Investigating the Feasibility of Laser Scanners for Autonomous Navigation of Hexacopters in Forests

Daniel Axtens

ENGN4718

Supervisor: Dr Jon Kim

November 2014

Abstract

This report works towards autonomous navigation of Unmanned Aerial Vehicles (UAVs) in dense, GPS denied forests by demonstrating that a laser scanner is feasible for estimating position. A payload for a hexacopter was built, mounting a gimbal-stabilised and vibration-damped laser scanner and a miniature PC. The laser scans are processed using polar scan matching to provide a position estimate. Trees are then detected in the laser scan, added to an approximate map, and used as fixed points to correct the position estimate. The algorithms run in real time, even on low-powered computers. The system is tested by both walking and flying the payload through different sets of trees. The results show that this approach achieves excellent localization in diverse locations without *a priori* knowledge.

Contents

Glossary

CSM Canonical Scan Matching.

IC Integrated Circuit.

ICP Iterated Closest Point.

IMU Inertial Measurement Unit.

MDF Medium Density Fibreboard.

PSM Polar Scan Matching.

ROS Robot Operating System.

SLAM Simultaneous Localization and Mapping.

SVD Singular Value Decomposition.

UAV Unmanned Aerial Vehicle.

Chapter 1

Introduction

Multirotor craft have been the subject of extensive research within the academic community for some time. They are popular for several reasons. They are relatively cheap. They are highly flexible, especially compared to fixed wing craft. The use of multiple rotors enables each rotor to be smaller, and allows all the rotors to be enclosed in a protective ring, providing safety benefits over helicopters. Quadcopters have been particularly popular, but hexacopters are often used when a greater payload is required.

A common problem in the research is the question of how to navigate these craft. At the most basic level, outdoor navigation with good GPS availability and no unknown obstacles is largely a solved problem, with systems already available in commercial and hobbyist products.

Indoor navigation where a map and GPS are unavailable is currently the subject of considerable research. Approaches include vision with a monocular camera [1], or more commonly a stereoscopic or depth-perceiving camera [2]. Laser scanners are also a common approach [3]. The absence of a global, absolute reference and freely available maps has led to a proliferation of Simultaneous Localization and Mapping (SLAM) systems to convert sensor data into a position with respect to a discovered map.

An area of considerable practical interest is outdoor navigation, and in particular, navigation in forests. There are significant real-world applications for this, such as the following motivating problem:

> Two hikers are lost in a forest. The forest has dense trees and inhospitable terrain, such that locating the hikers with a conventional search and rescue

party would take too long. Therefore, we wish to locate the hikers with UAVs.

How do we navigate the UAVs?

GPS is often limited in forests. Furthermore, the trees in forests are not known in advance, and are in close proximity (well below the margin of error for unaided GPS). Outdoor environments also provide significant additional challenges to the approaches adopted for indoor navigation: sensor data tends to be sparser, and objects considerably less regular. This is the area considered by this report.

While vision has been attempted in this area [4], this report investigates the feasibility of using a laser scanner to provide real-time laser odometry for navigating in forests, using the forest trees as fixed reference points to aid in the localization process.

To that end, the following method was adopted:

- A laser scanner was mounted in a gimbal-stabilised vibration-damped hexacopter payload.
- Software to determine the position of the hexacopter from the laser scans was developed. The software has two main stages:
  1. Use Polar Scan Matching to rapidly determine an approximate position.
  2. Build up a map of trees in the forest, and use that map to correct for drift in the polar matching process.
- The payload was walked through several sets of trees to determine its effectiveness under idealised conditions.
- The payload was attached to a hexacopter and flown through sets of trees to determine its effectiveness under actual flight conditions.

An important limitation in scope to note is that autonomous control was not attempted: the hexacopter was flown manually in flight tests.

1.1 Aims

This project set out to achieve the ambitious goal of hovering a hexacopter in a fixed position, outdoors, by using a laser scanner with polar scan matching to provide position information. During the course of the project, it was determined that unaided polar scan matching would not be sufficient, due to the drift inherent in the technique. Therefore, the project explored how the position can be corrected by detecting trees, and using their fixed positions. This author wrote new code to explore this, and concluded that is possible, and indeed feasible in real time. As a result of this new direction, hovering control was not attempted.

The revised goal can be summarised as providing an answer to the following question:

> **Are laser scanners feasible for autonomous navigation of hexacopters in forests?**

1.2 Contribution

The outcomes of this project can be expressed in terms of the **hardware** designed and built, the **software** written, and the **knowledge** gained towards answering the main question. In addition, this project assisted in **skill development**.

1.2.1 Hardware

This project required *designing and building* a gimbal-stabilised, vibration-damped mount for a laser scanner, and mounting it on landing gear for a hexacopter.

The major parts — gimbal, anti-vibration mount, onboard computer, laser scanner, hexacopter and landing gear — were supplied, but they needed to be put together, both mechanically and electrically.

The outcome of this part was a working payload with those parts mounted, that was tested and proven to work. This required both physical hardware development, and electrical work. All the designs are original. The parts were fabricated on a laser cutter by Alex Martin.

During flight tests, the hexacopter was flown by Jon Kim.

1.2.2 Software

The project required significant development within the Robot Operating System (ROS) environment. Chapter 3.3 details which software components were new to this project, but in summary, this author:

- Undertook significant work updating, debugging and adapting the polar scan matcher node. This node was written prior to this project, but required significant work throughout the course of this project. This node was written in C++.
- Wrote completely new nodes in Python to detect trees and determine the position of the hexacopter based on the tree positions.

The software was designed to be reusable in future projects, so the software itself is an outcome.

1.2.3 Knowledge

The software and hardware aimed to provide the tools necessary answer the original question: are laser scanners feasible for autonomous navigation of hexacopters in forests?

The project contributes to answering this question, by testing the hardware and software developed in real unstructured environments. The results provide evidence that laser scanner navigation is indeed feasible. The project also provide suggestions for further implementation work and for future research.

1.2.4 Skill Development

The following new skills were developed by this author during this project.

- SolidWorks, for design of physical components.
- Arduino programming, for the gimbal.
- Familiarity with the ROS environment.

The following existing were skills used/developed:

- Basic electronics and soldering skills
- Basic hardware skills
- C++ programming
- Python programming

1.3 Reproducible Research

Reproducibility is a core requirement of scientific research. Results can only be trusted if they can be replicated by others. In addition to increasing transparency and confidence in the scientific process, reproducibility accelerates scientific progress by making it easier to verify, continue and extend prior work.

Traditionally, reproducibility was accomplished by publishing papers that clearly and concisely explained the work done. However, with the enormous complexity of many projects, it is impossible to include the complete details within a report of reasonable length. Fortunately, the internet makes it easy to provide not just all the details, but all the code and all the data. Full reproducibility — at least of software elements — should be a key goal.

As such, this report, the code written for it, and the data sets analysed, have been made freely available online. They can be accessed at `https://github.com/daxtens/laser-scanner-forests-report`. While their application is more limited, all the relevant CAD files are available from the same address.

1.4 Outline

The report proceeds as follows. Chapter 2 lays out the relevant background. The system for real-time laser odometry is described in Chapter 3. The hardware built and software developed are discussed in Sections 3.2 and 3.3 respectively. The experimental results are in Chapter 4. Chapter 5 concludes this report and makes suggestions for future work.

Chapter 2

Background

Two key concepts for this project are localization and SLAM. *Localization* refers to the process whereby an agent determines its position with respect to a known map. *Simultaneous Localization And Mapping* (SLAM) goes a step further: what if an agent is placed in an unknown environment? To determine its position, it must also create a map of its environment, and simultaneously localize itself within the map.

2.1 Literature Survey

There is a significant body of work on GPS-denied indoor areas.

One approach is to use a single camera, and use computer vision techniques such as object detection and optical flow. This is the approach taken by [1], and requires significant image processing. The processing, while successful, is specialised to indoor environments, and not suitable to the outdoor case.

A popular and more flexible approach in recent years has been to use commercial stereoscopic or depth sensing cameras. For example, [2] uses the Microsoft Kinect to navigate autonomously indoors. The Kinect provides an RGB colour image and a depth map generated by an IR projector and a monochrome camera.

The images from the Kinect are processed on board for basic localisation, and also sent to a separate, more powerful computer, which performs full SLAM, transmits corrections back to the on-board processor, and builds a dense 3D map.

A limitation of the Kinect that is not explored in [2] is the limited range of the depth

sensor on the Kinect. Depth sensing makes use of an IR projector, which limits its range compared to stereoscopic vision. It's unclear if this would severely compromise its performance outdoors, where obstacles are often much further away.

Of particular relevance to this project is [3]. The authors constructed 2 quadcopters: one that used the a laser scanner, and one that used a custom stereoscopic camera. (More detail on the laser based operation is given in the companion paper [5].)

In [3], the performance of both the laser scanner approach and the camera approach are compared using a Vicon vision capture system to provide ground truth. This approach provides for informative data, but is unsuitable for use outside.

The team successfully ran autonomous flight trials and concluded that both approaches were feasible, and that each had their own strengths and weaknesses.

Notably, the paper details the considerable amount of effort required to have the stereoscopic image processing happen in real-time. The quadrotor itself was equipped with a 1.6GHz onboard computer, which used WiFi to offload image processing to a 2.4GHz Core2 Duo laptop. The laser scanner data, on the other hand, was processed onboard on a Gumstix board — highlighting the computational disparity between the two approaches.

The problem of navigation in forests is considered in [4]. Here the authors developed a SLAM algorithm based on an Inertial Measurement Unit (IMU) and a monocular camera. The monocular camera is lightweight, but can only provide bearings to obstacles, not distances, complicating processing.

The test-bed used in [4] was a remote controlled car, rather than a copter, their focus being solely on testing the algorithm. The processing for the algorithm was done entirely from a remote computer; none was done on-board. Furthermore, the obstacles used were artificial: red "blobs" mounted on poles. This vastly simplified the vision processing (especially compared to [1]) but this would need to be addressed before the approach was feasible in real environments. In addition, *a priori* knowledge of the position of closest trees was given to the robot.

The main contribution of this paper is the comparison of a regular Kalman filter with an unscented Kalman filter.

2.1.1 Where does this work fit into the literature?

This project is novel in the following areas:

- Heavy emphasis on outdoor navigation in real environments.
- Algorithm selection and coding for real-time, on-board operation, without the requirement of a base station computer to do heavy processing.

This project is most similar to [3] (with respect to laser scanners), except outdoors instead of indoors. Therefore, the additional challenges of vastly sparser scans in outdoor environments are addressed.

2.2 Hexacopter Considerations

While hexacopters have a number of advantages, they have a number of limitations that must be taken into consideration when designing for them.

Firstly, they have a constrained payload. While the additional two motors give hexacopters a significant advantage over quadcopters, they are still severely weight constrained. This has two effects: constrained sensors, and constrained onboard computation.

The constraint on sensor weight limits the suite of sensors available, restricting both the variety and quality of sensors. For example, a UAV could have a high precision IMU, multiple laser scanners, and multiple Kinects and use sensor fusion to get an even better estimate of position and a high quality 3D map, but that is simply too heavy.

Similarly, constrained onboard computational power requires selection of algorithms which are fast, often at the cost of accuracy. Alternatively, sensor data can be transmitted to a base station for computation. However, this comes at the cost of increased complexity and latency.

Secondly, the hexacopters have unstable flight dynamics. Rapid control is required, meaning that the chosen algorithms must operate in real time and with low latency. This exacerbates the problems caused by constrained onboard computation.

Finally, because the UAV is flying, sensors cannot give direct odometry measurements, and the craft has limited ability to ‘hold position’ and get a more accurate location reading.

2.3 Laser Scan Matching

Laser scan matching is a process whereby a laser scan is matched with a scan from a previous instant in time. The process determines the translation and rotation that most accurately matches the new scan with the old scan. This rotation and translation is an estimate of the movement that has occurred between the two time instants.

There are a wide variety of laser scan matching algorithms, with a range of speed and accuracy trade-offs.

A particularly common algorithm is Iterated Closest Point (ICP) [6]. This algorithm matches each point in the first scan to its closest point in the second scan, determines the rotation and translation that will reduce the mean square distance between the pairs of points, applies that transform, and then iterates the process repeatedly to minimise the error.

A naive implementation of ICP has time complexity $O(n^2)$ in the number of points. If the search angles are constrained, this can be reduced to $O(kn)$, where k is proportion to the angular resolution. Alternatively, with the use of appropriate data structures, the time can be reduced to $O(n \log n)$

Canonical Scan Matching (CSM) [7] is a popular ICP variant that is provided in the `scan_tools` package in ROS. CSM does point to line matching rather than point to point matching, making it faster than ordinary ICP and more robust, especially in indoor case where there are lots of lines.

Originally, these algorithms only matched a scan with the immediate preceding scan. Multiple previous scans were not stored, so fixed points did not remain constant over time. As such, errors accumulate which cause the estimated position to drift with respect to the ground truth over time. To address this, a some implementations use keyframes: rather than matching with the most recent scan, a scan is chosen as a keyframe, and subsequent scans are repeatedly matched with it. New keyframes are chosen at regular intervals or when the change in points is significant. This gives reduced error over time at the cost of greater complexity and storage requirements. This is also mostly suitable to an indoor environment where scans are dense and full of features such as walls and corners.

Chapter 3

Real-time Laser Odometry

3.1 Polar Scan Matching

As a key requirement of this project is to make efficient use of limited computational power, Polar Scan Matching (PSM) was chosen as the scan matching algorithm, over other more well known methods such as ICP.

PSM is defined in [8]. The key idea of PSM is to take advantage of the polar structure of laser scans in order to increase efficiency. The efficiency gain over ICP is quite significant: PSM has time complexity of $O(n)$.

The PSM implementation does not support keyframes, and so can only match a scan with the immediate preceding scan. As such, errors accumulate which cause PSM to drift over time.

Determining the qualitative extent of this drift, and correcting for it, is thus a focus of this report.

3.2 Hardware Construction

3.2.1 Laser Scanner and Gimbal

The hardware was built in two phases: Mk I around a Hokuyo UBG-04LX-F01 scanner,[1] and Mk II around a Hokuyo UTM-30LX scanner.[2]

Mk I

Initially, the assembly was built around the UBG-04LX-F01 laser scanner.

The UBG-04LX-F01:

- Has a 240° scanning range.
- Produces scans of 682 points, each separated by 0.36°.
- Has a maximum range of 4000mm.
- Draws 12V.
- Provides both serial and USB output.
- Produces 40 complete scans per second.

The scanner was fit into a X-CAM X140B gimbal set.[3]

The X140B:

- Is a 2-axis brushless gimbal.
- Is designed to fit Sony NEX5 series cameras.
- Draws 12V.

After removing one of the supports designed for the Sony NEX5 cameras, the scanner fit neatly into the gimbal and was secured on a temporary basis with velcro and foam.

However, as the scanner provides only a USB socket, rather than a USB cable, it had to be positioned so as to provide access to the USB port through a hole in the top panel of the gimbal. This necessitated placing the scanner on one side of the gimbal, and quite

[1] http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/UBG-04LX-F01_spec1.pdf
[2] http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/UTM-30LX_spec_en.pdf
[3] http://www.x-camtech.com/Downloads/X-CAM_X140B_User_Manual.pdf

far forward, as shown in Figure 3.1. To counterbalance this, a U-bolt was placed on the other side of the gimbal and loaded with nuts to roughly balance the platform.

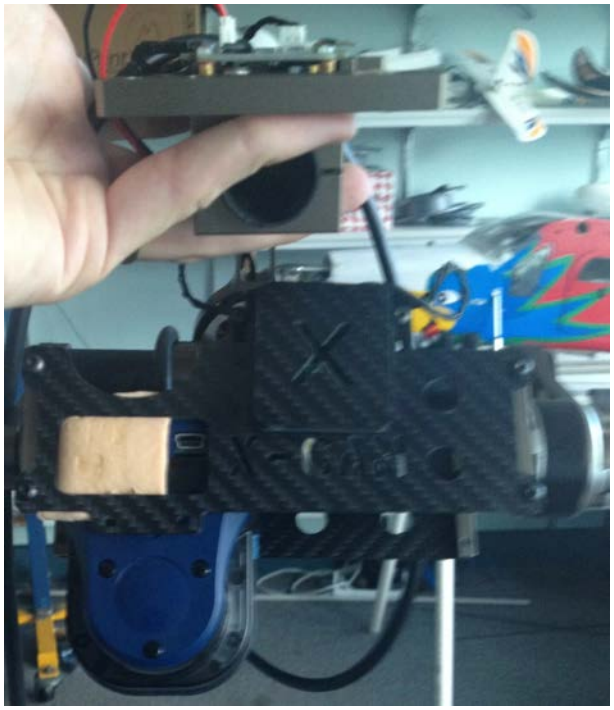

Figure 3.1: Picture of the Mk I, showing the laser scanner mounted with the USB port accessible. This is before the installation of the U-bolt.

While this was sufficient to allow testing of the software and mount, it was excessively heavy and degraded gimbal performance. Designing a replacement platform was identified as a priority for later stages of the project.

Mk I Mod II

During development, power was accidentally connected backwards across the gimbal, blowing an Integrated Circuit (IC) on the control board. Unhelpfully, the gimbal ships with a custom controller board, and all the identifying marks on the ICs were ground off before the gimbal was shipped, making it impossible to replace the damaged IC. Instead, the entire gimbal control system had to be replaced.

The gimbal control board was replaced with a Martinez V3.[4]

The mounting holes in the new board did not fit the existing mounts for the old board, requiring an adaptor board, shown below in Figure 3.2. The large holes mount the new board on nylon spacers, with the small holes attaching to gimbal.

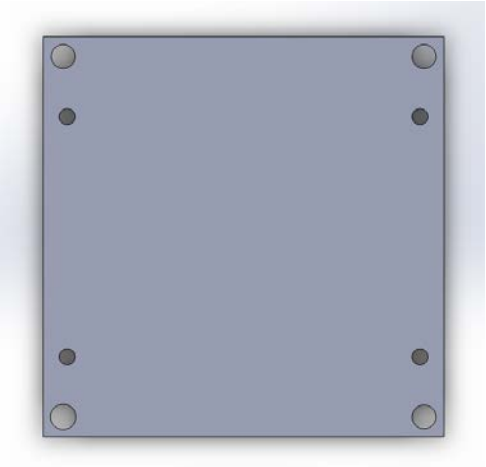

Figure 3.2: The gimbal adaptor board.

This was initially laser cut out of 3mm Medium Density Fibreboard (MDF), however this proved too thick: the original mounting screws were too short to go through the board and in to the mounts below. A 2mm thick board was lighter and more successful.

Replacing the board also necessitated replacing the gimbal's IMU. The new IMU fortunately fit in the space taken up by the old gimbal, and was hot-glued in place.

Unlike the old board, this board did not come pre-programmed. The board ran an Arduino, and the controller software is open-source and available online.[5] The software was installed and tuned. It performed poorly with the heavy Mk I arrangement, but fixing this was postponed due to the upcoming Mk II.

[4] `http://brushlessgimbal.de/`, and see also `http://www.hobbyking.com/hobbyking/store/__41386__2_Axis_Brushless_Camera_Gimbal_Stabilization_Control_Board_w_IMU.html`

[5] `http://sourceforge.net/projects/brushless-gimbal-brugi/`

Mk II

As explained in Section 4.2.2, it was decided to replace the laser scanner to improve range and thus hopefully improve performance outdoors.

The replacement laser scanner was the Hokuyo UTM-30LX scanner.

The UTM-30LX:

- Has a 270° scanning range.
- Produces scans of 1080 points, each separated by 0.25°.
- Has a maximum range of 30m.
- Draws 12V, between 0.7A and 1A.
- Provides both serial and USB output.
- Produces 1 scan every 25ms (40 scans per second).
- Minimum detectable width at 10m: 130mm.

However, this scanner is too tall to fit neatly into the gimbal mount. As such, the top plate and base plate were removed and a new gimbal base plate was designed. The plate is pictured in Figure 3.3. To allow the laser scanner to be positioned in such a way as to balance the gimbal, the board is designed with 3 sets of holes giving 3 possible mount positions.

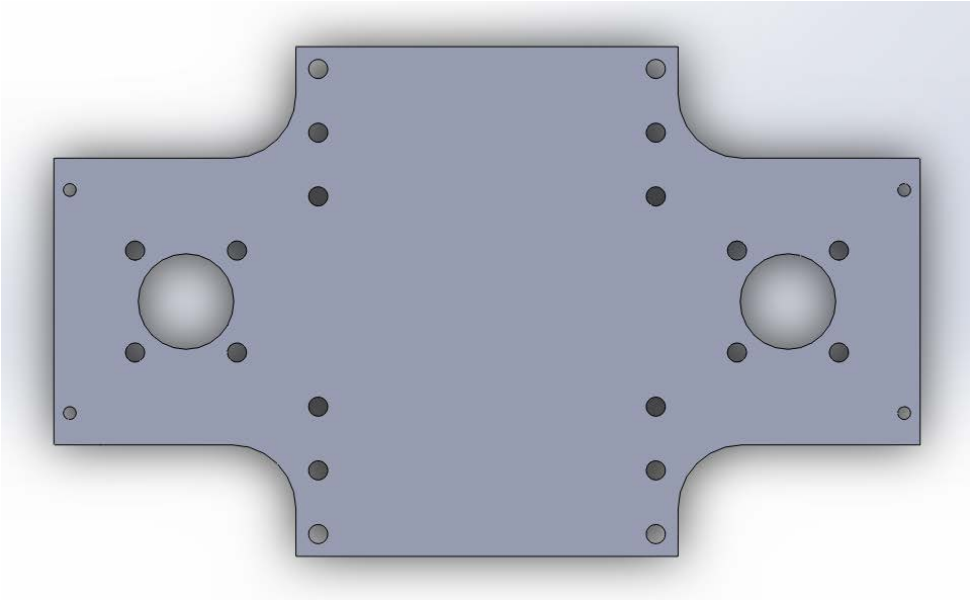

Figure 3.3: The gimbal base plate.

Only one base plate was fabricated, being laser cut out of 3mm MDF. The laser scanner was mounted and the gimbal was re-adjusted. Performance was *considerably* better than with the Mk I.

After fabrication, the following mistakes were discovered, which should be corrected should another base plate be fabricated in future:

- The holes for mounting the rest of the gimbal were designed as M2, where as in fact they are M3. (The holes were manually enlarged for the sake of expediency.) Version 2 of the board should enlarge the holes before laser cutting.
- Enlarging the holes also requires extending the board so that there's enough material on either side of the hole to support the weight. (The present board is cutting it perilously close.)

The following optimisations could also be made:

- The existing board has two cut-outs on either side of the scanner that were put there for the height sensor. These are unnecessary, and should be replaced with a more comprehensive weight-reduction strategy.
- Only the middle set of holes are required. This will allow the board to be simplified and reduced in size.

3.2.2 Onboard Computer

A Pico-ITX form-factor single-board computer was installed to process the scanner data in real time.

The board is a LP-170C Pico-ITX:[6]

- 1.8GHz Atom processor
- 4GB memory
- 4xUSB2 port
- 24GB Compact Flash card for main storage
- Draws 12V

The operating system and software set up on the computer is detailed in Chapter 3.3.

[6] http://www.commell.com.tw/Download/Datasheet/LP-170_Datasheet.pdf

3.2.3 Power Supply

The three powered components (gimbal, scanner and PC) all require 12V power. However, the hexacopter uses 4 cell LiPo batteries, which provide 14.8V. As such, a buck converter is used to down-convert the voltage. The converter is shown in Figure 3.4 below.

Figure 3.4: One of the switching converters

The converters are rated at 25W. At 12V, this is approximately 2A. As such, it was necessary to split the loads between two converters. One converter powers the PC and the scanner, and the other powers the gimbal. A block diagram is shown in Figure 3.5 (the battery, regulator and load all share a common ground).

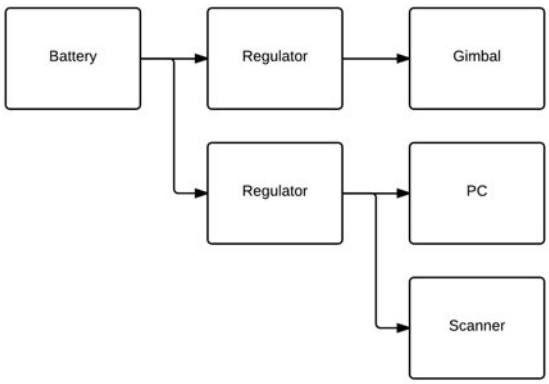

Figure 3.5: Block diagram of the power system

In addition, a locking connector was installed onto the gimbal controller board instead of the original two pin connector. Unlike the previous connector, this connector can only be inserted one way, eliminating the risk of destroying another board.

3.2.4 Height Measurement Setup

It is desirable for the laser scanner to also be able to provide information on the altitude of the hexacopter. To this end, a downwards facing mirror is required, to reflect the last few degrees of the scan down.

The mirror assembly was designed to be adjustable after fabrication. As such, the mirror was held by two circular rings, which could be rotated through a mount.

This was modelled as shown in Figure 3.7. The completed mirror assembly is shown below in Figure 3.6.

Figure 3.6: The completed mirror assembly.

The mirror is a front-faced mirror — silver deposited on silicon — in order to prevent error from refraction caused by having glass in front of the reflective surface.

When the mirror was installed, it was discovered that it was too high, and the entire assembly had to be tilted down and held in place with hot glue. This issue occurs because the assembly mounts on the top of the support. Future assemblies should mount on the bottom side of the top of the support, with screws going upwards into the support, rather than downwards.

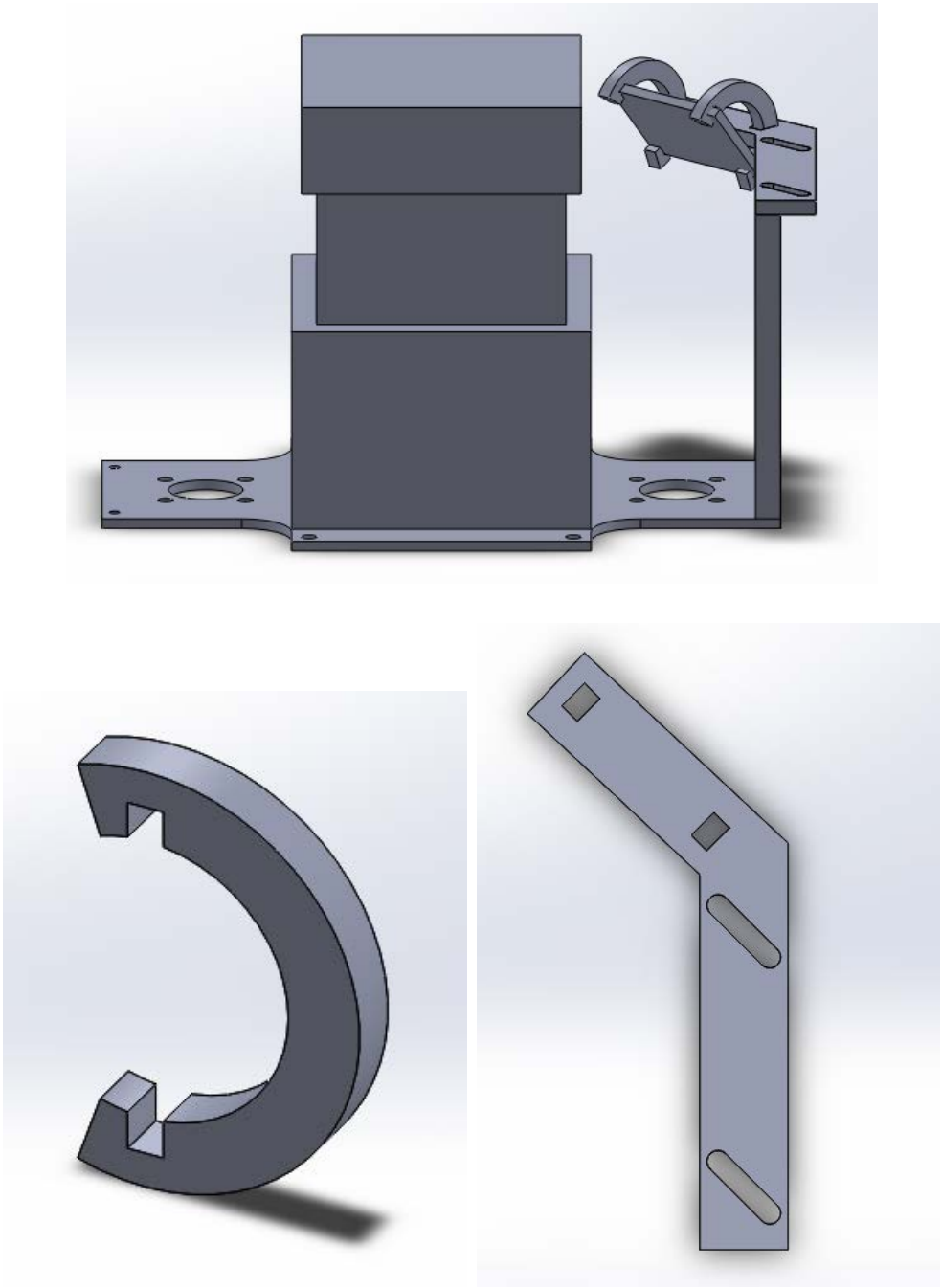

Figure 3.7: The system model (top), the individual rings (bottom left) and the piece holding the rings (bottom right).

3.2.5 Anti-vibration Mount

The gimbal was mounted to an anti-vibration mount with cable ties. The gimbal control board — which would otherwise have protruded too high and hit the mount — was mounted inside the anti-vibration mount with velcro. Various adjustments were made to ensure it did not hit the upper plate of the anti-vibration mount.

The anti-vibration mount was attached to a laser cut board designed to fit the landing gear and the onboard computer. It is attached with cable ties to bear the load and hot glue to prevent the parts moving with respect to each other.

The board is reproduced below in Figure 3.8.

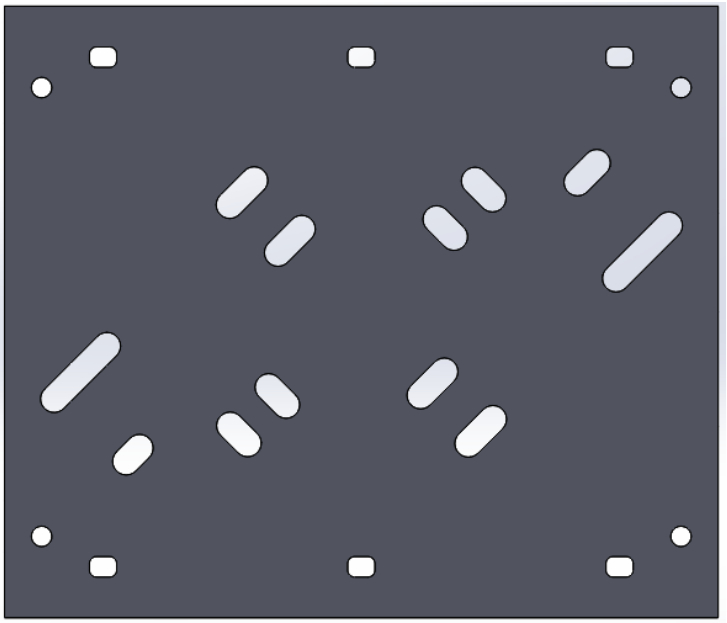

Figure 3.8: The PC mounting board.

It's not immediately obvious how the board mates with the anti-vibration mount, so Figure 3.9 shows the designed plate mating with a reconstruction of the critical parts of the anti-vibration plate.

The board is mounted to the landing gear with cable ties to bear the load and hot glue to damp vibrations.

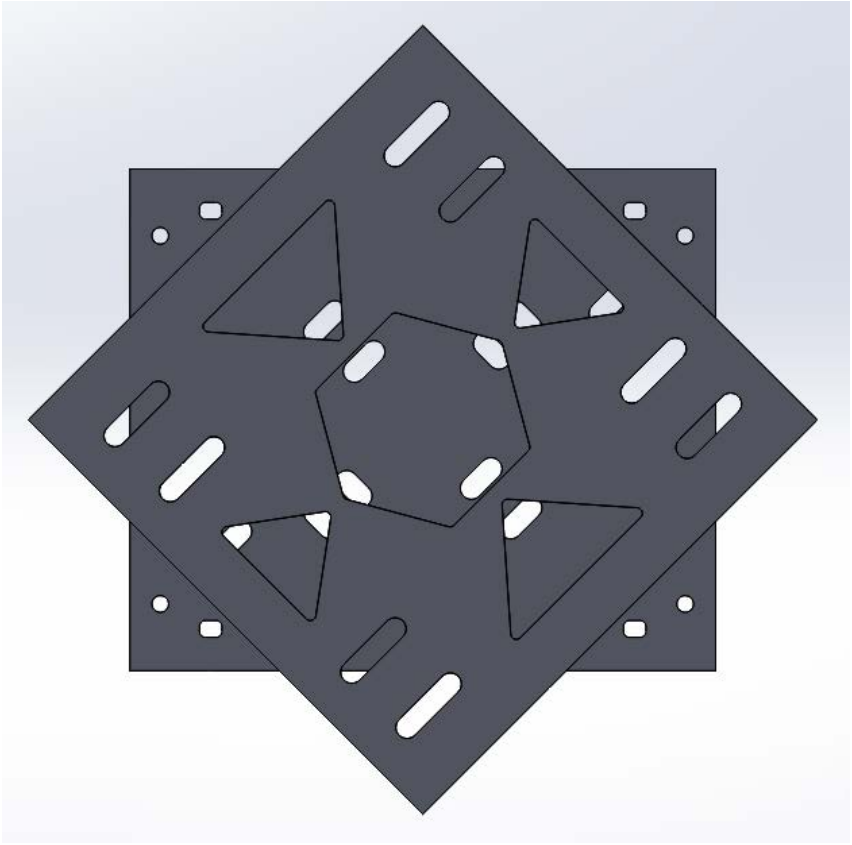

Figure 3.9: The key (interior) parts of the anti-vibration mount (front), with the PC mount plate (back) showing the shared cutouts for cable ties.

3.2.6 The Complete Assembly

Figure 3.10 is a picture of the complete assembly.

Figure 3.10: The completed assembly. The battery sits on top of the assembly.

The total weight of the assembly is around 1.4kg, not including batteries. The battery used weighs 330g.

3.3 Software Construction

The software is built on top of ROS, the Robot Operating System [9], using the Indigo version. ROS is built around the concept of *nodes* — lightweight, reusable programs that perform a small part of robot operation — that are coupled together by passing around *messages* through use of a publish-subscribe mechanism.

The publish-subscribe mechanism is a key to the flexibility of ROS. Various types of *message* can be defined — from numbers to odometry information to laser scans and many others. Messages pass over *topics*, which are named channels that pass a specific type of message. For example, the interface to a laser scanner may define the `scan` topic, over which messages of the `LaserScan` type are then *published* by that node. Other nodes — such as a scan matcher or a visualisation system — can *subscribe* to the `scan` topic in order to receive the messages as they are published. This setup allows new nodes to be added that use the data without having to reconfigure the node that supplies the data.

ROS also provides a *transform* mechanism. Loosely, a transform gives the position of one frame of reference in terms of another frame of reference. For example, a transform could provide the position of a sensor with respect to the robot body — a fixed transform. Another example would be a transform that gives the position of the robot body with respect to some reference frame. For example, GPS provides the position of the robot in an absolute frame. This is a dynamic transform.

Because ROS is a modular system, it is helpful to consider an overview of the setup before the parts are discussed in detail. Figure 3.11 shows a graphical representation of ROS nodes used in the program, as well as the main topics and transforms. Grey nodes were provided by ROS, while white nodes were either written or significantly modified during the project.

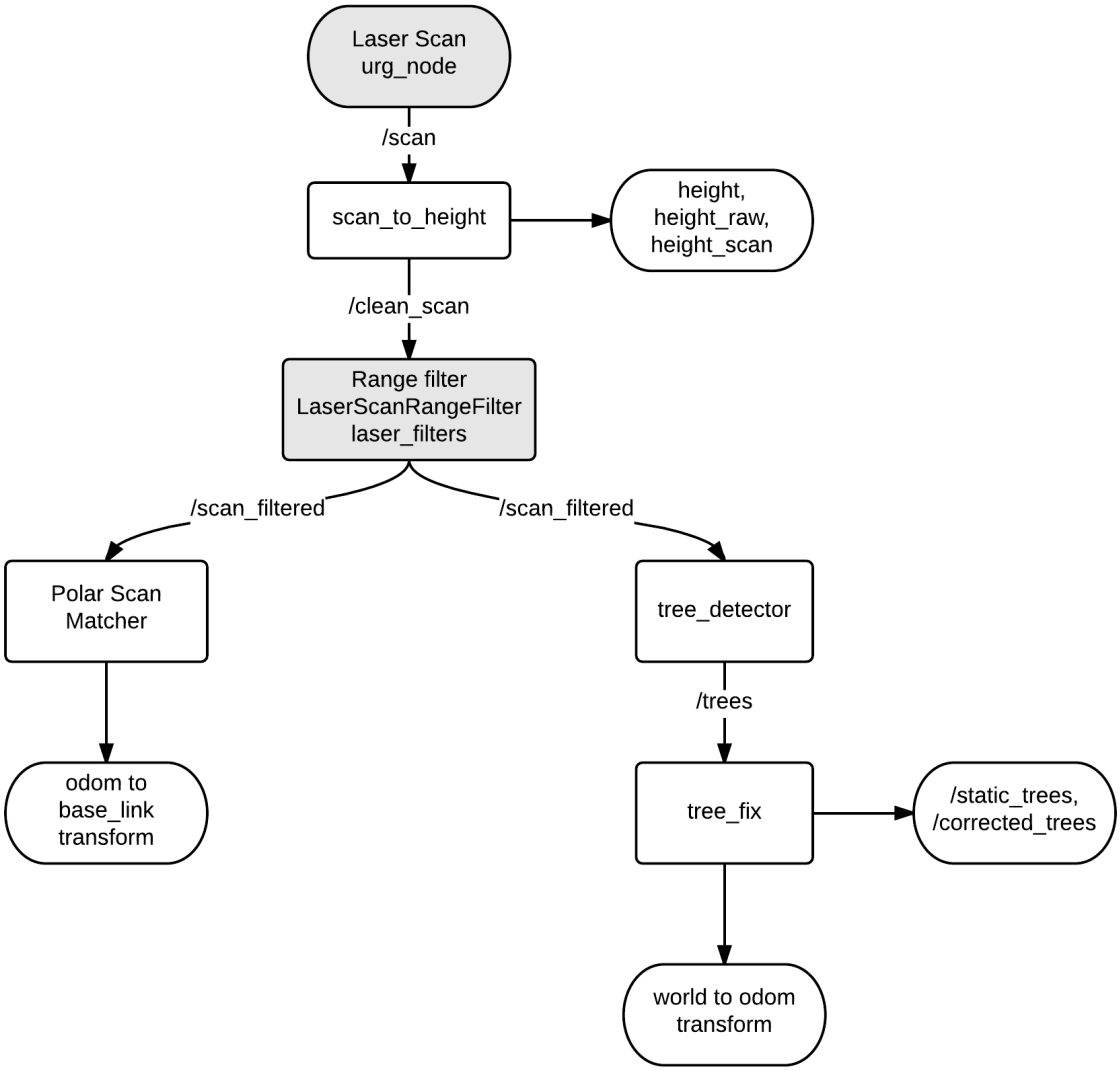

Figure 3.11: Graphical representation of ROS nodes, data passed between them, and key transforms published.

There are three frames of reference in Figure 3.11: `world`, `odom`, and `base_link`. They are (mostly) named in accordance with ROS standard REP105 [10], and are defined as follows:

- `world` provides a fixed world reference frame. It is defined such that the robot starts at the origin.[7]
- `base_link` is a frame of reference that is fixed to the robot. That is, if there's a sensor attached to the robot, the sensor's position remains at a constant position with resepect to this frame.

[7]For compliance with REP105, this frame should be named `map`. However, early development wasn't REP105 compliant and the name hasn't been updated yet.

- `odom` is difficult to define except in terms of its relationship to other frames. The position of the `base_link` frame is determed by the polar scan matcher with respect to the `odom` frame, and the `odom` frame is positioned by `tree_fix` in order to make sure the `base_link` frame is positioned correctly. In other words, the `world` to `odom` transform is a correction only: `odom` doesn't represent any real position.

This is understood most easily through Figure 3.12.

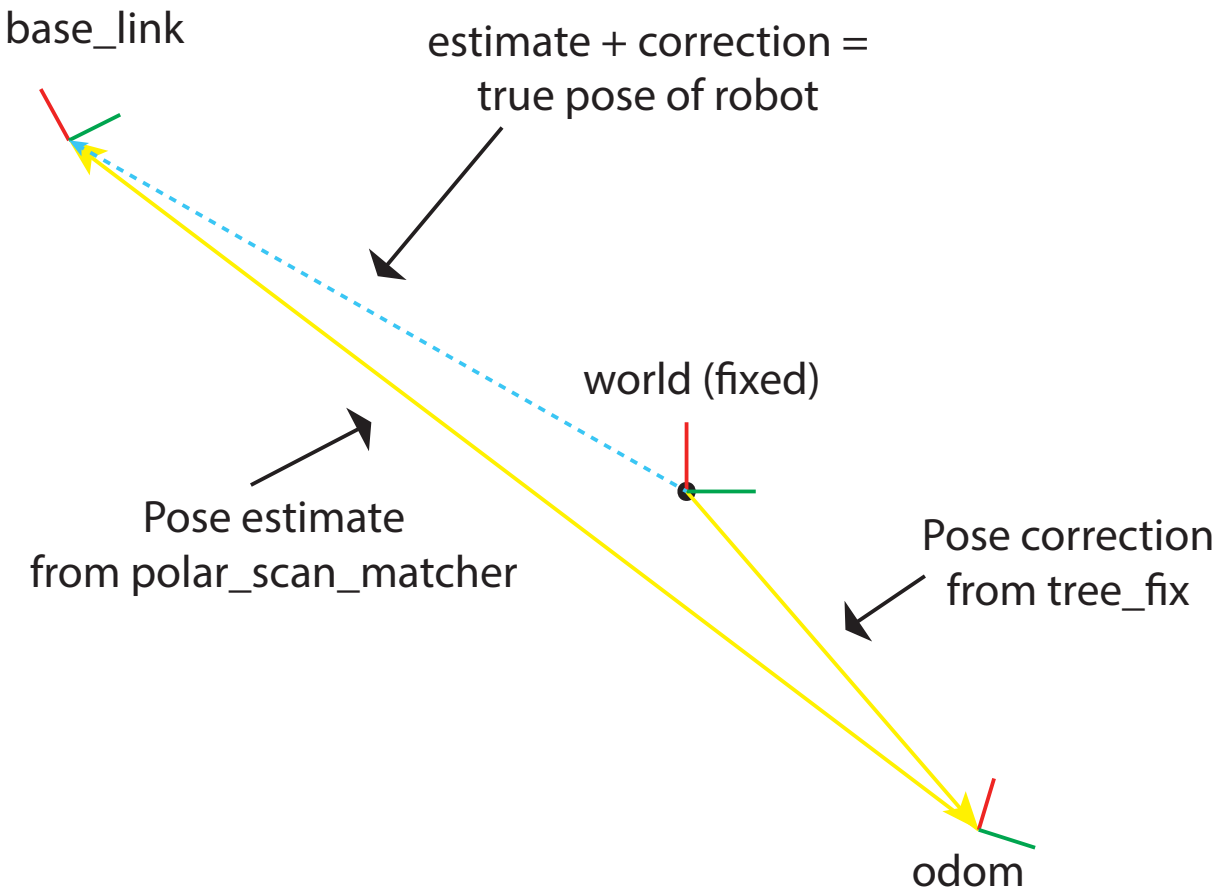

Figure 3.12: A diagramatic explanation of the ROS transforms used.

3.3.1 scan_to_height

As explained in Subsection 3.2.4, the last few points of the laser scan are reflected downwards to get a measurement of height.

This node splits the received laser scan (topic `/scan`) into two scans: `/height_scan`, which contains just the points for the downward reflected height, and `/clean_scan` — the scan with the downward reflected points removed.

The median of the downward points is published to `/height_raw`. However, this value is very noisy. To reduce the noise, the value is then placed in a 10 point moving average, and that value published in `/height`. This has a very low noise — in the order of a couple of centimetres — but because there are 40 scans per second, still very rapid response to changes in altitude.

So far, this value is largely unused. Future work should integrate this into control of the hexacopter.

The node also still needs to be calibrated. There are two major variables or sources of error. Firstly, the mirror is not at a perfect downwards angle, so there is a multiplicative error. Secondly, there's an additive error due to the distance between the sensor and the mirror, and due to losses caused by the mirror surface.

3.3.2 Scan Filtering

Having the laser scanner mounted in the landing gear means that various points of the scan will always hit the landing gear. If these are not filtered out, they cause the polar scan matching to perform very poorly.

A node provided with ROS (`laser_filters`) is used to filter points from the scan that would be within the footprint of the hexacopter. This significantly improves performance.

Various other filters provided with the node were briefly explored, including a shadow remover, and an interpolation filter. Neither improved performance.

3.3.3 Polar Scan Matcher

The polar scan matcher is based on [8], as explained in Section 3.1.

This node posed a number of challenges.

Firstly, while code was purported to exist for ROS, it no longer existed in the primary `scan_tools` repository.[8] A copy was located in another repository and copied over.

Secondly, the code was old — circa ROS Electric, and would not compile on more recent versions of ROS due to changes in various libraries, specifically the transform datatypes.[9] Fortunately, the provided migration scripts proved sufficient to migrate the code and have it compile.

Thirdly, the code once compiled did not run, but would crash almost immediately after launching. The code initially expected the scanner to report a range of 0 for out of range points, but the scanner was instead reporting NaN (not a number) values instead. Fixing the assumption lead to the polar scan matcher running reasonably well indoors.

Despite these changes, outdoor performance remained completely inadequate. The first attempt to fix this was by replacing the scanner with a much more capable scanner (see the Mk II: Subsection 3.2.1). This lead to some improvement but not enough to make the system usable.

The following changes were also required for acceptable outdoor performance:

Points required for a match: By default, the polar scan matcher node requires 100 points be matched from between two scans for the match to be considered successful. This worked well inside but poorly outside. Dropping the threshold to 10 points improved matching noticeably, but not to the point of being acceptable.

Negative Distances: Error reports from the polar scan matcher node showed that occasionally negative distances were found. Tracing these back, it was discovered that the scanner node itself was outputting negative distances. These were discarded in an early stage of processing, leading to slightly improved performance.

Median Filter: A close, time-consuming examination of the code revealed that one of the pre-processing steps was a median filter.

[8] `https://github.com/ccny-ros-pkg/scan_tools`

[9] See: `http://wiki.ros.org/geometry/bullet_migration` and `http://wiki.ros.org/fuerte/Migration`

The median filter replaces each point with the median of the points around it. Formally, it works as follows:

Input: Laser scan s
Output: Laser scan s'

for $i \leftarrow 0$ **to** (*length of* $s - 1$) **do**
    `// Collect 5 points: the point and two points either side.`
    $r \leftarrow s[(i-2)\ldots(i+2)]$;
    $r' \leftarrow \text{bubble\_sort}(r)$;
    `// Replace the point with the central (3rd) point of` r'
    $s'[i] \leftarrow r[2]$;
end

Algorithm 1: Pseudo-code for the median filter algorithm

While this algorithm works well indoors, where there are long straight runs, the algorithm dramatically impedes outdoor performance. Disabling this filter entirely lead to vastly superior outdoor behaviour.

3.3.4 Tree Detection & Localization

In keeping with the ROS philosophy of having small, specialised and reusable nodes, the process is broken up into two nodes: one which takes laser scans and detects trees, and one which uses the detected trees to correct the polar scan matching-derived position to a true position.

Tree Detection: `tree_detector`

The tree detector node simply detects trees from a laser scan. It works from the assumption that tree trunks are basically circular, and so attempts to detect circles in the scan.

It is presently a very primitive process, but works sufficiently well when the following assumptions are met:

- Trees are close. More precisely, a *map range* is defined, and points that compose the trees must be within the square (`-MAP_RANGE`, `-MAP_RANGE`) to (`MAP_RANGE`, `MAP_RANGE`). The choice of `MAP_RANGE` is a trade-off between detection range and speed and memory usage. However, increasing range has diminish returns, as the

distance between detected points grows with distance, making it harder to accurately detect trees. Experimentally, 6 metres provides good results.

- There are no significant clusters of points other than trees within the map range. For example, there must be no walls and no people.

The process proceeds as follows:

Drop scans: Only every Nth scan is processed, where N is a parameter currently set to 10. This is necessary to ensure that there's not a backlog of messages out of the node. Drop the other $N - 1$ scans without processing them.

Convert the laser scan into an image: For speed and memory usage reasons, only points within the map range are included. Presently, the intensity of the reading from the laser scan is ignored, and readings are mapped either to pure white if they fall within the map square, and black if they don't fall within the map.

Blur: A gaussian blur is applied to the image. This serves two purposes:

- It reduces the effect of noise on future stages — a stray reading will be blurred into basically nothing.
- A scan of a tree at a few metres distance will not be a contiguous line, due to the angle between the readings. This leads to it not being detected as one feature, but as several, which is undesirable. Blurring tends to create smoother, contiguous runs that are easier to detect.

Threshold: A threshold is applied to further reduce noise.

Circle detection: Actually detecting circles was more difficult than expected. Four major approaches were attempted:

- The OpenCV built-in "Hough Circles" detection process. This failed to yield any results at all, even on artificial perfect circles and with supplied sample code. This may have been due to OpenCV version problems, or a misunderstanding of the parameters.
- Feature detection with a semi-circle also failed; which may be assumed to be due to the same root cause as the Hough Circles failure.
- MSER—maximal stable external regions—a way to detect 'blobs', was mostly successful, but far too slow.

- The approach ultimately adopted was to find features with more points than a given threshold, and find the minimum enclosing circle. While not strictly accurate, it provides sufficient accuracy, especially when there are a sufficiently large number of points, for the algorithm to work well.

Publish points The centers of the detected trees are published as a point cloud to the `/trees` topic. They're published in the same frame of reference that they're detected in — `/laser`, which in the current implementation is tied to `/base_link`.

Significant future work is needed to make this node more robust and versatile, especially in an outdoor environment that contains both trees and other objects. The biggest improvement needed is to replace the circle detection algorithm with a proper one. Either the Hough Circles function in OpenCV could be debugged and used, or perhaps RANSAC circle detection (as described in [11]) could be used in order to reduce computational requirements. Either way, a proper circle detection algorithm should prevent features like walls from being inadvertently detected as trees.

Furthermore, there are a number of improvements that can be made to the speed and robustness of the algorithm.

- The map uses 1 pixel per centimeter. This provided good early experimental results, but is quite costly in terms of both memory and speed. This could probably be reduced, especially if a different algorithm was used.
- Presently, the points are placed into a map, which, for memory and speed reasons, is fairly small. However, the map is very sparse, because in an outdoor environment most of the points are out of range. It should be fairly straightforward to create several smaller maps wherever points are detected, decreasing memory usage, increasing speed, and increasing detection range.
- A gaussian filter is used with quite a large kernel (11×11 pixels). Alternative, faster filters should be evaluated.
- The entire Python code should be re-written in C++, which would probably result in a *significant* speed up. Alternatively, Cython could be considered as a way to significantly increase the code speed.

Finally, it would be highly desirable for many of the hardcoded parameters to be made configurable in the standard ROS way.

Tree Localization: `tree_fix`

The `tree_fix` node uses the tree positions published by `tree_detector` to perform primitive SLAM. Specifically, `tree_fix`:

- Builds up an approximate map of the trees in the environment.
- Uses that map and the detected trees to localise the robot, providing a correction to the pose estimate provided by the polar scan matcher.

In very broad brush strokes, the process is as follows:

- Assume that the change between this scan and the last scan is probably not too great, seeing as the `tree_fix` package publishes a new point cloud on `/trees` several times a second. Therefore, the last correction can be used to gain a good first estimate of what the current correction should be.
- Find the static points closest to the new points, resulting in two sets of points of equal length.
- Then it becomes the case that the new points (P') are the rotation (R) of the original points (P) plus a translation (T) plus some noise/error (N):

$$P' = RP + T + N \tag{3.1}$$

  Therefore, the R and T that minimises the error between the known points and P' can be determined.
- R and T are used to transform the complete set of new points.
- Points that cannot be matched to existing map points are determined to be new trees and added to the map.

Implementation details make the algorithm more complex in practise. The implemented algorithm is shown in Figure 3.13 and fleshed out in more detail below.

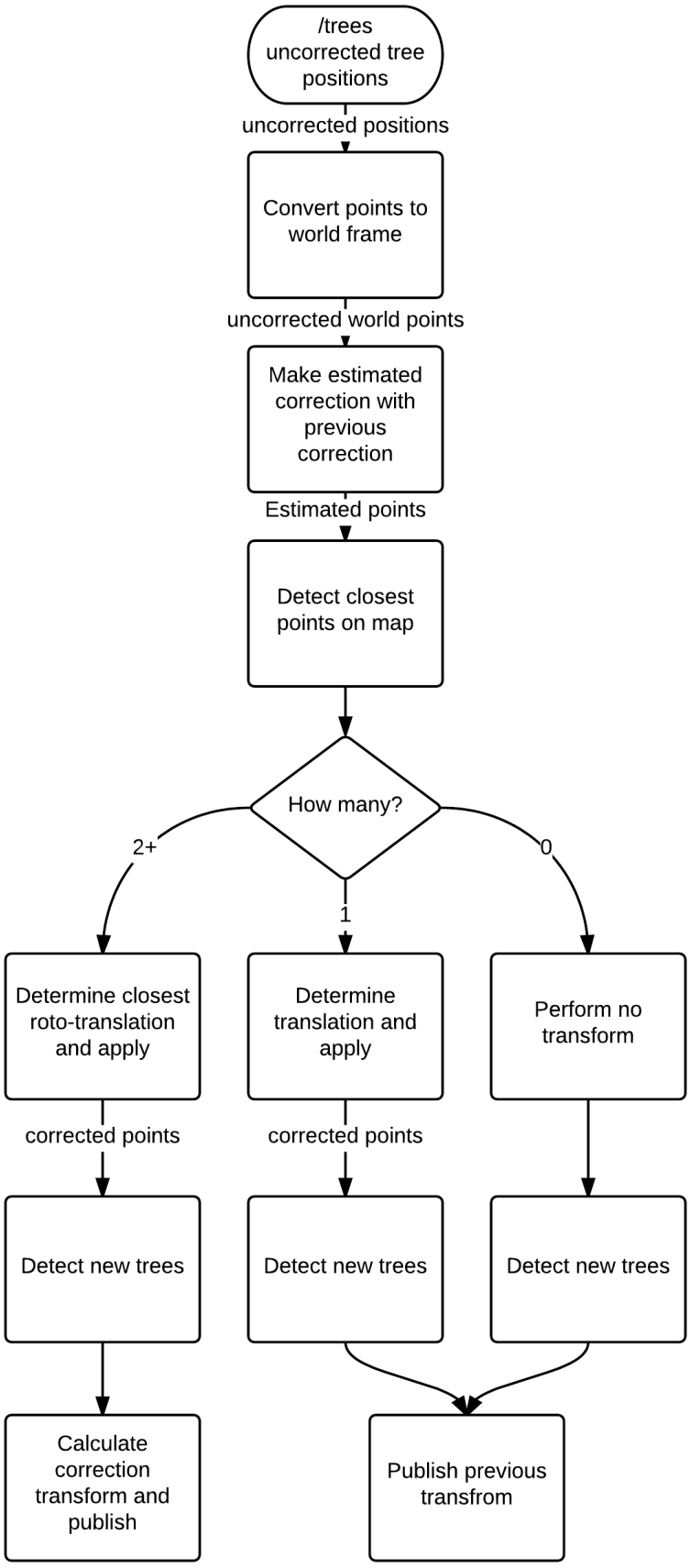

Figure 3.13: Flowchart describing the operation of `tree_fix`

The algorithm as implemented is:

- Keep track of the last translation and angle correction. Initially this starts at a translation of (0,0) and an angle of 0 radians.
- Use the last known translation and rotation to transform the detected points into 'uncorrected points'.
- The uncorrected points are then compared to the known points. Points are matched to their nearest known point, so long as the nearest known point is no further than a given threshold away.
- If there are two or more points in common, a roto-translation (a rotation plus a translation) is then calculated such that the RMS error is minimised. This is done as follows.

Input: Sets of points A, B
Output: Rotation matrix R, translation matrix T

`// Center A and B.`
$A_{centroid}$ =mean(A);
$A' \leftarrow A - A_{centroid}$;
$B_{centroid}$ =mean(B);
$B' \leftarrow B - B_{centroid}$;

`// Try a set of rotations.`
`// To be precise, try` N `points in` $[-W, W]$.
$err_{min} = \infty$;
$points \leftarrow$linspace$(-W, W, N)$;

foreach $\theta \in points$ **do**
    R = rotation matrix for θ ;
    $B_{rotated} = B' \times R$;
    err =rms_error$(B_{rotated}, A')$;
    if $err < err_{min}$ **then**
        $err_{min} = err$;
        $\theta_{best} = \theta$;
    end
end

`// Find the translation that aligns the rotated center of` B `with the center of` A.
R = rotation matrix for θ_{best} ;
$T = -R \times B_{centroid} + A_{centroid}$;

return *(R, T)*

Algorithm 2: The brute force algorithm for determining roto-translations.

Previously, this was implemented using the Singular Value Decomposition (SVD) method. However, this was found to be less reliable, because the SVD can include a *reflection* as well as a rotation and a translation. This is physically impossible, but due to the dense and largely similar nature of the forest, it was returned quite often by the SVD based algorithm. The brute-force roto-translation finder described, while vastly less efficient, is much more reliable.

- If the common set only contains one point, then a simple translation is calculated, and it is assumed there is no rotation. (It is impossible to determine whether there is a rotation in addition to a translation, and any rotation is likely to be small relative to any translation.)
- If there are no points in common, the transform is assumed to be the identity: no translation or rotation, simply because it is impossible to determine any transform.
- The translation is applied to all the uncorrected points, such that they become corrected points.
- The corrected points are then tested against the known map points. If there is no map point within a given threshold distance of a corrected point, the corrected point is added to the map.
- The map and the corrected points are published for visualisation purposes.
- If two or more common points were detected, the correction converted from being in terms of the previous correction into being in terms of the world frame. Because the correction is prone to considerable jumps, it is averaged with the previous correction before being published.
- If less than two points were detected, the previous correction is republished. It is important for smooth operation that a correction is published, even if it's the same — otherwise transforms can start to fail, leaving the system without an estimate of position.

Significant future work is needed on this node. Firstly, it can be prone to large jumps: it would be good to integrate it with some form of filter that provides more smooth positioning. Perhaps this could be done by integrating the position information from the polar scan matcher and the tree matcher into a single processing node, utilising something like a Kalman filter.

Secondly, it has no ability to update its idea of known trees. It cannot perform meaningful loop closure or indeed any form of update other than recognising a new tree. This should be addressed.

Lastly, this work replicates work done by others on scan matching and SLAM. Replacing the point matching algorithm described with a point matching algorithm described in the literature should be considered. As many scan matchers are designed for laser scans or other dense point clouds, it may be necessary to make some changes for the vastly more sparse point cloud from the tree detector. Furthermore, the literature relating to SLAM should be consulted to allow more meaningful map updates.

Chapter 4

Experimental Results

4.1 Experimental Setup

The onboard PC used `rosbag` to record scan data and save it to the onboard storage. In early stages of the project, the polar scan matcher and visualisation were also conducted on the onboard computer, but the lack of OpenGL support make the visualisation program `rviz` very slow, which had knock-on effects on the rest of the processing.

As such, for much of the project, data was recorded on the onboard PC, then transferred to a low-powered laptop for processing. The processing was required to be real-time, using `rosbag`'s ability to output timestamped messages. Furthermore, any advantage that the laptop had from being dual-core was obviated by the significant processing requirements of `rviz`. As such, the processing that was run on the laptop should also run in real time with no or minimal tuning on the onboard computer.

4.1.1 Environments

Three main environments were tested, all on them on the ANU campus. The "tennis court" location is a clump of trees near some tennis courts, and is pictured in Figure 4.1. The "roadside" location is another clump of trees on the ANU campus, pictured in Figure 4.2. The clump of trees that forms the Research School of Information Sciences and Engineering (RSISE) location is shown in Figure 4.3.

Figure 4.1: Various pictures of the "tennis court" clump of trees.
Top: View from the start of the loop (the dented metal rail).
Middle: View from mid-way though the loop.
Bottom: View from towards the end of the loop.

Figure 4.2: Roadside trees.
Top: View from near the start of the loop.
Middle: View from towards the end of the loop, looking forwards.
Bottom: View from towards the end of the loop, looking backwards towards the start.

Figure 4.3: RSISE Trees
Top: View showing the front/side showing the big tree, 2 front trees and the 2 back trees.
Bottom: View from the side, looking in the opposite direction.

4.2 Preliminary Tests

4.2.1 Polar Scan Matching Alone: Indoor

Firstly, consider just the polar scan matcher by itself, indoors.

The polar scan matcher provides generally adequate performance inside a room, but tends to be subject to drift and jumps.

This is illustrated in Figure 4.4, which was taken in the Research School of Information Sciences and Engineering (RSISE) building at the Australian National University (ANU). The laser scanner started in the room in the top right corner of the scan, then exited the room, walked straight to the corner, then turned. The angles should be right angles, rather than the non-right angles observed.

It's worth noting that the laser scans tend to go in 'blocks': matching will be very good for a short period (for example in a room) then will have a 'jump' where error rises significantly.

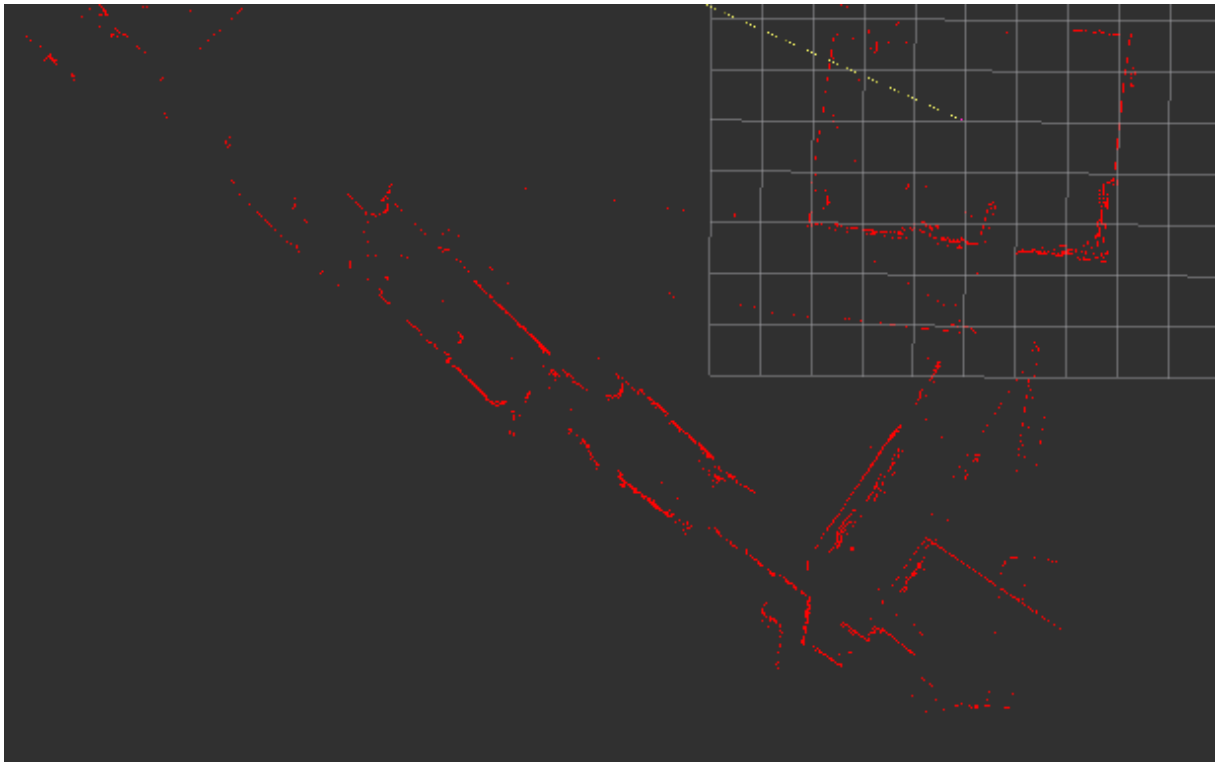

Figure 4.4: Performance of polar scan matcher alone, indoors.

A few observations can be made about the jumps:

- Part of the reason was traced back to sensor/recording error. `rosbag`, which was used to record the data, saves the data out in chunks of a certain size. By default, the chunk size is 768 KB. Unfortunately, writing that much data out to the compact flash card used in the onboard computer caused noticeable delay. This was reduced later in development to 256 KB, which reduced the delay.
- Performance in corridors was particularly bad. This was at least partially due to the self-similarity of corridors: without either being able to see the end of a corridor or other distinguishing landmarks, and with no IMU or other form of odometry, it becomes difficult to distinguish movement from staying stationary.
- Turning corners and leaving rooms seemed to be particularly large sources of error. This could be due to the particularly large number of points changing quickly, with very few points remaining across the turn.

It was also interesting to note that if one of the SLAM systems that is available with ROS was added to the system (for example `slam_gmapping`), performance improved significantly indoors. However, the chunking issue identified above still occurred.

As indoor navigation was not a priority, merely an early way to test the system, no major work was invested in ironing out defects.

4.2.2 Polar Scan Matching Alone: Outdoor

The polar scan matcher was then tested outdoors.

Initially, the system was entirely unusable, not detecting motion in any usable way whatsoever. Firstly, a greater range laser scanner was tried. (This necessitated significant hardware work, resulting in the Mk II: see Section 3.2.1.) However, ultimately various fixes to the software were required. These are explained in Section 3.3.3.

Once those fixes were deployed, the polar scan matcher provided good estimates in the short term, but with unacceptable levels of long-term drift. The drift was especially visible on tree trunks, which became "blurred", as shown below in Figure 4.5.

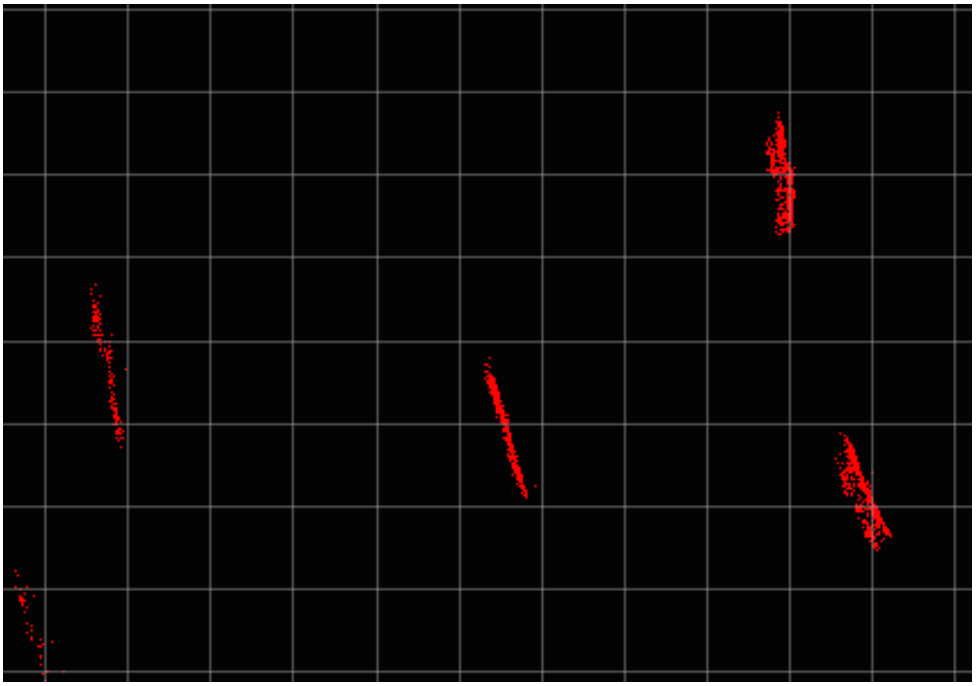

Figure 4.5: "Blurring" experienced by uncorrected outdoor polar scan matching. The picture shows the cumulative laser scan measurements (red dots) over a few seconds. Were there no drift, the five trees would appear as five semicircles.

The cumulative drift and error is significant. Figure 4.6 shows the result of walking the scanner assembly around a clump of trees at ANU. (See Section 4.3.1 for full details of the location.) The path taken was a loop, so the net translation should be zero. However, a large translation is observed; trees do not stay in the same place and there is significant noise.

Evidently further work was required for accurate localisation.

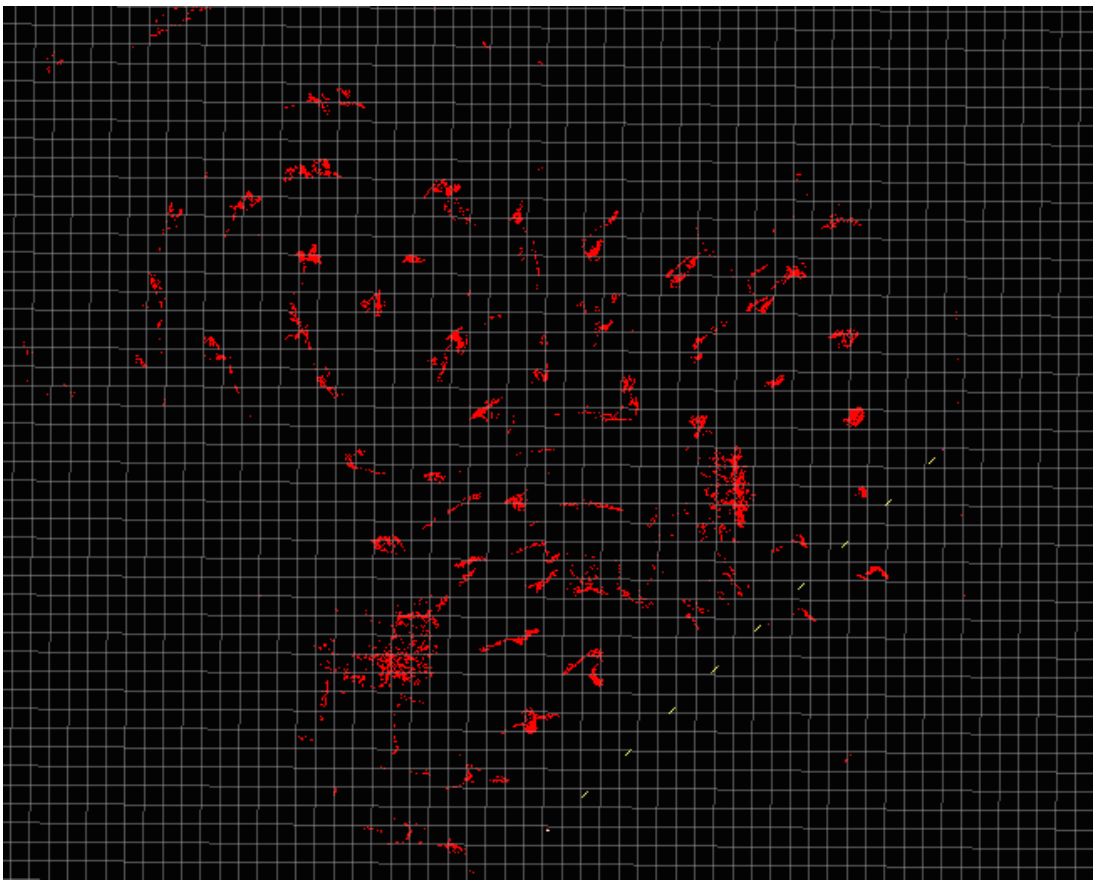

Figure 4.6: PSM only, outdoors. The yellow line shows the final translation, which should be zero. The red dots show the accumulated laser scans.

4.3 Walking Results

Two major outdoor locations were tested.

4.3.1 Tennis Court

The hexacopter landing gear assembly was walked in a loop through the tennis court location trees. The results—the path taken and the map of trees detected—is shown in Figure 4.7. The data bag for this and other trials is available on the project repository.

In the results:

- The *red line* represents the path travelled. The line is a collection of arrows, where the base of each arrow represents the position at that instant, and the arrow is facing in the direction the landing gear was facing.
- The *green dots* represent trees *currently being detected* by the tree detection algorithm. In this case, two trees in view of the start/end position are detected.
- The *blue dots* represent the position of every tree known to the system, the 'map'.

The results have been annotated to highlight several features. Firstly, the arrow shows the initial direction of movement along the loop. The labelled regions are as follows:

A: This region shows the landing gear being rotated around a point. This was done to ensure that the scanner remained in view of multiple trees for as much of the route as possible.

B: This region shows the landing gear being walked sideways: unlike early in the path, the arrows are perpendicular to the direction of travel. Again this was done to ensure that the laser scanner had a good view of trees.

C: This shows a simple 'jump' in position, as the system goes from being able to see one tree, and therefore not being able to do a position correction, to seeing two trees, and being able to do a position correction. The averaging algorithm reduces the level of discontinuity in the position, but the discontinuity is still observable.

D: This shows a more complicated 'jump', as the system became momentarily confused about its position, and then recovered.

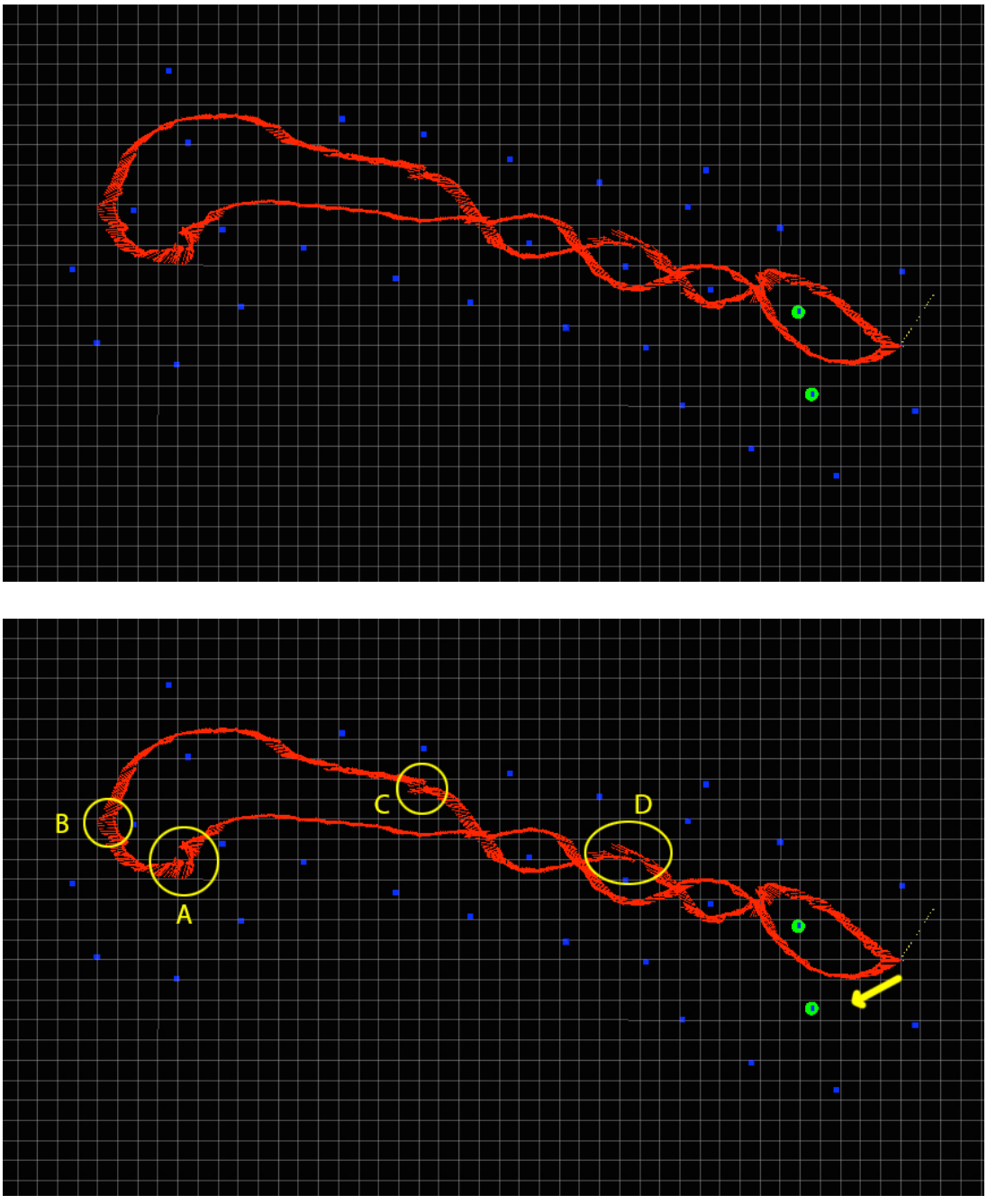

Figure 4.7: Tennis Results. **Top:** raw results. **Bottom:** annotated results.

Because the path was a loop, it is possible to consider how accurately the path closes. Over a distance of around 40m each way, the final position was virtually indistinguishable from the original position, with error of around 10cm.

As such, it can be concluded that the algorithm was highly successful for this set of trees.

4.3.2 Roadside

The hexacopter assembly was walked in a loop in the "roadside" location, with the results plotted in Figure 4.8. The symbols are the same as in the previous location.

Points to note include:

A: The landing gear was again walked perpendicular to the direction it was facing at the end of the loop in order to get a clear reading. (As with B in the previous section.)

B: There were a number of significant jumps. This jump has been highlighted because it occurs after a reasonably large chunk of having no trees detected at all (note the large 'gap' in trees directly above it on the map), so it provides a good example of the amount of drift experienced over moderate distances when using PSM alone.

C: Unlike the previous example, this example does not end with two trees visible, but only one. As shown here, the detected tree starts to drift off from the known tree, because corrections can only be correctly estimated with two trees. This demonstrates that PSM drifts away from the ground truth even in the absence of motion.

The return to origin behaviour is excellent when in view of two trees, but does, as this example shows, tend to drift away when only one tree is in view. However, once that is accounted for, the accuracy is very good — this time over a distance of about 60 meters each way, or over 120 meters in total.

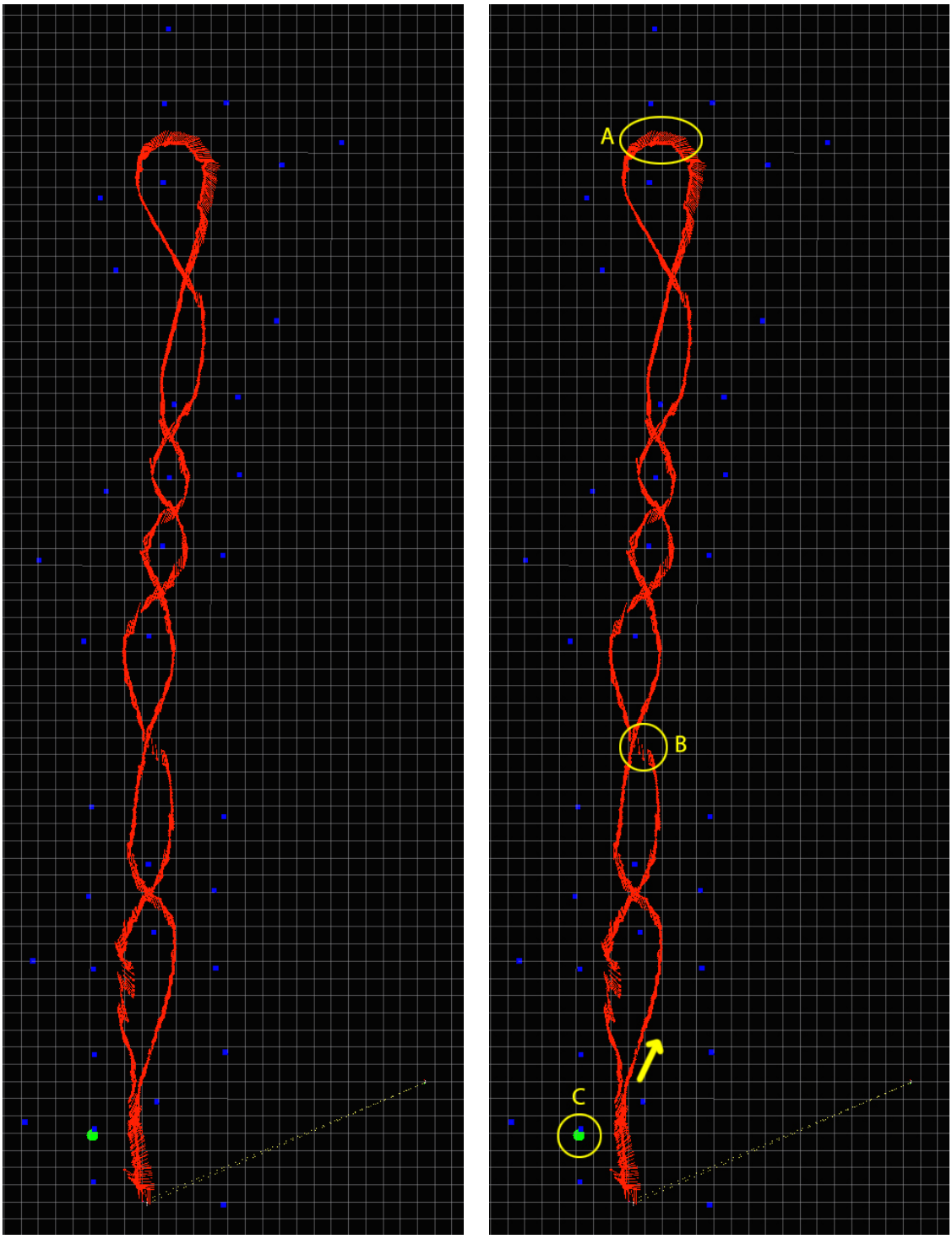

Figure 4.8: Roadside results. **Left:** unannotated results. **Right:** annotated results

4.3.3 Negative Results

Along with the two successful results above, there were a number of unsuccessful trials. While all the trials were different, a number of observations can be made, which further characterise the system.

There are two major reasons for unsuccessful trials: sensor and recording errors, and algorithm errors. Recording errors will be detailed here, and algorithm errors discussed in Section 4.5.

There were two major sources of recording error. Firstly, the chunking error discussed in Section 4.2.1. Secondly, there were some issues with the recorded data missing the start and/or end of the route. This was mitigated by using a smaller chunk size, and ultimately eliminated by waiting 10–15 seconds from when the system started.

4.3.4 Conclusion — Walking

The results from walking the landing gear around the two locations are highly positive.

In short, they demonstrate that:

- the hardware works and is capable of handling the low-frequency vibrations and general 'wobble' caused by being walked around.
- the software works, both in terms of being able to record and replay data, and also in the sense of being able to successfully localise the platform as it is being walked around.
- the software works in both locations, so it is not overly specialised for a single location or set of assumptions.

However, just walking around leaves a few questions: can the hardware handle flying, especially in terms of high frequency vibrations; and can the software handle the extra 'noise' in the data caused by flying?

4.4 Flying Results

The landing gear was attached to one of ANU's hexacopters. The hexacopter power was kept separate from the payload power, in order to prevent any drops in voltage from sudden acceleration from causing issues with the computer.

The hexacopter was able to take off, but required around 70% throttle, suggesting the overall weight should be reduced.

Two flight tests were undertaken, which are explored below.

4.4.1 RSISE

The first flight test was conducted outside the Research School of Information Sciences and Engineering at ANU. The hexacopter in flight is shown in Figure 4.9.

Figure 4.9: The hexacopter flying near the RSISE trees. The photo was taken facing away from the trees, so the trees identified in the results are not in frame.

The flight path did not attempt to weave between the trees, rather it simply flew from side to side, facing the clump of trees, at a distance of a few meters.

The flight was highly successful. Once the hexacopter had taken off, it was able to correctly locate the trees and use them for localization, even with changes in altitude. It also shows the altitude measurement is both smooth and responsive.

Because the flight was not a loop, plotting the entire path is not particularly useful or informative. Instead, Figure 4.10 shows a snapshot of the 3D path taken by the hexacopter, as it ascends.

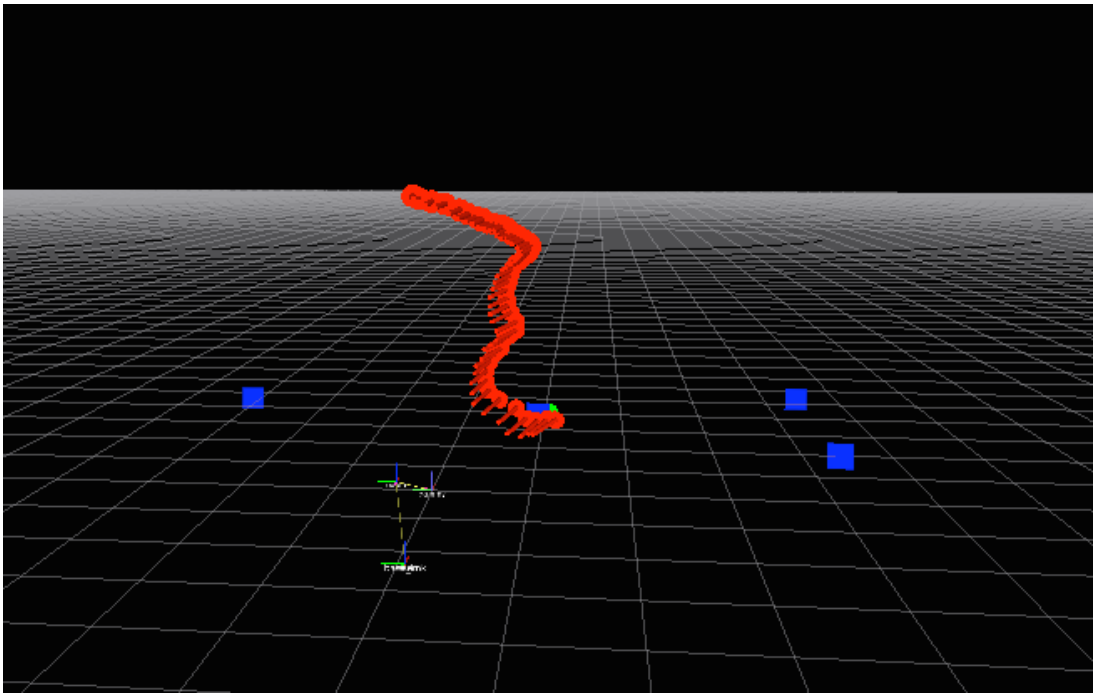

Figure 4.10: RSISE flight results, showing a snapshot of the flight.

The software kept accurate localization so long as the hexacopter was at a reasonable height: more than 1m or so above the ground.

4.4.2 Tennis Court

The second flight test was conducted in the same location as in Section 4.3.1.

Determining the flight path was slightly more complex in this case. The data was noisy, particularly around takeoff and landing. In particular, the landing involved a turn conducted when the polar scan matcher was pointed across a car-park, and was too high to observe any points. Thus the turn cannot be computed, and the system gets horribly confused.

As such, the flight path with the take-off and landing omitted is shown below in Figure 4.11. The symbols are the same as for the walking tests.

Some stills from the 3D view are shown in Figure 4.12. Figure 4.13 shows a picture of the hexacopter in flight.

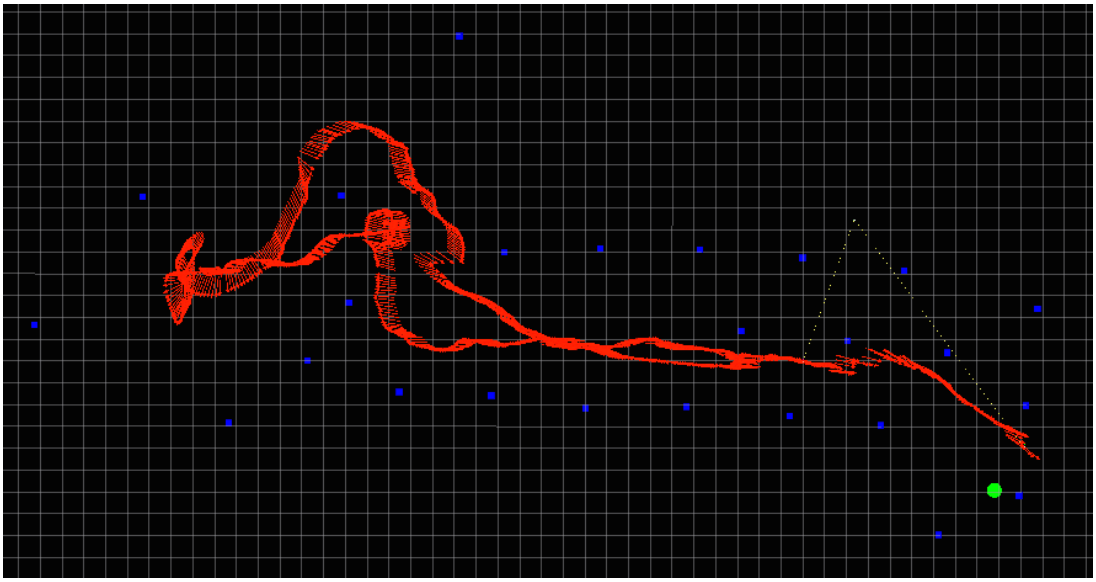

Figure 4.11: The path taken by the hexacopter through the tennis court trees.

Because of the noise at the start and end of the loop, it is not possible to determine how accurate the return to origin behaviour is. However, it is worth noting that the map derived is similar to the map derived when walking through the trees, and that the path computed accurately matches the path actually taken when flying the craft.

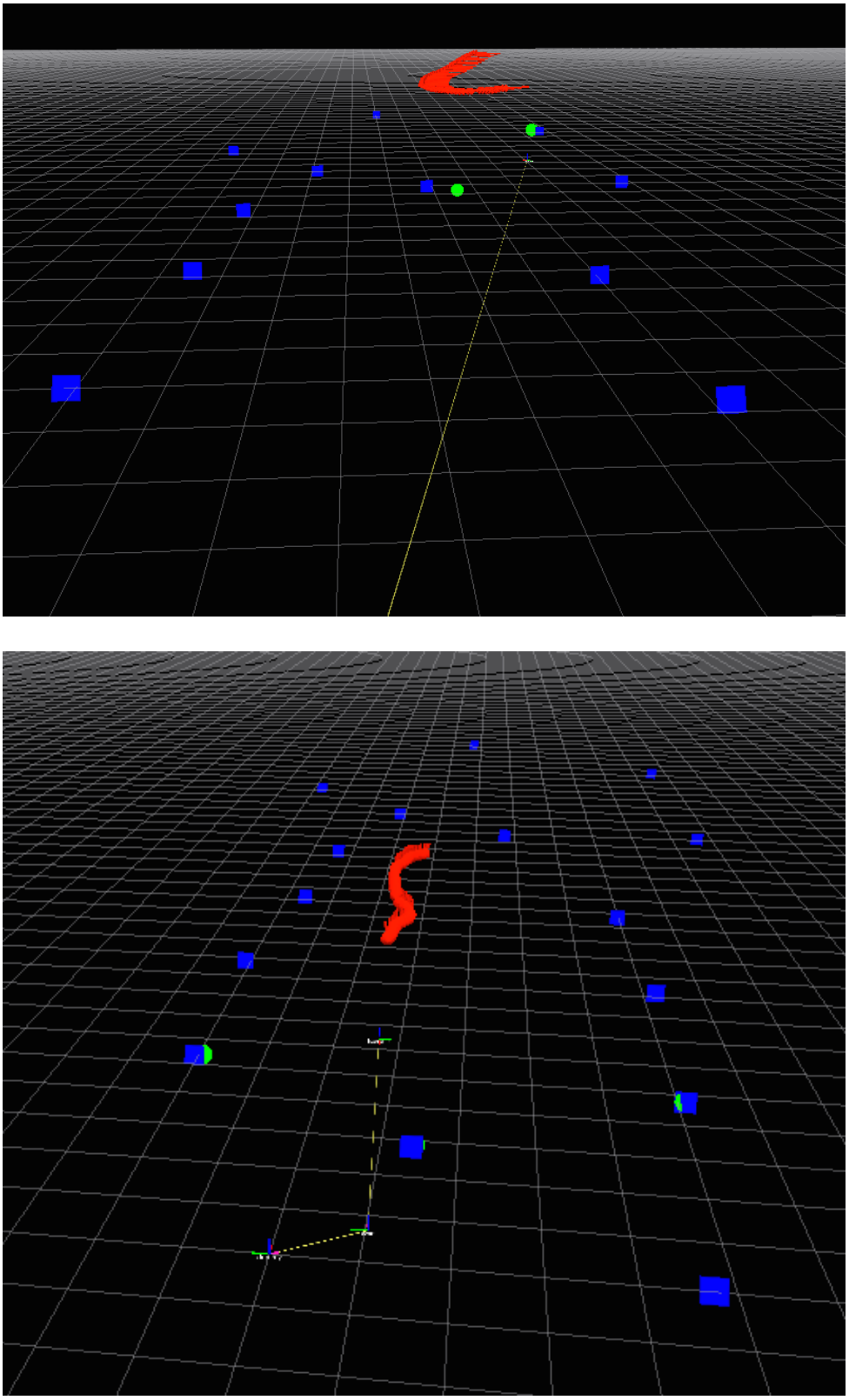

Figure 4.12: Two 3D snapshots of the flight.

Figure 4.13: A picture of the hexacopter in flight through the tennis court trees.

4.4.3 Conclusion — Flying

The big question remaining after the successful walking tests was whether or not the system could handle the high frequency vibrations caused by being flown.

The results show that the system can accurately localise itself when flying.

4.5 Discussion

Understanding the errors in the results is not straightforward. Broadly, error can be categorised into *sensor error*, and *algorithmic error*. Sensor error from the laser scanner appears to be small, and vibrations from flying the hexacopter does not seem to increase it. Some error which could most accurately be described as sensor-related was due to timing issues in recording data, but that was easy to address (see Section 4.2.1).

The algorithmic error comes from the polar scan matcher, the tree-based localization and the interaction between the two. These issues are explored in turn.

4.5.1 Analysis of PSM-related error

The negative results from walking around provided a lot of examples of causes of error.

Firstly, the polar scan matcher does not deal well with circling a single tree. This makes sense: a circle from any angle looks the same, so it's not surprising that the algorithm struggles. This is especially the case when there is only 1 tree in view.

More broadly, the polar scan matcher works much better the more points it can match. Sparse clumps of trees surrounded by clear space tend to work much worse than denser clumps of trees, which is why many of the negative results were for the RSISE trees and the results for the tennis courts and roadside were much more positive.

Lastly, foliage did not seem to interact well with the PSM.

Furthermore, errors in PSM tended to interact badly with the tree-based localisation: see Section 4.5.3.

4.5.2 Analysis of Tree-based Localization Error

Tree based localisation is itself prone to two different types of error: error in detecting trees, and error in using those trees for localization.

Tree Detection

The primitive nature of the tree detector affects its accuracy.

Firstly, there's a tradeoff in the number in tree points required for the detection of trees. If too many points are required, trees have to be very close before they're detected. If too few points are required, a single tree can be detected as multiple trees, and there are increased spurious detections.

More broadly, the detection algorithm detecting features rather than circles creates problems.

Inability to deal with different classes of feature: Using the minimum enclosing circle (MEC) with features means that any shape is treated as a circle. This makes the

algorithm unsuitable to be used in mixed environments, and also makes it perform poorly in the presence of people in the field of view.

Cannot use radii: The MEC detection is quite unstable: rather than detecting a stable circle with a near-stationary centre, the centre and especially the radius fluctuate wildly. This means the radius cannot be published and used for tree matching.

Cannot detect distant trees: An inevitable feature of laser scanners is that the detected points spread out. The minimum detectable width of an object at 10m is 130mm with the UTM-30LX scanner used (see Section 3.2.1). A decent sized tree should still get 3 or 4 points, which is enough to fit a circle. However, because the MEC detection requires many more points, and requires them to be close together when mapped to pixels, it cannot detect distant trees.

Ultimately, these issue can only be solved by moving to an algorithm that actually detects circles.

Further issues:

- The tree detector is also easily confused by foliage.
- The system models trees as perfect vertical columns of infinite height. Obviously this is not correct. A more appropriate model should allow trees to change height, to fork, to have foliage, and to have finite height.

Tree Localization

The tree localization is hamstrung by its nearest-neighbour algorithm. In short, there is no way to accurately locate a small set of points within a larger set of points without first having a position estimate. This means that any estimate that pushes the trees either beyond the threshold for nearest-neighbour, or pushes them closer to a different recorded tree will cause the points to fall out of sync, and once they are out of sync they cannot go back into sync of their own accord.

This is not an easy problem to solve, especially in a time-efficient way.

For example, finding the best match for 2 points in a map with 20 points requires testing 20×19 different pairs. Trying to match 3 points requires at worst $20 \times 19 \times 18$, and the complexity goes up factorially with additional trees.

The process is further complicated because the algorithm cannot be sure that all the points that have just been detected map to points on the map: some — or all! — of them could be new detections.

In addition, forests tend to look similar at small scales. Perhaps it's simply a quirk of the two environments mapped, but the a lot of pairs of trees have similar distances. This makes generic matching with only small numbers of trees particularly difficult.

Integrating the radii of trees into the matching algorithm would reduce the similarity. However, at the moment there's the assumption that the trees are a perfect column, with a constant radius the whole way around. If the measurements were not made at very similar heights, this assumption would fall apart, again complicating matters.

However, all of this is ignoring part of the available information. The current system is binary — for any given position, there is a tree or there is not. This is a nice simple way of understanding the world, but it does not encapsulate all the information available: it is also *known that some areas are empty.* This can be used: if an alignment would place a tree in an area known to be is empty, it is likely to be an incorrect alignment. In other words, accuracy can be improved with a probabilistic occupancy grid mapping, albeit at the cost of greater resources. This is a common approach taken by SLAM systems, including the ROS package `slam_gmapping`.

This approach could also be extended to three dimensions by taking up the idea of voxels: 'volume pixels'. This would enable the system to work across a range of heights. However, it would massively increase both complexity and storage requirements, especially with a naive 3D grid approach. Indeed, the complexity would have almost reached the level of the stereoscopic camera approach.

4.5.3 Interaction

The tree localization system is unfortunately rather sensitive to the errors occurring in PSM stage. If an error occurs — even momentarily — that causes a tree to be pushed away from it's proper nearest neighbour and into the neighbourhood of another tree, it will be matched with that tree, and there is presently very little way to recover from such an occurrence. The best way to rectify this would be to replace or integrate the `tree_fix` node with a proper SLAM system.

4.5.4 Flight

The flight trials raise a number of interesting questions and issues.

Firstly, how should take-off and landing be handled? The localisation is particularly likely to fail near the ground, especially during takeoff and landing. Looking at the laser scans, the data seems to be mostly "noisy", and not meaningful.

This could be due to a number of different factors. The uneven nature of the ground near trees is an obvious candidate. Any deviation of the scan plane from parallel will also be much more noticeable on near the ground.

To deal with this, the `tree_fix` package was patched to also consume the `/height` topic from `scan_to_height`, and only process the laser scan when the height is above a threshold, currently 1m.

Future work could look at integrating the low altitude data into a more complete model, although it would be advisable to discard very low data — say below 0.5m or so.

Secondly, how should other noisy/erroneous/confusing data be handled? The tennis court flight in particular features two kinds of noise: a person getting in the laser scan, and tree foliage.

With regards to people, the algorithm is particularly hamstrung by the inability to *remove* points from its view of the world. The noise point stays on the map, and tends to cause problems with matching. Replacing the 'map' with a proper representation that takes uncertainty into account would help to alleviate this.

With regards to foliage, again better image processing is likely to be key to fixing this. The problem can be reduced by increasing the points threshold for a match to 90, but this comes at the cost of reduced detection. Perhaps a more appropriate fix is to try and find a circle that includes all the noise and is roughly centred on the tree trunk. This again ties in well with the concept of a 3D model of trees, rather than a pure 2D one.

Another apparent fix observed during testing was simply to process scans at a lower rate: instead of processing 1 in 10 (4 Hz), processing 1 in 20 (2 Hz) reduced the problem. This is probably an artefact of the particular data set — it just managed to skip the problematic frames by chance. It should not be considered as a general approach to the problem.

Chapter 5

Conclusion

This report set out to answer the question:

> **Are laser scanners feasible for autonomous navigation of hexacopters in forests?**

The results and contributions of this project can be summarised as follows.

In terms of hardware, this project has contributed a payload with a gimbal and anti-vibration mount which successfully kept the scanner stable when walking and when flying, such that the data was not unduly affected by vibrations, and accuracy was not degraded.

In terms of software, this project has contributed a modified polar scan matcher and tree based SLAM system with the following characteristics:

- The modified polar scan matcher provides a position estimate in a forest-like environment that is accurate in the short term but drifts in the long term.
- The tree detection SLAM provides corrections to the position that provide excellent accuracy over the long term. The system is, however, somewhat unstable in the presence of error.
- The entire process can be completed in real time on the on-board computer, without requiring that any computation be offloaded to a base station.

Therefore, this project supports the conclusion that laser scanners can provide real-time laser odometry for autonomous navigation of hexacopters in forests.

5.1 Future Work: Software

5.1.1 Summary of key Node-related Work

A number of pieces of future work are detailed in the Software section (Section 3.3). This section seeks to highlight what is considered to be the biggest and most pressing aspects for future work.

Firstly, the filter used in the tree-localization node is a primitive averaging filter. This should be replaced with a Kalman filter. Both the polar scan matcher node and the tree matching processes provide covariance estimates, and provide or can trivially provide velocity estimates as well, making them ideally suited to being combined in this way.

Secondly, the tree detection used in the tree-detector node is so basic that it is surprising that it works as well as it does. Updating it to actually fit circles properly will vastly increase the stability of tree detection, reduce error, make it possible to provide and use radius data, and — implemented carefully — it will make the processing much faster.

5.1.2 Future Directions in Tree Detection

The tree-detector is based around converting the scan from its polar form into a cartesian form, and then processing that.

A number of laser scan matching algorithms take a similar approach, but significant efficiency gains were realised in [8] by processing the scan in its polar form, without conversion to cartesian form. Would it be possible to realise similar efficiency gains by detecting trees in polar form? Does the structure of the polar scan provide opportunities to increase detection accuracy?

To investigate, some simple simulation was conducted in MATLAB.

Firstly, as shown in Figure 5.1, a 'generic' scene was transformed from the cartesian form to a polar representation, where the scan angle forms the x-axis, and the radius from the scanner forms the y-axis. This shows the features one would readily expect from a polar coordinate form: for example, lines become curves. This seems to show the trees with a somewhat distinctive shape, but it's hard to determine quite what.

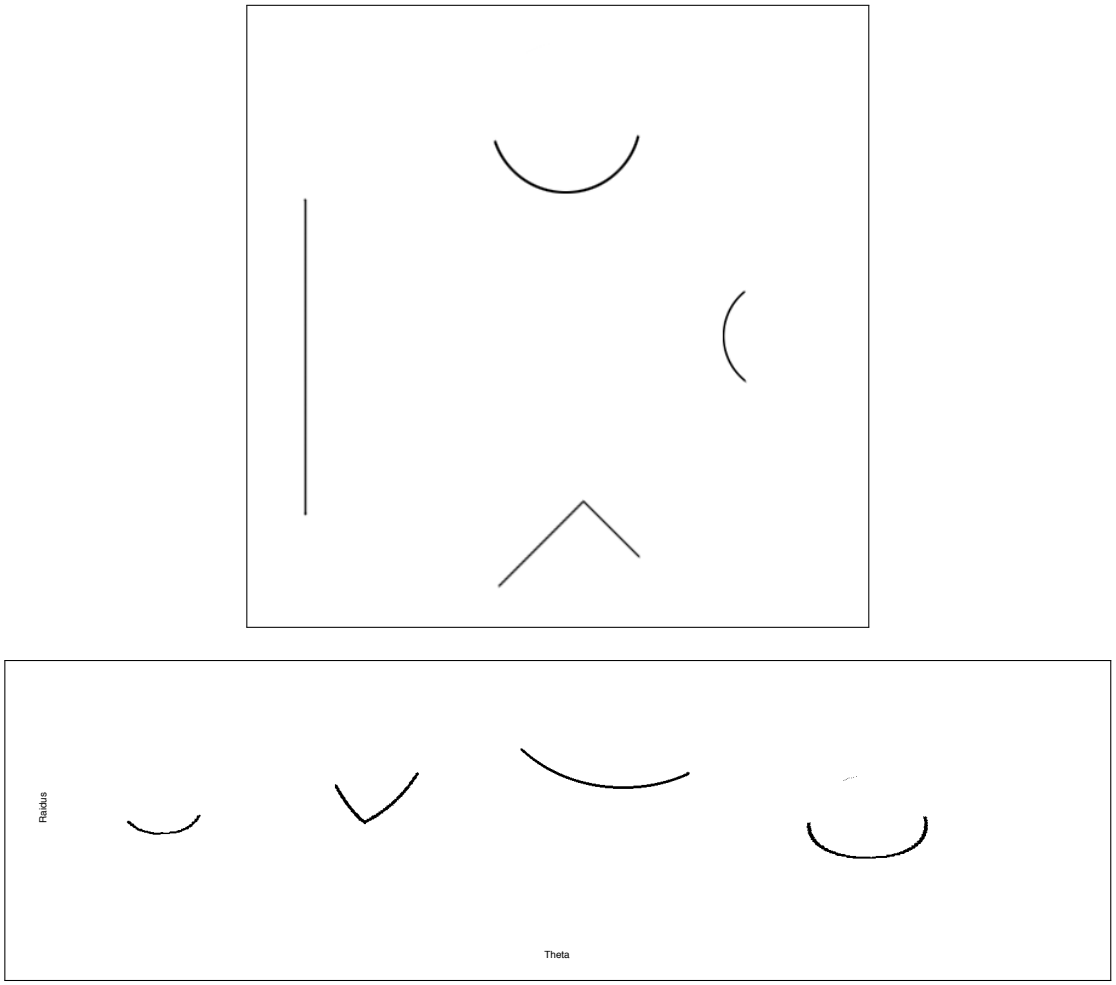

Figure 5.1: A generic scene (top) transformed from cartesian to polar and plotted (bottom). The origin angle has been shifted to aid comprehension.

To get a better idea of what we're looking for, Figure 5.2 shows what would happen if the entire tree was known in the polar coordinate-system. (Obviously by the nature of the laser scanner this isn't possible in an actual scan, but it's nonetheless helpful to understand the shape.) This reveals an 'egg' shape.

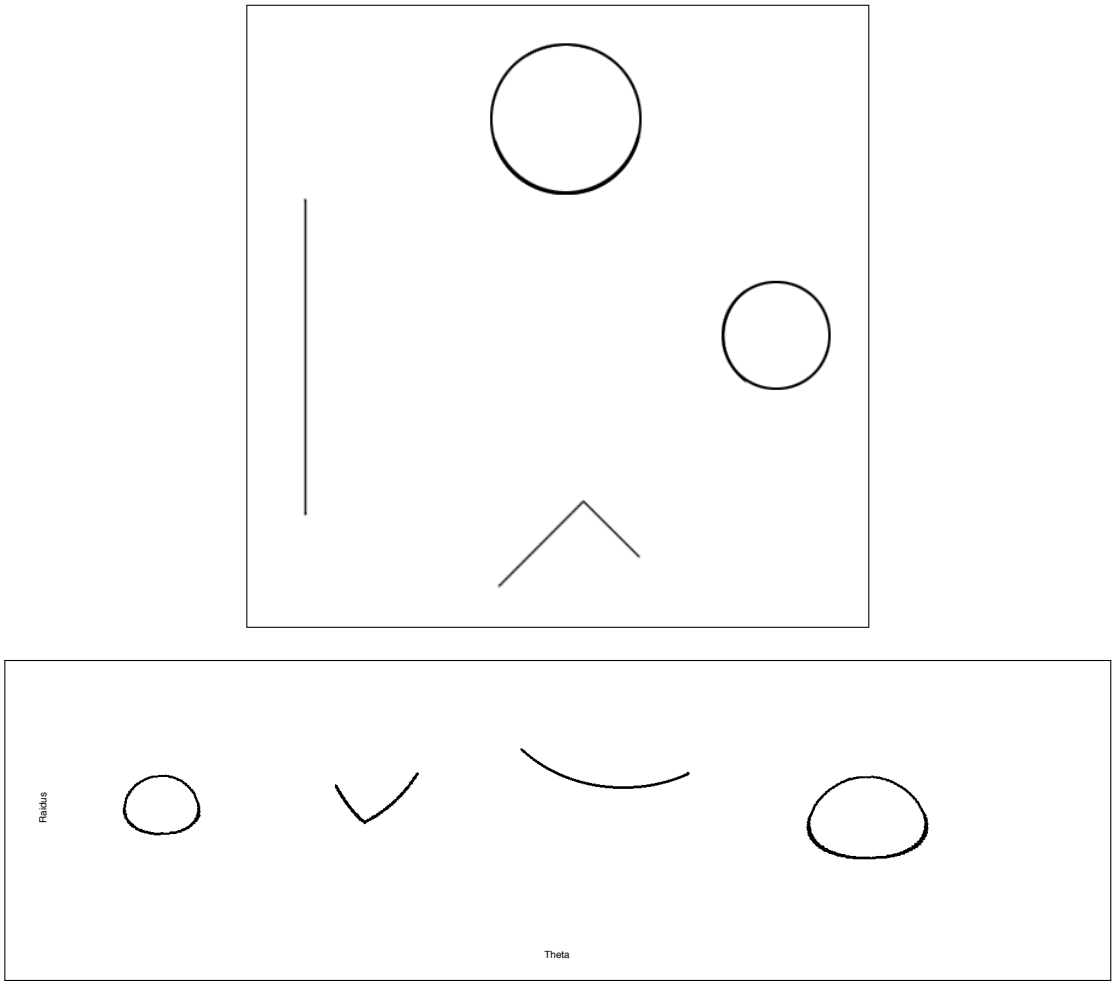

Figure 5.2: A generic scene with complete circles (top) transformed from cartesian to polar and plotted (bottom). The origin angle has been shifted to aid comprehension.

A further test in Figure 5.3 shows a scene with just trees, different sizes and at different radial distances; investigating the change this makes in shape.

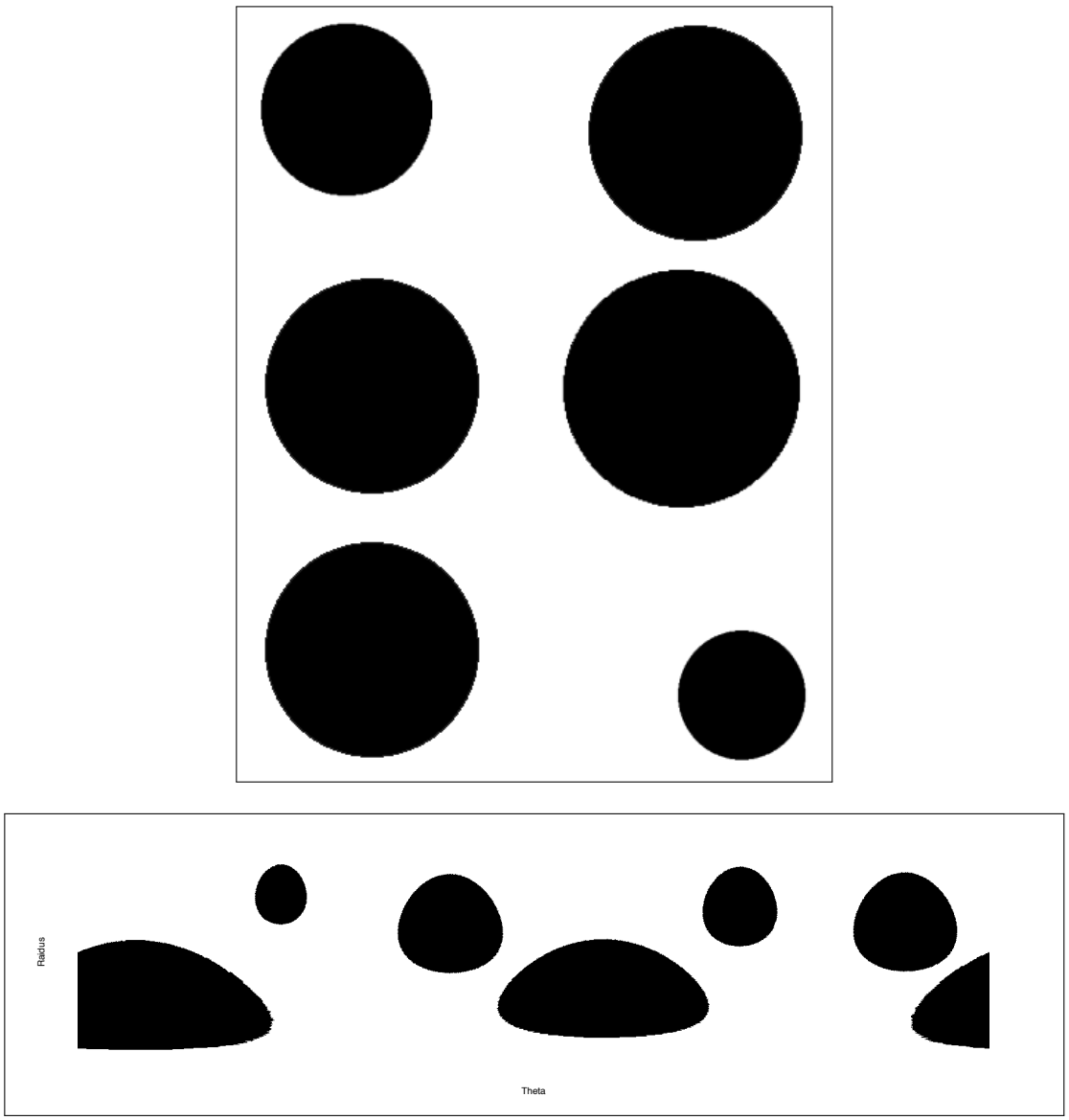

Figure 5.3: A generic scene with complete circles (top) transformed from cartesian to polar and plotted (bottom). The origin angle has been shifted to aid comprehension.

Overall, these figures show that cartesian circles, when examined in polar form, regardless of location, transform into quite a unique 'egg' shape, which can be distinguished from other types of shape, including lines and corners.

A more rigorous mathematical approach would be required to quantify the parameters of the shape, but once that is done, it may be possible to detect circles in the polar form, leading to time and memory savings.

Furthermore, this process should be more resilient against false detections because it can take advantage of the polar nature of the scan: not only does it avoid a large number of

multiplications and a memory accesses, it does not require a continuous (in the Cartesian frame) set of points that the current detector does. Processing int he polar frame should also allow the blurring step to be removed, which should improve accuracy: while the blurring step is important for the current detection method to work, it makes it easier to fit a circle to noise. Furthermore, methods derived from the polar frame may require fewer points to get a meaningful result, allowing for detection at greater range.

Ultimately, this has potential to be quite a useful avenue for future investigation.

5.1.3 Maintenance

A final point worth highlighting is software maintenance. A large amount of effort went into updating the Polar Scan Matcher node to work with recent versions of ROS. For this code, and the other code written for this project to continue to be useful in the future, it will need to be kept up to date, preferably as new versions of ROS come out, rather than in one big jump several versions down the line. Fortunately, by placing the data and code on GitHub, it becomes much easier to use a Continuous Integration platform like Travis CI[1] to test against multiple versions of ROS. Those who continue this project into the future are strongly encouraged to take up this opportunity to bring modern software practises to bear on the project.

[1] `https://travis-ci.org`

5.2 Hardware: Future Work

5.2.1 Weight

A major concern regarding the hardware is the weight. Requiring 70% throttle severely limits the flight time of the hexacopter, and also makes it harder to fly. How can weight be reduced?

Battery

The current setup uses separate batteries for the hexacopter (motors, radios, etc) and for the payload. This was to reduce the likelihood that a sudden spike in current draw (e.g. from climbing suddenly) from causing a voltage drop that would cause the onboard computer to shut down, or lead to corrupted data.

However, the payload battery lasts for upwards of an hour, whereas the flight battery lasts for mere minutes. This is evidently not optimal. How could this be addressed?

Can we use a single battery pack, keeping to computer and scanner voltage stable by use of large capacitors? Probably not: once the voltage drops exceed the millisecond time scale, the amount of capacitance required becomes incredibly onerous.

Instead, it ought to be possible to better configure the batteries so that the *endurance* of the packs is roughly matched — that is, that the flight time roughly equals the length of time for which the payload runs. An outline of how that might be achieved follows.

Firstly, the gimbal set could be connected to the main battery. Not only is the gimbal the single biggest consumer of power in the payload, its firmware can be configured to expect and compensate for fluctuating power.

This leaves just the scanner and the PC connected to the secondary battery. On average these together use around 1.5A. Say they are to run for 30 minutes; that gives $1.5 \times 12 \times 0.5 = 9$Wh, or $1.5 \times 0.5 \times 1000 = 750$mAh (assuming a 12V source). This could be supplied with a lighter battery.

Unfortunately, LiPo batteries don't exactly hit 12V: a 3 cell battery will supply 11.1V, and a 4 cell battery 14.8V. To minimise weight, it would be preferable to use an 11.1V

battery. If a boost converter is used with 90% efficiency:

$$I_{\text{batt}} = 1.5 \times \frac{12}{11.1} \times \frac{1}{0.9} = 1.80 \text{ A} \tag{5.1}$$

At 1.80A, a battery would need 900mAh of capacity to last half an hour, or if we want to ensure that we never drop below 25% capacity:

$$0.75c = 1.80 \times 0.5 \Rightarrow c = 1200 \text{ mAh} \tag{5.2}$$

A 1200mAh 3 cell LiPo would weigh significantly less than the current 3300mAh, 4 cell battery (330g). (For instance, a 1800mAh battery weighs in at 110g, and a 2200mAh battery weighs in at 163g.)

PC Selection

The previous section took as fixed the computer used in the system. What if it wasn't? It may be possible to change the computer for one which runs on 5V only, allowing a much greater voltage variation to occur before problems are experienced. Two obvious candidates are the Raspberry Pi,[2] and the BeagleBone Black.[3] Both of these boards run on 5V, consume vastly less power, and weigh much less than the LP-170C board currently in use. However, they come with their own set of complications:

Computational Power The LP-170C has a 1.8GHz Atom CPU, whereas the BeagleBone has a 1GHz ARM Cortex A8, and the Raspberry Pi has a 700MHz ARMv6 CPU.

This obviously gives the LP-170C a significant (almost 2-fold) advantage in sheer computational grunt. Whether or not this will actually be an issue is unknown. Certainly with the current inefficient tree detection, the difference is likely to require a lower frame rate, but if the process can be done without conversion from polar to rectangular and the use of costly OpenCV operations, this advantage may fall away.

Memory The LP-170C has 4GB of memory, compared to 512MB on both the BeagleBone and the Raspberry Pi. While that's significantly less, it is very likely that

[2] `http://www.raspberrypi.org/`
[3] `http://beagleboard.org/Products/BeagleBone+Black`

most of the extra memory on the LP-170C is sitting idle. The board is not doing extensive graphics processing; it's not running a desktop environment; it's not doing particularly memory-intensive processing. (For instance, the map produced by the tree detector is presently 1200 by 1200 pixels, which given it's greyscale, equates to a mere 1.4 MB.)

I/O The LP-170C uses a Compact Flash (CF) card, the Raspberry Pi uses a Secure Digital (SD) card, and the BeagleBone has 4 GB of onboard flash.

There are two relevant considerations: capacity and speed. While the BeagleBone has the most limited storage, it's still more than enough for a complete ROS distribution, especially if installed with care to avoid bringing in unnecessary items like the visualisation package `rviz` and a desktop environment.

Of more pressing concern is the speed of the relevant storage technologies. The speed of writing a 768KB chunk has already proven to be problematic on the LP-170C (see Section 4.2.1), and Raspberry Pis are notorious for poor IO performance. The BeagleBone's performance would need to be tested, but it's difficult to imagine it being significantly worse than either of the alternatives.

Extensibility All the solutions proposed support USB, which is by far the most likely avenue for extension peripherals. The LP also supports PCI-E mini-card. While many peripherals are available in that form factor, they're usually also available in USB form, so it's difficult to see how this would be of benefit. The Raspberry Pi and the BeagleBone Black support GPIO with readily accessible sets of pins, which may be useful for integrating with the autopilot system.

Graphics The three have varying graphics hardware. However, the only use for this hardware is in setup and debugging: no graphics card processing is being contemplated. All the systems are perfectly suitable for this.

In conclusion, the Raspberry Pi is probably unwise, given its low grunt and dismal IO, but serious consideration should be given to the BeagleBone Black, especially if it becomes possible to process the scan for trees using the polar method discussed in Section 5.1.2, as that would largely obviate the processing power issue.

If a BeagleBone Black could be deployed, it could safely be run on the main power with an appropriate regulator. That would leave only the laser scanner requiring a regulated 12V

source. The scanner is also probably less likely to be significantly affected by temporary fluctuations in voltage, although this would need to be verified. Ultimately this may enable the system to run on a single battery, which would be a significant weight saving.

Other Ways of Reducing Weight

As the landing gear was assembled largely as a prototype, there are other potential weight savings that could be realised by a redesign and rebuild.

With reference to Figure 5.4, the following aspects should be considered in future work:

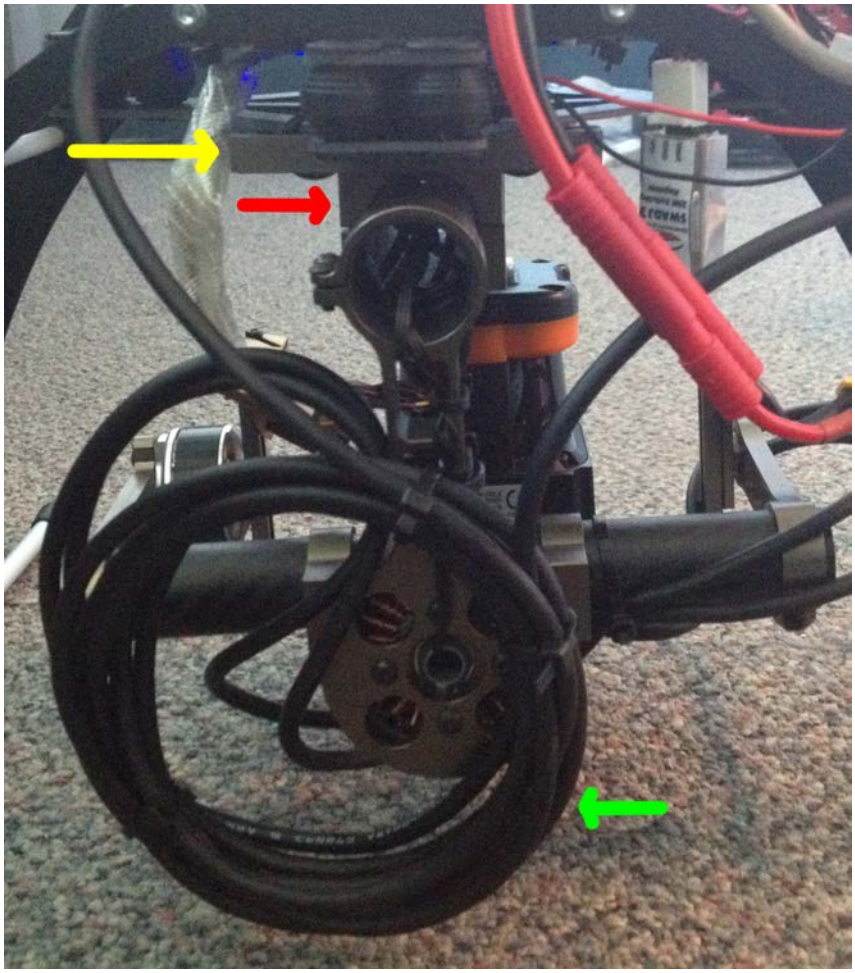

Figure 5.4: Potential savings in hardware weight.
Yellow: Gimbal top plate
Red: Gimbal clamps attaching to top plate
Green: Long power and USB cable from scanner.

- Presently, the gimbal is mounted to the anti-vibration mount by means of cable ties attaching the mount to the top plate of the gimbal (yellow arrow). The top plate itself is attached to the central beam by means of two metal clamps (the front clamp is shown by the red arrow). Future work should investigate removing the top

plate entirely, and attaching the beams to the anti-vibration mount either directly or though the existing clamps. This may require a redesign of the bottom plate of the anti-vibration mount, which has very unhelpfully positioned holes and cutouts.

- There is a lot of excess cable bulk (green arrow), comprising both a USB cable and a combined power/serial cable for the scanner. Due to the prototype nature of the project, the excess cable was not cut out. However, if the project proceeds with this scanner, the cables should be shortened.

An additional design goal that should be considered when reducing weight is to balance the hexacopter as a whole, which will increase its efficiency and make it easier to fly. This will require a more holistic design process than what was undertaken previously.

5.2.2 Landing Damage

A second concern is damage caused to the hardware by a hard landing on the second flight trial.

Regrettably, no pictures were taken immediately after the landing, so Figure 5.5 highlights areas of interest for discussion.

In the figure, the yellow circle identifies a gimbal joint that 'popped out', leading to a partial break along the red line. Fortunately the break was not all the way through, so it was possible to pop the joint back into place.

However, future work should address the obvious weakness this reveals. Instead of merely having a plate on the base, there should also be a plate at the top of the gimbal to provide the extra rigidity. The frame will need to go around the scanner, and the design will need to consider relocating the downwards reflecting mirror in line with Section 3.2.4.

The base plate should also be redesigned in line with Section 3.2.1: accidentally designing the screw holes to be too small may well have been a contributing factor. When re-fabricating the plate, it may also be worth considering a more durable material than 3mm MDF.

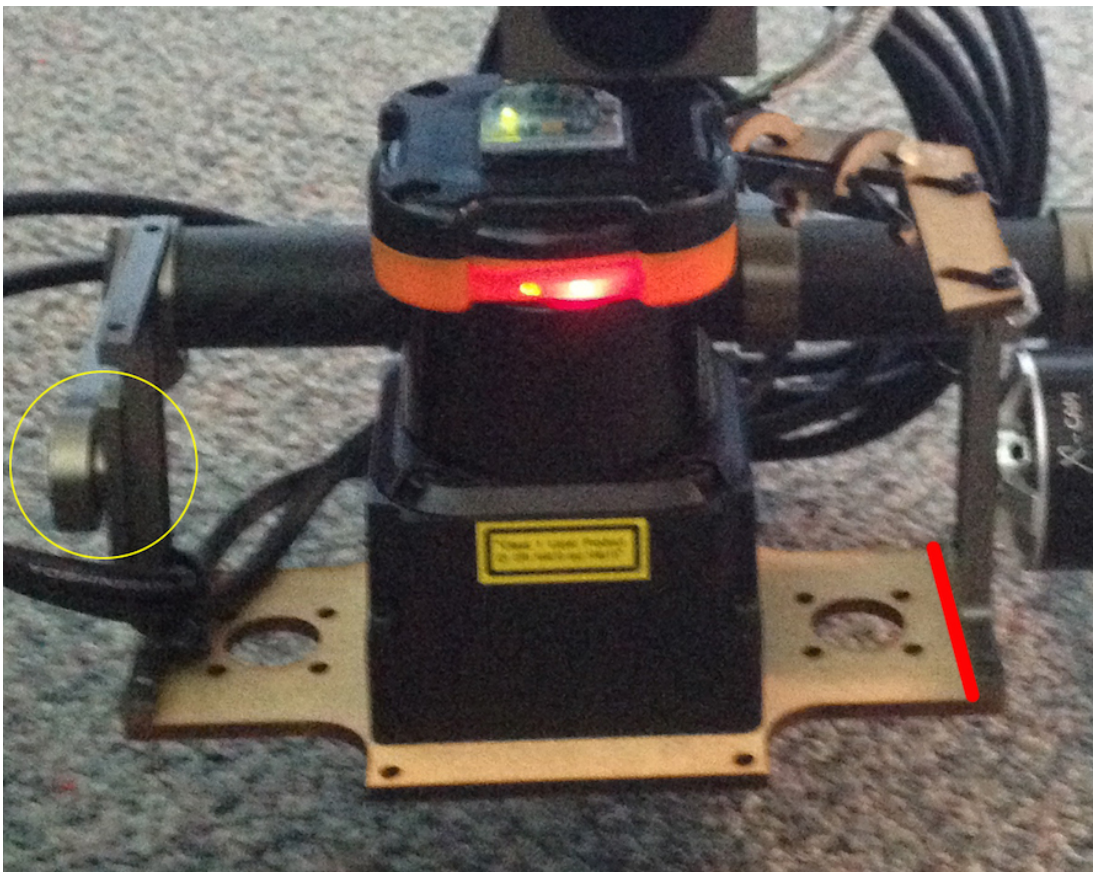

Figure 5.5: Identification of areas of interest from crash.

5.3 Future directions

This section considers future work for the system and project more generally.

5.3.1 Developing Autonomy

The obvious future step for this project is to link the positioning information into a control system. This ties in with the broader context of forest exploration and the problem posed in the introduction of locating people in a forest.

This will involve coordinating manual and automatic control, especially for takeoff and landing, as well as having a manual override in case of problems. However, ROS-MAVLink interaction is not a new area, and so existing resources can be found to aid in this process.

5.3.2 Other Directions

Following [3], it would be interesting to integrate the laser scanner with some sort of camera.

- Adding a Kinect or a stereoscopic camera (as in that paper) would open up a lot of interesting sensor fusion opportunities. More practically, it would help to obviate the fact that the laser scanner only provides a single plane, and would help the hexacopter change altitude with confidence that it wasn't going to hit something above or below it.
- Even adding a monocular camera would be quite interesting. It would enable simple colour-based detection of foliage and other non-tree obstacles, which could be used quite fruitfully to prune the map of trees.

In addition, getting access to data from the onboard IMU — even given its limited accuracy — would be useful. For example, in the final moments of the tennis court flight, there were no useful points for the laser scanner. Having access to IMU data would have given at least a rough estimate of the rotation of the hexacopter, which would have been very helpful.

A particularly interesting direction that could be considered, especially with reference to the problem of forest exploration posed in the introduction, is the use of multiple cooperative hexacopters.

5.4 Acknowledgements

Lastly, I wish to acknowledge:

- Jon Kim, for his able supervision, his consistently helpful suggestions and encouragement, and for flying the hexacopter.
- Alex Martin, for his work fabricating the laser-cut parts, providing the front-faced mirror, and for his helpful advice on construction.

Bibliography

[1] S. Ahrens, D. Levine, G. Andrews, and J. How, "Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, May 2009, pp. 2643–2648.

[2] 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)*, 2011, pp. 1–16.

[3] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Stereo vision and laser odometry for autonomous helicopters in GPS-denied indoor environments," in *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 2009, pp. 733 219–733 219.

[4] J. Langelaan and S. Rock, "Towards autonomous UAV flight in forests," in *Guidance, Navigation and Control Conference*, 2005.

[5] A. Bachrach, R. He, and N. Roy, "Autonomous flight in unstructured and unknown indoor environments," in *Proceedings of EMAV*, 2009.

[6] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, Feb. 1992. [Online]. Available: http://dx.doi.org/10.1109/34.121791

[7] A. Censi, "An icp variant using a point-to-line metric," in *2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 19–25.

[8] A. Diosi and L. Kleeman, "Fast laser scan matching using polar coordinates," *The International Journal of Robotics Research*, vol. 26, no. 10, pp. 1125–1153, 2007.

[9] M. Quigley, K. Conley, B. P. 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*, 2009. [Online]. Available: http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

[10] W. Meeussen. (2010) REP105: Coordinate Frames for Mobile Platforms. [Online]. Available: http://www.ros.org/reps/rep-0105.html

[11] T. chuan Chen and K. liang Chung, "An efficient randomized algorithm for detecting circles," *Comput. Vis. Image Underst*, vol. 83, p. 2001, 2001.