AN INVESTIGATION OF TRAJECTORY TRACKING AND REPLICATION ON COMMODITY DRONES

CHRISTOFFER SCHMIDT(201408694)

JONAS LE FEVRE SEJERSEN(201407676)

MASTER'S THESIS

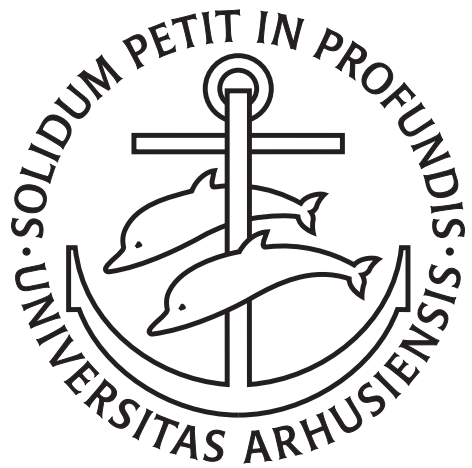
June 2019

Advisor: Kaj Grønbæk

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

AN INVESTIGATION OF TRAJECTORY TRACKING AND REPLICATION ON COMMODITY DRONES

CHRISTOFFER SCHMIDT, JONAS LE FEVRE SEJERSEN



Master's Thesis
Department of Computer Science
Science & Technology
Aarhus University

June 2019

: *An Investigation of Trajectory Tracking and Replication on Commodity Drones* , Master's Thesis, © June 2019

Ohana means family.
Family means nobody gets left behind, or forgotten.

— Lilo & Stitch

ABSTRACT

Based on an inspirational use case in which a drone is tasked to survey a parcel sorter, this report details an investigation of the possibility of performing automated drone flight based on a given ground truth trajectory. More specifically, with a focus on the condition that the equipment used should be affordable commodity hardware. The different existing technologies used are analyzed and a system solving the problem is constructed, based on a visual SLAM technique where we contribute with additional modules for alleviating ground truth decay through keyframe paths, scale estimation problems through a control based scale estimation, and a look at different controller types. Additionally then the options for a technique employing inertial data is investigated. All of these solutions are tested in an experimental process. The investigation produces a system which solves the problem with a surprising level of accuracy and with an average approximate RMS of about five centimeters. In addition, then several reflections concerning the solution of the task are made, which do not necessarily contribute directly to the full solution.

A video demonstrating the solution and the process of running the solution may be found on https://youtu.be/haKltQHmPk4

RESUMÉ

Inspireret af en use case hvori en drone har ved opgave af flyve rundt on et pakkesorteringsbånd, beskriver denne rapport en undersøgelse af mulighederne inden for automatiseret droneflyvning baseret på en given ground-truth rute. Der fokuseres specifikt desuden på den begrænsning, af det brugte udstyr bør være prismæssigt overkommeligt. De eksisterende teknologier er analyseret, og et system som løser problemet er sammensat, baseret på en visual SLAM teknik hvori vi udbygger med moduler som lindrer problemer med henfald i grouth truth, estimering a skalering via. en kontrolbaseret teknik. Der kigges desuden også på forskellige kontroltyper. Yderligere undersøges mulighederne for inkorporering af inertidata også. Alle disse løsninger testes i en eksperimentprocess. Undersøgelsen producerer et system med en overraskende høj mængde precision med en gennemsnitlig RMS på cirka 5cm. Der laves desuden også adskillige refleksioner vedrørende løsningen af opgaven, som ikke nødvendigvis er i direkte relation til den endelige løsning.

Der kan findes en demonstrationsvideo af løsningen of den fulde process af at køre løsningen på https://youtu.be/haKltQHmPk4

CONTENTS

Foreword 1

1 INTRODUCTION 3
- 1.1 Use Case 3
- 1.2 Project Scope 4
  - 1.2.1 Commodity Equipment 5
- 1.3 Central Hypotheses 6
- 1.4 Project Objectives 6
- 1.5 Thesis Structure 7

2 RELATED WORK 9
- 2.1 Previous Drone Research 9
  - 2.1.1 Localization 10
  - 2.1.2 Scale Estimation 11
  - 2.1.3 Drone Navigation 12
- 2.2 Theory of the engineering of navigation controllers 12
  - 2.2.1 Proportional–Integral–Derivative Control 13
  - 2.2.2 Model predictive Control 14
- 2.3 Computer Vision 16
  - 2.3.1 What is visual SLAM? 16
  - 2.3.2 ORBSLAM 18
  - 2.3.3 Visual Inertial SLAM 20
- 2.4 Filtering of Inertial Data 22
- 2.5 Synchronization 22

3 CHOOSING METHODS FOR THE PROJECT 25
- 3.1 Localizing the Drone 25
  - 3.1.1 A Visual-Only Solution 26
  - 3.1.2 A Visual Inertial Solution 27
- 3.2 Navigation and Control 28

4 DESIGN AND IMPLEMENTATION OF DRONE SYSTEM 31
- 4.1 The Design Vision 31
- 4.2 Architecture of the System 34
  - 4.2.1 Input Data 35
  - 4.2.2 Location and Mapping 35
  - 4.2.3 Scale Estimation 37
  - 4.2.4 Path Planning and Segmentation 40
  - 4.2.5 Drone Controller and Output 41
- 4.3 Choice of Hardware 45
- 4.4 Experiments with vision-only solution 48
  - 4.4.1 Simulated experiments and setup 48
  - 4.4.2 Real world experiments and the environmental setup 53
  - 4.4.3 A note on the current status of the drone 62
- 4.5 Experiments with visual-inertial solution 62

5 REFLECTING ON ATTEMPTED METHODS 65
5.1 Choosing the right SLAM method 65
5.2 Model based controller 66
5.3 Discussion of ground-truth 66
5.4 Tracking camera errors in the experimental results 68
6 CONCLUSION 71
6.1 Future Work 72

I APPENDIX 73
A THE ORIGINAL USE CASE 75
B INERTIAL MEASUREMENT UNIT DATA SHEET 79
C ROS RQT GRAPHS 81

BIBLIOGRAPHY 83

LIST OF FIGURES

Figure 1 An example of the loopsorters present at BEUMER Group and the use case (and in public advertizing material). Image used with permission by BEUMER Group Aarhus. 4

Figure 2 A block diagram of a PID controller in a feedback loop. 14

Figure 3 A block diagram of a Model Predictive Controller. Outside the Model Predictive Controller (on the right), we have a plant which is the object which we are manipulating, where as in our case this is the drone. The state Estimator is the actual result of performing the command τ on the plant, where the prediction model is updated with the new state of the plant, x. 15

Figure 4 Direct vs indirect approach[13]. 17

Figure 5 On the left, we see a sparse mapping created by PTAM, where the coloured points are the extracted features. In the middle, a semi-dense mapping created by LSD-SLAM, where the coloured points are the high gradient areas. Last, on the right, a dense mapping by DTAM, where all the points on the surface are part of the map[14]. 18

Figure 6 The workflow of ORBSLAM2. As ORBSLAM2 was an extension to the original utilizing stereo and RGBD cameras, this picture describes the frame as a stereo or RGBD frame input, but a single frame would work as well. 19

Figure 7 An overview of the VINS system, split into the four processes. 20

Figure 8 Drone pathing using a Proportional–integral–derivative (PID) controller and how drift affects the error. 29

Figure 9 A schematic view of the full design vision imagined to solve the primary task. 33

Figure 10 The architecture which is focused on in this implementation. The colored background indicates separate machines, depending on the choice of solution. The yellow background signifies a singular machine for the non-distributed inertial solution, and the separate blue and red signify devices on the distributed solution. Of course, the visual only solution does not include anything from the red section. 34
Figure 11 A sketch of the structure of the control based scale estimation. 39
Figure 12 On the left the filtering of noisy segments is shown. The goal point of Segment B is within the error margin of Segment A, so a new segment C is created to be a direct trajectory from the goal point of segment A to the goal point of the old Segment C. On the right is the error proportional to the size of the error margin for a segment. 41
Figure 13 Proportional controller with drift correction. In this image, the error is corrected in 3 commands, hence the three color error, but in reality this would take a lot more. 42
Figure 14 The IMU + Raspberry Pi mount. 49
Figure 15 An example of the drone being simulated in the environment of Chillon Castle. 50
Figure 16 A set of plots showing the scale estimate over time. All of these were performed in a simulation of Chillon Castle. 52
Figure 17 An example of two separate scalings performed in the same area for two different maps. This image displays the most disparate examples, as most runs would look too similar. 54
Figure 18 The physical testing grounds. The first showcases the environment as a whole. The blue circle indicates the placement of one of the cameras for recording. The second image displays the two downwards-facing cameras used for the horizontal axis tracking. 55
Figure 19 Controllers evaluated on a Z path. On the left we see the result of the PID controller (red) replicating the ground truth path (green). On the right, the result of the proportional controller with drift correction. Here, the focus is on the overshooting, while trying to correct the drift error. 56

Figure 20 The L and Z trajectories of the flights plotted together. Please note that these plots have been scaled, and that the axes measure in pixels. 60
Figure 21 The R and C trajectories of the flights plotted together. Please note that these plots have been scaled, and that the axes measure in pixels. 61
Figure 22 The RQT graph of the system, showing all active topics 82

LIST OF TABLES

Table 1 tuning of the PID controllers 45
Table 2 Bebop 2 specifications 46
Table 3 The calculated root-mean-square errors of the flights. The flight paths calculated upon were trajectories L, R, C, and Z 62

ACRONYMS

UAV Unmanned Aerial Vehicle

MAV Micro Air Vehicle

SLAM Simultaneous Localization and Mapping

IMU Inertial Measurement Unit

ROS Robot Operating System

GPS Global Positioning System

GLONASS GLObal NAvigation Satellite System

LIDAR Laser Imaging Detection And Ranging

PID Proportional–integral–derivative

MPC Model predictive controller

NTP Network Time Protocol

PTP Precision Time Protocol

DfD Depth from Defocus

FOREWORD

Throughout the execution of this project, we have received assistance from plenty of helpful people, whom we would like to thank. First of all we would like to thank our advisor Kaj Grønbæk, who has helped us obtain the most results, by setting our limits realistically, and by helping with many aspects of the project planning and presentation. We would also like to thank Astrid Høegh Tyrsted and anyone else who helped us gaining access to the ESK Hall for experimentation with the drone. In conjunction, we would like to thank everyone who helped look for an alternate flying location, especially all of the users of the cafeteria of Kristiansvej 13 8660 Skanderborg, for permitting us to perform preliminary experimentation at their location, in particular Michael Schmidt who originally suggested it. We would also like to thank all of our proofreaders: Anna Luise Bülow Row, Michael Schmidt, Jill Schmidt, and Susan Schmidt Mortensen.

1 INTRODUCTION

Before getting into the specifics of this project we begin with an overall introduction of the use case which the project is inspired by, the considerations that follow, and how this eventually leads to the set of central hypotheses which we have investigated in this project. We give a description of the project as a whole and how the final conclusions have formed. Following this, we give a brief overview of the report structure, so we may proceed onto the related work.

1.1 USE CASE

The thesis presented in this report takes inspiration from a use case originally provided by BEUMER Group Aarhus. This use case goes as follows: In the express parcel industry, an increasing number of B2C parcels, as a result of ecommerce is observed. This increase has lead to a solution involving automatic parcel sorting, involving one or more large "loop sorters" (see figure 1). With these loop sorters carrying parcels at speeds up to 2.5 m/s, a certain risk of parcels falling off this sorter is detected, commonly due to low friction or incorrect positioning. When such incidents occur, the parcels will usually end up in safety netting on the sides of the sorter, which will then need to be checked regularly by a manual labourer. The problem presented is then to eliminate the necessity of these checkups by making use of one or more Unmanned Aerial Vehicle (UAV)s or "drones".[1] Primarily, the idea posed in the use case is thus a system which performs the following tasks:

1. A drone supervising a stretch of a specified area for parcels by flying automatically around the area, ensuring that it sees the specified nettings (or other points of interest).
2. The drone identifies any incidents in the form of parcels having fallen off and entered the nettings.
3. Finally, the identified incident is lifted away from the netting.

While this certainly would be very impressive, this thesis is not a full implementation of this system, as certain parts of it are somewhat

1 There are many different pieces of nomenclature surrounding the device in question. These include UAV, Micro Air Vehicle (MAV), "drone", and at times a more specific descriptor such as "small quadrotor" for the most common four-winged example. As most of these names are quite unwieldy, we will refer to these throughout this report as "drones".

Figure 1: An example of the loopsorters present at BEUMER Group and the use case (and in public advertizing material). Image used with permission by BEUMER Group Aarhus.

out of scope. The idea, however, has been used as inspiration for identifying new ways of experimenting with autonomous patrolling drones. The area which we then wished to investigate specifically was the first point, namely flying over the specified stretch automatically, whilst ensuring that any points on the path were reached by the drone by an acceptable margin. The original proposal for the use case can be seen in appendix A.

1.2 PROJECT SCOPE

The primary task can thus be summarized as such: After providing a map with a given trajectory through an environment, the drone will then, when prompted, be able to patrol this environment with a sufficient level of accuracy. The central idea is then, to apply some of the techniques of modern drone research, i.e. computer vision based solutions together with tested control methods. With a few tweaks, such as methods for scale estimation, applied to alleviate certain issues introduced by the use of these techniques and as a result would make it possible to perform the navigation well, and consistently. Additionally, we also investigate the possibility of making use of inertial data in the navigation, and whether it is possible to perform the task given its visual and inertial input from two different devices in a distributed fashion. If not, the discoveries should at least speak to the capabilities of this technology, given the description of the primary task. As an added challenge, the use case motivates yet another quite common restriction when it comes to drones, namely the subject of pricing.

1.2.1 *Commodity Equipment*

An essential part of this investigation is the fact that the original use case places itself in an industrial setting, presumably with tens, if not hundreds, of the aforementioned drones doing their patrol. The reason why this is an important distinction is that the field of autonomous flying robots or the entire field of robotics as a whole is an expensive one. The machines themselves are expensive, but for the automation to be applicable, using existing research, one would typically find themselves in need of a high-quality global shutter camera with an array of high-frequency hardware synchronized sensors which function in symbiosis with the drone and the camera(s). This quickly amounts to hefty sums of money and major configuration of sensors and cameras, which then becomes a further problem as a multitude of them are required in the air for the full use case. Thus, it often becomes unrealistic for a setting such as the one provided in the use case. With this in mind, we set to determine whether it would be possible to apply some of the existing findings in the fields of computer vision, ubiquitous computing and robotics with affordable equipment. This includes things such as single rolling shutter cameras whose Simultaneous Localization and Mapping (SLAM) output may cause photometric distortions, potentially noisy sensors outside of the drone hardware, etc. It is then hypothesized that a sufficient amount of accuracy still may be achieved within a certain bound using a visual SLAM technique for localization only. Based on the results of such an experiment, the possibilities within the area of inertial data may then be investigated. Here, the expectation is to see whether one can enhance the localization by applying enough careful adjustment in the form of accurate calibrations, various forms of synchronization, and meticulous noise integration/filtering. All of these points are attempts to remove common sources of error in inertial solutions such as sensor bias and noise, which are especially relevant in the context of the "affordable" equipment. Additionally, the use of disconnected sensors makes for an interesting investigation of applications of the mentioned technologies in distributed settings, i.e. the setting in which the inertial data and the visual data originate from two different devices, as synchronization is often essential in conventional solutions.

The existence of the use case should make it clear why investigating these hypotheses and potentially confirming the possibility of such an implementation could be important. As it confirms that it is possible to make such a setup, satisfying at least one of the requirements of the use case directly. Such a setup could then theoretically run in an industrial setup using machines equipped only with a single camera and possibly an Inertial Measurement Unit (IMU), which are both common in drones.

1.3 CENTRAL HYPOTHESES

This overall description of the tasks at hand motivates the central hypotheses of this thesis. Namely:

- Given a sufficiently good visual mapping, the drone will consistently replicate a specified flight path without drifting so much that the task may no longer be carried out. For this we expect at most an approximate average error of about the length of the device used, in our case about 20 cm.
- Given inertial data from cheap embedded hardware, it is possible to filter and stabilize the inertial data enough to utilize a visual-inertial technique in the above solution.
- Given a setting in which the inertial and visual data comes from two different sources, rather than a single integrated device, a visual-inertial solution using clock-synchronization may be constructed.

As indicated by the last point, we also investigate the possibilities of the visual-inertial solution acting in a distributed context, i.e. the inertial data, and the visual data coming from two completely different devices. Of course, with synchronization being a major part of these sorts of solutions, it is expected for there to be challenges with this, but by applying sufficient software-based synchronization it will still provide usable results.
Note that the idea of making the path more "efficient" pertains to the idea of the overall path being shorter without skipping any points of interest and without putting the drone in unnecessary danger. The central idea essentially being that the drone can perform more patrols quicker. Note that a big part of this thesis is the investigation of whether these hypotheses can be fulfilled given different setups, such as incorporating inertial data either into the localization or as a separate entity.

1.4 PROJECT OBJECTIVES

The primary objective is to base the system's localization on a robust, visual SLAM solution, for which it is possible to employ as much of the method for potential improvements. For this, we investigate the possibilities with regards to some of the existing problems visual-only solutions bring, for example, scale-estimation. Additionally, we investigate the options of incorporating inertial data into the system, with the limitations of a cheap hardware setup. The chosen solution estimates the camera position and attempts to create a mapping of the area around it based on identified features. The localization is fed

into a preprocessing of the path, and then further fed into a navigation system, which guides the drone along a prespecified trajectory. The mapping from the SLAM algorithm is continuously reinforced and changed as the process continues, and by employing a keyframe-based path specification, even the path should be enhanced as the map improves. Given the context of commodity hardware, this setup is likely to have multiple sources of error. These include, as mentioned earlier, the existence of sensor bias and noise, potential communication issues of supplying a video stream at a high enough rate, possible drifts in the mapping created by the SLAM algorithm, etc. Of course, these all need to be addressed to a certain degree.

Following the implementation of this system, it is tested thoroughly by qualitative tests, and a subsequent physical test to confirm the validity of the system in real life. The objectives of these tests and experiments are to investigate:

- Whether navigation is possible
- The error of the navigation
- The effectiveness of individual elements such as trajectory tracking or the scale estimation
- The possibilities regarding utilization of inertial data, either in an embedded or distributed context.

In section 4.4.2.2, it proves highly successful for a visual-only solution, however, the inertial solution, while getting close to viable, still presents issues which prove the navigation too dangerous to test at the stage discovered. While the conclusion for the distributed version of this solution also proves too dangerous to run physically, the results of the synchronization and careful calibration prove exceptionally close to usable despite existing precautions concerning meticulous calibration of the sensor and camera. All of these results are finally evaluated and considered under the scrutiny of both the qualitative and quantitative tests.

1.5 THESIS STRUCTURE

In Chapter 2 we investigate the existing research in the field of robotics and autonomous drones while digging a little deeper into the specifics of computer vision, control theory, sensor data, and synchronization. In addition, we provide an overview of related works with regards to SLAM algorithms, the internal representations we made use of, and relevant algorithms. In Chapter 3, we explain how all of these make for interesting use in this thesis and the actual final design. We identify issues which require addressing, and where things may prove challenging. Said design is described in detail in Chapter 4. Finally,

we design a solution for the case and implement a physical prototype. These implementations are tested, and experiments are conducted. The experimental results derived from this, are those from which we draw our conclusions and provide perspective for potential future work in Chapter 6. Chapter 5 briefly reflects on a couple of works in which much effort was put into, but which did not prove useful for the final solution of the problem.

2 RELATED WORK

The range of fields that can be covered with regards to the desired design/implementation is far and wide, depending on the final solution. As is such, firstly we will cover some of the existing discoveries that relate to this task, and how they may or may not prove useful for this particular case. This includes literature specifically relating to the completion of the entire task, literature on the general subject of drone research and related literature, such as literature relating to the methods used, for example, computer vision research. In this chapter, we summarize these areas of the literature, their methods, their advantages, their disadvantages. We will cover the field in the most straightforward fashion, by gradually increasing the specificity of the contents, starting with the general field of drone research, then by covering the relevant parts of control theory, computer vision, sensor data, and synchronization. By the end, we will have looked at all that needs to be considered for the remaining analysis and the choices of which will be made in the next chapter.

2.1 PREVIOUS DRONE RESEARCH

To obtain a general sense of the given field, investigating existing research, specifically with regards to drones and automated navigation of them, is a must. This is both in regards to how one expects to see with the device and how one expects to control them. In this section, we explain some of this research, as it will later become clear that knowledge of past endeavours will prove useful when considering solutions for this particular task. This investigation will also describe the logic behind the eventual choices of technology, and overcoming of possible problems, which will be discussed in chapter 3.

The field of research which specifically pertains to drones is a relatively young one. With most active research seemingly beginning around the mid 00's with the development of proper quad-rotors and proper control algorithms, according to [28].[1]. As one would expect, a plethora of specific research has been carried out during the time since then.

1 For the sake of relevance, we will focus on research that either does not concern itself with the type of device, or works with the quad-rotor style drone, as these are by far the most common devices on the market.

2.1.1 *Localization*

The field of navigation and localization is in no particular way specific to drones. In fact, it is far older. However, there is a vast limitation in relative to usefulness present when it comes to the localization of drones in particular, at least in terms of the execution of the task at hand. Due to the nature of the task, precision is essential, as an error margin of a few meters would fail to perform the task. Additionally, solutions involving heavy or unwieldy extra devices could affect the lifting capacity or the balance in commodity drones. As such, we shall attempt to focus on the findings which have in fact shown to be useful on drones or which have the potential to be as such.

In general, the phase of localizing a drone in an automation system concerns itself with solving the SLAM problem. As the abbreviation exhibits, this is the problem of determining a camera's location, based on the data collected, while simultaneously creating an internal representation of what the environment looks like. The problem originates from the cyclical problem of one needing to know one's surroundings to appropriately localize oneself based on sensor-data, but creating such a map requires one to know the location from which the sensor data comes from.

Performing SLAM to solve a problem such as the one in question is a natural choice. Most modern solutions concern themselves with relatively accurate localization by having an internal representation which can be useful when one requires the device to be "aware" of its environment.

While the problem may sound paradoxical, it has been solved in various ways with many different sensors over the last several years. Many of these sensors have been employed to achieve some degree of autonomy on drones.

Much like the more common research in the field of localization, a lot of findings pertain specifically to outdoor localization and makes use of a fusion of Global Positioning System (GPS)/GLObal NAvigation Satellite System (GLONASS) and onboard sensors to enable accurate positioning. An example of this includes [24] in which such navigation is implemented on an AR Drone (a relatively common commodity drone) to obtain an accuracy of $\pm 0.5\text{m}$ at hovering state. However, the article presents no accuracy measurements for the drone while in movement.

In general, outdoor drone solutions come with their restrictions. The strongest one being restricted to solutions which depend on using outdoor environmental data, as precision is nowhere near good enough. The idea is also very conflicting with the use case, initially motivating this thesis. Additionally, half a meter of accuracy is not quite sufficient for indoor use. Luckily, there is also substantial drone research explicitly concerned with indoor localization. In general, these

can be segmented into solutions that make use of a Laser Imaging Detection And Ranging (LIDAR) sensor, and solutions that concern themselves with the usage of computer vision (sometimes with additional sensors).

A LIDAR based solution is one in which laser-based sensors provide an exceptionally precise set of data, for which a plethora of useful SLAM techniques exist. LIDAR based solutions are by far the most mature of solutions, as they ensure quite a high level of accuracy. Examples of this include [2] in which a high level of accuracy is obtained from a drone by merging a visual odometry with laser sensor data.

The computer vision based solutions employ a very popular solution to the SLAM problem, which includes the usage of a camera sensor. Examples of visual SLAM solutions may be seen in [29]. As can be seen, there are many solutions on the market, but certain ones are particularly well fit for the drone problem. For example, [23] used monocular visual SLAM to navigate areas presumed to be feature-rich, and [1] performs navigation in unknown areas by merging visual and inertial data into one vision-based solution. Further details with regards to the possibilities of visual and visual-inertial SLAM will be discussed in section 2.3.

As can be seen in the respective articles, the amount of resources necessary to make these methods work varies greatly, which will also play a part due to the commodity limitation of this project.

2.1.2 *Scale Estimation*

As briefly mentioned in the last section, and as will be detailed in section 2.3, the application of a monocular SLAM method brings with it its limitations in accuracy. One of these constraints includes the very problem of scale. The problem occurs when the monocular SLAM method generates a map of the environment, estimating the movement of the camera between frames. This estimation is good, but the absolute scale of the map is substantially harder to estimate as it requires some information about the environment which goes beyond the camera's view. The most straight-forward solution to this problem is the solution of adding more cameras, or by using RGBD cameras instead. Of course, this undercuts the definition of the problem being applicable for monocular solutions. Other solutions to the problem have been proposed in the past. The most common one is to implement an extra sensor into the system which can provide an additional sense. This is most often a laser-based sensor such as LIDAR or inertial from an IMU. The IMU is often a part of a visual-inertial solution (see section 2.3.3), or as standalone scale-estimation modules which use the data to determine the absolute metric scale of the camera movements. An example of the use of IMU to scale-estimate can be seen in

[19] where the simple inclusion of a 3D accelerometer enables a live scaling, close to metric scale.

Research which exclusively makes use of the camera while still managing to produce a scale estimation does also exist, like in [26] wherein they use a method called Depth from Defocus (DfD) which relies on the blurring of the image, eventually enabling a metric scale estimate through camera movement. This research is, however, still ongoing, and with it its future work.

Only a few works are available which directly employ the fact that a drone is observing the environment, but a few do exist, such as [11] in which a laser range finder is used for scale estimation, or [15] where a scale estimation is obtained by adapting gain and observing self-caused drone oscillations to estimate a metric scale. The latter is more interesting, as it requires no extra sensors implemented to the system. The procedure consists of applying control to the Z-axis of the drone, adapting the gain based on the oscillations caused by the control signals.

2.1.3 *Drone Navigation*

In terms of the field of navigation, it is hard to avoid control theory[2], since it is the core of how the system handles the input and produces output. In relation to drones, it is used in almost every drone to handle hovering and controlling the magnitude of the propellers to get a specified altitude. In general, it is used to stabilize a system variable at a certain value. This is not the first time a controller is used to solve a path-following trajectory problem. In fact, [12] did a good job comparing the accuracy of two different control methods, a proportional–derivative controller and an Model predictive controller (MPC). How these controllers work will be further described in section Section 2.2. The result presented in [12] shows a more stable controlling using an MPC rather than a PD controller. The main factor here is the overshooting of the PD controller, which is applying to much force in the desired direction. This is already compensated in the MPC, since a model of the drone is used to optimize the next N moves into the future. While [12] shows an improvement using MPC over a PD controller, it is hard to relate the result to any new control methods since the details of which unit and which value the error is computed on is nowhere to be found.

2.2 THEORY OF THE ENGINEERING OF NAVIGATION CONTROLLERS

Control theory is a sub-branch of a bigger topic called "Guidance, Navigation and Control" (GNC), where control theory deals mainly

2 Note that the term "Control Theory" is a highly overloaded one, and that in this particular report, it refers to the theory and engineering of feedback controllers

with the Navigation and Control aspect of the GNC system. As described in [7], the terms of GNC should be understood as follows:

- Guidance refers to finding the desired trajectory from a vehicle's current location to a designated goal.
- Navigation is a field of study that focuses on the process of monitoring and controlling the movement of a craft or vehicle from one place to another.
- Control refers to the manipulation of the forces of the vehicle to execute a specific command. An example could be moving the vehicle forward which translates to applying force on certain motors.

In control theory, navigation and control are submerged into one, where the navigation aspect of establishing the current state and where to go next is handled while control actions to match the current state with the next are computed. In the previous section, we shortly touched upon two types of controllers commonly used in drone-related scenarios. As these particular controllers will play a essential role in the final design/implementation, a brief description of how these controllers work in general will be presented in this section.

2.2.1 *Proportional–Integral–Derivative Control*

A PID controller is, in its essence, a control loop feedback mechanism used to continuously compute the error $e(t) = r(t) - y(t)$ between a desired setpoint $r(t)$ and a measured process variable $y(t)$ and applies a correction based on proportional, integral, and derivative terms. In our case, the error is the distance between the desired coordinate on the trajectory and the actual coordinate of the drone. The working principle behind a PID controller is that the proportional, integral and derivative terms must be individually adjusted or "tuned". Here is how each portion affects the output results.

- Proportional tuning involves correcting a target proportional to the difference. Thus, the target value is never achieved because as the difference approaches zero, the applied correction does so too. So in our case, the drone would never hit the target coordinates.
- Integral tuning attempts to correct this by effectively accumulating the error result from the "P" action to increase the correction factor. Hereby, the function of "I" is to increase the output result to hit its desired value. However, a side effect of giving "I" too much influence on the output, will result in a case where when the target is reached, "I" attempts to drive the cumulative error to zero, resulting in an overshoot. We can write "I" as $\int_0^t e(t')dt'$.

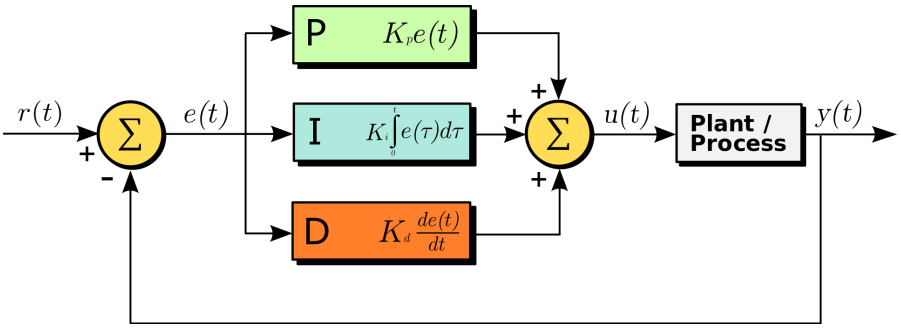


Figure 2: A block diagram of a PID controller in a feedback loop.

- Derivative tuning attempts to minimize this overshoot by slowing the correction factor applied as the target is approached. We can express "D" as $\frac{de(t)}{d(t)}$

Then we define tuning the system, by adding a weight to each of these components, respectively K_P, K_I, K_D. This results in the output equation of the PID controller to be

$$u(t) = K_P e(t) + K_I \int_0^t e(t')dt' + K_D \frac{de(t)}{dt}.$$

A block diagram of the PID system can be observed in figure 2. In [12] they created a PD, meaning they set the weight $K_I = 0$ leaving only "P" and "D" to influence the output. A more detailed description of PID can be found in [3]. As a result of [12], they proved better accuracy using an MPC approach over the PD controller. In the next section, we will consider the MPC, how it functions and why it might be better.

2.2.2 *Model predictive Control*

An MPC is an advanced control technique for difficult multivariable control problems. Many alterations of the concept MPC exist. Presented in [25] there is a detailed overview of a few of these, whereas in this section, we will only describe the general concept. The MPC model consists of a prediction model, an optimizer and three kinds of variables; input, output and disturbance. The disturbance variables are input variables that influence the system outputs, but cannot be adjusted by the control system. A block diagram of a general MPC is presented in figure 3.

Suppose that we wish to control a system with multiple input/output variables, while still satisfying inequality constraints on these variables. If a reasonable accurate dynamic model of the system is available, a prediction of future output values can be computed by considering the input-output relationships represented by the model. If we can compute the output variable given a set of input values, an optimization of the input variable can be produced to get the desired

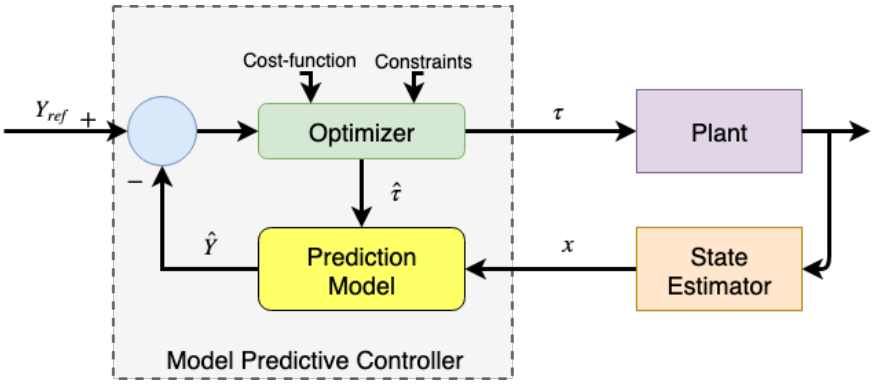


Figure 3: A block diagram of a Model Predictive Controller. Outside the Model Predictive Controller (on the right), we have a plant which is the object which we are manipulating, where as in our case this is the drone. The state Estimator is the actual result of performing the command τ on the plant, where the prediction model is updated with the new state of the plant, x.

output. This process is a recursive process where the output of the optimization is feedback into the prediction model for the next iteration of the optimization. This process is completed N times, where only the first prediction is fed into the plant for execution. The variable N varies depending on the type of plant that is used in the system since N is the number of control steps we want to predict into the future. Using an MPC model creates several advantages, which is why it is used so much:

- The model will capture the dynamic and static interaction between input, output and disturbance variables.
- Soft and hard constraints on inputs and outputs are considered in a systematic manner.
- The control calculations can be coordinated with the calculation of optimum set points.
- Accurate model predictions can provide early warnings of potential problems.

One thing to keep in mind is the success of MPC (or any other model-based approach) heavily depending on the accuracy of the process model. Inaccurate predictions can make matters worse, instead of better [25]. The function of the state estimator in figure 3 is to find the actual outcome of performing the command τ on the plant. The state estimator then updates the prediction model to match the true state of the plant. In the next section, we are diving into how we can use SLAM as our state estimator to find the actual location of the drone after a control command is performed.

2.3 COMPUTER VISION

Computer vision is defined as a field of study that seeks to help computers "see" and understand the content of their surroundings using a camera or sensor data [5]. Computer vision is a broad field of complexities from advanced artificial intelligence to simple computable feature detection. In this section, we will focus on an old well-known problem of SLAM.

2.3.1 *What is visual SLAM?*

SLAM is the art of simultaneously finding the localization of the device while mapping the surroundings into a localized map. This can be done by utilizing a camera and several sensors to improve accuracy. In visual SLAM only the visual inputs are used to perform location and mapping, meaning that the only sensor required is a camera. The visual SLAM algorithms are designed to take advantage of the vast amount of information from the world available from image data. The way that the SLAM systems use these data can be classified as sparse/dense and direct/indirect, where the former describes the quantity of regions used in each received image frame. The latter describes different ways in which the image data are used. This means there are four possible types of SLAM system, in terms of how the image data are used [8][14].

2.3.1.1 *Direct and indirect methods*

The way a SLAM system uses the information from a received image can be used to classify it as either a direct or indirect method.
The indirect method is processed in two steps. Firstly, the raw image data is processed to extract and match features in the image. Extracting and matching features can be done in many different ways, but the overall idea is to keep the various features unique and rotation invariant. By extracting the features and building an intermediate representation, one solves part of the overall problem, such as establishing correspondence. Secondly, the intermediate values are interpreted as noisy measurements Y in a probabilistic model to estimate geometry and camera motion. This is typically done by a Maximum Likelihood approach, which finds the geometry and camera motion that maximizes the probability of a match:

$$X^* := \text{argmax}_X P(Y|X)$$

The indirect method seeks to optimize the geometric error since the computed values of the map are either point-position or flow-vectors [8].
The Direct method skips the pre-processing step and uses the actual

pixel intensities by measuring light received from certain directions over a certain period of time. Instead of optimizing the Geometric error, which we do not have without the pre-process, it is optimizing the photometric error [8]. In figure 4, a visual representation of the differences in direct and indirect (feature-based) is presented.

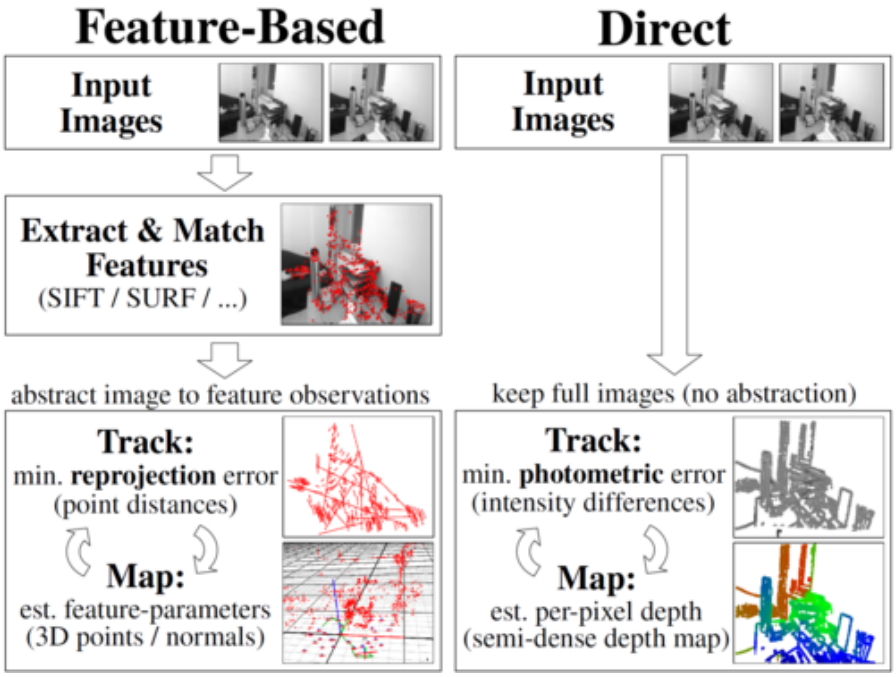


Figure 4: Direct vs indirect approach[13].

Since feature extraction can take a lot of time, using the direct methods may allow more time for other calculations while maintaining the same frame rate as indirect methods. On the other hand, indirect feature-based methods provide better tolerance for changing light conditions, which unlike direct methods, do not directly use pixel intensities.

2.3.1.2 *Sparse and Dense Methods*

How the image is utilised in a visual SLAM system can most commonly be classified as either sparse or dense. *A sparse system* makes use of a small selected subset of the pixels in each of the images, typically in the shape of computationally repeatable features, of which there are many kinds, or as something more novel like detection of edges. *A dense system*, however, makes use of most if not all of the pixels present in an image, often making use of the photometric properties of the camera (i.e. knowledge of things like exposure and vignette). As they use a different number of pixels and regions in a given area, the generated maps from sparse and dense methods are very different. The maps generated from sparse methods are point clouds, which are a coarse representation of the scene and mainly

used to track the camera pose (localisation). On the other hand, dense maps provide much more detail of viewed scenes; but because they use much more pixels than sparse methods, more powerful hardware is usually needed, and most current dense SLAM systems require a GPU[14].

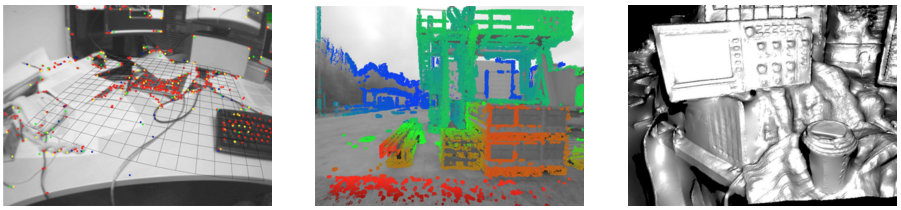


Figure 5: On the left, we see a sparse mapping created by PTAM, where the coloured points are the extracted features. In the middle, a semi-dense mapping created by LSD-SLAM, where the coloured points are the high gradient areas. Last, on the right, a dense mapping by DTAM, where all the points on the surface are part of the map[14].

In the next two subsections, Section 2.3.2 and Section 2.3.3, we are going to look at an indirect-sparse visual SLAM method called ORBSLAM and an indirect-sparse visual inertial SLAM method called VINS-mono.

2.3.2 *ORBSLAM*

In 2015, Raul Mur-Artal, Juan D. Tardos, J. M. M. Montiel and Dorian Galvez-Lopez invented monocular ORBSLAM, and in 2016 they extended it to work with stereo and RGBD [18]. In this subsection, we are going to take a very brief look at the monocular ORBSLAM.

Essentially ORBSLAM is built on three proccesses running parallel (see figure 6):

- Tracking
- Local Mapping
- Loop closing

The first of these processes is tracking in which the main functionality is to localize the camera within every frame by finding feature matches within the local map and minimizing the reprojection error by applying motion-only bundle adjustments. To explain motion only bundle adjustments, one has to understand bundle adjustments first. Bundle adjustments are described well in [31] as a problem of refining a visual reconstruction to produce jointly optimal 3D structures (map of features) and viewing parameter estimates (camera pose and calibration). The parameter estimates are found by minimizing some

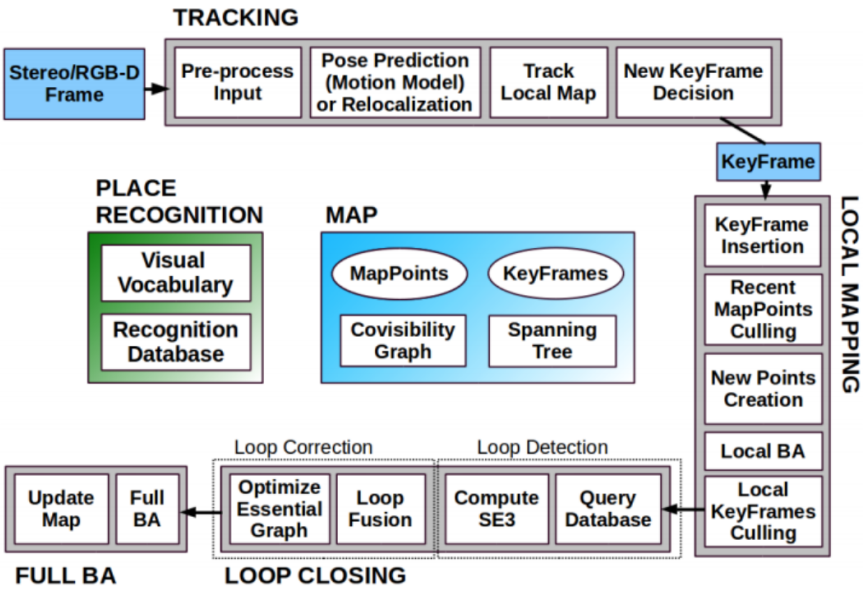


Figure 6: The workflow of ORBSLAM2. As ORBSLAM2 was an extension to the original utilizing stereo and RGBD cameras, this picture describes the frame as a stereo or RGBD frame input, but a single frame would work as well.

cost function using the Levenberg-Marquardt algorithm to find a local minimum that quantifies the model fitting error. The solution is simultaneously optimal with respect to both map structure and camera variations. The name, bundle adjustment, is referring to the bundle of light rays leaving each 3D feature and converging on each camera centre, which are 'adjusted' optimally with respect to both feature and camera positions. A motion-only bundle adjustment is when we are only optimizing the motion of the camera, leaving the feature map unchanged.

The features are found using the ORB feature detection and are used for tracking, mapping and place recognition tasks. The use of this feature detector comes with the benefits of being very fast and robust in rotation and scale. For further information about the ORB feature detector see [20]. After feature detection, a test on the resulting feature point is done to show if the frame can be categorized as a keyframe. The test is done to help us not to process all of the frames received. Doing so, we are leaving the tracking thread to sort out the most important frames in which we are passing on to the next step of the system. These important frames are called keyframes and are determined by maximizing the time between two frames without loosing too many feature correspondences in the map.

The next process is the local mapping process, where the local map is managed, optimized and adjusted using local bundle adjustments. Local bundle adjustment is done instead of looking at the whole map of feature points, only optimizing the last N keyframes with respect to

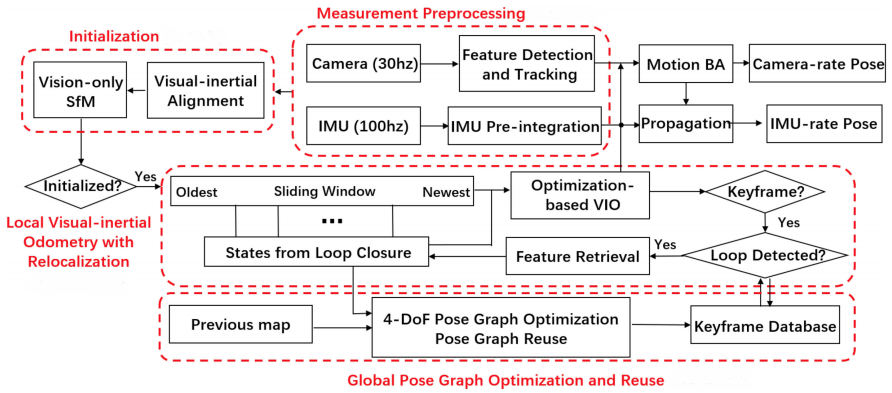


Figure 7: An overview of the VINS system, split into the four processes.

each other. In this processing step, all is managed by only processing the keyframe found in the tracking process and updating the current local map.

Lastly, the loop closing process is used to detect large loops and correct the accumulated drift by performing a pose- graph optimization. This thread launches the fourth thread to perform full bundle adjustment after the pose-graph optimization, to compute the optimal structure and motion solution.

In the next section, a Visual inertial SLAM technique is presented, with the focus on how the inertial data improve the existing system.

2.3.3 *Visual Inertial SLAM*

One of the known disadvantages of vision-only systems is that they are incapable of recovering the metric scale. There are a few ways of solving this problem by applying data-fusion, but recently, we have seen a growing trend of assisting the monocular vision system with an IMU. Presented in [6], is a comparison of existing monocular visual inertial techniques. The results show that VINS is the most accurate SLAM technique on the market, but comes with a cost of processing power. The focus in this section will be to explain how [21] combines the IMU with visual SLAM to improve the accuracy.

One of the primary advantages of using visual-inertial system (VINS) is the observing of the metric scale, as well as roll pitch angles. Observing these parameters enables the navigation tasks that require real size metric state estimations. Besides, the integration of IMU measurements can dramatically improve the motion-tracking performance by connecting the gap between losses of visual tracks due to illumination change, textureless areas, or motion blur [21]. The VINS SLAM system can be split into four processes, as shown in figure 7. The four processes are:

- Measurement processing

- Initialization
- Local Visual-inertial odometry with relocalization
- Global Pose Graph Optimization and Reuse

The first stage of the system is the measurement processing, in which features are extracted and tracked, and IMU measurements between two consecutive frames are preintegrated. The measurements collected are the accelerations from the accelerometer, and the orientation and angular velocity from the gyroscope. Since the camera is running 30 Hz and the IMU is at 100 Hz, a preintegration is performed to ensure bias correction and noise handling — for more information on how this is accomplished read [21].

The next stage in the system is the initialization. In this stage, the bias of the gyroscope is calibrated and the preintegration values of the first process are updated. Additionally, initializing velocity, gravity Vector, and metric scale to compute the correct scaling of the state estimation, are used in later processes. When retrieving data at this stage, an online alignment of the visual and IMU data is performed to find the time difference between the same action recorded in camera and IMU. This is an online process, since it depends on the individual recording modules.

The Local Visual-inertial odometry with relocalization stage, is the stage where all the data gathered and processed is used to update the map and finetune the position based on the visual and inertial data combined. They introduce a sliding window technique, which has the same function as local bundle adjustment, which is to bound the size of data which is being optimized on. The inertial data improves the accuracy by adding value to the optimization stage in Visual-inertial odometry based optimization. This stage is equal to the "Local Mapping" and the loop detection aspect of the "Loop closing" process in ORBSLAM.

The last stage is the global pose graph optimization and reuse. This stage is equivalent to the loop correction part of the "Loop closing" and the full bundle adjustment aspect of the ORBSLAM. The last two stages are performed in different ways compared to ORBSLAM, but the overall functionality of these are the same.

While the potential presented by inertial solutions is undoubtedly higher, this does not necessarily make it exclusively superior to a vision-only solution. The reliance on inertial data, while excellent for high-grade IMUs, any potential noise may cause deviations whose importance depend on the use. Inertial solutions are also somewhat less mature than the visual-only solution. Even so, their improved accuracy is undeniable and certainly worth further investigating.

2.4 FILTERING OF INERTIAL DATA

An inertial SLAM solution such as VINS described in section 2.3.3 relies upon data provided by an extra sensor, most commonly an IMU with angular and linear acceleration data of some sort. As the inertial data provides quite a lot of information about the eventual localization and mapping, it is expected for this data to be as consistent as possible. This is particularly important for this specific task, as there is a focus on affordable commodity hardware, which has a high risk of being particularly susceptible to different types of noise. With this in mind, it becomes worth looking into ways of preprocessing the sensor data to make it as usable as possible. The most common inertial measurement model is one in which one needs to be aware of two main noise factors when performing this preprocessing. These are *random noise* (or *white noise*) causing sudden random fluctuations to the sensor data. These do not reflect reality, and *sensor bias*, in which error slowly accumulates to the sensor output, causing a sort of "drifting away" if given enough time. For inertial sensors, this model is usually applied in such that these noise factors can exist individually for each axis on the measurements (see examples [4], [30]). In [21], the article originally describing VINS, one will also see that they also apply this model, and attempt to counteract noise by preintegrating noise data based on a noise calibration of the sensor. This calibration consists of a *noise density* for random noise, describing the strength of the white noise on the sensor, and a *random walk* variable for sensor bias also modelling the strength of the accumulating bias noise on the sensor. By knowing these variables, one can attempt to counteract the noise. The described variables can be estimated from any inertial sensor by measuring a sufficient amount of data from the stationary sensor and plotting the Allan standard deviation plot of the data as done in [4]. Additional ways of counteracting, especially random noise, are by applying certain filtering techniques to the sensor data before relaying it further. Examples of this include a simple moving window average, low-pass-filters, and merging data from other sensors.

2.5 SYNCHRONIZATION

As described in the motivational introduction, most visual-inertial drone configurations are equipped with hardware-synchronized visual-inertial data. This means that on the actual hardware itself, a trigger is set up straight onto the board, which causes the camera to capture a frame exactly when the proper amount of inertial frames are captured. This extreme precision, while not always a strict requirement, is a consistent consensus across many of the current state-of-the-art visual-inertial SLAM solutions that currently exist. For one part of this project, we look into software synchronization of separate devices to

still obtain usable SLAM data. Two pieces of related work play an essential part in this, one originating from the creators of VINS in [22], in which they develop a method for estimating the exact delay between the two elements. They do this by performing a visual-inertial alignment, jointly optimizing on the time offset between the visual and the inertial data. Another piece of work, which is used for the said experiment, is the usage of Precision Time Protocol (PTP) to perform synchronization between devices in the system on the given network. In particular, the implementation used is that of *chrony* ([16]) which is an implementation of PTP (and Network Time Protocol (NTP)), which has been tested to obtain synchronization delays all the way down to a scale of microseconds.

In this chapter, we gave a brief overview of existing works in the field of automation and drone navigation have been detailed. Many parts of the field are still quite young and growing rapidly so there are several potential solutions but we could not go through them all. As a result, we narrowed it down to what we think are the most essential and promising findings for this project. In the next chapter, a thorough analysis of existing work will be presented, determining which of these findings would make for a suitable solution in our use case and in which areas these findings might be in need of improvement.

3 CHOOSING METHODS FOR THE PROJECT

In the previous chapter, we presented some of the literature which is important to take into consideration when it comes to generally performing automation, but particularly with regards to solving the posed problems and investigating the hypotheses. Having described all of this, the next step is the analyze of the literature and the chooses of technologies used in the final solution/prototype. As part of this, certain uncertainties will arise and be considered in order to improve upon existing solutions. These uncertainties include how to establish ground-truth for trajectories, how to perform scale estimation for monocular SLAM solutions, and how exactly to perform the actual control. All of these were considered while also keeping the commodity restriction in mind. This chapter describes all of the above so we, in the following chapter, may describe the final design, how it was implemented, and the subsequent experiments.

In one way or another, all of these challenges must be addressed in order to obtain a sufficient, usable, automation using a vision based solution.

3.1 LOCALIZING THE DRONE

As the original use case directly describes the operation in a closed off indoor environment, most of the literature pertaining to outdoor localization is less than useful for this implementation. This is due to the fact that it would be taking place indoors in an, almost certainly, GPS-denied environment. Even if this was not the case, the general error of the literature which made use of it, even with advanced fusion techniques, was still somewhere in the area of 0.5 meters, as mentioned in chapter 2. This is not sufficient, when one wishes to obtain an error of the original path which is much smaller than this.

As is such, we look to indoor solutions instead. For these, while it would certainly obtain the greatest accuracy using something like the existing LIDAR solutions, such sensors are commonly quite expensive, and commonly require very expensive drones, going by the discussed literature. As of writing this report, it would also appear that the general concept of applying LIDAR sensors for automation of vehicles has recently been criticized, some even describing the practice as "doomed", citing them as being too expensive while still being too unnecessary[1]. This would very much go against the spirit of the task, which is to accomplish good automation with affordable com-

1 https://techcrunch.com/2019/04/22/anyone-relying-on-lidar-is-doomed-elon-musk-says

modity hardware. With these observations, it seems clear that the solution would either require a way to enhance and enable affordable commodity LIDAR sensors or that the solution is likely to be based on camera-based computer vision.

While there are many different ways of performing visual SLAM, as described in chapter 2, we will focus on two primary versions which have shown, to a certain degree, levels of success in the field. Each method has its challenges in this particular context. These include the execution of a purely visual solution and based on the achieved results of that: possibly a visual-inertial solution.

3.1.1 *A Visual-Only Solution*

With a purely visual solution, a lot of potential noise from commodity IMUs would be avoided, as the solution would be based entirely on the data received from a camera. As it is near impossible to find an affordable commodity drone with multiple front pointing cameras or RGBD cameras, this restricts the potential to a monocular solution. This does, however, still come with its own set of challenges. The lack of an element outside of the visual solution will result in not having any way of estimating the absolute scale of the scene. Having no absolute scale could lead to errors when performing navigation, due to the potentially dramatic shift in the correlation between the digital representation of the environment and the real environment. This could further result in any sort of tuning of a drone controller being drastically incorrect, leading to even more issues. This particular issue was investigated in [15], and we will further show these suspicions in later chapters. Finally, the sole reliance on the mapping provided from the camera data poses the risk of not knowing what to do when loss of tracking occurs, or if the map is simply not good enough. Several methods alleviating mapping issues were covered in chapter 2 when describing, in particular, ORB-SLAM. The loop closure functionality proves very useful when attempting to combat drift in the positioning of keyframes, so that they are placed at the most accurate location on the map whenever possible. The ability to relocalize efficiently, enables one to obtain a method of recovery when tracking is lost, and the efficiency of the ORB features means that it would be possible to run it on a ground-station without too much extra effort (especially if one is in possession of a GPU). A big issue which required addressing for this combination, was the concept of ground truth, with relation to a continually improving map. If one records a path of estimated poses as the path, there is a risk of this ground truth path becoming decreasingly similar to the actual ground truth path of the actual environment. In the next chapter, we propose a method for solving this issue.

The idea of the visual-only solution would thus be to utilize ORB-SLAM as the sole localization technique but with modifications addressing the above mentioned problems, together with a scale estimation algorithm and a control-scheme.

3.1.1.1 *Scale estimation*

A few different methods for estimating scale were covered in chapter 2. For this particular problem, we did not wish to employ any of the sensor-based solutions, unless applying an inertial solution, as it would be a lot of extra heft to employ in one single part of the process exclusively. This leaves either a drone-specific solution, like the one of [15], or one like the DfD solution of [26]. The latter is still very new and has a fair few open issues which are still being researched. As such, it was deemed a little premature for usage at least in automated drone flight. With these considerations in mind, a drone-specific solution was considered the best way to achieve something relatively stable. For the particular case of [15], we took issue with its very localized method for establishing scale, namely the fact that the entire estimate of the scale is achieved by causing small oscillations in a single area. While the results presented are quite good, one concept which one gets quite familiar with, when using visual SLAM techniques, is the idea of scale drift localizing itself in small areas if it is not rectified properly. This leads to small clusters of points essentially consisting of the map itself repeated as a smaller version of itself. Such a cluster may lead to very brief changes in movement on the map, which is normally mostly harmless, but if one were to cause oscillations right at such a cluster, it could prove problematic. With this in mind, it was decided that a new method would be more relevant to investigate. Most desirably a method which requires movement throughout the map, and which takes plenty of measurements to investigate such issues. Such a method is designed in the next chapter.

3.1.2 *A Visual Inertial Solution*

Using a solution which incorporates both visual data and inertial data, there would no longer be a challenge with scale estimation, as many visual-inertial SLAM solutions use it to more accurately estimate the scale to begin with. A visual-inertial solution is much less prone to lose tracking as a result of sudden changes, or dips in tracking, as the inertial data is available as a fallback. Unfortunately, this benefit does not come without risk, as an estimation based on inertial data requires quite specific data. This is often a lesser problem when applied in grand-scale research projects. This was eluded in chapter 2 with regards to filtering of sensor data, wherein the nature of inertial sensor noise is described. Cheaper sensors pose much stronger noise/bias, and thus, a bigger risk when applied in a project of this

kind. This could pose a possible solution to the commodity issue, by applying proper filtering, and thus removing enough noise to make the sensor data reliable for localization. However, this poses a risk of the filtering technique eliminating too much of the data, making the sensor data useless. The goal of a visual-inertial solution would thus be to perform many of the same steps as the visual-only solution, with exception of scale-estimation. The legibility of such a solution will be investigated in the next chapter.

3.1.2.1 *Distributed Input*

In conjunction with this problem, another thing worth investigating, is the possibility of having the sensor present as its own entire unit in a distributed system. The motivation behind this extra goal is that many commodity drones are often quite closed when it comes to development. This is especially when it comes to the data received in their IMU to such a degree that it is impossible to receive raw sensor-data. Even so, with most visual-inertial SLAM techniques, the inertial data must often come from the same device which provides the visual data to enable proper synchronization. This is frequently stressed when it comes to implementations of inertial solutions which often cause the developers to recommend hardware synchronization, so the timestamps align as much as possible. The "distributed" solution presented here proposes, that it might be possible to have the inertial unit be on its own device, and then applying a software-based synchronization between the two devices to obtain timestamps that line up with a very low margin of error (using PTP one may get all the way to microseconds in misalignment), thus causing proper visual-inertial alignment. The idea is that this, or perhaps a combination of this and the online-temporal calibration of VINS would suffice to solve the problem. This possibility is also investigated in this report.

Both of the inertial possibilities require careful construction, as any source of error may cause the localization to become insufficient for real-time flight. First of all, the actual placement of the sensors with relation to one another needs to be carefully maintained, and may not change throughout the flight, lest the calibration data becomes useless, causing localization to suffer the same fate. This means that one needs not only to perform a careful extrinsics calibration, but also needs to ensure the consistency of such.

3.2 NAVIGATION AND CONTROL

There are many different ways to implement and design controllers. In chapter 2, a few solutions were highlighted, PID and MPC, whereas, in this section, a critical analysis of these controllers in the scenario of our use case is presented.

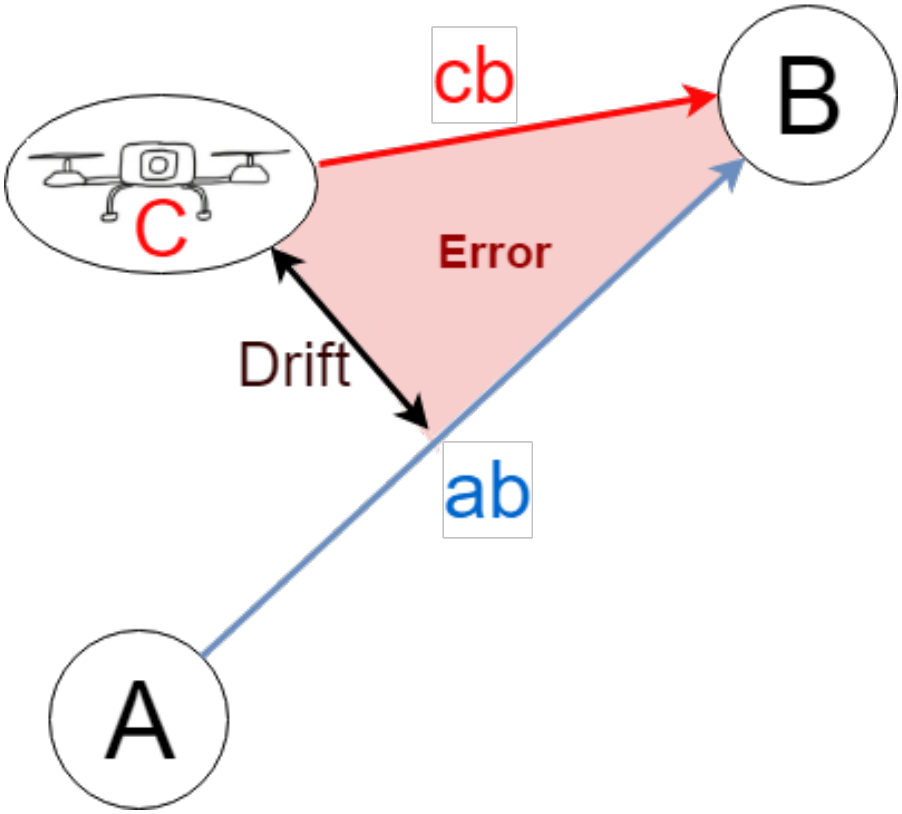


Figure 8: Drone pathing using a PID controller and how drift affects the error.

We start with the usage of the PID controller, which is a controller which focuses on navigating the drone from point A to point B. However, the goal of our posed use case is having the drone follow a specific trajectory, ab, between these to points instead. The consequence of only focusing on getting to point B, instead of following a specific trajectory, is that it makes it vulnerable to drift. Affected by drift, the controller will respond with creating a new trajectory, cb, and follow it, which will resolve in an error which is not accounted for by the controller. This scenario is shown in figure 8.

Another simple or straightforward solution for a controller, is to create a proportional controller with drift correction. The idea is to add the proportional-vector from the drone to point B, with the drift vector from the drone to trajectory ab. This solution will actively account for the drift error and command the drone to get close to the trajectory in each command. A problem with this solution is the risk of potential overshoot. This occurs, when a proportional control without the derivative aspect only takes the distance between the drone and point B into account without looking at the current rate of change in the error. Another unwanted feature in both the proportional controller with drift correction and the PID controller, is the narrow vision

of the path as a whole. Both of these controllers focus on a single step of the entire route. As a result, the drone slows down the closer it gets to the point B, but when it gets there a new goal is given, and the drone has to speed up again. While this does not lead to worse accuracy, it slows down the drone, resulting in more time spent on the route and more battery usage. Depending on the nature of this slowdown, one may argue that it helps to fulfil the goal of "capturing points of interest" in the trajectory, but even so, the risk must be observed.

A controller that accounts for more than a single step on the entire path, is the MPC controller. The ability to predict how fast the drone should move to be able to follow the trajectory perfectly, was what we were looking for in a controller module. One of the problems, as mentioned in chapter 2 section 2.2, is that a very accurate dynamic model needs to be computed before the MPC performs well at all. Typically, one does not have the full dynamic model of a drone available.

As a conclusion of the controller analysis, we chose to take the proportional-controller with drift correction and the PID controller on to further testing. The decision was based on the high level of difficulty of the MPC where a precise dynamic model had to be computed, before resulting in an accurate controller. While it is possible to compute the dynamic model, we deemed it out of scope given the time available. The proportional controller with drift correction and PID -controller makes for a good testing pair since they solve the flaws of each other. Even though a combination of the two controllers is not possible possible, due to the fact that having two PID controllers deciding on the same axis could result in two different command actions, the internals of the PID would start outputting higher numbers in order to overcome each others commands

In this chapter, a thorough analysis of existing work has been completed, whereas the following techniques have been selected to be further proved upon and combined into a unified system. As our localization techniques, we chose two different approaches. The first being the visual-only technique which utilizes only the camera feed to complete a map and localize the drone within it. Here the ORB-SLAM is chosen for its robustness and fast recovery, and a scale estimation technique is developed using drone-specific moves to improve the quality of the map. The other approach was the visual-inertial technique which utilizes both camera and IMU, where the function of the IMU is to strengthen the weakness of the visual-only and to estimate the scale. As for controllers, a proportional controller with drift correction and a PID controller were selected based on the differences in strengths and weaknesses. In the next chapter, we go through the overall design of the system and how a prototype is implemented to showcase the results.

4

DESIGN AND IMPLEMENTATION OF DRONE SYSTEM

In the previous chapter, we looked into the specific tools which were deemed the most useful to solve the given problem. These tools included choices of mapping and localization, controller type, and the problems that may follow from the usage of them, such as scale estimation or ground-truth drift. With all of these considerations in place, the actual system itself was designed and implemented in order to enable the subsequent investigation. This includes the general vision of how the full system would look, but also how to implement the eventual prototype. In this chapter, we contribute with a design for solving the primary task, including solutions to the missing layers, modifications to the individual elements, and control algorithms. Thereafter, we describe the implementation of a prototype which makes use of visual SLAM. Following this, we investigate a version of the system which enables the inclusion of inertial data, and different ways of processing it. One being in a distributed setting with software-based synchronization, the other with the two sensors running in a centralized setting. Additionally, this also includes the execution of the experiments testing these solutions and the results that follow.

4.1 THE DESIGN VISION

In this section, an envisionment of the system as it would look in its fully realized setup is described as a solution to the industrial installation. Note that not all elements of the design vision have been tested for the final prototype, but it will be pointed out whenever this is the case.

The design vision of the entire system can be split into three layers. The lowest layer being the drone flying its route, collecting information from sensors and camera. The main functionality of the drone is to collect the data and send it to the next layer of the system and wait for a corresponding response. The information collected would be the monocular camera feed from the front of the drone and depending on which solution visioned, data from an IMU would be collected as well.

The second layer is the ground station, which is a powerful stationary device with the purpose of processing the information collected to provide control feedback to the drones. A ground station can have multiple drones connected to it, where the data collected is used to

strengthen a single map and locate the position of each drone within. The architecture of the system, on the ground station, consists of five modules; Drone API, Location and Mapping, Scale estimation, Path planner and lastly a controller module. The drone API module is a transmission layer between the drone and the ground station. The core functionality of the module is to route drone data into the correct modules and when receiving a move command from the controller module, to convert it from a vector into the speed of each motor on the drone. The Location and Mapping module will be receiving the camera and sensor data from the drone API module. Depending on which solution chosen; visual-only or visual-inertial, the module will be implemented differently since the visual-only solution does not use the IMU data. Given the data, the module will achieve a mapping of the environment surrounding the drone which gave the data, and the location of this drone is found within this map. Besides, this module records every movement of the drone, both under construction of a route using keyframe-based paths, but also under patrolling. These information is sent to both the third layer and the next module in this system. By recording only the keyframe-based path, a lot of noise generated by the drone is filtered and will not be recreated under the autonomous flight, more of this will be described under the implementation of the module in section 4.2.2. Another positive feature is when the map updates from a bundle adjustment or a loop closure, the recorded path automatically updates as well. The Scale estimation module will work differently depending on the solution. In the visual-only solution, the module will be responsible for scaling the map given from the location and mapping module consistently, since monocular SLAM methods have problems with inconsistent scaling. For the visual-inertial solution, the scaling will be based on the IMU data and will be handled in this module or in the location and mapping module. The functionality of the path planning module is to receiving the position of a drone, handling which segment of the path is currently being processed, and sending the segment to the controller. A segment consists of a start coordinate, and a goal coordinate, wherein between these coordinates a trajectory is computed. This is the trajectory which the controller module should perform. The last module is the controller module. It receives a position from the location and mapping module and a segment from the path planner module. The information is processed into a vector in which it determines which direction the drone should move next in order to minimize the error between it and the planned trajectory. The magnitude of the vector correlates with how fast the drone should move in the corresponding direction.

The last and third layer is a cloud-based interface for visualizing information for users and providing the result of each patrolling drone. The cloud-based interface will be receiving information from multi-

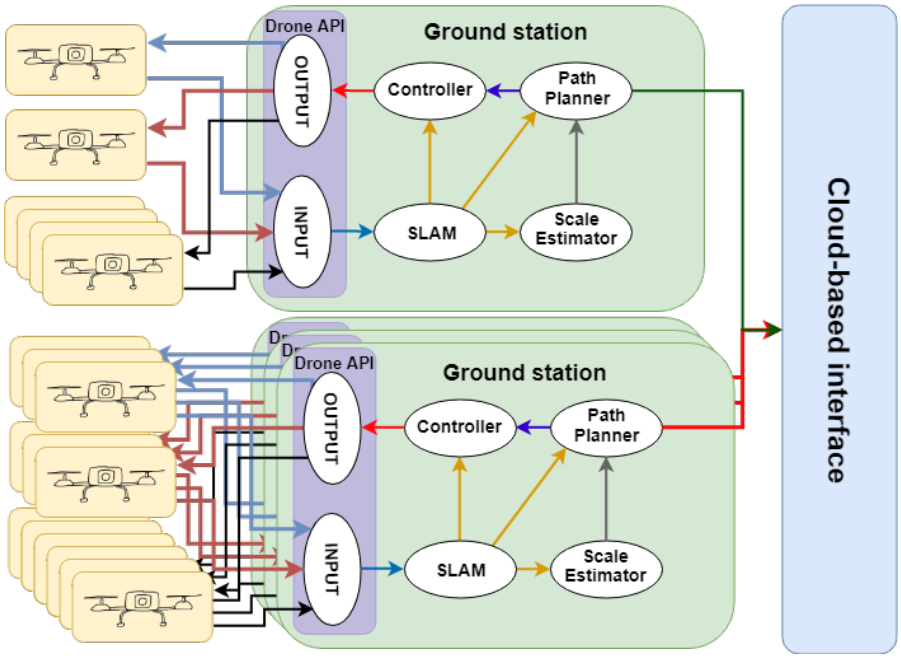


Figure 9: A schematic view of the full design vision imagined to solve the primary task.

ple ground stations, where a single map is formed and shown to the user. The information given is the camera feed from each drone, and the position computed. Here, customized data will also be shown if any is needed. The intricacies of the cloud-based interface are determined on a more individual basis, i.e. the specific metrics provided will depend highly on the specific use case. For example, for the use case given in chapter 1, there could be instances of dropped parcels or other kinds of analytics pertaining to the loop sorter. The only consistent layer would be, that one can process information about drone flight and camera data.

The entire system is described as a diagram in figure 9, where each layer are shown as outer boxes and the modules in the layer is shown within.

For this report, the elements which will be focused on will be the drone input/output itself, and the specifics of the ground station. Any elements pertaining to the cloud-based interface are left as a part of the full design vision. A visualization of these elements may be seen in figure 10. As a prototype, we have implemented a single ground station that operates over a single drone.

With the overall design vision in place, a prototype emerges. The elements of the design vision which are being investigated is primarily the Drone API and the ground-station, such that the hypotheses may be tested.

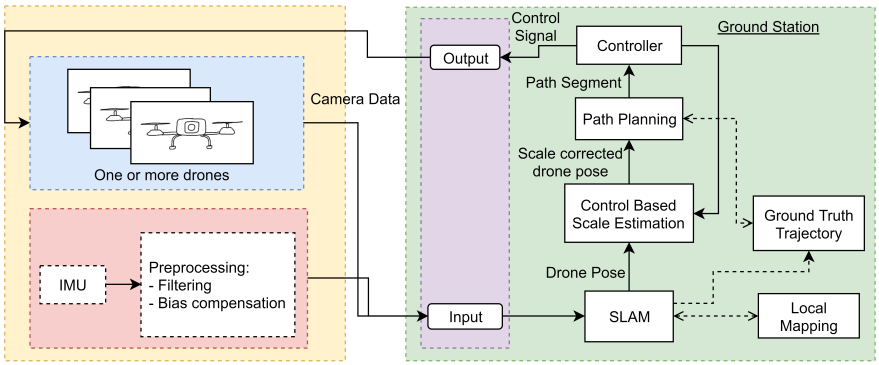


Figure 10: The architecture which is focused on in this implementation. The colored background indicates separate machines, depending on the choice of solution. The yellow background signifies a singular machine for the non-distributed inertial solution, and the separate blue and red signify devices on the distributed solution. Of course, the visual only solution does not include anything from the red section.

4.2 ARCHITECTURE OF THE SYSTEM

The system itself is designed to be as modular as possible, i.e. it should be constructed in such a fashion that it is possible to, for example, make use of another SLAM method, to filter inertial data in a different fashion, or perhaps to adapt the navigational system to a different drone control system, with as little trouble as possible. With this in mind, the entire system is formulated as a set of "nodes", which communicate certain messages using a messaging system. This messaging system chosen was Robot Operating System (ROS)[1], as it enables the exact desired modular structure in the form of a graph-based architecture wherein individual processes are implemented as *nodes* with specific message types being defined as ways for nodes to interface with the node. ROS has been tried and tested on several occasions as being a robust system for implementing systems in robotics, and it even comes with capabilities to create simulations and methods to interface with a system using common JSON based communication (should one wish to interact over a more common web-based interface). With an implementation following this, one could theoretically use any piece of technology desired as long as they can fit the message types defined for the individual nodes. An illustration of the node architecture can be seen in appendix C.

In an attempt to follow the flow of data throughout the system, we shall explain the individual elements in the order they are expected to function in the system, starting at the input to the system, and

1 Note that while the name may indicate otherwise, ROS is actually a collection middleware and frameworks for robot software development, rather than an actual operating system

ending at the output to the device. Throughout this explanation we will ensure to specify exactly how the different elements play into the concrete prototype, i.e. which parts were not concretely implemented, and which were. With this, we may then proceed straight into testing/experiments following the explanation.

4.2.1 *Input Data*

Depending on which solution is employed, the explanation of how the system receives its input may vary somewhat. By figure 10, one may see where the input comes from depending on which kind of system is in use. For the visual-only solution, the input data comes solely from the drone's camera, or in a fully realized system, multiple drone cameras (indicated by a blue area). For the visual-inertial solution, additional data is supplied by a separate IMU which can provide a sufficient amount of data (indicated by the yellow and red areas for the same device and the distributed version respectively). As discussed in both chapter 2 and 3, then the data often requires some middleware, should one wish to eliminate any potential sources of error. For the camera, this involves metadata, or rectification of images, both of which are quite simple to perform once one has obtained the proper data/calibrations, and for the IMU it involves filtering and preintegration.

For the prototype, these elements were naturally implemented. The chosen camera setup was set up to stream the rectified images at a rate between 20 and 30 Hz (see section 4.3) The sensor filtering attempted (ignoring the preintegration which follows from the SLAM method) included a simple averaging window, a low-pass filter, and, of course, raw data.

4.2.2 *Location and Mapping*

After the input has been appropriately handled, it is fed into a SLAM module. This module has the ability to publish the camera's current pose, all transformation data required, that is a transformation between the world coordinates, the camera coordinates, and the drone body coordinates. The module must also be capable of recording and later publishing a path for the ground truth, under the assumption that the path may change (in our case due to the keyframe-based approach of recording the path).

4.2.2.1 *Visual-only Solution*

As described in chapter 3 then the ideal localization/mapping technique for a visual only construction was chosen to be ORBSLAM. Our idea was that the problem of one has recorded ground truth

path becoming inconsistent as the map improves may be improved by making use of the internals of ORB-SLAM, i.e. by having the trajectory consist of keyframe locations which are in the instance of the ORB-SLAM map itself. With this, the trajectory will automatically be corrected to the most correct when the map is updated throughout flights, continuously updating the quality of the map, the accuracy of the trajectory itself, and thus the overall execution of the flight. This is also what was eventually used for the primary experiments. However, the system itself is built such that one can adapt it to their own SLAM method (at the cost of not having our changes present, of course). The system is constructed for a solution which provides its map in the form of point clouds, and for the navigation to work, the method should also publish a path in some fashion. Our visual SLAM node makes use of a version of ORBSLAM very similar to that of [18], with the exception of it being fit to perform communication over the ROS network, and it has been modified such that it publishes a path based on collected keyframes regularly. This is done in an attempt to not only reduce the amount of communication over the network but also in an attempt to make the path as initially smooth as possible, as this would enable any path processing to be performed quickly. As mentioned, then another advantage of using the keyframes while they are still in the system is, that if the map is corrected due to the continuous observation of the area, the keyframes will be corrected as well such that they best fit into the newly updated map. It is thus performing a sort of continuous optimization of the points observed by said keyframes, potentially improving the overall flight. Configuration wise all this requires is a carefully calibrated camera, as to enable the best possible projection of points into the mapping coordinate system. As any consumer level drone is bound to be equipped with a rolling shutter camera, the reprojection error of the camera calibration is required to be quite low, as a rolling shutter error poses a certain risk of causing substantial a photometric error which could affect the final input. A reprojection error of at least 0.5 pixels was arbitrarily chosen as a sufficient restriction, as it is quite low.

The prototype follows all of the above demands, including the modified ORBSLAM implementation and the reprojection error of the calibration clocked in at 0.13 pixels using a long series of carefully made chessboard calibration images.

4.2.2.2 *Visual-Inertial Solution*

The inertial version follows a lot of the same structure of the visual-only solution. However, the configuration is a little more involved. Naturally, very careful camera calibration is absolutely necessary for the projection and distortion parameters, but due to the setup of the camera and the IMU, there are two more adjustments required, namely the extrinsics calibration, and the noise calibration.

The extrinsics calibration is performed to make it clear to the system, how the IMU and the camera are oriented with regards to one another. This is particularly important to get right as any error in the extrinsics will cause a major misalignment in the visual-inertial data. For this reason, it is also important that the extrinsics remain as correct as possible. This means that the positioning of the IMU should be robust and in no way susceptible to shifting as a result of the drone moving about.

The noise calibration of the sensor pertains to an estimation of the Allan standard deviation table explained in chapter 2 and briefly in chapter 3. This is required to perform the best preintegration of the data, and remove any potential noise which made it past the filtering. Considering the limitation of price, it is very likely for the sensor to contain a fair amount of noise, so the integration of the sensor data may be very important, and will at the very least improve the stability of the data. The subject of visual inertial alignment when it comes to SLAM that incorporates inertial data was discussed in chapter 3 and is a major source of error when one attempts to use them for automation. Being unable to incorporate something as strict as hardware synchronization in the system inspired us to attempt to keep the distributed nature of the system, whilst performing software-based synchronization. Going by figure 10 then the main idea is to have all devices in the orange area synchronize their clocks with that of the "master" of the system. The two input nodes make use of a relative clock-synchronization protocol (in this case PTP) to have them obtain an internal clock which is, theoretically, within microseconds of the same time as the "master" system. The idea is then that the two systems, while not technically synchronized in the traditional sense to one another, will be timestamping their data with their synchronized clocks, presumably obtaining a good enough synchronization to have the visual and inertial data align properly. Any desired filtering of the inertial data may be applied before sending it to the "master".

4.2.3 *Scale Estimation*

Specifically for the challenge of scale estimation for the monocular visual solution, an idea based on the system's description (i.e. the fact that it is specific to drones) for approximating a sense of scale was thought up. The essential idea sprouts from the fact that drones can employ quite specific navigation simply by the virtue of their construction. They do not pose issues with things like friction from moving on a surface and thus the only factor which would affect a functional drone in movement is air-resistance (a substantially less worrisome issue when moving indoors). Naturally, this also applies to commodity/cheaper hardware, at least to a sufficient degree for this idea. This idea while inspired from [15], but with the added gain

that by having movement through the map in the method, and by applying statistical analysis to the data, one may use the method to validate suspicions of scale drift.

The primary goal of the solution is to attempt to estimate the ratio between the scale of the modelled world, and the scale of the real world. Naturally, with many devices, this is a non-trivial task, as one needs to apply some known constant actuation in the real world, like moving on an axis at a constant velocity. Luckily, for the role of the drone, one can get quite close to such an ideal by applying a small control signal, which naturally very quickly stops accelerating. The point from then is to find a way to best approximate the ratio of:

$$\frac{\Delta p_{real}}{\Delta p_{model}}$$

with Δp naturally being the shift in position for the particular "realm". For this, one can quite naturally estimate Δp_{model} by choosing a time interval of measurement within the near-constant movement and taking two pose-measurements provided by the SLAM method. The task of approximating Δp_{model} is slightly more fastidious. In the implementation that follows of this, the approximation of this is applied as

$$v \cdot (t_2 - t_1)$$

With v being a constant which serves as an approximation of the velocity the control signal creates, and t_i being time-stamps of the poses provided in the time interval being measured upon. In summary, the "scaling factor" which would then be applied to the coordinate system of the model would be calculated as the mean of several instances of the following measurement while moving at a near-constant velocity:

$$\frac{p_2 - p_1}{v \cdot (t_2 - t_1)}$$

Naturally, it would be unrealistic to assume that one could apply a single method for calculating the velocity of the device based on a control signal, however, for the purposes of this system, the shift does not necessarily have to represent reality accurately. Instead, it merely needs to be possible to obtain a consistent measurement, which makes all maps provided to the system have the same sense of scale. In this sense, while a near-real scale estimation is the optimal (and naturally the desired) goal, it is not a necessity for the control system to prosper. Ideally, one would be in possession of the dynamic model of the device, and would such be able to know the exact speed of the device no matter the control signal, but this is often considered a matter of much secrecy when it comes to commodity hardware. It should for this be clear, that one could decide on one particular control signal and then make the necessary measurements, but for the sake of generalization, it is technically possible to avoid doing so.

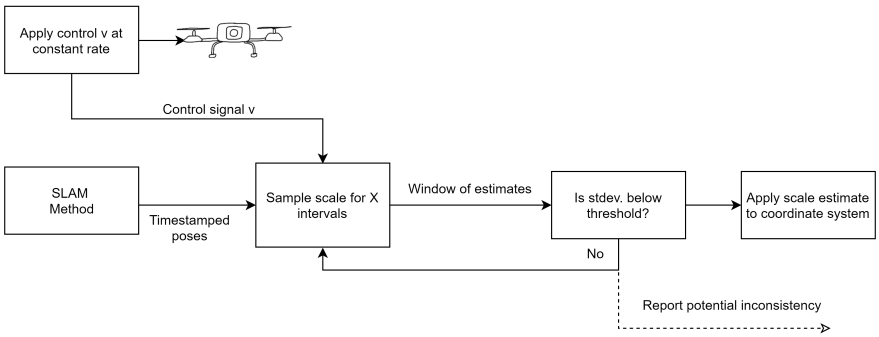


Figure 11: A sketch of the structure of the control based scale estimation.

It should be said, that the final result will be a rough estimation, and it is quite possible that the quality of the map will play into how well the scale can be approximated. For example, if the mapping process has experienced a substantial amount of scale-drift, the different measurements of the scale should vary significantly. An example of this would be if one happened to encounter a localized area of clustered points as a result of scale-drift. Then one would encounter a significantly different sort of scale estimate should one estimate the scale from that one area. Of course, this would in itself prove an interesting find, as it could provide insight into the quality of the said map, and perhaps indicate when it is good enough, as a map with such a sufficient scale drift would be problematic to navigate through. Based on this, the idea was decided to be worth pursuing together with the rest of the system by having the primary task of the estimator be the apparent task of estimating the scale, and the secondary task be to analyze a collection of estimates, concluding based on how it varies, whether the map (and by extension the estimate) is good enough yet.

The control-based scale estimation was implemented as a separate module which may be activated when desired. As theorized, then the estimation of the scale is assumed to differ as one moves through the map if the SLAM model is subject to significant drift in scale. Based on this, it would make sense to investigate the standard deviation of the estimated scales, either throughout the entire measurement process or over a sliding window if one needs to save memory. The view of the entire set of estimates could be used to investigate the quality of the map, i.e. the data would vary significantly if there are areas of significant scale drift present in the map. The sliding window can be used to get a statistical view of the most recently analyzed area by the drone. If the scale estimates satisfy a low enough level of deviation, it is presumed to be a "good" estimation of the scale, to the degree that it will provide sufficiently consistent map sizes. The hypothesized structure of the scaling module can be seen in figure 11. Of course, all of this is hypothetical and must be tested in an experimental context.

4.2.4 *Path Planning and Segmentation*

By applying the tools described above, we will achieve a location and a mapping of the environment around the drone. The system is designed such that one may take the keyframe-based paths from the SLAM nodes, and save/modify them, so they best fit the navigation of the device. These paths are at some point marked as being the proper paths for navigation, and the "path-segmenter" may then be toggled into "navigation mode". In "navigation mode" the segmenter provides the controller node with chunks of the path in the form of "segments". These segments consist of simple lines forming a linear path for the drone to fly. By providing as simple a task as possible, this enables the controller module to focus on getting as much of the provided stretch into the viewport of the drone's camera.

The segmenter decides on segments by responding whenever the SLAM method provides a newly acquired pose of the drone camera. Of course, even with keyframe-based paths, it is possible for there to be spurious shifts of small errors in the path that could make navigation more difficult. As is such, the segmenter keeps track of the general area of the drone, checking whether any of the following path segments are within a certain error margin of the device. If this is the case, it skips those segments. For the experiments, a couple of different margins for this area were tested, with the chosen setting being a constant circular radius around the device. Naturally, should the visual solution not have a sense of scale, this would not be possible, as the assumed error radius would become inconsistent between flights, but with the assumption of some estimated scale, a constant radius should work. A visualization of the segmentation can be seen in figure 12. How big this error margin is, depends on how much error we allow the drone to have at each goal of the segments. The consequence of having the error margin too big, is that we allow the drone to follow an only near correct trajectory, leaving an unresolved error in the system for each segment. By having the error margin smaller, the error will be between the trajectory of the drone and the ground truth will be much smaller. The consequence can be seen in figure 12 on the right. Although having the error margin smaller resolves theoretically in a smaller error. Having it too small will generate more error. The noise from the drone will start to matter if the error margin is set too low. Upon trying to reach the goal point of the segment, the noise may prevent it from hitting the exact coordinates needed to continue to next segment, leaving the drone circling the point and accumulating an even bigger error. Finding the correct size of the error margin is to find the balance of the observed amount of noise on the drone and the strictly allowed error of the system, then tune the error margin based on that noise. In our implementation of the path planner node, we chose an error margin of 0.089 using the metric of

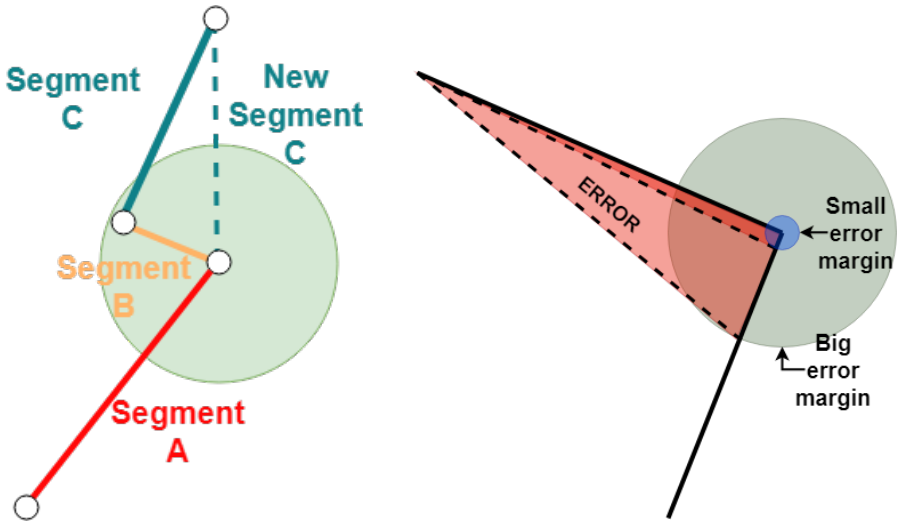


Figure 12: On the left the filtering of noisy segments is shown. The goal point of Segment B is within the error margin of Segment A, so a new segment C is created to be a direct trajectory from the goal point of segment A to the goal point of the old Segment C. On the right is the error proportional to the size of the error margin for a segment.

the scale estimator when applying a metric approximation of the control signal, but this may, of course, be adjusted to whichever absolute scale one works with. One of the significant choice factors for this error margin was that the drone should never miss hitting the goal point. The reason for valuing this feature is based on observing the drone in action. If the drone is not hitting the goal point in the first try, it will have very slow movement to correct the error, leaving the drone hanging in the air for longer, and it may be circling the goal point for some time accumulating the error. With this in mind, we have tuned the error margin around these scenarios finding a sufficient balance for the error margin.

4.2.5 *Drone Controller and Output*

The drone controller node is the last node to process data before sending it back to the drone API node. The drone API node is the intermediate node between the drone and ground control system, sending and receiving information from the drone through a local network. So the core functionality of the controller node is given the current state of the drone, to find the best control command minimizing the distance from the drone to the goal position.

As mentioned in section 3.2, two different solutions are implemented, each one with its strength and focus. The proportional controller with drift correction is focusing on minimizing the accumulated drift as fast as possible, where the PID controller is focusing on getting to

the goal position as accurate as possible, leaving the drift as an error. Both controllers take two inputs and return a vector describing the direction and magnitude of the next control command. Given as input is the current segment controlled by the path planner node and the position given by the SLAM node, in the next couple of section, a description of how these controllers were implemented and tuning of the parameters.

4.2.5.1 *Implementation of proportional controller with drift correction*

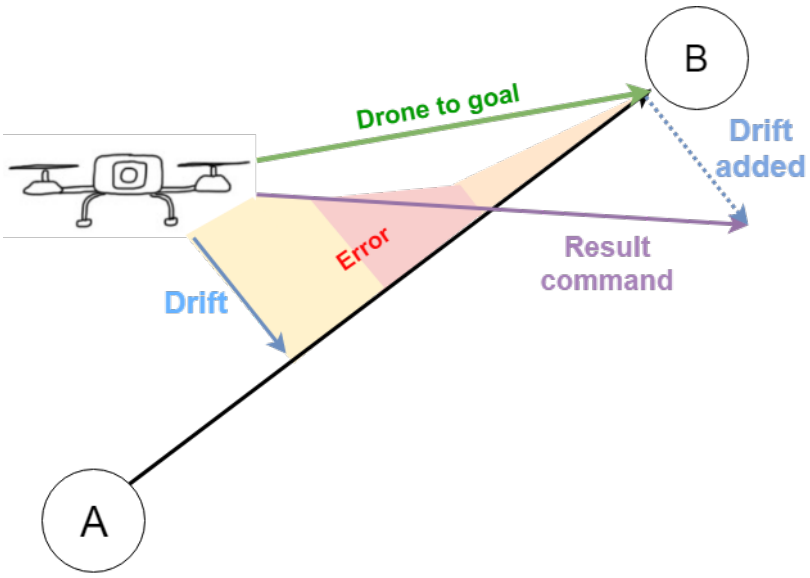


Figure 13: Proportional controller with drift correction. In this image, the error is corrected in 3 commands, hence the three color error, but in reality this would take a lot more.

The implementation of this controller is based on minimizing both the drift error and the distance from the drone to the goal point of the current segment.

To compute the drift, we find the orthogonal vector from the drones current position $p = (x_p, y_p, z_p)$ to the line segment. The segment given as input is two *start* $= (x_s, y_s, z_s)$ and *goal* $= (x_g, y_g, z_g)$, and the line between these point is given by:

$$v = \begin{bmatrix} x_s & +(x_g - x_s) & \cdot t \\ y_s & +(y_g - y_s) & \cdot t \\ z_s & +(z_g - z_s) & \cdot t \end{bmatrix}$$

So the squared distance between a point on the line with parameter t and p is therefore:

$$\begin{aligned} d^2 =& [(x_s - x_p) + (x_g - x_s)t]^2 \\ &+ [(y_s - y_p) + (y_g - y_s)t]^2 \\ &+ [(z_s - z_p) + (z_g - z_s)t]^2 \end{aligned}$$

Finding the orthogonal vector, is finding the vector minimizing the distance between the drones position and some point on the line, but to find this point on the line with parameter t, we set the $\frac{d(d^2)}{d(t)} = 0$ and solve for t to obtain:

$$t = -\frac{(start - p) \cdot (goal - start)}{|goal - start|^2}$$

By inserting t, computed by the above definition, into the formula of the line v, we get the vector to the point on the line segment which results in the shortest distance. We call this vector v_l and the vector to the current drone position for v_p. The drift vector is then computed by:

$$Drift = v_l - v_p$$

This is also the orthogonal vector from the point to the line segment.

After finding the drift vector, the vector from the drone to the goal point of the segment is found.

$$v_{d2g} = v_{goal} - v_p$$

To find the resulting move command, we combine the drift vector and the v_{d2g}, by adding them together into the final command. This results in a command which takes into account the accumulated drift and the trajectory. The higher the drift distance, the more will this method tries to correct it. This is illustrated in Figure 13, where the computation and concept of the controller are shown.

To tune this controller, we added more focus on correcting the drift distance, by introducing a scaling factor of 1.5, which is added onto the drift vector. Additionally, In an attempt to minimize the speed in which the drone is approaching the goal point, a proportional method is applied. Meaning the amount of speed is proportional to the magnitude of the vector v_{d2g}.

4.2.5.2 *Implementation of proportional–integral–derivative controller*

For each axis; x,y,z in the drone coordinate system, a PID controller is designed and tuned to minimize the distance between the current drone position and the goal position of the segment. The axis in drone coordinates is as follows; x is forward and backwards, y is strafing left and right, z is up and down. We split the commanding task into three PID controllers because each controller can only control one output, and secondly, we have the option to tune the controllers separately.

For each controller we are given; the distance $e(t)$ between the drone and the goal in the corresponding axis, the time when the drone recorded this data.

For the proportional part of the controller, it is multiplying the distance $e(t)$ by the tuned weight k_p of the corresponding axis, resulting

in $e_p = e(t) * k_p$. The output e_p is proportional to the distance.

The Integral action of the controller is tricky since tuning this too high, and it will apply to much force and the drone will overshoot its goal, but too low will result in no change will be applied. The Integral action is found by taking the integral of the error over the time from the start of the segment to the current time.

$$e_i = \int_0^t e(t)\Delta t$$

The Δt is the time difference between the last command given $t - 1$ and the command on time t. A proper PID tuning will calculate precisely how much Integral to apply for the specific process since it can very quickly get out of hand. To prevent the integral action from getting too high and dominate the command control, a capped value is implemented to ensure the output e_i never deviate more than 0.05, resulting in less overshooting on longer segments, but still enough to overcome dampening effect of the derivative action.

To prevent the drone from overshooting its goal, we add the derivative action to the equation. The derivative action is an estimate of the future distance, based on the current rate of change. The core functionality is to slow down the drone when closing into its goal, but tuning this to high will result in never hitting the goal at all, this is what we define as the dampening effect. In general, we are adding the derivative action to allow bigger k_p and k_i and still keep the system stable, resulting in a faster response, better loop performance and quicker movements on the drone without overshooting. To compute the derivative action, we define $\Delta e(t) = e(t) - e(t-1)$ as the change in distance and compute the output as:

$$e_d = \frac{\Delta e(t)}{\Delta t}$$

which is the current rate of change in distance. After computing all these elements of the controller, the move command is computed by

$$C = e_p + (k_i * e_i) + (k_d * e_d)$$

The tuning of the PID controllers is to find the weights; k_p, k_i, k_d leading to a satisfied result. The tuning of our prototype was performed by allowing it to fly a specified route and observe the amount of overshooting, dampening effect and speed the route was performed in. There is no single universally recognized way of consistently tuning a PID controller, and while there have been suggestions for such, then for this particular implementation the tuning was based on observed behaviour, with an initial guess coming from the simulation.

The resulting tuning values used for the prototype in the final solution is described in table 1. The z-axis is tuned without any integral

actions since the drone is generation a lot of noise on this axis. So naturally, the drone will prevent the dampening effect from happening and without any integral actions, the overshooting is minimal. The highest tuned value is in the derivative actions, since we preferred a slower performance with less overshooting, leading to a more accurate flight.

Table 1: tuning of the PID controllers

axis / tuning	PID on x-axis	PID on y-axis	PID on z-axis
P	0.3	0.3	0.3
I	0.01	0.01	0
D	0.4	0.4	0.4

This concludes the description of the individual elements in the fully fledged prototype. The tuning of controllers will result very differently, depending on the use case and the type of hardware used. Therefore, the next section will provide a description of the hardware used during testing and experimentation of the prototype and modification made to fit the use case.

4.3 CHOICE OF HARDWARE

While the eventual experiments typically rely on confirmation by simulation, the prototype system had to be implemented and tested in a real physical environment for an actual physical test. In the last section, the actual design of the system, and the prototype was described, where as this section describes all the hardware required for eventual tests and experiments. There is a wealth of challenges present when choosing the hardware for such experiments, especially considering the limitations of affordability. This section also elaborates on some of these challenges.

The choice of the actual drone was determined based on the following factors:

- As the main motivation of the entire task, it must be affordable.
- Many of the existing affordable drones with an exposed development platform are mainly developed on using an Android API, as to more easily facilitate usage on smartphones. However, this would cause a substantial development tax both in erroneous development, but also in the form of significant communication interception for what may be a very busy system. This is commonly the case especially with systems employing computer vision to any degree. Preferably, a machine with a more robust development platform such as ROS, would be pri-

Table 2: Bebop 2 specifications

Drone type	Quadrotor
Camera sensor resolution	14 megapixels
Camera resolution	Up to 1920 by 1080
Camera streaming framerate	30 frames per second
Carry capacity	Approx. 200-300g
Sensor frequency	State changes available at 5 Hz

oritized, as this would make the distribution of the system very simple

- It should go without asking, that the drone should be equipped with a camera which it can stream over an internet connection.

Based on these factors, the chosen drone became the Parrot Bebop 2 (see table 2). The Bebop 2 is an affordable, consumer-level drone with a decent camera capable of streaming to a ground-station at 30 frames per second (reduced to 856x480 to allow the subsequent processing at the ground-station), acceptable flight capabilities, and sufficient carrying capabilities to carry whatever other components would prove useful such as the sensors for the subsequent experiment. The Bebop 2 is also armed with a low level bottom-facing odometry based on a low-resolution camera and a depth sensor.

Additionally, then there are open source libraries available that fit the native development code of the machine's firmware to the aforementioned ROS requirements. For this project, we employed Bebop Autonomy [27]. From an implementational standpoint, outside of the application of this library, very little logic specific to the drone's build and type was applied in the prototype. By the design of the system, the control elements themselves exposes a topic which sends a "twist" message, describing the desired linear and angular movements. This follows the desire of designing the system to be as modular as possible, so any type of drone hypothetically could be used, as long as it publishes a camera stream (and assuming one has performed the necessary calibrations), and that one can turn these "twist" messages into a control signal.

One thing which is generally the case for consumer-level drones is the fact that the usefulness of the natively provided sensor data is relatively low. Essentially, then they only provide the data when the data changes to some degree. This is often sufficient for hobby projects and is a common method of preserving power by only presenting sufficiently different data at a lowered rate. Unfortunately, to enable the usage of inertial data for the purposes of this report's experiments raw or filtered raw data is usually preferred. As an example, then the authors of [21] recommend at least a camera frequency of 20 Hz and a raw sensor frequency of 100 Hz, whereas the Bebop 2 natively only presents state data at approximately 5 Hz, which is naturally not remotely sufficient to perform a visual inertial solution. This was previously mentioned in chapter 3, and as concluded in there, this was considered inevitable, as very few consumer-level drones present raw sensor data either way. Besides, then the purpose of one of the setups is to investigate the viability of distributed visual inertial SLAM, and as such, the challenge must be investigated either way.

To demonstrate the full process of the visual-only solution, we have constructed a short video demonstration which showcases a few

of the executed steps in the eventual experimentation process. This video may be seen on https://youtu.be/haKltQHmPk4

For the version of the system that makes use of inertial data for the localization, an IMU is an obvious necessity. Once again, as affordable drones do not present sufficient data for this, an external IMU was required. Following the literature discussed in chapter 2, at the very least, this sensor, should present linear acceleration and angular velocity, however, the inclusion of a magnetometer is preferred. In adherence to the requirement of affordability, the sensor must also be of sufficiently low price, as the challenges of noisy/biased data of a consumer level IMU is a part of this experiment. The chosen sensor for this experiment is the `LSM9DS1` (partial data sheet available in appendix B) mounted on a Raspberry Pi sense-hat [9]. Note that the usage of the sense-hat is entirely due to the fact that it enables ease of use by simply mounting it onto a Raspberry Pi model with GPIO pins and a power source, which also makes it easier to apply any filtering desired simply by writing additional software. This sensor enables all the degrees of freedom described earlier, and it presents data which has shown to be substantially noisier than the more commonly used high-grade IMUs for projects such as this.

To ensure the most consistent camera to IMU extrinsics, a mounting device for the drone was created using a 3D printer which enables fastening the entire sensor setup tightly onto the drone (see figure 14), ensuring the any calibrated extrinsics remain as consistent as possible. For inertial experiments in which both input devices are to be in the same system, the camera on the drone is exchanged for a Raspberry Pi camera module (v2) mounted onto the same board which runs the IMU. The Pi board itself is then built to stream both sources to the home station.

4.4 EXPERIMENTS WITH VISION-ONLY SOLUTION

The vision-only solution was the first to be implemented, tested, experimented on, and evaluated. Of course, as there were several elements to the full system which were to be investigated, multiple different tests and experiments were carried out. We will first describe the process behind the tests, after which we will delve into the concrete results.

4.4.1 *Simulated experiments and setup*

To ensure a certain level of certainty with regards to the implementation, a set of preliminary experiments were carried out which did not involve the hardware described in section 4.3. Rather, the implementation was made, after which it was run multiple times in a simulated

Figure 14: The IMU + Raspberry Pi mount.

environment. This brings along another advantage of the selected hardware, namely the existence of a ROS library, called Gazebo [10]. The library introduces a full physics enabled simulation of a given robot, of which there exist methods for simulating a Bebop Parrot 2, with all the ROS interfacing one would expect. It simulates running on the drone's local internet connection, and it simulates the availability of the front-facing camera. When running, the simulations were run in either simple geometric environments with enough textured surfaces to map the area or on a model of Castle Chillon, due to it having plenty of room for movement. An example of the simulated environment may be seen in figure 15. Note that the involvement of the simulated environment depended greatly on the purpose of the test. This will be delved further into the individual tests. While we did test all the nodes of the system in simulation, the scale estimation node was the one where we did experiments to test the effect and stability of the scaling found. In the next section, the experimental results of the scale estimation node in simulated are presented and

Figure 15: An example of the drone being simulated in the environment of Chillon Castle.

analysed for later use in the full prototype experiments in section 4.4.2.2.

4.4.1.1 *Experiments with scale estimation*

For testing and experimenting with the proposed scale estimation algorithm, the environment would first be mapped using the proposed SLAM method. Following this, the scale estimation scheme would be run as described in the implementation, that is by applying a constant control signal at a fixed rate, collecting estimates and following the flowchart of figure 11. All scale estimates were measured and analyzed. This was done in simulation, and to confirm the behaviour, it was also done with physical experiments[2]. The primary goals of these tests were to investigate:

- Whether the scale estimation showed any sign of converging consistently towards a result. Determining when a result is consistent was determined to be in such a fashion that the results no longer deviate by more than 0.4, which while seemingly arbitrary, did seem like a reasonable level at which the scaling would still be usable.
- Whether the eventual estimate could consistently be obtained for unique mappings of the environment
- Whether the quality of the map affects the ability to estimate the scale properly

2 While a few of the results are observed from the physical experiments, most come from simulation due to the unfortunate natural restriction of battery life, and the fate of the drone

- Whether applying the estimate to the mapping resulted in a more accurately scaled version of the map

A couple of examples of the progression of the scale estimate may be seen in figure 16 in which the drone is flown across a mapped area on a couple of randomly available stretches, following the premise of the experiment described above. As one may see, then it is quite clear that the estimate itself slowly settles towards a specific area small area it will then fluctuate towards. Based on the results, we believe this is due to the fact that the entire map of course technically does have the variable scale to a certain degree, as it is all just a broad set of optimal results. As is such, the scale is unlikely to be a single constant, but rather a (hopefully small) range. As one may see in the plots, then this is indeed the case, as a lot of them even look like an actual convergence. Note also, that the scale is not in the same area for each individual plot, as the runs showed were performed on separate mapping instances, so the scale assumed by ORB SLAM would differ slightly (or sometimes even quite a bit). Even so, the desired level of convergence was indeed observed.

While expected, this technically only addresses the first goal of observing convergence, however, by applying the scaling one may observe the scale when applied to the mapping itself. Examples of this may be seen in figure 17, as one may see from that example, then it was indeed possible to obtain consistent results using the scale estimation algorithm. However, as detailed in section 4.2.3, then the graphs displayed were run using the length of the control vector (0.05 for this example) as the variable v. When running in a real physical environment, it was quickly shown, that one would need either a coincidentally accurate model for control vectors, in which everything is specified in meters per second, or one would need to know how fast the chosen control vector is to obtain a metrically scaled map. However, as also described in the implementation, then it matters little, as long as the control is tuned to the consistently obtained scale. As for the connection to the quality of the mapping, this was most apparent in the physical experiments, wherein the standard deviation of the overall scale estimations would vary greatly depending on the amount of mapping applied throughout the scaling-flights. This was, however, very hard to recreate, as ORB-SLAM, for the very logical reason of optimization, does its best to avoid scale drift. When performed physically, the standard deviation for specific windows would fluctuate, comparatively, a lot when the mapping was insufficient. In an attempt to further confirm this, a set of mappings of simulated environments specifically avoiding things that alleviate error (e.g. avoiding loop closure), and attempting to induce scale drift as much as possible, were performed, and scale estimation was then run with these. This did indeed also turn out to be the case, although to a slightly less damning degree as it is hard to create a sufficiently

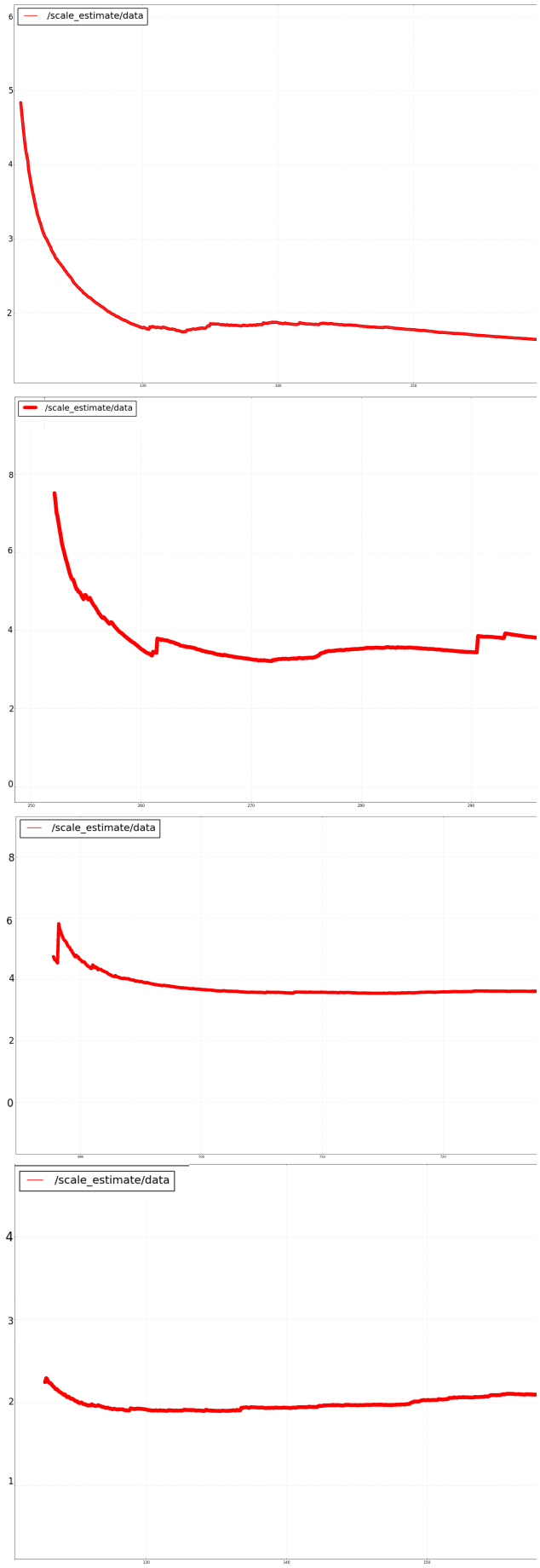


Figure 16: A set of plots showing the scale estimate over time. All of these were performed in a simulation of Chillon Castle.

bad map in a simulation, even when simulating common noise factors like wind, and simple random movement noise.

4.4.2 *Real world experiments and the environmental setup*

All physical experiments were run in a hall which sufficiently mimics the conditions of something like the industrial use case without running the risk of any physical damage due to having any workers on the scene. For the measurement; ROS messages were recorded, behaviour in the SLAM method was recorded, and additionally, to capture the real physical behavior of the drone itself a set of three Xiaomi Action CAM 4K[3] cameras were mounted (using gorilla pods) as to capture the drone's movement on both axes, two pointing downward to capture movement in the X and the Y-axis and one mounted on the side to capture the height-data. All cameras were shooting at a resolution at 1080p at 100 frames per second and had its images rectified as much as possible to avoid too many distortions as a result of the lens. With this, one could then record the drone mid-flight, and track its location with relatively good accuracy. The physical testing ground may be seen in figure 18. To enable the best possible tracking, all automated drone flight was performed without the drone rotating to face the directions supplied by the poses. We do this without too much loss of generality, as one can have the drone perform rotations between control segments should this be important. This is doable, as keyframes store a full camera pose, and not just a position. In the next section, a preliminary experiment in this environment is presented to test which controller is best suited for the full prototype in section 4.4.2.2.

4.4.2.1 *Experiments with navigation control*

As a preliminary experiment, the controllers where tested in the real world setup, to find the best performing and most accurate controller for the final experiment on the prototype. The test was to complete a trajectory formed as a Z. This shape of the trajectory was chosen for its sharp edges, where the drone has to complete a full stop before continuing onto a new segment.

The controllers were evaluated on the path performance recorded by the SLAM node, which records every key-frame the drone has collected and the position it was collected. These positions were combined to form a path, which could be compared to one another. By moving the drone by hand, we could create an almost perfectly formed Z path recorded by the SLAM node, without it being affected by mechanical noise or drift. This path is what we define as the ground truth, which the drone had to replicate using the different controllers.

3 https://www.mi.com/us/mi-action-camera-4k/

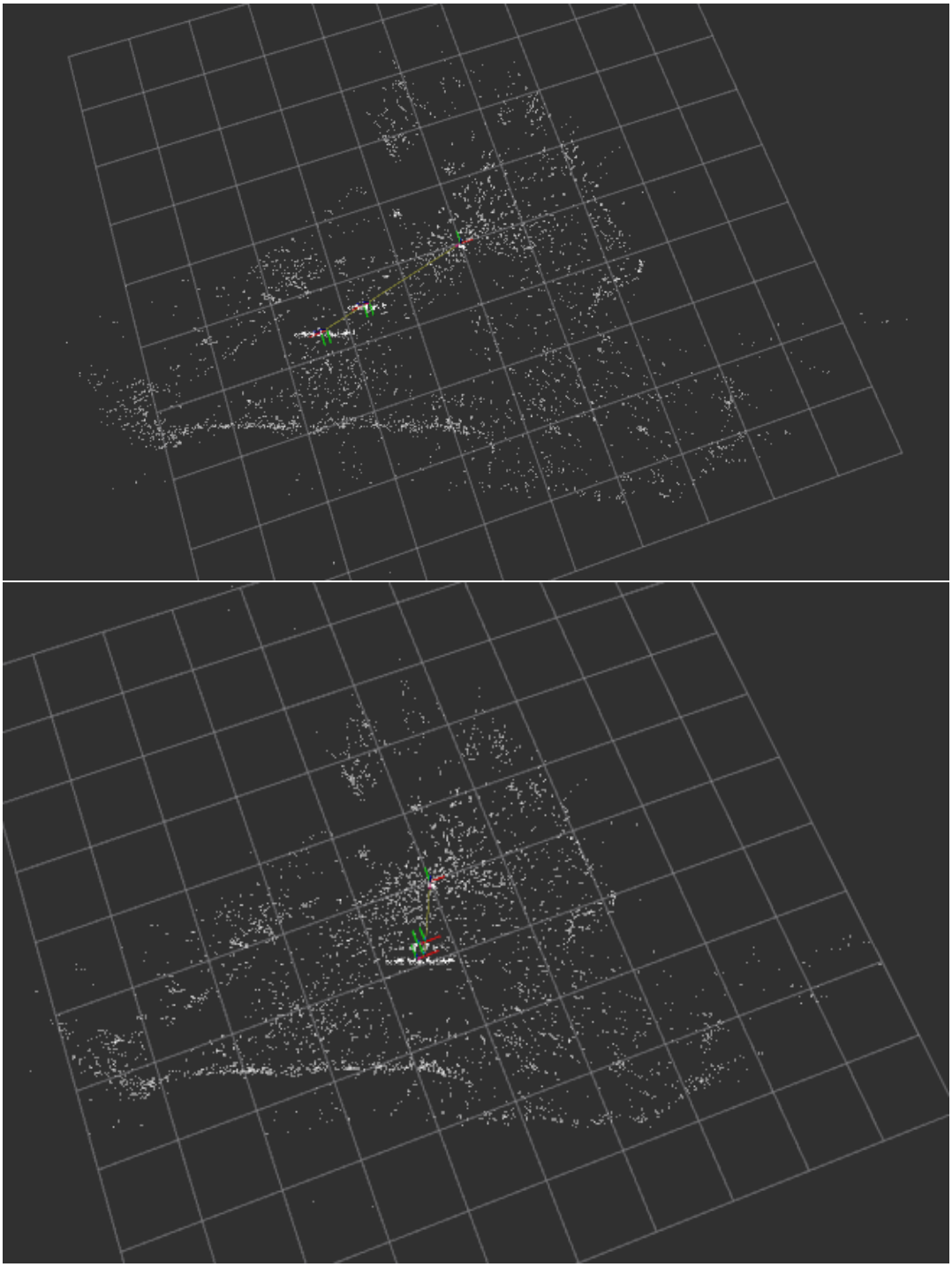


Figure 17: An example of two separate scalings performed in the same area for two different maps. This image displays the most disparate examples, as most runs would look too similar.

Figure 18: The physical testing grounds. The first showcases the environment as a whole. The blue circle indicates the placement of one of the cameras for recording. The second image displays the two downwards-facing cameras used for the horizontal axis tracking.

The controllers were then evaluated on the difference in how far the replicated path was from the ground truth path.

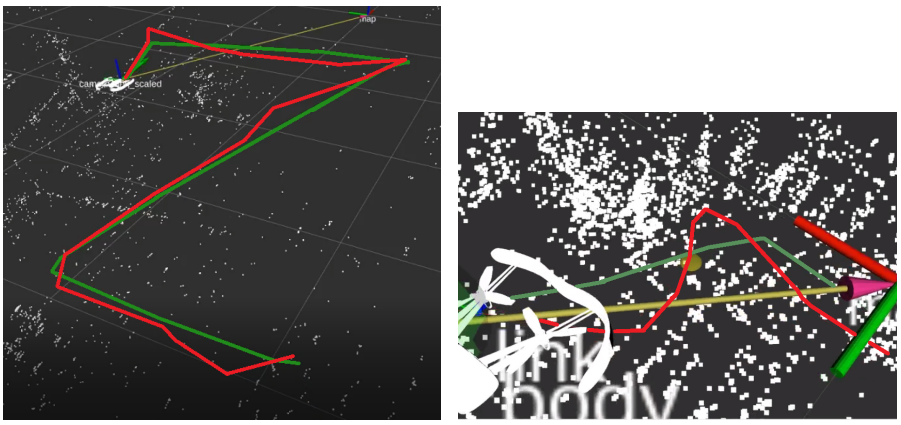


Figure 19: Controllers evaluated on a Z path. On the left we see the result of the PID controller (red) replicating the ground truth path (green). On the right, the result of the proportional controller with drift correction. Here, the focus is on the overshooting, while trying to correct the drift error.

While no error metric was computed to find the best controller, the result was clear. As can be seen in figure 19 on the right, the proportional controller with drift correction tended to overshoot the goal, leading to more of an S shaped trajectory. So even though the controller did correct the drift error, it had too much force in the resulting direction. What the controller was missing was the derivative action from the PID controller. As shown in figure 19 on the left, the PID controller had little to no issue with overshooting. So as a conclusion of the experiment, to have the best accuracy indoor, we chose the PID controller as it resulted in the most accuracy. The PID controller was used in the final prototype for the full experiment.

4.4.2.2 *Experiments with the full prototype*

Having established the viability of scale estimation and decided which controller was deemed vastly superior, the fully implemented prototype could then be tested through a set of (non-simulated) experiments. The primary goals of this particular set of experiments were then to investigate:

- How much does the scale-estimation matter for the quality of the navigation?
- To which degree does the drone manage to reproduce the ground truth?
- Are there benefits or disadvantages from the navigation controller chosen, for example with regards to different types of paths?

- Does the path remain consistent, even as a result of local map updates caused by the SLAM solution?
- Can the implemented prototype in fact carry out the task we originally set out to perform?

To answer these questions, several executions of the actual full navigation task were performed, under different circumstances, namely by varying the ground truth paths (and, in theory, with and without scale estimation). The recording setup described earlier was then put to use to obtain all the necessary data to answer the questions above.

We first began by completely confirming the first goal. While the original intention was to perform the navigation both with and without scale-estimation, it proved nigh-impossible to obtain any sort of consistent navigation without some sort of scale estimation first. Not only would the scale of the map vary, making the tuning of the controller unfit between executions, but even when tuning the controller in simulation and then running this in the real setting, the drone would move very slowly, and very unsteadily. To the degree that it would probably not be able to finish a lap before running out of battery. Luckily, then running the tasks with scale estimation made for good results, consistent enough for one to tune a controller to. Do note, that for the sake of accuracy, we estimated the speed of the drone at a certain control estimate, and applied this to the scale-estimator as described in earlier sections. This was primarily for the sake of having results which were as close to metric as possible. As mentioned earlier, then this does not affect the validity of the results, as all that was needed was an "absolute scale". This was confirmed as the navigation worked exactly as well with any arbitrary scaling coefficient (as long as the controller was tuned for it). Due to the abysmal results when running without scale estimation, it was decided not to gather results on full flights, as the inconsistency and constant retuning of the controller would cause the results to vary wildly, while still being vastly inferior.

To see how well the drone could stick to a ground truth, a set of four predetermined paths were flown several times, and the drone's error from the ground truth was then calculated. The four paths were:

- A simple path which was simply a line, referred to as the "L" path.
- A rectangular path, referred to as the "R" path.
- A circular path, referred to as the "C" path.
- A zigzag-like path forming the shape of the letter "Z", therefore having the same name.

The visualized paths may be seen in figures 20 and 21, and the calculated errors may be seen in table 3.

We first bring our attention to the visualized paths of figures 20 and 21, later confirming all of these by looking at the calculated average errors. By looking at all the trajectories, one can observe that the drone managed to follow its ground truth to a certain degree, i.e. at no point did it lose tracking beyond repair, nor did it ever skip parts, or stop following the ground truth. This was, in itself, a success of the task. In particular, impressively low deviance can be seen in trajectories for the "R" and "C" paths of figure 21, wherein the worst error seems to mainly originate from the beginning of the "R" path, and near the end. The initial error comes from the drone approaching the beginning of the ground truth, which is the default setting before beginning the flight. However, the more significant gap in error near the end of the trajectory, speaks to a source of error which may be observed again in the "Top 2 Z" plot of figure 20, in which the final line of the "Z" trajectory starts deviating again. As the PID controller does not have an explicit element for correcting drift in the way described for the proportional controller tested. This means, that if there is a sufficiently large gap between keyframe locations, the drone will not correct any error caused by residual approach, such as what may be caused by the angle of the "Z" trajectory, or the corner of the "R" trajectory. This may be alleviated in a variety of ways, one is to weigh the drift higher on the controller, another may be to increase the density of the keyframes, either by allowing ORBSLAM to take in more keyframes, or by synthetically populating the ground truth with more points which must be visited based on the existing keyframes. There appears to be some precedence for this, as the "C" path was the best reproduced path considering its complexity. However, even with the presence of the drift, the resulting trajectory is still quite good overall with a surprisingly low error. After all, the drift present was in itself quite low, and was mainly too small to see, like in the "L" trajectory in which the drift was technically present, but so low that it is hard to see in its plot. The side camera was placed to observe any dramatic deviance in height, which luckily did not appear to be the case. It does, however, appear in all of its plots, that the given trajectory was slightly noisier when it came to this axis. Even so, the overall result shows no overly dramatic shifts in height, which would imply a subpar reproduction.

These statements about the quality of the reproduction may be confirmed by looking at the calculated error of 3. An important point of note here is that the errors were initially calculated in pixels based on the actual image-based tracking and then later recalculated to centimetres. The conversion was accomplished by observing the length of the drone throughout the flight in pixels, concluding the ratio from pixel to centimetres, i.e. if the drone is 120 pixels wide in the video, then the ratio would be $\frac{23\mathrm{cm}}{120\mathrm{pixels}}$. The final ratio used for the different videos was then the one which would cause the worst error if

true. For example, for a top-based camera, the smallest instance of the drone available in the frames would cause the error to be most significant. Using this method means that the errors calculated are approximated, but they are also the worst case approximation. Including the root-mean-squared error penalty also being larger due to it coming from the pixel-based value. Even with all of this, the mean of all the mean RMS errors for all flights still turned out to be substantially lower than the initially set anticipations of 20cm. The worst observed RMS error being the deviance observed in the "R" path described earlier, causing the RMS error to hit 11.780cm which is still almost half of the expected error. Combining the RMS error from all the flight and finding the mean of RMS error results in a mean of RMS error at 5.4cm, which is considerably lower than the anticipations of 20cm. All the observations about the visualized paths may be confirmed here by the calculated error for the trajectories noted, which was slightly higher than the others, but not so far off as to make it a problem.

Based on these reflections, it was determined that the drone managed to perform the flight, and quite well too, with the worst average root-mean-square error being about 6.4cm for the first top camera, which was quite well within the expectations originally set. Even when adding the two axes, the average error is still only just below 7cm. From a purely qualitative point of view, the paths themselves are quite good. However, the lack of drift control is still visible in particular in the "Z" path in which some occasional overshoot is visible. This does not diminish the fact that the error measured on it is still quite good, but it does seem to indicate that there is room for improvement. Overall, however, the results are still surprisingly excellent. Throughout the flights, local map updates were observed in ORBSLAM, which is a very much so intended benefit of the system, as it continues to gather keyframe data and feature-points for the overall map. These improvements occurred without any issue, and any time the map updated, the ground-truths would update with them to best optimize the reprojection error as described in chapter 2. A disadvantage which must be noted here is of course that if the map is updated with a drastic new change, which shifts the entirety of the map, while the ground-truths would follow troop, the estimated scale will remain the same. However, such drastic changes are typically a sign of a lousy mapping and do not occur when one has even a moderately good map.

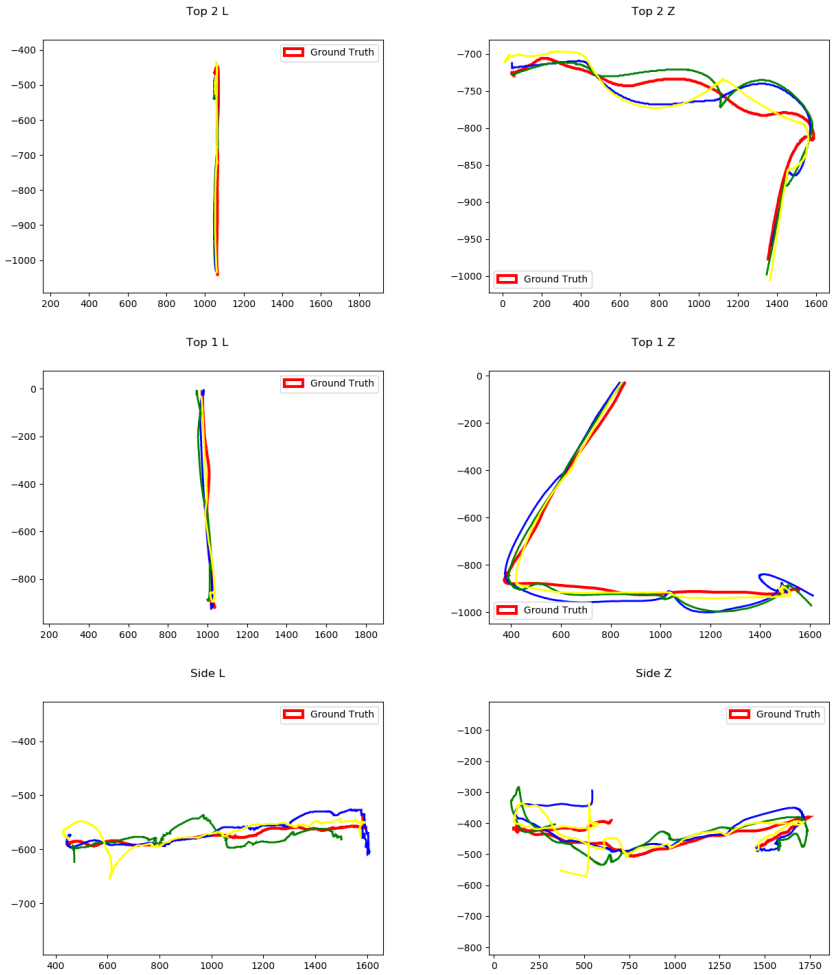


Figure 20: The L and Z trajectories of the flights plotted together. Please note that these plots have been scaled, and that the axes measure in pixels.

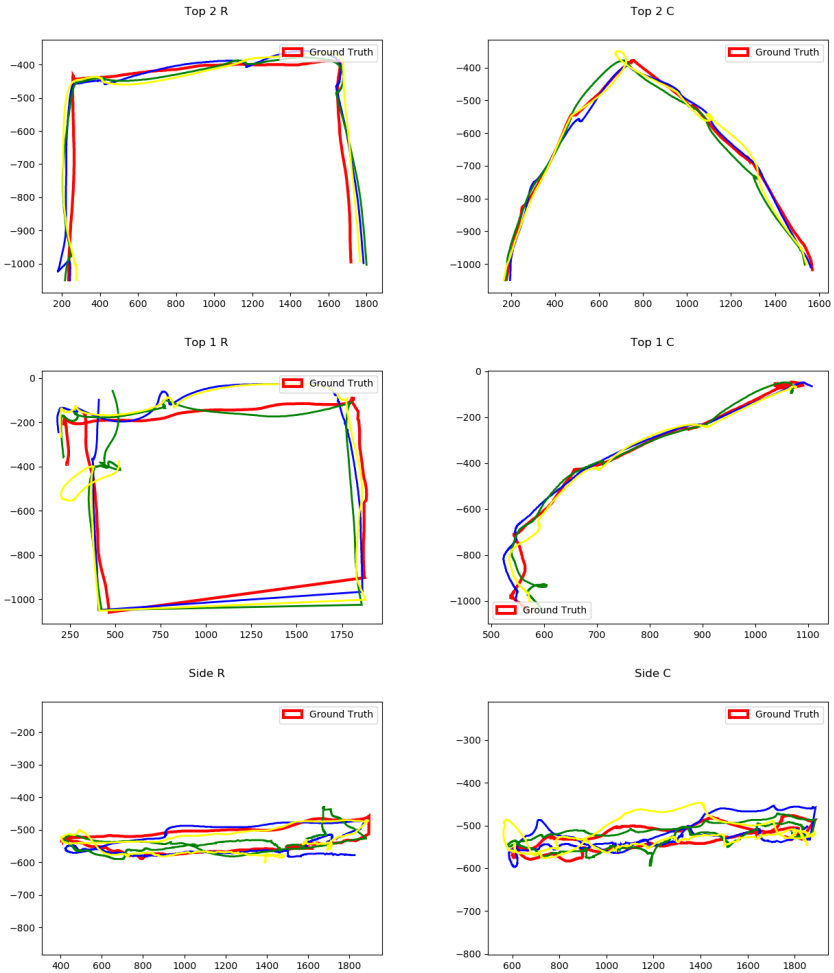


Figure 21: The R and C trajectories of the flights plotted together. Please note that these plots have been scaled, and that the axes measure in pixels.

Camera - Trajectory	L	R	C	Z	Mean
Top 1 (pixels)	12.926	56.099	12.998	40.128	30.537
Top 1 (cm)	2.714	11.780	2.729	8.426	6.412
Top 2 (pixels)	14.348	24.382	12.846	19.487	17.766
Top 2 (cm)	2.755	4.681	2.467	3.741	3.411
Side (pixels)	25.999	13.977	16.329	27.527	20.956
Side (cm)	7.799	4.193	4.898	8.258	6.287

Table 3: The calculated root-mean-square errors of the flights. The flight paths calculated upon were trajectories L, R, C, and Z

4.4.3 *A note on the current status of the drone*

We would like to note on an unfortunate event which occurred during the process of the final physical experiments featuring the drone. As it were, then due to an unexpected barometer error on the drone, the experiment in question finished with the drone flying upwards uncontrollably, causing it to collide with a lamp, and subsequently crashing into the floor. We believe this is important to point out, as the results, while glowingly positive, it was clear that it could possibly be improved even further by applying more fine tuning to, for example, the drone controller, or by attempting some of the fixes described in the evaluation. All experiments performed up until the incident in question were executed as expected, and we are certain that the error was in no way related to the implementation.

4.5 EXPERIMENTS WITH VISUAL-INERTIAL SOLUTION

Having fully accomplished the task, it was deemed worth attempting to implement the inertial solution as well. The primary goals of this part of the investigation were:

- Whether it was possible to utilize the inertial data into the prototype.
- Whether it was possible to do the same, but where the inertial data and the visual data come from two different sources.
- If the above proves true, how this affects the flight of the drone.

As predicted, then running the monocular version of VINS using a camera mounted onto the Raspberry Pi with the sensor module, resulted in a sort of tracking, but even with the preintegration performed by VINS, the tracking was still much too hectic to be suitable for drone flight. With this in mind, the next step was to attempt to mediate some of this random noise by applying some of the filtering techniques mentioned, namely by trying with the moving aver-

age and trying with the low-pass filter. Through a long and arduous process of adjusting the parameters, a couple of good settings were obtained, including the rolling average at a low window, and a few settings of the low-pass filter. Unfortunately, while the tracking got exceptionally close to optimal for the drone, the full practical application of the prototype was not attempted. When tracking the drone would occasionally, especially when being jolted by sudden movements, "drift off" as if a sudden very high bias was introduced to the camera, causing the tracking location to jet off. While not a problem for many applications, and certainly an excellent result, this is a problem in particular with drones, as they often change their movements through somewhat sudden changes in angle, and sometimes translation depending on the task. As this tracking was still deemed too dangerous for real flight, especially given the fact that the drone had been damaged in the last experiment, the determination of the last goal of this investigation was left as potential future work. We believe this issue originates from the fact that the IMU is a cheaper model, and has a wider range of noise, which means that attempting to integrate this noise is harder for the system, causing it not to be as reliable as one would wish. We believe that given more time it would be possible to do this, even with a cheap sensor, but it would require more time than what was available for this project (and possibly a couple more spare drones).

With the result of this experiment only being partial, it was decided that the distributed experiment would still be performed, but with slightly more relaxed constraints. Specifically, it was changed to *Whether it is possible to obtain tracking as good as the one where everything is integrated on one device, through a distributed setting*. This experiment thus consisted of having the visual data be provided by the drone's onboard camera, while the sensor data continues to be supplied from the Raspberry Pi sensor. To alleviate the fact that the two systems have wildly different clocks, the PTP software framework of chrony was utilized to synchronize the local network clock of the Raspberry Pi and the drone-camera with the ground-station as the base. Essentially then this makes it, so the red and blue section in figure 10 are synchronized with the green section with a synchronization error of all the way down to microseconds, in theory. Doing this led to another long process of adjustment, but ultimately, the results came surprisingly close to the tracking originally obtained using fully onboard tracking, with the same issues of drift. It should be noted that while the tracking got quite close to the same, it was subject to more fragility as a subject of time. That is to say that while the tracking could be obtained, it would not last further than a few minutes before some issues would occur with visual-inertial misalignment. We believe this is due to the nature of the synchronization platform which attempts to estimate the synchronization error, and then slow down, or speed

up, the clock of the system which is adjusting. We believe this process has a risk of causing an error to accumulate in the SLAM process as the occasional adjustment causes data to arrive in the same "slowed down" or "sped up" fashion, leading to eventual misalignment. As all of these tools were used in a somewhat separate way, there is, of course, no way for the system to know of these adjustments. As is such, the result of this part of the experiment can be formulated as such: While one can get very close to obtaining tracking as good as the last, it is a more prominent subject to issues as a result of the bigger reliance to the network. Even so, we believe there is some precedence here to state, that it may be possible to perform good distributed visual inertial SLAM given a more tightly coupled setup, perhaps a custom synchronization platform which can report adjustments to the system in some fashion. Sadly, we find such a solution is out of scope for this report.

In this chapter, the design vision has been formed and expressed in details. From this design vision, a prototype has been assembled, implemented and evaluated to show that a part of this design vision is entirely possible in a real-life scenario. As concluded by the results shown in 4.4.2.2, the vision-only solution to the use-case was a success. With a worst-case result of an RMS error on 11.780cm, it was still almost twice as good as the initial set anticipations of 20cm. As for the results of the visual-inertial solution in section 4.5, while the localization was deemed too risky to fly, we did find the performance to be very close. With the distributed version of the same solution, some potential problems which lead to a complication with flying the drone for an extended period were exhibited, but nevertheless the synchronized, yet entirely distributed version came close to performing as well, like the one in which everything runs on the same device. In the next chapter, we have some honourable mentions of techniques we tried to implement to improve the quality or user experience but did not lead to any new result.

5 REFLECTING ON ATTEMPTED METHODS

Being the investigation that it is, it should be pointed out that the works detailed throughout this report mainly describe the analysis and execution of something which carried positive results. This is, however, not the only work that were worked on during the process. Throughout the process, we implemented many different solutions which could potentially improve the quality or user experience of the final product. In the end, all had different demands which required a lot more work to fulfil or which we did not have the correct equipment to fulfil. In this chapter, to give an overall idea of the full process, we give a summary of two of these solutions which were attempted in the implementation but had to be let go, despite great potentials. We reflect over these solutions, and parts of the fully implemented process, namely the ground-truth recording and the experimental data, and highlight where potential issues and/or solutions arise.

5.1 CHOOSING THE RIGHT SLAM METHOD

In chapter 2, we go into detail with how visual SLAM techniques work, and what should be considered when choosing a final solution. What was not detailed, were our experiments with some of the other kinds of techniques which exist.

Both ORBSLAM and VINS are indirect sparse methods, meaning that they find features in the image and create a sparse map from these features. However, the sparse map may be hard for some humans to understand, as they largely consist of single points of interest, rather than a fully "textured" map. A different idea for our choice of SLAM method, was to use a dense method to create a *dense* map so every pixel in the image would be mapped, leading to better user experience than the one currently provided from a visualisation standpoint. Unfortunately, it would require a very strong ground-station to be able to render and compute SLAM in real-time using such a solution. As a conclusion, a *sparse* method had to be used in our prototype. The next best solution would be to use a *direct* SLAM approach, where instead of feature points, it would observes the intensity of each pixel and project it onto the map. This results in a more detailed map while still being *sparse* for better performance. However, this comes with the cost of being vulnerable to light changes. The method we attempted to use in our implementation was the Direct Sparse Odometry (DSO) [8]. While being left with a slight loss in map accuracy it would make the map much more understandable. After testing this module in iso-

lation on various prerecorded datasets and implementing a module into our system, we had to make this solution work with the chosen hardware. Unfortunately, this brings the explanation to a larger issue with regards to the use of methods which optimize on photometric error, as methods such as DSO do, rather than the geometric error. Namely, the problem of performing a strong photometric calibration. As opposed to the calibrations of the methods used, which calibrated the intrinsic and extrinsic of the camera, then a photometric calibration is a more involved process which involves one obtaining data on the magnitudes of the exposure in the camera. It also includes the determination of any potential vignette in the camera. All of these elements are to be determined before one may make use of any direct visual SLAM methods, and while this was most certainly attempted, the calibration ended up needing more meticulous control options of the camera than expected, as an example being able to set the exposure time manually. The control options needed are unfortunately not always available on off-the-shelf hardware, and so despite a substantial amount of attempts, it was eventually deemed too time-consuming. Especially considering the fact that methods such as DSO were already known to be sensitive to light changes, which is very likely to happen in the use case given, if one assumes that the drones will survey a loop sorter at multiple times throughout a day.

5.2 MODEL BASED CONTROLLER

As described throughout the report, and discarded in section 3.2, an MPC is a model based controller, which is used to predict future control command and optimizes the command based on the prediction. Since Parrot refused to share their mathematical model, we attempted to implement an MPC navigation module using an estimated dynamic model of the previous version of the Bebop Drone. The model is described in [17] and is a first-order non-linear model relating the roll and pitch angles of the Bebop to its lateral and forward velocities. Implementing the dynamic model into the MPC navigation module, confirmed the suspicions that one cannot merely adapt an existing model, as the prediction was not accurate enough. To correct the model, one needs a motion capture system with high precision and frequency of around 120 Hz, to be able to measure the true values of the Drone in action. We had no such system and therefor had to discard the MPC solution.

5.3 DISCUSSION OF GROUND-TRUTH

The ground-truth path recorded in the system is technically not the true ground-truth but a filtered version using the keyframe-based recording from SLAM described in section 4.2.2. The keyframe-based

recording was chosen above the more straightforward recording of all poses produced by the SLAM method, even though the recording of all the poses would have given a more authentic path of the true ground-truth.

Using a keyframe-based path provides us with several benefits, the main benefit being that it is adaptable when an update on the local map is performed. A local update on the map corrects the position of the keyframes surrounding the affected area, resulting in the recorded ground-truth being updated to fit the new map and along with this prevents deterioration of the trajectory. Another benefit of the keyframe-based recording is that it functions as a natural filter for excluding noisy drone movements and small drifts, by the definition of a keyframe. A keyframe is created when a frame fulfils all of the following conditions:

1. 20 frames have passed since the last global-relocalization.
2. The local mapping thread is idle, or 20 frames has passed since the last keyframe.
3. Current frame tracks at least 50 features.
4. Current frame tracks less than 90% similar features of the current keyframe.

The fourth condition functions as a filter, excluding frames or poses where nothing new is happening in the frame, hence the small drifts and noisy moments. It is possible to change these parameters in the configuration of the system.

One of the problems about using a filtered ground-truth is the potential of filtering out crucial manoeuvres to avoid crashing into objects along the path. One way to avoid this problem would be to find the noisy sections of the path and use spline fitting technique to smooth the path without completely filtering out the noise or in this scenario, the crucial manoeuvres. To use a fitting technique, will however require all the poses to be recorded, resulting in the path not being corrected when the map is updated.

Filtering out crucial manoeuvres was, however, determined to be less of a problem for this project as condition three ensures that the frame is very recognizable since it has at least 50 unique features. Being recognizable ensures that when a drone is at the same location as the keyframe, the system can determine, with high accuracy, the exact location. The fourth condition is what saves us from not excluding crucial manoeuvres. When the drone is turning or is about to perform a crucial manoeuvre, the drone will be exposed to a lot of new features, triggering the fourth condition, and a new keyframe is saved to the path.

In the real world experiments, in section 4.4.2, the difference in the true ground-truth and the keyframe-based ground-truth may be of

significance. The drone is never shown the true ground-truth, due to the expressed worries about the path becoming worse over time with map updates. The drone's flight, however, is evaluated on how well it can follow the original ground-truth. The drone is following instructions based on the keyframe-based ground-truth, and so, one question which could be worth pursuing, is whether the deviation between the true ground-truth and the keyframe-based may be affecting the result. This is not a guarantee, as the noise of a completely recorded path may cause the navigation to become too busy for control, but it does stand to reason, that having all the points of a recorded path may increase accuracy of the flight, at the expense of not being able to react to map updates.

5.4 TRACKING CAMERA ERRORS IN THE EXPERIMENTAL RESULTS

Part of the proceedings for the experiments which are worth reflecting on, is the fashion in which we decided to record the trajectory of the drone for the experimental data. The usage of cameras and tracking for determination of error data is a delicate process in which one must address potential errors within the data. These error sources originate from the position and configuration of the cameras mounted in the testing environment. The potential problems with regard to position originate from the fact that the cameras used have a default wide angle view, causing the positioning gained from tracking to be based on the view after being distorted by the lens. While less of a problem for this experiment, as the distortions would be somewhat minor, we did alleviate this issue by applying the built in image rectification of the cameras to remove some of this distortion. This caused the image to be mostly stable, and the tracking was good. This does not remove the error entirely, but it removes it enough for the potential error to be relatively minor.

The positioning of the cameras was another source of error which could cause it to be harder to gain precise tracking data. That is, if the camera is not in line with the axis being tracked, we risk a certain inaccuracy, not to mention that the drone may simply go off screen depending on the error. We attempted to combat this by positioning the cameras as cleanly as possible, with the awareness of their field of view, and carefulness with where to place the drone throughout recording and flight. This process was an iterative process of placing cameras, observing their view, marking the ground and general area, and then adjusting the cameras. This process proved fruitful for the top mounted cameras, for which the positioning was quite clean, with the exception of one drop in circular paths, which means that only half of the flight was actually measured. Unfortunately, the side camera did suffer from slight misalignment with the drone throughout flight, which resulted in a slightly off angle for the data to be

perfect. Based on careful investigation of the data, and calculation of the eventual data, it was still determined that the data was useful for determining the height of the drone, due to the angle still being consistent, and due to the off angle not being so off that it would prove unusable.

As this section describes, there are certain elements in this way of recording data which may lead to error. However, we believe that we managed to combat these issues sufficiently through careful consideration and careful evaluation.

We hope that these extra reflections prove useful for imagining the full structure of the investigation, and clarifies why certain elements were not further delved into throughout the later chapters of the report, despite the efforts put into them. With all information on the investigation in place, we now proceed to the final chapter, in which we conclude the findings of the project.

6 CONCLUSION

Taking inspiration from a real use case, the project described in this report is an investigation of the potentials for providing an automated drone system in which one or more drones traverse a set of flight paths automatically. The main objective of the project is the accuracy in which the drone can record and replicate a trajectory given the ground truth. It has been of importance to develop this on a simple consumer drone, rather than a big expensive research drone. Based on the description of the problem, we investigated the possibilities based on the existing literature in fields of drone research, computer vision, localization, and navigation. Also, based on these possibilities, we applied them to our problem and our hypotheses, ultimately leading to not only the identification of the possible tools but also to the challenges that would come along with them, given the context. This all culminated in what is ultimately our contribution, namely the fully automated system. The contributed works outside of the investigation itself, are visible in the way we have modified the internal tools of the final implementation. These contributions include modification of how the localization keeps track of paths, how to obtain a sense of scale, how to control the devices for the specific problem of trajectory replication, and how all of these things fit together into an efficient networking structure, i.e. the eventual solution. We supported the viability of this implementation through systematic experiments which showed great potential, at the very least for the visual-only solution. In addition to this, we also looked into the possibility of using inertial data, given the nature of the problem, making great strides towards such a solution, even given the limitations of commodity hardware. We even made an intriguing observation, motivating the possible idea of a visual inertial SLAM technique, in which one is not required to use hardware-based synchronization to get results. While these inertial solutions proved too dangerous to execute in flight, given the amount of time given, we still believe that it provided some excellent insight into the possibilities of the central problem, as a supplement to the very positive results of the visual-only solution.

The overall result of this investigation was a success, considering a worst-case result of an RMS error at 11.780cm for one single type of path, but even the worst-case average RMS was only a little more than 6cm. We obtained a highly accurate trajectory reproduction through careful adjustment and solutions to some of the more common issues. This was managed despite the limitations of commodity hardware set by the use case. Even though the inertial solution had its issues,

the results were also interesting from the perspective of the overall field. While the entirety of the use case may not be solved with this, it certainly speaks to the viability of using automated drones in an industrial setting.

6.1 FUTURE WORK

Naturally, one does not solve the entirety of the field of drone automation in four months, and so there is room for continued work outside of our investigation. There are areas of this project which could very well encourage work in the future which builds further upon it. A few examples of this include:

- The work into the incorporation of inertial data was cut short, but did get very close, and we believe much more work could be done into obtaining the desired results.
- The observations made with regards to the possibilities of a distributed visual inertial SLAM technique could very well motivate the full development of such a technique which alleviates some of the potential issues with the synchronization. We believe that it would be possible to obtain a method which nearly rivals existing techniques if performed right.
- As described in chapter 4, then there is a full element of the design vision which would be worth investigating, namely the cloud-based element of the design, in which many drones provide data to a SLAM technique, constructing the mapped area in parallel. We believe this is not only possible, but that it could be useful in other areas which require mapping of an environment using camera data.
- As mentioned in section 3.2, a future improvement would be to use a modern controller such as MPC. To do so, a dynamic model of the drone used would have to be constructed.

We hope that the combination of this report and these possible suggestions for future work may lead to more exciting research in the field of automated drone operation in the future.

Part I

APPENDIX

A

THE ORIGINAL USE CASE

When this thesis was originally put together, it took its basis in the ideas of this use case. Note that the thesis does not aim to create the specified product, nor is it entirely about solving the problem itself. It is simply used as a way of generating the base idea, and also as an example of how the developments of the stated hypotheses could cause further development in the area. The use case was posed by BEUMER Group, Aarhus in early 2019. What follows is the proposal:

Drone based vision feed in logistics systems

Supervisors: XXX
Collaboration: BEUMER Group, Aarhus

Background and motivation

The express parcel industry is currently experiencing high growth rates world wide. This growth is driven by the rising number of B2C parcels resulting from ecommerce.

BEUMER Group delivers turn key solutions for automatic parcel sorting. An integral part of a high capacity parcel sorting system is often one or more loop sorters. A large parcel centers can sort over 100.000 parcels/hour.

A loopsorter runs at up to 2.5 m/s, so an inherent risk is that a parcel can fall of the loopsorter. This happens if the parcel does not have sufficient friction, was not placed correctly onto the loopsorter tray etc. Though this happens rarely, it is still a factor to be considered, and safety netting is always installed around the loopsorter to prevent such parcels from falling onto the floor. Loopsorters might also be installed hanging from the ceiling to free up floor space in the sorting center. Thus, it might require the use of a ladder to access the netting around the sorter.

A large parcel center will typically have someone to walk around the parcel center on a scheduled basis, to check if any parcels are located in the netting around the sorter.

Project description

The aim of this project is to replace human inspection of the sorter netting, with that of a drone. A successful implementation of this concept can lead to both lower cost (reduced manning hours) and higher quality (parcels spend less time in the netting, hence potentially reaching the recipient earlier).

Hence, the main research questions considered in this project could be:

- Is it possible to fly a drone autonomously within a parcel sortation center and are there any requirements to the building for it to be possible?
- Is it possible to identify parcels in the netting from a camera mounted on the drone?
- What options exists for automatically lifting the parcel away from the netting?

BEUMER Group offers a test system in our factory in Aarhus, which can be used for field testing. There is also the option for further testing at an actual sorting centre at one of our end customers.

Collaboration

BEUMER Group Denmark (BDK) is a world leading supplier of high speed sortation solutions for the logistics and airport industry. As a turn key supplier, our deliveries often span from early conceptual design, full mechanical, electrical and software scope as well as possibly assuming full responsibility for the operation and maintenance of the completed system (O&M).
Our 24/7 hotline call center, operating out of the headquarter in Aarhus, Denmark supports over 500 installations around the globe.
https://www.beumergroup.com/

Contact

For further information, please contact XXX

B

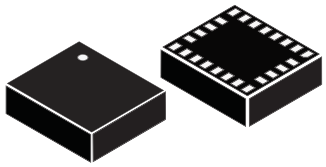
INERTIAL MEASUREMENT UNIT DATA SHEET

The following appendix contains the first page of the data sheet of the chosen IMU, the LSM9DS1:

LSM9DS1

iNEMO inertial module: 3D accelerometer, 3D gyroscope, 3D magnetometer

Datasheet - production data



LGA-24L (3.5x3x1.0 mm)

Features

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
- ±2/±4/±8/±16 *g* linear acceleration full scale
- ±4/±8/±12/±16 gauss magnetic full scale
- ±245/±500/±2000 dps angular rate full scale
- 16-bit data output
- SPI / I^2C serial interfaces
- Analog supply voltage 1.9 V to 3.6 V
- “Always-on” eco power mode down to 1.9 mA
- Programmable interrupt generators
- Embedded temperature sensor
- Embedded FIFO
- Position and motion detection functions
- Click/double-click recognition
- Intelligent power saving for handheld devices
- ECOPACK®, RoHS and “Green” compliant

Applications

- Indoor navigation
- Smart user interfaces
- Advanced gesture recognition
- Gaming and virtual reality input devices
- Display/map orientation and browsing

Description

The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.

The LSM9DS1 has a linear acceleration full scale of ±2*g*/±4*g*/±8/±16 *g,* a magnetic field full scale of ±4/±8/±12/±16 gauss and an angular rate of ±245/±500/±2000 dps.

The LSM9DS1 includes an I^2C serial bus interface supporting standard and fast mode (100 kHz and 400 kHz) and an SPI serial standard interface.

Magnetic, accelerometer and gyroscope sensing can be enabled or set in power-down mode separately for smart power management.

The LSM9DS1 is available in a plastic land grid array package (LGA) and it is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

Table 1. Device summary

Part number	Temperature range [°C]	Package	Packing
LSM9DS1	-40 to +85	LGA-24L	Tray
LSM9DS1TR	-40 to +85	LGA-24L	Tape and reel

C

ROS RQT GRAPHS

This appendix includes the full RQT graphs generated from the ROS network which the system prototype creates. This is included as an appendix, as it may comes off as confusing for the reader if they are not familiar with the intricasies of ROS.

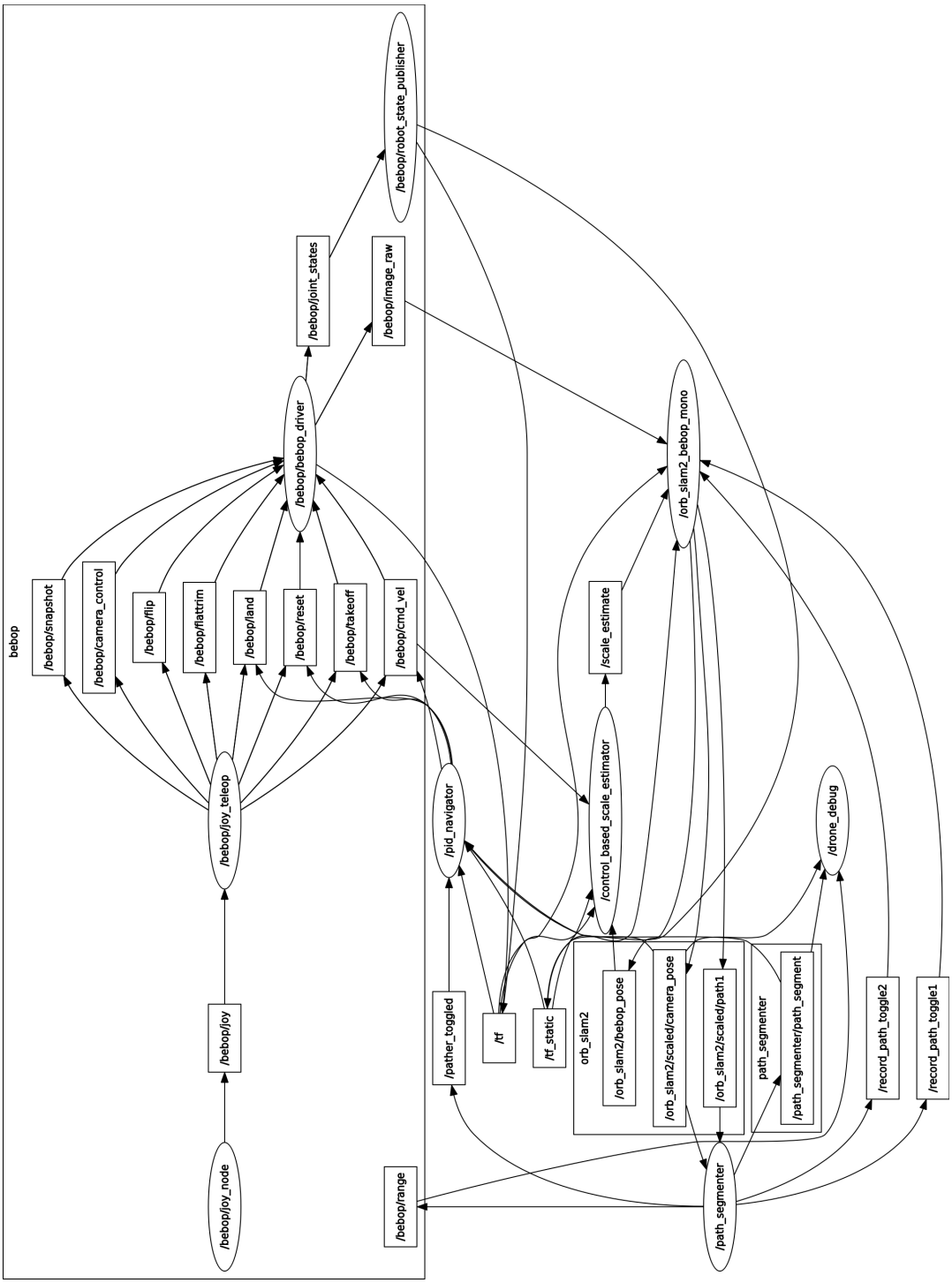


Figure 22: The RQT graph of the system, showing all active topics

BIBLIOGRAPHY

[1] Ignacio Alzugaray, Lucas Teixeira, and Margarita Chli. "Short-term uav path-planning with monocular-inertial slam in the loop." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2739–2746.

[2] Abraham Bachrach, Samuel Prentice, Ruijie He, and Nicholas Roy. "RANGE–Robust autonomous navigation in GPS-denied environments." In: *Journal of Field Robotics* 28.5 (2011), pp. 644–666.

[3] Hari Bansal, Rajamayyoor Sharma, and Shreeraman Ponpathirkoottam. "PID Controller Tuning Techniques: A Review." In: 2 (Nov. 2012), pp. 168–176.

[4] ISS Board. "IEEE standard specification format guide and test procedure for single-axis interferometric fiber optic gyros." In: *IEEE Std* (1998), pp. 952–1997.

[5] Jason Brownlee. *A Gentle Introduction to Computer Vision*. 2019. URL: https://machinelearningmastery.com/what-is-computer-vision/.

[6] Jeffrey Delmerico and Davide Scaramuzza. "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2502–2509.

[7] Jacco Drabbe. "Space engineering - Control engineering." In: *ESA Publications Division ESTEC, 2200 AG Noordwijk, The Netherlands* (2004). EUROPEAN COOPERATION FOR SPACE STANDARDIZATION.

[8] J. Engel, V. Koltun, and D. Cremers. "Direct Sparse Odometry." In: *pami* (Mar. 2018).

[9] RASPBERRY PI FOUNDATION. *Raspberry Pi Sense Hat*. https://www.raspberrypi.org/products/sense-hat/. [Online; accessed in the period from January to June 2019]. 2019.

[10] Open Source Robotics Foundation. *Gazebo website*. http://gazebosim.org/. [Online; accessed in the period from January to June 2019]. 2019.

[11] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. "Towards a navigation system for autonomous indoor flying." In: *2009 IEEE international conference on Robotics and Automation*. IEEE. 2009, pp. 2878–2883.

[12] Andres Hernandez, Harold Murcia, Cosmin Copot, and Robin De Keyser. "Model predictive path-following control of an AR. Drone quadrotor." In: *Proceedings of the XVI Latin American Control Conference (CLCA'14), Cancun, Quintana Roo, Mexico*. 2014, pp. 14–17.

[13] *Informatik IX Chair for Computer Vision Artificial Intelligence*. 2016. URL: https://vision.in.tum.de/research/vslam.

[14] Kudan. *Different Types of Visual SLAM Systems*. 2017. URL: https://www.kudan.io/post/different-types-of-visual-slam-systems.

[15] Seong Hun Lee and Guido de Croon. "Stability-based scale estimation for monocular SLAM." In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 780–787.

[16] Richard Curnow (currently maintained by Miroslav Lichvar). *chrony website*. https://chrony.tuxfamily.org/. [Online; accessed in the period from January to June 2019]. 2017.

[17] Mani Monajjemi, Sepehr Mohaimenianpour, and Richard Vaughan. "UAV, come to me: End-to-end, multi-scale situated HRI with an uninstrumented human and a distant UAV." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4410–4417.

[18] Montiel J. M. M. Mur-Artal Raúl and Juan D. Tardós. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System." In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. DOI: 10.1109/TRO.2015.2463671.

[19] Gabriel Nützi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. "Fusion of IMU and vision for absolute scale estimation in monocular SLAM." In: *Journal of intelligent & robotic systems* 61.1-4 (2011), pp. 287–299.

[20] Opencv. *Opencv ORB (Oriented FAST and Rotated BRIEF)*. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html. Accessed: 2019-4-10. 2011.

[21] Tong Qin, Peiliang Li, and Shaojie Shen. "Vins-mono: A robust and versatile monocular visual-inertial state estimator." In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020.

[22] Tong Qin and Shaojie Shen. "Online temporal calibration for monocular visual-inertial systems." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3662–3669.

[23] Seyed Abbas Sadat, Kyle Chutskoff, Damir Jungic, Jens Wawerla, and Richard Vaughan. "Feature-rich path planning for robust navigation of MAVs with mono-SLAM." In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3870–3875.

[24] Lucas Vago Santana, Alexandre Santos Brandao, and Mario Sarcinelli-Filho. "Outdoor waypoint navigation with the AR. Drone quadrotor." In: *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2015, pp. 303–311.

[25] Dale E Seborg, Duncan A Mellichamp, Thomas F Edgar, and Francis J Doyle III. *Process dynamics and control*.

[26] Tomoyuki Shiozaki. "Scale estimation for monocular SLAM using depth from defocus." PhD thesis. 2018.

[27] Autonomy Lab of Simon Fraser University. *Bebop Autonomy*. https://bebop-autonomy.readthedocs.io/en/latest/. [Online; accessed in the period from January to June 2019]. 2015.

[28] S Suzuki. "Recent researches on innovative drone technologies in robotics field." In: *Advanced Robotics* 32.19 (2018), pp. 1008–1022.

[29] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. "Visual SLAM algorithms: A survey from 2010 to 2016." In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (2017), p. 16.

[30] Nikolas Trawny and Stergios I Roumeliotis. "Indirect Kalman filter for 3D attitude estimation." In: *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep* 2 (2005), p. 2005.

[31] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. "Bundle adjustment—a modern synthesis." In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.