Contents

1 Mapping **1**

1.1 Evaluation procedure . . . 1

1.1.1 Experimental setup . . . 2

1.2 Triangulation . . . 2

1.2.1 Midpoint Method . . . 3

1.2.2 Optimal Correction . . . 3

1.2.3 Comparison of triangulation methods . . . 3

1.3 Bundle Adjustment . . . 3

1.4 Tuning the Bundle Adjustment . . . 4

1.4.1 Convergence of the solver . . . 4

1.4.2 Rejecting outliers . . . 5

1.5 Map Initialization . . . 5

A Benchmark results **7**

A.1 Effect of convergence criterion during bundle adjustment . . . 7

A.2 Effect of outlier threshold after bundle adjustment . . . 7

A.2.1 Threshold after first bundle adjustment . . . 7

A.2.2 Threshold after second bundle adjustment . . . 8

A.3 Effect of limiting the number of matches between keyframes . . . 8

Chapter 1

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.

1.1 Evaluation procedure

The evaluation will happen in two phases: first the drone will initialize its map and then 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 1.1. During the initialization 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 place in a know position on a table (position A in figure 1.1). There, the drone is turned on, and it begins initializing its map. The drone is then successively places in 3 other known locations (B, then C, then D), and at each of these locations, the drone in manually communicated its position. Every time this happens, the drone adds landmarks into the map, and optionally updates existing landmarks' position, or even removes some landmarks. In reality, the drone would not know its exact position when taking views 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 second phase of the evaluation, 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 postition estimation, and the time taken to compute the map.

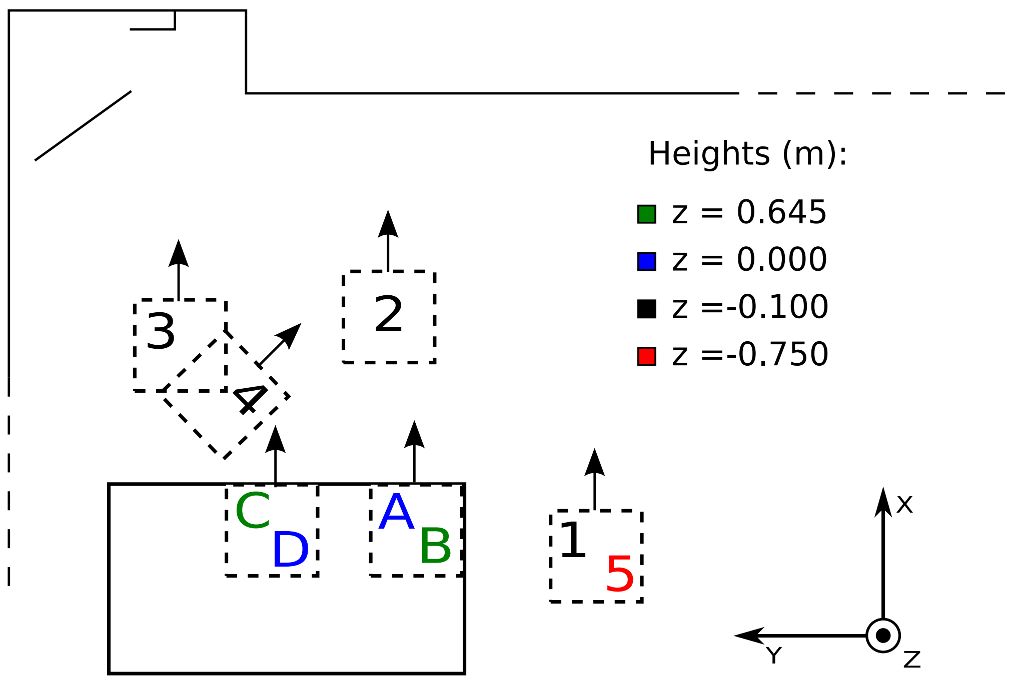

Figure 1.1: Different positions of the drone during the validation

1.1.1 Experimental setup

We will conduct two different finds of experiments: one to measure the accuracy, and one to measure the robustness of the mapping method. In both cases, we begin by placing the drone at the corner of a desk (position A on figure 1.1). After the drone has saved its view, it is placed on a stool exactly above its first position. Its exact pose is then manually communicated to the drone to simulate the measurements from its IMU and ultrasonic sensor. The drone then again takes a snapshot of its camera input, and matches points with the first view, and then adjusts its second pose and the position of the points through bundle adjustment. The drone and stool are then moved to the left (position B on fiugre 1.1), and again manually given its exact position, after which it takes another snapshot, matches points with the first two views, and readjusts the poses of the keyframes anf the positions of the points through bundle adjustment. This entire process is rpeated a third time when the drone is placed on the desk again at position D. The initialization of the map consists in making these four keyframes and adjusting their position and that of the landmarks 3 times. After this initialization is done, we place the drone at different locations, and measure how close it is to where it thinks it is. We evaluate the initialization based on how accurate its position estimation is, as well as on how much time the successive bundle adjustments took.

1.2 Triangulation

In the problem of triangulation, we try to find the 3D coordinates of a point from the 2D coordinates of the projection of this point on two images that were seen from different positions. We assume that the position of the camera taking these images is known exactly at both locations. If the camera positions is known exactly, and if the projection of the points into the image planes was perfect, then the two rays going from the camera centers, and through the images of the points would intersect at the location of the 3D point. In practice however, these two rays never cross exactly, so a method has to be found to find the best possible location of a 3D point from the pair of images.

1.2.1 Midpoint Method

The simplest solution would be 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. The optiaml 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 the assumption that the error of their projection on the image planes follows gaussian noise.

1.2.2 Optimal Correction

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 [1], 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 Strum method. It also has better numerical properties, as unlike the Hartley-Sturm method, it does not have singularities at the epipoles.

1.2.3 Comparison of triangulation methods

Using the evaluation procedure described in section 1.1, we can compare these 2 triangulation methods.

Table 1.1: 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. Therefore, we will use the optimal correction method to triangulate points.

1.3 Bundle Adjustment

Unfortunately the main source of errors when mapping points is not inaccuracy of the camera, but uncertainty on the camera's position. This is bad, as a bad estimation of the camera's position will result in badly located landmarks, which in turn will result in a bad estimation of the camera position. 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, not only relative to the drone. The problem of adjusting camera positions 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.

Table 1.2: 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.

1.4 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.

1.4.1 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 1.3: Effect of convergence criterion on Bundle Adjustment speed and performance

Convergence Threshold	Accuracy Distance (m)	Accuracy Angle (rad)	Accuracy Time (s)	Robustness Distance (m)	Robustness Angle (rad)	Robustness 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 1.3, 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.

1.4.2 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.

1.5 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.

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 Threshold	Accuracy Distance (m)	Accuracy Angle (rad)	Accuracy Time (s)	Robustness Distance (m)	Robustness Angle (rad)	Robustness 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 Threshold	Accuracy Distance (m)	Accuracy Angle (rad)	Accuracy Time (s)	Robustness Distance (m)	Robustness Angle (rad)	Robustness 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 Distance (m)	Accuracy Angle (rad)	Accuracy Time (s)	Robustness Distance (m)	Robustness Angle (rad)	Robustness 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 1^{st} b. a.	Outlier Threshold 2^{nd} b. a.	Accuracy Distance (m)	Accuracy Angle (rad)	Accuracy Time (s)	Robustness Distance (m)	Robustness Angle (rad)	Robustness 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 Distance (m)	Accuracy Angle (rad)	Accuracy Time (s)	Robustness Distance (m)	Robustness Angle (rad)	Robustness 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**