AUTONOMOUS VISUAL-INERTIAL
NAVIGATION FOR QUADROTOR MAVs

A Thesis
Submitted to the Faculty of Graduate Studies and Research In Partial Fulfillment of the Requirements For the Degree of Master of Applied Science in Electronic Systems Engineering University of Regina

By
Shehryar Khurhsid
Regina, Saskatchewan, Canada
April, 2016

Copyright © 2016: S. Khurshid
UNIVERSITY OF REGINA

FACULTY OF GRADUATE STUDIES AND RESEARCH

SUPERVISORY AND EXAMINING COMMITTEE

Shehryar Khurshid, candidate for the degree of Master of Applied Science in Electronic Systems Engineering, has presented a thesis titled, **Autonomous Visual-Inertial Navigation for Quadrotor Mavs**, in an oral examination held on April 7, 2016. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner: Dr. David Gerhard, Department of Computer Science

Supervisor: Dr. Raman Paranjape, Electronic Systems Engineering

Committee Member: Dr. Mehran Mehrandezh, Industrial Systems Engineering

Committee Member: Dr. Irfan Al-Anbagi, Electronic Systems Engineering

Chair of Defense: Dr. Chris Oriet, Department of Psychology
Abstract

The aim of this thesis is to develop a system which would enable a quadrotor MAV (Micro Aerial Vehicle) to estimate its position and orientation and to autonomously navigate in unknown environments using vision as the primary source of information. To navigate in three-dimensional space, an autonomous MAV should not only possess knowledge of its current position and orientation (pose for short), but also of the world around it. While the former can be obtained using a GPS for large outdoor environments and the latter can be provided as a map, a truly autonomous navigation system should enable an MAV to infer its pose in indoor, GPS-denied environments using only the on-board sensors. While images from a camera are rich in data, they are devoid of any depth information. Extracting depth information from a single camera therefore requires the presence of reference objects with known geometry such as artificial fiducial markers in the field of view, or state-of-the-art monocular structure from motion techniques.

In this thesis, we will study and develop solutions for environments which the MAV possesses no a priori information. We will present our modular approach to the problem of autonomous navigation which comprises of three separate blocks. The first block uses a state-of-the-art monocular simultaneous localization and mapping algorithm to transform the camera into a real time pose sensor. The second block uses an extended Kalman filter to refine the pose information from the camera by fusing it with data from the MAV’s onboard sensors. The third block uses a proportional-integral-derivative controller to generate control commands for the MAV.

We implemented our system on a commercially available quadrotor MAV and tested it in real world scenarios. The accuracy of our system will be compared against a highly accurate motion capture system and the findings will be presented/analyzed.
Acknowledgements

I would like to thank my supervisor Dr. Raman Paranjape and my parents for their unrelenting patience, support, guidance and encouragement over the last few years.
In the land of the blind, the one-eyed man is king

Desiderius Erasmus
## Contents

1 Introduction .......................... 1
   1.0.1 Problem Statement ............... 5
   1.0.2 Outline ......................... 5
   1.0.3 Contributions ................... 6

2 Quadrotor Dynamics .................. 7
   2.1 Quadrotor MAV Coordinate Frames ... 7
      2.1.1 Inertial Frame $\mathcal{K}_i$  ... 7
      2.1.2 MAV Frame $\mathcal{K}_{m}$ ........ 8
      2.1.3 MAV-1 Frame $\mathcal{K}_{m1}$ .... 8
      2.1.4 MAV-2 Frame $\mathcal{K}_{m2}$ .... 9
      2.1.5 Body Frame $\mathcal{K}_b$ ........ 10
   2.2 Forces and Moments ............... 12
   2.3 Quadrotor State Model ............ 14

3 Multiple-View Geometry ............ 15
   3.1 Notation .......................... 16
   3.2 Camera Model ...................... 17
      3.2.1 Pin-hole Model ................. 17
      3.2.2 Extrinsic Parameters .......... 18
      3.2.3 Intrinsic Parameters .......... 20
   3.3 Single-View Geometry ............. 22
      3.3.1 2D-3D Pose Estimation .......... 22
      3.3.2 Non-Linear Refinement .......... 25
   3.4 Two-View Geometry ................ 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>Estimating the Essential Matrix</td>
<td>27</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Recovering Camera Pose from Essential Matrix</td>
<td>28</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Triangulating Landmarks</td>
<td>28</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Non-linear Refinement</td>
<td>29</td>
</tr>
<tr>
<td>3.5</td>
<td>Feature Points</td>
<td>29</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Finding feature points</td>
<td>29</td>
</tr>
<tr>
<td>3.5.2</td>
<td>FAST Corner Detector</td>
<td>31</td>
</tr>
<tr>
<td>3.5.3</td>
<td>LoG Blob Detector</td>
<td>31</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Multiscale Feature Point Detection</td>
<td>31</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Tracking a Feature Point</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>Monocular SLAM</td>
<td>33</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Algorithm Outline</td>
<td>35</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Map Initialization</td>
<td>36</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Map Optimization</td>
<td>36</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Adding Keyframes and Landmarks</td>
<td>37</td>
</tr>
<tr>
<td>3.6.5</td>
<td>Further Map Managing</td>
<td>37</td>
</tr>
<tr>
<td>3.7</td>
<td>Camera Tracking</td>
<td>37</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Pose Estimation</td>
<td>38</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Tracking Recovery</td>
<td>38</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Identifying New Keyframes</td>
<td>38</td>
</tr>
<tr>
<td>3.7.4</td>
<td>Further Tracking Aspects</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>Kalman Filter Sensor Fusion</td>
<td>40</td>
</tr>
<tr>
<td>4.1</td>
<td>Linear Kalman Filter</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Extended Kalman Filter</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Unscented Kalman Filter</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>Sequential Monte Carlo Filters</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>PID Controller</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>The Proportional Term</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>The Derivative Term</td>
<td>47</td>
</tr>
<tr>
<td>5.3</td>
<td>The Integral Term</td>
<td>47</td>
</tr>
</tbody>
</table>
6 Implementation 48

6.1 Outline ................................................. 48
6.2 Software Architecture ................................ 49
6.3 Visual SLAM ............................................. 50
  6.3.1 Scale Estimation .................................. 50
  6.3.2 Integrating Sensor Data ............................ 51
6.4 State Estimation and Prediction ....................... 51
  6.4.1 State Vector ....................................... 51
  6.4.2 Observation Model ................................ 52
  6.4.3 State Transition Model ............................ 54
  6.4.4 Estimation of Model Coefficients ................. 56
6.5 PID Controller ......................................... 56

7 Results 57

8 Conclusion 63

  8.1 Limitations ............................................ 64
  8.2 Future Work .......................................... 64

A Least Squares Minimization 68

  A.1 Solution of Linear Equations ....................... 68
  A.2 Solution of Nonlinear Equations ................... 69
# List of Figures

1.1 The de Bothezat quadrotor ................................................. 2

2.1 The Inertial Reference Frame $\mathcal{K}_i$ ................................ 7
2.2 The MAV Frame $\mathcal{K}_m$ .................................................. 8
2.3 The MAV-1 Frame $\mathcal{K}_{m1}$ .............................................. 9
2.4 The MAV-2 Frame $\mathcal{K}_{m2}$ .............................................. 10
2.5 The Body Frame $\mathcal{K}_b$ .................................................. 11
2.6 Side view of the quadrotor MAV .......................................... 12
2.7 The four basic movements of a quadrotor ............................... 14

3.1 Pin-hole imaging model ............................................... 18
3.2 Projection geometry with Camera and World coordinate frames .. 19
3.3 Pose Estimation .......................................................... 23
3.4 Non-Linear Pose Estimation ............................................. 25
3.5 Two-View Geometry ...................................................... 26
3.6 Schematic outline of a keyframe-based SLAM algorithm ........... 35

4.1 Example of filtering ..................................................... 40
4.2 Prediction and update step for a one-dimensional Kalman filter .. 41

5.1 Schematic representation of a general control loop ................. 45
5.2 Proportional and Proportional-Derivative controllers .......... 46
5.3 The PID Controller ..................................................... 47

6.1 Software Architecture .................................................. 49

7.1 The developed system in action ...................................... 57
7.2 Trajectory Comparison: Autonomous Control, W-figure .......... 58
Chapter 1

Introduction

The term “autonomous robot”, even for a reader who possesses knowledge in the field of robotics, has a degree of ambiguity associated with it. Robotic arms in assembly lines, vacuum cleaning robots and self-driving cars for example are all classified as autonomous robots even though they vary vastly in application, scope and complexity. The ambiguity therefore arises from the lack of a universally agreed-upon definition of the term “autonomous robot”. It can be argued that a truly autonomous robot should possess human-like capabilities of thought, self-awareness, reasoning and emotion. In fact, meeting or even exceeding human capabilities has been the long standing goal in the fields of robotics and artificial intelligence. The current state of robotics in comparison can best be described as being in its infancy. Although modern computers and robots out-perform humans in certain tasks, they are thoroughly incapable of free thought. Therefore, a more flexible requirement for a robot to be autonomous is the ability to execute tasks without human intervention. Furthermore, the extent of autonomy is limited to environment and tasks a robot is designed for e.g. a robotic arm programmed to assemble parts of a car can only perform the specialized task of assembling parts of a car within the strict confines of an assembly line.

MAVs (Micro Aerial Vehicles) are a class of MAVs (Unmanned Aerial Vehicles) that are characterized by size and weight restrictions. MAVs can either be controlled remotely by a human or by an autonomous software agent. Quadrotor MAVs in particular have gained significant popularity in recent years. Also known as quadcopters, quadrotor MAVs use a configuration of four fixed pitch rotors to achieve lift and thrust. Their abilities of being able to take off and land vertically, staying still in the air, moving to another position without changing its heading, and maneuvering through tight, confined spaces make them an ideal choice for use in both indoor and outdoor environments.
MAVs have successfully been used in numerous real-world tasks such as search and rescue of a missing person using a quadrotor MAV equipped with a thermal imaging infrared camera\cite{6}, inspection of the disaster site at Japan’s Fukushima Daiichi nuclear power plant\cite{58} or aerial surveillance of potentially threatening situations. Other applications include aerial coverage of sports events, inspecting pipelines and power lines for damage and delivering purchased items to the customers within minutes\cite{15}.

![The de Bothezat quadrotor](Source: www.verticalmag.com)

The earliest reported use of fixed four rotor configuration in a flying aircraft can be traced back to 1922, with the introduction of the ill-fated de Bothezat helicopter \cite{4} which over a course of 100 test flights, reached a maximum altitude of 5 meters. Developed by the US Air Service, the de Boethezat project was promptly scrapped after the test flights proved the aircraft to be unreliable, underpowered and mechanically complex. The mechanical complexity of an otherwise simple arrangement can be attributed to the inherent instability of the quadrotor setup. Without the use of advanced electronic control systems or sophisticated stabilization mechanisms, manual control of a quadrotor aircraft turned out to be too tedious and complex if not impossible. In addition, compared to their two-rotor counterparts, quadrotor aircrafts are less energy efficient.

While the quadrotor configuration may not be feasible in large aircrafts where it has been dominated by the two-rotor configuration, it has nonetheless found considerable success in the MAV domain. Unlike a two-rotor helicopter, all the rotors in a quadrotor configuration have a fixed pitch angle and therefore do not have to rely on a complex swashplate mechanism to alter the MAV’s
orientation. In addition, it affords the possibility of enclosing the rotors inside a lightweight protective hull, enabling the MAV to fly safely in confined indoor environments with a large number of obstacles. Finally, using four smaller rotors for propulsion instead of a single, large one effectively reduces the amount of energy carried by each rotor, minimizing damage in the event of a collision, making quadrotor MAVs particularly suitable for use in human dense environments.

Keeping true to the definition of a robot, modern quadrotor MAVs can perceive the environment using sensors and manipulate it with their rotors. The choice of sensors for use on MAV platforms is dictated by their size, weight and power consumption attributes. The position and orientation of a rigid airborne robot can be described as six independent parameters, each corresponding to one degree of freedom (i.e. position and orientation in all three axes), and can be estimated in real-time using specialized sensors.

The obvious choice for measuring the position of the MAV is a GPS (Global Positioning System) device but position data from a GPS is not only way beyond the acceptable levels of inaccuracy, it is also not updated frequently enough to be useful in MAV applications where a high refresh rate is necessary for the quadrotor to constantly make minor corrections to its trajectory. Furthermore, the unavailability of GPS signals in indoor environments would restrict a system solely relying on GPS for position estimation to outdoor environments.

Alternatively, each of the three position perimeters can be estimated using range sensing devices. For example a remote, range sensing device such as a laser range sensor can measure the distance to the nearest object in one particular direction by analyzing the time difference between an emitted laser beam and its reflection. LiDAR (Light Detection And Ranging) systems extend the one-dimensional functionality of the laser range sensor to two or three dimensions by scanning sections of space in vertical or horizontal directions for two-dimensional maps or both vertical and horizontal directions for three-dimensional maps. Although the maps produced by a LiDAR system are highly detailed, dense and accurate, the system itself is too heavy and power consuming for use in a payload and power constrained quadrotor MAV platform.

The accuracy of a laser range scanner can be compromised for low weight and power consumption of ultrasonic SONAR (SOund Navigation And Ranging) range sensors. Sonar range sensing devices work in a manner similar to laser range sensing devices. The distance to the nearest object in a particular direction is measured by analyzing the time difference between an emitted ultrasonic sound pulse and its received reflection. The loss in accuracy compared to laser based counterparts can be attributed to higher absorption and scattering properties of sound waves resulting in distorted reflections. Sonar sensors are however small and lightweight with low power requirements,
making them ideal candidates for use in MAVs.

Pressure sensors, also known as altimeters are another example of lightweight, power efficient sensors which can be used to estimate one degree of freedom i.e. the altitude of the MAV with respect to the ground. Pressure sensors achieve this by constantly comparing the difference in pressure at the MAV’s current altitude with that at the ground level. Data from a pressure sensor is sensitive to changes in ambient temperature and pressure. In indoor environments, pressure changes from closing and shutting doors for example, can cause undesirable spikes in altitude measurements. Furthermore, low data resolution and high measurement uncertainties make pressure sensors an unreliable source of altitude information in confined indoor environments.

Unlike position estimation, accurate estimates of all three orientation angles can be obtained rather reliably using solid-state MEMS (Micro-Electrical-Mechanical Systems) based IMU (Inertial Measurement Unit) devices. IMUs usually come equipped with a combination of rate gyroscopes, accelerometers, and magnetometers. Rate gyroscopes measure the rate of change of orientation (angular velocity), accelerometers measure the rate of change of velocity (acceleration) and magnetometers measure the strength of magnetic field. Although, data from an IMU can theoretically be used to compute complete 6-DOF (Degrees Of Freedom) pose estimates, it is only used to estimate orientation angles.

Of all the senses that humans and animals rely on to perceive the environment, vision is arguably the most dominant. Therefore, the idea of a robot using vision to infer its position and orientation in three-dimensional space from what it can see is a natural extension of our desire for robots to mimic human behavior as closely as possible. Cameras provide vast amounts of information about the scene under view. In fact, one of the challenges for any real-time computer vision application is to process the sheer quantity of data a camera provides, within strict time limits. While rich in visual data, images from a camera lack depth information. Therefore, any distance and orientation measurements from a single image requires the presence of reference objects with known geometry in the field of view. Additionally, these reference objects need to be sufficiently unique in appearance and textured enough to be distinguishable from the rest of the scene.

The aim of this thesis is to therefore use vision to estimate the MAV’s position and orientation. The pose information from the camera will be fused with noisy IMU measurements to obtain more accurate state estimates. Once the MAV’s position is estimated, it can be used to approach and hold a given target position or follow a fixed path. Furthermore, such a system can be used to significantly reduce a pilot’s workload, making manual control of the MAV much easier by automatically compensating for the inherent instability of the aircraft and in particular horizontal drift.
In order to cope with previously unseen environments and allow for truly autonomous behavior however, knowing the MAV’s position is not sufficient - one also needs ways to detect obstacles, walls, and maybe objects of interest. While the map built for SLAM can partly be used to infer information regarding the surrounding environment such as the position of obstacles, in general additional methods are required.

1.0.1 Problem Statement

The purpose of this thesis is to develop a system that will enable a quadrotor MAV to navigate in environments of which it possesses no a priori information. Using only the information available from the on-board sensors, with the front camera serving as the primary sensor, the system will not rely on external help such as visual markers or a motion capture system. The system will combine unreliable data from multiple, asynchronous sources to obtain reliable estimates of MAV’s position and orientation in three-dimensional space, enabling it to

- hold its position in the three-dimensional space in the presence of external disturbances, wind,
- autonomously fly to a specified position, given its three-dimensional coordinates without human intervention,
- follow a specified path represented as a set of coordinates in three-dimensional space

1.0.2 Outline

In Chapter 2, we will examine quadrotor dynamics and the various coordinate frames associated with it. The following three chapters explain and detail techniques and methods used in our approach: In Chapter 3, techniques and mathematical methods used in keyframe-based visual SLAM algorithms, in particular the components of the parallel tracking and mapping algorithm by Georg Klein and David Murray [24] are presented. In Chapter 4, we present the sensor fusion and prediction method used in our system, the well-known and widely used extended Kalman filter (EKF). In Chapter 5, we introduce the proportional-integral-differential (PID) controller, a method widely used in industrial applications and in our approach for controlling the MAV. Chapters 1 through 4 were written to be independent, free-standing sections which can be read separately from the rest, highlighting the 3 distinct blocks (Chapters 3, 4 and 5) of our system as well as a brief introduction to the quadrotor dynamics (Chapter 2). The distinct blocks are then put together in Chapter 6, where we present our approach, the developed system and its three main blocks: a keyframe-based visual
SLAM algorithm, an EKF to combine data from the SLAM algorithm and on-board sensors, and the PID controller controlling the MAV. In Chapter 7 we analyze the accuracy of the developed system in real-world test. Finally, in Chapter 8 we summarize the developed system and propose ways to improve it and exploring directions for future research.

1.0.3 Contributions

The contribution of this thesis is a portable, platform-agnostic, autonomous visual-inertial control system for quadrotor MAVs that are equipped with a camera and an IMU. The developed system enables an MAV to not only localize itself in the three-dimensional space without using GPS or any external sensors but also to build a representation of its environment in the form of a sparse point map. As a result the system enables the MAV to fly pre-defined trajectories and to hold its position in the presence of external disturbances.

The system in this thesis is built upon and closely related to the past research work done at University of Regina’s MAV lab [56][52][43][19][18][34][2].
Chapter 2

Quadropter Dynamics

2.1 Quadropter MAV Coordinate Frames

Consider a quadropter MAV in flight. In order to specify the motion of an object, we need to describe the motion of every single point on the object. But since a quadropter MAV can be generalized as a rigid object, we only need to specify the motion of a single point on the MAV, and the motion of the coordinate frame attached to that point.

2.1.1 Inertial Frame $K_i$

The inertial frame $K_i$ is an earth-fixed coordinate frame of reference which, for the remainder of this thesis, will have its origin and axes aligned with axes of the MAV at the initial pre-takeoff position (See Figure 2.1).

Flight trajectories, way-points, ground speed and absolute position data of the MAV is measured in the inertial reference frame. Furthermore, map points and the position of external reference
objects is also specified in the inertial reference frame.

2.1.2 MAV Frame $\mathcal{K}_m$

The origin of the MAV frame $\mathcal{K}_m$ is translated by a three-dimensional vector $t_m^i \in \mathbb{R}^3$ and lies at the MAV’s center of mass. The axes of the MAV frame however, are aligned with the axes of the inertial reference frame, which can be observed in Figure 2.2.

![Figure 2.2: The MAV Frame $\mathcal{K}_m$](image)

The yaw angle $\psi \in [-\pi, \pi]$ of the MAV is measured with respect to the MAV frame. A vector $v_{(i)} \in \mathbb{R}^3$ in the inertial frame is transformed to vector $v_{(m)} \in \mathbb{R}^3$ in the MAV frame by a three-dimensional translation vector $t_m^i \in \mathbb{R}^3$ which represents the absolute position of the MAV:

$$v_{(m)} = v_{(i)} + t_m^i \quad (2.1)$$

2.1.3 MAV-1 Frame $\mathcal{K}_{m1}$

The MAV-1 frame $\mathcal{K}_{m1}$ is the rotation of the MAV frame about the $z$-axis by an angle $\psi \in [-\pi, \pi]$. The angle $\psi$ represents the yaw angle of the MAV. Figure 2.3 illustrates the relation between $\mathcal{K}_m$ and $\mathcal{K}_{m1}$. Note that the origin and the axis of rotation are preserved, therefore the origin and the $z$-axis of the MAV-1 frame are identical to that of the MAV frame.

The pitch angle $\theta \in [-\pi, \pi]$ of the MAV is measured with respect to the MAV-1 frame. The transformation between the MAV frame and the MAV-1 frame is a three-dimensional rotation matrix.
\( R_{m}^{m1} \in SO(3) \) that transforms a vector \( v_{(m)} \in \mathbb{R}^3 \) in the MAV frame to vector \( v_{(m1)} \in \mathbb{R}^3 \) in the MAV-1 frame:

\[
v_{(m1)} = R_{m}^{m1} v_{(m)}
\]  

(2.2)

\[
R_{m}^{m1} = \begin{pmatrix}
\cos(\psi) & -\sin(\psi) & 0 \\
\sin(\psi) & \cos(\psi) & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]  

(2.3)

### 2.1.4 MAV-2 Frame \( K_{m2} \)

The MAV-2 frame \( K_{m2} \) is the rotation of the MAV-1 frame about the \( y \)-axis by an angle \( \theta \in [-\pi, \pi] \). The angle \( \theta \) is the pitch angle of the MAV. The origin and the \( y \)-axis of the MAV-2 are identical to that of the MAV-1 frame, which can be seen in Figure 2.4.

The roll angle \( \phi \in [-\pi, \pi] \) of the MAV is measured with respect to the MAV-2 frame. The transformation between the MAV-1 frame and the MAV-2 frame is a three-dimensional rotation matrix \( R_{m1}^{m2} \in SO(3) \) that transforms a vector \( v_{(m1)} \in \mathbb{R}^3 \) in the MAV-1 frame to vector \( v_{(m2)} \in \mathbb{R}^3 \) in the MAV-2 frame:

\[
v_{(m2)} = R_{m1}^{m2} v_{(m1)}
\]  

(2.4)
Likewise, the body frame $\mathcal{K}_b$ is the rotation of the MAV-2 frame $\mathcal{K}_{m2}$ about the $x$-axis by an angle $\phi \in [-\pi, \pi]$, (roll). Similarly, the origin and the $x$-axis of the body frame are identical to that of the MAV-2 frame (See Figure 2.5).

Forces and torques acting on the MAV are specified in the MAV frame. Additionally, on-board sensors like accelerometers and gyroscopes measure data with respect to the MAV frame.

The transformation between the MAV-2 frame and the body frame is a three-dimensional rotation matrix $R^b_{m2} \in SO(3)$ that transforms a vector $v_{\{m2\}} \in \mathbb{R}^3$ in the MAV-2 frame to vector $v_{\{b\}} \in \mathbb{R}^3$ in the MAV-2 frame:

$$v_{\{b\}} = R^b_{m2} v_{\{m2\}}$$

$$R^b_{m2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix}$$
Figure 2.5: The Body Frame $K_b$

The transformation from the MAV frame $K_m$ to the body frame $K_b$ is the three-dimensional rotation matrix $R_m^b \in SO(3)$:

$$R_m^b = R_m^b R_m^2 R_m^1$$

$$R_m^b = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\phi) & \sin(\phi) \\
0 & -\sin(\phi) & \cos(\phi)
\end{bmatrix}
\begin{bmatrix}
\cos(\theta) & 0 & -\sin(\theta) \\
0 & 1 & 0 \\
\sin(\theta) & 0 & \cos(\theta)
\end{bmatrix}
\begin{bmatrix}
\cos(\psi) & \sin(\psi) & 0 \\
-\sin(\psi) & \cos(\psi) & 0 \\
0 & 0 & 1
\end{bmatrix}$$

(2.8)

$$R_m^b = \begin{bmatrix}
c(\psi)c(\theta) & -s(\psi)c(\theta) & s(\theta) \\
c(\psi)s(\theta)s(\phi) + s(\psi)c(\theta) & -s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & -c(\theta)s(\phi) \\
-c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) & s(\psi)s(\theta)c(\phi) + c(\psi)s(\phi) & c(\theta)c(\phi)
\end{bmatrix}$$

(2.8)

And the transformation $R_m^m \in SO(3)$ from the body frame $K_b$ to the MAV frame $K_m$ is the inverse of $R_m^b$, i.e:

$$R_m^m = (R_m^b)^{-1} = (R_m^b)^T$$

$$R_m^m = \begin{bmatrix}
c(\psi)c(\theta) & c(\psi)s(\theta)s(\phi) + s(\psi)c(\theta) & -c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) \\
-s(\psi)c(\theta) & -s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & s(\psi)s(\theta)c(\phi) + c(\psi)s(\phi) \\
s(\theta) & -c(\theta)s(\phi) & c(\theta)c(\phi)
\end{bmatrix}$$

(2.9)

where $c(\cdot) = \cos(\cdot), s(\cdot) = \sin(\cdot)$

It should be noted that since matrix multiplication is generally not commutative, changing the
order of multiplication will in essence change the sequence of rotations and therefore result in a different rotation matrix. The elemental rotation angles \((\psi, \theta, \phi)\) about the three axes are known as Euler angles.

### 2.2 Forces and Moments

Unlike fixed-wing aircraft, quadrotor MAVs do not rely on aerodynamic aerofoils to generate lift or thrust, which is instead provided by the four fixed-pitch rotors, aerodynamic effects therefore, other than drag, will be ignored.

![Figure 2.6: Side view of the quadrotor MAV. Each rotor produces an upwards force \(f_r\) and torque \(\tau_r\). Rotors B and D rotate clockwise and rotors A and C rotate counter-clockwise.](image)

The total thrust \(f_t \in \mathbb{R}^3\) measured in the body frame \(K_b\) is the sum of the upwards force \(f_r \in \mathbb{R}\) generated by the four rotors. Since the rotors do not produce any lateral force, the \(x\) and \(y\) components of \(f_t\) are zero.

\[
\begin{bmatrix}
0 \\
0 \\
f_{tz}
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
f_a + f_b + f_c + f_d
\end{bmatrix}
\]

(2.10)

For the MAV to hover at a fixed elevation, the total upwards thrust should be equal to the force of gravity acting on the MAV i.e. \(f_{tz} = mg\) and the force generated by each rotor \(f_h = \frac{1}{4}mg\).

In order for the MAV to take off or to ascend once it is in the air, the combined force from the four rotors must be greater than the force of gravity i.e. \(f_{tz} > mg\) and the force exerted by each rotor...
should be greater than $f_h$ i.e. $f_{asc} = f_h + \Delta f_*$ (See Figure 2.7(a)). Similarly, to descend $f_{tz} < mg$ and $f_{des} = f_h - \Delta f_*$.

A yawing torque about the $z$-axis acts on the system when one pair of diagonal rotors spins faster than the other due to drag. For example, an effective positive yawing torque is produced if rotors B and D spin faster than rotors A and C. This scenario is illustrated in Figure 2.7(b). Similarly, rotors A and C spinning faster than rotors B and D produces a negative net yawing torque. Expressed mathematically, the total yawing torque $\tau_\psi$ on the system can be written as

$$\tau_\psi = (\tau_b + \tau_d) - (\tau_a + \tau_c) \quad (2.11)$$

$$\tau_\psi \begin{cases} = 0, & (\tau_b + \tau_d) = (\tau_a + \tau_c) \\ > 0, & (\tau_b + \tau_d) > (\tau_a + \tau_c) \\ < 0, & (\tau_b + \tau_d) < (\tau_a + \tau_c) \end{cases} \quad (2.12)$$

A positive rolling torque about the $x$-axis is produced when rotors A and B spin faster than rotors C and D (See Figure 2.7(c)). Conversely, a negative rolling torque is produced when rotors C and D spin faster than rotors A and B. The net rolling torque $\tau_\theta$ is given as:

$$\tau_\theta \propto (f_a + f_b) - (f_c + f_d) \quad (2.13)$$

$$\tau_\theta \begin{cases} = 0, & (f_a + f_b) = (f_c + f_d) \\ > 0, & (f_a + f_b) > (f_c + f_d) \\ < 0, & (f_a + f_b) < (f_c + f_d) \end{cases} \quad (2.14)$$

Continuing in a similar manner, it can be seen in Figure 2.7(d) that a positive pitching torque about the $y$-axis is produced when rotors A and D spin faster than rotors B and C and a negative pitching torque is produced when rotors B and C spin faster than rotors A and D. The overall pitching torque about the $y$-axis is then:

$$\tau_\varphi \propto (f_a + f_d) - (f_b + f_c) \quad (2.15)$$
2.3 Quadrotor State Model

The state variable of a quadrotor MAV can be defined as the following 12-element vector:

\[
\mathbf{v}_t = \begin{pmatrix} x_t, & y_t, & z_t, & \phi_t, & \theta_t, & \psi_t, & \dot{x}_t, & \dot{y}_t, & \dot{z}_t, & \dot{\phi}_t, & \dot{\theta}_t, & \dot{\psi}_t \end{pmatrix}^T \in \mathbb{R}^{12}
\]  

(2.17)

- \( x_t, y_t \) and \( z_t \) are inertial frame quantities that correspond to the position of the MAV’s center in metric units.
- \( \phi_t, \theta_t \) and \( \psi_t \) are the three orientation angles (roll, pitch, yaw) measured w.r.t to MAV-2, MAV-1 and MAV frames respectively.
- \( \dot{x}_t, \dot{y}_t \) and \( \dot{z}_t \) measured in the Body frame, correspond to the velocity of the MAV.
- \( \dot{\phi}_t, \dot{\theta}_t \) and \( \dot{\psi}_t \) are the three rotation rates also measured in the Body frame.

Figure 2.7: The four basic movements of a quadrotor (a)Ascend/Descend (b)Yaw (c)Roll (d)Pitch

\[
\begin{align*}
\tau_\phi & = \begin{cases} 
0, & (f_a + f_d) = (f_b + f_c) \\
> 0, & (f_a + f_d) > (f_b + f_c) \\
< 0, & (f_a + f_d) < (f_b + f_c)
\end{cases} \\
\end{align*}
\]  

(2.16)
Chapter 3

Multiple-View Geometry

The problem of estimating three-dimensional structure of a scene (i.e. three-dimensional coordinates of the scene points in the camera coordinate frame) from its two-dimensional image(s) has been one the long-standing areas of research in the field of computer vision. The problem can be generalized as the inverse of the image formation process: where the latter deals with the study of objects giving rise to images and the former seeks to infer the structure of a scene or an object under view from images. Owing to its inverse nature, the problem is inherently ill-posed or sometimes even impossible to solve. Therefore, we must first seek to develop a suitable mathematical representation of the image formation process that strikes a balance between complexity and ease of manipulation (the model should be easily invertible). This trade-off between accuracy and simplicity can be achieved by imposing constraints on the image formation process, scene geometry and lighting conditions.

It will become clear later in this chapter, that one of the three dimensions of the scene structure (depth data) is lost during the image formation process, rendering depth estimation through the means of geometry from single images impossible. Although texture, local intensity variations and shading cues in a single image can be directly analyzed to infer three-dimensional structure, these methods are often too unreliable and require extensive knowledge of the scene under view - a requirement that prohibits their use in arbitrary, unknown environments. Alternatively, supervised machine learning methods \[20\] \[45\] can be employed to extract the three-dimensional structure of a scene from single images by learning from a set of pre-annotated training samples: a large, diverse set of images along with their corresponding depth data. While learning based methods exhibit reasonable accuracy on benchmark data, the depth estimates on their own are still too coarse to be
used in any real-time application. However their use as prior estimates in a probabilistic framework, or as initial estimates that can be further refined in an optimization framework appears to have merit.

Perhaps then the human visual system is a good place to gain insight on the process of depth perception from images. Although machines are still not capable of emulating the human visual system, owing in equal parts to its complexity, our lack of understanding and the contrasting manner in which human brains and microprocessors handle data, our knowledge of how it combines two different images, one from the left eye and the other from the right eye to infer three-dimensional structure of the scene under view can nonetheless be expressed in a more familiar form that both humans and machines can work on - mathematics.

Human visual system is an example of a two-view imaging system, also known as stereoscopic or binocular vision. The two-view paradigm can be extended to multiple-views where multiple two-dimensional images of a scene, taken from different vantage points are used to estimate its three-dimensional structure. What should be of note here is the use of the term multiple-views, and the deliberate avoidance of the term multiple cameras. The images of the scene need not be taken at the same time from different cameras, an image sequence or a video of a static scene from a single camera, moving in three-dimensional space is another way of describing a multiple-view system. Where snapshots and camera locations at different time intervals respectively serve as the multiple images and vantage points. The use of a moving camera to recover the three-dimensional geometry of a scene under view is known as structure from motion in computer vision, a concept that will be use pervasively in this thesis.

3.1 Notation

- Vectors will be denoted as lower-case, bold letters and matrices as upper-case, regular emphasis letters.

- Images will be denoted as a function \( I(x, y) \in [0, 255] \subseteq \mathbb{Z}_+ \) defined over a discretized two-dimensional image space of positive integers \( \Omega \subseteq \mathbb{Z}_+^2 \) such that indices \( x \) and \( y \) are mapped to intensity values in the interval of positive integers \([0, 255] \subseteq \mathbb{Z}_+\).

- Coordinate frames will be denoted as uppercase calligraphic letters. For instance \( \mathcal{W} \) represents the world coordinate frame and \( \mathcal{C}_n \) represents the \( n \)-th camera coordinate frame.

- Transformations between coordinate frames will be denoted as \( 4 \times 4 \) matrices.
3.2 Camera Model

A camera can be broadly generalized as a composition of two basic components: An optical lens to "direct" the propagation of light (through reflection, refraction and diffraction) and a photosensitive surface on which light directed from the lens forms an image. The photosensitive surface in modern digital cameras, is a two-dimensional array of discrete CCD (charge-coupled device) or CMOS (complementary metal-oxide semiconductor) photo sensor elements, with each sensor element representing a pixel. Intensity values at each pixel are calculated by integrating the total light energy received by the sensor element over both time (exposure time) and space (dimensions of the CCD/CMOS sensor). The image formation process of a camera (camera model) can therefore be constructed by modelling the two components.

In literature, camera models of varying complexity have been proposed, ranging from simplified non-linear \[12] to higher-order non-linear models for wide-angle lenses \[23][9]. Like most mathematical models of a real-world physical system, the complexity of a camera model is a direct trade-off between accuracy and ease of manipulation.

3.2.1 Pin-hole Model

One of the simplest ways to model an optical lens is by approximating its aperture by an infinitesimally small hole, such that all the light rays propagating through the lens are forced to pass through a single point: the optical center.

Consider the ideal pin-hole imaging model shown in Figure 3.1. If we let scene point \( p \) have coordinates \( X_{(c)} = (X_{(c)}, Y_{(c)}, Z_{(c)})^T \in \mathbb{R}^3 \) defined in the camera coordinate frame \( C \), centered at the optical center \( O \) and its projection \( x \) have coordinates \( x_{(c)} = (x_{(c)}, y_{(c)})^T \in \mathbb{R}^2 \) on the two-dimensional image plane, then it can be seen that the coordinates of \( p \) and \( x \) are related by an ideal perspective projection:

\[
x_{(c)} = -f \frac{X_{(c)}}{Z_{(c)}}, \quad y_{(c)} = -f \frac{Y_{(c)}}{Z_{(c)}}
\]

(3.1)

where \( f \): the distance in metric units along the \( z \)-axis between the image plane and the optical center \( O \) is the focal length of the imaging system.

The negative sign in Eq 3.1 represents the fact that the image of an object is formed upside down on the image plane. The negative sign can be dropped by flipping the image: \((x_{(c)}, y_{(c)}) \mapsto (-x_{(c)}, -y_{(c)}) \) or \(-f \mapsto f\), effectively bringing the image plane in front of the optical center.

It is important to note that since the amount of light that can pass through an infinitesimally
small aperture is also infinitesimally small, the pin-hole model serves as an idealized geometric approximation of well-focused imaging systems instead of representing the actual aperture size of the lens.

It is evident from the mathematical expression of perspective projection (Eq 3.1) that the inverse problem of determining the three-dimensional coordinates \((X_{(c)}, Y_{(c)}, Z_{(c)})^T\) of the scene point \(p\) from its two-dimensional image coordinates \((x_{(c)}, y_{(c)})^T\) and the camera’s focal length \(f\) is not possible without explicit knowledge of depth \(Z_{(c)}\). This depth ambiguity can also be observed directly from Figure 3.1 where any point on the line passing between \(p\) and \(O\) is projected onto the same point \(x\).

### 3.2.2 Extrinsic Parameters

In the perspective relation of Eq 3.1, the three-dimensional coordinates \((X_{(c)}, Y_{(c)}, Z_{(c)})^T\) of the scene point \(p\) were expressed with respect to the camera coordinate system \(C\). Since the position of the camera and the coordinate frame \(C\) associated with it is not static, the geometry of the scene can instead be expressed with respect to an earth-fixed inertial coordinate frame, which will be referred to as the world coordinate system \(\mathcal{W}\). With the introduction of an additional coordinate frame the perspective geometry of Eq 3.1 needs to be modified to accommodate the transformation between the two coordinate frames.

Figure 3.2 shows scene point \(p\) with respect to both the camera \(C\) and the world coordinate system
If we let the coordinates of point \( p \) be defined in \( \mathcal{C} \) as \( X_{(c)} = (X_{(c)}, Y_{(c)}, Z_{(c)}) \in \mathbb{R}^3 \) and in \( \mathcal{C} \) as \( X_{(w)} = (X_{(w)}, Y_{(w)}, Z_{(w)}) \in \mathbb{R}^3 \), then \( X_{(c)} \) and \( X_{(w)} \) are related by a three-dimensional rigid transformation \( M_{c}^w \in SE(3) \) such that

\[
X_{(c)} = M_{c}^w X_{(w)} = R_{c}^w X_{(w)} + t_{c}^w
\]  \hspace{1cm} (3.2)

where \( R_{c}^w \in SO(3) \) is a \( 3 \times 3 \) rotation matrix and \( t_{c}^w \in \mathbb{R}^3 \) is a three-dimensional translation vector. A more convenient way of writing Eq (3.2) is to use the homogeneous representation of points, so that:

\[
\bar{X} = \begin{pmatrix} X \\ 1 \end{pmatrix} \in \mathbb{R}^4, \quad \bar{M}_{c}^w = \begin{pmatrix} R_{c}^w & t_{c}^w \\ 0 & 1 \end{pmatrix} \in \mathbb{R}^{4 \times 4} \]  \hspace{1cm} (3.3)

Using homogeneous representation, Eq (3.2) can now be written as:

\[
\bar{X}_{(c)} = \bar{M}_{c}^w \bar{X}_{(w)} = \begin{pmatrix} R_{c}^w & t_{c}^w \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_{(w)} \\ 1 \end{pmatrix}
\]  \hspace{1cm} (3.4)

Similarly, combining Eq (3.4) and Eq (3.1) and re-writing using homogeneous representation, we get a relationship between the three-dimensional coordinates \( \bar{X}_{(w)} = (X_{(w)}, Y_{(w)}, Z_{(w)}, 1) \) of scene point \( p \) in the world coordinate system \( \mathcal{W} \) and its two-dimensional projection \( \bar{x}_{(c)} = (x_{(c)}, y_{(c)}, 1)^T \).
3.2.3 Intrinsic Parameters

The camera model of Eq. 3.5 needs to be further modified in order to express projection points \( \tilde{x}_{(c)} \) in pixel units instead of metric units. This can be achieved by introducing a scaling matrix \( S \in \mathbb{R}^{3 \times 3} \) such that:

\[
\begin{bmatrix}
  x_{(c)} \\
  y_{(c)} \\
  1
\end{bmatrix}
= \begin{pmatrix}
  s_x & 0 & 0 \\
  0 & s_y & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{bmatrix}
  x_{(c)} \\
  y_{(c)} \\
  1
\end{bmatrix}
\]

(3.7)

where \( s_x \in \mathbb{R}_+ \) and \( s_y \in \mathbb{R}_+ \) represent the size of a pixel along \( x \) and \( y \) axes in metric units (pixels/m). The coordinates \( \tilde{x}_{(c)} \) are still specified with respect to the principal point: the point where the \( z \)-axis of the camera coordinate system \( C \) intersects the image plane. Pixel coordinates however, are positive integers defined in relation to the top-left corner of the image. The pixel coordinates \( \tilde{x}_{(c)} \) therefore need to be translated appropriately:

\[
\tilde{x}_{(c)}_{im} = \tilde{x}_{(c)} + c_x
\]

(3.8)

\[
\tilde{y}_{(c)}_{im} = \tilde{y}_{(c)} + c_y
\]

where \( c_x \) and \( c_y \) are the pixel coordinates of the principal point with respect to the origin of the image plane. The scaling and translation operations of Eq. 3.7 and Eq. 3.8 can be combined into a single matrix \( K_s \in \mathbb{R}^{3 \times 3} \), such that:

\[
\begin{bmatrix}
  x_{(c)}_{im} \\
  y_{(c)}_{im} \\
  1
\end{bmatrix}
= \begin{pmatrix}
  s_x & 0 & c_x \\
  0 & s_y & c_y \\
  0 & 0 & 1
\end{pmatrix}
\begin{bmatrix}
  x_{(c)} \\
  y_{(c)} \\
  1
\end{bmatrix}
\]

(3.9)
Combining Eq 3.9 with the perspective projection model of Eq 3.5 yields a more complete representation of the transformation between three-dimensional coordinates $\bar{X}_{(w)}$ of scene point $p$ in the world coordinate system $W$ and its corresponding two-dimensional pixel coordinates $\bar{x}_{(c)im}$ in the camera coordinate system $C$:

$$Z_{(c)} \begin{pmatrix} x_{(c)im} \\ y_{(c)im} \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & c_x \\ 0 & s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R^c_w & t^c_w \\ 0 & 1 \\ s & x_0 \\ c & x_0 \end{pmatrix} \begin{pmatrix} X_{(w)} \\ Y_{(w)} \\ Z_{(w)} \\ 1 \end{pmatrix}$$

(3.10)

Matrices $K_s$ and $K_f$ can be multiplied together to obtain a single matrix $K$:

$$K = K_s K_f = \begin{pmatrix} s_x & 0 & c_x \\ 0 & s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} f s_x & 0 & c_x \\ 0 & f s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{3 \times 3}$$

(3.11)

Substituting $K_s K_f$ in with $K$ in the pin-hole perspective projection model of Eq 3.5, we get:

$$Z_{(c)} \bar{x}_{(c)im} = KP \bar{X}_{(w)} = KPM^c_w \bar{X}_{(w)}$$

(3.12)

(3.13)

where:

- $Z_{(c)} \in \mathbb{R}_+$ is the depth (z-coordinate) of the scene point $p$ in the camera coordinate system $C$.
- $\bar{x}_{(c)im} \in \mathbb{R}^2$ are the coordinates of the projection point $x$.
- $K \in \mathbb{R}^{3 \times 3}$ is the matrix of intrinsic parameters.
- $f s_x \in \mathbb{Z}_+$ and $f s_y \in \mathbb{Z}_+$ are focal lengths in $x$ and $y$ direction (pixels).
- $c_x \in \mathbb{Z}_+$ and $c_y \in \mathbb{Z}_+$ are the coordinates of the principal point (pixels).
- $P \in \mathbb{R}^{3 \times 4}$ is the projection matrix.
• $M_w^c \in SE(3)$ is the three-dimensional world-to-camera transformation matrix. Also referred to as the pose of the camera with respect to the world coordinate system $\mathcal{W}$.

• $\bar{X}_{\{w\}} \in \mathbb{R}^3$ are the coordinates of the scene point $p$ in the world coordinate system $\mathcal{W}$.

3.3 Single-View Geometry

With a suitable model of image formation at our disposal, we can now proceed to analyze the inverse process of retrieving the three-dimensional scene geometry and camera pose from images. It was observed in the previous section that the inherent depth ambiguity arising from the process of image formation renders geometric estimation of the three-dimensional scene structure from the sole knowledge of two-dimensional image point coordinates impossible. However, provided that the geometry of the scene under view and its corresponding image projections along with the intrinsic camera parameters are already known, we can use the camera model of Eq 3.9 to estimate the pose of the camera $M_w^c \in SE(3)$.

3.3.1 2D-3D Pose Estimation

Pose estimation forms the basis of many augmented reality based applications. The problem statement can be summarized as determining the optimal rotation and translation of a calibrated camera with respect to a known three-dimensional object or scene and its corresponding two-dimensional image coordinates. The geometry of the scene under view can be expressed as a collection of points, lines or planes.

In this thesis we will only consider the point representation case where a scene is defined as a set of $n$ sparse points $p_i$ with three-dimensional homogeneous coordinates $\bar{X}_{\{w\}} i \in \mathbb{R}^3$ in the world coordinate system $\mathcal{W}$. Each scene point $p_i$ has a corresponding image projection $x_i$ with two-dimensional homogeneous coordinates $\bar{x}_{\{c\}}i \in \mathbb{R}^2$ in the camera coordinate system $\mathcal{C}$. Figure 3.3 shows the case for $n = 2$ correspondences.

$$Z_{\{c\}}i\bar{x}_{\{c\}}i = K\bar{X}_{\{c\}}i = KM_w^c\bar{X}_{\{w\}}i$$  \hspace{1cm} (3.14)
Figure 3.3: Pose Estimation

\[ M_w^c = [R_w^c \mid t_w^c] \]

\[
Z_{(c)i} \begin{pmatrix} x_{(c)i} \\ y_{(c)i} \\ 1 \end{pmatrix} = \begin{pmatrix} f s_x & 0 & c_x \\ 0 & f s_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \begin{pmatrix} X_{(w)i} \\ Y_{(w)i} \\ Z_{(w)i} \\ 1 \end{pmatrix} \]

\[ (3.15) \]

\[ \tilde{x}_{(c)i} = K^{-1} x_{(c)i} \]

\[ (3.16) \]

\[ Z_{(c)i} \tilde{x}_{(c)i} = M_w^c \tilde{X}_{(w)i} \]

\[ (3.17) \]

\[
Z_{(c)i} \begin{pmatrix} \tilde{x}_{(c)i} \\ \tilde{y}_{(c)i} \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \begin{pmatrix} X_{(w)i} \\ Y_{(w)i} \\ Z_{(w)i} \\ 1 \end{pmatrix} \]

\[ (3.18) \]
\[ Z_{(c)} = r_{11}X_{(w)i} + r_{12}Y_{(w)i} + r_{13}Z_{(w)i} + t_x \]  
(3.19)

\[ Z_{(c)} = r_{21}X_{(w)i} + r_{22}Y_{(w)i} + r_{23}Z_{(w)i} + t_y \]  
(3.20)

\[ Z_{(c)} = r_{31}X_{(w)i} + r_{32}Y_{(w)i} + r_{33}Z_{(w)i} + t_z \]  
(3.21)

\[ r_{11}X_{(w)i} + r_{12}Y_{(w)i} + r_{13}Z_{(w)i} + t_x - \tilde{x}_{(c)i}(r_{31}X_{(w)i} + r_{32}Y_{(w)i} + r_{33}Z_{(w)i} + t_z) = 0 \]  
(3.22)

\[ r_{21}X_{(w)i} + r_{22}Y_{(w)i} + r_{23}Z_{(w)i} + t_y - \tilde{y}_{(c)i}(r_{31}X_{(w)i} + r_{32}Y_{(w)i} + r_{33}Z_{(w)i} + t_z) = 0 \]  
(3.23)

Equations (3.22) and (3.23) can be written as a homogeneous system of linear equations of the form \( Ax = 0 \). Where \( x \in \mathbb{R}^{12} \) is a vector of the 12 unknown parameters of transformation matrix \( M^C \in SE(3) \) and \( A \) is a \( 2n \times 12 \) matrix of their respective coefficients.

\[ Ax = 0 \]  
(3.24)

\[ A = \begin{pmatrix} \tilde{X}_{(w)1}^T & 0_{1\times4} & -\tilde{x}_{(c)1} & \tilde{X}_{(w)1}^T \\ 0_{1\times4} & \tilde{X}_{(w)1}^T & -\tilde{y}_{(c)1} & \tilde{X}_{(w)1}^T \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{X}_{(w)n}^T & 0_{1\times4} & -\tilde{x}_{(c)n} & \tilde{X}_{(w)n}^T \\ 0_{1\times4} & \tilde{X}_{(w)n}^T & -\tilde{y}_{(c)n} & \tilde{X}_{(w)n}^T \end{pmatrix} \in \mathbb{R}^{2n\times12} \]  
(3.25)

\[ x = (r_{11}, r_{12}, r_{13}, t_x, r_{21}, r_{22}, r_{23}, t_y, r_{31}, r_{32}, r_{33}, t_z)^T \in \mathbb{R}^{12} \]  
(3.26)

The better known closed-form solutions of Quan[12], Hartley[18], Abdel[1], Slama[18], LePetit[27] and Ansari[3] estimate the pose by directly solving the above system of linear equations or a variation of it.

The solution which minimizes the algebraic error function \( \|Ax\| \) lies in the null-space (kernel) of matrix \( A \), and can be obtained in closed-form using either the singular value decomposition of the \( 2n \times 12 \) matrix \( A \) as the right singular vector corresponding to the smallest singular value of \( A \) or using the eigenvalue decomposition of the \( 12 \times 12 \) square matrix \( B = A^TA \) as the eigenvector corresponding to the smallest eigenvalue of \( B = A^TA \). (See Appendix A.1)
3.3.2 Non-Linear Refinement

Closed-form methods, like the one described in the previous section have the undesirable trait of being sensitive to noise. Even a small error in the measured image point coordinates $x_{i,c}$ can lead to pose estimates that are significantly different from the true solution. This can be remedied by iteratively minimizing a nonlinear geometric error function instead, such that:

$$
m_{w}^{c*} = \arg\min_{m_{w}^{c}} \sum_{i=1}^{n} \| \bar{x}_{i} - f_{i}(m_{w}^{c}) \|^2 \quad \forall i = 1, \cdots, n \tag{3.27}
$$

where $m_{w}^{c*} \in \mathbb{R}^6$ is the vector parametrization of the pose that minimizes the reprojection error i.e. the distance between the measured images point coordinates $x_{i,c}$ and the hypothesized image point coordinates $x'_{i,c} = f(m_{w}^{c})$

![Non-Linear Pose Estimation](image)

Figure 3.4: Non-Linear Pose Estimation

Equation 3.27 is a nonlinear least squares minimization problem which can be solved using iterative unconstrained optimization methods (See Appendix A.1).

3.4 Two-View Geometry

Consider the two-camera system shown in Fig 3.5. $x_1$ and $x_2$ are the coordinates of projections of the same point $p$ onto the image planes of the two cameras. If we let $X_1$ and $X_2$ be the three-dimensional coordinates of point $p$ in camera frames $C_1$ and $C_2$, then the transformation between
the two camera frames can be written as:

$$X_2 = RX_1 + t$$

(3.28)

The relationship above can also be expressed in terms of projection coordinates with unknown depth \((z)\) components \(\lambda_i\) as:

$$\lambda_2 x_2 = R\lambda_2 x_1 + t$$

(3.29)

The unknown depths can be eliminated by multiplying both sides with a skew symmetric matrix parametrization \(\hat{T}\) of the translation vector \(t\):

$$\lambda_2 \hat{T} x_2 = \hat{T} R \lambda_1 x_1$$

(3.30)

Since the vector \(\hat{T} x_2 = t \times x_2\) is perpendicular to the vector \(x_2\), the inner product \(<x_2, \hat{T} x_2> = x_2^T \hat{T} x_2\) is also zero. This implies that the quantity \(x_2^T \hat{T} R \lambda_1 x_1\) is also zero. Eq 3.30 is therefore reduced to:

$$x_2^T \hat{T} R x_1 = 0$$

(3.31)
where $E = \hat{T}R \in \mathbb{R}^{3 \times 3}$ is known as the Essential matrix.

The essential matrix plays a central role in two-view geometry and encodes the relation between two images, taken of the same scene but from different viewpoints. Note that in this thesis, we focus only on the essential matrix $E$, describing the relationship between points expressed in homogeneous, normalized image coordinates. However, the same methods can be applied when the two calibration matrices $K_1$ and $K_2$ are unknown, using the fundamental matrix $F$. The essential and the fundamental matrix are related by $E = K_1FK_2$.

### 3.4.1 Estimating the Essential Matrix

Since the essential matrix encodes the rigid-body transformation between two camera frames, it has 6 degrees of freedom. Only 5 of these can however be determined from point correspondences - as mentioned, $E$ can only be determined up to scale, leaving the scale of the translation vector $t$ unknown. As each pair of observations introduces two additional constraints on $E$ and one additional unknown, namely the depth of the point, a minimum of 5 points is required to estimate it. In fact there are many different methods to estimate $E$ from five, six or seven correspondences [29].

A more common approach however is the comparatively simple eight-point algorithm, which requires a minimum of 8 point-correspondences. For each pair $x_1, x_2$, the relationship $x_2^T E x_1 = 0$ leads to one linear constraint on the entries of $E$. Combining $n$ such constraints results in a linear system:

$$A b = 0 \quad (3.33)$$

with $A \in \mathbb{R}^{n \times 9}$ being built by stacking the constraints, while $b \in \mathbb{R}^{9}$ contains the entries of $E$. In the absence of noise and for $n \geq 8$, Eq 3.33 will have a unique solution (up to scale), namely the null-space of matrix $A$. In the presence of noise and with $n \geq 8$ however, Eq 3.33 will only yield the trivial solution $e = 0$. In order to still get an estimate, it is relaxed to

$$\min_b A b, \quad \text{such that } \|b\| = 1 \quad (3.34)$$

This minimization problem can be solved using the singular value decomposition (SVD) of $A$, the solution being the right singular vector of $A$ corresponding to the smallest singular value. Note that this procedure yields a matrix $E \in \mathbb{R}^{3 \times 3}$ which is the least-square solution to equation Eq 3.32.
to the presence of noise however, this matrix in general does not satisfy the properties of an essential matrix as described above and therefore additional normalization steps are required. Furthermore it has been shown that other methods, exploiting the internal constraints of an essential matrix and solving analytically for the five unknown degrees of freedom are both faster and more stable in practice, however nonlinear and therefore significantly more complex \cite{19}.

### 3.4.2 Recovering Camera Pose from Essential Matrix

Once $E$ has been estimated, it is possible to recover $R$ and $t$ from it: Let $E = UDV^T$ be the SVD of $E$. Algebraic resolution leads to four candidate solutions satisfying $E = \hat{T}R \in \mathbb{R}^{3 \times 3}$.

$$
\hat{T} = VWDV^T \tag{3.35}
$$

$$
R = UW^{-1}V^T \tag{3.36}
$$

using:

$$
W = \begin{pmatrix}
0 & -1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{pmatrix} \quad \text{or} \quad W = \begin{pmatrix}
0 & 1 & 0 \\
-1 & 0 & 0 \\
0 & 0 & 1
\end{pmatrix} \tag{3.37}
$$

It can be shown that only one of those four solutions is valid, as for any pair of corresponding observations $x_1, x_2$, three solutions will generate a point $X$ lying behind at least one camera plane. Furthermore, in practice $\hat{T}$ will not be a skew-symmetric matrix due to a noisy estimate of $E$. In order to avoid this, the element $(3,3)$ of matrix $W$ can be set to zero, forcing $\hat{T}$ to be a valid skew-symmetric matrix \cite{18}.

### 3.4.3 Triangulating Landmarks

With $R$ and $t$ known, the 3D-position $X \in \mathbb{R}^3$ of point $p$ can be triangulated. As each observation yields two constraints on $x$, this is an over constrained problem. The maximum-likelihood solution $x^*$ is given by the minimization of the two-sided reprojection error:

$$
X^* = \arg\min_x e_{\text{rep}}(X, R, t) \tag{3.38}
$$

$$
e_{\text{rep}}(X, R, t) = \|\text{proj}(K_1 X) - x_1\| + \|\text{proj}(K_1 R(X - t)) - x_2\| \tag{3.39}
$$

where $x_1$ and $x_2$ are the known pixel-coordinates of the projection of point $p$ onto the two image
planes. An analytic solution from any three constraints can be used as initialization, such a solution however is very ill-conditioned and may lead to very poor estimates.

### 3.4.4 Non-linear Refinement

Although the eight-point algorithm can easily incorporate more than 8 points, it turns out that doing so does not significantly increase the accuracy of the estimate for $R$ and $t$. This is due to the fact that it minimizes the algebraic error function (Eq 3.33), which does not have a direct geometric interpretation: while without noise this error function does have its minimum at the correct position, in the presence of noise the minimum of this error function does not correspond to the optimal solution in a maximum-likelihood sense. This is particularly true if the point-correspondences are in pixel-coordinates, and Eq 3.33 is solved for the fundamental matrix.

In order to get a better estimate, a nonlinear minimization using the result of the method described above as initialization is applied, minimizing the total reprojection error and solving for $R$ and $t$ and $X_1, ..., X_n$ simultaneously:

$$\min_{X_1, ..., X_n, R, t} \sum_{i=1}^{n} e_{rep}(X_i, R, t)$$  \hspace{1cm} (3.40)

### 3.5 Feature Points

Feature points are small two-dimensional patches of texture that are unique within a local neighbourhood in an image. The idea behind extracting feature points is to minimize the amount of data to be processed. Instead of processing the entire image, a minimal set of “unique” features in the image can be used for tasks such as detecting and tracking objects, estimating pose etc. In an image sequence, feature points are chosen as distinguishable image segments which can be easily identified and tracked over subsequent frames. Examples of features points include edge-based features\cite{14}, corner-based features and region-based features \cite{11}\cite{36}. In this thesis, a feature point is a 2D location on the image, while its corresponding 3D location will be referred to as a landmark. Therefore, feature points are image projections of landmarks on the image plane.

#### 3.5.1 Finding feature points

Introduced in 1988 by Harris and Stephens \cite{17}, the Harris corner detector is one of the earliest feature detectors. The Harris corner detector does not strictly identify corners in an image. Instead, it
identifies image patches as potential feature points if they exhibit low self-similarity. Mathematically, this can be expressed as demanding the change in appearance to be large in every direction when the patch is shifted on the unit circle \((x, y)\). Measured as a weighted sum of squared difference (SSD), the change in appearance is denoted by \(S(x, y)\). The weighing function \(w(u, v)\) is Gaussian with variance \(s^2\) i.e:

\[
S(x, y) = \sum_u \sum_v w(u, v) (I(u + x, v + y) - I(u, v))^2
\]  

(3.41)

Using the first-order Taylor series expansion, \((I(u + x, v + y) - I(u, v))\) can be decomposed into terms of image gradients \(I_x\) and \(I_y\) as:

\[
(I(u + x, v + y) - I(u, v)) \approx I(u, v) + I_x(u, v)x + I_y(u, v)y
\]  

(3.42)

\(S(x, y)\) is now simplified to:

\[
S(x, y) \approx \sum_u \sum_v w(u, v) (I_x(u, v)x + I_y(u, v)y)^2
\]  

(3.43)

Which can also be written as:

\[
S(x, y) \approx (u, v) \begin{pmatrix} x \\ y \end{pmatrix}
\]  

(3.44)

Where matrix \(A\) is referred to as the structure tensor:

\[
A = \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}
\]  

(3.45)

The structure tensor \(A\) is a symmetric, positive semi-definite matrix, therefore the two extrema \(\max_{x,y} S(x, y)\) and \(\min_{x,y} S(x, y)\) are the two eigenvalues \(\lambda_1\) and \(\lambda_2\) of \(A\). A patch is a corner only if \(\lambda_1\) and \(\lambda_2\) both have large real values.

Instead of finding the eigenvalues of matrix \(A\), which can be computationally expensive, Harris and Stephens \[17\] suggest finding the response function \(M_c\):

\[
M_c = \lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2
\]  

\[
= \det(A) - \kappa \text{trace}^2(A)
\]  

(3.46)

Where \(\kappa\) is a positive constant. Shi and Tomasi claimed that \(\min(\lambda_1, \lambda_2)\) is a more stable response
function. The improvement over the Harris corner detector referred to as the Shi-Tomasi corner detector [47].

### 3.5.2 FAST Corner Detector

Compared to the Harris corner detector, the FAST corner detector of Rosten and Drummond [44] is around 20 to 30 times faster, a trait that makes it particularly suitable for use in real-time applications. The FAST corner detector explicitly finds corners in an image by determining whether a pixel is at the center of a corner. A pixel is classified as a corner if the pixels surrounding it in a circle contain continuous sequences of brighter or darker pixels.

### 3.5.3 LoG Blob Detector

Instead of finding corners in an image, the Laplacian of Gaussian (LoG) blob detector detects blobs. Blobs are approximately circular contiguous regions which are brighter or darker compared to their surrounding. These blobs correspond to the local extrema of the LoG response function \( M_L(x, y; t) \), which is the Laplacian of the smoothed image:

\[
M_L(x, y; t) = t \Delta L(x, y; t) \tag{3.47}
\]

\[
L(x, y; t) = g(x, y, t) * I(x, y) \tag{3.48}
\]

Where \( g(x, y, t) \) is a Gaussian kernel with variance \( t \):

\[
g(x, y, t) = \frac{1}{2\pi t^2} e^{-\frac{x^2+y^2}{2t^2}} \tag{3.49}
\]

### 3.5.4 Multiscale Feature Point Detection

The FAST and Harris corner detection algorithms can only find corners of a certain size. To be able to find corners at different scales, a multi-scale approach needs to be adapted. The image pyramid method allows larger scaled corners to be detected by building a pyramid of successively down-sampled images, with increasing relative patch size. The Lukas-Kanade tracking algorithm finds Harris corners at each pyramid-level, and removes the points that do not correspond to a local maximum in scale-dimension with respect to their Difference-of-Gaussian DoG response.
3.5.5 Tracking a Feature Point

The task of tracking a feature point involves finding the exact location of a known feature point in an image, under the assumption of good initialization. When processing an image stream, the location of the patch in the previous frame can be used as initialization, assuming that the displacement between two consecutive frames is small. Tracking is not to be confused with methods for matching image descriptors such as SIFT\[30\] and SURF\[4\] (which build an invariant representation of a patch) or Ferns\[41\] (training a classifier to recognize a patch from any viewpoint), although these methods can also be used for tracking, referred to as tracking by detection. A general formulation of tracking is to find parameters $p$ of a warp function $f(x, y; p)$, such that the difference between the original patch $T(x, y)$ and the transformed image $I(f(x, y; p))$ becomes minimal, i.e. the sum of squared differences (SSD) is minimized:

$$p^* = \arg\min_p \sum_x \sum_y (I(f(x, y; p) - T(x, y))^2$$

(3.50)

In order to achieve invariance to affine lightning changes, a normalized cross-correlation error function can be used instead. Note that this error function is highly non-convex and hence only a local minimum can be found, underlining the need for a good initialization. The warp function $f(x, y; p)$ can take different forms, for tracking a two-dimensional image patch, two important warp functions are:

**Pure Translation**: It is often sufficient to consider only translation for frame-to-frame tracking. The resulting transformation has two degrees of freedom, the displacement in two dimensions:

$$f(x, y; \delta x, \delta y) = \begin{pmatrix} x + \delta x \\ y + \delta y \end{pmatrix}$$

(3.51)

**Affine Transformation**: An affine transformation allows for displacement, non-uniform scaling and rotation, leading to 6 degrees of freedom:

$$f(x, y; p) = \begin{pmatrix} p_1 & p_2 \\ p_3 & p_4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} p_5 \\ p_6 \end{pmatrix}$$

(3.52)

For practical applications, choosing a good warp function and parametrization for the task at hand is essential.
3.6 Monocular SLAM

Starting without any prior knowledge of the environment, simultaneous localization and mapping (SLAM) algorithms [46] continuously estimate the pose of a robot while at the same time build a map of its environment. Since, estimating the pose of a robot requires a map of the environment and building a map of the environment requires accurate pose estimates, SLAM is often referred to as a chicken and egg problem. SLAM algorithms form the basis of many autonomous robotic applications where a robot is required to navigate in unknown environments with no a priori knowledge. SLAM algorithms are also a central component of many marker-free augmented reality applications which allow the user to project virtual objects on to a real scene.

The map of the environment generated by a visual SLAM algorithm is represented as a number of 3D landmark points i.e. points in 3D space whose 2D projections (feature points) can be recognized and localized in the camera image, typically appearing as distinguishable regions or patches (See Section 3.5). From the locations of the feature points in the image, the position of the camera can be estimated. As the camera explores the environment and new parts of it become visible, the map is augmented by adding newly identified 3D landmark points and trimmed by removing obsolete landmark points.

In spite of being the subject of significant research over the past few decades, the SLAM problem is still marred with several challenges such as handling large and dynamic environments, detecting loop closures, and efficiently managing the computational complexity with a growing map. A problem that is particular to visual SLAM algorithms is the scale ambiguity of the map. Without the presence of external calibration objects in the scene, such as markers with known geometry, one degree of freedom i.e. the scale remains unknown.

SLAM algorithms have played an important role in the development of truly autonomous robots [39]. Historically, the formulation of the SLAM problem is attributed to Durrant [13]. Early approaches to the solution of the SLAM problem employed an extended Kalman filter. The current pose of the robot as well as the locations of all the landmarks in 3D space are represented as single current state vector. Also known as EKF-SLAM, the extended Kalman filter based SLAM algorithm is severely limited by its computational complexity which is quadratic in the number of landmarks present in the state vector, since each iteration requires an update of the covariance matrix of the state vector. The quadratically increasing complexity of EKF-SLAM therefore sets a practical limit on the number of landmarks. The limitations of the EKF-SLAM algorithm are mitigated by replacing the
extended Kalman filter with a particle filter\cite{39}. The use of a multi-modal probability distribution made possible by employing a particle filter, as opposed to the uni-modal Gaussian distribution of the extended Kalman filter enables each particle (mode) to represent a possible trajectory of the robot. The FAST-SLAM algorithm has a complexity logarithmic in the number of particles and landmarks.

A stark contrast to the filter-based approach, keyframe-based SLAM algorithms eliminate the probabilistic relationship between tracking and mapping. The parallel tracking and mapping (SLAM) algorithm of Klein and Murray\cite{24} splits tracking (localization) and map management (mapping) into two separate asynchronous threads. As opposed to filter-based methods which only retain the knowledge of the past states as a probability distribution, keyframe-based methods explicitly retain a sub-set of the past states as a set of observations known as keyframes.

Recently, these two approaches have become more and more interconnected, with recent publications such as \cite{22};\cite{25};\cite{37} adopting the advantages of both approaches. Strasdat et al.\cite{50} have shown both by theoretical considerations as well as practical comparisons, that in general keyframe-based approaches are more promising than filtering. More precisely, they argue that increasing the number of landmarks is more beneficial than incorporating more observations in terms of accuracy gained per unit of computation.

Particularly the second task - optimizing a map consisting of keyframes, landmarks and observations - has received significant attention: Strasdat et al. have explored different bundle adjustment strategies \cite{52} and proposed large scale and loop closure methods \cite{51}. Kummerle et al. developed a general framework for graph optimization called g2o\cite{26}, generalizing the idea of bundle adjustments: In particular, this framework allows to include explicit constraints on the relationship between two keyframes (relative rotation and translation) into the bundle adjustment process, allowing for truly constant-time visual SLAM. Using this approach, the size of the mapped area is only constrained by the memory required to store the map, and not by computational power available.

At this point it is important to make the distinction between visual SLAM and visual odometry. The former maintains and continuously optimizes a map of the environment in addition to tracking the pose of the camera while the latter only estimates the camera pose without necessarily maintaining a map of the environment \cite{46}.
3.6.1 Algorithm Outline

The Visual SLAM algorithm used in this thesis is the monocular parallel tracking and mapping algorithm of Klein and Murray [24]. The algorithm splits the task of camera tracking and map management into two separate asynchronous threads which run in parallel. An additional map initialization step is required to generate an initial map of the environment since both camera tracking and map management threads need a map to operate on. The interaction between these three components is visualized in Figure 3.6.

![Figure 3.6: Schematic outline of a keyframe-based SLAM algorithm. Three main parts can be identified: Map Initialization, Camera Tracking and Map Management. While the Map Initialization stage only generates the initial map, Camera Tracking and Map Management run simultaneously in two continuous loops, both operating on the same map.](image)

- **Map initialization**: Generates an initial map of the environment from pure translational motion of the camera. Is only performed once before the start of the algorithm or when tracking is lost.

- **Map management**: Running in a loop, the map management thread continuously updates the maintained map by incorporating new keyframes and optimizing the existing map.

- **Camera tracking**: Running in a loop, the camera tracking thread continuously estimates the 6-DOF pose of the camera for each new image frame of the image stream.
3.6.2 Map Initialization

One of the fundamental difficulties in keyframe-based, visual SLAM is the inherent chicken-and-egg like nature of the problem: to build a map, the ability to track the camera motion is required, which in turn requires the existence of a map. This issue arises from the lack of depth information. A common approach to solve this is to apply a separate initialization procedure:

1. Take the first keyframe \( K_1 \) and detect promising feature points \( x_1, \ldots, x_n \) using e.g. the FAST corner detector (See Section 3.5.2).

2. Track feature points using a simple frame-to-frame tracking approach as described in Section 3.5.5 until the camera has translated far enough such that the landmark positions can be triangulated.

3. Take the second keyframe \( K_2 \) and extract new feature point positions \( x'_1, \ldots, x'_n \)

4. Generate the initial map from these point-correspondences (Section 3.4). The scale of the map as well as the position and orientation of the first keyframe can be defined arbitrarily, due to numerical considerations the initial map is often scaled such that the average distance between camera and landmarks is one.

3.6.3 Map Optimization

Given all observations \( x_{ij} \), the goal is to refine the keyframe and landmark positions \( K_j \) and \( p_i \), such that they best coincide with these observations. Assuming independent, Gaussian noise on each observation in pixels, the maximum-likelihood solution to this problem is obtained by minimizing the total reprojection error \( E_{rep} \). Let the reprojection error of a single observation \( x \) of landmark \( p \) with three-dimensional coordinates \( X \), and from a camera pose \( C \) be defined as

\[
e(x, X, C) = \bar{x} - \text{proj}(K_{cam}C X) \in \mathbb{R}^2
\]

which corresponds to the pixel distance between the location where the landmark actually was observed and its projection onto the image. In the remainder of this section, we use \( e_{ij} = e(x, X, C) \) for the reprojection error of an observation \( x_{ij} \) contained in the map. The total reprojection error is now given by:

\[
E_{rep}(x_1, \ldots, x_n, K_1, \ldots, K_m) = \sum_{i=1}^{n} \sum_{j=1}^{m} \rho \left( \frac{|e_{ij}|}{\sigma_{ij}^2} \right)
\]
where \( \rho \) is a robust kernel function. Minimizing this error function, using an iterative method such as Levenberg-Marquardt is referred to as global bundle adjustment (BA).

For a growing number of landmarks and keyframes, optimizing this error function as a whole each time a new keyframe or landmark is added quickly becomes computationally unfeasible. This gives rise to the concept of local bundle adjustments. Optimization of equation (3.54) is performed by only considering a small subset of keyframes and a corresponding subset of landmarks, keeping everything else fixed: after adding a new keyframe, optimizing only over the most recently added keyframes and a corresponding set of landmarks may be sufficient, assuming that the estimates for older keyframes and landmarks already are comparatively accurate. It is to be mentioned, that the total reprojection error of equation (3.54) is a non-convex function and hence only a local and possibly suboptimal minimum will be found.

### 3.6.4 Adding Keyframes and Landmarks

When a new keyframe has been identified by the tracking part, it can simply be added to the set of keyframes using the known camera position and observations of existing landmarks. To generate new landmarks, additional points from the new frame are extracted and matched to points in other keyframes, using epipolar search - if a match is found, the respective landmark position can be triangulated and is added to the map. Afterwards, a local bundle adjustment step, including the new keyframe and all new landmarks is performed.

### 3.6.5 Further Map Managing

Apart from bundle adjustments and incorporation of new keyframes, additional routines are required to remove outliers, merge landmarks corresponding to the same three-dimensional point, or identify additional observations and landmarks. The map management loop continuously executes these tasks.

### 3.7 Camera Tracking

The camera tracking loop is executed once for each new video-frame \( I \), and calculates the corresponding camera position \( C \), based on the known coordinates \( X_1, \ldots, X_n \) of landmark points \( p_1, \ldots, p_n \). It requires an initial guess of the camera pose \( C_0 \), for example the pose in the previous frame.
3.7.1 Pose Estimation

First, all potentially visible landmarks are projected into the image based on the expected camera position $C_0$, and for each such landmark, a warped template of its expected appearance is generated from a keyframe it was observed in. Using a tracking approach such as the one described in Section 3.5.5, the exact location of the landmark in the image is then computed to subpixel accuracy. The result of this stage is a set of $k$ 3D-2D point correspondences, $X_1, \ldots, X_k$ and $x_1, \ldots, x_k$.

Based on these 3D-2D point correspondences, the camera position $C$ is estimated using the method described in Section 3.3.1. Note that the tracking process as described here does not require any feature descriptors (SIFT, SURF, ORB). This is in contrast to many other approaches, in particular filtering-based ones where feature points are frequently matched based on such descriptors.

3.7.2 Tracking Recovery

When tracking is lost, the above method cannot be applied as no initial estimate of the camera position $C_0$ is available. A separate recovery procedure is therefore required. There are several possibilities to achieve this, for example using viewpoint invariant feature descriptors such as SIFT and SURF. The SLAM algorithm tries to recover by comparing the downscaled current frame with all existing keyframes, trying to find an approximate match. Based on this match, an initial guess of the camera orientation, assuming the position to be that of the keyframe is computed. The resulting camera pose is then used as $C_0$ and the normal tracking procedure as described in Section 3.7.1 is applied.

3.7.3 Identifying New Keyframes

The tracking part also decides if a frame will be added as a new keyframe based on heuristic criteria such as:

- tracking quality is good (a high fraction of landmarks has been found),
- no keyframe was added for some time,
- no keyframe was taken from a point close to the current camera position.
3.7.4 Further Tracking Aspects

In practice, several improvements are possible. For example, a motion-model can be applied to compute a better initial estimate of the camera position $C_0$, or an additional coarse tracking stage, only considering keypoints found on a higher scale may be added in order to track fast motion.
Chapter 4

Kalman Filter Sensor Fusion

![Figure 4.1: Example of filtering: the true state is shown in black, the (noisy) measurements are shown in green. The blue line corresponds to a simple running-average low-pass filtered data - while this does lead to significantly smoother results, an artificial delay is introduced. The red line shows the result of a Kalman filter.](image)

In robotics one generally deals with dynamic systems, i.e. (physical) systems that change their state over time. For a quadrotor MAV, for example, the state might consist of its current position, orientation and velocity. In order to accurately control such a system, sensors are used to collect
information and infer the current state as accurately as possible. Measurements from real-world sensors however are always subject to measurement errors (also called noise), hence using only the most recent measurement often leads to unstable and poor results. Furthermore in many cases multiple sensors can be used to acquire information about the same state variable - in order to get an optimal estimate, these measurements can then be combined to give a single estimate with higher accuracy. The goal of data fusion and filtering algorithms is to use sensor measurements and knowledge about the dynamics of the system to gain an accurate estimate of the system’s current state.

Figure 4.1 shows an example of a one-dimensional Kalman filter. The true state is shown in black, the noisy raw measurements are shown in green. For comparison, the blue line plots the result of a simple running average low-pass filter, and the red-line shows the result of a Kalman filter. The filtered output of the Kalman filter does smooths the data without introducing an artificial delay, unlike the running average filter.

4.1 Linear Kalman Filter

Figure 4.2: Prediction and update step for a one-dimensional Kalman filter. While the prediction step corresponds to a convolution of two Gaussians, the update step is an application of Bayes formula: \( \text{posterior} \propto \text{likelihood} \cdot \text{prior} \). For both steps, the resulting distribution again is an analytically computable Gaussian.

The Kalman filter is a well-known method to filter and fuse noisy measurements of a dynamic system to get a good estimate of the current state. It assumes that all observed and latent variables have a (multivariate) Gaussian distribution, the measurements are subject to independent, Gaussian noise and the system is linear. Under these assumptions it can even be shown that the Kalman
filter is the optimal method to compute an estimate of the current state as well as the uncertainty (covariance) of this estimate. In the remainder of this section, we use the following notation:

- $n, m, d$: dimension of the state vector, measurement vector and control vector respectively,
- $x_t \in \mathbb{R}^n \sim \mathcal{N}(\mu_t, \Sigma_t)$: true state at time $t$. The estimate of this vector, incorporating measurements up to and including the measurement at time $t - 1$ is denoted by $\hat{x}_t \sim \mathcal{N}(\hat{\mu}_t, \hat{\Sigma}_t)$,
- $\mu_t \in \mathbb{R}^n$: mean of the state vector $x_t$,
- $\Sigma_t \in \mathbb{R}^{n \times n}$: covariance of the state vector $x_t$,
- $\bar{\mu}_t \in \mathbb{R}^n$: estimated mean of $x_t$,
- $\bar{\Sigma}_t \in \mathbb{R}^{n \times n}$: estimated covariance of $x_t$,
- $B_t \in \mathbb{R}^{n \times d}$: control-input model, mapping the control vector $u_t \in \mathbb{R}^d$ to its effect on the internal state,
- $A_t \in \mathbb{R}^{n \times n}$ state transition model, mapping the state $x_{t-1}$ at time $t - 1$ to the state $x_t$ at time $t$. This transition is assumed to be subject to zero-mean Gaussian noise $w_t \sim \mathcal{N}(0, R_t)$, where $R_t$ is known: $x_t = A_t x_{t-1} + B_t u_t + w_t$,
- $z_t \in \mathbb{R}^m$: observation at time $t$. Again this observation is assumed to be subject to zero-mean Gaussian noise $v_t \sim \mathcal{N}(0, Q_t)$, where $Q_t$ is known: $z_t = C_t x_t + v_t$ where $C_t \in \mathbb{R}^{m \times n}$ is the observation model, mapping the internal state to the respective expected observation.

Using the above definitions, the linear Kalman filter now operates in a continuous prediction-update-loop:

1. **Predicting** the state $\hat{x}_t \sim \mathcal{N}(\hat{\mu}_t, \hat{\Sigma}_t)$ ahead in time, increasing uncertainty:

   $$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$
   $$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$  \hspace{1cm} (4.1)

2. **Updating** the state $x_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ by incorporating an observation, decreasing uncertainty:

   $$K_t = \Sigma_t C_t^T (C_t \Sigma_t C_t^T + Q_t)^{-1}$$
   $$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$
   $$\Sigma_t = (I - K_tC_t)\bar{\Sigma}_t$$  \hspace{1cm} (4.2)
A full derivation and proof of these formulae is beyond the scope of this work, for further details the reader is referred to Thrun et al.\[53]\.

### 4.2 Extended Kalman Filter

The extended Kalman filter drops the assumption of a linear system, making it applicable to a much wider range of real-world problems. The only difference is that observation, state transition and control model can now be defined by any two differentiable functions $g: \mathbb{R}^n \times \mathbb{R}^d \rightarrow \mathbb{R}^n$ and $h: \mathbb{R}^n \rightarrow \mathbb{R}^m$:

\[
x_t = g(x_{t-1}, u_t) + w_t \\
z_k = h(x_t) + v_t
\]

The difficulty now lies in the fact that when we apply a nonlinear transformation to a Gaussian random variable, the resulting random variable is no longer Gaussian: in order to still make the above framework applicable, $g$ and $h$ are approximated by a first-order Taylor approximation, which however leads to the result no longer being optimal. Writing $g$ and $h$ in terms of their first-order Taylor approximations:

\[
g(x_{t-1}, u_t) \approx g(\mu_{t-1}, u_t) + g'(\mu_{t-1}, u_t)(x_{t-1} - \mu_{t-1}) \\
= g(\mu_{t-1}, u_t) + G_t(x_{t-1} - \mu_{t-1})
\]

\[
h(x_t) \approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)(x_t - \bar{\mu}_t) \\
= h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t)
\]

The prediction-update loop for extended Kalman filter can now be written as:

1. **Prediction**

\[
\bar{\mu}_t = g(\mu_{t-1}, u_t) \\
\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t
\]
2. Update

\[ K_t = \Sigma_t H_t^T (H_t \Sigma_t H_t^T + Q_t)^{-1} \]
\[ \mu_t = \mu_t + K_t (z_t - h(\mu_t)) \]
\[ \Sigma_t = (I - K_t H_t) \Sigma_t \]

\[ (4.7) \]

4.3 Unscented Kalman Filter

The unscented Kalman filter is a further improvement on the extended Kalman filter, often leading to more robust and accurate results \[21\][55]. It addresses the problem that using a first order Taylor approximation to transform a probability distribution in general leads to poor results when \( f \) is highly nonlinear: instead of linearization, the transformed distribution is estimated from a number of nonlinearly transformed sample points from the original distribution.

4.4 Sequential Monte Carlo Filters

Sequential Monte Carlo also known as particle filters further relax the assumptions made: both state and observation are no longer required to have a Gaussian distribution - allowing for example to track multiple hypothesis simultaneously. This is achieved by characterizing the distribution using a set of sample points called particles. The major drawback of this method is, that the number of particles required grows exponentially in the dimension of the state - often rendering this approach computationally unfeasible for large \( n \).
Chapter 5

PID Controller

Control theory deals with the problem of controlling the behavior of a dynamic system, i.e. a (physical) system that changes its state over time and which can be controlled by one or more system input values. The general goal is to calculate system input values $u(t)$, such that the system reaches and holds a desired state. In other words, the measured error $e(t)$ between a given setpoint $w(t)$ and the measured output of the system $y(t)$ is to be minimized over time. In particular, the goal is to quickly reach the desired setpoint and hold it without oscillating around it, counteracting any random disturbances introduced into the system by the environment. This process is schematically represented in Figure 5.1.

![Schematic representation of a general control loop. The goal is to calculate system input values $u(t)$ such that the measured error $e(t) = w(t) - y(t)$ is minimized.](image)

In this chapter, we present the proportional-integral-derivative controller (PID controller), a generic control loop feedback mechanism widely used in industrial control systems [54]. A PID controller is used in our approach to directly control the quadrotor MAV. It is based on three separate control mechanisms, the control signal being a weighted sum of all three terms:

- The **proportional** part depends on the current error $e(t)$,
• The integral part depends on the accumulated past error $\int_0^t e(\tau) d\tau$

• The derivative part depends on the predicted future error, based on the derivative of the error with respect to time $\dot{e}(t)$.

If the integral and derivative of the error cannot be measured directly, they are approximated by numeric integration and differentiation: $\int_0^t e(\tau) d\tau \approx \Sigma_{\tau=1}^t e(t)$ and $\dot{e}(t) \approx \frac{1}{\delta_t} (e(t) - e(t - \delta_t))$. The PID controller now calculates the system input values according to:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \dot{e}(t) \quad (5.1)$$

where $k_p$, $k_i$ and $k_d$ are tunable filter coefficients that typically are determined experimentally by the means of trial-and-error, however there are heuristic methods and guidelines to help this process. The effect of these three parts of a PID-controller is explained below.

Figure 5.2: (a) Proportional control. (b) The same system with an added derivative term.

### 5.1 The Proportional Term

The proportional part is always required, and is the part responsible for reducing the error: the bigger the error, the stronger the control signal. In real-world systems however, a purely proportional controller causes severe overshoots, leading to strong oscillations. The behavior of a typical system, controlled by a purely proportional controller is visualized in Figure 5.2a.
5.2 The Derivative Term

The derivative part has the effect of dampening occurring oscillations: the higher the rate of change of the error, the more this term contributes towards slowing down this rate of change, reducing overshoot and oscillations. The effect of adding a derivative term to the controller is visualized in Figure 5.2b.

5.3 The Integral Term

The integral part is responsible for eliminating steady-state errors: for a biased system requiring a constant control input to hold a state, a pure PD-controller will settle above or below the setpoint. Depending on accumulated past error, the integral term compensates for this bias - it however needs to be treated with caution as it may increase convergence time and cause strong oscillations. The effect of adding an integral term to a PD-controller is visualized in Figure 5.3.

Figure 5.3: (a) PD-control of a biased system, resulting in a steady-state error. (b) the same system when adding an integral term. While for this system overshooting is unavoidable, the integral term causes it to still settle at the correct value. (c) the three distinct control terms working together.
Chapter 6

Implementation

With all the building blocks now at our disposal, we will, in this chapter, describe our approach to the problem and the system developed and implemented in the course of this thesis. In Section 6.1, we first outline our approach, summarizing the methods used and how they interact. We briefly present the software architecture in Section 6.2. In the following three sections, we describe the three main components of our approach: in Section 6.3, we describe the visual SLAM system, and how it is augmented to benefit from the additional sensor measurements. In Section 6.4, we describe the extended Kalman filter used to combine information from the different sensors available and the prediction model. In Section 6.5, we describe the PID controller to control the MAV.

6.1 Outline

Our approach comprises of the following parts

- **Visual SLAM**: a visual SLAM algorithm as described in Chapter 3 is applied to incoming video frames and computes an estimate of the MAV’s pose, based on a predicted pose calculated by the Kalman filter.

- **Extended Kalman filter**: to combine the pose estimate provided by visual SLAM with available sensor information as well as the predicted effect of sent control commands, an extended Kalman filter (EKF) as introduced in Chapter 4 is developed.

- **PID Controller**: from an estimate of the MAV’s pose and velocity provided by the EKF, a PID-controller as introduced in Chapter 5 is used to generate control commands.
6.2 Software Architecture

The first task to be solved in this thesis was to build a reliable, lightweight software system to control the AR Drone 2.0. In particular, the challenge was to develop a fault-tolerant system, for example it needs to quickly and automatically detect and restore a lost connection to the MAV without losing the internal state of the control system. We achieved this by encapsulating all direct MAV-communication in a separate process: the MAV proxy process offers a clean local interface for accessing video and navigational data, as well as sending control commands to the MAV. Furthermore it can record and replay all data sent by the MAV, simulating a real flight and significantly facilitating the debugging process. The software architecture of our system is shown in Figure 6.1.

A second process, the UI process monitors the state of the MAV proxy and automatically restarts it when a connection loss or error is detected. It also offers the possibility of manually controlling the MAV via keyboard and serves as a graphical interface for managing connections and recording or replaying flights.

The main algorithmic component runs in a third process: the control process contains the SLAM system, the Kalman filter and the PID-controller, as well as video and map visualization. When controlling the MAV manually, this process is not required - allowing for immediate manual takeover when it is interrupted for debugging purposes.

These three processes communicate using named pipes for interchanging messages, as well as shared memory regions, mutexes and events for asynchronously interchanging video, navigational
and control data.

6.3 Visual SLAM

For the task of visual SLAM, our solution is based on the open-source parallel tracking and mapping system by Klein and Murray [24], a well-known and widely used keyframe-based visual SLAM system as described in Chapter 3. It is augmented to make use of the additional sensor information available, in particular the main drawback of the visual SLAM algorithm, the impossibility to recover the scale of the scene, is resolved.

6.3.1 Scale Estimation

One of the major shortcomings of visual SLAM algorithms using a single camera is, that the ambiguous scale $\mu$ of the map cannot be determined without using either additional sensor information or knowledge about objects present in the scene. To use a visual SLAM method for navigation however, an estimate for this scale factor is important - it is required for specifying a flight path in euclidean space and calculating appropriate control commands. The scale value is updated in regular time intervals using a pair of position measurements from the sensors and the SLAM algorithm. The horizontal speeds are calculated on-board by the MAV, using the ultrasound altimeter and an optical-flow based algorithm on the bottom camera. The ultrasound altimeter measures the relative height, which is subject to drift for uneven ground.

The distance traveled in time interval $t : t + \delta t$ is measured by both the SLAM system and the available sensors, creating a pair of measurements $x_{SLAM}$ and $x_{IMU}$

$$x_{SLAM} = \|d_{SLAM}(t + \delta t) - d_{SLAM}(t)\|$$

$$x_{IMU} = \left\| \int_{t}^{t+\delta t} v_{IMU}(t)dt \right\|$$

where

- $d_{SLAM}(t)$ and $d_{SLAM}(t + \delta t)$ are position estimates from the SLAM algorithm at times $t$ and $t + \delta t$,

- $v_{IMU}(t)$ is a vector of velocity estimates from the MAV’s sensors.

The scale $\mu$ can be found from a collection of $n$ past pairs
\[
\mu = \text{median} \left( \frac{x_{\text{IMU}_i}}{x_{\text{SLAM}_i}} \right) \quad i = 1, \ldots, n
\]  

(6.2)

6.3.2 Integrating Sensor Data

In addition to scaling and orientating the map correctly, the IMU measurements available are used to increase both resilience and accuracy of the SLAM algorithm. The Kalman filter is used to generate an accurate prior estimate of the MAV’s position for each video frame, which is then utilized in three ways:

- it is used as initialization for tracking \((C_0)\), replacing the SLAM algorithm’s built-in decaying velocity model.
- when tracking is lost, it is used as alternative initialization for the recovery process, in some cases speeding up recovery.
- when the roll or pitch angle measured by the IMU deviate strongly from SLAM’s pose estimate, tracking is assumed to be lost and the respective result discarded, in particular it is not added as keyframe. This basic validity check drastically reduces the probability of permanently corrupting the map by adding false keyframes.

6.4 State Estimation and Prediction

6.4.1 State Vector

The internal state of the Kalman filter is defined as a vector

\[
v_t = \left( x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi \right)^T
\]  

(6.3)

where

- \(x, y, z\) correspond to the world-coordinates of the MAV center in meters,
- \(\dot{x}, \dot{y}, \dot{z}\) correspond to the velocity of the MAV in meters per second, expressed in the world (inertial) coordinate system,
- \(\psi, \theta, \phi\) correspond to roll angle, pitch angle and yaw angle in degrees, representing the MAV’s orientation. While in general such a representation of a 3D orientation is problematic due to
ambiguities and loss of one degree of freedom at certain configurations (Gimbal lock), the fact that both roll and pitch angle are always small makes this representation well suited.

• $\psi$ corresponds to the yaw-rotational rate in degrees per second.

While the three position and orientation parameters, for a rigid object are independent of each other, the same is not true for a quadrotor MAV in flight. For instance, a change in roll $\psi$ and pitch $\theta$ angles, will cause a change in the lateral position $(x, y)$ of the MAV. A change in yaw $\phi$ or the height $z$ has no affect on any of the other parameters. The coupling of the state parameters will become more evident in the rest of this section.

As the state changes over time and measurements are integrated in irregular intervals, the respective values will be treated as continuous functions of time when appropriate. For better readability, the time argument is omitted when clear from context.

6.4.2 Observation Model

The observation model calculates the expected measurements based on the current state of the MAV. As two distinct and asynchronous observation sources are available, two separate observation models are required. We adopt the notation introduced in Chapter 5, that is $z \in \mathbb{R}^m$ denotes the measurement vector, while $h$ denotes the observation model, mapping the current state to the respective expected observation.

Visual SLAM Observation Model  The visual SLAM system generates an estimate of the MAV’s pose for every video frame, that is approximately every 30 ms, which due to the very limited resolution of 640 x 480 pixel is subject to significant noise. This pose estimate is treated as direct observation of the respective state parameters, that is

$$h_{\text{SLAM}}(x) = \left( x, y, z, \phi, \theta, \psi \right)^T \in \mathbb{R}^6$$

(6.4)

and

$$z_{\text{SLAM}} = \log(M_{\text{SLAM}}^P M_{\text{SLAM}})$$

(6.5)

where $M_{\text{SLAM}}$ is the camera pose as estimated by the SLAM algorithm, $M_{\text{SLAM}}^P$ the (constant) transformation transforming the camera coordinate system to the MAV coordinate system, and $\log$ is the transformation from a pose matrix to a pose vector $(t_x, t_y, t_z, \psi, \theta, \psi)^T$. 

52
**Sensor Observation Model**  The AR.Drone 2.0 sends updated IMU measurements every 5 ms. These measurements however do not correspond to raw sensor measurements, but have already been subject to filtering and other pre-processing steps - as a result, they heavily violate the assumption of independent measurement noise. To compensate for this, the respective measurement variances are chosen to be comparatively high. The sensor values used are:

- **Horizontal Speed**: $\dot{x}_b, \dot{y}_b$ measuring the forward and side way movement of the body frame (See Chapter 2.0)

- **Relative height**: $h$ this value corresponds to the MAV’s relative height, and is measured every 40 ms. Assuming a flat ground surface with occasional discontinuities, changes corresponding to a vertical speed of more than $3 \, \text{m/s}$ are filtered out. Nevertheless, treating it as direct observation of the MAV’s height would be problematic: Deviations due to an uneven ground or an inaccurately estimated scale $\mu$ would cause unstable and oscillating values for the MAV’s estimated height and in particular its vertical speed. Instead, only the change in height is used: Let $z(t - \delta t)$ be the height of the MAV according to the filter after the last height-observation, and $h(t - \delta t)$ and $h(t)$ the measured relative heights at time $t - \delta t$ and $t$. The observed absolute height at time $t$ is then given by $z(t - \delta t) + h(t) - h(t - \delta t)$.

- **Roll and Pitch Angles**: $\dot{\phi}, \dot{\theta}$ as these sensor values are very accurate, drift-free and calibrated with respect to the SLAM map after the initialization procedure, they can be treated as direct observations of the respective state variables.

- **Yaw Angle**: $\dot{\psi}$ as this sensor value is subject to significant drift over time, it is handled the same way as the relative height measurements.

The resulting measurement vector and observation functions are:

$$h_{IMU}(\mathbf{x}) = \begin{pmatrix} x \cos \psi - y \sin \psi \\ x \sin \psi + y \cos \psi \\ z \\ \phi \\ \theta \\ \psi \end{pmatrix}$$

and

53
\[ \mathbf{z}_{IMU}(x) = \begin{pmatrix} \dot{x}_b \\ \dot{y}_b \\ z(t - \delta t) + h(t) - h(t - \delta t) \\ \dot{\phi} \\ \dot{\theta} \\ \psi(t - \delta t) + \dot{\psi}(t) - \dot{\psi}(t - \delta t) \end{pmatrix} \]  

(6.7)

6.4.3 State Transition Model

The state transition model \( g \) propagates the state through time, that is based on the state \( x(t) \) and the active control command \( u(t) \) at time \( t \), a prediction for the state \( x(t + \delta t) \) at time \( t + \delta t \) is calculated. As the AR.Drone 2.0 sends accurate timestamps in microseconds with each sensor measurement package, these can be used to determine the exact prediction intervals \( \delta t \). Prediction is thus performed in irregular time steps of \( \delta t \approx 5\text{ms} \). In this section, we describe the state transition model and the control model used, and how it is derived. In Section 6.4.4 we describe how the required model coefficients are estimated from test-flights.

**Horizontal Acceleration**  The horizontal velocity of the MAV changes according to the horizontal acceleration, which depends on its current attitude. It is given by

\[ \mathbf{f} = \begin{pmatrix} m\ddot{x} \\ m\ddot{\hat{x}} \end{pmatrix} \]  

(6.8)

where \( m \) is the MAV's mass and \( \mathbf{f} \) the sum of all horizontal forces acting upon the MAV. The horizontally acting force \( \mathbf{f} \) consists of two components:

- **Drag Force**: \( f_d \) for a comparatively slow-moving object, the force acting upon it due to air resistance is approximately proportional to its current velocity, that is

\[ f_d \approx \begin{pmatrix} -b_x \ddot{x} \\ -b_y \ddot{\dot{x}} \end{pmatrix} \]  

(6.9)

where \(-b_x, -b_y\) are constants.

- **Rotor Thrust**: \( f_t \) the rotors are assumed to generate a constant force acting along the MAV's \( z \)-axis. If it is tilted, a portion of this force acts horizontally, which can be found by projecting
the thrust vector onto the MAV frame

Assuming \( f_d \) and \( f_t \) to be constant over the short time period the MAV’s horizontal acceleration is given by

\[
\begin{align*}
\dot{x} &= \frac{1}{m} \left( k_x (cos\phi sin\theta cos\psi + sin\phi sin\psi) - b_x \dot{x} \right) \\
\dot{y} &= \frac{1}{m} \left( k_y (cos\phi sin\theta sin\psi - sin\phi cos\theta) - b_y \dot{y} \right)
\end{align*}
\]  

(6.10)

where \( k_x, k_y, b_x, b_y \) are unknown model coefficients.

**Influence of Control Commands** The control command \( u = (\dot{\phi}, \dot{\theta}, \dot{\psi}, \ddot{z})^T \) defines the desired roll and pitch angles, the desired yaw rotational speed as well as the desired vertical speed of the MAV as a fraction of the maximal value permitted. These parameters serve as input values for a low-level flight controller running on-board the MAV, which then adjusts the engine speeds accordingly. The on-board flight controller also automatically stabilizes the MAV when it is in flight. The behavior of our high-level controller is modeled by approximating the roll and pitch rotational speed, the vertical acceleration as well as the yaw rotational acceleration based on the current state and the sent control command. We use a model similar to the one derived for the horizontal accelerations, that is

\[
\begin{pmatrix}
\dot{\phi} \\
\dot{\theta} \\
\ddot{\psi} \\
\ddot{z}
\end{pmatrix} =
\begin{pmatrix}
\lambda_1 (\lambda_2 \ddot{\phi} - \phi) \\
\lambda_3 (\lambda_4 \ddot{\theta} - \theta) \\
\lambda_5 (\lambda_6 \ddot{\psi} - \dot{\psi}) \\
\lambda_7 (\lambda_8 \ddot{z} - \dot{z})
\end{pmatrix}
\]  

(6.11)

where \( \lambda_1 \) to \( \lambda_8 \) are model constants which are determined experimentally in Section 6.4.4.

**State Transition Function** The complete state update \( x(t + \delta t) \) for a time period of \( \delta t \) is given by:

---

55
\[
\begin{pmatrix}
  x \\
y \\
z \\
\hat{x} \\
\hat{y} \\
\hat{z} \\
\phi \\
\theta \\
\psi \\
\end{pmatrix}_t + \delta t \begin{pmatrix}
  \delta \hat{x} \\
  \delta \hat{y} \\
  \delta \hat{z} \\
  \delta t\lambda_7 (\lambda_8 \hat{z} - \hat{\dot{z}}) \\
  \delta t\lambda_1 (\lambda_2 \hat{\phi} - \hat{\phi}) \\
  \delta t\lambda_3 (\lambda_4 \hat{\theta} - \hat{\theta}) \\
  \delta t\lambda_5 (\lambda_6 \hat{\psi} - \hat{\dot{\psi}}) \\
\end{pmatrix}_t
\] (6.12)

\[k_x, b_x = \arg\min_{k_x, b_x} \sum_t (\hat{x}(t) - \tilde{x}(t))^2\] (6.13)

6.4.4 Estimation of Model Coefficients

The constant coefficients of the model derived above, \(k_x, k_y, b_x, b_y\) and \(\lambda_1\) to \(\lambda_8\), are estimated by minimizing the difference between the values predicted by the model and the true values obtained by a motion capture system over a series of test flights.

In order to control the MAV we use a PID controller, taking the predicted MAV state as input. In particular we directly use the speed estimates maintained by the Kalman filter. Let \(x = (x, y, z, \hat{x}, \hat{y}, \hat{z}, \phi, \theta, \psi, \tilde{\psi})^T\) be the predicted state of the MAV, and \(p = (\hat{x}, \hat{y}, \hat{z}, \tilde{\psi})^T\) the target position and yaw angle. The control signal \(u = (\hat{\phi}, \hat{\theta}, \tilde{\psi}, \tilde{z})^T\) is now calculated by applying PID control to these four parameters:

\[
\begin{pmatrix}
  \phi \\
  \theta \\
  \tilde{\psi} \\
  \tilde{z} \\
\end{pmatrix} = \begin{pmatrix}
  k_p^\phi (\hat{x} - x) + k_i^\phi \int \hat{x} - x \\
  k_p^\theta (\hat{y} - y) + k_i^\theta \int \hat{y} - y \\
  k_p^\psi (\tilde{\psi} - \psi) + k_i^\psi \int \tilde{\psi} - \psi \\
  k_p^z (\tilde{z} - z) + k_i^z \int \tilde{z} - z \\
\end{pmatrix}
\] (6.14)
Chapter 7

Results

Figure 7.1: Left: The real-time trajectory (plotted in green) of the MAV flying a Vertical-Square as it is estimated by the Visual SLAM algorithm, Right: The feature points tracked by the SLAM algorithm. The color of the points correspond to the pyramid-level they were detected in.

In this chapter we will evaluate the accuracy of the developed system by comparing the estimated state of our system with ground-truth MAV trajectories obtained using an OptiTrack motion capture system, which will alternatively be referred to as the ground truth. Figure 7.1 shows the visual output of our developed system. The image on the left shows the real-time trajectory of the MAV plotted in green and map points in red. The image on the right shows the live image stream from MAV’s camera with the ground plane and tracked points drawn on top.

Before the data recorded from two separate sources can be compared, it needs to first be synchronized in time and space (i.e. both sets of data need to be represented in the same reference frame). Let the raw position estimates from the EKF and the OptiTrack be written as 3 dimensional vectors $x_t \in \mathbb{R}^3$ and $y_t \in \mathbb{R}^3$ respectively. The transformation $M^i_t \in SE(3)$ between the inertial reference
frame $K_i$ and the OptiTrack reference frame $K_o$ and the time difference $\tau$ can then be found by minimizing:

$$\arg\min_{M_i^o} \sum_{t=0}^{n} \sum_{\tau=0}^{m} \|x_t - M_i^o(y_t + \tau)\|^2 \quad \forall t = 0, \ldots, n$$

(7.1)

Once the optimal transformation $M_i^o \in SE(3)$ and the time difference is determined, the data can be aligned for comparison by mapping both sets of data to a single reference frame and correcting for the time difference.

Figure 7.2: Trajectory Comparison: Autonomous Control (W-figure): comparison of the system’s estimated position (solid red) with the ground truth (solid black) of an autonomously flown W-figure.

Figure 7.2 plots the trajectory measurements of an autonomously flown W-figure. Starting from an arbitrary initial hovering position, the MAV was instructed to fly through four way-points, completing a W-figure before flying to the final destination point which was set at the top-center of the W-figure.

The dashed red line plots the estimated trajectory of the MAV without incorporating the Visual SLAM pose. The blue cross marks the start of visual SLAM pose incorporation. The abrupt jump in the filtered position estimate is caused by the correction of the accumulated drift of the IMU-only trajectory upon the inclusion of Visual SLAM algorithm’s estimates into the EKF. The solid black line plots the ground truth trajectory. The R.M.S error of the estimated position was $[0.072, 0.086, 0.101]$ meters.
Figure 7.3 shows the $x$, $y$ and $z$ components of the postion estimated by both the EKF (solid red line) and the OptiTrack system (solid black line) during an autonomously flown horizon-square figure. The R.M.S error of the estimated position was $[0.011, 0.019, 0.023]$ meters.
Figure 7.4: Trajectory Comparison: Autonomous Control (Position Hold)

Figure 7.4 depicts the trajectories of the system’s estimated position (solid red) and the ground truth (solid black) during an autonomous position hold maneuver. The R.M.S error of the estimated position (solid red) was [0.136, 0.321, 0.172] meters. The position maneuver represents a scenario where the SLAM system is least effective due to the lack of enough translational movement to incorporate new keyframes into the map, which leads to a higher error in the position estimates.
Figure 7.5 shows the comparison of the system’s estimated position (solid red) with the ground truth (solid black) during an autonomously flown position-hold maneuver. The R.M.S error of the estimated position (solid red) was [0.312, 0.517, 0.193] meters.

Of particular note is the relatively low error in the vertical z estimates during a position hold. This can be attributed to the additional range information from the bottom facing sonar sensor.
Figure 7.6: Orientation Angles Comparison: Autonomous Control (Position Hold)

Figure 7.6 shows the comparison of the system’s estimated orientation angles (solid red) with the ground truth (solid black) during an autonomously flown Position Hold maneuver. The R.M.S error of the estimated position (solid red) was [0.009, 0.007, 0.018] degrees.
Chapter 8

Conclusion

In this thesis, we developed a system enabling a quadrotor MAV (AR Drone 2.0) to navigate in environments of which it possesses no a priori information. The system is particularly suited for indoor environments where access to GPS is not available. Our approach uses a single camera on-board the quadrotor, and does not require artificial markers or external sensors.

To achieve this, our approach employs a keyframe-based, visual simultaneous localization and mapping (SLAM) system to build a sparse map of the environment, and to compute a visual pose estimate within this map for each video frame. We developed an extended Kalman filter (EKF) to fuse this visual pose estimate with additional sensor measurements available, to synchronize the different data sources. This is achieved by formulating and implementing a full model of the MAV’s flight dynamics and the effect of sent control commands. Proportional-integral-differential (PID) control is applied to control the pose of the MAV, and fly to and hold a desired target position; our approach allows the MAV to fly to a specified target position and to accurately hold a flying position in three-dimensional space.

On one hand, the system is robust to occlusions or when the MAV loses sight of the landmarks, it is able to approximately return to its original pose using the IMU and altimeter measurements. On the other hand, the visual tracking eliminates drift, allowing the MAV to fly in a large environment and land close to its takeoff position.

In summary, we developed a system that enables a MAV to be controlled and navigated in an unknown environments using only on a single low resolution camera, provided there are enough landmarks in its field of view. We showed how IMU-based odometry and a visual SLAM algorithm
can be combined in a consistent framework to balance out their respective weaknesses.

With the developed system, we contribute a complete solution that facilitates the use of low-cost, robust and commercially available quad-rotor MAVs as a platform for future robotics research by enabling them to autonomously navigate in previously unknown environments using only onboard sensors.

8.1 Limitations

The closed-hardware nature of our test platform, the AR.Drone 2.0, required the system to be run on an external workstation instead of the MAV itself. The use of an external workstation in the loop over Wifi introduces unpredictable and often large delays (of up to 2 secs). With an open hardware platform the system can however be ported to run entirely on the MAV’s onboard computer, eliminating the need for an external workstation in the loop. The visual SLAM algorithm requires the scene to be highly textured and static. In a scene with little to no texture, for example, a white wall, the SLAM algorithm will fail to register any keypoints. Another fail case is a highly dynamic scene which violates the static scene assumption on which the SLAM system is built upon.

The use of an extended Kalman filter requires the noise distribution from all the sensors to be normally distributed. The assumption was not valid for our test case but was effectively approximated by assuming large covariances.

8.2 Future Work

There are a number of interesting research directions to augment and build upon the current system, which we will briefly discuss in this chapter.

Soft IMU constraints and Increasing Map Size

One of the major limitations of the current system is the size of the visual map. The time required for tracking a video frame grows linearly with the number of landmarks in the map, severely limiting the map size in practice. There are several interesting approaches to enable constant-time tracking. For example enforcing euclidean consistency of the map only in a local window, while globally the map is treated topologically as recently proposed by Strasdat et al. [cite Strasdat et al]. Essentially
this allows the map to grow arbitrarily without affecting the complexity of tracking a video frame or incorporating new keyframes.

An interesting topic for future research is how IMU and altimeter measurements - in particular with respect to the camera’s attitude - can be integrated into this pose-graph framework as additional soft constraints on and in between consecutive keyframes. This additional information can not only be used to eliminate three out seven drift dimensions (scale, as well as roll and pitch angle), but can also be used as additional constraint between poorly connected or unconnected parts of the visual map. Consider flying through an open door or window: only very few landmarks will be visible from both sides of the door, causing visual tracking to be inaccurate or even fail completely. IMU measurements can then be used to impose relative constraints between keyframes taken on both sides of the door, allowing to continue visual tracking within the same map.

Using Multiple Cameras

A further interesting research opportunity is given by the problem that when only one camera is used, visual tracking is bound to fail if the camera loses sight of the landmarks. This is particularly true for a low-resolution camera such as the one used for this thesis. An interesting topic for future research would be how incorporating multiple cameras can improve the tracking process by providing redundancy to e.g. a white wall in one direction.

Additionally, differently orientated cameras can provide strong redundancy to different types of motion: while tracking rotation around the optical axis of a camera is comparatively easy, large rotations around the x and y axis typically cause visual tracking to fail as the field of view changes rapidly, and for a visual SLAM algorithm using a single camera, new landmarks cannot be initialized due to the lack of sufficient parallax. Using three orthogonally orientated cameras, any type of motion can be tracked well by at least one of these cameras.

In this thesis we showed that visual tracking can be achieved using a camera with a resolution of only 640 x 480 - in terms of computational complexity tracking four such cameras rigidly attached to each other (i.e. undergoing the same motion) is equivalent to tracking one 1280 x 720 camera, as the tracking time only depends on the image size, the number of potentially visible landmarks and the number of landmarks found. Incorporating multiple low-resolution cameras looking at different scenes might hence produce more accurate results, without any added computational cost. For instance, most MAVs come equipped with a bottom facing camera for optical flow based position
estimation. The bottom facing camera can be used in both single and multiple camera configurations for the SLAM system. Due to the very low cost, weight and power consumption of such cameras, this is of particular interest in the context of MAV navigation.

Onboard Computation

The current system depends on a ground station for computationally complex tasks such as visual SLAM, which is mainly due to the very limited access to the on-board processing capabilities of the ARDrone 2.0. While this does allow for more computational power to be used, it is also the reason for some of the challenges encountered, in particular for the large delays due to wireless LAN communication.

A next step would be to make the MAV fly truly autonomously by performing all computations required on-board - the comparatively low computational power required by the current system suggests that a similar approach can well be performed on integrated hardware. An interesting research topic would be how the approach presented can be divided in two parts, the time-critical tracking, filtering and controlling part running on integrated hardware while the computationally expensive map optimization is done on a ground station PC.

Such an approach would efficiently combine the benefit of visual tracking on-board (little delay) and the computational power of a ground station for optimizing and maintaining a large map. Additionally, little bandwidth would be required as only sparse representations of keyframes and landmarks need to be communicated, instead of streaming each video frame. This could be extended as far as to allow the MAV to autonomously explore small new areas of the environment and then fly back in range of the ground station to drop new observations and keyframes, while receiving an optimized version of the global map.

Dense SLAM

An interesting and very promising research topic is the transition from sparse, keypoint-based SLAM to dense methods, either using depth-measuring cameras or by inferring the three-dimensional structure of the observed scene from camera motion alone. In particular an online generated, three-dimensional model of the environment greatly facilitates autonomous obstacle avoidance and recognition, path planning or identification of e.g. possible landing sites - after all the MAV is not supposed to fly into a white wall just because there are no keypoints on it. Furthermore, the capability
of easily generating accurate, three-dimensional models of large indoor environments is of interest in industrial applications - such a model can also be generated offline, after the MAV used to acquire the visual data has landed.

Due to the heavy dependency on GPU hardware of all dense SLAM methods, this would have to be combined with a hybrid approach, using a ground station for computationally complex tasks. The current development of GPUs for mobile phones however indicates that it will be possible to run such methods on integrated hardware in the near future.
Appendix A

Least Squares Minimization

A.1 Solution of Linear Equations

For a system of linear equations of the form $Ax = b$ where $A$ is an $m \times n$ matrix, the solution depends on the following criteria:

1. $m < n$: The system of linear equations is underdetermined (fewer equations than unknowns) and therefore does not have a unique solution. There is instead a vector space of solutions.

2. $m = n$: The system of linear equations has a single unique solution which can be obtained through the inversion of the square matrix $A$ as: $x = A^{-1}b$.

3. $m > n$: The system of linear equations is overdetermined (more equations than unknowns) and therefore generally does not have a solution unless some equations either appear more than once or are linear combinations of the others.

For the case of an overdetermined system of linear equations, we can instead try to find the vector $x^* \in \mathbb{R}^n$ that minimizes $\|Ax - b\|^2$:

$$x^* = \min_x \|Ax - b\|^2$$  \hspace{1cm} (A.1)

where $\|\cdot\|$ is the Euclidean norm. The solution $x^*$ is referred to as the least squares solution of the system of linear equations. Expanding Equation [A.1]:

$$\|Ax - b\|^2 = (Ax - b)^T (Ax - b) = x^T A^T Ax - x^T A^T b - b^T A x + b^T b$$  \hspace{1cm} (A.2)
It is easy to see that the derivative of \( \|Ax - b\|^2 \) with respect to \( x \) at a minimum should vanish. Setting the derivative to zero gives rise to the following normal equation which can be solved to get the solution \( x^* \):

\[
-A^T b + (A^T A)x^* = 0 \tag{A.3}
\]

\[
x^* = (A^T A)^{-1} A^T b \tag{A.4}
\]

**Homogeneous System of Linear Equations**

Similarly, the solution to a system of linear equations of the form \( Ax = 0 \), better known as homogeneous system of linear equations for the overdetermined case \( (m > n) \) can be formulated as a constrained least squares minimization problem:

\[
x^* = \min_x \|Ax\|^2 \quad \text{subject to:} \quad ||x|| = 1, \quad x \neq 0 \tag{A.5}
\]

The solution \( x^* \in \mathbb{R}^n \) lies in the null-space of \( m \times n \) matrix \( A \) and can be found using the singular value decomposition of \( A \) such that:

\[
A = UDV^T \tag{A.6}
\]

where \( U \) is an \( m \times m \) matrix of right singular vectors, \( D \) is an \( m \times n \) diagonal matrix of non-negative singular values and \( V \) is an \( n \times n \) matrix of left singular values. The solution \( x^* \) to the minimization problem is the right singular vector corresponding to the lowest singular value.

Alternatively, the solution can be found using eignevalue decompositon of the \( n \times n \) matrix \( A^T A \) as the eigenvector corresponding to the lowest eigenvalue.

**A.2 Solution of Nonlinear Equations**

Given a model, defined by functional relation \( \tilde{y} = f(x) \), where \( f : \mathbb{R}^d \rightarrow \mathbb{R}^m \) is a vector-valued nonlinear function, which takes as input the perimeter vector \( x \in \mathbb{R}^d \) and produces as output the measurement vector \( \tilde{y} \in \mathbb{R}^m \), we seek to find the solution vector \( x^* \in \mathbb{R}^d \) that minimizes the difference (residual) between a set of \( n \) hypothesized measurements \( \tilde{y}_i \in \mathbb{R}^m \) and true measurements \( y_i \in \mathbb{R}^m \).

Interpreted as a minimization problem:

\[
x^* = \arg\min_x \sum_{i=1}^{n} ||r_i(x)||^2 \tag{A.7}
\]
where
\[ r_i(x) = y_i - \hat{y}_i = y_i - f(x), \quad \forall i = 1, \cdots, n \]  \hspace{1cm} (A.8)

is the residual function and \( \rho(r_i) = \|r_i(x)\|^2 \) is the objective function. The minimization problem can be solved by noting that the derivative of the objective function at a minimum should be zero, giving rise to the optimality condition:
\[ r_i(x) \frac{\partial r_i(x)}{\partial x_j} = 0, \quad \forall j = 1, \cdots, d \]  \hspace{1cm} (A.9)

For convex objective functions with a single minimum, the solution of the optimization problem can be obtained in closed-form by solving Equation (A.9). For the class of non-convex objective functions with multiple minimums, the optimality condition can not be solved analytically in closed-form. The problem can instead be solved by starting at an initial estimate \( x_0 \in \mathbb{R}^d \) and iteratively stepping in the direction of negative gradient until a minimum is reached.

**Newton Method**

If we let \( x_t \in \mathbb{R}^d \) be the estimated solution after the \( t \)-th iteration. Then the second-order Taylor series approximation of the objective function around \( x_t \) (where \( \Delta x = x_{t+1} - x_t \)) is:
\[ \rho(r_i(x_t + \Delta x)) \approx \rho(r_i(x_t)) + g_t^T \Delta x + \frac{1}{2} \Delta x^T H_t \Delta x \]  \hspace{1cm} (A.10)

The derivative of the objective function with respect to the perimeter vector at a minimum should vanish, i.e.
\[ \frac{\rho(r_i(x_t + \Delta x)) - \rho(r_i(x_t))}{\Delta x} = 0 \]  \hspace{1cm} (A.11)

Substituting \( \rho(r_i(x_t + \Delta x)) \) from Equation (A.10) into Equation (A.11), we get:
\[ g_t + H_t \Delta x = 0 \]  \hspace{1cm} (A.12)
\[ \Delta x = -H_t^{-1} g_t \]  \hspace{1cm} (A.13)

Which leads to the estimate \( x_{t+1} \) for the next iteration:
\[ x_{t+1} = x_t - H_t^{-1} g_t \]  \hspace{1cm} (A.14)
where vector $g_t \in \mathbb{R}^d$ and matrix $H_t \in \mathbb{R}^{d \times d}$ are the respective gradient (vector of first-order partial derivatives) and Hessian (matrix of second-order partial derivatives) of the objective function at the current estimate $x_t$.

**Gauss-Newton Method**

The gradient vector $g_t$ and the Hessian matrix $H_t$ can be written in terms of partial derivatives of the vector-valued residual function $r_i(x_t)$ as:

\[
    g_t = 2 \sum_{i=1}^{n} r_i(x_t) \frac{\partial r_i(x_t)}{\partial x_j}, \quad \forall j = 1, \ldots, d \tag{A.15}
\]

\[
    H_t = 2 \sum_{i=1}^{n} \left( \frac{\partial r_i(x_t)}{\partial x_j} \frac{\partial r_i(x_t)}{\partial x_k} + r_i(x_t) \frac{\partial^2 r_i(x_t)}{\partial x_j \partial x_k} \right) \tag{A.16}
\]

where $J_t$ is the Jacobian (matrix of second-order partial derivatives) of the residual function $r$ at the current estimate $x_t$.

For large optimization problems, computation and inversion of the Hessian at each iteration is computationally infeasible. A suitable alternative are the Quasi-Newton methods which approximate the Hessian as a positive definite matrix. The Gauss-Newton method approximates the Hessian by dropping the second-order term in Equation (A.16), which leads to:

\[
    H_t = 2J_t^T J_t \tag{A.18}
\]

Together with $g_t = 2J_t^T r_i(x_t)$ and $H_t = 2J_t^T J_t$, the update step of the Newton method leads to the Gauss-Newton update step:

\[
    x_{t+1} = x_t - H_t^{-1} g_t \tag{A.19}
\]

\[
    x_{t+1} = x_t - 2(2J_t^T J_t)^{-1} J_t^T r_i(x_t) \tag{A.20}
\]

\[
    x_{t+1} = x_t - (J_t^T J_t)^{-1} J_t^T r_i(x_t) \tag{A.21}
\]

In contrast to the Newton method, the Gauss-Newton method does not require explicit computation of a Hessian matrix. The Hessian is instead approximated from the Jacobian. Furthermore, since the Hessian is positive definite by construction, Gauss-Newton method ensures convergence.
to a minimum instead of a saddle point.

**Levenberg-Marquardt Method**

Kenneth Levenberg [28] proposed the addition of a damping term $\lambda$ to the Hessian matrix for the Gauss-Newton method, such that:

$$x_{t+1} = x_t - (J_t^T J_t + \lambda I)^{-1} J_t^T r_t(x_t)$$  \hspace{1cm} (A.22)

The introduction of a damping term results in a hybrid between the Gauss-Newton method ($\lambda = 0$) and the gradient descent method ($\lambda = \infty$):

$$x_{t+1} = \begin{cases} 
  x_t - (J_t^T J_t)^{-1} J_t^T r_t(x_t), & \text{if } \lambda = 0 \\
  x_t - g(x_t), & \text{if } \lambda = \infty
\end{cases} \hspace{1cm} (A.23)$$

The value of the non-negative damping term $\lambda$ is updated at each iteration. When the current estimate is not close to the solution, $\lambda$ is raised, making the algorithm behave like gradient descent, leading to a faster approach towards the minimum. Conversely, when the current estimate is close to the minimum, the value of $\lambda$ is lowered, bringing the algorithm closer to the Gauss-Newton method.

Marquardt [35] further improved upon Levenberg’s method by introducing an adaptive damping scheme which replaces the identity matrix $I$ with a diagonal matrix consisting of the diagonal elements of the approximated Hessian matrix $H_t = J_t^T J_t$, such that:

$$x_{t+1} = x_t - (J_t^T J_t + \lambda \text{diag}(J_t^T J_t))^{-1} J_t^T r_t(x_t)$$  \hspace{1cm} (A.24)

Marquardt’s modification avoids slow convergence when the current estimate is close to the minimum by scaling the step size with the curvature of the residual function at the current estimate.
References


