8-2016

Autonomous Android: Autonomous 3D Environment Mapping with Android Controlled Multicopters

Tate Glick Hawkersmith
University of Tennessee, Knoxville, thawkers@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Tate Glick Hawkersmith entitled "Autonomous Android: Autonomous 3D Environment Mapping with Android Controlled Multicopters." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Lynne E. Parker, Major Professor

We have read this thesis and recommend its acceptance:

Syed K. Islam, Mongi A. Abidi

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Autonomous Android:
Autonomous 3D Environment Mapping with Android Controlled Multicopters

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Tate Glick Hawkersmith

August 2016

Acknowledgements

I would like to thank my advisor, Dr. Lynne E. Parker, who has challenged me from the classroom to the laboratory. Her guidance and passion for the field of robotics have been an invaluable influence on my continuing academic and professional success. I would also like to thank my friends, EECS faculty, and my fellow lab members in the Distributed Intelligence Laboratory for their support and guidance throughout this adventure. Finally, I would like to thank my parents and grandparents, without whom none of this would be possible.

Abstract

Autonomous robots are robotic platforms with a high degree of autonomy, programmed to perform various behaviors or tasks. They can either be semi-autonomous, only operable within the strict confines of their direct environment, or fully autonomous, capable of sensing and navigating their environments without any human interaction.

In this thesis, I focus on fully autonomous robotic platforms, specifically multicopters, controlled by an onboard Android-driven device, a widely available operating system for smartphones and tablets with over 1.4 billion active monthly users worldwide [Callaham 2015]. The main objective of this research is to create a plug and play solution for autonomous 3D aerial mapping using consumer off-the-shelf multicopters and an Android device. I begin with an overview of 3D mapping using a depth sensor and fully autonomous multicopters as separate entities and then discuss the process of combining them into a single, self-contained unit using a modified version of a computer vision technique called SLAM (Simultaneous Localization and Mapping). My modified SLAM uses an internal map of locations and altitudes already visited, as well as obstacles detected, by the onboard depth sensor while creating a 3D map of the environment.

While using a combination of a major phone and tablet operating system and an affordable consumer multicopter makes scanning the real-world into a digital form available to millions of people, there are currently still limitations of this autonomous platform. My Android-based autonomous aerial mapping application serves as a base for future research into more detailed aspects of autonomous mapping for multicopters, such as object detection, recognition, and avoidance. I conclude with an analysis of the

current applications of this technology with a more robust platform and the possible real-world applications for the future.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

We live in a digital world with the prevalence of new technological advances every day, but how can we capture more elements of the real-world in the virtual world? Building rich 3D maps of environments is an important task for mobile robotics, with applications in navigation, manipulation, semantic mapping, and telepresence [Henry, Krainin, Herbst, Ren, Fox 2014]. Robots are able to traverse and capture data throughout many different types of terrain, regardless of whether it is impassable or hazardous for a human operator. While this research can be applied to various applications of mobile robotic platforms, the initial purpose is to explore the options in multicopter platforms and onboard processors in order to provide millions of consumers the ability to scan portions of the physical world around them into the digital world.

Current multi-rotor platforms yield high barriers to entry, whether it is price or technological expertise, which dissuades consumers from experimenting with this new hobby. This thesis focuses on providing the average consumer the ability to purchase an affordable, ready-to-fly (RTF) multicopter and couple it with a compatible Android device without any technological knowhow. This opens up the possibility for any consumer to take a commercially available robot platform and a smartphone and automate the process of creating a 3D map for commercial and residential real-estate, environmental and natural disaster areas, hazardous environments, SWAT, police departments, and military applications [Loianno et al. 2015]. In this chapter I discuss the

challenges involved in solving the problem (Section 1.1), present my approach overview (Section 1.2), and summarize my results (Section 1.3).

1.1 Challenges

Creating a consumer ready, plug-and-play, autonomous 3D mapping multicopter doesn't come without its own unique challenges. Stable and precise control of an autonomous micro air vehicle (MAV) demands fast and accurate estimates of a vehicle's pose and velocity. In cluttered environments such as urban canyons, under a forest canopy, and indoor areas, knowledge of the 3D environment surrounding the vehicle is additionally required to plan collision-free trajectories [Bachrach et al. 2012]. In order to keep the platform plug-and-play for the average user, the Global Positioning System (GPS) must remain enabled even though navigation systems based on wirelessly transmitted information technologies are not typically useful in these scenarios due to limited range, precision, and reception [Bachrach et al. 2012].

In addition to stable and precise controls, the MAV must be able to carry and balance the onboard controller and sensors throughout its flight mapping. Stabilizing the payload can be a challenge itself because most affordable consumer multicopters only weigh 1000 g – 5 lbs and can only lift up to 800 g, causing even a slightly uneven weight distribution to destabilize the entire autonomous platform. While combining the onboard controller and sensors into a single Android device complements a plug-and-play approach and limits the number of payload components that need stabilizing, it also presents its own challenges. With only a single front facing sensor and a 120 degree Field of View (FOV), the multicopter only has a limited sense of the immediate environment ahead.

1.2 Approach Overview

To address the problem and challenges presented in this research, I propose an android controlled multicopter. My initial mobile robotic platform of choice for this application is a quadrotor, due to its mechanical simplicity and ease of control. Moreover, its ability to operate in confined spaces, hover at any given point in space, and perch or land on a flat surface makes it a very attractive aerial platform with tremendous potential [Loianno et al. 2015]. The goal is to create a 3D map of an unknown environment with no prior knowledge or information about the surroundings, using a fully autonomous multicopter controlled by an onboard Android device equipped with a depth sensor.

The approach is divided into two main components: (1) 3D modeling using the depth sensor on the Android device, a Google Tango Tablet; and (2) Full automation of the multicopter controls using the Android device. Component (1) includes creating an Android application which captures the 3D data points from the environment using the Google Tango's depth sensor and synchronizing each frame of the data with the position of the tablet, for reconstructing after completion, and saving it to memory. Component (2) involves creating an Android application to send automatic movement controls to the multicopter using the Pixhawk autopilot Java API. In order to reduce the error of the combined 3D mapping and autonomous flight platform, each component needs to be tested and refined independently of the other. The final step is to merge the components into a single Android application and refine the custom, SLAM based movement algorithm using the depth sensor data and create an internal movement map of visited, unmapped, and blocked locations.

1.3 Contributions and Lessons Learned

The fundamental contributions and lessons learned in this research include:

- Indoor aerial navigation requires hypersensitive movement controls and multiple sensors, excluding GPS, to minimize motion drift.
- Data-rich RGB-D images are extremely helpful in detecting loop closure, identifying objects, such as windows and doors, and avoiding obstacles.
- Commercially available flight controllers are currently robust enough to allow autonomous flight.
- The major limitations currently preventing a viable consumer off-the-shelf autonomous 3D multicopter mapping solution are a low minimum payload and minimal battery life.
- There are realistic real-world applications of this autonomous mapping technology including: commercial and residential surveying, environmental and natural disaster areas, hazardous environments, SWAT, police departments, and military applications.
- I provide a base autonomous flight control Android application, which is compatible with any multicopter using the Pixhawk autopilot controller and any smartphone or tablet running the Google Tango software, setting a foundation for more complex visual odometry concepts.

1.4 Organization of the Thesis

The remainder of this thesis is structured as follows. Chapter 2 introduces related work in RGB-D mapping, visual odometry, Simultaneous Localization and Mapping (SLAM), and automated flight enabled by consumer electronics. Chapter 3 describes the hardware, software, and implementation method of the ready-to-fly autonomous platform, both as separate entities and a combined unit. In Chapter 4 I discuss the

results, modifications to the autonomous platform, and different potential directions for a more refined autonomous “multi-mapper”. I conclude the thesis with Chapter 5 and summarize the possible future applications in consumer, commercial, and military mapping.

Chapter 2

Related Work

Research on visual odometry and mapping for autonomous aerial robotics has seen dramatic advances over the last decade. The same drop in price-performance ratio of processors and sensors has fueled the development of micro unmanned aerial vehicles (UAVs) that are between 0.1 and 1 m in length and 0.1-2 kg in mass [Loianno et al. 2015]. This chapter examines visual odometry, 3D modeling with respect to autonomous flight control, and prior work on multicopter mapping.

2.1 RGB-D Cameras

2.1.1 RGB-D 3D Mapping

RGB-D cameras are sensing systems that capture RGB images along with per-pixel depth information. RGB-D cameras rely on either active stereo or time-of-flight sensing to generate depth estimates at a large number of pixels [Henry, Krainin, Herbst, Ren, Fox 2014].

Most 3D mapping systems contain three main components (Figure 2.1): first, the special alignment of consecutive data frames; second, the detection of loop closures; third, the globally consistent alignment of the complete data sequence. While 3D point clouds are extremely well suited for frame-to-frame alignment (Figure 2.2) and for dense 3D reconstruction, they ignore valuable information contained in images. Color

cameras, on the other hand, capture rich visual information and are becoming more and more the sensor of choice for loop closure detection [Henry, Krainin, Herbst, Ren, Fox 2014]. The rich visual information contained in these images can be essential to making the correct calculations for the next position to move to, or to detect an obstacle the multicopter needs to avoid.

The mapping component of my research uses a RGB-D camera to capture only the depth information from the environment, align each frame in synchronization, and recreate dense 3D point clouds.

[Henry, Krainin, Herbst, Ren, Fox 2014]

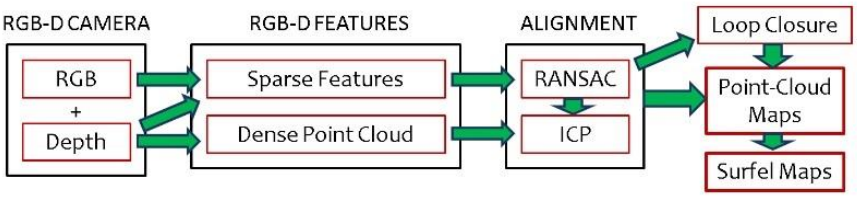

Figure 2.1. Overview of RGB-D Mapping. The algorithm uses both sparse visual features and dense point clouds for frame-to-frame alignment and loop closure detection. The surfel (surface element) representation is updated incrementally.

[Henry, Krainin, Herbst, Ren, Fox 2014]

Figure 2.2. Example frame for RGB-D frame alignment. Left, the locations of Scale Invariant Feature Transform (SIFT) features in the image. Right, the same SIFT features shown in their position in the point cloud.

2.1.2 Visual Odometry

Estimating a vehicle's 3D motion from sensor data typically consists of estimating its relative motion at each time step by aligning successive sensor measurements such as laser scans or RGB-D frames, a process most often known as "visual odometry" when comparing camera or RGB-D images [Bachrach et al. 2012]. Visual odometry aims at recovering only the trajectory of a vehicle. Nevertheless, it is not uncommon to see results showing also the 3D of the environment which is usually a simple triangulation of the feature points from the estimated camera pose. An example visual odometry result using a single omnidirectional camera is shown in Figure 2.3 [Siegwart, Nourbakhsh, Scaramuzza 2011 187].

All visual odometry algorithms suffer from motion drift due to the integration of the relative displacements between consecutive poses which unavoidably accumulates errors over time. This drift becomes evident usually after a few hundred meters. But the results may vary depending on the abundance of features in the environment, the resolution of the cameras, the presence of moving objects like people or other passing vehicles, and the illumination conditions [Siegwart, Nourbakhsh, Scaramuzza 2011 187].

Visual odometry concepts can be applied to autonomous flight controls for multicopters in indoor environments since GPS coverage is severely limited and motion drift is a major contributing factor in error accumulation. My research uses visual odometry to calculate the movement and orientation changes and adjusting for motion drift by using the data captured from successive RGB-D camera frames.

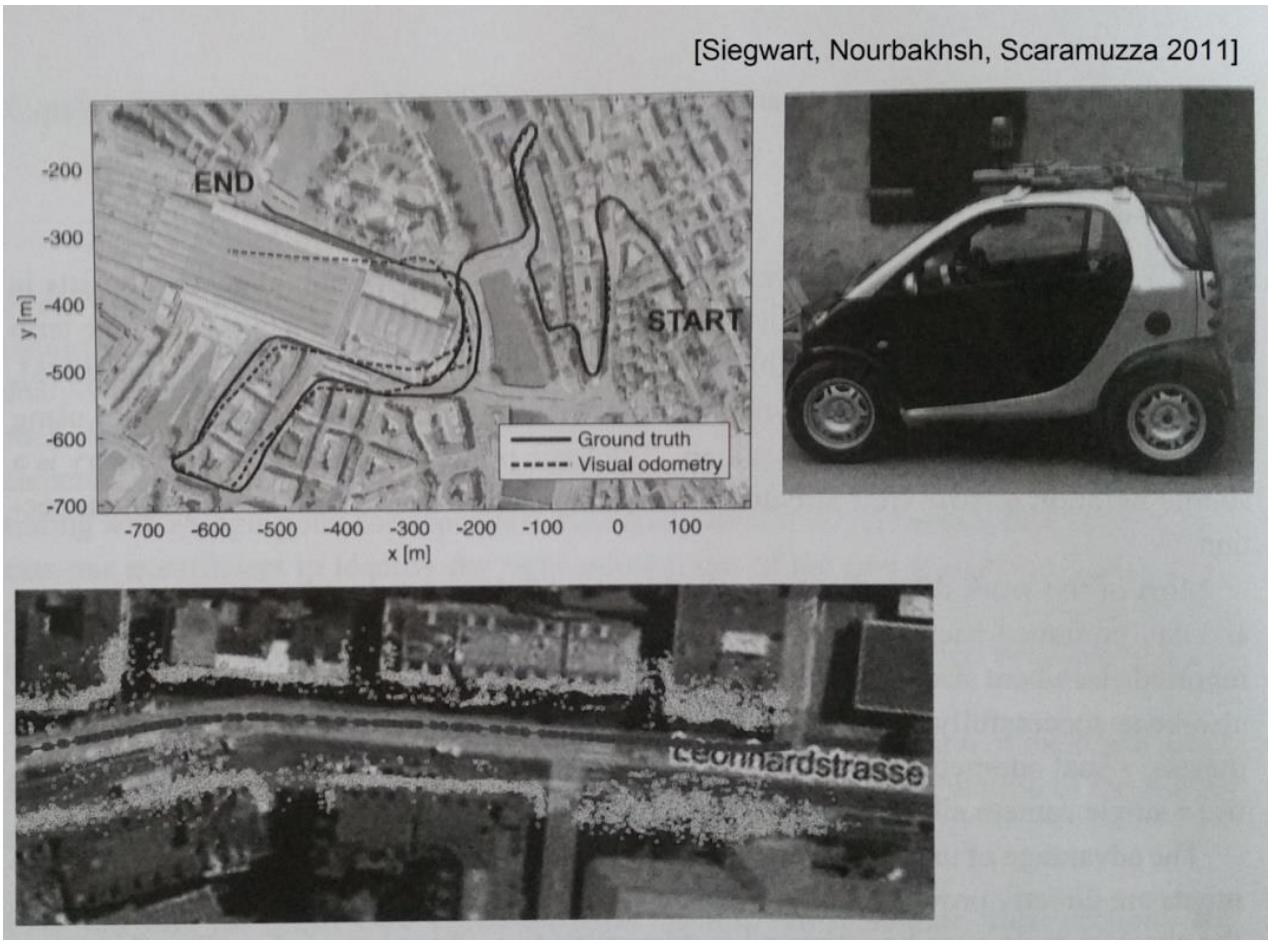

Figure 2.3. (upper left) An example visual odometry result with related map (bottom) obtained using a single omnidirectional camera mounted on the roof of a vehicle (right).

2.2 Simultaneous Localization and Mapping

Concurrent mapping and navigation are critical tasks for a mobile robotic platform when navigating in an unknown environment. Simultaneous localization and mapping (SLAM) solves this problem and can be implemented in a variety of different ways. The most relevant solutions to SLAM are focused on the feature-based approach, where feature descriptors are extracted from laser scans or images to solve the problem of matching observations to landmarks [Cortes, Sole, Salvi 2011]. SLAM consists of multiple parts (Figure 2.4): Landmark extraction, data association, state estimation, state update, and landmark update [Riisgaard, Blas 2005].

Since GPS and robot odometry are unreliable within a certain error threshold, the robot must rely on additional sensors to correct its position and orientation by extracting features from the environment and re-observing the same landmarks after every single movement action has taken place. An Extended Kalman Filter (EKF) is responsible for updating where the robot thinks it is based on these features. The EKF keeps track of an estimate of the uncertainty in the robots' position and also the uncertainty in these landmarks it has seen in the environment [Riisgaard, Blas 2005]. A detailed breakdown of the SLAM iteration process can be seen in Figures 2.5-2.9.

Visual SLAM (V-SLAM) is defined as a real-time EKF SLAM system using only a single perspective camera by simultaneously tracking interest points in the environment from captured images and the motion of the camera. V-SLAM is also called bearing-only SLAM, to emphasize the fact that it uses only angle-observations in contrast to laser-based or ultrasound-based SLAM, which need instead both angle and range information. Because of this, bearing-only SLAM is more challenging than range-bearing SLAM [Siegwart, Nourbakhsh, Scaramuzza 2011 357].

SLAM concepts can be applied to aerial robotic navigation through concurrent mapping and navigation. The main autonomous flight control component of my research uses a bearing-only V-SLAM for navigation due to its singular front facing camera.

[Riisgaard, Blas 2005]

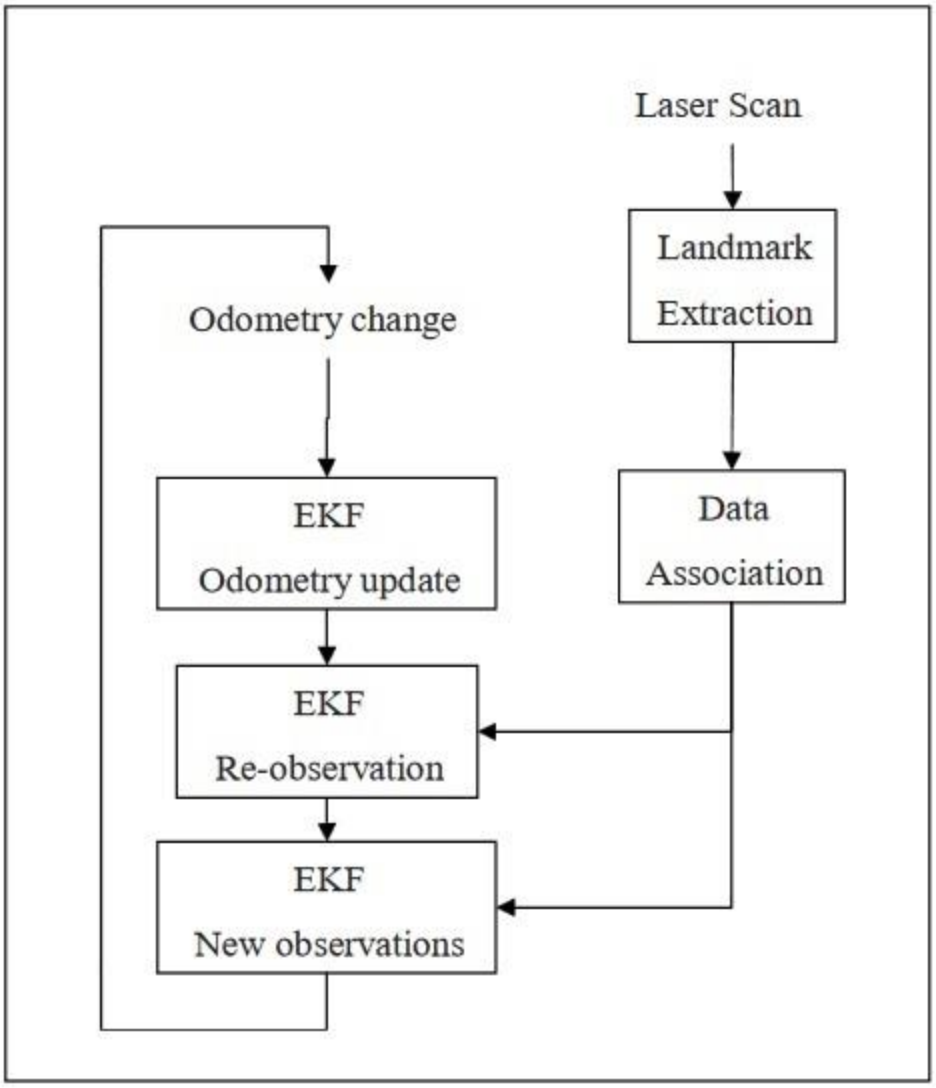

Figure 2.4. Overview of the SLAM process

[Riisgaard, Blas 2005]

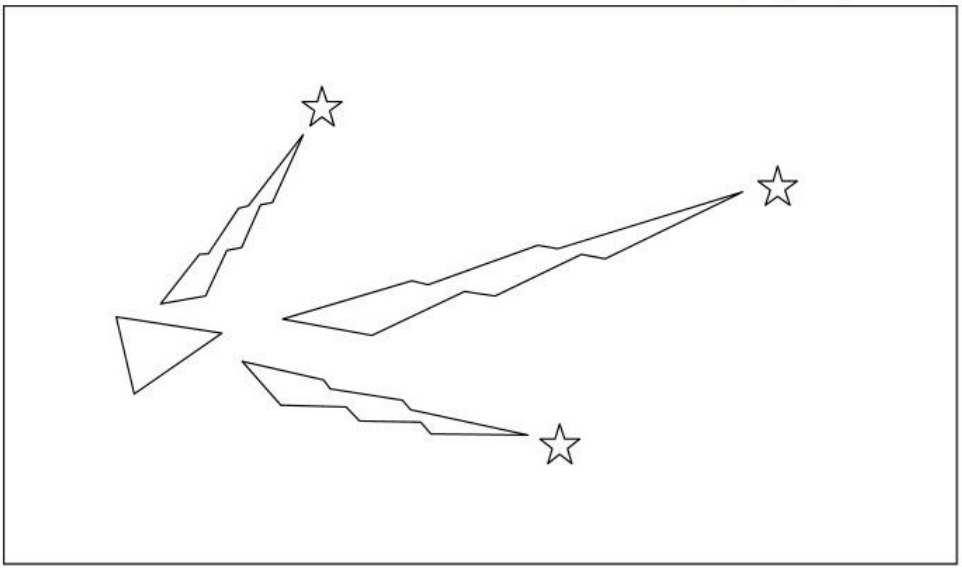

Figure 2.5. The robot is represented by the triangle. The stars represent landmarks. The robot initially measures using its sensors the location of landmarks (sensor measurements illustrated with lightning).

[Riisgaard, Blas 2005]

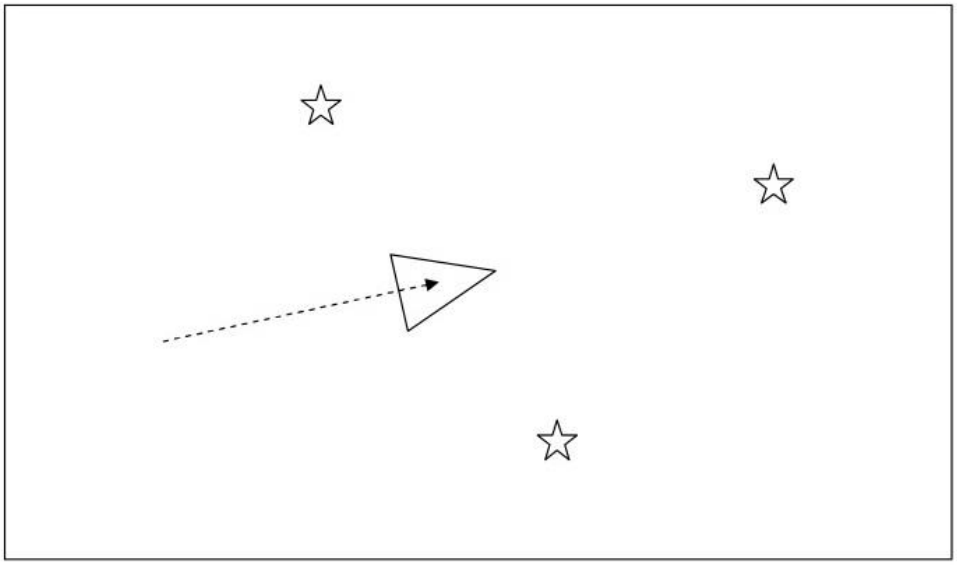

Figure 2.6. The robot moves so now it thinks it is in the location of the solid triangle. The distance moved is given by the robot's odometry.

[Riisgaard, Blas 2005]

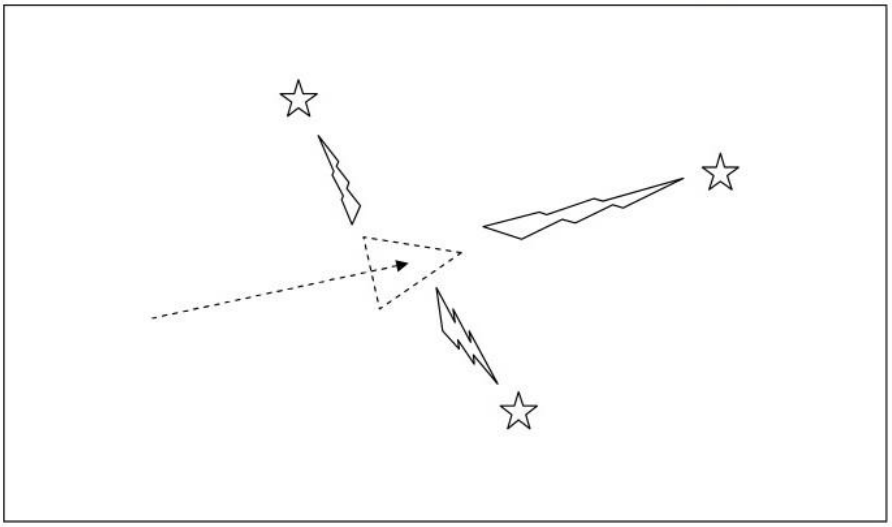

Figure 2.7. The robot once again measures the location of the landmarks using its sensors but finds out they don't match with where the robot thinks they should be (given the robots location). Thus the robot is not where it thinks it is.

[Riisgaard, Blas 2005]

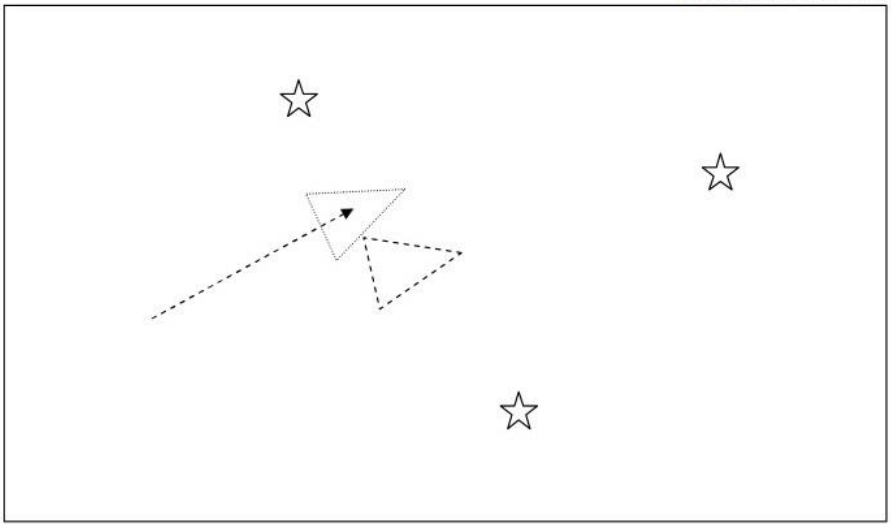

Figure 2.8. As the robot believes more its sensors than its odometry it now uses the information gained about where the landmarks actually are to determine where it is (the location the robot originally thought it was at is illustrated by the dashed triangle).

[Riisgaard, Blas 2005]

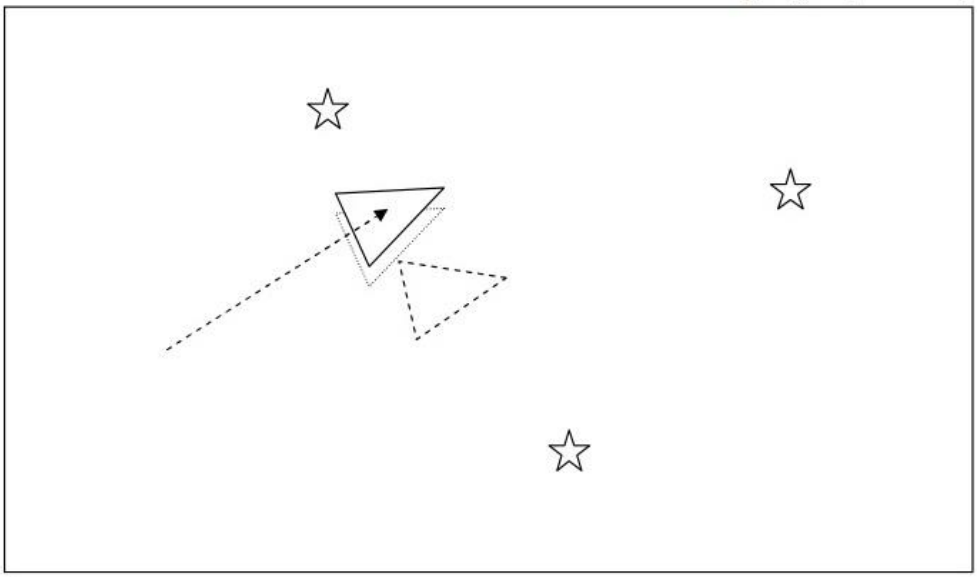

Figure 2.9. In actual fact the robot is in the location of the solid triangle. The sensors are not perfect so the robot will not precisely know where it is. However this estimate is better than relying on odometry alone. The dotted triangle represents where it thinks it is; the dashed triangle where odometry told it it was; and the last triangle where it actually is.

2.3 Multicopter Mapping

Early quadrotor vehicles were primarily experimental systems but improved design and software tools have led to significant increases in reliability and reduction in cost. Today, quadrotors have reached the maturity of consumer-grade devices [Loianno et al. 2015].

One example of a mapping multicopter uses consumer off-the-shelf (COTS) components and is equipped with an AutoPilot board consisting of an IMU, user-programmable ARM7 microcontroller, and a Google Tango smartphone enabling 3D navigation at speeds of 1-1.5 m/s with an average error of 4 cm in position [Loianno et al. 2015].

Another example uses a custom AscTec pelican quadcopter equipped with an Asus Xtion Pro Live sensor and an external ground station which performs camera tracking and 3D reconstruction in real-time, and sends the estimated position back to the quadrocopter at 30 fps with a delay of approximately 50 ms [Sturm, Bylow, Kerl, Kahl, Cremers 2013]. While these platforms use SLAM based movement algorithms, they are both customized quadcopter platforms. My research goes further and uses a COTS ready-to-fly multicopters for an easier plug-and-play approach.

2.4 Summary

In this chapter I described the research fields related to autonomous aerial robotics from mapping using a RGB-D camera, using individual camera frames for calculating trajectory, and multicopter mapping. Chapter 3 describes the hardware, software, and implementation method of the ready-to-fly autonomous platform, both as separate entities and a combined unit.

Chapter 3

Materials and Methods

In this chapter I describe my approach to creating the autonomous 3D mapping robotic platform of consumer off-the shelf components (Table 3.1) from hardware (Section 3.1) to software (Section 3.2) selection and the process of integrating them into a single, fully autonomous unit. A modular approach is taken to ensure that each component of the platform, the depth sensor with a 3D mapping program and an Android compatible multicopter, are working individually (Section 3.3) before merging each component into the combined architecture (Section 3.4).

Table 3.1. Consumer off-the-shelf components of the autonomous platforms and their respective links.

COTS Component	Link
Iris+ Quadcopter	https://store.3dr.com/products/IRIS
X8+ Octocopter	http://drnes.com/product/3dr-x8-plus/
Pixhawk	https://pixhawk.org/
Google Tango Tablet	https://get.google.com/tango/
Multicopter Mount	https://www.amazon.com/Robotics-Universal-Gimbal-Adapter-Bracket/dp/B00QPA3K92

3.1 Hardware

3.1.1 Iris+ Multicopter

In order to enable everyday consumers the financial ability to scan their physical surroundings into the digital world I chose two affordable multicopters. The first multicopter I chose is the Iris+ (Figure 3.1) due to its price ($599), ready-to-fly capability, telemetry radio, and autopilot controller. The Iris+ is a quadcopter, propelled by four rotors, developed by 3D Robotics and capable of lifting 400 g for 16-22 minutes of flight time. It is equipped with four 9.5 x 4.5 tiger motor multi-rotor self-tightening propellers, four 950 kV motors, uBlox GPS with integrated magnetometer, 5100 mAh 3S battery, 915 mHz telemetry radio, and 32-bit Pixhawk autopilot controller with Cortex M4 processor (Figure 3.2).

Figure 3.1. Iris+ base quadcopter

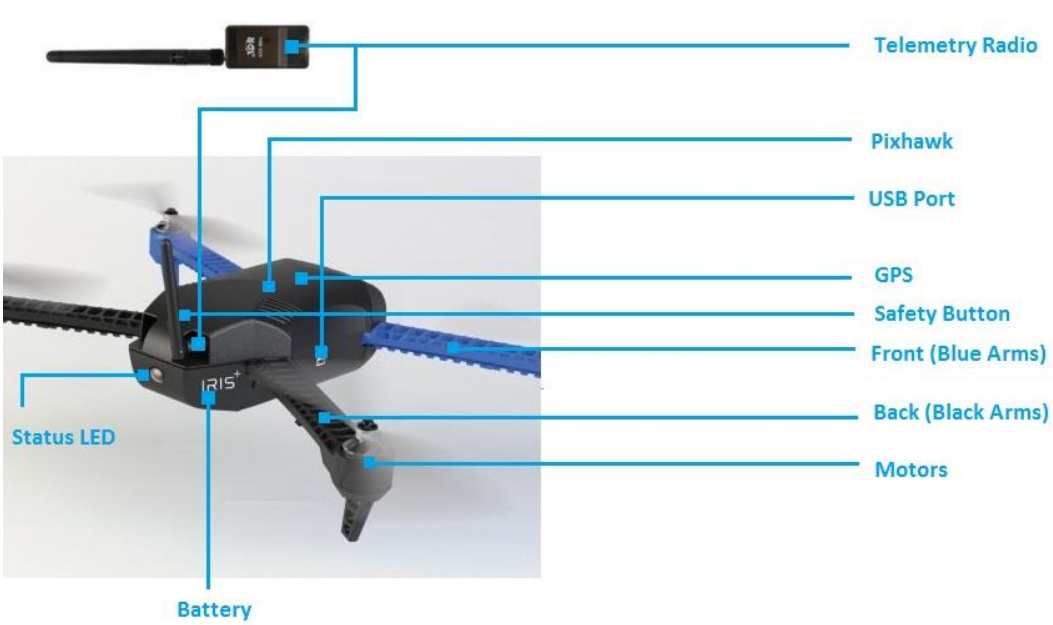

Figure 3.2. Iris+ detailed component diagram

3.1.2 X8+ Multicopter

The second multicopter I chose is the X8+ (Figure 3.3) due to its increased lift capacity and identical control hardware to the Iris+ quadcopter. The X8+ is an octocopter, propelled by eight rotors, developed by 3D Robotics and capable of lifting 800 g for 15 minutes of flight time or over 1000 g with reduced flight time. It is equipped with eight 11 x 4.7 APC multi-rotor propellers, eight 800 kV motors, uBlox GPS with Compass, 10000 mAh 4S battery, 915 mHz telemetry radio, and 32-bit Pixhawk autopilot controller with Cortex M4 processor (Figure 3.4).

Figure 3.3. X8+ base octocopter

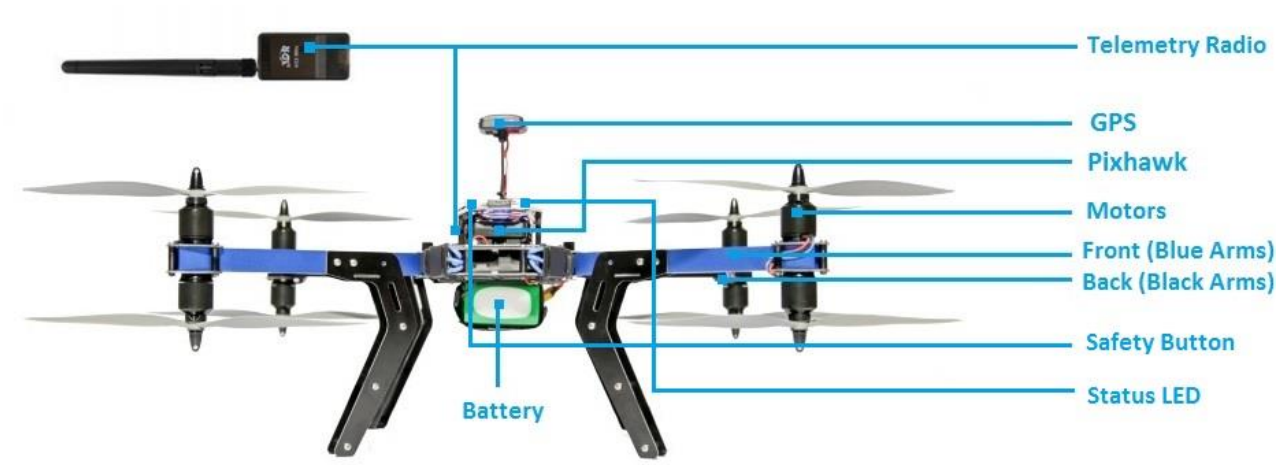

Figure 3.4. X8+ detailed component diagram

3.1.3 Pixhawk

To allow fully autonomous navigation, the 32-bit Pixhawk (Figure 3.5) autopilot is an all-in-one controller that combines a Flight Management Unit (FMU) and Input/Output (IO) into a single package. The Pixhawk is equipped with a 168 MHz 32-bit STM32F427 Cortex M4 core with FPU, 256 KB RAM, 2 MB Flash, and 32-bit STM32F103 failsafe co-processor. Its sensors include a ST Micro L3GD20H 16-bit gyroscope, ST Micro LSM303D 14 bit accelerometer / magnetometer, Invensense MPU 6000 3-axis accelerometer/gyroscope, and MEAS MS5611 barometer. The Pixhawk interfaces include 5x UART (serial ports), one high-power capable, 2x with HW flow control, 2x CAN (one with internal 3.3V transceiver, one on expansion connector), Spektrum DSM / DSM2 / DSM-X® Satellite compatible input, Futaba S.BUS® compatible input and output, PPM sum signal input, RSSI (PWM or voltage) input, I2C, SPI, 3.3 and 6.6V ADC inputs, Internal micro USB port, and external micro USB port extension.

3.1.4 Google Tango

The Android-driven Google Tango tablet (Figure 3.6) promises the future of mobile 3D motion and depth sensing. Using its integrated depth sensing, powerful computing, and 3D motion and feature tracking, the tablet is able to sense and map its immediate environment while maintaining its position relative to the world around it.

The Tango tablet is equipped with 128 GB internal storage, 4 GB RAM, USB 3.0 host via dock connector, Bluetooth 4.0, NVIDIA Tegra K1 with 192 CUDA cores, 4960 mAH cell battery, 1 MP front facing, fixed focus camera, 4 MP 2µm RGB-IR pixel sensor, 7.02” 1920x1200 HD IPS display, and dual-band WiFi. Its sensors include a motion tracking camera, 3D depth sensing, accelerometer, barometer, compass, aGPS, and a gyroscope.

[Computer Vision and Geometry Lab 2016]

1 Spektrum DSM receiver
2 Telemetry (radio telemetry)
3 Telemetry (on-screen display)
4 USB
5 SPI (serial peripheral interface) bus
6 Power module
7 Safety switch button
8 Buzzer
9 Serial
10 GPS module
11 CAN (controller area network) bus
12 I²C splitter or compass module
13 Analog to digital converter 6.6 V
14 Analog to digital converter 3.3 V
15 LED indicator

1 Input/output reset button
2 SD card
3 Flight management reset button
4 Micro-USB port

Figure 3.5. 32-bit Pixhawk autopilot controller

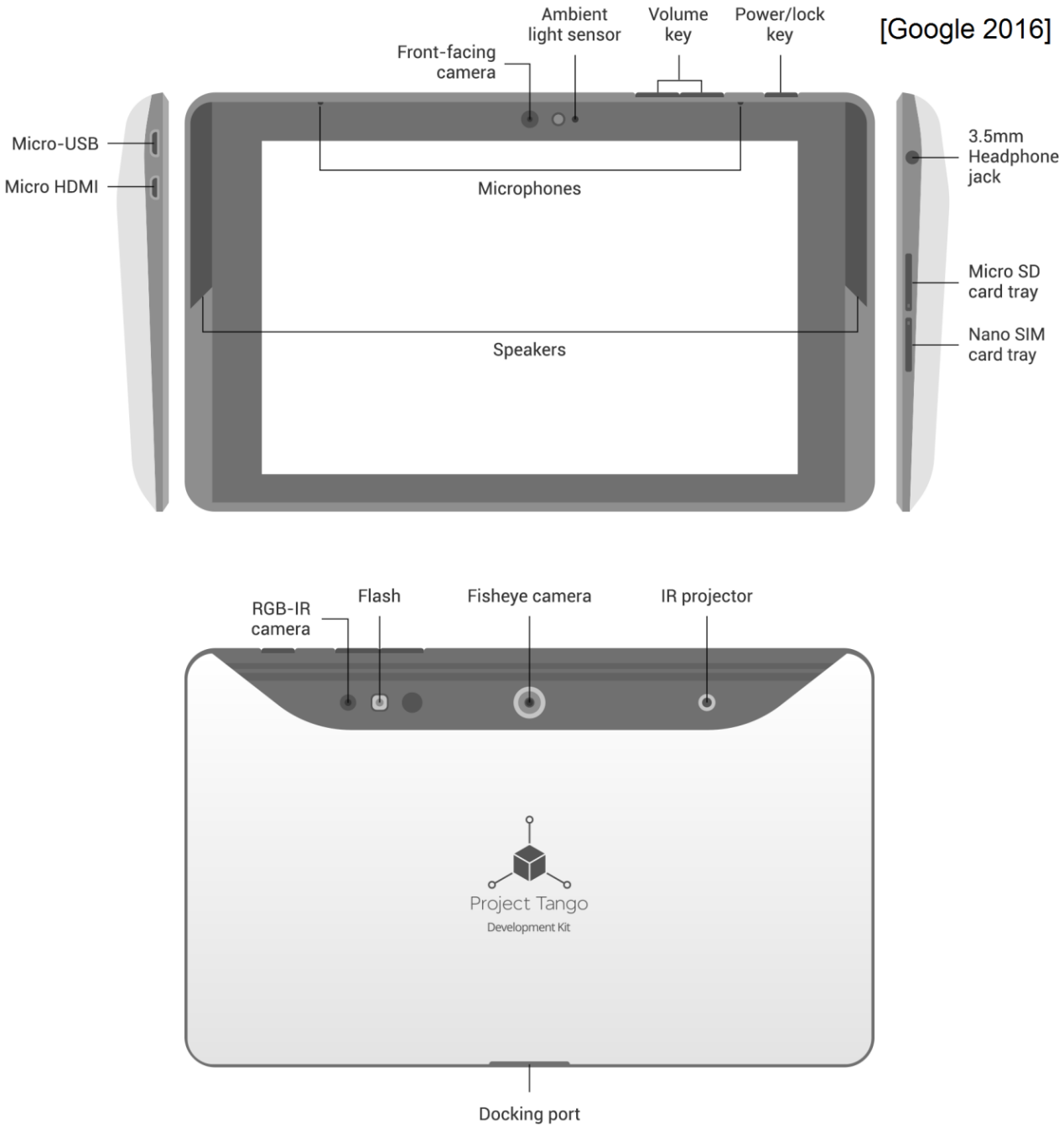

Figure 3.6. Hardware diagram for the Tango tablet development kit

3.2 Software

To achieve an all-in-one program interface and ensure compatibility between the Pixhawk autopilot controller and the Google Tango tablet, Android studio is chosen as the integrated development environment (IDE). This allows the entire autonomous flight mapping platform to operate untethered and independent of a ground station. The consumer off-the shelf software can be seen in Table 3.2.

Table 3.2. Consumer off-the-shelf software of the autonomous platforms and their respective links.

COTS Component	Link
Android Studio	https://developer.android.com/studio/index.html
Dronekit API	http://dronekit.io/
Google Tango API	https://developers.google.com/tango/
Meshlab	http://meshlab.sourceforge.net
Android Studio	https://developer.android.com/studio/index.html

3.2.1 Dronekit API

The Dronekit API is available for Python, Java, and for cloud-based web applications through the MAVLink protocol and enables aerial platform connectivity in the air with an onboard computer or on the ground for ground control with a laptop or desktop computer. This API allows for universal platform development, can be integrated with additional sensors, intelligent path planning, and autonomous flight.

The Python API is the suggested route for directly communicating with the APM flight controller with a companion computer for tasks in path planning, computer vision, or 3D modeling. The Java API is the suggested route for in-flight interactions or ground station controls. The cloud-based application is the suggested route for downloading data while in flight, from live logs to photos and videos.

3.2.2 Google Tango API

The Google Tango API is available in C, Java, and for Unity applications. Tango is a platform that uses computer vision to give devices the ability to understand their position relative to the world around them [Google 2016]. The Tango's API is capable of motion tracking by understanding position and orientation, area learning using visual cues and self-correcting motion tracking, and depth perception to understand the shape of the world around it.

The Java API is the suggested route for integrating the Tango's functions into other applications built with Java or Android Studios. The C API enables native flexibility but lacks the graphical user interface (GUI) required to display while the Unity SDK is solely for graphical applications such as games and 3D visualizations.

3.3 Implementation of 3D Mapping and Autonomous Controls

In order to reduce the error of the combined 3D mapping and autonomous flight platform, each component is created as its own Android application and is thoroughly tested and refined independently of the other components.

3.3.1 Google Tango Point Cloud 3D Mapping

To implement computer vision techniques required for path planning and 3D modeling, the depth sensor on the Google Tango is utilized to create a cloud of 3D points representing its position relative to the world. The Tango's orientation and position, relative to the world, are synchronized with each other by creating a writeable storage directory and saving the file buffer, pose translation, rotation, field of view, and the data points for each camera frame.

Once the 3D mapping process is completed, the scan data files are transferred to a computer and assembled using a python script. The python 3D assembler reads in each data file the same way it is written to memory, translates the data points from their

axis and angle coordinates to XYZ data points, and saves all the data points from a single camera frame as a 3D model of colored point cloud models. A modeling program, such as Meshlab, is used to assemble the final XYZ data points into a 3D environment model.

3.3.2 Autonomous Android Flight Guidance

Autonomous flight controls are made possible through the Dronekit API which enables Android applications access to the APM flight controller. To test the robustness of the Pixhawk autopilot controller with respect to on-the-fly, basic control commands are implemented for arming the multicopter, initializing home variables, turning right and left, moving forward and backwards, and correcting relative latitude, longitude, and altitude based on current location and motion drift.

3.4 Autonomous Mapping Platform

3.4.1 Architecture

To ensure compatibility between the Google Tango tablet and the Pixhawk autopilot controller, Java is chosen as the appropriate programming language and Android studio as the IDE. The hardware architecture of the combined autonomous platform, both the Iris+ and X8+, is shown in Figure 3.7. The ground station computer in the diagram represents the onboard android-driven Google Tango and the MAVLink represents the connection between the Pixhawk and the micro USB telemetry radio connected to the tablet.

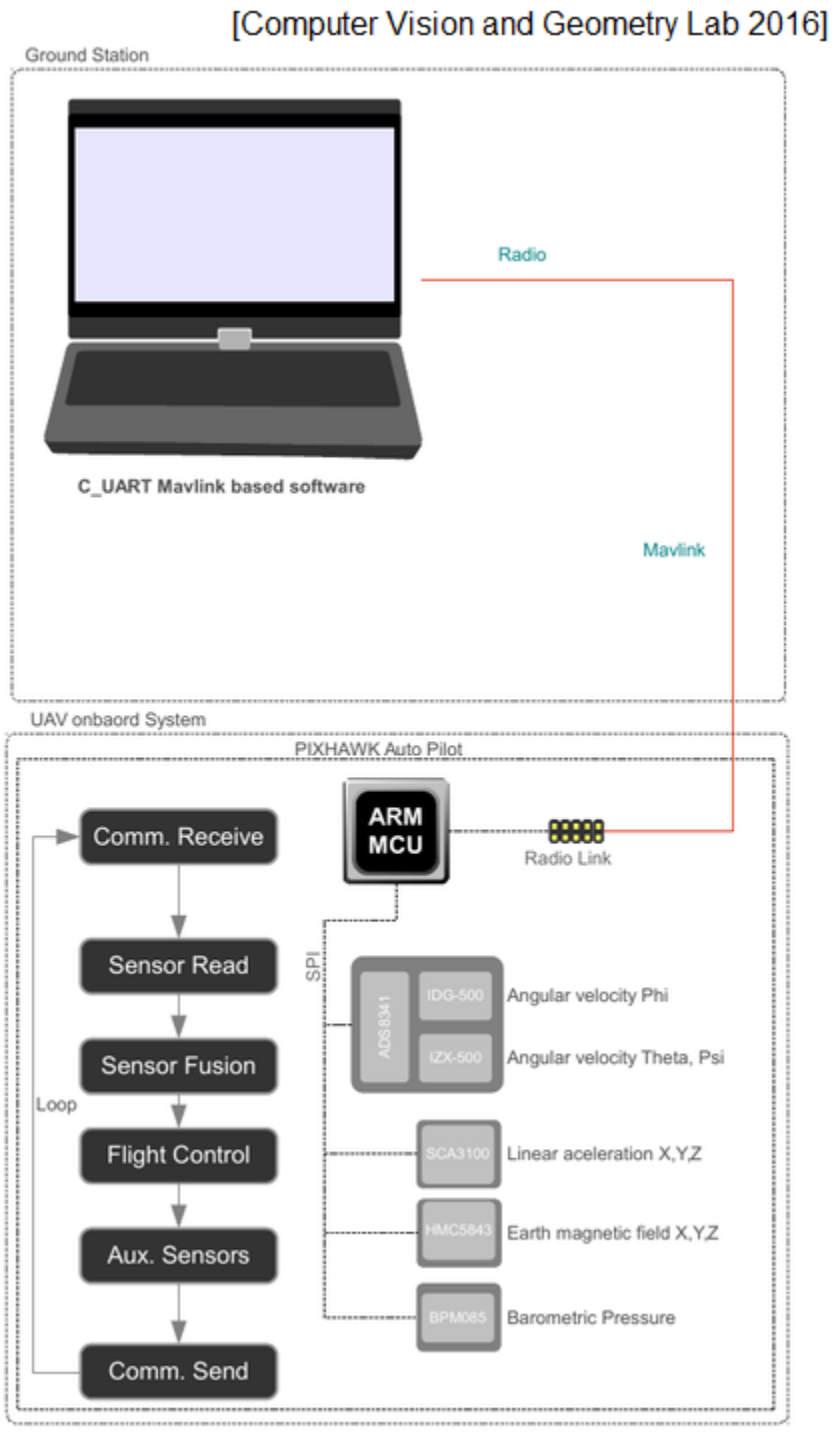

Figure 3.7. Autonomous 3D mapping multicopter architecture. The ground station represents the onboard Google Tango tablet and the MAVLink represents the telemetry radio.

3.4.2 Payload Stabilization

To reduce motion drift and increase flight stability, the payload must contain only mandatory components to remain as light as possible while centered on the bottom of the multicopter. Table 3.3 gives a breakdown of the Iris+ autonomous platform's payload components, their individual weights, and the total payload weight. Table 3.4 gives a breakdown of the X8+ autonomous platform's payload components, their individual weights, and the total payload weight.

Table 3.3. Iris+ autonomous platform payload components.

Iris+ Payload Item Description	**Weight**
Google Tango	370.7 g
USB 3.0 mounting base	81.5 g
Micro USB cable and radio	57.4 g
Gimbal and mounting hardware	15.4 g
Total Payload	**525 g**

Table 3.4. X8+ autonomous platform payload components.

X8+ Payload Item Description	**Weight**
Google Tango	370.7 g
USB 3.0 mounting base	81.5 g
Micro USB cable and radio	57.4 g
(4) 12" Aluminum leg extenders	194.5 g
(2) Wooden leg supports	162.7 g
Wooden base platform	285.7 g
Mounting hardware	46.2 g
Total Payload	**1198.7 g**

3.4.3 Component Integration

Combining each component of the autonomous platform is broken down into three steps: (1) Mounting payload components on the multicopter as a combined unit; (2) integrating the 3D mapping Android application into the autonomous flight control application; and (3) adding a modified SLAM based algorithm using the depth sensor data collected from the Google Tango tablet.

Step (1) involves calculating the payload weight for each platform, as described in Section 3.4.2, and attaching the payload to the center of mass on each multicopter. In order to evaluate this step for success or failure, each multicopter platform is flight tested manually to ensure minimal motion drift when operating in a hovering position.

Step (2) involves merging the 3D mapping component described in Section 3.3.1 with the autonomous flight control component described in Section 3.3.2. To retain a modular approach in implementation, the 3D mapping component is integrated into the basic autonomous flight control application without initially making any additional modifications to either component. In order to evaluate this step for success or failure, each multicopter platform is flight tested autonomously to ensure minimal motion drift while operating in a hovering position.

Step (3) involves creating a basic V-SLAM based algorithm for making autonomous flight control decisions using the single front facing depth sensor on the Google Tango tablet. The modified V-SLAM contains three operations that are reiterated at each time step:

1. The multicopter moves using a combination of its GPS sensor and motion tracking and the depth sensor on the Tango tablet to minimize motion drift.
2. The Google Tango tablet takes sensor readings from its new location in the environment and updates an internal 2D obstacle map.
3. The multicopter determines the next position to move to based on its internal 2D obstacle map and changes orientation respectively.

At the home position, the multicopter initializes its absolute latitude, longitude, altitude, and generates a blank 2D array representing the geo-fence where it is safe for

operation. The first operation is implemented by using a combination of the tablet position and GPS position of the multicopter in order to move in one foot increments. When a movement command is sent, the relative latitude, longitude, and altitude for loitering are updated to the new position. Due to the mounting position of the Tango tablet on the Iris+ quadcopter, the motion tracking orientation of the X and Z movement axes are flipped (Figure 3.8) with respect to the standard, upright mounting orientation on the X8+ (Figure 3.9). Once the new position is reached the multicopter corrects motion drift using constant function calls to center latitude, longitude, and altitude at a relative location.

The second operation is implemented by using the depth sensor on the Google Tango tablet to take measurements from its new location. If the pixel depth threshold exceeds a predetermined value, the multicopter uses its 2D obstacle map (Figure 3.10) to update the most recent position.

The third operation involves using the internal 2D obstacle map the multicopter is generating throughout its automation in order to determine the next movement position and pose. Initially the multicopter runs a simple wall-follow movement based algorithm. Once the basic V-SLAM movement and mapping function is successful, supplementary computer vision applications in object detection, motion tracking, and area learning are implemented with additional sensors for more intelligent in-flight decisions.

Figure 3.11 shows a detailed breakdown of the Iris+ autonomous platform, including its payload. Figure 3.12 shows a detailed breakdown of the X8+ autonomous platform, including its payload.

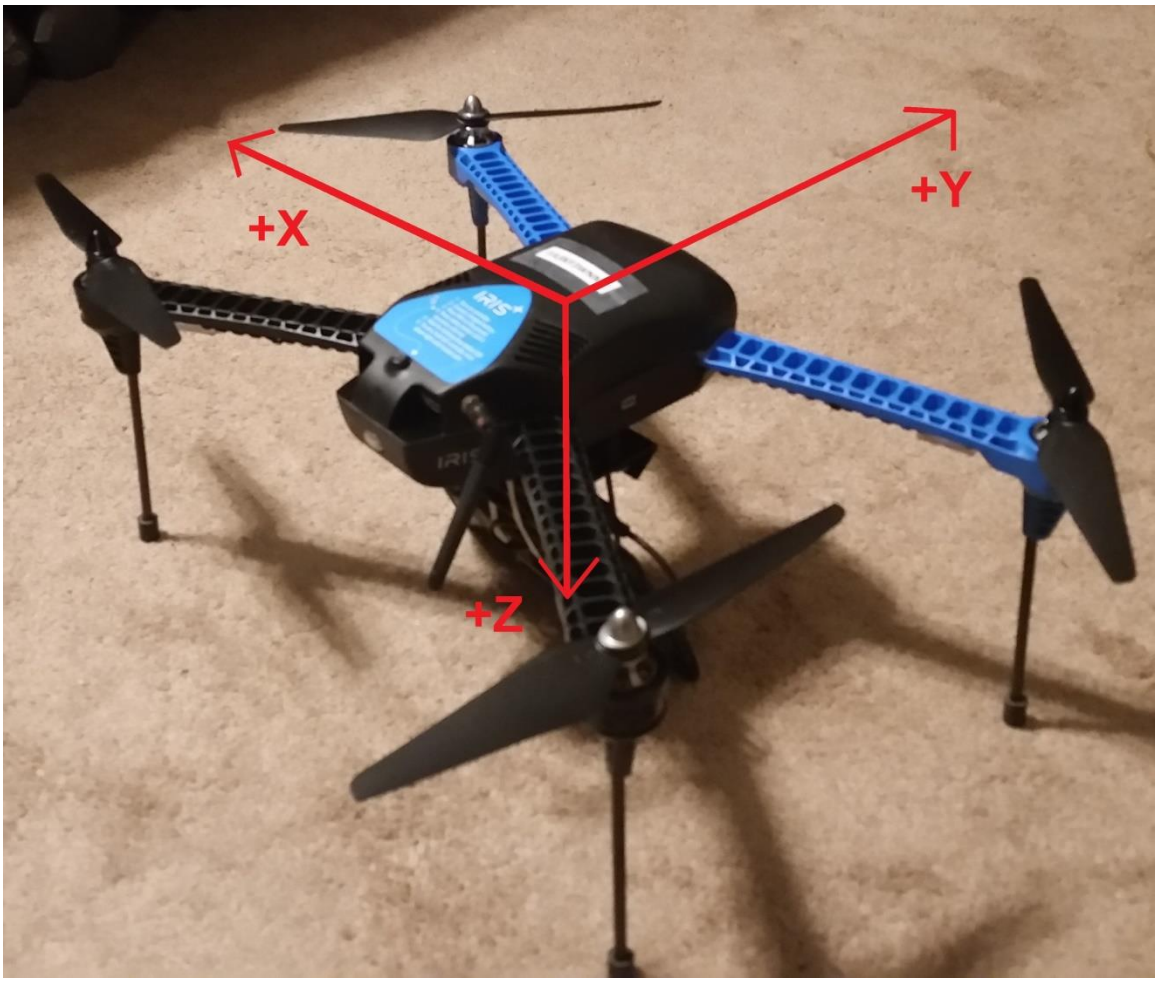

Figure 3.8. Iris+ Tango movement axis

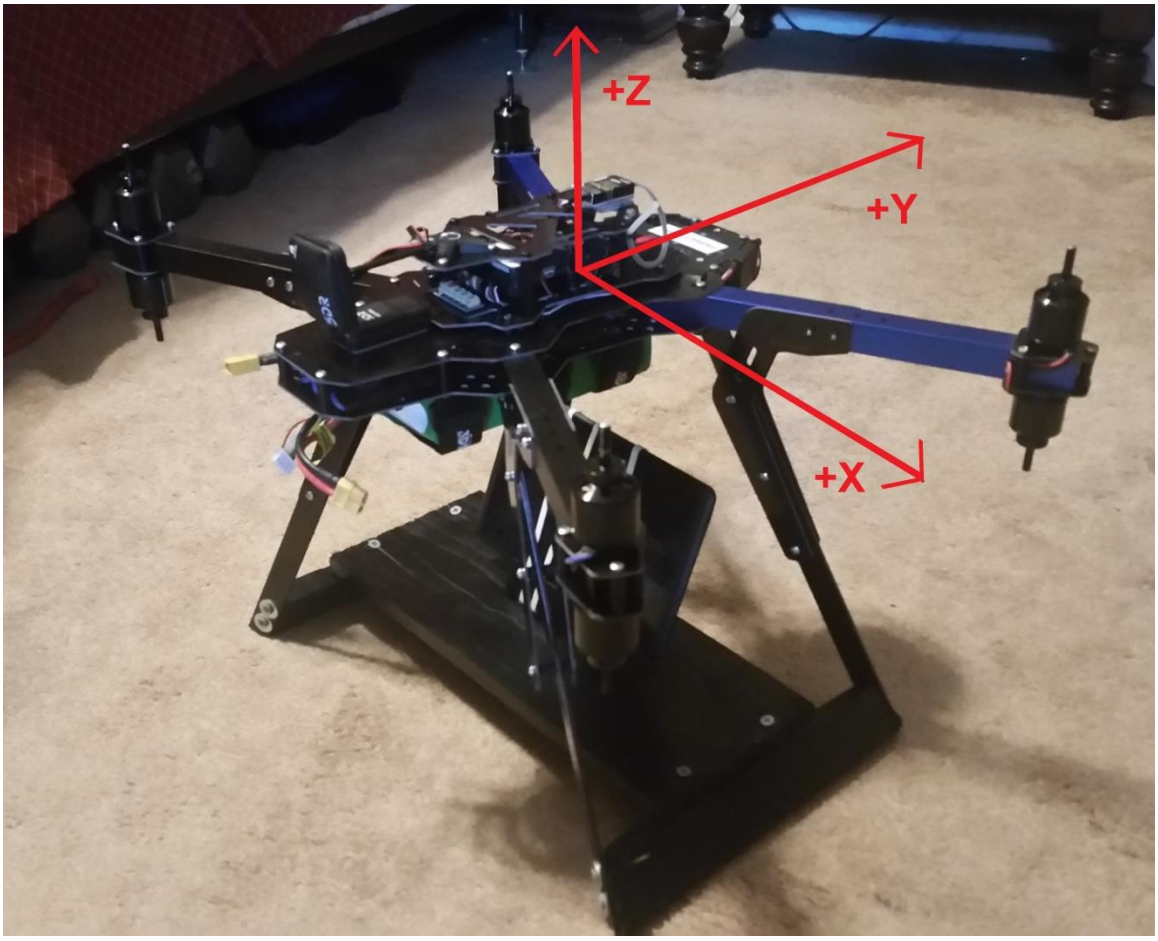

Figure 3.9. X8+ Tango movement axis

2D Multicopter Array Map															
16x16 ft Empty Room															
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3
3	1	0	0	0	0	0	1	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	1	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	1	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	1	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	1	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	H	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	3
3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	3
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Symbol Descriptions
9 – Home Location
3 – Impassible
0 – Unvisited
1 – Clear

Figure 3.10. Example of the multicopter’s 2D obstacle map

Figure 3.11. Iris+ detailed autonomous platform

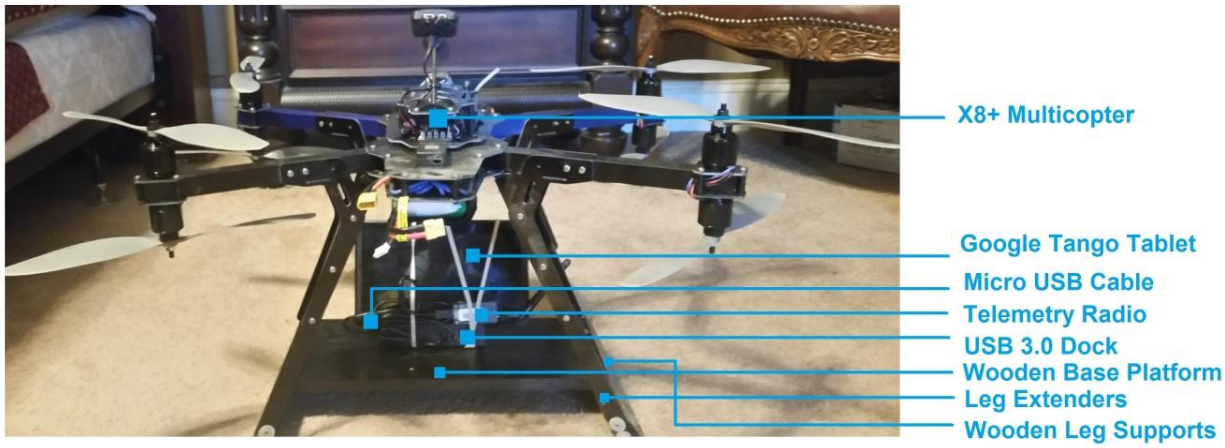

Figure 3.12. X8+ detailed autonomous platform

3.5 Summary

In this chapter I described my hardware and software selections, implementation of each component of the autonomous multicopter mapping platform independently of the other, and the combined implementation of all the components. Chapter 4 discusses the test results of both the separate and combined platform, as well as future work and potential applications of this research.

Chapter 4

Results and Discussion

This chapter presents the results of the autonomous mapping platform test runs on Iris+ and X8+ multicopters described in Section 3.3 and 3.4, both as separate implementations (Section 4.1), and a single combined unit (Section 4.2), with future directions and modifications to the current platform discussed in Section 4.3.

4.1 Separate Platform Implementation

4.1.1 Google Tango Point Cloud 3D Mapping

The point cloud mapping discussed in Section 3.3.1 is created by synchronizing the position, orientation, and data points captured by the Google Tango tablet and writing them to a file. To evaluate this component for correctness, an empty 16x16 foot room with two windows and one door is mapped on the tablet and then assembled on a PC using the python script. The point cloud model of the room is shown in Figure 4.1, a model of the 16x16 foot room overlaid on top of the point cloud data for verification is shown in Figure 4.2, and the actual room is shown in Figure 4.3. More complex objects are mapped to determine the level of detail obtained using only point cloud models and are shown in Figure 4.4.

Figure 4.1. Point cloud model of an empty 16x16 foot room assembled in Meshlab.

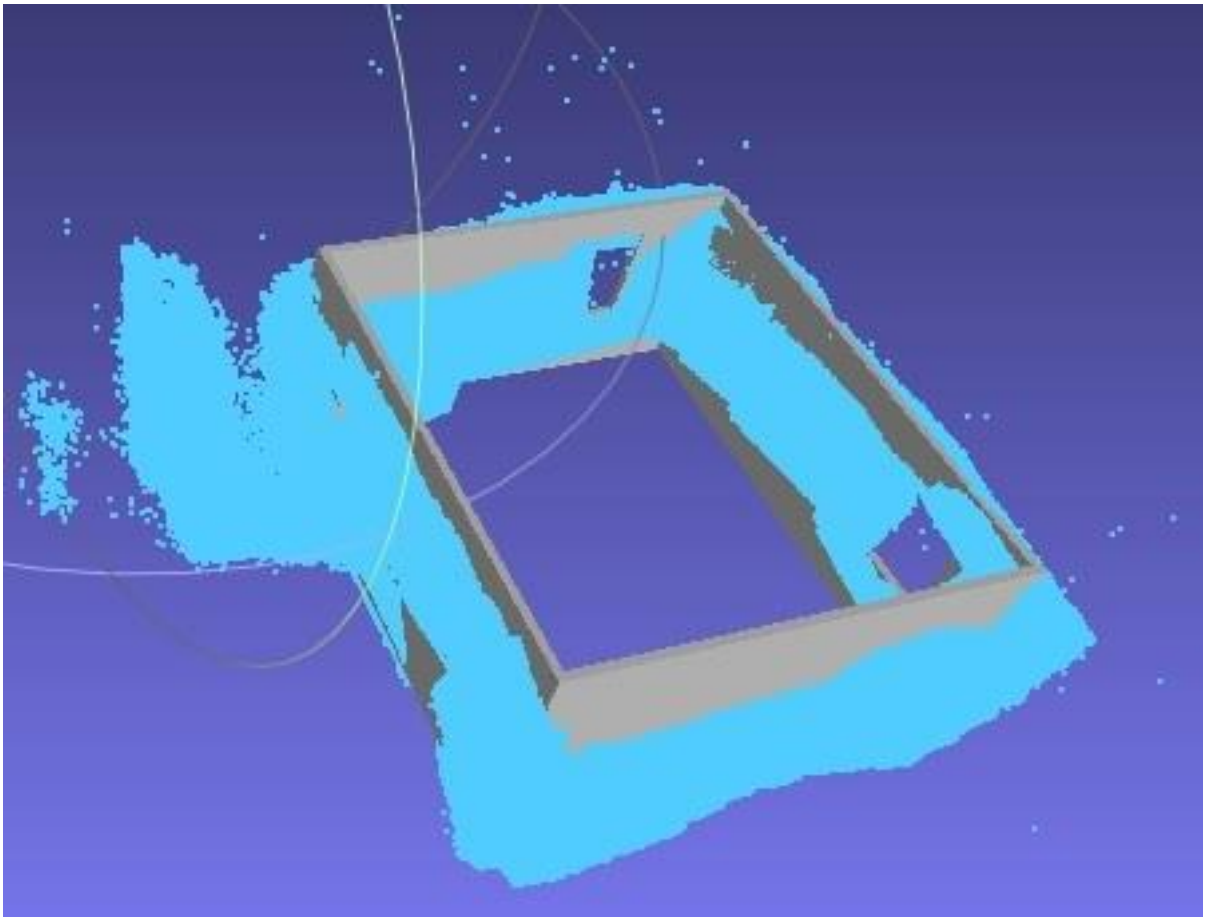

Figure 4.2. Model of an empty 16x16 foot room overlaid on the point cloud model.

Figure 4.3. Photos of the actual 16x16 foot room.

Figure 4.4. Complex objects mapped with the actual photo, to the right respectively, for comparison.

4.1.2 Autonomous Android Flight Guidance

The Autonomous flight guidance is implemented by creating a sequence of autonomous flight controls for variable initialization, turning left and right, moving forward and backwards, correcting latitude, longitude and altitude based on its current position and motion drift. To evaluate the controls for success or failure, both multicopters are tested using a command sequence to arm the autonomous robotic platform, takeoff to a predetermined height, loiter in place at the home location, turn left or right at various angles, move forward or backward preset distances, and finally land back at the home location. Without the Google Tango tablet payload and the inability to access its sensors for stabilization and motion drift correction, both the Iris+ and X+ multicopters successfully launched to the correct altitude, flew its preset flight path and landed with minimal motion drift using only the onboard GPS for corrections.

The Iris+ trial flight results are shown in Tables 4.1-4.3 and the X8+ trial flight results are shown in Tables 4.4-4.6. Scatter plots with linear trend lines are shown in Figure 4.5 for takeoff, Figure 4.6 for moving, and Figure 4.7 for turning. The y-intercept of each trend line represents the amount of motion drift accumulated while the multicopter is loitering in place and waiting for flight commands. The y-intercept for the takeoff trend line represents altitude drift, the movement trend line y-intercept represents forward or backward drift, and the turning trend line y-intercept represents drift to the left or right. The slope of each trend line represents the average error given a specific flight command. A trend line slope closest to one represents the highest level of automated movement control accuracy. A slope less than one represents an undershooting of the target altitude, movement position, or turn angle. A slope greater than one represents an overshooting of the target altitude, movement position, or turn angle.

Table 4.1. Iris+ automated takeoff trial runs programmed height, actual height reached, and percent error. A measurement error of approximately 3 inches is not included in the percent error calculation.

Iris+ Automated Takeoff		
Programmed	**Actual**	**% Error**
24.00”	27.75”	15.63%
24.00”	30.00”	25.00%
24.00”	31.00”	29.17%
30.00”	28.00”	6.67%
30.00”	32.50”	8.33%
30.00”	34.25”	14.17%
36.00”	33.75”	6.25%
36.00”	38.25”	6.25%
36.00”	40.00”	11.11%
42.00”	40.00”	4.76%
42.00”	43.50”	3.57%
42.00”	44.75”	6.55%
48.00”	45.00”	6.25%
48.00”	46.00”	4.17%
48.00”	49.75”	3.65%
72.00”	66.00”	8.33%
72.00”	67.75”	5.90%
72.00”	69.25”	3.82%

Table 4.2. Iris+ automated movement trial runs programmed distance, actual distance reached, and percent error. A measurement error of approximately 6 inches is not included in the percent error calculation.

Iris+ Automated Movement		
Programmed	**Actual**	**% Error**
+12.00”	+10.00"	16.67%
+12.00”	+14.25"	18.75%
+12.00”	+15.00”	25.00%
+18.00”	+17.25”	4.17%
+18.00”	+21.75”	20.83%
+18.00”	+23.00”	27.78%
+48.00”	+45.50”	5.21%
+48.00”	+46.00”	4.17%
+48.00”	+50.25”	4.69%
-12.00”	-11.25”	6.25%
-12.00”	-14.00”	16.67%
-12.00”	-16.00”	33.33%
-18.00”	-19.25”	6.94%
-18.00”	-20.00”	11.11%
-18.00”	-22.50”	25.00%
-48.00”	-45.75”	4.69%
-48.00”	-47.00"	2.08%
-48.00”	-51.50”	7.29%

Table 4.3. Iris+ automated turning trial runs programmed angle, actual angle reached, and percent error. A measurement error of approximately 10 degrees is not included in the percent error calculation.

Iris+ Automated Turning		
Programmed	**Actual**	**% Error**
Right 45°	Right 38°	15.56%
Right 45°	Right 42°	6.67%
Right 45°	Right 46°	2.22%
Right 90°	Right 87°	3.33%
Right 90°	Right 91°	1.11%
Right 90°	Right 94°	4.44%
Right 360°	Right 347°	3.61%
Right 360°	Right 351°	2.50%
Right 360°	Right 358°	0.56%
Left 45°	Left 40°	11.11%
Left 45°	Left 44°	2.22%
Left 45°	Left 47°	4.44%
Left 90°	Left 80°	11.11%
Left 90°	Left 83°	7.78%
Left 90°	Left 96°	6.67%
Left 360°	Left 343°	4.72%
Left 360°	Left 351°	2.50%
Left 360°	Left 371°	3.06%

Table 4.4. X8+ automated takeoff trial runs programmed height, actual height reached, and percent error. A measurement error of approximately 3 inches is not included in the percent error calculation.

X8+ Automated Takeoff		
Programmed	**Actual**	**% Error**
24.00”	26.75”	11.46%
24.00”	21.25”	11.46%
24.00”	26.50”	10.42%
30.00”	29.75”	0.83%
30.00”	30.25”	0.83%
30.00”	31.50”	5.00%
36.00”	34.75”	3.47%
36.00”	35.50”	1.39%
36.00”	36.00”	0.00%
42.00”	41.25”	1.79%
42.00”	42.00”	0.00%
42.00”	42.50”	1.19%
48.00”	47.25”	1.56%
48.00”	48.00”	0.00%
48.00”	49.25”	2.60%
72.00”	70.00”	2.78%
72.00”	72.50”	0.69%
72.00”	73.00”	1.39%

Table 4.5. X8+ automated movement trial runs programmed distance, actual distance reached, and percent error. A measurement error of approximately 6 inches is not included in the percent error calculation.

X8+ Automated Movement		
Programmed	**Actual**	**% Error**
+12.00”	+11.25”	6.25%
+12.00”	+12.50”	4.17%
+12.00”	+13.00”	8.33%
+18.00”	+19.00”	5.56%
+18.00”	+20.25”	12.50%
+18.00”	+21.00”	16.67%
+48.00”	+48.75”	1.56%
+48.00”	+49.25”	2.60%
+48.00”	+51.00”	6.25%
-12.00”	-12.00”	0.00%
-12.00”	-13.50”	12.50%
-12.00”	-14.00”	16.67%
-18.00”	-17.50”	2.78%
-18.00”	-18.75”	4.17%
-18.00”	-20.00”	11.11%
-48.00”	-46.75”	2.60%
-48.00”	-49.00”	2.08%
-48.00”	-50.00”	4.17%

Table 4.6. X8+ automated turning trial runs programmed angle, actual angle reached, and percent error. A measurement error of approximately 10 degrees is not included in the percent error calculation.

X8+ Automated Turning		
Programmed	**Actual**	**% Error**
Right 45°	Right 41°	8.89%
Right 45°	Right 44°	2.22%
Right 45°	Right 51°	13.33%
Right 90°	Right 83°	7.78%
Right 90°	Right 86°	4.44%
Right 90°	Right 92°	2.22%
Right 360°	Right 349°	3.06%
Right 360°	Right 356°	1.11%
Right 360°	Right 364°	1.11%
Left 45°	Left 42°	6.67%
Left 45°	Left 43°	4.44%
Left 45°	Left 49°	8.89%
Left 90°	Left 85°	5.56%
Left 90°	Left 88°	2.22%
Left 90°	Left 90°	0.00%
Left 360°	Left 352°	2.22%
Left 360°	Left 357°	0.83%
Left 360°	Left 365°	1.39%

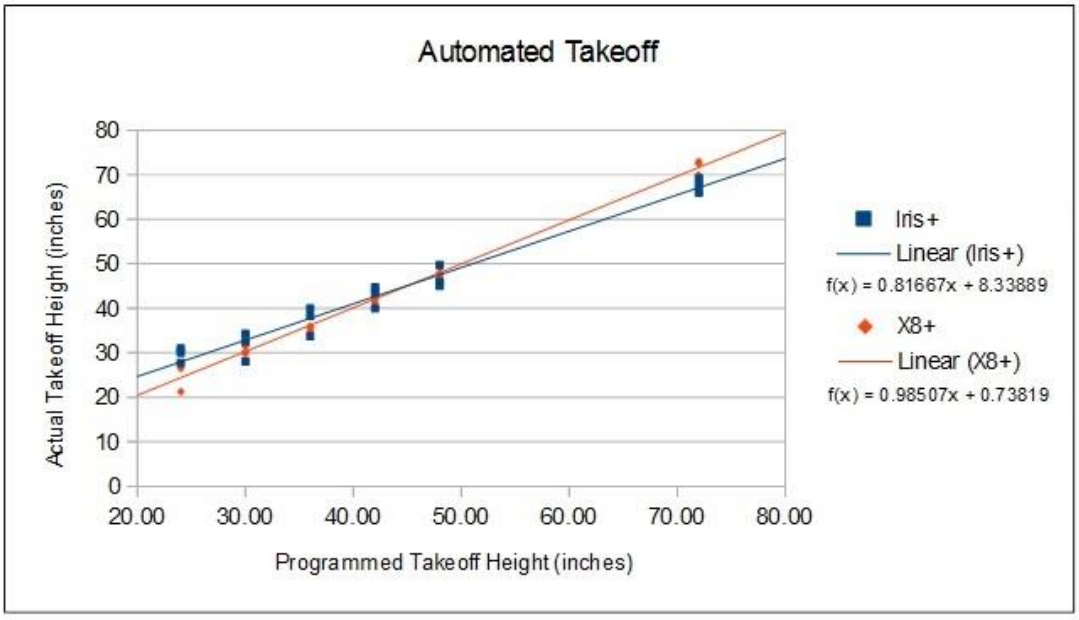

Figure 4.5. Iris+ and X8+ autonomous takeoff scatter plot.

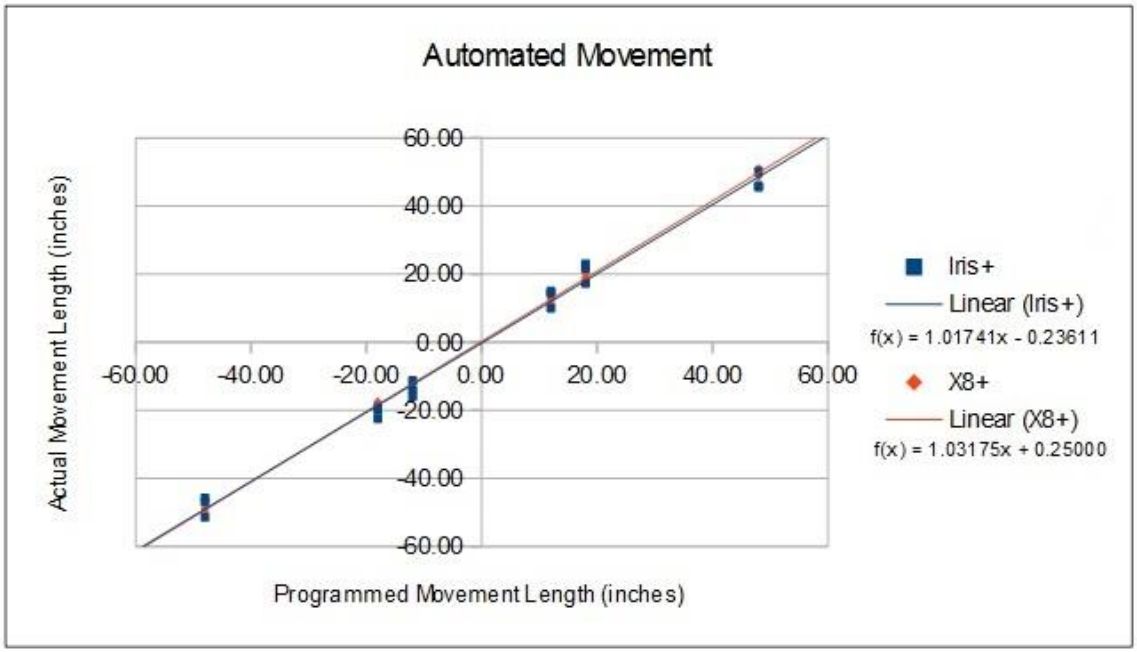

Figure 4.6. Iris+ and X8+ autonomous movement scatter plot.

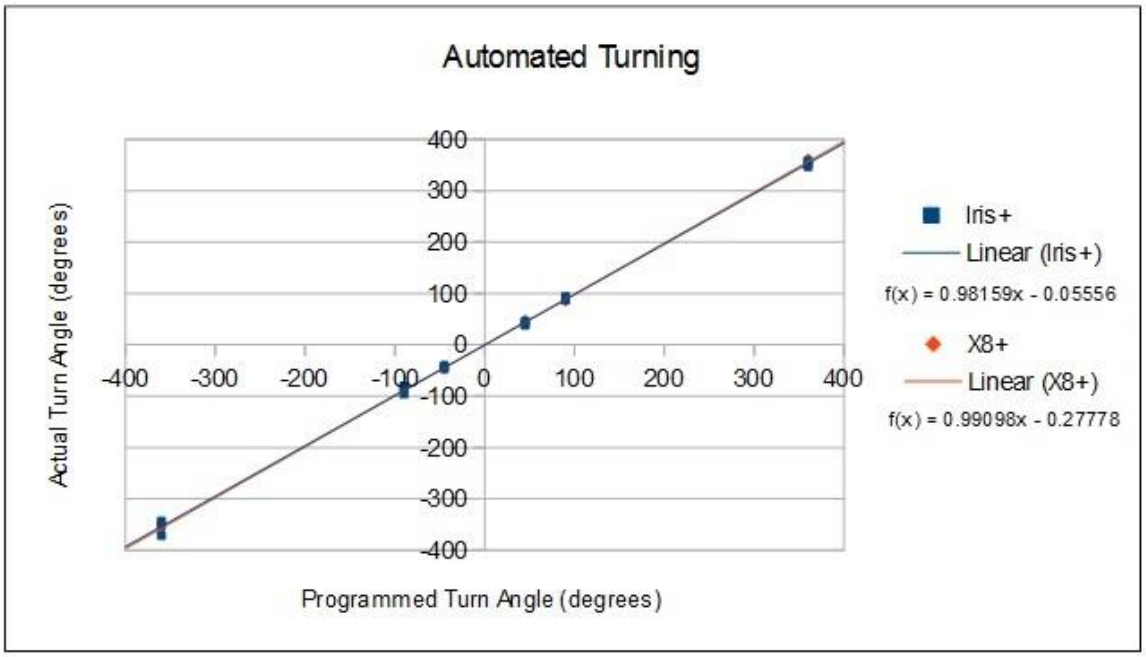

Figure 4.7. Iris+ and X8+ autonomous turning scatter plot.

4.2 Combined Platform Implementation

4.2.1 Iris+ Autonomous Platform

The weight of the Iris+ platforms payload, as shown in section 3.4.2, exceeded the maximum operating capacity of 400g; however, a combined platform test is still run in order to determine the adverse effects of an encumbered multicopter on the autonomous flight controls.

During takeoff the Iris+ struggled to get off the ground. When the platform finally took flight the launch is unstable at best. It is unable to reach the pre-designated altitude for takeoff due to desensitized controls from an overburdened payload. After altering the takeoff altitude to 6 feet, from the initial 4 feet, the platform is able to takeoff and attain the original altitude of 4 feet while loitering. An example of a failed autonomous 3D mapping test is shown in Figure 4.8. During autonomous flight with an overburdened payload, the Iris+ experiences an excessive amount of motion drift while loitering, discrepancy in autonomous takeoff altitude to actual altitude reached, a flipped X and Z axes due to the Google Tango tablet's mounting position, and unpredictable flight controls. Therefore, the Iris+ is upgraded to a more robust multicopter platform. Test flight video of the Iris+ autonomous platform can be seen at https://www.youtube.com/playlist?list=PLwzUC7_hiftAgcEvLwcU5HLltVmjSM4sG.

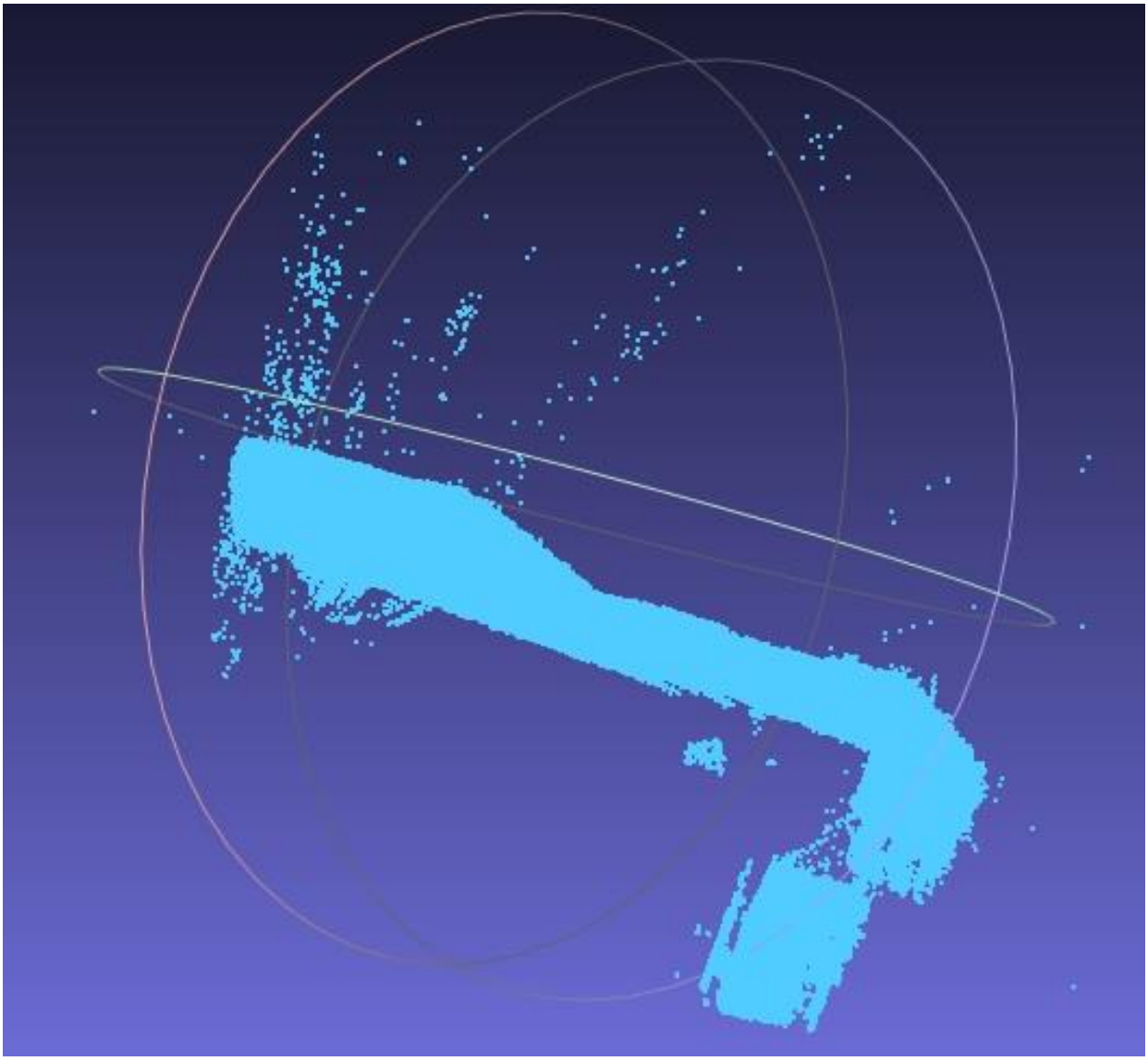

Figure 4.8. Iris+ autonomous point cloud scan assembled in Meshlab

4.2.2 X8+ Autonomous Platform

The weight of the X8+ platforms payload, as shown in section 3.4.2, is within the acceptable operating capacity, with a reduced flight time, and allowed the Google Tango the ability to mount in an upright, forward facing direction with the X and Z axes pointing in the proper orientations.

After a successful takeoff to the initial altitude of 4 feet, relative to the initialized home altitude, the autonomous X8+ began to encounter problems. If the testing environment is too bright or too dark the IR sensor on the Google Tango used for depth sensing and obstacle avoidance is unable to detect anything, causing the multicopter to smash directly into obvious walls. To eliminate some of known issues, the preset depth threshold is increased to 6 feet from 2 feet for a higher sensitivity and the multicopter platform is moved to an evenly lit room. Indoor tests continued to encounter issues with inconsistencies and, in some cases, complete loss of GPS signal, resulting in unpredictable and erratic movements. When the GPS is able to retain a signal throughout flight, a lack of robustness in the Java application caused over half of all the move, turn, and relative location stabilization commands to fail in execution and timeout. An example of a failed autonomous 3D mapping test is shown in Figure 4.9. Test flight video of the X8+ autonomous platform can be seen at https://www.youtube.com/playlist?list=PLwzUC7_hiftAgcEvLwcU5HLltVmjSM4sG.

4.3 Lessons Learned

This thesis makes several contributions to the field of autonomous aerial robotic mapping but the most important lesson learned is that hypersensitive controls, combined with synchronized visual odometry concepts in object detection and recognition, are required for aerial navigation and obstacle avoidance. An autonomous MAV also needs to be equipped with other sensors, such as a RGB-D camera, which is limited by several factors, such as the low maximum payload, power consumption, and processing resources [Schauwecker, Ke, Scherer, Zell 2012].

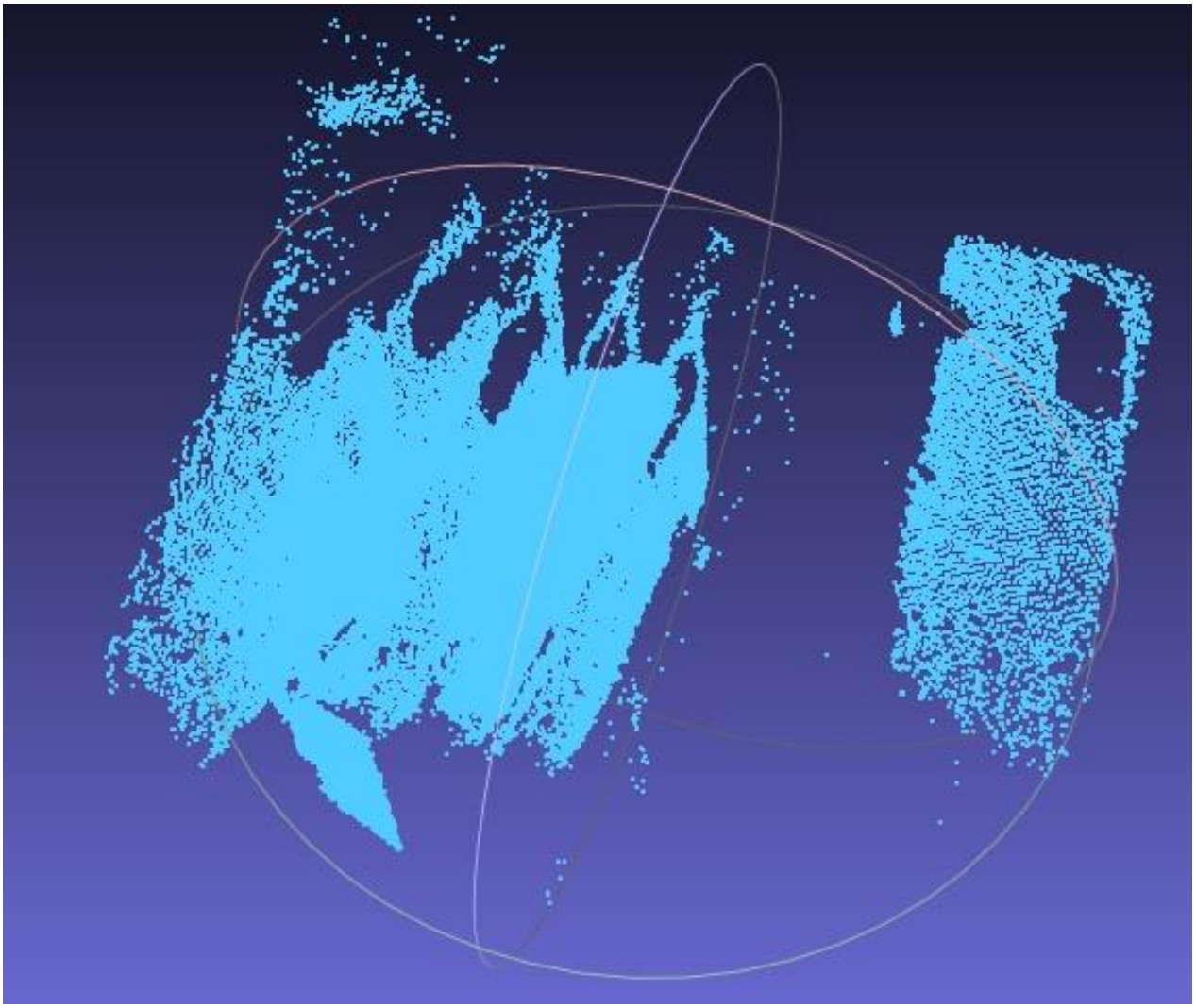

Figure 4.9. X8+ autonomous point cloud scan assembled in Meshlab

4.4 Future Directions

New innovative technologies and the future of computing both incorporate important computer vision concepts for acquiring, processing, analyzing, and understanding high-dimensional data from the real world. This research encompasses one way to integrate computers into the physical-world by autonomously creating 3D generated models from any real-world environment, allowing a user to view the environment around them digitally, whether it is a mile away or a thousand miles away. The technological applications span a variety of fields including: commercial and residential real-estate, environmental and natural disaster areas, hazardous environments, SWAT, police departments, and military applications.

Chapter 5

Conclusions

This thesis presents one approach to creating a fully autonomous, self-contained 3D mapping multicopter Android application using only consumer available electronic devices.

In order to implement a potential solution, two affordable multicopters and an Android tablet are chosen for the hardware based on their Java API compatibility and simple connectivity. The minimum payload is an overburden on the smaller Iris+ and causes it to behave erratically and unpredictably with respect to the commands it is given, which is ultimately its downfall. The autonomous X8+ platform has more success with the ability to loiter in place and map portions of the environment without moving; however, once the depth sensor and movement commands are integrated, compounding problems including loss of signal from the Pixhawk GPS, light interference with the Google Tango IR sensor, and lack of programming robustness due to all the components running in a single application, resulting in smashed propellers and obstacle collisions.

While this entire approach has not been completely successful, it has established a solid foundation for consumer electronic driven multicopters. The multicopters and their Pixhawk autopilot controllers are more than sufficient for accurate flight control and the Tango enabled tablet is more than capable of 3D mapping and sensing. Although multicopter hardware is rapidly changing and evolving, the combined Android based

Pixhawk control software and Tango 3D point cloud mapping program can be used as a foundation for future research in autonomous aerial 3D mapping. The Pixhawk hardware has the ability to control a multitude of different multicopters from ready-to-fly to completely customized, enabling the Android app to be continuously tested and refined on a variety of different aerial vehicles for future research. Supplementary sensors should be added for increased loiter stabilization, 360 degree environment field of view (FOV), more accurate flight controls, and improved obstacle avoidance. Additionally, the sensors and 3D modeling components should be broken into their own process and an onboard processor such as a Raspberry Pi 2 or an ODROUD-XU to provide a more robust interface between the sensors and the autonomous flight controls while synchronizing the camera frames with the multicopter movement commands.

Autonomously creating 3D digital maps of the real world could allow users to: View real-estate globally without leaving the comfort of their living room; Map and monitor environmental and natural disaster areas during and after their destruction; Reconstruct hazardous or uninhabitable environments digitally, such as Chernobyl and Fukushima; and create 3D recon models for police, SWAT, and military before entering an unknown area. However, due to the current limitations in multicopter payload lift capacity, battery life, GPS accuracy, API robustness, component compatibility, and price, the technology is still in the early development stages for consumers.

Bibliography

[3D Robotics Iris+ 2016] 3D Robotics. "IRIS | 3DR | Drone & UAV Technology." *3DR Drone UAV Technology,* (2016). https://3dr.com/kb/iris/.

[3D Robotics X8+ 2016] 3D Robotics. "X8 | 3DR | Drone & UAV Technology." *3DR Drone UAV Technology,* (2016). https://3dr.com/kb/x8/.

[3D Robotics DroneKit 2016] 3D Robotics. "DroneKit by 3D Robotics." *DroneKit*. 3D Robotics, (2016). http://dronekit.io/.

[Bachrach et al. 2012] Bachrach, A., S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy. "Estimation, Planning, and Mapping for Autonomous Flight Using an RGB-D Camera in GPS-denied Environments." *The International Journal of Robotics Research* 31.11 (2012): 1320-343.

[Callaham 2015] Callaham, John. "Google Says There Are Now 1.4 Billion Active Android Devices Worldwide." *Android Central*. 29 Sept. (2015).

[Carrillo, López, Lozano, Pégard 2011] Carrillo, Luis Rodolfo García, Alejandro Enrique Dzul López, Rogelio Lozano, and Claude Pégard. "Combining Stereo Vision and Inertial Navigation System for a Quad-Rotor UAV." *Recent Developments in Unmanned Aircraft Systems* (2011): 373-87.

[Computer Vision and Geometry Lab 2016] Computer Vision and Geometry Lab. "PX4 Autopilot." *Pixhawk Flight Controller Hardware Project* (2016). https://pixhawk.org/.

[Cortes, Sole, Salvi 2011] Cortes, B. Bacca, X. Cufi Sole, and J. Salvi. "Indoor SLAM Using a Range-augmented Omnidirectional Vision." *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)* (2011).

[Google 2016] Google. "Tango." *Tango* (2016). https://get.google.com/tango/.

[Henry, Krainin, Herbst, Ren, Fox 2014] Henry, Peter, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. "RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments." *Experimental Robotics Springer Tracts in Advanced Robotics* (2014): 477-91.

[Loianno et al. 2015] Loianno, Giuseppe, Gareth Cross, Chao Qu, Yash Mulgaonkar, Joel A. Hesch, and Vijay Kumar. "Flying Smartphones: Automated Flight Enabled by Consumer Electronics." *IEEE Robotics & Automation Magazine IEEE Robot. Automat. Mag.* 22.2 (2015): 24-32.

[Riisgaard, Blas 2005] Riisgaard, Søren, and Morten Rufus Blas. "SLAM for Dummies." *Cognitive Robotics*. MIT, (2005).

[Schauwecker, Ke, Scherer, Zell 2012] Schauwecker, Konstantin, Nan Rosemary Ke, Sebastian Andreas Scherer, and Andreas Zell. "Markerless Visual Control of a Quad-Rotor Micro Aerial Vehicle by Means of On-Board Stereo Processing." *Informatik Aktuell Autonomous Mobile Systems 2012* (2012): 11-20.

[Siegwart, Nourbakhsh, Scaramuzza 2011] Siegwart, Roland, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*.Cambridge, MA: MIT, (2011).

[Strauss 2015] Strauss, Daryll. "Experiments with Project Tango." *GitHub*. (2015). https://github.com/daryllstrauss/tango.

[Sturm, Bylow, Kerl, Kahl, Cremers 2013] Sturm, J., E. Bylow, C. Kerl, F. Kahl, and D. Cremers. "Dense Tracking And Mapping With A Quadrocopter." *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-1/W2 (2013): 395-400

Vita

Tate Hawkersmith was born in Tullahoma, Tennessee. He received his B.S degree in Computer Engineering from the University of Tennessee in 2014. In 2016 he completed his M.S. degree in Computer Engineering under the supervision of Dr. Lynne E. Parker. During his time at the University of Tennessee he worked on the EcoCar2 center stack development team. His research interests include autonomous robotic platforms and computer vision. After graduation, he intends to continue working on his start-up, Virtual Xperience, which provides virtual 3D interactive walk-throughs of commercial and residential real-estate.