Contents

Acronyms

UAV Unmanned Aerial Vehicle.

Chapter 1

Introduction

1.1 A brief history of drones

UAVs (Unmanned Aerial Vehicles), more commonly called drones, are defined as flying vehicles without human operators on board. They can be remote-controlled, or controlled by on-board computers. The earliest recorded use of UAVs dates back to 1849, when Austria launched about 200 unmanned balloons armed with bombs against de city of Venice [6]. Due to unfavorable wind conditions, this attack failed, and the experiment was not repeated. The first functional UAVs were made towards de end of World War 1 and their use was, like the Austrian balloons, military. One example is the Kettering Bug (Figure 1.1), which was a torpedo with wings and a propeller developped by the US Army in 1918 [8].

Figure 1.1: The Kettering Bug (1918)

Throughout the 20th century, UAVs become more and more sophisticated, and were used more and more, but always for military purposes. In the more recent years, civilian UAVs have started to appear on the markets and their number quickly exceeded that of military UAVs. In february 2017, the FAA (Federal Aviation Administration) of the United States estimated that around 1.1 million units were in use in the US alone, and expected that number to rise to 3.55 million by 2021 [4]. These civilian drones are very different from military drones, in both their form and their function: civilian drones are usually smaller, and use rotors to take off vertically. They are used in a wide variety of applications.

1.2 Motivation

The ability to remote-control small and agile flying objects over large distances through the air, and to bring them to previously inaccessible locations, makes many new things possible. With the increasingly lower prices and better performances of civilian UAVs, people keep finding more and more uses for these high-tech gadgets. Some examples of these applications are: crop monitoring in agriculture [1], delivery of mail or parcels, construction [2], cinematography, entertainment, or search and rescue operations. In all these applications, the more autonomous a drone is, the more efficient it will be at its task. One of the main challenges to achieve autonomy is for an UAV to be able to correctly identify its surroundings, and localize itself within them. In outdoor environments, GPS systems allow UAVs to know their position with great accuracy, but this is not possible in GPS-denied environments, such as indoors. The main subject of this thesis will be fully autonomous navigation by a quadcopter in a GPS-denied environment.

1.2.1 Ethical considerations

The new possibilities brought by drones also pose ethical questions about security and privacy. Even though this technology can improve people's quality of life, it also has the potential to diminish it. If drones start to be widely used comercially, we could reach a point where the sound nuisances that they cause seriously impacts people who live in densely populated areas. Also, they can make us feel less at home, knowing that we could be observed from the sky. For this reason, it is important to adopt strict reulations regarding the use of drones in public spaces. Fortunately, many countries are already adopting legislation in this direction.

1.3 Context

This thesis is part of a project at the UCL that spans over several years and several masters theses. This project was launched by professor Julien Hendrickx in the 2012-2013 academic year, and had as long-term goal to develop a program that would enable low-cost UAVs to navigate autonomously in indoor environments. This means creating a map of their environment, localizing themselves in this map, and avoidig obstacles during exploration, using only on-board sensors. Another goal is to allow several drones to collaborate to speed up exploration. Five theses have already been written on this subject, each taking the work of the previous a little further.

2012-2015: First three theses In each of the three academic years (2012-2013, 2013-2014, 2014-2015), one masters thesis on the subject of indoor navigation for autonomous low-cost drones was written. These masters theses formed the base of the future work. They implemented visual SLAM methods to allow drones to build a two-dimensional map based on keypoints (first red pucks, then visual landmarks that the drone detected from a textured field of view), and to localize itself within this map. During this time, inter-drone communication was also established, and was used to allow a drone to communicate the location of a target to another drone.

2015-2016: Recent work Last year, two groups of students simultaneously wrote theses on this subject. Before doing so, they joined forces to reimplement what had been done previously, but using the ROS interface, an interface to work with robots that would make many things simpler, and allow more flexibility (see section The work of the first group of students allowed a drone to search and follow a mobile target, and call a second drone to continue this task when its battery was low.
The second group of students extended to SLAM algorithm to allow to use a 3D map to localize the drone. Unfortunately, they did not implement triangulation to allow to project seen points

into 3D space, but rather made the assumption that all points were located on the ground when building the map. The end result was a drone capable of using a 3D map to localize itself, but not capable of building one from its observations.

1.4 Objectives

For my own thesis, my goal is to continue the work of last year's second group, to allow true 3-D SLAM: to build a 3D map based on observations by the monocular camera. To achieve this goal I will follow the following steps:

- Research the current state of the art for 3D Keyframe based monocular visual SLAM
- Implement a way to triangulate points based on observations
- Bundle Adjustment
- Dense reconstruction
- Obstacle Avoidance

1.5 Ressources

1.5.1 Hardware

For the practical implementation of this project, a Parrot AR.Drone 2.0 was used. This quadrotor was commercialized in 2012 and is an updated version of the original AR.Drone that was launched in 2010. This drone was marketed as a high tech toy, and is designed to be controlled from a smarphone application (connected to the drone via Wi-Fi). A few augmented reality games are available for the AR.Drone, in which it can visually recognise some predefined tags using its cameras, and interact with objects or other drones with a tag. To encourage the creation of more games for their drones, Parrot has released an open SDK that allows to effectively reprogram the drones. This early release of an open SDK has made it quite popular in the scientific community to do research on autonomous flight. The drone consists of 4 rotors, each with their own electric motor and microcontroller, an internal computer with a 1GHz ARM Cortex A8 processor and 1GB DDR2 RAM at 200MHz, and various sensors.

Sensors

The AR.drone has the following sensors:

- 3 axis accelerometer with ± 50 mg accuracy
- 3 axis gyroscope with $\pm 2000°/s$ accuracy
- Pressure sensor with ± 10`Pa` accuracy
- 3 axis magnetometer with $\pm 6°$ accuracy
- Ultrasound sensor (facing downwards)
- Frontal camera (HD 720p 30fps)
- Ventral camera (QVGA, 60 fps)

The accelerometer and gyroscope together form the IMU.

Internal sensor fusion When remote-controlled during normal use, the internat controller needs a pose estimation to be able to stabilize itself. For safety reasons, it needs a reliable estimate of its altitude, and speeds, so that it can stop and hover. For convenience, it also needs a somewhat reliable estimation of its position, to cancel drifts when hovering. For these estimations, the internal controller fuses data from three sources: its IMU, its ultrasound sensor, and Odometry integrated from the optical flow observed by the bottom camera. Unfortunately, the values of these three sources ara available to be read from the drone, but not the internally fused pose, which is used directly by the controller.

1.5.2 Software

Parrot SDK

The SDK released by Parrot allows to send commands and receive information from the drone. However, it does not allow acces to code running in the drone during normal operation. It is possible to send the drone commands to take off, land, emergency stop, hover, move in a certain direction, but not to directly control the command send to the motors. Similarly, it is possible to read the IMU, ultrasound, odometry and video data from the drone, but not to access its own internal data fusion.

ROS

For the practical programming of the drone, the ROS (robot operating system) toolbox was used. This toolbox provides useful abstraction to program robots: the program is decomposed into "nodes", each node running simultaneously in a separate thread. Each node implements part of the drone's task, and the nodes can communicate between them by sending structured messages. The objective is to make the code modular, to be able to use existing nodes uploaded by other developpers. One such node, called "ardrone autonomy" created by handles communication with a physical AR.Drone, so by using this node, we can completely disregard Parrot's specific SDK, and send all commands in ROS format. This way, we hope to make it easy to reuse code when changing hardware. ROS also provides a number of convenient tools for developping software to be used on robots: it provides ways to record and replay data, to easily set and change parameters, to run only specific parts of the program for testing.

OpenCV

OpenCV is an open-source computer vision library. It provides functions and data structures to manipulate images, and implements many state-of-the-art computer vision algorithms.

PCL

PCL stands for Point Cloud Library and is an open-source library for 3D point cloud processing. Currently is is only used to visualize the map created by the drone, but in the future, it can be used for higher-level functions on this map like combining maps created by multiple robots, or reconstructing a dense representation of the map from a point cloud.

Ceres Solver

Ceres solver is an open-source library for modeling and solving non-linear least squares problems. It will be used in this thesis to solve Bundle Adjustment problems.

1.6 Outline

First, the state of the art of will be explored in chapter 2. Chapters 3 through 5 will present the main subject of this work: computer vision (Chapter 2), localisation (Chapter 3), Mapping (Chapter 5). The results will then be presented in chapters 6: Simultaneous Localisation and Mapping. Finally, we will conclude in chapter 7.

Chapter 2

State of the art

2.1 Hardware

When talking about autonomous drone navigation, it is important to be aware of what the current hardware is capable of doing, and how we can expect it to evolve in the near future. We will talk about the three main aspects of this hardware: the multirotor systems themselves, with the different possible configurations, the sensors, and finally the embedded computers.

2.1.1 Multirotor systems

Multirotors, ot multicopters, are defined as rotorcrafts with three or more rotors. Having more rotors enables them to maneuver in 3D space with with fixed-pitch rotors, unlike helicopters, which have articulations at the bases of their rotors. The most common multirotors have 3, 4, 6, or 8 rotors, and are respectively called tricopters, quadcopters (or quadrotors), hexacopters, and octocopters. Having more rotors has the advantage of giving more agility, at the cost of more energy consumption, and therefore a shorter battery life. A free solid object in 3D space, such as a multicopter, has 6 degrees of freedom: 3 for translation and 3 for rotation. To be able to directly control each of there 6 degrees of freedom, it must be possible to give 6 independent controls to the drone. This means that tricopters and quadcopters are always under-actuated: they can't directly control all 6 degrees of freedom. For example, quadrotors whose rotors are all in the same plane (as is almost always the case), can only directly control their translational movement along the axis parallel to the rotation of their rotors, and their roll, pitch and yaw angles, so to control their position in the plane perpendicular to the direction of gravity, they have to first adapt their roll and pitch, so that the resulting force of gravity and the thurst of their motors points inside that plane. Most hexacopters also work this way, as their rotors are also often in the same plane. Some hexrotors, however, have tilted rotors, and are fully actuated [9].
Octocopters on the other hand, are always over-actuated. One example, the Omnicopter, developped at ETH Zurich [3] can perform a 360°rotation along any axis, and move in a straight line in any direction, which enables it to perform complex and precise manoeuvres. It is able to catch thrown ping-pong balls with a little net.

2.1.2 Sensors

2.1.3 Embedded computers

2.2 Computer Vision

2.3 Simultaneous Localization And Mapping

Simultaneous Localization and Mapping (SLAM) refers to the joint task of creating a map of a robot's surroundings, while also keeping track of the robot's location in this map. The word "robot" should be understood very broadly in this context, for example it could be a simple handheld camera. Because there are countless different types of robots that do SLAM, SLAM is also a very diverse field, with different algorithms for different kinds of sensors. Here, we will focus on monocular visual SLAM, which is SLAM where the main sensor is a monocular camera.

2.3.1 Localization

2.3.2 Mapping

2.4 Bundle Adjustment

Chapter 3

Computer Visions

Chapter 4

Localization

Chapter 5

Mapping

The task of mapping was the main challenge of this work. This task consists in placing recognizable features in a map, that can later be used as landmarks by the drone to estimate its own position. The main challenge in the 3D case, is that a point needs to be observed from at least 2 different positions to be mapped. The simplest approach to map a point, is to simply triangulate its position from two different views. We will begin by exploring this approach. We will find that although this does work reasonably well when we are certain of the position of the cameras, it does not when this position is uncertain. In addition, this method does not allow to take more than two views into account. To remedy these problems we will implement a bundle adjustment step, that allows to build a map that is globally consistent. Throughout this section, we will have to make design choices to try to obtain a method that is both fast enough to work in real time, and accurate enough for the drone to control its position. To evaluate these performances, we will perform a standardized test.

5.1 Structure of the map

We will keep a keyframe-based structure for the map, but this structure will need to be adapted so that each landmark does not belong to just one keyframe. The map will contain two main data structures: a list of keyframes, and a list of landmarks. Each keyframe will contain the following information: the drone's pose estimation at the moment the keyframe was created, the estimated corrected pose of the keyframe, a list of observed keypoints, meaning their descriptor and their 2D position in the image plane of the keyframe, and for each keypoint, whether it corresponds to a landmark of the map, and which one. Each landmark will contain a descriptor and estimated 3D coordinates.
The map grows every time we decide to add a keyframe. When we do, the keypoints of the current image seen by the camera, along with their 2D coordinates and descriptors, are saved in a new keyframe. The descriptors are then matched with descriptors of the other keyframes, and when there is a match between two keypoints of two different keyframes, we can triangulate a new landmark into the map.
The main challenges are :
- How to triangulate new points?
- How to take into account when more than two keyframes see a same keypoint ?
- How to prevent errors from accumulating over time?

5.2 Evaluation procedure

All along the creation of the mapping algorithm, we will need to make design choices and to set a number of parameters. To make these decisions, we will evaluate the performance of the

map using a benchmark test. The evaluation will happen in two phases: in the map creation phase, the drone will initialize its map using four keyframes (a somewhat arbitrary number to ensure a large and enough map), and in the map testing phase, it will be moved to various known locations, and we will measure the accuracy of its position estimation. The setup used for this evaluation is illustrated on figure 5.2. During the map creation phase, we will try to emulate the way the drone would initialize its map just after taking off during a real flight mission (see more in section). First the drone is placed in a know position on a table (position A in figure 5.2). There, the drone is turned on, and it begins initializing its map. The drone is then successively placed in 3 other known locations (B, then C, then D), and at each of these locations, the drone manually receives its position. Every time this happens, the drone adds landmarks into the map, and optionally updates existing landmarks' positions, or even removes some landmarks. In reality, the drone would not know its exact position when making keyframes at points B, C, and D (position A is defined as the origin), especially at the beginning, as it would have to rely on its IMU and other internal sensors to estimate its position. To emulate this, we implement a second type of test: the robustness test, where the position that is communicated to the drone is slightly different from its real position, at each of the uncertain points (B, C, and D).
In the map testing phase, the drone is again placed at different known locations. This time, no information is communicated to the drone from the outside, and the drone does not modify its map. The drone estimates its position using only its camera and the map that it built during the initialization phase. We compare the drone's estimated position with its real position to evaluate the quality of the map. The two quantities we will seek to optimize are the accuracy of the drone's visual pose estimation, and the time taken to create the map.

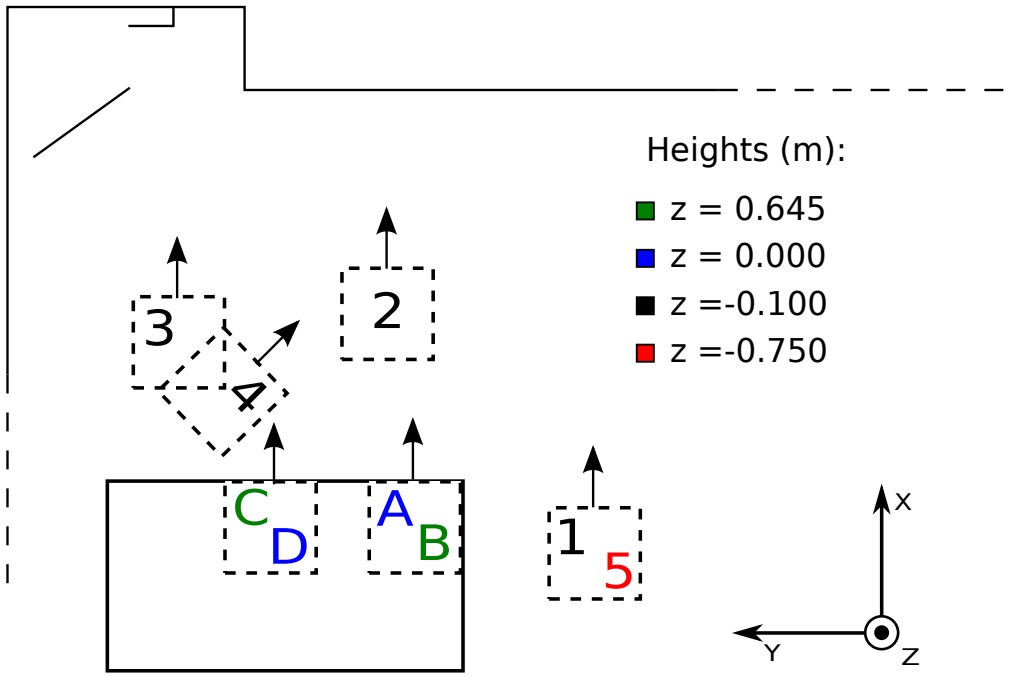

Figure 5.1: Different positions of the drone during the validation

5.3 Triangulation

Our first problem is where to put a landmark in the 3D world from two observations at two different keyframes. If all measurements were perfect, we could simply draw a ray at each keyframe that goes from the camera center and passes through the keypoint in the image plane, and those two rays would intersect at the position of the landmark.
Unfortunately, those lines never intersect in practice, due to various errors (measurement errors,

errors in the model of the cameras, errors on the position estimation of the cameras), so we need to find a method to find the best possible location of a 3D point from the pair of images.

5.3.1 Midpoint Method

The simplest and most obvious solution is to take the midpoint of the common perpendicular of the two rays. This method is intuitive to understand geometrically, and is quite easy to compute. In practice, however its results are not very good, as there is no theoretical reason for this point to be the best.

5.3.2 Linear least squares

A more advanced method for finding the coordinates of a point from a correspondence is called Linear Least squares, and is described in Hartley & Zisserman's *Multiple view geometry in computer vision* [7]. This method also has the advantage that it is already implemented in the OpenCV library.

5.3.3 Optimal Correction

If we assume that the error on observed points is random and follows a Gaussian distribution with zero mean, then the optimal solution would be to displace the pixels on both images until the resulting rays meet, keeping the displacement of the pixels as small as possible in the least squared sense. Such a solution would give the maximum likelihood estimator of the position of the 3D points, under those assumptions. There are several algorithms in the litterature that triangulate the position of a point using optimal correction. The most popular one, proposed by Hartley and Sturm [5], computes the solution directly but requires finding the root of a 6th degree polynomial. Kanantani et. al.'s method [**kanatani**] finds a solution iteratively, but requires very few iterations to have an accurate solution, and in practice, is faster than the Hartley Sturm method. It also has better numerical properties, as unlike the Hartley-Sturm method, it does not have singularities at the epipoles.

5.3.4 Comparison of triangulation methods

Using the evaluation procedure described in section 5.2, we can compare these 2 triangulation methods.

Table 5.1: Comparison between triangulation methods

	Accuracy		Robustness		Computation
	Distance (m)	Angle (rad)	Distance (m)	Angle (rad)	Time (s)
Midpoint Method	0.122	0.052	0.845	0.270	0.012
Optimal Correction	0.099	0.041	0.776	0.219	0.024

The computations were timed during the accuracy test, but these timings should be the same for the robustness tests. We can see that the optimal correction method is more performant, at the cost of being computationally heavier. We will see, however, that this computation time is negligible in comparison with the time taken for bundle adjustment. Because the time difference is not significant, and because optimal correction is better motivated theoretically while also giving better results in practice, we will use the optimal correction method to triangulate points. The performance increase from optimal correction is not very big however, because the biggest source of error is not gaussian noise on the measurements of the points, but errors in the position estimate of the cameras at the moment of creating the keyframes (especially in the robustness test).

5.4 Bundle Adjustment

Having a bad estimation of the keyframe's pose is problematic, as it will result in badly located landmarks, which in turn will cause a bad estimation of the camera's position when future keyframes are created. In the long term, errors will accumulate, and the map will be completely distorted. Luckily, if we have enough point correspondences between two images, it is possible to deduce the relative displacement between the two images. This means that from a set of images, we can reconstruct a scene, without even needing a prior estimation of the position of the cameras that took the images. This is good news as it means that the images can give us some absolute information about the scene, that can be used to cerrect the errors on the camera poses of the previous keyframes. The problem of adjusting camera poses and 3D point locations in order to minimize the reprojection errors of the 3D points onto the image planes is known as bundle adjustment. As stated above, bundle adjustment has the advantage of being absolute with respect to the world, and so not having errors accumulate. Another advantage of bundle adjustment, is that it can easily take into account points that are seen by more than two cameras, which is not trivial for the triangulation techniqhes described above. The main disadvantage of bundle adjustment is that it is computationally heavy, so it is important to adapt it to be useable un real time.
To show the advantages of bundle adjustment, we compare it with triangulation using the optimal correction method. When using bundle adjustment, optimal correction triangulation is also used to obtain an initial solution, from which we optimize.

5.4.1 How it works

5.4.2 Performance

Table 5.2: Performance of Bundle Adjustment

	Accuracy			Robustness		
	Distance (m)	Angle (rad)	Time (s)	Distance (m)	Angle (rad)	Time (s)
No Bundle Adjustment	0.099	0.041	—	0.776	0.219	—
With Bundle Adjustment	0.090	0.030	15.703	1.374	0.492	13.802

We see that in the accuracy test, bundle adjustment gives a significant improvement to the performance of the pose estimation. In the robusteness test, however, bundle adjustment makes the results quite worse. This is surprising, as it is just in the case where the pose of the camera is uncertain that bundle adjustment should improve de result, by adjusting the pose of the camera. But because the problem is quite non linear, it is possible to fall in local minima. We will later see that by tuning the bundle adjustment phase, we can make it perform much better, and actually improve the results in the robustness test. The other big problem of bundle adjustment is the time it takes. In both tests, this time was over 13 s, which is a prohibitively large amount of time for real time applications. In addition to improving the performance on the robustness test, we will try to recude the time taken by the bundle adjustment.

5.4.3 Tuning the Bundle Adjustment

The main drawback of Bundle Adjustment is that it requires an iterative method to be solved and can take a lot of time, which of course is a limiting factor for a robot that builds a map in real time. Therefore it is important to optimize both the speed of the computations, and their accuracy. As is often the case, there will have to be a tradeoff between these two. To measure both the speed of the Bundle Adjustment and the accuracy of the map obtained, we will conduct some experiments using different parameters to initialize the map.

Convergence of the solver

The first element we can tune is the convergence criterion of the solver that solves the bundle adjustment problem. We will stop when the ratio of the change in the objective function to the value of this function arrives below some threshold. This ensures that the criterion scales with the problem, which is important as the size of the problem can vary during operation (as the map grows, for example). The default value of the Ceres solver is 10^{-6}, but experimentally, we find that we can use a softer threshold to significantly speed up the computations, without impactig the quality of the results too much (and even improving them in the robustness test).

Table 5.3: Effect of convergence criterion on Bundle Adjustment speed and performance

Convergence	Accuracy			Robustness		
Threshold	Distance (m)	Angle (rad)	Time (s)	Distance (m)	Angle (rad)	Time (s)
1.000×10^{1}	0.099	0.041	0.529	0.776	0.219	0.527
5	0.099	0.041	0.533	0.776	0.219	0.560
1	0.099	0.041	0.537	0.776	0.219	0.529
5×10^{-1}	0.099	0.041	0.532	0.284	0.086	0.615
1×10^{-1}	0.116	0.037	0.671	0.244	0.098	1.176
5×10^{-2}	0.068	0.016	0.743	0.291	0.113	2.117
1×10^{-2}	0.090	0.032	1.421	0.823	0.316	2.770
5×10^{-3}	0.091	0.031	1.533	0.142	0.050	4.186
1×10^{-3}	0.085	0.023	3.620	1.035	0.397	4.907
1×10^{-4}	0.084	0.026	5.058	1.362	0.480	10.484
1×10^{-5}	0.090	0.030	12.453	1.483	0.528	13.206
1×10^{-6}	0.090	0.030	15.703	1.374	0.492	13.802
1×10^{-7}	0.088	0.028	15.795	1.559	0.567	12.935

As we can observe on table 5.3, the more severe the convergence threshold, the better the performance on the accuracy test. On the robusteness test, however, this is true for high convergence thresholds, but when the convergence threshold is lower than 0.01, the performance starts to degrade as the convergence threshold becomes lower. In both the accuracy and robustness tests, the lower the threshold, the longer it takes. We will chose a threshold of 0.05, as it is a nice compromize of performance on the accuracy test, performance on the robustness test, and speed.

Rejecting outliers

When looking closer at the results of bundle adjustment, we find that a large part of the errors come from a small number of points. One possible explanation is that these points do not correspond to real world points, so that even if we have the correct position of all cameras exatly, the rays corresponding to these points will not intersect, or even be close to intersection. For this reason, after every bundle adjustment pass, we could look at the contribution of every mapped point to the objective function, and eliminate points whose error is too high. Before doing this we have to take some things into account that as a point is observed by an increasingly large number of cameras:

- Its total contribution to the cost function will increase, as each observtion of a point adds a term to the objective funtion
- Its contribution per observation to the cost function will also increase, because every time an observation is added, the location of the point will change a bit, and its position will become less optimal if we only consider the cameras that already saw the point before.

For these reasons, we will put a different threshold on the points depending on the number of cameras that see the point. If a point is seen by a large number of cameras, we will allow it to

have larger errors in the objective function before removing it.

5.5 Map Initialization

A robot needs a map lo localize itself within it, but it needs to know its position to find the postion of surrounding objects and build a map. Because the mapping and localization tasks are mutually dependent on one another, there needs to be a special procedure to build a map from nothing when it does not exist yet. Arbitrarily, we decide that the position of the drone when it starts flying is the origin (in all 6 degrees of freedom) of the map. However, with only a monocular camera, it is not possible to find the exact location of any visual features from one observation only, views from at least two different positions are needed to triangulate points.

To reach the position from which the drone will take a second view and triangulate points, the drone has to fly blindly. Blindly here means without using a map la localize itself visually, but the drone can still use its other sensors (IMU and ultrasonic sensor) to obtain an estimate of its postion. Because the ultrasonic sensor is much more accurate than the IMU, and gives an absolute measure, we will mostly rely on this sensor to estimate the relative position from where we take the second view. Because the ultrasonic sensor only gives the distance from the bottom of the drone to the ground, the drone should fly straight up from its first position (the origin) to reach its second position.
Once in its second position, the drone can match seen keypoints from both views, and from its estimated position, triangulate those points to the map. However, the drone's estimation of its pose is prone to errors, especially as it only used its ultrasonic sensor and IMU. There errors can be corrected with the information from the cameras, because of we have matching observations, we can compute the fundamental matrix, and know exactly the displacement between the two views. Refining the poses of the views and the location of the landmarks simultaneously such as to minimize the reprojection error is a nonlinear optimization problem known as Bundle Adjustment. Using Bundle adjustment, we can refine the position of the second view, and use the ultrasonic sensor information to fix the scale. Another advantage of Bundle Adjustment is that it allows to take into account more than 2 views of a point.

Chapter 6

Simultaneous Localization and Mapping

Chapter 7

Conclusion

Bibliography

[1] Chris Anderson. “Agricultural Drones”. In: *MIT Technology Review* vol. 117 | no. 3 (2014).

[2] “Bâtir avec des drones ?” In: *https://uclouvain.be/fr/sciencetoday/actualites/batir-avec-des-drones.html* (2016).

[3] Dario Brescianini and Raffaello D’Andrea. “Design, Modeling and Control of an Omni-Directional Aerial Vehicle”. In: *IEEE International Conference on Robotics and Automation* (2016).

[4] Marcus Chavers. “Consumer Drones By the Numbers in 2017 and Beyond”. In: *newsledge.com* (2017).

[5] Richard I. Hartley and Peter Sturm. “Triangulation”. In: *CVIU - Computer Vision and Image Understanding* Vol. 68, No. 2 (1997).

[6] Russell Naughton. *Remote Piloted Aerial Vehicles : An Anthology*. Tech. rep. Monash University, 2003.

[7] Andrew Zisserman Richard Hartley. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[8] Jimmy Stamp. “Unmanned Drones Have Been Around Since World War I”. In: *smithsonian.com* (2013).

[9] Richard Voyles. *Dexterous Hexrotor Research*. http://web.ics.purdue.edu/ rvoyles/research.Hexrotor.html.

Appendix A

Benchmark results

A.1 Effect of convergence criterion during bundle adjustment

Table A.1: Effect of convergence criterion on Bundle Adjustment speed and performance

Convergence	Accuracy			Robustness		
Threshold	Distance (m)	Angle (rad)	Time (s)	Distance (m)	Angle (rad)	Time (s)
1.000×10^{1}	0.099	0.041	0.529	0.776	0.219	0.527
5	0.099	0.041	0.533	0.776	0.219	0.560
1	0.099	0.041	0.537	0.776	0.219	0.529
5×10^{-1}	0.099	0.041	0.532	0.284	0.086	0.615
1×10^{-1}	0.116	0.037	0.671	0.244	0.098	1.176
5×10^{-2}	0.068	0.016	0.743	0.291	0.113	2.117
1×10^{-2}	0.090	0.032	1.421	0.823	0.316	2.770
5×10^{-3}	0.091	0.031	1.533	0.142	0.050	4.186
1×10^{-3}	0.085	0.023	3.620	1.035	0.397	4.907
1×10^{-4}	0.084	0.026	5.058	1.362	0.480	10.484
1×10^{-5}	0.090	0.030	12.453	1.483	0.528	13.206
1×10^{-6}	0.090	0.030	15.703	1.374	0.492	13.802
1×10^{-7}	0.088	0.028	15.795	1.559	0.567	12.935

A.2 Effect of outlier threshold after bundle adjustment

A.2.1 Threshold after first bundle adjustment

Table A.2: Effect of removing outliers (convergence tolerance = 0.05)

Outlier	Accuracy			Robustness		
Threshold	Distance (m)	Angle (rad)	Time (s)	Distance (m)	Angle (rad)	Time (s)
3	0.083	0.025	1.039	0.214	0.061	1.705
4	0.074	0.021	1.018	0.168	0.061	1.792
5	0.075	0.021	0.941	0.127	0.046	1.849
6	0.079	0.023	0.967	0.135	0.048	2.034
7	0.079	0.022	1.006	0.134	0.048	1.979
8	0.079	0.022	0.960	0.137	0.050	2.108
9	0.078	0.023	0.975	0.187	0.070	1.747

Table A.3: Effect of removing outliers (convergence tolerance = 0.005)

Outlier Threshold	Accuracy			Robustness		
	Distance (m)	Angle (rad)	Time (s)	Distance (m)	Angle (rad)	Time (s)
3	0.085	0.026	2.232	0.370	0.137	3.597
4	0.084	0.026	2.158	0.347	0.122	3.494
5	0.086	0.026	2.048	0.319	0.113	4.200
6	0.086	0.027	2.087	0.309	0.109	3.295
7	0.086	0.026	2.076	0.202	0.072	3.390
8	0.086	0.026	2.147	0.185	0.065	4.522
9	0.085	0.028	2.066	0.213	0.076	3.578

A.2.2 Threshold after second bundle adjustment

Table A.4: Effect of removing outliers (convergence tolerance = 0.05)

Outlier Threshold		Accuracy			Robustness		
1^{st} b. a.	2^{nd} b. a.	Distance (m)	Angle (rad)	Time (s)	Distance (m)	Angle (rad)	Time (s)
5	4	0.074	0.025	1.059	0.133	0.049	1.892
5	5	0.078	0.026	1.046	0.122	0.044	1.866
5	7	0.072	0.023	1.068	0.128	0.046	1.928
5	1.000×10^1	0.074	0.023	1.063	0.136	0.049	1.822
5	1.500×10^1	0.071	0.023	1.095	0.136	0.048	1.900
6	4	0.078	0.025	1.090	0.124	0.046	1.955
6	5	0.072	0.023	1.098	0.127	0.047	1.960
6	7	0.072	0.022	1.067	0.123	0.045	1.930
6	1.000×10^1	0.073	0.022	1.082	0.121	0.045	1.944
6	1.500×10^1	0.073	0.023	1.094	0.121	0.045	1.933
7	4	0.076	0.024	1.090	0.123	0.045	1.915
7	5	0.074	0.023	1.071	0.129	0.048	1.913
7	7	0.073	0.022	1.111	0.124	0.046	1.950
7	1.000×10^1	0.071	0.021	1.114	0.121	0.045	1.923
7	1.500×10^1	0.073	0.022	1.073	0.121	0.045	1.959

A.3 Effect of limiting the number of matches between keyframes

Table A.5: Effect of limiting the number of matches between keyframes

Matches limit	Accuracy			Robustness		
	Distance (m)	Angle (rad)	Time (s)	Distance (m)	Angle (rad)	Time (s)
2.500×10^1	0.070	0.020	0.293	0.177	0.074	0.419
5.000×10^1	0.082	0.029	0.429	0.173	0.061	0.755
7.500×10^1	0.097	0.034	0.608	0.176	0.063	0.938
1.000×10^2	0.113	0.038	0.732	0.175	0.059	1.167
1.500×10^2	0.090	0.030	0.849	0.110	0.038	1.705
2.000×10^2	0.074	0.023	0.859	0.112	0.035	1.579
2.500×10^2	0.073	0.021	0.919	0.112	0.040	1.894
3.000×10^2	0.073	0.022	1.085	0.121	0.045	2.018

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**