Ego-Motion Estimation of Drones

EMRE AY
Ego-Motion Estimation of Drones

Positionsestimering för drönare

EMRE AY

Master’s Thesis at RPL
Supervisor: Patric Jensfelt
Examiner: Joakim Gustafson
Abstract

To remove the dependency on external structure for drone positioning in GPS-denied environments, it is desirable to estimate the ego-motion of drones on-board. Visual positioning systems have been studied for quite some time and the literature on the area is diligent. The aim of this project is to investigate the currently available methods and implement a visual odometry system for drones which is capable of giving continuous estimates with a lightweight solution. In that manner, the state of the art systems are investigated and a visual odometry system is implemented based on the design decisions. The resulting system is shown to give acceptable estimates.
Sammanfattning

För att avlägsna behovet av extern infrastruktur så som GPS, som dessutom inte är tillgänglig i många miljöer, är det önskvärt att uppskatta en drönarens rörelse med sensor ombord. Visuella positioneringssystem har studerats under lång tid och litteraturen på området är ymnig. Syftet med detta projekt är att undersöka de för närvarande tillgängliga metoderna och designa ett visuellt baserat positioneringssystem för drönare. Det resulterande systemet utvärderas och visas ge acceptabla positionsuppskattningar.
Contents

1 Introduction 1
  1.1 Problem Statement 2
  1.2 Organization 2
  1.3 Scope of the Project 2

2 Background 4
  2.1 Visual Odometry 4
  2.2 Feature Based VO 5
    2.2.1 Feature Detection and Extraction 6
    2.2.2 Matching & Tracking 7
    2.2.3 Motion Estimation 8
  2.3 Appearance Based VO 10
  2.4 Hybrid Approaches 11
  2.5 Random Sample Consensus 12
  2.6 Optimization 13
    2.6.1 Key Frames 13
    2.6.2 Pose Graph Optimization 13
    2.6.3 Bundle Adjustment 13

3 Related Work 14

4 Investigation of the Current Systems 16
  4.1 SVO 16
    4.1.1 Background 16
    4.1.2 Details and Review 18
  4.2 ORB-SLAM 19
    4.2.1 Background 19
    4.2.2 Details and Review 20
  4.3 ORB-SLAM 2 21
    4.3.1 Background 21
    4.3.2 Details and Review 21

5 Design & Methods 23
  5.1 Design Decisions 23
5.2 Methods ................................................. 25
  5.2.1 FAST Detector .......................................... 25
  5.2.2 BRIEF Descriptor ....................................... 27
  5.2.3 ORB (Oriented FAST and Rotated BRIEF) ............. 28
5.3 Feature Matching Methods ................................. 29
  5.3.1 Stereo Matching ........................................ 29
  5.3.2 Matching with Window Search ......................... 29
  5.3.3 Matching with Projection .............................. 30
5.4 Motion Tracking .......................................... 30

6 Practical Information & Implementation ........................ 32
  6.1 Hardware ................................................. 32
  6.2 Software .................................................. 33
  6.3 Dataset Collection ........................................ 33
  6.4 Implementation .......................................... 33

7 Evaluation .................................................. 34
  7.1 Criterion ................................................ 34
  7.2 Experimentation .......................................... 34
    7.2.1 Experiment 1 ......................................... 34
    7.2.2 Experiment 2 ......................................... 38
    7.2.3 Experiment 3 ......................................... 39

8 Conclusion ............................................... 43
  8.1 Summary ................................................. 43
  8.2 Discussion ............................................... 43
  8.3 Conclusion ............................................... 44
  8.4 Future Work ............................................. 44

Appendices ................................................. 45

A Ethics, Society & Sustainability .............................. 46
  A.1 Ethics .................................................. 46
  A.2 Society ................................................ 47
  A.3 Sustainability .......................................... 48

Bibliography ............................................... 49
Chapter 1

Introduction

There is an increasing interest on drones—or Unmanned Aerial Vehicles (UAV) as a general name—as their application spectrum grows wider and they become more reachable. Applications, sizes and structures of drones vary, making them suitable for assorted environments and usages; from large military/defence drones to personal drones for taking photographs. Recent improvements and high market demand on microelectronics and MEMS made smaller, cheaper and better hardware possible. Hence the availability of fast and light processing units together with sensors was a significant factor on the recent popularity of drone technology.

Autonomous applications of drones require integration of various sub-systems. A key sub-system to accomplish any autonomous task with drones is positioning. It is essential to have reliable information on a drone’s position or trajectory since it directly limits or affects the quality of applications. Using GPS (Global Positioning System) information is a common practice where possible. For GPS-denied environments such as indoor areas accurate position information can be supplied to drones by using external structures such as several cameras or radio signal receiver/transmitters mounted inside an indoor area. Yet this limits the mobility of the drones, requires special hardware and increases the cost. Such systems are useful in research since the cost and limited movement space are usually tolerable. However, end-user products would require no reliance on such costly external structure. Hence on-board solutions with relatively cheap hardware are demanded.

On-board positioning systems can be created using visual sensors on drones. Visual positioning systems have been studied for several decades and ego-motion estimation is still an active research area. Although the methods are theoretically well formulated, robust pose estimation is still a challenge due to practical constraints and many theoretical assumptions. One major constraint is the computational cost. The visual pose estimation systems can be computationally expensive and even though the current CPUs are capable of handling this, it is still desirable to have computationally more lightweight solutions. As previously stated, there are numerous other sub-parts in an autonomous system such as a drone, and the available computational capacity as well as the memory is to be shared by all of them.
CHAPTER 1. INTRODUCTION

Therefore, being lightweight is sought on a visual positioning system. But, in order to accomplish that, one should compromise from another system parameter. In that manner, the price of constricting the computational expense is usually paid with the system performance.

Most civil drones are operated with commercially available Flight Controller Units (FCUs) such as Pixhawk autopilot boards [1]. The FCUs are separate processing boards which are equipped with sensors (e.g. IMU, magnetometer...) and run an autopilot software. FCUs are low level units; they can take commands from other computers which perform the higher level computations such as mapping, path planning or obstacle avoidance. These commercial boards are also capable of receiving external position estimates [2] and performing on-board sensor fusion [3] together with their internal sensor data. In that manner, compensating the reduced accuracy (due to constricting the computational expense) on the pose estimate of a lightweight system might be possible using the pose fusion function of the drone’s FCU. Based on this assumption, the design criterion of the visual positioning system can be relaxed.

1.1 Problem Statement

The objective of this project is to investigate the state of the art vision based motion estimation systems and implement a lightweight visual odometry system for indoor drone applications. The research question of the project is;

- Can a similar estimation performance be achieved with a visual positioning system to be used indoors, that is more lightweight in terms of CPU load and memory consumption, than the current state of the art methods?

1.2 Organization

The organization of this report is as follows; in Chapter 2, the necessary background information on the subject is supplied. In Chapter 3, the related work is given. Then, an investigation on state of the art implementations is done in Chapter 4. Based on the outcomes of investigation, it is proceeded with design decisions and details of the methods in Chapter 5. The implementation is covered in Chapter 6 together with practical information and the evaluation is covered in Chapter 7. Finally, it is concluded with Chapter 8.

1.3 Scope of the Project

Investigating the current successful implementations and creating a lightweight visual odometry system are within the scope of this project whereas generating a globally consistent map or sensor fusion are not. The output of the visual odometry
1.3. SCOPE OF THE PROJECT

system might be further used in on-board sensor fusion of drone FCUs to improve the accuracy, however this is outside the project’s scope.
Chapter 2

Background

2.1 Visual Odometry

Visual odometry (VO) is the estimation of positional change by incrementally calculating the motion from visual input. The analogy with wheel odometry comes from the incremental nature of calculations [4]. A major convenience of visual odometry over wheel odometry and GPS is the independence on interaction with the surroundings [5] such as wheel slippage or failures in GPS because of obstacles [6].

Monocular or stereo configurations as well as RGB-D cameras can be used for VO. The methods can be categorized as: feature based where distinguishable and repeatable features are matched or tracked in consecutive frames, appearance based (or global/direct) where the pixel intensities are used to minimize the photometric errors between frames, and hybrid (or semi-direct) where both feature and appearance based methods are combined [4][7][8]. Calculation of visual odometry is made through a pipeline of processes.

![Figure 2.1: Visual odometry methods and configurations](image-url)
2.2. FEATURE BASED VO

Using monocular and RGB-D/stereo vision for VO mainly differs in obtaining the absolute scale of the scene. The absolute scale can be directly determined in a calibrated stereo system since the baseline (i.e. the distance between two camera centers) is known or even more directly with an RGB-D camera where a depth image is retrieved. Whereas the estimation of the trajectory can be only made up to a scale factor in a monocular VO system [9].

Using stereo or RGB-D cameras overcome this situation since the absolute scale is observable in both cases. However as the distance between the elements and the camera increases, stereo vision starts to function as monocular. This is because the maximum depth to be triangulated is related to the stereo baseline. For far objects in the scene with no observed parallax, the system cannot retrieve depth as in the monocular case.

In fact, monocular configuration might be the least costly solution. However, to recover the trajectory with absolute scale in monocular vision, additional information is needed such as dimensions of an object in the scene or information from an other sensor. Also, the drift of the scale is an issue where it is needed to be estimated together with the motion.

Visual Odometry vs. Structure From Motion

The techniques in visual odometry are sometimes referred to Structure From Motion (SFM). SFM is the problem of recovering the 3D structure of the scenes and relative poses of the views from different images which might be unordered and uncalibrated [4]. Therefore, VO is a more specialized problem with ordered images, therefore it might be said that SFM encapsulates VO.

Visual Odometry vs. Visual SLAM

The main difference between visual odometry and visual SLAM (vSLAM) is the consistency. vSLAM uses solutions such as loop closing to detect the overall drift in the motion and in the map to correct them. However even though there exists correction techniques such as bundle adjustment (covered in Section 2.6), the correction in VO is more local and the global consistency might not be achieved.

2.2 Feature Based VO

Features are parts of the image that are of interest which would be matched between different images of the same scene. For feature based visual odometry, usually point features are used where every feature is a point (pixel) on the image. In a feature based pipeline, the features are extracted from images, matched between frames and the motion is estimated using corresponding features between frames. An example feature based VO pipeline is shown in Figure 2.2.
2.2.1 Feature Detection and Extraction

Feature detection is the process of determining and finding features in the image. According to Kneip et al. [10], the feature detector selection affects the feature based visual odometry performance significantly.

The feature can be a blob—i.e., part that shows difference in color or intensity from its neighbours in the image—or a corner. In order to make the feature scale-invariant, the extraction is usually made on the scale-space [11] of the image. Once the feature is detected, a descriptor of that feature is needed to match or track it.

Feature extraction is the process of creating descriptors of features which can be used to uniquely identify the features. The feature descriptor is a vector that is represented using the information from the region around the feature in the image. A simplistic way to create a descriptor is using the intensities of the pixels around the feature. However, this naïve approach is sensitive to spatial changes. Additional information from the neighbours is usually exploited to create robust descriptors.

Scale Invariant Feature Transform (SIFT) [12], one of the most renowned methods, represents the descriptor using the gradient information. In SIFT, a grid is placed around the feature point and gradient magnitudes and orientations are calculated for each cell in the grid. Orientation histograms are created by accumulating the cell contents and the feature descriptor is formed with size of 128 elements that correspond to 4-by-4 array with 8 orientation bins in each of them.

Speeded-Up Robust Features (SURF) [13], another well-known method, uses wavelet responses on x and y directions in a patch around the feature to create the descriptors with size of 64 elements.


2.2. FEATURE BASED VO

Figure 2.3: Feature detection from the drone’s camera


2.2.2 Matching & Tracking

Feature matching relates to comparing separately extracted features to find correspondence. The most straightforward method for matching is to compare the descriptor vectors of different features based on a distance metric such as Euclidian or Hamming distance. This brute-force approach would not be suitable for large number of features and images or for specific time constraints such as visual odometry calculations. Therefore, nearest-neighbor algorithms can be used to perform matching faster [12].

Feature tracking relates to finding feature correspondences on another image (or image part) without necessarily detect and extract features on the new image. Therefore, the features are extracted once and tracked in the next images. This would be useful since it avoids the processes of detection and extraction on every single frame. Kanade-Lucas-Tomasi (KLT) tracker is a renowned method in the literature. The algorithm consists of the Lucas-Kanade method [14] for image tracking/alignment and works of Tomasi & Kanade [15] and Shi & Tomasi [16] for selecting suitable features to track. KLT tracker assumes small motion, spatial coherence and constant intensity at the corresponding pixels between frames. The tracker is often used with a pyramidal scheme [17] where the feature tracking is made using down-sampled versions of the image to cope with small inter-frame motion assumption.
CHAPTER 2. BACKGROUND

2.2.3 Motion Estimation

Motion estimation is an essential part of visual odometry. Between every consecutive frame, the transformation between frames is estimated. The trajectory of the body can then be retrieved using all the past transformations and the transformation between the camera and the body. Several different approaches are possible for motion estimation based on feature dimensionality (i.e. whether the features are described in image or world coordinates). The corresponding features between frames can be represented in 2D image coordinates or 3D world coordinates by using triangulation. Therefore, based on the feature representation in previous and current frames it is possible to estimate the motion using:

- **2D - 2D correspondences**, where features are described in image coordinates in both frames
- **3D - 2D correspondences**, where the features from the previous frame are triangulated and described in 3D coordinates and the current features are in 2D image coordinates
- **3D - 3D correspondences**, where the features from both frames are triangulated and described in 3D coordinates

**Motion Estimation Using 2D - 2D Correspondences**

The transform between two camera views can be calculated with epipolar geometry or homography using corresponding features described in 2D image coordinates. These motion estimation techniques can be used with monocular and stereo camera configurations.

As shown in Figure 2.4, intersecting image planes between two camera views with baseline as their axis forms the epipolar geometry \[ F \]. The rays between the camera center and scene points in the first view are projected as lines in the second view and vice versa. These lines are called epipolar lines.

The fundamental matrix is a 3-by-3 matrix and represents the mapping between corresponding points and formulates the epipolar constraint;

\[
x'TFx = 0
\]

(2.1)

where \( x' \) and \( x \) are corresponding points in two images represented with homogeneous image coordinates and \( F \) is the fundamental matrix.

Once the fundamental matrix is known, the relative pose between two views can be retrieved since:

\[
F = K_2^{-T}EK_1^{-1}
\]

(2.2)

\[
E = [t]_xR
\]

(2.3)
where $K_1$ and $K_2$ are the calibration matrices, $E$ is called the essential matrix, $[t]_x$ is the skew symmetric translation matrix and $R$ is the rotation matrix.

For non-planar scenes, the 8-point algorithm can be used to calculate the fundamental matrix [19] [18]. The algorithm requires eight corresponding image points to calculate the nine unknowns of the fundamental matrix. However, the algorithm tends to degenerate in case of having four or more points lie in a straight line, or if most of the points are coplanar [19].

The 5-point algorithm [20] calculates the essential matrix with a calibrated system and is unaffected by the scene planarity. The estimation process usually involves a RANSAC scheme which is described in Section 2.5, thus it is also beneficial to require fewer points to form a hypothesis in terms of computational time. The algorithm, however, might yield multiple solutions to deal with.

Homography is another way to estimate the relative pose between views using scene planarity. Homography is the projective transformation between corresponding points and it is formulated as:

$$x' = Hx$$

(2.4)

where $x'$ and $x$ are corresponding points in two images represented with homogeneous image coordinates and $H$ is the projective transformation.

The motion hypotheses can then be retrieved with homography such as with the method of Faugeras et al. [21].

**Motion Estimation Using 3D - 2D Correspondences**

The 3D-to-2D approach is called the Perspective-n-Point (PnP) problem which Li [22] defines as "determining the pose of a calibrated camera from $n$ correspondences between 3D reference points and their 2D projections". There are numerous solutions for PnP in the literature [23]. The 3D - 2D approach is suitable for both monocular...
and stereo configurations. However, initially it is required to track three frames for monocular configuration to use the first two frames for triangulation.

The core of the PnP problem is to find the transform between views by minimizing the reprojection error. The previously triangulated 3D points are reprojected to the next image and the transform is found through \[ 2.5 \]

\[
T_k = \arg \min_{T_k} \sum \| x^i_k - \hat{x}^i_{k-1} \|^2
\]

where \( x^i_k \) is the \( i^{th} \) 2D point in the image at step \( k \) and \( \hat{x}^i_{k-1} \) is the 2D reprojection of the corresponding 3D point, both in homogeneous coordinates.

It is required to have at least three points (P3P) that are not collinear in order to derive a solution [4]. With three points, there could be up to four solutions. With four points (P4P) there would be a unique solution if all of them are coplanar [24]. There can be finitely many solutions for P4P and P5P problems with points in general position, whereas there will be a unique solution in the case of P6P. However, as stated in the previous part, since a RANSAC scheme is commonly used for outlier rejection, it is beneficial in terms of computational time to use fewer points to generate hypotheses.

Motion Estimation Using 3D - 3D Correspondences

3D-to-3D approach is the 3D point set alignment problem where the distance between two point sets is minimized to find the transform between views. It is suitable for stereo configuration where the features can be triangulated using left and right images at every instant and then matched to the previous view. The problem is formulated as [25]:

\[
T_k = \arg \min_{T_k} \sum \| X^i_k - T_kX^i_{k-1} \|^2
\]

where \( X^i_k \) is the \( i^{th} \) 3D point at step \( k \) in homogeneous coordinates.

Nister et al. [6] states that 3D-2D approaches are more accurate than 3D-3D approaches due to the higher uncertainty in depth direction of triangulated points [4].

2.3 Appearance Based VO

Appearance based methods use directly the intensity information of each pixel rather than extracting sparse distinct points from the image to estimate the motion and structure. Therefore, appearance based methods are also called direct or global methods.

There are several advantages in using appearance based methods:

- **No need to extract distinct features:** Feature extraction is not needed since each pixel contributes for the estimation.
2.4. HYBRID APPROACHES

- **Robustness for scene texture**: Direct methods exploit information from all of the pixels, including the areas that are homogeneous with very small gradient [8]. Therefore, it is beneficial to use direct methods at environments with low distinctive texture [20].

- **Sub-pixel parameter precision**: Motion parameters are estimated with high precision because direct methods use every pixel to estimate a few parameters [8].

- **Outlier rejection**: It is possible to extract the dominant motion in case of outliers using analysis in the frequency-domain [8].

- **Dense reconstruction**: The structure can be densely recovered [8], which would be useful in case of dense scene mapping.

In contrast with their advantages, according to Scaramuzza et al. direct methods are "less accurate than feature based methods and are computationally more expensive" [4]. Therefore, feature based implementations are more common in the literature.

Brightness constancy lies in the core of the direct methods (as in Lucas-Kanade method in Section 2.2.2) where the pixel intensity of a structure assumed to be constant between frames:

\[ I(x, y, t) = I(x + u(x, y), y + v(x, y), t + 1) \]  \hspace{1cm} (2.7)

where \( I(x, y, t) \) is the pixel intensity of the image at pixel coordinate \((x, y)\) and time \(t\) and \((u, v)\) represents the pixel displacement. By reordering the equation and linearizing it using the Taylor series assuming small motion, the brightness constancy constraint can be retrieved as:

\[ 0 = I_x u + I_y v + I_t = \nabla I < u, v > + I_t \]  \hspace{1cm} (2.8)

where \( \nabla I \) is the image gradient and \( I_t = I(x, y, t+1) - I(x, y, t) \) is the temporal derivative. Hence the error function is:

\[ E(p_1, \ldots, p_N) = \sum (\nabla I < u, v > + I_t)^2 \]  \hspace{1cm} (2.9)

where \( p_N \) are the motion model parameters. In order to recover the camera transformation between views, a 3D motion model should be used such as instantaneous velocity motion, discrete 3D motion or plane-parallax model [8]. In order to ensure the small motion assumption, a hierarchical pyramid scheme is used.

2.4 Hybrid Approaches

Hybrid approaches combine feature and appearance based methods in different ways to exploit good aspects of both. Feature based methods are usually more accurate...
and appearance based methods have high precision (as stated in Section 2.3). For certain cases it is beneficial to choose one over the other, such as in the work of Scarammuzza et al. [27] where the feature based methods are used for estimating the translation and appearance based methods are used for estimating the rotation of a ground vehicle.

### 2.5 Random Sample Consensus

Random Sample Consensus (RANSAC) [24] is a method for robust model fitting of a data with outliers. In conventional estimation methods, all of the data is included and used without regarding whether some parts of the data might be "bad" (i.e. datum with gross error). This might result in significantly less accurate estimates.

In order to reject bad datum, an iterative algorithm is used in RANSAC. In the algorithm, $n$ points are randomly picked from the data where $n$ is the minimum number of points required to calculate the model. Using these points, a model is calculated and the subset $S^*$ of the data (called the consensus set) is formed with points that lie within an error tolerance of the calculated model. If the number of points in $S^*$ exceeds the threshold, then the points in $S^*$ are used with a conventional method (such as least squares) to calculate a new model. If there are not enough inlier, the procedure is repeated with new $n$ random points. The algorithm would run for a fixed number of iterations. The best of the calculated models would be the estimate.

**Algorithm** RANSAC

\[
i \leftarrow 0, M_{\text{best}} \leftarrow \text{Null}
\]

\[
n \leftarrow \text{minimum number of points required for the model}
\]

**while** $i < \text{max}$ **do**

\[
S \subset D, \text{ form } S \text{ with } n \text{ random points from data } D
\]

Calculate model $M$ using $S$

\[
S^* \subset D, \text{ form } S^* \text{ with points within error tolerance of } M
\]

**if** $\text{length}(S^*) > \text{threshold}$ **then**

Calculate new model $M^*$ with least squares using $S^*$

**if** $M^*$ is better than $M_{\text{best}}$ w.r.t. some metric **then**

\[
M_{\text{best}} = M^*
\]

**end if**

**end if**

\[
i \leftarrow i + 1
\]

**end while**

**return** $M_{\text{best}}$

RANSAC scheme is commonly used in computer vision algorithms. As stated in previous sections, motion estimation is usually done inside a RANSAC scheme in feature based visual odometry.
2.6 Optimization

In visual odometry, the motion is estimated with consecutive frames. Since the pose is incrementally calculated, the uncertainty at each time stamp is the combination of uncertainties in previous poses and transformations \[28\]. Therefore, the uncertainty in the estimate tends to accumulate, which causes a drift error. In order to reduce the error, optimization should take place not only between the consecutive frames, but between more distant ones.

2.6.1 Key Frames

Due to errors and noise, it is beneficial in terms of depth uncertainty \[4\] to do triangulation using frames with high enough parallax (based on an heuristic), which might not be the case for consecutive frames. Therefore, during the operation a subset of frames with high enough parallax can be extracted as key frames. The optimization process would use these key frames to refine the estimate.

2.6.2 Pose Graph Optimization

The pose optimization problem can be represented with a graph, where poses are the nodes and transformations are the edges. In order to reduce the accumulation of the error, several previous nodes can be used to optimize edge constraints \[28\]:

\[
e_{ij} = \sum \| P_i - T_{ij} P_j \|^2
\]

(2.10)

where \(e_{ij}\) is the edge constraint between nodes \(i\) and \(j\), \(P_i\) is the pose at node \(i\) and \(T_{ij}\) is the transform between poses at nodes \(i\) and \(j\).

With pose graph optimization only the motion parameters are optimized and not the structure.

2.6.3 Bundle Adjustment

Bundle adjustment (BA) can optimize both the motion and the structure. Therefore, it has higher complexity. Usually it is used with a windowed scheme, where only the previous \(N\) frames are included in the optimization with the window size \(N\). It is also possible to use motion-only BA or structure-only BA where either motion parameters or structure parameters are optimized.
Chapter 3

Related Work

The first appearance of the term visual odometry (VO) in an academic resource seems to be in Olson’s study [29]. However, the work of Nister et al. [6] is seen as a landmark for the term in literature [1]. The term VO is relatively new, but estimating ego-motion from visual input dates back more than three decades. Moravec’s study [30] is one of the earliest research on the subject where he used a slider stereo configuration. In his study, Moravec also invented one of the primal corner detector [4] which he called the interest operator. It is currently referred as Moravec corner detector in the literature.

The two-part tutorial on visual odometry by Scaramuzza and Fraundorfer [4, 28] gives the history, describes the concept and references the studies in the area, therefore it is very useful to have a grasp on the subject and its development. Below we will give a brief account of the history and some of the most important related work.

The essence of feature based VO is the motion estimation from 2D-to-2D, 3D-to-2D or 3D-to-3D image/structure correspondences as described in Section 2.2.3. Huang et al. [25] has a review study on the topic for structure and motion.

The 8-point algorithm of Longuet-Higgins [19] is the earliest implementation in computer vision for relative pose estimation from 2D-to-2D feature correspondences. The work of Hartley et al. [18] is a renowned reference book in the area and it includes formulation of the 8-, 7- and 6-point algorithms as well as the computer vision fundamentals for multiple views. For calibrated systems with unconstrained motion, the minimum solution requires five points and Nister’s efficient 5-point solver [20] is seen as the standard method in the literature. Using additional IMU data, Fraundorfer et al. [30] showed that the minimum number of points can be further reduced to three for calibrated systems with unconstrained motion. For constrained motion, Scaramuzza et al. [32] reduced the number of points even down to one by exploiting the non-holonomic constraint, however their method cannot be implemented on drones since it requires planar constrained motion (such as the motion with Ackermann steering [33]).

The Perspective-n-Point (PnP) problem, as described in Section 2.2.3 is the
3D-to-2D problem. The term PnP seems to be coined by Fischler et al. [24] in their well-known study where they also defined the RANSAC method and gave a solution for the P3P case. As Moreno-Noguer et al. [23] points out, the PnP problem and especially the P3P case are studied broadly in the literature. The studies of Dhome et al. [34], Haralick et al. [35] and Gao et al. [36] are some of the examples from the literature for the P3P problem and the studies of Horaud et al. [37] and Quan et al. [38] are the examples for other PnP problems.

The 3D-to-3D problem requires a minimum of three points and Arun et al. [39] shows the solution using least squares and singular value decomposition.

The RANSAC scheme as described in Section 2.5, is commonly used for robust estimation with feature based VO. Newer variations of RANSAC exist to detect bad hypothesis earlier and preempt them as in works of Nister [40] and Chum et al. [41].

For the appearance based VO, the photometric error minimization is the essence as described in Section 2.3. The review of Irani et al. [8] gives a brief on direct methods. The work of Hanna [42] is an early study of ego-motion estimation with direct methods. Lovegrove et al. [26] successfully used direct methods using the input from a rear parking camera of a car where the frames mostly lack of distinctive features. In recent years, RGB-D cameras started to be used for VO applications. The studies of Tykkala [43] and Kerl [44] are examples of using RGB-D cameras for VO.

There are significantly fewer studies in the literature for hybrid approaches in comparison with feature based and appearance based methods. The study of Scaramuzza [27] is the first in the literature to use both approaches where two separate trackers were used. The rotation was estimated using the direct methods and the translation was estimated with feature based methods. The work of Forster et al. [7] is the state of the art hybrid approach and it is investigated in Section 4.1.
Chapter 4

Investigation of the Current Systems

There are several open-source implementations for visual odometry and visual SLAM systems and it is a part of the objective of this project to investigate them. Visual positioning is well established in theory. However, applying theory with hardware and software also has its own problems especially when the systems are desired to be running online with time constraints. Tackling these problems require theoretical and hands-on experience and insight in the domain to write optimized code. The available successful implementations are the product of accumulated experience of their respective researchers. Therefore, it is beneficial to investigate them.

Another reason to investigate the current open-source implementations is to study different approaches to make the implementation decisions on this project. In that manner, not merely the VO systems but also SLAM implementations are studied since the core techniques are mostly shared.

4.1 SVO

Semi-direct visual odometry (SVO) is the current state of the art hybrid VO approach for monocular cameras developed by Forster et al. [7].

4.1.1 Background

The motive behind SVO is to get the advantage of feature based methods with the sub-pixel accuracy of appearance based methods. It is a monocular solution and the method was designed to be used on drones, therefore, it is for unconstrained motion in 6 DoF (degrees of freedom).

The motion estimation happens in three phases in which the first two are based on direct methods and the last one uses feature based methods. In the first phase, ‘sparse model-based image alignment’ is used [7] to retrieve the first estimate of the pose. This step aligns the current frame to the previous frame by applying direct methods on sparse patches located around the features corresponding to the same world points by solving the nonlinear least squares problem;
4.1. SVO

Figure 4.1: Copyright © 2014, IEEE [7]. (4.1a) Image alignment using the patches located around the features. The 2D points in the patches are reprojected from first frame to the second and the transform that minimizes the photometric error is retrieved. (4.1b) Separate patch alignment. (4.1c) Pose refinement with bundle adjustment.

\[
T_{k,k-1} = \arg\min_{T_{k,k-1}} \frac{1}{2} \sum_{i \in R} \| \delta I(T_{k,k-1}, u_i) \|^2
\]

(4.1)

where \( T_{k,k-1} \) is the transform between frame \( k \) and \( k - 1 \), \( R \) is the region in frame \( k - 1 \) where the depth is known, \( u_i \) is the 2D feature in \( R \), and \( \delta I(T_{k,k-1}, u_i) \) is the photometric error:

\[
\delta I(T_{k,k-1}, u_i) = I_k(\pi(T_{k,k-1} \cdot \pi^{-1}(u_i, d_{u_i}))) - I_{k-1}(u_i)
\]

(4.2)

in which \( \pi \) is the projection matrix, \( d_{u_i} \) is the depth at point \( u_i \) and \( I_k(u_i) \) is the pixel value at point \( u_i \) in frame \( I_k \). The visualization of the first phase is in Figure 4.1a.

After the alignment in the first phase, a rough estimate of the transform between frames is retrieved. However, the 3D pose of the world points are not accurate. Hence, this implies that the estimate should be improved. In the second phase, the corresponding image patches in current and previous frames are aligned separately to further reduce the photometric error using:

\[
\hat{u}_i = \arg\min_{\hat{u}_i} \frac{1}{2} \| I_k(\hat{u}_i) - A_i I_r(u_i) \|^2, \quad \forall i
\]

(4.3)
where \( \hat{u}_i \) is the aligned 2D point, \( A_i \) is the warping matrix and \( I_r \) is the reference key frame. The key frame selection in SVO is made using the Euclidean distance between the new frame and other key frames. If the calculated distance passes the threshold a new key frame is added.

This separate alignment of corresponding patches would violate the epipolar constraint, therefore, this step is seen as a "relaxation" step to have sub-pixel accuracy [7]. Due to this violation, there occurs a "reprojection residual" [7]. In the last phase, the pose is again optimized to minimize the residuals with motion-only bundle adjustment (BA). Following to it, structure only and local BA are applied.

Mapping in SVO happens separately and it produces the 3D world points from extracted 2D FAST features to be used in motion estimation. Every world point has a depth represented with a probability distribution which is initialized with high variance and SVO uses a probabilistic Bayesian filter to estimate its depth [7]. The filter updates the points distribution in every new frame. A point would only be placed in the map to be used in motion estimation once it has low enough variance in its depth distribution [7]. Inverse depth parametrization is used in the Bayesian filter.

### 4.1.2 Details and Review

SVO is developed in C++ and is open-source. SVO supports and runs on ROS (see Section 6.2) and it depends on some other open-source packages, for example, to detect FAST features or to use Lie algebra. At any instant, SVO operates on four parallel threads. Motion estimation and mapping happens in two separate threads. Another thread listens to keyboard inputs to get user instructions and the main thread triggers other threads and visualizes the data for the ROS environment. When it ran on the drone’s computer which equipped with an i7-5557U (details in Section 6.1) using the dataset from University of Zurich-RPG [45], SVO was using approximately 28% of the CPU (except visualization program) and 22 MiB of the memory with the monocular input of size \( 752 \times 480 \) at 20 Hz.

Since it is a monocular solution, SVO reconstructs the trajectory up to a scale factor and the absolute scale of the system is unknown. In order for SVO to run, an initial map with enough points should be created. Due to the depth filter, the creation of new 3D points would happen when the uncertainty of the points decreased below a threshold. In order for depth uncertainty to converge, small motion of the body is necessary [7]. In the trials made with the dataset collected with our drone, this is confirmed. When there was fast motion, such as during take off, SVO was either unable to initialize a map or if there was a map, the track was getting lost since new points could not be put in the map. In the latter case, SVO had to relocalize the drone using the current map if it could.
4.2 ORB-SLAM

ORB-SLAM is a state of the art visual SLAM system for monocular cameras developed by Mur-Artal et al. [46].

4.2.1 Background

ORB-SLAM is a full SLAM system using feature based methods. As it is clear from its name, it uses ORB features. The novelty of the method includes [46];

- Extracting the features once for using them in all parts of the system such as in loop closure or tracking instead of extracting new features at every part
- Robust initialization strategy which computes two different models for the scene and uses the one with better score based on the developed heuristic
- Using different graphs for different tasks;
  - **Covisibility graph:** (Figure 4.2b) where each key frame is a node and two nodes share an edge if they observe the same points in the map with the number of points being the edge weight. Using covisibility graph, it is possible to cope with large maps.
  - **Essential graph:** (Figure 4.2c) a sparser sub-graph of covisibility graph with fewer edges. The edges are eliminated based on their weight. It is used in loop closing since the reduced number of connections reduces the optimization time.
- Frequent key frame insertion in tracking and a key frame elimination strategy in local mapping
- Utilizing bag of words (BoW) representation for feature descriptors for place recognition

The system consists of three main parts; tracking, local mapping and loop closing.

In tracking, the input frames are processed, the features are extracted, the motion is estimated and the key frame selection is made. Initialization also happens inside tracking.

The motion estimation in tracking is based on motion-only bundle adjustment for the first rough estimate from the previous frame and then minimizing the reprojection error using the map.

The initialization strategy in ORB-SLAM is based on calculating two different models for the scene. A fundamental matrix is computed using the 8-point algorithm simultaneously with a homography (see Section 2.2.3). Then both models are scored and the selection is made based on a formulated heuristic using the model scores. Then the scene is constructed using the selected model and the initial map is created.
CHAPTER 4. INVESTIGATION OF THE CURRENT SYSTEMS

Figure 4.2: Copyright © 2015, IEEE. (4.2a) Key frames on the trajectory (blue) and map points (black and red). (4.2b) Covisibility graph. (4.2c) Essential graph.

Local mapping has an elimination procedure for key frames and bad map points. Then it inserts new points to the map and optimizes the poses of the current key frame and its neighbours in the covisibility graph as well as the poses of the points in the map observed by those key frames.

For loop closure (and also for relocalization but it happens in the tracking) bag of words representation is utilized. In the loop closure, the similarity of the current key frame is compared with other nodes in the covisibility graph with no connected edges and the candidates are found and the consistency is checked. Then the similarity transform is computed and applied to the current key frame and its neighbours. Then using the essential graph an optimization is made.

### 4.2.2 Details and Review

ORB-SLAM is written in C++ and is open-source. It uses other packages such as DBoW2 and g2o. DBoW2 is a library to utilize the bag of words representation of feature descriptors as well as to create a database and offline vocabulary. g2o is a framework for graph optimization which includes useful methods for nonlinear least square problems such as Levenberg-Marquardt, the one used in ORB-SLAM. Also, the system is implemented in the ROS environment.

The system runs on four main threads, the main thread calls others and also publishes visualization information. The other threads are for tracking, mapping and loop closing. Also during the initialization, the systems computes two motion models parallel for a very short period. When it ran on the drone’s computer which equipped with an i7-5557U (details in Section 6.1) with RPG dataset, ORB-SLAM was using approximately 43% of the CPU and 340 MiB of the memory for the monocular input of size $752 \times 480$ at 20 Hz.

As ORB-SLAM is a monocular solution, the absolute scale of the system is unknown. When testing with the datasets collected with our drone, there occurred problems with repeatability and the map initialization was again a problem as in the case of SVO. Also, ORB-SLAM does not provide an adjustment of the initial
4.3 ORB-SLAM 2

ORB-SLAM 2 [49] is the newer and extended version of ORB-SLAM for stereo and RGB-D cameras.

4.3.1 Background

The core structure of ORB-SLAM is preserved in ORB-SLAM 2. The architecture (e.g. tracking, local mapping and loop closing parts) are kept the same and the system enabled to operate with stereo and RGB-D input. The new features are:

- Separation of close and far points
- Updated key frame insertion criterion also based on the number of close points
- Utilizing BoW representation during tracking to match features
- Localization-only mode

Since the stereo and RGB-D input are supported in the ORB-SLAM 2, the distinction of close and far feature points is made. Feature points which have depth larger than 40 times the baseline are considered far. For RGB-D data, the input is converted to stereo by creating a virtual right coordinate and the baseline is approximated based on the used camera. The distinction is made because far points have low disparity for accurate triangulation and the mentioned threshold is empirically set by the study of Civera et al. [50, cited in 49]. Only the close points are used for motion estimation.

The key frame insertion criterion is updated with the close and far points. A new key frame is inserted if the number of close points in track is decreased below a certain threshold [49]. In ORB-SLAM 2, the BoW representation is used also in tracking part for matching. This was mentioned in the first ORB-SLAM paper [46], however it was not used in the code base for tracking.

Lastly, a localization mode where the local mapping and loop closing parts are idle and the system only tracks the features based on the map. No new key frames or map points are created or inserted and relocalization is used in case of getting lost. This mode can be enabled using the GUI of the viewer, and it might be used if the current map is good.

4.3.2 Details and Review

ORB-SLAM 2 became a stand-alone program and the usage in ROS is made optional, therefore, another thirdparty library is used for visualization.

The system uses four threads. The main thread calls other threads and also does the tracking and the other threads are for local mapping, loop closing and
optionally visualizing. Additionally, in the case of stereo input, the system extracts ORB features in left and right images in parallel threads for every new frame pair. When it ran on the drone’s computer which equipped with an i7-5557U (details in Section 6.1) with RPG dataset \cite{rpg_dataset}, ORB-SLAM 2 was using approximately 40% of the CPU for and 320 MiB of the memory with the monocular input of size $752 \times 480$ at 20 Hz. The CPU usage was approximately the same when tested with collected stereo dataset with both channels of size $752 \times 480$ at 20 Hz. In comparison with the original ORB-SLAM, there is a slight improvement on the CPU usage for the monocular case for the same dataset.

The stereo performance of ORB-SLAM 2 is satisfactory and with absolute scale. Initialization is not a problem as it was for the monocular case. The features can be triangulated directly from the first pair and the map can be initialized without any delay. However, the repeatability is an issue since the results are varying due to the randomization induced by the methods (i.e. RANSAC) as well as the operating system and ROS. RANSAC randomly selects datum for model calculations (see Section 2.5), hence the algorithm is not deterministic. Also, the operating system schedules all the tasks in the computer based on their priorities. The changes in the schedule might result in lost messages (images in this case) in the ROS node’s message filter cache. This would induce a randomization to the system as well.
Chapter 5
Design & Methods

Using the insight from investigating the current systems, the design decisions are made and the methods that are used in the implementation are given in this chapter.

5.1 Design Decisions

The objective of this project includes implementing a lightweight visual odometry system. The requirements from the VO system are:

- To have high rate of estimation
- To have a continuous estimate of the drone’s pose which might be used as input to the on-board sensor fusion of the drone’s FCU
- The pose estimate should be with absolute scale
- Global map consistency is not required but low drift is desired

With SVO; the absolute scale is unknown since it is a monocular solution. Thus, additional sensor data and fusion is required to recover the absolute scale, as well as its drift. Once the absolute scale is recovered, SVO might be quite accurate. However, SVO already runs on four threads and uses approximately 28% of the CPU (see Section 4.1.2). Additional sensor fusion for recovering the scale would consume even more computational power. This approach would be far from lightweight. Initialization is also problematic with SVO and it requires small motion (see Section 4.1.2).

With ORB-SLAM; absolute scale is also a problem as it is monocular as well. The same things apply as SVO in terms of absolute scale recovery. It is globally consistent since it is a full SLAM system, which is not sought. In fact, large loop closures might not be desirable since they make large jumps in the pose estimate. Additionally, it requires a BoW vocabulary to be trained offline. Using BoW is very efficient for place recognition, however the images used for training should be
### Table 5.1: A brief comparison of SVO, ORB-SLAM and ORB-SLAM 2.

<table>
<thead>
<tr>
<th>SVO</th>
<th>ORB-SLAM</th>
<th>ORB-SLAM 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute scale is unknown</td>
<td>Absolute scale is unknown</td>
<td>Absolute scale is observable</td>
</tr>
<tr>
<td>Additional computations are required to</td>
<td>Additional computations are required to</td>
<td>Map can be initialized with the</td>
</tr>
<tr>
<td>retrieve absolute scale and its drift</td>
<td>retrieve absolute scale and its drift</td>
<td>first frame pair</td>
</tr>
<tr>
<td>(i.e. sensor fusion)</td>
<td>(i.e. sensor fusion)</td>
<td></td>
</tr>
<tr>
<td>Map initialization is a problem</td>
<td>Map initialization is a problem</td>
<td></td>
</tr>
<tr>
<td>Small motion is required</td>
<td>Globally consistent, however</td>
<td>Globally consistent, however</td>
</tr>
<tr>
<td></td>
<td>loop closure might result in large</td>
<td>loop closure might result in large</td>
</tr>
<tr>
<td></td>
<td>jumps in pose estimate</td>
<td>jumps in pose estimate</td>
</tr>
<tr>
<td></td>
<td>A trained BoW vocabulary is needed</td>
<td>A trained BoW vocabulary is needed</td>
</tr>
<tr>
<td>Repeatability is a problem</td>
<td>Repeatability is a problem</td>
<td>Repeatability is a problem</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>Localization mode requires a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>built map and heavily dependent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>on relocalization</td>
</tr>
</tbody>
</table>

general enough. Initialization is also an issue as in SVO and lastly ORB-SLAM uses a significant portion of the CPU power (approximately 43%, see Section 4.2.2).

With ORB-SLAM 2; the absolute scale of the scene is known in case of stereo and RGB-D configurations. Therefore no sensor fusion for scale is necessary. Same things apply for loop closure as in ORB-SLAM. Additionally, ORB-SLAM 2 utilizes BoW representation also for tracking (see Section 4.3.1). There is a localization mode in ORB-SLAM 2 where it is more lightweight since mapping and loop closing threads are idle. However, this mode requires an already built map, is heavily dependent on relocalization and it does not create new map points or key frames in order to keep the global consistency.

Considering these points, several decisions are made for the implementation;

- It is determined to use stereo configuration because;
  - The absolute scale is directly observable
  - The features can be triangulated directly from the first image pair
  - Initialization is not a problem as it is in the monocular case
  - Stereo vision degenerates into monocular for far points, yet the implementation is for indoor usage where the distances are usually more limited

- It will be based on ORB-SLAM 2 because;
  - It supports stereo input
  - Handles the close and far points

- The solution would be `orb-vo` where;
5.2 Methods

- The tracking system of the ORB-SLAM 2 is used
- It is limited to two main threads, one for tracking and one for visualization/calling the other thread
- No local mapping or loop closing as in the case of ORB-SLAM 2; the goal is to create a more lightweight system and the global consistency is not required. Thus, the operations for creating a consistent can be removed.
- BoW representation won’t be used, therefore relocalization and tracking would be made using matching functions from the first ORB-SLAM (window search based matching). BoW requires an offline trained vocabulary and it requires some time to load it for every launch. BoW is advantageous for fast matching but it is mostly used in loop closing, which we decided not to include as well.
- Reinitialization would be possible from the last known frame when relocalization fails
- The initial pose of the camera would be able to set to any configuration

Thus, orb-vo would be a simplified and more lightweight visual odometry solution based on ORB-SLAM 2 for stereo input.

5.2 Methods

In this section the methods in orb_vo are presented. As stated in the previous section, ORB-SLAM 2 is the base of our implementation, but some methods from the first ORB-SLAM (window based matching) are also combined. Here, the FAST and BRIEF methods which together forms the ORB method are first given with details. Then it is proceeded with the feature matching and motion tracking methods which originate from both ORB-SLAMs.

5.2.1 FAST Detector

Despite their quality, computational costs of feature detectors such as Harris or SIFT make them unsuitable to be used in real time systems such as visual odometry or SLAM [51]. It is needed to have more rapid solutions. The FAST detector is developed concerning constraints in online systems originally in [52] and then further improved in [51].

The core idea of FAST is simple, in order to detect a point $p$ as a corner, the pixels on a discretized circle (Bresenham circle) around $p$ are checked as shown in Figure 5.1. Several tests are applied using pixel intensities of the interested point $p$ and the pixels on the circle, then a comparison is made using a threshold $t$.

A point $p$ is defined as a corner if there are $n$ connected points on the circle that have intensity values higher than $p + t$ or lower than $p - t$, according to the
original algorithm [52]. In order to make the progress faster and to eliminate points that are not corners, an initial rejection test was applied to four pixels on the circle corresponding to vertical and horizontal directions (pixels 1, 5, 9, 13 in Figure 5.1) [51].

The original algorithm had several issues including; low performance of initial rejection test if $n < 12$, the pixels to be tested were chosen and their tests were ordered based on assumptions, the output of the initial rejection test was not re-used once it is passed and several corners could be detected next to each other [51].

In order to tackle these issues, non-maximal suppression and methods from machine learning are used in a later improvement [51]. To find a corner detector with a desired $n$ [51]:

- FAST algorithm is ran without using the initial rejection test on a dataset of images to detect corners

- For each detected corner, a vector of size 16 is stored containing the pixel $x$’s state on the corresponding circle. Therefore, each vector is divided into three states based on the pixel intensity (brighter than $p + t$, darker than $p - t$ or similar)

- A feature vector $P$ is formed using all the individual vectors

- For all states separately, a decision tree algorithm is used, where the pixel $x$ which gives the most information is selected.

- Finally, machine generated C code is retrieved for the decision tree.
5.2. METHODS

5.2.2 BRIEF Descriptor

Despite their success factors, descriptors such as SIFT and SURF have high dimensional floating point descriptor vectors and are computationally intense. For systems with relatively more limited memory (such as embedded boards) and computational power, they won’t be suitable.

Memory constraints can be tackled with several approaches such as using algorithms for reducing the dimension or hashing the descriptors to binary strings [53]. However, these approaches still require to create descriptors with high dimensions.

The BRIEF descriptor [53] is developed considering the memory and speed constraints. Given a feature point, it produces a descriptor vector of optional number of bits (128, 256 or 512) i.e. a binary string. Since every descriptor element is one bit, BRIEF descriptors occupy 16, 32 or 64 bytes in the memory in comparison with 512 bytes in SIFT [12] (128 floating points) and 256 bytes in SURF [13] (64 floating points).

The descriptor is created using binary tests of pixel pairs of the image patch positioned around the features. Given a patch \( p \), a binary test \( \tau \) is:

\[
\tau(p; x, y) = \begin{cases} 
1, & p(x) < p(y) \\
0, & \text{otherwise} 
\end{cases} 
\] (5.1)

Where \( x = (u_x, v_x)^T \), \( y = (u_y, v_y)^T \) are pixels inside \( p \) and \( p(k) \) is the intensity value at pixel \( k \). For an \( n_d \) dimensional descriptor, \( n_d \) binary tests are applied to the patch.

The selection of the binary tests affects the performance of the descriptor. Given the patch size \( S \) where the origin of the patch coordinate system is in the center, there are several approaches for selecting pixel pairs \((x_i, y_i)\) [53]:

- **Uniform distribution;** \((x_i, y_i)\) are sampled from \(\text{uniform}(-S/2, S/2)\)
- **Gaussian distribution;** \((x_i, y_i)\) are sampled from \(N(0, \sigma^2)\)
- **Separate Gaussians;** \(x_i\) is sampled from \(N(0, \sigma_x^2)\), \(y_i\) is sampled from \(N(x_i, \sigma_y^2)\),
- **Systematic;** \(\forall x_i = (0, 0)^T\) and \(y_i\) takes all values from a grid
- **Random;** \((x_i, y_i)\) are randomly sampled

Once a test strategy is selected, it should sample \( n_d \) test pairs within the patch and the same spatial pair configurations should be used for all features to be described. Then the descriptor is calculated [53]:

\[
f_{n_d}(p) = \sum_{i=1}^{n_d} 2^{i-1} \tau(p; x, y) 
\] (5.2)
One major advantage of BRIEF is the usage of binary vectors. Therefore the comparison of two descriptor vectors can be made using the Hamming distance instead of Euclidian so it would be faster.

There are two main problems with BRIEF, noise and orientation sensitivity [53]. Since it directly uses single pixel values, it is very sensitive to noise, therefore the patch should be smoothed with a kernel prior to the operation [53]. Also, BRIEF is not invariant for rotations, therefore it has low toleration to in-plane rotations.

5.2.3 ORB (Oriented FAST and Rotated BRIEF)

ORB (Oriented FAST and Rotated BRIEF) is an alternative descriptor to SIFT and SURF which combines FAST edge detector and BRIEF binary descriptor with several modifications.

The important highlight of the ORB method are;

- Adding orientation information to FAST features,
- Using a pyramid scheme to achieve scale invariance
- Using Harris corner response [54] to score and order FAST features
- Making BRIEF descriptors rotation-aware
- Developing a learning algorithm to regain high variance on rotation aware BRIEF descriptors

FAST features as described in Section 5.2.1, are not multi-scale and the algorithm does not produce a metric for cornerness. Therefore, in the ORB method, FAST features are detected in a scale pyramid scheme and to avoid edges rather than corners, Harris corner response [54] is used [53].

ORB uses intensity centroids as described by Rosin [56] to produce orientation information of detected FAST features. In order to calculate the intensity centroid, the moments of the image patch are defined as;

\[ m_{pq} = \sum_{x,y} x^p y^q I(x, y) \]  \hspace{1cm} (5.3)

Then the centroid is defined as;

\[ C = \left( \frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \]  \hspace{1cm} (5.4)

The orientation of the feature patch is then the angle of the vector connecting the center of the patch to the intensity centroid;

\[ \theta = \arctan2(m_{01}, m_{10}) \]  \hspace{1cm} (5.5)
5.3. FEATURE MATCHING METHODS

Since the orientation of the feature patch is retrieved, the descriptor should be made "rotation-aware" [55]. Given \((x_i, y_i)\) pixel location pairs at \(n\) binary tests a matrix can be constructed:

\[
S = \begin{bmatrix}
    x_1 & \cdots & x_n \\
    y_1 & \cdots & y_n
\end{bmatrix}
\]  \hspace{1cm} (5.6)

The trick which ORB uses is to rotate location pairs according to the feature patch orientation prior to binary tests [55]:

\[
S_\theta = R_\theta S
\]  \hspace{1cm} (5.7)

where \(R_\theta\) is the rotation matrix of patch orientation \(\theta\).

Then, the binary tests can be performed at rotated pixel locations and the binary string is formed as described in Section 5.2.2. This method is named Steered BRIEF [55].

One problem with this approach is the variance. Rublee et al. [55] states that the descriptor matches lost the high variance property of the original BRIEF once the locations are rotated. In order to tackle this problem, a learning method is developed where the good binary test locations can be retrieved to achieve high variance for Steered BRIEF descriptors. The name of this method is called rBRIEF [55].

5.3 Feature Matching Methods

In orb_vo, when every image pair is received as input, the ORB features are extracted in both images and the matches between left and right channels are found. Then, the matches between different frames are needed to be found for motion estimation. In this section, the matching methods used in orb_vo will be described. The methods here are taken either from ORB-SLAM or its second version.

5.3.1 Stereo Matching

The received images are rectified, i.e. the image planes for left and right channels are aligned to be coplanar such that the corresponding points lie on the same horizontal line.

The matching method is then straightforward, for every feature point on the left image, a match is searched on the same horizontal line within a range on the same scale level. The hamming distance of the feature vectors is used to find the match. Once a match is found, then depth of each feature point can be calculated using the base line and focal length.

5.3.2 Matching with Window Search

Instead of brute-force matching by comparing all the features in both frames, this method looks for matches in a local area with some radius around each feature,
assuming that the motion between frames was small to make the feature stay in the search radius (i.e. window size). For a feature point \( p_i^k \) in frame \( k \), all the feature points \( p_{ij}^{k+1} \) that are found within the window size \( r \) around the image coordinates of \( p_i^k \) in frame \( k+1 \) are compared using the hamming distance of the feature vectors.

5.3.3 Matching with Projection

In order to match image features between two frames or between map points and a frame projection is can be used. It is assumed that the association of the image features and the 3D map points in the first frame is made, this operation would be done in another part of the program.

The 3D map points seen in the first frame are then projected on the second frame as 2D points. Then for each projected point a window search is made. The projective matching is used in the motion tracking.

5.4 Motion Tracking

The motion tracking in \( \text{orb}_v \) is made in two main steps. In the first step, an initial motion estimation is made and in the second step, the pose is refined using the map points. All the motion estimation is based on optimization with bundle adjustment.

For the first step there are two approaches, either using a motion model or using a reference key frame. In both approaches, the structure is similar; an initial guess of the current frame pose is picked, the features matches are found and the pose is refined with motion-only BA. The difference is in formulating the initial guess to perform the optimization.

A motion model is simply the transform between current view and the previous one as visualized in Figure 5.2. Assume that the two camera views \( C_1 \) and \( C_2 \) have the transforms \( T_1 \) and \( T_2 \) according to the world frame. Then the transform between these two views would be:

\[
T_c = T_1^{-1} \cdot T_2 \tag{5.8}
\]

For tracking, the motion between the consecutive frames assumed to be constant. Therefore, the transform \( T_c \) between the current and previous frames is applied to the pose of the current frame to give the initial guess;

\[
T_{\text{guess}} = T_c \cdot T_{\text{previous}} \tag{5.9}
\]

If a motion model is not available or the track was not successful with it, then the second approach is used. In this approach, the pose of the current frame’s reference key frame is picked as the initial guess.

Once the initial guess is computed, the matches between the current and previous frames are found and the guessed pose is refined with BA. The motion model
5.4. MOTION TRACKING

Figure 5.2: Transforms of different views

approach uses projective matching whereas the key frame approach uses window search to find the matches.

The output of the first step then refined using the map points visible in current frame and another BA is performed to calculate the system’s output estimate.
Chapter 6

Practical Information & Implementation

6.1 Hardware

In order to gather datasets and carry out the development, a drone is built and used. The drone is a quadcopter with Quad-X airframe motor configuration as shown in Figure 6.1b. The FCU on the drone is Pixhawk Pixracer X1 [1] which has several embedded sensors and it has control schemes for stable flight of the drone. The motors are driven with Electronic Speed Controllers (ESCs) which receives the control signals from the FCU.

The board inside an Intel NUC5i7RYH mini PC is removed and mounted on the drone. The processor is an Intel i7-5557U with up to 3.40 GHz frequency. Therefore, it is a powerful board for its small size and low weight.

Figure 6.1: (6.1a) Drone in the test environment. (6.1b) Quad-X airframe configuration. Image retrieved from [57] under CC BY 4.0 license.
Various cameras including IDS UI-1221 and UI-3241 are used during the investigation. Later, it is decided to use a Duo MLX stereo camera which is factory-calibrated and has global shutters.

6.2 Software

ROS (Robot Operating System) framework is used throughout the project. ROS is an open-source middleware which was first released in 2009 by Willow Garage. Since then, it showed an increasing popularity in academia and applications. ROS is designed to be "peer-to-peer, tool based, multi-lingual, thin, free and open-source" hence it provides libraries for communicating different parts of the project (nodes) and features for debugging.

As discussed in Section 5.1, ORB-SLAM 2 served as a software basis for the implementation.

6.3 Dataset Collection

Datasets are collected using the camera(s) mounted on the drone while it was flying in a test area where it was trackable with the OptiTrack motion capture system. The system has infrared cameras mounted around the test area such that it can detect and track the pose of the reflective objects. Therefore, several small reflective objects are placed on the drone and they are identified for the motion capture system. The output of the motion capture track, which would serve as the ground truth, as well as the camera on the drone are recorded using ROS as bag files.

6.4 Implementation

The development of the orb_vo system is made following the design decisions in Section 5.1. A new package is created and the necessary parts from the code bases of ORB-SLAM and ORB-SLAM 2 are fetched. The modifications and additions are made in C++ and the tests are applied using the collected dataset.
Chapter 7

Evaluation

7.1 Criterion
The system is designed to be relatively more lightweight with the price of consistency. However, it is assumed that the accuracy of the system would be improved if the pose fusion of the drone’s FCU is utilized. Therefore, the criterion from the visual odometry estimate are;

- The standard deviations of position errors should be less than the drone’s diameter (0.25 m in this case), ideally even smaller
- The mean of the position errors should be less than the drone’s radius (0.125 m)
- The estimate should be continuous, therefore it should handle getting lost. This is designed to be achieved by the reinitialization property developed for orb_vo. If the system loses the track and can’t relocalize with the map, it is expected to initialize a new map with the current scene at the last known pose
- The estimation rate should be high, so that the estimation can happen between frames in order to keep the track of every input stereo couple

7.2 Experimentation
Since a more lightweight system is implemented, it is desired to answer whether its performance would be similar to its 'parent' method ORB-SLAM 2. In that manner three experiments with two different datasets are performed.

7.2.1 Experiment 1
In this experiment the same collected dataset is ran on both orb_vo and ORB-SLAM 2. The dataset included stereo images of size 752 × 480 in both channels at
7.2. EXPERIMENTATION

Table 7.1: Standard deviations and means of the errors in x, y and z coordinates for the orb_vo in Experiment 1. Units are in meters. Row color code; green rows are the results that meet the criterion, yellow are borderline (i.e. not within the range but close), red are not acceptable.

<table>
<thead>
<tr>
<th>Launch</th>
<th>$\sigma_x$</th>
<th>$\sigma_y$</th>
<th>$\sigma_z$</th>
<th>$\mu_x$</th>
<th>$\mu_y$</th>
<th>$\mu_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.23697</td>
<td>0.21971</td>
<td>0.21317</td>
<td>-0.11054</td>
<td>0.093933</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.11679</td>
<td>0.11586</td>
<td>0.10929</td>
<td>-0.094728</td>
<td>0.11186</td>
<td>0.10037</td>
</tr>
<tr>
<td>3</td>
<td>0.099944</td>
<td>0.16061</td>
<td>0.21927</td>
<td>0.074016</td>
<td>0.056948</td>
<td>-0.059269</td>
</tr>
<tr>
<td>4</td>
<td>0.19436</td>
<td>0.13792</td>
<td>0.19193</td>
<td>0.28571</td>
<td>0.04368</td>
<td>-0.041978</td>
</tr>
<tr>
<td>5</td>
<td>0.14586</td>
<td>0.18115</td>
<td>0.16898</td>
<td>0.028677</td>
<td>0.068355</td>
<td>0.13229</td>
</tr>
<tr>
<td>6</td>
<td>0.27603</td>
<td>0.2277</td>
<td>0.22051</td>
<td>0.66072</td>
<td>-0.43069</td>
<td>0.27201</td>
</tr>
<tr>
<td>7</td>
<td>0.30435</td>
<td>0.20167</td>
<td>0.25905</td>
<td>0.72946</td>
<td>-0.21344</td>
<td>0.43014</td>
</tr>
<tr>
<td>8</td>
<td>0.19844</td>
<td>0.15324</td>
<td>0.21975</td>
<td>-0.096479</td>
<td>0.13874</td>
<td>0.19191</td>
</tr>
<tr>
<td>9</td>
<td>0.099384</td>
<td>0.12088</td>
<td>0.14469</td>
<td>-0.030288</td>
<td>0.013515</td>
<td>0.05109</td>
</tr>
<tr>
<td>10</td>
<td>0.10949</td>
<td>0.13127</td>
<td>0.245</td>
<td>-0.063174</td>
<td>0.057712</td>
<td>-0.0015017</td>
</tr>
</tbody>
</table>

Table 7.2: Standard deviations and means of the errors in x, y and z coordinates for the ORB-SLAM 2 in Experiment 1. Units are in meters. Row color code; green rows are the results that meet the criterion, yellow are borderline (i.e. not within the range but close), red are not acceptable.

<table>
<thead>
<tr>
<th>Launch</th>
<th>$\sigma_x$</th>
<th>$\sigma_y$</th>
<th>$\sigma_z$</th>
<th>$\mu_x$</th>
<th>$\mu_y$</th>
<th>$\mu_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.20024</td>
<td>0.37411</td>
<td>0.33109</td>
<td>0.076162</td>
<td>-0.014255</td>
<td>0.45632</td>
</tr>
<tr>
<td>2</td>
<td>0.11008</td>
<td>0.099345</td>
<td>0.12144</td>
<td>-0.0095578</td>
<td>0.006543</td>
<td>-0.10112</td>
</tr>
<tr>
<td>3</td>
<td>0.37464</td>
<td>0.20791</td>
<td>0.44928</td>
<td>1.0561</td>
<td>-0.57665</td>
<td>0.77542</td>
</tr>
<tr>
<td>4</td>
<td>0.10332</td>
<td>0.10625</td>
<td>0.12347</td>
<td>-0.032516</td>
<td>0.046114</td>
<td>-0.10381</td>
</tr>
<tr>
<td>5</td>
<td>0.1178</td>
<td>0.1228</td>
<td>0.1508</td>
<td>-0.019737</td>
<td>0.031782</td>
<td>-0.1176</td>
</tr>
<tr>
<td>6</td>
<td>0.12475</td>
<td>0.11525</td>
<td>0.12981</td>
<td>-0.00092652</td>
<td>-0.0070255</td>
<td>-0.14138</td>
</tr>
<tr>
<td>7</td>
<td>0.12975</td>
<td>0.14391</td>
<td>0.13797</td>
<td>0.03055</td>
<td>0.043081</td>
<td>-0.052403</td>
</tr>
<tr>
<td>8</td>
<td>0.10844</td>
<td>0.09945</td>
<td>0.14019</td>
<td>0.039304</td>
<td>-0.013058</td>
<td>-0.053999</td>
</tr>
<tr>
<td>9</td>
<td>0.12236</td>
<td>0.11035</td>
<td>0.18836</td>
<td>0.35542</td>
<td>-0.21002</td>
<td>-0.024025</td>
</tr>
<tr>
<td>10</td>
<td>0.14663</td>
<td>0.15947</td>
<td>0.12402</td>
<td>-0.014628</td>
<td>0.15766</td>
<td>-0.078454</td>
</tr>
</tbody>
</table>

20 Hz, where the drone started from the ground, took off, flew and landed.

As mentioned in Section 4.3.2 the results of the ORB-SLAM 2 was varying between launches for the same input due to the randomization induced by RANSAC, ROS and operating system. Therefore, variance in the repeatability was expected in orb_vo as well. In that manner, the experiment is performed 10 consecutive times for each system. The results for ORB-SLAM 2 and orb_vo are presented in Tables
7.2 and 7.1 respectively including the standard deviations and means of the errors in x, y and z coordinates.

Figure 7.1: Estimated position from orb_vo (purple) and ground truth (green) in 3D and 2D views for Experiment 1. 7.1c and 7.1a are the results of launch 7 and 7.1d and 7.1b are the results of launch 9.

All launches with the orb_vo gave a continuous estimate; in some of the launches the track was not lost at all and in the others relocalization or reinitialization was performed. Therefore this requirement is checked. The output rate was also approximately 20 Hz or slightly less, hence the requirement on the rate is also checked. The orb_vo was using approximately 34\% of the CPU and 30 MiB of the memory for this dataset whereas the ORB-SLAM 2 was using approximately 40\% of the CPU and 320 MiB of the memory. Therefore it can be said that the CPU load is only slightly reduced in the orb_vo, but the memory consumption is significantly lower. This is due to the removal of loop closing and local mapping operations in the orb_vo.
7.2. EXPERIMENTATION

Figure 7.2: Estimated position from ORB-SLAM 2 (purple) and ground truth (green) in 3D and 2D views for Experiment 1 launch 3. The bad performance is due to the rapid error accumulation in pitch error due to fast motion during take off.

When looking at the Table 7.1, it can be observed that the results vary drastically between launches. Only four of the launches have results that are acceptable according to the criterion and in two launches the results are not within the range but close. In the other five launches, the system output is not acceptable and have larger errors. The trajectory estimates of launches 7 and 9 together with the ground truth from motion capture system are plotted in Figure 7.1 as an example for good (launch 7) and bad (launch 9) launches. Also the errors in x, y and z are plotted in Figure 7.3. The bad performance in launch 7 is the result of huge error accumulation in pitch angle during take off. This can be seen by the tilt in the trajectory in Figure 7.1a. Even though the performance of launch 9 is within the tolerance defined in criterion, it can be seen that it diverges from the ground truth in x and y at many parts of the trajectory. We believe the low performance is a result of the sudden motion during take off and landing where they largely contributed to the error accumulation. From these results of the launches, it is concluded that orb_vo is very sensitive for fast motion.

The results of the launches for ORB-SLAM 2 as shown in Table 7.2 are slightly better than the orb_vo. Five launches have results matching the criterion, three of them have close values and two of them have not acceptable results. In fact, it is observed that the bad results occur during take off as well. As seen in Figure 7.2 the estimates diverges during take off due to rapid error accumulation in pitch angle. It is apparent that orb_vo inherited the sensitivity to fast motion from ORB-SLAM 2.

From Experiment 1, it is concluded that the orb_vo gives continuous estimation with high rates and its performance of was slightly different than ORB-SLAM 2.
Also, it is observed that both methods are sensitive to fast motion. Therefore, in order to test whether fast motion such as during take off or landing really reduces the performance, it is proceeded with Experiment 2.

### 7.2.2 Experiment 2

Experiment 1 showed that both ORB-SLAM 2 and hence orb_vo are sensitive to fast motion and that orb_vo has a similar performance to ORB-SLAM 2. Thus, Experiment 2 is designed to see whether the performance of orb_vo would be better without fast motion. In that manner, the same dataset used in Experiment 1 is edited and the take off and landing parts are removed.

The experiment is again launched 10 consecutive times and the results are presented in Table 7.3 including the standard deviations and means of the errors in x, y, and z coordinates. The results are indeed confirmed that when the take off and landing parts are removed from the dataset, the performance of the orb_vo is significantly improved where nine out of ten launches had successful estimates. It is also observed that, there was an improvement even on the bad results (i.e., they had lower errors in comparison with the bad results in Experiment 1). The only bad result Experiment 2 was launch 2 where the error occurred due to a bad reinitialization. The trajectory estimates of launches 2 and 8 together with the ground truth from motion capture system are plotted in Figure 7.4 as an example for one
7.2. EXPERIMENTATION

Table 7.3: Standard deviations and means of the errors in $x$, $y$ and $z$ coordinates for the orb_vo in Experiment 2. Units are in meters. Row color code; green rows are the results that meet the criterion, yellow are borderline (i.e. not within the range but close), red are not acceptable.

<table>
<thead>
<tr>
<th>Launch</th>
<th>$\sigma_x$</th>
<th>$\sigma_y$</th>
<th>$\sigma_z$</th>
<th>$\mu_x$</th>
<th>$\mu_y$</th>
<th>$\mu_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.068527</td>
<td>0.13737</td>
<td>0.045115</td>
<td>-0.0058523</td>
<td>0.045494</td>
<td>0.070439</td>
</tr>
<tr>
<td>2</td>
<td>0.1424</td>
<td>0.29403</td>
<td>0.12282</td>
<td>-0.14914</td>
<td>0.28588</td>
<td>-0.065352</td>
</tr>
<tr>
<td>3</td>
<td>0.10862</td>
<td>0.12672</td>
<td>0.030344</td>
<td>-0.0072013</td>
<td>0.079969</td>
<td>0.051467</td>
</tr>
<tr>
<td>4</td>
<td>0.052717</td>
<td>0.15643</td>
<td>0.041676</td>
<td>-0.0072413</td>
<td>0.079969</td>
<td>0.051467</td>
</tr>
<tr>
<td>5</td>
<td>0.10172</td>
<td>0.12848</td>
<td>0.15137</td>
<td>0.053428</td>
<td>-0.079969</td>
<td>0.051467</td>
</tr>
<tr>
<td>6</td>
<td>0.079358</td>
<td>0.15651</td>
<td>0.041026</td>
<td>-0.0729</td>
<td>-0.012643</td>
<td>0.015634</td>
</tr>
<tr>
<td>7</td>
<td>0.094399</td>
<td>0.13407</td>
<td>0.036734</td>
<td>-0.063899</td>
<td>0.046242</td>
<td>0.064596</td>
</tr>
<tr>
<td>8</td>
<td>0.060351</td>
<td>0.088649</td>
<td>0.038779</td>
<td>0.0067014</td>
<td>0.020807</td>
<td>0.069001</td>
</tr>
<tr>
<td>9</td>
<td>0.083402</td>
<td>0.1103</td>
<td>0.045306</td>
<td>-0.04191</td>
<td>0.040936</td>
<td>0.053963</td>
</tr>
<tr>
<td>10</td>
<td>0.065522</td>
<td>0.11873</td>
<td>0.095933</td>
<td>-0.0033448</td>
<td>-0.049263</td>
<td>0.060079</td>
</tr>
</tbody>
</table>

of the good (launch 8) and bad (launch 2) launches.

The results of Experiment 2 confirm the findings in Experiment 1 that orb_vo is sensitive to fast motion and show that the its performance is increased for the same dataset if take off and landing parts are removed. In order to test validity of this situation on other datasets, it is proceeded with Experiment 3.

7.2.3 Experiment 3

Experiment 3 is performed in order to test the performance of orb_vo on other datasets with no fast motion (i.e. take off or landing). The new dataset is ran with the orb_vo 10 consecutive times as in previous experiments. The results are presented in Table 7.4 including the standard deviations and means of the errors in $x$, $y$ and $z$ coordinates. The results of Experiment 3 confirmed that orb_vo performance would be successful most of the time if the drone’s motion are not fast such as during take off or landing. The trajectory estimates of launches 8 and 9 together with the ground truth from motion capture system are plotted in Figure 7.5 as an example for one of the good (launch 9) and bad (launch 8) launches.
Figure 7.4: Estimated position from orb_vo (purple) and ground truth (green) in 3D and 2D views for Experiment 2. 7.4c and 7.4a are the results of launch 2, the only bad launch in the experiment, and 7.4d and 7.4b are the results of launch 8.
7.2. EXPERIMENTATION

Table 7.4: Standard deviations and means of the errors in $x$, $y$ and $z$ coordinates for the $\text{orb}_\text{vo}$ in Experiment 3. Units are in meters. Row color code; green rows are the results that meet the criterion, yellow are borderline (i.e. not within the range but close), red are not acceptable.

<table>
<thead>
<tr>
<th>Launch</th>
<th>$\sigma_x$</th>
<th>$\sigma_y$</th>
<th>$\sigma_z$</th>
<th>$\mu_x$</th>
<th>$\mu_y$</th>
<th>$\mu_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.080017</td>
<td>0.10744</td>
<td>0.052252</td>
<td>-0.012446</td>
<td>-0.023846</td>
<td>-0.013542</td>
</tr>
<tr>
<td>2</td>
<td>0.082543</td>
<td>0.18531</td>
<td>0.043161</td>
<td>-0.009497</td>
<td>0.023948</td>
<td>-0.011039</td>
</tr>
<tr>
<td>3</td>
<td>0.084762</td>
<td>0.17248</td>
<td>0.054499</td>
<td>-0.024707</td>
<td>-0.0039513</td>
<td>0.008045</td>
</tr>
<tr>
<td>4</td>
<td>0.078681</td>
<td>0.10362</td>
<td>0.062724</td>
<td>-0.022608</td>
<td>-0.06526</td>
<td>0.011154</td>
</tr>
<tr>
<td>5</td>
<td>0.075251</td>
<td>0.11929</td>
<td>0.046984</td>
<td>-0.011118</td>
<td>-0.033347</td>
<td>-0.0059382</td>
</tr>
<tr>
<td>6</td>
<td>0.07451</td>
<td>0.11714</td>
<td>0.046235</td>
<td>-0.032623</td>
<td>-0.057045</td>
<td>-0.022709</td>
</tr>
<tr>
<td>7</td>
<td>0.066879</td>
<td>0.1155</td>
<td>0.042769</td>
<td>-0.037777</td>
<td>-0.018982</td>
<td>-0.010194</td>
</tr>
<tr>
<td>8</td>
<td>0.10177</td>
<td>0.15774</td>
<td>0.099341</td>
<td>0.016746</td>
<td>-0.12785</td>
<td>0.039563</td>
</tr>
<tr>
<td>9</td>
<td>0.065706</td>
<td>0.094102</td>
<td>0.045771</td>
<td>-0.035596</td>
<td>-0.024488</td>
<td>-0.0088036</td>
</tr>
<tr>
<td>10</td>
<td>0.062196</td>
<td>0.12442</td>
<td>0.048592</td>
<td>-0.055278</td>
<td>-0.040051</td>
<td>-0.017614</td>
</tr>
</tbody>
</table>
Figure 7.5: Estimated position from orb_vo (purple) and ground truth (green) in 3D and 2D views for Experiment 3. (a) and (b) are the results of launch 8, the only bad launch in the experiment, and (c) and (d) are the results of launch 9.
Chapter 8

Conclusion

8.1 Summary

Motion estimation from visual input is a rich study area. There are many methods developed in the last three decades, which requires diligence to study. The literature in the area is studied and fundamentals as well as related works are reported.

The state of the art implementations, SVO, ORB-SLAM and ORB-SLAM 2, are investigated. Based on this investigation and desired features of the positioning system, design decisions are made. A new visual odometry solution called orb_vo is implemented based on ORB-SLAM 2.

orb_vo is a "relaxed" version of the original system where it was aimed to have a more lightweight solution that is capable of giving continuous estimates in the cost of consistency. Several experiments are launched with orb_vo on the two datasets and the results are shown. Based on the outcomes of the experiments, it is concluded that orb_vo is capable of giving estimates meeting the criterion most of the time for smooth motions.

8.2 Discussion

The goal of the implementation was to have a relatively more lightweight solution in terms of CPU load and memory consumption. This is achieved in the design by compromising the global consistency and removing operations for loop closing and local mapping from the ORB-SLAM 2 method.

Experiment 1 showed that, orb_vo fits the criterion for estimation rate and continuous estimation. The latter is an improvement to the ORB-SLAM 2, where the system is designed to try relocalization in case it got lost. This is to keep the consistency, however the system can stop giving any estimates if it fails to relocalize itself. The orb_vo reinitializes the map on the last frame’s pose to tackle this situation. Experiment 1 also pointed out that the orb_vo inherited the issues with repeatability and sensitivity for fast motion such as during take off or landing.
Experiment 2 and 3 confirmed that, the `orb_vo` is capable of giving successful estimates most of the time if the motion is not fast, and the performance is significantly improved when the aggressive motion is removed from the dataset. This is a constraint on the robustness to the motion. However, this issue might be tackled by launching the `orb_vo` after the drone has taken off. Since the landing is usually the end of the drone’s trajectory, having bad estimates during or after the landing should not be a problem.

Based on the smooth motion assumption, we believe that the output of the `orb_vo` should be able to fly the drone safely in the indoor environments once the on-board sensor fusion of the drone’s FCU utilizes it.

The `orb_vo` is in fact more lightweight than ORB-SLAM 2, however the difference in CPU load is approximately 6%, a slight difference. The memory consumption, on the other hand, is significantly lower for the `orb_vo` which would be useful for systems memory constraints. The choice between `orb_vo` and ORB-SLAM 2 is based on the design constraints. If it is affordable, using ORB-SLAM 2 might be favorable due to consistency.

### 8.3 Conclusion

Based on the investigation of the state of the art methods and the development of the designed system it is concluded that:

- Robust visual pose estimation is a highly challenging problem. Even the state of the art solutions tend to fail from time to time.

- Repeatability and sensitivity to fast/aggressive motion are common issues in the current visual positioning systems. The repeatability of the results vary between different launches for the same input due to the nature of methods and the software frameworks. The sensitivity to fast motion is a shared problem because either it tends to fail the initialization process (as in the case of SVO), or it might result in rapid error accumulation.

- It is indeed possible to achieve a similar performance to the current ORB-SLAM 2 system with a more lightweight solution in terms of CPU and memory usage called the `orb_vo` with the assumption of smooth motion of the drone.

- Based on the assumption on the motion, the developed `orb_vo` solution is a successful method and the news value of this project.

### 8.4 Future Work

It is assumed that the pose fusion of the drone’s FCU would improve the accuracy. This assumption can be tested on the drone. The performance of the fusion to fly the drone indoors can then be seen.
8.4. FUTURE WORK

To improve the performance and to tackle with the sensitivity to fast motion during landing and take off, it might be beneficial to use an additional distance sensor mounted at the bottom of the drone. In this way, the errors during take off might be reduced. Also, this information can be used for system to automatically adjust the initial height in case of launching the system after take off. Currently, the initial position of the drone is supplied manually with a parameters file.

Another way to improve performance might be using IMU data. The drone’s FCU already processes the IMU data from its raw values. Therefore, incorporating this information in a lightweight way to the system can be investigated.
Appendix A

Ethics, Society & Sustainability

A.1 Ethics

Drones are heavily subject to ethical discussions due to the variety of applications including in military and law enforcement. Military drones are increasingly adopted for use, by 2009 more than 50 nations have been using drones for militaristic applications [60]. Singer states that "the US Air Force now trains more unmanned-systems operators than fighter and bomber pilots combined" [61]. There are some aspects to concern in this manner. Governments point out their precision in strike and provision of accurate surveillance to avoid causalities whereas the opponents argue with their lack of capacity to distinguish targets from innocents [62, cited in 63]. From Kantianist perspective, if all the governments were significantly investing into military drones, one might argue that any possible war would be without causalities. However, this would be very far from realistic since universalizing this maxim would end up in an arms-race where resources otherwise be allocated for development, education and health would spent on military. Also since the economic volumes of countries are different, there would always be better powers which would possibly result in more wars and eventually causalities. Another aspect in military drones is the "playstation" effect which describes a mentality of pilots who teleoperate the drones from a distance might objectify the situation less realistically and violate "acceptable ethical practice" [64] [65].

From another point of view, higher defense and security but lower causalities are possible with military drones. Ethical analyzes are related to the motives of military drone usage. One might argue that, it would be ethical to use them in case of a direct attack for defense.

There are discussions on civil or non-militaristic drone applications as well. Issues on data and personal privacy is important. It is very common to drones to have visual sensors on-board. Using cameras on drones might damage the personal privacy. Culver [66] gives an example with two situations; in one case a drone from a news agency takes photographs to map and information from a rural area, in the second case the drone is recording videos of a celebrity in her property
from outside. Culver concludes as "through a utilitarian framework, then, use of drones in newsgathering may be justified when organizations: avoid infringing moral or legal rights of their subjects; consider the connection between their information gathering and the level of scrutiny subjects deserve; keep their promises to subjects and communities; and approach their work with impartiality" [66].

A.2 Society

Increasing popularity of drones raise questions on societal aspects in terms of safety, privacy, labor and regulations. Safety (in terms of military drones) and privacy are addressed from ethics point of view in Section A.1. The effects of drone technology to society will be emphasized in this section.

Concerns about safety includes militaristic usages of drones, their airworthiness and interference with air traffic. Using more unmanned vehicles for military definitely affect the society on perceiving military and security. According to Boucher’s study, which focuses on the society’s visions and acceptance criterion on drones, the visions of sample groups were very positive on military drones and support for them was remarkably higher than for personal or recreational drones [67]. Applications perceived as 'life saving' seem to make drones more acceptable. Airworthiness of drones is another safety issue. There are many commercial drones available in the market for ranging prices, also it is possible to make one with kits. The stability of drone software (such as controllers) or hardware are up to the user or owner most of the time. Hence possible accidents might affect the society when the drone technology become even more common. For example Li-Po batteries are commonly used with drones which can be dangerous when used without enough care or knowledge. Another safety related concern for society is the possible interference of drones with air traffic. However, restrictions on drone usage around airports seem to be already in effect.

Almost every drone has a camera or can be equipped with one. This makes them suitable for surveillance and intelligence. Privacy is rather a vague term and legal basis to define the boundaries might be unable react to technological developments with the same pace. Also, anybody can access to a drone with visual sensors, the use of personal drones still lack of detailed regulations. The excessive number of drones with cameras in the environment for either governmental surveillance or personal tasks might result in the effect on society which is described by Bentham (as in Panopticon building) and Foucalt where individuals act as if they are being observed all the time even if they are not [64] [68].

Labor is an aspect which cannot be overlooked as in the case of all robotic systems. Drones are being used in movie sector, aerial mapping, agriculture and recently transportation of goods. There are concepts where they can be used to built structures and co-operate with other drones as a swarm. The use of drones as labor can create new jobs but also might lead to unemployment in some sectors. In the above mentioned study, Boucher states about the study groups that "while
many accepted that the development of a European civil drone sector would create jobs for operators, manufacturers and repair work, they expected job losses elsewhere, particularly in the traditionally working-class sectors such as factories and deliveries.” [67]. There are some thoughts on unemployment due to autonomous/automation systems. Bill Gates states that there should be taxes for companies on using autonomous/automation systems [69], whereas Elon Musk justifies the idea of “universal basic income” [70].

Regulation and understanding the use of technology are major factors. Many of the aspects which seem to have negative effects on society can be overcome with following the technology and creating regulations accordingly. Because there are also many useful applications and effects of drone technology as well. Some of them are investigated in the next section in connection with sustainability.

A.3 Sustainability

The words “sustainability” or “sustainable” are being frequently used in media, news, books and articles. The definitions of sustainability and sustainable development should be understood as well as the differences. Sustainable development refers to a global level with clear system, function and time parameters whereas sustainable use can be refer to any specific system [71].

Drone technology can contribute sustainable development in several ways; by using in maintenance of renewable energy plants, by using renewable energy on themselves, by monitoring the wildlife and environment and by agricultural applications.

Maintenance of renewable energy plants such as huge solar farms or wind turbines can be made easier and possibly cheaper with drone technology. Even if the direct interaction might not be possible due to limited payload and battery life, just using them for investigation is helpful. Solar powered drones are designed and in use, therefore there won’t be carbon emission from such drones. Drones can be used to monitor environment and wild life for protection and data gathering for forming strategies.
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


