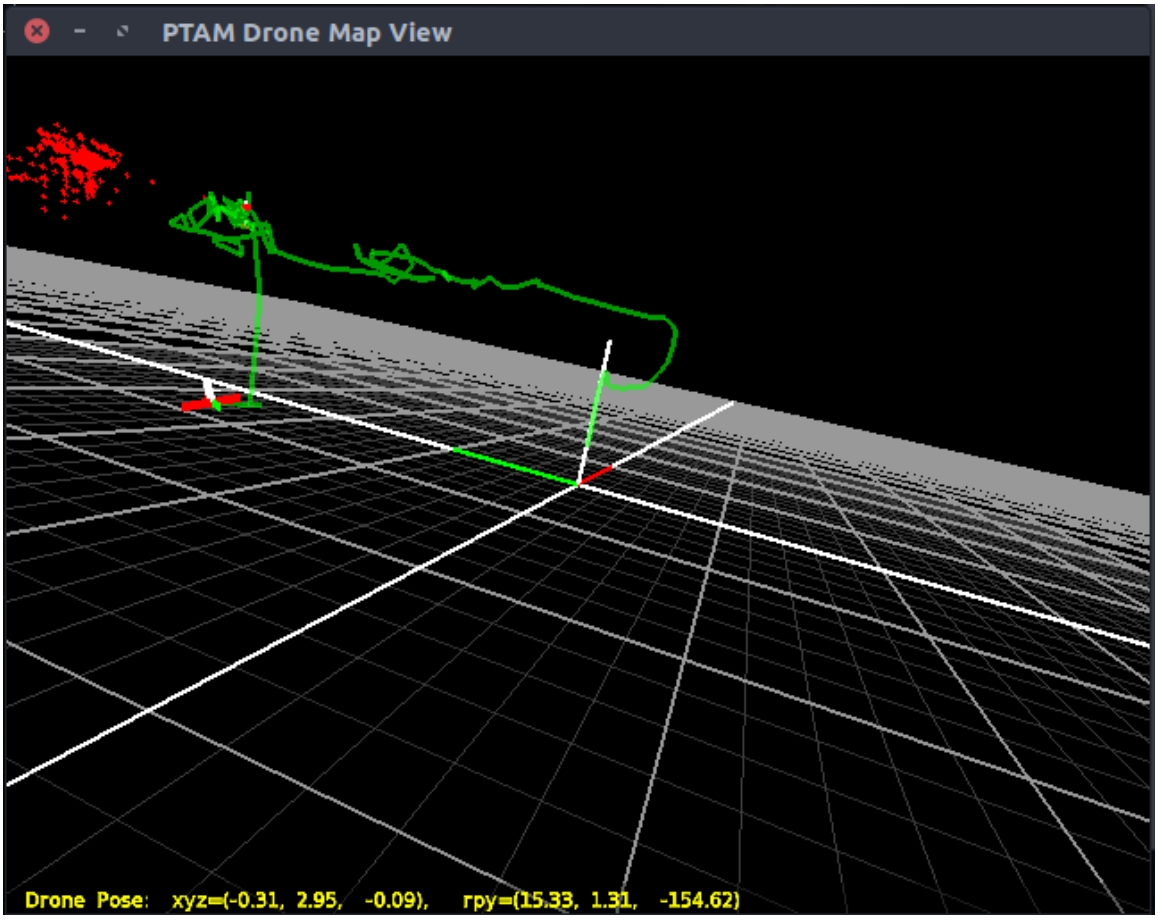

Figure 4.1 Early Experiments with the AR.Drone 2.0. Shows a 2-dimensional map of the environment and path taken by the robots.

was lost.

Figure 4.2 shows the more stable performance of the Parrot Bebop 2. Similarly to Fig. 4.1, in this trial run the UGV traveled along the corridor on Swearingen's first floor. This trial used the center-of-the-corridor algorithm. As can be seen in Fig.4.2, the UAV very closely followed the UGV as it navigated the hallway. The tag remained within view as the robots rotated, and tracking was maintained. No operator assistance was provided during this run. This experiment ended when the Bebop's battery died. Since the Bebop is only capable of 25-30 minutes of flight, care must be taken to recharge the battery after every flight. One solution to extending the

life of the Bird's Eye View system can be seen in the work of Rezelj and Skocaj [37]. The UAV could land on the UGV when the battery was low and recharge before resuming its exploration.

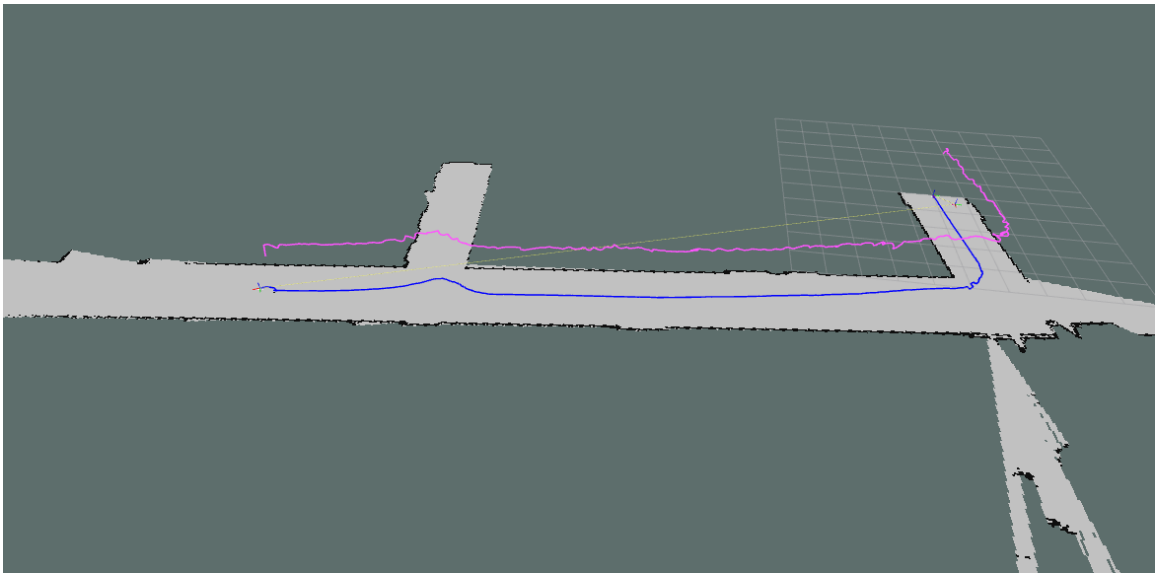

Figure 4.2 Traversing down the corridor with the walls mapped. The UGV trajectory is shown in blue and the UAV trajectory, above, in pink.

Figure 4.3 presents a more challenging scenario where the UGV performed a random walk inside a bounded arena. The UAV followed along even when the UGV performed a 180 degree rotation. Since the Bebop is prone to drift when rotating, maintaining track of the tag is more difficult during pure rotation than translation with rotation. As result, more noise can be seen in the path of the UAV than in the previous center-of-the-corridor experiment. In addition, the rotations must be done slowly in order for ORB-SLAM to be able to maintain feature detection. Despite this, the UAV was still capable of following the UGV as it performed these rotations.

Figure 4.4(a) shows the Pangolin GUI generated by ORB-SLAM. It shows the feature points and keyframes used in calculating the pose of the UAV. This pose is given in a world reference frame. However, since ORB-SLAM does not have any knowledge of the scale of the system, it is not the same world coordinate frame used

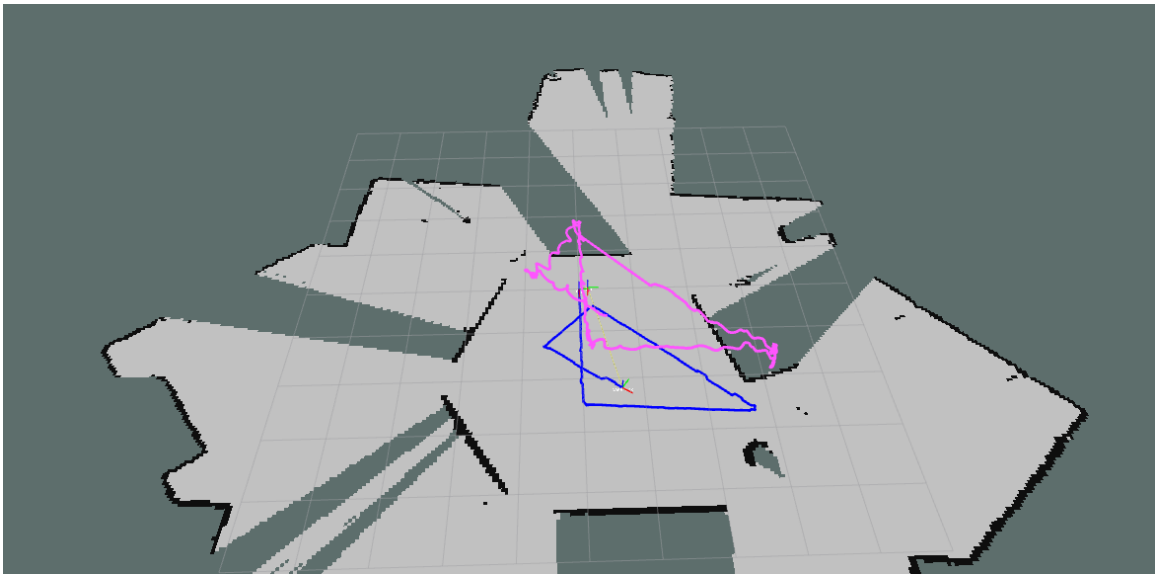

Figure 4.3 Random walk experiment with the walls mapped. The UGV trajectory is shown in blue and the UAV trajectory, above, in pink.

by the UGV and cooperative localization. In order to be compared directly to those poses, the ORB-SLAM data must be scaled accordingly. Figure 4.4(b) shows this data scaled and plotted next to the CL data and UGV's trajectory. As can be seen in this figure, the ORB-SLAM data is not as accurate as the CL data. The trajectory drifts upward over time and is not corrected. However, the x- and y-dimensions of the ORB-SLAM trajectory are very similar to those found using cooperative localization. This scaled pose could be used as an estimate for the UAV's world frame pose when the tag is not visible by the system. Using the UGV's global pose and ORB-SLAM, the UAV could attempt to find the UGV without the help of a trained operator if tracking is lost.

Figure 4.5 shows a long trial run down the 3rd floor of Swearingen. This experiment ran for 808 seconds, or 13 minutes. At a speed of .15 meters/second, the UAV and UGV covered 120 meters. For increased distance, the speed of the UGV could be increased. However, if the speed is too high, the UAV will be unable to follow the tag. This is because at high speeds the UAV will move out of view of the

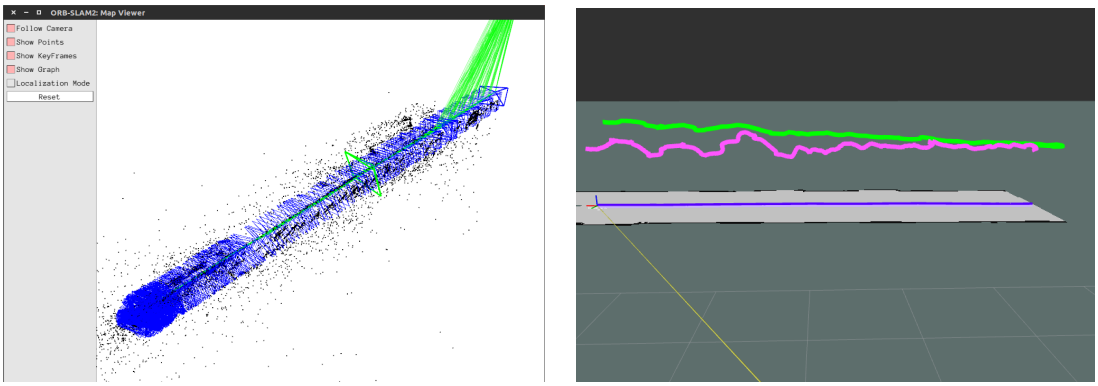

Figure 4.4 Center-of-the-corridor ORB-SLAM experiments: (a) The left image shows a map of features and keyframes generated by ORB-SLAM. (b) In the right image, the UGV trajectory is shown in blue, the UAV trajectory as calculated by CL in pink, and the UAV trajectory according to ORB-SLAM in green.

UGV's camera. During this experiment, the operator intervened one time when the UAV flew directly under an A/C vent and was blown off course. The length of this experiment shows that the UAV is able to follow the UGV for long distances without drifting off course. In all of the experimental data collected, the pose of the UAV

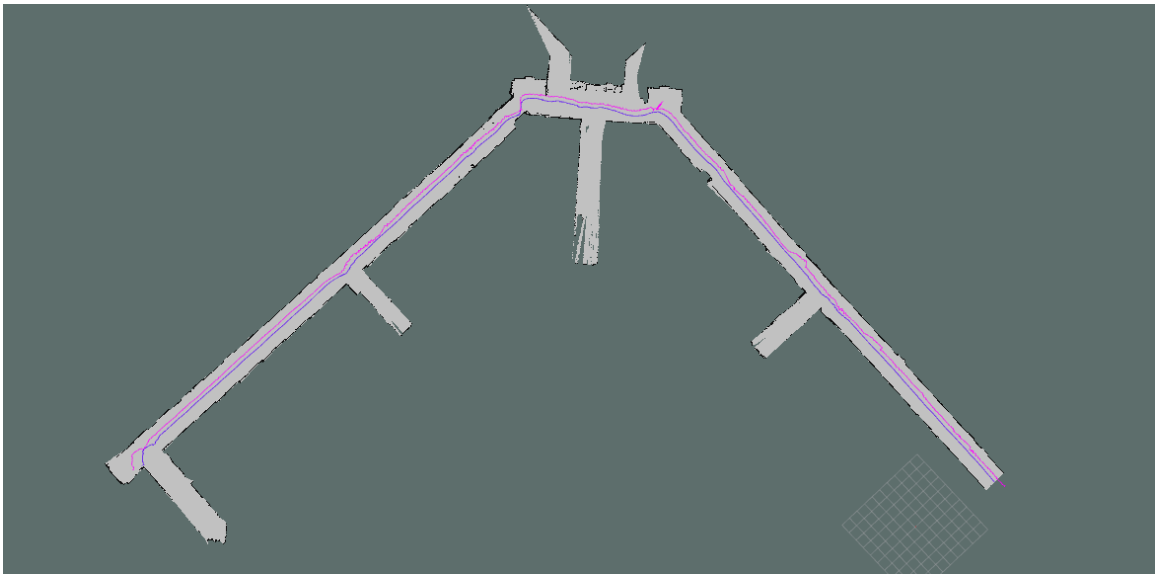

Figure 4.5 Traversing Swearingen's 3rd floor with the walls mapped. The UGV trajectory is shown in blue and the UAV trajectory, above, in pink.

and UGV are very similar, excluding the difference in height above the ground. The UGV successfully navigates the corridors of the Swearingen building, staying in the center of the hallway, and the maps produced by rviz and ORB-SLAM accurately show the robots' paths through the hallways. As all of these objectives are met, one can say the two robots are able to co-operatively and autonomously explore and map an unfamiliar environment.

Chapter 5

Conclusion

This thesis aims to determine the abilities of a UAV and UGV team to autonomously search, map, and explore an indoor environment. The two vehicles remain in contact with each other visually and over a WiFi network. Cooperative localization of the UAV using the UGV's pose and relative pose between the two is combined with the UAV's pose as calculated by ORB-SLAM for more accurate localization. The framework developed in this thesis is for application in search and rescue.

The results show that the UGV is able to autonomously navigate the environment and that the UAV is able to follow the UGV using vision-based techniques for long periods of time. The pose of the UAV is calculated both using cooperative localization and the monocular SLAM package ORB-SLAM for enhanced localization. These poses are plotted in rviz alongside the TurtleBot's pose as calculated from the laser data to show the viability of this approach.

Future work not covered in this thesis could include more robust communication over the WiFi network. For example, if the UAV loses track of the tag, once the UGV has stopped moving due to the lost tag signal it receives, the UGV could send its current pose data to the UAV. Once this data has been transformed to the UAV's coordinate frame, the UAV could then fly to the location of the UGV using ORB-SLAM to localize. Once the tag is back within sight, the UAV would notify the UGV and the UGV would continue on navigating the environment.

Another way to improve upon this project would be expanding ORB-SLAM to perform 3-D mapping of an area using both the UAV's front camera and UGV's laser

scanner. Currently the maps provided by the laser rangefinder and ORB-SLAM are kept separate, but could be combined for increased accuracy.

Bibliography

[1] A Bahr, JJ Leonard, and MF Fallon. Cooperative localization for autonomous underwater vehicles. *Robotics Research*, 28(6):714—-728, 2009.

[2] M Bazin, I Bouguir, D Combe, V Germain, and G Lassade. Parallel tracking and mapping of a fleet of quad-rotor. *International Journal of Engineering and Technology*, 9(3), 2017.

[3] P Bison, G Chemello, C Sossai, and G Trainito. Cooperative Localization using Possibilistic Sensor Fusion. In *IFAC Symposium on Intelligent Autonomous Vehicles*, pages 621–626, Madrid, 1998.

[4] Y Uny Cao, Alex S Fukunaga, and Andrew Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27, 1997.

[5] Luiz Chaimowicz and Vijay Kumar. Aerial shepherds: Coordination among uavs and swarms of robots. In *Distributed Autonomous Robotic Systems 6*, pages 243–252. Springer, 2007.

[6] Anjan Chakrabarty, Robert Morris, Xavier Bouyssounouse, and Rusty Hunt. Autonomous indoor object tracking with the parrot ar. drone. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 25–30. IEEE, 2016.

[7] CH Chang, SC Wang, and CC Wang. Vision-based cooperative simultaneous localization and tracking. *IEEE International Conference on Robotics and Automation*, 2011.

[8] Howie Choset and Joel Burdick. Sensor based planning. i. the generalized voronoi graph. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on Intelligent Robots and Systems*, volume 2, pages 1649–1655. IEEE, 1995.

[9] HaiBin Duan and SenQi Liu. Unmanned air/ground vehicles heterogeneous cooperative techniques: Current status and prospects. *Science China Technological Sciences*, 53(5):1349–1355, 2010.

[10] D. MacArthur E. MacArthur and C. Crane. Use of cooperative unmanned air and ground vehicles for detection and disposal of mines. In *Proc. SPIE*, volume 5999, page 9, 2005. Intelligent Systems in Design and Manufacturing VI.

[11] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[12] A. Quattrini Li et. al. Experimental comparison of open source vision based state estimation algorithms. In *International Symposium on Experimental Robotics (ISER)*, 2016.

[13] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.

[14] D Fox, W Burgard, H Kruppa, and S Thrun. Efficient multi-robot localization based on monte carlo approximation. *International Symposium of Robotics Research*, 1999.

[15] N Frietsch, O Meister, C Schlaile, and GF Trommer. Teaming of an ugv with a vtol-uav in urban environments. In *2008 IEEE/ION Position, Location and Navigation Symposium*, pages 1278–1285. IEEE, 2008.

[16] M Gilioli and KJ Schilling. Autonomous cooperative localization of mobile robots based on ranging systems. *SPIE Unmanned Ground Vehicle Technology*, 5083, 2003.

[17] Ben Grocholsky, James Keller, Vijay Kumar, and George Pappas. Cooperative air and ground surveillance. *IEEE Robotics & Automation Magazine*, 13(3):16–25, 2006.

[18] Yuxi Huanga, Ming Lva, Dan Xionga, Shaowu Yangb, and Huimin Lu. An object following method based on computational geometry and ptam for uav in unknown environments. *IEEE International Conference on Information and Automation*, 2016.

[19] C Jennings, D Murray, and JJ Little. Cooperative robot localization with vision-based mapping. In *IEEE International Conference on Robotics and Automation*, pages 2659 —-2665, 1999.

[20] Konstantinos Kanistras, Goncalo Martins, Matthew J Rutherford, and Kimon P Valavanis. A survey of unmanned aerial vehicles (uavs) for traffic monitoring. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2013.

[21] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE, 2009.

[22] R Kurazume, S Nagata, and S Hirose. Cooperative positioning with multiple robots. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1250–1257, 1994.

[23] A. Lawrence and A. Turchina. Tag tracking in ros. `https://github.com/ablarry91/ros-tag-tracking`, 2014.

[24] Dunhao Liu, Xun Li, and Wei Liu. The research about collaboration techniques for aerial and ground mobile robots. *Proceedings of Science*, 2015.

[25] D. MacArthur and C. Crane. Unmanned ground vehicle state estimation using an unmanned air vehicle. In *2007 International Symposium on Computational Intelligence in Robotics and Automation*, pages 473–478, June 2007.

[26] L Merino, F Caballero, and P Forssen. *Advances in Unmanned Aerial Vehicles: Single and multi-UAV relative position estimation based on natural landmarks.* 2007.

[27] Michael J Milford, Gordon F Wyeth, and David Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 403–408. IEEE, 2004.

[28] C. Minhua and G. Jiangtao. Mobile robotics lab spring 2015 automatic landing on a moving target. `https://robotics.shanghaitech.edu.cn/sites/default/files/files/robot_land_project2_report.pdf`, 2015.

[29] Raul Mur-Artal, JMM Montiel, and Juan D Tardós. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[30] Robin R Murphy, Eric Steimle, Chandler Griffin, Charlie Cullins, Mike Hall, and Kevin Pratt. Cooperative use of unmanned sea surface and micro aerial vehicles at hurricane wilma. *Journal of Field Robotics*, 25(3):164–180, 2008.

[31] Christos Papachristos and Anthony Tzes. The power-tethered uav-ugv team: A collaborative strategy for navigation in partially-mapped environments. In *22nd Mediterranean Conference of Control and Automation (MED*, pages 1153–1158. IEEE, 2014.

[32] C. Phan and H. H. T. Liu. A cooperative uav/ugv platform for wildfire detection and fighting. In *System Simulation and Scientific Computing*, pages 494–498, Oct 2008. Asia Simulation Conference - 7th International Conference.

[33] Meghshyam G Prasad, Sharat Chandran, and Michael S Brown. A motion blur resilient fiducial for quadcopter imaging. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 254–261. IEEE, 2015.

[34] I M Rekleitis, G Dudek, and E Milios. Probabilistic Cooperative Localization in Practice. In *IEEE International Conference on Robotics and Automation*, pages 1907–1912, 2003.

[35] Ioannis Rekleitis, Gregory Dudek, and Evangelos Milios. Multi-Robot Exploration of an Unknown Environment, Efficiently Reducing the Odometry Error. In *International Joint Conference on A.I. (see also CIM Tech. Report TR-CIM-97-01)*, 1997.

[36] Ioannis Rekleitis, Gregory Dudek, and Evangelos E. Milios. On multiagent exploration. In *Proceedings of Vision Interface 1998*, pages 455–461, Vancouver, Canada, June 1998.

[37] A. Rezelj and D. Skocaj. Autonomous charging of a quadcopter on a mobile platform. In *Austrian Robotics Workshop*, pages 65–66, 2015.

[38] F Rivard and J Bisson. Ultrasonic relative positioning for multi-robot systems. *IEEE International Conference on Robotics and Automation*, 2008.

[39] F. Shkurti, A. Xu, M. Meghjani, J. C. Gamboa Higuera, y. Girdhar, P. Giguere, B. B. Dey, J. Li, A. Kalmbach, C. Prahacs, K. Turgeon, I. Rekleitis, and G. Dudek. Multi-domain monitoring of marine environments using a heterogeneous robot team. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1447–1753, Portugal, Oct. 2012.

[40] H. G. Tanner. Switched uav-ugv cooperation scheme for target detection. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3457–3462, April 2007.

[41] Pratap Tokekar, Joshua Vander Hook, David Mulla, and Volkan Isler. Sensor planning for a symbiotic uav and ugv system for precision agriculture. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5321–5326. IEEE, 2013.

[42] Steven L Waslander. Unmanned aerial and ground vehicle teams: Recent work and open problems. In *Autonomous control systems and vehicles*, pages 21–36. Springer, 2013.

[43] S Yuta and Suparerk Premvuti. Coordinating autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots. In *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1566–1574. IEEE, 1992.

Appendix A

Platform Documentation

A.1 Software Installation

The Bird's Eye View project depends upon a functioning Kobuki TurtleBot 2 with a Hokuyo laser scanner, a Parrot Bebop 2, a portable router, and a Logitech joystick controller. A functioning ROS installation is also needed. The software needed in order for all these parts to work together is specified below, along with installation instructions.

A.1.1 ROS

A complete guide to installing and using ROS, the robot operating system, can be found in the ROS wiki at `http://wiki.ros.org/ROS/Installation`. This project uses ROS Indigo on an Ubuntu 14.04 machine.

A.1.2 Kobuki TurtleBot 2

The software needed for the TurtleBot 2 and its installation is described in detail in the platform documentation by Adem Coskun. Instructions are also available in the ROS wiki at `http://wiki.ros.org/Robots/TurtleBot`. These instructions assume a Microsoft Kinect is used as a sensor; however, the TurtleBot used for this project instead uses a Hokuyo 20m laser scanner.

This laser scanner requires the ROS package `urg_node` to be installed. This can be done by running the command

```
$ sudo apt-get install ros-indigo-urg-node
```

on an Ubuntu machine. Before running this node, ROS must be granted access to the laser scanner. To do this, run

```
$ sudo chmod a+rw /dev/ttyACM0
```

This package is used instead of `hokyo_node` because this model laser scanner connects using an Ethernet port, which is not supported by `hokuyo_node`. If using a Hokuyo laser that connects via USB, run

```
$ sudo apt-get install ros-indigo-hokuyo-node
```

A.1.3 Parrot Bebop 2

The Bebop requires multiple packages in order to navigate, to map its surroundings and track the position of the robot, to track a fiducial tag, and to switch between the front and bottom cameras. The packages required in order for the robot to perform all tasks necessary for this project are: `bebop_autonomy`; `orb_slam`; `ar_track_alvar`; `usb_cam`; and the `ar_tag_following` package, which will later be available on the official Bird's Eye View GitHub. These packages depend upon OpenCV, which comes preinstalled with Ubuntu 14.04. However, it should be noted that OpenCV 3.0 will give compilation errors; instead, version 2.4.10 should be used. To check which version you have, run

```
$ dpkg -l | grep libopencv
```

Installation instructions for OpenCV 2.4.10 can be found at `http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html`.

Bebop Autonomy

The `bebop_autonomy` package should be installed first, as the packages described below depend upon it. `bebop_autonomy` is an open-source ROS driver for the Parrot

Bebop quad-copter (both 1 and 2). It can be installed by first running

```
$ sudo apt-get install build-essential python-rosdep
  python-catkin-tools
```

to insure these necessary packages are installed. Next, in the catkin workspace directory,

```
$ git clone https://github.com/AutonomyLab/bebop_autonomy.git
  src/bebop_autonomy
# Update rosdep database and install dependencies (including
# parrot_arsdk)
$ rosdep update
$ rosdep install --from-paths src -i
# Build the workspace
$ catkin build -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

should be run to install `bebop_autonomy` and to insure that all dependencies are installed. Documentation for the Bebop is available at `http://bebop-autonomy.readthedocs.io/en/latest/installation.html`.

ORB-SLAM 2

The ORB-SLAM 2 package is made up of three different threads: tracking, local mapping, and loop closure [29]. The package is available on GitHub. To begin, first make sure to install OpenCV, Eigen 3, and Pangolin, as ORB-SLAM depends on these packages.

After installing the above, ORB-SLAM 2 can be installed by running the following commands

```
$ git clone https://github.com/raulmur/ORB_SLAM2.git
  ORB_SLAM2
```

```
$ cd ORB_SLAM2
$ chmod +x build.sh
$ ./build.sh
```

in order to install this package. Then run

```
$ export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:PATH/
  ORB_SLAM2/Examples/ROS
$ chmod +x build_ros.sh
$ ./build_ros.sh
```

to add ORB-SLAM to the ROS_PACKAGE_PATH environment variable and build the package. Further documentation is available at `https://github.com/raulmur/ORB_SLAM2`. Sample data sets are available at `http://vision.in.tum.de/data/datasets/rgbd-dataset/download`.

Before using ORB-SLAM, a settings .yaml file must be created. This file contains camera parameters and ORB-SLAM parameters. An example .yaml file is provided in the ORB-SLAM source.

AR Track Alvar

The `ar_track_alvar` package is an open source AR tag tracking library. This package is capable of generating AR tags, as well as identifying and tracking the pose of individual AR tags. To install this package, tun

```
$ sudo apt-get install ros-indigo-ar-track-alvar
```

Pre-made fiducial tags can be downloaded from the `ar_track_alvar` ROS wiki page at `http://wiki.ros.org/ar_track_alvar`. Alternately, a tag with custom ID numbers, border widths, or sizes can be generated by running

```
$ rosrun ar_track_alvar createMarker
```

The size of the tag must then be updated in the launch file of your choice (e.g. `launch/pr2_indiv_no_kinect.launch`) by changing 4.4 in the line shown below to the size of the tag in cm.

```
<arg name="marker_size" default="4.4" />
```

A.1.4 Logitech F710 Joystick

This project requires the use of a joystick controller to move the Bebop into an initial position above the TurtleBot. To get the joystick up and running, first install the ROS `joy` package.

```
$ sudo apt-get install ros-indigo-joy
```

To identify your joystick, run

```
$ ls /dev/input
```

Take note of any jsX devices (where X is a number). Next, connect your joystick to the computer and retype the previous command. The new jsX device that appears is your controller.

Now that you've identified the controller, to test that the installation is successful run

```
$ sudo chmod a+r /dev/input/jsX
$ jstest /dev/input/jsX
```

A.2 Robot Initialization

Now that the required software is all installed and compiled, the robots can be initialized. Because this project heavily depends on the network connection between the two robots, initializing the networking will be discussed first, followed by a brief explanation of initializing the Bebop and TurtleBot.

A.2.1 Networking

The Bird's Eye View project requires two PCs, a Ground Control Station (GCS) and a TurtleBot PC. The TurtleBot PC is placed on the TurtleBot and is connected to the TurtleBot and Hokuyo laser scanner via Ethernet. A portable router and power source (for the router) are also placed on the TurtleBot. The specific router used in this project is the TP-Link 150Mbps Wireless-N Nano Pocket Router. The TurtleBot PC should be connected to this router using an Ethernet cable. The Ground Control Station should then connect wirelessly via WiFi to the router. These two connections allow the TurtleBot PC and GCS to communicate with each other.

Next the TurtleBot PC should connect to the Bebop's on-board network via WiFi. This connection allows the TurtleBot PC to communicate with the drone. Since the TurtleBot PC can be controlled wirelessly using the GCS, the GCS can now be used to control both the Bebop and the TurtleBot. The GCS must be kept within range of the portable router in order for both robots to operate correctly. All components of the Bird's Eye View network are shown in the diagram below [**??**]. The joystick controller should be connected to the TurtleBot PC.

To control both robots from the Ground Control Station,

```
$ ssh user@ipaddress
```

must be run on the GCS, where user is your username on the TurtleBot PC, and ipaddress is the IP address of the TurtleBot PC's wireless adapter. This can be found by running the command

```
$ ifconfig
```

and locating the IP address associated with the WiFi adapter.

Currently the GUI graphics provided by `bebop_autonomy` cannot be forwarded via SSH, so will only appear on the TurtleBot laptop. As a result, before initializing the Bebop, the command

```
$ export DISPLAY=:0
```

must be run.

A.2.2 Bebop and TurtleBot

To initialize the TurtleBot, first power the robot up using the power switch on the bottom right of the robot. Next plug the Hokuyo Ethernet and TurtleBot USB into ports on the TurtleBot laptop. Move the TurtleBot to its starting position. Then from the Ground Control station, which should be connected to the TurtleBot PC through SSH, run the following command:

```
$ roslaunch afrl_driver turtlebot-20-hokuyo.launch
```

This will initialize the TurtleBot and the Hokuyo laser.

To initialize the Bebop, first insert a battery and secure it in place. Then, if flying indoors, the robot's rotors can be protected with a guard. From the GCS, run

```
$ roslaunch ar_tag_following bebop.launch
```

after ensuring that the TurtleBot PC is connected to the Bebop's WiFi. This command initializes the joystick controller, the `ar_track_alvar` fiducial tag tracking, and the `bebop_autonomy` package. Alternatively,

```
$ roslaunch bebop_tools bebop_nodelet_iv.launch
$ roslaunch bebop_tools joy_teleop.launch
```

can be run. If it is not done automatically by the code controlling the drone, you must make sure the emergency lights are set to green, and that flat trim has been called. This can be done by using the joystick controller. To properly initialize ORB-SLAM, after first taking off the robot should fly up a meter and then back down a meter to allow ORB-SLAM to take keyframes. Both robots are now ready to begin exploring the surrounding area.

A.3 Controlling the Robots

The goal of this project is for the TurtleBot and the Bebop to work together autonomously. As a result, the two robots are primarily controlled through code which instructs the TurtleBot to explore and the Bebop to follow the TurtleBot around. This code can be run by typing

```
$ roslaunch afrl_driver driver.launch
$ rosrun ar_tag_tracking controller
```

The first line starts the exploration code on the TurtleBot, and the second starts tag following on the Bebop. Note that these commands must be run on the GCS in terminal with an open SSH session to the TurtleBot PC.

However, in order to get the Bebop into position above the TurtleBot, it can be controlled using the joystick controller. These controller layout is specified in the `bebop_tools/config/log710.launch` file (or a similar file). The code listed below can be modified to assign the buttons to their functions.

```
teleop:
  piloting:
    type: topic
    message_type: "geometry_msgs/Twist"
    topic_name: cmd_vel
    deadman_buttons: [7]
    axis_mappings:
        -
          axis: 3  # Right thumb stick (up/down)
          target: linear.x
          scale: 1.0
          offset: 0.0
```

```
-
axis: 2  # Right thumb stick (left/right)
target: linear.y
scale: 1.0
offset: 0.0
-
axis: 1  # Left thumb stick (up/down)
target: linear.z
scale: 1.0
offset: 0.0
-
axis: 0  # Left thumb stick (left/right)
target: angular.z
scale: 1.0
offset: 0.0
```

A deadman's switch can be set, preventing any velocity commands from being sent unless the button is held. By default this is the right trigger. To identify which button is which, run the command

```
$ sudo jstest /dev/input/jsX
```

where jsX is the device ID of the joystick identified previously [A.1.4].

Further details on controlling the TurtleBot can be found in the paper by Adem Coskun or on the ROS TurtleBot wiki at `http://wiki.ros.org/Robots/TurtleBot#turtlebot.2BAC8-Tutorials.2BAC8-indigo-1.Navigation`

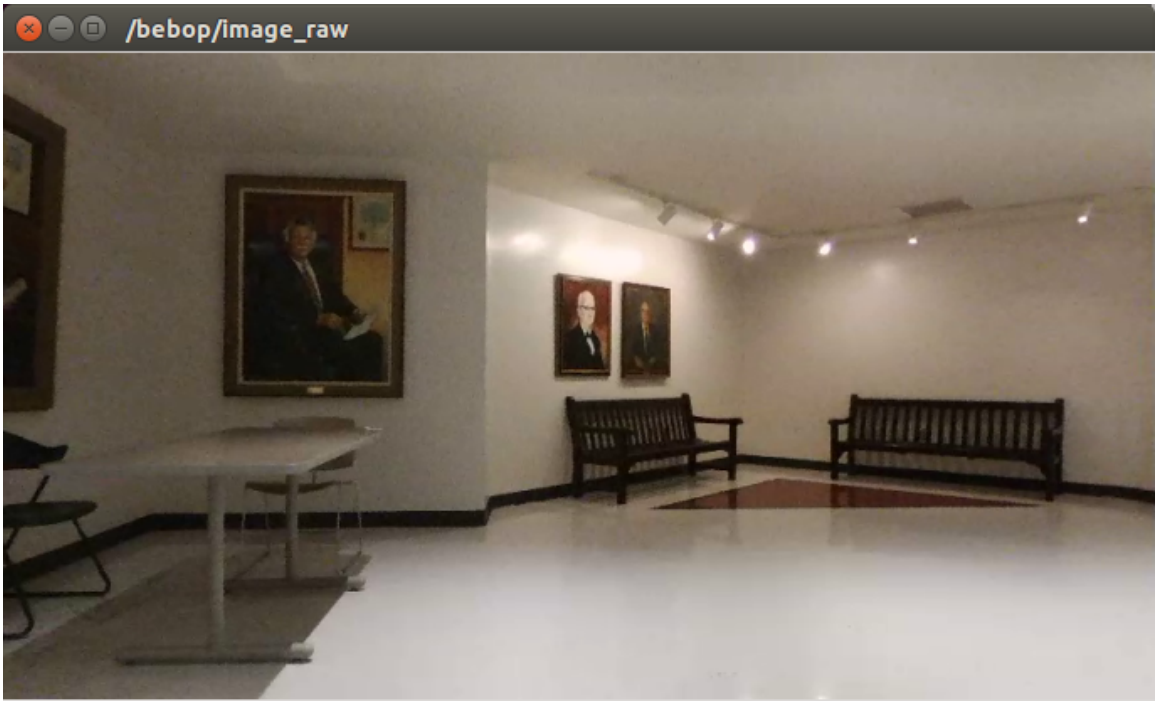

Figure A.1 Front camera view of the Parrot Bebop 2.

A.4 Sensors and Actuators

The TurtleBot 2 sensors and how to operate them are described in detail in the paper by Adem Coskun. The sensors and actuators of the Bebop and the Hokuyo laser scanner are described below.

A.4.1 Bebop Sensors

The drone comes with one camera installed. A 1080p front camera sensor with a recording up to 30fps; The drone also comes with a 3-axis gyroscope, a 3-axis accelerometer, and an ultrasound altimeter enhanced with an air pressure sensor.

The cameras first need to be calibrated. In order to do this, the ROS camera calibration package must be installed. If it is not, run

```
$ rosdep install camera_calibration
$ rosmake camera_calibration
```

The camera calibration files for all of the AFRL drones are available on the Bird's Eye View GitHub. The available rostopic for camera data published by the drone is `bebop/image_raw` To begin calibrating, run

```
$ rosrun camera_calibration cameracalibrator.py
  --size 8x6 --square 0.108 image:=/bebop/image_raw
  camera:=/camera
```

To get a good calibration, move the checkerboard around in the camera frame. More detailed calibration instructions are available at the following web page: `http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration`.

The `bebop_autonomy` package calculates and publishes Odometry data based on velocity estimates reported by the drone. The data is published to the `bebop/odom` rostopic as standard `nav_msgs/Odometry` messages. The corresponding TF transform is also published as odom -> base transformation in `/tf`.

IMU data including linear acceleration, angular velocity, and the orientation of the drone is published as a standard ROS `sensor_msgs/Imu` message. The necessary TFs are also published by `bebop_autonomy`.

A.4.2 Hokuyo Laser Scanner

The Hokuyo laser scanner on-board the TurtleBot currently has a maximum range of 20 meters. Before launching, the network must be configured to allow communication between the Hokuyo and the UGV. This can be done using the provided script file or set permanently using the Network Manager. The script file, `hokuyo_20.sh` is shown below

```
$ sudo ip addr flush dev eth1
$ sudo ip link set eth1 up
$ sudo ip addr add 192.168.1.253/24 dev eth1
$ sudo route del -net 192.168.1.0 gw 0.0.0.0 netmask
```

```
  255.255.255.0 dev eth1
$ sudo route add -host 192.168.1.20 dev eth1
$ route -n
```

To launch the Hokuyo laser scanner, place the following code in a launch file.

```
<node name="hokuyo" pkg="urg_node" type="urg_node" >
        <param name="frame_id" value="base_link" />
        <param name="ip_address" value="192.168.1.20" />
        <remap from="scan" to="base_scan" />
        <param name="angle_min" value="-1.57"/>
        <param name="angle_max" value="1.57"/>
</node>
```

The data collected by the scanner can then be viewed in rviz, or can be used with gmapping. Both the Hokuyo node and gmapping are launched by the `afrl_driver/turtlebot-20-hokuyo.launch` file.

A.5 Shutting Down

To shut down the project, close all commands currently running in the terminal of the Ground Control Station either by typing exit or by pressing CTRL+C. Also make sure to end all SSH sessions. Shut down both the GCS and the TurtleBot PC. To turn off the Bebop, remove the guard (if applicable) and unplug the battery. The battery should then be charged for later. To shut down the TurtleBot, unplug the Hokuyo Ethernet and the TurtleBot USB from the TurtleBot laptop. Then flick the switch located at the bottom right side of the TurtleBot. The TurtleBot should also be charged for later use. Unplug the power bank from the portable router, and if needed, charge the power bank.

Appendix B

Cooperative Localization Code

```
void tagCallback(const ar_track_alvar_msgs::AlvarMarkers::
                    ConstPtr& msg) {
  // Get tf from base_link to map
  listener.waitForTransform("map", "base_link", ros::Time(0),
  ros::Duration(7.0));
  try {
    listener.lookupTransform("map", "base_link",
    ros::Time(0), tfTransform);
  } catch(tf::TransformException &exception) {
    ROS_ERROR("%s", exception.what());
  }
  turtlePose.header.stamp = ros::Time::now();

  // Get TurtleBot position
  turtlePose.pose.position.x = tfTransform.getOrigin().x();
  turtlePose.pose.position.y = tfTransform.getOrigin().y();
  turtlePose.pose.position.z = tfTransform.getOrigin().z();
  turtlePose.pose.orientation.w=tfTransform.getRotation().w();
  turtlePose.pose.orientation.x=tfTransform.getRotation().x();
  turtlePose.pose.orientation.y=tfTransform.getRotation().y();
  turtlePose.pose.orientation.z=tfTransform.getRotation().z();
```

```
  // Add the Turtlebot's current position to its path
  turtlePath.poses.push_back(turtlePose);

  // If this is the first data recieved about the TurtleBot
  // save the header stamp, and publish the path
  if(!turtlePath.poses.empty()){
    turtlePath.header.stamp=turtlePath.poses[0].header.stamp;
  }
  turtlePathPub.publish(turtlePath);

  // If the UAV's tag is within view of the TurtleBot,
  // calculate the pose of the UAV using CL
  if(!msg->markers.empty()) {
    tagPose = msg->markers[0].pose;
    dronePose = getDronePose(turtlePose);
    dronePath.poses.push_back(dronePose);

    // If this is the first data recieved about the drone
    // save the header stamp, and publish the path
    if(!dronePath.poses.empty()){
      dronePath.header.stamp=dronePath.poses[0].header.stamp;
    }
    dronePathPub.publish(dronePath);
  }
}

// Method for computing the UAV's pose
```

```
geometry_msgs::PoseStamped getDronePose(geometry_msgs::
                                PoseStamped turtlePose) {
  //Set the drone's pose equal to that of the TurtleBot's
  geometry_msgs::PoseStamped dronePose;
  dronePose = turtlePose;

  //Add the pose relation between the TurtleBot and the
  // fiducial tag, accounting for the height of the TurtleBot
  dronePose.pose.position.x += tagPose.pose.position.x;
  dronePose.pose.position.y += tagPose.pose.position.y;
  dronePose.pose.position.z += tagPose.pose.position.z
                            + TURTLEBOT_HEIGHT;

  dronePose.pose.orientation.w += tagPose.pose.orientation.w;
  dronePose.pose.orientation.x += tagPose.pose.orientation.x;
  dronePose.pose.orientation.y += tagPose.pose.orientation.y;
  dronePose.pose.orientation.z += tagPose.pose.orientation.z;

  return dronePose;
}
```