Inertial-aided EKF-based Structure from Motion for Robust Real-time Augmented Reality

JOAKIM HUGMARK

Degree project in
Computer Science
Second cycle
Stockholm, Sweden 2013
Inertial-aided EKF-based Structure from Motion for Robust Real-time Augmented Reality

Master’s Thesis in Computer Science
Computer Vision and Active Perception Lab

JOAKIM HUGMARK (891109-0556)

Course Code: DD221X
Supervisors: Hedvig Kjellström & Fredrik Stridsman
Examiner: Danica Kragic

June 12, 2013
Abstract

The aim of this project was to develop a system that enables the overlay of computer graphics in a video sequence recorded by a moving camera. To do this, the camera's position relative to different landmarks in the picture needs to be estimated, a problem commonly referred to as “Structure from Motion” within the Computer Vision community, or “Monocular SLAM” within the robotics community. The system should be robust in the sense that it can handle partial occlusions and dynamic environments, as well as large open spaces without much texture in the image. To enable this, an IMU-unit is used to complement the visual input from the camera.

The system is based on previous work by Civera et al. and uses the Extended Kalman Filter (EKF) to fuse the sensor inputs. A 1-point RANSAC method is used to efficiently detect and discard outliers in the sensor data. Inverse depth parameterization is used to enable the use of landmarks far away from the camera. The IMU used in the project is developed by X-IO Technologies, and uses an on-board algorithm developed by Madgwick to determine a precise and drift-free orientation.

The system is evaluated using multiple video sequences recorded in a setting similar to where the system is intended to be used. Results indicate that the IMU really helps the system to differentiate ambiguities between translational and rotational movements, as well as keeping the system stable during smaller occlusions. Some cases where the system often fails are also identified. The performance of the system is evaluated, as well as how some different parameters involved impacts the results.
Referat

EKF-baserad Structure from Motion för robust förstärkt verklighet i realtid

Målet med detta projekt var att utveckla ett system som möjliggör placerandet av datorgrafik i en videosekvens inspelad av en kamera i rörelse. För att åstadkomma detta mäste kamerans position och rotation relativt landmärken i dess omgivande miljö uppskattas. Detta problem benämns ofta ”Structure from Motion” (inom datorseendeområdet) eller ”Monocular SLAM” (inom robotikområdet). Systemet ska även vara robust så tillvida att det kan hantera partiella ocklusioner och icke-statiska miljöer, liksom att operera i stora öppna områden vilket resulterar i att bilden kan innehålla väldigt lite texturer. För att kunna uppnå detta används en IMU-enhet som complement till den visuella information som kameran ger.

Systemet är baserat på tidigare forskning av framförallt Civera et al. och använder ett Extended Kalmanfilter (EKF) för att sammanfoga de olika sensorernas data. En 1-punkts RANSAC-metod används för att detektera och avfärdra outliers i sensordatan. Invers djupparameterisering används för att möjliggöra användandet av avlägsna landmärken. IMU:n som används är utvecklad av X-IO Technologies och använder en integrerad algoritm utvecklad av Madgwick för att beräkna en exakt och driftfri absolut orientering.

Contents

1 Introduction 1

2 Related work 5
   2.1 Monocular SLAM ........................................ 5
   2.2 Structure from Motion ................................ 6
   2.3 IMU Fusion .............................................. 7
   2.4 Fusion of Vision and IMU Data ......................... 8
   2.5 Other related work .................................... 9

3 Background theory 11
   3.1 Requirements ........................................... 11
   3.2 Quaternions ............................................. 12
   3.3 EKF-SLAM ............................................... 14
   3.4 Parametrization ........................................ 16
   3.5 Image features ......................................... 16
   3.6 Homographies .......................................... 17
   3.7 Data association .............................. 18
      3.7.1 Active search .................................... 19
      3.7.2 RANSAC ........................................... 20

4 Method 25
   4.1 The vision system ...................................... 25
      4.1.1 Camera calibration ............................... 25
      4.1.2 State vector ..................................... 26
      4.1.3 Kalman prediction ................................ 26
      4.1.4 Measurements ................................... 27
      4.1.5 Detecting and matching landmarks ................ 31
      4.1.6 Choosing corresponding matches ................. 31
      4.1.7 Kalman update ................................... 32
      4.1.8 Patch-warping ................................... 32
   4.2 The IMU-system .......................................... 32
      4.2.1 EKF parameters .................................. 32
   4.3 Fusion of IMU and vision data ......................... 34
Augmented Reality (AR) is used in various areas to enrich a video sequence with additional information. Recent applications include video games\(^1\) information about relative distances in football games\(^2\) and tracking of ball trajectories in golf and baseball (see Figure 1.1). Another emerging area that uses AR is visual tutorials, where the user is guided with symbols and text that is overlaid on a live video displayed to the user. With the recent explosion in the smartphone market, we will certainly see new applications, where the smartphone camera is used to create an AR-interface for the user. In all AR applications, it is of vital importance to create a convincing illusion that the overlaid information seems to be part of the real world.

In many of these applications, the camera can not be assumed to be static. In these non-static cases, it is of great importance to determine exactly where to place the overlaid objects in the image for each and every video frame. This can be achieved either by tracking the 3D position where the object is supposed to be placed, and project it onto the image, or by tracking the motion and pose of the camera itself. It is crucial that this tracking-process can run in roughly the same frequency as the frame-rate of the camera, so that the AR-interface does not seem to “lag” behind.

This second option, recovering the pose of a monocular camera from a video sequence, is a problem that has been actively researched in two different fields, originally using slightly different approaches. Both approaches either need to know a map of the environment beforehand, or tries to estimate it simultaneously. Within Computer Vision, and especially Multiple View Geometry, research in this area has been done under the term “Structure from Motion” (SfM). SfM refers to determining the structure of an area or object that is represented by a image or video sequence, as well as the position of the camera within the sequence. Traditionally, the whole sequence is assumed to be known beforehand, which has lead to global optimization techniques such as bundle adjustment becoming the gold-standard.\(^1\) 

\(^2\)http://www.tracab.com/products.aspx
CHAPTER 1. INTRODUCTION

Figure 1.1: Example of Augmented Reality used for displaying the tracked trajectory of a golf ball.

Within the robotics community, the Simultaneous Localization and Mapping (SLAM) problem refers to determining the pose of a mobile robot, as well as a map of what the robot has “seen”. The last decade’s increase in available computing power, also on small mobile platforms, means that the camera has started to become one of the most widely used sensors for robot navigation. Navigating using a single camera is sometimes referred to as Monocular SLAM, and concerns pretty much the same problem as SfM: recovering robot (camera) pose, as well as the map (structure of environment) from an image sequence. The main difference is that the mobile robot get new data sequentially, and immediate knowledge of the whole sequence can not be assumed. Instead, filtering techniques such as the Extended Kalman Filter (EKF) has become the standard approach, with recent advances in using a combination of sequential tracking, while optimizing the map at certain key-frames in a parallel thread. [11]

One inherent problem with using a single camera to determine the pose, is that it can only be determined up to a scale factor, without prior knowledge. This means that the relative positions of the landmarks in the map, as well as the relative motion of the camera can be determined, but not their exact metric values. Another difficulty is to differentiate between rotations and translations. While a rotation of the camera results in a different transformation of the image than a translation does, the differences can sometimes be very subtle, depending on the environment. Yet another problem is that non-restrictive motion models for the camera movement often can be misused to explain noise in the video sequence as (non-existent) motions of the camera.
Another problem inherent to any approach based on exteroceptive sensors such as a camera is that the scene may change during observation. In the case of Computer Vision, persons may move about in the scene, occluding static landmarks, while tricking the system into trying to track moving landmarks. In a worst-case scenario, the whole scene might be occluded for short periods of time, making visual tracking impossible. Problems may also arise during fast and jerky motions, where motion blur may cause the landmark tracking and matching to fail.

One way to overcome these problems is to use an additional, proprioceptive sensor. In this report, different methods of fusing the data from an Inertial Measurement Unit (IMU) with the data provided by a state-of-the-art vision system will be evaluated and tested. The IMU may both provide measurements with an absolute scale (the absolute acceleration of the camera, not used in this report), as well as a way to decouple rotational movements from translational ones. Furthermore, since it is proprioceptive, it will not suffer from occlusions or changes in the environment. This makes it perfect to combine with a vision-based system.

In this report, the tests will concentrate on how the methods for fusing IMU with vision perform in an especially difficult case of an arbitrary movement on a golf course. This means that the high-parallax-points close to the camera will have very little texture, making them harder to track. Most trackable landmarks will instead be very far away and show little to no parallax (see Figure 1.2).

![Figure 1.2: Example showing the nature of the environment where the algorithm described in this report is supposed to be used. Notice how most of the noticeable landmarks are in the far-away parts of the image, which means that they will show little to no parallax when the camera moves.](image)

First, the individual sensors and their corresponding filtering-schemes’ perfor-
mance are measured on synthetic video sequences consisting of simple camera-movements. Analysing the results gives some knowledge about the strengths and weaknesses of the two sensors, and provides some guidance for how to fuse them together.

Then, different methods for fusing the two sensors’ data are evaluated, both on the already mentioned synthetic sequences, but also on more complex sequences consisting of more general movement.

The rest of this report is organized as follows. In Chapter 2, similar work and the current state-of-the-art within the field is presented, as well as references for some of the concepts that will be widely used in this report. In Chapter 3, the background theory needed for the method is presented. This allows the different parts of the system to be presented and discussed in a more isolated and orderly manner. Chapter 4 provides more implementation details about the system described in the report, and presents equations and models used specifically for this system. It also presents the experimental setup, to allow for easy replication. The experiments themselves, as well as the results, are presented in Chapter 5. Finally, a summary and discussion of further work can be found in Chapter 6.
Chapter 2

Related work

As mentioned, in Chapter 1, the problem of reconstructing camera positions from a sequence of monocular images has been approached in two different fields, with slightly different methods.

In the robotics community, the methods are usually based on a filtering-scheme, such as the Kalman filter. This has become such a popular method, that there are surveys considering the use of Kalman filter for robot vision. Using filter methods to obtain the pose of a camera is sometimes referred to as Monocular SLAM.

2.1 Monocular SLAM

Real-time Monocular SLAM was first achieved by Davison [8], [9]. In contrast to previous work (which operated on batches of images), Davison concentrates on the case where the computations has do be done sequentially in real-time, which is the case for many robotics applications. He formulates the problem of camera pose estimation as a state estimation problem, where the camera is described by a state-space model. This requires the camera to be calibrated, since it operates in a metric Euclidean space. Formulating a motion model for the camera, he proceeds with an Extended Kalman Filter (EKF)-based approach, with careful choices of the noise matrices etc. to motivate smooth trajectories. The novelty in the approach lies in how the features used (image-patches) have their depth initialized. He assumes a uniform 1D distribution for the position of the features, a line from the camera center, through their projections in the image, extending toward infinity. He then uses a Particle Filter, and iteratively evaluates the possible depth-hypotheses (particles). The locations of these features are then used to determine the camera pose via the EKF.

Two of the main problems with monocular SLAM, in particular for augmenting live TV, is initialization of landmarks to track and low-parallax environments. Both of these problems are addressed in [5]. There, a new parametrization for the position of the landmarks in the state vector is introduced, which allows infinite depth, and thus infinite depth uncertainty. Using this parametrization, new features are added
with a probability density function over the depth that allows them to be infinitely far away, thus being used only for bearing estimation. When a parallax is detected, due to camera movement, this pdf will be updated to reflect the finite depth of this point. Results show that this new parametrization both allows for far-away features to be used, as well as immediate initialization. Which means that the system can be initialized without the help of fiducials or other objects with known forms.

A further improvement of monocular SLAM is presented in [6]. Civera et al show that a camera-centred EKF-SLAM approach can be used to map longer trajectories with accuracy approaching that of using visual odometry with bundle adjustment. They also propose a RANSAC-based outlier-detector, that requires just 1 point to form an hypothesis. This is achieved using the first half of the EKF’s update-step, and the resulting innovation to estimate support for the RANSAC hypotheses. Centering the coordinate frame to the sensor results in lower covariances and better linearisation. This will be the basis of the work in this report.

2.2 Structure from Motion

In the Computer Vision community, the term Structure from Motion (SfM) is used to describe the process of recovering the structure of the environment, as well as the motion of the camera from a sequence of images, or a video. Most approaches assume knowledge of the whole image sequence, and then performs global optimization over it to compute the optimal camera positions [10]. While the goal is pretty much the same as with monocular SLAM, the means by which it is achieved has traditionally been slightly different.

Recently however, the methods of the two fields have approached each other, and in [11] an algorithm called Parallel Tracking and Mapping (PTAM) is presented. It combines a sequential tracking process with another, parallel, map-optimization process, to refine the map based on semi-global optimization over certain key-frames in the image sequence. This provides previously unparalleled performance for AR-applications in smaller environments (see Figure 2.1).

And in [14], the Parallel Tracking and Mapping algorithm presented in [11] is extended with the help of an IMU that is used to estimate the unknown scale factor. They try two different methods, one based on spline-fitting, and the other on an EKF. Both methods are evaluated offline, where EKF outperforms the spline-fitting method. The EKF is then implemented in an online-fashion within the PTAM framework (as two parallel threads) and performs quite good on both simulated and real data. Other interesting parts are in their motion model, where they incorporate an acceleration part, and in the multi rate Kalman filter, which is used every time either vision or IMU data is available.
2.3 IMU Fusion

Regarding the inertial measurement unit, different approaches to fuse the data from the accelerometers, gyroscopes and magnetometers to calculate an absolute orientation have been suggested. Quite recently, a new, non-EKF-based approach was suggested in [12]. By representing the orientation as quaternions, the filtering problem is turned into an optimization problem, which is solved using a gradient descent method, based on an analytically derived Jacobian. It also compensates for error and drift in the gyroscopes. The proposed filter is compared to a commercially available Kalman-based filter, and turns out to be slightly better, while requiring significantly less computational power. An RMS-error of about $0.6^\circ$ for pitch and roll, and about $1^\circ$ for yaw, is achieved. Only one (IMU-version) or two (MARG-version) parameters need to be tuned, and instructions for how to determine these optimally are presented.
2.4 Fusion of Vision and IMU Data

A couple of different methods to fuse vision and IMU-data to obtain a refined camera pose have emerged the last decade. Most, almost all, of them are based on filtering schemes, such as Kalman filters or Particle filters.

Armesto et al. [2] propose a method to fuse sensor input from an IMU as well as from a vision system using a multi-rate EKF and a multi-rate UKF. They evaluate the results using the two methods, and while the performances are similar, the EKF is considered better, since it is faster. The UKF also shows some numerical instability when used to calculate the rotational movements. They also conclude that using a fusion of the camera and the IMU is better than using any one of them alone. The vision sensor is better for slow movements, while the IMU is better for fast movements.

In [15] an inertial sensor is used in conjunction with a magnetometer to determine the pose of a person. They develop a sensor fusion algorithm based on the Kalman filter, which is evaluated and compared to a commercial filter from Vicon. The magnetometer works at a very slow frequency (1.67 Hz), compared to other work.

Torr and Zisserman [20] present an overview of methods for estimation of camera motion and structure. The report advocates the use of feature-based approaches, rather than direct approaches, where you work with all pixels in the image directly. It provides good instructions for how to compute homographies, as well as the fundamental matrix between different camera views. The method does however require the whole sequence to be available from the start, in order to perform bundle adjustment.

In [18], a method for tracking the pose of a camera for augmenting football games is presented. It uses a spatial-aware variant of the Hough transform to detect lines on the field, and uses the positions of the lines to track the motion. It depends on an initial estimation of the pose, which is determined with the help of multiple images, and then uses the assumption of a fixed mounted camera to further simplify the process. It also relies on the fact that the lines on the field form a known geometry in 3D space, which means that the structure of the environment in some sense is already known, and the problem is reduced to one of only localization or geometric reconstruction.

Ababsa and Mallem [1] propose a camera pose estimation method based on fusion of IMU and vision data. The Computer Vision-part is based on pose estimation from fiducials, where a corner detector is used to detect and identify fiducials, which provide an initial scale of the problem. Then they construct a motion model of the camera, that relates the camera’s position, orientation, angular velocity, speed and acceleration in the different coordinate frames. The data from the IMU is then directly used in this motion model, together with the output from the vision tracker. A SIR particle filter is suggested as the fusion algorithm, and an overview of how it should be implemented is given. The method has been evaluated by letting the sensor platform move in pre-determined patterns with a high precision robot arm,
and RMSE-values of the estimated trajectory are calculated. The results show that the method is better than previously suggested methods, while being a bit faster, and suitable for real-time use.

2.5 Other related work

A discussion on whether to use filtering or bundle adjustment for monocular SLAM is given in [17]. The difference between the two methods are presented, as well as an experimental evaluation of their performance. Bundle adjustment of key-frames is considered superior in terms of accuracy per CPU cycle. On the other hand, concentrating on key-frames, and performing bundle adjustment on only a part of the frames will probably work poorly for AR-purposes, since the estimation of the camera’s position will be vital for every frame, in order to place graphics in the environment in a visually appealing way.

A more detailed description of the theoretical foundations of the Kalman filter, as well as the Extended Kalman Filter, can be found in [21] and [19].
Chapter 3

Background theory

This section presents the theory of the concepts that will be used in implementing the SfM-system that this report describes. A basic knowledge of Computer Vision concepts such as features, corners and epipolar geometry is assumed, as well as basic knowledge of the Kalman filter.

3.1 Requirements

The high-level requirements of the system is that it should be able to track the movement of a camera by analysing the output image sequence, as well as the data from a rigidly attached IMU. The output of the system should be the extrinsic parameters of the camera along with the image coordinates of a specific point specified by the user in the first image frame. This allows for an object initially placed at the tracked point to be correctly placed in all the consecutive frames of the video sequence. See figure 3.1.

Figure 3.1: The concept of SfM: when the camera moves, the structure of the environment is reconstructed, together with the camera movement, based on measurements from the image sequence.
CHAPTER 3. BACKGROUND THEORY

There are a couple of special requirements on the SfM-system this report describes that puts it apart from many of the other, already existing camera pose estimation systems. First of all, since it is to be used for AR, sequential, real-time computation of the pose is a requirement. The camera pose must be reconstructed on-the-fly, without access to the whole sequence, and should be feasible to implement without any lag.

Furthermore, the environment will mainly consist of large, open spaces, where the most notable corner points will give little or no parallax. The foreground will mainly consist of low-texture grass and moving people. This will require the system to be able to track even non-distinct features, while quickly discarding more distinct features originating from moving objects, or people. See Figure 1.2.

On the positive side, the video sequences will normally be quite short, less than a minute, which makes long-term drift, otherwise a huge problem in SLAM, less of an issue.

3.2 Quaternions

Quaternions are an extension of the complex numbers, formed by four components as \( q = (q_0, q_x, q_y, q_z) : q_0 + q_x i + q_y j + q_z k \). While they have numerous interesting algebraic properties, the focus of this section is to explain how they can be used to represent rotations in 3-space. Let's begin with why we need quaternions at all.

The more intuitive way of representing rotations in 3-space is Euler angles. By using three separate angles, representing rotations about the three axes (yaw, pitch and roll), a general rotation might decomposed of three, easily visualizable rotations about a single axis. The main problem with this representation is the so-called “gimbal lock”. The gimbal lock occurs when the pitch changes to \( 90^\circ \), then the roll and yaw rotation axes coincide, which results in the rotated object only being able to rotate about two of the axes (see Figure 3.2).

The gimbal lock causes the Euler angles to have a singularity whenever the pitch-angle approaches \( \pm 90^\circ \), which will cause problems with the system described in this report.

Quaternions, on the other hand, does not suffer from any such singular orientations. Other advantages include:

- Subsequent rotations can be formed with simple multiplication.
- Faster to normalize than rotation matrices.
- Nice, continuous representation of rotations allows for easy interpolation between orientations.

\[ \textit{http://en.wikipedia.org/wiki/File:Gimbal_lock.png} \]
3.2. QUATERNIONS

Figure 3.2: Example of gimbal lock: yaw and roll coincide, and the plane can only rotate about two axes. Image courtesy of Wikimedia Commons.

Quaternions can be converted to Euler angles via:

\[
\psi = \text{atan2}(2q_xq_y - 2q_0q_z, 2q_0^2 + 2q_x^2 - 1) \quad (3.1)
\]
\[
\theta = -\text{asin}(2q_xq_z + 2q_0q_y) \quad (3.2)
\]
\[
\phi = \text{atan2}(2q_yq_z - 2q_0q_x, 2q_0^2 + 2q_z^2 - 1) \quad (3.3)
\]

The reverse transformation is:

\[
q = \begin{pmatrix}
\cos(\phi/2) \cos(\theta/2) \cos(\psi/2) + \sin(\phi/2) \sin(\theta/2) \sin(\psi/2) \\
\sin(\phi/2) \cos(\theta/2) \cos(\psi/2) - \cos(\phi/2) \sin(\theta/2) \sin(\psi/2) \\
\cos(\phi/2) \sin(\theta/2) \cos(\psi/2) + \sin(\phi/2) \cos(\theta/2) \sin(\psi/2) \\
\cos(\phi/2) \sin(\theta/2) \sin(\psi/2) - \sin(\phi/2) \cos(\theta/2) \cos(\psi/2)
\end{pmatrix} \quad (3.4)
\]

A rotation \( q^1 \) followed by another rotation \( q^2 \) is formed by the quaternion multiplication \( q^1 \otimes q^2 \), defined as:

\[
q^1 \otimes q^2 = \begin{pmatrix}
q_{0}^{1}q_{0}^{2} - q_{x}^{1}q_{x}^{2} - q_{y}^{1}q_{y}^{2} - q_{z}^{1}q_{z}^{2} \\
q_{0}^{1}q_{y}^{2} + q_{x}^{1}q_{z}^{2} - q_{y}^{1}q_{z}^{1} + q_{x}^{1}q_{y}^{1} \\
q_{0}^{1}q_{x}^{2} + q_{y}^{1}q_{y}^{2} - q_{x}^{1}q_{x}^{1} + q_{y}^{1}q_{z}^{1} \\
q_{0}^{1}q_{z}^{2} - q_{y}^{1}q_{x}^{2} + q_{y}^{1}q_{y}^{1} + q_{z}^{1}q_{z}^{1}
\end{pmatrix} \quad (3.5)
\]
3.3 EKF-SLAM

In order to estimate the motion path of a camera (or robot) based on its sensor-input, the structure of the scene must generally be known. If both the scene and the motion path are unknown, the solution is to recover both at once, a coupled chicken-and-egg-problem often referred to as Simultaneous Localization and Mapping (or SLAM) in the robotics community. In this report, only the motion path of the camera is of interest, but a map of the structure of the environment is still needed in order to solve the problem, which means that the whole SLAM problem needs to be solved. One of the most widely used SLAM algorithms is based on the Extended Kalman Filter, and is thus called EKF-SLAM.

The Extended Kalman Filter is a non-linear version of the normal Kalman Filter. A Kalman Filter is used to filter noisy measurements and combine them to estimate some (optimal) state based on these measurements.

The main difference between EKF and EKF-SLAM when estimating the motion of a camera, is that the system state in the EKF only consist of the pose of the camera (in this case position $r$, a rotation quaternion $q$, velocity $v$ and angular velocity $\omega$):

$$
\hat{x}_E = \begin{pmatrix}
    r_c \\
    q_c \\
    v_c \\
    \omega_c
\end{pmatrix},
$$

while the state in EKF-SLAM consist of estimates of both the pose of the camera, but also of landmarks $y_i$ in the map:

$$
\hat{x}_{ES} = \begin{pmatrix}
    \hat{x}_E \\
    y_1 \\
    \vdots \\
    y_N
\end{pmatrix},
$$

Similarly, the state covariance $P$ in EKF-SLAM not only contains the covariances of the camera-pose-variables, but also of the landmark-coordinates.

Just as in a normal EKF, a motion model $f$, which is used to predict the motion of the system over small time steps, is needed.

$$
\hat{x}_{k|k-1} = f(\hat{x}_{k-1})
$$

In the case of monocular SLAM, a common motion model is the constant velocity motion model, which assumes a constant velocity and angular velocity over each time step. A constant velocity model is a good way to describe a smooth general motion, without jerkiness. This fits well with the movement of a hand-held camera.

In order to predict the state covariance, the Jacobian $F$ of the motion model w.r.t. the state parameters is needed (the landmarks are not predicted in this step,
3.3. EKF-SLAM

so their covariance is not updated either):

\[ P_{k|k-1} = F_{k-1} P_{k-1} F_{k-1}^T + Q_{k-1}, \]  

(3.9)

where \( Q_k \) is the model noise.

Next, a measurement model \( h \), which relates the measured values \( z_i \) to the landmarks in the system state, is needed:

\[ z_i = h(x_i) \]  

(3.11)

as well as the Jacobian \( H \) of this function w.r.t. the state parameters:

\[ H_k = \left( \frac{\partial h}{\partial r_k}, \frac{\partial h}{\partial q_k}, \frac{\partial h}{\partial v_k}, \frac{\partial h}{\partial \omega_k}, \ldots, \frac{\partial h}{\partial y_1}, \ldots, \frac{\partial h}{\partial y_N} \right)^T \]  

(3.12)

These are used in the update step of the filter, which, again, works in a similar way as in a normal EKF. The update step begins by computing the “innovation” of the state and covariance:

\[ \tilde{y}_k = z_k - h(\hat{x}_{k|k-1}) \]  

(3.13)

\[ S_k = H_k P_{k|k-1} H_k^T + R_k \]  

(3.14)

where \( z_k \) are the measurements at time step \( k \) and \( R_k \) is the measurement noise.

Then the Kalman gain \( K \) is computed:

\[ K_k = P_{k|k-1} H_k^T S_k^{-1} \]  

(3.15)

and is used to update the state according to the measurements:

\[ \hat{x}_{k|k} = \hat{x}_{k|k-1} + K \tilde{y}_k \]  

(3.16)

\[ P_{k|k} = (I - K_k H_k) P_{k|k-1} \]  

(3.17)

An approximative, but faster, way of computing the covariance is:

\[ \tilde{P}_{k|k} = P_{k|k-1} - K S K^T \]  

(3.18)

\[ P_{k|k} = 0.5 \tilde{P}_{k|k} + 0.5 \tilde{P}_{k|k}^T \]  

(3.19)

For more details about how the motion model and measurement model are defined, as well as how the Jacobians are computed, refer to Section 4.1. More information, and the theory behind the Extended Kalman Filter can be found in [21] and [19].
3.4 Parametrization

In order to keep track of the landmarks, a “map” with their positions are stored in the state vector of the EKF. It has been shown \(^3\) that representing the positions of the landmarks relative to the camera results in a lower linearization error during the EKF process. This way of representing the map is sometimes referred to as “camera-centric”, meaning that the position of the camera is the origin of the map. In order to restore the absolute position of a landmark, the position of the camera when that landmark was initialized also needs to be stored.

Intuitively, representing the positions of the landmarks in Cartesian space may seem well enough, but it turns out that using Cartesian coordinates causes problems when new features are initialized. When a landmark is measured for the first time, only its direction can be estimated with any certainty; the distance from the camera to the landmark is unknown, since no parallax can be measured from a single image frame. Using Cartesian coordinates to represent this infinite uncertainty with regard to the distance between the camera and the landmark, is impossible. Instead, many systems have waited until the camera have translated as far as needed before a parallax can be measured. Only after that, when an estimate of the distance to the landmark can be computed, can the feature be used. This makes the system unable to use any landmarks far away from the camera.

Using the so-called “inverse depth parameterization”, first introduced by Civera et al. \(^5\), the features can be initialized with an infinite uncertainty for the depth-coordinate, transforming to a finite depth uncertainty naturally when a parallax is observed. This allows for using very far-away features to estimate the bearing of the camera, as well as avoiding the need to wait for features to initialize. It also removes the necessity of having some landmarks with known 3D positions, present in many of the previous works, for example in \(^9\).

The inverse depth parameterization consists of the camera’s position when the feature was initialized: \((x_i, y_i, z_i)\); the azimuth and elevation angles: \((\theta_i, \phi_i)\); and the inverse distance to the point: \(\rho_i = \frac{1}{d_i}\) (see Figure 3.3). The conversion from inverse depth parameterization to Cartesian coordinates becomes:

\[
\begin{pmatrix}
X_i \\
Y_i \\
Z_i
\end{pmatrix} = \begin{pmatrix}
x_i \\
y_i \\
z_i
\end{pmatrix} + \frac{1}{\rho_i} \begin{pmatrix}
\cos \phi_i \sin \theta_i \\
-\sin \phi_i \\
\cos \phi_i \cos \theta_i
\end{pmatrix}
\] (3.20)

3.5 Image features

The measurements of landmarks provided by the camera is in the form of interest points in the images. The idea is to detect notable high-contrast points in the first image, and then try to track how they move throughout the sequence, while adding more points if necessary.
3.6 Homographies

A homography is a transformation from a projective space to itself, and is in the domain of Computer Vision often used to warp images of the same environment taken from different angles, to make them fit together. A very common application is in stitching together multiple images into a panorama (see Figure 3.5).

In this case, however, the application is another: homographies will be used in order to predict how the tracked landmarks (image-patches) will look in the new frame. This means that instead of determining the rotation and translation from the homography (which would be the case when creating a panorama), the translation and rotation is already known, and will be used in order to determine the homography.

Two cameras looking at a plane \( n \), have the poses defined by the rotation matrices \( R_a, R_b \) and translation vectors \( t_a, t_b \). The homography is then given by:

\[
H_{ba} = R_a^{-1} R_b - \frac{(t_b - t_a)n^T}{d}
\]  

(3.21)
where $d$ is the distance from the camera to the plane from camera $b$. See figure 3.6. More information about how the normal $n$ is estimated can be found in Section 4.1.8.

### 3.7 Data association

In order to track the landmarks, a mere feature detector is not enough. Some kind of scheme that matches different point sets, and thus determines the correspondence, is needed. It turns out, however, that it is infeasible to construct a feature matcher that is not prone to a large amount of deviations, or outliers. A combination of a relatively robust matching approach, such as active search, where the EKF model’s prediction is used to determine an area where to search for corresponding points, and outlier-rejection algorithm, such as RANSAC, is usually enough.
3.7. DATA ASSOCIATION

Figure 3.5: Example of using homographies to warp pictures when stitching them together to create a panorama.

Figure 3.6: Image of camera $b$ looking at the plane from a distance $d$. Image courtesy of Wikimedia Commons. [Homography-transl-bold.svg](http://en.wikipedia.org/wiki/File:Homography-transl-bold.svg)

3.7.1 Active search

First suggested in [8], the active search method works by using the prediction of the features positions in 3D-space relative the camera, $y_{iC}^C$, together with the pinhole camera model, to predict the positions of the features in the current image frame,
An area around this predicted point is then searched for a matching landmark. The size of the area is determined by the uncertainty of the prediction, $\lambda$, which are the eigenvalues of the covariance matrix $S_i$ for landmark $i$:

$$\lambda_i = \text{eig}(S)$$  \hspace{1cm} (3.23)

$$S_i = H_i P_{\hat{e}_{k-1}} H_i^T + R_i$$  \hspace{1cm} (3.24)

where $H_i$ is landmark $i$’s part of the Jacobian of the measurement model $[3.12]$ and $R_i$ is the measurement noise for landmark $i$.

### 3.7.2 RANSAC

As mentioned in Section 3.7, just a matching method is not enough. An algorithm to find and reject outliers from the data is necessary to get a robust set of corresponding points.

A very powerful and popular method for detecting outliers is RANdom SAmple Consensus, RANSAC. RANSAC is a statistical method that tries to determine a set of inlier data points that all agree to some extent. The general assumption made by RANSAC is that the data consists of inliers - points generated by a specific model, subject to noise, and outliers - points that does not fit with the model. The algorithm then works in the following way $[10]$:

1. Choose the smallest possible random sample from the observed data that allows for the construction of an hypothesis for the model.
2. Construct an hypothesis model based on this sample.
3. Fit all the data to the hypothesis model.
4. All data points that deviate less than a certain threshold from the hypothesis model are considered “in support” of the model.
5. If the support is good enough, continue to the next step. Otherwise, try another random sample.
6. When an hypothesis model with good enough support is found, re-estimate the model with all data points that are in support of the model.
7. Re-evaluate the new model on all the data. Collect all points in support for this new model as inliers.

\[\text{Note that compensation for distortion also is performed in this step, see Section 4.1.4}\]
3.7. DATA ASSOCIATION

(a) Input data to RANSAC; some are part of the model and some are outliers.
(b) Output: Line fitted with RANSAC. No outliers are used.

Figure 3.7: Input and output from the RANSAC algorithm. Images from Wikimedia Commons.

See Listing 3.1 for pseudocode for RANSAC outlier rejection. Figure 3.7 illustrates the basics of the RANSAC algorithm applied to fitting a straight line to a (noisy) dataset of points.
CHAPTER 3. BACKGROUND THEORY

Listing 3.1: Pseudocode for RANSAC outlier rejection.

```python
max_support = 0
max_hypotheses = ∞
i = 0
while (i < max_hypotheses):
    sample = random_sample(dataset)
    hyp = create_hypothesis(sample) #See next page(s)
    support = 0
    for point in dataset:
        prediction = predict(hyp, point) #See next page(s)
        if (prediction - point < threshold):
            support++
    if (support > max_support):
        max_support = support
        best_hyp = hyp
        ϵ = 1 - support / dataset.size()
        max_hypotheses = log(1 - p) / log(1 - (1 - ϵ)m))
    i++

j = 0
for point in dataset:
    prediction = predict(hyp, point) #See next page(s)
    if (prediction - point < threshold):
        inlier(j) = true
    else:
        inlier(j) = false
j++
```

1-point RANSAC

One of the main problems with RANSAC is the complexity. The required number of hypotheses to test in order to find at least one without outliers with a probability \( p \), is given by:

\[
    n = \frac{\log(1 - p)}{\log(1 - (1 - \epsilon)^m)}
\]

where \( \epsilon \) is the ratio of outliers and \( m \) is the number of data points required to formulate a hypothesis [10]. In this case, five corresponding points are needed in order to recover the pose change of a camera [13]. The number of hypotheses needed for different outlier ratios, probabilities and required data points can be found in Table 3.1.

From the table, it is clear that it would be a huge advantage to be able to reduce the number of points needed to form an hypothesis. Recent work by Civera et al. [6] presents a method, by which the number of data points needed in order to form an hypothesis is reduced to 1, thus only needing 6 hypotheses in order to have at
3.7. DATA ASSOCIATION

<table>
<thead>
<tr>
<th>m</th>
<th>p</th>
<th>ε</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.2</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.3</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.4</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.5</td>
<td>146</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.6</td>
<td>448</td>
</tr>
<tr>
<td>1</td>
<td>0.99</td>
<td>0.1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0.99</td>
<td>0.2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0.99</td>
<td>0.3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0.99</td>
<td>0.4</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>0.99</td>
<td>0.5</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0.99</td>
<td>0.6</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.1: Number of RANSAC hypotheses $n$ that needs to be tested in order to achieve at least one outlier-free with a probability $p$, for different number of points needed to form an hypothesis $m$ and different outlier ratios $\epsilon$.

least one free from outliers with 99% certainty if the outlier ratio is 40%, as opposed to 57, if 5 points are needed for form an hypothesis.

The method works by using the a priori probability distribution for model parameters present in an Extended Kalman Filter to form an hypothesis with only one point. The hypothesis is formed by performing a partial EKF-update, where only the state (not the covariance) is updated (see Listing 3.2). This updated state is then used to predict the positions of all the other corresponding points (see Listing 3.3). These predictions are compared with the actual positions of the matches: all that lie within a specified threshold (the RANSAC “low-innovation”-threshold), are considered in support for the hypothesis.

This is repeated with more hypotheses until there is a 99% probability that at least one of the hypotheses tested was free of outliers. The most supported hypothesis is then selected, and an EKF update based on all the points considered inliers by this hypothesis is performed.

```python
def create_hypothesis(matching_points):
    z = matching_points.z  # Matched position
    z = matching_points.h  # Predicted position
    H = matching_points.H
    S = H*P_e[k-1]*H.T + matching_points.R
    K = P_e[k-1]*H*S
    x = x_e[k-1] + K*(z - x)
    return new_hyp(x)
```

Listing 3.2: Pseudocode for hypothesis creation for 1-point RANSAC.
def predict(hyp, point):
    x = hyp.x
    h = calculate_h(x, point)  # See eq. 4.11 - 4.16
    return h

Listing 3.3: Pseudocode for prediction of a point with an hypothesis for 1-point RANSAC.

The next step is to rescue potentially high-innovation inliers that were missed by the RANSAC algorithm. To do this, we begin by performing a Kalman update based on the inliers found by RANSAC, to obtain a new state $x_{k|k-1}^{li}$ and covariance matrix $P_{k|k-1}^{li}$. Then all outliers are again predicted by this new state, and gets a predicted position $h^{hi}$ and covariance $H^{hi}$. Then their correspondence with the current model of the state is evaluated using the squared Mahalanobis distance:

$$v = \nu_i^T S^{-1} \nu_i$$  \hspace{1cm} (3.26)

where $\nu_i = z_i - h^{hi}_i$  \hspace{1cm} (3.27)

and $S = H^{hi} P_{k|k-1}^{li} H^{hiT}$  \hspace{1cm} (3.28)

If the distance $v$ is below a threshold, the point is considered as a “high-innovation” inlier. For more information, refer to [6].
Chapter 4

Method

This chapter describes the implementation in more detail, and describes how the concepts described in Chapter 3 were combined into a working system.

The system is based on previous work by Civera et al. [6], and the corresponding 1-point RANSAC framework for monocular SLAM written in MATLAB. This system has been reimplemented in C++ with OpenCV, and some parts have been slightly changed, which is described in the following sections. Support for IMU input has also been added.

The system basically uses two different sensors for estimation of the camera pose: a monocular camera and an IMU. While the implementation integrates these two sensors quite closely, their implementations will be explained separately in different sections.

4.1 The vision system

The vision system is the system handling the monocular image data, and is more complex than the IMU system. It will here be described by its impact on the different components of the EKF.

4.1.1 Camera calibration

In order for the system to operate optimally, the camera used needs to be calibrated. During the experiments, the camera has been calibrated using the camera calibration module bundled with OpenCV 2.4.4, using a check-board pattern and 5 distortion parameters. See Figure 4.1. The distortion model in OpenCV is based on Bouguet’s model, which is different from the model used in the original implementation of 1-point RANSAC by Civera et al. [6].

\[http://www.vision.caltech.edu/bouguetj/calib_doc/\]
4.1.2 State vector

The state vector consist of the camera pose, expressed in the Cartesian position $\mathbf{r}$ of the camera along with a rotation quaternion $\mathbf{q}$, together with the velocity $\mathbf{v}$ and the rotational velocity $\mathbf{\omega}$:

$$\dot{\mathbf{x}}_s = \begin{pmatrix} \mathbf{r}, \mathbf{q}, \mathbf{v}, \mathbf{\omega} \end{pmatrix}^\top$$ (4.1)

Since EKF-SLAM is used, the map is also stored in the state. This means that the positions of all the landmarks are stored in the state vector, in their inverse-depth coordinates (See Section 3.4):

$$\mathbf{y}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \\ \phi_i \\ \theta_i \\ \rho_i \end{pmatrix}$$ (4.2)

4.1.3 Kalman prediction

A constant velocity model is used to describe the camera movement. This model is used to predict the next state of the system:
4.1. THE VISION SYSTEM

\[
\dot{x}_{k|k-1} = f(\check{x}_{k-1}) = \begin{pmatrix}
    \mathbf{r}_{k-1} + (\mathbf{v}_{k-1} + \mathbf{V}_{k-1}) \Delta t \\
    \mathbf{q}_{k-1} \otimes \mathbf{q}(\mathbf{\omega}_{k-1} + \mathbf{\Omega}_{k-1} \Delta t) \\
    \mathbf{v}_{k-1} + \mathbf{V}_{k-1} \\
    \mathbf{\omega}_{k-1} + \mathbf{\Omega}_{k-1} \\
    \vdots \\
    \mathbf{y}_N
\end{pmatrix}
\]

Where \( \otimes \) denotes the quaternion product, and \( \mathbf{q}(\bullet) \) denotes the transformation from a rotation vector to a quaternion. \( \mathbf{V} \) and \( \mathbf{\Omega} \) is the model acceleration and angular acceleration noise times \( \Delta t \) respectively. As evident from the equation, the landmarks are left unchanged during this step. This is the case, even their positions are relative to the camera, because the camera positions when they were initialized is stored.

The prediction Jacobian \( F \) is given by:

\[
F = \begin{pmatrix}
    \mathbf{I} & 0 & \Delta t \mathbf{I} & 0 \\
    0 & \partial \mathbf{q}_k / \partial \mathbf{r}_{k-1} & 0 & \partial \mathbf{q}_k / \partial \mathbf{v}_{k-1} \\
    0 & 0 & \mathbf{I} & 0 \\
    0 & 0 & 0 & \mathbf{I}
\end{pmatrix}
\]

(4.4)

Since the landmarks are left unchanged, they are also left out of \( F \). The noise-Jacobian \( G \) is computed as

\[
G = \begin{pmatrix}
    \Delta t \mathbf{I} & 0 \\
    0 & \partial \mathbf{q}_k / \partial \mathbf{\Omega} \\
    \mathbf{I} & 0 \\
    0 & \mathbf{I}
\end{pmatrix}
\]

(4.5)

The derivation of these Jacobians is quite tedious and outside the scope of this report. And the closed-form solutions are too long to print in a readable way. A detailed description of how it is done can however be found in the Appendix of [7].

4.1.4 Measurements

The measurements \( z_i \) of this sensor are 2D image-coordinates of detected landmarks in the image. These measurements are compared to predicted image positions of the same landmarks \( \mathbf{h}_i \) (3.11), and related to the current state via the measurement Jacobian \( H \) (3.12).

Initialization of new features

Sometimes (most notably during the first iteration of the filter), new features need to be found and added to the filter. How new features are actually found is described
in Section 4.1.5. But once they are found, the filter needs to be updated with the new landmark, as well as its covariance.

First, the actual parameters of the landmark needs to be calculated. The first three components of the landmark is the camera’s position at the time of initialization:

\[
(x_i, y_i, z_i) = r^C
\]  
(4.6)

Next is the direction of the landmark, defined by the azimuth and elevation angles:

\[
\begin{pmatrix}
\theta_i \\
\phi_i
\end{pmatrix} = \begin{pmatrix}
\arctan (u_x, u_z) \\
\arctan (-u_y \sqrt{u_x^2 + u_z^2})
\end{pmatrix}
\]  
(4.7)

where \(u\) is the directional vector through the image coordinates \((u_i, v_i)\) of the measurement, expressed in the world-coordinate frame:

\[
u = R(q^C) \begin{pmatrix} u_i \\ v_i \end{pmatrix}
\]  
(4.8)

Finally, the inverse-depth-parameter, \(\rho\), is chosen just as Civera et al. [5] do, to be \(\rho = 1.0\), with a standard deviation of \(\sigma_\rho = 0.5\).

Next is to update the state covariance, \(P_{k|k}\) to reflect the new landmark.

\[
P_{k|k}^{\text{new}} = J \begin{pmatrix} P_{k|k} & 0 & 0 \\ 0 & R_i & 0 \\ 0 & 0 & \sigma^2_\rho \end{pmatrix} J^T
\]  
(4.9)

\[
J = \begin{pmatrix}
I & 0 \\
\frac{\partial y}{\partial x^C} & 0 \\
\frac{\partial y}{\partial (u_i, v_i, \rho)} & \partial (u_i, v_i, \rho)
\end{pmatrix}
\]  
(4.10)

Where the \(0\) on the second row reflects the zero-vector corresponding to the partial derivatives \(\frac{\partial y_{N+1}}{\partial y_1} = \ldots = \frac{\partial y_{N+1}}{\partial y_N} = 0\).

**Landmarks and Measurement equation**

Via the measurement equation, the measured image-coordinates can be transformed to corresponding 6D Inverse depth coordinates, which describes the landmarks’ positions in the environment relative to the camera. The measurement equation is simply a combination of equations (4.6) - (4.8).

Storing this information allows for the prediction of their position in the image frame, which enables the computation of \(h_i\) for every feature \(i\) via the reverse transform.

First, the inverse depth coordinates are transformed into Cartesian coordinates relative to the predicted camera frame:

\[
h_i^C = R^{CW} \begin{pmatrix}
\rho_i \\
\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - r^{WC} + \begin{pmatrix}
\cos \phi_i \sin \theta_i \\
-\sin \phi_i \\
\cos \phi_i \cos \theta_i
\end{pmatrix}
\end{pmatrix}
\]  
(4.11)
4.1. THE VISION SYSTEM

Due to lens-distortion in the camera, these coordinates are corrected according to

\[ x' = \frac{x}{z} \]  
\[ y' = \frac{y}{z} \]  
(4.12)

\[ x'' = x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2 (r^2 + 2x'^2) \]  
(4.14)

\[ y'' = y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1 (r^2 + 2y'^2) + 2p_2 x' y' \]  
(4.15)

where \( r^2 = x'^2 + y'^2 \). This is followed by the pinhole camera model being applied:

\[ h_i = \begin{pmatrix} C x + f x'' \\ C y + f y'' \end{pmatrix} \]  
(4.16)

In (4.11) \( R_{CW} \) is the predicted camera rotation in the world coordinate frame, and \( r_{CW} \) is the predicted camera position in the world coordinate frame.

Measurement Jacobian

The measurement Jacobian, \( H \), defined in (3.12), is also quite tedious to derive, and the closed-form solution is too long to fit here. It can however be found in the Appendix of [7].

Since another lens distortion model has been used here, the partial derivatives representing the undistortion changes to:

\[ \frac{\partial h_i}{\partial h_j} = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix} \]  
(4.17)

where

\[ p_{11} = k_1 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right) + k_2 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)^2 + k_3 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)^3 + 1 - \frac{2 p_1 (c_y - v)}{f} + (c_x - u) \left( \frac{k_1 (2 c_x - 2 u)}{f^2} \right) + \frac{2k_2}{f^2} \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right) (2 c_x - 2 u) \]  
(4.18)

\[ + 3 k_3 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)^2 (2 c_x - 2 u) \]  
\[ - 3 p_2 \frac{(2 c_x - 2 u)}{f} \]
\[ p_{12} = -\frac{2p_2}{f} \left( c_y - v \right) - \left( c_y - v \right) \left( \frac{k_1 (2c_x - 2u)}{f^2} \right) \]
\[ + \frac{2k_2 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)}{f^2} \left( 2c_x - 2u \right) \]
\[ + \frac{3k_3 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)^2}{f^2} \left( 2c_x - 2u \right) + p_1 \left( 2c_x - 2u \right) \]  
\[ (4.19) \]

\[ p_{21} = -\frac{2p_1}{f} \left( c_x - u \right) - \left( c_x - u \right) \left( \frac{k_1 (2c_y - 2v)}{f^2} \right) \]
\[ + \frac{2k_2 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)}{f^2} \left( 2c_y - 2v \right) \]
\[ + \frac{3k_3 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)^2}{f^2} \left( 2c_y - 2v \right) + p_2 \left( 2c_y - 2v \right) \]  
\[ (4.20) \]

\[ p_{22} = k_1 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right) + k_2 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)^2 \]
\[ + k_3 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)^3 + 1 - \frac{2p_2}{f} \left( c_x - u \right) + \left( c_y - v \right) \left( \frac{k_1 (2c_y - 2v)}{f^2} \right) \]
\[ + \frac{2k_2 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)}{f^2} \left( 2c_y - 2v \right) \]
\[ + \frac{3k_3 \left( \frac{(c_x - u)^2}{f^2} + \frac{(c_y - v)^2}{f^2} \right)^2}{f^2} \left( 2c_y - 2v \right) - 3p_1 \left( c_y - 2v \right) \]  
\[ (4.21) \]
4.1. THE VISION SYSTEM

Measurement Noise

The Kalman Filter requires a measurement noise matrix $R$, that represents the uncertainty in the measurement. This matrix was found experimentally. High values indicates that the measurements are unreliable, and more emphasis should be put on other sensors, or the prediction. Results for an experiment on how the measurement noise impacts the results can be found in Section 5.1.

4.1.5 Detecting and matching landmarks

Landmarks are initialized using the FAST-detector \[10\] applied to randomly selected regions of interest in the image frame. The detection threshold depends on the nature of the image and should be tweaked for each occasion. The landmark descriptor is a $41 \times 41$ px image patch extracted from around the landmark. When new landmarks has been initialized, the state and covariance vectors are updated according to Section 4.1.4.

To find the area where to search for and already initialized landmark, an active search-approach, described in Section 3.7 is used. This area is exhaustively searched using template-matching, looking for matches of the middle $17 \times 17$ px in the patch descriptor. The matching-criteria is the correlation between the predicted landmark appearance and the neighbourhood around each point. Implementation-wise this is done via template-matching in OpenCV \[3\] using the normalized cross-correlation-coefficient as criteria:

$$R(x, y) = \frac{\sum_{x',y'} T(x', y') \cdot I(x + x', y + y')}{\sqrt{\sum_{x',y'} (T(x', y')^2 \cdot I(x + x', y + y')^2}$$  (4.22)

Where $T(x, y)$ denotes the pixels at position $(x, y)$ in the template, and $I(x, y)$ denotes pixels at position $(x, y)$ in the image.

4.1.6 Choosing corresponding matches

In order to get a coherent set of corresponding matches, outlier-rejection is important. This is achieved using 1-point RANSAC (see Section 3.7.2). This set of matches are called “Low-innovation inliers”, and are used in the first of two update-steps in the EKF.

Rescuing rejected inliers

Just as in \[6\], so-called “High-innovation inliers” are rescued after the first update step. This set is then used to perform a second update-step in the EKF. See Section 3.7.2.

\[3\] http://docs.opencv.org/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#matchtemplate

31
4.1.7 Kalman update

The update-steps are performed just as usual, using the state, covariance and measurements provided. First, a Kalman update is performed using the low-innovation inliers found by 1-point RANSAC. Then high-innovation inliers are rescued, and finally a second Kalman update is performed, using measurements and covariances of these rescued landmarks.

4.1.8 Patch-warping

The patches are warped according to the landmarks’ positions relative to the camera, to allow for more robust matching. This is done via a Homography (see Section 3.6). The homography is calculated as in eq. (3.21):

\[ H = C \left( R^C - t^C \hat{n} \right) / d \]  \hspace{1cm} (4.23)

where \( C \) is the camera matrix, \( R^C \) is the difference between the current rotation and the rotation when the feature was initialized, \( t^C \) is the translation since the feature was initialized, \( \hat{n} \) is the patch’s normal and \( d = 1/\rho \), the distance to the feature. \( \hat{n} \) is estimated as the initial direction from the camera to the patch:

\[ n = \begin{pmatrix} -u_i \\ -v_i \\ f \end{pmatrix} \]  \hspace{1cm} (4.24)

\[ \hat{n} = \frac{n}{\|n\|} \]  \hspace{1cm} (4.25)

Implementation-wise, the OpenCV-functions \texttt{perspectiveTransform} and \texttt{warpPerspective} are used to execute the warping.

4.2 The IMU-system

The IMU used is called x-IMU and is produced by x-io Technologies. It provides an onboard algorithm that filters the output from the accelerometers, gyroscopes and magnetometers to produce a drift-free orientation quaternion. The algorithm used is the same as the one presented in \cite{12}.

4.2.1 EKF parameters

The first step is to create a Kalman filter for just the IMU, using the provided orientation-quaternion as measurements. The state vector is the same as in the vision system, but without the landmarks (essentially Equation 4.1). Using the same motion model, the prediction equations and Jacobians become exactly the same as in the vision system \( 4.33 - 4.5 \).

Using the x-IMU, part of the state in the Kalman filter is directly observable, which leads to the following trivial measurement equation and not-as-trivial Jacobian:
4.2. THE IMU-SYSTEM

Figure 4.2: The x-IMU from x-io Technologies.

\[ z_k = h(x_k) = q_k \] (4.26)

\[ H = \begin{pmatrix} 0 & 1 & 0 & \frac{\partial q_k}{\partial \omega_{k-1}} \end{pmatrix} \] (4.27)

Where the last component can be computed via the chain rule:

\[ \frac{\partial q_k}{\partial \omega_{k-1}} = \frac{\partial q_k}{\partial q(\omega_{k-1} \Delta t)} \frac{\partial q(\omega_{k-1} \Delta t)}{\partial \omega_{k-1}} \] (4.28)

These factors can be computed with the quaternion product formula and the conversion formula between rotation vectors and quaternions:

\[ \frac{\partial q_k}{\partial q(\omega_{k-1} \Delta t)} = \begin{pmatrix} q_0^2 - q_x^2 - q_y^2 - q_z^2 \\ q_x^2 - q_y^2 - q_z^2 - q_0^2 \\ q_y^2 - q_z^2 - q_x^2 - q_0^2 \\ q_z^2 - q_x^2 - q_y^2 - q_0^2 \end{pmatrix} \] (4.29)

where \( q^2 = q_{k-1} \). (4.30)

\[ \frac{\partial q(\omega_{k-1} \Delta t)}{\partial \omega_{k-1}} = \begin{pmatrix} \frac{\partial q_0}{\partial \omega_0} & \frac{\partial q_1}{\partial \omega_0} & \frac{\partial q_2}{\partial \omega_0} & \frac{\partial q_3}{\partial \omega_0} \\ \frac{\partial q_0}{\partial \omega_1} & \frac{\partial q_1}{\partial \omega_1} & \frac{\partial q_2}{\partial \omega_1} & \frac{\partial q_3}{\partial \omega_1} \\ \frac{\partial q_0}{\partial \omega_2} & \frac{\partial q_1}{\partial \omega_2} & \frac{\partial q_2}{\partial \omega_2} & \frac{\partial q_3}{\partial \omega_2} \\ \frac{\partial q_0}{\partial \omega_3} & \frac{\partial q_1}{\partial \omega_3} & \frac{\partial q_2}{\partial \omega_3} & \frac{\partial q_3}{\partial \omega_3} \end{pmatrix} \] (4.31)

where \( q^4 = q(\omega_{k-1} \Delta t) \). (4.32)

The measurement noise \( R \) for the IMU was found experimentally to be: \( R = 0.0001 \times I \).
CHAPTER 4. METHOD

4.3 Fusion of IMU and vision data

Since both the IMU and vision systems are based on an EKF, the fusion becomes quite natural: multiple update steps of the filter. The filter-loop can be seen in Listing 4.1.

```python
image = retrieve_initial_image()

while(new_measurements_available()):
    initialize_new_features(image)
    kalman_predict()
    new_image = retrieve_new_image()
    imu_data = retrieve_imu_data()
    measurements = match_features(new_image)
    li_inliers = ransac(measurements)
    kalman_update(li_inliers)
    hi_inliers = rescue(measurements)
    kalman_update(hi_inliers)
    kalman_update(imu_data)
    image = new_image
```

Listing 4.1: Pseudocode for the filter loop.

As can be seen from the code, the IMU-data and image data are retrieved directly after one another. This is to keep the time difference between the IMU and image measurement as low as possible. When running the filter offline, on a previously recorded sequence, retrieve_imu_data() becomes retrieve_imu_data_at(time), where time is the timestamp of the image.

Note however, that the timestamping of the data is a problem on its own, since the data received from the camera and/or IMU may have been buffered for an unknown period of time. Thus, it is often needed to analyze the data beforehand, and determine the offset between the timestamps of the IMU data and the timestamps of the image data.

4.3.1 Other fusion approaches

Other approaches for fusing the IMU with the monocular data were also tested. Since the IMU makes part of the state (the rotation quaternion) directly observable, it would make sense to use this as a priori information for the filter, and thus use it in the prediction step. Instead of trying to predict the orientation of the camera in the next frame, one could use what the IMU says directly. This would enable the vision system to search for matching landmarks in the predicted areas, and explain the discrepancies with the prediction with pure translation (since the rotation was already given by the IMU). This method was also implemented and tested, using MATLAB. Only a modified prediction function, \( \mathbf{f} \) is needed (compare to (4.33)):  

34
4.4. EXPERIMENTAL SETUP

\[
\hat{x}_{k|k-1} = f(\hat{x}_{k-1}, q_{k,IMU}) = \begin{pmatrix}
    x_{r_{k-1}} + (v_{k-1} + V_{k-1}) \Delta t \\
    q_{k,IMU} \\
    v_{k-1} + V_{k-1} \\
    \omega_{k-1} + \Omega_{k-1} \\
    \vdots \\
    y_N
\end{pmatrix}
\] (4.33)

Tests using the mean value between the IMU-measurement and the predicted rotation were also made.

Another approach would be to use the individual sensors on the IMU (accelerometer, gyroscope and magnetometer) to measure the acceleration and angular rotation of the camera directly, and let the EKF do the rest. This would require the state to be extended with acceleration components, and more complex pre-processing of the data would be needed in order to subtract the gravitational vector. This method was not fully developed or tested in this project.

4.4 Experimental setup

The x-IMU was mounted on a Photon Focus MV1D2080 camera, as depicted in Figure 4.3. The IMU was placed as close to the camera’s optical center as possible, in order for them to get approximately the same rotational axes.

The camera was mounted on a tripod, as depicted in Figure 4.4. The mount allows for locking the different rotational axes, which allowed for recording sequences containing pure panning, tilting and rolling motions. More generalized motions that included translations as well as rotations were also recorded.

To obtain a “ground truth” for the pure rotational movements, visual landmarks in the image were used to ensure that the final orientation of the camera was the same as its initial orientation for some of the sequences. For the translational movements, markers on the ground were also used, in combinations with the mentioned visual landmarks, to ensure that both the position and orientation of the last frame was the same as the initial frame (see Figure 4.4).

4.5 Test sequences

In this section, the test sequences used to evaluate the system are described. The indoor sequences were shot in a static office environment, with lots of visual landmarks and short distances. The outdoor sequences were shot at a driving range, a dynamic environment with fewer visual landmarks in the image, and large open spaces. See Figure 4.5. The indoor sequences were first filtered with a median filter, to remove the salt and pepper-noise present in low-light conditions.
CHAPTER 4. METHOD

4.5.1 Indoor tilt sequence

A short sequence recorded in an office environment, where a mounted camera is tilted upwards about 55°. Then the camera is tilted downwards to an inclination of about -15° followed by a tilt upwards, until it reaches its a final inclination of about 18°. No translational movement. See Figure 4.6a.

4.5.2 Indoor pan sequence

A short sequence recorded in an office environment, where a mounted camera is first rotated in a panning motion of about 80°. The camera is then panned back again, passing its initial orientation, and ending rotated about 8° in the other direction. No translational movement. See Figure 4.6b.

4.5.3 Outdoor pan sequences

Two short sequences recorded at a golf driving range, where the mounted camera was panned to the side. One of the sequences features an occlusion in the form of...
4.5. TEST SEQUENCES

Figure 4.4: The mount was moved between the two sets of markers (blue circles), a translation of 139 cm.

a person moving about in front of the camera. IMU-references are in Figure 4.7. A picture demonstrating the occlusions is in Figure 4.8.

4.5.4 Outdoor roll sequence

A short sequence used to test the system’s ability to handle roll-rotations (which are not very common in filmed footage but nonetheless may occur). The camera was mounted on a stand, and carefully rolled counter-clockwise. See Figure 4.9 for the IMU reference.

4.5.5 Outdoor translational sequences

Two short sequences recorded at the driving range, where a mounted camera was moved by hand between two sets of markers placed 139 cm apart. See Figure 4.4. Major occlusions in one of the sequences. See Figure 4.10 for IMU references.
(a) Indoor environment.  

(b) Outdoor environment.

Figure 4.5: Example images from the office and the driving range.

---

(a) Tilt  

(b) Pan

Figure 4.6: Reference rotation curve from the IMU for the indoor tilt and pan sequences.
4.5. TEST SEQUENCES

(a) Pan without occlusions  (b) Pan with occlusions

Figure 4.7: Reference rotation curve from the IMU for the outdoor pan sequences.

Figure 4.8: Example of occlusion in the occluded outdoor pan sequence.
Figure 4.9: IMU reference for the outdoor roll sequence.

(a) Translation sequence without occlusions  (b) Translation sequence with occlusions

Figure 4.10: Reference rotation curve from the IMU for the outdoor translation sequences.
Chapter 5

Experiments

In this chapter, the results of the experiments performed to evaluate the system are presented. In Section 5.1, only the vision system is evaluated to determine how to tweak it for different environments, as well as to determine its inherent strengths and weaknesses. In Section 5.2, the two different fusion approaches are evaluated. Then it continues in Section 5.3 by focusing on one of them, and different situations where the addition of the IMU data is invaluable are demonstrated. Finally, Section 5.4 evaluates the feasibility of using the system in a real-time environment by looking at possible bottlenecks in the algorithms.

Overall, the results look promising, albeit far from perfect. Qualitatively, the markers in the image, indicating where matched landmarks are found, seem to follow the landmarks well throughout the sequences. This indicates that landmarks are tracked well, and should allow for a robust placement of virtual objects in the image. See Figure 5.1.

Throughout the chapter, you will see a lot of plots. They are divided into two subplots - the left shows the estimated position of the camera throughout the sequence, while the right plot shows the estimated rotation, expressed in Euler angles. The shaded areas around the lines represent ±1 standard deviation from the estimated value (taken from the covariance matrix $P$). Please note that while the rotations are measured in degrees $^\circ$, the positions are in arbitrary units, since the scale parameter is unknown. Thus, positions between plots can not really be compared to each other, only relative motions within the plot as well as the general “shape” of the movements.

5.1 The influence of the vision approach

In this section, different parameters of the vision system, and their impact on the overall results are evaluated. All results in this section have been produced without using the IMU, to put emphasis on the vision part of the system.
(a) Identified landmarks at time $t$ frames. (b) Identified landmarks at time $t + 5$ frames.

Figure 5.1: Matched landmarks at two different time-steps. Many of the landmarks identified in the first frame, are found at the same places five frames later, which indicates that the system is capable of tracking salient landmarks. Some landmarks are however lost, while others have been added. Red indicates low-innovation inliers, while green indicates high-innovation inliers.

5.1.1 Patch warping

Implementation of the patch-warping component of the system provided a noticeable challenge. Experimental results indicate that the patch warper sometimes reduces the performance of the system. Since the motions performed are quite restricted to probable motions performed with a shoulder-mounted camera, the absence of a working patch warper should not influence the results very much. Therefore, patch-warping has been turned off throughout the results presented here, except for when it is necessary. Basically, if the sequence contains considerable changes in roll, patch warping is needed (see Figure 5.2).

5.1.2 Feature detection

One of the parameters of the system is the threshold for the FAST feature detector. A lower threshold results in more features being detected, which might be good if features in less-texturized areas are really needed. It does however also lead to less salient features, meaning that some of them will be very hard to track over time. Or in short: lower threshold leads to better landmarks in terms of their position in the 3D world, but worse features in terms of their trackability. Recent research indicates that more landmarks to track is one of the best ways to improve the accuracy of
5.1. THE INFLUENCE OF THE VISION APPROACH

(a) Roll: With patch warping.

(b) Roll: Without patch warping.

(c) Pan: With patch warping.

(d) Pan: Without patch warping.

Figure 5.2: The patch warping is necessary when the sequence contains considerable roll-rotations (since the whole image rotates), but actually makes the result worse for the panning sequence.
a Monocular SLAM system [17]. More features does however also lead to a slower system.

To test how the FAST threshold impacts the results, the system has been evaluated of two different sequences, where everything was kept constant except for the threshold of the FAST detector. Figure 5.3 shows the results for the indoor panning sequence, while Figure 5.4 shows the results for the outdoor panning sequence. Since the indoor sequence contains many more salient landmarks, a higher threshold may still result in many landmarks being found, which leads to both precise and robust estimation results. The outdoor sequence, on the other hand, has less texture and fewer salient landmarks to track. In order to get as many landmarks (and with it as good accuracy) as in the indoor sequence, the threshold of the detector must be lowered, thus risking less salient landmarks to be tracked.

Another important parameter of the vision system is the nature of the descriptor. The main descriptors used in this system are image patches, i.e. just tiny patches of the image at the positions of the landmarks. Other types of descriptors, such as the SIFT and SURF descriptors were also tried, but deemed to slow during the matching step (since a FLANN-based NN-classifier is needed to match them).

5.1.3 EKF noise parameters

The model noise and measurement noise parameters in the EKF represents the expected random deviations in the motion model and in the measurements respectively. Sometimes, increasing the noise leads to better performance with non-linear models.

Here, the results of changing the measurement noise $R$, the linear acceleration noise $\sigma_a$ and the linear angular acceleration noise $\sigma_\alpha$ have been evaluated on the indoor tilt sequences (see Section 4.5). The reference motion, recorded by the IMU, can be seen in Figure 4.6a. The results of changing the noise parameters can be seen in Figure 5.5.

Increasing the measurement noise $R$ is equivalent to telling the filter that it should emphasize the model more and the measurements less. Thus, a higher $R$ should lead to a smoother curve, since the model uses the Constant Velocity assumption made in Section 3.3. Looking at Figure 5.5, this seems to be the case. $R = 50$ was chosen as the default value for the system.

A high value on $\sigma_a$ indicates more uncertain linear accelerations, which corresponds to faster and more jerky translational movements of the camera. Since the indoor tilt sequence only contains rotational movements, a low value on $\sigma_a$ is more appropriate, as indicated by Figure 5.6. But to accommodate for more general motions, the default value of the system is set to $\sigma_a = 0.5$.

The standard deviation of the angular acceleration is denoted $\sigma_\alpha$. Analogous to $\sigma_a$, a high $\sigma_\alpha$ will allow for fast and jerky rotations of the camera. Since the tilting sequence only contains rotational movement, too low a value will result in the model not being able to predict the rotations performed, while too high a value will cause oscillations of the estimated angles due to noisy measurements. See Figure 5.7.
5.1. THE INFLUENCE OF THE VISION APPROACH

(a) FAST threshold: 10.

(b) FAST threshold: 50.

(c) FAST threshold: 100.

Figure 5.3: In the indoor panning sequence, the abundance of salient landmarks in the scene allows for a higher FAST threshold, thus easily skipping “bad” landmarks.
CHAPTER 5. EXPERIMENTS

Figure 5.4: In the outdoor panning sequence, a lower FAST threshold is needed to find enough landmarks to get good estimations.
default value is $\sigma_\alpha = 5.5$.

### 5.1.4 RANSAC threshold

There are basically two thresholds governing the RANSAC algorithm. First is the threshold for whether to consider a landmark a low-innovation inlier or not. Too low a value will make it difficult (or impossible) to form RANSAC hypotheses with high support. This may force the system to choose a hypothesis that is the “best of the worst”, which may lead to spurious or oscillatory behaviour. A too large value on the other hand, leads to clear outliers being classified as inliers, thus impacting the results. This may also lead to spurious behaviour and large “jumps”. Which RANSAC threshold is optimal largely depends on the nature of the sequence: is the environment static or dynamic? Are there occlusions in the sequence? Is the sequence very noisy, etc.?

The other important threshold is in the rescuing part, namely how far from the hypothesis an outlier may be to be regarded as a high-innovation inlier. If everything about the problem at hand is known, it is possible to determine exact values for these parameters for different probability levels. For example that 99% of the points classified as inliers actually are inliers.

### 5.2 The influence of the fusion algorithm

In this section, different fusion approaches, as well as different parameters within these approaches, are evaluated.

#### 5.2.1 Different fusion approaches

**IMU in prediction-step**

This approach is very sensitive to sync-issues between the camera and the IMU. If the IMU-data lags behind, the prediction will also lag behind, causing the update-steps to always over-compensate, eventually leading to severe drift. The same applies if the image data lags behind a bit. This approach might be successful if it is possible to perform hard synchronization between the camera and the IMU.

**IMU in update-step**

This worked quite well, and is the method that was used throughout the rest of the experiments.

#### 5.2.2 When to perform the IMU update

Assuming that the second fusion approach is used, it is possible to perform the Kalman update with the IMU-provided quaternion at different points in the loop.
CHAPTER 5. EXPERIMENTS

Figure 5.5: Impact of measurement noise, R, on the estimation of the camera orientation for the indoor tilt sequence.
5.2. THE INFLUENCE OF THE FUSION ALGORITHM

Figure 5.6: Impact of the acceleration noise in the model, $\sigma_a$, on the estimation of the camera orientation for the indoor tilt sequence.

(a) $\sigma_a = 0.005$

(b) $\sigma_a = 0.05$

(c) $\sigma_a = 0.5$ (standard)
Figure 5.7: Impact of the angular acceleration noise in the model, $\sigma_\alpha$, on the estimation of the camera orientation for the indoor tilt sequence.
5.2. THE INFLUENCE OF THE FUSION ALGORITHM

In practice, there are four different alternatives for when to perform this update step:

1. At the start of each iteration, right after the Kalman prediction step, but before predicting the features’ positions in the current frame.

2. After predicting the features’ positions in the current frame, but before using RANSAC to find inliers.


4. After both Kalman updates that uses vision data.

Results using these four different update-orders are found in Figures 5.8 - 5.11. The results are very similar, but there is a slight time-offset in the curves between performing the IMU-update before or after looking for low-innovation inliers.

![Figure 5.8: Camera pose when performing the Kalman update before predicting the features’ positions (Alternative 1).](image-url)
CHAPTER 5. EXPERIMENTS

Figure 5.9: Camera pose when performing the Kalman update after predicting the features’ positions, but before RANSAC (Alternative 2).

Figure 5.10: Camera pose when performing the Kalman update based on IMU data after Kalman update using low-innovation inliers, but before looking for high-innovation inliers (Alternative 3).

Figure 5.11: Camera pose when performing the Kalman update based on IMU data after both Kalman updates using vision data (Alternative 4).
5.3 Usefulness of the IMU

In this section, evaluations of the same sequences, with and without IMU, are compared. Generally speaking, the IMU helps “smoothen” the estimated rotations, allowing for higher values of the model noise. This is good, because it also allows the model to capture faster accelerations and angular accelerations. See Figure 5.12 for an example of this.

5.3.1 Translations and free movements

The largest difficulty for the vision system in an ideal setting is to differentiate between rotations and translations. Using prior knowledge, it is possible to restrict the motion model to a pure rotational or translational movement, which vastly improves the result to ones comparable with the fused system presented in this report. In many cases, however, no assumptions can safely be made about the motions, and the more general constant velocity model needs to be used. In such cases, the IMU may provide valuable rotational information to the system, which put restrictions on how the movements can be estimated.

Results from the outdoor translational sequence can be seen in Figure 5.13. Reference rotations are in Figure 4.10. The translation performed in the sequence is a free-hand translation in the X-direction. It is quite clear from the figure, that the system without the IMU prefers to explain the observed movements as combinations of rotations and various translations. The reason is that the motion model gives equal probability for all kinds of motions, and a more complex explanation may also account for noise and outliers in the sequence. With the IMU telling the EKF that the camera is facing in the same direction throughout the sequence, the system is restricted to use translations for explaining the movement.

5.3.2 Occlusions

One of the requirements of the system is that it should be able to handle more dynamic scenes, with partial, and sometimes total occlusions. Occlusions is a common problem with computer vision systems, and using a complementary proprioceptive sensor like an IMU may help a great deal with this.

By using the system on the two occlusion sequences, with and without using the IMU data, the usefulness of the IMU has been evaluated. First is the occluded panning sequence, in Figure 5.14. In Figure 5.15 the same graphs for the occluded translation sequence can be found. The corresponding IMU references are found in Figures 4.7b and 4.10b respectively. In the case of the panning sequence, the IMU clearly helps with the estimation: without the IMU, there is a clear spike in both the rotation and translation plot at the moment when the occlusion passes in front of the camera.

In the translational sequence however, the difference is quite small, and the translational movement is not captured in any of the tests. This is the case because
(a) With normal or high model noise, the estimated position and rotation is not very smooth.

(b) With the IMU added, however, the result is smoother and more precise.

(c) A similar effect can be achieved by lowering the model noise (in this case the linear acceleration have been lowered from 0.5 to 0.005). But this put more restrictions on the type of motions that the model can capture.

Figure 5.12: How the use of the IMU “smoothen” the results, even with rather high model noise.
5.3. USEFULNESS OF THE IMU

(a) With IMU

(b) Without IMU.

Figure 5.13: How the system performs with and without the IMU for translational sequences.

the vision system mostly found features far away from the camera. Far-away features does not show any parallax, which means that they indicate that the camera is still. Thus, even if a few closer features are found, they will be considered as outliers by the RANSAC-method since they are initialized as being at the same distance. This is an important problem with this system, that will need to be addressed.

5.3.3 Feature-starved environments

Another requirement is for the system to be able to handle feature-poor environments, such as a golf course. In these environments, it may be hard to find salient features, and even harder to keep track of them. This is especially true for features close to the camera. And it is these features that show parallax, which is used to determine translational movements. The problem here is that the IMU only pro-
Figure 5.14: How the system performs with and without the IMU for the occluded panning sequence.

provides rotations, and no information regarding translations. It may, however, force the vision system to describe the observed small changes in feature positions as parallaxes caused by translational movements instead of rotations.
5.3. USEFULNESS OF THE IMU

Figure 5.15: How the system performs with and without the IMU for the occluded translation sequence.
CHAPTER 5. EXPERIMENTS

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU time (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ransac</td>
<td>35.7</td>
</tr>
<tr>
<td>» compute_hypothesis_support</td>
<td>21.6</td>
</tr>
<tr>
<td>» hi_inverse_depth</td>
<td>15.7</td>
</tr>
<tr>
<td>» projectPoints</td>
<td>5.8</td>
</tr>
<tr>
<td>» Rodrigues</td>
<td>4.6</td>
</tr>
<tr>
<td>» Compute hrl</td>
<td>3.2</td>
</tr>
<tr>
<td>» Compute S</td>
<td>3.6</td>
</tr>
<tr>
<td>» Compute K</td>
<td>3.0</td>
</tr>
<tr>
<td>match_features</td>
<td>23.6</td>
</tr>
<tr>
<td>» predict_features</td>
<td>17.3</td>
</tr>
<tr>
<td>» Compute S</td>
<td>8.6</td>
</tr>
<tr>
<td>» Hi_inverse_depth</td>
<td>4.8</td>
</tr>
<tr>
<td>» predict_descriptor</td>
<td>2.5</td>
</tr>
<tr>
<td>» template_matching</td>
<td>5.6</td>
</tr>
<tr>
<td>update_li</td>
<td>22.3</td>
</tr>
<tr>
<td>» kalman_update</td>
<td>21.7</td>
</tr>
<tr>
<td>» Compute P</td>
<td>7.9</td>
</tr>
<tr>
<td>» Compute K</td>
<td>7.7</td>
</tr>
<tr>
<td>» Compute S</td>
<td>7.6</td>
</tr>
<tr>
<td>rescue_hi_inliers</td>
<td>8.0</td>
</tr>
</tbody>
</table>

Table 5.1: Performance analysis of the C++-implementation.

5.4 Performance analysis

Since one of the requirements of the system is that it should be feasible to implement in real-time, it is of interest to measure how the system performs, and to identify potential bottlenecks. Overall, the system runs in near-real-time for video sequences running in 25 fps. Depending on the sequence and its complexity, some slowdowns can occur, which slows the system down to about 5 fps.

A performance analysis using Visual Studio 2012 performance analyzer is summarized in Table 5.1. Two of the most resource-hungry functions, compute_hypothesis_support and predict_features are called inside loops iterating over all features, and are independent of each other. This means that they are suitable to be called in a parallel manner, which could be achieved quite easily. Other resource-hungry parts are the computations of the matrices S, K and P. These are achieved using simple matrix multiplication, which can be much more efficiently handled by the GPU.

All in all, since the performance of the system is quite close to real-time as it is; implementing the suggestions above should allow the system to run faster than real-time.
Chapter 6

Conclusions

6.1 Summary

The goal of this project was to implement a camera pose estimation system that can be used for augmented reality in a dynamic environment. The end-use of the system is to enable the overlay of graphics on a video sequence recorded by a moving camera in a visually realistic way. In order to accomplish this, two sensors were used: the camera that records the image sequence, and an IMU attached to the camera.

The idea is that the two sensors may complement each other: the IMU provides a very good estimate of the orientation of the camera at all times, even when the scene is occluded. The camera, on the other hand, provide the means to get a pixel-perfect estimation of the landmarks in the image, as well as means to measure the parallax of the landmarks in order to estimate translational movements.

The system developed is based on EKF-SLAM, as well as previously published work, especially by Civera et al. [6], to get a foundation of the vision-part of the system. One of the requirements on the system was that it should be usable in large open spaces, where most landmarks are far away. To accomplish this, inverse depth parameterization of the landmarks was used. That allowed immediate use of newly initialized landmarks as well as far away ones for estimating the orientation.

Another requirement was the ability for the system to cope with occlusions in the image. Even though the IMU can help a lot when there are occlusions in the image, a robust outlier-rejection scheme for the vision system is also needed. For this, the newly developed 1-point RANSAC algorithm for EKF-SLAM was used. Fusion of IMU and vision data came naturally with the EKF-framework, as multiple update-steps.

The system has been evaluated on real image sequences recorded in an environment very similar to where it is supposed to be used (a golf course). The results clearly indicate that using the combination of a camera and IMU sensor allows for more robust camera pose estimations during more difficult circumstances. Examples of such circumstances include occlusions, feature-poor environments and diverse types of combined movements.
The experiments do however also uncover some “failure-modes” for the system. One being that translations are sometimes not registered by the system, especially in feature-poor environments. Another is that the system is very sensitive to synchronization-issues between the IMU and the camera. When the sensors are out-of-sync the EKF may often diverge.

The performance analysis indicate that although the specific implementation evaluated here is not ready for real-time use as-is, this can probably be achieved quite easily by introducing matrix multiplications on a GPU, as well as parallelizing a few loops.

6.2 Future work

While the system performs relatively well in many situations, there are a number of tweaks and additional features which can be used to both increase the precision as well as the performance and robustness of the system. This section aims to present some pointers for future work on how some of them can be implemented and what they would improve.

First of all, a more thorough qualitative analyse of the results should be made, where actual virtual 3D-objects are placed in the video sequence, and see how it looks when the camera moves. As of now, the infrastructure to do this is not finished, and the only qualitative analyses have been to look at how the markers indicating the landmarks have moved throughout the sequence. This indicates whether the placement of the object in the sequence works, but does not tell much about its perspective.

One of the main problems with the system is that the very general constant velocity model (see Section 4.1.3) allows for very complex motion patterns of the camera. This is especially apparent when the IMU is turned off. One would probably want some model that still allows for complex motions, but prefer the simple explanations if they are viable (think Occam’s razor). One method to do this is suggested in [7]. The idea is to perform a maximum-likelihood-estimate at every iteration in the Kalman loop to determine the most probable model $M$, given the data in the previous iteration $y_i$:

$$\max_{M} P(M | y_1, ..., y_N).$$

(6.1)

But even without such an estimate, it is possible to use a more restricted motion model, which restricts the movements to those probable when using a shoulder-mounted camera.

Another problem inherent to Monocular SLAM and SfM is the unknown scale parameter. All positions and translations estimated are only relative. This makes it difficult to compare results from different sequences or with different parameters, as there is no relevant metric absolute ground-truth to compare with. Nüeti et al. [14] use the direct accelerometer and gyroscope values from the IMU to estimate the scale of translations and positions. They use the PTAM framework, but since
6.2. FUTURE WORK

they use an EKF to fuse everything together, it could naturally be integrated in this framework.

A somewhat practical, but very important difficulty with the specific implementation of the system in this project, is the synchronization between the IMU and the camera. If the two sensors are out-of-sync, the EKF will try to merge two slightly disagreeing sensors. Since the measurement noise for the IMU is so much lower than for the camera, the rotation will be taken more or less directly from the IMU, and the difference due to sync-problems will most likely be explained by a translation instead. This problem could be avoided entirely by getting timestamps directly from the hardware driver, which unfortunately does not seem to be supported by the X-IMU. Another idea is to use a multi-rate filter that uses the sensor data as it comes, instead of using a fixed loop-rate that just uses the “latest” measurements.

One of the requirements for the system was that it should be able to track a specific point in the image, to allow the placing of a virtual object there. While the framework needed to accomplish just that is already implemented (since landmarks in the image are tracked), it has not yet been added to the system, and is not evaluated. Using only the camera pose to blindly place an object in the image probably is not robust enough. This small addition of allowing a user-specified point in the image to be tracked at all times, is thus needed before the system can be used as intended.

While the performance of the system is rather good, especially for indoor-sequences, it still does not operate at real-time (see Section 5.4). Since the most CPU-time is spent on matrix multiplications, which can be performed more efficiently on a GPU, and on loops where each iteration is independent of the others, real-time performance can most probably be achieved.

It would be interesting to see how a different type of filter, more suitable for non-linear estimations, would perform using the concepts of 1-point RANSAC and inverse depth parametrization together with an IMU. In [1], a particle filter is used to fuse IMU and vision data in a manner quite similar to the system described here, but without for example inverse depth parametrization, which means that they require fiducials present in the video.
Bibliography


