

https://dspace.org/

Autonomous 3D mapping and surveillance of mines with MAVs

Edwards, Stuart Robert

2017

https://hdl.handle.net/10539/25005

Edwards, Stuart Robert (2017) Autonomous 3D mapping and surveillance of mines with MAVs, University of the Witwatersrand, Johannesburg, https://hdl.handle.net/10539/25005

Downloaded from a DSpace repository. https://dspace.org/

Autonomous 3D Mapping and Surveillance of Mines with MAVs

Author: Stuart Edwards
Supervisors: Turgay Celik and Richard Klein



A Dissertation Submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, for the degree of Master of Science.

12 July 2017

Abstract

The mapping of mines, both operational and abandoned, is a long, difficult and occasionally dangerous task especially in the latter case. Recent developments in active and passive consumer grade sensors, as well as quadcopter drones present the opportunity to automate these challenging tasks providing cost and safety benefits. The goal of this research is to develop an autonomous vision-based mapping system that employs quadrotor drones to explore and map sections of mine tunnels. The system is equipped with inexpensive, structured light, depth cameras in place of traditional laser scanners, making the quadrotor setup more viable to produce in bulk. A modified version of Microsoft's Kinect Fusion algorithm is used to construct 3D point clouds in real-time as the agents traverse the scene. Finally, the generated and merged point clouds from the system are compared with those produced by current Lidar scanners.

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.



Stuart Edwards

July 2017

Acknowledgements

First and foremost, I would like to thank my supervisors, Professor Turgay Celik and Richard Klein, for their guidance over the course of my studies. Professor Celik's mentorship and patience has been a driving force for me and his vast wealth of experience has helped me overcome many an obstacle during my research. Richard Klein, despite being swamped with his own research, has always been eager to answer any questions I had or advise me when I faced difficulties. I would therefore once again like to express my deepest gratitude to Prof Celik and Richard Klein, as this dissertation would not have been possible without them.

I would also like to thank the School of Mining Engineering at the University of the Witwatersrand and more specifically the members of the Digital Mining Group for their partnership and assistance with this project. Professor Frederick Cawood's passion and enthusiasm for the project has given me much encouragement and his regular, eager assistance has been greatly appreciated. Additionally, I would like to thank Dr Bekir Genc, Tariq Feroze and Faiq Javaid for their help during the different phases of my research.

I would like to thank Jeremy Green for sharing his ideas and research findings with me, Jarryd Bekker for his technical assistance and Atif Muhammad for his frequent and useful input. Finally, I would like to thank my father, Robert Edwards, for helping me with the numerous hardware repairs that are necessary when working with an autonomous quadcopter.

Contents

1 Introduction **1**

2 Background and Related Work **4**
2.1 Introduction . . . 4
2.2 Quadcopters . . . 4
2.2.1 Overview . . . 4
2.2.2 History & Development . . . 5
2.2.3 Modern Popularity . . . 6
2.2.4 Design and Functionality . . . 7
2.3 3D Reconstruction & Mapping . . . 8
2.3.1 Overview . . . 8
2.3.2 Range imaging techniques . . . 10
2.3.3 Processing depth data . . . 13
2.3.4 Kinect Fusion . . . 21
2.3.5 RGB-D mapping . . . 24
2.4 Autonomous navigation . . . 26
2.4.1 Overview . . . 26
2.4.2 Simultaneous Localization and Mapping . . . 26
2.4.3 Path Planning . . . 29
2.5 Similar Projects . . . 30
2.6 Conclusion . . . 31

3 Research Methodology **32**
3.1 Introduction . . . 32
3.2 Equipment Setup . . . 33
3.2.1 Choice of range imaging device . . . 33
3.2.2 Choice of quadcopter . . . 33
3.2.3 The on-board computer . . . 34
3.2.4 Mounting the payload . . . 34
3.2.5 The ground station . . . 34
3.3 3D Reconstruction . . . 35
3.3.1 Choice of 3D reconstruction method . . . 35
3.3.2 Implementation . . . 36
3.4 Control System . . . 39
3.4.1 Controller Overview . . . 39
3.4.2 Autopilot . . . 39
3.4.3 PID Position Controller . . . 40
3.4.4 Navigation controller . . . 42
3.5 Experiment 1: 3D reconstruction performance . . . 43
3.5.1 Data collection . . . 43
3.5.2 Data processing . . . 44

3.6 Experiment 2: Autonomous mapping of a mine tunnel 45
3.6.1 Set up 45
3.6.2 Data Collection 45
3.6.3 Data Processing 45
3.7 Conclusion 46

4 Results and Discussion **47**
4.1 Experiment 1 47
4.1.1 Results 47
4.1.2 Discussion 47
4.2 Experiment 2 51
4.2.1 Results 51
4.2.2 Discussion 52

5 Conclusion **54**

List of Figures

1.1 Nick's tunnel, located at the school of mining engineering, at the University of the Witwatersrand. 2

2.1 The record setting Oehmichen No 2 Quadcopter designed by French engineer Étienne Oehmichen [Shosa, 2015; Krossblade, 2016]. 6
2.2 Diagram showing the various forces involved in the quadcopter flight. The front, back, left and right rotors are denoted by M_f, M_b, M_l and M_r respectively. θ, ϕ and ψ denote the pitch, roll and yaw angles of the quadrotor respectively [Raza & Gueaieb, 2010]. 9
2.3 The four methods used to alter a quadcopter's dynamics. [Raza & Gueaieb, 2010]. 9
2.4 The Z+F IMAGER 5010C, 3D Laser scanner [zflaser.com, 2016]. 12
2.5 The IR pattern used by the Microsoft Kinect [bbzippo, 2017]. 14
2.6 A diagram of a single point structured light setup and the values used for calculating the relative depth of the point from its respective disparity value [Khoshelham, 2012]. Z_0 and Z_k are the initial and final distances between the object and the camera (C) respectively. The shift between Z_0 and Z_k results in a displacement distance D between the observed point k and its expected position in the reference image. d is the disparity value corresponding to this shift. Finally, b denotes the distance between the projector (L) and the camera, and f denotes the focal length of the camera. 16
2.7 The 15 possible cases for an isosurface intersection through a cube [polytech, 2016]. 20
2.8 A reconstructed scene using the Kinect Fusion algorithm. 23

3.1 An overview of the components of the autonomous 3D mapping system. 33
3.2 The final quadcopter set up after mounting the payload. 35
3.3 An overview of the quadcopter controller. 39
3.4 Nick's tunnel and the laser scanning equipment. 44

4.1 The laser scan point cloud. 48
4.2 One of the KinFuLS Scan point clouds. 53

List of Tables

3.1 The final gains for the quadcopter position PID controller. 42

4.1 Experiment 1 results: 8 bit KinFuLS scan 49
4.2 Experiment 1 results: 16 bit KinFuLS scan 49
4.3 Experiment 2 results: Tunnel wall scans 51

List of Abbreviations

CT computerized tomography.

EDM Electronic Distance Measurement.

EKF Extended Kalman Filter.

FAST Features from Accelerated Segment Test.

GPS Global Positioning System.

ICP Iterative Closest Point.

IMU Inertial Measurement Unit.

MAV Micro Aerial Vehicle.

MRI Magnetic resonance imaging.

PCA Principle Component Analysis.

RANSAC Random sample consensus.

RC Radio Controlled.

SAD Sum of absolute distances.

SfM Structure From Motion.

SLAM Simultaneous Localisation and Mapping.

SSD Sum of squared distances.

ToF Time of Flight.

TSDF Truncated Signed Distance Function.

UAV Unmanned Aerial Vehicle.

VTOL Vertical Take Off and Landing.

Chapter 1

Introduction

Mining has always been a hazardous occupation and although the fatality rate from mining related accidents has dropped in recent times due to modern safety procedures and equipment, the profession still carries significant risk. Common hazards that miners face include physical dangers such as rock fall, falls from height and entrapment, as well as less noticeable ones such as long term exposure to various environmental factors involved in mining [Donoghue, 2004]. In South Africa, the Gold mining industry continues to display the highest fatality rate when compared to other commodity mining operations, contributing 33 of the 77 reported fatalities in the South African mining industry for 2015 [Zwane, 2016].

The mining industry in South Africa is also affected by the country's high crime rate: illegal mining is becoming more prominent in both abandoned and operational mines, and it is estimated that there are over 14 000 people involved in these activities [CoM, 2016]. It is difficult to determine the financial impact caused by these activities on legal operations and on the country as a whole, but the most credible estimates put the annual losses at R6 billion. Additionally, illegal mining is accompanied by gang related activities, prostitution, mining accidents and irreparable damage to the environment [Jamasmie, 2016].

Both the safety and the security issues faced by mines are progressively being resolved with the introduction of improved operating procedures and advanced equipment. Modern security systems, such as thermal cameras and contactless smart card access points have helped to reduce the number of crime-related incidents on mine sites over the past few years [isds.com, 2017; minealert.com, 2017]. These measures are, however, expensive and have proven to be insufficient in some regions of the world.

The Digital Mine project at the University of the Witwatersrand, School of Mining Engineering aims to take these measures further by developing a digitised "smart mine" [Cawood, 2015]. The goal of the project is to work towards the notion of an intelligent mine. Several research projects have been initiated, which include:

- Bringing real time, two-way, high-speed communications and environmental surveillance technologies that are currently available at the surface to the underground mining environment;
- Underground positioning, mapping and navigation;
- Action recognition and detection of abnormalities;
- Remote, visual inspections; and
- Visual environmental and rock monitoring.

Ultimately, the project aims to be able to fully automate a mining operation, thus removing the need to deploy people in the dangerous conditions present underground. In order to achieve this, a 67 meter long mock-tunnel, as shown in Figure 1.1, has been constructed using authentic mining materials in order to allow research and development in this space.

One of the mining processes that is the focus of research in the Digital Mine Project is that of mine surveys. Mine surveying is vital to any mining operation and the data produced by surveyors is critical during mine management, construction and planning. Total stations are one of the most popular tools in mine surveying and consist of an electronic theodolite, used for measuring horizontal and vertical angles, and an Electronic Distance Measurement (EDM) device for measuring distances. Surveyors can use this equipment to measure the locations of mine tunnel features for later analysis. Laser scanners are another useful, but expensive tool for mine surveys, enabling the surveyor to capture high precision point clouds from the scene [minesurveyor.net, 2016].



Figure 1.1: Nick's tunnel, located at the school of mining engineering, at the University of the Witwatersrand.

Despite having access to these tools, surveying mines can still be a lengthy process. Even modern laser scanners can take up to 10 minutes to complete a single scan, which may not include much of the scene when working in an underground environment. Additional measures must also be taken to ensure accurate registration of the resulting point clouds and often involve the deployment of markers in the scene.

Taking the above issues into account, this research aims to develop an affordable and autonomous indoor mapping system that uses quadrotor drones to autonomously survey and patrol a series of mine tunnels while constructing a 3D map of the tunnels in real-time. This 3D map could then be used for:

- drone navigation;
- providing management with a semi-real-time overview of the mine system, including the locations of mining staff.

Additionally, future work could look at using these maps for:

- search and rescue of injured mine workers trapped underground;
- hazard detection, such as potentially dangerous structural changes to the mine shaft;
- security surveillance, with the help of pose and facial recognition techniques.

This work develops a system using a modified version of Microsoft's Kinect Fusion algorithm [Newcombe *et al.* , 2011] to perform real-time, markerless, point cloud capture, registration and mesh reconstruction. Cloud registration is done using an optimised implementation of the Iterative Closest Point (ICP) algorithm that is designed to take advantage of modern GPU hardware. The current implementation only records projected infra-red light, which it uses to infer depth and does not record and make use of any RGB data for cloud alignment or presentation. This is due to the inherent lack of light in the underground mining environment, which adversely affects RGB-dependant systems. This dissertation does, however, investigate and compare RGB-D related methods with the implemented depth-only method.

While not on par with industrial laser scanners, the accuracy of consumer depth sensors is sufficient for use in mine management and planning, and for search and rescue operations. The rapid improvements to consumer level depth sensors also mean that the accuracy of future systems will likely approach that of current laser scanners. Current laser scanners have millimeter or even sub-millimeter accuracy, which is ideal for detecting gradual changes to the underground structure of a mine which may indicate a potential collapse. This work investigates whether this system is able to detect similar changes to the environment using only the consumer depth cameras.

As a result of consumer depth sensors' prominence in the gaming industry, there already exist a variety of libraries for object and person recognition, as well as pose estimation. These libraries and methods show the potential for use with the developed system and could allow for future research in person identification and general security. Such a system could provide smart, mobile security surveillance, which would be far more difficult to predict and avoid than traditional static surveillance systems. Pose estimation techniques and machine learning methods could potentially be used to detect suspicious behaviour of individuals recorded along an agent's route. This may help identify potential criminal activities. Object recognition could also be used to tag equipment, vehicles and people within the mine, thus allowing management to better keep track of resources and personnel.

This research is composed of two sub-problems: The problem of generating and merging 3D maps of a scene in real-time, and the problem of autonomous positioning and navigation in a GPS-denied environment. The next chapter looks at previous work done in the field of 3D mapping and autonomous navigation; and describes the techniques used in this research. Chapter 3 presents the methodology that was followed during the completion of the autonomous mapping system and describes the two experiments that were performed in order to evaluate the system. Chapter 4 lists the results of these two experiments and discusses their implications with regards to the goals of this research. Finally, chapter 5 summarises the contents and findings of this research and concludes the dissertation.

Chapter 2

Background and Related Work

2.1 Introduction

In recent years, quadcopters (quadrotor helicopters) have become increasingly popular due to their decreasing cost and growing availability. These small, four-rotor UAVs are currently used in a wide variety of applications ranging from film production to archaeological field surveys [Seitz & Altenbach, 2011]. Most commercial quadcopters are controlled manually via a radio controller (RC) or smart-phone application, but often make their underlying software API available to developers. This has allowed them to be the focus of many computer vision and artificial intelligence related projects.

The majority of quadcopters are deployed in outdoor environments, be it for hobbyist activities or for industrial applications. In these cases, the availability of GPS allows for easy automation of the quadcopters. Existing data sources, such as Google Maps [Google, 2016], can be used to implement waypoint following systems with minimal effort. For indoor applications, on the other hand, the GPS denied environments mean that autonomous quadcopters must employ other positioning techniques for navigation. In such cases, the common approach is to implement some form of Simultaneous Localisation and Mapping (SLAM), which allows the agent to map its environment and estimate its position within its surroundings.

A second technology that has grown in popularity and availability over the past decade is consumer depth cameras, such as the Xbox Kinect [Microsoft, 2015]. As with quadcopters, the emergence of these devices into the entertainment and gaming industry has helped to significantly boost the research and development of improved depth cameras, while gradually reducing retail costs.

In this research, these two technologies are employed in order to develop the autonomous indoor mapping system. This chapter presents the background and relevant studies in the context of these technologies, with focus on the aspects that are discussed in Chapter 3.

2.2 Quadcopters

2.2.1 Overview

Quadrotor helicopters (or quadcopters) are Vertical Take-Off and Landing (VTOL) Micro Aerial Vehicles (MAVs) with four fixed-pitch rotors arranged around the ends of a cross frame [Hoffmann *et al.* , 2004]. This simple, symmetric design allows the quadcopter to manoeuvre in any direction and without the need for much adjacent space. This versatility, coupled with their rapidly falling prices, has lead to a tremendous rise in popularity over the past few years, initially

as a toy and later as a research tool. Today the vast amount of attention that quadcopters and similar micro-drones receive in research and industry has led to an almost exponential growth in the capabilities of these devices that is reminiscent of Moore's Law. Section 2.2.2 recounts the important milestones in the history and development of quadrotor aircraft with a discussion of their modern popularity and applications in Section 2.2.3. Finally, the technical aspects of the quadrotor design and operation are discussed in Section 2.2.4.

2.2.2 History & Development

Unmanned Aerial Vehicles (UAVs) have been the subject of much interest from as early as 1917 [Mueller, 2009]. A number of prototype unmanned aeroplanes were commissioned and built during World War I, but the first successful unmanned aircraft, called the "Kettering Bug", was completed in 1918 by Charles Kettering and was too late to be deployed in the war. Research continued after the war, producing a number of increasingly sophisticated unmanned aerial drones over the last century. The first mass produced UAVs were used for target practice by anti-aircraft gunners during World War II [Naughton, 2016], shortly before a variety of UAVs were deployed in various combat roles.

Alongside the development of UAVs, which usually consisted of manned aircraft that were adapted for unmanned flight, came the development of Micro Aerial Vehicles (MAVs). UAVs with wingspans shorter than 6 meters and weighing under 25kg are generally classified as MAVs, although there is no exact definition [Mueller, 2009]. The small scale of MAVs added additional complexity to their development, such as the requirement of smaller combustion engines, radio components and actuators. The pre-existing model airplane community made large contributions in these areas, as well as in the design and development of MAVs as a whole. The first successful flight of a radio controlled (RC) model airplane took place in Germany, 1936. Following World War II, the popularity of model RC planes increased, leading to numerous improvements over the next few decades. It was not until the 1970s, however, that MAVs were produced for any serious applications [Mueller, 2009]. It was around this time that video camera and other remote sensing technologies became suitable for use in unmanned aviation. Over the next decade, the United States military began the research and development of small UAVs mainly for reconnaissance applications and produced a number of small unmanned aircraft in the process, some of which fall under the MAV classification.

The majority of these first MAVs were fixed winged aircraft and were not designed with high manoeuvrability in mind. Multi-rotor MAVs only became viable in the early 21st century despite multirotor aircraft being the first VTOL aircraft to be developed [Krossblade, 2016]. When engineers first experimented with the idea of VTOL aircraft at the beginning of the 20th century, a multi-rotor design was chosen for the first prototypes rather than the single main rotor design seen in modern helicopters. The main rotor of modern helicopters generates torque that must be counter balanced by a secondary tail rotor. Early VTOL pioneers saw this as an inefficient use of engine power and opted for a multi-rotor design instead. Early experiments however, proved the quadrotor design to be highly unstable and completely infeasible for practical use. The most successful quadcopter during this period was the Oehmichen No 2 shown in Figure 2.1. It was developed by Étienne Oehmichen, who set a world record by flying it a distance of 360m [QuadcopterArena, 2015]. More efficient quadcopter designs were produced over the following decades, but were temporarily replaced with single rotor designs for the following reasons:

1. The rotors of a quadcopter must be constantly adjusted while in use in order to provide stable flight. This made quadcopters very unstable and made the piloting of the quadcopter an extremely difficult endeavour. Single rotor helicopters, on the other hand, have their center of gravity below the main rotor, which allowed them to easily maintain stable flight

[Krossblade, 2016].

2. The size of early combustion engines meant that it was only feasible for a quadcopter to carry a single engine, usually positioned at the center of the quadcopter. The engines torque then had to be transferred to each of the four rotors using shafts and belts. This added to the mass of the vehicle and added undesirable weak points to the quadcopter. The engines of single rotor helicopters are also positioned centrally within the vehicle, but can be connected almost directly to the main rotor, which is directly above it.

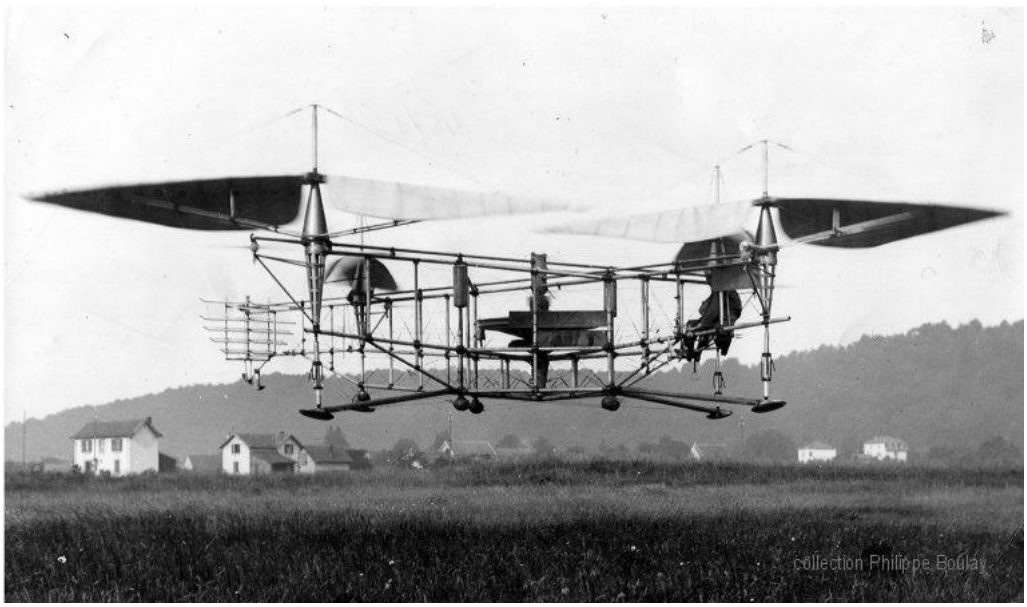


Figure 2.1: The record setting Oehmichen No 2 Quadcopter designed by French engineer Étienne Oehmichen [Shosa, 2015; Krossblade, 2016].

Roughly a century after the first quadcopters were prototyped, modern electronics had advanced to the point where developing stable multi-rotor aircraft was feasible. The existence of microelectronics meant that stable unmanned quadcopters could be built at a considerable fraction of the size and cost of the multi-rotor aircraft of old. Small electric motors could be placed directly under each of the four rotors and on-board electronics could act as the flight controller in order to regulate the rotors and maintain stability. The resulting quadcopters that we see today are therefore small enough to be classified as MAVs, cheap enough to appeal to hobbyists and researchers alike, and manoeuvrable enough to be flown both indoors and outdoors.

2.2.3 Modern Popularity

The past few years have seen a massive surge in the consumer drone industry, with an estimated 700 000 quadcopters sold in 2015, marking a 63% increase from 2014 [CNBC, 2016]. While the current value of the drone industry is estimated to be around $3.3 billion, it is projected to hit $90 billion by 2025 [Inc.com, 2015].

As this new technology has begun to flood the world markets, authorities are still in the process of developing laws and regulations to keep these potentially dangerous tools in check. Before the emergence of multi-rotor MAVs onto the market, the majority of unmanned aircraft belonged to the model airplane community. This almost century old community follows a strict set of regulations and community members usually belong to clubs where they are provided with a safe and open airspace in which to fly. There have thus been extremely few accidents involving

model airplanes over their lengthy history. On the other hand, the wide availability of multi-rotor drones to the general public has already lead to numerous, potentially fatal, incidents mostly involving passenger aircraft. The ease of use of these drones and the still limited regulations on their use has increased the likelihood of amateur drone pilots causing harm or flying into dangerous areas, such as airports. Gettinger & Michel [2015] studied 921 drone-related incidents that were reported by aircraft pilots during the period of 2014-2015 and found that there were 327 close encounters during this time. Furthermore, in 12 of these incidents, the pilot was forced to take action in order to evade the drone. Fortunately there have been no reported fatalities yet, but even a single collision with a multi-engine passenger plane could result in hundreds of deaths.

Potential risks aside, the development of MAV drones has opened up a wide range of opportunities for both hobbyists and professionals. The high manoeuvrability and low cost of quadcopters means that they have a wide range of applications, both in indoor and outdoor environments. The gradually increasing carrying capacity of these drones has made them ideal for use in the film industry and have almost become part of any film crew's standard equipment. In fact, the use of quadcopters for aerial photography is not just limited to professionals, with products such as the DJI Phantom series [DJI, 2016] allowing amateur film makers and photographers to capture high, sweeping shots that would have once required the expensive use of a helicopter. Indeed, the ability of a quadcopter to act as a portable eye-in-the-sky has proven highly useful in many sectors. Police forces in some areas have begun using drones in situations requiring reconnaissance, such as sting operations or search and rescue efforts. Southern Africa has also seen drones used in anti-poaching efforts where impromptu aerial views provide invaluable information to security staff [savetherhino.org, 2015]. Many such applications have been investigated over the past decade with extensive efforts in drone-related research and commercial ventures in progress today. The popularity of quadcopters is therefore unlikely to be a mere passing fad and these devices should prove to be powerful and versatile tools in years to come.

2.2.4 Design and Functionality

As mentioned in Section 2.2.1, quadcopters consist of a cross-shaped frame, with a rotor at each of the four ends. Each of the four rotors is powered by a dedicated electric motor and the remaining electronics, including the flight computer and sensor payload, are situated at the center of the quadcopter frame. Other multi-rotor drone designs follow a similar format with an arm for each rotor extending outwards from the drone's center of mass. Figure 2.2 on page 9 shows the basic design of a quadcopter, as well as the angles that define its roll (ϕ), pitch (θ) and yaw (ψ).

Apart from the one or two processors used for flight control, quadcopters are usually equipped with a suite of sensors. This sensor payload may differ from drone to drone depending on the quality and intended uses of the drone, but almost always includes:

- An Inertial Measurement Unit (IMU) comprised of:
  - An accelerometer for measuring the linear acceleration of the quadcopter along 3 axes;
  - A Gyroscope for measuring the angular acceleration of the quadcopter around 3 axes;
- A Barometer for measuring the air pressure in order to estimate the drone's altitude; and
- A compass (Magnetometer) for determining true north in order to estimate the drone's heading.

A GPS module is also commonly found on many drones, but is not vital for the drone's operation and is only functional outdoors. In industrial and research applications, a quadcopter

may be fitted with specialised sensors in addition to these standard sensors. These often include various distance sensors, including ultrasonic sensors and laser range finders. Occasionally optical flow sensors are also used, which help to maintain a drone's position in GPS denied environments. The final major component of a quadcopter is its LiPo battery, which is often the heaviest component on the drone. A drone's battery determines the maximum flight time of the drone, with a larger capacity battery providing a longer flight time. This is only true up to a point however, as the weight of a battery is proportional to its capacity. The combined weight of a drone (including the battery) and its payload have a significant impact on the flight time thereof and so there is a trade-off between a drone's payload capacity and its range.

Unlike the single rotor helicopter, quadcopters only possess upwards-facing, fixed-pitch rotors and are not equipped with any form of tail rotor. The flight dynamics of quadcopters is therefore naturally different from that of the traditional, single rotor helicopter. There are a number of different helicopter designs, but the most common design features a single main rotor mounted above the fuselage and a secondary anti-torque rotor mounted on the tail, perpendicular to the main rotor. The pilot is able to control the pitch and roll of the aircraft by tilting the individual blades of the main rotor, thus changing the thrust vector. By altering the pitch of the tail rotor, the pilot is also able to adjust the yaw of the vehicle.

The rotors of a quadcopter, on the other hand, usually consist of a single piece of plastic and, hence, provide no control over the individual blades. Instead, the MAV is controlled by adjusting the relative speeds of the four rotors, as shown in Figure 2.3 on the next page. The front and back rotors (M_f and M_b) rotate in the opposite direction to the left and right motors (M_l and M_r) in order to balance the torque generated by each pair of rotors [Raza & Gueaieb, 2010]. The altitude can be controlled by simply increasing or decreasing the thrust of all four motors simultaneously. The pitch is adjusted by changing the thrust of the front rotor while sending the opposite command to the back rotor, thus pitching the MAV forwards or backwards, while keeping the resultant thrust constant. Similarly, the quadcopter can roll left and right by adjusting the relative thrusts of the left and right rotors. Finally the yaw is controlled by adjusting the relative rotor speeds of the two rotor pairs. The rate of change of a quadcopter's yaw is proportional to the sum of the torques of its rotors. When the torques of the clockwise pair and the counter-clockwise pair sum to zero, the yaw of the quadcopter is kept constant. If the combined torque of the clockwise pair is not equal to that of the counter-clockwise pair, the yaw will change accordingly. For example, increasing the speed of the clockwise pair and decreasing the speed of the counter-clockwise pair, will cause the quadcopter to rotate clockwise without changing its total thrust.

The quick and precise control over the four rotors has only been made possible in the last two decades by the decreasing cost and growing capabilities of microprocessors. Quadcopters are equipped with an onboard control system that performs the necessary functions required for stabilisation and manoeuvres at a sufficient frequency. By using sensor data from onboard sensors such as the IMU and GPS, the quadcopter is usually able to maintain its stability and position without input from the pilot.

2.3 3D Reconstruction & Mapping

2.3.1 Overview

The camera has been around for over 300 years and has allowed the capture and distribution of vast amounts of visual information from the world around us. With modern image processing and machine learning techniques, it is possible to extract some additional information from standard images. Despite this, the 2 dimensional representation of the world that traditional

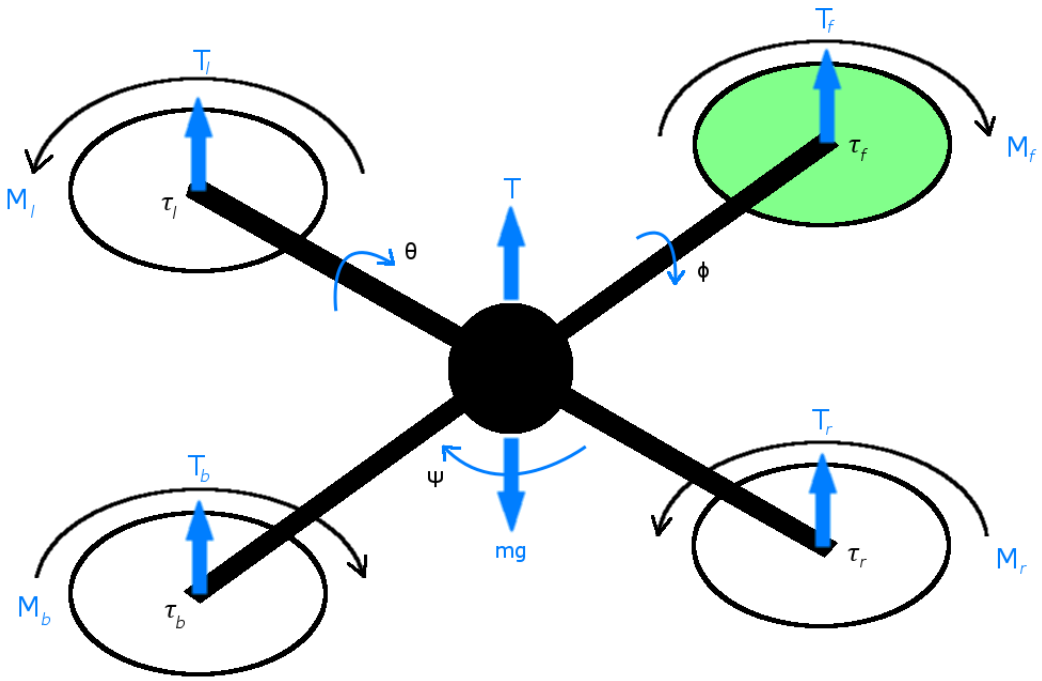


Figure 2.2: Diagram showing the various forces involved in the quadcopter flight. The front, back, left and right rotors are denoted by M_f, M_b, M_l and M_r respectively. θ, ϕ and ψ denote the pitch, roll and yaw angles of the quadrotor respectively [Raza & Gueaieb, 2010].

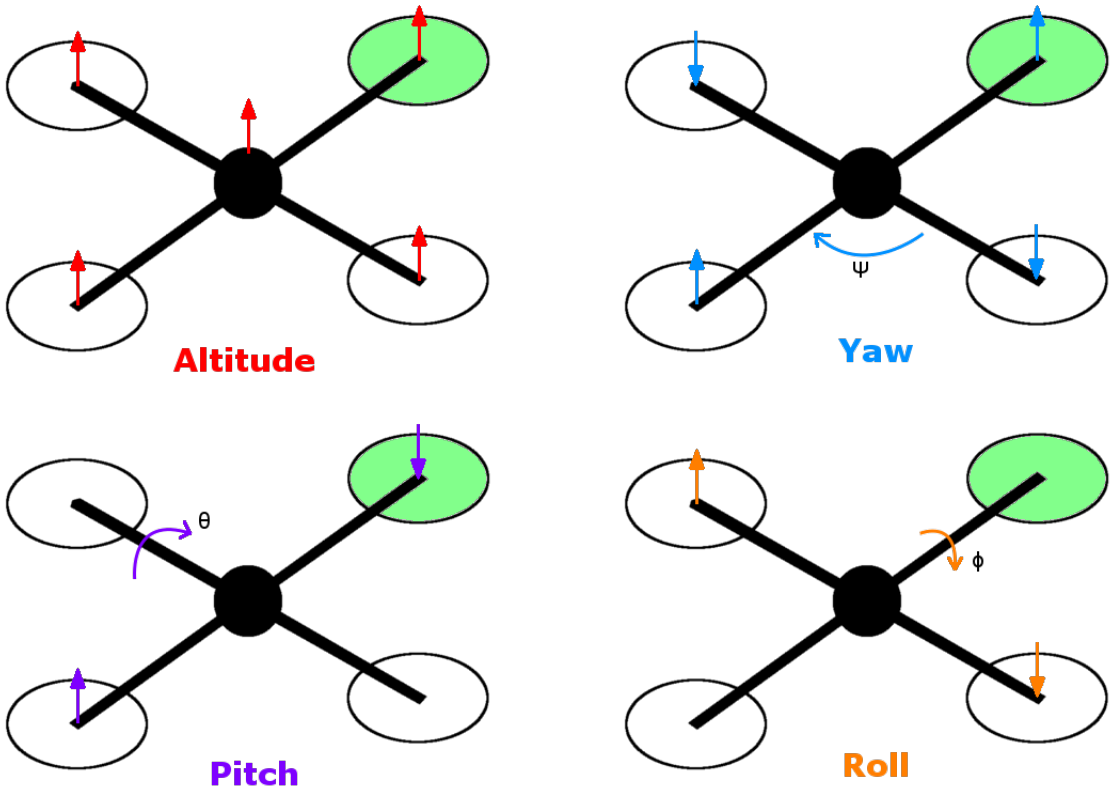


Figure 2.3: The four methods used to alter a quadcopter's dynamics. [Raza & Gueaieb, 2010].

cameras offer is still severely limiting in many applications. We live in a 3D world and while our brains are capable of inferring some 3D information from images, there is still much information that is lost entirely when the world is projected onto a 2D space. Computers have an even harder time using visual cues to infer structure and depth from images, as most current systems do not use contextual information or prior knowledge during this process. Hence there is the need to use specialised hardware, either in addition to or instead of traditional cameras in order to capture 3D information. Furthermore, specialised software is required both for capturing the 3D information and for processing it afterwards.

3D reconstruction is the process of generating virtual representations of real-world objects and is one of the major challenges in computer vision [Zhu *et al.* , 2008]. These reconstructions are extremely useful across many fields, but are especially applicable in robotics. In order to successfully navigate its environment without colliding with obstacles, a robot needs to be able to estimate its distance from other objects or even its 3D position within its surroundings. Until recently, successful 3D mapping systems have required expensive equipment, such as laser scanners, thus limiting their use in research. With the recent release of inexpensive depth sensors in the gaming industry, such as the Microsoft Kinect [Microsoft, 2015], this is no longer the case and this area of research has become more accessible. Along with the appearance of these consumer-level depth sensors, a number of open-source libraries such as the Point Cloud Library (PCL) [Rusu & Cousins, 2011], have begun to emerge and allow developers to fully exploit the abilities of these devices. Coupled with existing computer vision libraries, such as OpenCV [Bradski, 2000], this has provided researchers and developers with a wide range of powerful tools in the field of 3D reconstruction and mapping.

The method of acquiring 3D data (called range imaging) can either be active or passive depending on whether or not the method interferes with the real-world object being reconstructed. Active methods often emit some form of wave, be it electromagnetic or ultrasound, and analyse the reflections in order to infer the distance between the sensor and the object. Passive methods, on the other hand, analyse pre-existing waves or other phenomena from the target object in order to infer depth [Buelthoff & Yuille, 1991]. Once the data has been collected, it is usually processed to create a depth map, which is a 2D image in which each pixel represents the distance from the sensor to that point in the real world. The data can also be represented as a set of 3D points representing the surface of the environment or object, called a point cloud. Finally, a point cloud can be further processed to generate a 3D mesh of the scene, which is more aesthetically pleasing and easier for people to analyse.

2.3.2 Range imaging techniques

Stereo Vision

In stereo vision, images taken from two or more different positions within a scene are compared in order to extract 3D information about it. Usually a stereo vision set-up consists of two cameras that are displaced at small, fixed distance from one another on a rigid frame. By using the known distances between the cameras and the disparity between the two images captured by these cameras, it is possible to extract depth information about the scene [Brunet & Chao, 2015]. Stereo vision usually involves solving the following two problems [Lemmens, 1988]:

- The correspondence problem: The problem of determining which pixels in the two or more images correspond to the same real-world feature.
- The calibration problem: The problem of obtaining the intrinsic parameters of the cameras used and pre-processing the obtained images to remove distortions and make them appear co-planar.

Stereo vision set-ups are generally cheap, but inaccurate due to noise and small errors in the camera calibrations. Additionally, this method produces very poor results for scenes with bad lighting conditions or few interest points.

Structure from motion

Humans use a variety cues, both binocular and monocular, to perceive depth in their environment [Goldstein, 2013]. Computer stereo vision is similar to the biological process of stereopsis, in which 3D information is obtained by comparing the images received by two eyes. Humans are also able to obtain a lot of 3D information from monocular cues, such as how images change over time when the observer or the objects in their environment are moving [Shapiro & Stockman, 2001].

In computer vision, structure from motion (SfM) is the process of recovering depth information from a sequence of 2D images captured by a moving camera [Vedaldi *et al.* , 2007]. As in stereo vision, SfM requires the correspondence problem to be solved and additionally imposes constraints or assumptions on the scene in order to resolve inherent ambiguities involved in the SfM problem [Robertson & Cipolla, 2008]. SfM is thus a more difficult problem than most other range imaging methods and is usually reserved for applications in which there is a large distance between the scene and the camera, such as aerial surveys.

LiDAR

Light Detection and Ranging (LiDAR) [Oceanservice.noaa.gov, 2016] is a range imaging technology that has been popular since the 1960s and although alternative technologies have since been developed, LiDAR is still used extensively today. A LiDAR scanner typically pulses infrared laser beams off a mirror and towards a target while measuring the time the pulse takes to reach the target point and reflect back to the mirror [Lidar-uk.com, 2016]. Using the measured time and the known speed of light, it is then possible to calculate the distance that the pulse travelled. Since a single pulse can only measure a single point, the mirror needs to be rotated at a high frequency in order to scan the desired area. When used in conjunction with moving vehicles, the vehicle's IMU and GPS data is used to stitch the data together.

Depending on the quality of the device, LiDAR can produce dense, highly accurate point clouds at rates that can vary from the millihertz range to the megahertz range. The Z+F Imager 5010C LiDAR device shown in Figure 2.4 on the following page for example, is able to measure upwards of 1 million pixels per second. They have therefore been the preferred range imaging technology for decades. The cost of these devices, however, is a huge obstacle in many applications, with prices usually ranging in the hundreds of thousands of dollars. While these prices are affordable to many of the companies that require LiDAR, they also make the use of LiDAR in fields such as robotics a difficult and costly choice. Even low resolution laser scanners can significantly increase the cost to produce a robot. This makes it difficult to use them in areas in which one desires to use a robot model en masse, such as in swarm robotics.



Figure 2.4: The Z+F IMAGER 5010C, 3D Laser scanner [zflaser.com, 2016].

Time of Flight Cameras

Time of Flight (ToF) Cameras form a sub-class of LiDAR, which use emitted and timed pulses of light to calculate distances. The major difference between traditional ToF cameras and traditional LiDAR is that ToF cameras are scannerless, and thus non-mechanical [Iddan & Yahav, 2001]. Conversely, LiDAR devices usually measure depth in a point-to-point fashion by rotating the instruments to scan the scene, ToF cameras illuminate the scene in a single pulse of infra-red (IR) light and analyse the entire reflection in order to calculate each pixel of the resulting depth image. This difference makes ToF cameras superior to LiDAR in terms of speed, but inferior in terms of resolution. While ToF cameras can reach frame rates of up to 160 fps, they generally operate at resolutions of around 320×240 pixels [Design, 2016]. This equates to a maximum measurement rate of $12,288,000$ pixels per second, which is an order of magnitude higher than the device in Figure 2.4.

ToF cameras initially came with a significant price tag, but recent uses of these devices in the gaming & entertainment industry have brought it down to an affordable level, making them significantly cheaper than traditional LiDAR products. A popular example of a consumer ToF camera is Microsoft's Kinect v2 [Microsoft, 2015], which is both easily affordable by the general public and reliable enough for use in commercial applications.

ToF cameras are active sensors as they project light onto the environment that they are measuring. As a result, these cameras have a number of drawbacks that occasionally hinder their use in research. Surfaces that are either translucent or specular will produce incorrect depth readings for ToF cameras, as these cameras are not designed to handle IR reflections or refractions [Hansard, 2013]. External sources of IR light also interfere with depth readings as the cameras cannot differentiate between emitted IR radiation and that from global illumination. Since the sun is a massive source of radiation across the electromagnetic spectrum, outdoor applications of ToF cameras are severely limited. This also means that multiple ToF cameras can interfere with each other, putting heavy limitations on the simultaneous use of these cameras.

Structured Light

Structured light sensors share many similarities with both stereo vision and ToF cameras. A core challenge in stereo vision is the correspondence problem, which involves matching each pixel in the left image with the pixels in the right image [Jecić & Drvar, 2003a]. Structured light devices greatly simplify this problem by projecting unambiguous features onto the scene in the form of a light pattern [Jecić & Drvar, 2003b]. The chosen light pattern often varies across different applications, but is most commonly chosen to be either a stripe pattern, a grid matrix or a speckle pattern, such as in Figure 2.5 on the following page. While a wide range of light wavelengths (both coherent and incoherent) can be used, near visible infra-red wavelengths are most common as they are invisible to the human eye, but still detectable by most digital cameras. Once the light pattern is projected onto the scene, the distorted result is recorded by an IR camera. By comparing the recorded pattern with a reference pattern, the sensor produces an image representing the disparity between the corresponding features in the two images (disparity map).

Early structured light cameras comprised of a single camera and a single pattern projector, with the latter effectively acting as a secondary camera in a stereo vision setup [Jecić & Drvar, 2003b]. Nowadays, structured light devices with two cameras and a single projector are more commonly found, as this allows for the application of photogrammetric principles that require multiple viewpoints. The projector often uses a single laser beam, which is then split by a diffraction grating in order to create the projected light pattern. Unlike stereo vision cameras, structured light cameras are, by definition, active sensors. In fact, the need for structured light sensors to illuminate the scene with IR light is reminiscent of ToF cameras, and indeed these two approaches share many of the same advantages and disadvantages.

Like ToF cameras, structured light does not involve any mechanical parts and therefore has a higher frame rate than LiDAR. The resolution of structured light cameras is slightly higher than most ToF cameras, but still significantly lower than most LiDAR devices. The heavy reliance on projected IR light also means that structured light cameras suffer from the same environmental challenges that ToF cameras face, including global illumination, and surface reflections and refractions. Additionally, because the intensity of the projected pattern quickly decreases with distance from the camera, the range of structured light cameras is limited.

The biggest advantage that structured light cameras share with ToF cameras is their low cost. In fact, with just a projector and a camera, it is possible to perform structured light scans without the need for any specialised equipment at all [FabCentral, 2016]. It is unsurprising then that structured light cameras are amongst the cheapest 3D scanning technologies available on the consumer market. Two of the most widely available structured light cameras are the ASUS Xtion Pro Live [Asus.com, 2015] and the Microsoft Kinect v1 [Microsoft, 2015]. Both of these cameras are lightweight and compact, with the Xtion Pro being the smaller of the two. This makes them ideal for use in robotics, where the weight of an agent's payload is a critical factor.

2.3.3 Processing depth data

The raw output of range imaging devices can vary from device to device, but this section focuses on the output and post-processing of structured light cameras. Like many range imaging sensors, the output of structured light cameras is in the form of a disparity map. The disparity between two corresponding pixels in two different images is given by their difference in position when the images are superimposed. The most common method for calculating a disparity map involves the use of either a sum-of-squared-differences (SSD) or sum-of-absolute-differences (SAD) window operation on two images. For each pixel, the SSD window calculates the difference in pixel intensities between the the first image and the second, starting at the location of the target

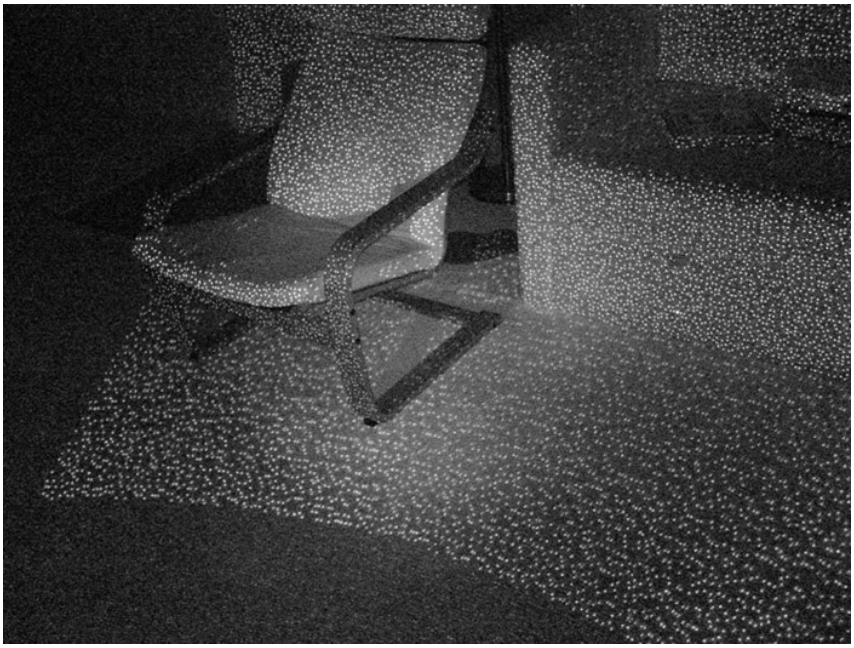


Figure 2.5: The IR pattern used by the Microsoft Kinect [bbzippo, 2017].

pixel and moving outwards horizontally. The window shift value that results in the smallest SSD value for a pixel is the disparity value of that pixel [Rambhia, 2017]. A disparity map then represents the set of disparities between the pixels in two images. Since a pixel's disparity is inversely proportional to the depth of the pixel, a disparity map can be used to calculate a corresponding depth map.

Computing the depth map

Figure 2.6 shows a simple structured light setup in which a single reference point is projected onto an object at k [Khoshelham, 2012]. In this diagram, the IR camera (C) is positioned at the origin, facing along the Z-axis. The distance between the projector and the camera is measured along the X-axis and is denoted by b. The object is moved forwards from some reference plane at a distance Z_0 from the camera to a distance Z_k. D is the resulting displacement between the observed point, k, and its expected position in the reference image, and d is the measured disparity in the image plane. Finally, the focal length of the camera is given by f and is an intrinsic parameter of the sensor, along with b and Z_0.

Applying theorems for similar triangles gives us:

$$\frac{D}{b} = \frac{Z_0 - Z_k}{Z_0} \tag{2.1}$$

and:

$$\frac{d}{f} = \frac{D}{Z_k} \tag{2.2}$$

which can be rearranged to give:

$$D = \frac{dZ_k}{f} \tag{2.3}$$

Next, Equation 2.3 is substituted into Equation 2.1:

$$\frac{dZ_k}{fb} = \frac{Z_0 - Z_k}{Z_0} \tag{2.4}$$

The depth of point k is given by Z_k, so we rearrange equation 2.4 to obtain the final expression for depth in terms of disparity:

$$Z_k = \frac{Z_0}{1 + d\frac{Z_0}{fb}} \tag{2.5}$$

Using this equation, it is then fairly easy to obtain a depth map from a given disparity map, assuming that the intrinsic camera parameters are known for the device. This principle applies not only to structured light sensors, but also to other range imaging devices that produce disparity maps, such as stereoscopic sensors.

The remaining two components of the point k's 3D coordinate can be found using the coordinates of the principle point on the reference plane (x_0 and y_0) and the lens distortions coefficients (δ_x and δ_y), all of which are included in the camera's intrinsic parameters. Since Z_k and f give the imaging scale for point k [Khoshelham, 2012], the X and Y coordinates of the 3D point can be expressed in terms of their corresponding image coordinates (x_k and y_k):

$$X_k = -\frac{Z_k}{f}(x_k - x_0 + \delta_x), \tag{2.6}$$

$$Y_k = -\frac{Z_k}{f}(y_k - y_0 + \delta_y). \tag{2.7}$$

Equations 2.5, 2.6 and 2.7, therefore, give the 3D coordinate of point k in object space. In real-world devices, which project more than one point, the set of calculated 3D points forms a 3D point cloud, which can either be used as is or processed further in order to obtain a format that is easier for humans to work with. If a depth map is accompanied by RGB data, it is also possible to colour a point cloud by simply applying pixel values in the RGB image to the points generated from the pixels in the corresponding depth or disparity map.

Point cloud registration

Point clouds can be used to represent the current view of a depth camera in real-time and information can be extracted from them on a frame-to-frame basis, with no need to keep the depth images or point clouds from the previous frame. An example of this is the pose estimation software that is used in conjunction with the Microsoft Kinect for gaming applications [Shotton *et al.* , 2013]. Certain applications, however, require information that is spread over many consecutive depth frames or point clouds, and thus need to retain some or all of the features from each frame. Two such examples of this are the 3D scanning of objects and 3D mapping of environments. In these two cases, the final output of the system should include depth information captured from multiple perspectives and locations. The complete 3D point cloud of a scanned object should be the set of point clouds generated after recording the object from various angles and the 3D map of a scanned area would include point clouds scanned at different locations.

There is therefore the need to merge consecutive point clouds, either online or offline, to obtain a more complete representation of an object or scene. The aforementioned applications that require multiple point clouds naturally involve translating and/or rotating the depth camera within the scene to capture it from multiple perspectives. The resultant point clouds are therefore not in a common reference frame and thus cannot be immediately merged to obtain the final point cloud. Ideally, the agent responsible for the range imaging is equipped with an IMU and/or GPS in addition to the depth sensor. In this case, the GPS data and integrated IMU data can be used to pinpoint the orientation and position of the depth camera for each frame. It is then simply a matter of using this information to transform each point cloud into a common reference frame before merging them.

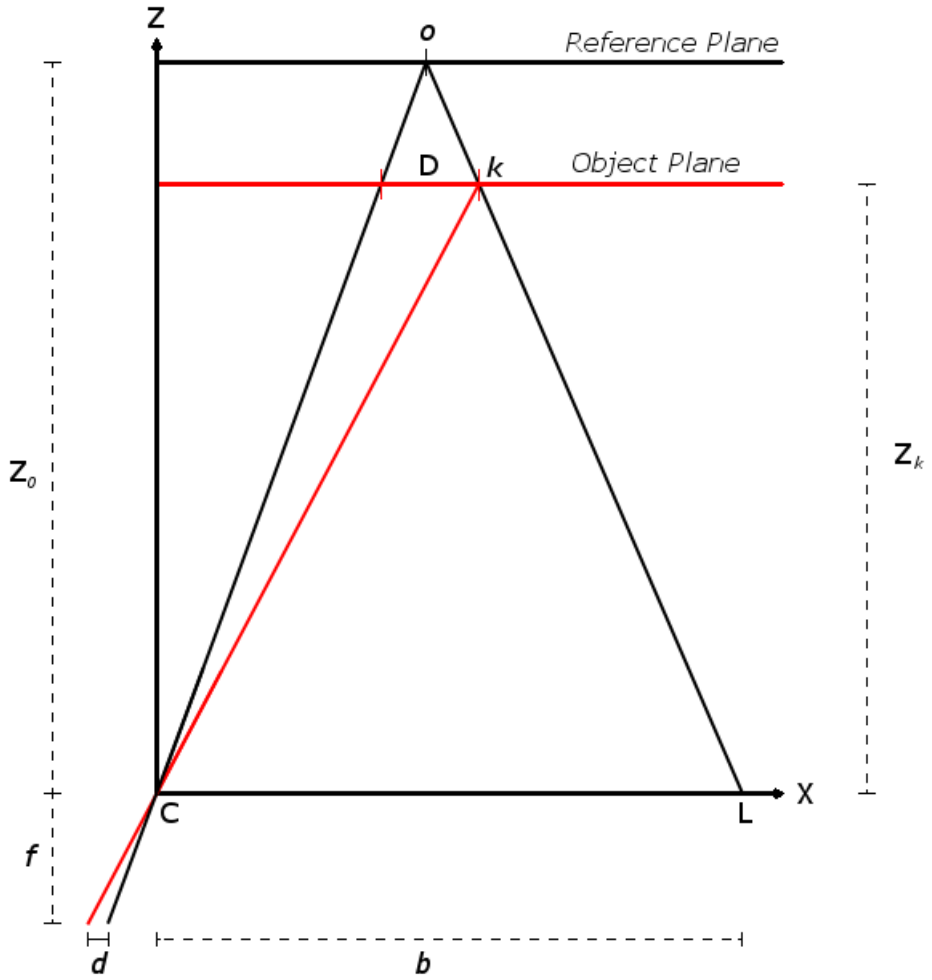


Figure 2.6: A diagram of a single point structured light setup and the values used for calculating the relative depth of the point from its respective disparity value [Khoshelham, 2012]. Z_0 and Z_k are the initial and final distances between the object and the camera (C) respectively. The shift between Z_0 and Z_k results in a displacement distance D between the observed point k and its expected position in the reference image. d is the disparity value corresponding to this shift. Finally, b denotes the distance between the projector (L) and the camera, and f denotes the focal length of the camera.

When only the point clouds are available, without additional localisation information, the task of transforming the clouds into a common frame of reference becomes more complicated. A number of algorithms have been developed over the passed few decades in order to solve this problem with varying results. One of the most widely used registration algorithms today is the iterative closest point (ICP) algorithm that was first developed by Besl & McKay [1992]. The purpose of ICP is to find the optimal rotation and translation for a given 3D data set that aligns it with a target 3D model and while it was initially proved to converge, it only did so under certain constraints. Champleboux *et al.* [1992] noted that the algorithm sometimes failed for non-identical data sets, with the failure rate increasing the less the data sets overlapped before the transformations were applied. Over the following two decades, many variations of the initial ICP algorithm were developed; each optimising or addressing flaws in the original. Around the year 2000, when laser scanners began to replace sonar in robotics, research involving ICP exploded and hundreds of additional variations and applications for the algorithm were found [Pomerleau *et al.* , 2015].

The wide availability of differing ICP variations means that it can be difficult to select the optimal variation for a given application. One of the simplest variants that is widely used is the ICP point-to-point algorithm first described by Rusu [2010a]. This algorithm takes a nearest neighbours approach to find corresponding points and is therefore useful when working with point clouds only (as opposed to both point clouds and surfaces). Given a source cloud P and a target cloud Q with N points, we define the nearest neighbour distance for each point $\mathbf{p_j} \in P$ and its nearest neighbour $\mathbf{q_i} \in Q$ as:

$$k_j = \arg\min_i ||\mathbf{p_j} - \mathbf{q_i}||^2, \tag{2.8}$$

where $i \in [1, ..., N]$ [Bellekens *et al.* , 2014]. In order to find a transformation that registers the two point clouds, we need to find a rotation R and a translation T that aligns the source cloud P with the target cloud Q. We therefore define an error function in terms of these values and Equation 2.8:

$$E_{pp} = \sum_{i=1}^{N} ||(R\mathbf{p_i} + T) - \mathbf{q_i}||^2. \tag{2.9}$$

This point-to-point ICP algorithm works best when the two point clouds are identical, as there exists a perfect transformation the aligns P with Q. When the clouds contain outliers and noise, however, the accuracy of the algorithm can rapidly drop. The ICP point-to-surface algorithm takes these problems into account when it improves upon the point-to-point variation [Bellekens *et al.* , 2014]. In point-to-surface ICP, it is assumed that the point clouds represent the surface of an object or scene, and further assumed that the local neighbours of each point in a cloud are co-planar. The surface of a local plane can then be defined by its normal vector, $\mathbf{n}$, which is the eigenvector corresponding to the smallest eigenvalue of the covariance matrix of the local points. The error function is then defined in terms of the scalar projection of point $\mathbf{p_i}$ onto a surface defined by $\mathbf{n_i}$ in Q,

$$E_{ps} = \sum_{i=1}^{N} ||((R\mathbf{p_i} + T) - \mathbf{q_i})\mathbf{n_i}||^2. \tag{2.10}$$

By composing the rotation and translation matrices into a single transform M such that

$$M = T \cdot R \tag{2.11}$$

$$= \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.12}$$

where R is written in terms of rotations of α, β and γ radians around the x, y and z axes respectively

$$\begin{aligned}
r_{11} &= \cos\gamma\cos\beta, \\
r_{12} &= -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha, \\
r_{13} &= \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha, \\
r_{21} &= \sin\gamma\cos\beta, \\
r_{22} &= \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha, \\
r_{23} &= -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha, \\
r_{31} &= -\sin\beta, \\
r_{32} &= \cos\beta\sin\alpha, \\
r_{33} &= \cos\beta\cos\alpha.
\end{aligned}$$

The error function is then,

$$E_{ps} = \sum_{i=1}^{N} ||(M\mathbf{p_i} - \mathbf{q_i})\mathbf{n_i}||^2. \tag{2.13}$$

In order to minimize the error function and hence align the clouds, we need to find an optimal transformation (M) between the clouds. This is done iteratively, with each successive iteration improving on the previous estimates for R and T. To solve for M in each iteration, we use the approximation that for some angle $\theta = 0$, we have $\sin\theta \approx 0$ and $\cos\theta \approx 1$ [Low, 2004]. Thus if $\alpha, \beta, \gamma \approx 0$ then,

$$\begin{aligned} R(\alpha,\beta,\gamma) &= \begin{pmatrix} 1 & \alpha\beta - \gamma & \alpha\beta + \gamma & 0 \\ \gamma & \alpha\beta\gamma + 1 & \beta\gamma - \alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &\approx \begin{pmatrix} 1 & -\gamma & \beta & 0 \\ \gamma & 1 & -\alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \end{aligned}$$

Substituting into M then gives,

$$\hat{M} = \begin{pmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ -\beta & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.14}$$

Using this transformation matrix with Equation 2.13, we obtain an equation to approximate the optimal transform in each iteration,

$$\hat{M}_{opt} = \arg\min_{\hat{M}} \sum_{i=1}^{N} ||(M\mathbf{p_i} - \mathbf{q_i})\mathbf{n_i}||^2. \tag{2.15}$$

Writing the inner terms of the summation in terms of the update parameters gives,

$$\begin{aligned} (Mp_i - q_i)n_i = \\ &[\alpha(n_{iz}p_{iy} - n_{iy}p_{iz}) + \beta(n_{ix}p_{iz} - n_{iz}p_{ix}) + \gamma(n_{iy}p_{ix} - n_{ix}p_{iy}) + \\ &n_{ix}t_x + n_{iy}t_y + n_{iz}t_z] - \\ &[n_{ix}q_{ix} + n_{iy}q_{iy} + n_{iz}q_{iz} - n_{ix}p_{ix} - n_{iy}p_{iy} - n_{iz}p_{iz}]. \end{aligned}$$

Writing this in terms of a matrix expression, we get

$$Ax - b, \tag{2.16}$$

where

$$b = \begin{pmatrix} n_{1x}q_{1x} + n_{1y}q_{1y} + n_{1z}q_{1z} - n_{1x}p_{1x} - n_{1y}p_{1y} - n_{1z}p_{1z} \\ n_{2x}q_{2x} + n_{2y}q_{2y} + n_{2z}q_{2z} - n_{2x}p_{2x} - n_{2y}p_{2y} - n_{2z}p_{2z} \\ \vdots \\ n_{Nx}q_{Nx} + n_{Ny}q_{Ny} + n_{Nz}q_{Nz} - n_{Nx}p_{Nx} - n_{Ny}p_{Ny} - n_{Nz}p_{Nz} \end{pmatrix}, \tag{2.17}$$

and

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & n_{1x} & n_{1y} & n_{1z} \\ a_{21} & a_{22} & a_{23} & n_{2x} & n_{2y} & n_{2z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & n_{Nx} & n_{Ny} & n_{Nz} \end{pmatrix}, \tag{2.18}$$

where
$a_{i1} = n_{iz}s_{iy} - n_{iy}s_{iz}$,
$a_{i2} = n_{ix}s_{iz} - n_{iz}s_{ix}$,
$a_{i3} = n_{iy}s_{ix} - n_{ix}s_{iy}$.

Finally, x is simply the update parameter vector,

$$x = \begin{pmatrix} \alpha & \beta & \gamma & t_x & t_y & t_z \end{pmatrix}^\top. \tag{2.19}$$

Thus, the task of minimizing Equation 2.15 with respect to $\hat{M}$ requires that we solve

$$x_{opt} = \arg\min_x (Ax - b)^2, \tag{2.20}$$

which can be solved using standard singular value decomposition. Transforming the source cloud using the approximated transformation may change the cloud's nearest neighbours, thus the nearest neighbours are recomputed in the next iteration and a new transform is estimated. This iterative process continues until the maximum number of iterations is reached or the transform becomes similar to that of the previous iteration by a predefined threshold value.

Surface Reconstruction

While point clouds can be used directly, it is often preferable to convert them to a mesh to make them easier to work with. Many 3D applications also do not support point clouds or provide few tools for manipulating them, making point cloud meshing a requirement. This process of recovering geometric representations of real world objects is called surface reconstruction. Research interest in this area has grown in recent years following the rapid increase in availability and use of point cloud data. As such, there are a number of surface reconstruction techniques available, many of which are designed for specific types and conditions of point cloud data.

In many cases, point clouds are accompanied by additional information that is either recorded alongside the depth information, can be calculated from known information or is an intrinsic property of the point cloud. Depending on the quality of the point cloud, it can be possible to determine the surface normals for each point in the cloud, which can be extremely useful when reconstructing the surface [Berger *et al.* , 2014]. By analysing the properties and outputs from the scanner itself, one can usually obtain additional information that is useful in surface reconstruction, such as the orientation of surface normals. Many scanners also record RGB information as well as depth information, which can be used to augment the accuracy of the point clouds and provide surface texture information.

An important feature in most surface reconstruction algorithms is the signed distance function of the point cloud surface [Chang, 2008]. Given a 3D point cloud B, if D is the minimum sub-domain of Euclidean space $\mathbb{R}^3$ that contains all points in B, then we define the signed distance function for any point $x \in B$ by [Armitage & Kuran, 1985]:

$$u(x) = \begin{cases} d(x, \partial D) & \text{if x in } \overline{D} \\ -d(x, \partial D) & \text{if x in } D' \end{cases} \tag{2.21}$$

Where ∂D is the boundary of D, $\overline{D}$ is the closure of D in $\mathbb{R}^3$ and $D' = \mathbb{R}^3 \backslash \overline{D}$. Finally d is the distance function that reflects a point's distance to the boundary. Since $u(x) = 0$ at the boundary of D, the surface of a point cloud can be easily reconstructed by finding the zero-crossing of the signed distance function. There are numerous approaches to estimating the signed distance field for a point cloud, which are usually determined by the surface reconstruction algorithm.

One of the most well known surface reconstruction algorithms is the marching cubes algorithm developed by Lorensen & Cline [1987] for the purpose of visualizing 3D medical scans. Before the development of the marching cubes algorithm, the results of MRI and CT scans were visualised as 2D slices of the 3D data, which made analysis difficult and time consuming. Marching cubes provided a way to construct a 3D isosurface from these 2D slices, thus making the results of 3D medical scans significantly more intuitive and readable. Due to the nature of the 2D slices produced by medical imaging, the original marching cubes algorithm was designed to take in a set of 3D scalar data with a regular structure as input [Newman & Yi, 2006]. Specifically, the 3D points that form the data set are the lattice points of a cubic lattice. The core idea of marching cubes is then to process each cube of this lattice sequentially in order to find intersecting regions of the final isosurface.

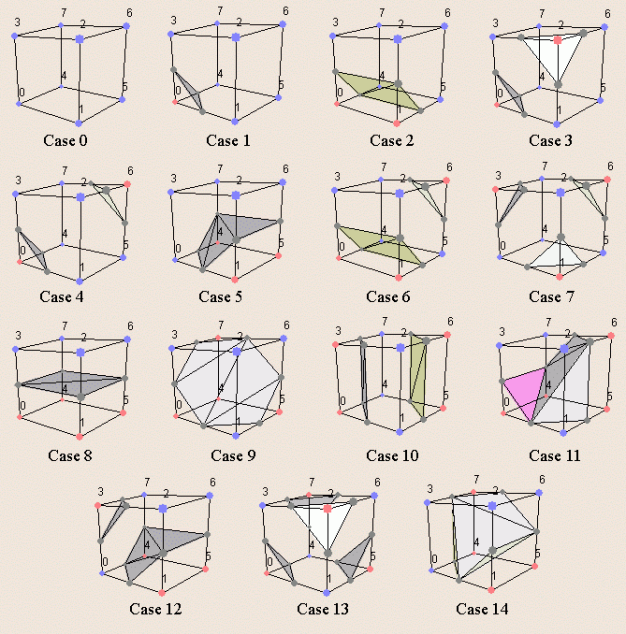


Figure 2.7: The 15 possible cases for an isosurface intersection through a cube [polytech, 2016].

Each cube in the cubic lattice is composed of 8 lattice points (vertices), with a scalar value for each point. The first step for processing a cube is to mark the vertices with values that are greater than or equal to a given threshold value, α. There are therefore 256 possible ways to mark the vertices of a cube, which can be reduced to the 15 base cases shown in Figure 2.7 by exploiting rotation and symmetry. Each of the possible marking cases represents an isosurface intersection and the cases are usually stored in a pre-built lookup table for efficiency. Once an intersection scenario has been identified for a marked cube, the intersection points of the isosurface are calculated for each of the affected edges using linear interpolation [Newman & Yi, 2006]. That is, given an intersected edge with vertices V_1 and V_2 with scalar values β_1 and β_2 respectively, the intersection point is given by:

$$I_{\{x,y,z\}} = V_{1_{\{x,y,z\}}} + \rho(V_{2_{\{x,y,z\}}} - V_{1_{\{x,y,z\}}}), \tag{2.22}$$

with

$$\rho = \frac{\alpha - \beta_1}{\beta_2 - \beta_1}. \tag{2.23}$$

Since a single edge is contained in four adjacent cubes in the lattice, the calculated intersection points for a cube can be reused when visiting the neighbouring cubes. Once all the intersection points for a cube have been found, the last step is to construct triangular faces to represent the surface for the cube, which can be simply done with the help of a lookup table. The resulting surface of the input data set is the set of all calculated triangles for each cube.

Numerous improvements have since been introduced to this original version of marching cubes, mostly with the intention of reducing the computational demand of the algorithm to the point where it is viable for use in real-time. Some variations exploit mirror symmetry in cubes to further reduce the number of intersection cases [Chan & Purisima, 1998; Miguet & Nicod, 1995; Van Gelder & Wilhelms, 1994], while others have focussed on optimising the process of encoding active and inactive cubes in the grid. Above all else, the rising computational power, availability, and affordability of modern GPUs has presented the opportunity for increased parallelism, and provides significant speed improvements in implementations of marching cubes.

2.3.4 Kinect Fusion

In 2011, Newcombe *et al.* [2011] developed the Kinect Fusion algorithm for real-time 3D scene reconstruction using the Kinect depth camera. Their aim was to develop a system that produces both detailed surface reconstructions and accurate camera pose tracking. While originally developed for the Microsoft Kinect [Microsoft, 2015], the algorithm can accept input from any consumer depth camera, regardless of the range imaging method used and open source implementations of Kinect Fusion have made provision for this.

Despite the fact that the Kinect is equipped with both a depth and an RGB camera, Kinect Fusion only processes depth information during surface reconstruction. This means that the algorithm does not make use of available colour information to optimise its output. On the other hand, since Kinect Fusion is not dependant on colour input, it is able to function unhindered in poor lighting conditions or even complete darkness. The algorithm thus has the potential for use in applications such as mine surveying and search and rescue, where the environment is usually dark and difficult to illuminate. For each input depth frame, the algorithm performs four stages of processing in order to obtain the updated surface and camera pose.

Surface Extraction

When a new depth map frame is obtained, a bilateral filter is first applied in order to remove noise, while preserving edges [Newcombe *et al.* , 2011]. A multi-resolution depth map pyramid is then constructed using the filtered depth image, with each level obtained by block averaging and then sub-sampling the previous level. For each level of the depth map pyramid, a point cloud is obtained by using the sensor's calibration matrix to back-project the corresponding level's depth image. The normals for the point cloud are calculated by taking the cross-product of the two vectors defined by adjacent points in a cloud. That is, given a point cloud, P, represented as a vertex map, V, the corresponding normal vector at point (x, y) is given by:

$$N(x, y) = (V(x + 1, y) - V(x, y)) \times (V(x, y + 1) - V(x, y)). \tag{2.24}$$

This too, is done for each level of the depth map pyramid, thus resulting in a normal map pyramid. Throughout this pre-processing stage, missing depth map pixels are ignored and thus not represented in the corresponding point clouds or normal maps.

Alignment

In order to estimate the alignment between the current depth frame and the global surface, Newcombe *et al.* [2011] make two assumptions. It is assumed firstly that depth data is recorded and processed at a high frequency and secondly that the sensor does not undergo large motions between consecutive frames. These assumptions allow for the use of an ICP algorithm variant for aligning similar point clouds. More specifically, for each frame in kinect fusion, an optimised version of point-to-surface ICP is employed to estimate the transformation between the current frame's point cloud and that of the global model. Thus using Equation 2.10, the error function for frame k becomes:

$$E(T_k) = \sum_u ||(T_k V_k(u) - V^g_{k-1}(u))^\top N^g_{k-1}(u)||^2, \tag{2.25}$$

where T_k is the transform that aligns vertex map $V_k(u)$ with the previous frame's map $V^g_{k-1}(u)$, which is in the global space. While ICP generally fails for point clouds with large displacements, the assumptions made mean that there are only subtle changes between consecutive frames, which significantly increase the chance of convergence, as well as reducing the number of iterations required. The performance of the algorithm is additionally increased by making two notable improvements. Firstly, the algorithm takes advantage of modern GPU hardware to parallelise a number of steps in the kinect fusion algorithm, such as some of the steps required for solving Equation 2.20. Secondly, ICP is performed on the lowest three levels of the vertex and normal map pyramids, starting with the uppermost of the three levels. This reduces usual computational demand of the ICP algorithm, by using lower resolution images to approximate the transform before refining it at higher resolutions.

Surface Reconstruction Update

Once an alignment between the new cloud and the global cloud has been estimated, the new cloud is merged with the global cloud in order to obtain an updated global surface. This is done using a truncated signed distance function TSDF variant that has been designed for use with GPUs. Specifically, the TSDF of both the global and the input cloud's are represented as discreet voxels, which can be efficiently represented in GPU memory. This allows for the efficient parallelization of the surface reconstruction step and hence for real-time throughput. It was because of this and the ability to merge TSDFs using a weighted average that Newcombe *et al.* [2011] chose this representation for this step.

Given an input depth frame D_k with estimated pose T_k, the global TSDF for point $\mathbf{p}$ is given by,

$$F_k(\mathbf{p}) = \psi(||T_k - \mathbf{p}|| - D_k(\hat{\mathbf{p}})), \tag{2.26}$$

where $\hat{\mathbf{p}}$ is the perspective projection of point $\mathbf{p}$ and ψ is a function that truncates the TSDF to within a desired distance, μ, from the surface.

$$\psi(m) = \begin{cases} \min(1, \frac{m}{\mu})\ \text{sign}(m) & \text{iff } m \geq -\mu \\ \text{null} & \text{otherwise} \end{cases}. \tag{2.27}$$

In the case where a point is both not visible and further from the surface than the maximum distance, μ, the function $\psi(m)$ does not measure the point and returns a null value. Once the TSDF has been computed for the new frame, it is then merged with the global model by computing the weighted average of the two clouds. Thus for each point $\mathbf{p}$, the average TSDF is

calculated as,

$$F_k^{avg}(\mathbf{p}) = \frac{W_{k-1}(\mathbf{p})F_{k-1}^{avg}(\mathbf{p}) + W_k(p)F_k(\mathbf{p})}{W_{k-1}(\mathbf{p}) + W_k(\mathbf{p})}, \tag{2.28}$$

where, given some maximum weight value W_θ, the weight associated with the current frame is,

$$W_k(\mathbf{p}) = \min(W_{k-1} + 1, W_\theta). \tag{2.29}$$

The zero crossing of the resulting average TSDF then represents the surface of the updated model. Rather than using the filtered depth map for this step, the raw data is used instead in order to retain detail in the final surface.

Surface prediction

The final step in the kinect fusion algorithm is obtain the new point cloud and normals from the updated global TSDF. This is achieved by ray casting against the zero crossing of the new TSDF for each pixel. Thus for each pixel a ray is propagated forwards in steps starting from some minimum ray length. When the ray moves from a positive to a negative region, a zero crossing has been located and the intersection point is recorded. In order to compute the surface normals, Newcombe *et al.* [2011] make the assumption that the gradient of the TSDF is orthogonal to the set of points on the zero crossing. Thus the normal for point $\mathbf{p}$ in the updated point cloud can be computed by taking the derivative of the TSDF,

$$N_k^g(\mathbf{p}) = \left[\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right]^\top. \tag{2.30}$$

The intersection points and normals are stored as a vertex map and normal map respectively, and are then used as the global model during the alignment step of the next iteration.

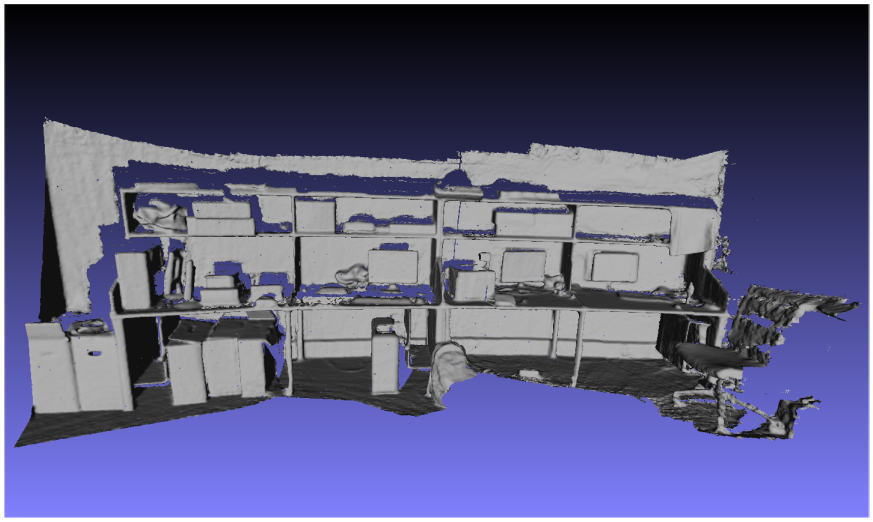


Figure 2.8: A reconstructed scene using the Kinect Fusion algorithm.

Since the development of the Kinect Fusion algorithm, a few open source implementations have begun to emerge, the most notable of which is KinFu [Pirovano, 2012]. KinFu was developed using the PCL library [Rusu & Cousins, 2011] and is now included in its stable releases. KinFu is able to work with any depth camera that is compatible with OpenNI [OpenNI Consortium *et al.* , 2011], providing more freedom in using this application.

2.3.5 RGB-D mapping

The kinect fusion algorithm takes the natural approach of using 3D point clouds to align depth frames with the global model, but fails to incorporate any form of RGB information during the alignment process. RGB-D mapping, on the other hand, takes advantage of both types of input from typical consumer RGB-D cameras. Developed by Henry *et al.* [2014] for use in navigation in robotics, the RGB-D mapping algorithm uses recorded RGB information to roughly align input frames, before applying RGB-D ICP to obtain a more precise alignment. Whereas traditional ICP variants can sometimes converge to local minima, the addition of RGB information to this step helps to avoid this, thus improving the quality of alignments. The rough alignment step using colour information also prevents cases of large displacements between clouds that would otherwise cause ICP to fail.

Rough alignment

When a new input is received, consisting of a depth frame and an RGB frame, the algorithm starts by extracting sparse features from the current RGB frame. This can be done using a variety of feature detectors, but Henry *et al.* [2014] found that the FAST feature detector [Rosten & Drummond, 2006] provided efficient performance and reliable results. The extracted feature points together with their corresponding depth values then form a point cloud of 3D feature points which, together with the feature points from the previous frame, can be used in the rough alignment step.

In order to obtain a transformation, $\hat{M}$, between these the two feature sets, Henry *et al.* [2014] employ the random sample consensus (RANSAC) algorithm. RANSAC iteratively estimates a transformation between the feature sets by finding a model that fits the data set with the maximum number of inliers. Prior to entering the main loop, the algorithm finds the matching feature points between the two feature sets. In each iteration, RANSAC then randomly samples 3 pairs of matching features and finds the optimal transformation between these two sets of three. The optimal transformation is found using the same method used to solve Equation (2.15) on page 18, except that for structured light cameras, the error function is based on the distance between the re-projected points rather than points in a point cloud. That is, given the set of matching feature points, F_m, the optimal transformation between the two sets is given by,

$$\hat{M} = \arg\min_{M} \left(\frac{1}{|F_m|} \sum_{i \in F_m} |Proj(M\mathbf{s_i}) - Proj(\mathbf{a_i})|^2 \right), \tag{2.31}$$

where $\mathbf{s_i}$ and $\mathbf{a_i}$ are points in the source and target set of feature points respectively. The re-projection function is defined as $Proj((x, y, z)) = (u, v, d)$, where u and v are the coordinates of the pixels in the RGB image and d is the value of the corresponding pixel in the disparity map. These values are given by,

$$u = \frac{f}{z}x + C_u, \tag{2.32}$$

$$v = \frac{f}{z}y + C_v, \tag{2.33}$$

$$d = \frac{f}{z}b, \tag{2.34}$$

where f and b are respectively the focal length and baseline of the camera, and (C_u, C_v) define the center of the re-projected image. Once the transformation has been found, the set of inliers (consensus set) is found by picking the pairs of matching features that lie within a specified threshold distance of each other when the estimated transformation is applied. This process is repeated for different samples until a predefined maximum iteration count is reached. The

transformation that produced the most amount of inliers is then optimised by re-computing the transformation using all the inliers, instead of just the 3 feature pairs. This produces a good, but not perfect alignment between the two frames, which is then optimised in the next step.

RGB-D ICP

After aligning the two point clouds using the transformation obtained in the previous step, each point in the current frame and its nearest neighbour in the previous frame are treated as a matching pair and are added to the set of matching points, P_m. Next, RGB-D ICP is used to optimise the alignment between the two clouds. This process is similar to the point-to-plane ICP algorithm described in Section 2.3.3, except that an extra term is added to the error function in order to account for visual features found in the previous step. Thus the point-to-plane ICP optimal transformation function becomes,

$$\hat{M}_f = \arg\min_M \left[\left(\frac{1}{|F_m|} \sum_{i \in F_m} |Proj(M\mathbf{s_i}) - Proj(\mathbf{a_i})|^2 \right) + \beta \left(\frac{1}{|P_m|} \sum_{j \in P_m} w_j |(M\mathbf{p_j} - \mathbf{q_j})\mathbf{n_j}|^2 \right) \right].$$

The inner function of this equation is made up of two terms. The first term represents the error produced by misaligned visual features points from the previous step, while the second term is the error function from the standard point-to-plane ICP algorithm with two additional weighting factors. The weight, w_j controls the contribution of each individual point pair in P_j to the calculated cost, while β controls the relative contributions of the two terms. The joint error function is minimized using the Levenberg-Marquardt algorithm [Levenberg, 1944; Marquardt, 1963].

Loop closure

Most depth cameras, especially those that are not designed for industrial applications, produce varying amounts of noise in their outputs depending on their type and environment. Despite filtering depth frames as a pre-processing step, this noise may introduce small errors into the frame alignment process and result in a sub-optimal solution. Additionally, the various assumptions and approximations made in order to solve the alignment problem further introduce errors into the result. While not usually noticeable between consecutive frames, these errors accumulate over time and eventually lead to a deformed global mapping of the scene. This can be particularly obvious when the sensor returns to a previously scanned point and the resulting mapping is severely misaligned with the previous mapping of the same point.

This problem, known as the loop-closure problem, is currently ignored in Kinect fusion. The RGB-D mapping algorithm, on the other hand, takes steps to detect and correct such cases. In order to detect a loop-closure, the algorithm needs to recognise when the sensor has returned to a previously visited location. Two frames could be said to represent the same location if a RANSAC alignment between them returns inliers above a specified threshold value. For the sake of reaching real-time mapping speeds, however, Henry *et al.* [2014] chose not to compare the current frame with every previously recorded frame, and instead defined a subset of key-point frames against which the current frame could be tested.

In RGB-D mapping, a key frame is selected whenever the current frame's distance or change in rotation from the previous key-frame exceeds a specified threshold. This method therefore

produces key-frames that are dissimilar to one another, as they do not contain many of the same visual features. The set of key-frames is then considered to be the set of possible loop closure frames and is tested against whenever a new key-frame is created. However, even this subset of frames may be computationally strenuous to test against, especially for lengthy sessions, and so the current frame is only tested against a subset of key-frames to detect a loop-closure. This subset is chosen to only include key-frames that are within a certain distance of the current global pose. Further more, a bag of words approach is used to efficiently identify similar key-frames from the set, so that the final subset is small enough to test in real-time.

Once a loop-closure has been detected, the algorithm then needs to correct the global model in order to realign the loop-closure frames. Henry *et al.* [2014] propose two methods for achieving this in an efficient manner. The first method involves using a graph to represent frame poses and their geometric constraints. If a pose constraint is represented by an edge between two vertices that represent a frame, then the graph would mostly consist of vertices with one child and one parent. A closure frame, however, would have an edge between itself and a prior frame that is not its neighbour. In order to readjust the global model when a loop-closure is found, the errors in the pose graph are minimized using a gradient descent algorithm. The second method involves re-projecting the feature points of each frame and then minimizing the alignment error between matching features, as well as their corresponding camera pose estimations.

2.4 Autonomous navigation

2.4.1 Overview

Any mobile autonomous agent is faced with the challenge of navigating its environment without human input, which usually includes the need for obstacle avoidance and path planning. The difficulty of this challenge depends heavily on the nature of the environment that the agent is designed to traverse, which can determine available sensory input, movement constraints and the quality of input features. Autonomous navigation is therefore the subject of many research endeavours in fields such as robotics and artificial intelligence.

For an agent to be able to navigate its environment, it usually requires a map of the environment as well as knowledge of its position within that map. For most agents with access to complete or near-complete information of the world, such as agents within a game or GPS-equipped outdoor robots, it is easy to acquire a map and the agent's location. In these cases, the agent can proceed directly to planning a path through the map. Some agents, however, do not have direct access to mapping and/or localisation information and therefore need to estimate this information using sensory input. In this research, the quadcopter drone is deployed in an underground environment and as a result, does not have access to GPS positioning. Additionally, since the goal of the drone is to map its environment, it will enter the environment without any prior knowledge of the layout. Agents with these constraints need to perform some form of simultaneous localization and mapping before they can begin with path planning.

2.4.2 Simultaneous Localization and Mapping

Simultaneous localisation and mapping (SLAM) is the problem of using an agent's sensory data to concurrently build a map of its environment and determine the agent's location within that map [Williams *et al.* , 2002]. In order to achieve complete autonomy in a robot, the agent requires the ability to localise itself within its environment and make navigation decisions without external assistance. For this reason, a solution to the SLAM problem is crucial to the field of robotics and while the problem has been mostly solved on a theoretical level, SLAM is still far

from perfect when used in real-world applications [Durrant-Whyte & Bailey, 2006].

Efforts towards solving the SLAM problem first began in the mid 1980s, but came to a temporary halt at the end of the decade following a paper by Smith *et al.* [1990]. In their paper, Smith *et al.* [1990] showed that a solution to SLAM would require the agent to keep track of every observed landmark in its environment, as well as its pose. This result seemed to imply that computations would grow quadratically more expensive as more landmarks were observed and that errors in the constructed map would grow with time. A few years later, however, it was shown that this problem does in fact have a convergent solution and that this solution was heavily dependent on correlations between observed landmarks [Csorba, 1997; Csorba *et al.* , 1996].

Whenever an agent observes a landmark, there is an inherent amount of error between the observed location and the actual location. This error is, in turn, due to error in the agents estimated pose when the observation is made. As a result, the position errors of multiple landmarks are very similar, and are in fact highly correlated. Furthermore, it was shown that these correlations always increase whenever new observations are made, which implies that estimates for the relative positions of landmarks get progressively more accurate over successive measurements. Thus as an agent scans an environment, it is able to build an increasingly accurate map in a separate frame of reference. All that remains is for the agent to estimate its position relative to this map, which gets easier as the map errors decrease.

This realisation lead to a resurgence in the SLAM research community and numerous methods were developed and improved over the following two decades to solve the SLAM problem. While not perfect, SLAM has already been incorporated into many products, either heading for, or on the market. Much focus is still on methods for improving the performance of SLAM algorithms, both in terms of speed and accuracy.

Problem description

In SLAM it is assumed that an agent enters an environment without any prior knowledge of its structure or layout. The goal of SLAM is thus to construct of set of landmarks from the environment in real-time, while simultaneously keeping track of the agent's pose and velocity [Durrant-Whyte & Bailey, 2006]. Two vectors are thus defined to represent these desired outputs,

- x_k which describes the pose of the robot at time k;
- m_i which contains the location of the ith landmark.

Furthermore, two more vectors are defined which are used to formulate the problem mathematically,

- u_k which describes the inputs to the robot at time $k-1$ that moved it to the pose x_k;
- z_{ik} which is the observation of landmark i at time k.

These four vectors, x_k, m_i, u_k and z_{ik} recorded from times $[0, k]$ then form the sets $X_{0:k}$, M, $U_{0:k}$ and $Z_{0:k}$ respectively. Thus given the robots starting pose x_0, the set of control inputs $U_{0:k}$ and the set of observations made $Z_{0:k}$, SLAM requires that we find the robots current pose x_k and a set of correlated landmarks m at time k. Rather than attempting to solve this problem deterministically, it has long since been realised that a probabilistic approach is better suited for such a noisy problem. As a result, the problem is usually described in terms of a probability distribution,

$$P(x_k, m | Z_{0:k}, U_{0:k}, x_0), \tag{2.35}$$

which needs to be solved at each time k. This is solved in a two-step update procedure using Bayes Theorem,

$$P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0) = \int P(x_k|x_{k-1}, u_k) \times P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0)dx_{k-1} \tag{2.36}$$

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) = \frac{P(z_k|x_k, m)P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k|Z_{0:k-1}, U_{0:k})}. \tag{2.37}$$

The two most important probability distributions from these equations are,

$$P(x_k|x_{k-1}, u_k), \tag{2.38}$$

and

$$P(z_k|x_k, m), \tag{2.39}$$

which are known as the motion model and observation model respectively [Durrant-Whyte & Bailey, 2006]. It should be noted that the motion model is dependent only on the robots previous position and the most recent control input, and that it is unaffected by environmental features.

Solution

In order to solve Equations (2.36) and (2.37), one needs to be able to solve the motion and observation models. The most common approach to this is to use the extended Kalman filter (EKF) with Gaussian noise [Durrant-Whyte & Bailey, 2006]. The motion model is described in terms of the vehicle kinematics, which are modelled by $f(a, b)$, and the Gaussian noise, w_k. Thus Equation (2.38) becomes,

$$x_k = f(x_{k-1}, u_k) + w_k. \tag{2.40}$$

Similarly, the observation model can be written in terms of its geometry, which is modelled by $h(a, b)$, and Gaussian noise, v_k. The observation model is then described by,

$$z_k = h(x_k, m) + v_k. \tag{2.41}$$

Both w_k and v_k have zero mean, and v_k has covariance R_k. With the motion models in this form, we can apply EKF to obtain solutions for the two update steps at each time k. The means of the desired vectors are then given by,

$$\begin{bmatrix} \hat{x}_{k|k} \\ \hat{m}_k \end{bmatrix} = E\begin{bmatrix} x_k \\ m \end{bmatrix} |Z_{0:k}\Big], \tag{2.42}$$

and the covariance,

$$\begin{aligned} P_{k|k} &= \begin{bmatrix} P_{xx} & P_{xm} \\ P_{xm}^T & P_{mm} \end{bmatrix} \\ &= E\left[\begin{pmatrix} x_k - \hat{x}_k \\ m - \hat{m}_k \end{pmatrix}\begin{pmatrix} x_k - \hat{x}_k \\ m - \hat{m}_k \end{pmatrix}^T |Z_{0:k}\right], \end{aligned}$$

where $\hat{x}_{k|k}$ is the estimated state at time k, given all measurements. Letting ∇f be the Jacobian of f, the first update step then becomes,

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}, u_k), \\ P_{xx,k|k-1} &= \nabla f P_{xx,k-1|k-1} \nabla f^T + Q_k, \end{aligned}$$

and letting ∇h be the Jacobian of h, the second update step becomes,

$$\begin{bmatrix} \hat{x}_{k|k} \\ \hat{m}_k \end{bmatrix} = \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{m}_{k-1} \end{bmatrix} + W_k[z(k) - h(\hat{x}_{k|k-1}, \hat{m}_{k-1})], \tag{2.43}$$

$$P_{k|k} = P_{k|k-1} - W_k S_k W_k^T, \tag{2.44}$$

with

$$S_k = \nabla h P_{k|k-1} \nabla h^T + R_k, \tag{2.45}$$

$$W_k = P_{k|k-1} \nabla h^T S_k^{-1}. \tag{2.46}$$

Problems

SLAM does not a have perfect solution at this time and most implementations display the occasional flaw or inaccuracy at varying levels. Although efficient implementations have been developed, the computations done during SLAM are still proportional to the square of observed landmarks [Durrant-Whyte & Bailey, 2006]. This can mean that mapping large areas or producing maps with high detail can be computationally strenuous. Loop closure, the problem of detecting when an agent has returned to a previous location and using that knowledge to correct its map, is still in need of further work as well.

In terms of the robotic agents themselves, SLAM is more difficult for MAVs than for terrestrial vehicles. Unlike many terrestrial vehicles, MAVs often do not have direct access to odometry information and in the absence of visual input, need to estimate it by double-integrating accelerations or using visual cues [Achtelik *et al.* , 2009a]. MAVs are also in constant motion while airborne, even while hovering in place, which may cause position estimates to drift over time.

2.4.3 Path Planning

Path planning strategies for indoor aerial robots are widely varied as information constraints and environmental factors are different in each application. Dolgov *et al.* [2008] used a modified version of the popular A* search algorithm for the autonomous driving of their entry in the DARPA Urban challenge in 2007. The A* algorithm was extended to capture the continuous 3D kinematic state space of the vehicle and the algorithm's solutions were improved using a non-linear optimisation. A number of subsequent studies have used variations of the A* and D* algorithms [Grzonka *et al.* , 2012; Shen *et al.* , 2011a] for quadcopter path planning, but often required the user to predefine a target destination.

Studies that involve agents that traverse unknown environments with little or no prior knowledge of the environment and its layout have often used a frontier-based approach for path planning [Achtelik *et al.* , 2009a; Yamauchi, 1997]. In this approach, frontiers are defined as the boundaries between explored and unexplored space and an agent explores unexplored space by moving to new frontiers. In a simple implementation, the agent could repeatedly navigate to the nearest frontier until the entire space is explored. Other approaches could assign priorities to different frontiers depending on the reason for exploration.

2.5 Similar Projects

Autonomous 3D mapping of both indoor and outdoor environments has been the subject of studies for almost two decades now, with indoor mapping focused mainly on robot navigation and obstacle avoidance without external assistance. These studies mostly chose LIDAR and laser range finder devices for acquiring depth data and only in the last 5 years has focus shifted to alternative depth sources, such as ToF and structured light cameras.

The first autonomous 3D mapping systems employed laser range finders in order to retrieve depth information from their surroundings. In 1998, Miller & Amidi [1998] developed an autonomous helicopter equipped with an integrated laser scanning system for use in outdoor environments. The scanning module of their system was tested by having it scan an area of roughly $300m^2$ while the helicopter was flown manually. The scan collected and registered 2.5 million data points with a half-meter accuracy.

Thrun *et al.* [2004] developed an autonomous robot for exploring and mapping abandoned mines. The Groundhog robot was a $680kg$ vehicle equipped with an on-board computer and a variety of sensors, including a laser range finder. The robot was deployed to explore and map a subset of tunnels in the abandoned Mathies mine near Courtney, PA. The experiment was mostly successful despite a software malfunction near the end that required manual intervention. The robot was able to generate a 2D map of the explored area as well as a 3D local map, which it used for obstacle detection. The test also demonstrated a few shortcomings in the system, such as the robot's inability to traverse deep water, heavy mud or small openings in the tunnels. Thrun *et al.* [2004] also experienced difficulties with wireless communications underground and proposed a system of wireless repeater stations to resolve this.

Nuchter *et al.* [2004] later improved upon this system by developing a six degree of freedom SLAM algorithm in order to merge the robot's local 3D scans into a global 3D map. The algorithm employed a fast variant of the ICP algorithm to incrementally register point clouds. Nuchter *et al.* [2004] tested the algorithm by using the same Groundhog robot that was used in the previous mine mapping attempt to traverse and scan 250m of the Mathies mine. The collected data was then processed offline using the algorithm in question. The algorithm was able to register the 3D points and produce a final map in under a minute, which shows promise for real-time implementations of the algorithm.

As quadrotor MAVs become more popular, many researchers turned to them for surveying applications. Researchers looked at using quadcopters for surveying construction sites, archeological digs and other outdoor projects, both autonomously and manually [Siebert & Teizer, 2014; Bemis *et al.* , 2014]. Today, there are a number of surveying companies that use MAVs for surveying open-cast mines. Underground mines, however, are a lot more difficult to survey using MAVs due to the lack of space and GPS availability, and most researchers have remained focussed on using terrestrial robots, rather than aerial robots. While not mine related, research pertaining to indoor mapping using quadrotor MAVs is still on-going. Achtelik *et al.* [2009b] designed a system that utilized a quadcopter equipped with a laser range finder and stereo camera to navigate and map an indoor office. Shen *et al.* [2011b] produced a similar system that used a quadcopter and a laser scanner to scan a multi-floor indoor environment.

2.6 Conclusion

This chapter discussed the various topics and research efforts related to this work. Despite being over a century old, quadcopters have only recently become practical and affordable due to the dropping costs of micro-electronics. This has lead to a massive surge in popularity of these drones, both amongst hobbyists and professions in many fields. The difference between quadcopters and traditional single rotor helicopters was also discussed in terms of kinematics and control methodologies.

The field of range imaging was then introduced, noting a large number of range imaging techniques, which are each suited for different applications and each with a different price tag. LIDAR devices are by far the most accurate tool for range imaging, but come at a significant cost. At the other end of the spectrum, there exist techniques that can process photographs captured using non-specialised cameras and extract depth information from them, often with low-reliability. Modern consumer depth cameras, which were originally aimed at the gaming community, offer both reasonable accuracy and reliability, while keeping costs low.

Next, the various steps and approaches to processing depth data was covered, from the raw input to the final mesh. The ICP algorithm is widely used in this area for point cloud registration, with many different variations and adaptations available. Modern GPU hardware has even allowed the use of ICP for registration of dense point clouds in real-time, thus opening up the possibility for use in online mapping using autonomous robots. Lastly, two successful approaches to 3D mapping using consumer depth cameras were discussed in detail. Kinect fusion makes efficient use of GPU hardware to optimise its global model update step and is well suited for use in low-lighting environments. RGB-D mapping, on the other hand, uses both RGB and depth information to improve the accuracy of cloud alignments, but displays poor performance in dark environments.

This chapter then gave an overview of concepts related to autonomous navigation, including SLAM and path finding. SLAM is vital for any autonomous robot and while the theory has been well developed over the passed two decades, many obstacles still remain for successful implementations. Kalman filters are still popular tools in the field of SLAM, but other successful approaches also exist, including the previously discussed Kinect fusion and RGB-D mapping algorithms, which essentially perform SLAM. Path planning is a much simpler problem to solve, although complexities arise when moving from terrestrial robots to MAVs due to the increase in dimensions of the state space.

Finally, a brief summary of research efforts that are related to this work was presented, with focus on 3D mapping using MAVs and autonomous mine surveying. The vast majority of autonomous mine surveying projects involve the use of terrestrial robots, rather than MAVs. There are, however, a number of projects in which quadcopters are used for autonomous indoor mapping. Many of these projects rely on laser scanners for depth data acquisition, while a handful have investigated other means of range imaging, such as stero-vision.

Chapter 3

Research Methodology

3.1 Introduction

This chapter discusses the methodology that was followed during the development of the autonomous indoor mapping system. As discussed in Chapter 1, the aim of this research is to develop a cheap and autonomous 3D mapping system that uses quadrotor drones to survey and map sections of mine tunnels in real-time. This multi-faceted objective naturally means that such a system would be comprised of many complex sub-systems, all of which need to function efficiently in order to produce acceptable results. These sub-systems can be divided into two main components, namely the 3D reconstruction component and the autonomous navigation component.

Figure 3.1 gives a broad overview of the system and its components. The payload of the quadcopter consists of a range imaging device and an on-board computer that is separate from the quadcopter's processor. The on-board computer is responsible for recording depth data using the range imaging device and, after some pre-processing, transmitting it to the ground station via the communication module. The ground station uses the received data to perform real-time 3D reconstruction, SLAM and path planning. Finally, the ground control station transmits a new set of flight instructions to the control unit of the quadcopter in order to manoeuvre it according to the determined path.

Due to several constraints, only a single quadcopter was used in the system and the addition of multiple, cooperative quadcopters is left to future work. It is also assumed that:

- the walls of the mine tunnels to be mapped will have a rough texture;
- there will be no moving objects other than the quadcopter during the mapping process;
- the tunnels will be equipped with WiFi access points at regular intervals, providing near constant WiFi coverage in the tunnels.

This final assumption is made in light of the fact that the product of this research is intended for use in a digitised smart-mine [Cawood, 2015].

This chapter presents the development of the various components in the system, up to and including their integration into the final product. Justification is also given for the equipment and materials used during this research, including any modifications made to off the shelf equipment.

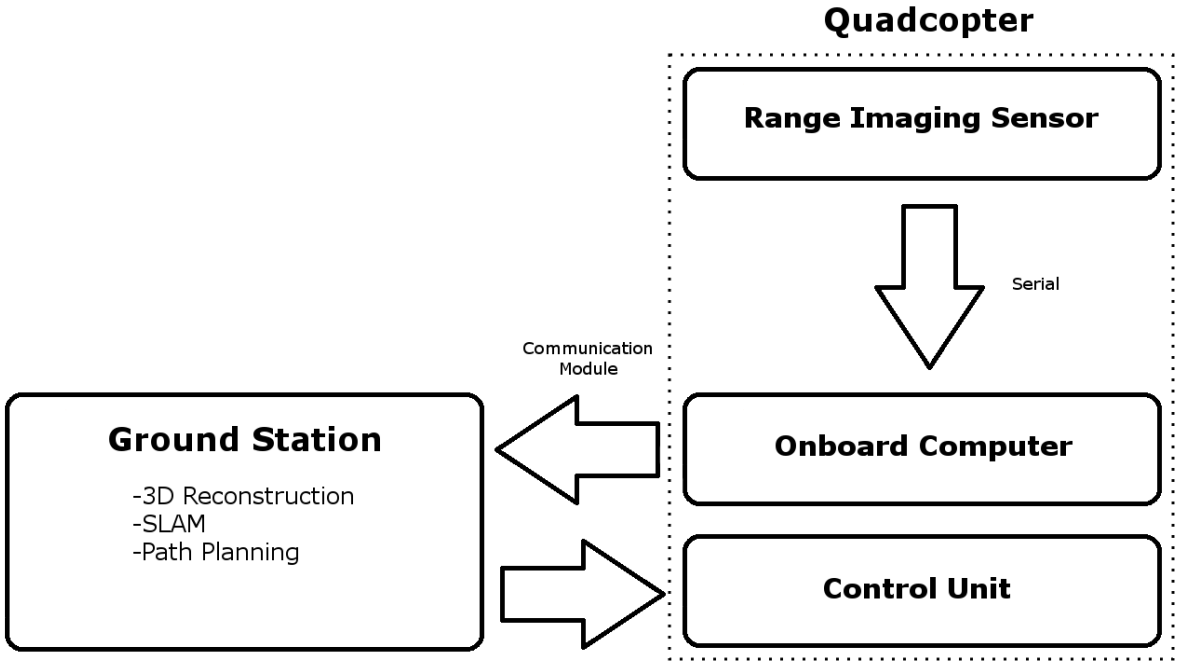


Figure 3.1: An overview of the components of the autonomous 3D mapping system.

3.2 Equipment Setup

3.2.1 Choice of range imaging device

For the range imaging and 3D reconstruction component of the system, a wide variety of sensors and algorithmic approaches were considered. One of the aims of this research is to keep the cost of producing the system relatively low and so LiDAR devices were immediately deemed non-viable. The extremely poor lighting conditions found in underground mines means that range imaging techniques that rely on colour information, such as stereo vision and SfM, would be unreliable in this environment. The final two options for range imaging techniques were either ToF or structured light. While both techniques have been shown to produce similar results, structured light cameras are lighter and cheaper. An Asus Xtion Pro Live structured light camera [Asus.com, 2015] was therefore chosen as the range imaging device for the system.

The Xtion Pro live is able to capture depth information at a resolution of 640×480 at 30fps. Additionally, the camera can record RGB information at a resolution of 1280×1024 and audio via its two microphones. Finally the camera's low power consumption of at most 2.5W is ideal for a situation where the weight of an extra battery could be detrimental to the quadrotor's flight time.

3.2.2 Choice of quadcopter

The quadcopter selected for the system is the AscTec Hummingbird [GmbH, 2015], which is used extensively in research involving precise and dynamic control of quadrotor drones. The hummingbird has a maximum payload capacity of 200g and a flight time of between 10-20 minutes, depending on the payload. Equipped with two microprocessors and an advanced control unit, the hummingbird is able to maintain high stability and offers multiple modes of control including on-board user-written code, manual control and GPS assisted control. These features have made the hummingbird one of the most popular quadcopters in research and make it suit-

able for use in indoor environments where there is little room to manoeuvre.

It should be noted that underground mining operations have very strict regulations that govern the design and use of electronics underground. Furthermore it is common for unprotected electronics to fail in the extremely humid conditions found in some mines. This research focuses on the viability of the mapping system itself, rather than on the development of quadcopters that are suited for underground mining applications. Thus while the hummingbird isn't suited for use in real-world mines, it serves as an adequate proof of concept for future work.

3.2.3 The on-board computer

Raspberry Pi B+ [Raspberrypi.org, 2015] was mounted on the quadcopter and connected to the depth camera. The B+ has a Broadcom SoC 700MHz processor, 512MB RAM and 4 USB ports, all of which are sufficient for the system. In order to power the depth camera through its USB ports, the Raspberry Pi was over-volted to 6V. Additionally, the CPU was over-clocked from its default 700MHz to 1000MHz. For WiFi connectivity, a 54mbps USB WiFi dongle was used.

3.2.4 Mounting the payload

The Hummingbird quadcopter that was used in this system is not equipped with dedicated payload bays and so additional considerations had to be taken in order to mount the sensor payload onto the drone in an efficient and unobtrusive manner. The payload itself consists of the Xtion Pro Live RGB-D camera and the Raspberry Pi. Initially a separate battery pack was also included to power the Raspberry Pi and the camera, however the total weight of the initial payload was found to be almost 500g, which significantly exceeds the hummingbirds's maximum payload capacity of 200g.

In order to reduce the weight of the payload to a more viable level, the battery pack was first removed and the hummingbird's 11.1V 2100mAh LiPo battery was instead used to power the Rasberry Pi after stepping the voltage down to 5V. The Xtion Pro Live, being the heaviest component in the payload, was stripped of its casings, leaving only the front cover to protect the lenses. The camera's USB cable was also shortened, leaving just enough length to reach the Raspberry Pi that is adjacent to it. The Raspberry Pi was also removed from its protective casing and fitted with an anti-static sleeve instead. The final weight of the payload after these modifications is 250g and although this still exceeds the specified payload capacity of the drone, the quadcopter was still able to fly with most of its original stability, with a flight time of roughly 6-10 minutes. Figure 3.2 on the next page shows the hummingbird quadcopter with the lightened payload mounted.

3.2.5 The ground station

The ground station computer used in this system was equipped with 32GB RAM, an Intel Core i7-4770K 3.5GHz quadcore CPU and an Nvidia Titan X GPU. The computer was connected via ethernet to a 2.4GHz WiFi router, which provided the WiFi coverage necessary for this system. For direct communication with the quadcopter's control unit, the ground station was also equipped with a XBee PRO module and linked with the hummingbird's on-board XBee which connects directly to the control unit. In order to avoid interference between the XBee, the radio controller and the WiFi router, separate channels were explicitly allocated to both the WiFi and the radio controller.



Figure 3.2: The final quadcopter set up after mounting the payload.

3.3 3D Reconstruction

3.3.1 Choice of 3D reconstruction method

The relatively high quality of the depth data provided by the depth sensor meant that the Kinect Fusion algorithm [Newcombe *et al.* , 2011] could be used for reconstructing the environment in real-time. Since Kinect Fusion does not depend on RGB information during its reconstruction process, it is suitable for use in the poorly lit conditions in which the system operates. However, the algorithm's assumption that the distance between the camera position in consecutive frames remains low, may not always hold true while the camera is mounted on a quadcopter. Errors caused by this assumption thus need to be detected and dealt with by the system.

There are a handful of open source implementations of the Kinect Fusion algorithm with the most well-known being KinFu [Pirovano, 2012], which was developed using and included in the Point Cloud Library [Rusu & Cousins, 2011]. There are a few differences between KinFu and the original algorithm. PCL's cross-device support allows for KinFu to be used with any OpenNI-compatible depth camera, which includes the Xtion Pro Live. Secondly, eigenvalue estimation is used to estimate point cloud normals instead of the cross-product approach discussed in Section 2.3.4. The final difference found in the implementation is the use of the marching cubes algorithm for rendering the surface instead of ray casting against the TSDF. These two implementation differences are described in detail in the next section.

Like the original Kinect Fusion algorithm, KinFu is limited to scanning within a fixed volume that is set when the scanning process begins. KinFu is thus suited for scanning a small object rather than an expansive area. This limitation, however, was lifted in a later extension to KinFu, called KinFu Large Scale (KinFuLS), which adds moving volume capabilities to the original algorithm. Another notable extension of KinFu is the RXKinFu library [Vona, 2017], which also adds moving volume capabilities to the algorithm. RXKinFu was initially forked from the KinFu project, after which the developers reorganised the project to make it more easily accessible and user friendly. The aim of the project was to make the Kinect Fusion algorithm available for use in robotics, and so many of the parameters in KinFu were exposed for tweaking by the users and several robotics-related libraries were added. One major drawback of the RXKinFu library software is that while it allows for moving volumes, it discards information contained in previous volumes when a volume shift occurs. For this reason, the Kinect Fusion Large Scale software was selected to do the 3D reconstruction in the mapping system and RXKinFu was kept as a reference for comparing the performance of the moving volume component.

3.3.2 Implementation

Data capturing

The biggest constraint on the 3D reconstruction system is the lack of physical connection between the depth sensor and the ground station that is doing the processing. Despite the cross-platform support provided by PCL, KinFuLS does not support receiving depth data from a networked sensor. In order to use KinFuLS in the mapping system, it was thus necessary to add networking functionality to KinFuLS that would allow the software to receive depth data both via a physical connection and wirelessly.

This task was complicated by the fact that depth data cannot be compressed using traditional image compression techniques. Even minor noise resulting from lossy compression was found to significantly affect the alignment process in Kinect Fusion, resulting in either increasingly misaligned point clouds in the global model or lost tracking. We therefore chose to transmit the uncompressed depth data to the ground station, but included optional parameters for reducing the data size. In addition to transmitting the raw 16bit depth maps that the Xtion Pro Live produces, options were incorporated to reduce the bit-depth of the maps to 8-bits. Depending on how this bit-depth reduction is done, the precision of the resulting depth map may be significantly reduced. However, the accumulative nature of the Kinect Fusion algorithm means that the accuracy of the global model increases as additional depth frames are merged with the model, effectively filling in the gaps left by the bit-depth reduction. For the same reasons, provisions were also made to transmit the depth images at either full resolution (640×480) or half resolution (320×240).

Flexibility aside, the depth transmission implementation on the Raspberry Pi is necessarily simple and computationally inexpensive. Retrieving the depth map from the depth sensor was done using the OpenNI [OpenNI Consortium *et al.* , 2011] library and the resulting data was transmitted over a TCP socket after some optional pre-processing steps. During these pre-processing steps, the scale and bit-depth of the depth map is reduced as necessary. Scaling is done linearly, while the bit-depths are reduced in one of two ways. For a target bit-depth of 8-bits, the depth values of the original (unsigned) 16-bit map are uniformly scaled down from a range of $[0, 65535]$ to a range of $[0, 255]$. While this reduction constitutes a vast drop in apparent precision, the resulting drop in precision for the KinFuLS output is less severe for the reasons discussed previously. In addition to the bit-depth reduction, the transmission of a 12-bit copy of the depth map was also tested. Instead of scaling down the depth values for this step, the available 12-bits offer a large enough range to keep the full precision of the original 16-bit map by thresholding the values. Thus any depth value greater than 4095 is set to 0 in the 12-bit depth map and ignored in 3D reconstruction phase. This thresholding reduces the maximum range of the depth map, but but does not affect the 3D reconstruction step. This is due to the fact that the Kinect Fusion algorithm performs the reconstructions in cubic volumes located in front of the depth sensor. These cubic volumes are usually between 2-5 meters in width and any depth information that lies beyond this distance is ignored during the reconstruction. Furthermore, any data that is lost in the reconstruction due to the thresholding is recovered when the depth sensor is moved closer to the missing points.

Depth map processing

The ground station receives each depth frame by reading the data byte by byte from the TCP socket and constructing an image from the pixel values. This process is shown in Algorithm 1. Once the depth image has been received by KinFu, it is then converted to the standard dimensions and bit-depth that KinFu expects. That is, this depth image is resized to a resolution of 640×480 and the bit-depth is scaled to 16 bits.

Algorithm 1: The pseudocode for reading a single depth frame from the TCP socket.

Data: frame width L, frame height H, number of bytes per pixel B and TCP socket handle S
Result: The most recent depth frame.

```
// Initialisation
imgSize = L × H × B;
Create buffer with length = imgSize;
// Read data from socket into buffer
for i ← 0 to imgSize −1 do
    Read byte from S into buffer;
end
// Assign pixel values to the image
frame = new image of size W × H;
p = 0;
for y ← 0 to H − 1 do
    for x ← 0 to W − 1 do
        frame(x,y) = buffer[p];
        p++;
    end
end
```

KinFu Large Scale performs the real-time 3D reconstruction on the input depth data according to the steps described in Section 2.3.4 in the previous chapter, with 3 notable differences. Firstly, KinFuLS allows for shifting volumes when the depth camera leaves the volume area. This means that scanning is not limited to a single real-world area, but can be done dynamically over a larger, unconstrained space without increasing the computational requirements. This is achieved by calculating the shift between the new TSDF volume and the old TSDF in GPU memory and discarding the portion of the old volume that does not overlap with the new one. Before the non-overlapping data is discarded, KinFu Large Scale first compresses and transfers it to the CPU, after which it can be saved to disk. As mentioned previously, at this point, the RXKinFu library does not save the data and simply discards it. Gaps present in the new volume from the shift are filled in when the next frame is processed.

The second change made to the Kinect Fusion algorithm in KinFu and KinFuLS is the use of eigenvalue estimation for calculating the surface normals of a point cloud. Whereas the original algorithm took the cross-product of the vectors defined by adjacent points in order to compute a point's normal, KinFuLS uses the principle component analysis (PCA) over adjacent points to estimate the normal [Rusu, 2010b]. For each point $\mathbf{p_i}$ in the point cloud, the covariance matrix of the k-nearest neighbours is given by:

$$C = (1/k) \sum_{i=1}^{k} (\mathbf{p_i} - \bar{\mathbf{p}})^T, \tag{3.1}$$

where $\bar{\mathbf{p}}$ is the centroid of the points in the nearest neighbours cluster. The normal for a point can then be determined from the eigenvalues, λ_j, and eigenvectors, $\mathbf{v_j}$, of its covariance matrix

with:

$$C \cdot \mathbf{v_j} = \lambda_j \cdot \mathbf{v_j}, \tag{3.2}$$

and $j \in (1, 2, 3)$.

The final difference found in KinFu and KinFuLS is the use of the marching cubes algorithm instead of ray casting for rendering the surface. Initially KinFu used the greedy triangulations algorithm to construct the final mesh, but later switched to marching cubes in order to obtain a smoother final reconstruction.

Optimisation

By default, the size of the volume in KinFuLS is set to $3m^3$ and while this is large enough to contain both walls of a narrow mine tunnel when the camera is positioned at the centre looking in the direction of the tunnel, it does not offer much room for lateral movement. This value was therefore increased from $3m^3$ to $4m^3$, while keeping the volume resolution at its original value. While this has the effect of slightly reducing resolution of the final model, it is necessary for scanning environments in which the useful features are large. In addition to the volume size, the shifting distance threshold was also reduced from the default value of $1.5m$ to $0.5m$. When the camera moves further from the volume's centre than the shifting distance threshold, the volume is shifted in order to once again position its centre at the camera. Thus reducing the shifting distance results in more frequent volume shifts and smaller updates to the global model during each shift. In most cases this change would be undesirable as each shift involves transferring data between the GPU and CPU, which may threaten the real-time performance of the software. In cases where the only features available are on the edge of the camera's view while the camera moves forwards, the features would no longer be visible by the time the camera reaches the edge of the volume and the tracking is lost. This situation occurs when scanning mine tunnels by keeping both walls in view at all times and so a shorter shifting distance ensures that there is always a reasonable amount of wall within the volume at all times.

Even with the default parameters, KinFuLS suffers from frequent ICP tracking loss and requires the camera to be returned to the position it was in when tracking was lost in order to resume scanning. This requirement is not suitable for the quadcopter mounted scanning system which relies on continuous scanning in order to position itself. Furthermore, when the ICP tracking is lost, the scanned model is often distorted and unusable as a 3D mapping. We therefore opted to replace the recovery mechanism with a more reliable procedure that would allow the quadcopter to continue to position itself uninterrupted: when a tracking loss occurs, the entire world model is written to the hard drive and then reset in the application. In order to minimize alignment errors between the new world model and the old model, the origin of the new model is set to be the origin of the final volume cube of the old model. When the new model is eventually written to disk, either as a result of the scan ending or due to another tracking loss, a separate process then aligns the newly written world model with the previous model using an ICP procedure. Depending on the processing power of the computer, this step can either be performed during the KinFuLS scanning process or after it has been completed. To reduce the rate at which models are corrupted due to tracking loss, an optional parameter was also defined that if set writes the world model to disk and resets the model in memory every n volume shifts, regardless of whether or not tracking was lost. After some testing, it was found that setting $n = 6$ results in world dumps that are small enough that they do not take long to write to disk, while still large enough to contain a significant portion of the local environment. While these world dumps do result in a delay between the last processed frame and the first

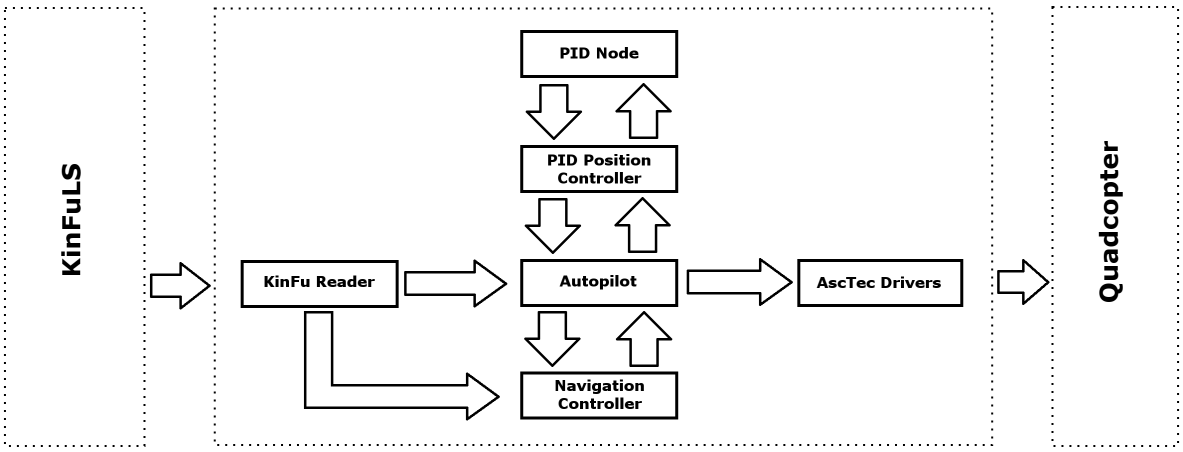


Figure 3.3: An overview of the quadcopter controller.

processed frame of the new model, with the high performance computer used in the system this delay is at most 200ms. Thus, the delays in position tracking caused by these world dumps do not affect the flight stability of the quadcopter by much.

Global transformations

With the changes made to the tracking loss recovery method, it was also necessary to keep track of the global position of the volume origin between resets. This global origin value is simply set to the most recent volume origin value unless a reset occurs, at which point the origin of the new volume is set to that of the global volume origin. Since KinFuLS tracks the local position of the sensor with respect to the volume origin, the global position of the camera is simply the sum of the local camera position and the global volume origin. This global camera position then gives the position of the drone within the coordinate space in the final system.

3.4 Control System

3.4.1 Controller Overview

The control system for the quadcopter was implemented using ROS [ROS, 2016] and consists of 5 components as shown in Figure 3.3. For communicating with the hummingbird quadcopter, the AscTec drivers and AscTec autopilot packages [Lab, 2016] were used, which provide tools and drivers for communicating directly with the quadcopter's low level processor. The communications frequency was set to 50Hz at the default baud rate of 57600. The purpose of the "KinFu reader" is to act as a communications bridge between the 3D reconstruction system and the quadcopter controller. This node simply reads any available data that is piped from the KinFu software and publishes it on the relevant ROS topic. The autopilot node functions as the central node for the system and is responsible for formulating and issuing commands to the drone, as well as monitoring its status. The autopilot node relies on the remaining two components, namely the PID position controller and the navigation controller, for obtaining stable thrust and attitude control values.

3.4.2 Autopilot

As mentioned above, the autopilot node acts as the primary node in the controller and is responsible for issuing direct commands to the quadcopter, with the help of the other nodes. The

autopilot node is designed to allow for both autonomous control over the quadcopter, as well as manual control using the keyboard. Algorithm 2 on the following page gives an overview of the main loop of the autopilot node.

In each iteration of the main loop, the most recent estimate for the quadcopter's current position is received from KinFuLS via the "KinFu Reader" ROS node. The narrow environment found in mine tunnels means that momentary delays in the system could give the quadcopter enough time to drift into a side-wall or enter into an unrecoverable trajectory. Thus if no new position values have been received from KinFuLS for more than a second, the autopilot enters into an emergency landing mode. In this state, all controls except for the thrust are handed over to the pilot and the thrust is gradually lowered in order to gently land the quadcopter at its current position.

If a new position value is received, it is then integrated with respect to time in order to estimate the current velocity of the quadcopter within the KinFuLS coordinate system. Next, any user input is processed and handled accordingly, including manual waypoint updates and emergency landing commands. If the navigation controller is running, the autopilot checks for new waypoints and if required, the desired destination point is updated. With the latest destination point and quadcopter position known, the relevant values are passed to the PID position controller in order to update the control signals that control the pitch, yaw, roll and thrust of the quadcopter. These control signals are returned in a callback function which sets the attitude and thrust controls as described in the PID position controller section.

The last step in the loop is then to publish the controls to the quadcopter via the driver node. During this step, the controls are first thresholded to limit their magnitude and a checksum is calculated. A control byte is also set at this point, which determines which controls the quadrotor can manipulate and which controls the pilot has. Finally, the controls, the checksum and the control byte are published to the AscTec driver's node, which in turn transmits them to the quadcopter.

3.4.3 PID Position Controller

Quadcopters that are stabilised using only IMU data have a tendency to drift over time and while the Asctec Hummingbird has far more precise on-board sensors than the average quadcopter, it still experiences a small amount of drift over time. Furthermore, the relatively heavy payload that is mounted on the quadcopter has a significant destabilising effect on the drone, thus reducing the degree by which the low level processor can compensate for drift. As a result, the system requires a PID controller that uses the KinFu implementation's tracking in order to stabilise the drone's position about a given point. This PID controller is constrained by KinFu's requirement for the camera to move smoothly between consecutive frames, meaning that fast, jerky movements should be avoided.

The PID position controller was implemented using the ROS PID package [Zelenak, 2016] and consists of two layers for each control input except yaw. For the pitch and roll controls, the first PID loop of each controller determines target velocity given the quadcopter's current position and the target position. The second loop of each controller then determines the respective pitch and roll outputs, given the current and target velocity of the quadcopter. The thrust controller follows the same structure, except the output signal of the controller is summed with the thrust value of the previous time-step in order to linearise its movement. Finally, the yaw controller consists of a single loop PID controller that calculates the required change in yaw given the quadcopter's current heading and target heading, as defined by its IMU. The gains of these sub-controllers were tuned manually, starting with the pitch, roll and yaw controllers and

Algorithm 2: The pseudocode for the autopilot node.

```
1  while Autopilot is running do
2      if Emergency landing then
           // Slowly reduce the thrust
3          UpdateEmergencyLanding();
4          continue;
5      end
       // Get the new position estimates from KinFu
6      UpdatePosition();
7      if Time since last response from KinFu is greater than 1 second then
           // Hand pitch, yaw and roll controls over to pilot and begin
              decreasing thrust
8          ExecuteEmergencyLanding();
9      end
       // Integrate position values to get velocity
10     UpdateVelocity();
       // Handle any keyboard input
11     UpdateControls();
       // Get any new waypoints from the navigation controller
12     UpdateWaypoints();
       // Sends the state and setpoint values for position and velocity to the
          PID position controller.  The control signals are received in a
          callback.
13     SendValsToPositionController();
       // Sets the pitch, yaw, roll and thrust to the received control signals
          from the PID position controller
14     UpdateControlCommands();
       // Sends the control commands to the quadrotor via the drivers
15     PublishCommands();
16 end
```

Table 3.1: The final gains for the quadcopter position PID controller.

	k_p	k_i	k_d
Velocity Controller (X-axis)	0.250	0.0750	0.015
Velocity Controller (Y-axis)	0.300	0.0010	0.005
Velocity Controller (Z-axis)	0.250	0.0750	0.015
Pitch Controller (Z-axis)	2.000	0.4000	0.060
Roll Controller (X-axis)	2.000	0.4000	0.060
Thrust Controller (Y-axis)	0.005	0.0001	0.005
Yaw Controller (Heading)	0.015	0.001	0.000

finishing with the thrust controller. The final gains are shown in Table 3.1.

The local coordinate system used for the position controller is based on that of the original KinFu software, with the positive Z-axis corresponding to the forward direction of the camera, the positive X-axis corresponding to the camera's right and the Y-axis pointing in an upwards direction. The global coordinate system is similar, with the X and Z axes defining the horizontal plane and the Y-axis pointing upwards. Thus the X coordinate of the target and desired positions are used as input for the roll controller and the Z coordinates are used for the pitch controller. Finally, the Y coordinates of the relevant points are used as input for the thrust controller. The quadcopter heading, which is used as the input for the yaw controller, is measured by the quadcopter's on-board electronic compass, which outputs a value in the range of $[0°, 360°]$. For ease of use, this value is converted to a range of $[-180°, 180°]$, where $0°$ corresponds to the quadcopter's starting heading.

While the autopilot node issues commands to the quadcopter at 50Hz, the fact that the modified KinFu software runs at a maximum of 30 frames per second means that the position controller only receives updated position coordinates at a rate of at most 30Hz. This rate is somewhat slower than the average quadcopter PID controller, but is still sufficient for keeping the quadcopter stable about a point, albeit with some noise. During the process of tuning the gains for the PID position controller, the movement of the quadcopter was intentionally damped in order to reduce the occurrence of sudden, jerky actions which could cause tracking loss for the KinFuLS software. This damping, along with the payload weight and the fact that the position controller receives position updates at a maximum of 30Hz, means that there is more noise in the quadcopter's movement when moving to or hovering about a point than is usually desired. When hovering about a specified position, the quadcopter tends to drift within a half meter radius around the position instead of converging to the exact position. This drift is not large enough to cause problems though, and in fact provides better scanning view-points for the 3D reconstruction system than if it remained almost static.

3.4.4 Navigation controller

Our navigation controller is designed for use in underground tunnels, with the assumption that there would be no moving obstacles. The system was therefore kept relatively simple and designed to be modular enough that it could be later used for larger, more complicated tasks at a later stage. The pseudocode for the navigation controller can be seen in Algorithm 3 and functions by assigning waypoints for the autopilot based on the quadcopter's current position.

On initialisation, the navigation controller waits for the autopilot node to launch the quadcopter via a take off command. Once the quadcopter has reached its desired altitude, the navigation controller begins issuing waypoints to the autopilot, with the aim of keeping the quadcopter in the center of the tunnel and directing it along the tunnel to explore unmapped areas. Keeping the quadcopter centered in the tunnel is achieved by simply positioning the waypoints along the midpoints of the two tunnel wall. The waypoints themselves are spaced at 0.5m intervals and a new waypoint is issued whenever the quadcopter reaches a distance less than 0.2m from the current waypoint. This exploration procedure progresses until the quadcopter reaches a dead-end, at which point the navigation controller issues a command for the autopilot to land the quadcopter.

Algorithm 3: The pseudocode for the main loop of the navigation controller.

```
1  while Autopilot is running do
       // Don't do anything if the quadcopter is grounded
2      if Quadcopter has not taken off then
3          continue;
4      end
       // If a dead end has been reached, land the quadcopter
5      if dead end then
6          PublishLandCommand();
7      end
       // If the current waypoint has been reached, update it
8      if Distance(Quadcopter position, CurrentWaypoint position) < 0.2 then
9          CurrentWaypoint.Z += 0.5;
10         CurrentWaypoint.X = midpoint between adjacent walls;
11     end
       // Publish the current waypoint
12     PublishCurrentWaypoint();
13 end
```

3.5 Experiment 1: 3D reconstruction performance

3.5.1 Data collection

Before the final system was assembled and tested, the performance of the Kinect Fusion algorithm was first evaluated on its own. This was done in order to compare the accuracy of the algorithm with that of current LiDAR systems that are used in industry, as well as to obtain scans done without additional noise from the quadrotor and WiFi.

Shown in Figure 3.4, the scene that was selected for the experiment was "Nick's Tunnel", a 67m long mock mine tunnel located under the School of Mining Engineering at the University of the Witwatersrand. As mentioned in Chapter 1, this tunnel was built using authentic materials in order to imitate the structure and conditions of real mine tunnels. The tunnel is equipped with high-speed WiFi access points and an efficient ventilation system. The first 48 meters of the tunnel is coated with a smooth reinforcing cement, while the final 18 meters consists exposed rock and ends in a blast face. The entrance to the mine tunnel is secured with a barred gate, which allows some sunlight into the first portion of the tunnel.

The depth camera used to record the data was the same Xtion Pro Live that would be

used in the complete system, connected directly to a laptop. For this experiment no live 3D reconstruction was performed, but rather the data was recorded for offline reconstruction on the ground station. This was done in order to ensure that the results were produced using the same Nvidia GPU that would later be used in the complete system. The recorded data consisted of 16bit depth images recorded at 640×480 pixels, and accompanying RGB images recorded at 1280×1024 pixels. The pairs of RGB-D images were captured at a constant rate of 30 frames per second, which is the the maximum possible rate for the depth camera.

The mine tunnel was recorded using 3 different camera poses so that the optimal pose could be determined for scanning the tunnel in the final system. Starting at the end of the tunnel furthest from the blast face, the depth camera was held up at the desired pose and then slowly carried to the other end of the tunnel while the data was recorded. The poses tested were: Facing the tunnel wall, facing the far end of the tunnel, and facing the far end of the tunnel while moving backwards. For each pose, 8 scans were recorded resulting in a total of 24 scans.

A ground truth data set was created by scanning the tunnel with a high-precision laser scanner. The scanner used for this was the Z+F IMAGER 5010 3D laser scanner [zf laser, 2016], which can produce 3D point cloud representations of a scene with sub-millimeter accuracy. The scanning process was performed over the course of two hours, and consisted of repeatedly repositioning the scanner to scan different segments of the tunnel. To ensure that the scans of each tunnel segment were aligned accurately in the final model, markers were positioned at regular intervals throughout the tunnel. During the registration phase, these markers were used as feature points to accurately align adjacent point clouds.



Figure 3.4: Nick's tunnel and the laser scanning equipment.

3.5.2 Data processing

Once the scans were complete, the data was transferred to the ground control station that would be used in the final system. For each scan, the depth frames were piped to KinFuLS at a constant 30 frames per second, at bit-depths of 16 bits and 8 bits. The data was also processed using RXKinFu in order to gauge the differences between the moving volume tracking capabilities of the two pieces of software. The RXKinFu test did not however save any results due the its inherent inability to retain shifted volumes.

The output models from the KinFuLS step were compared against the laser scan point cloud in order to evaluate their precision. Since the KinFuLS output clouds and the laser scanned

clouds used different coordinate systems, each of the KinFuLS clouds had to first be aligned with the laser scan cloud. This was done by first matching the bounding box scale and origin of the KinFuLS cloud with that of the laser scan in order to remove translational and scaling offsets. The rotational offsets were then minimized by manually rotating the KinFuLS clouds so that they roughly aligned with the laser scan. Finally, a precise alignment between the KinFuLS and laser scan clouds was obtained using the ICP algorithm. Once aligned, the precision of each KinFuLS cloud was measured by computing the mean nearest neighbour distances between the test cloud and the laser scan cloud. That is, for each point in the KinFuLS cloud, the nearest neighbour was found in the laser scan cloud and the absolute distance between these two points was recorded and later used to calculate the mean distance. This mean distance is then treated as the error between the KinFuLS cloud and the laser scan cloud. These results are presented in Chapter 4.

3.6 Experiment 2: Autonomous mapping of a mine tunnel

3.6.1 Set up

As with the first experiment, the second experiment was carried out in the 67m long mock mine tunnel, called "Nick's Tunnel". Although the tunnel is accompanied by an external control room, a dedicated control station was set up in the tunnel itself for this experiment. The control station consisted of the ground control computer and the WiFi router, which was used instead of the tunnel's built in WiFi access points in order to keep the network isolated. The ground control station was connected to the router via ethernet and the quadcopter's connection was alternated between ethernet and WiFi throughout the experiment.

3.6.2 Data Collection

Before running the system autonomously, it was evaluated without running the navigation controller node over a 10m long subsection of the tunnel. The quadcopter was positioned at the center of the tunnel facing the blast face and take-off command was issued. Once airborne, waypoints were manually assigned from the ground control station, directing the quadcopter towards the blast face where it was commanded to land. During this process, the flight-path and mapping performance of the quadcopter was carefully monitored. This step was repeated using both a WiFi connection as well as an ethernet connection between the drone and the router. Next, the quadcopter was positioned at the starting point and orientated to face the tunnel wall. The above steps were then carried out again in order to evaluate the effect that the quadcopter's view direction had on its position tracking and flight capabilities. This was necessary as a view direction that is aligned with the tunnel direction contains far less features than a view direction that pointed towards the tunnel wall.

The system was next run with the autonomous navigation component enabled, over larger sections of the tunnel. Since the autonomous system needs to be able to see where it is going in order to avoid collisions, the system was only tested during this step with the quadrotor facing in the direction of the tunnel, towards the blast face. Once again, the steps were repeated using both a WiFi connection and an ethernet connection between the quadcopter and the router. For safety reasons, a backup pilot equipped with the RC controller was on standby for the duration of the autonomous phase of this experiment.

3.6.3 Data Processing

The meshes obtained during this experiment were compared against the laser scanned point cloud by following the same steps followed in the first experiment. That is, the origins of the

bounding boxes of the meshes were first aligned with that of the laser scan cloud and any rotational offset was minimised manually. An ICP procedure was then used to precisely align the input mesh with the laser scan cloud, before computing the mean distance between the two data sets.

3.7 Conclusion

In this chapter, the methodology that was followed over the course of this research has been presented. The system developed during this research is comprised of numerous components, including both hardware and software, which have been adapted for use in an underground mapping system. The methodology therefore involves both the modification of existing solutions and the implementation of new, problem-specific solutions.

This chapter begins by presenting and justifying the use of equipment in this research. For the sake of prototyping this system, it was decided that only a single quadcopter would be used in the implementation. The quadcopter is equipped with a Raspberry Pi B+ and a structured light Xtion Pro Live RGB-D camera, the combination of which only just exceeds the maximum recommended payload capacity of the quadcopter. The ground station consists of a high performance computer equipped with an Nvidia GPU and communicates with the quadcopter unit via both WiFi and XBee links.

Next the chosen 3D reconstruction software for the system was discussed, which is the KinFuLS application from the Point Cloud Library. While based on the kinect fusion algorithm, KinFuLS does make some changes in its implementation. These changes were made in order to take advantage of the features of the point cloud library and do not affect the performance of the algorithm in a negative way. A fork of the KinFu implementation was also considered, called RXKinFu, which separately added moving volume features to the algorithm. While more organised than KinFuLS, RXKinFu's volume shifting procedure is not designed or optimised for retaining the global volume, and any data that is shifted out of the local volume is lost. It was thus decided that RXKinFu would not be suitable for use in the autonomous mapping system. A number of modifications were also made to the KinFuLS implementation including networking capabilities, graceful tracking loss handling and hole filling.

The control system that is responsible for autonomously piloting the quadcopter was then introduced in a modular fashion. The controller comprises of a number of ROS nodes, some of which are optional depending on the application. The autopilot node forms the central part of the controller, bringing together the remaining nodes and issuing commands to the quadcopter based on their output. The commands are relayed to the quadcopter using the AscTec drivers node at a rate of 50Hz. Similarly, the KinFu reader node relays data from the KinFuLS app to the ROS environment at a rate equal to the fps of the app. This data is used by the navigation controller when issuing waypoints for the autopilot and by the autopilot node itself when setting the states and setpoints of the PID position controller.

Finally, the two experiments that were carried out during this research were described, from the data collection steps to the data processing steps. The aim of the first experiment was to evaluate the performance of the KinFuLS software before mounting the depth camera on the drone. The 3D mapping in this experiment was therefore done by hand and with a direct connection between the sensor and the computer running the KinFuLS software. The second experiment followed a similar procedure to the first, except the completed system was tested instead of just a subcomponent. In both experiments, data was collected multiple times and using a variety of approaches. These results are presented and discussed in the following chapter.

Chapter 4

Results and Discussion

4.1 Experiment 1

4.1.1 Results

The laser scan of "Nick's Tunnel" was conducted by a team of two surveyors over the period of roughly 2 hours. After registering the scanned point clouds using the placed markers and removing areas that lay outside of the tunnel, the final point cloud contained $211,418,856$ points. Figure 4.1 shows a render of this point cloud from various angles.

The scan of the tunnel using the Xtion Pro Live was then conducted by a single person as described in the previous chapter. After processing the data using the 3D reconstruction system, 8 point clouds were obtained for each bit-depth. Figure 4.2 shows a render of one of these clouds from various perspectives. Out of the 3 tested poses, the 3D reconstruction system was only able to produce acceptable point clouds for the pose in which the camera was facing the far end of the tunnel and moving towards it. For the pose in which the camera was facing the far end of the tunnel, while moving away from it, the distant parts of the world model experienced decimation as their distance from the camera increased, causing numerous tracking losses and corruptions of the world model. The pose in which the camera was directed towards the wall while scanning along the tunnel allowed for successful tracking along a single wall, but resulted in large rotational errors when scanning was moved from one wall to an adjacent perpendicular wall. This meant that the angles between intersecting walls were far greater in the scan than they are in the original scene, regardless of depth map resolution or bit-depth.

For the successful pose, KinFuLS experienced an average of 7 tracking losses, all of which occurred while scanning the first section of tunnel that is coated with smooth cement. By comparison, the RXKinFu software experienced an average of 5 tracking losses for the same data. Additionally, both the KinFuLS and the RXKinFu software produced some minor rotational error between consecutive point cloud volumes containing scans of the smooth portion of the tunnel. Finally, the point clouds for the successful pose were compared against the laser scan point cloud to produce the results shown in Tables 4.1 and 4.2.

4.1.2 Discussion

Experiment 1 provides a clear comparison between conventional 3D mapping systems and the modified KinFu mapping system, in terms of both ease of use and mapping quality. The system that was extended in this research took an average of 3 minutes and 4 seconds to scan the 67m long tunnel, while it took the laser scanner 2 hours to scan the same section. Additionally, while the laser scanner required manually placed markers to register adjacent point clouds, KinFuLS performed the scan without any additional equipment. However, the results also show that the

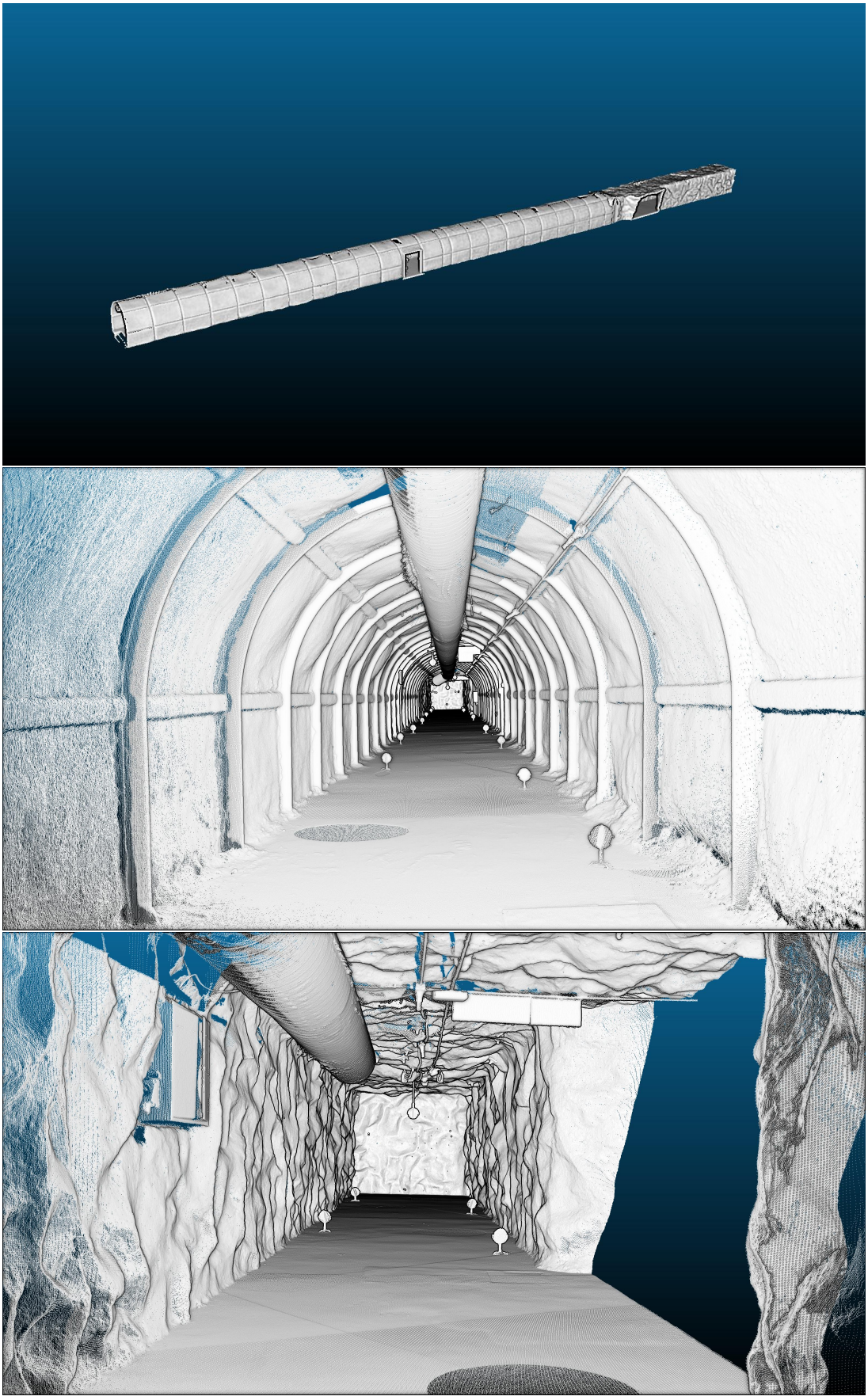


Figure 4.1: The laser scan point cloud.

Table 4.1: Experiment 1 results: 8 bit KinFuLS scan

	Points	**Scan time (secs)**	**Error w.r.t laser scan (cm)**
Scan 01	115, 982, 498	154	3.995
Scan 02	118, 616, 431	161	4.540
Scan 03	119, 172, 228	164	3.745
Scan 04	118, 789, 459	181	3.815
Scan 05	119, 415, 851	189	4.058
Scan 06	117, 855, 160	213	3.984
Scan 07	117, 969, 662	198	4.280
Scan 08	116, 913, 573	210	3.724
Mean	118, 089, 358	184	4.018

Table 4.2: Experiment 1 results: 16 bit KinFuLS scan

	Points	**Scan time (secs)**	**Error w.r.t laser scan (cm)**
Scan 01	113, 982, 357	154	3.668
Scan 02	117, 556, 180	161	5.368
Scan 03	116, 600, 241	164	3.666
Scan 04	117, 031, 939	181	4.344
Scan 05	117, 467, 634	189	3.834
Scan 06	117, 091, 520	213	3.762
Scan 07	118, 814, 411	198	4.240
Scan 08	115, 640, 602	210	3.500
Mean	116, 773, 111	184	4.048

pose of the camera has a significant impact on the ability of the KinFuLS software to scan a tunnel, with a poor pose causing the scan to fail completely. The ideal camera pose for KinFuLS is one that places the whole local tunnel section within the scanning volume simultaneously, which requires the camera to face in the direction of the tunnel. The system developed in this research is also more efficient than the laser scanner in terms of the man-power required to operate it. Although the laser scanner can be used by a single person, its weight and the use of multiple registration markers means that it is more efficient to assign it to a two man team. On the other hand, our system is extremely lightweight and can thus be used at maximum efficiency by a single person.

The ease of use of this system comes at the cost of its mapping quality and accuracy. When compared to the laser scan of the tunnel which has sub-millimeter accuracy, the KinFuLS clouds have a mean error of about 4 cm. If it is assumed that the laser scan is a perfect measurement of the actual scene, then the modified KinFuLS system is accurate to within 4 cm on average for a camera pose that scans the whole tunnel section. Since the system uses a structured light depth camera, which has an accuracy that is inversely proportional to the square of the distance between the camera and the scene surface, the accuracy of the scan for a given tunnel surface does improve when the camera is positioned closer to said surface. However, the constraints on the pose of the camera while scanning mean that it is not currently possible to scan a tunnel wall by wall at a close distance.

Surprisingly, lowering the bit-depth of the input depth data for this system has very little impact on the accuracy of the output model. The results from Tables 4.1 and 4.2 show that depth images with 8-bit data actually produced models with a slightly higher accuracy (4.018 cm) than those produced from the corresponding 16-bit data (4.048 cm). However, only scans 2 and 4 display a higher accuracy for the 8-bit data than the respective 16-bit data and high errors in these two results indicate that there was an alignment error during the scanning process. Treating these two scans as outliers and recalculating the mean error for each table gives an error of 3.964 cm for the 8-bit data and 3.778 cm for the 16-bit data. These revised errors show that the 16-bit data does produce more accurate scans that the 8-bit data, but not by a large margin. This 1.86 mm increase in error when switching from 16-bit depth input to 8-bit depth input is negligible relative to the total errors and so it is acceptable to use the lower bit-depth for scans in situations where the data transfer rate is low. The condition for this substitution is that the scanned environment consists almost entirely of large features, such as the texture of a rock-face. The introduction of smaller features to a scene would result in a further increase in the error margin between the 8-bit scan and the 16-bit scan.

The centimeter-range accuracy of the KinFuLS system does put limitations on its potential uses and it cannot yet be considered a complete replacement for current laser scanner systems. This system is not suitable for any task that requires precise, millimeter-scale scans of a mine tunnel, such as detecting potential faults or gradual changes to the tunnel. There are however, a number of applications for which this system is suited, such as mapping for navigation, object and people detection, and exploration. Furthermore, the results of this first experiment indicate that the accuracy of the structured light camera and KinFuLS software is sufficient for use in the final quadcopter-mounted system. The rapidly growing interest in virtual and augmented reality technologies, should also result in further improvements and cost reductions to consumer depth sensors, which in turn would lead to immediate accuracy improvements for KinFuLS.

Table 4.3: Experiment 2 results: Tunnel wall scans

	Points	**Error w.r.t laser scan (cm)**
Scan 01	4,566,855	2.992
Scan 02	3,329,622	3.052
Scan 03	3,692,344	2.536
Scan 04	4,037,841	2.025
Scan 05	4,118,239	1.633
Mean	3,948,980	2.448

4.2 Experiment 2

4.2.1 Results

The first step in experiment 2 was to evaluate the performance of the flight controller in the mining tunnel by manually assigning waypoints for the quadcopter. The flight capabilities of the quadcopter varied drastically depending on the section of tunnel through which it was flying and the drone's orientation with respect to the tunnel direction. For the end section of tunnel, consisting of bare rock, the quadcopter was able to track its position and fly smoothly between waypoints while it was facing the tunnel wall and flying sideways. When facing forwards however, the noise in the positioning system increased dramatically and the quadcopter was unable to maintain controlled flight. The first section of tunnel consisting of smooth, cement walls produced similar results with an increased occurrence of tracking losses. The flight time of the quadcopter during this step was found to be between 6 and 10 minutes due to the battery capacity of the quadcopter and the weight of its payload.

One major problem that was found during this experiment is that the quadcopter radio controller created significant interference for the WiFi network, which in turn negatively impacted on the frame-rate of the depth stream between the on-board computer and the ground station. Thus while the quadcopter was active, the frame rate of the depth stream varied erratically, often dropping as low as 1-2 frames per second. Despite using separate radio channels, the high power of the remote controller still had a significant impact on the quality of the WiFi network. The confined environment found in the mine tunnel further exacerbated this problem, effectively rendering the WiFi network unusable for large data transmission. It was thus decided that a wired connection should be used between the quadcopter and the ground station for the purposes of this research and the problem of radio interference is left for further research.

For the second step, the autonomous navigation module was enabled and the system was allowed to scan both sections of the tunnel while facing the wall. In this pose, the system was unable to detect incoming collisions from the side and so the land command had to be issued manually when the quadcopter neared the blast face. While the quadcopter was able to fly autonomously when facing the wall, all attempts to run the system while facing the quadcopter down the length of the tunnel resulted in crashes. This step was carried out 5 times in order to obtain 5 scans of the tunnel sections, which were then compared against the laser scan as discussed in Section 3.6. Table 4.3 lists the errors for each scan with respect to the ground truth laser scan cloud. By the end of the experiment, the quadcopter and its payload were coated in a fine layer of dust which did not seem to affect the performance of the on-board computer or interfere with the camera's recording ability.

4.2.2 Discussion

The results of experiment 2 show that the positioning ability of the prototype system is heavily dependant on the quality and scale of the features found in the scanned environment. For the scans in which the quadcopter was facing towards the far end of the tunnel, the only available features were the tunnel ceiling and walls, all of which lay on the very edge of the captured depth images. Features on the edge of a depth image are further from the depth camera than features in the center of the image plane and are therefore less accurate due to the structured light sensor's accuracy versus distance relationship. In addition to this, when the quadcopter is moving forwards through a tunnel, the edge features very quickly move out of frame and are therefore less present during cloud alignment steps than features in the center of the image would be. For 3D reconstruction purposes, alignment errors caused by these issues are smoothed out over consecutive frames. The prototype system however, uses the information obtained from each frame in order to estimate the position of the quadcopter and so any noise that is present directly affects the positioning estimates. Despite the damping of the PID positioning module gains, the low update frequency of the system means that noisy positioning estimates have a significant destabilising effect on the flight of the quadcopter.

The scans that were completed with the quadcopter facing the wall were far more successful, as all features were fully visible and closer to the camera. It is unsurprising then that the average error of the scans when compared to the laser scan (2.448 cm) is significantly lower than the errors in experiment 1. As discussed in Section 4.1.2, these high accuracies would drop significantly if the system were to attempt to reorientate the quadcopter to scan a perpendicular wall. While it is not viable for the system to scan an entire tunnel in this manner, the results do indicate that the system has potential for use in scanning and analysing rock-faces. Errors as low as 1.633 cm mean that the close-range scans of blast-faces and other surfaces of importance could potentially be used to critically evaluate the surface, either by a human or by a machine.

The results of experiment 2 indicate that the prototype system is not ready for use in real world applications but are very informative in terms of what changes are necessary in order to address the flaws that are present in this prototype in future work. First and foremost, a single medium should be used for all communications between the quadcopter and the ground station so as not to cause interference. Depending on the quadcopter used, the on-board processing tasks could even be shifted from the Raspberry Pi to the quadcopter's on-board processor so as to further reduce the weight of the payload. For this to be achieved, the libraries and drivers used by the Raspberry Pi to capture, process and transmit the depth data would need to be adapted for use on the quadcopter's processor. In order to address problems related to the pose of the quadcopter while scans are performed, some modification may be needed for the depth camera. Ideally, two structured light camera's could be mounted on the quadcopter, such that each one faces diagonally towards the left and right of the quadcopter's heading instead of straight forwards. This would provide a much wider field of view and allow both walls of a tunnel to be scanned in detail simultaneously. The extra weight of such a setup could be reduced by using only the depth components of the structured light cameras, as the RGB camera is not necessary for the system.

The noticeable amount of dust kicked up by the quadcopter during the experiment once again draws attention to the specialised hardware requirements that an autonomous mapping system would require in an actual mine. The dust may not have caused any issues over the course of the experiment, but continued use of a quadcopter in such an environment may lead to a severe shortening of the lifespan of the motors and on-board computer. Furthermore, actual mines present additional environmental hazards such as high humidity and temperatures, and the possibility of flammable gasses. In future work, it would therefore be necessary to develop a custom-built quadcopter for use in underground mines that is able to resist the harsh conditions within.

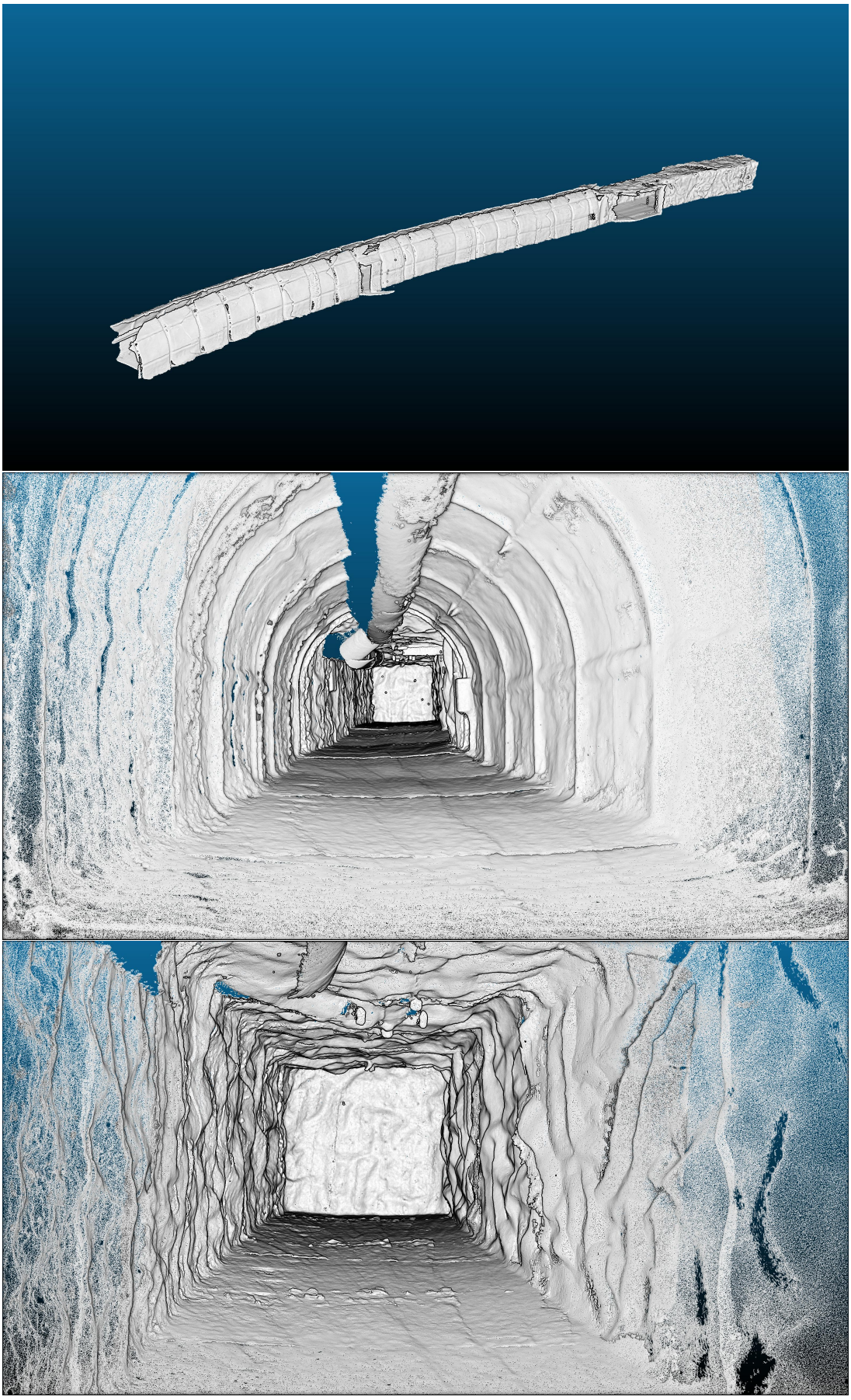


Figure 4.2: One of the KinFuLS Scan point clouds.

Chapter 5

Conclusion

This dissertation presents the first steps towards developing a fully autonomous mine mapping system. Chapter 1 considered the mining sector in South Africa and how it is still rife with danger, despite continuous attempts in recent years to improve safety. Accidents, environmental hazards and the high crime rate all contribute to a high fatality rate every year, as well as notable financial losses for the mining sector. This research aims to provide a multi-purpose tool that can be used in the mining sector for reconnaissance and surveillance in underground mines, thus providing dynamic security and removing the need for humans to enter dangerous areas.

Chapter 2 discusses quadcopters and their dynamics, as well as different approaches to 3D reconstruction. Quadcopters have only been a viable technology in recent times, but their vast range of applications have driven their popularity and development. This is gradually resulting in the introduction of cheaper and more powerful quadcopters into the consumer market, making the technology ideal for use in research. Depth imaging technologies have also gained popularity in recent times due to their introduction into the gaming sector. Numerous algorithms have begun to emerge alongside these technologies that take advantage of their abilities to perform tasks that were previously limited to expensive industrial equipment. One such task is 3D reconstruction, which the Kinect Fusion algorithm is able to perform in real-time over a wide area. This too is ideal for use in this research and the combination of consumer depth imaging equipment and quadcopters allows for the development of a relatively cheap and autonomous 3D mapping system.

Chapter 3 presents the methodology that was followed over the course of this research. The autonomous 3D mapping system is comprised of a ground station and a single quadcopter. The quadcopter is equipped with a payload consisting of a structured light depth camera and a Raspbery Pi, which acts as the on-board computer. The weight of the payload had a significant destabilising effect on the flight capabilities of the quadcopter and so modifications were required in order to reduce this weight as much as possible. The 3D reconstruction is performed by the KinFuLS software, which was extended to allow for depth map streaming over a network. The flight controller for the quadcopter was implemented using ROS and consisted of several modules that could later be extended for use in a larger system. The PID controller module was tuned manually and intentionally damped in order to allow for smooth scanning. Finally, two experiments were performed in order to evaluate the performance of the prototype system. The mock mining tunnel located at the University of the Witwatersrand was used for both experiments and provides a detailed recreation of the structure of actual mine tunnels. The first experiment tested the performance of the 3D reconstruction system against that of traditional laser scanning systems, while the second experiment evaluated the complete autonomous mapping system.

Chapter 4 lists the results of the two experiments and discusses their implications. The first experiment shows that while the 3D mapping system is not yet an adequate replacement for laser scanners, it is suitable for general, low precision mapping tasks. The prototype system is also much easier to use than laser scanners and scans can be performed in a matter of minutes instead of hours, as is the case for laser scanners. Experiment 2 demonstrated some of the drawbacks of using consumer depth cameras for this system, but also gave insight into potential avenues for future work. The quadcopter was not able to scan the tunnel while flying down the center, as the high amounts of positioning noise had a severe impact on its flight capabilities. The system was however, able to scan the tunnel walls while flying side-on, which indicates that even the current prototype is not without its applications. Scenarios that require the mapping and analysis of rough surfaces, such as rock faces, would be suitable for this system. Thus the prototype, as it currently stands, has applications in certain aspects of mine surveying, architecture and archaeology.

Future work is required to perfect this prototype and this research provides the groundwork and knowledge for such an endeavour. The use of multiple depth cameras would increase the field of view of the mapping system and allow the quadcopter navigate the tunnel while facing down the center. The weight constraints of the quadcopter mean that these additional depth cameras would need to be modified, removing any unnecessary components, such as RGB cameras. Much work is also required to design a quadcopter that is suitable for this mapping task. First and foremost, the quadcopter needs to be able to endure the harsh conditions found within most mines, including the high humidity, heat and potentially flammable gasses. In addition to this, the quadcopter should not pose a threat to the safety of mine personnel or equipment. For such a drone, a balance needs to be found between its carrying capacity, stability and flight range. Finally, the communications system needs to be redesigned in such a way that the depth stream and the control system do not interfere with each other. The addition of multiple, cooperative quadcopters would require further considerations during the design of this communications system, as well as the autonomous navigation system.

References

Achtelik, Markus, Bachrach, Abraham, He, Ruijie, Prentice, Samuel & Roy, Nicholas. 2009a. Autonomous navigation and exploration of a quadrotor helicopter in GPS-denied indoor environments. *In: First Symposium on Indoor Flight*.

Achtelik, Markus, Bachrach, Abraham, He, Ruijie, Prentice, Samuel & Roy, Nicholas. 2009b. Autonomous navigation and exploration of a quadrotor helicopter in GPS-denied indoor environments. *In: First Symposium on Indoor Flight*. Citeseer.

Armitage, DH & Kuran, Ü. 1985. The convexity of a domain and the superharmonicity of the signed distance function. *Proceedings of the American Mathematical Society*, **93**(4), 598–600.

Asus.com. 2015. *Xtion PRO LIVE - Overview*. `http://www.asus.com/Multimedia/Xtion_PRO_LIVE/`. Accessed: 2016-3-22.

bbzippo. 2017. `https://bbzippo.wordpress.com/2010/11/28/kinect-in-infrared/`. Accessed: 2017-7-2.

Bellekens, Ben, Spruyt, Vincent, Berkvens, Rafael & Weyn, Maarten. 2014. A survey of rigid 3D pointcloud registration algorithms. *Pages 8–13 of: Fourth International Conference on Ambient Computing, Applications, Services and Technologies*. Citeseer.

Bemis, Sean P, Micklethwaite, Steven, Turner, Darren, James, Mike R, Akciz, Sinan, Thiele, Sam T & Bangash, Hasnain Ali. 2014. Ground-based and UAV-based photogrammetry: A multi-scale, high-resolution mapping tool for structural geology and paleoseismology. *Journal of Structural Geology*, **69**, 163–178.

Berger, Matthew, Tagliasacchi, Andrea, Seversky, Lee, Alliez, Pierre, Levine, Joshua, Sharf, Andrei & Silva, Claudio. 2014. State of the art in surface reconstruction from point clouds. *Pages 161–185 of: EUROGRAPHICS star reports*, vol. 1.

Besl, Paul J & McKay, Neil D. 1992. Method for registration of 3-D shapes. *Pages 586–606 of: Robotics-DL tentative*. International Society for Optics and Photonics.

Bradski, G. 2000. OpenCV. *Dr. Dobb's Journal of Software Tools*.

Brunet, Tom & Chao, Leo. 2015. *CS 766 Project 4 - Uncalibrated Stereo Vision*. `http://pages.cs.wisc.edu/~chaol/cs766/`. Accessed: 2016-3-22.

Buelthoff, Heinrich H & Yuille, Alan L. 1991. Shape-from-X: Psychophysics and computation. *Pages 235–246 of: Fibers' 91, Boston, MA*. International Society for Optics and Photonics.

Cawood, Fred. 2015. Digital Mine laboratory prepared for digital mining. *PositionIT*.

Champleboux, Guillaume, Lavallee, Stephane, Szeliski, Richard & Brunie, Lionel. 1992. From accurate range imaging sensor calibration to accurate model-based 3D object localization. *Pages 83–89 of: Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*. IEEE.

Chan, Shek Ling & Purisima, Enrico O. 1998. A new tetrahedral tesselation scheme for isosurface generation. *Computers & Graphics*, **22**(1), 83–90.

Chang, William. 2008. *Surface reconstruction from points.* Department of Computer Science and Engineering, University of California, San Diego.

CNBC. 2016. `http://www.cnbc.com/2016/01/20/faa-drone-registry-may-not-be-enough.html`. Accessed: 2016-8-4.

CoM. 2016. *Illegal and Artisanal Mining.* Accessed: 2016-7-26.

Csorba, Michael. 1997. *Simultaneous localisation and map building.* Ph.D. thesis, University of Oxford.

Csorba, Michael, Uhlmann, Jeffrey K & Durrant-Whyte, Hugh F. 1996. New approach to simultaneous localization and dynamic map building. *Pages 26–36 of: Aerospace/Defense Sensing and Controls.* International Society for Optics and Photonics.

Design, Beauty. 2016. *Depth Sensing - Bluetechnix.* `http://ww2.bluetechnix.com/en/products/depthsensing/product/argos3d-p100/`. Accessed: 2016-7-8.

DJI. 2016. `http://www.dji.com`. Accessed: 2016-7-5.

Dolgov, Dmitri, Thrun, Sebastian, Montemerlo, Michael & Diebel, James. 2008. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, **1001**, 48105.

Donoghue, AM. 2004. Occupational health hazards in mining: an overview. *Occupational Medicine*, **54**(5), 283–289.

Durrant-Whyte, Hugh & Bailey, Tim. 2006. Simultaneous localization and mapping: part I. *IEEE robotics & automation magazine*, **13**(2), 99–110.

FabCentral. 2016. `http://fab.cba.mit.edu/content/processes/structured_light/`. Accessed: 2016-7-8.

Gettinger, Dan & Michel, Arthur Holland. 2015. Drone sightings and close encounters: An analysis. *Bard College Center for the Study of the Drone.*

GmbH, Ascending Technologies. 2015. *AscTec Hummingbird - Drohne fur Dynamik Schwarmfluge.* `http://www.asctec.de/en/uav-uas-drone-products/asctec-hummingbird/`. Accessed: 2016-3-22.

Goldstein, E. 2013. *Sensation and perception.* Cengage Learning.

Google. 2016. *Google Maps.* `http://maps.google.com`. Accessed: 2016-7-25.

Grzonka, Slawomir, Grisetti, Giorgio & Burgard, Wolfram. 2012. A fully autonomous indoor quadrotor. *Robotics, IEEE Transactions on*, **28**(1), 90–100.

Hansard, Miles. 2013. *Time-of-flight cameras.* Springer.

Henry, Peter, Krainin, Michael, Herbst, Evan, Ren, Xiaofeng & Fox, Dieter. 2014. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. *Pages 477–491 of: Experimental robotics.* Springer.

Hoffmann, Gabe, Rajnarayan, Dev Gorur, Waslander, Steven L, Dostal, David, Jang, Jung Soon & Tomlin, Claire J. 2004. The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC). *Pages 12–E of: Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, vol. 2. IEEE.

Iddan, Gavriel J & Yahav, Giora. 2001. Three-dimensional imaging in the studio and elsewhere. *Pages 48–55 of: Photonics West 2001-Electronic Imaging*. International Society for Optics and Photonics.

Inc.com. 2015. http://www.inc.com/will-yakowicz/1-million-drones-to-take-to-skies-after-holidays.html. Accessed: 2016-8-4.

isds.com. 2017. http://www.isds.co.za/news/SecuSystems-stops-illegal-mining-AND-steps-up-on-safety.php. Accessed: 2017-7-2.

Jamasmie, Cecilia. 2016. *Illegal mining expands in South Africa at Sky-high rates*. Accessed: 2016-7-26.

Jecić, Stjepan & Drvar, Nenad. 2003a. The assessment of structured light and laser scanning methods in 3D shape measurements. *Pages 237–244 of: Proceedings of the 4th International Congress of Croatian Society of Mechanics*. Citeseer.

Jecić, Stjepan & Drvar, Nenad. 2003b. The assessment of structured light and laser scanning methods in 3D shape measurements. *In: 4th International Congress of Croatian Society of Mechanics*.

Khoshelham, KouroshElberink, Sander Oude. 2012. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors*, **12**(12), 1437–1454. Accessed: 2016-7-9.

Krossblade. 2016. http://www.krossblade.com/history-of-quadcopters-and-multirotors/. Accessed: 2016-8-3.

Lab, CCNY Robotics. 2016. *Asctec drivers*. https://github.com/ccny-ros-pkg/asctec_drivers. Accessed: 2016-12-19.

Lemmens, MJPM. 1988. A survey on stereo matching techniques. *International Archives of Photogrammetry and Remote Sensing*, **27**(Part 5), 11–23.

Levenberg, Kenneth. 1944. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, **2**(2), 164–168.

Lidar-uk.com. 2016. http://www.lidar-uk.com/what-is-lidar/. Accessed: 2016-7-6.

Lorensen, William E & Cline, Harvey E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *Pages 163–169 of: ACM siggraph computer graphics*, vol. 21. ACM.

Low, Kok-Lim. 2004. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, **4**.

Marquardt, Donald W. 1963. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, **11**(2), 431–441.

Microsoft. 2015. *Kinect for Windows*. https://www.microsoft.com/en-us/kinectforwindows/. Accessed: 2016-3-22.

Miguet, Serge & Nicod, Jean-Marc. 1995. A load-balanced parallel implementation of the marching-cubes algorithm. *Pages 229–239 of: Proceedings of High performance computing Symp*, vol. 95.

Miller, Ryan & Amidi, Omead. 1998. 3-D site mapping with the CMU autonomous helicopter.

minealert.com. 2017. http://minelert.com/improving-mine-safety-productivity-access-control-systems/. Accessed: 2017-7-2.

minesurveyor.net. 2016. http://www.minesurveyor.net/. Accessed: 2016-7-26.

Mueller, Thomas. 2009. On the Birth of Micro Air Vehicles. *International Journal of Micro Air Vehicles*, **1**(1), 1–12.

Naughton, Russell. 2016. *The Dennyplane.* http://www.ctie.monash.edu.au/hargrave/dennyplane.htm. Accessed: 2016-7-11.

Newcombe, Richard A, Izadi, Shahram, Hilliges, Otmar, Molyneaux, David, Kim, David, Davison, Andrew J, Kohi, Pushmeet, Shotton, Jamie, Hodges, Steve & Fitzgibbon, Andrew. 2011. KinectFusion: Real-time dense surface mapping and tracking. *Pages 127–136 of: Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on.* IEEE.

Newman, Timothy S & Yi, Hong. 2006. A survey of the marching cubes algorithm. *Computers & Graphics*, **30**(5), 854–879.

Nuchter, Andress, Surmann, Hartmut, Lingemann, Kai, Hertzberg, Joachim & Thrun, Sebastian. 2004. 6D SLAM with an application in autonomous mine mapping. *Pages 1998–2003 of: Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 2. IEEE.

Oceanservice.noaa.gov. 2016. http://oceanservice.noaa.gov/facts/lidar.html. Accessed: 2016-7-6.

OpenNI Consortium, *et al.* . 2011. OpenNI, the standard framework for 3D sensing.

Pirovano, Michele. 2012. Kinfu an open source implementation of Kinect Fusion + case study: implementing a 3D scanner with PCL.

polytech. 2016. http://users.polytech.unice.fr/~lingrand/MarchingCubes/algo.html. Accessed: 2016-7-8.

Pomerleau, François, Colas, Francis & Siegwart, Roland. 2015. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends in Robotics (FnTROB)*, **4**(1), 1–104.

QuadcopterArena. 2015. *The history of drones and quadcopters.* http://quadcopterarena.com/the-history-of-drones-and-quadcopters/. Accessed: 2016-8-3.

Rambhia, Jay. 2017. *Disparity Map.* http://www.jayrambhia.com/blog/disparity-mpas. Accessed: 2017-1-22.

Raspberrypi.org. 2015. *Raspberry Pi.* https://www.raspberrypi.org/products/model-b-plus/. Accessed: 2016-11-1.

Raza, Syed Ali & Gueaieb, Wail. 2010. *Intelligent Flight Control of an Autonomous Quadrotor.* Citeseer.

Robertson, DP & Cipolla, Roberto. 2008. Structure from motion.

ROS. 2016. http://www.ros.org/. Accessed: 2016-11-7.

Rosten, Edward & Drummond, Tom. 2006. Machine learning for high-speed corner detection. *Pages 430–443 of: European conference on computer vision.* Springer.

Rusu, Radu Bogdan. 2010a. Semantic 3D object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, **24**(4), 345–348.

Rusu, Radu Bogdan. 2010b. Semantic 3d object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, **24**(4), 345–348.

Rusu, Radu Bogdan & Cousins, Steve. 2011 (May 9-13). 3D is here: Point Cloud Library (PCL). *In: IEEE International Conference on Robotics and Automation (ICRA)*.

savetherhino.org. 2015. `https://www.savetherhino.org/rhino_info/thorny_issues/the_use_of_drones_in_rhino_conservation`. Accessed: 2017-3-17.

Seitz, Christian & Altenbach, Holger. 2011. Project ARCHEYE: the quadrocopter as the archaeologists eye. *International Archives of the Photo grammetry, Remote Sensing and Spatial Information Sciences*, **38**(1), C22.

Shapiro, Linda G & Stockman, George C. 2001. *Computer vision*. Prentice Hall.

Shen, Shaojie, Michael, Nathan & Kumar, Vijay. 2011a. Autonomous multi-floor indoor navigation with a computationally constrained MAV. *Pages 20–25 of: Robotics and automation (ICRA), 2011 IEEE international conference on*. IEEE.

Shen, Shaojie, Michael, Nathan & Kumar, Vijay. 2011b. Autonomous multi-floor indoor navigation with a computationally constrained MAV. *Pages 20–25 of: Robotics and automation (ICRA), 2011 IEEE international conference on*. IEEE.

Shosa, Peter. 2015. *A Brief History of Quadcopters and Multirotors - EyeOnDrones.com*. `http://www.eyeondrones.com/brief-history-quadcopters-multirotors/`. Accessed: 2016-7-3.

Shotton, Jamie, Girshick, Ross, Fitzgibbon, Andrew, Sharp, Toby, Cook, Mat, Finocchio, Mark, Moore, Richard, Kohli, Pushmeet, Criminisi, Antonio, Kipman, Alex, *et al.* . 2013. Efficient human pose estimation from single depth images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**(12), 2821–2840.

Siebert, Sebastian & Teizer, Jochen. 2014. Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system. *Automation in Construction*, **41**, 1–14.

Smith, Randall, Self, Matthew & Cheeseman, Peter. 1990. Estimating uncertain spatial relationships in robotics. *Pages 167–193 of: Autonomous robot vehicles*. Springer.

Thrun, Sebastian, Thayer, Scott, Whittaker, William, Baker, Christopher, Burgard, Wolfram, Ferguson, David, Hahnel, Dirk, Montemerlo, D, Morris, Aaron, Omohundro, Zachary, *et al.* . 2004. Autonomous exploration and mapping of abandoned mines. *Robotics & Automation Magazine, IEEE*, **11**(4), 79–91.

Van Gelder, Allen & Wilhelms, Jane. 1994. Topological considerations in isosurface generation. *ACM Transactions on Graphics (TOG)*, **13**(4), 337–375.

Vedaldi, Andrea, Guidi, Gregorio & Soatto, Stefano. 2007. Moving forward in structure from motion. *Pages 1–7 of: Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE.

Vona, Marsette. 2017. *RXKinFu*. `http://www.ccs.neu.edu/research/gpc/rxkinfu/index.html#acknowledgements`. Accessed: 2016-11-1.

Williams, Stefan B, Dissanayake, Gamini & Durrant-Whyte, Hugh. 2002. An efficient approach to the simultaneous localisation and mapping problem. *Pages 406–411 of: Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1. IEEE.

Yamauchi, Brian. 1997. A frontier-based approach for autonomous exploration. *Pages 146–151 of: Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on.* IEEE.

Zelenak, Andy. 2016. `https://bitbucket.org/AndyZe/pid.git`. Accessed: 2016-11-29.

zf laser. 2016. *zf-laser.* `http://www.zf-laser.com/Z-F-IMAGER-R-5010.3d_laserscanner0.0.html?&L=1`. Accessed: 2016-11-17.

zflaser.com. 2016. `http://www.zf-laser.com/Z-F-IMAGER-R-5010C.3d_laserscanner.0.html?&L=1`. Accessed: 2016-7-8.

Zhu, Jiejie, Wang, Liang, Yang, Ruigang & Davis, James. 2008. Fusion of time-of-flight depth and stereo for high accuracy depth maps. *Pages 1–8 of: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on.* IEEE.

Zwane, Mosebenzi. 2016. *Mining Health and Safety Statistics 2015.* Accessed: 2016-7-25.