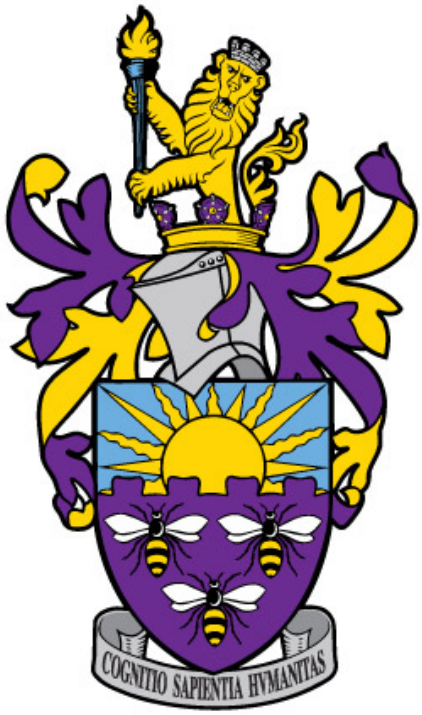
The University of Manchester



A study for autonomous indoor mapping with AR.Drone

Third year project report

May 2016

Author:
Ciprian Tomoiagă
BSc (Hons) Computer Science
with Industrial Experience

Supervisor:
Dr. Tim Morris

Abstract

Our project investigates the outstanding problem of obtaining a 3D map of an environment from a monocular camera mounted on a quadcopter.

In our work we use the ORB-SLAM, one of the best feature-based SLAM algorithms for building the map and estimating the pose of the system. These are then fed into two separate sub-systems dealing with autonomy and reconstruction. The autonomy driver takes care of acquiring an initial scale and driving the quadcopter to the next goal through a PID controller. The reconstruction module uses a Random Sample Consensus method to simplify the 3D model into a collection of planes. Implementation and testing has been carried out with Parrot's AR.Drone 2.0, a commercially available and affordable quadcopter.

The future work is suggested based on observed challenges and limitations of the system, such as goal acquisition and generating visually rich maps.

Acknowledgements

First, I would like to thank my supervisor, Tim Morris, whose support and advice was invaluable for this work.

I would also like to thank my family for their continuous support, especially during my studies.

Contents

List of Figures

Introduction

This report supports the associated third year project of autonomous indoor mapping with the AR.Drone. It investigates and describes different approaches for solving the critical parts of autonomous navigation through an unknown environment. It also presents our experiments and argues our choices when implementing such a system on an affordable *Micro Aerial Vehicle* (MAV).

The rest of this chapter presents the context of the problem, including motivation, aims and limitations. Chapter 2 is dedicated to the biggest body of work and research, the problem of Simultaneous Localisation and Mapping. It presents some of the early approaches and builds up to the most recent ones. The rest of our work consists of autonomy and reconstruction modules and is described in chapter 3. It also presents the overview of our system. We explain our evaluation methods in chapter 4 and conclude with chapter 5 by listing our achievements and future work.

1.1 Motivating examples

Autonomous robots, especially flying ones, can be used to accomplish a wide variety of tasks. Some notable examples which are usually unsuitable for humans or can be automated are:

- inspection of tunnels such as the Large Hadron Collider (TeraRanger)
- inspection of planes for lightning strikes (BBC)
- exploration of caves and natural mazes; these can be deep inside mountains or under oceans here on Earth or, why not, on remote planets or asteroids (Mars Curiosity Rover [Cheng et al. 2005])
- search for victims and survivors after disasters such as Fukushima or April 2015 Nepal earthquake
- spying of terrorist settlements by intelligence agencies (with tools such as Nano Hummingbird by AVInc & DARPA)

To some degree, these all represent instances of the same problem, namely navigating through a maze. To accomplish such goals, the agent (human or robot) needs to have a high level understanding of the environment and a good strategy for coverage and path planning. Moreover, in real life these environments are usually dynamic so mapping them poses a challenge even to humans.

1.2 The system and its restrictions

To implement and test our approach we used the AR.Drone 2.0, an affordable quadcopter from Parrot. This is a closed source architecture which exposes a public *Application Programming Interface* (API) to send data from its sensors (camera, inertial measurement

unit, altimeter, magnetometer etc.) and to receive commands for navigation. Therefore our programs have to run on a ground station and interact with the drone via a wireless connection. In addition, next we present the two most solid reasons that defined our challenge and influenced all our choices.

1.2.1 No global position

Although our drone provides readings of the Global Positioning System (GPS), this is usually unavailable in interior spaces. Moreover, even if it were available, its precision is in the range of several meters which makes is unsuitable for our tasks. Another method for providing a global position is through a marker-based triangulation system which uses distinctive visual cues attached to the drone and a setup of 3 cameras for tracking. While such a system is highly accurate, it is restricted to the controlled environment of the laboratory.

As a result, it is necessary to rely solely on information provided by the *local* sensors to infer and track the position of the drone.

1.2.2 No direct depth measure

The choice of sensors available further limits the capabilities of reading the environment. The Parrot AR.Drone provides two RGB cameras, one pointing forward and another pointing downward, but no depth sensors such as range finders or RGB-D camera. In addition, due to limited battery power, it allows no extra payload so customisations are infeasible.

Therefore, the only option is to infer depth from *visually detected* movement. Since we want exploration of unknown environments and only one of the two cameras is active at a time, it is obvious that the forward facing camera is better suited for the task.

1.3 Aims

Given the big challenge domain of possible applications and the imposed restrictions, our aims are mostly *exploratory* in nature. These are, in order of their priority:

1. to research and understand the problem of monocular SLAM;
2. to test and compare existing implementations and choose the most suitable one;
3. to transform and save the generated map for further processing;
4. to develop a mean of autonomy on top of the chosen SLAM;
5. to give a visually rich rendering of the saved map .

During our first planning session we established it was mandatory to get item 3, which includes the previous ones. This would provide a starting point for other projects wishing to extend our work later on. In addition, item 4 is a desirable goal to have a complete system. Finally, we estimated that item 5 is a body of work too big for the given time frame when combined with the previous tasks, so it was marked as an extension.

Simultaneous Localisation And Mapping

> Using sensory information to locate the robot in its environment is the most fundamental problem to provide a mobile robot with autonomous capabilities.
>
> Ingemar J Cox, 1991 [10]

This task, now known as SLAM, was first introduced by Smith and Cheeseman (1986) and then formalised as a whole concept by Durrant-Whyte and Bailey [14, 3]. It refers to the process through which a robot builds a map of its environment and keeps track of its position within this map. Both of these are regularly updated based on new observations of landmarks in the environment so the precision and the accuracy of taking these have a direct influence on the result. In general, this is a very large class of algorithms and, while close to perfect solutions exist, they rely on carefully tailored systems. In real life scenarios however, the systems are fixed and the algorithms have to fit to them.

The first solutions, including that of Smith and Cheeseman, were based on an Extended Kalman Filter (EKF). Besides high computational complexity, their main problem was inconsistency due to the linearisation of an unliniar problem, as shown by Castellanos et al. (2004). A later approach, the FastSLAM algorithm and its enhancement FastSLAM 2.0, by Montemerlo et al. (Montemerlo et al., 2002, 2003), brings great improvement by taking a Bayesian point of view to the problem through the use of a modified particle filter. In essence, their method uses the independence between the robot's paths and the individual landmark measurements in order to decompose the SLAM problem. Moreover, the second variant brings, for the first time, a proof of convergence for the linear SLAM problem with a single particle.

Another important challenge in this topic is the *kidnapped robot problem* where the robot has to determine its correct position in the map after being kidnapped from one location and woken up at another one, usually unknown. The wake-up robot problem is an instance of this, when the robot is told that it was kidnapped; it represents the initialisation state for most systems. Solving this challenge depends on the type of sensors used as it involves recognising or discarding previously registered locations with minimum number of false positives.

The rest of the section focuses on the *visual* SLAM challenge and algorithms, where the only information is coming from cameras.

2.1 Visual (and monocular) SLAM

Due to the restrictions, our setup relies solely on visual information to solve the SLAM problem. This section provides the basics of this task, detailing the pinhole camera model along with the mathematics of the rigid body motion. Then it builds on to explain the visual odometry, its weaknesses, and how a SLAM system solves them. Lastly, it compares and contrasts between two approaches of modern SLAM: appearance-based versus feature-based and mentions two mixed approaches.

2.1.1 The camera

Cameras are ubiquitous these days especially on robots and micro aerial vehicles because they:

- are the richest source of information from the environment
- are passive sensors, immune to interference
- have a very low footprint in terms of size, weight and power
- are low-cost, and easy to use and calibrate

In order to abstract them to a mathematical representation, most systems use the *pinhole model*, shown in figure 2.1. This is also known as the *perspective model* as it describes the mathematical projection from 3D world coordinates to 2D image coordinates. It is implemented as a *projection matrix* and comprises both the *world* $\rightarrow$ *camera* transformation ($\mathbb{R}^3 \rightarrow \mathbb{R}^3$) and *camera* $\rightarrow$ *image* projection ($\mathbb{R}^3 \rightarrow \mathbb{R}^2$). The matrix provides an association between points in camera's image space and locations in 3D world space.

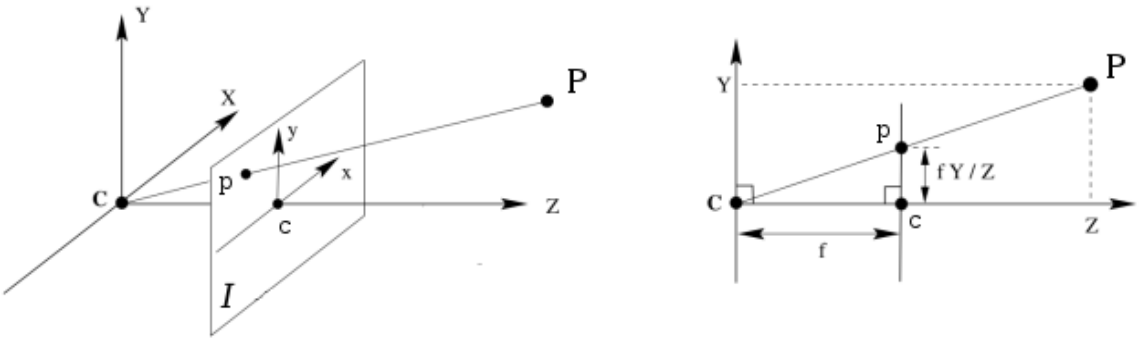


Figure 2.1: In the perspective camera model the point P in camera coordinates is mapped to a pixel p on the image plane I by the ray CP where C is the *Centre Of Projection (COP)*. The perpendicular from C to the plane I is called the *principal axis*, with the intersection point $c = (c_x, c_y)^\intercal$ known as the *principal point* or *optical centre*. The distance f is the camera's focal length.

The first mapping, *world* $\rightarrow$ *camera*, uses the extrinsic parameters which define the location and orientation of the camera origin with regards to the world frame. For a point P in world coordinates, it is described as:

$$P_{camera} = RP_{world} + T,$$

where $R \in \mathbb{R}^{3\times3}$ is the rotation matrix denoting the camera's heading and $T \in \mathbb{R}$ is the translation vector of the world's origin with regards to the camera's coordinate system.

The second mapping, *camera* $\rightarrow$ *image*, uses the intrinsic parameters to derive the projection of the 3D point $P_{camera} = (X, Y, Z)^\intercal$. By the means of similar triangles (see figure 2.1), its location (x, y) is expressed as:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z} \begin{pmatrix} X \\ Y \end{pmatrix}.$$

Since the origin of the image plane I may not be at its principal point c but at the lower or upper left corner of the image, the mapping $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ from camera coordinates to image coordinates will account for this:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \pi(X, Y, Z) := \frac{f}{Z} \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix}. \tag{2.1}$$

Passing everything to homogeneous coordinates allows us to use matrix multiplication for describing the whole projection process from world coordinates to image coordinates:

$$w \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} R|T \end{bmatrix}}_{M} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{2.2}$$

with K being the camera *calibration matrix* that combines the intrinsic parameters, and M combining the extrinsic ones.

It is important to note that in both forms (Equation 2.1 and Equation 2.2) the depth Z is lost, so the inverse mapping which we are interested in, from image space to 3D camera (and then world) coordinates needs to know or approximate it:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \pi^{-1}(x, y, Z) := Z \begin{pmatrix} f^{-1}x - c_x \\ f^{-1}y - c_y \\ 1 \end{pmatrix}.$$

While the pinhole model is a good abstraction of a simple camera, it also has its drawbacks: it does not take into account that modern cameras use lenses to refract the rays and get a sharp projection. These lenses are not perfect and usually come with radial and tangential distortion (figure 2.2). The former manifests in form of "barrel" or "fish-eye" effect, and the latter occurs in the manufacturing process when the lenses are not perfectly parallel to the image plane. However, these can be modelled and corrected by introducing further nonlinear transformations from the distorted point (x, y) to the undistorted (x_u, y_u):

$$\begin{aligned} x_u &= x \cdot \Delta x_{radial} + \Delta x_{tangential} \\ y_u &= y \cdot \Delta y_{radial} + \Delta y_{tangential}, \end{aligned}$$

where

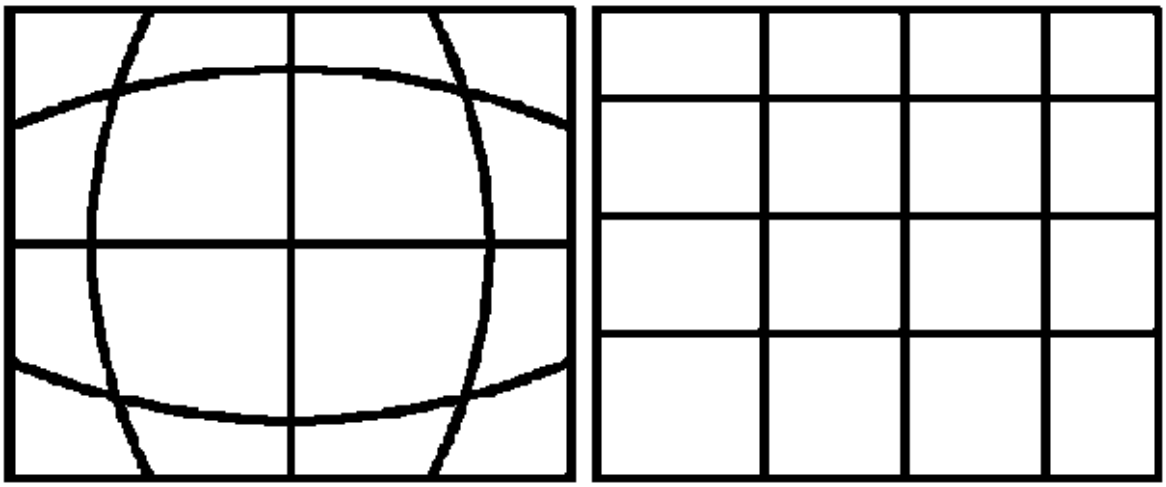
$$\begin{aligned} x_{radial} &= 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots \\ y_{radial} &= 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots \\ x_{tangential} &= 2p_1 xy + p_2(r^2 + 2x^2) \\ y_{tangential} &= 2p_2 xy + p_1(r^2 + 2y^2) \\ r &= x^2 + y^2 \end{aligned}$$

Usually only the first 3 parameters are taken for the radial distortion, which gives the 1-row distortion matrix $D = \begin{pmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{pmatrix}$.

Since the calibration of the camera doesn't change during operation, it can be calculated once and used many times. The calibration matrix K and the distortion matrix D are estimated by performing a least-squares minimisation on a known pattern of input, such as a checker board.

Figure 2.2: Lens distortions

A combination of these is present in most cameras



(a) Barrel (b) Tangential

2.1.2 Twist coordinates in rigid body motion

Any transformation in the Euclidean space will represent the displacement of a *rigid body* if it keeps invariant:

a) the distance between any two points

b) the orientation of the body (a right-handed coordinate system remains right-handed)

The family of such transformations describing how coordinates of every point X on the object change as a function of time can be denoted by a mapping g:

$$\begin{aligned} g(t) : \mathbb{R}^3 &\to \mathbb{R}^3 \\ X &\mapsto g(t)(X). \end{aligned}$$

In terms of vectors, this mapping has to preserve both the norm and the cross product of any two vectors. Because of item b), these transformations are also called the *special* Euclidean transformations and they form the group $SE(3)$.

A more physical interpretation of the definition is in terms of coordinate frames: a rigid body motion on a orthonormal coordinate frame preserves right-handed frames. Under this, the mapping g becomes:

$$\begin{aligned} g : \mathbb{R}^3 &\to \mathbb{R}^3; \\ g(X) &= RX + t. \end{aligned}$$

In homogeneous coordinates this can be expressed as a single transformation matrix $T \in \mathbb{R}^{4\times4}$, comprising both the rotation R and the translation t:

$$g \begin{pmatrix} x \\ 1 \end{pmatrix} = Tx = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix}. \tag{2.3}$$

Its advantage is that multiple transformations can be concatenated through the left multiplication of their respective matrices:

$$T_i = T_{i-1}T_{i-2}...T_1T_0.$$

However, a major drawback is that the usage of a 3×3 rotation matrix makes for an over-parametrised system of nine parameters and only 3 degrees of freedom. This sets a challenge bigger than necessary for numerical minimisation problems. In order to reduce the computational effort, we note that $SE(3)$ is a Lie Group and we use its associated Lie Algebra $se(3)$. In this context, the transformation T can be represented by a *twist* vector $\xi = (\nu, \omega)^\intercal \in se(3)$, which represents it as as an angular velocity ω and a translational velocity ν. These have 3 coordinates each (one for each dimension) so the system overall has 6 parameters for 6 degrees of freedom, forming a minimal representation of a rigid body motion.

2.1.3 Visual odometry

In robotics, odometry is the process of estimating change in position over time, based on data from motion sensors such as wheel rotary encoders. *Visual* odometry is a sub case of the structure from motion problems, dealing only with estimation of camera poses. The term was introduced by the work of Nistér et al. (2004) which presents a system for real-time ego-motion estimation over a long trajectory. Similarly to wheel odometry, it makes incremental estimates of the pose by examining motion-induced changes on the images of on-board cameras. However, while being robust to adverse conditions such as wheel slip in uneven terrain, it needs sufficient illumination, texture and scene overlap between consecutive images. Scaramuzza and Fraundorfer (2011) have shown that when these conditions are fulfilled, "visual odometry provides more accurate trajectory estimates than wheel odometry, with relative position error ranging from 0.1 to 2%", which makes it ideal in GPS-denied environments.

Building up from the previous 2 parts, we use a rigid body transformation $T_{i,i-1} \in SE(3)$ to describe the camera motion observed in consecutive images taken at discrete time steps i:

$$T_{i,i-1} = \begin{bmatrix} R_{i,i-1} & t_{i,i-1} \\ 0 & 1 \end{bmatrix}.$$

Given an initial pose C_0, usually set by the user, and the set of transformations between consecutive frames $T_{1:n} = T_{1,0}, \ldots, T_{n,n-1}$, we can estimate the current camera pose C_i with regards to a global coordinate system through concatenation:

$$C_i = T_{i,i-1} \cdot T_{i-1,i-2} \cdots T_{1,0} = C_{i-1}T_{i,i-1}.$$

Each of the transformations $T_{i,i-1}$ has an uncertainty associated with it. Therefore, the camera pose C_i depends on the uncertainty of past transformations. As the algorithm progresses, the error on previous pose estimates is accumulated which leads to drift over time, causing the real path and the estimated trajectory to differ.

2.1.4 Correcting drift

Visual odometry is used as a building block of visual SLAM, but due to accumulation of uncertainty and errors, it suffers from trajectory drift over time. A true SLAM solution

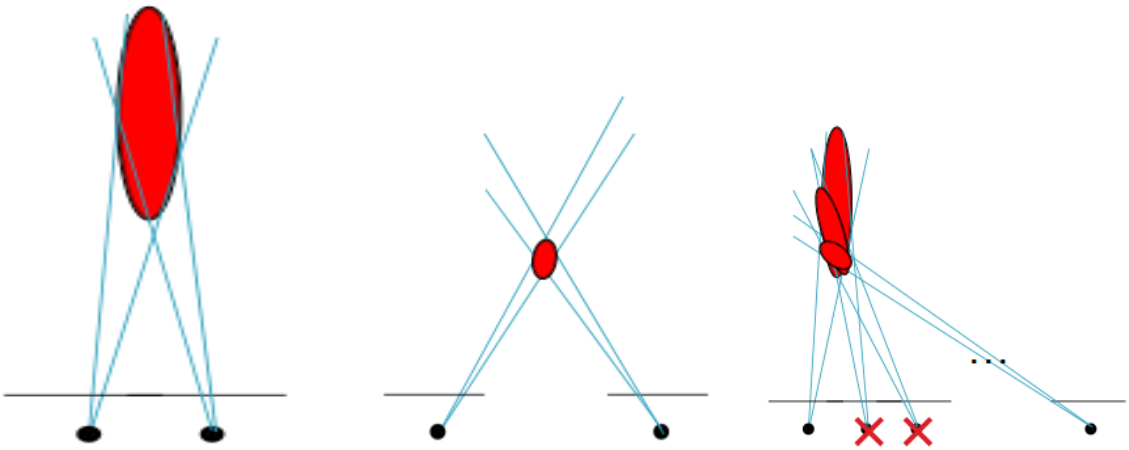
adds different optimisations to VO in order to build a global and consistent map; these include bundle adjustment, pose-graph optimisations with loop closure and fusion with other sensors.

Bundle adjustment is an iterative refinement to obtain a more accurate map of the local trajectory. It is a *windowed* algorithm as it tries to minimise an error function over the last m frames. This is calculated as the image reprojection error:

$$\underset{X^k, C_i}{\operatorname{argmin}} \sum_{k,i} \|p_i^k - g(X^k, C_i)\|^2, \tag{2.4}$$

where p_i^k is the point corresponding to landmark X^k in image i and $g(X^k, C_i)$ is the reprojection of the same landmark according to the current camera pose C_i. The choice of window size m depends on the computational capabilities of the system since the reprojection error is a nonlinear function requiring an expensive algorithm such as Levenberg-Marquardt. A small window limits the number of parameters making bundle-adjustment tractable in real-time.

A very important observation is that the rays reprojecting two observed instances (p_i^k, p_j^k) of the same landmark X^k from the poses (C_i, C_j) will *never* truly intersect due to noise, errors in camera calibration and feature matching uncertainty. Therefore, the estimated 3D position of the point will be the one closest to all intersecting rays in a least-squares sense. The *quality* of a 3D point can be defined based on the standard deviation of the distances from all the rays. Points of low quality appear when frames are taken at nearby intervals compared to the distance (figure 2.3 (a) vs. figure 2.3 (b)). These can be avoided by a *keyframe* selection step which skips frames until the average uncertainty decreases below a certain threshold (figure 2.3 (c)).



(a) (b) (c)

Figure 2.3: Uncertainty of landmark reprojection
The red area indicates possible locations for a landmark observed from different points. Observations which undergo significant motion (b) yield more accurate results than those which are close to each other (a); high-uncertainty observations should be discarded (c). (Image from Scaramuzza)

If additional transformations T_{ij} between non-consecutive time steps i and j are known, these can be used to obtain global optimisation. A graph is built with nodes

representing camera poses and edges denoting the rigid body motion between them. The accumulated drift can then be corrected by imposing constraints e_{ij} over the edges and minimising a non-linear cost function:

$$\sum_{e_{ij}} \|C_i - T_{ij} C_j\|^2.$$

Loop closures have great contributions to the pose graph as they connect graph nodes that are far apart and between which large drift has accumulated. These occur when the agent re-observes a landmark or a place after not seeing it for a long time and are usually detected by doing a constraint search over a window of nearest frames.

Finally, trajectory drift can also be corrected by fusing data from other sensors with the visual odometry estimates and then filtering the result. These include readings from IMU, GPS and even laser scanners and are crucial to monocular systems in order to recover the absolute scale of the scene.

2.2 SLAM approaches

With the SLAM problem being so old, a number of different approaches have evolved over time. The introduction part of this chapter lists the very early work, all of which employed filtering approaches. A more complete history is given by Scaramuzza and Fraundorfer in their 2011 tutorial, which has already become outdated.

The first criterion for differentiating SLAM solutions is the system setup. Some use single camera [37, 47], but since this suffers from lack of scale, visual-inertial approaches emerged in order to fix it [45; 58; 16; 33]. Others prefer multi-camera setup, varying from stereo [52; 27] to trinocular [12; 53] and even more cameras [30; 51]. While these are all interesting approaches and bring something innovative to the problem, we will focus on monocular ones due to restrictions of the AR.Drone.

A second criterion for analysing a visual SLAM implementation is whether it is based on feature matching or on direct appearance of an image. The rest of the section will detail the basics of each of them, and investigate the most recent work in the field. The last part presents hybrid approaches which aim at combining the strengths of both sides.

When considering our choice, we have looked into three of the most popular algorithms, two feature-based (PTAM [31; 17], ORB-SLAM [41]) and one appearance-based (LSD-Slam [18]). We also considered the hybrid approach of SVO [20] but deemed it unsuitable as it uses downward oriented camera. Our investigations led to choosing ORB-SLAM as the most appropriate for our goals, being robust in initialisation and tracking and dealing well with large scale environments.

2.2.1 Appearance-based (direct) methods

Direct methods for estimation of motion and shape recover unknown parameters from measurable image quantities at each pixel of the image by minimising an error measure such as brightness or brightness-based cross-correlation (Irani and Anandan (1999)). They work on the assumption of the brightness constancy equation between two consecutive frames I_i, I_{i+1}:

$$I_i(p_i) = I_{i+1}(p_{i+1}), \tag{2.5}$$

where p_i and p_{i+1} are the projections of a world point $P_W = \begin{pmatrix} X & Y & Z \end{pmatrix}^\intercal$ in those frames; in other words, projections of the same point should have the same intensity in consecutive frames. When this holds, it leads to a huge number of constraints even for a relatively small camera resolution (e.g. 640 × 480 for VGA yields over 300,000 constraints).

In practice, however, Equation 2.5 is often violated due to sensor noise, lighting conditions and non-Lambertian surfaces. This means there is a non-zero residual

$$r = \|I_i(p_i) - I_{i+1}(p_{i+1})\|$$

for each pixel in the corrected image. This can be used to define the *photometric error* function by combining the errors for each pixel along with the camera motion ξ and the projection function $\pi(\xi, P)$ from Equation 2.1:

$$E = \sum_{p \in \Omega} \|I_i(p_i) - I_{i+1}(\pi(\xi, P))\|. \tag{2.6}$$

As seen in section 2.1.2, ξ presents the camera motion through twist parameters, forming a minimal system with 6 DoF. After linearising the error function E using a second-order approximation, a numerical minimisation algorithm can be employed to find the best solution. Moreover, this gives sub-pixel accuracy since the system is over-parametrised, with constraints from all the pixels and only the 6 elements of ξ to estimate.

Another optimisation step is to add confidence weighting to the local constraints depending on their gradient magnitude. This favours highly texturised areas of the image while giving less importance to the homogeneous parts. Also, to allow for motion steps bigger than one pixel, it is usual to apply a coarse-to-fine refinement, starting the estimation at a high level of an image pyramid and refining it on lower levels. The improvement makes it possible to track motions up to 15 percent of the image size (Irani and Anandan, 1999). However, it is important to note that this approach essentially ignores multiple or combined motions, locking to the dominant one. Depending on the needs, this could be a strength or a weakness.

Since the system of constraints is so large, the estimation step is usually very costly to be implemented on the CPU. *Dense Tracking and Mapping (DTAM)* by Newcombe et al. (2011) was one of the first approaches to make use of the GPU in order to solve this problem and has shown great performance in tracking and mapping, especially with regards to motion blur.

More recently, the work of Engel et al. (2014b) called *LSD-SLAM* (Large-Scale Direct monocular SLAM), has become the state-of-the-art in direct methods. We have thoroughly investigated it for our task, but it has proven very difficult to set up and configure for our system. Besides installation and running problems, we have experienced poor initialisation and frequent loss of tracking.

Its novelty relies in approximating the inverse depth map with a 2D array of random Gaussian values, which makes it *scale-aware*. This is done only on the *good* pixels of the image, their quality being defined by the direction of intensity gradient. New observations either become new keyframes or refine the current one through Gaussian merge of the depth map. The 3D environment map is stored in a topological graph of keyframes K_i with nodes $N_i = (I_i, D_i, V_i)$ and edges ϵ_{ji}, where I_i is the image of K_i, D_i is the inverse depth map and V_i is the variance of the depth map. The edges between keyframes j and

i contain their relative alignment as a *similarity* transform $\xi_{ji} \in sim(3)$ [1]. Global map optimisation is performed in the background with *g2o* framework (Grisetti et al., 2011). A summary of the update loop in LSD-SLAM is listed in algorithm 1.

2.2.2 Feature-based methods

As opposed to direct methods, this class of algorithms uses only a sparse set of points to estimate motion and perform reconstruction. Since there is a wide choice of feature detectors (see subsection 2.3.1) and matching works well, this approach is more robust and allows dealing with significant motion between frames. The detector choice is application dependent as (usually) there is a trade-off between speed and distinctive matching. Their biggest disadvantage is lack of visual richness in the resulting map, since it is built from the same set of sparse features.

Traditionally, there are two ways of working with feature points: tracking or matching. The former finds interesting points in one frame and then searches for them in the next frames using local techniques such as correlation. To limit the search space, the images should be taken from nearby views. Therefore it appears mostly in early work that focused on small environments. The latter approach independently detects features in all the images and then matches them against each other based on the descriptor's similarity metric. It is more suitable when a large viewpoint change is expected so it is common in recent work, focused on large environments (Fraundorfer and Scaramuzza (2012)).

The feature correspondences are stored in a map, ready to be used for localising newly tracked frames. For monocular systems, the features can be stored either in 2D, as detected, or in estimated 3D, after back projection; this results in two approaches for expressing the motion between previous feature vector f_{i-1} and current one f_i :

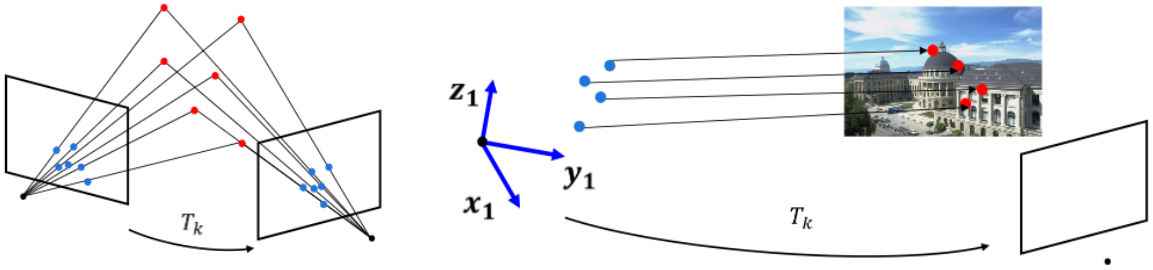
- 2D-to-2D: when *both* f_{i-1} and f_i are specified as 2D image coordinates (figure 2.4 (a)); the transformation is found by minimising the reprojection error (Equation 2.4) of the triangulated points in each image.

- 3D-to-2D: when f_{i-1} is specified as 3D coordinates and f_i is in 2D (figure 2.4 (b)); similarly to previous case, the transformation corresponds to the minimal reprojection error, but it also needs a preparation step to triangulate p_{i-1} from two adjacent camera views I_{i-2} and I_{i-1}. The error function for the transformation T_i then becomes:

$$T_i = \begin{bmatrix} R_{i,i-1} & t_{i,i-1} \\ 0 & 1 \end{bmatrix} = \underset{T_i}{\operatorname{argmin}} \sum_k \| p_i^k - p_{i-1}^k \|^2. \tag{2.7}$$

Finally, the algorithms should perform optimisations and corrections. Besides the regular ones for visual odometry (subsection 2.1.4), feature-based solutions can also employ a Random Sample Consensus (RANSAC) method to eliminate outliers in matched features. For 6 DoF, a minimum of 5 point correspondences is needed, as scale is an unobserved parameter. An efficient implementation was proposed by Nistér (2004).

[1] Analogous to $se(3)$ from section 2.1.2, $sim(3)$ is the Lie algebra of the group $Sim(3)$ in which the transformation T also includes the *scale*. Equation 2.3 becomes: $T = \begin{pmatrix} sR & t \\ 0 & 1 \end{pmatrix}$.



(a) 2D-to-2D (b) 3D-to-2D

Figure 2.4: Monocular motion estimation. (Image from Scaramuzza)

MonoSLAM (Davison et al. (2007)) is the first successful approach to use only monocular vision for SLAM. It tracks 100 features as *natural visual landmarks* and couples them with the local camera poses in a probabilistic map. Since its goal is re-localisation, the features are only selected once and then tracked for indefinite time, being updated with more observations through an Extended Kalman Filter.

Another approach that proved very robust and is still in use today is the Parallel Tracking and Mapping, PTAM (Klein and Murray (2007)). Essentially, it notes the independence of data association between tracking and mapping which allows splitting the work into separate threads. For tracking it uses fast, *every-frame* visual odometry with a robust estimator and coarse-to-fine optimisations. For mapping, it uses the batch technique of bundle adjustment, which is more accurate than incremental approaches but also more expensive to compute. However, it is tractable in real-time since it is decoupled from tracking and can be run only at key-frames, when there is sufficient motion to update the map.

In our work, we have investigated PTAM as part of the tum_ardrone[2] package (Engel et al. (2012a)), an open-source implementation which is very similar to ours. The package's strength relies in having an accurate and drift-free scale obtained by visual-inertial fusion of PTAM with an EKF. However, it cannot handle a large environment nor does it build a map of the exploration. Due to these limitations, we have opted to use the first open-source version of ORB_SLAM[3] package, which is detailed in section 2.3.

2.2.3 Hybrid methods

While having obvious potential advantages, the combination of direct and feature-based methods is theoretically difficult due to the computational effort involved in both. Forster et al. (2014) have proposed a *semi-direct* approach for downward oriented cameras on a MAV called SVO. Similarly to Klein and Murray (2007), it splits the work into two threads, one for constant-time tracking and the other for extending the map decoupled from real-time constraints. To get an accurate motion tracking it uses two steps from direct methods and a feature-based step for refinement:

1. find relative camera pose by minimising photometric error on hundreds of small patches (Equation 2.6);

[2] https://github.com/tum-vision/tum_ardrone
[3] https://github.com/raulmur/ORB_SLAM

2. align 2D feature patches with their correspondences in the keyframe with the closest observation angle of the same world point;

3. refine pose and structure by minimising 2D-to-2D and 3D-to-2D reprojection errors (Equation 2.4 and Equation 2.7)

The mapping thread uses a probability distribution to estimate the depth of 2D features with unobserved 3D correspondence. Subsequent observations of the same feature update its probability in a Bayesian manner up to a given threshold, after which the depth estimate is converted to a 3D point and used for motion tracking. To keep an even distribution of features representing the depth filters, the image is divided into fixed-size cells from which it selects the FAST corner with the highest Shi-Tomasi score.

While SVO gives very accurate tracking results at real-time frame-rates, it is not a complete SLAM algorithm as it does not build a global map. It addition, it struggles with forward motion and pure rotations due to its downward oriented design.

Krombach (2016) also proposes a semi-dense approach for visual stereo SLAM. It uses the direct monocular lsd_slam [4] and extends it with open-source package LIBVISO2 [5] . It recovers absolute scale from triangulation based on stereo feature-matching. It also corrects its drift via additional pose-graph constraints derived from feature-based odometry. This computation is done in-between the keyframes of LSD-SLAM and provides estimated priors for direct tracking.

2.3 ORB-SLAM

ORB-SLAM, proposed by Mur-Artal et al. (2015) is an extremely accurate, robust and efficient solution to the SLAM problem, from the feature-based category. Contrary to other SLAM implementations, it does not use an external odometry system, which allows it to use the same feature detector for all the sub-tasks (tracking, mapping, optimisation).

This section presents its inner workings, starting with the building blocks of feature detectors, explaining the *bag of words* approach used for optimisations and outlining the implementation details. It concludes with the output of the algorithm and our extension to it.

2.3.1 Feature detection and extraction

After camera calibration, extracting features in incoming images is the first step performed by any feature-based algorithm. These can then be used for depth recovery (stereo systems), motion estimation, loop detection and bundle adjustment optimisation. A feature descriptor encodes information about an interest point in an image to differentiate or match it among others. A *good* feature descriptor is characterised by:

- photometric and geometric invariance
- distinctiveness
- repeatability

[4] https://github.com/tum-vision/lsd_slam
[5] http://www.cvlibs.net/software/libviso/

- localisation accuracy
- robustness

Initially such descriptors focused on detecting corners (Förstner, 1986; Harris and Stephens, 1988), then moved to finding blobs, with the notable example of *Scale Invariant Feature Transform* (SIFT) (Lowe, 2004) and its upgrade into *Speeded Up Robust Features* (SURF) (Bay et al., 2008). More recently, with the introduction of *Binary Robust Independent Elementary Features* (BRIEF) (Calonder et al., 2010), they advanced to fast intensity tests between *sets of pairs* of points.

The *Oriented FAST and Rotated BRIEF* (ORB) approach (Rublee et al., 2011) fuses the FAST keypoint detector with the BRIEF descriptor along with several performance enhancements to obtain a robust, invariant and efficient descriptor. Like BRIEF, it represents the feature f as an n-vector of binary tests τ for the smoothed image patch p :

$$\tau(p; x, y) := \begin{cases} 1 & , p(x) < p(y) \\ 0 & , p(x) \geq p(y) \end{cases},$$

$$f_n(p) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(p; x, y),$$

with the pairs (x, y) chosen as the top n FAST keypoints, ordered by the Harris corner measure. Feature comparison is done via the Hamming distance between the two vectors, which is very efficient on any architecture.

In order to achieve rotation invariance, ORB computes an orientation θ for each FAST keypoint which is used to "steer" the BRIEF descriptor before sampling. Given the $2 \times n$ coordinate matrix $S = \begin{pmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{pmatrix}$ of the sampling points (x_i, y_i) for feature f, it rotates S by θ: $S_\theta = R_\theta S$. The angle θ is defined by the slope of axis $\overline{CO}$, where O is the corner's centre and C the intensity-weighted centroid of the patch.

The last enhancement, and probably the most important, is in the way the sampling pattern is chosen. After noting the high correlation introduced by the previous step, the authors suggest learning the best set of points for the application. In essence, the approach performs a search among all possible tests τ between the pairs of a training set of keypoints. It chooses in a greedy manner 256 tests which are uncorrelated (up to a given threshold) and have means near 0.5.

Similarly to SVO, ORB-SLAM keeps a homogeneous distribution of features by dividing the search space in a grid and imposing a minimum of features per cell. For small resolutions (up to 752×480) 1000 corner keypoints are recommended, with a minimum of 5 per cell. In case of failure to reach the minimum, it maintains flexibility by adapting either the detector threshold or, on texture-less patches with zero corners, the amount of corners retained per cell.

2.3.2 Bag of words

A bag-of-words model (BoW) for the visual domain was proposed by Sivic and Zisserman (2009), which treats images as documents and builds an index of visual words. First, it builds a vocabulary of features from images across the domain (e.g. a film) by using

feature extraction (subsection 2.3.1). Then it abstracts them to *codewords* by grouping similar features together. This can be done with a clustering algorithm such as *k-means*. Finally, it builds the *codebook* of the domain, which is a dictionary of codewords. For efficient querying with new features into the learned domain, the codebook keeps an index that maps codewords to the images (or frames) in which they appear.

In order to scale well with time, a SLAM algorithm should recognise places it visited before and use them to optimise its knowledge. Since the database of possible places gets big in large environments, it is important to do it efficiently. Gálvez-López and Tardos (2012) proposed a *bag-of-words* approach which is achieving this, and released it as the open-source package DBoW2[6]. However, it is limited as it uses BRIEF which is not invariant to rotation nor scale. Their work is extended by Mur-Artal and Tardós (2014) who adapt it to work with the ORB feature descriptor. This is embedded as a submodule of the ORB_SLAM package and used for several optimisations. The vocabulary is built offline from a large database of indoor and outdoor images in order to get good performance on different environments.

2.3.3 Implementation

The ORB_SLAM implementation builds on the main ideas of PTAM (Klein and Murray, 2007) by splitting different concerns on separate threads of execution: one for tracking, one for mapping and one for loop closing.

One of the key insights into its performance relies on using covisibility information for several tasks. This arranges keyframes as nodes in an undirected graph, connected by edges when they share observations of the same map points. The edges are weighted by the number of common map points. Besides this covisibility graph, a spanning tree is also built incrementally from the first keyframe, called the *Essential Graph*. It covers all keyframes but only *heavy* edges from the covisibility graph along with loop closure edges.

The tracking module estimates the current camera motion ξ_i in three big steps:

1. make initial ξ_i estimate based on state:
   a) **successful tracking for $i-1$:** get 3D-to-2D correspondences from ξ_{i-1}; $\xi \leftarrow$ solution of Perspective-and-Point (PnP) problem
   b) **tracking lost:** get bag-of-words representation of I_i and find candidate KFs from re-localisation module; for each of them get 3D-to-2D correspondences and solve PnP problem; $\xi \leftarrow$ best solution
2. project map into frame and get more correspondences to map points by using covisible KF and their immediate neighbours
3. minimise reprojection error using initial ξ_i and all correspondences from step 2.

The robustness of the algorithm relies on a good initial estimate of the map. This is performed on the tracking thread and good generalisation is achieved by parallel computation of two models along with a heuristic selecting the appropriate one. The first model assumes a planar scene and uses the homography estimation H_{cr} while the second assumes a non-planar scene and computes the fundamental matrix F_{cr} of the motion.

[6] https://github.com/dorian3d/DBoW2

Given the current and previous frames I_c, I_r and the ORB feature matches between them $x_c \leftrightarrow x_r$, H_{cr} and F_{cr} are calculated as:

$$x_c = H_{cr} x_r,$$
$$x_c^\intercal F_{cr} x_r = 0.$$

The heuristic function then uses the number of inliers for each estimate to select the best model or to reinitialise the calculation if none passes a given threshold. This basically completely solves the *wake-up robot problem* as it gives correct and accurate initialisation without requiring any particular scene setup or exterior intervention.

The mapping thread processes new frames, keeps track of the map points and the covisibility graph, inserts new keyframes and removes duplicate ones. This last step is important for keeping the size fixed in time when the same environment is explored.

The loop closing thread watches new keyframes for detecting loops. Upon detection it calculates the accumulated drift, aligns the ends of the loops and fuses the map points. Then it inserts edges into the *essential graph* which contribute to global optimisation over similarity constraints.

All optimisations use the Levenberg-Marquardt algorithm from g2o[7] framework with a Huber cost function for robustness of camera reprojection error.

2.3.4 Extensions

By default, the ORB_SLAM package only saves the trajectory of the camera through a sequence of keyframes. We have extended it to also generate map data for further offline processing. This includes map points along with their normals and colours, which are exported both in raw form and in the widely used *Polygon File Format* (.ply[8] extension).

In addition, we have added an interactive feature to help evaluate the generated map. The user can select points of the map from the current view and is shown the relative distance between them. Using the previously measured scale, this can then be compared to the real distance in order to get a measure of the accuracy.

[7] https://github.com/RainerKuemmerle/g2o

[8] linkhttp://paulbourke.net/dataformats/ply/

Autonomy & Reconstruction

Besides being able to get a map of the environment through which the drone had navigated, we wanted it do be done automatically and autonomously. Section 3.1 details the challenges this implies in the context of monocular SLAM and our approaches to them. In addition, we tried to get as much detail as possible from the resulting map so in section 3.2 we explore options in that regard. Finally, section 3.3 gives an overview of the whole system and details how everything is put together.

3.1 Autonomy

> "Would you tell me, please, which way I ought to go from here?"
> "That depends a good deal on where you want to get to," said the Cat.
> "I don't much care where–" said Alice.
> "Then it doesn't matter which way you go," said the Cat.
> "–so long as I get SOMEWHERE," Alice added as an explanation.
> "Oh, you're sure to do that," said the Cat, "if you only walk long enough."
>
> *Alice's Adventures in Wonderland*, Lewis Carroll

A truly autonomous system consists of three parts:

1. a robust wake-up and/or swift stealing recovery
2. an intelligent planner, in charge of selecting goals
3. a careful driver, in charge of navigating safely

As seen in section 2.3.3 the ORB-SLAM algorithm deals well with item 1. The rest of the section details the other two, emphasising the restrictions imposed by monocular, feature-based SLAM.

3.1.1 Goal and scale acquisition

Once it has an idea of where it is, the drone needs to acquire a point to navigate to, a goal. This can be done based on the map it that's already built or based on some other visual cues. Choosing the next goal, in general, is a type of *multi-armed bandit* problem, the agent having to choose between exploration of new areas and exploitation of existing ones. Heng et al. (2015) have shown an efficient strategy of achieving both with a depth-sensing MAV. However, if the resulting map cues are solely the ones from the feature-based SLAM, the exploitation part is non-existing since the same features will be detected every time.

To enforce exploration, one strategy is to extend the frontiers of the map (Yamauchi, 1997). To do so, it needs a densely populated 3D occupancy grid in order to find free paths and make assumptions about *true* empty spaces. While this has problems in outdoors environments, it works well for interior spaces like the ones we are interested in. However,

in the case of sparse maps like the one built by ORB-SLAM, it is close to impossible to infer whether the emptiness is due to lack of real objects or to objects with few interest points or texture.

Another strategy based on the 3D map is that of *wall following* and the implementation is relatively straightforward once we define the *walls*. The map kept by ORB-SLAM consists of individual 3D points, so additional work is needed to infer the position and orientation of a wall, which is detailed in subsection 3.2.1.

Regardless of the chosen strategy, the position of the goal has to be known in world coordinates as well. Since the built map is scale-free, we need a mean of acquiring one. For monocular SLAM, all successful approaches for this involve fusion with IMU sensors and filtering. Our approach takes advantage of the optimisations in ORB-SLAM which restrict the drift of the scale. As such, we measure the scale once, in an initialisation step, by flying the drone manually for a known distance. The navigation module can then use it to calculate the relative position of the goal. We tried to devise an automatic way scale initialisation by using the drone's downward facing camera. This would scan for a known pattern in the image, then travel forward until the pattern is found again. This way, we have better control of the travelled distance. However, this approach was limited by the API which does not allow changing the active camera at run time.

If we allow some knowledge of the environment to guide our exploration, then the goal acquisition does not have to rely on the map. Criminisi et al. (1999) have shown that vanishing points can be used for estimating 3D structure. Given the restricted environment of an office building, we know it would consist mostly of corridors and equally spaced walls. The drone can use this information to navigate by choosing the vanishing point as a goal and keeping equal distance from the walls. Wang et al. (2014) successfully employ a vanishing point strategy for goal acquisition and navigation. Bills et al. (2011) also use vision for this task but besides corridors, it also recognises stairs and rooms through a classifier. Then it dynamically adopts a different navigation strategy suitable to the current environment.

For our goal acquisition and navigation system we experimented with the vanishing point strategy. We used the well established method of searching for lines through the Hough transform from OpenCV[1]. While the detection worked well, navigation remains an open challenge due to lack of real distance to the wall ahead.

3.1.2 Navigation

Once a goal has been established, the autonomous system needs a driver to plan an obstacle-free path and follow it. To achieve real-time performance, collision-free trajectory are usually decomposed in two steps, both of which are desirable: static and dynamic. The former proposes a set of way-points based on a given view of the map and a heuristic such as shortest travelled distance or maximum coverage. The latter watches the environment in real time and adopts a reactive strategy based on motion constraints. The result is then applied to the static path in order to avoid collisions with moving objects. The second step is needed especially on systems in which the map building is not real-time and is robust to noise and temporary objects. The literature on the topic of path planning and following is vast and beyond our scope. Alonso-Mora et al. (2015) give

[1] https://github.com/Itseez/opencv

a good overview of the topic and list three optimisation-based approaches for a reactive driver.

Our driver follows a static, predefined path in the environment. Being restricted in the available commands by the API, it can only set the drone's speed on one axis at a time, and only as a percentage of a maximum value. While this suffices for basic tasks, it does not allow more aggressive manoeuvres. As such, we decided a *proportional-derivative* (PD) approach fits well with the given limitations. It works together with the earlier presented module of goal and scale acquisition to steer the drone through the map.

3.2 Reconstruction

Even though a visually rich reconstruction of the map was beyond our aims, explorations of ideas in the autonomy section pushed the research into the topic of reconstruction. A better rendering of the map is defined by two variables: structure and looks. While we were more interested in the former for the navigation part, the later gives a better purpose to the project.

3.2.1 Better structure

The ORB-SLAM stores the map as a collection of individual 3D points, each associated to a keyframe. In addition, it also estimates the perceived surface normal at each point based on the different poses which observe it. After extending the code to also keep track of the apparent colour of each point, we have exported this information into the *Point Cloud Library* (PCL[2]) where we extract structure from the map. Notably, we used the properties of corridors to highlight and estimate planes that could represent walls, useful in navigation (section 3.1.1). This employs a *Random Sample Consensus* method (RANSAC, Fischler and Bolles 1981), which is established as the standard of model estimation in presence of outliers. It works by computing model hypotheses from randomly sampled sets of data and verifies them among the remaining points. It chooses as solution the model that has the highest consensus with the other data (algorithm 2).

Our implementation uses the RANSAC provided by PCL to fit a plane through the point cloud. Besides the three regular parameters of a plane $p : ax + by + cz + d = 0$, we also impose other restrictions for selection of interesting planes. For example, to avoid detecting the floor or the ceiling, the plane we are looking for should be roughly perpendicular to them, meaning its normal should be parallel to the x axis (figure 3.1).

3.2.2 Better looking maps

For generating better looking maps, Mur-Artal and Tardós (2015) proposed a probabilistic semi-dense approach that works well with ORB-SLAM. However, it incurs an overhead which few functional benefits, since the map still cannot be used for navigation.

Instead we considered an offline *structure-from-motion* (SfM) approach. We can use ORB-SLAM to record a series of keyframes which cover the whole map and feed these into an established SfM solution. Moreover, we considered exporting the constraints from the already computed pose graph which could serve as priors to SfM and speed it up.

[2] http://pointclouds.org/

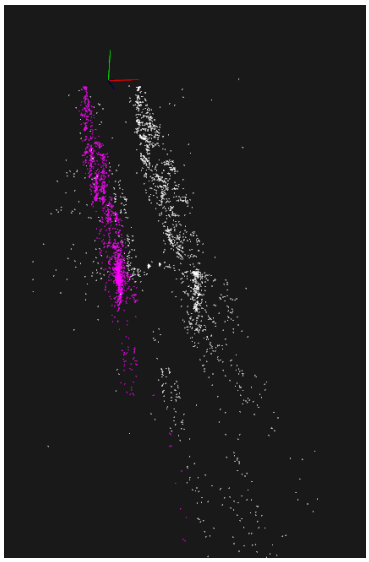


Figure 3.1: Plane segmentation is performed with the RANSAC method. This is a view from above and slightly to the left of a corridor. After detecting a plane, the points are highlighted in purple.

3.3 System overview

Since our aims were to explore different ideas and solutions to the challenges of autonomous mapping, we required a system that:

a) is modular, with low coupling

b) allows easy swapping of modules

c) facilitates communication between modules despite the language they're written in

d) allows instances of modules with same functionality to run in parallel

e) is centred around SLAM (optional)

The rationale behind item d) is related to the development phase of autonomy on the drone. Although inexpensive, damaging the hardware is time consuming. Therefore, we wanted to have a manual backup for the autonomous driver as a safety feature. They would both run in parallel and the manual driver can override the autonomous one.

Based on the above, we chose the *Robot Operating System* (ROS, Quigley et al. 2009) as the supporting architecture of our work (figure 3.2). This is a distributed framework designed to be modular and to provide, among many other things, a structured communications layer. It fulfils all the above requirements, provided the software is configured to work with it and all the dependencies are met. Unfortunately, this is one of its biggest weaknesses, especially since version *Groovy* which breaks backwards compatibility.

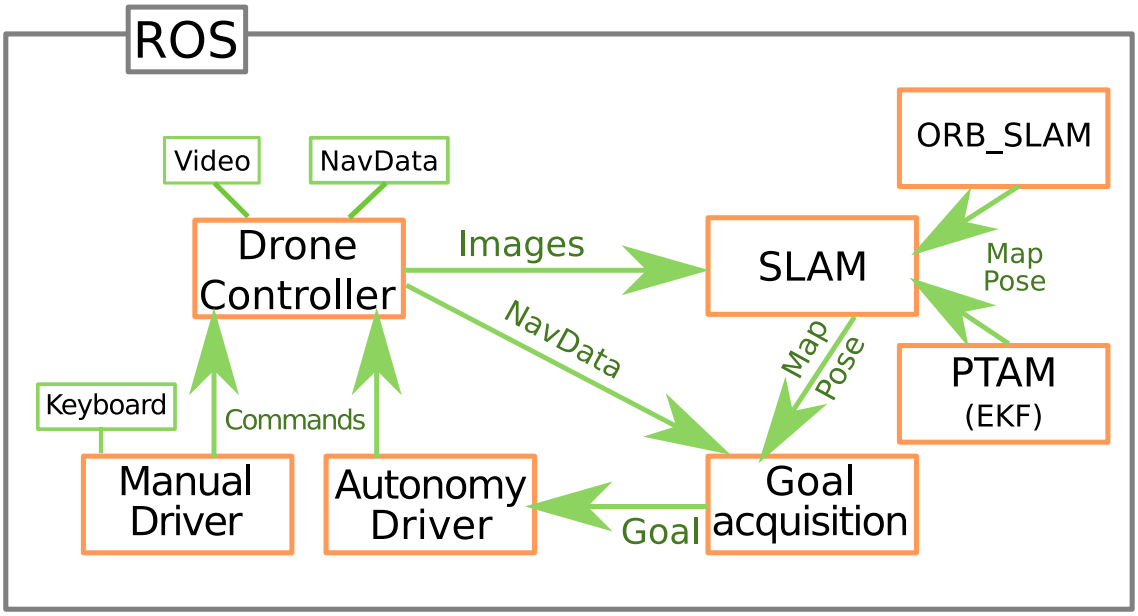


Figure 3.2: The diagram of our architecture shows the inputs in green boxes, the modules in orange and the communications between them with green arrows.

Evaluation

This chapter covers the evaluation methods used for assessing our work. We commence with the autonomy module in section 4.1 since it can be isolated from the other components. For the SLAM algorithms, extensive testing and benchmarking is carried out and published by their authors. These are implemented on public and well known data sets such as KITTI[1] and TUM_RGBD[2]. Therefore, our method is more concerned with the practicalities of the implementations and notes the interdependence with the reconstruction module. Section 4.2 presents combined ideas from the evaluation of the SLAM module in conjunction with the reconstruction one.

4.1 Autonomy

As detailed in subsection 3.1.2, our autonomous driver is based on a proportional-derivative controller. It takes a goal $p = (x, y)$ as 2D coordinates and the scale of the map and steers the drone by setting a relative *pitch* speed (moves the aircraft along the x axis). Note the elevation is kept constant.

4.1.1 The experiment

To test our driver we set up an experiment recording the error along the x axis. We performed 5 complete cycles of:

- **initialisation:** take-off; initialise SLAM; initialise scale
- **goal retrieval:** $p_{goal} = (x, y)$, constant z
- **flight loop:** take $p_{current} = (x_c, y_c)$; set $\dot{x} = k_1 \frac{x - x_c}{x} + k_2 \Delta x$
- **end:** land when $p_{current} \simeq p_{goal}$

and measured the error $e = \|p_{land} - p_{goal}\|$ as the distance between landing point and goal point. In each of them we set $p_{goal} = (6, 0)$, meaning the MAV should follow a straight line from the initial position, for 6 meters, with no side drift.

4.1.2 Results

Run number	Scale	p_{land}	Error
1	11.68	$(6.53, -1.07)$	1.19406867474
2	13.91	$(4.96, 0.81)$	1.31821849479
3	12.01	$(5.21, 0.39)$	0.881022133661
4	13.76	$(4.83, -0.54)$	1.28860389569
5	11.37	$(6.78, -1.23)$	1.45646833127
Mean	12.54	$(5.66, -0.32)$	1.227

[1] http://www.cvlibs.net/datasets/kitti/
[2] http://vision.in.tum.de/data/datasets/rgbd-dataset

The previous table lists the measurements taken and the error on each of them. Considering the accuracy of ORB-SLAM of only a few centimetres on various data sets (see figure 4.1), we exclude that as a source of errors. As such, the mean error of 1.227 meters can be justified mostly by the inaccurate scale measurement, as it is done manually. For better results, in section 3.1.1 we also proposed a mean of automatic scale acquisition.

We believe another significant source of errors is the hardware configuration of our system. The AR.Drone is pre-loaded with a driver of its own meant to keep its stability during flight. We noticed that in some environments this tends to amplify the commands it receives, resulting in unexpected drift. On top of that, there is also a communication cost in the system which incurs a delay between the processed frames and the moment the calculation is done, resulting in inaccurate measurements of position.

Lastly, we note the significant error in the y component of the landing point. We related this to a possible hardware fault as the drone tends to drift to the left (i.e. negative y) even during hovering, with no commands given.

4.2 SLAM and Reconstruction

One of the key factors supporting our choice of SLAM were the claimed results of ORB-SLAM (figure 4.1). Its robustness makes it the most desirable out of the three we considered (PTAM, LSD-SLAM, ORB-SLAM). However, we also experimented with the other two. It is worth mentioning that LSD-SLAM requires careful tuning and there is no *correct* parameter setting, choices being made empirically. This made it more difficult to evaluate since the cause of tracking failure was not clear.

	ORB-SLAM	PTAM	RGBD-SLAM	LSD-SLAM
fr1_xyz	**0.90**	1.15	1.34	9.00
fr2_xyz	0.30	**0.20**	1.42	2.15
fr1_floor	**2.99**	✗	3.51	38.07
fr1_desk	**1.69**	✗	2.52	10.65
fr2_360_kidnap	3.81	**2.63**	100.5	✗
fr2_desk	**0.88**	✗	3.94	4.57
fr3_long_office	**3.45**	✗	-	38.53

	ORB-SLAM	PTAM	RGBD-SLAM	LSD-SLAM
fr3_str_tex_far	**0.77**	0.93	-	7.95
fr3_str_tex_near	1.58	**1.04**	-	✗
fr2_desk_person	**0.63**	✗	2.00	31.73
fr3_sit_xyz	**0.79**	0.83	-	7.73
fr3_sit_halfsph	**1.34**	✗	-	5.87
fr3_walk_xyz	**1.24**	✗	-	12.44
fr3_walk_halfsph	**1.74**	✗	-	✗

Figure 4.1: An evaluation of SLAM algorithms on the public repository TUM_RGBD. The numbers represent the Root Mean Squared Error of the distance, in centimetres. Red crosses mean the algorithm failed to complete on that respective set. Image from Mur-Artal (2015).

For a better evaluation of the constructed map in terms of representation of the real environment, we proposed an interactive test in subsection 2.3.4, which we summarise here: the user measures a distance both on the map and in real scene and uses the acquired scale to compare them (figure 4.2). The selected points correspond to a distance of 1.29 m in the real environment, while the calculated distance is $d = s \cdot \|AB\| = 12.41 \cdot 0.091 = 1.13$ m. The difference, again, comes from imprecise scale measurement.

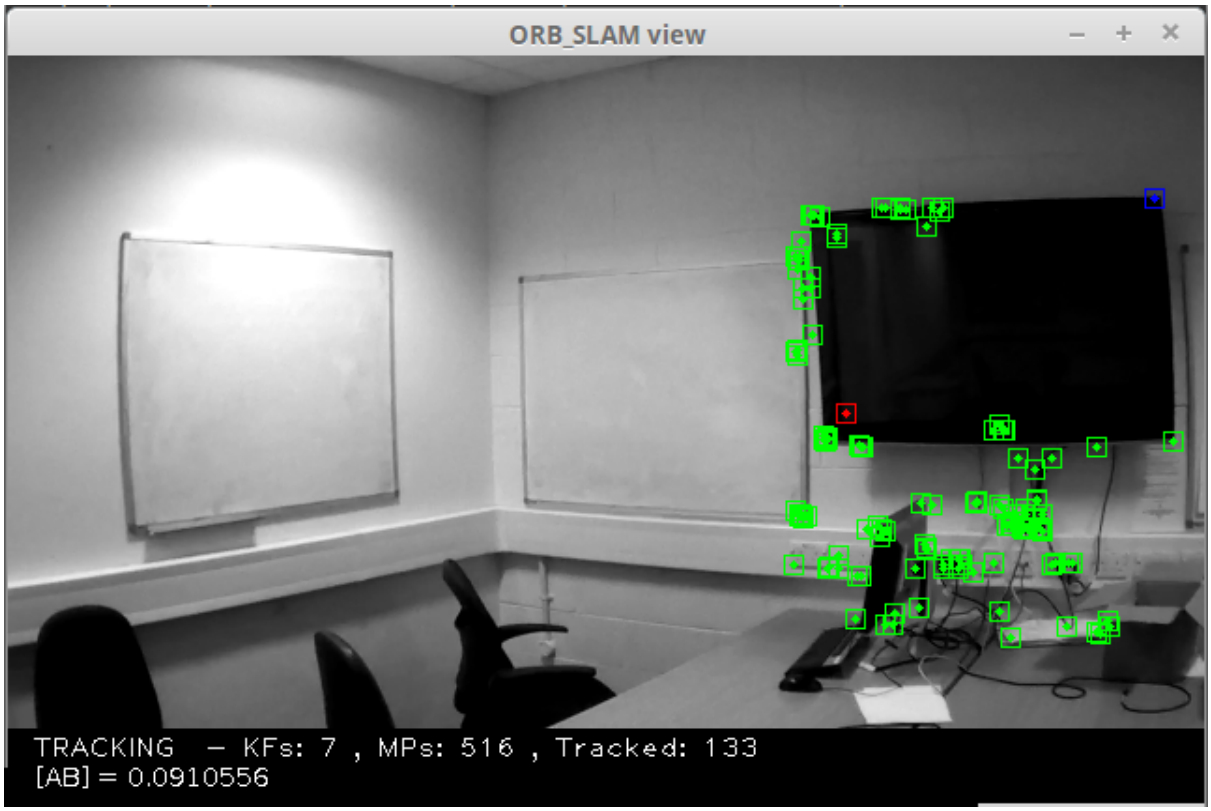
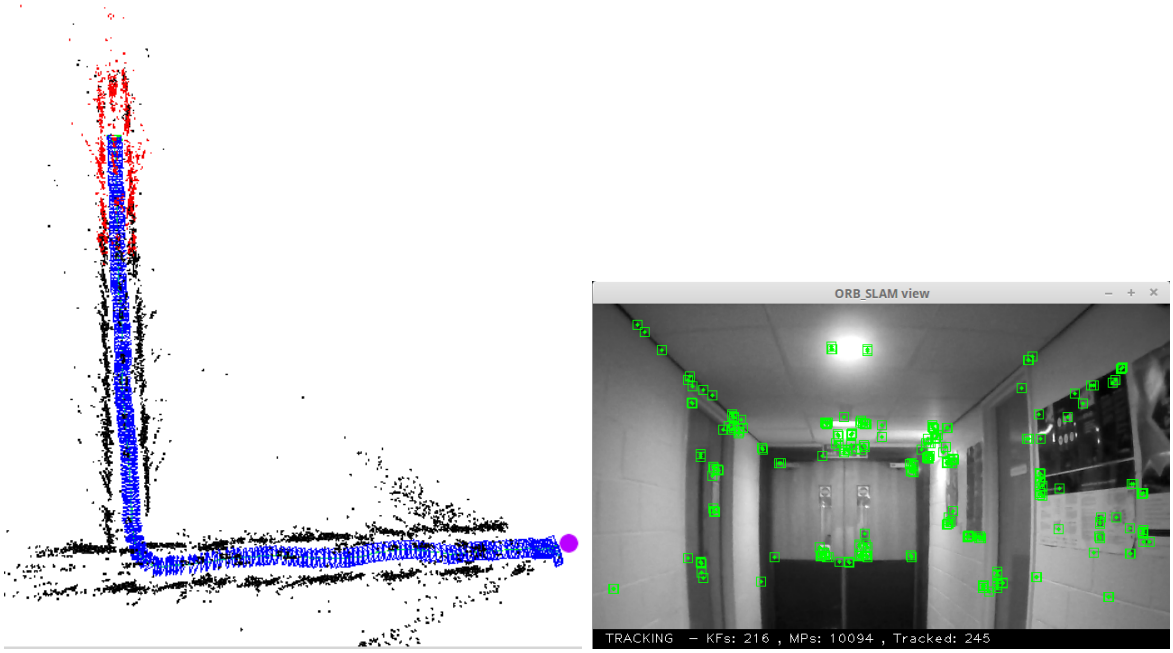


Figure 4.2: A view of the ORB-SLAM with two points of interest selected (red and blue). The acquired scale was 12.41.

One last measure for our system is in terms of map density and outliers. It is hard to define a sensible metric for this as the maps are intended for visual inspection. Looking at figure 4.3, we note the points follow closely the walls of the corridor but there are also a few outliers, especially close to the beginning of the exploration.



(a) The map (b) Current view

Figure 4.3: The map generated by ORB-SLAM and the view at the end. The starting point is marked with a purple dot. The points coloured in red represent the ones that are currently seen, while the ones in black are considered out of view and are not tracked. The blue squares represent keyframes along the trajectory.

Conclusion

This chapter concludes our work and this report by summarising the achievements, reiterating through the knowledge we gained and listing possible tasks for future work.

5.1 Achievements

Towards the end of the implementation we demonstrated an autonomous MAV mapping the environment it passes through and generating data for offline processing. We also added software for understanding higher level structures from the point cloud, in the form of planes. Finally, we tied everything together in a highly modular and configurable system which allows other modules to be added or swapped with the current ones.

On the theoretical side, we performed an in-depth research of the SLAM problem and analysed three modern implementations. Then we explored and implemented a number of ideas for autonomous navigation, based either on the generated map or on different visual cues such as vanishing point.

We based our choices on evaluations of the packages we used both in terms of performance (done by the authors) and in terms of fitness to the purpose (performed by us). Our results were tested through high-level experiments and the shortcomings were identified, along with ways to overcome them.

5.2 Knowledge gained

We climbed a steep learning curve in order to get a thorough understanding of the challenge and its solutions. This was supported by a number of papers and tutorials into the different domains of computer vision (see references):

- camera calibration
- feature detection and extraction
- visual odometry
- pose-graph optimisations and loop closure
- structure from motion
- object recognition

5.3 Future work

Perfect, real-time, monocular SLAM is still an open challenge. While almost completely solved on a powerful computer, the challenge is still open for embedded architectures. Likewise, the problem of monocular dense mapping is currently solved only through the use of GPUs.

When considering our achievements, we believe the following are realistic possible extensions:

- inertial fusion – use measurements from IMU for a truly drift-free SLAM
- offline structure from motion – use the approach described in subsection 3.2.2 to generate visually rich maps
- vanishing point navigation – needs depth measurement so a slightly different hardware configuration is recommended

5.4 Reflection

The project reached the goals set at the beginning and provided deep insight into its area, despite being faced with a number of challenges such as the restricted hardware. Although not yet ready to explore Mars, it has shown a proof of concept and ways to improve it. Overall, I gained extensive knowledge and a real passion into the field of computer vision.

References

Javier Alonso-Mora, Tobias Naegeli, Roland Siegwart, and Paul Beardsley. Collision avoidance for aerial vehicles in multi-agent scenarios. *Autonomous Robots*, 39(1):101–121, 2015. 3.1.2

AVInc. Nano hummingbird. *http://www.avinc.com/nano*. (Accessed 3 May 2016). 1.1

Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006. 2

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008. 2.3.1

BBC. Easyjet develops flying robots to inspect aircraft. *http://www.bbc.co.uk/news/business-27308232*. (Accessed 3 May 2016). 1.1

Cooper Bills, Joyce Chen, and Ashutosh Saxena. Autonomous MAV flight in indoor environments using single image perspective cues. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 5776–5783. IEEE, 2011. 3.1.1

Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary Robust Independent Elementary Features. *Computer Vision–ECCV 2010*, pages 778–792, 2010. 2.3.1

José A Castellanos, José Neira, and Juan D Tardós. Limits to the consistency of EKF-based SLAM. 2004. 2

Yang Cheng, Mark Maimone, and Larry Matthies. Visual odometry on the Mars exploration rovers. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 1, pages 903–910. IEEE, 2005. 1.1

Ingemar J Cox. Blanche-an experiment in guidance and navigation of an autonomous robot vehicle. *Robotics and Automation, IEEE Transactions on*, 7(2):193–204, 1991. 2.1.1

Antonio Criminisi, Ian Reid, and Andrew Zisserman. Single view metrology. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 434–441. IEEE, 1999. 3.1.1

Matthew N Dailey and Manukid Parnichkun. Landmark-based simultaneous localization and mapping with stereo vision. In *Proceedings of the Asian Conference on Industrial Automation and Robotics*, 2005. 2.2

Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007. 2.2.2

Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006. 2

J. Engel, J. Sturm, and D. Cremers. Accurate figure flying with a quadrocopter using onboard visual and inertial sensing. In *Proc. of the Workshop on Visual Control of Mobile Robots (ViCoMoR) at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012a. 2.2.2

J. Engel, J. Sturm, and D. Cremers. Camera-based navigation of a low-cost quadrocopter. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012b. 2.2

J. Engel, J. Sturm, and D. Cremers. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems (RAS)*, 62(11):1646––1656, 2014a. 2.2

Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014b. 2.2, 2.2.1

Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 3.2.1

Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014. 2.2, 2.2.3

Wolfgang Förstner. A feature based correspondence algorithm for image matching. *International Archives of Photogrammetry and Remote Sensing*, 26(3):150–166, 1986. 2.3.1

Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *Robotics & Automation Magazine, IEEE*, 19(2):78–90, 2012. 2.2.2

Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *Robotics, IEEE Transactions on*, 28(5):1188–1197, 2012. 2.3.2

Arren Glover, William Maddern, Michael Warren, Stephanie Reid, Michael Milford, and Gordon Wyeth. Openfabmap: An open source toolbox for appearance-based loop closure detection. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4730–4735. IEEE, 2012. 15

G Grisetti, H Strasdat, K Konolige, and W Burgard. g2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation*, 2011. 2.2.1

Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988. 2.3.1

Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous visual mapping and exploration with a micro aerial vehicle. *Journal of Field Robotics*, 31(4):654–675, 2014. 2.2

Lionel Heng, Alkis Gotovos, Andreas Krause, and Marc Pollefeys. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1071–1078. IEEE, 2015. 3.1.1

Michal Irani and P Anandan. About direct methods. In *Vision Algorithms: Theory and Practice*, pages 267–277. Springer, 1999. 2.2.1, 2.2.1

Michael Kaess and Frank Dellaert. Visual slam with a multi-camera rig. 2006. 2.2

Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007. 2.2, 2.2.2, 2.2.3, 2.3.3

Nicola Krombach. Combining Feature-based and Direct Methods for Semi-dense Real-time Visual SLAM from Stereo Cameras. Master's thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2 2016. 2.2.3

Chang Liu, Stephen D Prior, WT Luke Teacy, and Martin Warner. Computationally efficient visual–inertial sensor fusion for Global Positioning System–denied navigation on a small quadrotor. *Advances in Mechanical Engineering*, 8(3), 2016. 2.2

David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 2.3.1

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. IJCAI. 2

Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002. 2

Takayoshi Mori and Stefan Scherer. First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1750–1757. IEEE, 2013. 2.2

Raúl Mur-Artal. Should we still do sparse-feature based SLAM? In *The Future of Real-Time SLAM (ICCV Workshop)*, December 2015. 4.1

Raúl Mur-Artal and Juan D Tardós. Fast relocalisation and loop closing in keyframe-based SLAM. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 846–853. IEEE, 2014. 2.3.2

Raúl Mur-Artal and Juan D Tardós. Probabilistic semi-dense mapping from highly accurate feature-based monocular SLAM. *Proceedings of Robotics: Science and Systems, Rome, Italy*, 2015. 3.2.2

Raúl Mur-Artal, JMM Montiel, and Juan D Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *Robotics, IEEE Transactions on*, 31(5):1147–1163, 2015. 2.2, 2.3

Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011. 2.2.1

David Nistér. An efficient solution to the five-point relative pose problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):756–770, 2004. 2.2.2

David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652. IEEE, 2004. 2.1.3

Gabriel Nützi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Fusion of imu and vision for absolute scale estimation in monocular slam. *Journal of intelligent & robotic systems*, 61(1-4):287–299, 2011. 2.2

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5, 2009. 3.3

Susan Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debabrata Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1765–1772. IEEE, 2013. 2.2

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011. 2.3.1

Davide Scaramuzza. Visual odometry tutorial. *http://rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf*. (Accessed 3 May 2016). 2.3, 2.4

Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *Robotics & Automation Magazine, IEEE*, 18(4):80–92, 2011. 2.1.3, 2.2

Konstantin Schauwecker and Andreas Zell. On-board dual-stereo-vision for the navigation of an autonomous mav. *Journal of Intelligent & Robotic Systems*, 74(1-2):1–16, 2014. 2.2

Korbinian Schmid, Philipp Lutz, Teodor Tomić, Elmar Mair, and Heiko Hirschmüller. Autonomous vision-based micro air vehicle for indoor and outdoor navigation. *Journal of Field Robotics*, 31(4):537–570, 2014. 2.2

Stephen Se, David Lowe, and Jim Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *The international Journal of robotics Research*, 21(8):735–758, 2002. 2.2

Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(4): 591–606, 2009. 2.3.2

Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986. 2

TeraRanger. First CERN spin-off in france. *http://www.teraranger.com/first-french-cern-spin-off/*. (Accessed 3 May 2016). 1.1

Jialiang Wang, Hai Zhao, Yuanguo Bi, Xingchi Chen, Ruofan Zeng, and Shiliang Shao. Quad-rotor helicopter autonomous navigation based on vanishing point algorithm. *Journal of Applied Mathematics*, 2014, 2014. 3.1.1

Stephan Weiss, Markus W Achtelik, Simon Lynen, Margarita Chli, and Roland Siegwart. Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 957–964. IEEE, 2012. 2.2

Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE, 1997. 3.1.1

Algorithms

Algorithm 1 LSD-SLAM update loop

procedure UPDATE($KF_{current}$)
    $F_{new} \leftarrow capture\_image()$
  Using Gauss-Newton minimisation on photometric error function:
    $\xi \leftarrow estimate\_transformation(F_{new}, KF_{current})$
    $d = distance(F_{new}, KF_{current})$
    if $d < threshold$ **then**
        $new\_idm = estimate\_inverse\_depth\_map(F_{new})$
        merge new_idm into $idm[KF_{current}]$
    else
        $KF_{new} \leftarrow F_{new}$
        $idm[KF_{new}] \leftarrow projection(idm[KF_{current}], \xi)$
        remove outliers in $idm[KF_{new}]$
    end if
    add $KF_{current}$ to environment map
    perform **loop closure**:
        $\{KF_{proposed}\} \leftarrow appearance\_mapping(KF_{current})$ // OpenFabMap [24]
        $\{KF_{candidates}\} \leftarrow \{KF_{proposed}\} \cup \{10 \text{ closest } KF\}$
        for each $KF_{candidate} \in \{KF_{candidates}\}$ **do**
        // Check statistical similarity when estimated independently
            if back transformation **is similar** to forth transformation **then**
                $insert\_edge(KF_{current}, KF_{candidate})$
            end if
        end for
    perform **graph optimisations**
end procedure

Algorithm 2 Random Sample Consensus for planar estimation

(1) Sets of approximately co-planar points $x = [u\ v\ d]^\intercal$ satisfy $|au + bv + c - d| \leq \epsilon$

(2) Let $C(S, p)$ be a fitness function reflecting the quality of the consensus set

Input: A set of points S, a small value ϵ, maximum running time t_{max}

Output: Parameters of a plane p' and the set of inliers S'

1: **procedure** PLANAR ESTIMATION(S, ϵ, t_{max})
2: $C' \leftarrow 0$ // fitness of current estimation
3: $t_0 \leftarrow t$
4: **repeat**
5: Select x_i, x_j, x_k from S
6: Determine parameters of plane $p = [a\ b\ c]^\intercal$ from x_i, x_j, x_k
7: $S' \leftarrow \{x$ where $x \in S$ and x satisfies the co-planar equation (1) given $p\}$
8: **if** $C(S', p) > C'$ **then**
9: $C' \leftarrow C(S', p)$
10: $S \leftarrow S'$
11: **end if**
12: **until** $t - t_0 > t_{max}$
13: Determine the parameters p' of the plane which make the best lest-squares fit to the points in S'
14: **end procedure**