Author:
Bose, Laurie N

Title:
Edge Based RGB-D SLAM and SLAM Based Navigation

Edge Based RGB-D SLAM and SLAM Based Navigation

Laurie Bose

A dissertation submitted to the University of Bristol in accordance with the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering

August 2016

circa 40000 words

Abstract

Visual Simultaneous Localisation and Mapping (SLAM) is a vital technology for the advancement of autonomous robotics, providing a means of both mapping and pose estimation from visual data. Many advances in visual and RGB-D SLAM however have built upon established point feature and dense reconstruction methodologies, leaving other approaches relatively uncharted in comparison. This thesis instead explores semi-dense edge based approaches for RGB-D SLAM, in addition to path planning approaches for SLAM based indoor aerial robotics.

We first present an edge based RGB-D SLAM approach utilizing edges from both depth and RGB channels, along with a method of depth edge detection exploiting the temporal similarity between RGB-D video frames. Detected edge pixels are back-projected to form semi-dense point clouds and a fast edge based Iterative Closest Point (ICP) algorithm is utilized for sensor pose estimation. Evaluation of the proposed SLAM system demonstrates compelling results, achieving similar accuracy to a number of alternative systems while having significantly lower computational cost, thus making it fit for real-time application on small robotic platforms. An extension of this SLAM system to use 3D line features is then presented, along with methods of extracting and registering such line features from semi-dense edge point clouds.

Secondly we present a belief space planning approach for autonomous indoor MAVs relying on bearing measurements to environmental landmarks for localisation in cluttered environments. This approach can produce trajectories which minimize uncertainty in the vehicle's state from such bearing measurements, while reaching an end goal. The approach allows for a tunable trade-off between minimizing trajectory length and state uncertainty, and examples trajectories are given comparing these two extrema in various 3D environments and landmark layouts.

Lastly a SLAM-aware path planning approach is presented for indoor MAVs, utilizing the proposed edge based SLAM system for localisation. In addition to ensuring collision avoidance this approach ensures that SLAM localisation is maintained by producing trajectories along which the vehicle's RGB-D sensor always observes sufficient information from the existing SLAM map. These trajectories are "keyframe centric" consisting of many connected "segment" trajectories, each defined within the reference frame of a specific keyframe, together forming a complete trajectory across the map. This scheme allows the MAV to navigate across the SLAM map, even if it has become globally inconsistent due to issues such as sensor drift accumulation.

Declaration

I declare that the work in this dissertation was carried out in accordance with the Regulations of the University of Bristol. The work is original, except where indicated by special reference in the text, and no part of the dissertation has been submitted for any other academic award.

Any views expressed in the dissertation are those of the author and in no way represent those of the University of Bristol.

The dissertation has not been presented to any other University for examination either in the United Kingdom or overseas.

SIGNED: DATE:

Acknowledgements

First I would like to thank my supervisor Arthur Richards for his all guidance and sage advice over the years.

I would also like to thank all my friends and colleagues who I have been lucky enough to know over the course of my PhD, both within Queens building and the Visual Information Laboratory, Including Colin Greatwood, Tom Kent, Austin Greg Smith, Teesid Leelasawassuk, Shuda Li, Thei Zaza, Analiza Abdilla, Toby Perrett, Davide Moltisanti, Michael Wray, Eduardo Ruiz, Steve Bullock, Elham Asadi, Oliver Turnbull, Kieran Wood and Ujjar Bhandari.

Finally I am very grateful for the funding and support I received from DSTL who made this research possible.

To My Parents.

Contents

List of Figures

List of Tables

Nomenclature

SLAM

K : SLAM map / set of keyframes

K_i : i^{th} keyframe

K_T : Current tracking keyframe, used for sensor pose estimation

$\mathbf{X}_T$: Estimated pose of sensor in tracking keframes reference frame

$\mathbf{X}$: Estimated pose of sensor in world reference frame

F : Latest RGB-D frame of sensor

D : Depth edge point cloud associated with F

I : RGB intensity edge point clouds associated with F

$\mathbf{P}_i = (\mathbf{t}_i, \mathbf{q}_i)$: Estimated pose of i^{th} keframe

$\mathbf{t}_i$: Translation component of pose $\mathbf{P}i$ as a 3-D vector

$\mathbf{q}_i$: Rotational component of pose $\mathbf{P}i$ as a unit quaternion

$\mathbf{R}(\mathbf{q})$: Rotation matrix equivalent of unit quaternion $\mathbf{q}$

$\mathbf{R}(\widehat{\mathbf{n}}, \theta)$: Rotation matrix equivalent of axis angle rotation $(\widehat{\mathbf{n}}, \theta)$

F_i : RGB-D frame of i^{th} keframe

D_i : Depth edge point cloud of i^{th} keframe

I_i : RGB edge point cloud of i^{th} keframe

$K_i = \{\mathbf{P}_i, F_i, D_i, I_i\}$

R : set of all keframe loop closure constraints

$\mathbf{R}_{i,j} = (\mathbf{t}_{i,j}, \mathbf{q}_{i,j})$: relative pose constraint between pose $\mathbf{P}_i$ and $\mathbf{P}_j$

$\mathbf{t}_{i,j}$: Translation component of relative pose constraint $\mathbf{R}_{i,j}$ as a 3-D vector

$\mathbf{q}_{i,j}$: Rotational component of relative pose constraint $\mathbf{R}_{i,j}$ as a unit quaternion

Belief Space Planning

B : Set of bearing measurement beacons

O : Set of convex polygonal obstacles

O_i : i^{th} obstacle

V_i : Vertices of O_i

$\hat{E}_i$: Unit vector edge direction of O_i

$\hat{F}_i$: Unit vector face normals of O_i

$O_i = \{V_i, \hat{E}_i, \hat{F}_i\}$

N : Set of navigation graph nodes

E : Set of navigation graph edges

P : Set of path nodes

P_i : i^{th} path node

$\mathbf{n}_i$ graph node associated with i^{th} path node

Q_i Parent of i^{th} path node

$\mathbf{m}_i$ Graph node associated with parent node Q_i

l_i Total length of path associated with P_i

w_i Weighting score assigned to path node P_i

keyframe Based Path Planning

G : Set of all keyframe navigation graphs

G_i : Navigation graph of keyframe K_i

N_i : Nodes of graph G_i

$\mathbf{n}_i^j$: j^{th} node of G_i

E_i : Edges of graph G_i

$\mathbf{e}_i^j$: j^{th} edge of G_i

E_C : Set of connecting edges

$\mathbf{c}_i$: j^{th} connecting edge

Chapter 1

Introduction

This thesis explores Simultaneous Localisation And Mapping (SLAM) and its relation to autonomous navigation and path planning. Specifically, we target the problem of SLAM based indoor Micro Air Vehicle (MAV) navigation and present both a semi-dense RGB-D SLAM system and a SLAM-aware path planning approach (motivated by this scenario).

This introduction provides an overview of the content covered by this thesis, beginning with a motivation in 1.1 describing the problems relating to SLAM and navigation that we seek to address. The areas of SLAM, robot navigation and path planning are then discussed in Section 1.2. Sections 1.3 and 1.4 discuss the motivation behind our proposed SLAM and path planning approaches respectively. Finally, Section 1.5 presents a thesis outline along with a list of contributions and peer reviewed publications.

1.1 Motivation

The task of navigation requires one to possess both a map to decide upon the route to take and a means to determine where one is currently located within said map. Autonomous navigation by robots and vehicles is no different, requiring a representative map of the environment so that path planning can be conducted, in addition to a means to localise within the environment itself. Despite the rapid increase in the use of autonomous robots and vehicles over the last decade, navigation has often remained restricted to environments which have pre-existing maps, and for which there exists a means of reliable localisation by the presence of known environmental features or access to external systems such as GPS.

However in the most general navigation scenario, there may be no pre-existing map of the environment in question, the environment may have no known structural elements to exploit for localisation, and there may be no access to any external systems such as GPS for localisation (due to being indoors, within an urban canyon etc). A robot in such a scenario must instead conduct navigation with nothing but its on-board sensors, using them to both map the observed environment and localise within it, that is to perform SLAM. In this way SLAM is viewed as a key technology in enabling wide spread use of autonomous robotics and is a highly active area of research.

There are however, many issues related to such SLAM based navigation. If for example, the SLAM system was to lose localisation, the robot's ability to navigate would be severely impaired. To avoid this issue, it is vital to ensure that the robot's on-board sensors always obtain sufficient information for the SLAM system to maintain localisation at all times. This sensor information constraint must thus be factored into the path planning process to ensure safe navigation. Another issue is that of the map of the SLAM system itself, which in some scenarios, may become distorted or even globally inconsistent due to the accumulation of sensor drift and tracking errors, further complicating the path planning process.

In addition to these issues regarding path planning, the robot's SLAM system and on-board sensors must themselves be suited to the task of autonomous navigation. Specifically the SLAM must achieve consistent real-time performance upon the robot's own hardware, provide an accurate estimation of the robot's pose (at least relative to a local frame), and be capable of constructing a map which provides sufficient geometric information regarding obstacles within the environment.

Further discussion of these issues, along with our proposed SLAM and path planning approaches are now given in the following sections.

1.2 SLAM and Robot Navigation

Visual SLAM has seen significant advances over the last two decades with the development of a range of systems and innovations, notably the point feature based systems, Parallel tracking and mapping (PTAM) [58] and the recent Orb-SLAM [78] system in 2015, along with the dense reconstruction approaches of Kinetic Fusion [47] and the Dense Tracking and Mapping system (DTAM) [79]. These have bought significant improvements in terms of accuracy, robustness, map size and detail.

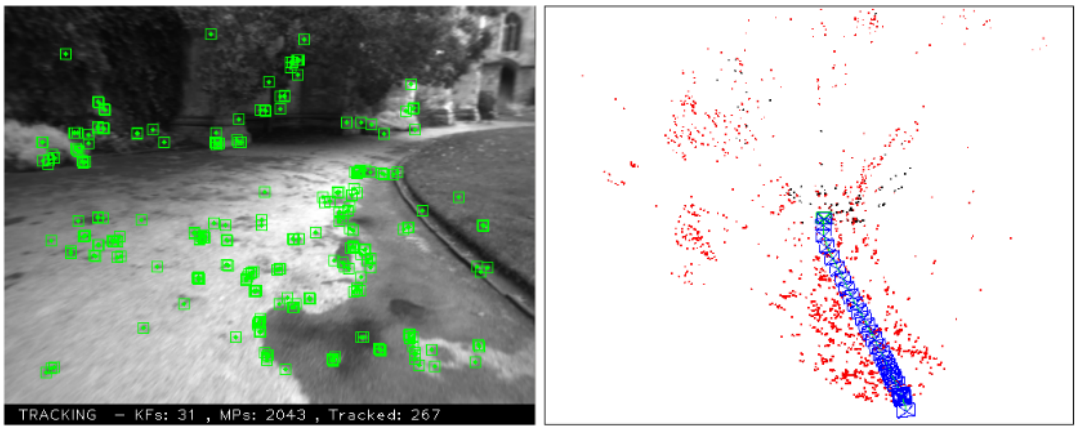


Figure 1.1: *Left, an example of point features extracted from a single video frame by ORB-SLAM. Right, an example of a sparse feature based map created by ORB-SLAM as presented in [78], estimated feature locations are shown in red and keyframe poses in blue.*

In order to conduct robot navigation however, the SLAM system and sensors utilized by the robot must be capable of providing a suitable representation of the environment, specifically a representation that provides sufficient 3D geometric information to ensure collision avoidance. Many such SLAM systems construct maps containing sufficient information to conduct sensor localisation, but which provide little explicit information regarding the geometry of the environment itself. For example, many traditional visual SLAM approaches, both monocular ([20],[14],[78],[15],[58],[60],[94],[93]) and stereo ([95],[80],[44],[74],[73]) are based on the concept of identifying and tracking salient visual point features, such as those produced by the well known SIFT [70], SURF [4], and ORB [85] feature descriptors. This results in maps consisting of the estimated locations of such features in 3D space as illustrated in Figure 1.3. These sparse 3D point feature maps are sufficient for localisation, however extracting a representation of underlying geometry of the environment from them is a challenging task. Indeed many areas of the environment may feature no point features whatsoever, leading to them having no representation within the SLAM map from which to estimate environmental geometry. In such cases these areas must be entirely treated as if they were obstacles and avoided, in-order to ensure safe navigation. Naturally these issues impose crippling restrictions on a robot's ability to navigate, and while indoor MAV navigation using point feature based SLAM systems has been investigated typically involve the construction of artificial point feature rich environments or severely constrains navigation to within a small volume of predetermined free space to ensure safety ([86],[88],[26],[5],[9],[105]), cases which bear little resemblance to real world scenarios. Other visual features such as lines and edges have also been utilized for indoor MAV navigation ([48],[55]) however once again

the sparsity of such features is often such that they alone are insufficient to provide the 3D geometric information required for navigation.

On the opposite end of the spectrum dense monocular visual SLAM methods produce highly detailed geometric maps, using every pixel of an image in estimating a highly detailed 3D model of the environment ([79],[47],[112],[111],[75],[56],[110]). However, this detailed representation is generally not required for navigation, where the scale at which navigation is conducted (determined by the size of the robot itself) dictates the level of geometric detail required. For example a quad-rotor navigating inside of an office space would require approximate geometry of any desks or furniture present so that it may navigate about such objects, and a detailed 3D reconstruction of other individual small objects however is not required, as they would have no affect on the decision making process used to navigate through the environment. Additionally the high computational cost associated with such dense approaches commonly necessitates the use of powerful parallel GPU hardware to achieve real-time performance, and the high memory requirements needed for such detailed mapping typically limits the size of the maps themselves. These factors again make such an approach ill-suited to use on many autonomous robotic platforms. Further, whatever the approach, camera based visual SLAM methods rely on sufficient visual information being observed in order to estimate the pose of the sensor. This can pose a problem in many environments with areas lacking in visual information.

Because of the differing information requirements between SLAM and navigation it makes intuitive sense to instead construct two maps forming different representations of the environment, one used for localisation and another to conduct navigation and obstacle avoidance. This still however, requires one to be able to accurately determine environmental geometry, and the navigation related complexities involved with monocular camera based SLAM approaches have turned many to using alternative sensors capable of observing environmental geometry directly. To this effect many autonomous vehicles have utilized LIDAR and laser range finders to produce geometric maps suitable for navigation. This includes considerable work concerning indoor MAV navigation ([38],[3],[1],[34],[35]), however the majority of these adopt a 2D representation of the environment, and the sensors involved are often highly costly limiting their potential widespread application.

An alternative to such sensors came with the development of cheap camera based RGB-D sensors such as the original Microsoft Kinect [115]. Using such RGB-D sensors, a geometrical map for path planning and collision avoidance can be created alongside of that of the SLAM system map used for localisation.

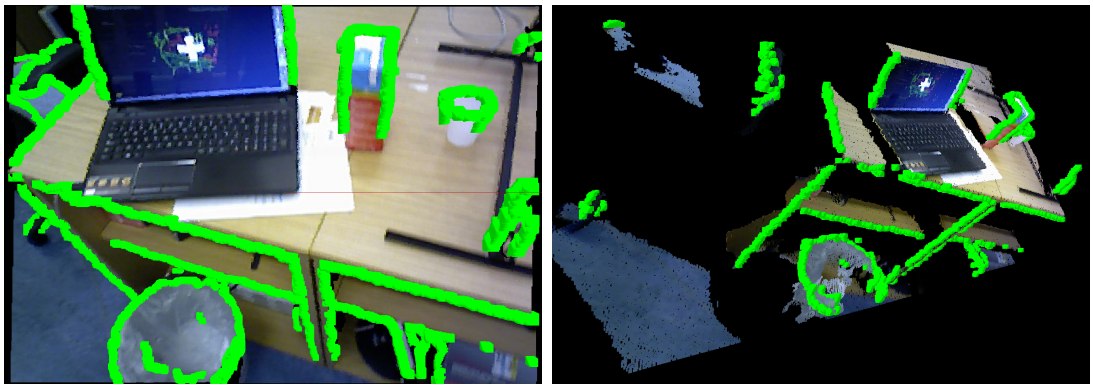


Figure 1.2: *Examples of back-projecting occluding depth edges to form semi-dense point clouds as shown in green.*

Similar to monocular SLAM, a wide array of RGB-D SLAM systems have been developed ([24],[47],[111],[112],[25],[39],[40],[109]) including both sparse point feature and dense approaches; many of which face similar challenges to their monocular counterparts. However there has been compelling work in the use of RGB-D SLAM for indoor MAV navigation including Huang et al [46] in which pose estimation and RGB-D odometry was performed on-board the vehicle while map optimization was performed from a grounds based PC, and the later work of Valenti et al in 2014 [106] in which the entire SLAM system ran on-board the vehicle. In this work we attempted to formulate a RGB-D SLAM approach to address some of the issues regarding use on MAVs, along with investigating how such a system should be utilized for navigation and path planning.

1.3 Edge Based RGB-D SLAM

The majority of SLAM systems have traditionally fallen into one of two approaches, being either sparse point feature based and only making use of specific areas of an image around which salient visual features are located, or taking a dense approach whereby every pixel of each image is utilized. Recent developments in the field of SLAM have also seen the emergence of so called "semi-dense" approaches notably with the introduction of the Large Scale Direct Monocular SLAM system LSD-SLAM [27] in 2014. The key idea behind this semi-dense approach being that instead of utilizing the entire image as in dense approaches, only pixels which are likely to provide useful localisation and tracking information should be utilized. In practice this can vastly decrease the amount of visual data that needs to be processed per frame, making such semi-dense approaches viable on a wide range of robotic platforms not equipped with the powerful parallel processing capabilities typically required for dense SLAM approaches.

This thesis specifically investigates applying such a semi-dense edge based approach to RGB-D SLAM by utilizing edges found in both the color and depth components of the RGB-D image. Intuitively such edge pixels are highly sensitive to change in sensor pose. Take for example a set of such edge pixels extracted from one RGB-D frame, and compare the values of those same pixels locations in the subsequent RGB-D frame, any small change in sensor pose between frames will likely cause a significant disparity between these pixel values. This is in contrast to pixels belonging to bland or textureless image regions (in either RGB or depth images), where the values of such pixels are unlikely to exhibit significant change in value from one frame to the next. In this way the use of only such edge pixels can be regarded as reducing the RGB-D image down to a smaller subset of pixels which contain useful information for sensor tracking and localisation. Additionally using such depth edges can allow such a system to maintain localisation even if there is a lack of visual detail in the scene (assuming that there is sufficient geometry detail), and similarly, vice versa for RGB color edges. A example of such depth edges extracted from an RGB-D point cloud is shown in Figure 1.2. Thus use of both RGB and depth edges allow such a SLAM system to operate in a wider range of scenarios.

As we intend such a SLAM system to be viable for use in autonomous navigation for small robotic platform it must naturally demonstrate real-time performance. The primary computational factors to consider are those of edge detection and edge registration, both of which need to be conducted for every RGB-D frame received. This thesis will address the implementation of these processes in detail, presenting approaches capable of achieving faster than real-time performance on a single CPU core.

1.4 SLAM Aware Path Planning

Let us again consider an autonomous robot using SLAM for mapping, localisation and navigation. A safe trajectory for such a robot is both collision free and also ensures that the SLAM system maintains localisation at all times. In an ideal hypothetical scenario the SLAM system used both never loses localisation and is perfectly accurate, effectively providing ground truth information with regards to localisation and environmental geometry needed for navigation. In such a scenario the SLAM system is effectively removed from the path planning problem and does not need to be considered when determining how the robot should move. In reality however, the SLAM system may lose localisation due to the vehicle's location (or motion) resulting in insufficient information for localisation being observed by the vehicle's sensors. Such a loss of localisation could cause

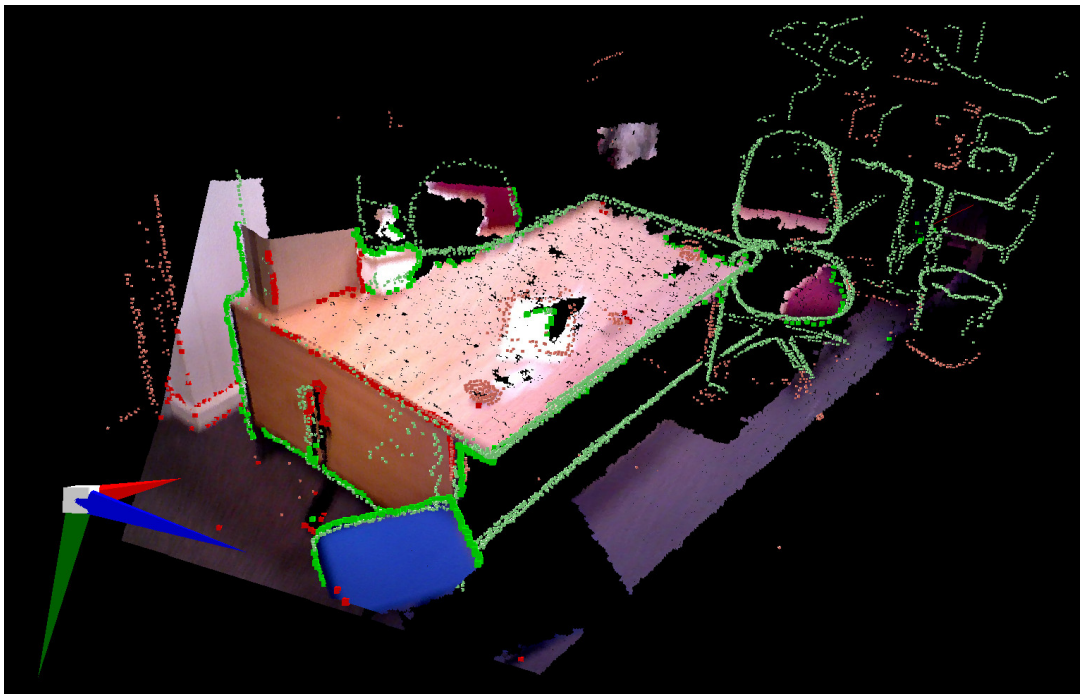


Figure 1.3: *Example of our proposed edge-based SLAM in operation.*

the vehicle to stray from the desired trajectory, potentially colliding with some obstacle in the environment.

Thus in practice the path planning process must carefully consider the SLAM system's ability to maintain localisation when determining safe trajectories. This involves checking that a potential trajectory always ensures that sufficient sensor information is available for the SLAM system to maintain localisation, based on the current constructed map. The specific implementation of this process however depends upon the SLAM system in question and what visual data is utilized for localisation.

Another relevant issue is that of changes to the SLAM map occurring during navigation due to loop closures, and also the accumulation of global drift leading to a map which, from a global reference frame is geometrically inconsistent. Such map changes may invalidate both calculated trajectories and the entire geometric map used for collision avoidance. To mitigate this path planning must be conducted in a manner that is robust to such global inconsistency. With this in mind we present a keyframe centric approach to path planning in Section 5, producing trajectories which consist of multiple segments each defined within the reference frame of a specific keyframe, which allows such trajectories to easily be warped in accordance with global changes to the SLAM map.

1.5 Contributions, Outline and Publications

1.5.1 Contributions

The work contained within this thesis describes a novel semi-dense RGB-D SLAM approach along with the associated methods of edge detection and point cloud registration. Additionally we present a novel path planning approach which conducts planning across multiple connected sub maps. This approach allows calculated paths to be robust to changes in the global map produced by the SLAM algorithm.

Our proposed SLAM algorithm utilizes edges extracted from both the RGB and Depth components of each RGB-D frame. The independent use of both color and depth features allows the proposed system to maintain tracking in situations where relying solely upon the depth or color components would result in tracking failure. At the time of writing this to our knowledge is the first semi-dense system making simultaneous use of color and depth from RGB-D data. Evaluation of this system shows that it compares favourably to a number of alternative CPU based RGB-D SLAM systems, displaying significantly lower computational cost, similar levels of accuracy, and consistent real-time performance at 30hz or higher.

We present a novel method of occluding depth edge detection which exploits the temporal similarity between frames of an RGB-D sequence. The locations of edges detected in the previous frame are used as a prior, to target edge detection to only image locations where edges are likely to be present. We show that by exploiting prior information in this manner our approach favourably compares to existing methods such as [12], having a significantly smaller computational cost.

Our SLAM system makes use of a custom edge based ICP registration method. We present extensive testing showing that compared to naive ICP registration our approach is more robust to down-sampling, converges in fewer ICP iterations, has a significantly lower computational cost per iteration, and can achieve greater accuracy within a given time frame. This core ICP component allows our entire SLAM system to achieve greater than real-time performance on standard CPU hardware.

Additionally we present a novel line based SLAM system. We propose methods of both performing line extraction from point cloud data, and performing registration between line segments and semi-dense edge point clouds. This line based SLAM is an extension of our semi-dense SLAM system, with a lower computational cost but reduced accuracy.

We present a method of belief space planning designed to navigate 3D environments while ensuring that localisation is maintained from bearing measurements to fixed landmarks. Finally we propose a keyframe sub map based method of path planning for use on MAVs using the prosed edge based SLAM to navigate around indoor environments, both ensuring obstacle avoidance and localisation via the SLAM system. We demonstrate the use of both this keyframe centric planner and the proposed SLAM system in use on-board of a quad-rotor MAV platform within an indoor flying arena.

1.5.2 Thesis Outline

A summary of the chapters within this thesis is as follows.

Chapter 2 : Edge Based RGB-D SLAM

This chapter presents our proposed Edge Based RGB-D SLAM approach, including our motivation behind creating such a system, and implementation of the various components. Our system uses a keyframe based approach in which both RGB and depth edges are extracted from each RGB-D frame. These edges are back-projected forming semi-dense "edge clouds" used for sensor pose estimation via ICP. Our method of edge based ICP registration is presented and evaluated along with our proposed method of fast depth edge detection exploiting the temporal similarity between RGB-D frames, and our proposed method of map optimization using an intuitive mass-spring system inspired approach. Finally the proposed SLAM system is evaluated using publicly available RGB-D datasets and mapping results are presented. This evaluation reveals that in comparison to a number of alternative systems, our proposed SLAM implementation is able to achieve a similar level of mapping and localisation accuracy, while having a significantly reduced computational cost many times smaller than that of other systems. This system was implemented from scratch in C# allowing easy portability between Linux and Windows, along with all other software described within this thesis.

Chapter 3 : Line Based RGB-D SLAM

This chapter presents an investigation into modifying the proposed edge based SLAM to utilize high level 3D line features. A method of extracting straight linear segments of semi-dense edge clouds is proposed, along with an ICP variant designed to perform registration between a set of such 3D linear features and a semi-dense edge cloud. Similar

evaluations are presented for this line feature based SLAM system as were for our edge based approach.

Chapter 4 : Belief Space Planning

This chapter presents an investigation into the use of belief space planning for generating trajectories for ensuring vehicle localisation based upon on-board sensor measurements. A belief space planning method is presented generating trajectories through complex 3D environments that aim to ensure localisation by measurement of various point landmarks placed within the environment. Results are presented comparing trajectories attempting to maintain localisation with those simply taking the shortest route between the desired start and goal locations.

Chapter 5 : Path Planning

This chapter presents our employed method of path planning targeted towards RGB-D SLAM based MAV navigation. A discussion of how to generate safe MAV trajectories is provided, laying out the necessity of not only collision avoidance but of ensuring the vehicle's RGB-D sensor always observes sufficient information for SLAM localisation to be maintained. A keyframe centric method of path planning is then proposed whereby separate navigation graphs are generated for each keyframe, ensuring both collision avoidance and sufficient RGB-D sensor data. These graphs are connected based upon detected loop closures, forming a global navigation graph robust to the effects of SLAM drift and global error. Trajectories generated by the proposed planning approach are presented, both in scenarios where the RGB-D sensor was being carried by hand and where it was mounted to a MAV navigating within an indoor flying arena.

Chapter 6 : Conclusions

Finally we summarize the results of the previous chapters, lay out contributions, and discuss potential directions of future work and investigation.

1.5.3 Publications

The work contained within this thesis has been peer-reviewed and published in the following publications:

1. Laurie Bose, Arthur Richards. Determining Accurate visual Slam Trajectories Using Sequential Monte Carlo Optimization. American Institute of Aeronautics and Astronautics (AIAA) Guidance, Navigation, and Control Conference (GNC) 2013. [7]
2. Laurie Bose, Arthur Richards. Mav Belief Space Planning In 3d Environments With Visual Bearing Observations. International Micro Air Vehicle Conference (IMAV) 2013. [8]
3. Laurie Bose, Arthur Richards. Fast Depth Edge Detection and Edge Based RGB-D SLAM. International Conference on Robotics and Automation (ICRA), 2016. [8]

The work of publication 3. is fully described across Chapters 2 and 3, while publication 2. is discussed in Chapter 4.

Chapter 2

Edge Based RGB-D SLAM

In this chapter we present our edge based RGB-D SLAM approach, along with a discussion of our motivation behind developing such a system and a brief background section. Much of this work was originally presented in Bose and Richards [6] at ICRA 2016. Our method extracts both RGB and depth edge pixels from each RGB-D frame, back-projecting these to form two semi-dense clouds which are used as the basis of sensor tracking and map construction.

2.1 Background

Consider the task of navigating a previously unseen environment. In order to keep track of how you are moving through the world it is necessary to memorize the layout of the environment and the locations of landmarks within it. This is the process of constructing a mental map of the environment, updating and refining it as new information becomes available. Simultaneously, this same mental map is also being used to track your location within the environment and determine the best route to take to achieve your current goal. The goal of SLAM (Simultaneous localisation and mapping) in robotics is to essentially replicate this process.

2.1.1 Probabilistic SLAM Description

Stated formally the SLAM problem is to determine the probability distribution,

$$P(x_t, m_t | z_{1,t}, u_{1,t}) \tag{2.1}$$

for the current time t, where x_t is the current state of the robot at time step t, m_t is the constructed map of the world, $z_{1,t}$ is the set of all sensor observations made up to time t and $u_{1,t}$ the set of all movement commands executed by the robot up to t. This probability distribution is referred to as the "SLAM posterior". This jointly describes the belief of a robot's current state and the constructed map of the world, given all previous sensor observations and movement commands. The form of x_t is dependant upon the robot, for example a simple wheeled robot's state may simply consist of a 2D position and orientation, while a flying quad-rotor's state would consist of a full 6DOF pose. Additionally the map m can take many forms depending upon the sensors and SLAM algorithm being used. For example m could take of the form of a occupancy grid, or a list of 3D locations of salient features observed by a camera. Thus the complexity of the SLAM posterior can vary significantly depending upon the SLAM problem in question.

From Bayes' rule it is possible to rewrite the SLAM posterior of Equation 2.1 in the form.

$$P(x_t, m_t | z_{1,t}, u_{1,t}) = \eta P(z_t | x_t, m_t, z_{1,t-1}, u_{1,t}) \times P(x_t, m_t | z_{1,t-1}, u_{1,t}) \tag{2.2}$$

Where η is a normalizing constant ensuring the integral over the entire distribution correctly sums to 1.

In practice a number of assumptions are made regarding the SLAM posterior in order to greatly simplify the problem.

- Firstly the assumption is made that the robot's state x_t is only dependant upon its previous state x_{t-1}, and the movement command issued at the same time step u_t. That is x_t follows a first order Markov process.
- It is assumed that the sensor measurements, and controls at each time step are uncorrelated.

- The environment is assumed to be static and not change over time, thus allowing a single variable m to replace $m_0, m_1, m_2 ... m_t$.

These assumptions allow Equation 2.2 to be simplified.

$$P(x_t, m|z_{1,t}, u_{1,t}) = \eta P(z_t|x_t, m) \times P(x_t, m|z_{1,t-1}, u_{1,t}) \tag{2.3}$$

Then using total probability theorem and these assumptions Equation 2.3 can be manipulated to the form.

$$P(x_t, m|z_{1,t}, u_{1,t}) = \eta P(z_t|x_t, m) \times \int P(x_t, m|x_{t-1}, z_{1,t-1}, u_{1,t}) \times P(x_{t-1}|z_{1,t-1}, u_{1,t-1}) \delta x_{t-1} \tag{2.4}$$

Finally it can be shown by using the definition of conditional probability the distributions inside the integral can be rewritten in the form.

$$P(x_t, m|z_{1,t}, u_{1,t}) = \eta P(z_t|x_t, m) \times \int P(x_t|x_{t-1}, u_{1,t}) \times P(x_{t-1}, m|z_{1,t-1}, u_{1,t-1}) \delta x_{t-1} \tag{2.5}$$

This form consists of three components, namely the "observation model".

$$P(z_t|x_t, m) \tag{2.6}$$

The "motion model" (or "process model").

$$P(x_t|x_{t-1}, u_{1,t}) \tag{2.7}$$

And the previous SLAM posterior.

$$P(x_{t-1}, m|z_{1,t-1}, u_{1,t-1}) \tag{2.8}$$

In this form the SLAM problem is given as a recursive Bayes filter.

The "motion model" as given by Equation 2.7 describes how belief of the robot's state evolves from one time step to the next. This distribution is of course dependant upon the form of movement used by the robot. The robot's motion will in practice never perfectly follow the desired controls. This is due to issues such as wheel slippage, wind or any number of other factors. Thus the uncertainty of the robot's state will naturally increase with each motion.

The "observation model" as given by Equation 2.6 gives the likelihood of a specific observation z_t being made given a specific map and vehicle state. The form of this distribution depends upon the type of sensor being used, along with it's the accuracy and noise characteristics.

In practice the continuous probability distributions of the SLAM posterior, motion model, and observation model must either be approximated discretely or greatly simplified using a set of assumptions in order for the SLAM problem to become tractable.

Common approaches include assuming that all distributions are Gaussian, using a particle filter based approach to approximate the distribution, or simply not explicitly modelling the entire distribution. Indeed EKF based SLAM approaches which model the entire belief but operate under the assumption the belief is Gaussian, still quickly become computationally intractable as the constructed map size increases.

2.1.2 Keyframe Based SLAM

Popularized by the work of Klein and Murray in [59],[58], the concept of keyframe based SLAM has seen wide adoption over the last decade. In general, a keyframe typically consists of a single set of sensor data, features extracted from said data, and an estimate of the global sensor pose at which this sensor data was obtained. A keyframe based system constructs a map consisting of a set of such keyframes, beginning with a single keyframe using the data acquired upon the system's initialization, with the addition of more keyframes as new areas of the environment are observed. The initial keyframe is typically taken to have its global pose fixed at the identity pose (at the origin), with

the global poses of subsequent keyframes being estimated relative to this initial one. Features are extracted from each new set of sensor data and registered with features stored within the keyframes of the map, providing an estimate of the sensor's current pose.

Such a key-frame based approach can also be considered as effectively partitioning the observed environment into a number of small sub maps, the locations of which are each estimated within some global reference frame, forming a global SLAM map of the environment. Such a partitioning into multiple independent submaps makes the process of global map optimization as discussion in the following Section 2.1.3 far more manageable and computationally tractable for large maps. Many works have used similar submap approaches enabling them to construct and manage large scale maps in real-time ([113],[49],[68]). We adopt such a keyframe based approach in our edge based RGB-D system presented in Section 2.3 onwards.

2.1.3 Pose Graph Optimization

The poses of such keyframes inevitably have some degree of error and uncertainty due to the accumulation of tracking errors leading to global drift in the sensor's estimated position. These issues can be mitigated however, by the detection of loop closures and map optimization.

Such loop closures define a set of relative pose constraints between various pairs of keyframes. In-order to incorporate this information into the map, the estimated keyframe poses need to be updated such that they comply with these constraints. This problem can be visualized as a pose graph in which the nodes of the graph represent the current estimated keyframe poses and the edges between nodes represent the relative pose constraints between pairs of keyframes from detected loop closures. The problem of finding the set of keyframe poses which best comply with these constraints can thus be regarded as a pose graph optimization (PGO) problem as originally introduced by [72], in which a set of poses are estimated from a set of noisy relative pose constraints. Solving such graph optimization problems in a robust and efficient manner is itself a large area of research, with many widely used solutions such as the well known "g2o" framework [61]. Such map optimization is also typically conducted continuously on a separate CPU thread from that conducting sensor tracking and localisation, in order to ensure one does not impede the performance of the other. In this work we investigated using an intuitive relaxation based approach to PGO as presented Section 2.8.

2.1.4 Depth Sensor Calibration

It should be noted that common consumer structured light based RGB-D sensors, such as the Microsoft Kinect and Asus Xtion, suffer from significant noise, discretization and distortion issues in regards to their depth measurements. These issue are proportional to the depth itself as demonstrated by [57], becoming a significant problem for depth measurements over 3 meters. Further such issues can adversely effect any SLAM or sensor tracking applications that rely on RGB-D depth data.

However, the distortion issue can be largely alleviated by calibration and post-processing of the raw depth data output by the sensor, as demonstrated in [42],[17],[100],[13]. As a brief overview, the distortion present in the depth images produced by the RGB-D sensor can be formulated on a per pixel basis. Let $d_{x,y}$ denote the distorted depth value associated with the depth image pixel at (x, y), and similarly let $t_{x,y}$ denote the true (undistorted) depth value associated with said pixel. The value of each pixel in the depth image is then given by $d_{x,y} = f_{x,y}(t_{x,y})$, where $f_{x,y}$ is the distortion function which when applied to $t_{x,y}$, produces the actual measured distorted depth value $d_{x,y}$. Similarly the true depth of each pixel is given by $t_{x,y} = f_{x,y}^{-1}(d_{x,y})$ where $f_{x,y}^{-1}$ is the associated undistort function. Note that each pixel has its own unique distortion functions $f_{x,y}$, $f_{x,y}^{-1}$, and further these distortion functions will vary between RGB-D sensors (even those of the same type i.e. two different Xtion sensors). If we observe a known piece of geometry (such as a flat planar surface) from a known sensor position, the true depth values $t_{x,y}$ which should be associated with each pixel can be calculated and then compared to the actual measured depth values $d_{x,y}$. By obtaining multiple such pairs of true and measured depth values for each pixel, a polynomial approximation of each pixel's undistort function $f_{x,y}^{-1}$ (and $f_{x,y}$) can be determined. We can then attempt to undistort each depth image produced by the sensor by applying $f_{x,y}^{-1}$ to each pixels measured depth $d_{x,y}$, greatly improving the quality of the produced RGB-D cloud at ranges above 3 meters. In our work we determined these distortion patterns by simply acquiring depth images of large flat surfaces at different distances from the sensor (with the sensor being approximately normal to the surface in question), an example of the resulting improvement in depth quality is shown in Figure 2.1.

2.2 SLAM System and Sensor Requirements

Since this work is primarily motivated towards SLAM based indoor autonomous vehicle navigation, primarily on MAV platforms, there were a number of important factors and

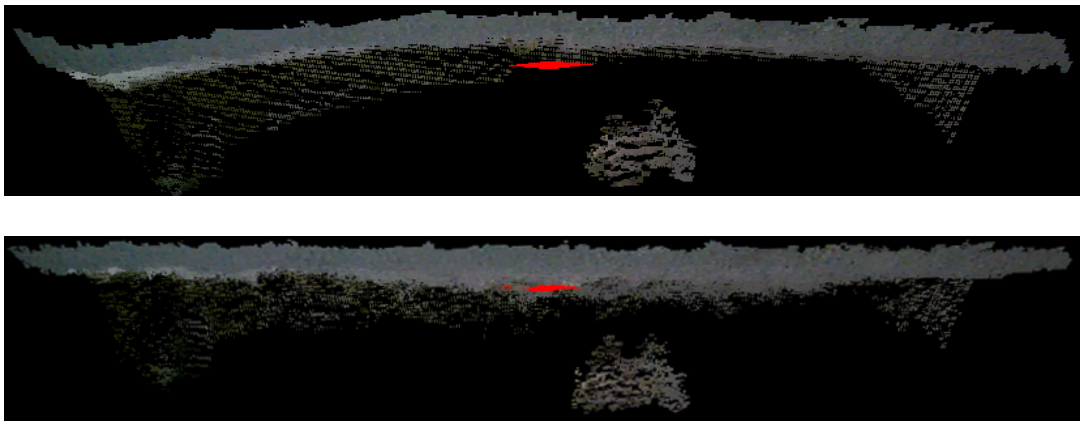


Figure 2.1: *A RGB-D cloud of a flat ceiling 2 meters from the sensor, before (top) and after (bottom) distortion correct has been applied.*

requirements to consider when determining which sensors and form of SLAM system were suited to the task at hand.

2.2.1 Geometric Configuration Space Construction

A typical indoor environment will consist of many obstacles and passageways, which have to be carefully navigated in order to ensure the vehicle does not experience a fatal collision during operation. Doing so requires some geometric representation or map of the environment, expressing where such obstacles (and open free space) are located, and where the vehicle may safely be positioned. This geometric map is then used in calculating safe obstacle free paths which the vehicle can use to navigate its surroundings.

In practice the larger the vehicle, the more approximate this geometric map of the environment can be since geometric details significantly smaller than the vehicle itself become largely irrelevant in terms of deciding how to navigate the environment. A group of such small geometric obstacles can simply be approximated by a single large obstacle with little effect on the final trajectories produced by the path planning process. Thus during autonomous navigation operations, an approximate geometric map of the environment is used to conduct path planning, while simultaneously a different feature based map is used by the SLAM system for tracking and localization.

Naturally this geometric map of the environment must be constructed during operation using information from the vehicle's sensors and SLAM system. The difficulty of this construction task varies greatly depending on the choice of sensor and SLAM system methodology, with some approaches being better suited than others.

2.2.1.1 Geometry From Monocular Methods

There is a rich history of literature tackling the problem of determining 3D geometry from 2D images. Methods such as space carving [62] and structure from motion (SFM) ([104], [99], [16]) are able to reproduce highly detailed dense 3D models, however, typically require accurate relative poses describing where each image was acquired. Estimates of such poses could be provided by the SLAM system; however, such dense reconstruction methods still remain computationally expensive and also provide a level of detail in their reconstruction unnecessary for conducting path planning. An alternative to using such a dedicated reconstruction approach is instead, to try and extract an approximate geometrical map from the map constructed by the SLAM system itself.

There are many forms of Visual SLAM systems using a single monocular RGB camera, with one of the most common approaches making use of sparse point features extracted from the sensor images. The sparse maps constructed by such systems then consist of the estimated locations of observed point features, such as that shown in Figure 2.2.

It is fairly obvious however, that such maps in general provide little information regarding the geometry of the environment and an approximate geometric map reconstructed from such a sparse set of points is highly unlikely to be fit for use in navigation and reliable obstacle avoidance. In order to construct a geometric map of decent quality that does not severely limit path planing and navigation, would require an abundance of point features to be present across all surfaces of the environment. Works such as [86] constructed arbitrary environments in which such an abundance of point features is present, and while successful path planning is demonstrated within these environments such a scenario is far removed from real-world application. Indeed, in a typical real-world environment, point features are often completely absent in many areas due to a lack of texture and poor lighting, and in such areas the SLAM system's sparse map would provide no usable geometric information at all.

For these reasons sparse point feature based approaches using a single monocular RGB camera are typically ill suited for autonomous indoor navigation.

Alternatively, dense monocular SLAM approaches such as the dense mapping demonstrated by DTAM [79], construct detailed geometric maps of the environment, from which an approximate geometric map for navigation purposes can be easily extracted. However such approaches require a high level of computing power typically unavailable on on-board computers carried by such MAVs systems, and so would not be able to achieve real-time performance. Additionally, the maps produced by such dense methods again provide a high level of detail not required for path planning purposes and quickly

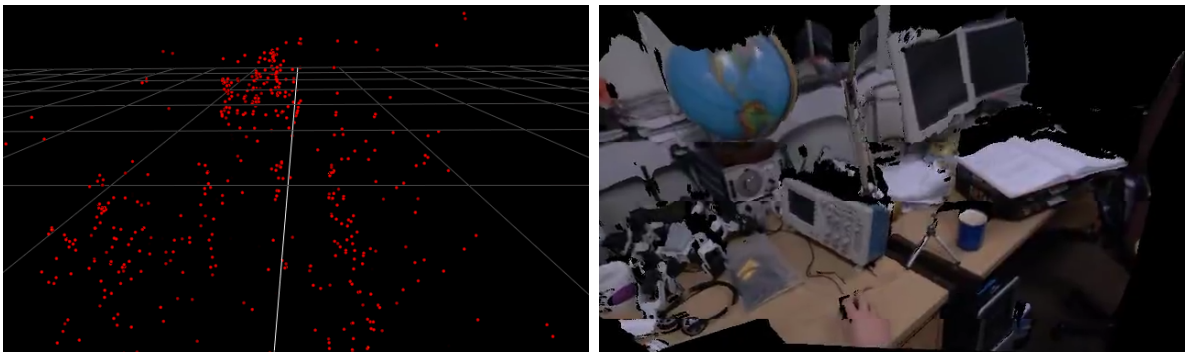


Figure 2.2: *Left, a sparse point feature based map constructed by PTAM [58]. Right, a dense map constructed by DTAM [79]. Note the difficulty in determining the underlying environmental geometry from the sparse map.*

consume a large amount of memory which greatly limits the maximum volume that can be mapped, placing further limitations on the vehicle's navigation capabilities.

2.2.1.2 Depth Sensor Geometry

As we have discussed previously, the task of constructing approximate 3D geometry using a single RGB camera can be a highly challenging task to perform reliably in real-time, especially on the limited on-board hardware typical of MAV systems. A currently far more practical approach is thus to simply equip the MAV with RGB-D sensors which can take direct measurements of the physical geometry of the environment. This greatly simplifies the task of constructing a geometric map for path planning, reducing the amount of computational work required and allowing more resources to be focused in other areas. Additionally RGB-D sensors remove a great deal of ambiguity, such as the physical scale of the constructed map, which otherwise would have to be estimated based on inertial measurements if relying purely on a standard monocular camera.

2.2.2 Real-Time Performance Limitations

An additional requirement is that the SLAM system used must be capable of real-time performance on the vehicle's on-board hardware. Ideally the SLAM system should be capable of running at least at the same rate at which data is acquired by the vehicle's primary sensor. In the case of standard RGB and RGB-D cameras this would require the SLAM to run at a frame rate of 30Hz or higher.

Falling far below this update rate of 30Hz has several degrading effects. First, since the SLAM is not able to keep up with the rate at which images are being produced

by the sensor, many images will be skipped over. This results in there being more time between the images actually being used for tracking and localization. The further apart two images are temporally, the more time there is between them in which camera motion can take place, generally leading to there being less common information shared between sequential images. Each of these factors increases the likelihood of tracking failure occurring whenever the sensor is undergoing motion. A MAV operating with such a low frame-rate SLAM system would be forced to greatly limit its rate of movement, in order to minimize the probability of such tracking failures occurring. Additionally lowering the frame rate of the SLAM system reduces the rate at which pose estimates are being provided to the vehicles controller. This in turn limits the rate which the vehicle's controller can make adjustments in order to accurately follow a specific trajectory, which must then be accounted for by taking wider paths about obstacles in order to ensure safety.

A wide range of on-board computers have been demonstrated on MAVs platforms, however compared to desktop PCs all of them are naturally somewhat limited in terms of raw computing power. Most notably, dedicated desktop GPUs are typically far too heavy, large, and power hungry to be practical on such a small flying vehicle. There are several computationally intensive dense SLAM approaches which typically require such GPUs in order to achieve real-time performance, DTAM and Kinect Fusion being two such examples. Both of these systems create dense geometric maps of the observed environment and both also perform tracking using all of the information in each camera image, rather than using only certain features extracted from these images. These two aspects are in general far too computationally intensive to be performed in real-time on such on-board MAV hardware (our target MAV platforms specifically carries a single desktop CPU only) and thus such dense methods are currently unsuitable for such autonomous navigation purposes.

2.3 RGB-D Edge SLAM Overview

The requirement from previous sections were used as guidelines in the development of a SLAM system, specifically with autonomous indoor navigation in mind.

Our proposed RGB-D SLAM system follows the standard keyframe based approach as laid out in Section 2.1.2, where the SLAM system map consists of a set of keyframes $K = \{K_0, K_1, ...\}$, each with an estimated pose and a set of observed features. The sensor's current pose is then estimated by registering features extracted from the current

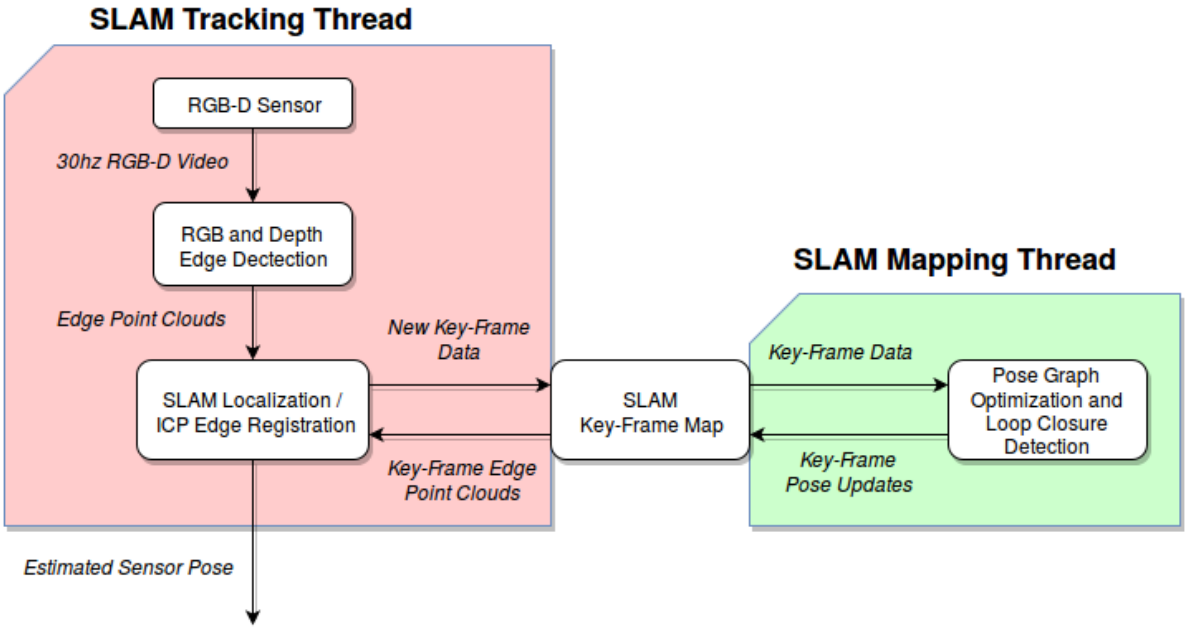


Figure 2.3: *Overview of the proposed SLAM system.*

RGB-D frame with features stored by the keyframes of the map.

Specifically our system makes use of edge features extracted from both the RGB and depth components of the RGB-D frames. The pixels of such edges are back-projected to form semi-dense point clouds, which are then used in both ICP based sensor tracking and loop closure detection.

An overview of the whole system is illustrated in Figure 2.3, Feature extraction and localization is performed on one thread while a second thread improves the consistency of the map by performing loop closure detection and pose graph optimization.

The remainder of this section introduces the adopted system formulation and discusses the reasoning behind the use of edge features and ICP based registration using their associated semi-dense point clouds. After which edge feature detection, edge based ICP registration, Loop closure detection, Pose-graph optimization and system evaluation are then given in the following sections.

2.3.1 Formulation

Each map keyframe has an associated estimated pose representing where the sensor was located when the keyframe's data and features were observed. We denote $P = \{\mathbf{P}_0, \mathbf{P}_1, ...\}$ to be the set of all estimated keyframe poses, with $\mathbf{P}_i \in P$ denoting the

pose of the i^{th} keyframe $K_i \in K$. The sensor's pose is assumed to have a full six degrees of freedom and hence, each $\mathbf{P}_i = (\mathbf{t}_i, \mathbf{q}_i) \in P$ is also a 6DOF pose with $\mathbf{t}_i$ and $\mathbf{q}_i$ denoting the translation and rotation components respectively. This translation component $\mathbf{t}_i$ is trivially represented as 3D vector and the rotational component $\mathbf{q}_i$ as a unit quaternion. Each of these estimated poses $\mathbf{P}_i \in P$ are in the same "world" reference frame, the origin of which we define as being located at the pose of the initial keyframe $\mathbf{P}_0$ where the SLAM system was initialized. In this way the pose of the initial keyframe $\mathbf{P}_0$ is defined to be fixed at the origin and each subsequent keyframe pose is estimated relative to it. Similarly the current estimated pose of the sensor denoted $\mathbf{X}$, is also given in the "world" reference frame.

In addition to an estimated pose each keyframe K_i also stores a complete frame of RGB-D data denoted F_i, along with two semi-dense point clouds formed from the features detected in F_i. These are the clouds associated with detected depth edges D_i and RGB edges I_i respectively. Thus in summary each keyframe $K_i = \{\mathbf{P}_i, F_i, D_i, I_i\}$ stores a frame of RGB-D data F_i, an estimate of where that data was observed relative to where the SLAM was initialized $\mathbf{P}_i$, and depth and RGB edge point clouds D_i, I_i formed from the edges detected in F_i.

2.3.2 RGB-D Edge Features

In color images the majority of such edges are either due to the borders of foreground objects contrasting against others located further from the camera, or due to sharp changes in texture or material on a particular surface. Similarly in depth images, edges due to local pixel value discontinuities are due to the geometric borders of foreground objects obscuring other objects and surfaces located at a further distance from the camera. Despite consisting only of a small fraction of the total pixels present in the original image, these sets of edge pixels (for both color and depth images) still retain most of the relevant information regarding the high level structure of the objects present in the image. Additionally such edge pixels can naturally be considered those which are most sensitive changes in camera pose; that is those pixels located upon the edges detected in one image are likely to experience a significant change in value (and no longer lie upon an edge) in the following image if the camera has undergone a change in pose. Contrast this to pixels lying within a area of blank or low contrast texture, even if the camera's pose has changed between two subsequent images, many of these pixel's values will not experience any significant change in value and providing little or no useful information for determining the change in camera pose.

Taking this into consideration, extracting an image's edge pixels can be viewed as an intelligent form of downsampling, producing a smaller subset of salient pixels which retain the majority of useful information for camera tracking. This is the primary motivation for choosing edges as the basis for the system's tracking and localisation. Working with such a reduced selective set of pixels substantially simplifies the task of registering and determining the relative camera pose between two RGB-D frames, making such edge points a strong feature choice for our requirements.

2.3.3 Edge Point Clouds and ICP Registration

The depth image component of an RGB-D frame provides an estimated depth for the majority of the pixels in both the RGB image (and the depth image itself), which can then be used to back-project these pixels forming a dense RGB-D point cloud typically consisting of hundreds of thousands of points. However, instead of back-projecting all pixels in this manner, we can instead only back-project the sets of edge pixels extracted from each RGB-D frame. This forms two semi-dense point clouds corresponding to edge pixels found in the RGB and depth image components respectively.

Under this set-up, the process of registering two RGB-D frames and thus estimating the relative camera poses at which they were acquired can be achieved by using a combined ICP process to estimate the transformation which best aligns their corresponding RGD and depth edge point clouds. This ICP based edge point cloud registration is typically multiple orders of magnitudes faster than using ICP to register the full RGB-D point clouds, due to the vastly reduced number of points. Despite this vastly reduced computational cost, this edge registration typically provides a similar or even superior level of accuracy in indoor scenarios as demonstrated in 2.6.3.

2.3.4 Relaxation Based Pose Graph Optimization and Loop Closure

Sensor tracking will inevitably contain some degree of error which can accumulate leading to substantial errors in the keyframes of the map P. Loop closure detection and pose graph optimization are used to correct for such errors and improve the accuracy and consistency of the map. These tasks are carried out in parallel on a separate thread to that conducting sensor tracking. A relaxation based Pose Graph Optimization (PGO) method is used to find the set of keyframe poses which best comply with the detected loop closure constraints, and thus are also likely to provide the most accurate and consistent

map. This relaxation based method is analogous to an intuitive mass-spring based system. Such a system will come to rest in an energy minima such that the positions of the masses are compliant with the springs connecting them. A full description of this system based on an intuitive mass-spring system analogy is given in Section 2.8.2.

2.4 Depth Edge Detection

The depth image component of an RGB-D video stream provides 3D geometric information of the scene in view. In such an image physical objects occluding other further away objects / surfaces from the camera's view give rise to discontinuities between the values of neighbouring pixels. Given any pair of such pixels, the pixel of smaller depth belongs to the occluding object (being nearer to the camera), while the other pixel belongs to the object (or surface) that is being occluded. Thus such depth image discontinuities give rise to two types of edges. Occluding edges whose pixels belong to the occluding foreground objects, and occluded edges whose pixels belong to objects and surfaces being occluded.

Note that the two edge types always occur in pairs, for every occluding edge there is an associated occluded edge. Further occluded edges do not physically represent objects in the scene and thus are not useful for SLAM or camera tracking purposes. This is illustrated in Figure 2.4, where occluding edges are shown to actually represent the boundaries of physical objects, while occluded edges are essentially the edges of "shadows" cast by those objects. The following proposed method of depth edge-detection, searches depth images for discontinuities and thus inherently detect both these types of edges, however, occluded edges are simply ignored.

2.4.1 Occluding Depth Edge Detection

We employ a two step search process to find pixels belonging to occluding edges within a depth image. This involves examining the pixels of the depth image twice, first by rows then by columns. In either case an identical process is used to locate local depth discontinuities within a given line of pixels (be it a row or column). This pixel search process, outlined in Algorithm 2.1, iterates over a given list of pixels, skipping over those of invalid value (pixels with a zero depth value) while keeping track of the last found valid pixel.

At each iteration the value of the current pixel (if valid) is compared to that of the last

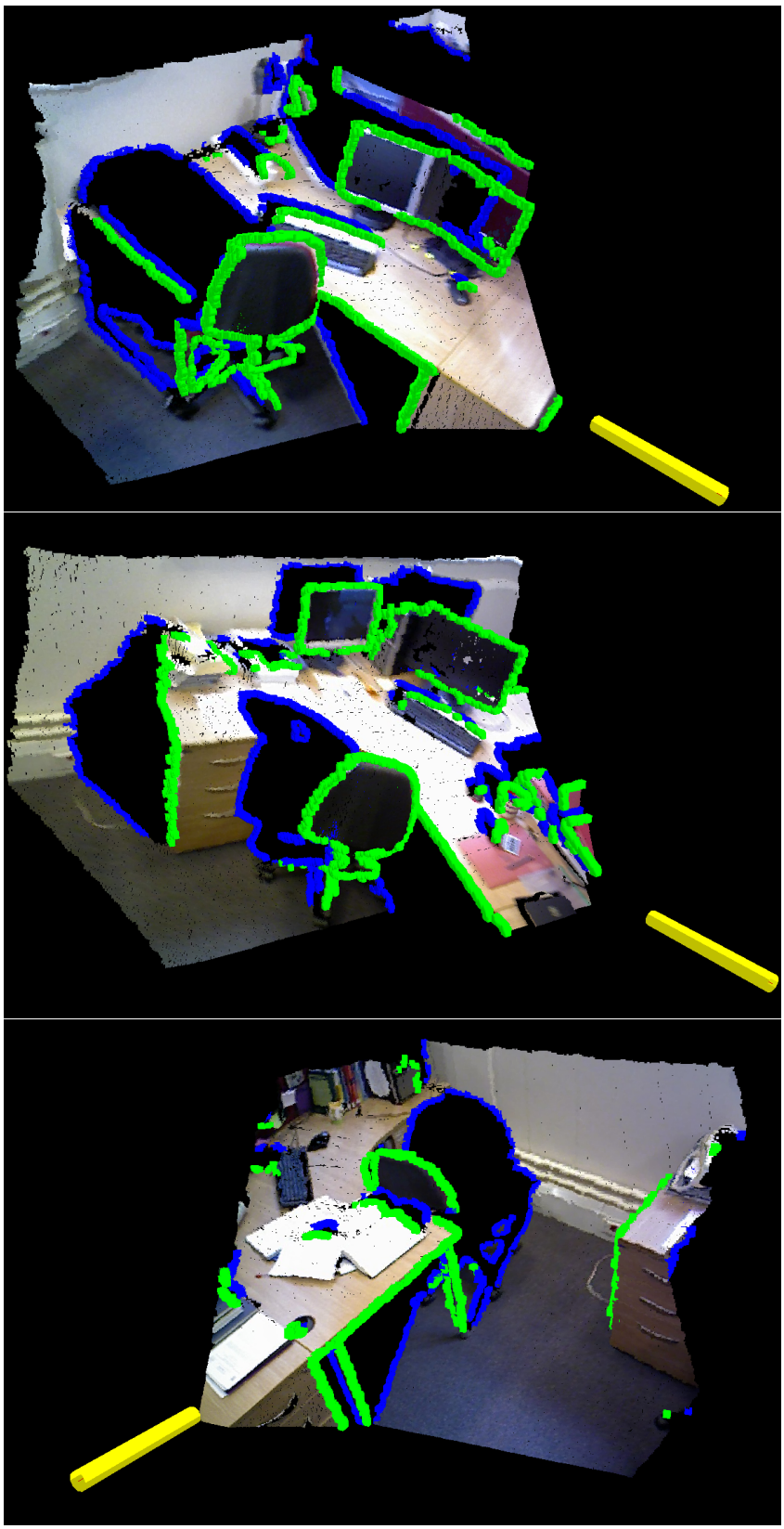


Figure 2.4: *Examples of edges due to depth image discontinuities. Occluding edges are drawn in green, occluded edges in blue and the sensor's position indicated by the yellow cylinder.*

INPUT: $P = \{(V_n, L_n) : n \in \{0...N\}\}$
// P is a list of depth pixels along a line
// V_n value of n^{th} pixel
// L_n image location of n^{th} pixel
T *// edge detection threshold ratio value*

OUTPUT: $E = \{\}$ *// edge pixel locations*

$last\_valid = 0$ *// index of the last observed valid pixel*
for $n = 0$ to N **do**
  if $V_n \neq 0$ **then** *// if current pixel value is valid*
    $threshold = Min(V_n, V_{last\_valid}) \times T$
    // check for large differences between pixel values
    if $(V_{last\_valid} - V_n) > threshold$ **then**
      $E = E \cup L_n$
    else
      if $(V_n - V_{last\_valid}) > threshold$ **then**
        $E = E \cup L_{last\_valid}$
      end if
    end if
    $last\_valid = n$
  end if
end for
return E

Algorithm 2.1: $P\_Scan(P, T)$
Detect large differences between the values of neighbouring pixels indicating the presence of occluding edges

found valid pixel. If the difference between these two values is above a certain threshold then this discontinuity in depth between nearby pixels is deemed large enough to indicate the presence of an occluding edge. If this is the case the pixel with the smaller of the two depth values (that being closer to the sensor) is identified as an occluding edge pixel. The other pixel which corresponds to an occluded edge is ignored. The previously mentioned threshold is given by $d \times T$ where d is the smaller of the two pixel values and T is a sensitivity constant. Having such a proportional threshold is necessary due to the nature of structured light RGB-D sensors where, both noise and spacing between readable values is proportional to depth [57, 69]. The two steps of this search process constitute scanning across the depth image in two orthogonal directions (by rows going left to right and by columns going top to bottom). Note that it is possible that certain pixels may be identified as belonging to an occluding edge in both of these steps (typically due to lying upon diagonal edges). However rather than having certain pixels appearing twice among the list of detected edge pixels, a final culling step is performed to remove such duplicates. In practice the organised structure of the image is exploited to minimize the computational time of this process.

2.4.2 Sub Image Depth Edge Detection

When working with a standard RGB-D video stream ($30hz$ 640×480), each depth image is highly likely to be similar to the previous one due to the relatively small camera movements between video frames. Occluding edge pixels are thus also likely to occur in similar locations from one depth image to the next due to this image similarity. This prior knowledge of where edge pixels are likely to occur can be exploited to significantly speed up the occluding edge-detection process. This is achieved by only searching for edge pixels in specific areas of the depth image (around which edge pixels were previously detected) instead of simply searching the entire image in a brute force manner. Our implementation of this concept considers each depth image as consisting of an $N \times M$ grid of smaller equally sized images (referred to here as "image patches") rather than a single image. For example a grid size of 4×3 would consider a 640×480 depth image as consisting of 12 image patches, each 160×160 pixels in size.

An $N \times M$ array of boolean edge-detection flags F is also stored and used to determine which image patches should be searched for occluding edge pixels. With each new depth image from the RGB-D video stream, all image patches whose associated flag is set to true are searched using the same occluding edge-detection process described in Section 2.4.1. The results of these searches are then used to update the edge-detection flags,

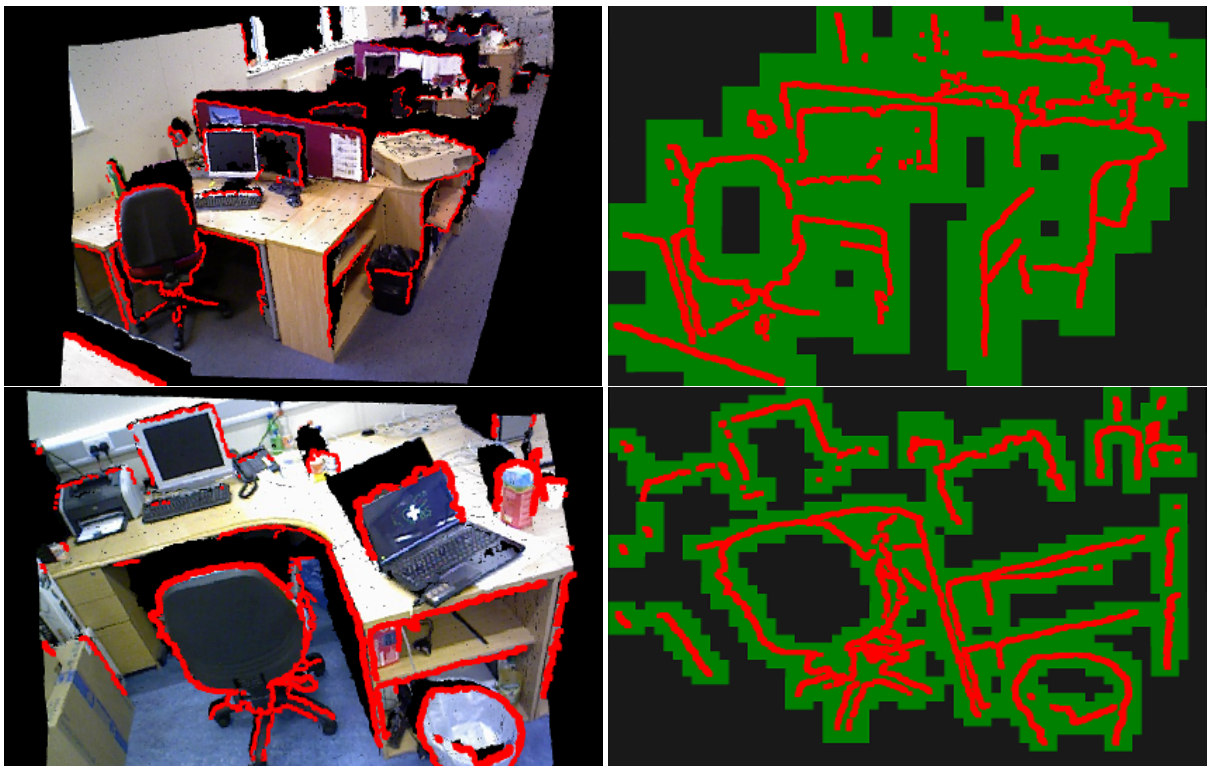


Figure 2.5: *Examples of detected occluding depth edges (Red) and the associated detection flag values for each image patch (Top, a grid consisting of* 16×12 *image patches. Bottom a grid of* 32×24 *image patches). Image patches flagged for detection are highlighted in green, while those not flagged are drawn in black.*

// image patches P and their associated boolean flags F
INPUT: $P = \{P_{xy} : x \in \{0...N\}, y \in \{0...M\}\}$
$F = \{F_{xy} : x \in \{0...N\}, y \in \{0...M\}\}$
T *// edge detection threshold value*
R *// number of patches to randomly search*
K *// row and column skip value*

OUTPUT: $E = \{\}$ *// edge pixel locations*

Set R randomly selected flags from F to True
for all $P_{ij} \in P$ **do**
  if F_{ij} **then** *// if patch flagged for edge detection*
    $row\_edges = \{\}$
    $col\_edges = \{\}$
    for all K^{th} *rows R of image patch P_{ij}* **do**
      $row\_edges = row\_edges \cup P\_Scan(R, T)$
    end for
    for all K^{th} *columns C of image patch P_{ij}* **do**
      $col\_edges = col\_edges \cup P\_Scan(C, T)$
    end for
    if $row\_edges \neq \{\}$ **or** $col\_edges \neq \{\}$ **then**
      $E = E \cup row\_edges \cup col\_edges$
      $F_{ij} = True$
      Set flags of neighbouring image patches to True
    else
      $F_{ij} = False$ *// reset patches flag*
    end if
  end if
end for

Remove any duplicate pixels from E
return E, F

Algorithm 2.2: *Occluding_Edge_Detection*(P, F, T, R, K)
Detect occluding depth edge pixels within flagged image patches and update edge detection flags

deciding which regions of the next depth image will be searched for edges. Figure 2.5 shows examples of such edge detection flag arrays. Image patches in which edge pixels are detected have both their own flag and those of their neighbouring patches set to true. All other patches which are not affected by this have their own flag reset to false. By setting the edge-detection flags in this manner, the next image will only be searched for edges in those image patches located around where edge pixels were detected in the current image. While this is sufficient for detecting edge pixels belonging to edges that were also present and detected in the previous image, this flagging scheme may fail to detect pixels belonging to new edges unique to the latest depth image (or edges that were present but not detected previously). To address this, with each new depth image a certain number edge-detection flags are randomly selected and set to true to facilitate the detection of new edges. The number of flags randomly selected in this manner (R) is determined by Equation 2.9 below.

$$R = Max(1, Round(N \times M \times rand\_search)) \tag{2.9}$$

$N \times M$ is the number of image patches and $rand\_search \in [0, 1]$ is a specified value relating to what percent of each depth image we wish to be randomly selected and searched for edges. Thus atleast one image patch is always randomly searched, increasing up to all image patches (and thus the entire image) being searched as $rand\_search$ approaches 1. In this scheme the detection of a new edge may be delayed a number of frames until the flag associated with the image patch it resides in is randomly selected and set to true. This potential detection delay is one of the trade-offs for the reduced computation this approach provides. Using higher values of $rand\_search \in [0, 1]$ will decrease the average number of frames of this detection delay but increase the average computational cost of the whole occluding edge detection process. The value of $rand\_search = 0.05$ was found to provide sufficiently small detection delay and is used as the default value adopted throughout the rest of this document. Algorithm 2.2 outlines this entire edge-detection process taking a set of image patches P, edge-detection flags F and number of patches to randomly search R as inputs and returning a list of edge pixel locations E and updated boolean flags F. This algorithm can also take an additional parameter K determining which rows and columns of each image patch should be skipped over instead of being searched for edge pixels. For greater clarity we will refer to this parameter as $rowcol\_skip$ from now on. The default value $rowcol\_skip = 1$ results in no rows/columns being skipped, with $rowcol\_skip = 2$ only every other row/column is searched, $rowcol\_skip = 3$ only every 3^{rd} row/column is searched and so on. This parameter can be viewed as allowing downscaled edge detection search to be conducted,

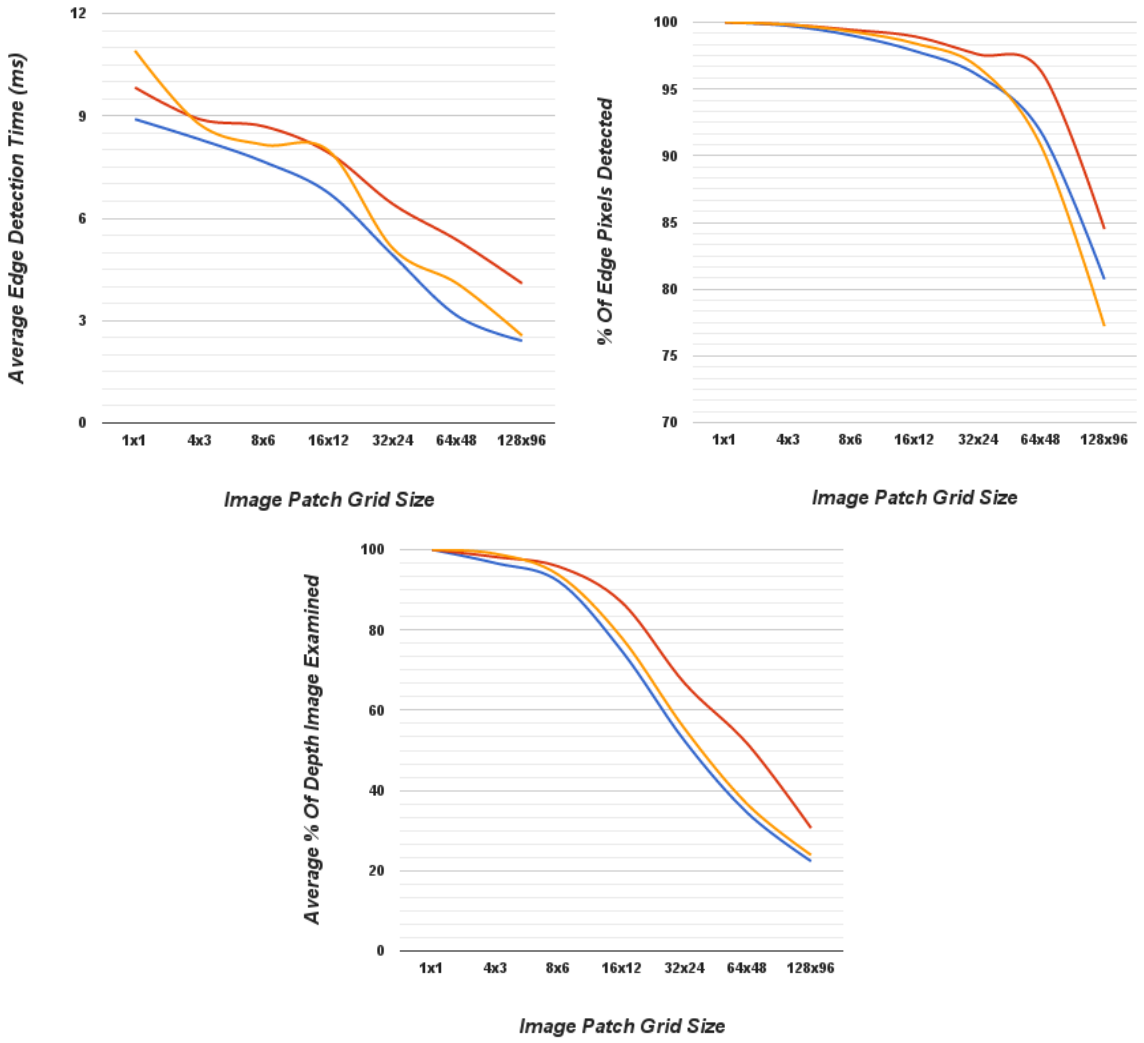


Figure 2.6: *Plots showing how various performance metrics of the occluding edge detection vary with the number of image patches the original depth image is divided up into (grid size). RGB-D sequences used, FR1 desk (Blue), FR1 plant (Red) and FR1 room (Yellow).*

resulting in fewer edge pixels being returned (and thus sparser edge point clouds being created from these pixels). Examples of this approach illustrating the image patches grid and corresponding edge-detection flag values are shown in Figure 2.5. Detailed results of running this proposed occluding edge detection on a number of RGB-D sequences are given in the following section.

2.4.3 Results

This section presents a sample of results from various experiments conducted to evaluate the performance of the edge-detection proposed in Section 2.4. We use the publicly

Table 2.1: *Depth edge detection results for Freiburg RGB-D sequences, showing how computation time and detection accuracy vary with the number of image patches the original depth image is split into.*

Sequence : FR1 desk			
Image Patches Grid Size	**Occluding Edges**	**Pixel %**	**avg % image searched**
Whole Image	8.9±2.25ms	100	100
4x3	8.31±2.57ms	99.7	96.65
8x6	7.65±2.25ms	99.1	92.13
16x12	6.74±2.26ms	97.8	74.85
32x24	4.91±1.95ms	96.0	52.29
64x48	3.13±1.32ms	91.8	34.28
128x96	2.41±1.04ms	80.7	22.38

Sequence : FR1 plant			
Image Patches Grid Size	**Occluding Edges**	**Pixel %**	**avg % image searched**
Whole Image	9.83±2.50ms	100	100
4x3	8.9±2.07ms	99.8	98.18
8x6	8.69±2.31ms	99.5	95.80
16x12	7.92±2.02ms	98.9	86.92
32x24	6.41±1.92ms	97.6	66.67
64x48	5.35±1.79ms	96.3	51.49
128x96	4.09±1.44 ms	84.5	30.62

Sequence : FR1 room			
Image Patches Grid Size	**Occluding Edges**	**Pixel %**	**avg % image searched**
Whole Image	10.91±1.97ms	100	100
4x3	8.75±2.03ms	99.8	98.98
8x6	8.15±2.11ms	99.2	93.74
16x12	7.98±2.22ms	98.4	78.11
32x24	5.11±1.68ms	96.6	55.34
64x48	4.08±1.7ms	90.7	36.43
128x96	2.56±1.03ms	77.2	23.92

Table 2.2: *Comparison of Occluding edge detection methods*

Sequence	Choi et al[12]	Proposed (32×24)	Proposed (16×12)
FR1 desk	24.06 ± 1.22 ms	4.91 ± 1.95 ms	6.74 ± 2.26 ms
FR1 desk2	24.71 ± 0.79 ms	4.88 ± 1.91 ms	6.64 ± 1.93 ms
FR1 room	23.86 ± 1.47 ms	5.11 ± 1.68 ms	7.98 ± 2.22 ms
FR1 plant	24.61 ± 1.71 ms	6.41 ± 1.92 ms	7.92 ± 2.02 ms
FR1 rpy	23.89 ± 0.99 ms	5.35 ± 1.83 ms	6.18 ± 1.94 ms
FR1 xyz	24.45 ± 1.36 ms	4.16 ± 1.30 ms	5.19 ± 1.03 ms

available Freiburg RGB-D datasets [97] to conduct this evaluation. These consist of a number of 640×480 RGB-D video sequences of various environments along with sensor ground truth trajectories obtained from motion capture. The results were obtained from a 2.60GHz Intel Core i5-3230M (2013), 4GB RAM laptop running Ubuntu 14.10.

We evaluated the image patch based occluding depth edge-detection described in Section 2.4.2 on a number of datasets and with various image patch grid sizes. In each case the total number of edge pixels detected across the entire sequence was recorded. This total was then compared to the total number of edge pixels obtained when using brute force occluding edge detection, which searches the entirety of each depth image by examining each pixel and its eight surrounding neighbours. From this comparison, the percentage of total edge pixels detected (compared to brute force) was calculated. The percentage of each depth image that was examined for edge pixels was also recorded in each case.

A detailed sample of these results is given in Tables 2.1, showing the average computation time of the edge detection process, percentage of total edge pixels detected and what percentage of each depth image was examined on average. These results are illustrated in the plots of Figure 2.6. As expected, the average computation time and percentage of the depth image examined generally decreases as the image patch grid size increases. This is due to the smaller image patches flagged for edge-detection more tightly fitting about the edges detected in the previous image. This tighter fitting then results in edge-detection being performed upon a smaller percentage of the entire depth image. The downside of larger grid sizes however, is an increased likelihood of edges failing to be re-detected from one image to the next due to changes in their locations between images (and thus moving into image patches not flagged for edge detection). This issue occurs most often when the sensor is undergoing a rapid change in orientation.

It can be seen however that using an image patch grid size such as 32x24 can provide up to a 50% saving in computation time while still detecting over 95% of the edge pixel

in the sequence. The average computation time when using such a grid size is also well within the 33ms required to process a standard 640×480 30Hz RGB-D video stream in real-time and leaves plenty of frame time remaining in which other processes can take place.

Table 2.2 shows a comparison between the computation times of our proposed occluding edge detection and that introduced in [12] across a number of Freiburg sequences. The use of prior knowledge to selectively only search certain areas of the depth image gives the proposed method a far lower computation time compared to the whole image search method conducted by [12]. All results in this section were obtained using the parameter value $rowcol\_skip = 1$.

2.5 RGB Edge Detection

Our RGB edge detection follows a similar design approach to the previously described depth edge detection, searching the image for sudden jumps in intensity value. It did not however, exploit any temporal similarity between RGB images, simply searching the entire image in around 9ms. It should be noted however, that due to such visual intensity edges being far more ambiguous in nature than occluding depth edges, this approach is not well suited and performs poorly compared to other methods such as the well known canny edge detector [19], however, for our purposes it was sufficient.

It is important to note that we reject all RGB edge pixels located in close proximity to occluding depth edges in the corresponding depth image. The back-projected points from such pixels can often be highly unreliable due to poor alignment and synchronisation between the depth and RGB data provided by the RGB-D sensor. Additionally, it is highly likely that such RGB edge pixels are the result of an occluding foreground object, in which case an occluding depth edge will already be present in the same location. The occluding depth edge detection flags described in Section 2.4.2 are used to enable fast rejection of such unwanted RGB edge pixels.

2.6 Edge Based ICP RGB-D Frame Registration

Section 2.4 and 2.5 described the process of extracting depth and RGB edge pixels from RGB-D frames. The depth component of an RGB-D frame can then be used to back-project such edge pixels into 3D space, generating a semi-dense point clouds referred

to hereon as edge clouds. The edge pixels of any image typically only make-up a tiny fraction of the image's total pixels. Similarly such edge clouds contain far fewer points than their raw RGB-D point cloud counterpart, typically being around two orders of magnitude smaller.

The well known Iterative Closest Point algorithm (ICP) can be used to register two point clouds, resulting in an estimate of the relative transformation between the sensor poses at which they were observed. The point clouds are referred to as the "source" and "target" clouds denoted here by S and T. ICP operates iteratively by first determining pairs of points between the two clouds. Let P denote the set of all determined pairs, with each point pair of the form $p = (\mathbf{p}_S \in T, \mathbf{p}_T \in S) \in P$. ICP then determines the transformation to minimize the sum of the distances between these paired points as given by $\sum_{p \in P} |\mathbf{p}_S - \mathbf{p}_T|$.

This transformation is applied to the source cloud and then the entire process is repeated until convergence or a maximum number of iterations have been performed. The final estimate of the transformation to align the two original clouds is simply the combination of all transformation applied to the source cloud. The proposed edge based SLAM system also uses such ICP registration for tracking the pose of the sensor and detecting loop closures between keyframes. However instead of using the raw RGB-D point clouds, ICP registration is performed using the semi-dense edge clouds previously mentioned.

Since the RGB edge pixels located within close proximity to occluding edge pixels are rejected during the RGB edge detection process as described in Section 2.5, the depth and RGB edge clouds generated from an RGB-D frame can be viewed as representing two types of non overlapping information. Utilizing both types of edge cloud simultaneously for registration requires a slight variation on the standard ICP algorithm. Instead of a single point cloud, the algorithm's "target" $T = (T_D, T_I)$ now consists of a pair of point clouds, a depth edge point cloud T_D and a RGB edge cloud T_I. In a similar manner, the "source" now also consists of a pair of edge clouds $S = (S_D, S_I)$. Each ICP iteration then involves two point pairing processes, one between the source and target depth edge clouds S_D, T_D, and similarly one between the sources and target RGB edge clouds S_I, T_I. All resulting pairs from both of these point pairing process are then used to update the estimated registration transformation, the remainder of the algorithm is by and large identical to standard ICP. This ICP process can then be used to register the edge point clouds generated from the latest sensor data $S = \{D, I\}$, with the edge point clouds of the current tracking keyframe K_T, i.e. $T = \{D_T, I_T\}$.

2.6.1 Advantages

There are several major advantages that such edge based ICP registration exhibits over full RGB-D point cloud ICP registration, especially with regards to real-time sensor tracking in which registration must be completed within a limited time period before the next RGB-D frame is acquired.

2.6.1.1 Convergence Speed and Robustness

ICP registration requires an initial guess at the relative transformation between the clouds and convergence to the correct transformation can be highly sensitive on this initial guess. The further this guess is from the actual correct registration transformation, the more likely it is that the ICP will produce an incorrect transformation due to it becoming trapped and converging to an incorrect local minima. However due to edge clouds consisting of a selective subset of the raw RGB-D cloud, they will typically feature far fewer incorrect ICP local minima. Additionally due to their semi-dense nature, the local minima of such edge clouds are typically far more sharply defined than those of raw RGB-D clouds. This means that in the majority of scenarios, given sufficient edges were present in the RGB-D frames being registered, edge point based ICP exhibits both a faster convergence rate (requiring fewer ICP iterations) and fewer convergences to incorrect local minima in comparison to ICP using entire RGB-D point clouds.

2.6.1.2 Reduced Computational Cost Per Iteration

Additionally, edge cloud based ICP is significantly cheaper computationally compared to ICP registration using raw RGB-D point clouds. This is due to the dominating computational factor in each ICP iteration being that of the nearest neighbour searches conducted to pair points between the source and target clouds. The computational cost of each of these searches (and the number of searches required) increases with the size of the point cloud being registered. Since edge clouds are typically multiple order of magnitude, smaller in size compared to their associated raw RGB-D cloud counter part, so too is their computational cost per ICP iteration.

2.6.2 Disadvantages

There are of course scenarios/environments in which there may be insufficient edges present in the RGB-D frames to conduct edge point based ICP registration, and in which

full RGB-D cloud registration would thus produce more accurate results. However in man made indoor environments such as those we are interested in operating in such a lack of present edges is unlikely, as the vast majority of man-made objects feature well defined edges both in their geometry and surface texture.

2.6.3 Evaluation

This subsection now presents a sample of results and evaluation of the proposed edge cloud based ICP registration, conducted using the indoor RGB-D sequences provided by the Freiburg RGB-D dataset [96] featuring a number of indoor environments.

We are primarily interested in evaluating this registration for use in tracking the pose of the RGB-D sensor, which typically involves registering two highly similar subsequent frames of RGB-D sensor data. Our evaluations thus consisted of attempting to register every frame from a given Freiburg sequence with a randomly transformed copy of itself. A different random transformation was used for each RGB-D frame. Naturally these random transformations are also the transformations that would perfectly align the original RGB-D frames with the transformed copy and is thus the transformation we wish the ICP registration to produce. The magnitude of these random transformations was constrained to reflect the typically limited degree of sensor movement that can occur in the 33ms between RGB-D frames for a $30hz$ sensor. Specifically each random transformation consisted of a translation of random direction and length between 0-25cm, and a random rotation up to 15 degrees (0.26 radians) about a randomly selected axis. The sensor undergoing transformations significantly greater in magnitude than these limitations in the space of 33ms would most likely cause extreme motion blur in the RGB-D frame rendering it of little use for registration. Additionally, during typical operation it is highly unlikely that the sensor will experience a transformation between two subsequent RGB-D frames exceeding these limits.

A Sample of the edge cloud registration evaluations obtained from the Freiburg "FR1 room" sequence is presented in Figures 2.7 and 2.8. The registration process was evaluated on the sequence multiple times, using different numbers of ICP iterations in order to observe how this parameter effects the accuracy and computation time. Each one of these evaluations made use of all 1467 frames in the FR1 room sequence, attempting registration between each frame and a randomly transformed duplicate. The translational and rotational errors from each attempted registration were recorded in order to generate the histograms as shown in Figure 2.7.

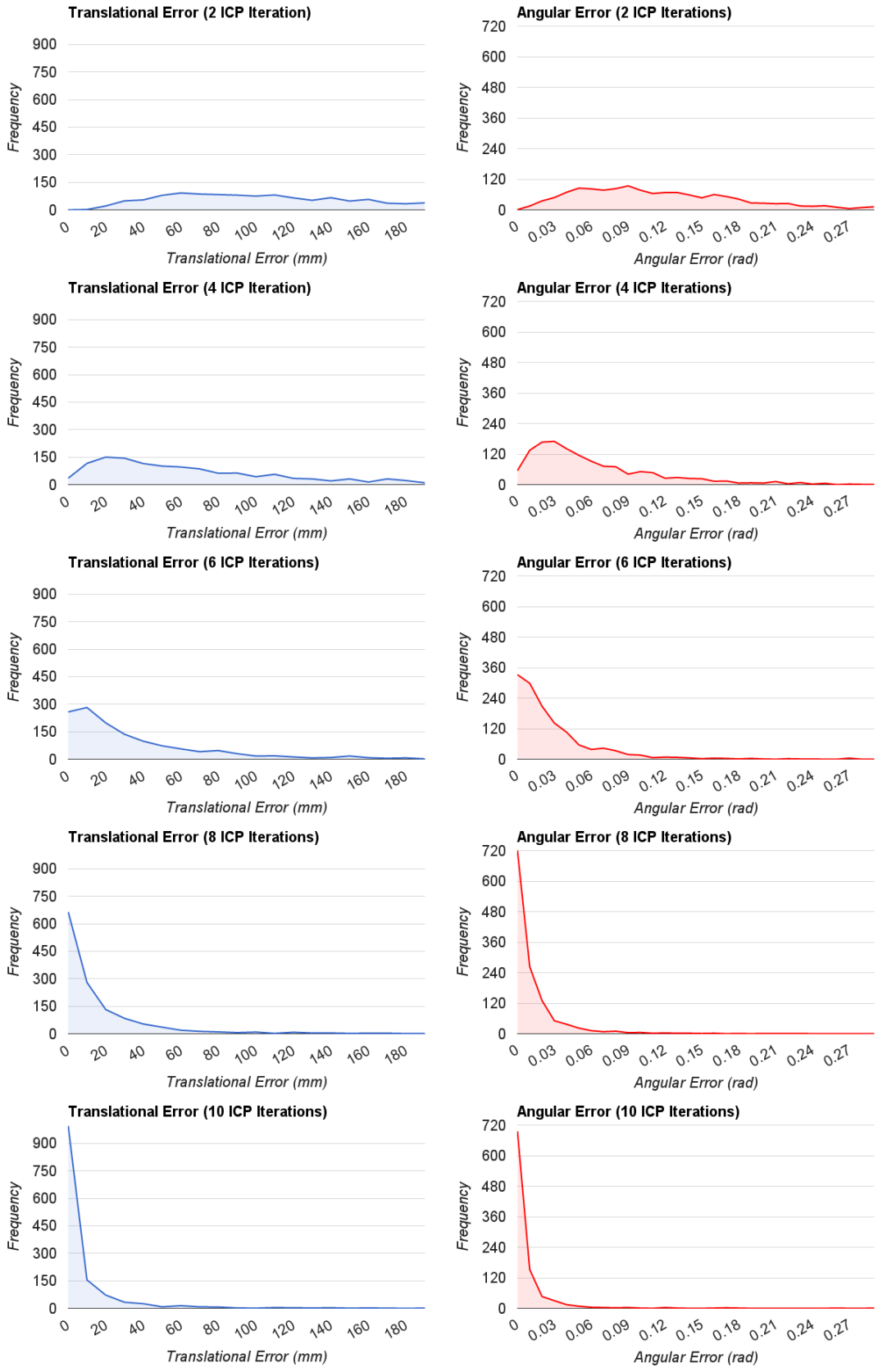


Figure 2.7: *Histograms illustrating how the errors associated with edge based ICP registration change with the number of ICP iterations used (RCS = 1).*

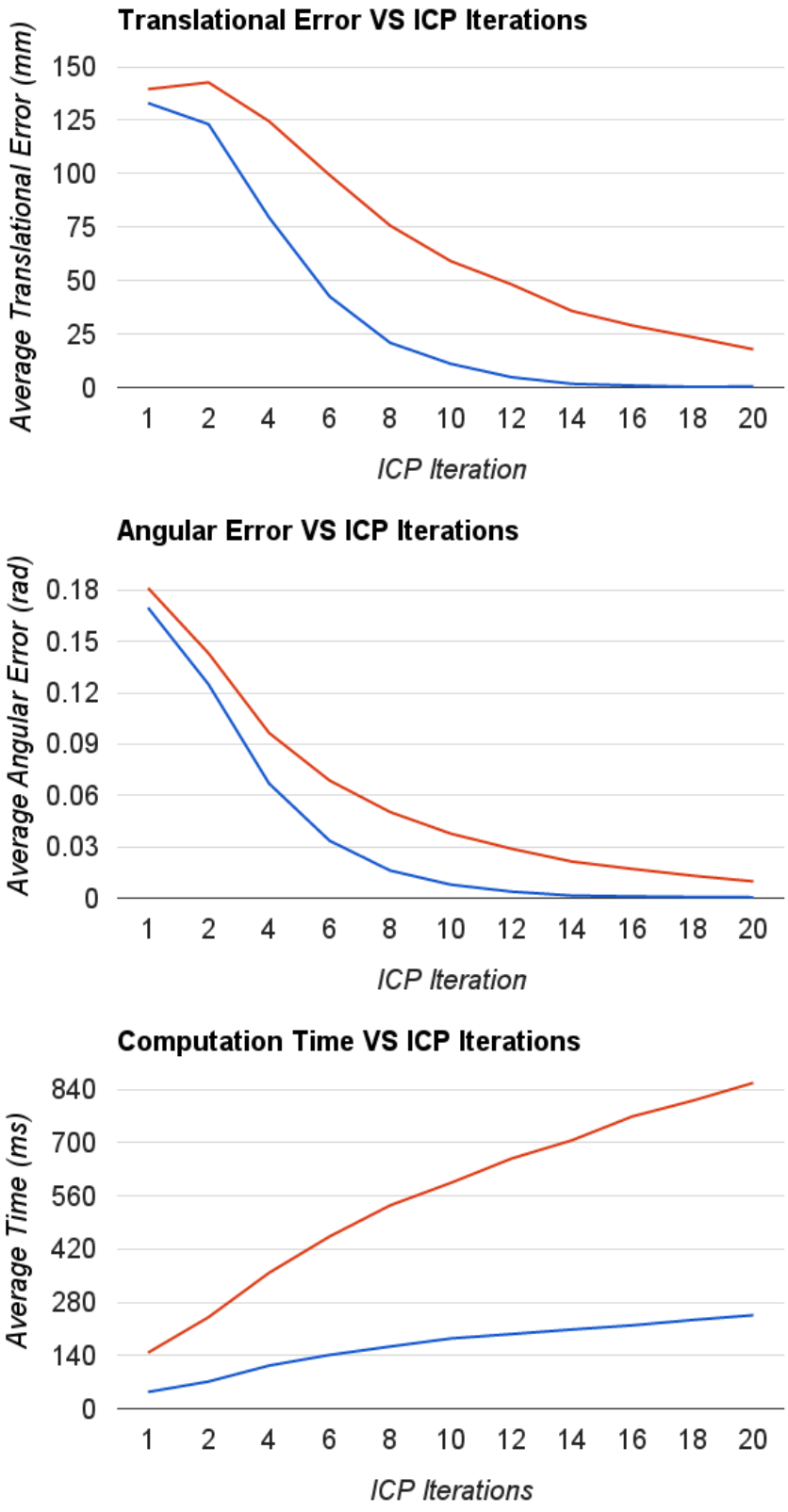


Figure 2.8: *Accuracy and computation time of edge based ICP registration (Blue, RCS=1) and raw point cloud registration(Red, Uniform downsampling x5).*

As is to be expected a decrease in both translational and rotational error occurs as the number of ICP iterations used for registration increases. This can seen in the shift of the distributions of Figure 2.7, and is further illustrated in the graphs of Figure 2.8 where a sharp decrease in registration error can be seen with increasing ICP iterations, plateauing around 10-14 iterations. The computation time of the registration process is also seen to increase linearly with the number of ICP iterations as expected.

We also conducted the same evaluations on raw RGB-D point cloud based ICP registration in order to compare to our edge cloud based approach. As seen in Figure 2.8 raw RGB-D cloud registration demonstrated significantly poorer accuracy for any given number of ICP iterations. The rate at which registration accuracy improves with increasing the number ICP iterations is also far slower than that of edge cloud based registration, and levels off at far higher values of translation and rotational error by comparison. Additionally edge cloud registration demonstrates a vast reduction in the amount of computation time required in comparison to raw point cloud registration.

The computation times of these edge cloud registrations however is still well above the required 33ms limit required to achieve real-time registration needed to conduct SLAM. In order to reduce the computation required each ICP iteration the Row Column Skip parameter (RCS) used in edge detection as described in Section 2.4 can be increased (i.e. $RCS > 1$). This has the effect of reducing the number of edge pixels produced by the edge detection by skipping over certain rows and columns of the RGB-D image and hence, also in turn reducing the size of the edge cloud being used for registration. In general this RCS parameter can be viewed as controlling the degree of uniform downsampling applied to the edge clouds ($RCS = 5$ will result in clouds five times smaller than $RCS = 1$ etc). We hence conducted the same edge cloud registration evaluation multiple times using various values for the RCS edge detection parameter.

Figure 2.9 illustrates a sample of the results showing the effects of this RCS parameter on the accuracy and computation time of the registration process, again obtained using all frames of the FR1 room sequence. It is immediately apparent that increasing the RCS value results in a dramatic decrease in computation time. Specifically for any value of $RCS > 1$ the computation time required is approximately $\frac{t}{RCS}$ where t is the time required by the edge cloud registration taking $RCS = 1$. This is as to be expected since the size of the edge clouds used for registration is inversely proportional to the RCS value. For registration involving 20 ICP iterations it can be seen that using any RCS value between 1 and 10 results in near identical registration accuracy in both rotational and translational error. When decreasing the number of ICP iterations to 10 we see slight increases in both translational and rotational registration error. This is

to be expected since in many sceneries this will be an insufficient number of iterations for the ICP registration to fully converge. Additionally it can be seen that the smaller edge clouds (associated with higher RCS values) demonstrate smaller translational error due to the fact that in general the rate of ICP convergence increases with smaller cloud size. This is illustrated further in Figure 2.10, showing how the errors in the estimated transformation and computation time of registration for various RCS values varies with the number of ICP iterations. For ICP iterations lower than 14, registration using RCS values of 5 and 10 is seen to provide slightly improved accuracy compared to when using an RCS value of 1, due to the faster rate of ICP convergence for smaller edge clouds.

From these evaluations it is clear that the RCS parameter can be increased to any value between 5-10 in order to save a significant amount of computation (both in the ICP registration and edge detection processes) while sacrificing little in terms of registration accuracy. Thus using such higher RCS values, accurate RGB-D frame registration can be achieved well within the required 33ms limit for real-time registration.

In order to draw a meaningful comparison between this real time edge cloud registration and the standard RGB-D point cloud based ICP registration, the latter must also be made to conduct real-time registration. This is achieved by simply uniformly downsampling the raw RGB-D point clouds in order to decreases the computation cost of the ICP registration as required. The same evaluation processes were then conducted for this down-sampled RGB-D point cloud ICP registration and the results compared to the real-time edge cloud registration using higher RCS values.

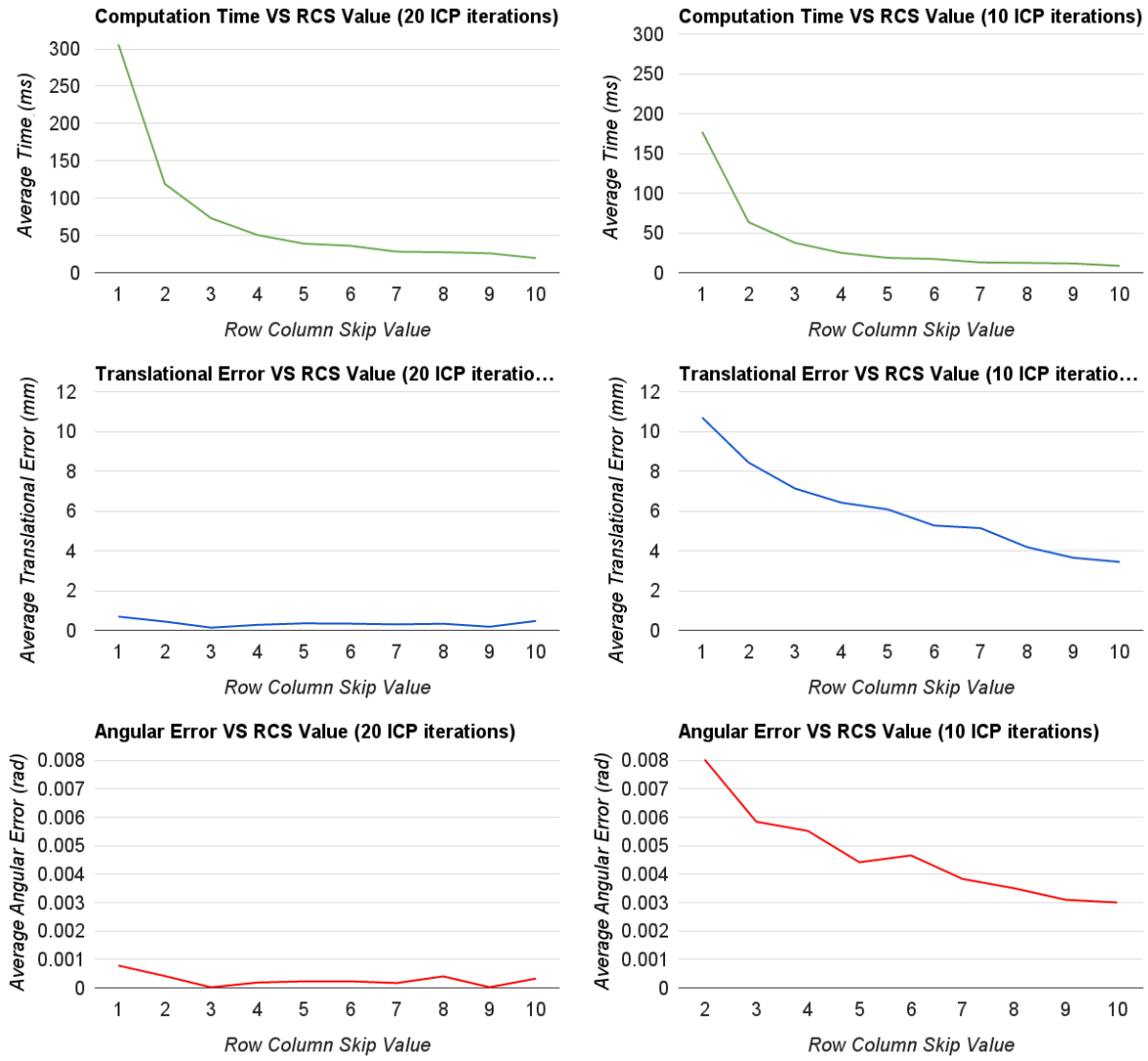


Figure 2.9: *Graphs illustrating the effect of the Row Column Skip (RCS) parameter on the accuracy and computation time of edge point based ICP registration. Increasing the value of the RCS parameter results in sparser edge point clouds being used for registration, and hence dramatically decreases computation time.*

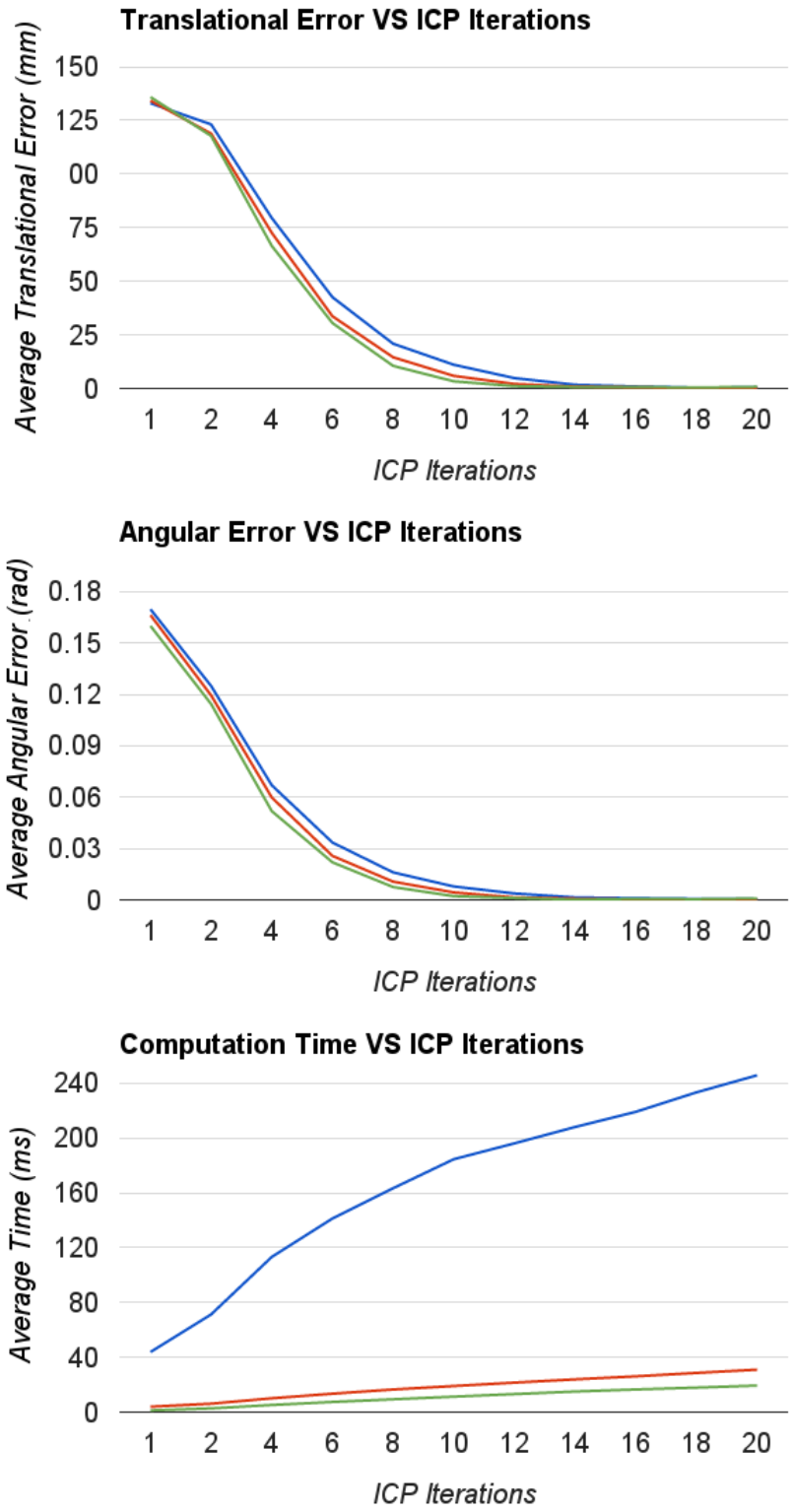


Figure 2.10: *Graphs comparing how the accuracy and computation time of edge point based registration varies with the number of ICP iterations, for various RCS values. Blue (RCS = 1), Red (5), Green(10).*

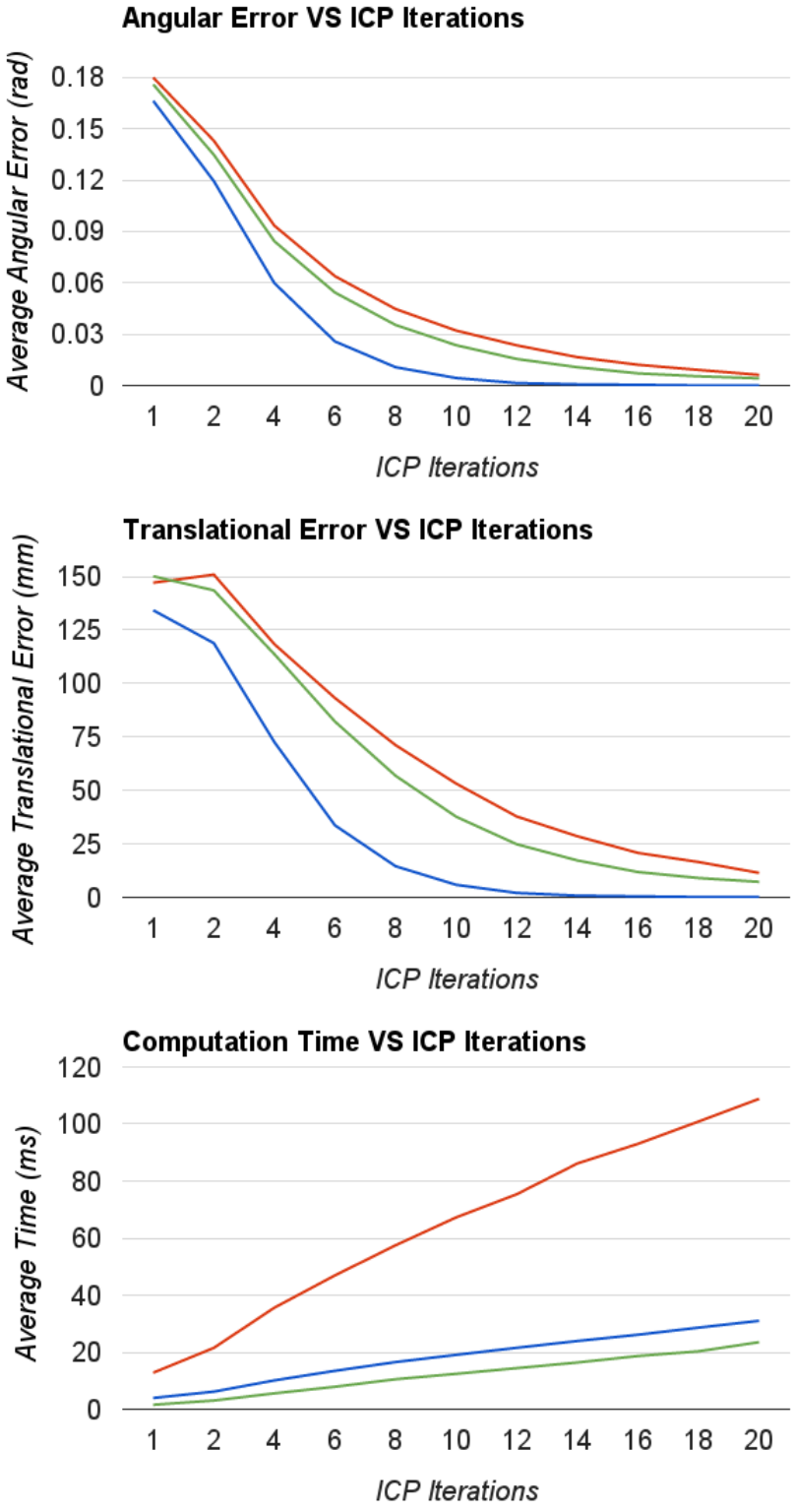


Figure 2.11: *Plots comparing the performance of real-time edge cloud registration (Blue, $RCS = 5$ and uniform down-sampled RGB-D point cloud registration ($\times 10$ Red, $\times 20$ Green) using the Freiburg FR1 room sequence.*

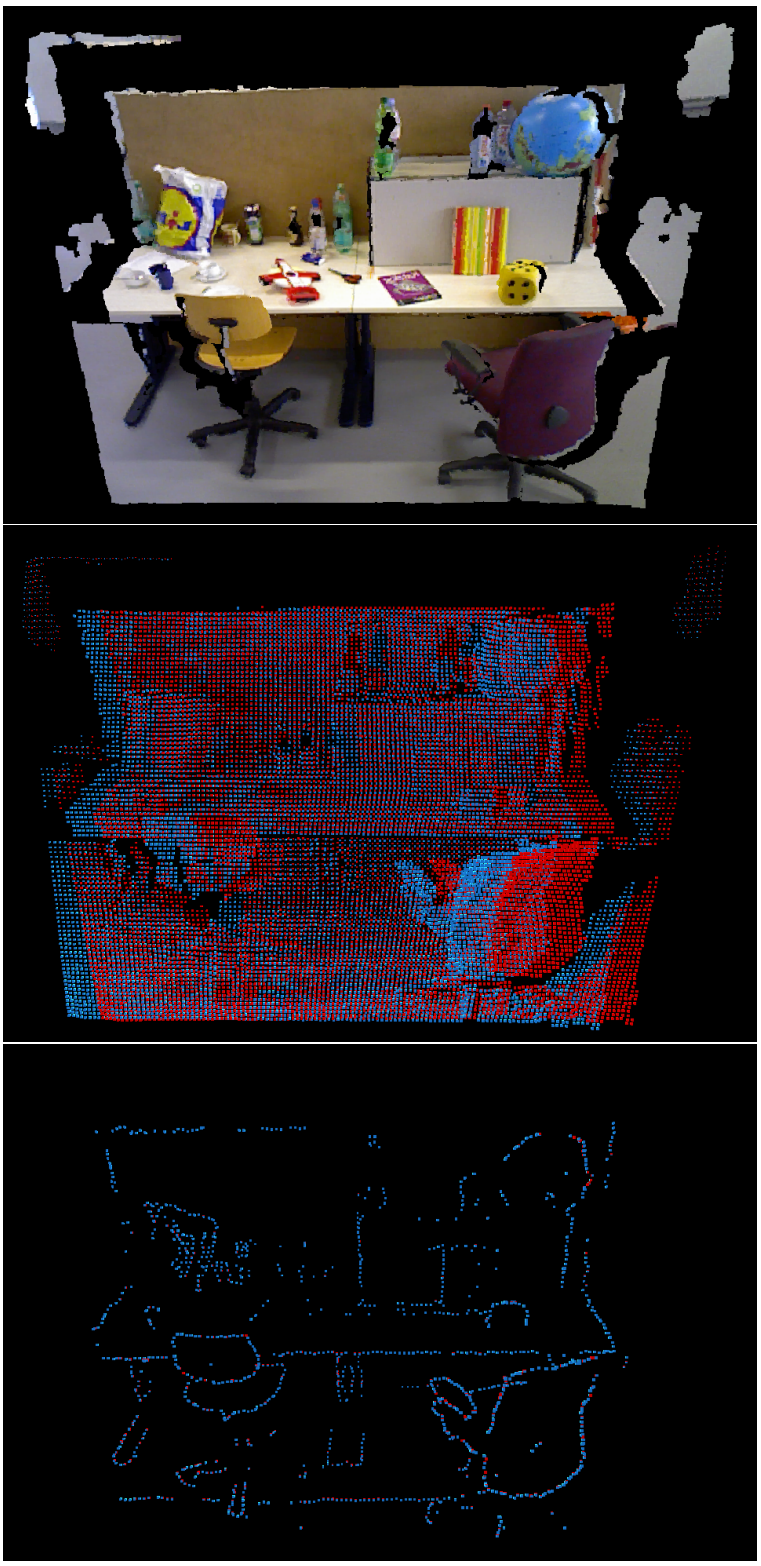


Figure 2.12: *Comparison of ICP registration quality when using raw RGB-D clouds (middle) and edge clouds (bottom). When using raw RGB-D clouds, there is obvious misalignment between the two clouds (shown in red and blue) compared to the edge based registration.*

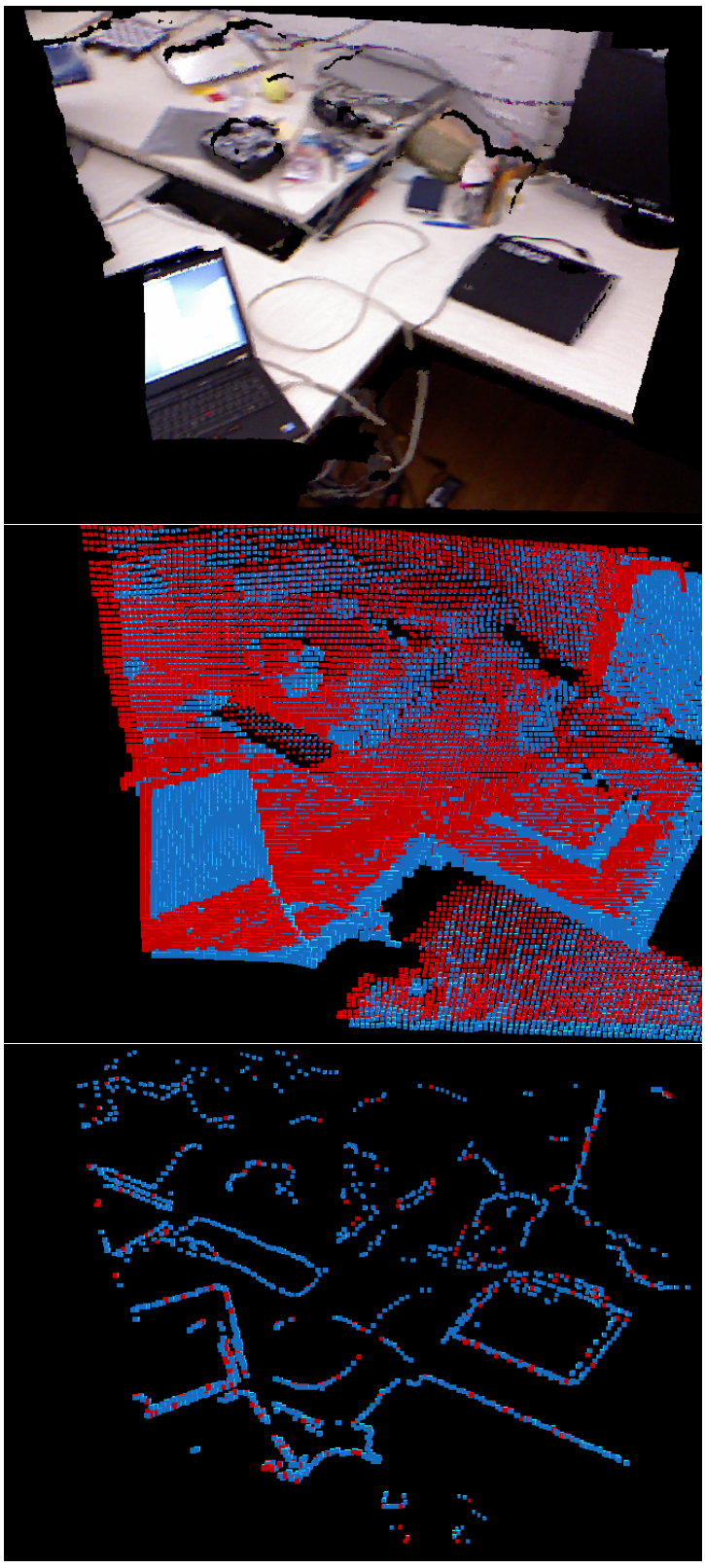


Figure 2.13: *Further comparisons of ICP registration quality when using raw RGB-D clouds (middle) and edge clouds (bottom). Again raw RGB-D cloud registration shows obvious misalignment between the two clouds (shown in red and blue) compared edge based registration.*

Figure 2.11 illustrates how the accuracy and computation time of both edge cloud registration ($RCS = 5$) and raw RGB-D cloud registration varies with the number of ICP iterations used, from 1 to 20. In every instance it is clear that edge cloud registration is able to provide superior accuracy in terms of both translational and rotational error. By comparison the RGB-D cloud registration using 10x uniform downsampling demonstrates poor registration accuracy and requires significantly more computation per ICP iteration making it unfit for real-time registration. The RGB-D cloud registration using 20x uniform downsampling achieves a slightly lower computational cost than the edge cloud registration, however its registration accuracy remains poor by comparison. This is largely due to the selective nature of the edge point clouds, which result in there being distinct local minima for the ICP registration of such clouds, which in turn results in a faster rate of ICP convergence towards such minima. By comparison the local minima of raw RGB-D cloud based ICP are nowhere near as distinct, as a result, requiring far more ICP iterations for the estimated transformation to convergence to such minima. This is illustrated in Figures 2.12 and 2.13 showing results from both edge cloud and RGB-D cloud based ICP (for the same registration problem). Clearly edge cloud ICP has converged to the correct transformation, where as raw RGB-D cloud ICP has only succeeded in aligning the major surfaces of the two clouds. Once these surfaces have been aligned it would require many further ICP iterations to "slide" the estimated transformation parallel to these surfaces and converge to a local minima.

Thus in summary, in the tested sequences edge point based ICP significantly outperforms raw point cloud ICP for any given number of ICP iterations both in terms of registration accuracy and computation time. Given the limited time window of 33ms between 30Hz RGB-D frames in which to conduct ICP registration (and leave sufficient time remaining for other processes) edge based registration is clearly the preferable registration approach so long as the environments being operated in feature sufficient geometric and or surface texture edges.

2.6.4 Registration Strength Evaluation

This edge based ICP registration is utilized by the SLAM system for a number of tasks such as sensor tracking. For such tasks it is necessary to have some measure of the strength and reliability of the resulting registration between the source and target edge point clouds.

Note how if the source and target clouds are identical and the ICP has produced the correct alignment transformation, then naturally every point in the source cloud will

end up being located in close proximity to some other point in the target cloud. On the other hand, if the ICP either failed to converge in the given number of iterations or converges to an incorrect transformation, then a significant proportion of the source cloud points will not be located in close proximity to any target cloud point. We refer to such source cloud points which end up in close proximity to some target cloud point as being "matched". Thus in such a scenario with identical source and target clouds, the strength of the ICP registration can be evaluated by determining the proportion of matched source cloud points.

In practice the source and target clouds utilized for sensor tracking will not be identical, and thus the correct alignment between them will not result in every source cloud point being matched. However despite this, the proportion of matched source cloud points still provides a useful metric for evaluating the strength of a particular ICP registration.

Let us denote the source and target point clouds by S and T respectively, then the proportion of matched source cloud points is given below in Equation 2.10.

$$\frac{1}{|S|}\sum_{\mathbf{p}\in S} f(\mathbf{p}) = \begin{cases} 1, & \text{if } |\mathbf{p} - NearestNeighbour(\mathbf{p}, T)| < \delta \\ 0, & \text{if } |\mathbf{p} - NearestNeighbour(\mathbf{p}, T)| > \delta \end{cases} \tag{2.10}$$

The value of $\delta \in \mathbb{R}$ in this equation gives the threshold distance used to determine if a point is considered successfully matched or not. Ideally a very small value of δ would be used, however, in practise, structured light RGB-D sensors which have considerable noise and spacing of readable depths [57]. These properties must therefore be accounted for in deciding the value of δ such that they are not a dominant factor. For example if the value of δ used is far smaller than the sensor's inherent noise then it will become a significant factor in determining if each source cloud point is considered matched or not, clearly an unwanted scenario.

2.7 Map Construction and Sensor Tracking

The SLAM system performs sensor tracking by using each new frame of RGB-D data F to update the estimated sensor pose. Depth and RGB edges are extracted from the sensor's latest data and the associated depth and RGB edge point clouds D and I are generated. These edge point clouds are used to estimate the pose of the sensor relative to the pose of the tracking keyframe $K_T \in K$ by making use of the edge based ICP process

described in the previous section. IPC registration is performed between the latest edge point clouds of the sensor D, I, and the edge point clouds D_T, I_T associated with the tracking keyframe K_T. The resulting transformation from this ICP registration then gives an estimate of the sensors pose $\mathbf{X}_T$ relative to the pose of the tracking keyframe P_T. The estimate of the sensors pose in the world reference frame $\mathbf{X}$ is then generated by simply transforming this relative pose estimate $\mathbf{X}_T$ by P_T.

Naturally it is necessary to switch between different tracking keyframes as the sensor moves through the environment. If the sensors estimated pose $\mathbf{X}$ is extremely different from that of a keyframe's pose $\mathbf{P}_i$, then it is unlikely that any common information exists between the RGB-D frames F and F_i that could be used to perform registration. Also in general, the further the estimated sensor pose $\mathbf{X}$ is from a specific keyframe pose $\mathbf{P}_i$, the less common information will exist between their associated RGB-D frames F and F_i. Intuitively the less common information between such RGB-D frames the more unlikely it is that the ICP registration will produce the correct alignment transformation, simply as there is less information between which correct registration can occur and more information that can contribute to incorrect registration occurring. Thus in order to attempt to ensure that sufficient common information always exists between the RGB-D frames F and F_T, the tracking keyframe K_T is always taken to simply be the keyframe $K_i \in K$ whose estimated pose $\mathbf{P}_i$ is closest to that of the sensor's estimated pose $\mathbf{X}$.

However, this alone is insufficient to ensure sensor tracking is maintained as the sensor may simply be located so far from any keyframe pose K_i, that no keyframes RGB-D data F_i has sufficient information in common with F to perform registration. In order to avoid encountering such a scenario, new keyframes are added to the map whenever either when the sensor's estimated pose $\mathbf{X}$ is significantly different from that of every keyframe of the map, or when registration with the current tracking keyframe is determined to be insufficiently strong.

Any such new keyframe K_i is initialized at current estimated sensor pose ($\mathbf{P}_i = \mathbf{X}$) and with the sensor's latest RGB-D data and associated edge features.

2.8 Map Optimization

The RGB-D registration process described in Section 2.6 is used for estimating the sensor's pose $\mathbf{X}_T$ relative to the current tracking keyframe K_T. However, this registration process is never perfectly exact and factors such as sensor noise and incomplete ICP convergence will negatively effect the registration process, resulting in errors being in-

troduced into the estimated sensor pose $\mathbf{X}_T$. $\mathbf{X}$ the estimated pose of the sensor in the world reference frame is formed by combining this relative pose estimate $\mathbf{X}_T$ with the estimated pose of the tracking keyframe $\mathbf{P}_T$. Thus the sensor pose $\mathbf{X}$ inherits errors from two different sources, the first being errors from the RGB-D frame registration process used to estimate the relative sensor pose $\mathbf{X}_T$ and second being errors in the estimated pose of the current tracking keyframe $\mathbf{P}_T$. The global pose of the initial keyframe $\mathbf{P}_0$ is taken to be fixed at the identity pose of the world reference frame and hence will contain no error by definition. The initial pose of subsequent keyframes however, are determined by the estimated sensor pose and hence will also be prone to exactly the same sources of error. Let us denote the true pose of the sensor relative to $\mathbf{P}_T$ by $True(\mathbf{X}_T)$, and the error in the estimated pose $\mathbf{X}_T$ by $\triangle\mathbf{X}_T$. Similarly let $True(\mathbf{P_T})$ and $\triangle\mathbf{P}_T$ denote the true pose of the tracking keyframe and the error in its estimated pose $\mathbf{P}_T$. The initial pose of a new keyframe P_i determined by the current sensor pose $\mathbf{X}$ can then be written as shown below.

$$P_i = True(\mathbf{X}_T) + \triangle\mathbf{X}_T + True(\mathbf{P}_T) + \triangle\mathbf{P}_T \tag{2.11}$$

From the above it is obvious how the small errors from the RGB-D frame registration process can accumulate since each keyframe's initial pose is dependent upon that of another keyframe. This can lead to significant errors in keyframe poses P resulting in a globally inaccurate and potentially inconsistent map. This is well demonstrated by extensively exploring an environment with the sensor and then returning back to around the location of the initial keyframe $\mathbf{P}_0$. Despite actually being around the same location, the estimated pose of the sensor $\mathbf{X}_T$ will likely be significantly different from $\mathbf{P}_0$ due to the accumulation of registration errors in poses of the map keyframe P. However since the sensor is located close to the initial keyframe pose P_0, it maybe possible to recognize common areas of the environment present in both in the current RGB-D sensor data and the RGB-D data of the initial keyframe F_0, and use this to determine how to correct the sensor's estimated pose $\mathbf{X}$ and improve the accuracy of the keyframe pose P.

In general terms, if two frames of RGB-D sensor data both contain some common area of the environment, then it may be possible to identify this common information and use it to estimate the pose at which one of these RGB-D frames was obtained relative to the other. This process of recognizing when some part of the environment is present in two different sets of sensor data, is referred to as loop closure detection, and the estimated relative pose between them, based on this common data is know as a loop

closure constraint.

For our system in particular, we wish to find loop closures between the various keyframes K of the map inorder to improve the estimated poses of the map keyframe P. It should be noted that the poses of certain keyframes may have become highly inaccurate due to error accumulation, and thus may not provide reliable a priori information to aid in loop closure detection with certain other keyframes.

Let the pose $\mathbf{R}_{i,j} = (\mathbf{t}_{i,j}, \mathbf{q}_{i,j})$ denote a relative pose constraint between the keyframes K_i and K_j arising from a detected loop closure. Specifically $\mathbf{R}_{i,j}$ gives the pose at which $\mathbf{P}_j$ should be located relative to $\mathbf{P}_i$ (i.e. in the co-ordinate reference frame of K_i) in order for these poses to be consistent with the associated loop closure. When considered from the world reference frame this constraint is fulfilled if K_j is located at $\mathbf{P}_j = \mathbf{P}_i\mathbf{R}_{i,j}$, or alternatively if $\mathbf{P}_i = \mathbf{P}_j\mathbf{R}_{i,j}$.

2.8.1 Loop Closure Detection

To detect a loop closure between two keyframes is to find a strong registration between their associated features or sensor data. In our edge based system this would mean finding a transformation that would align the edge clouds of two different keyframes, specifically such that a large proportion of the two clouds are overlapping giving us high confidence that both edge clouds were generated from observations of the same edge features. ICP registration as described previously in 2.6 can be used to produce an estimate of the transformation to align two edge clouds given an initial guess. This same process is used to find loop closures between the edge clouds of different keyframes. Once the ICP has produced an estimated alignment transformation the alignment strength is evaluated as described in Subsection 2.6.4. If this alignment is deemed strong enough, the estimated alignment transformation between the pair of keyframes is compared against all other existing loop closure constraints to the keyframes to determine if the new constraint from the estimated alignment is an outlier by comparison. If the alignment passes these tests it is used to generate a new loop closure constraint between the two keyframes. In all cases an initial ICP registration is attempted using highly down-sampled versions of the edge clouds to quickly gauge whether a loop closure may exist before performing a full edge cloud ICP registration.

Whenever a new keyframe is added to the SLAM map, loop closure detection is attempted between it and the previous n keyframes of the map. The estimated poses of both the new keyframe and keyframe with which loop closure is being detected are

used to form the initial alignment transformation estimate used for the ICP registration process. In most cases there should not have been significant sensor drift between the acquisition of two such keyframes and thus this initial transformation estimate generated from the estimated keyframe poses should be reasonably accurate to the true alignment transformation.

In addition to this sequential form of loop closure, a randomized search is performed for loop closures between all other pairs of keyframes. For each pair of keyframes this process first examines the composition of each keyframe's edge clouds, only then performing ICP registration between keyframes whose edge clouds (for both depth and RGB edges) have a similar number of points and similar bounding sphere size. This prevents attempting to conduct ICP between significantly different edge clouds which are unlikely to be observations of the same edge features. The initial transformation for ICP is again based on the estimated relative pose between said keyframes, calculated from their estimated poses, a random transformation is then also applied to this based upon the indexes of each keyframe. The maximum magnitude of this random transformation (for both rotational and translational components) is taken to be proportional to the difference between the indices of the keyframes. The justification for this is that in general the amount of camera motion that has occurred between two keyframes K_i and K_j, will be proportional to the difference between their indexes $\mid i - j \mid$. Greater camera motion increases the potential error in the estimated relative pose between the keyframes K_i and K_j due to the accumulation of drift, and hence the magnitude of transformation needed to correct said drift. The application of the aforementioned random transformation seeks to correct for such drift (even if by random chance), and thus is of magnitude proportional to $\mid i - j \mid$.

2.8.2 Pose Graph Optimization

Let R denote the set of all detected loop closures $\mathbf{R}_{i,j} \in R$, each of which imposes a constraint on the relative pose between a pair of keyframes K_i, K_j. This set R can be visualized in the form of a (pose) graph, in which each graph node represents the estimated pose of a certain keyframe $\mathbf{P}_i \in P$, and each edge connecting a pair of nodes represents a specific constraint $\mathbf{R}_{i,j} \in R$ between their associated poses. Pose graph optimization (PGO) is the task of finding a set of poses P complying with the set of relative poses constraints R. Such a set of poses would then be consistent with the detected loop closures and in general will provide a far more accurate and consistent map compared to a different set of poses which contradict these detected loop closure

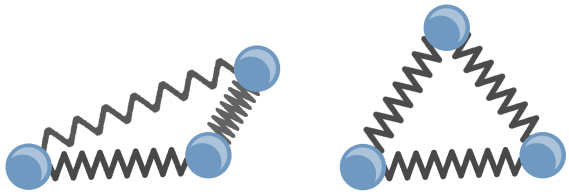


Figure 2.14: *A damped spring mass system starting from any initial configuration (Left) will come to rest in a local energy minima (Right)*

constraints.

2.8.2.1 Spring-Mass System Equivalence

Consider a constraint $\mathbf{R}_{i,j} \in R$ as representing a damped spring connecting two point masses with poses $\mathbf{P}_i$ and $\mathbf{P}_j$. This spring is defined such that it is at rest with zero stored energy, if the point mass poses $\mathbf{P}_i$, $\mathbf{P}_j$ comply with the relative pose constraint of $\mathbf{R}_{i,j}$. If instead the masses are closer or further apart than as dictated by $\mathbf{R}_{i,j}$, then the system is in a non zero energy state with the spring being either compressed or stretched. Similarly if the masses are such that their relative orientation does not match that of $\mathbf{R}_{i,j}$, then the spring is being bent or twisted, again storing potential energy. It is intuitive that given any two initial poses for $\mathbf{P}_i$ and $\mathbf{P}_j$, such a simple spring mass system would over time come to rest at its lowest energy configuration i.e. at some pair of poses $\mathbf{P}_i$,$\mathbf{P}_j$ which comply with the constraint $\mathbf{R}_{i,j}$. This same principle as illustrated in Figure 2.14 applies to any potential spring mass system though convergence to the global local energy minima is not guaranteed for many systems.

Extending this concept beyond a single pair of keyframes the entire pose graph can be considered as a spring mass system, with point masses located at the node poses P which are in turn connected by springs in accordance with the graph edges R. Given any set of initial poses P, such a spring mass system would come to rest at a lower energy configuration than it initially started at, resulting in a set of poses P more compliant with the constraints of R. It should be noted however that many potential spring mass set-ups are not guaranteed to come to rest at the global minimum in terms of lowest stored energy. Such a system may instead become trapped in an local minima, especially if the initial set of poses used for P deviate greatly from the required relative pose constraints.

2.8.2.2 Relaxation Based PGO

We developed a simple graph relaxation PGO algorithm, inspired by this spring mass analogy. As an overview, each iteration of this PGO algorithm makes adjustments to the poses of P (pose graph nodes) in accordance with the set of relative pose constraints R (the pose graph edges). These adjustments are such that the set of pose P becomes more compliant with constraints R, and thus over multiple iterations, the set P converges to minima in terms of disparity with the constraints of R. This is somewhat analogous to simulating a damped spring mass system, where the system will come to rest in an energy local minima as described at the beginning of this section and is a similar approach to that introduced by [45] but extended from 2-D positions to full 6DOF poses.

The pose adjustments made at each each iteration of this algorithm are determined by iterating through the constraints of R, for each $\mathbf{R}_{i,j} \in R$ making small alterations to both of its associated poses $\mathbf{P}_i$ and $\mathbf{P}_j$ such that they become more compliant with $\mathbf{R}_{i,j}$. These small per constraint pose alterations are split into translational and rotational components. The process of determining such pose alteration components for a certain constraint $\mathbf{R}_{i,j}$ are described in the remainder of this section.

2.8.2.3 Pose Alteration : Translational

The translational component of a constraint $\mathbf{R}_{i,j} = (\mathbf{t}_{i,j}, \mathbf{q}_{i,j})$ is fulfilled if the graph node pose $\mathbf{P}_j = (\mathbf{t}_j, \mathbf{q}_j)$ is located at position $\mathbf{t}_{i,j}$ in the co-ordinate reference frame of P_j as laid out in equation 2.12 below.

$$\mathbf{R}(\mathbf{q}_i)^{\mathbf{T}}(\mathbf{t}_j - \mathbf{t}_i) = \mathbf{t}_{i,j} \tag{2.12}$$

This can trivially be rearranged to express where the node $\mathbf{P}_j$ should be positioned relative to $\mathbf{P}_i$ (or vice versa) in the world reference frame. This then gives rise to the criteria given in equations 2.13 and 2.14, both of which would fulfil the translational component of the constraint $\mathbf{R}_{i,j}$.

$$\mathbf{t}_j = \mathbf{t}_i + \mathbf{R}(\mathbf{q}_i)\mathbf{t}_{i,j} \tag{2.13}$$

$$\mathbf{t}_i = \mathbf{t}_j - \mathbf{R}(\mathbf{q}_i)\mathbf{t}_{i,j} \tag{2.14}$$

In order to come closer to fulfilling the constraint $\mathbf{R}_{i,j}$ both the positions $\mathbf{t}_j$ and $\mathbf{t}_i$ can be linearly interpolated towards their associated criteria 2.13 and 2.14 by some smaller factor $\varepsilon \in \mathbb{R}$ as shown below.

$$\mathbf{t}_j = \mathbf{t}_j + \varepsilon(\mathbf{t}_i + \mathbf{R}(\mathbf{q}_i)\mathbf{t}_{i,j} - \mathbf{t}_j) \tag{2.15}$$

$$\mathbf{t}_i = \mathbf{t}_i + \varepsilon(\mathbf{t}_j - \mathbf{R}(\mathbf{q}_i)\mathbf{t}_{i,j} - \mathbf{t}_j) \tag{2.16}$$

2.8.2.4 Pose Alteration : Rotational

The rotational component of a constraint $\mathbf{R}_{i,j} = (\mathbf{t}_{i,j}, \mathbf{q}_{i,j})$ is fulfilled if the node $\mathbf{P}_j$ is at an orientation of $\mathbf{q}_{i,j}$ relative to the co-ordinate frame of the node $\mathbf{P}_i$ as shown in equation 2.17 below.

$$\mathbf{q}_i^{-1}\mathbf{q}_j = \mathbf{q}_{i,j} \tag{2.17}$$

In a similar fashion to the translational component, this rotation component of the constraint can be fulfilled by either of the criteria given by equations 2.18 or 2.19.

$$\mathbf{q}_j = \mathbf{q}_i\mathbf{q}_{i,j} \tag{2.18}$$

$$\mathbf{q}_i = \mathbf{q}_j\mathbf{q}_{i,j}^{-1} \tag{2.19}$$

And again by altering both of the node orientations $\mathbf{q}_j$ and $\mathbf{q}_i$ to be slightly closer to those above, the constraint $\mathbf{R}_{i,j}$ comes closer to being fulfilled. Such alteration are performed

by spherical linear interpolation (Slerp [90]), shifting both of the quaternions $\mathbf{q}_j$ and $\mathbf{q}_i$ by some small interpolation factor $\gamma \in \mathbb{R}$ towards 2.18 and 2.19 respectively.

2.8.2.5 PGO Algorithm Outline

Each iteration of the PGO algorithm performs the previously described pose alterations for each of the relative pose constraints of R. A complete iteration is outlined in Algorithm 2.3. Convergence of this algorithm can be determined simply by checking that all constraints of R are being complied with using a threshold based metric, however in practice this PGO algorithm is run continuously on a separate thread to the sensor tracking and thus convergence detection is not required. The values of $\gamma = 0.01$ and $\varepsilon = 0.03$ were used in all scenarios.

INPUT:
P *// estimated keyframe poses*
R *// relative pose constraints from detected loop closures*

OUTPUT: P *// Adjusted keyframe poses*

for all $\mathbf{R}_{i,j} \in R$ **do**

// alter the translation component of poses $\mathbf{P}_i, \mathbf{P}_j \in P$
$\mathbf{t}_j = \mathbf{t}_j + \varepsilon(\mathbf{t}_i + \mathbf{R}(\mathbf{q}_i)\mathbf{t}_{i,j} - \mathbf{t}_j)$
$\mathbf{t}_i = \mathbf{t}_i + \varepsilon(\mathbf{t}_j - \mathbf{R}(\mathbf{q}_i)\mathbf{t}_{i,j} - \mathbf{t}_j)$

// alter the rotation component of pose $\mathbf{P}_i, \mathbf{P}_j \in P$
$\mathbf{q}_j = Slerp(\mathbf{q}_j, \mathbf{q}_i\mathbf{q}_{i,j}, \gamma)$
$\mathbf{q}_i = Slerp(\mathbf{q}_i, \mathbf{q}_j\mathbf{q}_{i,j}^{-1}, \gamma)$

end for
return P

Algorithm 2.3: *Relaxation Based Pose Graph Optimization*

2.8.3 Loop Closure Examples

This section briefly illustrates a sample of results relating to map optimization. Figure 2.15 illustrates detected loop closure constraints between keyframes, while Figure 2.16 compares the mapping results of SLAM with and without the use of map optimization.

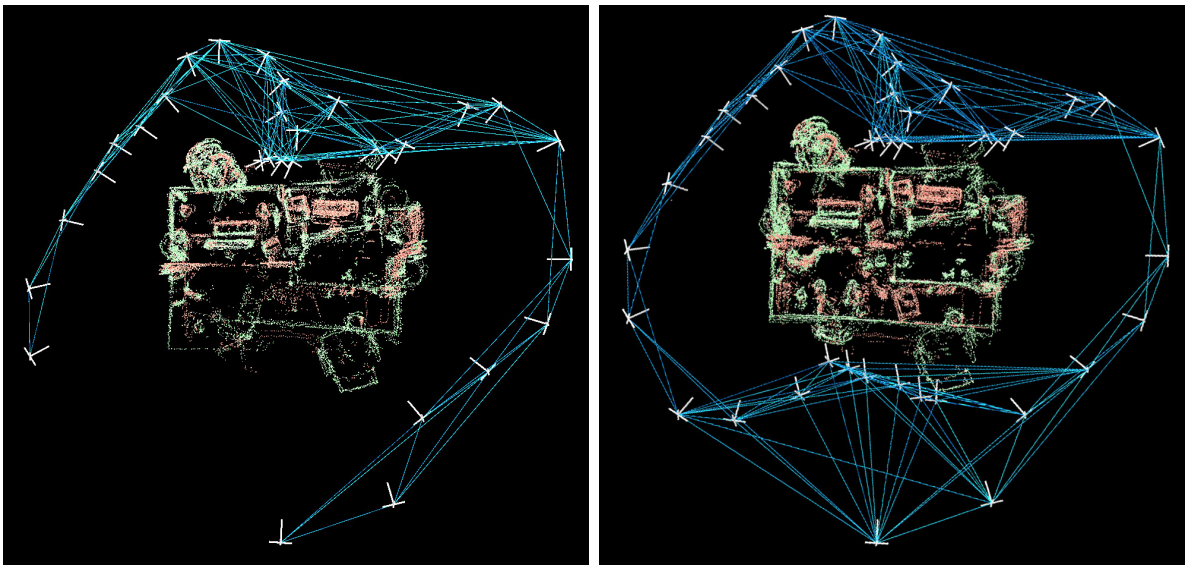
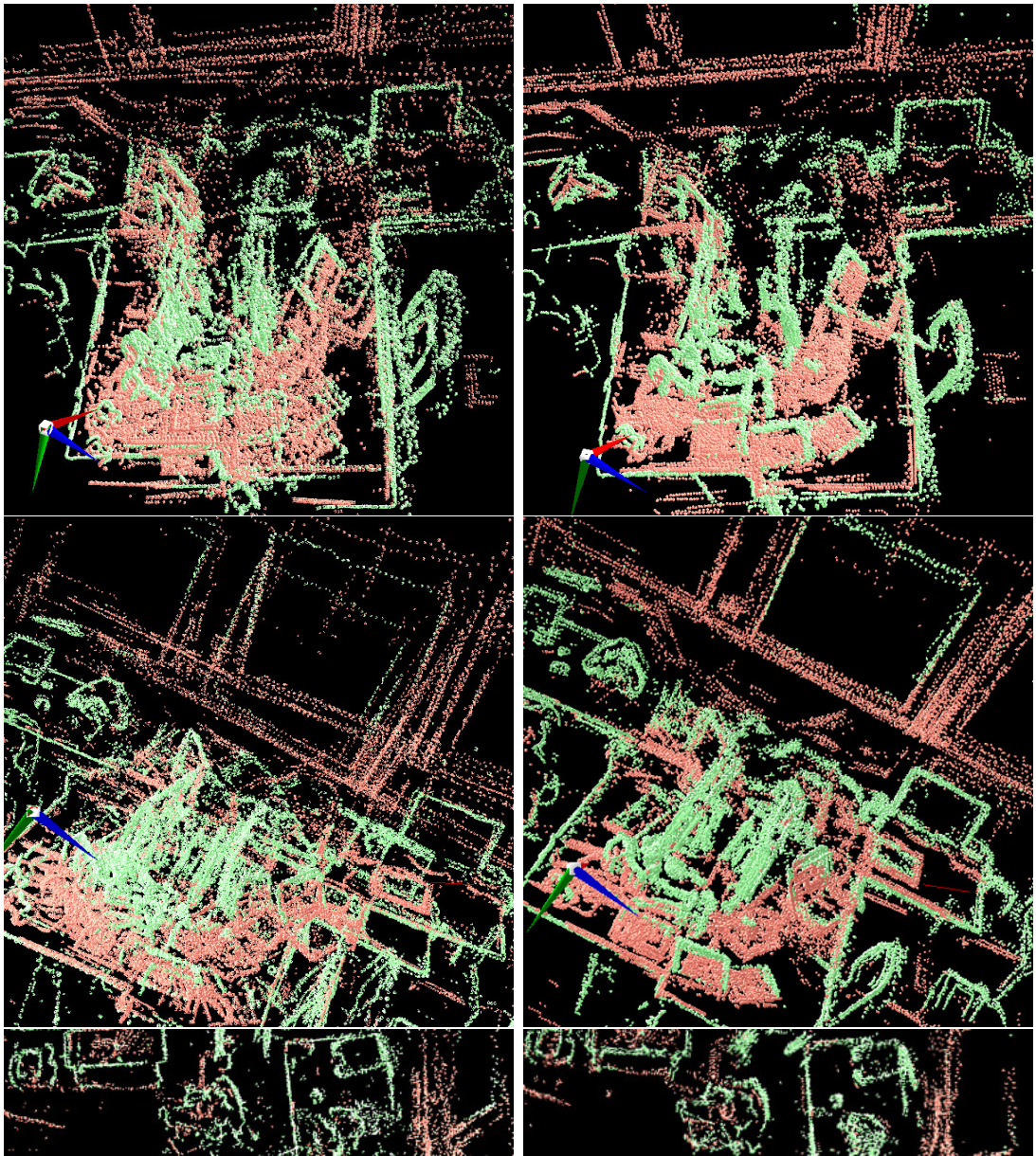


Figure 2.15: *An example of loop closures detected on the "FR3 long office household" RGB-D sequence provided by [96]. Keyframes axes are drawn in white, with loop closures between keyframes drawn in blue.*



2.9 Results

2.9.1 SLAM Evaluation

This section presents a sample of results from various experiments conducted to evaluate the performance of the edge based SLAM from Section 2.3. Again as in Section 2.4.3, we use the Freiburg RGB-D dataset [97] for this evaluation and the same 2.60GHz Intel Core i5-3230M(2013), 4GB RAM system running Ubuntu 14.10.

The estimated sensor trajectories produced were compared to the recorded ground truth trajectories using the evaluation tools provided by [97]. All results were obtained using an image patch grid size of 32×24 as discussed in Section 2.4.2.

Table 2.4 gives an evaluation of our proposed SLAM system on a number of datasets and also details the effects of altering the row/column skipping parameter $rowcol\_skip$ given to the depth edge detection process. Increasing the value of $rowcol\_skip$ results in fewer edge pixels being returned by the edge detection process and thus smaller down-sampled edge point clouds being used by the SLAM system. We observed that the system's accuracy displayed a surprising level of robustness to this form of downsampling. With $rowcol\_skip = 10$, only every 10^{th} row and column of each image patch is searched for edge pixels. On average this results in 10 times fewer edge pixels being detected (and thus 10 times smaller edge point clouds) compared to when $rowcol\_skip = 1$. Despite using such smaller point clouds the resulting sensor trajectories are still comparable in accuracy to those obtained when using no downsampling ($rowcol\_skip = 1$). Using larger values of $rowcol\_skip$ also greatly decreased the total runtime on each data set largely due to the smaller edge point clouds resulting in much faster ICP registration. Because of this robustness to downsampling and the desire of real time performance on limited hardware, the parameter value of $rowcol\_skip = 5$ was chosen to be the default for the proposed SLAM system.

Table 2.3 shows a comparison between results from the proposed SLAM system and other RGB-D SLAM systems on the same Freiburg datasets. These systems are the well known SIFT feature based RGB-D SLAM [24] (running on a "quad-core CPU with 8 GB of memory") and the occluding edge based SLAM system presented by [12] (running on a Intel Core i7 CPU, 8GB memory). The edge based SLAM system of [12] uses occluding depth edges in a similar manner to our own system but does not make use of RGB edges and utilizes a different method of occluding edge detection with significantly higher run times (as illustrated Table 2.2). We see that in general our system provides a comparable levels of accuracy while having far shorter total run-times, being able to

Table 2.3: *Evaluation results of the SLAM system proposed in Section 2.3 on various RGB-D video sequences along with comparisons to other SLAM systems. Reported results were obtained with the edge detection parameter rowcol_skip = 5.*

Translational RMSE			
Sequence (length)	**SIFT based RGB-D SLAM**	**Occluding edge based SLAM**	**Proposed RGB-D edge SLAM**
FR1 desk (23 s)	**0.049 m**	0.153 m	0.075 m
FR1 desk2 (25 s)	0.102 m	0.115 m	**0.098 m**
FR1 plant (42 s)	0.142 m	0.078 m	**0.076 m**
FR1 room (49 s)	0.219 m	**0.198 m**	0.210 m
FR1 rpy (28 s)	**0.042 m**	0.059 m	0.055 m
FR1 xyz (30 s)	**0.021 m**	**0.021 m**	0.038 m

Rotational RMSE			
Sequence (length)	**SIFT based RGB-D SLAM**	**Occluding edge based SLAM**	**Proposed RGB-D edge SLAM**
FR1 desk (23 s)	**2.42 deg**	7.47 deg	3.43 deg
FR1 desk2 (25 s)	3.81 deg	5.87 deg	**3.75 deg**
FR1 plant (42 s)	6.34 deg	5.01 deg	**4.09 deg**
FR1 room (49 s)	9.04 deg	6.55 deg	**5.66 deg**
FR1 rpy (28 s)	**2.50 deg**	8.79 deg	4.20 deg
FR1 xyz (30 s)	**0.90 deg**	1.62 deg	1.92 deg

Total Runtime			
Sequence (length)	**SIFT based RGB-D SLAM**	**Occluding edge based SLAM**	**Proposed RGB-D edge SLAM**
FR1 desk (23 s)	199 s	65 s	**14 s**
FR1 desk2 (25 s)	176 s	92 s	**16 s**
FR1 plant (42 s)	424 s	187 s	**29 s**
FR1 room (49 s)	423 s	172 s	**30 s**
FR1 rpy (28 s)	243 s	95 s	**16 s**
FR1 xyz (30 s)	365 s	111 s	**17 s**

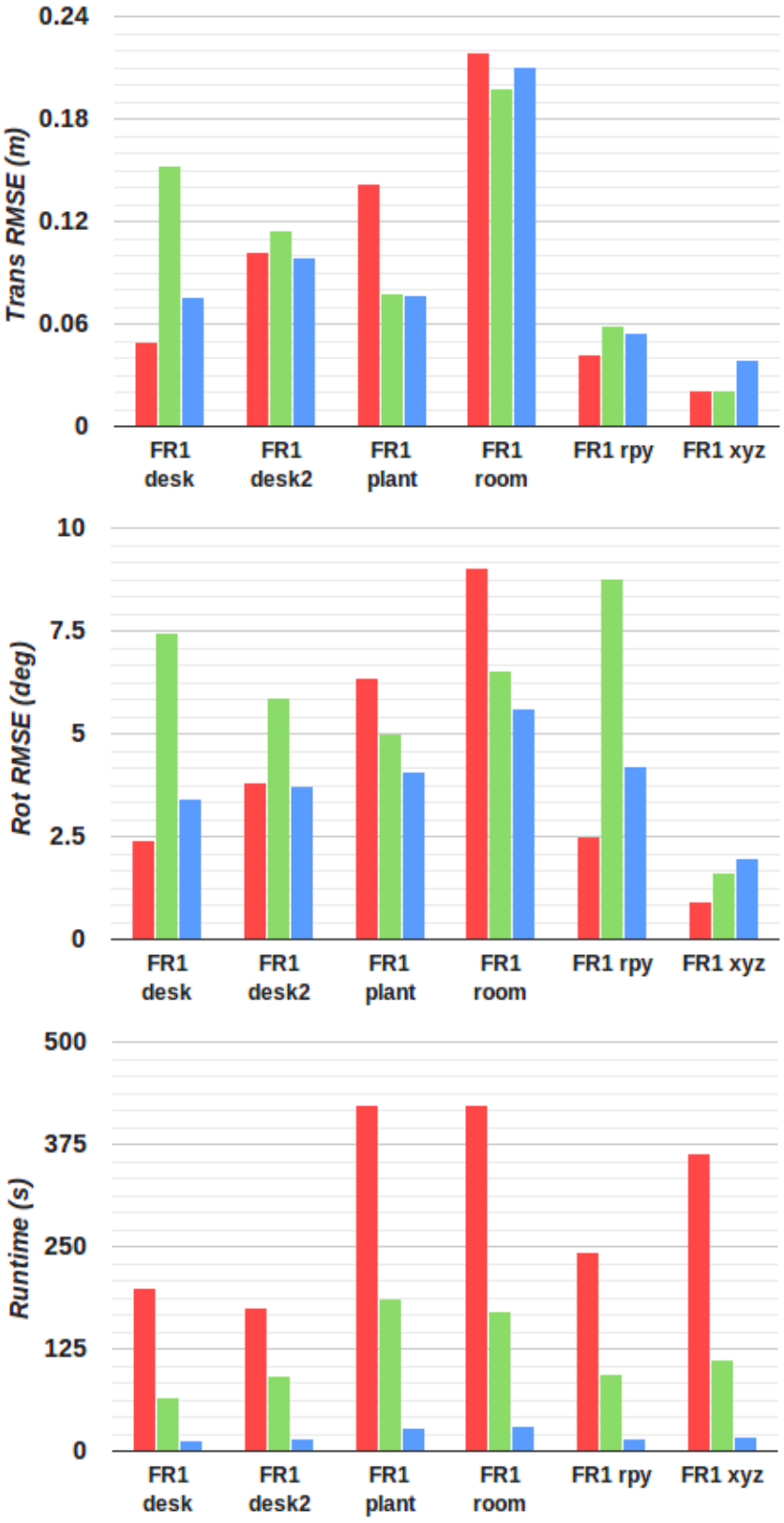


Figure 2.17: *Plots comparing results obtained by different SLAM systems on a number of Freiburg datasets. Our proposed edge based SLAM is shown in Blue, RGB-D SLAM [24] in red and occluding edge RGB-D SLAM [12] in green.*

Table 2.4: *Results detailing the effects of changing the rowcol_skip parameter on SLAM system performance (translational RMSE, rotational RMSE and total runtime).*

Row Col Skip Value	FR1 desk 23 s	FR1 desk2 25 s	FR1 plant 49s	FR1 rpy 28 s
1	0.081 m 3.65 deg 79 s	0.101 m 3.69 deg 93 s	0.078 m 4.22 deg 149 s	0.061 m 4.39 deg 101 s
5	0.075 m 3.43 deg 14 s	0.098 m 3.75 deg 16 s	0.076 m 4.09 deg 29 s	0.055 m 4.20 deg 16 s
10	0.085 m 3.76 deg 7 s	0.111 m 3.88 deg 8s	0.098 m 5.19 deg 15 s	0.056 m 4.25 deg 8 s
20	0.103 m 4.13 deg 4 s	0.122 m 4.21 deg 5 s	0.098 m 4.10 deg 10 s	0.056 m 4.24 deg 4 s

process the 30Hz sequences in real-time.

2.9.2 Further Results

In addition to the Freiburg RGB-D sequences, the proposed SLAM system was also tested live in a number of different environments. Although these tests have no ground truth from which to conduct a complete system accuracy evaluation, the resulting maps are still useful indicators of the system's capabilities. This section presents a number of such maps along with brief analysis.

Figures 2.21 and 2.23 were obtained from partially mapping the interior of a two storey house. Starting from the kitchen on the ground floor three other rooms were explored before returning back to the initial starting location. From Figure 2.21 it can be seen that some degree of drift has occurred during mapping as the rooms on the upper floor do not perfectly align with the rooms below and since such rooms are not observable simultaneously no loop closures may be detected to correct such errors. Despite this however it is clear that a reasonable level of accuracy has been achieved. In this particular environment at no point was there insufficient depth or rgb edge features present to cause a loss of slam tracking. Figure 2.23 shows close up views of some of the rooms present.

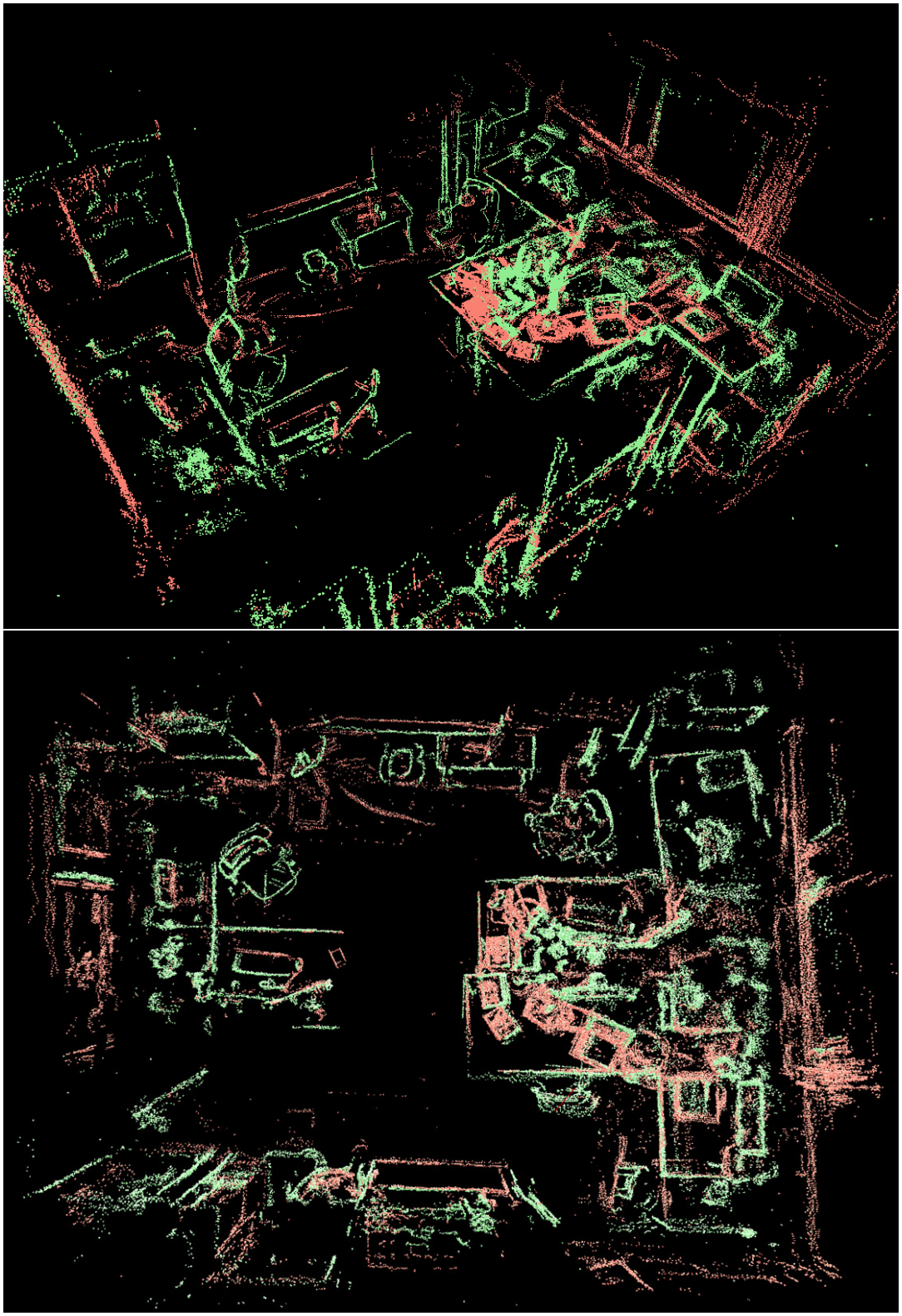


Figure 2.18: *Map created by the proposed SLAM from the FR1 room dataset. Point clouds of the keyframes are drawn with occulding edge point clouds in green and RGB edge point clouds in red.*

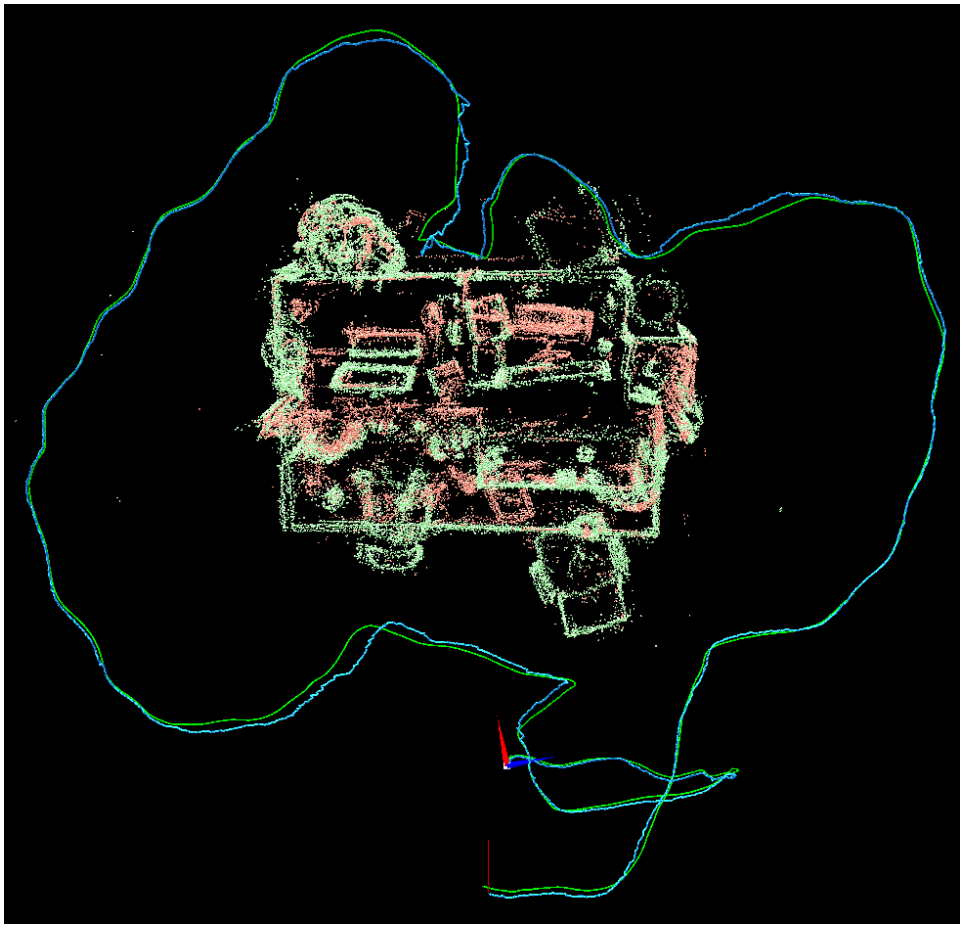


Figure 2.19: *Map created by the proposed SLAM from the FR3 "long office household" RGB-D sequence. The ground truth trajectory of the sensor is drawn in green, while that estimated by the SLAM system is shown in blue.*

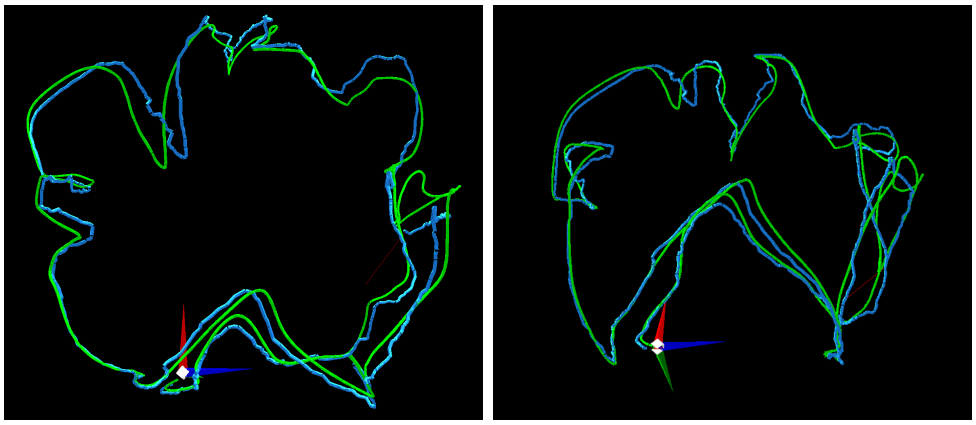


Figure 2.20: *Comparison of ground truth sensor trajectory (green) and SLAM's estimated sensor trajctory (blue) for the FR1 plant RGB-D sequence.*

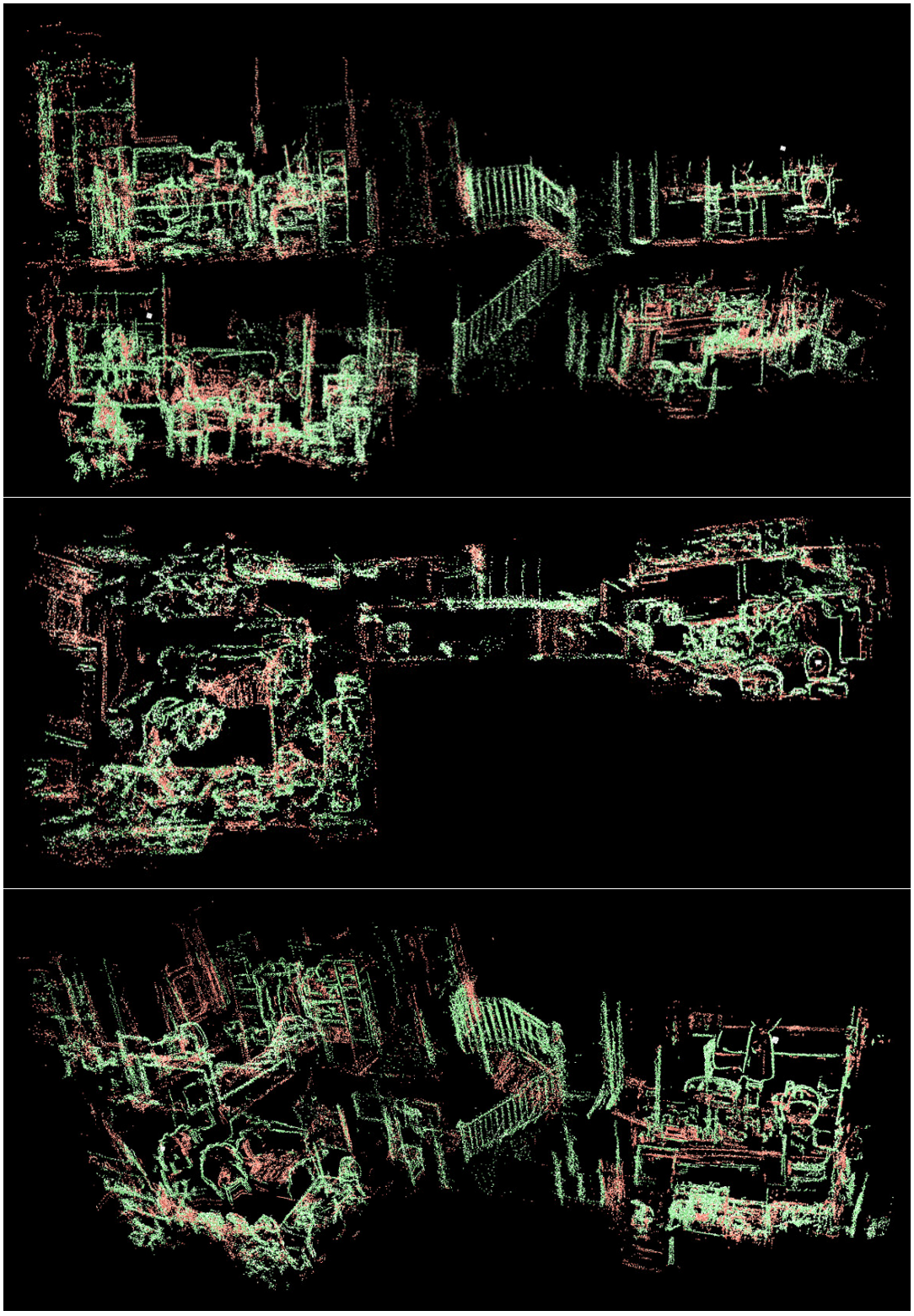


Figure 2.21: *Example map created by partially mapping a two story house. Point clouds of the keyframes are drawn with occulding edge point clouds in green and RGB edge point clouds in red.*

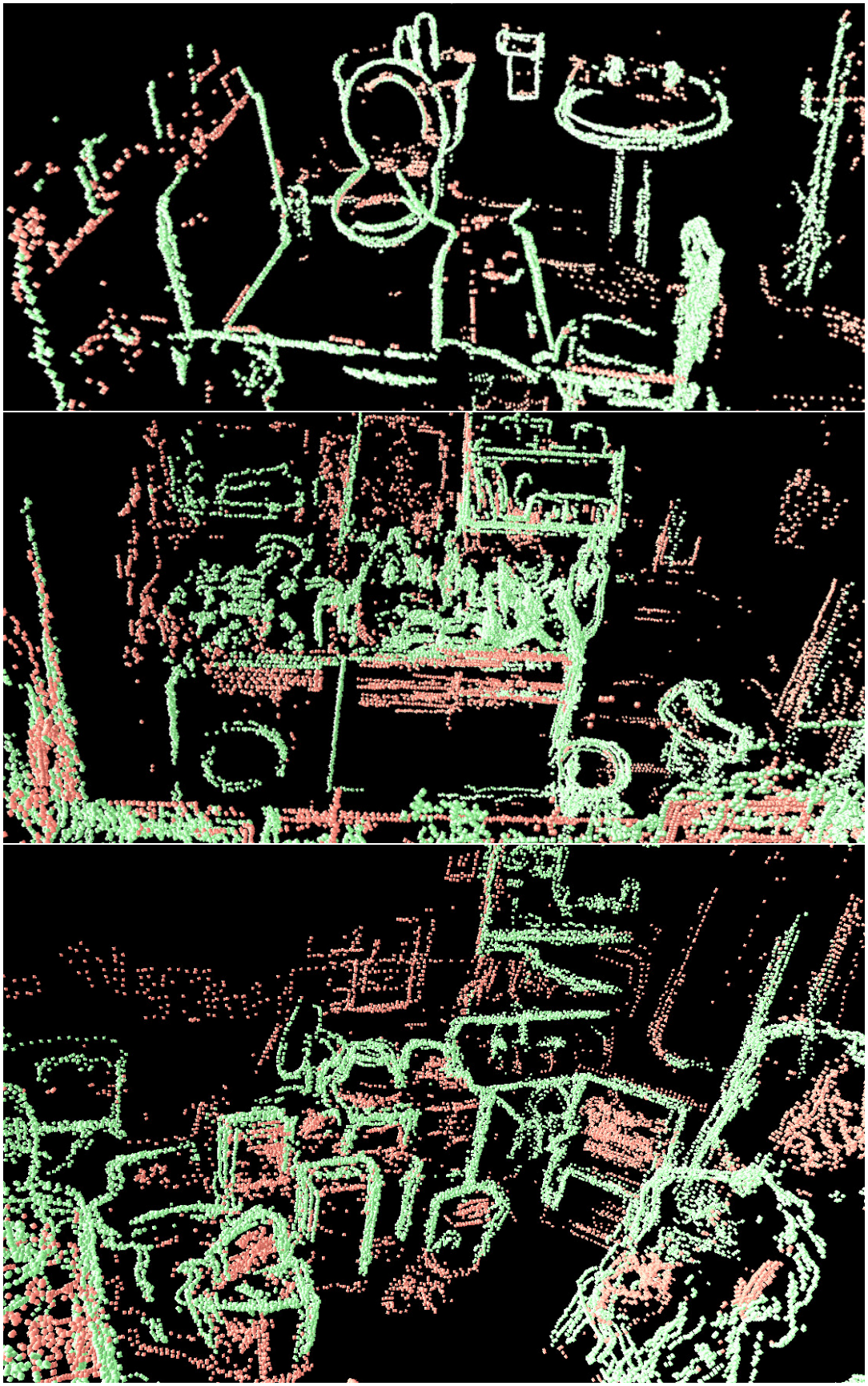


Figure 2.22: *Close views of the room featured in the map of Figure 2.21. Point clouds of the keyframes are drawn with occulding edge point clouds in green and RGB edge point clouds in red.*

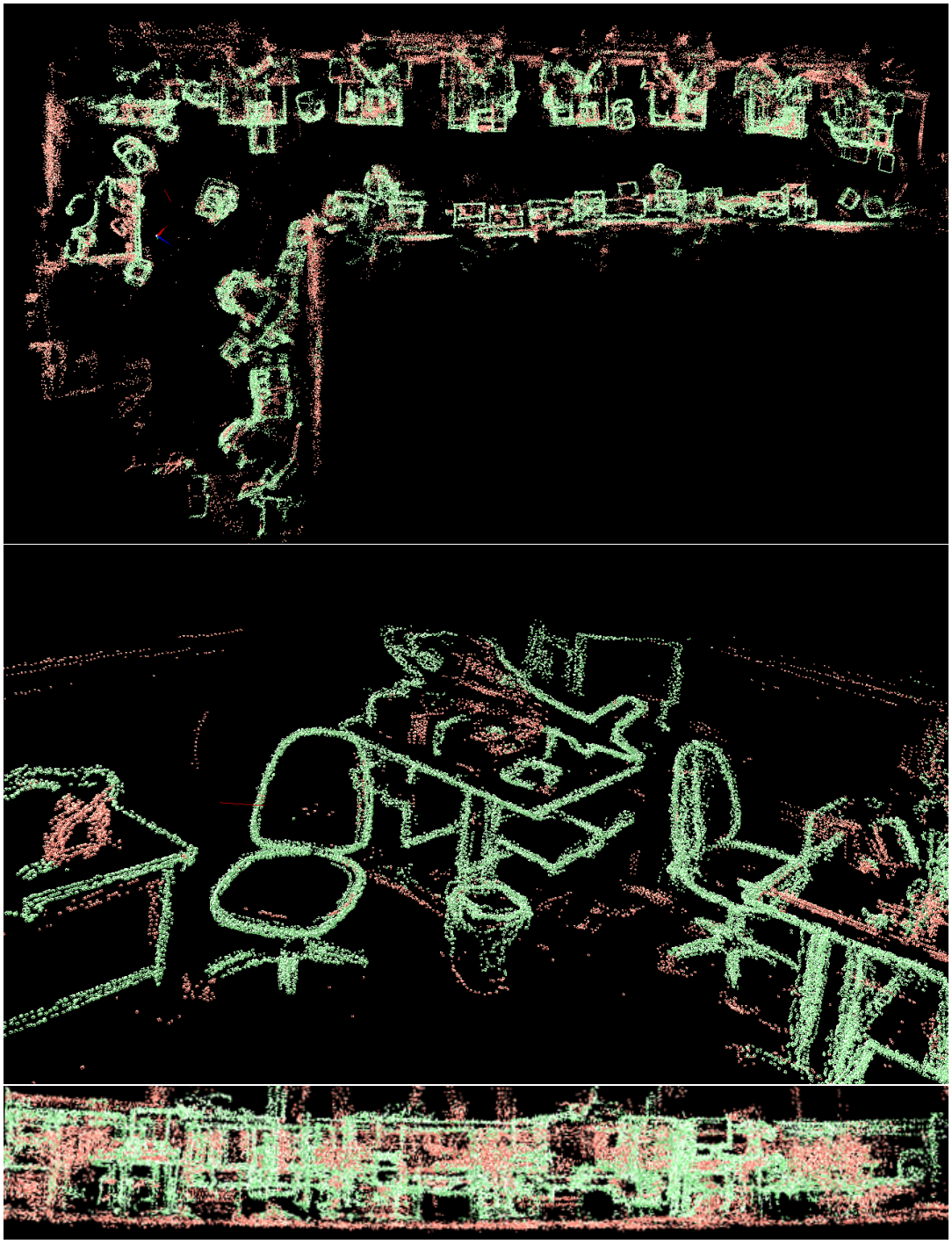


Figure 2.23: *Example map created by mapping an office space. Point clouds of the keyframes are drawn with occulding edge point clouds in green and RGB edge point clouds in red.*

2.10 Conclusions

In this chapter an edge based RGB-D SLAM system was proposed. Both RGB and occluding depth edges are utilized, being extracted from each RGB-D frame and back-projected to form semi-dense edge point clouds. Our occluding depth edge detection method exploits the temporal similarities between RGB-D frames to direct the search process to only specific image regions where edges are likely to reside, saving significant computation time compared to the edge detection employed by other works. A keyframe set-up is used and the sensor pose is estimated by registering the latest edge clouds with those of the map's keyframes. It is demonstrated how this edge based ICP may provide both significantly faster and more accurate registration compared to using uniformly down-sampled RGB-D cloud data, and how its performance does not significantly degrade with downsampling.

As demonstrated in the evaluation, the proposed system is competitive with similar systems at the time of writing, achieving similar levels of accuracy while having a computational overhead many times smaller, making it fit for real-time navigation applications on small robotic platforms as discussed in 2.2 at the start of this chapter.

Chapter 3

Line Based RGB-D SLAM

The depth and RGB edge point clouds discussed previously in Chapter 2 are far smaller in size compared to their raw RGB-D cloud counterparts. Despite this reduction edge point clouds may still contain various high level features within them. One such type of high level feature is that of straight 3D line segments represented by large subsets of collinear points in the edge point cloud. This chapter presents an investigation in extending the previously proposed edge cloud based SLAM system to utilize such high level linear features.

The majority of man-made objects such as monitors, desks and posters all feature many such straight lines and edges, either in their physical geometry or their surface texture. Since we are primarily interested in operating in such man-made environments we investigated the use of such features for SLAM, modifying the SLAM system described in the previous section to utilize such line features.

One of the major benefits of using such high level 3D line features is that of compactness. Edge point clouds are typically around two orders of magnitude smaller than their full RGB-D cloud counterpart. The process of extracting 3D line features from such edge point clouds typically involves hundreds of points being merged to form a single line feature. Thus the total number of line features is again around two orders of magnitude fewer than the number of points in the edge cloud they were extracted from. Because of this great reduction in the number of features, SLAM tracking and registration can be performed at a further reduced computational cost. Thus if the edge based SLAM system of the previous chapter is unable to achieve consistent 30Hz real-time performance on a certain platform, the line and edge cloud based system proposed here may provide a preferable alternative.

However, there are downsides of making use of such high level features. Many environments contain geometric objects and textured surfaces featuring curved or irregular edges. Naturally 3D lines cannot fully represent these curved edges, instead multiple lines must be used to form an approximate representation. Additionally extraction of these 3D line features adds yet another layer of post processing, further abstracting away from the original RGB-D data. Accurate, reliable and repeatable 3D line detection can be a challenging task in many scenarios. Certain detected line features may be inaccurate, and in some cases certain line features may fail to be detected entirely. As such, SLAM using such 3D line features for mapping is in general less accurate in comparison to SLAM using edge point clouds. This is the trade-off made for the decreased computational cost such features provide.

We denote such 3D line features with a pair of 3D vectors i.e. $L = (\mathbf{l}_1, \mathbf{l}_2)$ where $\mathbf{l}_1$ and $\mathbf{l}_2$ represent the start and end locations of the line respectively.

3.1 Incorporation of Line Features

As discussed previously line features provide a very compact description of a scene and thus can be used to obtain fast registration and sensor tracking. However, the computational cost of extracting such line features from edge clouds is not insignificant, and performing such extraction with every new frame of RGB-D data would have a significant performance impact dramatically increasing the average frame time. Thus instead of switching to a completely line feature based SLAM system, we adopt a hybrid approach creating a system that makes use of both edge clouds and line features.

This hybrid system is largely identical to the edge point cloud based system introduced in the previous sections, however, each map keyframe K_i now stores two sets of 3D line features in addition to the two edge point clouds D_i and I_i. These sets of line features are again associated with the depth and RGB components of the keyframes RGB-D data respectively. Registration and sensor tracking is achieved by extracting depth and RGB edges from the latest RGB-D frame F, generating the associated depth edge cloud D and RGB edge cloud I; and then finally registering said point clouds D and I with the depth and RGB line features stored by the current tracking keyframe K_T. Despite still involving edge point clouds (rather than purely sets of line features) such registration can still be performed at a significantly reduced computational cost compared to purely edge cloud based registration.

In this way the proposed hybrid system utilizes line feature to reduce the computational

cost of sensor tracking, while also keeping the total computational cost involved in line feature extraction low by only having to extract such line features once per map keyframe $K_i \in K$ (using each keyframe associated RGB-D frame F_i), rather than extracting line features from every RGB-D frame.

The following subsections now introduce the method used to extract such line features from edge point clouds, the method used to perform registration between such line features and edge point clouds, and finally results and evaluation of the hybrid SLAM system utilizing both of these types of features.

3.2 Line Segment Extraction from Edge Point Clouds

Given an edge point cloud E we wish to be able to identify all 3D line features present within it. These line features are to be used as features for the SLAM map keyframes themselves, and thus an accurate and robust method of extraction is required. There are numerous possible approaches to such a task with varying degrees of robustness and computational cost.

Our implemented method of 3D line segment extraction employs a split and merge based approach, partitioning the original edge point cloud up into multiple small subsets $c \subset E$. This partitioning is conducted such that each $c \subset E$ consists of an approximately collinear set of points, and the set of all such subsets collinear is denoted C. A merging process is then performed, which iterates through possible pairs of these sets $(c_i \in C, c_j \in C) | c_i \neq c_j$, merging any pairs into a single set provided that their union of points is also approximately collinear i.e $C = \{c_i \cup c_j\} \cup S - \{c_i, c_j\}$. This criteria of collinearity is checked by simply fitting a 3D line to the set of points in question and then determining the average distance from said line. This process is repeated until no further mergers can be made, resulting in the creation of the final sets of approximately collinear points. The final set of 3D line features is then formed by the lines of best fit to each of these collinear point sets $c \in C$.

The steps of this extraction process are illustrated in Figure 3.1.

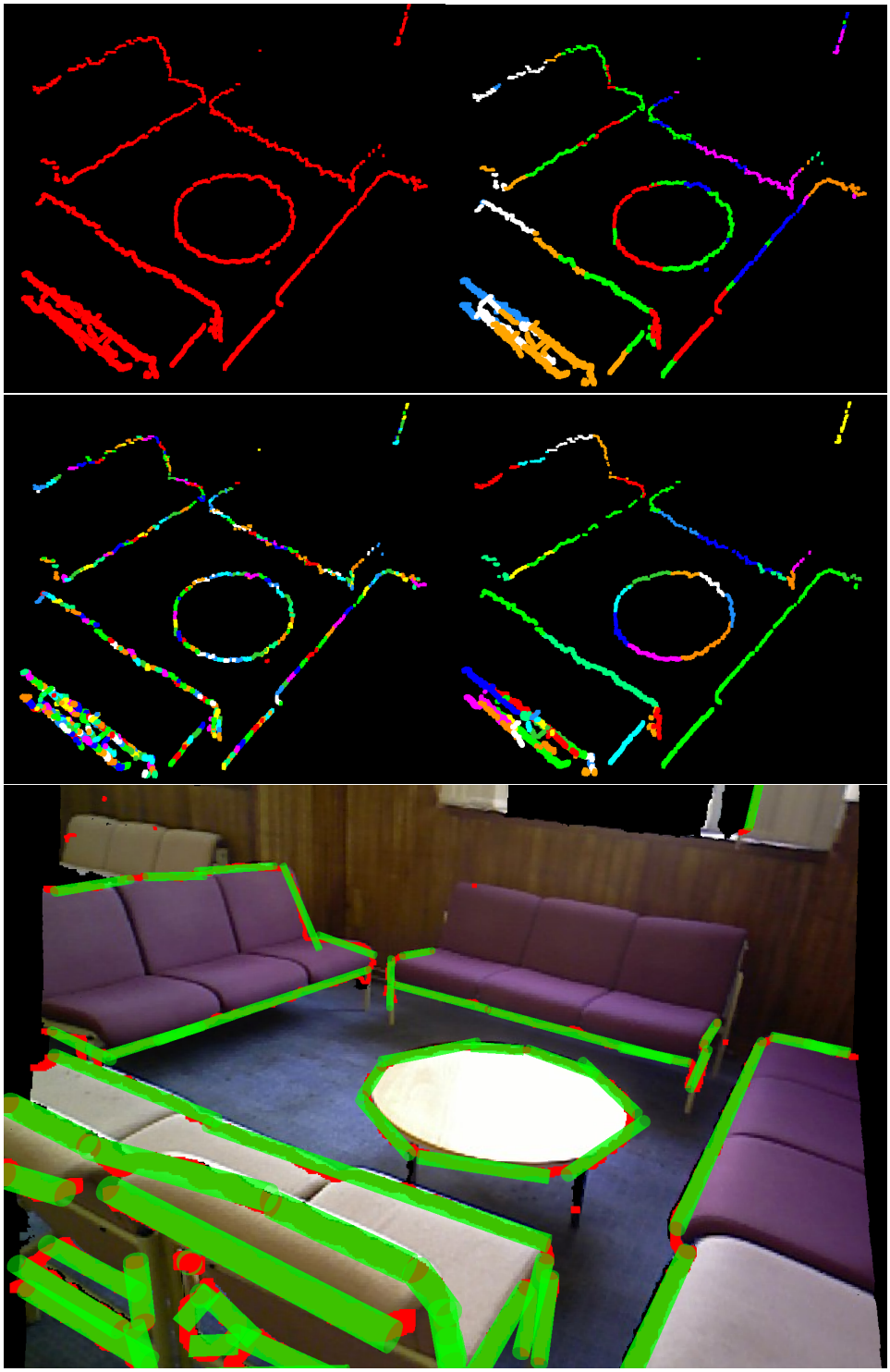


Figure 3.1: *An illustration of the steps involved in extracting linear segments from an edge point cloud. The set of 5741 points are partitioned into 277 subsets of collinear points which are then merged to form the final 43 collinear point sets and line features. which the final line segments are extracted.*

3.2.1 Grid Partitioning

The first step of this extraction process is to partition the edge point cloud E into multiple smaller sets, primarily to reduce the computational cost of nearest neighbour searches required in the following steps. This involves calculating the bounding box of the point cloud E comprised of the two vectors $\mathbf{B}^{min}$ and $\mathbf{B}^{max}$, which are then in turn used to determine a 3D grid partitioning the cloud. Let B^{dim} denote the size of each grid cell (i.e. each cell is of dimensions $B^{dim} \times B^{dim} \times B^{dim}$), and $b_{ijk} \subset E$ denote the subset associated with the cell at grid co-ordinates (i, j, k), consisting of all points of E contained within that grid cell. Naturally such a subset b_{ijk} must contain at least two points for a line feature to be present, however, it is of course always possible to draw a perfect line between two points and in general far more collinear points are required to be sufficient certainty that a line feature is present. We denote this minimum number of points that a line feature must be associated with by Z, and thus in turn we are only interested in those b_{ijk} which contain at least Z points. For simplicity we denote the set B as consisting of all such grid cell subsets, i.e.

$$B = \{b_{ijk} : |b_{ijk}| \geq Z\} \tag{3.1}$$

3.2.2 Collinear Subset Construction

The next step is to split each grid cell's subset of points $b \in B$ into multiple subsets denoted $c_0, c_1, c_2 ... c_n$. Each of these subsets $c \subset b$ consist of a small cluster of points, such that each point $\mathbf{p} \in c$ is within a minimum distance μ of at least one other point $\mathbf{q} \in c$ as given in Equation 3.2 below.

$$\forall(\mathbf{p} \in c)\exists(\mathbf{q} \in c) : \mathbf{p} \neq \mathbf{q} \wedge |\mathbf{p} - \mathbf{q}| \leqslant \mu \tag{3.2}$$

Due to the nature of edge point clouds, such clusters of points predominately consist of short strands of points lying upon a common edge. By limiting the size of such clusters to a small number of points $K \in \mathbb{N}$ (such that $|c| \leqslant K$) , it is likely that the majority of them will be sets of approximately collinear points which can be reasonably represented by 3D line features.

Algorithm 3.1 outlines the process of generating each such collinear subset $c \subset b$. This process begins by initializing a subset c by removing a randomly selected point from b

and adding it to c, after which this subset is iteratively grown in size by again removing points from the set b and adding them to c. The points used to grow c in this manner are selected by finding the current closest point in b to the latest point added to the set c. Nearest neighbour searches have to be conducted to find such closest points. However, since b typically does not contain a great number of points such searches can be conducted using brute force approach while still not becoming a major performance bottleneck. After a set c has been grown in size to the desired number of points K, the line of best fit of c is calculated and used to evaluate if the set is deemed to be sufficiently collinear as to represent a 3D line feature. This evaluation simply consists of calculating the average distance between each point from the subset $\mathbf{p} \in c$ and the line of best fit consisting of start and end points $\mathbf{l}_1$ and $\mathbf{l}_2$ as shown in Equation 3.3.

$$R^{c,(\mathbf{l}_1,\mathbf{l}_2)} = \frac{1}{\mid c \mid} \sum_{\mathbf{p} \in c} \mid (\mathbf{p} - \mathbf{l}_1) - (\mathbf{p} - \mathbf{l}_1) \cdot \frac{(\mathbf{l}_2 - \mathbf{l}_1)}{\mid \mathbf{l}_2 - \mathbf{l}_1 \mid} \mid \tag{3.3}$$

If this error $R^{c,(\mathbf{l}_1,\mathbf{l}_2)}$ is below a threshold value ϵ then the generated subset c is added to the set C containing all generated collinear subsets. Otherwise it is rejected as it is not deemed to represent a reliable 3D line segment.

3.2.3 Set Merger

The final step in this 3D line extraction process involves attempting to merge together pairs of collinear points sets from C, in an attempt to create larger collinear sets. This involves iterating through each possible pair of sets $(a \in C, b \in C) : a \neq b$ and determining if the set consisting of their union $a \cup b$ is also a set of approximately collinear points. This is done by calculating the line of best fit to the points of $a \cup b$ and then evaluating how well this line fits said points. If deemed adequate, then the sets a, b are removed from C and replaced by the set consisting of their union $c = a \cup b$. This process is repeated until no further mergers occur. A full outline of this merger step is given below in Algorithm 3.2.

The steps of this merging process are illustrated in Figure 3.1, along with the output linear segments drawn on top of the raw RGB-D point cloud.

for all $b \in B$ **do**
  while $\mid b \mid > 0$ **do**
    $\mathbf{p} \in b$ *// select a random point from b*
    $\mathbf{q} = \mathbf{p}$ *// store last used point*
    $c = \{\}$ *// initialize new empty subset*
    while $\mid \mathbf{p} - \mathbf{q} \mid < \mu$ **and** $\mid c \mid < K$ **do**
      $c = c \cup \{\mathbf{p}\}$ *// add point to subset*
      $b = b - \{\mathbf{p}\}$ *// remove point from b*
      $\mathbf{q} = \mathbf{p}$ *// store last used point*
      $\mathbf{p} = Closest(b, \mathbf{q})$ *// find the closest point to* $\mathbf{q}$
    end while
    if $\mid c \mid == K$ **then**
      $(\mathbf{l}_1, \mathbf{l}_2) = LineOfBestFit(c)$ *// calculate best fit line*
      if $R^{c,(\mathbf{l}_1,\mathbf{l}_2)} < \epsilon$ **then** *// evaluate line fitting*
        $C = C \cup \{c\}$ *// add to set of collinear subsets*
      end if
    end if
  end while
end for
return P

Algorithm 3.1: *Collinear subset generation .*

Merged:
for all $(a \in C, b \in C) : a \neq b$ **do** *// iterate through all possible set pairs*
  $d = a \cup b$
  $(\mathbf{l}_1, \mathbf{l}_2) = LineOfBestFit(d)$ *// calculate best fit line*
  if $R^{d,(\mathbf{l}_1,\mathbf{l}_2)} < \epsilon$ **then** *// evaluate line fitting*
    $C = C - \{a, b\}$
    $C = C \cup \{d\}$ *// add merged set to* C
    Goto *Merged*
  end if
end for

Algorithm 3.2: *Collinear set merger*

3.2.4 Extraction Results

A sample of results from this 3D linear feature extraction process are shown in Figure 3.2. Though some linear features that are obvious to the human eye have been missed in many cases the process typically produces sufficient features to conduct tracking. The computational of the the extraction process varies depending upon the input semi-dense edge cloud, however, is never seen to exceed 10ms.

3.3 Iterative Closest Line

The ICP process described previously in Section 2.6 was used to perform registration between two sets of input point clouds. Namely, the source clouds $S = (S_D, S_I)$ and target clouds $T = (T_D, T_I)$, where S_D, T_D are depth edge clouds and S_I, T_I are RGB edge clouds. Each iteration of this ICP process then determines pairs of points between these source and target clouds, specifically pairs between the two depth edge clouds S_D, T_D, and the two RGB edge clouds S_I and T_I. The calculation to best align all determined point pairs is then calculated and used to update the estimated alignment transformation. Over multiple iterations, this process causes the estimated alignment transformation to converge, producing a final estimate.

However, now that the keyframe features being used have been changed to sets of 3D line features a new registration process is required, one that can register edge point-clouds with sets of 3D line features. This subsection introduces a variant of the standard ICP algorithm which we used to achieve such registration.

Similar to ICP used for edge cloud registration, this variant takes two sets of features extracted from two different RGB-D frames, referred to as the "source" and target inputs. The source input $S = (S_D, S_I)$ again consists of a pair of depth and RGB edge clouds, however, the target T now consists of a pair of line feature sets. That is $T = (T_D, T_I)$ where T_D is a set of line features extracted from a depth edge cloud, and T_I is a set of line features extracted from a RGB edge cloud. As such each element of T_D is a line feature of the form described previously ($l = (\mathbf{l}_a, \mathbf{l}_b) \in T_D$), and similarly so is each element of T_I.

Each iteration then involves projecting each point from the depth edge cloud $\mathbf{p}_i \in S_D$ onto its nearest depth edge line feature $L \in T_D$, to form a projected point $\mathbf{q}_i$. Each point is then paired with its associated projected point (i.e. pairs of the form $(\mathbf{p}_i, \mathbf{q}_i)$), and added to a list of all point pairings generated in the current ICP iteration. An identical

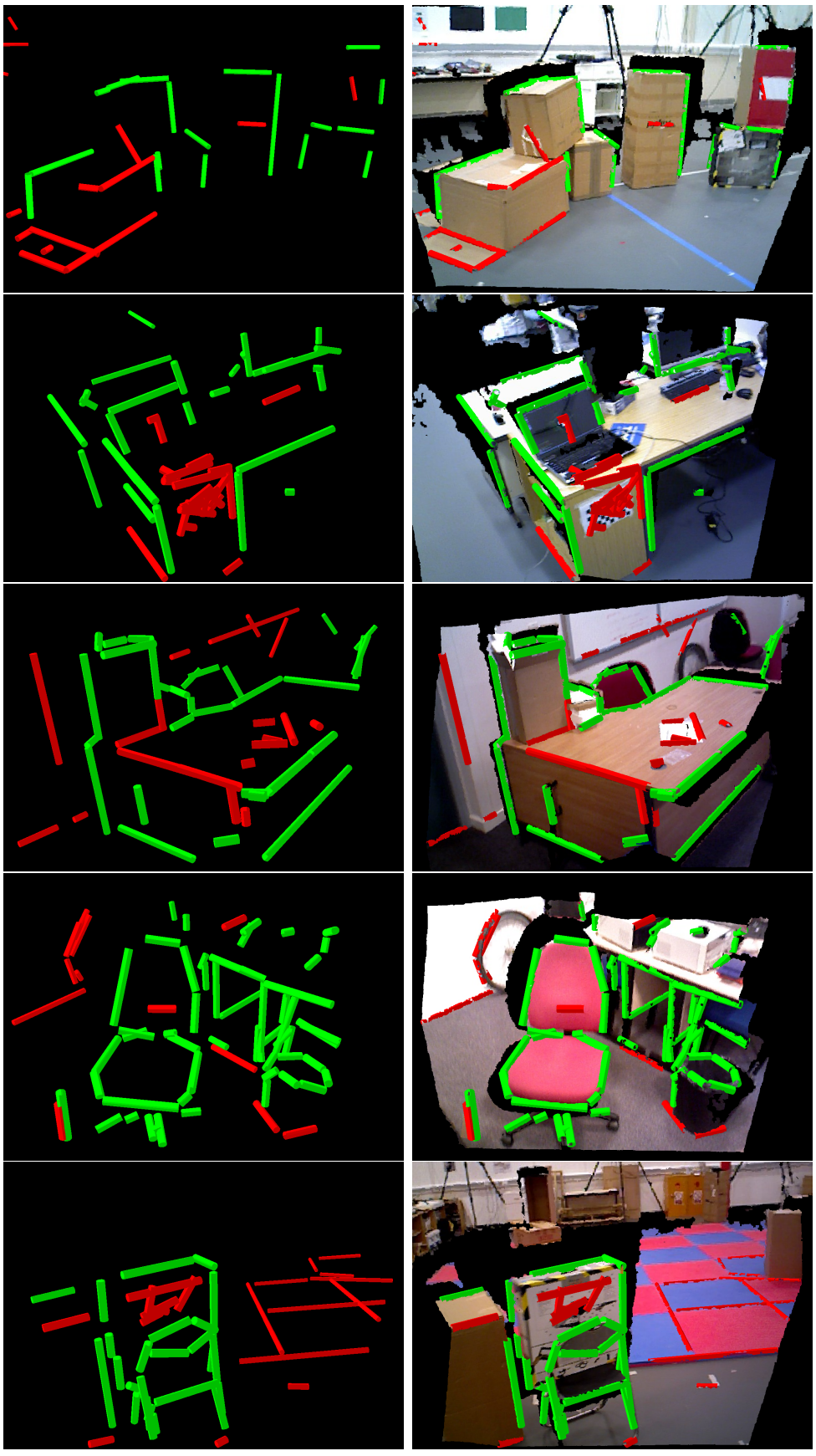


Figure 3.2: *Examples of extracted 3D linear features.*

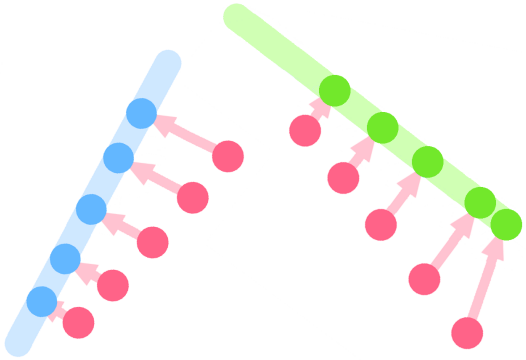


Figure 3.3: *A 2D illustration of point pair generation. Each point from the edge point cloud (shown in red) is paired with its projection onto the nearest line feature as indicated by the arrows. The transformation to minimize the total distance between these pairs is then calculated as in standard ICP.*

projection/pairing process is also conducted for the RGB edge cloud S_I and set of RGB edge line features T_I, generating additional point pairings. This process of determining point pairings by projection onto nearest line features is illustrated in 3.3.

After these point pairings have been determined the algorithm proceeds in an identical manner as standard ICP. The transformation to best align all determined pairs of points is calculated and used to update the estimated alignment transformation. Thus the core difference between this ICP registration process and that of Section 2.6 is simply in how the point pairings used to update the estimated transformation are generated each iteration.

3.3.1 Nearest Line Feature Search

As described above this point pairing generation involves the process of having to determine which line feature (from a set) is closest to a specific point. A naive approach to this would simply involve calculating the distance between the point and each of the line features in question. However the computational cost of this process would increases proportionally with the number of line features present. To avoid having to search an entire set of line features in this manner we make use of two spatially partitioning 3D grids, one associated with the target set of depth line features T_D, and a second associated with the target set of RGB line features T_I.

Such a partitioning grid G for a set of line features L, is constructed such that each grid cell C stores a specific subset of line features (either depth or RGB associated) denoted

$C_L \subseteq L$. This subset C_L is such that given any point $\mathbf{p}$ inside of the cell's bounding box volume, the closest line feature to said point must belong to C_L. Thus under this set-up, the process of finding the closest line feature to a point $\mathbf{p}$ consists of determining the grid cell C whose volume $\mathbf{p}$ resides in, and then examining the line features of C_L, to find that closest to $\mathbf{p}$. Since C_L is a subset of L this process avoids having to examine the entire set of line features, saving significant computational time. This is somewhat analogous to how K-D trees are used in standard ICP to improve the efficiency of nearest neighbour searches. Though the cost of calculating such grids may be too significant for real-time computation, in practice they only need be calculated once for each keyframe, the computation of which can be conducted on a separate thread from that conducting sensor tracking. The grid is uniform and hence each grid cell is the same size denoted C_D (so that each cell has volume C_D^3), this parameter is both tuned experimentally and by accounting for the computer hardware being used. Smaller values of C_D mean the grid cells have smaller volumes and thus in general the subsets C_L of possible closest line features will contain fewer elements increasing nearest line feature search performance, however this also leads to a greater number of grid cells requiring additional memory and thus the selection of the C_D value involves a trade-off between these two factors.

The construction of such a partitioning grid G involves a number of different steps, the first of which is to determine the volume of space which the grid should occupy in the form of a bounding box $B^G = (\mathbf{B}^G_{min}, \mathbf{B}^G_{max})$. This involves trivially calculating the bounding box of the set of line features L, expanding this bounding box equally in all directions by some fixed length G_{expand}, and finally decreasing and increasing the bounding box's minimum and maximum points such that it may encompass a whole number of C_D^3 volume grid cells. The step expanding the bounding of the box by G_{expand} is to allow the grid to still be used to find the closest line feature to points lying somewhat outside of the minimum bounding box of the set of line features L. Larger values of G_{expand} result in more grids cells requiring additional memory but allow for more robust registration, typically a value of $G_{expand} = 1m$ is sufficient, however, beyond this there is little observable benefit. This bounding box construction process is illustrated in Figure 3.4.

The volume of the grid's bounding box B^G is then uniformly split into multiple cubic bounding boxes, each of which represents the volume of a specific grid cell. The next step is to then calculate the subsets of line features associated with each cell of the grid. Let the bounding box of a specific grid cell C be denoted $B^C = (\mathbf{B}^C_{min}, \mathbf{B}^C_{max})$, and its associated subset of line features be denoted by $C_L \subseteq L$. Ideally C_L should consist of the smallest possible subset of L such for every point $\mathbf{p}$ inside of the cell's bounding box

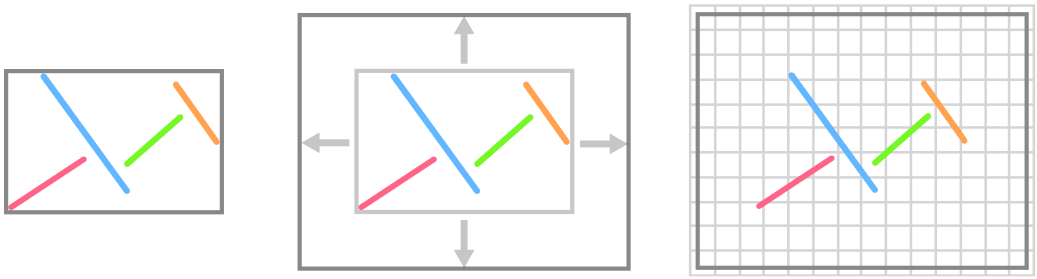


Figure 3.4: *A 2D illustration of the process of generating grid cells used in determining the closest line feature to a specific point. Left, the bounding box of all 3D line features is determined. Middle, this bounding box is then expanded by a fixed size in all directions. Right, finally the bounding box size is increased such that it can be partition into a uniform grid of cells of each of size C_D.*

volume B^C, the closest line feature to $\mathbf{p}$ belong to the subset C_L as is outlined below in Equation 3.4, where $NearestLine(\mathbf{p}, L)$ denotes the closest line features to $\mathbf{p}$ from the set L.

$$C_L = \{l \mid NearestLine(\mathbf{p} \in B^C, L) = l\} \tag{3.4}$$

However, the process of determining such a subset for a cubic volume of space requires significant computation which can significantly delay the addition of vital keyframes, causing temporary periods of tracking loss or poor quality tracking. Instead we settle for determining a similarly defined subset for the minimum bounding sphere of the cell's cubic bounding box B^C. The simplified geometry of this volume allows for an approximate solution to be quickly computed.

The minimum bounding sphere for a grid cell's bounding box B^C is simply located at position $\mathbf{x}$ in the center of the box, and has a radius r determined by the dimensions of the box itself as given in Equations 3.5 and 3.6.

$$\mathbf{x} = \mathbf{B}^C_{min} + 0.5 \times (\mathbf{B}^C_{max} - \mathbf{B}^C_{min}) \tag{3.5}$$

$$r = 0.5 \times |\mathbf{B}^C_{max} - \mathbf{B}^C_{min}| \tag{3.6}$$

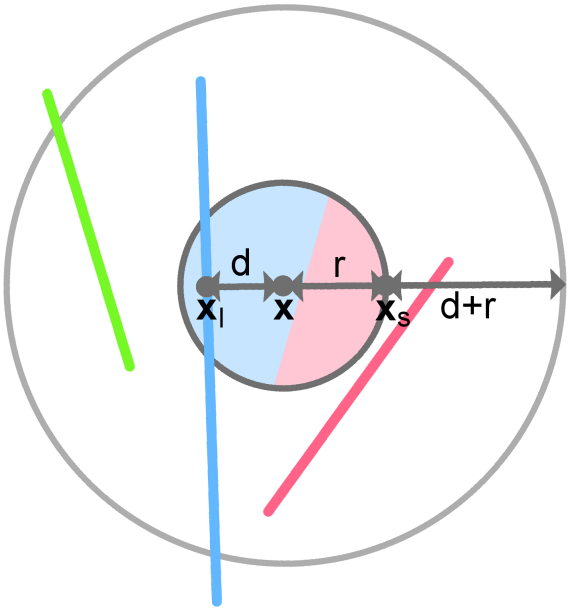


Figure 3.5: *Determining line features subset associated with grid cell bounding sphere.*

The process of determining the subset of line features for this bounding sphere first involves finding the line feature $l_1 \in L$ closest to the center of the bounding sphere $\mathbf{x}$. Let d denote the minimum distance between this line feature l_1 and the point $\mathbf{x}$, and let $\mathbf{x}_l$ denote the closest point to $\mathbf{x}$ on line feature l_1 (i.e. the projection of $\mathbf{x}$ onto l_1). Additionally, let $\mathbf{x}_s$ denote the point within the bounding sphere furthest from the line feature l_1, this point lies both upon the surface of the bounding sphere and along the line which passes through the points $\mathbf{x}$ and $\mathbf{x}_l$ as illustrated in Figure 3.5. Specifically $\mathbf{x}_s$ lies at a distance of $d+r$ from the line feature l_1, and is located at the position given by Equation 3.7 below.

$$\mathbf{x}_s = \mathbf{x} - r\frac{\mathbf{x}_l - \mathbf{x}}{|\mathbf{x}_l - \mathbf{x}|} \tag{3.7}$$

For any other line feature $l_2 \in L$ to possibly be the closer to the point $\mathbf{x}_s$ than the line feature l_1, the minimum distance between l_2 and $\mathbf{x}_s$ must be smaller than $d+r$. Since $\mathbf{x}_s$ is the furthest point from the line feature l_1 this same criteria hold for all points in the bounding sphere. That is for any point $\mathbf{p}$ within the bounding sphere and a line feature $l_2 \in L$, the minimum distance between l_2 and $\mathbf{p}$ must be smaller than or equal to $d+r$ for l_2 to possibly be the closest line feature to $\mathbf{p}$ (otherwise l_1 would be the closer feature). From the above statement it is clear that for a line feature l_2 to possibly

be the closest line feature to any point within the bounding sphere at all, l_2 must come within a minimum distance of $d+r$ of the bounding sphere itself. In other words l_2 must intersect the volume of the sphere which centred about $\mathbf{x}$ and has radius $r + (d + r)$ as illustrated in Figure 3.5.

Thus for any point $\mathbf{p}$ in the minimum bounding sphere of the cell C, the closest line feature to $\mathbf{p}$ will too belong to the subset of line features C_L consisting of those line features which intersect the sphere centred about $\mathbf{x}$ with radius $r + (d + r)$ as given by Equation 3.8.

$$C_L = \{l = (\mathbf{l}_a, \mathbf{l}_b) \in L \mid \left| \frac{(\mathbf{l}_b - \mathbf{l}_a) \times (\mathbf{x} - \mathbf{l}_a)}{(\mathbf{l}_b - \mathbf{l}_a)} \right| \leq 2r + d\} \tag{3.8}$$

This subset is somewhat conservative in that it may possibly contain line features which are not in fact the closest feature to any point in the bounding sphere as illustrated by the green line feature in Figure 3.5. However in practice it will still only consist of a fraction of the total line features of L in the majority of scenarios and additionally is computationally cheap to determine.

Thus the process of determining the subset of line features C_L for a grid cell C consists of determining the line feature $l_1 \in L$ which comes within some closest distance d to the center of the cell's bounding box $\mathbf{x}$, and then calculating the subset of line features which intersect the sphere centred at $\mathbf{x}$ and of radius $2r + d$.

Once fully constructed, such a grid can then be utilized in finding nearest line feature $l_1 \in L$ to a specific point $\mathbf{p}$, by simply determining the grid cell C the point resides in and then calculating which of the line features $l_1 \in C_L$ is closest to $\mathbf{p}$. $\mathbf{p}$ can then be projected onto it's closest line feature to form the point $\mathbf{q}$ and the point pair $(\mathbf{p}, \mathbf{q})$ used in updating the estimated transformation as laid out in Algorithm 3.3.

In order to examine how this partitioning grid based approach to the nearest line search compares with performing a simple brute force search, registration evaluations were carried out again using the Freiburg RGB-D datasets [97], and conducted in a similar manner to the evaluations of Section 2.6.3. As would be expected no differences in terms of translational and rotational accuracy were observed between brute force search and the partitioning grid based approach, however there was a significant difference in terms of computational cost. As shown in Figure 3.6 the grid based approach approximately halves the computational cost of each ICP iteration providing a substantial benefit over the brute force approach.

$P = \{\}$ *// initialize set of point pairings*
for all $\mathbf{p} \in E$ **do** *// iterate over all points in source point cloud*
  // find the grid cell containing $\mathbf{p}$
  $C =$ Grid Cell Containing Point $\mathbf{p}$
  $L =$ Line Features Stored by Cell C
  $d = \varepsilon$
  $\mathbf{q} = \mathbf{p}$
  for all $l = (\mathbf{l}_1, \mathbf{l}_2) \in L$ **do** *// iterate over line features of grid cell*
    $t = (\mathbf{p} - \mathbf{l}_1) \cdot \left(\frac{\mathbf{l}_2 - \mathbf{l}_1}{|\mathbf{l}_2 - \mathbf{l}_1|}\right)$
    $t = min(max(t, 0), |\ \mathbf{l}_2 - \mathbf{l}_1\ |)$
    $\mathbf{p}_2 = \mathbf{l}_1 + t\left(\frac{\mathbf{l}_2 - \mathbf{l}_1}{|\mathbf{l}_2 - \mathbf{l}_1|}\right)$ *// project point* $\mathbf{p}$ *onto line feature* l
    $d_2 =|\ \mathbf{p}_2 - \mathbf{p}\ |$ *// calculate distance from line feature*
    if $d_2 < d$ **then**
      $d = d_2$ *// update nearest line feature distance*
      $\mathbf{q} = \mathbf{p}_2$ *// update projected pairing point*
    end if
  end for
  if $\mathbf{p} \neq \mathbf{q}$ **then**
    $P = P \cup \{(\mathbf{p}, \mathbf{q})\}$ *// add point pairing* $(\mathbf{p}, \mathbf{q})$ *to* P
  end if
end for
Return P

Algorithm 3.3: *ICP point pairing generation with line features*

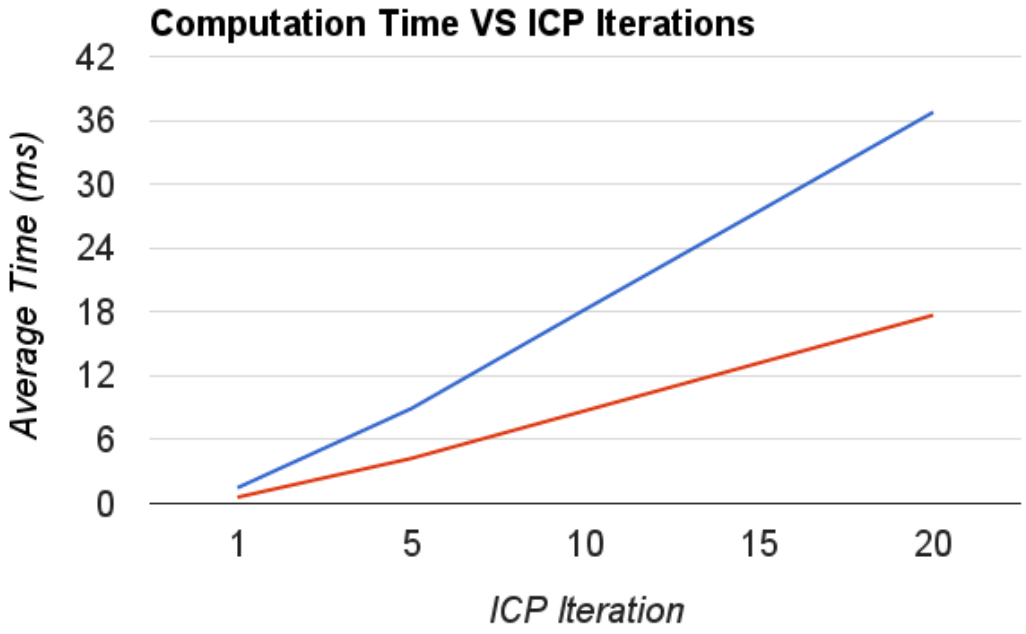


Figure 3.6: *Comparison of computational cost of point cloud to line feature registration when using brute force nearest line search (Red) or the partitioning grid based approach (Blue) as described in Section 3.3.1*

3.3.2 Point to Line feature Pair Selection

Consider a set of line features L extracted from an edge cloud D, it is important to note that in many scenarios the line features of L will not be fully representative of the entire edge cloud. There may be many areas of the cloud from which no accurate line features could be determined, and the points belonging to such areas will in no way be represented in the set of line feature L. Thus the environment/scene from any RGB-D sequence may have many features and objects which are represented in the depth and RGB edge point clouds extracted from the sequence, but which may have no representation in the sets of RGB and depth line features extracted from these edge clouds. We will refer to such edge cloud points as being unrepresented points. Figure 3.7 illustrates this issue of unrepresented points by overlaying extracted line features over the associated edge clouds (both depth and RGB), it is clear that the line features do not encompass all points of the clouds and many features of the scene have no line feature to represent them.

The ICP registration process described in this section estimates the alignment transformation from a set of point pairs. These are generated by projecting each edge cloud point onto its nearest line feature, and then pairing each edge cloud point with its projection. In many situations a significant number of such pairs may involve unrepresented

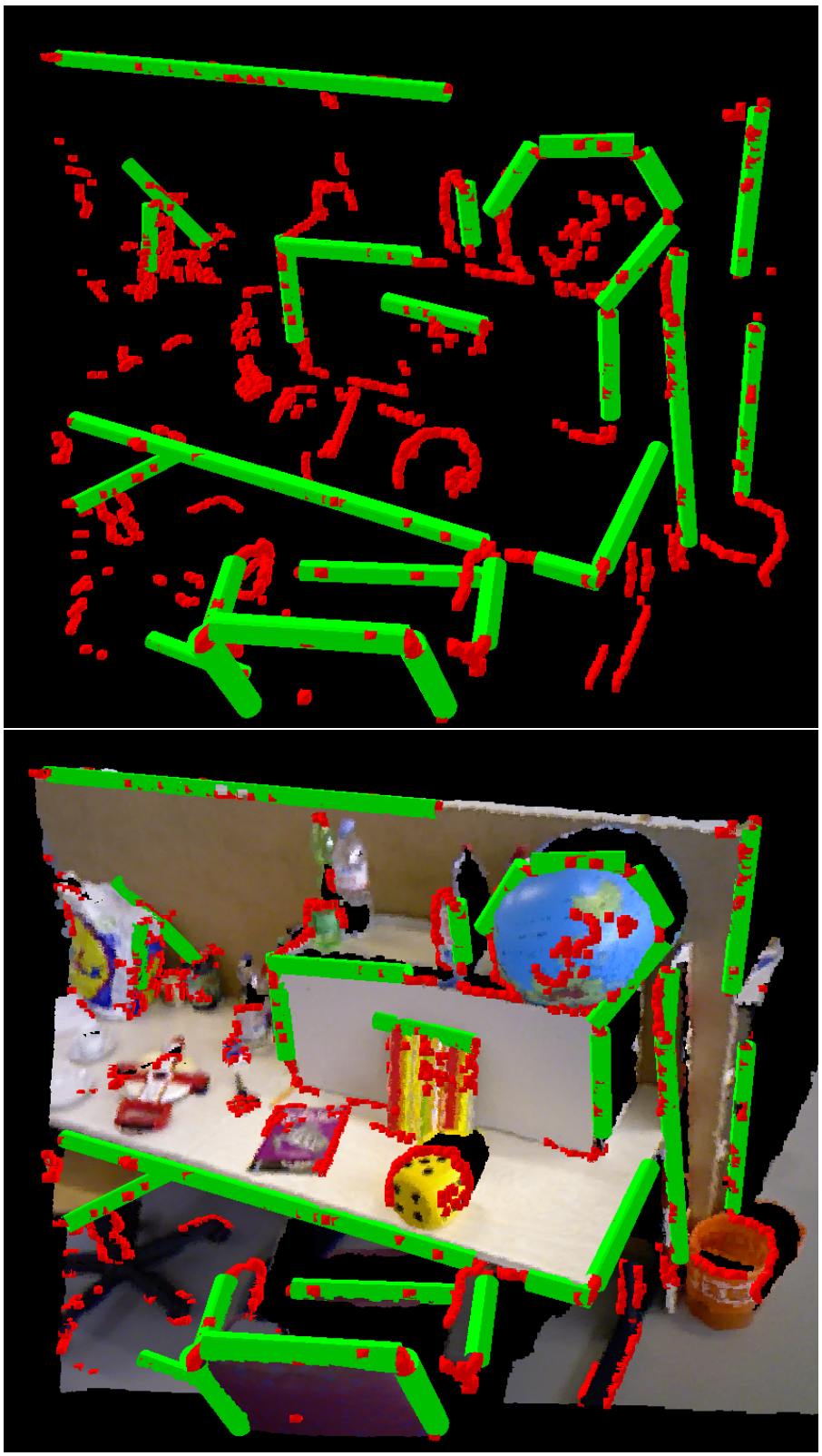


Figure 3.7: *Comparison between Depth and RGB edge clouds and the line features extracted from them.*

points, whose associated edge is not represented in the set of line features being used to generate the pairs. Such point pairs are of course incorrect as they are pairing together points which belong to completely different edges in the environment.

As such at any given iteration of the registration process even if the estimated transformation is in fact in accordance with the true registration transformation, incorrect pairings may still be present and cause the estimated transformation to converge to an incorrect solution. Thus these incorrect pairings only serve to corrupt the registration process, introducing error into the estimated transformation. One method to attempt to alleviate this issue is to be selective in deciding which point pairs should be used to update the estimated transformation each iteration, opposed to simply always using all determined pairings.

Given a pairing $(\mathbf{p}, \mathbf{q})$ between a point $\mathbf{p}$ and its projection onto its closest line feature $\mathbf{q}$, let us refer to distance between said points $\mid \mathbf{p} - \mathbf{q} \mid$ as the pairing distance $D(\mathbf{p}, \mathbf{q})$. If current estimated transformation is in accordance with the true alignment transformation, then points from the source edge cloud whose associated edge is also represented by some line feature will likely be located in close proximity to said line feature. The pairs involving these points will thus in general be associated with small pairing distances. On the other hand, unrepresented edge cloud points will be located at a relatively far distance from any line feature by comparison. The point to line pairings involving these unrepresented points will thus in general also have greater pairing distances associated with them in comparison to the points whose edges are represented in the target set of line features. In general whenever the estimated alignment transformation is comparable to the true alignment transformation, point to line pairings which have a high pairing distance are significantly more likely to be pairings involving unrepresented points, which will introduce error into the estimated transformation.

Thus by examining the distribution of pairing distances for all determined point pairings, and removing those outliers whose pairing distance is significantly greater than the average, it is likely that the majority of pairings involving unrepresented points will be removed. Only using the remaining pairings to update the estimated alignment transformation, will then result in a more accurate final transformation estimation in the majority of circumstances compared to simply using all determined pairings. The practical implementation and evaluation of this concept is now discussed.

Let P denote the set of point pairs determined at any given iteration of the point to line ICP registration. Each element $(\mathbf{p}, \mathbf{q}) \in P$ then has an associated pairing distance (simply the euclidean distance between the paired points) denoted $D(\mathbf{p}, \mathbf{q})$. In order to

be selective in which pairings are used to update the estimated transformation, the mean μ and standard deviation σ of the pairing distances are calculated as given by Equations 3.9 and 3.10 as shown below.

$$\mu = \frac{1}{|P|} \sum_{(\mathbf{p},l)\in P} D(\mathbf{p}, l) \tag{3.9}$$

$$\sigma = \sqrt{\frac{1}{|P|} \sum_{(\mathbf{p},l)\in P} (D(\mathbf{p}, l) - \mu)^2} \tag{3.10}$$

As described previously, point pairings whose associated pairing distance lies significantly above the average are more likely to be pairings involving unrepresented points. Thus all point pairings whose pairing distances are above a threshold Ψ distance give by Equation 3.11 are discarded, and the remainder used to update the estimated transformation as per usual.

$$\Psi = \mu + \alpha\sigma \tag{3.11}$$

The exact value of α used for this threshold was determined experimentally. Figure 3.8 shows an evaluation of the registration process described in this section (utilizing the same evaluation approach used in Section 2.6) using various values of α in order to determine which value to use during standard operation.

Examining these graphs we see that the extreme values of $\alpha = -0.5$ to 0 result in very high registration transformation error (both in the translational and rotational components). This is due to the very large proportion of point pairings rejected each ICP iteration under these values. Such high levels of rejection often results in the remaining point pairs not being sufficient to accurately update the estimated transformation, often resulting in convergence to an incorrect transformation. This issue can be addressed by simply increasing the value of α and hence the number of point pairs used each iteration to update the estimated transformation. This effect can be seen in the sharp decrease in transformation error between the α values of -0.5 to 0.

On the opposite end of the spectrum are the values of $\alpha > 0.5$ in which far fewer

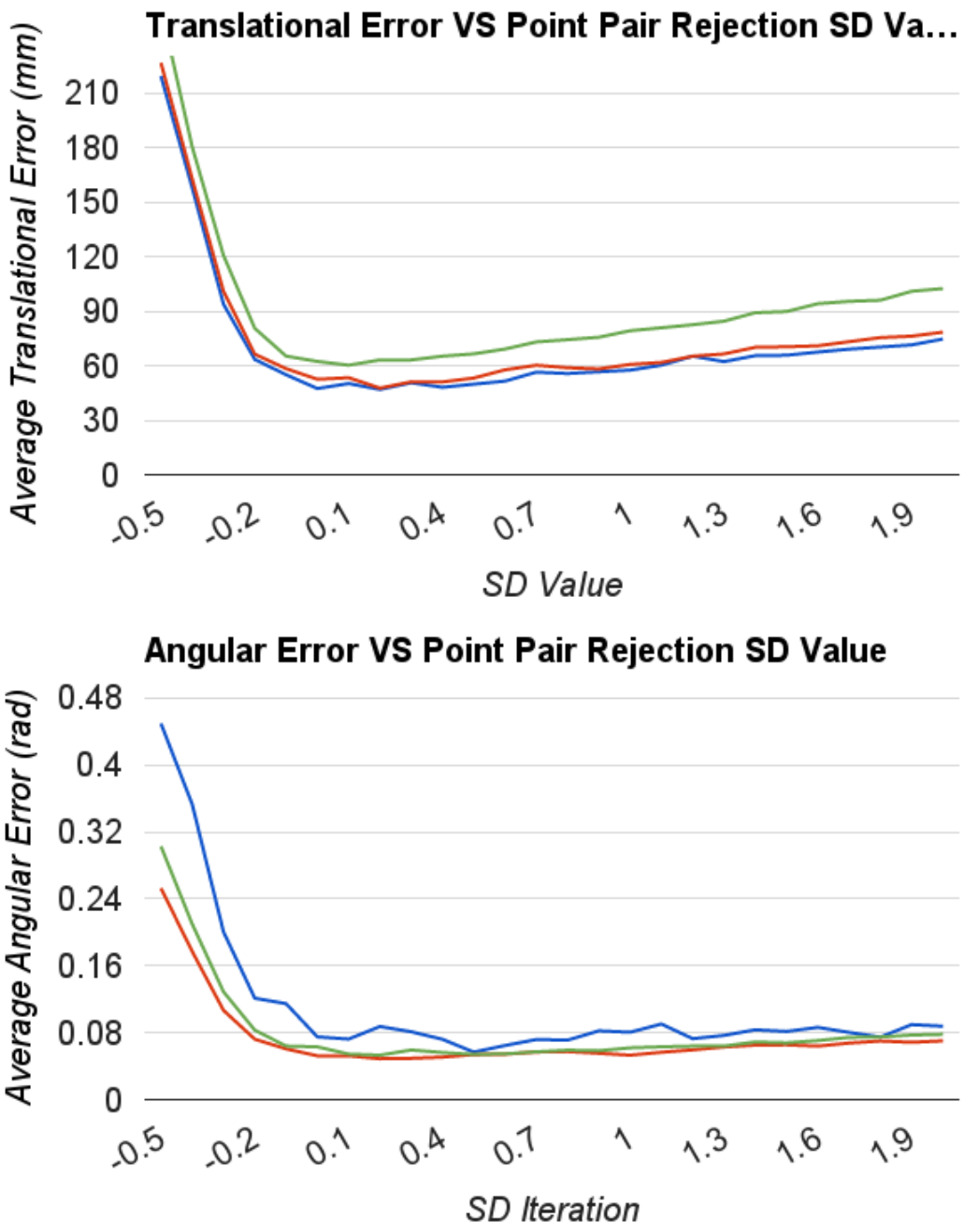


Figure 3.8: *Point to line feature registration accuracy using different standard deviation pair culling values for the datasets FR1 desk(red), FR1 plant(green), FR1 room(blue).*

point pairs are rejected. Thus many of the point pairs used to update the estimated transformation are associated with the unrepresented points previously described, which only serve to introduce error into the process of updating the estimated transformation each ICP iteration. As the value of α continues to increase from 0.5 to 2, a greater proportion of the point pairs used to conduct registration involve such unrepresented points, resulting in an increase in translational and rotational error.

Thus the optimal value of α resulting in the lowest rotational and translational error is seen to lie between these two extremes in the 0 to 0.5 interval. Such values of α provide the best trade-off in rejecting point pairs likely to involve unrepresented points, while at the same time leaving sufficient point pairs remaining such that accurate registration can be achieved in the majority of scenarios. Thus we adopted $\alpha = 0.3$ as the standard parameter value used for this point to line feature based registration.

3.3.3 Evaluation

The same approach used for evaluating the edge cloud registration (as described in Section 2.6.3) was again used in evaluating this proposed point to line ICP registration. This consisted of attempting to register frames of RGB-D data from the Freiburg RGB-D data sequences with transformed copies of themselves, and them evaluating the accuracy and computation time resulting from each such registration. Figure 3.9 shows a sample of evaluation results generated from the FR1 room RGB-D sequence, using a varying number of ICP iterations. Figure 3.10 also provides a comparison between the point to line based ICP registration and the edge cloud ICP registration described previously in Section 2.6.

Once again it can be seen that the average translation and rotational error decreases with the number of ICP iterations, with the rate of improvement in accuracy being greatest at low ICP iteration counts and then steadily decreasing before plateauing at around 10 to 12 ICP iterations. It is clear that the point to line registration approach has a significantly smaller computational cost per ICP iteration than the edge cloud registration approach. When comparing the two registration methods using the same number of ICP iterations for each, point to line registration consistently requires less than half the computation time of the comparative edge cloud registration. However, this significant reduction in computational cost comes at the price of reducing the average registration accuracy. From the plots of Figure 3.10 it can be seen that point to line registration demonstrates a similar level of improvement in registration accuracy per ICP iteration as that shown by edge cloud registration. This improvement in accuracy

continues until the 10-12 ICP iteration mark at which the translation and rotational error of both types of registration levels off with little further improvement with additional ICP iterations. The level of registration error at which these two registration methods plateau at however is significantly different, with point to line ICP registration averaging out with 5cm greater translational error, and two degrees more rotational.

3.4 Results

We now present both an evaluation of this proposed SLAM system along with a set of example result obtained from real-time mapping with a hand held sensor and laptop.

3.4.1 SLAM Evaluation

In this section we present an evaluation of this line based extension to the proposed SLAM systems as laid out in Chapter 2. Once again we use the same hardware, datasets, and evaluation methodology as described previously in 2.9.1.

Table 3.1 compares the results obtained by the line based SLAM of this chapter, to the edge purely edge cloud based approach as presented in Chapter 2, plot of these results are also illustrated in Figure 3.11 . It can be seen that on all datasets the line based approach demonstrated inferior accuracy. This is somewhat to be expected as the linear features used in the line approach are an approximation of the underlying edge clouds. On the positive side, the line based approach demonstrates lower total run times on every RGB-D sequence in comparison to the edge cloud based approach. Again this is to be expected due to the lower computational cost of the edge cloud to line segment registration process that is being utilized as presented in Section 3.3.

In summary the use of such linear features results in a decrease in computational cost, compared to the edge cloud based approach of the previous chapter. However this comes at the price of a significant loss in accuracy. Thus this approach should only be considered in circumstances in which the purely edge cloud based SLAM approach of the previous chapter is to computationally expensive to achieve real-time performance on the target hardware platform.

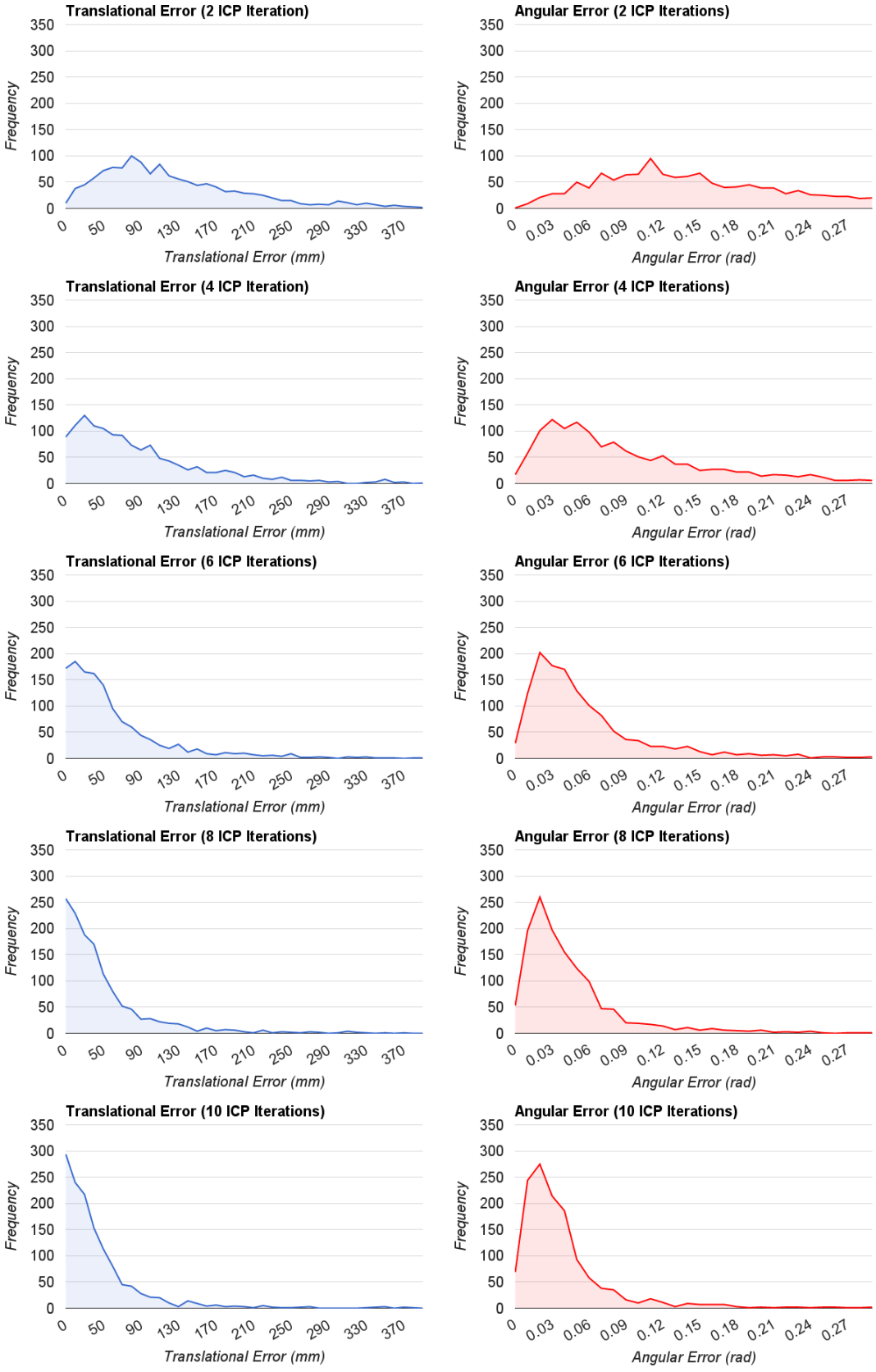


Figure 3.9: *Line feature based ICP registration accuracy histograms*

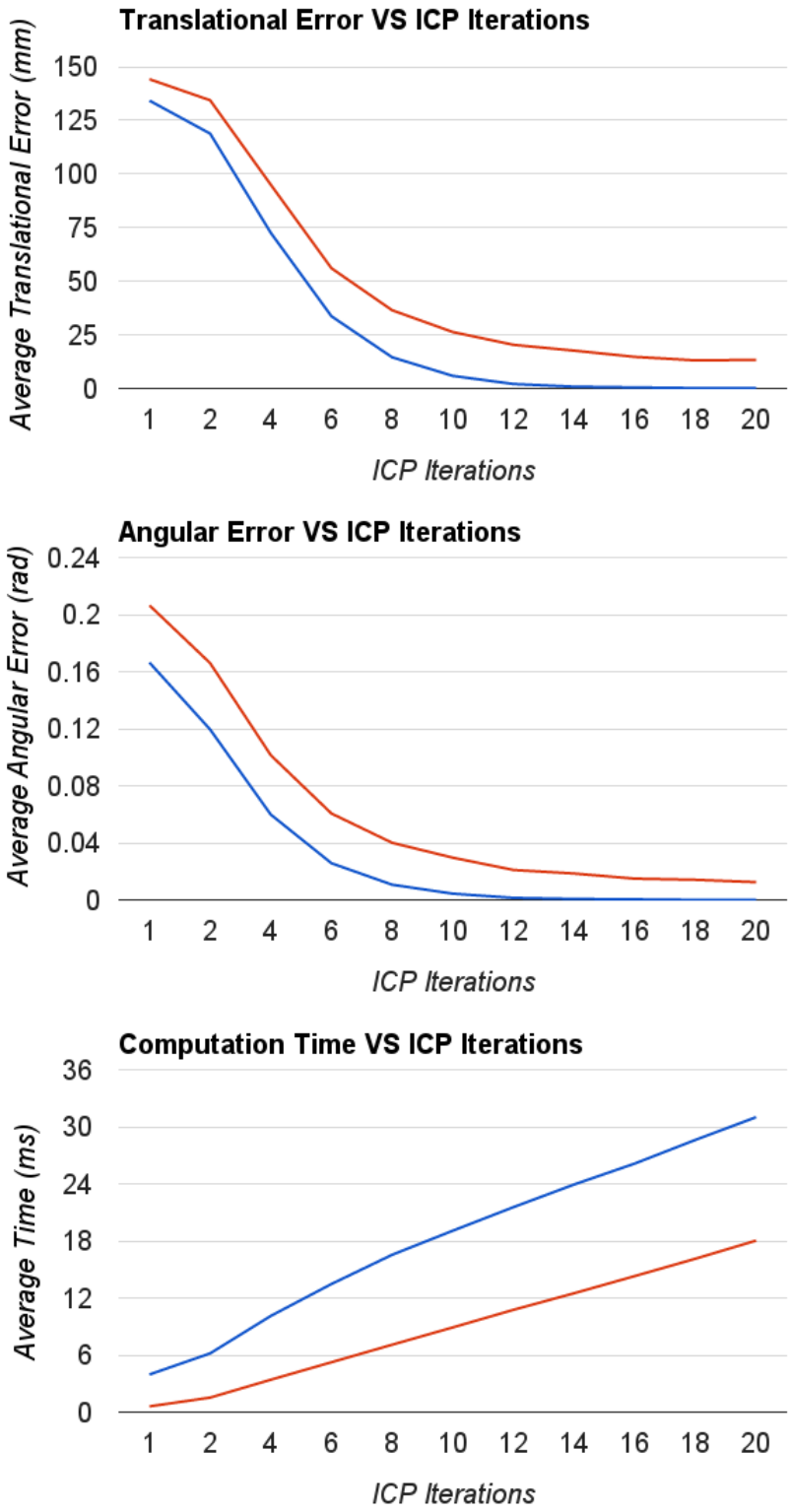


Figure 3.10: *Line Based Registration (Red) VS Edge Point Based Registration (Blue) with RCS = 5*

Table 3.1: *A comparison of the performance of the proposed line based SLAM extension to that of the purely edge cloud based system proposed previously in Chaper 2 using various RGB-D sequences from the Freiburg dataset [96].*

Translational RMSE		
Sequence (length)	**Proposed RGB-D Edge SLAM**	**Proposed RGB-D Line SLAM**
FR1 desk (23 s)	**0.075 m**	0.094 m
FR1 desk2 (25 s)	**0.098 m**	0.106 m
FR1 plant (42 s)	**0.076 m**	0.090 m
FR1 room (49 s)	**0.210 m**	0.232 m
FR1 rpy (28 s)	**0.055 m**	0.063 m
FR1 xyz (30 s)	**0.038 m**	0.048 m

Rotational RMSE		
Sequence (length)	**Proposed RGB-D Edge SLAM**	**Proposed RGB-D Line SLAM**
FR1 desk (23 s)	**3.43 deg**	4.63 deg
FR1 desk2 (25 s)	**3.75 deg**	4.09 deg
FR1 plant (42 s)	**4.09 deg**	5.52 deg
FR1 room (49 s)	**5.66 deg**	7.07 deg
FR1 rpy (28 s)	**4.20 deg**	4.54 deg
FR1 xyz (30 s)	**1.92 deg**	2.23 deg

Total Runtime		
Sequence (length)	**Proposed RGB-D Edge SLAM**	**Proposed RGB-D Line SLAM**
FR1 desk (23 s)	14 s	**10 s**
FR1 desk2 (25 s)	16 s	**11 s**
FR1 plant (42 s)	29 s	**20 s**
FR1 room (49 s)	30 s	**21 s**
FR1 rpy (28 s)	16 s	**11 s**
FR1 xyz (30 s)	17 s	**11 s**

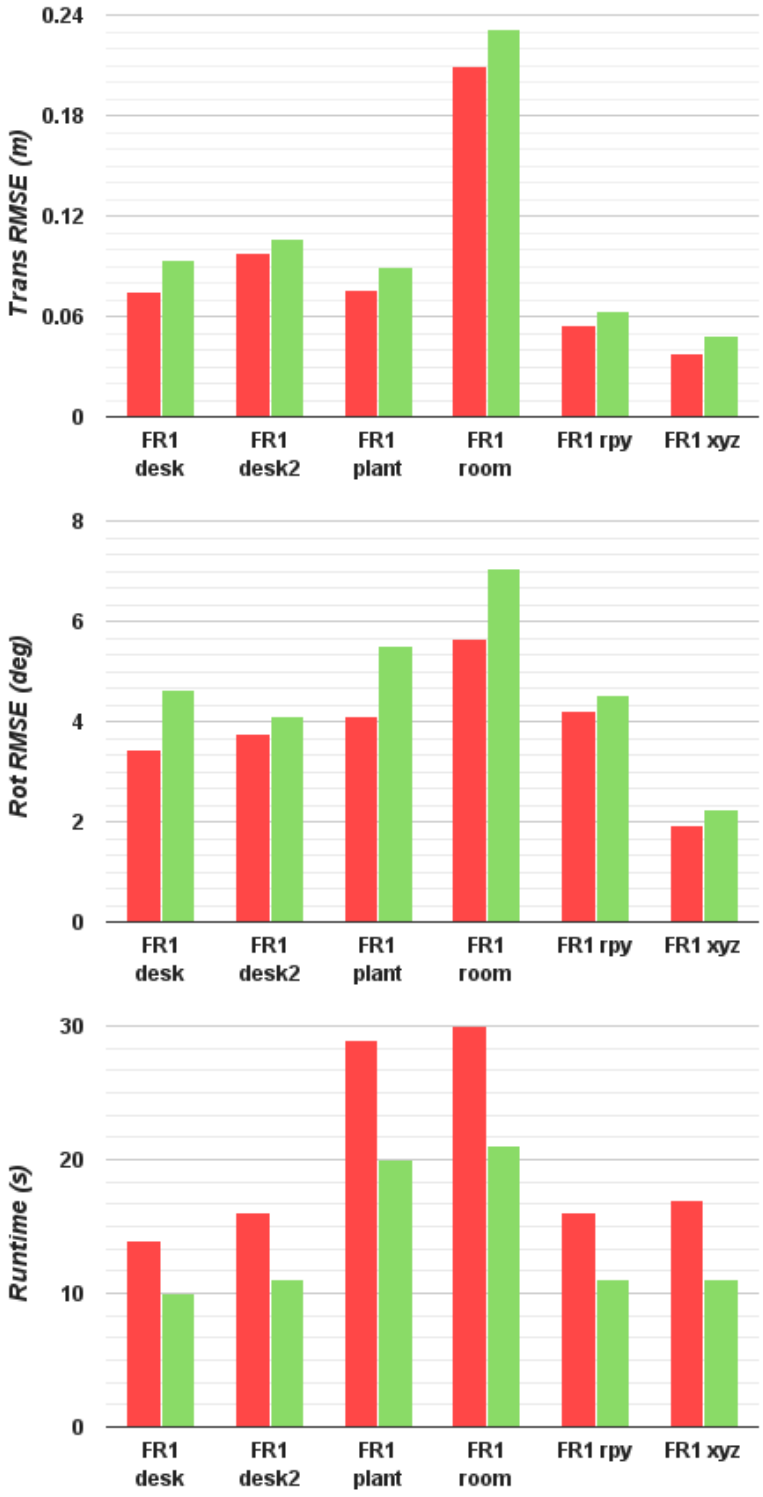


Figure 3.11: *Plots comparing results obtained from the line based SLAM extension of this chapter to the previously proposed edge cloud based approach on a number of Freiburg datasets. The edge based results are shown in red and while line based are shown in green.*

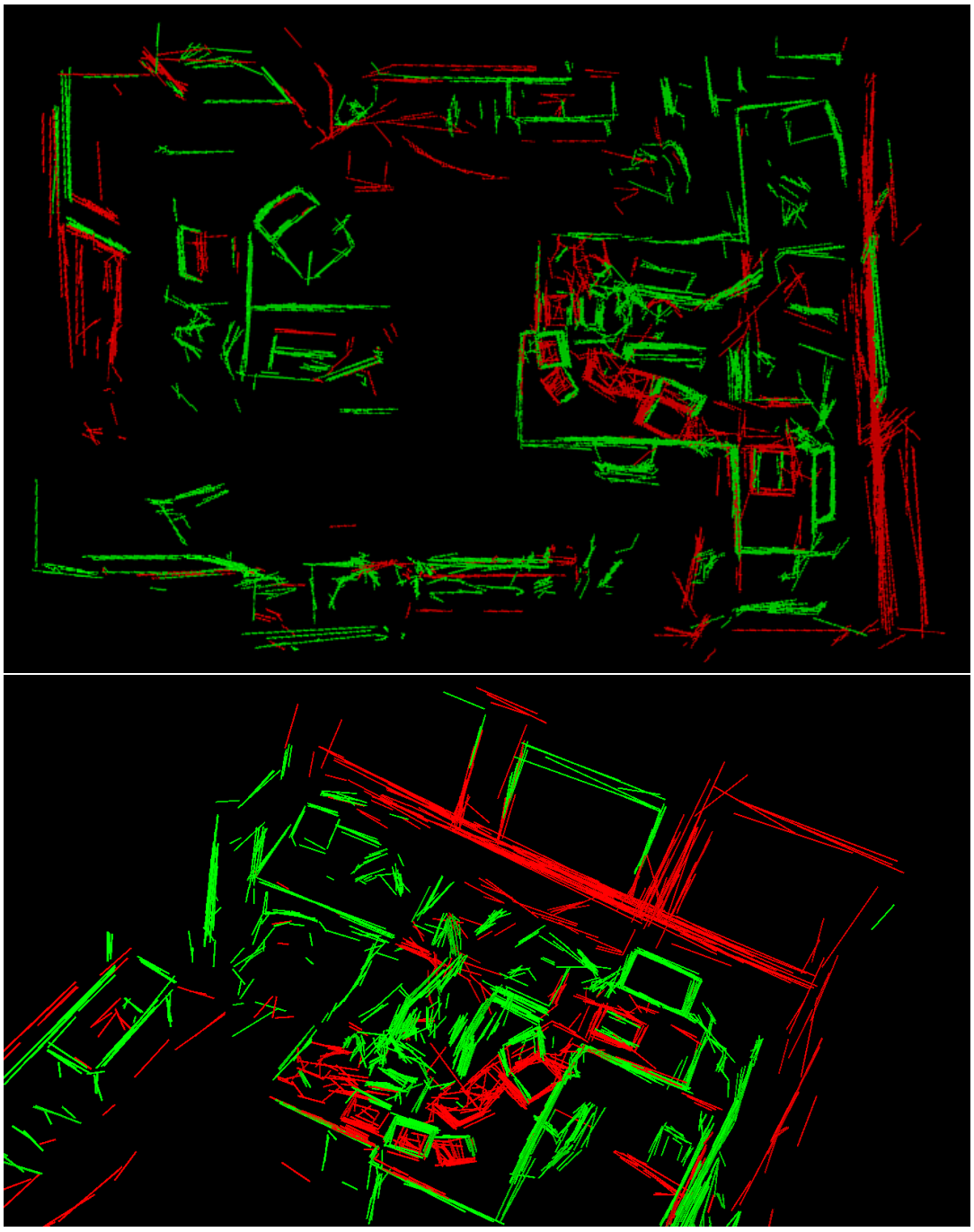


Figure 3.12: *The resulting map from the FR1 room sequence using.*

3.4.2 Further Results

The proposed line based SLAM system was also tested live and recorded dataset without ground-truth for evaluation. This sections presents a selection of maps produced in such scenarios. Again though there is no ground-truth to conduct formal evaluation, visual inspection can give some indication of the system's mapping performance.

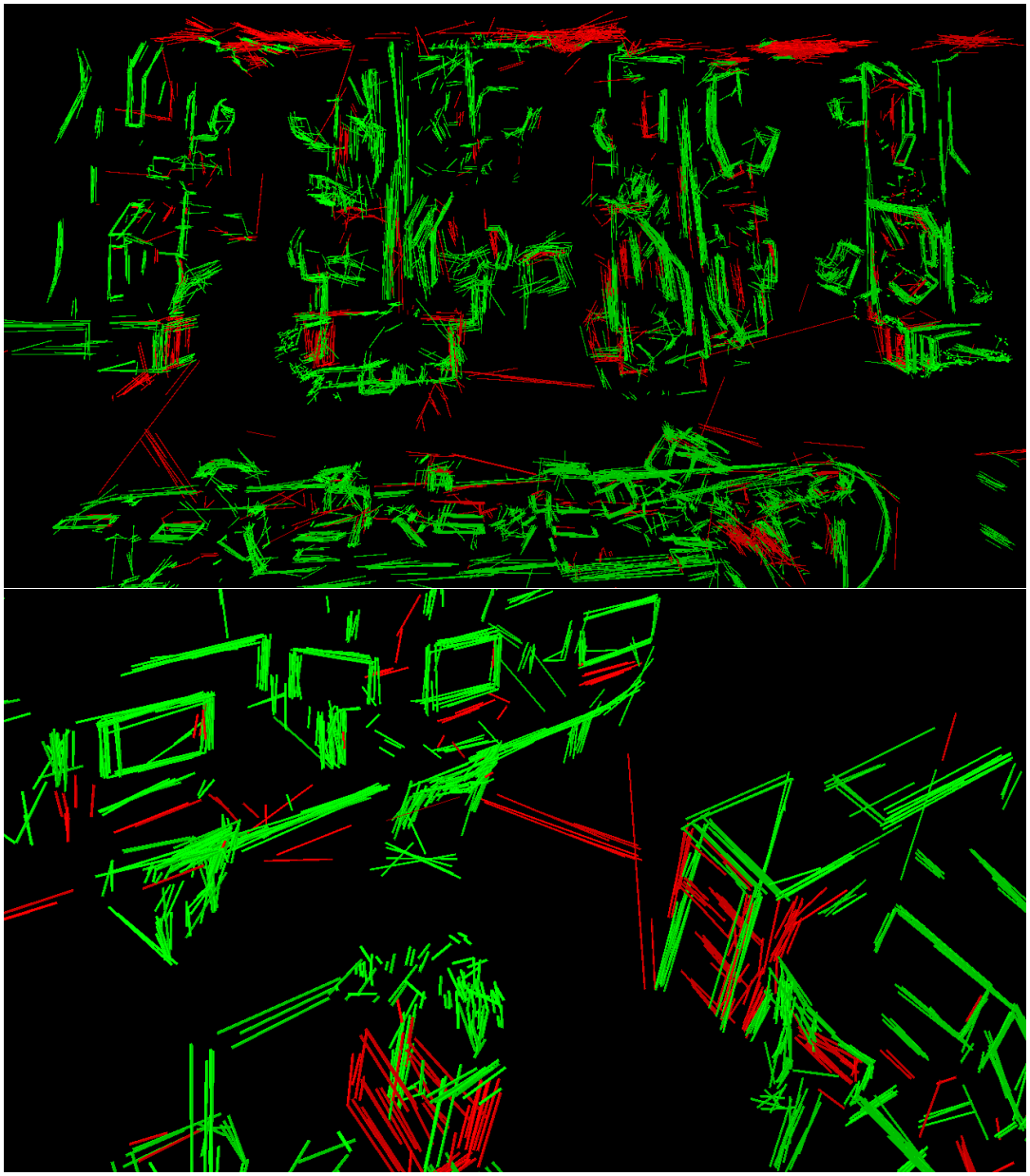


Figure 3.13: *The resulting map from inside an office space.*

3.5 Conclusions

This chapter presented an extension of the edge based SLAM approach presented in Chapter 2, utilizing high level linear features. The motivation behind the use of such features is that they provide a far more compact representation of the semi-dense edge clouds used previously, and in doing so allow for more computationally efficient registration. We presented a split and merge based approach for extracting such features from semi-dense edge clouds, along with our adopted ICP based method of registration between edge clouds and linear feature sets. This was shown to have a lower computational cost to the edge cloud to edge cloud registration proposed in Section 2.6, but also demonstrated an overall decrease in registration accuracy. When evaluated in the same manner as the edge based system of Chapter 2, this line based approach demonstrated a total computational cost around 30% (6.5ms) lower than that of the purely edge cloud based system of the previous chapter. However this came at the cost of translational and rotational accuracy being decreased on average by around 30% (2.5cm) and 20% (0.8 degrees) respectively.

Chapter 4

Belief Space Planning

This chapter investigates how Belief-Space planning could be applied to autonomous navigation for MAVs in cluttered environments, specifically relying on bearing measurements to landmark beacons in the environment for localisation. The following work was originally presented in Bose and Richards [8] at IMAV 2013.

Consider a MAV such as a quad-rotor based platform navigating a pre-mapped indoor environment, utilizing point feature based monocular visual SLAM for localisation. Accurate navigation in such a scenario relies upon the vehicle maintaining localisation. A loss of localisation would result in the vehicle being unable to account for any deviation from the desired trajectory. This would result in the vehicle drifting further away from the desired trajectory over time, potentially colliding with some obstacle within the environment. Such drift can be represented by an increase in uncertainty in the vehicle's estimated state, that is the longer the vehicle has remained without localisation, the further it has potentially drifted from its desired state and hence the greater the uncertainty in its estimated state. In contrast the process of acquiring sensor measurements used to localise and update the vehicle's estimated state correspond to a decrease in state uncertainty.

The set-up presented in the following work is designed to be somewhat analogous to this scenario, with a simulated MAV conducting localisation via bearing measurements to fixed beacons within a cluttered environment. In order to navigate the environment the vehicle must ensure that localisation is maintained at all times if possible. This involves incorporating the concept of state uncertainty into the path planning process, in order to generate vehicle trajectories which attempt to minimize state uncertainty (i.e. maintain localisation) by ensuring that sufficient beacon bearing measurements can

be obtained. Such trajectories are somewhat analogous to those for a vehicle relying on visual point feature SLAM, which ensures that sufficient visual features are always observed for localisation, by both accounting for the location of such visual features within the map and accounting for the pose of the vehicle's sensor itself.

The process of generating trajectories accounting for an agent's state uncertainty (typically attempting to minimize such uncertainty) is commonly referred to as belief space planning. There are many examples of such belief space planning implementations. The Belief Road Map [81] (inspired by the standard Probabilistic Road Map [54]) was used to produce simulated results in which a robot minimizes its state uncertainty during navigation by travelling within close proximity to "beacons" providing information for localization. The Rapidly-exploring Random Belief Trees algorithm [10] iteratively constructs a graph in belief space to determine safe trajectories for the vehicle. Simulated results showed the vehicle's trajectory deviates in order to pass through state measurement areas, reducing the state uncertainty to enable the vehicle to safely pass through narrow passageways that require precise manoeuvring. The Particle RRT algorithm [76] accounts for uncertainty by simulating each tree expansion multiple times with process noise, adding multiple new branches per expansion.

Note that trajectories which minimize the vehicle's state uncertainty typically do not provide the quickest route between the two state, however, they are in a sense the most reliable as are they likely ensure that localization is maintained at all times. Such an uncertainty minimizing trajectory in the proposed set-up may take a long indirect route through the environment in order to ensure sufficient beacon bearing measurements are obtained to keep state uncertainty low. By comparison the fastest direct route may involve the vehicle traversing through areas sparse in measurement beacons, greatly increasing the risk of losing localisation, putting the vehicle at risk. If given a complete or partial map of an environment, what trajectory should the vehicle take between two states in order keep the uncertainty relating to the vehicle's state to a minimum? (or to provide some desired trade-off between minimizing state uncertainty and travel time) This is the question which motivated the remaining work of this chapter.

It should be noted that such belief space planning methods aim to minimize the state uncertainty of the vehicle within a global coordinate frame. For visual SLAM based navigation problems however this aim is often not of much practical use. This is due to the fact that such visual SLAM systems typically estimate the current pose of the sensor relative to the pose of some key-frame or sub map. While the estimated global pose of such key-frames (and hence the sensor) may be highly inaccurate due to accumulated drift, the estimated pose of the sensor relative to the key-frame currently being used for

tracking (and other nearby key-frames) can remain highly accurate. Thus the SLAM system may provide sufficient information to successfully navigate about obstacles within a given proximity to the vehicle's current location, while not necessarily providing an accurate global estimate of the vehicle's pose. Thus high uncertainty in the vehicle's state in a global reference frame does not pose an issue to conducting safe navigation when using such a SLAM system, and thus the trajectories produced by such belief space planning methods do not provide any major benefit. Additionally such belief space planning approaches are typically of very high computational cost compared to analogous standard planning methods. For an autonomous flying vehicle with limited on-board computational power, such belief space methods may take an unacceptable length of time to produce valid trajectories for the vehicle, severely hampering navigation. For these reason the work described in this chapter was not used in practical experiments and was purely conducted in simulation. However this investigation into the viability of such methods prompted the work presented in the following Chapter 5, in which we propose a planning approach accounting for the formerly described issues, and which effectively supersede the work of this chapter.

4.1 Planner Overview

The belief space planning algorithm presented in this chapter operates within a simulation set-up designed to approximate that of a MAV, navigating though a mapped environment with sparse visual features akin to using a monocular visual SLAM system. These environments consist of a set of obstacles in the form of polygon meshes, and a set of so called "beacons" taken to represent visual landmark or regions rich in salient visual features. The vehicle is able to take relative bearing measurements to any such beacon, provided it is both within the field of view of the vehicle's camera sensor, and not occluded by any obstacle. Naturally the vehicle must avoid collision with any of the obstacles present.

An Octree is used to specially partition the environment based upon the position of the obstacles present. The leaf nodes of this Octree are then used to determine the nodes of a navigation graph. Edges are then added to this graph connecting any two nodes that are within line of sight of one another, and whose associated Octree nodes are also adjacent. This step involves determining if two points are in direct line of sight of one another by checking that the line segment connecting them does not intersect any of the polygonal obstacles. The computational cost of this step is reduced by using the partitioning Octree to quickly determine a subset of obstacles with which a connecting

line segment could potentially collide, and then only conducting the intersection tests against this subset of obstacles.

The planner itself recursively explores this navigation graph from a starting node, incrementally constructing different paths across the graph. The evolution of the vehicle's estimated state will vary across each of these paths, and determines which paths are preferable over others according to the desired trade off between minimizing distance travelled and state uncertainty. For each trajectory, a set of particles (each representing a potential vehicle state) is used to represent an approximation of the probability distribution for the vehicle's state. This set of particles is determined from the beacon measurements that the vehicle would observe along that specific trajectory, in a similar way to how particle filter localization is typically performed (as described in [29], [101], [102]). This set of particles captures the growth of uncertainty in the vehicle's state as it moves through the environment, and the subsequent uncertainty reduction due to localization from bearing measurements obtained from beacons. Each trajectory is evaluated based upon both its length and the associated evolution of the vehicle's estimated state. Trajectories that display the desirable trade-off between minimizing length and uncertainty are extended to generate further trajectories through the graph. Other trajectories with undesirable characteristics are discarded to avoid wasting computation time.

The adopted formulation of this belief space planning problem is presented in the following section.

4.2 Problem Formulation

Let the simulated environment consist of a set of obstacles O and measurement beacon positions $B \subset \mathbb{R}^3$. Each obstacle in $O_i \in O$ is in the form of a convex polygonal body such that $O_i = \{V_i, \hat{E}_i, \hat{F}_i\}$, where V_i is the set of vertices of the body's mesh, $\hat{E}_i$ the set of unit vectors aligned with each of the body's edges, and $\hat{F}_i$ the set of unit vector normals for each of the bodies faces. Note that non-convex obstacles are represented in this set-up by simply decomposing them into multiple convex obstacles. Each $\mathbf{b}_i \in B$ is the position of a measurement beacon.

The simulated MAV is taken to be equipped with a forward facing sensor which can take bearing measurements to a beacon $\mathbf{b}_i \in B$ provided that $\mathbf{b}_i$ is both within direct line of sight, and within the sensor's field of view. It is assumed that the MAV has a complete map of the environment (i.e. both O and B), and that there is no data association

ambiguity for any bearing measurements taken. Thus the vehicle can conduct belief space planning account for where the beacons of B are observably, and the location of the obstacles O within the map.

A navigation graph across the environment is constructed as described in Section 4.3 consisting of a set of nodes N, and a set of undirected edges E dictating how these nodes are connected (of the form $e_i = (\mathbf{n}_i, \mathbf{n}_j) \in E$). A graph node is also placed at the initial position of the vehicle denoted $\mathbf{n}_{start}$, and at the desired goal location $\mathbf{n}_{goal}$.

A set of "path nodes" P is used to store different paths generated through this graph, with each $P_i \in P$ consisting of a graph node $\mathbf{n}_i \in N$, a parent path node $Q_i \in P$, and estimation or "belief" of the vehicles state $\mathbf{x}_i$. For simplicity we use $\mathbf{m}_i \in N$ to denote the graph node associated with the parent path node Q_i (i.e. $\mathbf{m}_i = \mathbf{n}_{Q_i}$).

Each such path node P_i represents a short path between the nodes $\mathbf{n}_i$ and $\mathbf{m}_i$ of the navigation graph, with $\mathbf{x}_i$ representing an estimation of the vehicle's state upon traversing said path. Specifically $\mathbf{x}_i$ consists of a set of particles, each a potential state for the vehicle, together which form a discrete approximation of the probability distribution for the vehicle's estimated state. Such a representation of vehicle belief is standard in Monte Carlo (particle filtering) localisation methods as seen in [101], [29], [102]. Further as will be shown $\mathbf{x}_i$ is generated from the estimated vehicle state associated with the parent path node Q_i. Finally each path node P_i also stores a "total path length" l_i, and a weighting score w_i used in evaluating how desirable a path segment is relative to others. Thus a path node P_i consists of the components shown in 4.1.

$$P_i = \{\mathbf{n}_i, \mathbf{m}_i, Q_i, \mathbf{x}_i, l_i, w_i\} \tag{4.1}$$

Initially P contains a single parent-less path node P_{start} located at the $\mathbf{n}_{start}$ graph node, and with a belief reflecting the initial belief of the vehicle's state. This node is then used in the generation of additional path nodes, expanding to different nodes of the navigation graph. This process is then repeated iteratively, taking existing path nodes of P and using them to generate additional path nodes. Note that every path node P_i can be traced back to P_{start} by recursively iterating through the parents of P_i, and thus in this way each P_i represents some path through the navigation graph between the nodes $\mathbf{n}_{start}$ and $\mathbf{n}_i$. The path length l_i represents the length of this path between $\mathbf{n}_{start}$ and $\mathbf{n}_i$. Thus this process of iteratively generating path nodes can be viewed as expanding a tree of paths through the navigation graph, rooted at $\mathbf{n}_{start}$.

The weighting w_i of a path node P_i is based both off its total path length l_i, and measure of the uncertainty related to its estimation of the vehicle's state $\mathbf{x}_i$ calculated from the sample variance of the set of particles. These two factors can be weighted differently based on the desired type of path (for example minimum distance paths would have zero weighting on uncertainty).

4.3 Graph Construction

This section describes the algorithm used to construct the navigation graph on which the belief space planner operates.

4.3.1 Determining Graph Vertices

The space of environment is first partitioned via the construction of an Octree, initialized so that the root node encompasses the entire volume in which planning is to be conducted. Each Octree node is then recursively subdivided into eight equally sized octants provided the node fulfils a specific splitting criteria listed in Algorithm 4.1. This produces a partition as shown in Figure 4.1 in which Octree nodes are densely placed about object edges, but sparse near flat surfaces. During construction it is also determined which of the obstacles from O each Octree node intersects. This intersection information is stored for each node for later use in determining the edges of the navigation graph.

After the construction of this partitioning Octree is complete, navigation graph nodes are placed at the center positions of all Octree nodes (provided this center point does not lie within one of the obstacles of O). Similar to the navigation graph construction approaches desribed in [43], [50], [63], [41], this process results in an efficient node layout for path planning, with sparse nodes in large open areas and dense nodes close to obstacles edges, allowing more precise navigation about said obstacles. Note that the maximum node depth of the Octree in Algorithm 4.1 determines the density at which graph nodes V are placed about the edges of obstacles, and important shortcuts past certain obstacles (such as narrow passageways) may not be represented in the final navigation graph if this maximum node depth is too low.

Additional graph nodes are then placed at the start and goal positions of the specific

path planning problem that needs to be solved.

$$N = N \cup \{\mathbf{n}_{start}, \mathbf{n}_{goal}\} \tag{4.2}$$

```
if node depth < max node depth then
    for each o_i ∈ O do
        if node intersects an edge of o_i then
            Return True
        end if
    end for
end if
Return False
```

Algorithm 4.1: *Splitting Criteria for an Octree node*

4.3.2 Determining Graph Edges

It is assumed that if any two graph nodes $\mathbf{n}_A, \mathbf{n}_B \in N$ are within direct line of sight, then a direct collision free trajectory between them exists along the line connecting them. This can be represented in the graph by the addition of an edge $(\mathbf{n}_A, \mathbf{n}_B) \in E$, however simply connecting every possible pair of graph nodes in this way would lead to a vast amount of edges, many of which provide no great benefit to navigation but would substantially increase the computational cost of the planning process. Due to this edges are only added between pairs of graph nodes $\mathbf{n}_A, \mathbf{n}_B \in N$, if they are both within line of sight of one another, and their associated Octree nodes are adjacent. This results in a graph of the form as shown in Figure 4.2.

Determining if two positions $n_A, \mathbf{n}_B$ are within line of sight involves checking that the line segment connecting the two does not intersect any obstacle in O. A naive approach to performing this check would simply involve checking for intersection between this line and each obstacle of O sequentially until an intersection is detected or all obstacles been checked. However this method results in the number of intersection checks required being proportional to the number of obstacles present, potentially becoming a limiting factor in computation time if a large number of obstacles are present.

As mentioned previously each Octree node stores which obstacles of O intersect with its associate cubic volume. It is a simple task to determine the set of leaf Octree nodes

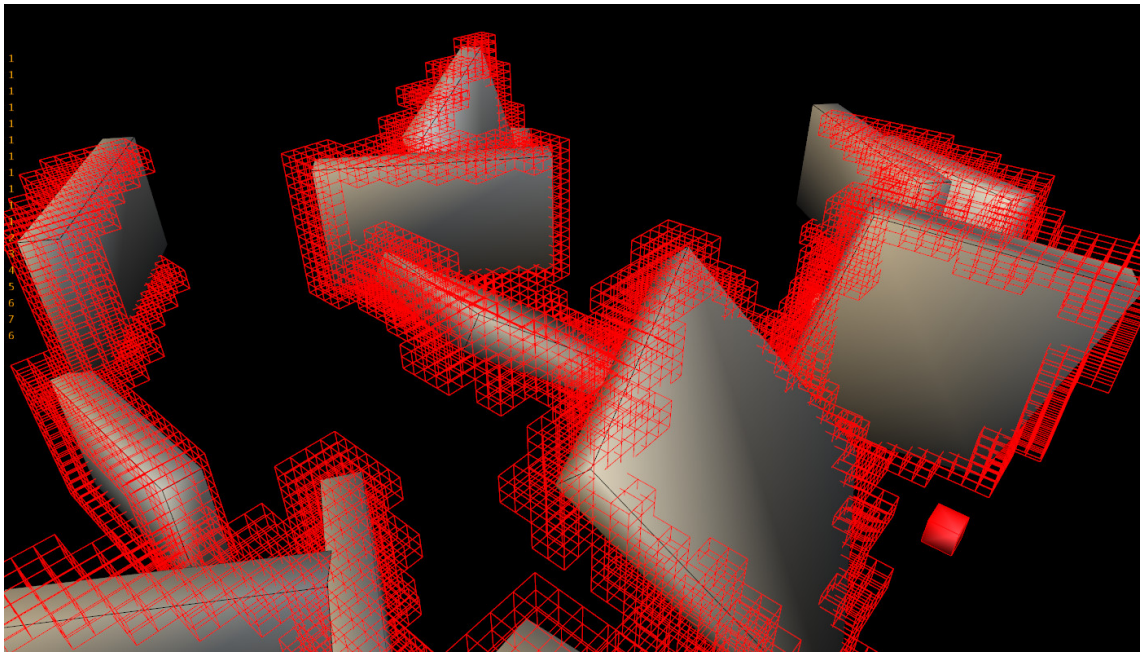


Figure 4.1: *Deepest nodes in a partitioning Octree showing how the splitting method biases the node density about obstacle edges.*

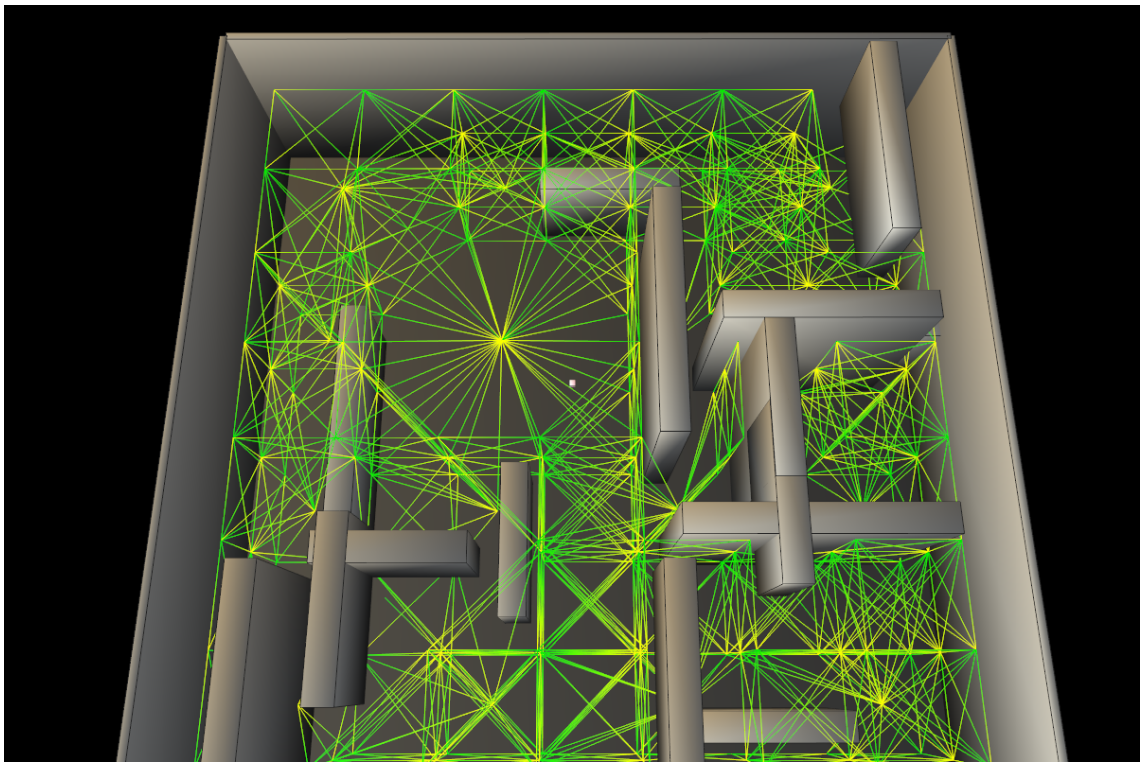


Figure 4.2: *Example of a graph constructed by the process described in Section 4.3. The partitioning Octree constructed with a low max node depth for clarity.*

that a line segment intersects. Naturally such a line segment can only possibly intersect those obstacles of O which also intersect with atleast one of said Octree nodes. This allows us to quickly determine a subset of obstacles which a line segment may potentially collide with and avoid the need to check for intersection with every obstacles in O, saving significant computational work.

4.3.3 Intersection Checks

When constructing the partitioning Octree as described in Section 4.3.1 it is necessary to check for intersection between the convex cuboid nodes of the Octree and the convex obstacles of O. The method used to conduct intersection checks between convex bodies makes use of the separating axis theorem [33], which simply states that if a plane can be placed between two convex objects such that each is fully contained on a different side of that plane then the objects are not intersecting. A unit vector $\hat{\mathbf{n}}$ for which there exists some separating plane of normal $\hat{\mathbf{n}}$, is called a separating axis. The process used here to determine if two convex obstacles are intersecting involves checking a specific set of unit vectors (dependent on the geometry of the obstacles) to see if atleast one is a separating axis.

In order to determine if a specific unit vector $\hat{\mathbf{n}}$ is a separating axis for two convex objects A and B, the projection of each of the convex objects onto the axis $\hat{\mathbf{n}}$ is calculated. This forms two projection intervals, one for each object denoted $I_A = [A_0, A_1]$ and $I_B = [B_0, B_1]$. If these two intervals do not overlap then it can be seen that $\hat{\mathbf{n}}$ is a separating axis for the objects, and hence they do not intersect. The analogous 2D equivalent of this concept is Illustrated in Figure 4.3. The case in which the intervals $[A_0, A_1]$ and $[B_0, B_1]$ do not intersect, indicating direction $\hat{\mathbf{n}}$ is a separating axis for objects A and B is shown in Figure 4.3a, and conversely Figure 4.3b shows the case where $\hat{\mathbf{n}}$ is found not to be a separating axis.

The interval formed from projecting a convex polygon mesh $o_i \in O$ onto a direction $\hat{\mathbf{n}}$ is determined by calculating the dot product of $\hat{\mathbf{n}}$ with each of the mesh's vertices $\mathbf{v} \in V_i$. This then forms the set $D = \{\mathbf{v} \bullet \hat{\mathbf{n}} \mid \mathbf{v} \in V_i\}$ and the interval formed by the projection is simply $[min(D), max(D)]$. If the convex mesh consists of a single edge (the object is a line segment) between two points $\mathbf{p}_1$ and $\mathbf{p}_2$ then $D = \{\hat{\mathbf{n}} \bullet \mathbf{p}_1, \hat{\mathbf{n}} \bullet \mathbf{p}_2\}$.

In order to fully determine if two convex meshes intersect, a number of directions must be checked to see if any would provide a valid separating axis. Let o_i and o_j be two convex polygon meshes in the same formats as the obstacles of O. The set of unit vector

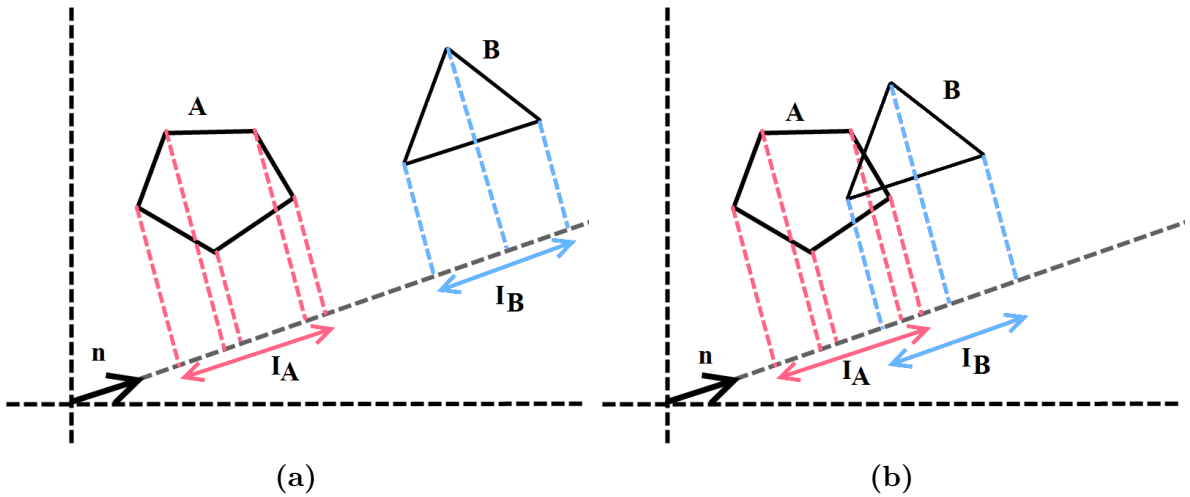


Figure 4.3: *Separating axis example cases*

directions $\hat{D}$ that need to be checked are thus given in Equation 4.3.

$$\hat{D} = \{\hat{F}_i \cup \hat{F}_j \cup \{\hat{\mathbf{a}} \times \hat{\mathbf{b}} \mid \hat{\mathbf{a}} \in \hat{E}_i, \hat{\mathbf{b}} \in \hat{E}_j\}\} \tag{4.3}$$

$\hat{D}$ is comprised of the face normals for both meshes, along with all possible directions formed by the cross product between elements of $\hat{E}_i$ and $\hat{E}_j$ the edge unit vector directions of the meshes. Without checking directions formed by such cross products, separating axes such as those shown in Figure 4.4 would not be accounted for. Each unit vector of $\hat{D}$ can be checked in turn to see if provides a separating axis. As soon as any separating axis is found, the meshes have been determined to not be intersecting, and thus it is not necessary to check any other directions.

Note if a specific direction $\hat{\mathbf{n}}$ has been checked to determine whether or not it is a separating axis, it is unnecessary to check any other direction parallel to $\hat{\mathbf{n}}$. For meshes such as those of the cuboid Octree nodes there will be faces whose normal direction is parallel to that of another face and edges parallel to others edges. It is important therefore to keep track of what directions have already been checked in order to avoid unnecessarily checking a direction parallel to one previous, potentially wasting computation time.

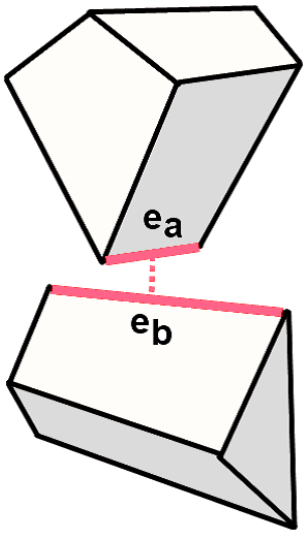


Figure 4.4: *Example of a separating axis formed from the cross product of edges e_a and e_b*

4.4 Belief Space Planner

Trajectories that feature high levels of uncertainty in the vehicle's estimated state are generally undesirable as the vehicle may stray off the desired trajectory, potentially colliding with some obstacle. However trajectories that minimize the uncertainty related to the vehicle's estimated state may also be undesirably lengthy in comparison. The belief space planning problem thus involves finding a trajectory between an initial position $\mathbf{n}_{start}$ and destination $\mathbf{n}_{goal}$ which provides some desired trade-off between minimizing distance travelled and maintaining low state uncertainty. The belief space planning method described in this section searches for paths through the previously constructed navigation graph which provide such a desired trade-off.

4.4.1 Algorithm

The planner searches for paths through the visibility graph constructed in Section 4.3, using a set of path nodes P (as described in section 4.2) to store the different paths generated thus far. This set of path nodes P is initialized by setting $P = \{P_{start}\}$. The initial path node P_{start} is located at the starting node of the navigation graph $\mathbf{n}_{start}$, and has a set of particles $\mathbf{x}_{start}$ approximating the initial belief of the vehicle's state. Being the path node from which all others will originate from, the parent of this initial path node is set to $Q_{start} = P_{start}$.

The planner works in an iterative manner, using path nodes from a subset of nodes to

update $U \subseteq P$ to generate new path nodes. Such generation of a new path node P_{new} involves taking an existing path node $P_i \in U$, and extending its existing path along some edge $e \in E$ which is connected to $\mathbf{n}_i$, that is $e = (\mathbf{n}_{new} \in N, \mathbf{n}_i \in N)$ or $e = (\mathbf{n}_i, \mathbf{n}_{new})$. This new path node will then have $\mathbf{n}_{new}$ as its associated navigation graph node, and $Q_{new} = P_i$ as its parent path node. In order to determine the set of particles for this new node P_{new} (representing its estimation of the vehicle's state upon reaching the graph node $\mathbf{n}_{new}$), a numerical particle filter localization simulation is conducted. This examines the sensor measurements the vehicle would observe while travelling from $\mathbf{n}_i$ to $\mathbf{n}_{new}$ along the edge e, and uses the set of particles of the parent path node $\mathbf{x}_i$ as the initial estimation of the vehicle's state. The simulation in question follows the standard steps involved in particle filter localization. Each iteration of the simulation process noise is applied to the set of particles (in accordance with the latest control inputs for the vehicle to follow the desired path), measurements to the beacons of B are then determined, and particle re-sampling is conducted based on how each particle's expected sensor measurements deviate from those that would actually be obtained. The resulting set of particles from this simulation upon reaching $\mathbf{n}_{new}$ is then used for $\mathbf{x}_{new}$ the new path node estimation of the vehicle's state. This new node P_{new} is then assigned a weighting score w_{new} based on the total path length of its complete path, and a measure of uncertainty related to $\mathbf{x}_{new}$. Each of these two factors can be weighted differently depending on the desired type of path.

A full outline of the planning algorithm is listed in Algorithm 4.2. The algorithm attempts to creates new paths through the graph by taking a path node $P_i \in U$, and then attempting to extend P_i's complete path from $\mathbf{n}_i$ to an additional navigation graph vertex $\mathbf{n}_{new}$ which is connected to $\mathbf{n}_i$ by some edge $(\mathbf{n}_{new}, \mathbf{n}_i) \in E$ or $(\mathbf{n}_i, \mathbf{n}_{new}) \in E$. If this new path node is deemed to provide an acceptable trade-off between path length and vehicle state uncertainty, and is not inferior in both respects to some other existing path node, it is added to the set of path nodes P, and the set of path nodes used for further path generation U. This process of attempting to generate a new path node from $\mathbf{n}_i \in U$ is attempted for each navigation graph node connected to $\mathbf{n}_i$ by some edge e, after which P_i is removed from U as all possible extensions of its associated path have been attempted. Once U no longer contains any path nodes there are no more potential paths to investigate and the algorithm terminates.

The outline Algorithm 4.2 involves a number of functions described in the following.

```
while
      do
    P_i =minimum scoring element of U
   for each n_new ∈ N | (n_i, n_new) ∈ E ∨ (n_new, n_i) ∈ E do
       P_new = Propagate(P_i, n_new)
       w_new = Assignweighting(P_new)
       if !(∃P_j ∈ P | (w_j < w_new ∧ n_j = n_new)) then
           P = P ∪ P_new
           Insert(P_new)
       end if
   end for
   U = U \ P_i
end while
```

Algorithm 4.2: *Planning method*

4.4.1.1 Propagate(P_i,$\mathbf{n}_j$)

The $Propagate(P_i, \mathbf{n})$ function takes a path node $P_i \in P$ and a graph vertex $\mathbf{n}_j \in N$ and returns a new path node P_{new} such that $\mathbf{n}_{new} = \mathbf{n}_j$. The function carries out a numerical simulation of the vehicle travelling along a straight path between $\mathbf{n}_i$ and $\mathbf{n}_j$ using a particle filter for localisation, taking $\mathbf{x}_i$ as the initial set of particles. At each iteration in this simulation, the subset of beacons in B that are both within line of sight of the MAV, and which can be brought within the MAV's limited field of view sensor are determined. These beacons are then examined to evaluate which would provide a bearing measurement resulting in the greatest decrease in state uncertainty. This is dependent both on the beacon's location, distance from the vehicle, and also the distribution of the current set of particles. The MAV is then made to face this beacon of greatest uncertainty reductions. The set of particles resulting from this simulation represent the belief of the vehicle's state after following the complete path of q, and are thus used for the new nodes belief $\mathbf{x}_{new}$.

4.4.1.2 Assignweighting(P_i)

This function assigns a weighting to a path node P_i based on both its path length l_i, and on evaluating the uncertainty associated of its set of particle states $\mathbf{x}_i$. This uncertainty evaluation simply sums the x,y,z sample variances in the positions of the particle of $\mathbf{x}_i$, the result of which we denote by $u_i \in \mathbb{R}$. The path nodes weighting is then assigned as

$$w_i = l_i\alpha + u_i\sigma \tag{4.4}$$

where $\alpha \in \mathbb{R}$ and $\sigma \in \mathbb{R}$ are constants that can be adjusted depending upon the desired type of path. For example setting $\sigma = 0$ would assign weightings based purely on path length, resulting in the planner attempting to produce minimum distance paths. On the other hand, setting σ to a value much greater than α results in the planner producing paths which attempt to maintain much lower state uncertainty by ensuring many informative measurements are taken to the localisation beacons of B.

4.4.1.3 Insert(P_i)

The function $Insert(P_i)$ inserts a path node $P_i \in P$ into the list of path nodes to update U. The position at which P_i is added to U is determined by its assigned weighting w_i such that U maintains a list of path nodes ordered by their weightings. By simply choosing the last element of U for expansion low weighted path nodes are expanded first. If we were instead not to order U, or simply select which path node to expand at random, it would lead to an extremely large number of paths being generated, all of which, are far inferior to the best existing paths which will at some point be identified later. This would result in a great amount of wasted computation creating and examining path which are extremely unlikely to provide the best desired trade-off.

4.5 Results

A number of trajectories produced by the planner in different environments are now presented (each environment is fully enclosed, however, the roof of each is not drawn). The uncertainty of the MAV's estimated state is visualized by error ellipsoids formed from the set of particle states of the path nodes as discussed in Section 4.2. In each set-up, robust paths produced by the planner when attempting to maintain low uncertainty (drawn in orange using weighting constants $\alpha = 0.01$, $\sigma = 1$) are compared with the paths produced when only minimizing distance (drawn in blue using weighting constants $\alpha = 1$, $\sigma = 0$). Localisation beacons are drawn as red markers and measurements taken of them at points along a path are indicated by lines and vision cones.

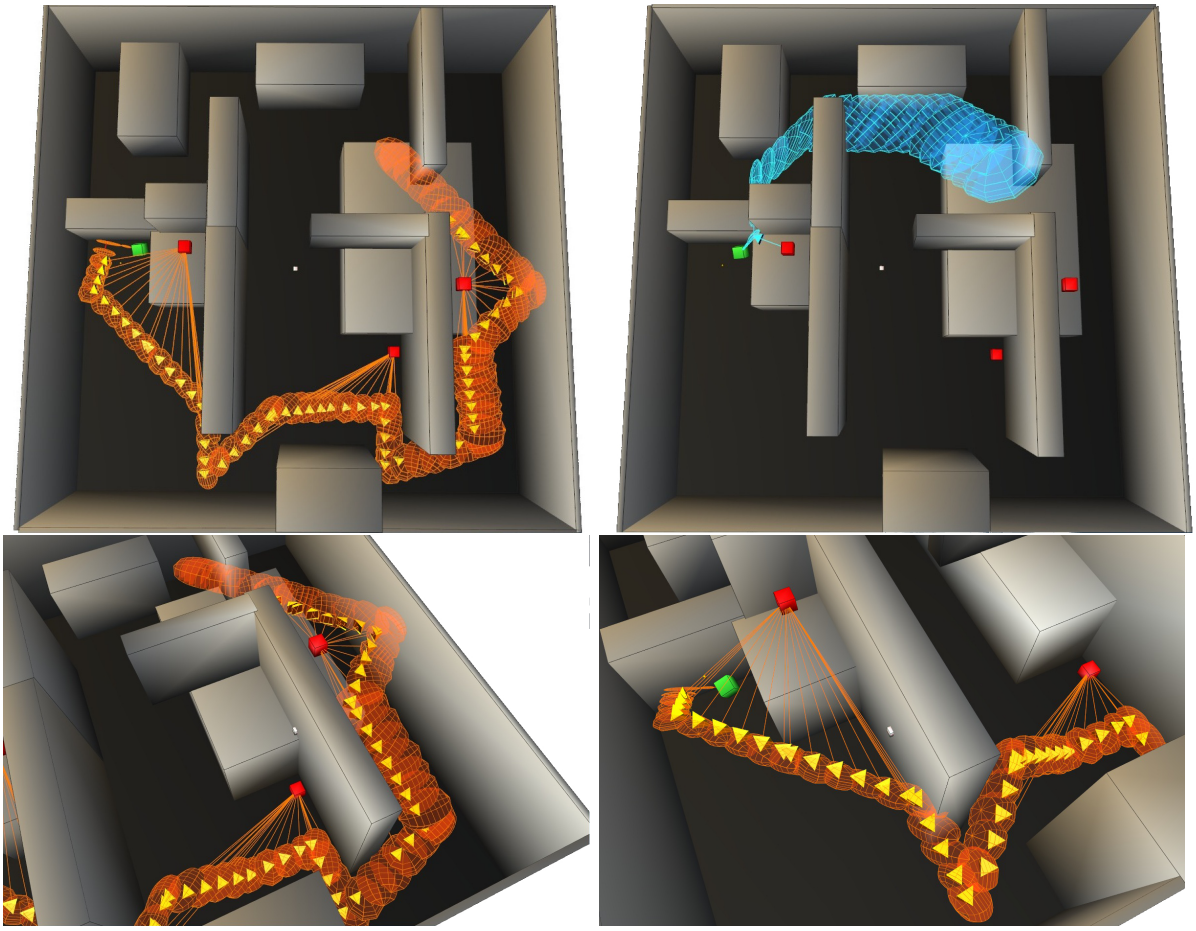


Figure 4.5: *Top: a comparison of trajectories minimizing state uncertainty (orange) and distance travelled (blue). Bottom : other views of the same uncertainty minimizing trajectory.*

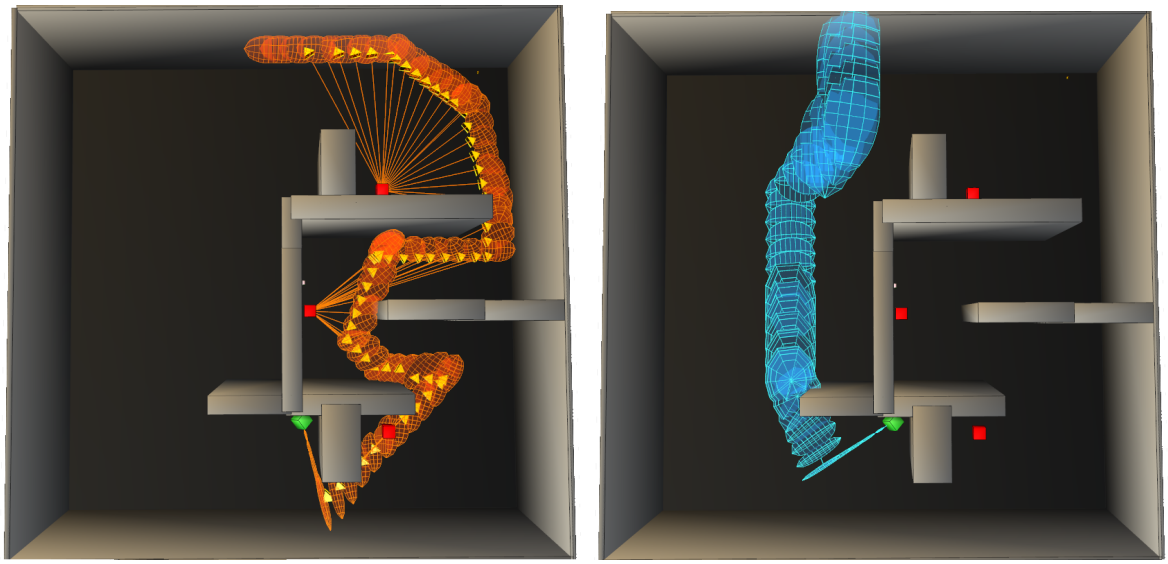


Figure 4.6: *Example comparison between robust and minimum distance paths.*

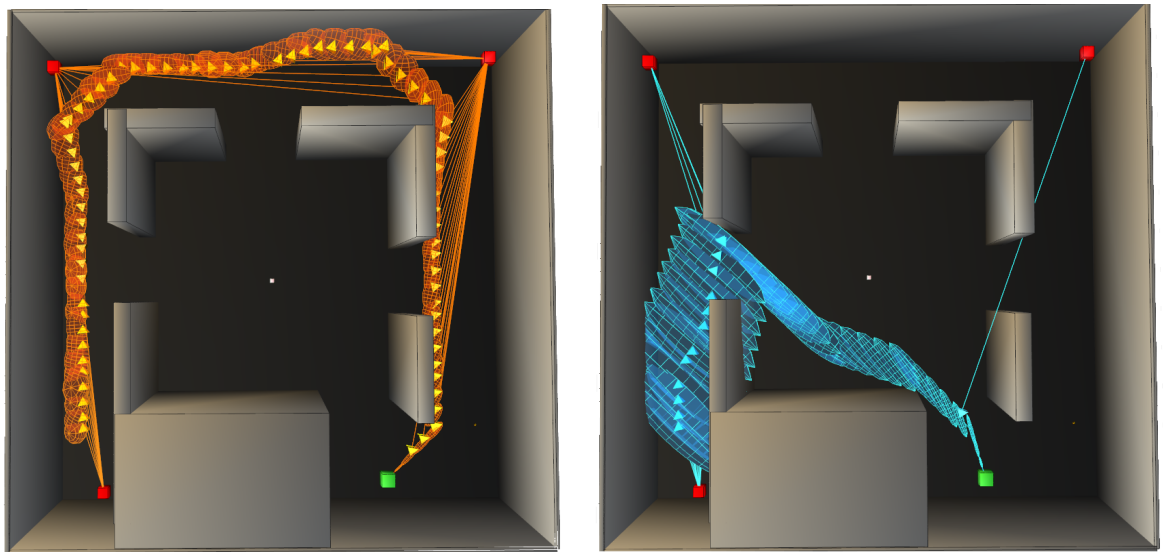


Figure 4.7: *Further example comparison between robust and minimum distance paths.*

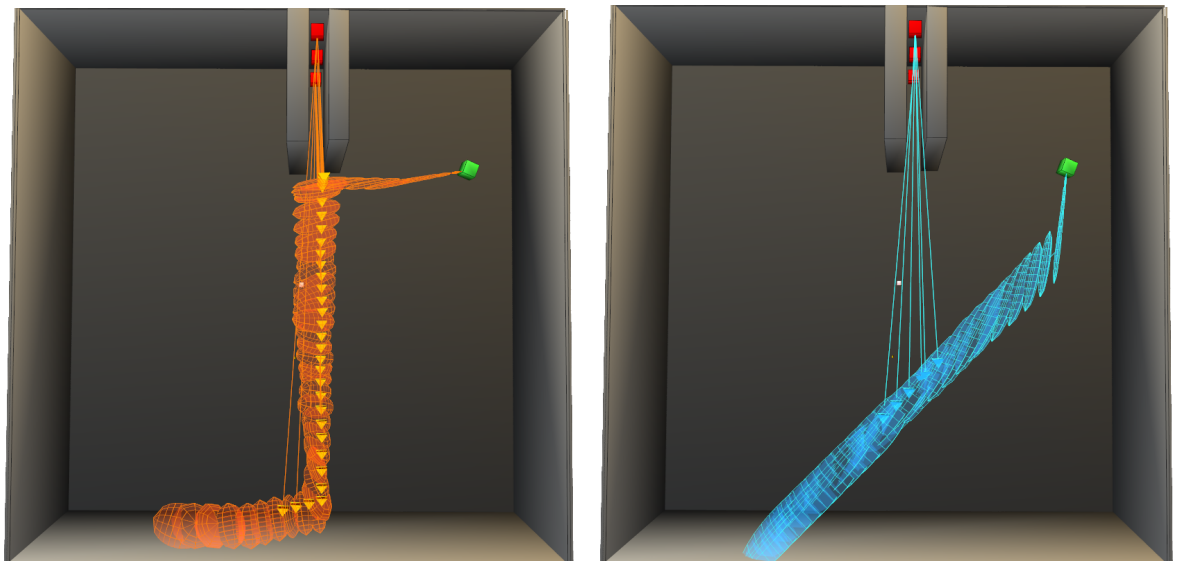


Figure 4.8: *Robust and minimum distance paths generated in space with limited visibilty to beacons.*

Table 4.1: *Path computation times and properties*

	Robust			Minimum Distance		
Figure	**Time**	**Path Length**	**End State Uncertainty**	**Time**	**Path Length**	**End State Uncertainty**
4.5	6.14 s	687	18.8	0.93 s	321	125.4
4.6	2.88 s	656	32.2	0.45 s	325	173.9
4.7	5.41 s	707	21.3	1.13 s	356	267.4
4.8	1.18 s	418	46.7	0.22 s	271	63.8

An example of a path generated by the planning algorithm between two points in a complex environment is shown Figure 4.5. The minimum uncertainty path is seen to take a route which enables it to obtain numerous beacon measurements, resulting in better localisation by the internal particle filter compared to the minimum distance path.

Examples of the planner operating in simpler environments are shown in Figures 4.6, 4.7, 4.8.

Figures 4.6 shows a path produced by the planner which attempts to maintain good localisation by choosing a long winding corridor with beacons rather than a direct route through empty space. Right by comparison shows minimum distance path.

Figure 4.7 highlights how the minimum distance path may risk collisions, as its uncertainty ellipsoid cuts deep into the wall near the destination. The robust path by comparison takes a long indirect route to the destination, but which always keeps the MAV in line of sight of a measurement beacon keeping its state uncertainty low.

In Figure 4.8 the environment features several localisation beacons are placed at the end of a narrow corridor. The MAV is thus only able to observe them when it aligns itself with the corridor, bringing them within clear line of sight of its sensor. The robust path is seen to remain in line with the corridor as long as possible before moving towards the destination target, thus minimizing the time in which no beacon measurements are available for localisation.

Table 4.1 shows the computation times and properties of the robust paths and shortest distance paths. As is to be expected the uncertainty measure of the robust path is seen to always be significantly lower than that of the shortest distance path. This indicates that in every scenario the MAV is more likely to successfully follow the robust path over the shortest distance path.

4.6 Conclusions

This section presented a belief space planning algorithm targeted towards navigation for indoor MAVs using monocular SLAM / localisation approaches. The planner generates trajectories for a simulated vehicle which is able to take bearing measurements to fixed beacons in the environment, used for localisation and estimation of the vehicle's own state. A navigation graph is constructed based on an Octree partitioning of the environment, followed by the generation of a tree of trajectories through said graph. The vehicle is simulated traversing said trajectories while also attempting to localise from bearing measurements to the beacons within the environment, this allows the evolution of the uncertainty of the vehicle's estimated state to be evaluated for each such trajectory. The tree generation process culls trajectories that either do not provide the desired trade-off between minimizing state uncertainty and trajectory length, or that are deemed inferior by these measures to other existing trajectories. The planner is demonstrated in a number of scenarios, and for each results from the planner are compared with the minimum distance trajectories through the constructed navigation graphs. As expected the planner is able to produce paths which involve greatly reduced state estimation uncertainty compared to minimum distance paths. Such "robust" low uncertainty paths are far safer for the vehicle as localisation is required to ensure obstacle avoidance.

Chapter 5

Path Planning

This Chapter presents our proposed path planning approach for indoor MAV systems utilizing the edge based SLAM presented previously in Chapter 2 to conduct navigation. A background overview of path planning methods is given before the outline of our keyframe centric planning approach, accounting for both collision avoidance and sensor information constraints to produce safe trajectories for the vehicle in question.

5.1 Background

5.1.1 Overview

Path planning problems regard determining how to transition between two states or "configurations"[65] for a given system. Depending upon the problem in question, such configurations may represent anything from the joint angles of a robotic manipulator to the 6DOF pose of a rigid flying vehicle. In the majority of such path planning problems however, many such configurations will be invalid or unreachable. Take for example a robot attempting to navigate a cluttered environment. In such a scenario the robot cannot simply pass through obstacles, and thus configurations resulting in the robot colliding with obstacles in the environment are deemed invalid. By then restricting robot navigation to only valid configurations such obstacle collisions are avoided.

This subset of valid configurations for a given path planning problem is referred to as the "configuration space" C. The solution to a path planning problem is then a continuous path (or "trajectory") across C which connects two desired valid configurations, typically referred to as "start" $\mathbf{c}_S \in C$ and "goal" $\mathbf{c}_G \in C$. Formally such a trajectory $t =$

$\{\mathbf{c}_S, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, ...\mathbf{c}_G | \mathbf{c}_i \in C\}$ is simply a sequence of configurations that the system (be it a robotic manipulator, wheeled vehicle etc) passes through in order to reach the desired goal configuration $\mathbf{c}_G$.

Classes of path planning problems can vary widely in terms of a number of factors such as configuration space dimensionality, obstacle field complexity, differential constraints/vehicle dynamics, and the quality of trajectories required. As such a great range of path planning algorithms and approaches have been developed, typically each targeted towards a specific class of planning problem exhibiting certain factors. Thus such problem factors must be carefully considered when determining what form of planning algorithm would best fit a specific planning problem. Some common concepts regarding such algorithms are now briefly discussed.

5.1.1.1 Completeness

One of the major concepts concerning path planning algorithms is that of completeness, which refers to a planning algorithm's ability to guarantee that a trajectory between $\mathbf{c}_S \in C$ and $\mathbf{c}_G \in C$ will be found given that one actually exists. Many algorithms are not in fact complete, instead employing various assumptions and approximations that in the majority of scenarios will result in a valid trajectory being found more quickly, at the cost of sacrificing completeness.

5.1.1.2 Optimality

Another major concept is that of trajectory optimality, that is that some possible trajectories are deemed preferable, or of higher quality than others, and whether or not a specific planning algorithm is able to determine the best possible trajectory between $\mathbf{c}_S \in C$ and $\mathbf{c}_G \in C$. Commonly this notion of trajectory quality is evaluated using a cost function F_{cost} to assign a cost to each potential trajectory, that is the cost of a trajectory t is given by $F_{cost}(t) \in \mathbb{R}$.

An optimal trajectory t_{opt} is one which minimizes (or maximizes depending on the cost function definition) said cost function i.e.

$$t_{opt} \in \{t : \underset{t}{argmin}(F_{cost}(t))\} \tag{5.1}$$

In many scenarios this cost function F_{cost} is simply the total trajectory length, with optimal trajectories then simply being the shortest possible trajectory between the start and goal configurations. However, many other problem specific variables such as fuel burn, sensor observations and information gain can be incorporated into the cost function, in order to tailor optimal trajectories to the specific planning problem.

5.1.1.3 Single-Query vs Multi-Query Algorithms

Path planning algorithms can for the most part be split into two categories, multi-query and single-query planners. Multi-query planners are able to re-use previous computation to aid in determining trajectories between a new set of start and goal configurations $\mathbf{c}_S \in C$, $\mathbf{c}_G \in C$.

Many multi-query planning algorithms initially compute an approximate representation of the configuration space C, which can then be used to determine trajectories between many different start and goal configurations $\mathbf{c}_S$, $\mathbf{c}_G$. The well known visibility graph algorithm [71] is a good example of one such planner, in that the navigation graph it initially constructs can be used multiple times for solving different planning problems across the same configuration space C. However multi-query planners that follow this approach typically operate under the assumption that the configuration space C remains fixed, which in many situations is not true (such as moving obstacles being present) restricting such planners to only certain planning problems.

On the other hand single-query planners such as the RRT algorithm discussed in Section 5.1.5, do not leverage any previous computation instead having to solve each planning problem separately from scratch.

5.1.1.4 Holonomic Vs Non-Holonomic Agents

Any vehicle / robot can be categorized as either being Holonomic and Non-Holonomic depending upon what movements it may make, and what degree of direct control it has upon its own configuration. If the number of directly controllable degrees of freedom is equal to the total number of degrees of freedom, then the robot has complete control over its configuration and thus is described as Holonomic. An example of this would be a robotic manipulator arm, formed of a set of motor controlled joints, where each joint angle can be controlled independently of the others. Conversely if not all degrees of freedom are controllable then the robot is described as being Non-Holonomic. One Non-Holonomic example would be a car like robot whose configuration consists three degrees

of freedom (its 2D coordinates and yaw angle) but which only has direct control of two degree of freedom (that being its acceleration and the angle of its front wheels).

Non-Holonomic robots require that such constraints are accounted for during path planning such that the trajectories produced may actually be followed by the robot in question. For our purposes regarding indoor MAV planning described later in Sections 5.2 and 5.3, similar to [35] we make the assumption that the vehicle only ever experiences small roll and pitch angles and that it has no direct control over such angles. These assumptions reduce the vehicle's configuration to 4D consisting of x,y,z,yaw. Each of these degrees of freedom are directly controllable and thus the vehicle maybe described as Holonomic under these assumptions.

5.1.2 Graph Based Planning

One of the most common path planning approaches involves generating a graph within the configuration space C, whose nodes consist of valid configurations and whose edges consist of direct trajectories between such nodes (i.e. trajectories that simply interpolate from one configuration to another). Once such a graph has been constructed a graph search algorithm such as A* [37] can be used to find the shortest path through the graph between two connected nodes. Note algorithms employing such an approach are considered multi-query as the constructed graph can be reused multiple times to find different trajectories across C.

The widely used visibility graph algorithm [71],[108] is one example of such an approach, typically requiring that the obstacles of C are in the form of a set of polygons (polyhedrons in a three dimensional C). This is due to the method of graph generation used, which places nodes at obstacle vertices and connects those which are in direct line of sight; resulting in a graph containing the shortest path in C between any two of its nodes (hence the visibility graph algorithm is optimal). The cost of generating such a graph however can increase exponentially with the number (and complexity) of obstacles present in C, this combined with the polygonal requirements, rule out the use of visibility graphs in many scenarios.

Grid based graph construction is a common alternative in such situations where there exists a large number of obstacles, or where strictly optimal paths are not required. This involves simply overlaying a uniform grid over the configuration space C, examining the configuration lying at the center of each grid cell, and placing nodes at those which are not lying within any obstacle. Finally edges are placed connecting nodes of adjacent

grid cells completing the graph. This approach has the benefit of being agnostic to what form the obstacles of C take, simply requiring the ability to check if a specific point (configuration) is contained within an obstacles. As resolution of the uniform grid used increases so too does the density of nodes in the constructed graph. This in turn increases the quality of the paths through the graph itself and the existence of paths within narrow obstacle fields. Intuitively as the size of the grid cells tends toward being infinitesimally small path through the graph tend towards optimality and the nodes of the graph tends towards being a direct copy of C itself. In this way such approaches are described as resolution complete and resolution optimal, however such high grid resolution are in practice not viable.

5.1.3 Stochastic Sampling Based Planners

Many planning algorithms scale very poorly to problems involving high dimension configuration spaces, particularly those algorithms which require the construction of an explicit (if approximate) representation of the entire configuration space C. A configuration space's size explodes with increasing dimensionality and as such, so does both the time and memory needed to compute and store such an explicit representation of C. This is most clearly demonstrated by grid based planning methods, where the number of grid cells needed to represent a space increases exponentially with dimensionality, making such algorithms unsuitable for problem of dimension 4 or above.

One way to deal with the issues caused by higher dimensionality is to simply do away with the notion of explicitly representing the configuration space C all together. This is the approach adopted by so called stochastic sampling based planning methods as catalogued in [52],[32],[54],[66]. Such methods instead work by iteratively sampling random configurations, rejecting those not within the configuration space C, and constructing some form of navigation graph from the remaining valid samples. This graph is then typically augmented with nodes and edges for the desired start and goal configurations $\mathbf{c}_S \in C$ and $\mathbf{c}_G \in C$.

Note however, it is not guaranteed that the constructed graph will contain a valid trajectory between $\mathbf{c}_S \in C$ and $\mathbf{c}_G \in C$, and thus such algorithms cannot be described as being complete. However, the proportion of the configuration space C spanned by the constructed graph increases with the number of randomly sampled configurations. Intuitively this means that the probability that a trajectory is found between $\mathbf{c}_S \in C$ and $\mathbf{c}_G \in C$ increases with the number of sampled configurations. Further it can be proven that the probability of finding a trajectory (given one exists) approaches one, as

the number of samples taken approaches infinity. Thus such sampling based planning algorithms are often described as being "probabilistically complete".

A number of such stochastic sampling based algorithms are now briefly described in the following sub sections.

5.1.4 Probabilistic Road-Maps (PRM)

Probabilistic road maps or PRMs [54], [92] are one form of stochastic sampling based planning, which function in a similar way to the visibility graph algorithm [71],[108]. Randomly sampled valid configurations from C are used to form the nodes of a navigation graph. Edges are added between pairs of nodes that are both within close proximity to one another, and which can be connected by a straight line across the configuration space C. The resulting navigation graph is then augmented by adding nodes (and appropriate edges) at start and goal configurations $\mathbf{c}_S$ and $\mathbf{c}_G$. A standard graph search algorithm such as A* [37] can then be used to determine trajectories between the nodes of this navigation graph.

The proportion of the configuration space covered by the PRM's navigation graph increases with the number configurations sampled, meaning that the algorithm is probabilistically complete. Additionally the quality of trajectories that can be found also increases with the number of configurations sampled in the graph's creation, as this leads to an increased density of nodes in the navigation graph. In this way the PRM algorithm is described as probabilistically optimal, that is, as the number of configurations sampled approaches infinity, the paths through that graph trend towards being optimal. The PRM is also a multi query algorithm due to the fact that the constructed navigation graph can be reused to solve multiple different planning problems.

5.1.5 Rapidly Exploring Random Trees (RRT)

Another common stochastic sampling based planning algorithm, the RRT algorithm [66],[67] iteratively generates a tree across the configuration space C beginning with a root node located at $\mathbf{c}_S$. Each such tree node holds a specific configuration from C, and has a single parent tree node representing the previous configuration from which that node originated. Additionally, every tree node may have multiple child nodes, each representing a potential configuration that may be reached from the current node in question. Due to the fact that every tree node can be traced back to the root node ($\mathbf{c}_S$)

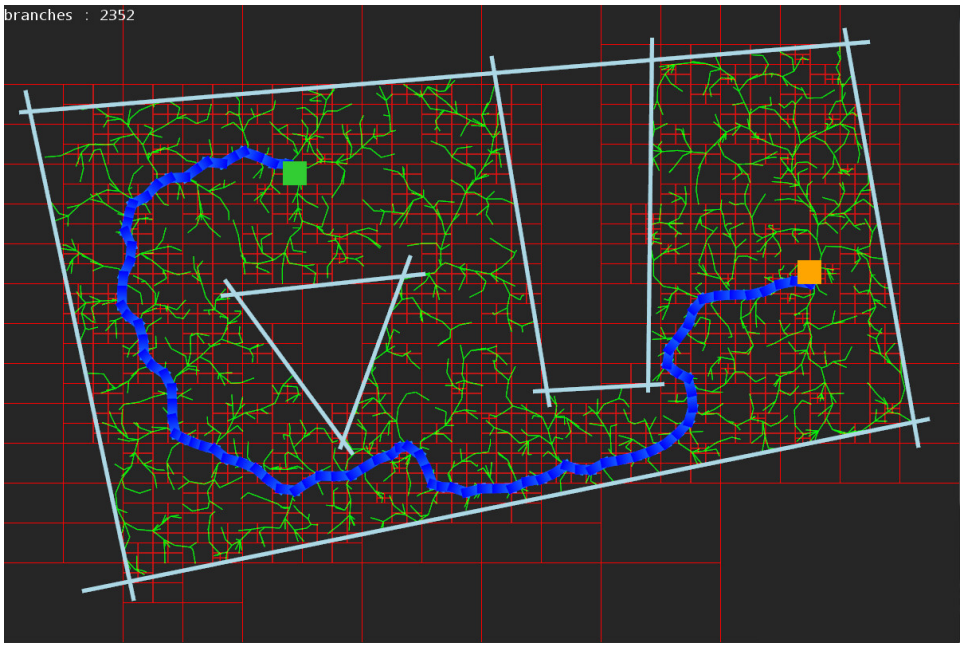


Figure 5.1: *RRT using a quad-tree structure to spatially partition the tree to allow for fast nearest neighbour searches.*

via the parent nodes, each tree node itself can be considered as representing a trajectory between its own configuration and $\mathbf{c}_S$.

The tree expansion across C operates in an iterative manner, randomly sampling an expansion target configuration $\mathbf{x} \in C$, finding the tree node $\mathbf{n} \in C$ closest to this sampled configuration $\mathbf{x}$, and then generating a new tree node by "steering" the configuration of $\mathbf{n}$ towards $\mathbf{x}$ in an attempt to expand the tree towards $\mathbf{x}$. This new node generation is performed using a so called "steering function" which attempts to guide one configuration towards another in compliance with the dynamics of the system/vehicle for which planning is being conducted. This allows RRT to be applied to a wide range of problems with various differential constraints [67], [53], [64], [30], however in the simplest case the steering function simply directly interpolates between two configuration vectors.

In scenarios where the constructed tree has grown to thousands of nodes, the process of merely determining the closest node to the expansion target becomes the dominant limiting factor in terms of performance. Thus a common RRT optimization is the use of spacial partitioning methods to achieve fast nearest neighbour searches. An example of a 2D RRT tree using a quadtree structure to achieve such fast NN searches is illustrated in Figure 5.1.

RRT demonstrates probabilistic completeness and scales well with higher dimensional planning problems. However, unlike the PRM it does not demonstrate probabilistic op-

timality. Thus despite being an excellent choice for high dimensional planning problems, RRT produced trajectories often require post processing in order to improve their quality to an acceptable level. The standard RRT algorithm is also single query due to the fact the tree is rooted at the starting configuration $\mathbf{c}_S$, and as such if the desired starting configuration $\mathbf{c}_S$ is changed the tree must be either altered to connect to this updated $\mathbf{c}_S$ or recreated from scratch.

5.1.6 Rapidly Exploring Random Tree Star (RRT*)

The RRT* algorithm [51] is one of the most widely used variations on the standard RRT algorithm largely due to it having the property of probabilistic optimality, resulting in it producing far superior trajectories to standard RRT. The tree expansion process is very similar to RRT, but an additional rewiring step takes place with the addition of each new node. This consists of examining all nodes in close proximity to the new node, and determining for each if the new node should replace their current parent node, due to the resulting trajectory being superior than their current trajectory. It is this rewiring step which guarantees probabilistic optimality, however, it does come at the price of a significant computational cost compared to standard RRT, largely due to the additional nearest neighbour searches being required. Figure 5.2 shows an example of the output from RRT* on a 2D path planning problem, demonstrating a path that is almost identical to the optimal in terms of minimum length. The same figure also clearly illustrates the differences between the tree structures produced by RRT* and standard RRT.

5.1.7 Informed RRT*

An additional improvement to RRT* for finding shortest length trajectories, informed RRT* as introduced by Gammell et al [31] uses the current best trajectory to restrict tree expansion to a subset of C. Specifically tree expansion is restricted to within a bounding ellipsoid in C, whose focal points located at $\mathbf{c}_S$ and $\mathbf{c}_G$, and whose major axis length is equal to the total length of the current best trajectory. It can be proven that any trajectory of shorter path length that may exist must be fully contained within this volume, and hence it is a waste of computation to conduct tree expansion outside of it. This restriction greatly improves the rate of convergence towards the optimal path (in terms of total path length). Figure 5.3 shows an example of this optimization in action on a 2D problem, note how a large amount of the configuration space which cannot possibly contain a better path remains unexplored.

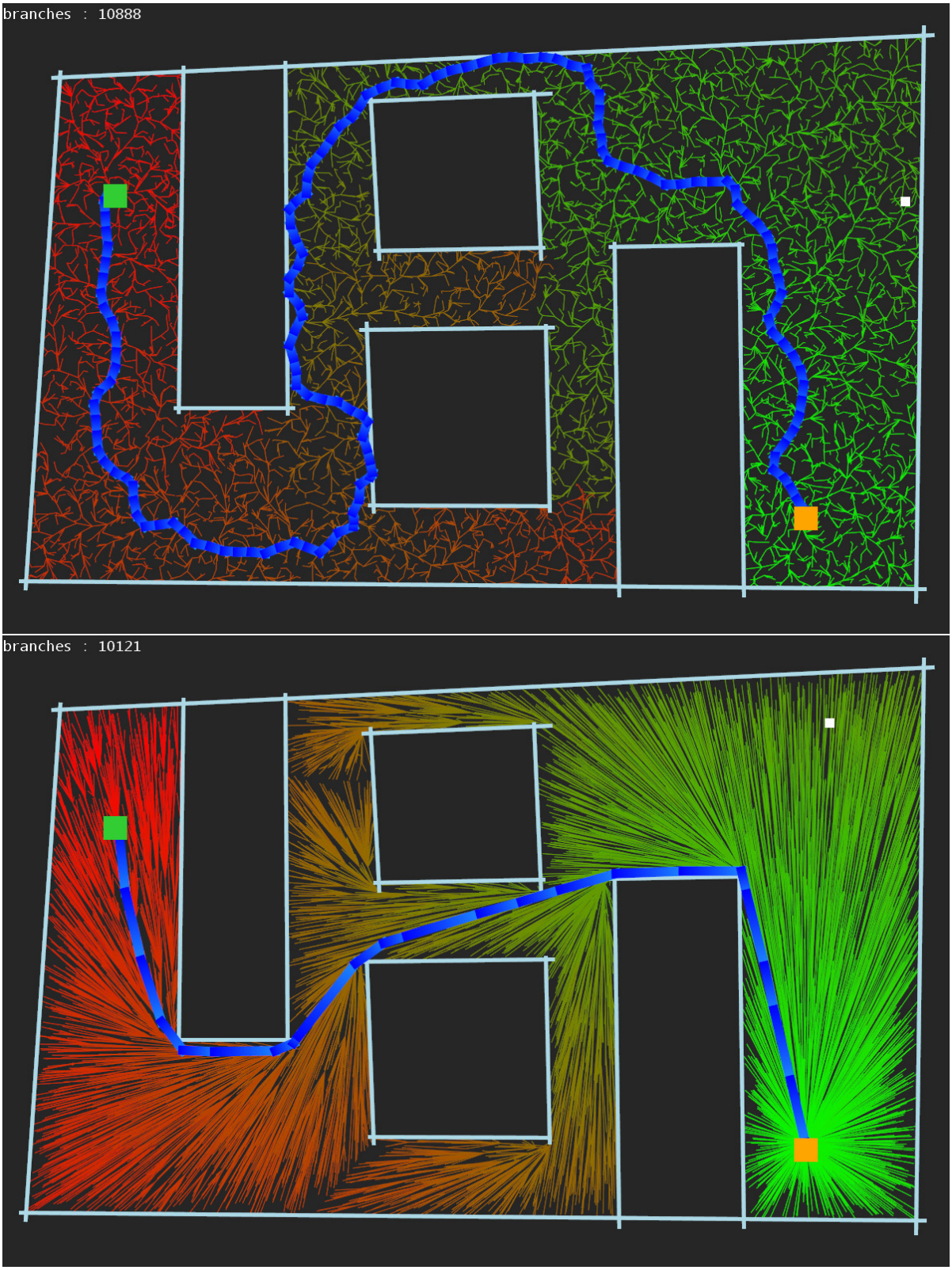


Figure 5.2: *Examples of both RRT (top) and RRT* (bottom) solving the same path planning problem, with both trees expanded to around 10000 branches. It is clear that RRT* produces a superior path in terms of total length.*

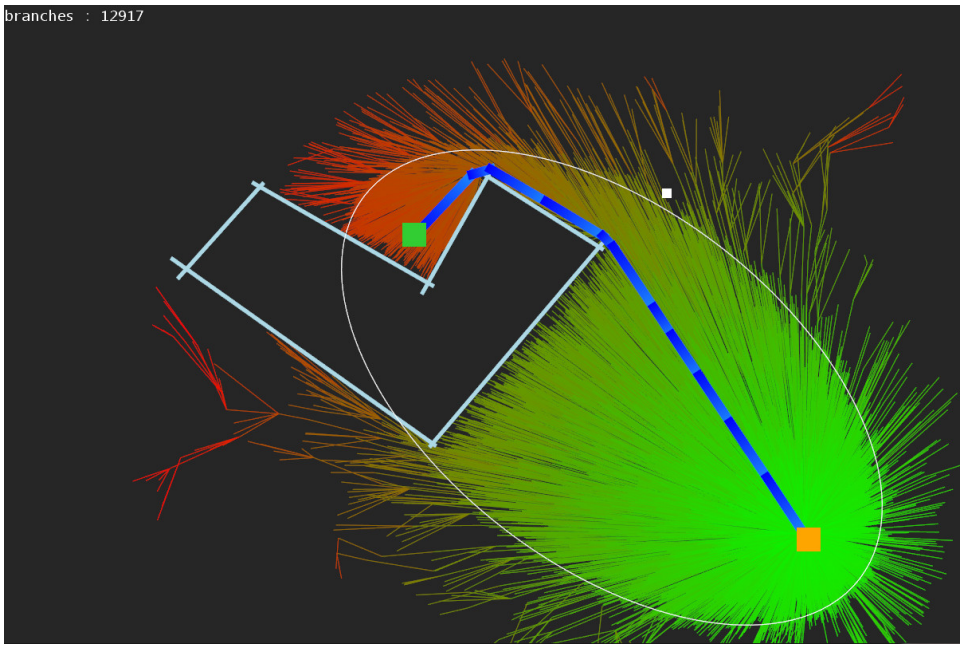


Figure 5.3: *An example of informed RRT*, tree expansion has been restricted to within the ellipsoid defined by the length of the current best path and start and goal locations in the configuration space.*

5.2 SLAM Aware Path Planning

This section provides and overview of the concepts and issues involved with SLAM based path planning. We then formally present our proposed approach in the follow Section 5.3.

5.2.1 Problem Overview

Our particular interest is in autonomous indoor navigation for Micro Air Vehicles (MAVs) using SLAM. We do make the assumption that the environment itself is static, however there is no guarantee that a complete map of the environment is available. Instead path planning must be conducted using the latest (typically incomplete) map constructed by the SLAM system. This SLAM map may expand as new areas are observed, or undergo significant global alterations due to the detection of new loop closures and map optimization as discussed in Section 2.8. Many of the planning methods described in Section 5.1 rely on the assumption that a complete and globally accurate map of the environment is always available, an assumption that no longer holds in SLAM based navigation scenarios.

Naturally great care must be taken to avoid collision with obstacles present in the en-

vironment. However, with no complete map of the environment itself there will exist regions of unknown space in which obstacles may or may not be present. Thus in order to ensure obstacle avoidance these potentially dangerous regions of unknown space must also be avoided during path planning in addition to known obstacles.

Additionally it is necessary to ensure that the SLAM system is able to maintain localisation throughout the course of navigation. This requires that sufficient information is observed by the vehicle's on-board sensors to maintain SLAM tracking at all times.

Thus at any one point in time, the SLAM system's current map along with a combination of constraints for obstacle avoidance and sensor information must be used to determine a configuration space C for the vehicle, in which safe path planning may be conducted. Note the SLAM map itself may be globally inaccurate and regularly undergo changes due to new loop closure detections, which then directly affect the configuration space C. Despite this, SLAM maps are typically accurate at a local level. That is to say that any small sub-map consisting of a subset of key-frames between which loop closures exist is likely to be accurate and consistent, even if its global location within the greater SLAM map is not, a fact that will be exploited in the proposed path planning approach. These concepts are now further expanded upon.

5.2.2 Empty, Occupied and Unknown Space

In any situation in which we do not have a complete map of the surrounding environment (or no map at all), we must consider the environment as being divided into known and unknown regions of space. Let $E \in \mathbf{R}_3$ denote the set of all points within the environment in question. Similarly let $U \subset E$ denote the set of all points inside currently unknown space, and $K \subset E$ all points within known space. There is no overlap between known and unknown space ($U \cap K = \emptyset$) and their union consists of all points within the environment itself ($E = K \cup U$).

Known space is that for which sufficient information via sensor observations has been obtained in order to accurately determine its contents. A specific point located within known space may either be occupied by some obstacle (such a wall, floor or other physical object), or be empty space which may be utilized for navigation. We refer to the set of all known space points occupied by physical obstacles as simply "occupied space" denoted by $K_{occupied} \subset K$, and similarly the set of all known empty space points as "empty space" denoted $K_{empty} \subset K$, with $K = K_{empty} \cup K_{occupied}$.

On the other hand unknown space is that whose contents are currently unknown. As such

each point $\mathbf{x} \in U$ could be revealed to be either within occupied space ($\mathbf{x} \in K_{occupied}$) or empty space ($\mathbf{x} \in K_{empty}$) upon acquiring additional sensor information. Naturally in order to conduct safe navigation it cannot be assumed that any region of unknown space $R \subset U$ is in fact a region of empty space, as there may exist within it some currently unobserved obstacle with which the vehicle may collide. Thus path planning must be restricted to within known empty space K_{empty}.

In practice it is neither practical nor necessary to determine an exact partitioning of the environment into empty, occupied and unknown space. As discussed in Section 2.2.1 the level of detail required of the configuration space used for path planning is relative to the size of the vehicle itself. Thus rather than determining and storing the exact nature of each point $\mathbf{x} \in E$ in the environment, the environment is partitioned into a number of discrete volumes of space, each taken to represent a region of empty(K_{empty}), occupied($K_{occupied}$) or unknown(U) space depending upon the belief of its content. Specifically a region must be considered occupied space if any obstacle is detected within it, on the other hand a region should only be considered as empty space if there is high certainty that it contains no such obstacles i.e. the entire region has been observed to consist of empty space. Such a conservative approach is required to safe navigation,

This partitioning based approach was first introduced with the use of Occupancy Grids to 2D mapping and navigation problems [22],[21],[103] partitioning the environment using a uniform 2D grid, with each cell holding a belief of its content (either empty, occupied or unknown space). Initially all cells begin representing unknown space, and as additional sensor measurements are obtained the beliefs of the observed cells are updated to reflect the new information.

This concept was extended to 3D mapping and navigation problems with the use of voxel grids [77],[84], partitioning the environment into a grid of uniform cubic volumes ("voxels") as illustrated in 5.7. We adopt such a voxel based approach, with each voxel's content (either empty, occupied or unknown space) determined by the SLAM system's map, sensor measurements and possibly other external information. An example of such a voxel based representation constructed from a map created by our edge based SLAM system from Section 2.3 is shown in Figure 5.4.

Decreasing the size of the voxels results in a more detailed approximation of the environment in question, but also results in a dramatic increase in the number of voxels required. Additionally the number of voxels needed to approximate said environment is simply proportional to the total volume of the environment itself. It is clear that such a uniform voxel approximation poorly represents large uniform volumes of empty, occupied

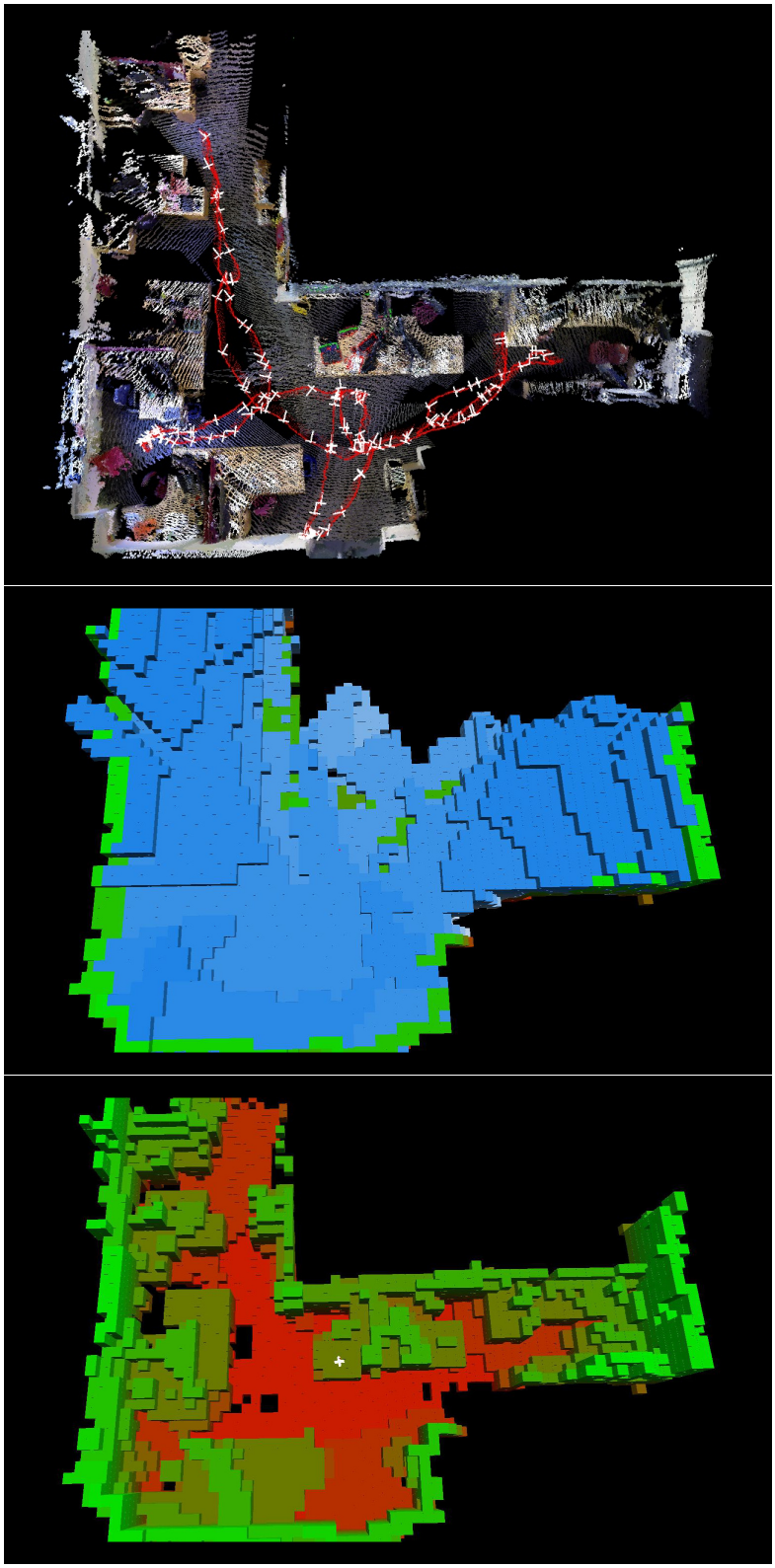


Figure 5.4: *A example of a uniform voxel map created for path planning purposes. Top shows the SLAM map from which the voxel map is generated. Middle shows both free and occupied space in Blue and green respectively. Bottom shows only occupied space with height indicated by color ranging from red (lowest) to green (highest).*

or unknown space in terms of memory consumption. For example, a large cubic region of empty space $R \subset K_{empty}$ would need to be represented using a large number of small volume voxels. This leads to many situations in which approximating an environment to the desired level of detail using a uniform voxel grid involves an extremely large number of voxels, resulting in significant computation and memory resources being required. However, for our navigational requirements a detailed representation of the environment is not necessary as geometric features smaller than the vehicle itself will not have an effect on the path planning decision making process, and thus these issue do not pose a problem.

If a finely detailed representation of K_{empty}, $K_{occupied}$ and U was required, then Octree based mapping approaches such as those demonstrated by [114] and [23] provides a viable alternative, addressing many of the issues inherent in a purely voxel grid based approach by providing a far more efficient partitioning.

5.2.3 Ensuring Sufficient Sensor Information for Localization

The path planning approaches discussed previously such as those of Sections 5.1.4, 5.1.5 and 5.1.6, have worked under the assumption that the agent conducting path planning (a MAV vehicle in our scenario) is always able to accurately measure its own configuration at all times (be that a 6DOF pose, joint angles or any other combination of variables). Since the agent can measure its own state it is able to follow any valid collision free trajectory.

However this assumption frequently does not apply in real world scenarios, such as those in which a vehicle must rely upon measurements from its on-board sensors to estimate its own configuration (i.e. localisation). In such a scenario there may exist many collision free trajectories through the environment along which the vehicle is likely to lose localisation due to poor sensor information. Once localisation is lost there is no guarantee the vehicle will be able to accurately follow a desired trajectory, and may stray off course colliding with some obstacle in the environment.

In the context of a vehicle relying upon a SLAM system for localisation, there may exist many locations within the environment at which the vehicle's sensors will not observe sufficient information for the SLAM system to maintain localisation. This may lead to the SLAM system only producing a partial estimation of the vehicle's configuration, or even non at all.

The configurations (i.e. vehicle poses) within the environment that will result in such

localisation failure depend on multiple factors such as the environment itself, the nature of the vehicle's on-board sensors, and what types of features extracted from the sensor data are used by the SLAM system. For example in the case of a vehicle using a monocular vision based SLAM system, sufficient visual information must be observed by the camera to maintain SLAM tracking and localisation. Thus a configuration resulting in the vehicle's camera observing nothing but a blank featureless surface is one example of a configuration that would result in SLAM localisation and tracking being lost completely due to the camera sensor providing no useful information.

Thus for a vehicle conducting autonomous SLAM based navigation, not only must path planning be restricted to known empty space K_{empty}, but it must also be ensured that the vehicle's sensors observe sufficient information for SLAM localisation to be maintained at all times. Let us use K_{info} to denote the set of all known space configurations at which the vehicle's sensors will observe sufficient information to maintain SLAM tracking based on the current map of the SLAM system. K_{info} can be viewed as an additional constraint on the vehicle's navigation to ensure SLAM localisation is maintained, much in the same way as how K_{empty} defines a constraint to ensure obstacle avoidance. The configuration space in which to conduct safe navigation ensuring both collision avoidance and SLAM localisation is thus given by.

$$C = K_{empty} \cap K_{info} \tag{5.2}$$

Naturally when determining K_{empty} the difference between the vehicle's frame of reference, and the frame of reference of each of its sensor must be accounted for. Typically these sensors are rigidly attached to the vehicle such that there is simply a fixed transformation between vehicle and sensor reference frames. In our application the MAV vehicle carries a single RGB-D sensor, whose reference frame is at a fixed transformation $\mathbf{s}$ from that of the vehicle. The sensor's pose in a given reference frame can thus be determined at any point by simple transforming the vehicle's pose (in the same reference frame) by $\mathbf{s}$. The vehicle's 6DOF pose is simply derived from its configuration (x,y,z,yaw), roll and pitch angles which control the vehicles translational velocities are taken to be 0 under the assumption that they are always of small magnitude.

5.2.4 Configuration Space Generation using SLAM Map Data

Without a complete prior map of the environment, the configuration space used for path planning must be generated from the map constructed by the SLAM itself. This requires that the configuration space be regularly updated as new information is incorporated into the SLAM map, resulting in the configuration space being expanded during the course of navigation. Additionally the SLAM map may also undergo large scale structural changes due to loop closures being detected between globally distant regions, which would then require the configuration space to be updated in order to reflect such changes and avoid becoming invalid.

In this way the configuration space may need to be regularly updated and revised during navigation to reflect any changes to the SLAM system map. However it is important to note again that any small sub-map, consisting of a subset of key-frames between which loop closures exist is likely to be accurate and consistent, and thus unlikely to be affected by such map changes. Thus while changes to the SLAM map may alter the configuration space associated with the entire global SLAM map, the configuration spaces associated with various sub-maps may not be affected. This fact can be exploited by thus avoiding the use of a single global configuration space, and instead utilizing multiple small configuration spaces resistant to such SLAM map alterations. We refer to a region of the map as being locally unchanged if it experiences either no change, or that the entire map region has simply undergone a rigid transformation, that is the global location of the region has changed, but the content and structure of the region has not itself been altered as illustrated in Figure 5.5.

Intuitively it is wasteful to recreate the configuration space for such locally unchanged regions, and instead it would be far more efficient to only recalculate the configuration space associated with regions of the SLAM map that actually have undergone local change. This however requires the ability to selectively update only certain parts of the configuration space in question. To achieve this we take a partitioning based approach, using many small configuration spaces denoted $C_0, C_1, C_2...$, each for a different region of the SLAM map. Thus when a change occurs to the SLAM map only those C_i locally affected by the change need be regenerated.

Our proposed SLAM system employs a keyframe based approach as described in Section 2.1.2, where each keyframe K_i holds a small subset of the whole SLAM map along with an estimated global pose $\mathbf{P}_i$ of where it is located. These keyframes themselves present a natural way to conduct the partitioning of the global configuration space into

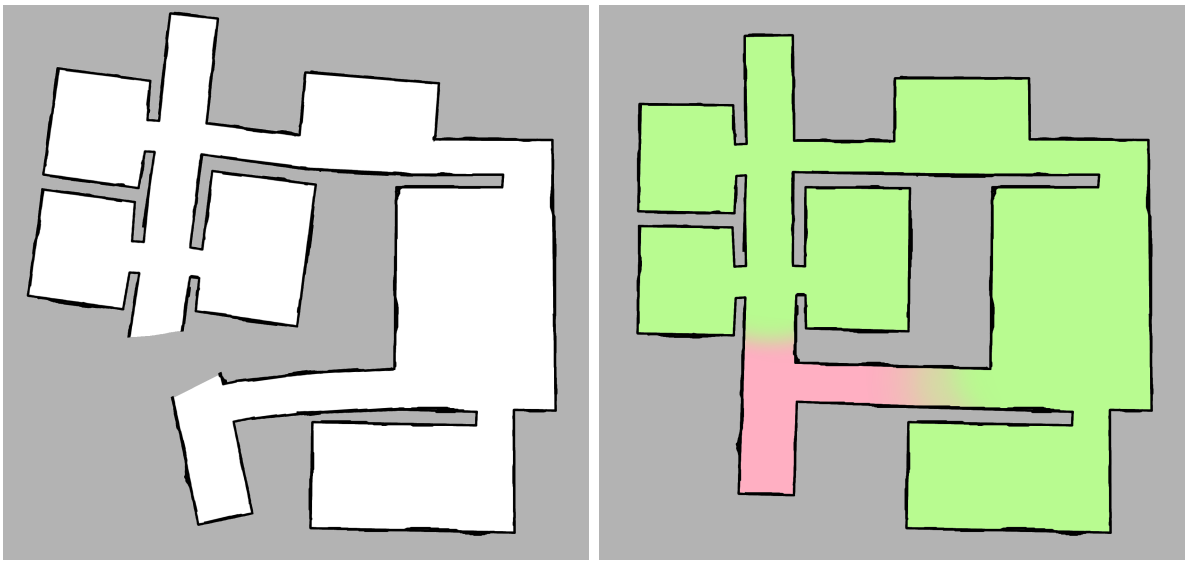


Figure 5.5: *2D Illustration of map changes due to loop closure. Areas highlighted in green largely remaining unaffected while red areas have undergone significant change.*

multiple smaller sub spaces C_0, C_1, C_2.... We make use of a set of keyframe configuration spaces C_0, C_1, C_2..., with each keyframe configuration space C_i associated with a specific keyframe K_i, and located at that keyframes estimated pose $\mathbf{P}_i$. As the SLAM map data of each keyframe K_i is static, each keyframe configuration space C_i need only be computed once (at the keyframe's creation).

In order to conduct full navigation however path planning cannot be restricted to a single keyframe configuration space C_i. Instead we must be able to determine trajectories passing through multiple such keyframe configuration spaces, which requires determining connections between said configuration spaces dictating where planning can transition from one such C_i to another C_j. Intuitively safe transition between two different configuration space C_i and C_j may occur wherever there is overlap between them. However determining such overlap requires an estimate of the relative pose between the locations of C_i and C_j. This relative pose could be determined from the estimated global keyframe poses $\mathbf{P}_i$, $\mathbf{P}_j$ however these are highly prone to error and drift, which could lead to invalid connections between configuration spaces. Alternatively loop closures between pairs of keyframes give an estimate of their relative pose that is generally far more reliable being based upon matching the common features within each keyframes sensor data. Thus loop closures are far more preferable as a means to determining keyframe configuration space connections, however they do have the downside that such loop closures must be detected before connections can be made.

This scheme of having many configuration spaces C_0, C_1, C_2... (one for each keyframe)

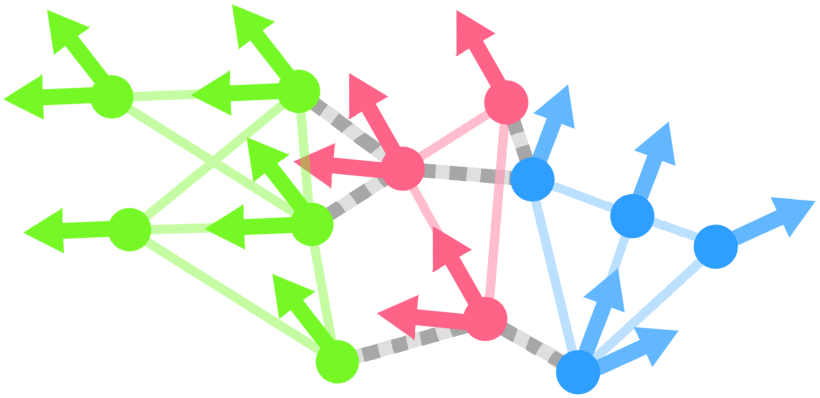


Figure 5.6: *Multiple small scale navigation graphs (as indicated by color) generated from keyframe RGB-D data connected together with edges determined by keyframe loop closure detections illustrated by the dashed edges.*

and with connections determined by detected SLAM loop closures, is illustrated in Figure 5.6. The overall resulting configuration space can be viewed as topological in nature, due to the fact the connections determining where it is possible to transition between two configuration spaces C_i and C_j are not determined by where they are currently estimated to be located (and instead by the relative pose estimates produced by loop closures).

In addition to providing a simple way to avoid having to recompute the entire configuration space whenever the SLAM system map changes, this approach is far more robust to inconsistency in the SLAM map caused by accumulating errors and drift. Path planning for conducting safe navigation can typically still be performed even if the SLAM map itself has become extremely inconsistent since the connection between the configurations space $C_0, C_1, C_2...$ are determined by loop closures alone. Another advantage is that an existing trajectory across some set of keyframe configurations spaces $\{C_i, C_j, C_k, ...\}$ will not be invalidated by changes to the SLAM map (in the form of changes to any of the associated keyframe poses $\{\mathbf{P}_i, \mathbf{P}_j, \mathbf{P}_k, ...\}$). Instead such trajectories remain inherently unchanged. These concepts are formalised in the following sections.

5.3 Keyframe Centric Path Planning

5.3.1 Overview

This section now formalizes the construction process of the keyframe navigation graphs used for path planning, adhering to the required collision avoidance and sensor information / SLAM localization constraints discussed previously.

Let us denote the set of m keyframes making up the current SLAM system map by $K = \{K_0, K_1, ..., K_m\}$. Each keyframe $K_i \in K$ then has an associated navigation graph $G_i = (N_i, E_i)$, consisting of a set of nodes N_i representing possible vehicle poses (each relative to the pose of K_i itself), and a set of edges E_i, each of which represents a safe (collision free and sufficient sensor information) trajectory between a pair of poses from N_i. In practice these vehicle poses are taken to have roll and pitch angles of zero, thus ensuring that each represents a valid configuration which the vehicle may take, working under the assumption that the vehicle is 4D Holonomic with respect to x,y,z and yaw. From here on will continue to refer to such $\mathbf{n} \in N_i$ as vehicle poses rather than configurations.

Each pose $\mathbf{n} \in N_i$ is collision free such that the vehicle may be located at such a pose $\mathbf{n}$ without being in collision with any obstacle in the environment or intersecting any region of unknown space. Additionally each $\mathbf{n} \in N_i$ ensure that the vehicle's on-board sensors would observe sufficient features from the keyframe K_i in order to maintain SLAM localisation. Thus it is safe for the vehicle to assume any of the poses $\mathbf{n} \in N_i$ from such a navigation graph G_i, since they are both obstacle free and ensure SLAM localisation.

Each of the navigation graph edges $\mathbf{e} = (\mathbf{n}_a, \mathbf{n}_b) \in E_i$, represents safe trajectories for the vehicle, interpolating between two different poses $\mathbf{n}_a, \mathbf{n}_b \in N_i$. Each such trajectory has been determined to be both collision free, and ensures that SLAM localisation is maintained using features from the keyframe K_i. These keyframe navigation graphs provide a way for the vehicle to travel between a small network of poses, while ensuring that both the obstacle avoidance and sensor information constraints discussed in the previous section are met. Further since each edge $\mathbf{e} = (\mathbf{n}_a, \mathbf{n}_b) \in E_i$ can be navigated using observations to the common set of features (the edge clouds D_i and I_i), and the relative poses between the nodes of $\mathbf{n}_a$ and $\mathbf{n}_a$ is fixed (as is the known collision free space between said nodes), the safety of the edge's trajectory is not affected by global map errors, loop closures or long term drift within the SLAM map. Structural changes in the map can change only the relative location of other keyframes relative to K_i.

In order to conduct navigation on a larger scale, an additional set of connecting edge C is determined, connecting together pairs of nodes from different keyframe navigation graphs, with each edge of the form $\mathbf{e} = (\mathbf{n}_a \in N_i, \mathbf{n}_b \in N_j) | i \neq j$. Similar to the edges of the keyframe navigation graphs, each of these connecting edges also represent safe interpolating trajectories between two different poses. However such edges are only generated between pairs of keyframes between which a loop closure has occurred. The local loop closure constraint between a pair of key-frames is unlikely to be affected by global structural changes to the SLAM map, and thus these connecting trajectories display a similar invariance to global structural map changes as the former keyframe navigation graph edges.

These connecting edges can then be used in forming a graph G_{map}, combining the nodes and edges of each of the keyframe navigation graphs.

$$G_{map} = (\{N_0 \cup N_1 \cup ..., N_m\}, \{C \cup E_0 \cup E_1 \cup ..., E_m\}).$$

Standard graph based path planning methods can then be used to determine routes through G_{map} allowing the vehicle to navigate across the map through multiple keyframe navigation graphs. If the edges of C are such that the graph G_{map} is connected, then path planning can be conducted across the entire SLAM map.

The process of constructing a keyframe navigation graph G_i is now described in the following section.

5.3.2 Keyframe Navigation Graph Construction

Each keyframe K_i stores a single frame of RGB-D data F_i, along with the associated depth and RGB edge point clouds D_i and I_i, as discussed in Section 2.3.3. The construction of a keyframe navigation graph G_i consists of a number of steps, using these edge point clouds and RGB-D frame data to determine a set of safe poses and trajectories, forming the nodes and edges of the graph.

5.3.2.1 Keyframe Voxel Maps

To construct the navigation graph G_i associated with the keyframe K_i, it is first necessary to construct a representation of the known empty space associated with K_i. This known empty space is determined from both the RGB-D data of the keyframe itself (F_i) and any subsequent frames of RGB-D data acquired by the sensor whenever the SLAM system is using K_i as the current tracking keyframe (K_T) for sensor tracking/localisation.

Specifically a uniform voxel grid is used to represent the known empty space associated with K_i (and simultaneously the known occupied, and unknown space). Once the initial content of this voxel grid has been determined a set of safe obstacle free poses for the vehicle $X = \{\mathbf{x_0}, \mathbf{x_1}, ...\}$ can be determined by examining the grid's contents (note that poses are relative to the keyframe's pose P_i). These safe poses can then be used in determining the location of nodes in the keyframe's navigation graph G_i.

In summary voxel maps are generated for each keyframe and utilized in the construction of each keyframe's navigation graph (and the connections between these graphs as will be described later in Section 5.3.2.7). Naturally each such map is given in the reference frame of its associated keyframe. The generation of these voxel maps is now described in further detail.

5.3.2.2 Voxel Map Initialization

The initial contents of a keyframe K_i's voxel map are determined using its associated frame of RGB-D data F_i. Initially all voxel cells are flagged as representing volumes of unknown space. The pixels of the keyframe's RGB-D frame F_i are then back-projected, forming the associated dense RGB-D point cloud in which each point lies upon the surface of some physical obstacle in the environment. Each point in this cloud implies the existence of a ray of empty space between the location of the sensor and the point itself. That is for each point $\mathbf{x}_j$ from the RGB-D cloud, there must exist a finite ray of empty space r_j between said point $\mathbf{x}_j$ and the sensor's location, which in this initialization case is located at the origin (since by definition the sensor acquired the keyframe K_i's associated RGB-D frame F_i at the keyframe's pose P_i) Each point $\mathbf{x}_j$ and its associated ray of empty space r_j is then used to determine the initial contents of the voxel grid as shown in Figure 5.7.

This process first involves examining each finite ray of known empty space r_j, determining which voxels the said ray intersect, and flagging these voxels as representing known empty space. This involes the well studies problem of voxel ray tracing and we implement a standard method of fast voxel tracing as introduced in [2] to perform this step. The next step then examines each point $\mathbf{x}_j$ from the RGB-D cloud, determines the voxel the point resides in, and flags that voxel as being known occupied space. In this way all voxels occupied by any environmental obstacle represented in the RGB-D cloud are flagged as known occupied space. An example of the results from this process are illustrated in Figure 5.7.

After the contents of the voxel grid have been updated in this manner, a shrinking process

is performed on the volume of known empty space approximated by the map. This simply involves finding those empty space voxels that are neighbours to at least one non-empty space voxel, and changing their value from empty space to unknown space. This shrinking process is performed recursively a number of times, determined by the dimensions of the vehicle, with the intent that the remaining empty space voxels are those in which the vehicle may be located anywhere within (and at any orientation), and still always be ensured of not colliding with any obstacle. Thus the center positions of all remaining empty space voxels are known safe positions at which the vehicle can be positioned with any orientation, we denote the set of these positions by $X = \{\mathbf{x_0}, \mathbf{x_1}, ...\}$.

It is important to note that as was shown in [57], the accuracy of depth data from standard RGB-D sensors greatly degrades with increasing depth value due to increasing distortion, spacing of readable depth values, and noise. Thus the content of voxels located at a far distance from the RGB-D sensor cannot be accurately determined. As such we restrict the uniform voxel grid to only cover a volume of 5 meters cubed.

Figure 5.9 shows an example of a small SLAM map along with the associated voxel maps for each keyframe, with occupied space voxels drawn in green and empty space voxels in blue.

5.3.2.3 Voxel Map Updates

The voxel maps associated with each keyframe may also be updated using the latest frame of RGB-D data F produced by the RGB-D sensor. Whenever a keyframe K_i is being used by the SLAM system as the current tracking keyframe K_T (as described in Section 2.7), the latest sensor RGB-D data F is used to update its voxel map. This update process is largely identical to the voxel map initialization process described previously in Section 5.3.2.2. First the estimated pose of the sensor is transformed into the reference frame of the keyframe K_i (by simply determining the relative pose between the sensor and P_i), after which the sensors RGB-D frame F is back-projected to form a point clouds which is then used to determine voxel content in the same manner as the voxel map initialization process. However it should be noted due to the large amount of data continuously produced by the sensor, and the fact that much of the space involved in these updates has already been observed prior, only a down-sampled selection of pixels from F are infact back-projected in order to reduce computational cost (typically x10 down-sampling).

This updating procedure allows the volume of empty space associated with each keyframe's voxel map to be expanded with additional sensor measurements, which is instrumentive

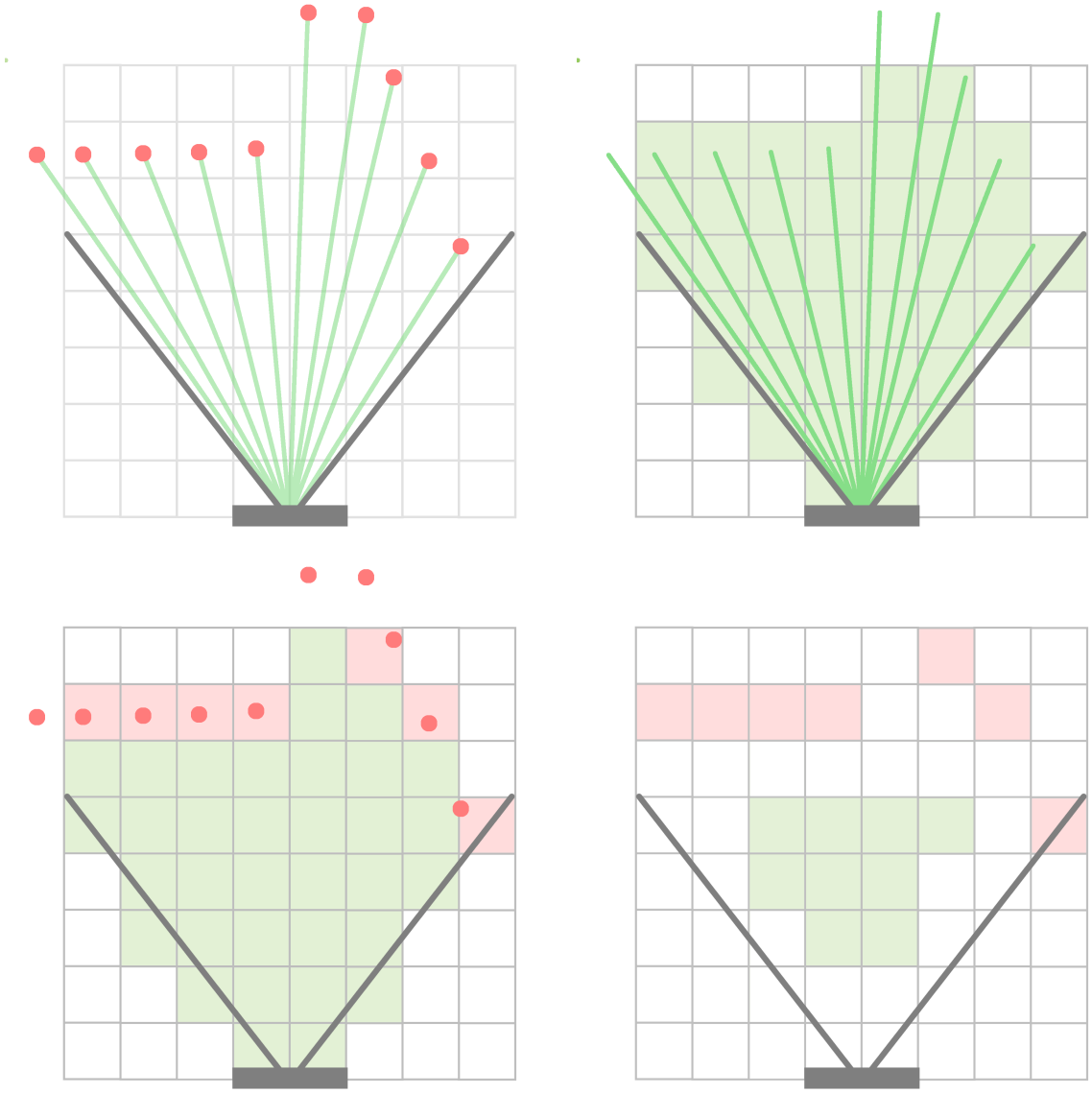


Figure 5.7: *A 2D illustration of voxel generation from point cloud data. Top left shows rays of know free space determined from each point in the cloud as shown in red. Top right, all voxels traversed by free space rays are flagged as free space (green). Bottom left, each voxel containing a point from the cloud is flagged as occupied space (red). Bottom right, a shrinking process is applied to free space voxels to ensure obstacle avoidance during path finding.*

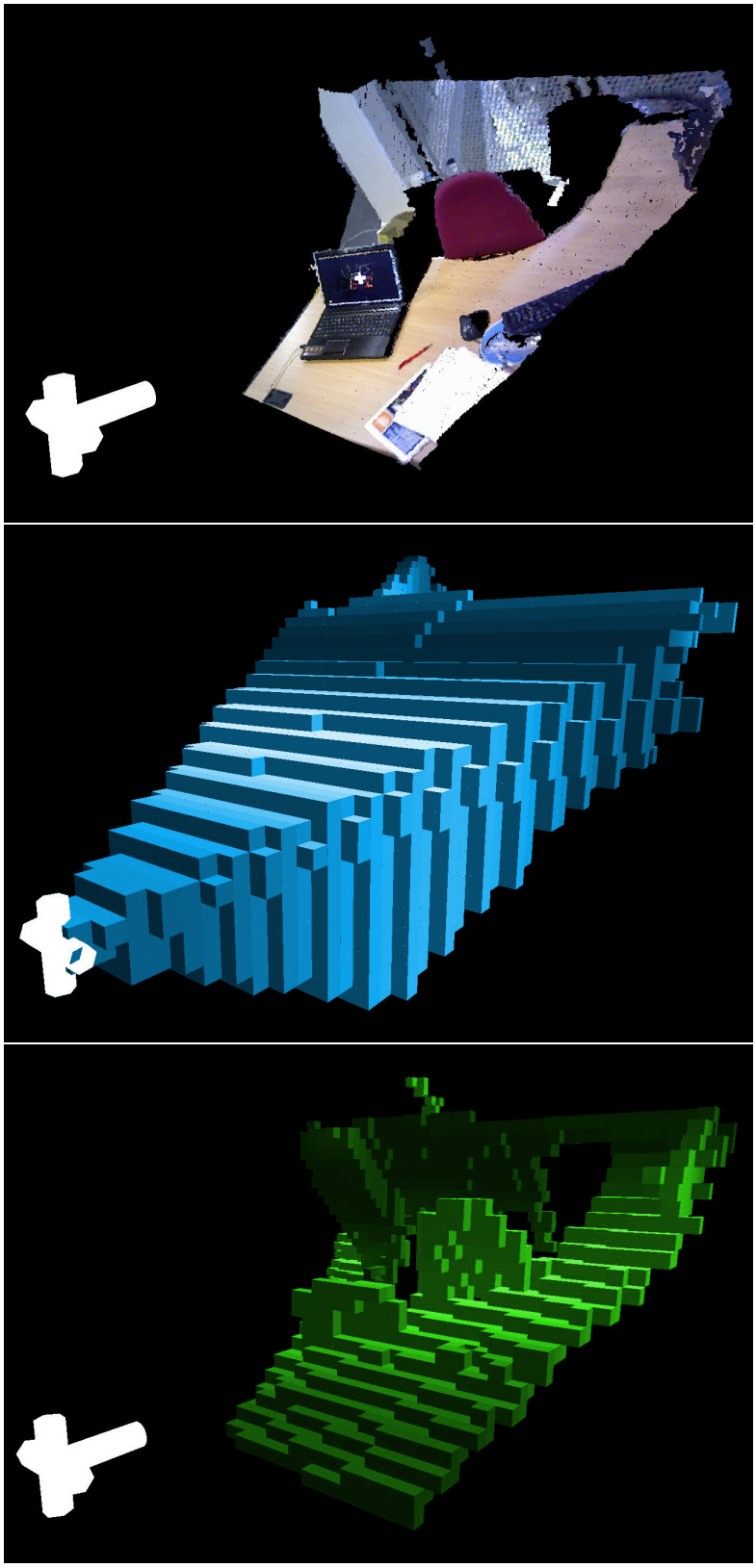


Figure 5.8: *An example of uniform voxel partitioning from a single dense RGB-D point cloud, known empty space voxels are drawn in blue, occupied space voxels in green. Unknown space voxels are not drawn.*

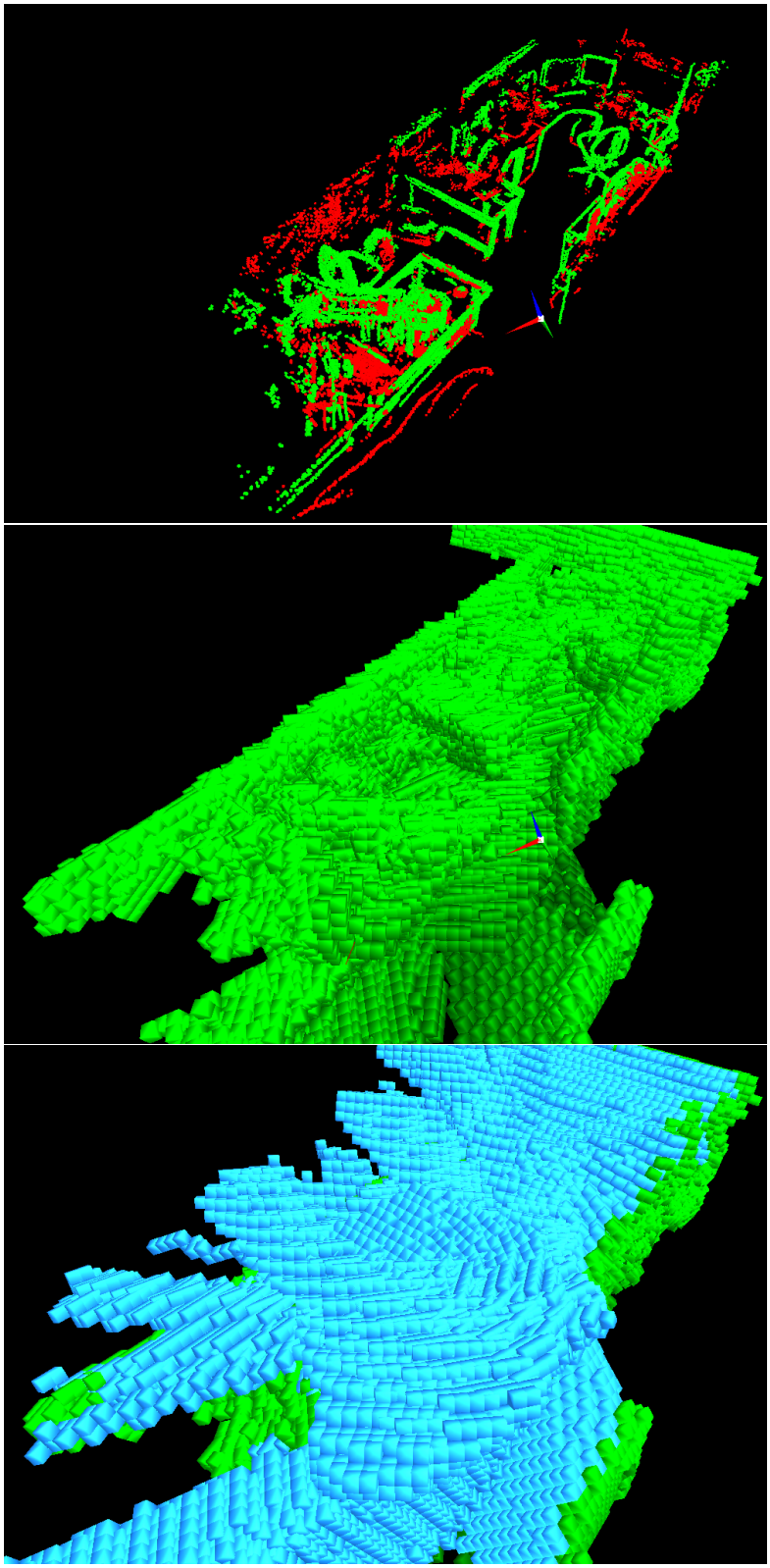


Figure 5.9: *Example of a SLAM map and the voxel maps associated with the various keyframes.. Top shows all edge features drawn relative to their respective keyframes. RGB edges are drawn in red and depth edges in green. Middle illustrates in green the occupied space voxels from each keyframe's voxel map. Bottom shows both occupied voxels and free space voxels (in blue) from each keyframe's voxel map.*

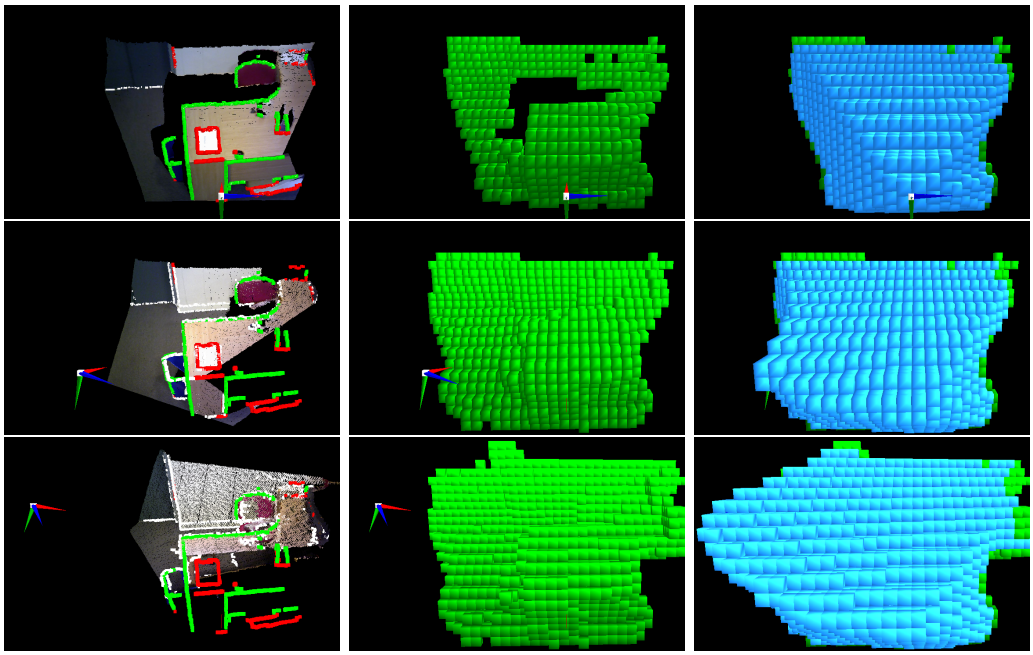


Figure 5.10: *Example of a voxel grid associated with a specific keyframe being updated over time (from top to bottom) as additional sensor data is aqquired. The left column shows the current observed RGB-D data and edge features of the keyframe. Middle shows the current occupied voxels in green, right shows both the occupied voxels and those voxels determined to be free space in blue.*

in allowing connecting between the keyframe navigation graphs to be formed as described later in Section 5.3.2.7. Examples of such keyframe voxel maps being updated are illustrated in Figures 5.10, again empty and occupied space are drawn in blue and green respectively.

5.3.2.4 Trackable Poses

As described previously in Section 5.3.2.2 a set of collision free vehicle positions $X = \{\mathbf{x}_0, \mathbf{x}_1, ...\}$ associated with a keyframe K_i can be determined from its associated voxel map. However in addition to simply avoiding obstacles, the vehicle's RGB-D sensor must observe sufficient features to maintain localisation. Thus for each position $\mathbf{x} \in X$ a set of associated vehicle orientations need to be determined which ensure that localisation can be maintained from the features of the keyframe K_i (i.e. the edge clouds D_i and I_i). These vehicle orientations and their associated vehicle positions then describe a set of vehicle poses which ensure SLAM tracking is maintain from K_i.

Determining such orientations first requires being able to determine what subset of features from a keyframe K_i would be observable from a specific sensor pose $\mathbf{p}$ (relative to

the pose P_i), and also determining if such a subset of features is sufficient to maintain localisation.

As stated, the first step requires determining what subset of keyframe K_i's features are reliably observable from a pose $\mathbf{p}$. In our proposed SLAM system, a keyframe K_i's stored features are in the form of an RGB edge point cloud I_i, and a depth edge point cloud D_i as discussed in Section 2.3.3.

Let $\mathbf{f} \in \{I_i \cup D_i\}$ denote some point from either one of these edge point clouds. For it to be possible to observe the point $\mathbf{f}$ from a candidate sensor pose $\mathbf{p}$ (and hence use it in localisation), it must both lie within the sensor's field of view, and be within a reasonable distance from the sensor in order to ensure an accurate depth measurement.

Thus two checks must be performed. The first examines the expected depth value δ associated with the point $\mathbf{f}$, when observed by the sensor at pose $\mathbf{p}$. Let the position and forward facing direction of the sensor at pose $\mathbf{p}$ be denoted by the vector $\mathbf{x}_p$, and unit vector $\widehat{\mathbf{n}_f}$ respectively. The expected observed depth value associated with $\mathbf{f}$ is given by

$$\delta = (\mathbf{f} - \mathbf{x}_p) \cdot \widehat{\mathbf{n}}_f$$

For the point to be observable, δ must be greater than the sensors minimum readable depth (around $0.8m$ for structured light RGB-D). Additionally since structured light sensor depth estimation greatly degrades with increasing distance, we also require that δ be smaller than a threshold depth $\delta_{max} = 4m$, for an observation of $\mathbf{f}$ to be considered sufficiently accurate for tracking, i.e. that $0.8 < (\mathbf{x} - \mathbf{x}_p) \cdot \widehat{\mathbf{n}_f} < \delta_{max}$.

If the point $\mathbf{f}$ passes this first depth check, a second is performed to determine if the point lies within the sensor's field of view. This involves checking if the point is inside the RGB-D sensor's view frustum. This consists of a pyramidal volume bounded by four planes whose geometry is dependent upon the sensor's horizontal and vertical field of view (in addition to the sensor's own pose $\mathbf{p}$). These planes are defined such that their normals point inwards, into the frustum volume itself. Using the standard point-normal form to define a plane in $\mathbb{R}^3$, let the four bounding planes of the frustum be defined (relative to the keyframe's pose P_i) by the point and unit normal pairs $(\mathbf{x}_j, \widehat{\mathbf{n}_j})$, $j \in [0, ..3]$. Determining if $\mathbf{f}$ lies within the frustum volume then involves checking that $\mathbf{x}$ is on the "inner" side of each of the four frustum planes, i.e. that

$$\forall j \in [0, ..3] : (\mathbf{f} - \mathbf{x}_j) \cdot \widehat{\mathbf{n}_j} > 0$$

If $\mathbf{f}$ passes both these checks it is assumed to be observable. Thus by performing the same observability check on each of the points in the clouds I_i and D_i, the subsets of

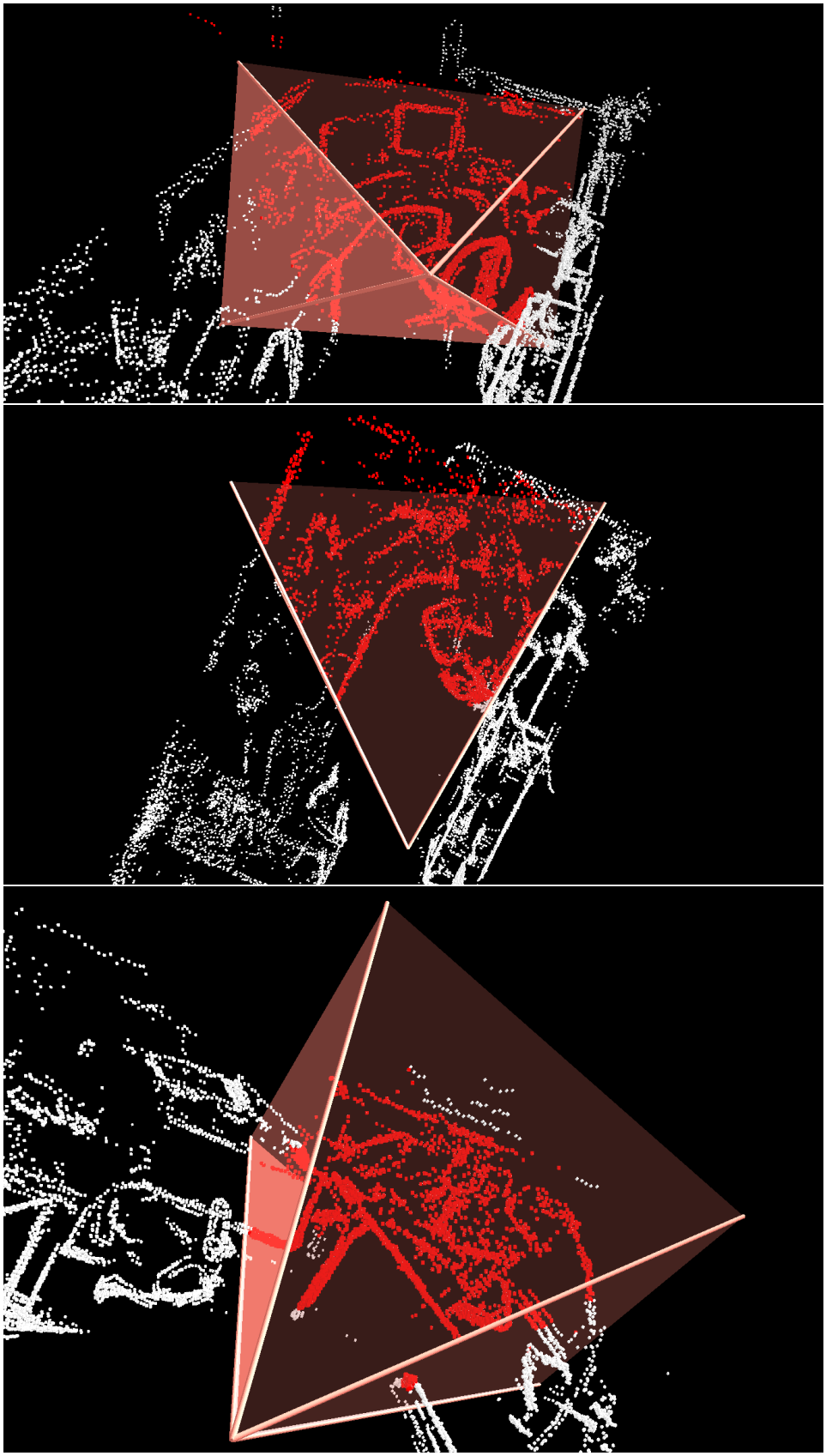


Figure 5.11: *Examples of sensor frustum culling of edge cloud points.*

edge points observable from sensor pose $\mathbf{p}$ can be calculated. Let us denote these subsets of edge points $I_i^{\mathbf{p}}$ and $D_i^{\mathbf{p}}$ respectively.

Having established sets of observable edge points $I_i^{\mathbf{p}}$ and $D_i^{\mathbf{p}}$, it is finally necessary to test if they provide sufficient information to maintain SLAM localization. This test involves evaluating if reliable registration can be achieved between the original keyframe edge clouds I_i and D_i, and the observable edge clouds $I_i^{\mathbf{p}}$ and $D_i^{\mathbf{p}}$ given a set various initial transformations T. The ICP registration process described in Section 2.6 is used to perform these registration tests. Given that $I_i^{\mathbf{p}}$ and $D_i^{\mathbf{p}}$ were transformed by a transformation $\mathbf{t} \in T$, correct registration should always result in an estimated transformation close to the inverse $\mathbf{t}^{-1}$ such that $\mathbf{t}^{-1}\mathbf{t}$ is the identity pose.

Let the set of initial transforms T which are tested consist of a set of uniformly spaced transformation (both in terms of translation and orientation) about the identity pose. Each transformation $\mathbf{t} \in T$ has a translational magnitude smaller than $\varepsilon = 0.5m$ and rotational component with angle magnitude smaller than $\theta = 0.3$ radians when expressed in axis angle form. Transformations of greater magnitude are unlikely to be encountered due to the limited rate of movement the sensor typically experiences and thus do not need to be evaluated.

The set of transformations T is then used as a set of initial ICP transformations used to evaluate registration. Registration between $(I_i^{\mathbf{p}}, D_i^{\mathbf{p}})$ and (I_i, D_i) is attempted for each initial transformation $\mathbf{t} \in T$. If for any $\mathbf{t} \in T$, ICP registration results in an incorrect alignment transform significantly different from the inverse $\mathbf{t}^{-1}$, it is likely due to the edge clouds $(I_i^{\mathbf{p}}, D_i^{\mathbf{p}})$ not being sufficient for reliable registration. In such a case it follows that the sensor pose $\mathbf{p}$ would not ensure reliable SLAM tracking and localisation from the features of keyframe K_i. Conversely any pose $\mathbf{p}$ which results in successful registration for all initial transforms $\mathbf{t} \in T$ is deemed to ensure such SLAM tracking.

5.3.2.5 Node Placement

To form the navigation graph G_i (for a keyframe K_i) we need to determine a set of safe vehicle poses to form the nodes of the navigation graph, i.e. a set of poses N_i which are both collision free and ensure the vehicle's sensor observes sufficient features from the key-frame K_i to maintain SLAM localisation. Previously Section 5.3.2.2 demonstrated how a voxel grid could be generated for each keyframe K_i, from which a set of collision free vehicle positions $X = \{\mathbf{x}_0, \mathbf{x}_1, ...\}$ relative to the pose $\mathbf{P}_i$ could be determined. Section 5.3.2.4 then described the process by which a potential sensor pose $\mathbf{p}$ (relative to some keyframe pose P_i) can be tested to determine if sufficient features from K_i would

be observed to maintain SLAM localisation. These two elements, the set of collision free vehicle positions X and the test to determine if a potential sensor pose maintains SLAM localisation, can now be used together in order to determine a set of safe vehicle poses N_i, which are both collision free and maintain SLAM tracking from K_i. This involves using X to generate a set of collision free poses P, and then determining the subset of these poses $N_i \subset P$ which maintain SLAM tracking from the features of K_i. This set of poses N_i can then be used to form the nodes of the K_i's navigation graph G_i.

Let $\Theta = \{\mathbf{q}_0, \mathbf{q}_1, ...\} \subset SO(3)$ denote a set of orientations relative to that of the keyframe pose P_i. The set of poses P is then generated by taking all possible combinations of positions from $X \subset \mathbb{R}^3$ and orientations from Θ, that is $P = \{(\mathbf{x_i}, \mathbf{q_j}) | \mathbf{x_i} \in X, \mathbf{q_j} \in \Theta\}$.

What set of orientations Θ to use for generating this set of poses P is dependant on the dynamics of the vehicle itself. A quad-rotor MAV platform cannot arbitrarily change its roll or pitch while holding its position constant, typically such a vehicle can only control the yaw component of its orientation independent of its position. As described previously, because of the lack independent control over yaw and pitch angles we restricted the vehicle's configuration space the 4D x,y,z,yaw configuration space in which the vehicle is Holonomic. This restriction is made under the assumption of small roll and yaw angles, and thus we restrict the set of orientations Θ to consist only of those corresponding to the vehicle varying its yaw component.

Intuitively orientations which result in the majority of K_i's edge feature lying outside of the sensor field of view are unlikely to ensure SLAM tracking can be maintained (at least from the features of K_i). Thus to avoid such useless orientations, Θ should consist of yaw only rotations lying within an interval $[-\theta, \theta] : \theta \in \Re^+$, where the magnitude of θ is derived from the field of view of the sensor. Additionally a fine degree of orientation granularity is typically not required in order to construct a navigation graph sufficient for navigation. Thus we take Θ to be a set of three yaw only rotations of magnitude $\{-\theta, 0, \theta\}$.

A set of obstacle free vehicle poses P is thus formed from all combinations of positions of X and orientations corresponding to the yaw rotations of magnitudes $\{-\theta, 0, \theta\}$. P also represents the set of potential navigation graph nodes, however each pose from P must be tested to ensure the vehicle's sensor would observe sufficient features to maintain SLAM tracking. This involves transforming each vehicle pose from P into a sensor pose (by simply applying $\mathbf{s}$ the rigid transformation between the sensor and vehicle reference frames), and then performing the test described previously in Section 5.3.2.4. Poses

which successfully pass said test are then used to form the set of nodes N_i for the navigation graph G_i.

Examples of such a set of poses is illustrated in Figure 5.12, with edge point features drawn in white, poses drawn in green, and the sensor view frustum at the keyframe's pose P_i in pink. Rejected poses that did not pass the test to ensure SLAM localisation is maintained are shown in red. In the top left example it can be observed how the distribution of features has restricted the set of poses to face to the right, while by comparison the top right example set of features is such that they impose no such restriction. The bottom left example shows the poses generated for a keyframe facing down a narrow blank walled corridor, poses have been restricted to both face down the corridor to observe sufficient features in the distance, in addition to being restricted by the physical bounds of the corridor itself as shown by the associated occupied space voxel grid shown in the bottom right.

5.3.2.6 Generating Graph Edges

Once the set of safe vehicle poses N_i has been generated, the final step in creating the complete navigation graph G_i is to determine a set of edges E_i, representing safe trajectories between the poses of N_i. Since our target Quad-Rotor platforms taken to be Holonomic with respect to x,y,z and yaw (under the assumption of small roll and pitch angles), we take an edge $\mathbf{e} = (\mathbf{n}_a, \mathbf{n}_b) \in E_i$ to represent a direct interpolating trajectory between the poses $\mathbf{n}_a, \mathbf{n}_b \in N_i$. That is a trajectory which takes the vehicle in a straight line linearly interpolating between the positions of $\mathbf{n}_a$ and $\mathbf{n}_b$, while also changing the vehicle's yaw at a fixed rate to linearly interpolate between the orientations of $\mathbf{n}_a$ and $\mathbf{n}_b$. Such trajectories are reversible and as such G_i is an undirected graph.

Let us denote the position and orientation components of the pose $\mathbf{n}_a$ by the vector $\mathbf{x}_{n_a}$ and rotation matrix $\mathbf{R}_{n_a}$ such that $\mathbf{n}_a = (\mathbf{x}_{n_a}, \mathbf{R}_{n_a})$, and similarly let $\mathbf{n}_b = (\mathbf{x}_{n_b}, \mathbf{R}_{n_b})$. The set of positions $\mathbf{e}$ passes through is thus simply given by

$$\{\mathbf{x}_{n_a} + t(\mathbf{x}_{n_b} - \mathbf{x}_{n_a}) | t \in [0, 1]\}$$

The orientation of $\mathbf{n}_b$ relative to orientation $\mathbf{n}_a$ is given by the rotation matrix $\mathbf{R} = \mathbf{R}_{n_b}\mathbf{R}_{n_a}^T$. Any rotation matrix can be expressed in axis angle form $(\widehat{\mathbf{n}}, \theta)$, consisting of a specific axis of rotation $\widehat{\mathbf{n}}$ and an angle giving the magnitude and direction of rotation about the said axis. Let $M(\widehat{\mathbf{n}}, \theta)$ denote the rotation matrix associated with an axis angle rotation $(\widehat{\mathbf{n}}, \theta)$, and let the axis angle representation of $\mathbf{R}$ be denoted by $(\widehat{\mathbf{n}}_{\mathbf{R}}, \theta_{\mathbf{R}})$. All the intermediate orientations between $\mathbf{R}_{n_b}$ and $\mathbf{R}_{n_a}$ along the trajectory $\mathbf{e}$ are then

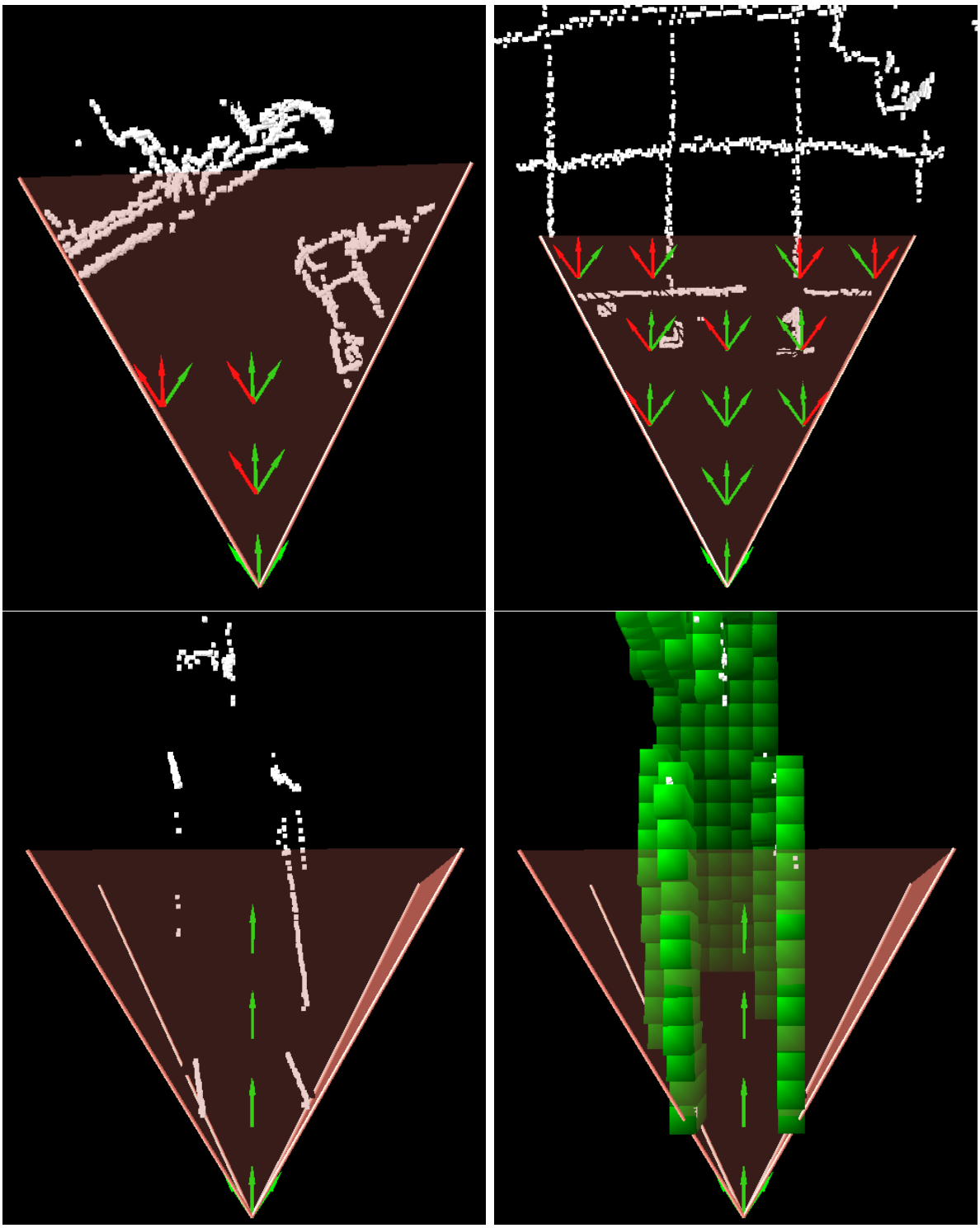


Figure 5.12: *Examples of safe vehicle poses generated from keyframe data. Safe poses are shown in green. Tested poses at which insufficient features are observed for reliable localisation are shown in red. Tested poses within occupied or unknown space are not shown. Bottom right shows the occupied space voxel grid from a section of corridor in green, showing how safe poses are restricted with the bounds of the observed corridor.*

of the form

$$M(\widehat{\mathbf{n}}_{\mathbf{R}}, t\theta_{\mathbf{R}})\mathbf{R}_{n_a} | t \in [0,1]$$

Thus the trajectory $\mathbf{e}$ passes through the set of poses given by

$$\{((\mathbf{x}_{n_a} + t(\mathbf{x}_{n_b} - \mathbf{x}_{n_a}), M(\widehat{\mathbf{n}}_{\mathbf{R}}, t\theta_{\mathbf{R}})\mathbf{R}_{n_a}) | t \in [0,1]\}$$

For such an interpolating trajectory between two poses $\mathbf{e} = (\mathbf{n}_a, \mathbf{n}_b)$ to be safe, it must both ensure collision avoidance and that the SLAM system is able to maintain tracking, from the features of K_i observed during the course of the trajectory. These are precisely the same criteria required of the set of safe vehicle poses N_i which form the nodes of the navigation graph being constructed G_i. Thus the same processes used to determine if a certain vehicle pose is collision free and can maintain SLAM tracking, can be adapted for determining the same criteria for specific a interpolating trajectory $\mathbf{e}$.

The uniform voxel grid associated with K_i described earlier was constructed to determine safe collision free regions of space for the vehicle. Thus determining if a trajectory $\mathbf{e}$ is collision free simply involves checking that the trajectory only passes through voxels flagged as known empty space. Since a direct interpolating trajectory simply involves moving in a straight line between the positions of two poses $\mathbf{n}_a$ and $\mathbf{n}_b$, the voxels which the trajectory passes through can quickly be determined using voxel ray tracing. After which all that remains is to check that each of the traced voxels has been flagged as known empty space.

If a trajectory $\mathbf{e}$ has been found to be collision free using the check described above, the next step to determine if $\mathbf{e}$ is safe is to ensure that the vehicle's sensor would always observe sufficient features from K_i to maintain SLAM tracking. In general it is sufficient to simply sample a subset of m poses from along the trajectory $\mathbf{e}$, and check that each one of these poses would in-fact maintain SLAM tracking. This set of m poses spaced out along the trajectory is defined as

$$\{(\mathbf{x}_{n_a} + \tau(\mathbf{x}_{n_b} - \mathbf{x}_{n_a}), M(\widehat{\mathbf{n}}_{\mathbf{R}}, \tau\theta_{\mathbf{R}})\mathbf{R}_{n_a}) | \tau \in \{\frac{1}{m}, \frac{2}{m}, ...\frac{m-1}{m}, 1\}\} \tag{5.3}$$

Naturally the greater the difference between the nodes $\mathbf{n}_a$ and $\mathbf{n}_b$ connected by the trajectory of $\mathbf{e}$, the greater the number of intermediate poses along the trajectory which need to be checked to ensure the trajectory is safe, i.e. the larger the value of m required in Equation 5.3. In practise we calculate this value based on the distance between nodes $\mid \mathbf{x}_{n_b} - \mathbf{x}_{n_a} \mid$, and the difference between their orientations as given by $\mid \theta_{\mathbf{R}} \mid$ the

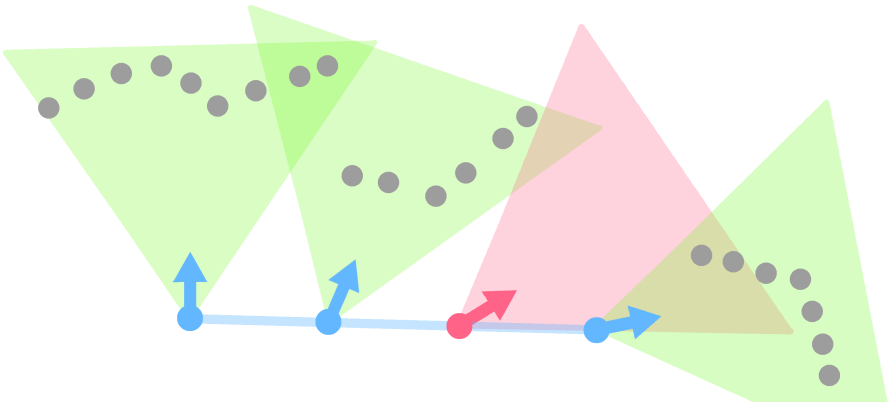


Figure 5.13: *Intermediate poses are sampled along a potential edge trajectory and checked to ensure each would result in SLAM localisation being maintained.*

magnitude of the angle component from the axis angle rotation $(\widehat{\mathbf{n}}_{\mathbf{R}}, \theta_{\mathbf{R}})$ between their orientations. The value of m used in Equation 5.3 is then simply given Equation 5.4, where the values of the constants were typically taken as $\mu = 0.2$ meters and $\gamma = 0.35$ radians derived experimentally and from the horizontal field of view of the Xtion RGB-D sensor used.

$$m = RoundUp\left(\frac{\mid \mathbf{x}_{n_b} - \mathbf{x}_{n_a} \mid}{\mu} + \frac{\mid \theta_{\mathbf{R}} \mid}{\gamma}\right) \tag{5.4}$$

The SLAM tracking checking process described previously in Section 5.3.2.4 which was used for determining the set of safe vehicle poses N_i, can be used once again for checking each of these poses along the trajectory in question. If each such pose does in fact pass this test then $\mathbf{e}$ has been determined to be a safe trajectory for the vehicle.

To generate the set of edge E_i, interpolating trajectories are tested between all pairs of nodes from N_i which are within some threshold distance from one another, typically taken to be 0.5 meters. All trajectories which pass both the collision and SLAM tracking checks described in the previous subsections are then added into E_i completing the creation of the navigation graph G_i for the keyframe K_i. Figure 5.14 shows examples of such navigation graphs created for keyframe data acquired during SLAM system operation.

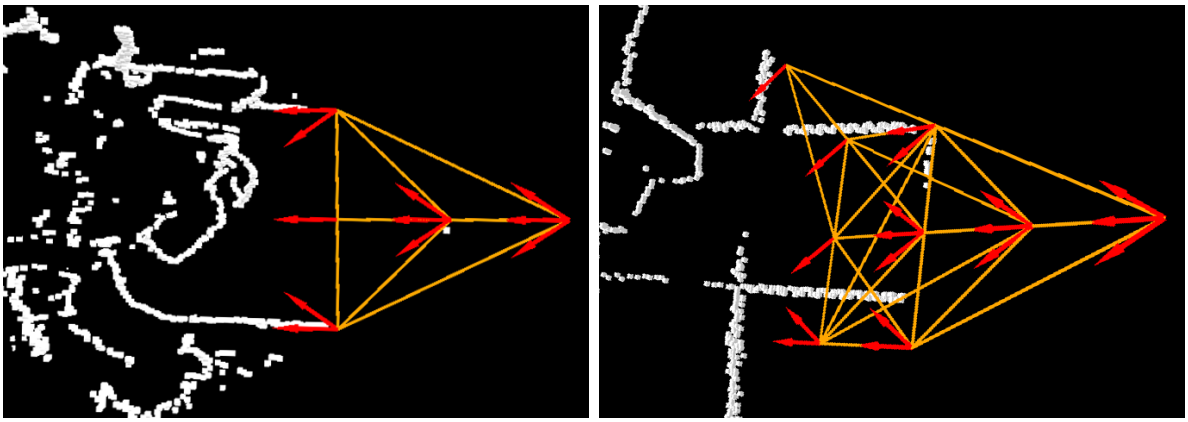


Figure 5.14: *Examples of keyframe navigation graphs, with graph nodes representing safe vehicle poses drawn in red and edges representing safe trajectories between such nodes drawn in orange.*

5.3.2.7 Graph Connecting Edges

Each keyframe navigation graph $G_i \in G$ provides a framework for safe path planning within a limited volume of empty space. Path planning on a larger scale however requires knowledge of how to safely transition between different keyframe navigation graphs during navigation. Switching between two navigation graphs G_i and G_j, requires that the vehicle follows some trajectory beginning at the pose of a node $\mathbf{n}_a \in N_i$ from one graph, and ending a different pose $\mathbf{n}_b \in N_j$ in another graph. Naturally such a trajectory must be contained within the overlap of known empty space for the keyframes K_i and K_j in order to ensure that it is collision free.

Such trajectories can be viewed as additional graph edges, connecting together the nodes belonging to different navigation graphs. This allows the various keyframe navigation graphs $G_i \in G$ to be combined into a single larger graph, allowing path planning to be conducted across multiple navigation graphs in a keyframe to keyframe based manner. We thus seek to determine a set of such "connecting edges" E_C connecting the nodes of different navigation graphs, and use G_{map} to denote the graph formed by the combination of these edges and the existing navigation graphs of G.

$$G_{map} = (\{N_0 \cup N_1 \cup ..., N_m\}, \{E_C \cup E_0 \cup E_1 \cup ..., E_m\})$$

Each connecting edge $\mathbf{c} \in E_C$ represents a simple interpolating trajectory between two graph node poses $\mathbf{n}_a \in N_i$ and $\mathbf{n}_b \in N_j$ where $i \neq j$. These interpolating trajectories are defined in exactly the same manner as those of the edges of the keyframe navigation graphs described previously.

As discussed in Section 5.2.4 the estimated global poses of the SLAM map keyframes may

be highly inaccurate, and thus are not a reliable source of information for determining valid connections between keyframe navigation graphs. In contrast loop closures between any pair of keyframes generally provide a far more reliable estimate of the relative pose between those keyframes. Thus connecting edges are only added between navigation graphs where there exists a loop closure between their associated keyframes providing a reliable estimation of their relative pose to one another.

Specifically whenever a new loop closure is detected between two keyframes K_i and K_j it provides an estimate of the relative pose $\mathbf{R}_{i,j}$ between said keyframes. This relative pose is then used in generating interpolating trajectories between all pairs of nodes $(\mathbf{n}_a \in N_i, \mathbf{n}_b \in N_j)$ within a threshold distance of one another (typically taken to be 0.5 meters).

Each such generated trajectory is then a potential connecting edge $\mathbf{c} = (\mathbf{n}_a \in N_i, \mathbf{n}_b \in N_j)$ between the two navigation graphs G_i and G_j. However this trajectory first needs to be tested to ensure it is both collision free and guarantees SLAM tracking and localisation. This can be achieved by using the same testing procedures used previously in Section 5.3.2.6 for testing the trajectory of a potential navigation graph edge. However for such a connecting edge trajectory $\mathbf{c} = (\mathbf{n}_a \in N_i, \mathbf{n}_b \in N_j)$, the test to ensure $\mathbf{c}$ is collision free must to be conducted twice for both of the navigation graphs G_i and G_j, and similarly the test ensure SLAM localisation has to be conducted for both the key-frames K_i and K_j. This is to ensure that the edge can be used to transition from either keyframe to the other, as the edge when used for path planning is assumed to undirected.

All connecting edges $\mathbf{c} = (\mathbf{n}_a \in N_i, \mathbf{n}_b \in N_j)$ which pass these tests are then added to the set of connecting edges E_C, expanding the graph G_{map} and providing a means to traverse between the navigation graphs G_i and G_j. Figures 5.15 and 5.16 show examples of such a G_{map} navigation graph, with each keyframe navigation graph drawn in a separate color for clarity, and the connecting edges of E_C drawn in white. Another example of such a graph along with the associated SLAM map is shown in Figure 5.16, in this instance the RGB-D sensor is rotated 360 degrees within an office environment with the generated graphs branching down the corridors while avoiding the obstacle boundaries.

5.3.3 Keyframe Planning

The steps laid out in the previous sections lay out the process of constructing navigation graphs for each keyframe of the SLAM map, and also the generation of connecting edges

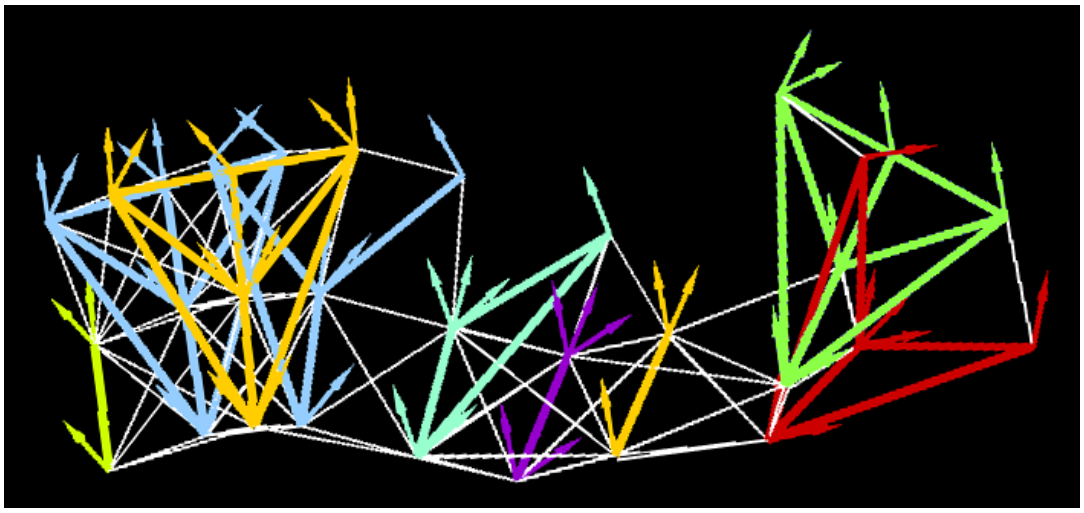


Figure 5.15: *Combination of keyframe navigation graphs (each shown in a different colour) from largely translational motion. Connecting edge between graphs are shown in white.*

providing a means to conduct navigation across these graphs. At any point in time the current graph used for path planning in navigation denoted G_{map} is formed by the combination of all nodes and edges from all navigation graphs $G_i \in G$.

As both additional keyframes and loop closures are added to the SLAM map new navigation graphs and connecting edges are generated as illustrated in Figure 5.17. In practice these tasks are performed on a separate threads from the two others conducting SLAM tracking and map optimization respectively (brining the total number of threads to three) in order to ensure that the SLAM system's performance is not compromised.

With the formation of the navigation graph G_{map}, all that remains is the problem of determining paths through G_{map} between any two connected nodes. Such a task can be performed using any standard graph search algorithm such as Dijkstra's [18] or the A* Algorithm [37].

5.4 Results

This section presents and evaluates a sample of the trajectories generated by the proposed path planning approach in different scenarios. The trajectories themselves are illustrated in blue, with the poses associated with the graph nodes visited along the trajectory drawn in red. Where shown, the occupied space voxels associated with each keyframe navigation graph are drawn in green.

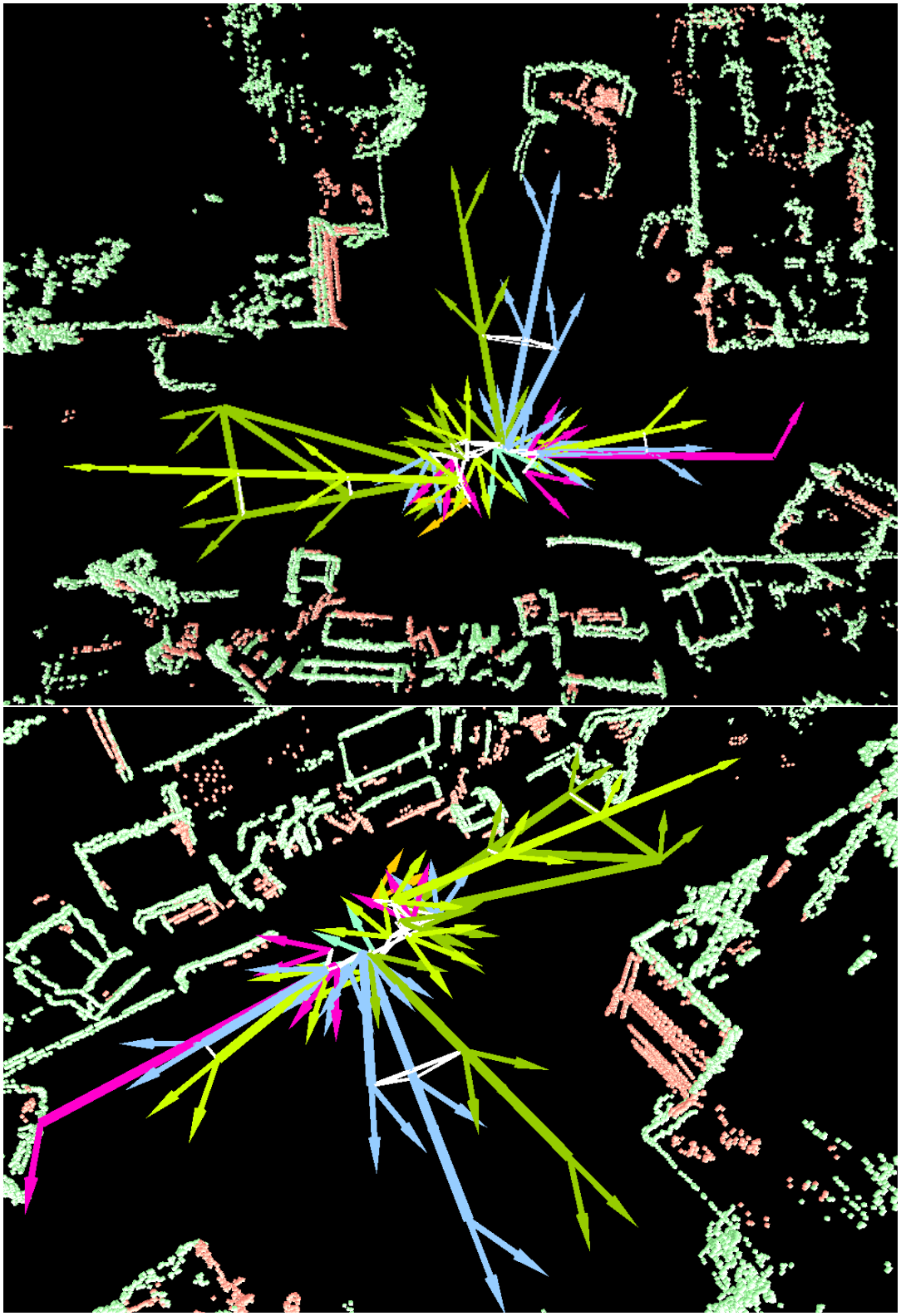


Figure 5.16: *Combination of keyframe navigation graphs (each shown in a different colour) from largely rotational motion. Connecting edge between graphs are shown in white.*

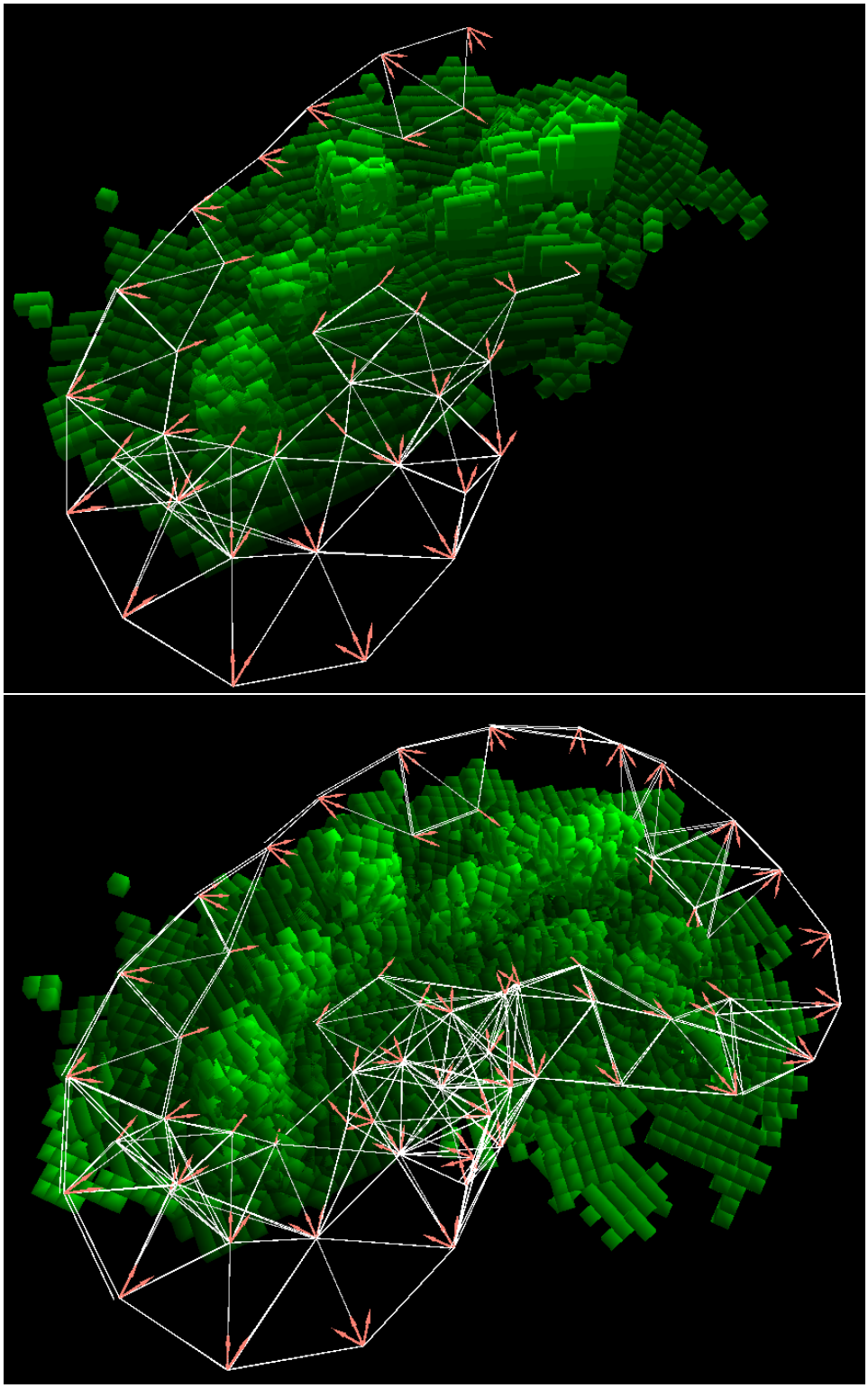


Figure 5.17: *Creation of a keyframe based navigation graph in a flying arena as described in 5.4.2. Occupied space voxels are drawn in green, while graph edges and nodes are drawn in white and pink. It can be seen that loop closure occurs as the vehicle moves back to around its starting location.*

In practice the proposed path planning is implemented on a separate thread from the sensor tracking and map optimization threads of the SLAM system, in order to ensure the performance of one system does not adversely affect the other. The generation of trajectories consists of performing a simple A* graph search through the connected keyframe navigation graphs. The computational cost of such searches may vary greatly depending upon the graph and the start and goal node locations in addition to the graph itself, however typically the computation time of such a search is far below 100ms. This is significantly less time than is required to traverse such a trajectory, and further such trajectory searches do not need to be frequently conducted.

5.4.1 Hand-held Results

This subsection present a sample of trajectories generated in environments which were mapped by hand using a standard laptop (2.60GHz Intel Core i5-3230M (2013), 4GB RAM, running Ubuntu 14.10) and Asus Xtion RGB-D sensor. Once mapped a destination was selected for the planners goal, and the generated trajectories were followed by hand in order to test the planner's ability to ensure SLAM localisation.

Figures 5.18 and 5.19 illustrate different trajectories generated to navigate between the different rooms of a house. Within the rooms themselves there is an abundance of detail such that there is seldom any sensor pose that would not receive sufficient information for SLAM localisation to be maintained, and thus there is effectively no sensor information constraint throughout many of these rooms. The hallways connecting these rooms however largely feature blank textureless walls, and as a result, the sensor is constrained to face down such hallways in order to avoid facing such blank surfaces and potentially losing SLAM localisation. Further trajectory examples are given in Figures 5.20 and 5.21.

5.4.2 MAV Hardware and Software Set-up

In addition to testing the proposed path planning approach by hand, we also performed experiments using a quad-rotor MAV equipped with an Asus Xtion RGB-D sensor. The vehicle also carried an on-board computer used to run both the proposed path planning software and edge based RGB-D SLAM system presented in Chapter 2. However this high payload (around $1.6kg$ take off weight) made vehicle highly unstable and difficult to control. Combined with severe latency issues we encountered in communication between the on-board computer and flight controller, we were not confident in conducting

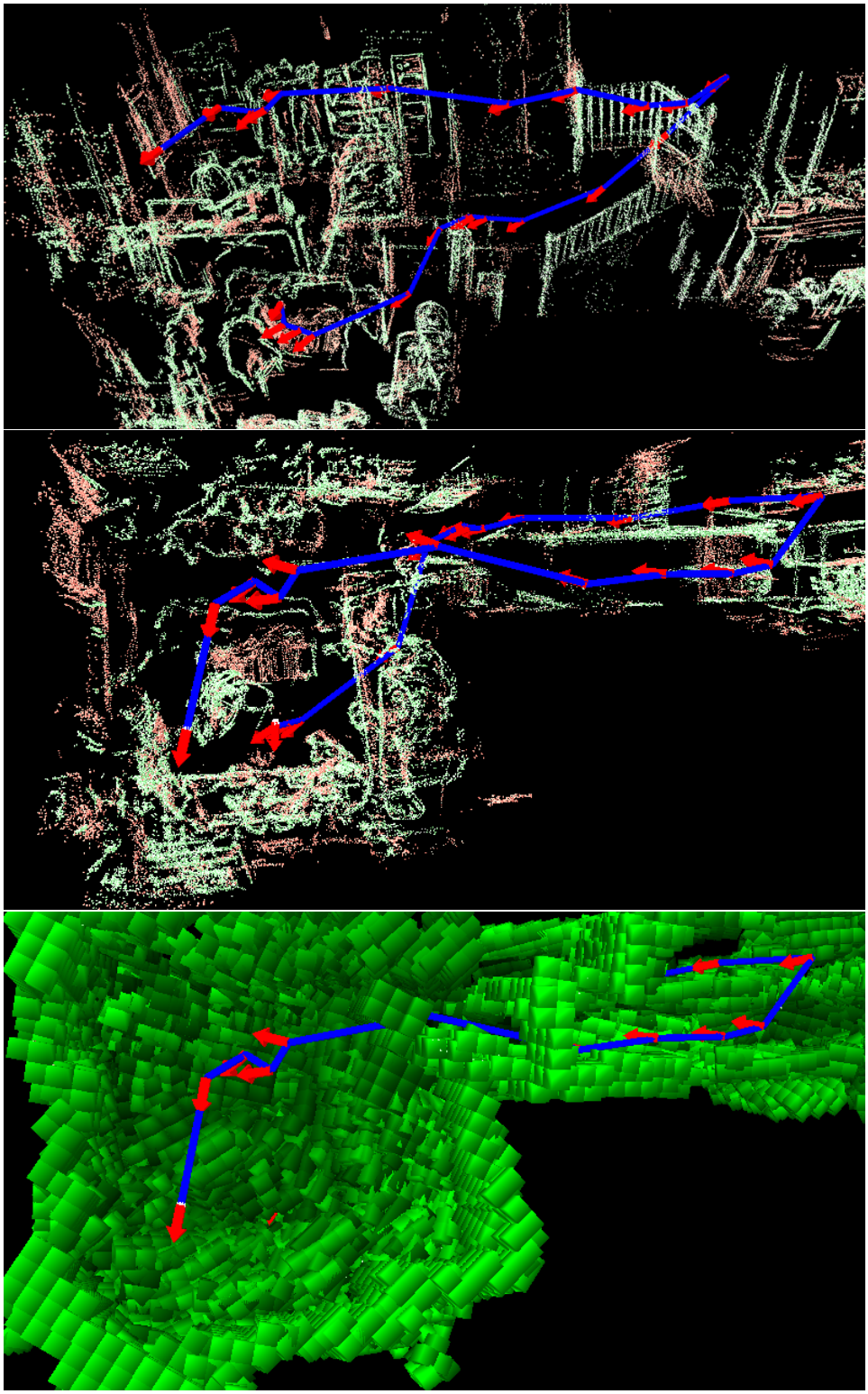


Figure 5.18: *Planning between rooms of an apartment.*

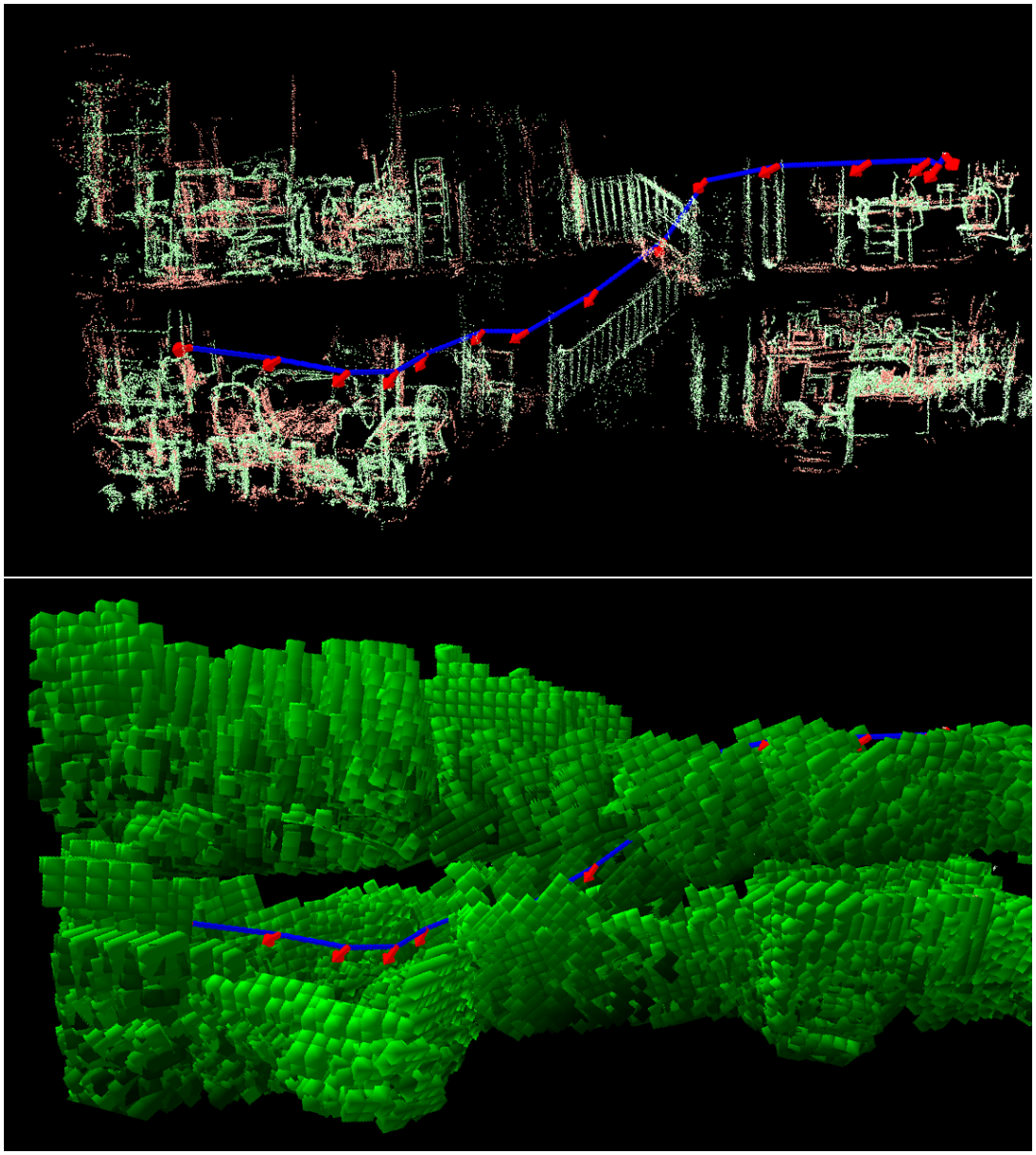


Figure 5.19: *Further planning between rooms of an apartment.*

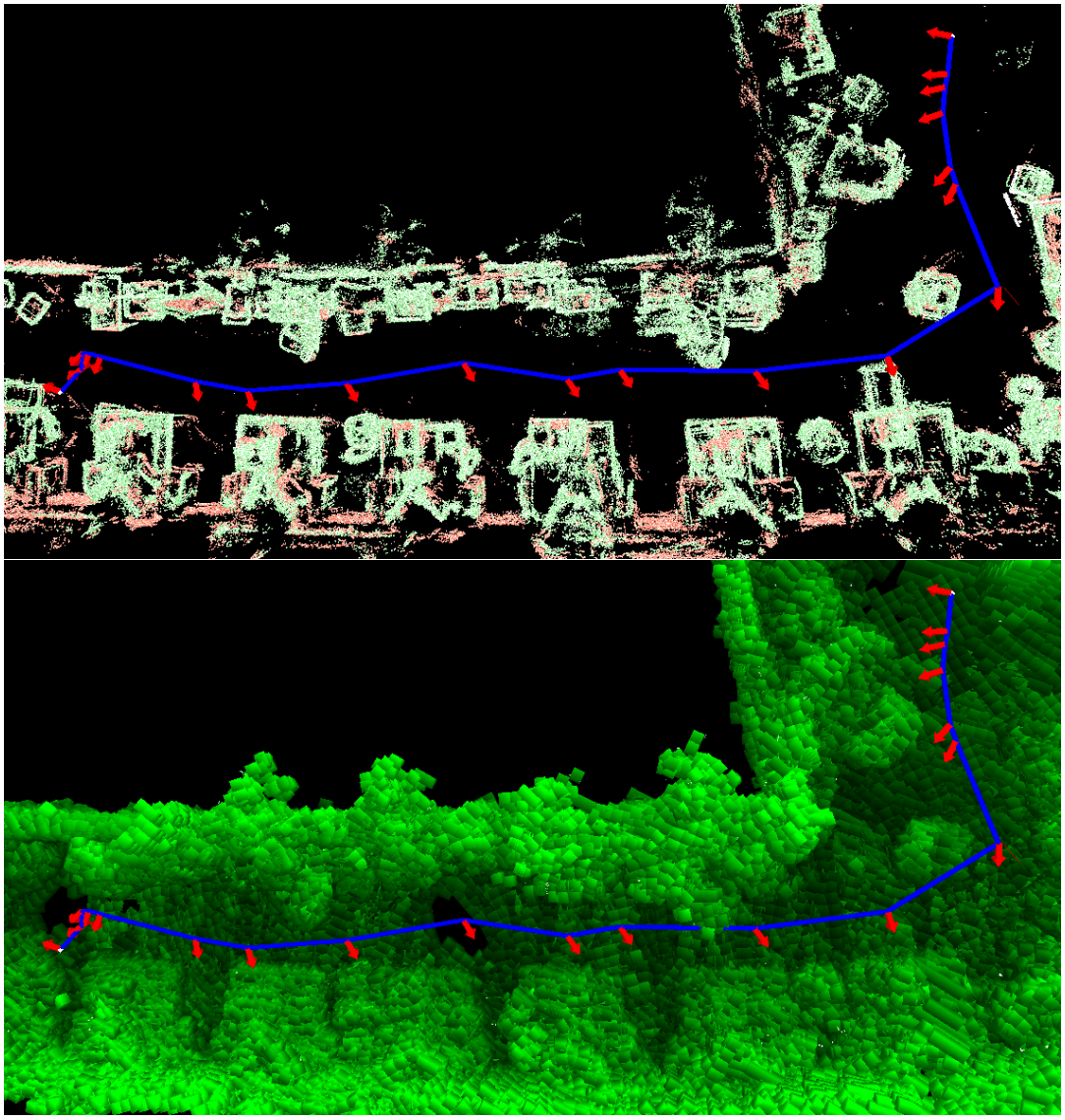


Figure 5.20: *Planning within an office space.*

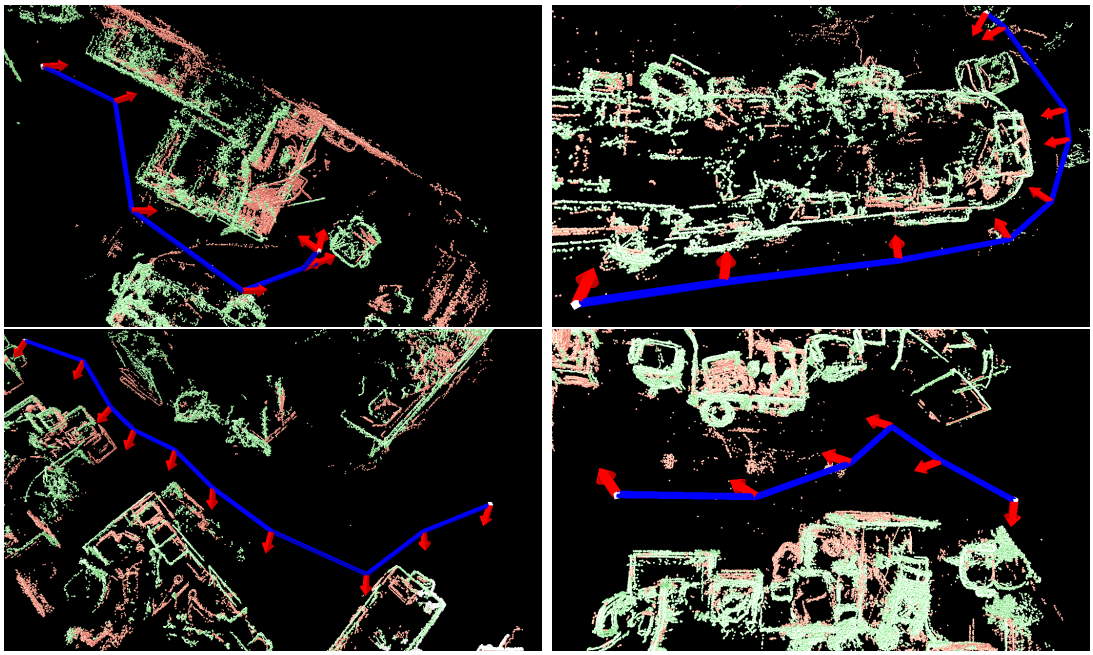


Figure 5.21: *Planning across various maps.*

prolonged flight relying on the SLAM system (running upon the on-board computer) for stabilisation. We thus instead made use of a simple ground based PID controller, using motion capture for control feedback to transmit commands to the vehicle's flight controller. The path planning and SLAM software running on-board the vehicle however had no access to this motion capture data, our path planning software simply transmitted commands in the form of 6DOF poses relative to the vehicle's reference frame to the ground based pc running the PID control. The PC then converted these poses from the vehicle's body frame to the global frame of the motion capture system before feeding them into the PID controller. Using this set-up with communication between a ground based PC and the on-board computer of the MAV we could command the vehicle navigate to a pose within its constructed SLAM map. The vehicle would then calculate and follow a valid trajectory avoiding obstacles and ensuring SLAM tracking.

We now briefly describe the components of this set-up in greater detail.

5.4.2.1 Vehicle

Two Asctec Pelican quad-rotor platforms as shown in Figure 5.22 were available to conduct flying experiments, both carrying an on-board computer, flight control computer, and a single ASUS Xtion RGB-D sensor. The on-board computers of these vehicles consisted of standard PC hardware in a compact form factor motherboard. One Pelican vehicle was equipped with a 1.86GHz Intel Core2Duo SL9400 (2008), while the other

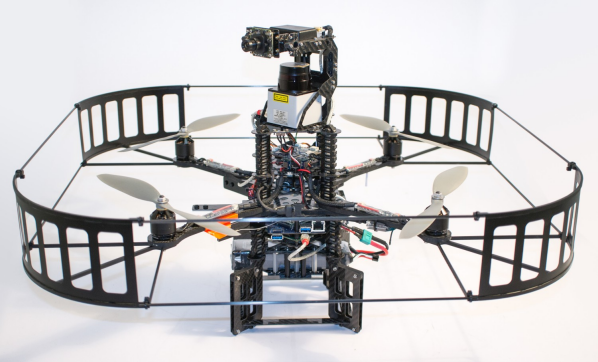


Figure 5.22: *AscTec Pelican.*

was equipped with a 2.10GHz Intel i7-3612QE (2012), our software was seen to achieve real-time 30Hz performance on both these platforms making them largely identical for our testing purposes. Both on-board computers were also equipped with 8GB ram and a solid state drive (SSD). A stripped down version of the Ubuntu Linux distribution was installed on these SSDs, allowing us easily port over our SLAM and navigation software which was developed on a desktop environment.

5.4.2.2 Motion Tracking

We make use of a commercial Vicon optical motion capture system, to provide reliable ground-truth measurements of the vehicle's full 6DOF pose. Such optical tracking systems make use of a set of cameras at known relative poses to one another. Rigid bodies that are to be tracked, typically must be visible from two or more camera's at once in order to get reliable measurements. The tracking volume is defined as the volume of space where such reliable tracking is available, and is determined by the overlap between the view frustums of multiple cameras. However obstacles in the environment will also occlude certain volumes of space from certain cameras, reducing the total tracking volume.

Vicon specifically is a passive marker based system using infra-red cameras, in which simple retro reflective markers are attached in a known configuration to the rigid bodies that are to be tracked. These markers are then illuminated with infra-red light, showing up in the camera images as bright points, from which it trivial to distinguish such makers from the rest of the scene by simply applying an intensity threshold to the images. These images are sent back to a desktop computer and processed in order to determine the 3D positions of the observed markers via triangulation, resulting in a sparse point cloud of marker positions. The 6DOF poses of a body being tracked can then be determined by

identifying a body's marker configuration within this sparse point cloud. Typically a method such as RANSAC [28] is used to perform this task, using the body's previous estimated pose to initialize the search for it marker configuration. It is important to avoid any sort of spacial symmetry when deciding upon a bodies marker placement, as such symmetries obviously will introduce ambiguities into the process of calculating the bodies 6DOF pose from the sparse cloud of marker positions. This motion tracking set-up is able to provide millimetre accuracy level tracking at $100Hz$ provided the body is within the reliable tracking volume.

5.4.2.3 Robot Operating System ROS

The Robot Operating System (ROS)[83] is an collection of open source software that aims to make development on robotic systems both more efficient and accessible, through a set of frameworks and tools that can be used to carry out many common functions and tasks such as interprocess communication, visualization, data logging, debugging and diagnostics. API facilities are also included making it simple to incorporate your own software into ROS, allowing it communicate with other software packages with ROS functionality, or make use of the set of core tools ROS provides.

5.4.2.4 ROS Communication

One of the most central aspects to ROS is the messaging passing system used for communicating between different ROS processes, running across various hardware platforms and computers. A ROS process can be any piece of software which uses the ROS API to enable such communication. Communication between such processes is performed using so called ROS topics. Each topic is identified by a specific name and may represent any type of information, such as data from an IMU sensor, or a camera's estimated pose from a SLAM system. ROS processes may then both publish data under a topic name, or subscribe to said topic to receive any data published under that topic name by other ROS processes. In this way various sensors and software processes running within the ROS framework may send and receive information from one another. The core ROS framework handles the task of actually ensuring data published on each topic is communicated to the relevant subscribing processes, even if such information has to be transmitted over wired or wireless local network connection to different machines. This communication set-up can be visualized as a graph structure, in which each ROS process is a node, and directed edges between nodes represent instances where one node has subscribed to a specific ROS topic, which the other node is publishing data for.

Such communication allowed us to transmit and receive data to and from the Pelican's on-board computer and a ground based PC. This enabled us to both command the pelican to path to a specific location in the SLAM map, and also visualize the SLAM and path planning software on the ground based PC while they were running on the Pelican's on-board computer. Examples of this are shown in Figure 5.23, where the SLAM system's map, estimated sensor pose, and estimated sensor trajectory are visualized in real-time from data being transmitted from the Pelican and using the ROS RVIZ tool.

5.4.3 MAV Results

MAV experiments were conducted within an indoor flying arena as can be seen in Figure 5.23, SLAM was initialized once the vehicle was in flight. The vehicle was then flown manually to acquire a map of the arena, while the path planning software simultaneously constructed the keyframe based navigation graph to be used for navigation. Once this step was complete the vehicle was switched over to being controlled by the ground based PID controller, whose setpoint was being controlled by the path planning software running on-board of the MAV.

Using the visualization of the vehicle's SLAM map displayed on the ground based PC we could select a desired pose for the vehicle within the map, this was then sent to the path planning software on-board the vehicle via ROS communication. The planner then determined a trajectory between the vehicle's current estimated pose, to the graph node closest to the desired pose. The path planning software would then use the SLAM system's current estimated vehicle pose to continuously calculate a pose which advance the vehicle slightly further along this desired trajectory. This pose was then communicated back to the ground based PC and used to update the setpoint of the PID controller, resulting in the vehicle advancing along the desired trajectory.

Figure 5.24 show a map created on-board of the MAV during flight along with some example trajectories. In this scenario obstacles were placed in the center of the flying space and the generated trajectories constrain the orientation of the vehicle to face towards nearby obstacles in order to ensure the RGB-D sensor is providing sufficient data to the SLAM system for localisation. Figure 5.25 shows the same flying arena but enclosed with obstacles along the boundaries of the motion capture system's tracking volume. It can be seen that generated trajectories avoid making the vehicle face directly across the empty arena from one side to the other as obstacles on the other side are deemed to far to provide reliable depth measurements. Instead trajectories skirt around the arenas edge where the vehicle can always be within range of obstacles to acquire

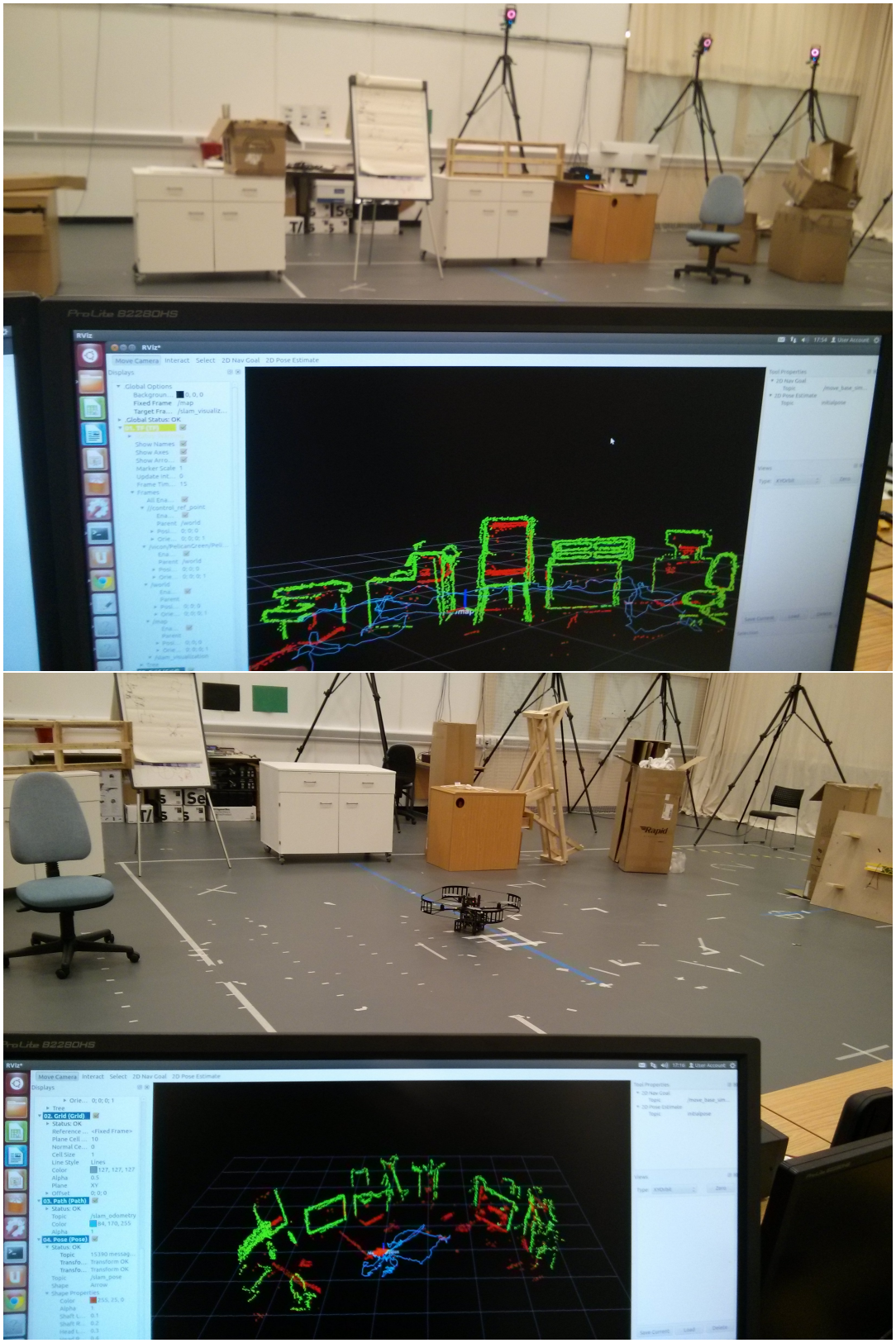


Figure 5.23: *Real-time visualizations of SLAM data received from the Pelican vehicle.*

accurate depth measurements.

The effect of the constraint on generated trajectories to ensure SLAM localisation is maintained was also examined for the enclosed flying arena. Figure 5.26 shows a comparison of two trajectories illustrating the result of removing this constraint. With the constraint active the trajectory generated across the arena forces the vehicle to both face and stay within a reasonable distance to the enclosing obstacles, thus ensuring that the vehicle's RGB-D sensor always acquires a large amount of visual and geometric information which the SLAM system can utilize. By comparison with this information constraint removed the generated trajectory takes a far more direct route across the arena but in doing so faces the vehicle's RGB-D sensor directly across the arena, resulting poor quality geometric information being produced by the sensor due to the far distance between it and the observed obstacles due to the nature of the sensor [57]. Using such data resulted in considerable noise being introduced into the SLAM system's estimated pose and even loss of tracking at certain points, significantly degrading the vehicle's ability to perform navigation.

5.5 Conclusions

This section proposed a keyframe centric path planning approach for RGB-D SLAM based navigation, specifically the system outline previously in Chapter 2. The approach is "SLAM aware" in that produced trajectories do not only ensure obstacle avoidance, but also that the vehicle's RGB-D sensor observes sufficient information when following said trajectory to maintain SLAM tracking and localisation. The approach constructs many small configuration spaces, each associated with a different keyframe of the SLAM map. Navigation graphs are generated for each keyframe's configuration space, providing collision free trajectories which also ensure SLAM localisation from the features of the associated keyframe. Safe connections between these graphs are determined from detected loop-closure constraints between keyframes, and provide a means for the planning process to transition between the different keyframe configuration spaces. In this way the many connected keyframe configurations space, form a larger configuration space which may be used to conduct navigation across the SLAM map. Further changes to the SLAM map only affect the connections between such graphs, avoiding the need to recalculate the entire configuration space. Results of generating various trajectories in different mapped environments are presented demonstrating the behaviour of the planner. These include trajectories generated for an AscTec Pelican quad-rotor to navigate within an indoor flying arena. In this scenario both path planning and SLAM were con-

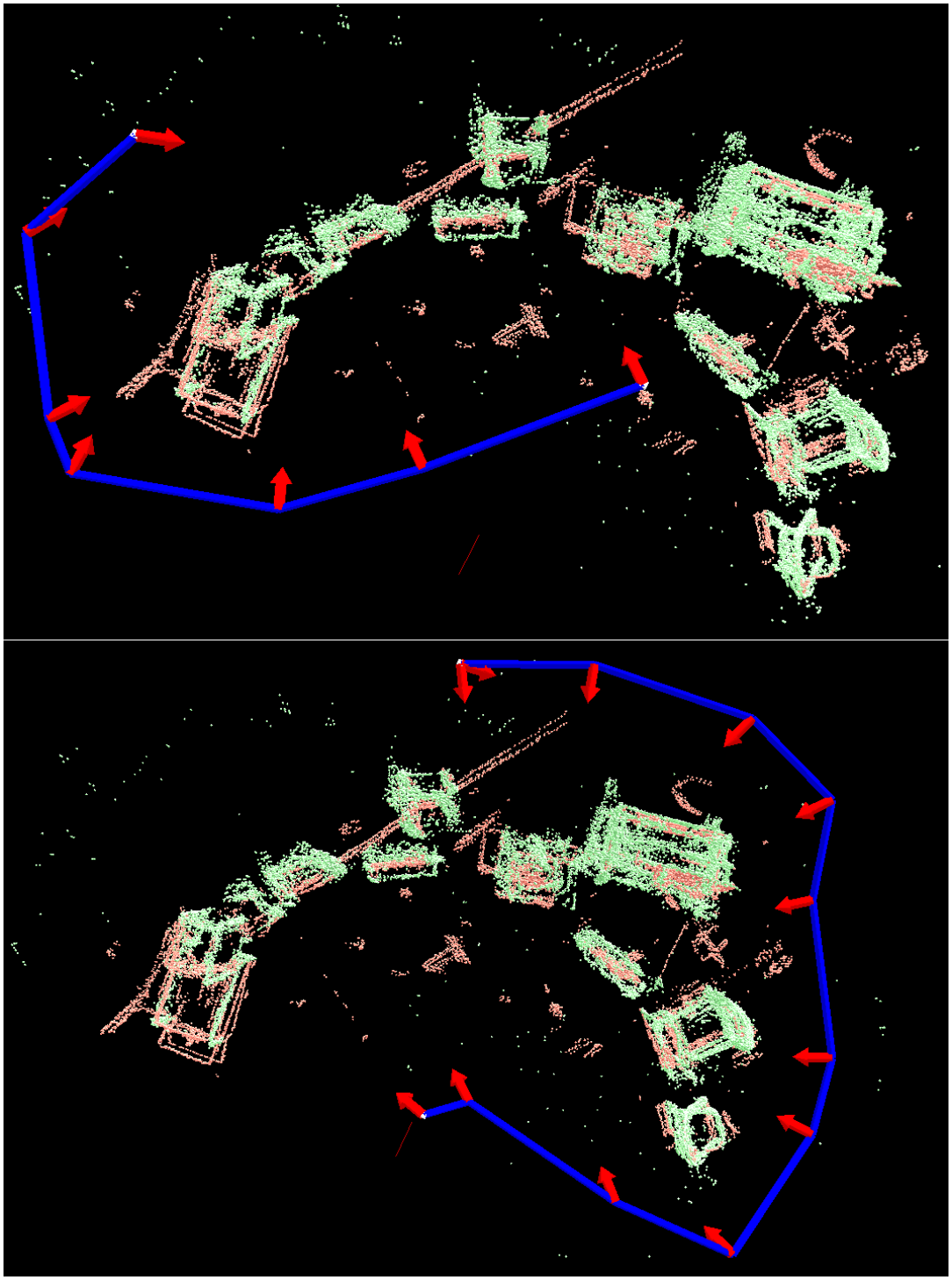


Figure 5.24: *Examples of trajectories generated within an obstacle filled flying arena.*

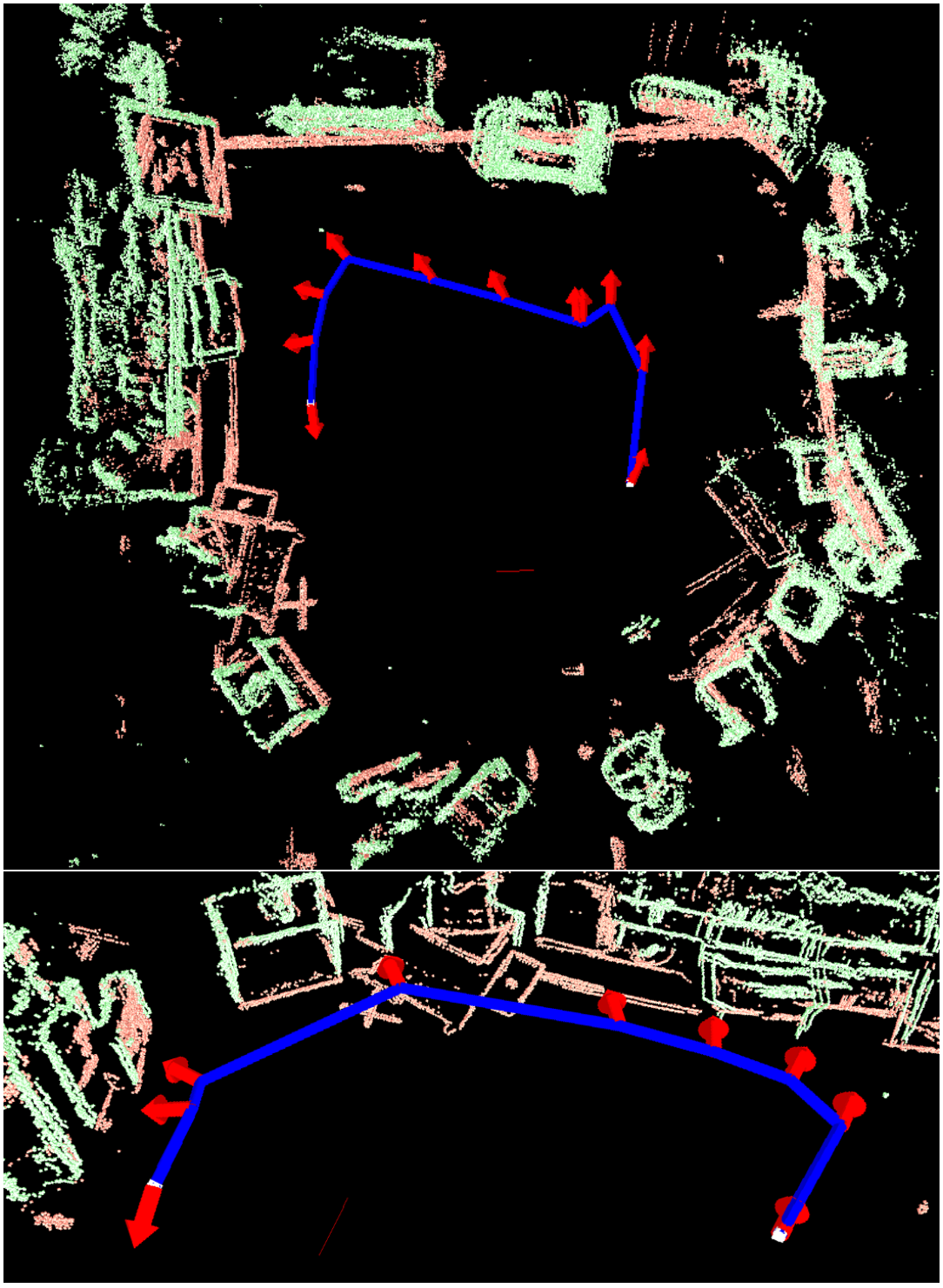


Figure 5.25: *Examples of trajectories generated within a flying arena with obstacles placed along the boundaries.*

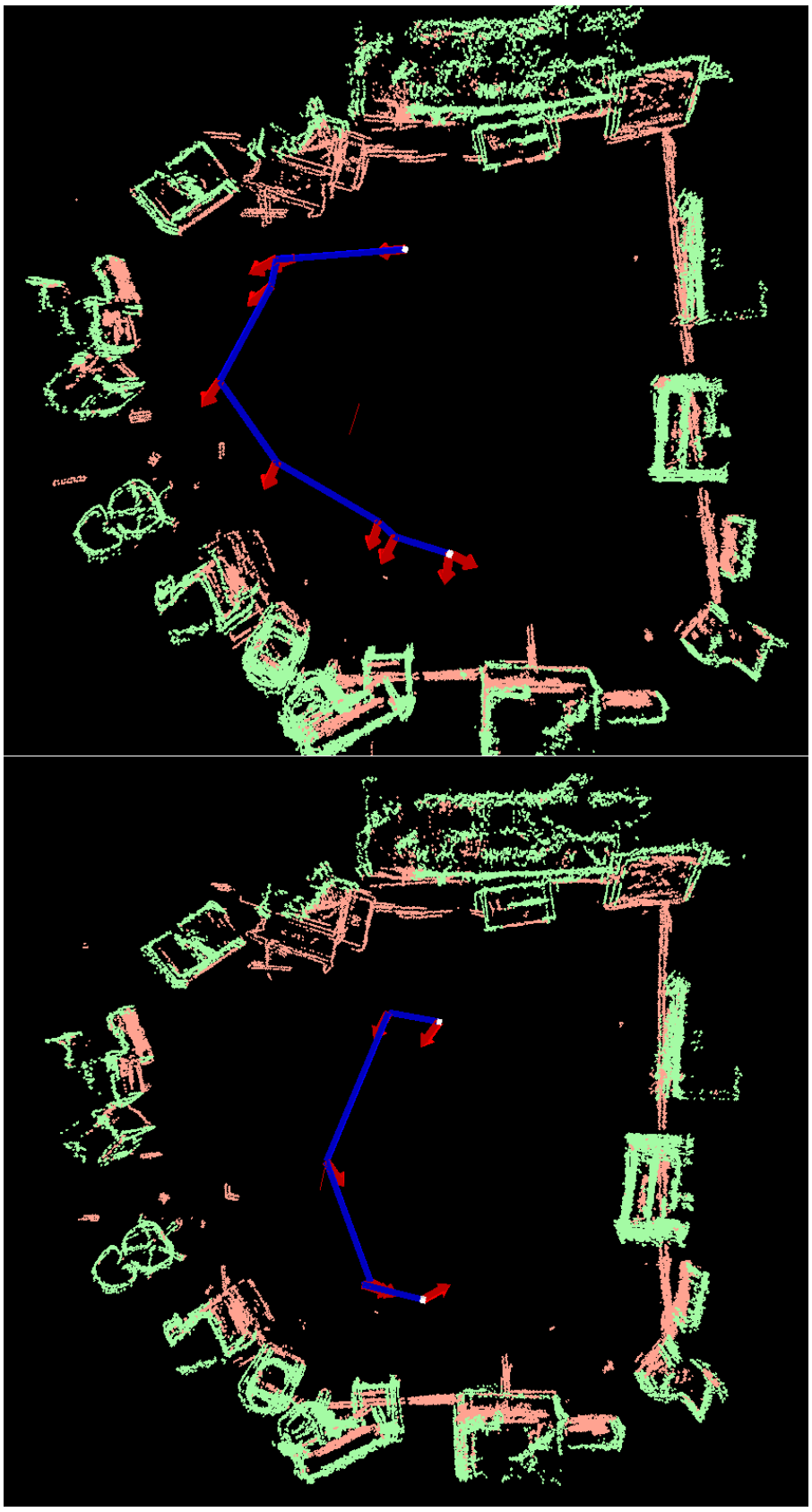


Figure 5.26: *Comparison between trajectories with (top) and without (bottom) enforcing SLAM localisation constraints.*

duct on-board the vehicle itself using its on-board computer. The path planning software generated a stream of pose setpoints based on the desired trajectory and estimated vehicle pose provided by SLAM. These setpoints were communicated to a ground based PC and fed into a PID controller which directly controlled the vehicle by transmitting commands to its flight controller.

Chapter 6

Conclusions

6.1 Chapter Summary

- Chapter 2 presented a novel edge based RGB-D SLAM system. This system simultaneously makes use of edges extracted from both the Depth and RGB components of the RGB-D video stream. This allows the system to continue to operate in many situation with either no visual information (such as from a lack of illumination), or a lack of geometric features (such as a scene of flat textured surfaces). The system back-projects detected edge pixels to form semi-dense point clouds, which are then used to achieve fast registration between RGB-D frames via ICP as described in Section 2.6. The evaluation of this edge cloud based ICP registration in Section 2.6.3 demonstrated that such an approach is highly robust to uniform downsampling, with no loss of registration accuracy even when x10 uniform downsampling is applied. It was also shown that this edge cloud based registration significantly outperforms naive RGB-D cloud IPC, having both a lower computational cost per ICP iteration and requiring fewer iterations to converge. The proposed SLAM system itself was evaluated in Section 2.9.1 and demonstrated compelling results. Compared to the similar CPU based RGB-D SLAM systems at the time of writing ([12], [24]), our proposed approach was seen to have a significantly lower computational cost, being able to process RGB-D video sequences well within real-time. Despite this our system was also still able to achieve a accuracy comparable to or exceeding these most costly SLAM approaches, thus making it viable for use in autonomous navigation on robotic platforms.
- Section 2.4 presented our proposed method of occluding depth edge detection,

exploiting the temporal similarity between sequential RGB-D frames. This method limits the search for edge pixels in the depth image to areas around which such pixels were detected in the previous frame. The intuition behind this being that the location of edges will not have significantly changed in the time between frames. Additional random regions are also searched to facilitate the detection of new edges not present in the previous depth image. To the best of our knowledge this was the first instance of such temporal similarity being exploited specifically for occluding depth edge, and at the time of writing is significantly faster than other demonstrated approaches such as [12].

- Chapter 3 developed an extension to the proposed edge cloud based SLAM system of Chapter 2, making use of 3D linear features extracted from the semi-dense edge clouds. The intuition behind this was that linear features could be used to form a reasonable approximation of such semi-dense edge clouds in the majority of situations, and in doing so greatly reduced the amount of data involved in registration (down from thousands of points to tens of lines). We presented both a method of 3D linear feature extraction from semi-dense edge points in Section 3.2, and an ICP based method of registration between semi-dense edge clouds and said 3D linear features in Section 3.3. Evaluation this proposed edge cloud and line feature hybrid SLAM system was conducted using the same approach used previously in Section 2.9. Results showed that this hybrid system was significantly faster than the purely edge cloud based system proposed in Chapter 2, reducing the total runtime on each RGB-D sequence by around 30 %. However, this came at the expense of accuracy due to the linear feature merely being approximations of the underlying edge clouds, with the purely edge cloud based SLAM system displaying superior accuracy in each sequence.

- Chapter 4 introduced a belief space planner, motivated by the notation that for any autonomous vehicle to accurately follow a desired path it must have access to an accurate estimate of its own state at all times i.e. it must maintain localisation. The planner was evaluated in generation of paths for simulated MAV relying upon bearing measurements to static landmarks (or "beacons") within the environment for localisation, in a scenario somewhat analogous to point feature based monocular visual slam. Section 4.5 demonstrated results obtained for various scenarios, each demonstrating how robust paths produced by the planner would deviate significantly from the shortest path where necessary, in-order to ensure that localisation was maintained via bearing measurements to the beacons within the environment. By comparison in many of these scenarios the shortest path would result in high

vehicle state uncertainty (i.e. loss of localisation) greatly increasing the risk of the vehicle colliding with the environment, and thus demonstrating how the proposed planner would be the preferable method of navigation.

- Chapter 5 presented our keyframe centric path planning approach, for vehicles making use of the previously proposed edge based SLAM system for localisation and obstacle avoidance, specifically MAVs platforms. For each keyframe the planner constructs a navigation graph of collision free trajectories which also ensure that the vehicle's RGB-D sensor observes sufficient features from said keyframe for the SLAM system to localisation. This sensor information constraint is vital to ensure the vehicle can follow the desired trajectory. Connections between these keyframe navigation graphs are then determined using the detected loop closures between keyframe pairs (as opposed to using the estimated global poses of them keyframes themselves), resulting in the construction of a navigation graph spanning the entire SLAM map. Such a navigation graph may still be constructed even in sceneries where the SLAM map has become globally inaccurate or inconsistent due to sensor drift and tracking error since it is based upon the consistent local connections between keyframes. The planner then generates trajectories through this navigation graph, passing through various keyframe reference frames. The vehicle follows such trajectories by navigating across different keyframe navigation graphs, at each navigating within the reference frame of the associated keyframe and thus never using its estimated global pose for navigation. Section 5.4 demonstrated results from this planner across various environments, for both maps constructed with a hand-held RGB-d sensor and with an RGB-D sensor carried on-board of a quad-rotor MAV platform inside of an indoor flying arena. For MAV experiments all computation (i.e. both SLAM system and path planning) was conducted in real-time upon its on-board computer. A computer on the ground was then used to command the MAV to move to a known position within the environment, with the MAV itself generating the trajectory. We observed that the MAV could successful navigate the arena in this manner, additionally when the sensor information constraint was removed from the trajectory generation process the MAV would encounter scenarios in which SLAM localisation was lost due to poor sensor information, indicating the importance of this aspect of the planner. Significant loop closures and map changes also did not adversely affect the vehicle during navigation, this is due to the nature of the planner which generates trajectories spanning across various keyframe reference frames, instead of a single global reference frame within which the SLAM map is subject to change.

6.2 Contributions Summary

This section summarizes the contributions of the work presented within this thesis.

- A novel edge based RGB-D SLAM system, making simultaneous use of both depth and RGB edge features. By back-projecting such edge pixels to form semi-dense point cloud fast registration between RGB-D frames can be achieved using ICP. An evaluation of the edge cloud based ICP registration utilized demonstrated that such an approach is both highly robust to uniform downsampling, and also significantly outperforms naive RGB-D cloud based registration in terms of both accuracy and computation time. An evaluation of the proposed edge based RGB-D SLAM system itself showed that it is able to achieve similar or better levels of accuracy than a number of comparable systems, while also having a significantly lower computational overhead, processing RGB-D sequences faster than real-time on a standard CPU core.
- A method of occluding depth edge detection exploiting temporal similarity between sequential RGB-D frames, to the best of our knowledge the first such method to do this, and at time of writing is significantly cheaper computationally compared to other published depth edge detection methods.
- A map optimization approach based on iterative relaxation in accordance to the relative pose constraints given by detected loop closures.
- A hybrid SLAM system making use of both 3D linear features and edge point clouds, along with methods of linear feature extraction from semi-dense edge clouds, and an ICP based method of registration between edge clouds and line features.
- A belief space planning algorithm, generating robust paths for simulated MAVs which attempt to ensure localisation can be maintained at all times from bearing measurements to fixed landmarks in the environment.
- A SLAM aware path planning method for MAVs, generating trajectories that are both collision free and also ensure the vehicle's RGB-D sensor observes sufficient information for SLAM to maintain operation at all times. This method is demonstrated both generating trajectories in environments where a RGB-D sensor was carried by hand to perform mapping, and where the sensor was mounted to a quad-rotor platform which was commanded from a ground based pc to navigate to known locations in the environment using this path planning approach.

6.3 Future Directions

6.3.1 Omnidirectional Sensing

For both SLAM and autonomous navigation, the use of a single limited field of view RGB-D camera sensor can be viewed as being a major limiting factor. In many environment great care must be taken to ensure the sensor is never positioned such that insufficient visual information is observed for localisation and sensor tracking. This problem can be addressed by obtaining a greater amount of visual information from the surrounding environment, across a wider field of view, either by using multiple sensors or an omnidirectional sensor as has been demonstrated in [11],[87],[98],[91],[36]. This would both greatly increases the robustness of the SLAM system, and significantly decrease the constraints imposed on the path planning process regarding the acquisition of sufficient sensor information.

6.3.2 Modern Small Form Factor Computer Hardware

The MAV used in our indoor flying experiments was equipped with an on-board computer consisting of standard desktop PC hardware. Despite this hardware being packaged into a small form factor it contributed significant weight and size to the vehicle, impacting its stability, agility and maximum flight time. Advances in small form factor computing, primarily driven by the smartphone market have since produced devices significantly smaller in size and weight, which have been shown to still have sufficient computing power to conduct SLAM among other computer vision tasks ([89],[82],[107]). Additionally such devices typically include GPU hardware, the parallel processing abilities of which could be used to accelerate many processes including the edge detection and ICP registration employed by our proposed SLAM system. In summary, using such a device for on-board computation would allow a smaller and more agile MAV platform to be used, the device's parallel processing capabilities would also open up a new avenue of potential visual processing methods.

6.3.3 Edge Cloud Refinement and Monocular Depth

Our current proposed SLAM systems are heavily reliant upon depth data as provided by an RGB-D sensor. However, as shown in [57] the accuracy of the depth data produced by many current RGB-D sensors (such as the Asus Xtion) is prone to noise, distortion,

discretization proportional to depth itself. As a result many keyframes may store semi-dense edge clouds generated from highly inaccurate low quality depth data, which in turn will result in poor registration and sensor pose estimation. This issue may be addressed by instead storing a semi-dense edge cloud for each key-frame, the estimated depth of each point of which is refined over many frames of RGB-D data (i.e. over multiple observations). Additionally stereo comparisons of pixels between RGB images may also be used in the estimation of pixel depths in a similar manner as employed by LSD SLAM [27], allowing the system to function in situations where the RGB-D sensor is unable to provide depth data.

6.3.4 Redundant keyframe Elimination

One issue with both proposed RGB-D SLAM systems is that of redundant keyframes that may accumulate during the course of mapping. By this we refer to keyframes which are highly similar to others present within the map, both in terms of estimated pose and associated features. These keyframes thus provide no useful contribution to the system's tracking and localisation capabilities, and cause the SLAM map to continuously grow in size even when re-visiting the same areas, increasing complexity and consuming additional memory and computational resources. It would thus be desirable to implement a method to detect and cull such keyframes from the map to address this issue, this would also decrease the complexity of the navigation graph used for navigation.

6.3.5 Tighter Integration Between SLAM and Edge Detection

Typically the majority of edge detection methods are designed to process single static images, with no use of external or a priori information. However for SLAM based applications, edge detection is performed on sequential video frames. We have already demonstrated how the temporal similarity between depth images from RGB-D video can be exploited to significantly accelerate up depth edge detection. A similar exploitation of temporal similarity should also be possible for edge detection regarding the RGB component. Additionally the motion of the RGB-D sensor itself as estimated by either the SLAM system or IMU sensors could be exploited in the edge detection process to further reduce computational cost. Specifically, estimated sensor motion could be used to better determine where edges from the previous RGB-D frame are likely to be located in the current image, and from what sides of the image are new edges likely to come into frame.

6.3.6 Slam Based Exploration

Our proposed keyframe based path planning approach in Chapter 5 was envisioned to allow a vehicle to safely navigate within an environment mapped by the SLAM system presented in 2. One possible avenue for future work is to introduce an exploration scheme based on this planning approach. Since the goal of this exploration would be allow the SLAM system to produce an accurate SLAM map of the environment it would be important to integrate exploration with the SLAM system itself. For example it would be important for the exploration scheme to navigate the environment in such a way to facilitate the detection of loop closures, ensuring that an accurate map of the environment is constructed.

References

[1] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Autonomous navigation and exploration of a quadrotor helicopter in gps-denied indoor environments. In *First Symposium on Indoor Flight*, number 2009. Citeseer, 2009. 7

[2] J. Amanatides, A. Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, page 10, 1987. 139

[3] A. Bachrach, R. He, and N. Roy. Autonomous flight in unknown indoor environments. *International Journal of Micro Air Vehicles*, 1(4):217–228, 2009. 7

[4] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006. 6

[5] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based mav navigation in unknown and unstructured environments. In *Robotics and automation (ICRA), 2010 IEEE international conference on*, pages 21–28. IEEE, 2010. 6

[6] L. Bose and A. Richards. Fast depth edge detection and edge based rgb-d slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1323–1330. IEEE, 2016. 15

[7] L. N. Bose and A. G. Richards. Determining accurate visual slam trajectories using sequential monte carlo optimization. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 4553, 2013. 14

[8] L. N. Bose and A. G. Richards. Mav belief space planning in 3d environments with visual bearing observations. In *International Micro Air Vehicle Conference and Flight Competition (IMAV2013)*, 2013. 14, 101

[9] R. Brockers, S. Susca, D. Zhu, and L. Matthies. Fully self-contained vision-aided navigation and landing of a micro air vehicle independent from external sensor inputs. In *SPIE Defense, Security, and Sensing*, pages 83870Q–83870Q. International Society for Optics and Photonics, 2012. 6

[10] A. Bry and N. Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 723–730. IEEE, 2011. 102

[11] D. Caruso, J. Engel, and D. Cremers. Large-scale direct slam for omnidirectional cameras. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 141–148. IEEE, 2015. 176

[12] C. Choi, A. J. B. Trevor, and H. I. Christensen. RGB-D edge detection and edge-based registration. *IEEE International Conference on Intelligent Robots and Systems*, pages 1568–1575, 2013. ISSN 21530858. doi: 10.1109/IROS.2013.6696558. v, 11, 37, 38, 62, 64, 172, 173

[13] S. Clarkson, J. Wheat, B. Heller, J. Webster, and S. Choppin. Distortion correction of depth data from consumer depth cameras. *Hometrica Consulting (Ed.) D*, 3: 426–437, 2013. 20

[14] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1403–1410. IEEE, 2003. 6

[15] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29 (6):1052–1067, 2007. 6

[16] F. Dellaert, S. M. Seitz, C. E. Thorpe, and S. Thrun. Structure from motion without correspondence. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 557–564. IEEE, 2000. 22

[17] M. Di Cicco, L. Iocchi, and G. Grisetti. Non-parametric calibration for depth sensors. *Robotics and Autonomous Systems*, 74:309–317, 2015. 20

[18] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. 155

[19] L. Ding and A. Goshtasby. On the canny edge detector. *Pattern Recognition*, 34 (3):721–725, 2001. 38

[20] E. Eade and T. Drummond. Scalable monocular slam. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 469–476. IEEE, 2006. 6

[21] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989. 130

[22] A. Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. *arXiv preprint arXiv:1304.1098*, 2013. 130

[23] J. Elseberg, D. Borrmann, and A. Nüchter. One billion points in the cloud–an octree for efficient processing of 3d laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76:76–88, 2013. 132

[24] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1691–1696. IEEE, 2012. v, 8, 62, 64, 172

[25] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, 30(1):177–187, 2014. 8

[26] J. Engel, J. Sturm, and D. Cremers. Accurate figure flying with a quadrocopter using onboard visual and inertial sensing. *IMU*, 320:240, 2012. 6

[27] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014. 8, 177

[28] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 164

[29] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. Particle filters for mobile robot localization. In *Sequential Monte Carlo methods in practice*, pages 401–428. Springer, 2001. 104, 105

[30] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, 2002. 125

[31] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *arXiv preprint arXiv:1404.2334*, 2014. 126

[32] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074. IEEE, 2015. 123

[33] S. Gottschalk. Separating axis theorem. Technical report, Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996. 109

[34] S. Grzonka, G. Grisetti, and W. Burgard. Towards a navigation system for autonomous indoor flying. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2878–2883. IEEE, 2009. 7

[35] S. Grzonka, G. Grisetti, and W. Burgard. A fully autonomous indoor quadrotor. *IEEE Transactions on Robotics*, 28(1):90–100, 2012. 7, 122

[36] D. Gutierrez, A. Rituerto, J. Montiel, and J. J. Guerrero. Adapting a real-time monocular visual slam from conventional to omnidirectional cameras. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 343–350. IEEE, 2011. 176

[37] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968. 122, 124, 155

[38] R. He, S. Prentice, and N. Roy. Planning in information space for a quadrotor helicopter in a gps-denied environment. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1814–1820. IEEE, 2008. 7

[39] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012. 8

[40] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *Experimental robotics*, pages 477–491. Springer, 2014. 8

[41] M. Herman. Fast, three-dimensional, collision-free motion planning. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1056–1063. IEEE, 1986. 106

[42] D. Herrera, J. Kannala, and J. Heikkilä. Joint depth and color camera calibration with distortion correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(10):2058–2064, 2012. 20

[43] J. Hirt, D. Gauggel, J. Hensler, M. Blaich, and O. Bittel. Using quadtrees for real-time pathfinding in indoor environments. In *Research and Education in Robotics-EUROBOT 2010*, pages 72–78. Springer, 2010. 106

[44] A. Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3946–3952. IEEE, 2008. 6

[45] A. Howard, M. J. Matarić, and G. Sukhatme. Relaxation on a mesh: a formalism for generalized localization. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 1055–1060. IEEE, 2001. 58

[46] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *International Symposium on Robotics Research (ISRR)*, volume 2, 2011. 8

[47] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011. 5, 7, 8

[48] N. G. Johnson. Vision-assisted control of a hovering air vehicle in an indoor setting. 2008. 6

[49] S. J. Julier and J. K. Uhlmann. Building a million beacon map. In *Intelligent Systems and Advanced Manufacturing*, pages 10–21. International Society for Optics and Photonics, 2001. 19

[50] S. Kambhampati and L. Davis. Multiresolution path planning for mobile robots. volume 2, issue: 3. *Journal of Robotics and Automation, IEEE*, 1986. 106

[51] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010. 126

[52] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. 123

[53] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the rrt*. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011. 125

[54] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996. 102, 123, 124

[55] C. Kemp. *Visual control of a miniature quad-rotor helicopter*. PhD thesis, Citeseer, 2006. 6

[56] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106. IEEE, 2013. 7

[57] K. Khoshelham and S. O. Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012. 20, 31, 52, 140, 167, 176

[58] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007. iv, 5, 6, 18, 23

[59] G. Klein and D. Murray. Improving the agility of keyframe-based slam. In *European Conference on Computer Vision*, pages 802–815. Springer, 2008. 18

[60] K. Konolige and M. Agrawal. Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077, 2008. 6

[61] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011. 19

[62] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000. 22

[63] Y. Kuwata and J. How. Three dimensional receding horizon control for uavs. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, volume 3, pages 2100–2113, 2004. 106

[64] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How. Motion planning for urban driving using rrt. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1681–1686. IEEE, 2008. 125

[65] J.-C. Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012. 119

[66] S. M. LaValle. Rapidly-exploring random trees a ew tool for path planning. 1998. 123, 124

[67] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. 124, 125

[68] J. J. Leonard and H. J. S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *ROBOTICS RESEARCH-INTERNATIONAL SYMPOSIUM-*, volume 9, pages 169–178. Citeseer, 2000. 19

[69] K. Litomisky. Consumer rgb-d cameras and their applications. *Rapport technique, University of California*, page 20, 2012. 31

[70] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999. 6

[71] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979. 121, 122, 124

[72] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997. 19

[73] C. Mei, G. Sibley, M. Cummins, P. M. Newman, and I. D. Reid. A constant-time efficient stereo slam system. In *BMVC*, pages 1–11, 2009. 6

[74] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid. Rslam: A system for large-scale mapping in constant-time using stereo. *International journal of computer vision*, 94(2):198–214, 2011. 6

[75] M. Meilland and A. I. Comport. On unifying key-frame and voxel-based dense visual slam at large scales. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3677–3683. IEEE, 2013. 7

[76] N. A. Melchior and R. Simmons. Particle rrt for path planning with uncertainty. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1617–1624. IEEE, 2007. 102

[77] H. P. Moravec. Robot spatial perceptionby stereoscopic vision and 3d evidence grids. *Perception*, 1996. 130

[78] R. Mur-Artal, J. Montiel, and J. D. Tardós. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. iv, 5, 6

[79] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011. iv, 5, 7, 22, 23

[80] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira. Large-scale 6-dof slam with stereo-in-hand. *IEEE transactions on robotics*, 24(5):946–957, 2008. 6

[81] S. Prentice and N. Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 2009. 102

[82] V. A. Prisacariu, O. Kähler, D. W. Murray, and I. D. Reid. Simultaneous 3d tracking and reconstruction on a mobile phone. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pages 89–98. IEEE, 2013. 176

[83] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009. 164

[84] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, 1989. 130

[85] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. IEEE, 2011. 6

[86] S. A. Sadat, K. Chutskoff, D. Jungic, J. Wawerla, and R. Vaughan. Feature-rich path planning for robust navigation of mavs with mono-slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3870–3875. IEEE, 2014. 6, 22

[87] D. Scaramuzza and R. Siegwart. Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles. *IEEE transactions on robotics*, 24 (5):1015–1026, 2008. 176

[88] K. Schauwecker and A. Zell. On-board dual-stereo-vision for the navigation of an autonomous mav. *Journal of Intelligent & Robotic Systems*, 74(1-2):1–16, 2014. 6

[89] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for ar on a smartphone. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 145–150. IEEE, 2014. 176

[90] K. Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254. ACM, 1985. 60

[91] C. Silpa-Anan, R. Hartley, et al. Visual localization and loop-back detection with a high resolution omnidirectional camera. In *Workshop on Omnidirectional Vision*. Citeseer, 2005. 176

[92] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000. 124

[93] H. Strasdat, J. Montiel, and A. J. Davison. Real-time monocular slam: Why filter? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664. IEEE, 2010. 6

[94] H. Strasdat, J. Montiel, and A. J. Davison. Scale drift-aware large scale monocular slam. *Robotics: Science and Systems VI*, 2010. 6

[95] H. Strasdat, A. J. Davison, J. Montiel, and K. Konolige. Double window optimisation for constant time visual slam. In *2011 International Conference on Computer Vision*, pages 2352–2359. IEEE, 2011. 6

[96] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012. v, ix, 41, 61, 96

[97] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012. 37, 62, 85

[98] J.-P. Tardif, Y. Pavlidis, and K. Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2531–2538. IEEE, 2008. 176

[99] C. J. Taylor and D. J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1021–1032, 1995. 22

[100] A. Teichman, S. Miller, and S. Thrun. Unsupervised intrinsic calibration of depth sensors via slam. In *Robotics: Science and Systems*, volume 248, 2013. 20

[101] S. Thrun. Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc., 2002. 104, 105

[102] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial intelligence*, 128(1):99–141, 2001. 104, 105

[103] S. Thrun et al. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1:1–35, 2002. 130

[104] P. H. Torr and A. Zisserman. Feature based methods for structure and motion estimation. In *International workshop on vision algorithms*, pages 278–294. Springer, 1999. 22

[105] G. P. Tournier, M. Valenti, J. P. How, and E. Feron. Estimation and control of a quadrotor vehicle using monocular vision and moire patterns. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, pages 21–24, 2006. 6

[106] R. G. Valenti, I. Dryanovski, C. Jaramillo, D. P. Ström, and J. Xiao. Autonomous quadrotor flight using onboard rgb-d visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5233–5238. IEEE, 2014. 8

[107] J. Ventura, C. Arth, G. Reitmayr, and D. Schmalstieg. Global localization from monocular slam on a mobile phone. *IEEE transactions on visualization and computer graphics*, 20(4):531–539, 2014. 176

[108] E. Welzl. Constructing the visibility graph for n-line segments in o (n 2) time. *Information Processing Letters*, 20(4):167–171, 1985. 122, 124

[109] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially extended kinectfusion. 2012. 8

[110] T. Whelan, M. Kaess, J. J. Leonard, and J. McDonald. Deformation-based loop closure for large scale dense rgb-d slam. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 548–555. IEEE, 2013. 7

[111] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, 2015. 7, 8

[112] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. Elasticfusion: Dense slam without a pose graph. *Proc. Robotics: Science and Systems, Rome, Italy*, 2015. 7, 8

[113] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte. Efficient simultaneous localisation and mapping using local submaps. In *Proceedings of the Australian Conference on Robotics and Automation*, pages 128–134, 2001. 19

[114] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2, 2010. 132

[115] Z. Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012. 7