Live 3D Reconstruction on Mobile Phones

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH Zürich

(Dr. sc. ETH Zürich)

presented by

Petri Tanskanen

MSc ETH in Computer Science
ETH Zürich

born 03.04.1984
citizen of Berikon AG and Finland

accepted on the recommendation of

Prof. Marc Pollefeys
Prof. Otmar Hilliges
Prof. Margarita Chli

2016
Abstract

This thesis presents a system for mobile devices with a single camera and an inertial measurement unit that allows to create dense 3D models. The whole process is interactive, the reconstruction is incrementally computed during the scanning process and the user gets direct feedback of the progress. The system fills the gap in currently existing cloud-based mobile reconstruction services by giving the user a preview directly on the phone without having to upload the images to a server. The on-device reconstruction enables new applications where it is not desirable to send the raw images to a remote server due to security or privacy reasons. In addition, since the system is actively analyzing the scanning process, it can use the inertial sensor data to estimate the objects real-world absolute scale. This is not possible by only processing the images on a server.

A novel visual inertial odometry algorithm that uses the Extended Kalman Filter framework to directly fuse image intensity values with the inertial measurements to estimate the camera motion is proposed. The fusion at this low level combines the advantages of the high accuracy from direct photometric error minimization with the robustness to fast motions when using inertial sensors. Thanks to the constrained model of the filter, it is possible to track scenes where other approaches using external correspondence algorithms will fail. The method works on a sparse set of image areas and can be efficiently implemented on mobile devices.
An efficient point cloud fusion algorithm is proposed that is based on a confidence weight computed from photometric and geometric properties to accurately combine depth measurements from different viewpoints into a consistent point cloud model. Thereby, visibility conflicts are detected and corrected and the measurements are then averaged by using their confidence scores as weight. The complete system is demonstrated to be working on various objects and in different environments and future applications are proposed.
Zusammenfassung


Ein neuer Algorithmus für Visual Inertial Odometry wird vorgeschlagen, welches das Extended Kalman Filter Framework benutzt, um die Intensitätswerte eines Bildes mit den Inertialsensormessungen zu fusionsieren. Das Verknüpfen der Daten auf diesem Level ermöglicht es, die Vorteile der Genauigkeit der photometrischen Optimierung und der Robustheit gegenüber schnellen Bewegungen durch die Verwendung der Inertialdaten zu kombinieren. Durch die inherenten math-
mathematischen Bedingungen im Modell des Filters funktioniert der Algorithmus in Umgebungen, in denen andere Systeme, die externe Correspondence-Algorithmen benutzen, versagen. Die vorgestellte Methode benutzt nur kleine Teile des Bildes für die Berechnungen und kann deswegen effizient auf einem mobilen Gerät implementiert werden.

Acknowledgement

This thesis would not have been possible without the support and contributions of many persons. Therefore, I want to express my sincere gratitude to the following persons:

First I would like to thank my advisor Prof. Marc Pollefeys, whose lecture about 3D photography I attended one year after he joined ETH and he got me so fascinated about the topic that I ended up doing my PhD study in his group. During all that time he was of great help and contributed many inspiring ideas that finally led to this thesis.

My appreciation goes to Lorenz Meier for pushing the Pixhawk project that allowed me to get in touch with the research group and pursuing many long technical discussions with me. And to Dr. Friedrich Fraundorfer, Dr. Gim Hee Lee and Dr. Lionel Heng for the great first two years in which we worked on micro aerial vehicles. Special thanks to Dr. Kalin Kolev, who enabled the Mobile Scanner project with his contributions and to all other collaborators on the project, Dr. Amaël Delaunoy, Federico Camposeco, Olivier Saurer and Pablo Speciale. I want to thank Dr. Torsten Sattler and all peers of the CVG for taking time for inspiring discussions and supporting each other.

I want to especially thank my colleagues in the dark basement lab, Dominik Honegger, Nico Ranieri and all others for our great time with Fondue and fruitful discussions. And to Tobias Nägeli for the many
ups and downs with our joined project on visual odometry, and to my co-advisor Otmar Hilliges who spent long nights in helping us getting our work done.

Thanks to the administration, Susanne Keller, who took care of so many problems on so many different levels, and Thorsten Steenbock for the numerous helpful discussions and his trust in me.

Lastly, I want to thank my parents and my twin sister for supporting me in many ways and making this thesis possible.
Contents

1 Introduction 1

2 Foundations 7
  2.1 Camera Models ........................................ 7
     2.1.1 Pinhole Camera .................................... 8
     2.1.2 Fish Eye Camera .................................... 10
  2.2 Optimization ........................................... 12
     2.2.1 Non-linear Least Squares ............................ 12
     2.2.2 Optimization on Manifolds .......................... 15
     2.2.3 Examples ........................................... 17
     2.2.4 Filtering ........................................... 20
  2.3 Stereo Depth Estimation ................................. 22
     2.3.1 Stereo ............................................... 22
     2.3.2 Depth Map Fusion .................................... 28

3 3D Reconstruction on Mobile Phones 31
  3.1 Related Work ........................................... 33
  3.2 System Overview ......................................... 35
  3.3 Visual Inertial Scale Estimation ........................ 36
  3.4 Visual Tracking and Mapping ............................. 43
     3.4.1 Two View Initialization ............................... 43
     3.4.2 Patch Tracking and Pose Refinement ................. 44
     3.4.3 Sparse Mapping ...................................... 44
3.5 Dense 3D Modeling ........................................ 45
  3.5.1 Image Mask Estimation ............................... 46
  3.5.2 Depth Map Computation .............................. 47
  3.5.3 Depth Map Filtering .................................. 52
3.6 Experimental Results ..................................... 52
3.7 Conclusion .................................................. 59

4 Semi-Direct Visual Inertial Odometry ................. 61
  4.1 Related Work ............................................. 63
  4.2 System Overview ......................................... 64
  4.3 Error State Filter Design .............................. 67
    4.3.1 Statespace structure ............................... 67
    4.3.2 Point Parametrization ............................. 67
    4.3.3 Continuous Time Model ............................ 68
    4.3.4 Prediction .......................................... 68
  4.4 Photometric Update ..................................... 69
    4.4.1 Patch extraction .................................. 72
    4.4.2 Image Pyramid Level Selection .................... 72
    4.4.3 Iterated Sequential Update ......................... 74
    4.4.4 Measurement Compression ........................... 74
    4.4.5 Anchor-centered Parameterization .................. 75
  4.5 Experimental Results ..................................... 76
  4.6 Conclusion ................................................. 81

5 Point Cloud Fusion ........................................ 83
  5.1 Related Work ............................................. 84
  5.2 Multi-Resolution Depth Map Computation ............ 86
  5.3 Confidence-Based Depth Map Fusion ................... 87
    5.3.1 Confidence-Based Weighting ....................... 88
    5.3.2 Measurement Integration ........................... 92
  5.4 Experimental Results ..................................... 98
    5.4.1 Comparison to Alternative Techniques ............ 98
    5.4.2 Live 3D Reconstruction on a Mobile Phone ....... 100
  5.5 Conclusion ................................................ 102
6 Applications 105
   6.1 Face Modeling ......................... 105
   6.2 Generic Modeling with Cloud Support ........ 110
   6.3 Sparse Scene Reconstruction .......... 112

7 Conclusion 115
   7.1 Future Work .............................. 116

Bibliography 119
Chapter 1

Introduction

3D modeling has become an increasingly important way of designing new products, planning buildings, running physically-based simulations and visualizing objects or environments. At the beginning, the only way of getting the 3D data of the objects or environments was to create them by hand with CAD or 3D modeling applications. This process is tedious and requires technical and also artistic skills. Often it is almost infeasible to recreate a real object as 3D model due to its complexity, therefore 3D scanning methods were developed to allow to measure the 3D surface. Subsequently, it can be imported to the modeling application to allow for comparison, measurements or modifications of real existing objects.

3D scanning technologies cover a large area of different technical solutions, the most used is laser scanning or LIDAR. Here, a laser range finder projects a laser beam or stripe on to the surface and its distance is measured by an optical sensor. Laser scanners allow for very precise measurements and very large range-spans. They range from table-top turntable scanners for smaller objects, and scanners with rotating heads that allow outdoor scanning of excavation sites up to ranges of several hundred meters to high power devices that can
be mounted on satellites to create elevation maps of the earths surface. Besides their precision and flexible range there are some drawbacks; since the scanning works with a single beam or stripe, scanning areas has to be done by moving the laser head, typically by rotating it. If the scanning device itself is moving, it cannot be assumed that the scanned area was observed from the same point in space anymore. In this case, processing the scanned data gets difficult and depending on the motion speed some part of the environment cannot be scanned at all.

For area scanning or mobile navigation devices that can measure a large amount of distances at once are better suited. For this purpose, 2D radars are available that can be used for distances in the range of kilometers or Time-of-Flight cameras that compute the distance per pixel by measuring the time that a light beam needs to travel to the surface and reflect back to the sensor through interference. Another set of 3D scanners is based on camera sensors. Here observations of the same scene point in two projective views are triangulated. One class of camera based 3D sensors are structured light scanners. These devices consist of a projector that sends out a pattern of light and a camera observing the image of the pattern. From the known pose of the projector and the camera relative to each other and the known projection of both lenses its then possible to recover the observed surface on the camera image. These scanners allow fast measurements of a large amount of pixels at video rate. One of the known sensors is the Microsoft Kinect sensor [1]. To hide the projected pattern from the camera image, typically infrared light is used for the projector. That way the scanner does not disturb the camera image in the visible spectrum, however, this approach has the drawback that the scanners have difficulties to work outdoors due to the strong infrared intensity of day light. The biggest advantage of these sensors is that they allow 3D measurements of dynamic scenes. Because all pixels are measured at the same time, it is possible to record even faster motions, for example on agile mobile robots like quadrotors, to allow for acquisition of useful scans for navigation.

Until now, all scanning technologies used radiated energy that was
sent out to make measurements, like a laser beam, radar waves or infrared light. Using two cameras allows for passive sensing - passive in the sense that light intensity which is already present in the scene is detected by the sensor. This allows to create energy efficient sensors but the drawback is that there needs to be enough light to be able to sense anything. Otherwise a light source needs to be mounted on the sensor, turning it again into an active sensor. Typical camera based 3D-scanners consist of two or more cameras with known relative poses that allow for precise and efficient processing of the images. Explained in a simple way, every pixel in one image shows a 3D point of the observed scene, the corresponding pixel in the other camera(s) image that shows the exact same 3D point is searched. In the same way as the structured light scanner, the 3D point can be reconstructed with the known relative pose and the known projection direction of the rays going through these two pixels and intersecting them in space.

During the last years, methods that enable 3D reconstructions from images taken from different view points with even different cameras were developed. These approaches allow to use large image collections from the Internet like Flickr to automatically create 3D models of points of interest where many people take images and upload them to a image database [2, 3]. The processing is done on powerful clusters of computers or on computers with several graphic cards. To bring the image based modeling on mobile devices like smartphones, cloud computing services like 123d Catch from Autodesk are offered. These services enable the user to use the camera in the smartphone to take pictures of an object from all sides and then upload them to the cloud where they are processed and the resulting 3D model is sent back to the smartphone. Thanks to the high computational power available behind this setup the models can be of high quality. The main issue is that average users cannot be supported in taking the right images from the right positions to ensure high accuracy and completeness of the final model. Occlusions, complex reflectance properties and shading effects often lead to failure in the reconstruction process since their effect on the appearance is difficult to predict in advance, especially for non-experts. This challenge is one of the motivations for this the-
Chapter 1 Introduction

Figure 1.1: This thesis deals with the problem of live 3D reconstruction on mobile phones. The proposed approach allows to obtain 3D models of pleasing quality interactively and entirely on-device.

sis. To address it, a monocular real-time capable systems which can provide useful feedback to the user in the course of the reconstruction process and guide their movements is needed. Through the interactivity, of the system the user can learn how to make useful image for creating models while he is scanning. This improves the situation for the user from the current state where all images are processed offline and if something goes wrong, the only feedback is an incomplete model or an error message, which is not helpful to learn what to do differently to improve. For some applications the fact that all computations are done on the device and data is not sent to some servers is very important. If the 3D data of the users face is for example used for authentication or some other body parts or even the whole body should be scanned, it is valuable to not have to sent all raw images first over the Internet.

Today’s smartphones provide an additional benefit, basically every phone is delivered with a built-in inertial measurement unit consist-
Figure 1.2: The scanned 3D models can be used to a large variety of applications, one of them being 3D printing.

In this thesis, we propose a system for 3D reconstruction that allows for full on-device processing and explore different extensions and im-
Chapter 1 Introduction

provements for this system and show some applications. We want to address the question of how to get a computationally demanding task such as image based modeling onto a smartphone. How to overcome the drawback of the unknown scale of a monocular camera system, as is the case on standard smartphones, by taking the Inertial Measurement Units (IMU) data into account? What way can we tightly combine the information from the camera with the IMU measurements on the raw level?

The core of the thesis is built upon the following peer reviewed papers:


The structure of the thesis is as follows; in the next chapter, a summarized overview of the theoretical background that will be used in the later chapters is given. In Chapter 3 the base system for mobile phone 3D reconstruction is described and first results are evaluated. Chapter 4 describes a novel Extended Kalman Filter-based algorithm for Visual Inertial Odometry that uses directly the image intensities of pixel patches to estimate camera motions. Chapter 5 extends the dense 3D modeling module of the system with a point cloud based fusion. Finally Chapter 6 shows some possible applications of the described system based on experiments done by the author and other people.
Chapter 2

Foundations

This chapter covers some of the theory used in the thesis and gives the reader a short introduction with references to more in depth information. First a short overview of different camera models and their geometry is reviewed. The second section covers the topic of optimization since almost all parts of the following chapters use optimization techniques. The last part discusses depth-map computation by stereo and depth-map fusion.

2.1 Camera Models

A fundamental part in computing 3D information from 2D images is the camera projection model. It defines how the light ray that goes through an image pixel projects out into the 3D world. The simplest model is the pinhole camera model that describes the camera with a simple projection. In practice, this model alone is not enough, since the actual camera lenses used in real cameras create a distortion of the simple projection.
Figure 2.1: The pinhole camera model with $C = (c_x, c_y)$ as projection center in the pixel coordinate frame and $f$ as focal length. A 3D point $P = (X, Y, Z)$ is projected onto the image plane at $p = (u, v)$ with $u = f \frac{X}{Z} + c_x, v = f \frac{Y}{Z} + c_y$.

2.1.1 Pinhole Camera

The pinhole camera model is the simplest model for camera projection and for standard field of view lenses it is works well if the lens distortion is modeled accordingly. Figure 2.1 shows the geometry of the pinhole camera. It simply projects a 3D point onto the image plane along the ray from the projection center through a pixel position in the image plane. The mapping between pixel values and ray direction can be represented in a $3 \times 3$ matrix $K$

$$K = \begin{bmatrix}
 f_x & 0 & c_x \\
 0 & f_y & c_y \\
 0 & 0 & 1
\end{bmatrix} \quad (2.1)$$

This matrix assumes that the image plane is set to $z = 1$. The projection of a 3D point onto the image plane is then simply $(u \ v) =$
2.1 Camera Models

Figure 2.2: Undistortion of typical lens effects. The image left shows the distorted image that the camera records, the pixels are warped relative to the projection center in such way that lines get straight again.

\[ K \begin{pmatrix} \frac{x}{v} \\ \frac{y}{v} \\ 1 \end{pmatrix} \text{.} \] To get the ray going through a pixel coordinate \( u, v \) its homogeneous representation is multiplied with \( K^{-1} \).

\[ \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \]

The simple pinhole model is not enough to model the camera image in general, all lenses introduce distortions to some extent, especially smaller and cheaper lenses. These distortions can be modeled as radial and tangential warps relative to a distortion center, that can be assumed on the projection center of the lens, see Figure 2.2.

The standard distortion model (i.e. \([4]\)) is defined as

\[
x_d = x_u (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2 p_1 x_u y_u + p_2 (r^2 + 2 x_u^2) \\
y_d = y_u (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1 (r^2 + 2 y_u^2) + p_2 x_u y_u
\]

where \( x_u \) and \( y_u \) are the pixel coordinates of the corrected image point and \( x_d \) and \( y_d \) are the original image point coordinates in the raw camera image. If these coordinates are represented in the \( z = 1 \) plane of the camera, the calibration coefficients \( k_1, k_2, k_3, p_1, p_2 \) are not dependent on the resolution and they can be used with any image resolution of a specific camera.
Chapter 2 Foundations

It might appear impractical to express the distortion coefficients in this way. But as will be explained in the following sections, during optimization of the location of a projection of a point in 3D will be expressed and compared to the measured position of that point. In this case the estimate of the projected point position is in undistorted image coordinates and we want to get its distorted location. If the whole image should be undistorted, a lookup-table is created to speed up this process. The lookup table contains for every undistorted pixel location the corresponding distorted pixel location where the pixel value has to be read from, so in this case the model is directly applicable as well.

2.1.2 Fish Eye Camera

For lenses that have a large field of view the introduced distortion gets too large to be correctly modeled by the combination of the pinhole and radial distortion. For these cases different fish eye camera models were proposed. One widely used model uses a polynomial function to model the deflection of the viewing rays [5]. This model can precisely describe the distortion but requires a rather large number of parameters. Another popular fish eye model, the so-called ATAN or FOV camera model, only requires one parameter [6] by modeling the distortion as a mapping from a spherical surface to a projection plane, see Figure 2.3.

The distortion function and the inverse of this model are

\[
\begin{align*}
    r_d &= \frac{1}{\omega} \arctan \left( 2r_u \tan \left( \frac{\omega}{2} \right) \right) \quad (2.4) \\
    r_u &= \frac{\tan (r_d \omega)}{2 \tan \left( \frac{\omega}{2} \right)} \quad (2.5)
\end{align*}
\]

where \( r_d \) and \( r_u \) are the distances of the image point to the distortion center and \( \omega \) is the distortion parameter of the camera model and can be seen as the virtual opening angle of the spherical lens (c.f. Figure 2.3). If the precision of the single parameter is not enough this model
2.1 Camera Models

Figure 2.3: The FOV or ATAN camera model. The distortion model assumes that the distance of the projection of a point to the principal point $r_d$ is roughly proportional to the angle between the optical axis and the projection ray. The undistorted projection $p_u$ of a 3D point $P$ is modeled as a mapping of the point on a sphere $p_d$ onto the projection plane at $z = 1$. 
can also extended with the radial distortion coefficients similar to the pinhole model.

2.2 Optimization

In general, a cost function $F : \mathbb{R}^n \rightarrow \mathbb{R}$ is defined which is then minimized over all or a set of its parameters $x$:

$$\min_{x \in \mathbb{R}^n} F(x)$$

(2.6)

In general, it cannot be guaranteed that a global minimum is found in a fixed number of iterations, therefore a local minimum near a starting position $x_0$ is tried to find instead, hoping this will be near the global minimum. In the following, different ways to minimize this cost function are discussed.

2.2.1 Non-linear Least Squares

For the case of structure from motion it is typically necessary to optimize over a large set of non linear functions, where we are looking for the least squares solution between some known values and their estimates. The problems arising in these cases can be expressed in the following way:

$$d_i = z_i - h_i(x)$$

(2.7)

$$f(x) = \sum_{i=0}^{n} d_i^2$$

(2.8)

Where $d_i$ is the residual between a measured data point $z_i$ and its estimated value $h_i(x)$. There are a number of iterative methods to minimize non linear functions in the least squares sense like Gradient Descent, Newton method or Gauss-Newton with possible improvements like variable trust-regions as in Levenberg-Marquart or Dogleg.
methods that are increased or decreased in size depending on the approximation quality of the cost function during optimization. The next two subsections will shortly summarize two of the methods used in this thesis.

**Gauss-Newton method**

The Gauss-Newton method requires that the cost function is twice differentiable and can only work on sum of squares. In return, it allows for a second order optimization without having to compute the second derivative of the cost function, the Hessian, which can be very challenging to compute in practice.

Least squares problems can also be weighted by a positive semi-definite matrix $\Lambda \in \mathbb{R}^{n \times n}$ depending on the estimated probability or accuracy of a measurement

$$f(x) = d(x)^\top \Lambda d(x)$$ (2.9)

where $d(x)$ is a vector of all $n$ stacked residuals $d_i$.

The Gauss-Newton method approaches the problem by approximating the Hessian of $f(x)$ by $J_d^\top \Lambda J_d$ where $J_d$ is the Jacobian of $d(x)$. This approximation is good if $d(x)$ is small, which is the case if the optimization happens near the true solution of $f(x)$. The least squares solution is then computed by computing an update step with the normal equations

$$(J_d^\top \Lambda J_d)\delta x = -J_d^\top \Lambda d$$ (2.10)

and adding the update step $\delta x$ to the current location $x$ of the solution and iterate until convergence.

In computer vision, measurements cannot only be noisy or less accurate in a defined way (for example when looking at images at a lower resolution), but there can be gross outliers, completely wrong measurements that happened due to wrong correspondence estimation. These outliers will typically introduce large residuals, which will
catastrophically disturb the least-squares solution. One way to minimize this possibility is to filter out the outliers, but since this cannot be done perfectly, it is very important to robustify the least squares solver for the case when outliers might slip through all filters. The standard way to handle outliers is to use robust weighting functions. These functions behave similarly to quadratic for small residuals but flat out for larger errors. Commonly used functions are the Huber, the Cauchy and the Tuckey (or Bisquare) robust weight functions:

\[
\begin{align*}
    w_{\text{Huber}} &= \begin{cases} 
    1 & |x| \leq k \\
    k/|x| & |x| > k 
    \end{cases} 
\end{align*}
\]
\[w_{\text{Cauchy}} = \frac{1}{1 + |x/c|^2}\]  
\[w_{\text{Tuckey}} = \begin{cases} 
    (1 - (x/c)^2)^2 & |x| \leq k \\
    0 & |x| > k 
    \end{cases} \]

Huber behaves exactly quadratic for errors \(x\) below the threshold \(k\) and increases linearly above the threshold, the biggest drawback is the discontinuity of the second derivative of the weight function at the threshold and its rather long tails where outliers will still influence the solution. The Cauchy function is continuous and has smaller tails than Huber but due to its descending first derivative, the Cauchy function can lead to wrong solutions that cannot be observed, the tuning factor \(c\) is chosen to be 2.3849 for 95% asymptotic efficiency. The Tuckey function is even more extreme in that it completely suppresses outlier measurements with a weight of 0. Here \(c = 4.6851\) for the 95% asymptotic efficiency. To overcome the issue of the discontinuity in the Huber weight function the so-called pseudo Huber function is defined that behaves similar to the Huber function otherwise:
2.2 Optimization

\[ w_{\text{Pseudo-Huber}} = \frac{1}{\sqrt{1 + (x/k)^2}} \quad (2.14) \]

\[ (2.15) \]

**Levenberg-Marquart method**

The Levenberg-Marquart method can be seen as an extension to the Gauss-Newton method, where a trust-region is used to interpolate between the Gauss-Newton and the Gradient Descent method since convergence is not guaranteed for Gauss-Newton, especially if the starting point is far away from the true solution. The Levenberg-Marquart algorithm, in general, will converge slightly slower but it will increase the possible convergence area. The normal equations are modified as follows to compute the update step:

\[ (J_d^\top \Lambda J_d + \lambda \text{diag}(J_d^\top \Lambda J_d))\delta = -J_d^\top \Lambda d \quad (2.16) \]

The so-called dampening factor \( \lambda \) is changed depending on how the cost function evolves, details about standard approaches can be found in [7]. \text{diag} creates a diagonal matrix from a vector, where the elements of the vector represent the diagonal of the matrix.

### 2.2.2 Optimization on Manifolds

In computer vision it is often necessary to optimize over parameters that lie on manifolds, for example rotations or rigid transformations. If the optimization approaches explained before are applied without special care, the solution is not necessarily on the manifold anymore. On the other hand, the accuracy and performance can be increased, if the optimization is done on the manifold itself. The standard way of achieving this is to work in the tangent space of the manifold, which itself is an Euclidian space with a mapping back onto the manifold, the exponential map. The update step is computed in the minimal representation in the tangent space of the transformation and is then
applied on the manifold by using the exponential map. A good explanation of the process can be found for example in [8].

As an example, a rotation in 3D belongs to the Special Orthogonal group $SO(3)$ that describes the group of orthogonal matrices whose determinant is 1 in three dimensions.

\[ y = R(x)p \quad (2.17) \]

This function describes a rotation of a vector $p \in \mathbb{R}^3$ by $R \in SO(3)$. $R$ is defined by its parameters $x$ in its tangent space. There are now two ways of defining the optimization: The first option is to use the parameters $x$, which requires to compute the Logarithmic map (the inverse mapping of the exponential map, see [8]) of $R$ to get the starting position $x_0$ for the optimization. But this is typically not necessary since the cost function often only contains expressions on the manifold only, in other words the cost function contains only $R$ and not its parameters $x$. This leads to the second option: since the optimization estimates at every step the correction $\delta x$ for the rotation that minimizes the cost function, it is not necessary to actually compute the Logarithmic map, but we define the update to start at 0. This way, the derivative of the cost function contains only the derivative of the Exponential map. The Exponential map itself is defined by $R(x) = \exp(x^\wedge)$ where $\wedge$ is the hat-operator mapping from the Euclidean tangent space to the Lie-algebra [9]. For the case of rotations the hat operator is

\[ \wedge : \mathbb{R}^3 \to so(3), x^\wedge = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} := [x]_x \quad (2.18) \]

where $so(3)$ is the Lie-algebra of $SO(3)$.

The exponential map $\exp([x]_x)$ is computed with the Rodriguez-Formula:
2.2 Optimization

\[
\exp([x]_\times) = \begin{cases} 
I + [x]_\times + \frac{1}{2} [x]_\times^2 = I & \text{for } (\theta \to 0) \\
I + \frac{\sin(\theta)}{\theta} [x]_\times + \frac{1 - \cos(\theta)}{\theta^2} [x]_\times^2 & \text{else}
\end{cases}
\text{ with } \theta = \|x\| 
\] 

(2.19)

And the derivative of Function (2.17) is given by

\[
\frac{\partial y}{\partial x} = -[Rp]_\times
\]

(2.20)

As can be seen, the Logarithmic map can be avoided completely, the next subsection gives another more complete example.

2.2.3 Examples

This subsection gives two practical examples of the described methods that are used in the thesis.

Camera pose optimization

One of the simplest cases is the optimization of the orientation and position of a camera observing points in 3D whose positions are known. In this case the projections of the real 3D points are measured in some way by for example extracting features and matching them or by tracking them between images.

\[
h_i(x) = \pi(T(x)P_i) 
\]

(2.21)

\[
d_i = z_i - h_i(x) 
\]

(2.22)

where \( h_i \) is the estimate of the projection of a 3D point \( P_i \) with the projection function \( \pi \), which is defined by the used camera model. \( T \) represents the orientation and position of the camera in 3D space represented as an element in the Special Euclidian group \( SE(3) \) that represents a rigid transformation in 3D by a rotation and a translation as a 6-dimensional vector \( x \).
The next step is to compute the Jacobian of \( h(x) \).

\[
P_c = T(x)P
\]

\[
\frac{\partial h(x)}{\partial x} = \frac{\partial \pi}{\partial P_c} \frac{\partial P_c}{\partial x}
\]

(2.23)

(2.24)

Here, the chain rule was used to separate the derivative into two parts, the derivation of the projection given a 3D point in the camera coordinate frame and the derivative of the transformation from world to camera frame. This separation is also useful since the projection is only dependent on the used camera model and the transformation depends only on the parametrization used for the camera poses (or in the case of bundle adjustment also on the parametrization of the map points).

\[
\frac{\partial P_c}{\partial x} = \frac{\partial \exp(x^\wedge)}{\partial x} = \begin{bmatrix} I & [P_c]_x \end{bmatrix}
\]

(2.25)

With the Jacobian it is possible to use the Gauss-Newton method to minimize the reprojection error with additional weighting with a robust cost function \( c \)

\[
f(x) = \sum_{i=0}^{n} c(d_i)d_i^2 = d^\top \Lambda d
\]

(2.26)

by iterating the algorithm with the update at iteration \( k \)

\[
(J_d^\top \Lambda J_d)\delta x = -J_d^\top \Lambda d
\]

(2.27)

\[
T_{k+1} = \exp(\delta x^\wedge) \cdot T_k
\]

(2.28)

until convergence.
### Bundle Adjustment

Bundle adjustment describes the case when not only camera poses are optimized but also the 3D points that have been observed from these poses. This typically leads to a very large system of equations that requires special care to stay computationally tractable [7]. One of the many approaches that try to keep the computational effort reasonable is to keep the Jacobians with respect to the camera poses and point locations separated by ordering the parameters with all camera parameters $a$ first and then all point parameters $b$: $p = (a^\top, b^\top)^\top$. This leads to a Jacobian with two blocks:

$$J = [ A \mid B ]$$

(2.29)

$$A = \frac{\partial p}{\partial a}$$

(2.30)

$$B = \frac{\partial p}{\partial b}$$

(2.31)

The benefit arises from the fact that there are typically many more point parameters than camera parameters, to compute the update step with Levenberg-Marquart, the following expression is formed:

$$\begin{bmatrix} U^* & W \\ W^\top & V^* \end{bmatrix} \begin{pmatrix} \delta a \\ \delta b \end{pmatrix} = \begin{pmatrix} \epsilon_a \\ \epsilon_b \end{pmatrix}$$

(2.32)

where $U^* = A^\top \Lambda A(1 + \lambda)$, $V^* = B^\top \Lambda B(1 + \lambda)$, $W = A^\top \Lambda B$, $\epsilon_a = -A^\top \Lambda^\top a$ and $\epsilon_b = -B^\top \Lambda^\top b$.

This expression can be solved in two steps, first

$$(U^* - WV^{*-1}W^\top)\delta a = \epsilon_a - WV^{*-1}\epsilon_b$$

(2.33)

is used to find the solution for $\epsilon_a$ and then

$$V^*\delta b = \epsilon_b - W^\top \epsilon_a$$

(2.34)
gives the solution to $\epsilon_B$. In the appendix of [7] more optimizations can be found that take care of the sparsity of the problem.

There exist several software libraries implementing solvers for non linear least squares problems with special algorithms that can take advantage of the structure of the bundle adjustment problems and use state of the art optimizations like Ceres-Solver [10], g2o [11] or GTSAM [12]. The latter two libraries offer a solver for factor graphs that are a generalization of the bundle adjustment problem. In the next section filtering techniques for visual odometry are discussed and possible combinations of both approaches (bundle adjustment and filtering) are sketched.

### 2.2.4 Filtering

In the previous section bundle adjustment was discussed, where the goal was to batch process all available information to achieve the best global solution. In the case of visual odometry, where the goal is to estimate the camera motion only without the need for a global map of point locations, different approaches are available. One class of the proposed approaches are filtering-based methods. The difference to the batch based approaches is that there is only one set of parameters for the current camera pose (or a small window of past poses in the case of a smoother). The rest of the information from the observations is marginalized out after one filter update step and kept as prior information in a covariance matrix. Different approaches are discussed in the related work section of Chapter 4.

**EKF-based visual inertial odometry**

Many approaches to filtering based Visual Odometry use the Extended Kalman Filter (EKF) framework. A Kalman filter recursively estimates the parameters of its state-space over time with noisy measurements. A motion model defines the propagation of the state space over time with some uncertainty, if measurements are available at
2.2 Optimization

some point in time a correction is computed by using a weighted average giving more weight to measurements with higher certainty.

One of the reasons to use an EKF-based approach for Visual Inertial Odometry (VIO) is the simplicity of the framework for implementation, even though the drawback is the bad scaling in terms of computational requirements in the number of filter states that limit the feasible number of estimated states. To tackle these issues solutions that reduce the number of necessary states have been proposed, see discussion of related work in Chapter 4.

**Batch-based inertial optimization**

Strasdat et al [13] investigated the accuracy and convergence properties of batch and filter based approaches and concluded that for the case of structure from motion batch based approaches have better characteristics. But if inertial measurements should be included into the estimation, special care need to be taken how they are included into the problem. The standard approach was to include relative pose estimates from an external source into the optimization problem and optimize for the map while staying near these prior poses. Very recently, solutions to efficiently solve the visual inertial odometry problem as batch-based problem were proposed [14, 15, 16] that use an efficient relative representation for the IMU measurements to speed up the optimization process, combined with a very efficient incremental solver for structure from motion based approaches [17] it can be shown that the inclusion of IMU measurement can be done without losing performance also in batch-based approaches. The only drawback here is that these optimization methods inherently work on keyframes, there needs to be a front-end that provides these keyframes. Here filtering-based approaches are still a good choice, since they inherently use the inertial data to help predicting the camera motion and make the frame-to-frame tracking more robust. This is demonstrated in different methods like [18] where a filtering front-end is feeding a mapping system with keyframes. The main difficulty here is on how to get the corrected probability estimates from the mapping back-end
back into the front-end filter. If this is not done properly the filter estimates can become inconsistent. A way to keep both the front and back-end using consistent estimates was proposed in [19] where both the back-end and the real-time filtering front-end work on the same Bayesian model in parallel and thus produce an globally optimal solution.

2.3 Stereo Depth Estimation

This section gives an short introduction into stereo depth estimation. This is a method that falls into the topic of image-based modeling which covers methods that compute the surface geometry of an observed object as complete as possible with only 2D images from known camera poses. There are many different approaches to extract the surface information, referred to as shape-from-X, where the X describes which visual cue is used. Some examples: Shape from silhouettes [20], Shape from stereo [21], Shape from texture [22], Shape from shading [23], Shape from focus [24].

2.3.1 Stereo

Looking at the methods for shape from stereo, there are numerous different approaches. This is a long evolved and broad field of research.
2.3 Stereo Depth Estimation

with more methods that can be covered in this section. Most methods focus on the highest possible accuracy or appealing appearance of the reconstruction. In this thesis, the goal is to allow for as fast as possible processing on a computing platform that has very limited computational power. One of the bottlenecks is directly at the beginning of the processing pipeline, the depth map estimation from stereo. The task is to estimate how far away the surface that projects into the corresponding pixel is from the camera center, see Figure 2.4. A straightforward approach is to project a ray from one of the camera through the pixel in question and search along the projection of that ray in the other image and select the pixel location where the pixel neighborhood (pixel patch) fits best to the neighborhood in the other image based on the intensity or color of the pixels, see Figure 2.5.

For the matching of the intensities or colors of the pixels in the patches different cost functions can be used. Often used methods are Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), Normalized Cross Correlation (NCC), see Equations (2.35)-(2.37). Another class of comparison functions are binary approaches like the Census Transform, where the pixel intensity is compared to the intensities of its neighbors and if the neighbor is darker the comparison results a 1 and 0 otherwise, the comparison score consists of the Hamming distance of two of these binary strings [27].

\[
SAD(x,y) = \sum_{p=-n/2}^{n/2} \sum_{q=-n/2}^{n/2} (I^2_{x+p,y+q} - I^1_{x+p,y+q}) 
\tag{2.35}
\]

\[
SSD(x,y) = \sum_{p=-n/2}^{n/2} \sum_{q=-n/2}^{n/2} (I^2_{x+p,y+q} - I^1_{x+p,y+q})^2 
\tag{2.36}
\]

\[
NCC(x,y) = \sum_{x,y \in W} \left( \frac{I^2_{x,y} \cdot I^1_{x,y}}{\sqrt{\sum (I^1_{x,y} - \bar{I}^1_{x,y})^2 \sum (I^2_{x,y} - \bar{I}^2_{x,y})^2}} \right) 
\tag{2.37}
\]

where \( I^1_{x,y} \) is the image intensity of image \( I^1 \) at pixel location \( x, y \)
Figure 2.5: Simple Stereo estimation, assuming two known camera poses and their respective images of a scene $I^1$ and $I^2$, for every pixel in image $I^1$ the ray $r_1$ going through a pixel is projected into image $I^2$. Along this ray $e_1$, at different locations the neighborhood pixels in image $I^2$ are compared to the neighborhood pixels in $I^1$. The best matching location on the ray in $I^2$ is selected as the matching pixel location and the 3D point $P$ can be triangulated by intersecting both rays $r_1$ and $r_2$ going through the respective pixels and thus its depth in both images can be computed.
and \( T_{x+p,y+q} \) is the mean intensity of the \( n \times n \) pixel patch \( W \) around pixel location \( x, y \).

SAD is very fast since it sums over integer differences but it is very sensitive to brightness changes. The sensitivity can be reduced by subtracting the mean difference of each patch from the sum, which compensates for a brightness change between the images. In this case its called Zero Mean SAD (ZSAD). SSD is performing slightly better but is also more expensive. Often NCC gives best results but is also the most expensive of the mentioned methods, the zero mean approach can also be applied to SSD and NCC. A good comparison of the different cost functions can be found in [28].

Even when using a very efficient patch comparison function, the computational effort is very high if the image resolution increases. If in addition the necessary patch comparisons per pixel are numerous since the depth range that needs to be covered is large, the depth map computation gets even more computationally expensive. The following subsections cover three different methods to increase efficiency of the matching, taking advantage of special hardware or randomize the matching.

**Pyramidal Stereo**

The main idea behind the pyramidal approach is to only run the exhaustive stereo computation for the depth map on a higher image pyramid level and then upsample the result level by level. During upsampling there exists a depth estimate for a window of \( 2 \times 2 \) pixels given the depth values of the neighboring pixels on the higher level. The depth value for the 4 pixels on the lower level can now be efficiently estimated by searching the range given by the higher image level depth values. The limited search range during upsampling reduces the number of mismatches and can also be seen as a guided matching approach that reduces the necessary computational effort by a large part. In this thesis the pyramidal stereo approach was used. More details on the overall system can be found in Chapter 3.
Plane Sweep

A different approach that allows making use of graphics shader hardware is called plane sweep \cite{29}. The idea is to project both camera images onto a plane in 3D space which can be efficiently done by projective texture mapping. If the plane is located at the same depth as the recorded scene area, the pixels from both images should be similar. The consistency check can be done with one of the comparison functions described in the section before. The plane is then moved (swept) to cover the desired depth range.

To efficiently handle the comparison computation of a patch around each pixel, the built-in box-filter mipmap generation functions of the graphics hardware can be used to very efficiently sum up window sizes of $2^n \times 2^n$. Starting with an image $I^0$ the following function is applied:

$$I_{x,y}^{i+1} = \frac{1}{4} \sum_{q=2x}^{2x+1} \sum_{p=2y}^{2y+1} I_{p,q}^i$$  \hspace{1cm} (2.38)

This only allows to compute the SSD score every $2^n \times 2^n$ pixel, but interpolation can be used to compute the values in between. Another way to sum pixel values is to use bilinear filtering and reading the value at the middle of 4 pixels with allows to compute the sum of a $2 \times 2$ window. Best results where shown when using a multi-level approach and trilinear interpolation combine the comparison results on different levels \cite{30}.

PatchMatch Stereo

PatchMatch Stereo transfers the idea of PatchMatch \cite{31} in structural image editing to stereo matching \cite{32}. The main idea here is to randomly initialize the depth map and then compare the matching cost of a plane at a certain depth value of the current pixel to the values of the neighboring pixels and take the plane with the lowest cost. After this propagation step a random plane value is generated and compared to the current plane, if the random plane matches better, it is taken in turn as the value for this pixel. Instead of only randomly
2.3 Stereo Depth Estimation

Figure 2.6: PatchMatch Stereo, image from [32]. (a) shows the algorithm in the middle of the propagation state of the right image. The lower part has still randomly initialized plane estimates per pixel. The red pixel is being processed, its random estimate is compared to the neighboring pixels (green arrows), the left image estimate (yellow) and if a video sequence is processed, also the previous estimate in time (blue). (b) is the result after 3 iterations. (c) shows the final result after post-processing.

Trying new planes, optimization techniques could be used to help to converge to the true value. Using the planes instead of just depth values allows to better handle slanted surfaces. The overview of the algorithm is shown in Figure 2.6.

The algorithm in [32] was implemented on a CPU and is too slow for real-time applications, especially on mobile hardware. An optimized version was proposed for a webcam based implementation allowing for real-time processing in [33], where no plane normals are estimated anymore, but only depths and to slightly increase accuracy the authors propose to use ZNCC as matching score instead of SAD. The resulting depth maps are worse in quality than with the original algorithm but since multiple depth maps are fused together, the lower quality effects are reduced in the final result.
Figure 2.7: Volumetric depth fusion, image from [34]. (a) and (b) show the truncated signed distance functions in the volume from two depth maps, the brown color shows empty space without distance values. (c) shows the fused result with the isosurface where the estimated surface will be extracted.

2.3.2 Depth Map Fusion

After getting a number of depth maps from different view pairs (or more images), the next step in getting a high quality reconstruction of the surface is to take all depth maps and fuse them together. This process can directly be included in the stereo depth map estimation or can be done as a separate step.

Volumetric Fusion

A very popular approach is to use a volumetric representation of the space where the surface lies in and store the distance to the surface in a 3D field of the volume [34]. Distances in front of the surface are negative and the ones behind are positive, the surface can then be extracted as the zero-isosurface from the volume, see Figure 2.7.

Well known implementations of volumetric fusion approaches are DTAM [35], KinectFusion [36, 37] and also the system described in [33] uses the KinectFusion backend to fuse the depth maps. One drawback of volumetric approaches is the large memory requirement, which can
be reduced by applying adaptive spatial data structures like octrees. Recently, voxel hashing has been proposed to finally overcome the issue of the high memory usage, allowing for streaming the required octree blocks into memory and freeing up the blocks that are not needed at a certain time [38]. In addition to high memory usage the volumetric fusion was mainly demonstrated on high performance graphics hardware in real-time which makes its application on mobile platforms difficult.

On the other hand, these methods offer a straightforward possibility to track the reconstructed scene in a dense manner by minimizing the photometric error of all pixels of the already reconstructed surface projected into the current camera view. This approach is robust and typically drifts less than methods based on sparse features.

**Point-Cloud Fusion**

In addition to volumetric fusion approaches, there are point cloud based methods that represent the depth values as 3D points and fuse the different depth maps by updating the 3D points. The 3D points can also be represented as surfels, as conceptually very small surface patch parts with a normal and possibly also color. Depending on the approach, the 3D points are directly updated [39] or the fusion happens by reprojecting the point cloud into the current view and updating it in 2D by using the pixels of the depth map. The latter approach with surfel-based fusion was used in this thesis and a detailed description is found in Chapter 5.
Chapter 3

3D Reconstruction on Mobile Phones

The flexible and accurate generation of 3D models of real-world environments has been a long-term goal in computer vision. Research efforts on 3D content creation from still images has reached a certain level of maturity and has emerged to popular industrial solutions like Autodesk’s 123D Catch. While high-quality 3D models can be obtained with such systems, the generation of an image set, which ensures the desired accuracy of the subsequently obtained 3D model, is a more challenging task. Camera sensor noise, occlusions and complex reflectance of the scene often lead to failure in the reconstruction process but their appearance is difficult to predict in advance. This problem is addressed by monocular real-time capable systems which can provide useful feedback to the user in the course of the reconstruction process and assist them in planning their movements. Impressive results were obtained with interactive systems using video cameras [35] and depth sensors [40, 36]. However, those systems require massive processing resources like multi-core CPUs and powerful GPUs. As a result, their usability is limited to desktop computers and high-end
Figure 3.1: Live 3D reconstruction in a museum. The smartphone display shows the dense 3D model registered to the object. Full results in Fig. 3.9.

laptops, which precludes applications of casual capture of 3D models in the wild. Moreover, the produced 3D models are determined only up to an overall scale and are not provided in metric coordinates for video camera based systems. This burdens their applicability in areas where absolute physical measurements are needed.

In the last few years, remarkable progress was made with mobile consumer devices. Modern smartphones and tablet computers offer multi-core processors and graphics processing cores which open up new application possibilities. Additionally, they are equipped with micro-electrical sensors, capable of measuring angular velocity and linear acceleration. The design of methods for live 3D reconstruction able to make use of those developments seems a natural step.

But still, the computing capabilities are still far from those of desk-
top computers. To a great extent, these restrictions render most of the currently known approaches inapplicable on mobile devices, giving room to research in the direction of specially designed, efficient on-line algorithms to tackle all the limitations of embedded hardware architectures. While first attempts for interactive 3D reconstruction on smartphones have already been presented [41, 42], their applicability is limited and their performance is still far from that of desktop systems.

In this chapter, we will present a fully on-device, markerless 3D modelling system that automatically takes images when the camera is held still and estimates the travelled distance between these motion segments, allowing estimation of the absolute scale. The dense reconstruction scheme is optimized to run at keyframe rate on current mobile phones.

3.1 Related Work

Our work is related to several fields in computer vision: visual inertial fusion, Simultaneous Localization And Mapping (SLAM) and image-based modeling.

Visual inertial fusion is a well established technique [43]. Lobo and Dias align depth maps of a stereo head using gravity as vertical reference in [44]. As their head is calibrated, they do not utilize linear acceleration to recover scale. Weiss et al.[45] developed a method to estimate the scaling factor between the inertial sensors (gyroscope and accelerometer) and a monocular SLAM approach, as well as the offsets between the IMU and the camera. Porzi et al.[46] demonstrated a stripped-down version of a camera pose tracking system on an Android phone where the inertial sensors are utilized only to obtain a gravity reference and frame-to-frame rotations.

Recently Li and Mourikis demonstrated impressive results on visual-inertial visual odometry, without reconstructing the environment [47].

Klein and Murray [48] proposed a system for real-time Parallel Tracking And Mapping (PTAM) which was demonstrated to work
well also on smartphones [49]. Thereby, the maintained 3D map is built from sparse point correspondences only. Newcombe et al. [35] perform tracking, mapping and dense reconstruction on a high-end GPU in real time on a commodity computer to create a dense model of a desktop setup. Their approach makes use of general purpose graphics processing but the required computational resources and the associated power consumption make it unsuitable for our domain.

As the proposed reconstruction pipeline is based on stereo to infer geometric structure, it is related to a myriad of works on binocular and multi-view stereo. We refer to the benchmarks in [50], [21] and [51] for a representative list. However, most of those methods are not applicable to our particular scenario as they don’t meet the underlying efficiency requirements. In the following, we will focus only on approaches which are conceptually closely related to ours.

Building upon previous work on reconstruction with a hand-held camera [52], Pollefeys et al. [53] presented a complete pipeline for real-time video-based 3D acquisition. The system was developed with focus on capturing large-scale urban scenes by means of multiple video cameras mounted on a vehicle. A method for real-time interactive 3D reconstruction was proposed by Stuehmer et al. [54]. Thereby, a 3D representation of the scene is obtained by estimating depth maps from multiple views and converting them to triangle meshes based on the respective connectivity. Even though these techniques cover our context, they are designed for high-end computers and are not functional on mobile devices due to some time-consuming optimization operations. Another approach for live video-based 3D reconstruction was proposed by Vogiatzis and Hernandez [39]. Here, the captured scene is represented by a point cloud where each generated 3D point is obtained as a probabilistic depth estimate by fusing measurements from different views. Similar to the already discussed methods, this one also requires substantial computational resources. Another key difference to our framework is the utilization of a marker to estimate camera poses, which entails considerable limitations in terms of usability.

Recently, the first works on live 3D reconstruction on mobile devices
3.2 System Overview

Our system consists of three main blocks: inertial tracking, visual pose estimation and dense 3D modeling, as depicted in Fig. 3.2. All three blocks operate asynchronous and thus allow us to optimally make use of the multi-core capabilities of the device. We take two main input streams: camera frames with resolution of 640 × 480 at tracking rates typically between 15-30 Hz and inertial sensor information (angular velocity and linear acceleration) at 200 and 100 Hz respectively. The inertial tracking module provides camera poses which are subsequently refined by the visual tracking module. The dense 3D modeling module is supplied with images and corresponding full calibration information at selected keyframes from the visual tracker as well as metric information about the captured scene from the inertial tracker. Its processing time is typically about 2-3 seconds per keyframe. The system is triggered automatically when the inertial estimator detects a salient motion with a minimal baseline. The final output is a 3D model in metric coordinates in form of a colored point cloud. All components of the system are explained in more detail in the following sections.
3.3 Visual Inertial Scale Estimation

Current smartphones are equipped with a 3D gyroscope and accelerometer, which produce (in contrast to larger inertial measurement units) substantial time-dependent and device-specific offsets, as well as significant noise. To estimate scale, we first need to estimate the current world to body/camera frame rotation $R_B$ and the current earth-fixed velocity and position using the inertial sensors. The estimation of this rotation is achieved through a standard Extended Kalman Filter. As the magnetometer and GPS are subject to large disturbances or even unavailable indoors as well as in many urban environments, we rely solely on the gyroscope and update the yaw angle with visual measurements $m_B$. We scale the gravity vector $g_B$ to the unit-length vector $z_B$ and estimate $y_B$ and $x_B$ using the additional heading information

$$r_{zB} = \frac{g_B}{\|g_B\|}, \quad r_{yB} = \frac{r_{zB} \times m_B}{\|r_{zB} \times m_B\|}, \quad r_{xB} = r_{yB} \times r_{zB},$$

with $R_B$ and dynamics given as

$$R_B = [r_{xB}, r_{yB}, r_{zB}] \in SO(3), \quad \dot{R}_B = \omega R.$$
3.3 Visual Inertial Scale Estimation

The filter prediction and update equations are given as

\[
\hat{R}_B = e^{\bar{\omega} dt} \bar{R}_B, \tag{3.3}
\]
\[
\bar{r}^+_i = \hat{r}_i + L_{ik} (z_i - \hat{r}_i) \quad \text{with } i \in \{x, y, z\}, \tag{3.4}
\]

where the Kalman gain matrix \(L_k\) is computed in every time step with the linearized system.

The camera and IMU are considered to be at the same location and with the same orientation. In the case of orientation, this is valid since both devices share the same PCB. As for the case of the location, this is a compromise between accuracy and simplicity. For the proposed framework, neglecting the displacement between sensors did not noticeably affect the results.

We initialize the required scale for visual-inertial fusion by first independently estimating motion segments. In order to deal with the noise and time-dependent bias from the accelerometer, an event-based outlier-rejection scheme is proposed. Whenever the accelerometer reports significant motion, we create a new displacement hypothesis \(\bar{x}\). This is immediately verified by checking a start and stop event in the motion. These are determined given that for sufficiently exciting handheld motion, the acceleration signal will exhibit two peaks of opposite sign and significant magnitude. A displacement is then estimated and compared to the displacement estimated by vision (\(\bar{y}\)) at the start and stop events, yielding a candidate scale. Due to visual or inertial estimation failures, outlier rejection is needed. Each new measurement pair is stored and the complete set is re-evaluated using the latest scale by considering a pair as inlier if \(\|\bar{x}_i - \lambda \bar{y}_i\|\) is below a threshold. If the new inlier set is bigger than the previous one, a new scale \(\lambda\) is computed in the least-squares sense using the new set \(I\) as

\[
\sum_{i \in I} \|\bar{x}_i - \lambda \bar{y}_i\|^2. \tag{3.5}
\]

Otherwise, the displacements are saved for future scale candidates.

As soon as the scale estimation converges, we can update the inertial position with visual measurements. In addition to providing an
estimate of the scene scale, we produce a filtered position estimation as show in Fig 3.3. This can be leveraged to process frames at lower rates or to mitigate intermediate visual tracking issues e.g. due to motion blur. Since the sample rate of the accelerometer is higher than the frame rate of the camera, we predict the position of the phone with each new accelerometer sample and update with the visual information whenever a new measurement is available. With every new IMU sample, the accelerometer data is rotated and the gravity is accounted for in the inertial frame. This acceleration is integrated using Velocity Verlet, which is in turn used for a decaying velocity model of handheld motion

\[
\vec{v}_f^{k+1} = \vec{v}_I^k + \tau \Delta t R_B (\vec{a}_B^k - g_B). \tag{3.6}
\]

Here \(\tau\) will account for timing and sensor inaccuracies (inherent of the operating system available in mobile phones) by providing a decaying velocity model, preventing unwanted drift at small accelerations (see Fig 3.3). To update with the visual data, the estimated velocity is first scaled to metric units using \(\lambda\) and then fused with the inertial prediction using a simple linear combination based on the variances of both estimates.

\[
\vec{x}_f = \kappa \left( \sigma_v^{-2} \lambda \vec{x}_v + \sigma_i^{-2} \vec{x}_i \right). \tag{3.7}
\]

Here the subscripts \(f, v\) and \(i\) denote fused, vision and inertial position estimates, respectively, and \(\kappa\) is the normalizing factor.

The visual updates become available with a time offset, so we need to re-propagate the predicted states from the point at which the vision measurement happened to the present [45]. This is done by storing the states in a buffer and, whenever vision arrives, looking back for the closest time-stamp in that buffer, updating, and then propagating forward to the current time.

Fig 3.4 shows the results of the combined vision and inertial fusion in a freehand 3D motion while tracking a tabletop scenario. It is evident that scale and absolute position are correctly estimated throughout the trajectory.
To evaluate the resulting scale accuracy of a reconstructed object a textured cylinder with known diameter was reconstructed. A cylinder was fitted into the reconstruction to measure the diameter. In a qualitative evaluation with multiple tries, the scale was estimated to have an error of up to 10-15% when working with objects of size of around 10-20cm. This is mostly due to the inaccuracy in the magnitude of the acceleration measured of the consumer-grade accelerometer in the device. This can be improved by an Extended Kalman Filter implementation that estimates the full camera pose by fusing the inertial data and the visual data in a tight manner and also allows to estimate all offsets and time delays. Such a filter implementation is discussed in Chapter 4.
Figure 3.3: Left: simple inertial prediction and decaying velocity vs ground truth. Right: visual-inertial estimate allows to partially reject tracking losses. Bottom: Convergence of scale estimation.
3.3 Visual Inertial Scale Estimation

Figure 3.4: Visual inertial pose estimate vs. ground truth from a VICON motion capturing system providing position information with 1mm accuracy.
Figure 3.5: Cylinder with texture used to evaluate the computed scale of the reconstruction.
3.4 Visual Tracking and Mapping

3.4.1 Two View Initialization

The map initialization in the system is a critical part since no undelayed pose estimation as in some Extended Kalman Filter based approaches [56] is used. Therefore, the system needs a reliable map from the beginning, if the initialization results in a suboptimal map the system can hardly recover from it. Two different initialization approaches were implemented: The first approach initializes the map from two keyframes. ORB features [57] are extracted from both frames and matched. Outliers are filtered out by using the 5-point algorithm in combination with RANSAC [58]. After that, relative pose optimization is performed and the point matches are triangulated. This approach sometimes gets bad initializations, especially if the user did not create enough baseline. To reduce the dependency on the user to apply enough baseline a second approach was designed to allow for dynamic initialization. The user taps on the screen to notify the system that he/she wants to start scanning now. The system extracts FAST corners [59] and tracks them with using the KLT feature tracking algorithm [60]. After every image the 5-point algorithm with refinement is applied, the points are triangulated and the median and mean angle of observation of all inlier features is checked. If the mean and median angle are above 5° the map is initialized as follows. In order to get a denser initial map, FAST corners are then extracted on four resolution levels and for every corner a 8x8 pixel patch at the respective level is stored as descriptor. The matching is done by comparing the zero-mean sum of squared differences (ZSSD) value between the pixel patches of the respective FAST corners along the epipolar line. To speed up the process, only the segment of the epipolar line is searched that matches the estimated scene depth from the already triangulated points. After the best match is found, the points are triangulated and included to the map which is subsequently refined with bundle adjustment. Since the gravity vector is known from the inertial estimator, the map is also rotated such that it matches the earth inertial frame.
3.4.2 Patch Tracking and Pose Refinement

The tracker is used to refine the pose estimate from the inertial pose estimator and to correct drift. The tracker has two main steps: a photometric pre-alignment and map based alignment. The first step follows the 6D semi-direct camera pose alignment approach described in [61]: 3D points that were used for pose estimation in the previous frame are projected into the current estimation camera image by using the estimated camera pose. Then the 6D camera pose is optimized by minimizing the photometric error of a unwarped $4 \times 4$ pixel patch around the estimated projected point location in the current image. This step already gives a very good estimate of the camera pose. To reduce drift and including more point measurements additional 3D points from the map are projected into the current camera frame. The matching is done by warping the $8 \times 8$ pixel patch of the map point onto the view of the current frame and computing the ZSSD score. It is assumed that its normal is oriented towards the camera that observed it for the first time. For computing the warp the appropriate pyramid level in the current view is selected. If the ZSSD score is below a threshold and the subsequent subpixel optimization converges, the projection is considered as a match. The matches are then optimized with a robust Levenberg-Marquart absolute pose estimator giving the new vision-based pose for the current frame. If for some reason the tracking is lost the small blurry image relocalization module from [48] is used. To further increase the change of successful relocalization, the camera pose can be aligned corresponding to the known gravity direction.

3.4.3 Sparse Mapping

New keyframes are added to the map if the user has moved the camera a certain amount or if the inertial position estimator detects that the phone is held still after salient motion. In either case, the keyframe is provided to the mapping thread that accepts the observations of the map points from the tracker and searches for new ones. To this end,
a list of candidates is created from non maximum suppressed FAST corners that have a Shi-Tomasi score [62] above a certain threshold. Another keyframe near the current one with similar orientation is selected to match the corners along the epipolar line, if the ZSSD score of the $8 \times 8$ pixel patch is below a threshold and the subsequent subpixel optimization converges, both corner locations are considered as a match and the point is triangulated and added to the map. To minimize the possibility that new points are created at positions where such already exist, a mask is created to indicate the already covered regions. No candidate is added to the map if its projection is inside a certain pixel radius. Since the typical scene consists of an object in the middle of the scene, only map points that were observed from an angle of 60 degrees or less relative to the current frame are added to this mask. This allows to capture both sides of the object but still reduces the number of duplicates.

Similar to [48], the mapper performs bundle adjustment optimization in the background. Its implementation is based on the method using the Schur complement trick that is described in [7]. After a keyframe is added, a local bundle adjustment step with the closest 4 keyframes is performed. With a reduced priority, the mapper optimizes the keyframes that are prepared for the dense modeling module. With lowest priority, the mapping thread starts global bundle adjustment optimization based on all frames and map points. This process is interrupted if new keyframes arrive. To remove outliers, observations that were given low robust cost function weights during bundle adjustment are removed, if a map point has only two observations after several added keyframes it is considered as an outlier and removed.

3.5 Dense 3D Modeling

At the core of the 3D modeling module is a stereo-based reconstruction pipeline. In particular, it is composed of image mask estimation, depth map computation and depth map filtering. In the following, each of these steps is discussed in more detail. Finally, the filtered depth map
is back projected to 3D, colored with respect to the reference image and merged with the current point cloud.

### 3.5.1 Image Mask Estimation

The task of the maintained image mask is twofold. First, it identifies pixels exhibiting sufficient material texture. This allows to avoid unnecessary computations which have no or negligible effect on the final 3D model and reduces potential noise. Second, it overcomes the generation of redundant points by excluding regions already covered by the current point cloud.

In particular, for an input color image we consider the structure tensor of its grayscale version \( I : \Omega \subset \mathbb{Z}^2 \rightarrow \mathbb{R} \) at pixel \((x_0, y_0)\)

\[
A(x_0, y_0) = \sum_{W(x_0, y_0)} \begin{pmatrix}
\frac{\partial I^2}{\partial x} & \frac{\partial I}{\partial x} \frac{\partial I}{\partial y} \\
\frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \frac{\partial I^2}{\partial y}
\end{pmatrix},
\]

(3.8)

where \( W(x_0, y_0) \subset \Omega \) denotes a window centered at \((x_0, y_0)\). Alternatively, the image could be smoothed in a pre-processing step to get rid of spurious texture response. However, this step is unnecessary in the constructed framework as pixels affected by noise are unlikely to give a consistent texture score over multiple images. An established criterion to assess the degree of texturedness of the neighborhood around \((x_0, y_0)\) is to analyze the eigenvalues \( \lambda_1, \lambda_2 \) of \( W(x_0, y_0) \). Here, we adopt the Shi-Tomasi measure which is given by \( \min\{\lambda_1, \lambda_2\} \) (see [62]). The respective texture-based image mask is obtained by thresholding the values at some \( \lambda_{\min} > 0 \). In our implementation, we set \( \lambda_{\min} = 0.1 \) and use windows of size \( 3 \times 3 \) pixels.

Additionally, another mask is estimated based on the coverage of the current point cloud. To this end, a sliding window, which contains a set of the recently included 3D points, is maintained. All points are projected onto the current image and a simple photometric criterion is evaluated. Note that points, that belong to parts of the scene not visible in the current view, are unlikely to have erroneous contribution.
to the computed coverage mask. The final image mask is obtained by fusing the estimated texture and coverage mask. Subsequent depth map computations are restricted to pixels within the mask.

3.5.2 Depth Map Computation

The depth map computation is the bottleneck of the dense 3D modeling module and requires special efforts. The requirement here is to design a method which is efficient enough to address the underlying hardware limitations as well as accurate enough to deliver useful feedback to the user in the course of the reconstruction.

Despite the rapid progress in mobile technologies, modern smart phones and tablets still suffer from some important hardware limitations which pose major challenges to the implementation of computationally intensive operations like dense stereo. Compared to commodity desktop computers and laptops, the most notable ones are slow memory access and low computational power of the GPU in terms of both speed and precision. As a result, the performance reduction of a demanding application by the transition from a desktop computer to a mobile device could be in the range of multiple orders of magnitude. We address these difficulties at both algorithmic and hardware level.

**Multi-resolution scheme.** We run binocular stereo by taking an incoming image as a reference view and matching it with an appropriate recent image in the provided series of keyframes. Instead of applying a classical technique based on estimating the optimal similarity score along respective epipolar lines, we adopt a multi-resolution scheme.

The proposed approach involves downsampling the input images, estimating depths, and subsequently upgrading and refining the results by restricting computations to a suitable pixel-dependent range.

Similar to the visual tracking stage (see Section 3.4), we rely on computations at multiple pyramid resolutions. At each level $i \in \{0, \ldots, L\}$, respective images are obtained by halving the resolution of their versions at level $i - 1$ in each dimension. Thereby, $i = 0$ contains the original images. Starting at the top of the pyramid, the multi-
### Table 3.1: Update scheme for computing $D_i(x, y)$, $(x, y) \in \Omega_i$. The notation $(x', y') := (\lfloor x/2 \rfloor, \lfloor y/2 \rfloor) \in \Omega_{i+1}$ is used. The respective depth range is given by the minimum and maximum value within the considered neighborhood.

<table>
<thead>
<tr>
<th>case</th>
<th>neighborhood</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \mod 2 = 1$</td>
<td>$D_{i+1}(x', y'), D_{i+1}(x' + 1, y')$, $D_{i+1}(x' + 1, y' + 1), D_{i+1}(x', y' + 1)$</td>
</tr>
<tr>
<td>$y \mod 2 = 1$</td>
<td>$D_{i+1}(x', y'), D_{i+1}(x' + 1, y')$, $D_{i+1}(x' + 1, y' - 1), D_{i+1}(x', y' - 1)$</td>
</tr>
<tr>
<td>$x \mod 2 = 1$</td>
<td>$D_{i+1}(x', y'), D_{i+1}(x', y' + 1)$, $D_{i+1}(x' - 1, y' + 1), D_{i+1}(x' - 1, y')$</td>
</tr>
<tr>
<td>$y \mod 2 = 0$</td>
<td>$D_{i+1}(x', y'), D_{i+1}(x', y' - 1)$, $D_{i+1}(x' - 1, y' - 1), D_{i+1}(x' - 1, y')$</td>
</tr>
</tbody>
</table>
3.5 Dense 3D Modeling

A resolution approach estimates a depth map \( D_i : \Omega_i \subset \mathbb{Z}^2 \rightarrow R \subset \mathbb{R} \) to each level \( i \) based on the image data at that level and the depth map from the consecutive level \( D_{i+1} \). While exhaustive computations have to be performed for the highest level \( L \), subsequent computational efforts can be reduced significantly by exploiting the previously obtained coarse result. In particular, we apply an update scheme based on the current downsampled pixel position and three appropriate neighbors, see Table 3.1.

We estimate the depth \( D_i(x, y) \) by searching an appropriate range given by the minimum and maximum value in \( \{ D_{i+1}^l | l = 0, \ldots, 3 \} \), where \( l \) denotes the image pyramid level. Thereby, depth values, that are not available due to boundary constraints or the maintained image mask, are omitted. In order to take into account uncertainties due to the coarse sampling at higher pyramid levels, we additionally include a small tolerance in the estimated ranges. As the uncertainty is expected to increase with increasing depth due to the larger jumps of the values from pixel to pixel, we use a tolerance parameter which is inversely proportional to the local depth \( D_{i+1}^0 \).

In our implementation, we used images of size \( 640 \times 480 \) pixels and 3 resolution levels (i.e. \( L = 2 \)). It should be noted that all estimated depth maps rely on a predefined range \( R \subset \mathbb{R} \) which can be determined by analyzing the distribution of the sparse map constructed in the camera tracking module (see Section 3.4).

The multi-resolution scheme comes along with some important advantages. First, it entails significant efficiency benefits compared to traditional methods as epipolar line traversals at higher image resolutions are restricted to short segments. Second, when applying a winner-takes-all strategy, potential mismatches can be avoided due to the more robust depth estimates at low image resolution. Third, regions in the image, which belong to distant scene parts outside of the range of interest, can be discarded at the lowest resolution level and subsequent refinement operations can be avoided for them. To illustrate these aspects, we present a comparison between the classical single-resolution winner-takes-all strategy and the developed multi-resolution technique on an example image pair (see Fig. 3.6). While
Figure 3.6: Single- vs. multi-resolution depth map estimation. *From left to right:* The reference image of a stereo pair, corresponding depth map estimated with a classical single-resolution winner-takes-all strategy and result obtained with the proposed multi-resolution scheme. Both solutions result in dense depth maps but at closer look it can be seen that the pyramidal multi-resolution approach can reduce the number of gross outlier depths.
both results look genuinely similar, a closer look reveals that the proposed multi-resolution scheme confers a higher degree of robustness by producing less outliers. Note that for the above example a conservative depth range was used so as to capture the entire field of view. However, the real benefit from it becomes evident when comparing the respective runtimes. In practice, the multi-resolution approach is about 5 times faster than the single-resolution counterpart.

**GPU acceleration.** Despite the utilization of a multi-resolution scheme, the developed method for dense stereo is not efficient enough to meet the requirements of the application at hand. For this reason, we made use of the parallelization potential of the algorithm with a GPU implementation (based on GLSL ES), which reduces the overall runtime of the 3D modeling module to about 2-3 seconds per processed image on a Samsung Galaxy S3. More concretely, we estimate depth maps at different pyramid levels in separate rendering passes. Thereby, some care should be taken due to the precision limitations of current mobile GPUs. We address this difficulty by using the Sum of Absolute Differences (SAD) as a similarity measure in the matching process (over $5 \times 5$ image patches) and transferring triangulation operations to get the final depth estimates to the CPU.

**Image pair selection.** A crucial step in binocular stereo is the choice of an appropriate image pair. An ideal candidate pair should share a large common field of view, a small but not too small baseline and similar orientations. As we have an ordered image sequence, a straightforward methodology would be to match each incoming image with its predecessor. Yet, this strategy is suboptimal in some cases, for example when the user decides to move back and recapture certain parts of the scene. Instead, we propose to maintain a sliding window containing the last $N_v$ provided keyframes ($N_v = 5$ in our implementation) and pick the one maximizing a suitable criterion for matching with the current view. For two cameras $j$ and $k$ this criterion is defined as

$$C(j, k) = \cos \theta_{\text{pose}}^{jk} \cdot \cos \theta_{\text{view}}^{jk} \cdot \cos \theta_{\text{up}}^{jk},$$

where $\theta_{\text{pose}}^{jk}$ denotes the angle between the viewing rays of both cam-
eras at the midpoint of the line segment connecting the mean depth range points along the camera principal rays, \( \theta_{jk} \) is the angle between the principal rays and \( \theta_{up} \) is the angle between the up vectors of both cameras. Additionally, we impose the following constraints

\[
5^\circ \leq \theta_{jk \text{ pose}} \leq 45^\circ, 0^\circ \leq \theta_{jk \text{ view}} \leq 45^\circ, 0^\circ \leq \theta_{jk \text{ up}} \leq 30^\circ
\]

An input image is discarded and not processed if none of the images in the current sliding window satisfy those constraints with respect to it.

### 3.5.3 Depth Map Filtering

The final step in the proposed 3D modeling pipeline consists in filtering the estimated depth map. The applied procedure is inspired by [63] and is based on checking consistency over multiple views. In particular, a sliding window containing the last \( N_d \) depth maps is maintained. The depth value at each pixel of the current map is tested on agreement with the maps in the sliding window, i.e. it is warped to the corresponding views and compared with the values stored there. The depth is considered consistent if the estimated difference is within a certain tolerance for at least \( N_c \) views. In our implementation, we set \( N_d = 5 \) and \( N_c = 2 \). It is important to note that the unfiltered depth maps have to be maintained here because parts of the scene, not visible in the views included in the current sliding window, would never get the chance to be reconstructed otherwise. This simple but very powerful filtering procedure is already able to remove most outliers and build a clean 3D model. In Chapter 5 a method for fusing the depth maps in an efficient manner is described further improving the obtainable results.

### 3.6 Experimental Results

All experiments were conducted on a Samsung Galaxy SIII I9300GT with Samsung Exynos 4 quad core CPU and ARM Mali-400 MP4
GPU and processed fully on-device during the capturing. The visual tracking was running in real-time and the dense processing was running at keyframe-frequency of roughly 2-3 seconds per keyframe, which limits the speed of scanning to a speed where the camera is moved in a way that there is not too much motion blur and the device can be hold without too much shaking.

The first set represents a typical use case for mobile 3D reconstruction: non-movable objects for which no 3D geometry exists yet. To this end, multiple objects were captured from the collection of a museum. Fig. 3.8 shows the generated model of a tribal mask which was created interactively on-device during the normal opening hours of the museum. Fig. 3.9 depicts an additional object from the same collection: an ancient Shakyaamuni Buddha statue. Both results were obtained under normal exhibition conditions. They illustrate the suitability for a number of practical cases where 3D objects cannot be simply taken to a full-fledged 3D scanning device.

To evaluate the generality of our approach, we complete the evaluation with two additional scenarios that were not envisioned initially: outdoor environments (see Fig. 3.10) and human faces (see Fig. 3.11).
Figure 3.7: Photo and front and left views of the reconstructed 3D model of a 0.6 m tall sitting Buddha statue.
Figure 3.8: Photo and front and right view of the reconstructed 3D model of a 0.5 m tall African tribal mask.
Figure 3.9: Photo and front and left views of reconstructed 3D model of a 1.6 m tall Shakyamuni Buddha statue.
3.6 Experimental Results

Figure 3.10: Photo and reconstructed 3D model of a building facade captured at street-level.
Figure 3.11: Front and left view of a reconstructed 3D model of a human face, including a corresponding photo of the test person.
3.7 Conclusion

We presented the first interactive on-device system for dense stereo-based 3D reconstruction on mobile phones. In order to address the major challenges posed by the underlying hardware limitations and to meet the robustness and efficiency requirements of the application, we integrated multiple novel solutions. In particular, we explored the capabilities of the inertial sensors available on modern mobile devices to improve the resilience of the camera tracking process to rapid motions and to automatically capture keyframes when the phone is static. Additionally, we showed how this sensor information can be exploited to derive the metric measures of the captured scene. Moreover, we proposed an efficient and accurate method for binocular stereo based on a multi-resolution scheme. The performance of the system was demonstrated on various indoor and outdoor scenes.
Chapter 4

Semi-Direct Visual Inertial Odometry

The system described in the previous chapter used mainly vision to compute its camera pose and estimated motion segments between moments when the phone was held still. This allowed to use zero-velocity updates to reduce the open-loop drift. This way of using the inertial sensor measurement is very limited, in this chapter we will look at how to improve the camera tracking with a Visual Inertial Odometry (VIO) algorithm that can work independent of a structure from motion map.

Recently a number of approaches have leveraged dense RGB-D data, available in real-time from depth sensing cameras such as the Kinect [1], in combination with ICP-like algorithms for pose estimation [37, 64, 36]. The core ideas of these algorithms where reformulated for monocular images, which would make them an option for our use case [65, 66, 67]. These methods use all data available for pose estimation and hence promise high tracking accuracy and robustness. However, they are computationally expensive and typically require powerful GPUs for real-time performance, prohibiting use in mobile
and computationally restricted setups.

Sparse direct methods as used in the previous chapter and other works [68, 61] offer higher precision and robustness than traditional feature extraction and tracking based methods [48] and have comparable or better runtime performance due to saved computation time by skipping the descriptor and matching computations. Being purely vision based, these methods struggle under fast motion, in particular rotation [68], when the camera is moving along it’s focal axis, and in scenes with few corner-like features [61]. Most direct photometric approaches are formulated as energy minimization problem and leverage Gauss-Newton like methods to solve for camera pose. Therefore, tightly coupling IMU and vision data for every processed image frame is non-trivial in these frameworks.

On the other hand filter-based approaches to VO or VIO [69, 47, 70] tightly couple inertial measurements with visual data and have demonstrated robustness to fast rotation, partial loss of visual tracking and relatively little drift over time. Also important extensions towards applying these algorithms on mobile phones that can directly estimate the time delay between camera images and the inertial sensor data and the rolling shutter readout time and compensate for the effects [71]. However, we are not aware of existing methods to incorporate direct methods (i.e., photometric error minimization) directly in the measurement model of the EKF framework.

In this chapter we propose a combination of direct photometric error minimization in an Extended Kalman Filter (EKF) framework. Allowing us to fuse vision and inertial data tightly, almost at the raw sensor level. Both signal sources measure the same motion but have different, complementary sensor characteristics which can provide additional constraints during the optimization camera pose. Fusing the complementary data sources at the lowest possible level allows the estimated system state to constrain and guide the image-space measurements, enforcing consistency between image-space feature positions and 6DOF camera motion. Our approach works with very few (10-20) and very small (as small as $3 \times 3$) image-patches. This sparsity allows for an efficient and fast implementation. Furthermore, the
method can handle scenes that do not have any corner-like features and hence is suitable for scenarios in which other methods fail.

4.1 Related Work

Dense methods

Dense direct methods operate on surface measurements directly, either using depth estimates of a stereo camera or a RGB-D sensor [64, 36], and do not extract sets of features from this data. These approaches require heavy GPU parallelization due to computational cost and tend to have restricted working ranges, due to sensor working principles. Dense monocular methods do not have special sensor requirements but have similar computational costs because they require the build-up of an explicit cost volume [66] or on computing constrained scene flow [72].

Semi-dense direct methods

Recently [68] proposed to estimate depth only for pixels in textured image areas and introduce an efficient epipolar search, enabling real-time visual odometry and semi-dense point cloud reconstruction on a standard CPU and even on mobile platforms [73]. Photometric alignment on sparse, known 3D points has been used by [61] to improve accuracy and robustness of the standard SLAM pipeline of [48]. Most of these approaches either do not use inertial data or treat both data sources mostly independently and only fuse the two at the camera pose [74] or sparse keyframe level [16] to estimate metric scale on top of vision based camera pose.

Visual Inertial Odometry

The very generic EKF framework has also been used for vision only camera tracking and structure from motion [69]. It allows for straightforward sensor fusion and hence it is very popular for algorithms
designed with mobile platforms in mind, which predominantly are shipped with cameras and IMUs [70, 47]. However, to make the problem computationally tractable typically EKF approaches operate on sets of image-space features. As outlined above this comes with certain issues. In the filtering context a further issue is that they are uncoupled from the estimated system. It is only possible to use predicted locations to support the feature correlation or matching but the correlation itself is completely unconstrained by the overall system state. This requires costly outlier rejection (e.g., RANSAC) to detect features that where not matched or tracked correctly.

To improve feature correlation results, early SLAM approaches have used photometric error and patch-wise normal estimation [75] to improve feature correlation but this was done separately from the standard EKF-SLAM steps. Instead of externally optimizing the homography between filter updates, [76] estimates the patch normal inside the EKF framework. The drawback with these methods is that the local patches have to be reasonably large (25 × 25 pixels or larger) for the normal to be estimated robustly. This increases computational cost and introduce problems with patches near depth discontinuities, where the texture in a patch would not change consistently with camera motion.

4.2 System Overview

In this chapter we propose a method that is based on sparse, very small patches and incorporates the minimization of photometric error directly into the EKF measurement model so that inertial data and vision-based surface measurements are used simultaneously during camera pose estimation. Our formulation allows for an efficient implementation that runs in real-time on a CPU and can be implemented on mobile platforms as well. The tight integration of direct surface measurements and inertial data allows to track image regions that are difficult to tackle with approaches that rely on feature trackers like KLT for example line-like structures in images.
Our technique is a visual-inertial odometry approach, this means that camera pose is estimated only from currently visible regions of the observed 3D scene and we do not maintain a global map of previously extracted feature points, we remove all features from the state space as soon as they leave the field of view of the camera. Note that the proposed approach could easily be extended with standard mapping back-end as for example in [77] or more advanced in a system like [19]. Following the approach in [78] we reformulate the EKF framework which has been used successfully for structure from motion [69] into an Error State Extended Kalman Filter ErKF.

Fig. 4.1 illustrates our approach. A small number of small patches were extracted in previous frames and the corner locations of the patches are projected into a predicted camera pose based on IMU data. An affine warp for the whole patch is computed (cf. Fig. 4.2). The algorithm then jointly optimizes the camera pose and the patch depth by minimizing the intensity residual. One advantage of this approach is that we do not rely on the extraction of features of a specific type (e.g., corners) and their 2D image space motion but can use any patches with sufficient gradient. In particular, patches which lie on lines (see highlighted region in Fig. 4.1) or they can be placed in image areas with good texture, similar to the pixel selection in (semi-)dense approaches [68]. Furthermore, we use an inverse depth parametrization [79] for the patch depth which allows us to start tracking without a special initialization sequence as it is necessary with other approaches [61, 77].
Figure 4.1: The left image shows the pixel patches selected for odometry computation on the current camera image. The middle two images show a selection of the pixel patches in the current image and the respective reference patches. The algorithm is optimizing the camera pose and the patch depth by minimizing the intensity residual. The rightmost image shows the intensity residuals with an arrow illustrating the patch motion that is needed to align both patches resulting from the image gradient of the current image.
4.3 Error State Filter Design

4.3.1 Statespace structure

The camera state $x_c = [p_{wi}, q_{iw}, v, o_a, o_\omega]^T \in \mathbb{R}^{16}$ contains the current IMU position $p_{wi}$, orientation quaternion $q_{iw}$, linear velocity $v$, the accelerometer and gyroscope offsets $o_a$ and $o_\omega$ respectively. The point state vector $x_m$ contains the states for the tracked patches (see Section 4.3.2 for description). The whole state is then $x = [x_c, x_m]^T$. We use an error state formulation $\tilde{x} = x - \hat{x}$ which is defined as the difference between the true state $x$ and the estimated state $\hat{x}$. The error state vector is defined as $\tilde{x}_c = [\tilde{p}, \tilde{\theta}, \tilde{v}, \tilde{o}_a, \tilde{o}_\omega]^T \in \mathbb{R}^{15}$, see [78] for more details. We used a static calibration for the transformation between camera and the IMU, with $p_{ic}$ the relative translation and $R_{ci}$ the relative rotation from camera to IMU. We want to point out that it is possible to include online camera-IMU calibration by following [47] and time delay and rolling shutter readout time can be estimated analogous to [71].

4.3.2 Point Parametrization

The estimated points are parametrized as anchored inverse depth bundles [79]. For every time step where new patches are initialized, the point state vector $x_m$ is augmented with $x_{new} = [p_k, q_k, \rho_{\text{init}}, ..., \rho_{\text{init}}]^T$ where $p_k$ and $q_k$ are the IMU pose after the filter update was computed at the current time step and for every initialized patch $\rho_{\text{init}}$ their respective inverse depths in the camera frame, which are set to an arbitrary value. So an anchor frame contains one or more patches that have individual inverse depth estimates. In addition to the point state vector the location of each patch in normalized image coordinates in the anchor frame is stored statically in a vector $m$. The 3D position of a point can be computed as follows:

$$ p_i = p_f + R(q_f)^T (R_{ci}^T m_i \rho_i - p_{ic}) \in \mathbb{R}^3 $$ (4.1)
where \( p_f \) is the position and \( R(q_f) \) the orientation of the according anchor frame and \( \rho_i \) the inverse depth of the point.

### 4.3.3 Continuous Time Model

The nonlinear process model follows the standard formulation of [78].

\[
\begin{bmatrix}
\dot{\hat{p}} \\
\dot{\hat{q}} \\
\dot{\hat{v}} \\
\dot{\hat{\omega}} \\
\dot{\hat{a}}
\end{bmatrix}_{x_{c,k+1}} = 
\begin{bmatrix}
v_k \\
q_k \times q(z_\omega - o_\omega_k + q_\omega) \\
R_{iw}^T(z_a - o_a + q_a) \\
q_o_\omega \\
q_o_a
\end{bmatrix}_{f(x_k,q_k,u_k)}
\]

with \( q = [q_a, q_\omega, q_o_\omega, q_o_a] \) the process noise and \( z_\omega \) and \( z_a \) the measurements of the gyroscope and accelerometer.

The Jacobians of the process model used in the EKF are given as

\[
F = \left. \frac{\partial f}{\partial x} \right|_{\hat{x}_{k|k+1}}, \quad G = \left. \frac{\partial f}{\partial q} \right|_{\hat{x}_{k|k+1}}
\]

The 3D points are modeled as static scene points assuming that they do not move in the 3D space. Therefore, the feature space dynamics are given as \( \dot{\hat{p}}_{f_i} = 0 \), \( \dot{\hat{q}}_{f_i} = 0 \) and \([\dot{\hat{\rho}}_1 \ldots \dot{\hat{\rho}}_N] = 0\).

### 4.3.4 Prediction

Following the continuous-discrete hybrid approach suggested in [80] we perform a \( 4^{th} \) order Runge-Kutta integration of the continuous motion equations given in 4.3.3. The error covariance \( P = [P_{CC} \ P_{CM} \ P_{MC} \ P_{MM}] \) is propagated by:

\[
P_{k+1|k} = 
\begin{bmatrix}
P_{CC,k+1|k} & \Phi(t_{k+1}, t_k)P_{CM,k|k} \\
P_{MC,k|k} & \Phi(t_{k+1}, t_k)^	op \\
\end{bmatrix}
\]

68
4.4 Photometric Update

The camera error covariance is numerically integrated as in [78] by

\[ \dot{P}_{CC} = FP_{CC} + P_{CC}F^\top + GQG^\top \]  \hspace{1cm} (4.5)

where \( Q \) represents the process noise and \( \Phi(t_{k+1}, t_k) \) is integrated by

\[ \dot{\Phi}(t_k + \tau, t_k) = F\Phi(t_k + \tau, t_k), \tau \in [0, T]. \]  \hspace{1cm} (4.6)

4.4 Photometric Update

The photometric update is different from standard visual odometry approaches that use 2D image positions from an external feature tracker or matcher. In our case the measurement model \( h(x) \) is used to directly predict the appearance of a pixel patch (the 1 dimensional intensity values of the pixels) of a reference view given the pixel values in the current camera view (see Figure 4.2).

More specifically, for every pixel patch the current estimate of the 3D location of the center pixel is transformed into the current camera frame:

\[ p_{c_{fi}} = p_{wf_r} - R_{iw_k}^\top p_{ic}, \quad p_{wck} = p_{wik} - R_{fw_r}^\top p_{ic} \]  \hspace{1cm} (4.7)

\[ h_c(x) = R_{ci}R_{iw_k}(\rho_i(p_{c_{fr}} - p_{wck}) + R_{fw_r}^\top R_{ci}^\top \pi^{-1}(u_{ri})) , \]  \hspace{1cm} (4.8)

where \( h_c \) is a vector from the predicted current camera center towards the 3D location of the patch center, \( R_{fw_r}, p_{c_{fr}} \) are the rotation and position (of the camera frame) of the reference view, \( R_{wck}, p_{wck} \) the predicted rotation and position of the current camera, \( R_{ic}, p_{ci} \) the camera-IMU transformation, \( u_{ri} \) is the stored center pixel of the tracked patch in the reference frame, \( \rho_i \) the inverse depth and \( \pi^{-1} \) is the camera back-projection function. Then the point is projected into image space with the pre-calibrated camera parameters. The measurement function \( h(x) \) is then used to compute the appearance of the reference pixels given the current state estimates and the current camera image:

\[ h(x) = I_k(\pi(h_c)) . \]  \hspace{1cm} (4.9)
\[ \rho_i = \frac{1}{d_i} \]

\[ \| \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]
Here, the measurement model equation $h(x)$ is given for a single pixel. In order to increase robustness we extend the single measurement to a patch around this point. In doing so we make the assumption that the scene around this point is planar because only the depth of the single point is modeled. However, since we are using small patches (3 × 3 pixels), the assumption of a locally flat scene can be made similar to [61]. To further reduce the degrees of freedom, we assume the patch normal to be orthogonal to the image plane in the anchor frame. This assumptions allows us to model the appearance of the pixels surrounding the center point via an affine warp $A$, encoding in-plane rotation of the patch, the depth dependent size of the patch and some shear caused by a camera observing the patch from a different angle.

The residual $r_i$, recursively minimized during camera pose estimation by the Kalman filter, is the photometric error between all the pixels in the reference patch and the pixels in the warped patch, extracted from the current camera view:

$$r_i = I_r(u_{ri}) - I_k(A(R_{wf_r}, p_{wf_r}, R_{wc_k}, p_{wc_k}, u_{ri}, \rho_i)),$$  \hspace{1cm} (4.10)

with $I_k$ the current image, $I_r$ the reference image, $R_{wf_r}, p_{wf_r}$ and $R_{wc_k}, p_{wc_k}$ the $[R|t]$ rotation and translations of the reference and the current view, $u_{ri}$ the location of the center pixel and $\rho_i$ the inverse depth of the point. This residual is computed for all points that are currently in the state space.

Finally, the Kalman filter update step requires the linearization of the measurement function $H$, computed as the derivative of $h(x)$ with respect to the states $x$:

$$H = \frac{\partial h(x)}{\partial x} = \nabla I_k A \frac{\partial \pi(h_c)}{\partial h_c(x)} \frac{\partial h_c(x)}{\partial x},$$  \hspace{1cm} (4.11)

with $\nabla I_k$ being the image gradient of the warped patch extracted from the current frame, $\frac{\partial \pi(h_c)}{\partial h_c(x)}$ is the $2 \times 3$ camera projection derivative matrix.

Assuming familiarity with the EKF framework, the equations given here and in the previous sections should be sufficient to implement...
Chapter 4 Semi-Direct Visual Inertial Odometry

Figure 4.3: The optimal pyramid level for pixel patch alignment is selected based on the estimated variance of the projected point location in pixel space. If the camera motion is fast, higher levels are used; if there is low variance on the camera pose lower levels are used for higher precision.

the proposed algorithm. However, there are a number of details that can be taken into consideration in order to improve robustness in real-world settings. We briefly discuss these in the following subsections.

4.4.1 Patch extraction

The patches can be selected using many different methods and in particular there is no requirement for patches to be centered on corners that are detected by corner detectors. In the experimental section we demonstrate the performance of our technique using only patches that are centered on single, non-intersecting lines. A simple implementation could just extract FAST keypoints [81] on an uniform grid. However, we noticed that selecting image areas based on the Shi-Tomasi score [62] that are stable over the whole scale space of the image pyramid leads to better and more stable results.

4.4.2 Image Pyramid Level Selection

One issue inherent with all direct methods is that of relatively small convergence radii during optimization. The methods only converge if the predicted and correct position in image space fall within a region in which the image gradient points towards the global minimum. In
some cases this region can be as small as a half-pixel radius from the correct position. This implies that if a predicted pose and hence the resulting projected point in the image lies more than half a pixel away, the optimization would diverge and the method would fail if this happens for many patches.

The standard approach to increase this convergence region is to apply image pyramids, starting the alignment on the highest level, where the pixels cover a larger area. After convergence on the highest level the optimization is repeated successively on the remaining levels of the pyramid, leading to many additional iterations and therefore additional computational cost.

In our method we attain predictions and associated uncertainties for all state variables and their covariances. This can be used to compute the variance on the point location in image space and consequently allows to select the optimal level in the image pyramid such that convergence is guaranteed (see Fig. 4.3) while providing the highest possible accuracy. Compared to the standard approach of iterating through the whole image pyramid starting from the highest (i.e. lowest resolution) level, this approach saves computation time while still offering the advantage of a larger convergence radius of the higher pyramid levels and the precision of the lower levels. In addition as the method selects the lowest possible level for convergence, it also reduces the risk of converging towards a wrong local minimum if the optimization on higher levels would not converge towards the correct image location.

The error covariance can be computed by omitting the image gradient when taking the derivative of the measurement function $\frac{\partial h}{\partial x}$:

$$H_{\pi} = \frac{\partial \pi}{\partial h_c} \frac{\partial h_c}{\partial x},$$  
(4.12)

$$S_{\pi} = H_{\pi} P_{k-1/k-1} H_{\pi}^T.$$  
(4.13)

The major axis of the error ellipsoid in image space is then the largest Eigenvalue of the $2 \times 2$ matrix $S_{\pi}$. To guarantee convergence the length of this axis should be smaller than 1 pixel at the respective
pyramid level. These calculations can be done while computing the derivate $H$ during the update step before the image gradient of the pixel patch is computed. The only overhead is the computation of the $2 \times 2$ matrix $S_\pi$ and its Eigenvalues.

### 4.4.3 Iterated Sequential Update

Inherently our formulation requires the processing of many measurements for each update step (every pixel is a measurement). Unfortunately this impacts runtime performance. The size of the Jacobian $\frac{\partial h}{\partial x}$ and as consequence, the size of the innovation covariance matrix $S$ will be $ns \times ns$, where $n$ is the number of patches and $s$ the patch size in pixels. Because $H$ needs to be inverted during every EKF update step, the size of $S$ directly impacts the runtime.

In the case of the linear Kalman filter, sequential updates [82] can be utilized to alleviate this situation. We observed that if one iteratively re-linearizes the measurement matrix $H = \frac{\partial h}{\partial x_{seq}}$ around each updated estimated state sequentially, the algorithm produces very good estimates in practice. The sequential update reduces the computations to $n$ inversions of a $s \times s$ matrix which drastically enhances runtime performance.

### 4.4.4 Measurement Compression

In addition to the sequential update, the dimension of the intensity measurements per patch can be reduced to 2. To this end, a QR decomposition of the $H$ is done and the reduced measurement matrix $H_r$ and residual $r_r$ is computed by

\[
H = [Q_r|Q_0]R \quad (4.14)
\]
\[
H_r = Q_r^T H \quad (4.15)
\]
\[
r_r = Q_r^T r \quad (4.16)
\]
4.4 Photometric Update

With this optimization the effect of the size of the pixel patch on the computation time of the update is reduced to the QR decomposition above, which allows for larger pixel patches to be used without a heavy computational impact.

4.4.5 Anchor-centered Parameterization

Representing the odometry filter with respect to the world origin poses issues in terms of consistency. One of them is that since absolute position measurements are never generated, as it would be the case for GPS position information, the uncertainty of the position will grow over time up to the moment when it will be so large that the filter will degrade in performance. One way to counter this effect is to move the frame of reference with the camera and move the uncertainty to the estimate of the world frames origin. This way the uncertainty near the camera and the observed points is always small. There are different approaches to do this, the most extreme is to bind the frame of reference to the camera and even propagate the system that way. But this is very costly since the whole local map and the covariance need to be transformed for every inertial measurement at high frequency. A better approach is to keep the reference frame fixed at the position of the last update and propagate the current pose normally and then after the next update move it to the latest position [56]. This keeps the propagation efficient and ensures that the increase in uncertainty is small during the short propagation phase until the next image. In [83] the authors use a similar anchor-based representation of the map and argue that it is better to chose the anchor with the lowest uncertainty as the next reference frame when the previous reference anchor lost the last observed feature. Because the estimation error from the previous reference to the new reference anchor is fixed forever when the transformation happens this way the error can be minimized. We implemented the latter version to improve the filter performance for longer tracks.
4.5 Experimental Results

We performed a comparison against ground truth acquired from a Vicon system and two of recently published methods that use a photometric approach in a semi-dense manner [68] and patch-based on corner locations [61]. We moved the camera around a regular office space, see an example image from the dataset in Figure 4.7 on the left. The top plot shows the 3D view of the final positions of the tracked points during the sequence and the trajectory of our method together with ground truth. The third plot shows the position in all axes, as can be seen, our method tracks the camera pose in typical scenes with equal quality than the compared methods. The initialization for [68] was difficult in this particular scene and its performance did not match the expected level.

The most compelling advantage of our constrained direct method is that it does not rely on the presence of corner like features. In particular, our implementation works on scenes that only contain (non-intersecting) lines. Figure 4.4 shows a demonstration of such a scene. It is clear that methods that rely on external trackers like KLT will fail in this scenario since the tracker is not able to fix the tracked points at a position and thus the point will start to randomly slide along the edge (see also Figure 4.5. Since in our implementation the location of the patches are constrained by the model in the filter, the algorithm is able to fully initialize with patches that lie on these kinds of edges even with a patch size of only 3x3 pixels.

Figure 4.6 shows the results of a challenging dataset with a camera moving in front of a curtain having almost only line-like structure in view. Only few patches where placed on corner-like areas, this was enough to fix the camera pose from drifting in vertical direction. This demonstrates that the proposed method can be used to track scenes that are rather hard for methods that rely on unconstrained feature correspondences as is the case in many indoor scenes.

The runtimes for the photometric update in C code on a Core i5 desktop computer is approximately 8.3 ms with 62 patches of 5x5 pixels, thus already allowing for real-time use. We plan to implement
Figure 4.4: Thanks to the constraints on the pixel patches, the algorithm is able to initialize even on this difficult scene consisting only of almost vertical lines. On the right side the used 3x3 pixel patches are visible.

a fully optimized version for mobile ARM CPUs.
Figure 4.5: Green border: red stars in the image show the center of the pixel patches tracked with the presented algorithm, on the right, the $16 \times 3 \times 3$ Pixel patches that were tracked in the sequence are shown, Red border: if KLT tracking is used, many of the tracked features (green cross) start to drift along the edge leading to complete failure without outlier rejection.
Figure 4.6: Visual-Inertial odometry on a scene with lines. Top: patches on lines. Bottom: comparison of the estimated camera position in the three axes with ground truth (VIC-CON data).
Figure 4.7: Blue: Trajectory from the presented algorithm, Magenta: Semi Direct Visual Odometry (SVO) [61], Green: Semi-Dense Visual Odometry (SDVO) [68], Black: Ground Truth (VICON data). The initialization of SDVO had issues in the selected scene, due to the suboptimal initial map the performance is not as good as can be expected.
4.6 Conclusion

In this chapter we presented a novel, Kalman filter-based semi-direct visual inertial odometry approach that combines the advantages of a tightly coupled visual-inertial Kalman filter and the robustness and precision of direct photometric methods. We demonstrated how the photometric update can be built into a standard error-state Kalman filter odometry algorithm. We proposed an efficient implementation that reduces the impact of the larger number of measurements when minimizing the photometric residual error and showed how to deal with the issue of the limited convergence radius of photometric approaches in the EKF context where iterating through multiple image pyramid level is not always an option due to the processing time necessary in that case. Finally, we demonstrated that our proposed algorithm matches the tracking quality of other state of the art direct photometric approaches. And in addition, thanks to the rigid scene constraints the proposed algorithm can work with pixel patches lying only on line-like structures and is even able to fully initialize without special procedure in such scenes. In future work, we want to evaluate the tracking quality in terms of the pose error compared to the running time of the different methods.
Chapter 5

Point Cloud Fusion

While the system in Chapter 3 produced already quite nice results, their quality is still limited when comparing to state of the art results from image based reconstruction.

This chapter can be regarded as an effort towards closing this gap between the capabilities of the system for live 3D reconstruction on mobile devices and the accuracy of similar interactive systems designed for high-end systems based on video cameras [35] or depth sensors [40, 36].

The main contribution is the development of an efficient and accurate scheme for integrating multiple stereo-based depth hypotheses into a compact and consistent 3D model. Thereby, various criteria based on local geometry orientation, underlying camera poses and photometric evidence are evaluated to judge the reliability of each measurement. Based on that, the proposed fusion technique justifies the integrity of the depth estimates and resolves visibility conflicts. We demonstrate the performance of the developed method within a framework for real-time 3D reconstruction on a mobile phone and show that the accuracy of the system can be improved while retaining its interactive rate.
Chapter 5 Point Cloud Fusion

5.1 Related Work

As this chapter deals with the problem of depth map fusion, which is a classical problem in multi-view 3D reconstruction as already mentioned in Chapter 2, it is related to a myriad of works on binocular and multi-view stereo. We refer to the benchmarks in [50], [21] and [51] for a representative list. However, most of those methods are not applicable to our particular scenario as they are not incremental in nature or don’t meet the efficiency requirements of embedded systems. In the following, we will focus only on approaches which are conceptually related to ours.

Building upon pioneering work on reconstruction with a hand-held camera [52], Pollefeys et al.[53] presented a complete pipeline for real-time video-based 3D acquisition. The system was developed with focus on capturing large-scale urban scenes by means of multiple video cameras mounted on a driving vehicle. Yet, despite its real-time performance, the applicability of the system on a live scenario is not straightforward. Nevertheless, we drew some inspiration from the utilized depth map fusion scheme, originally published in [84]. The first methods for real-time interactive 3D reconstruction were proposed by Newcombe et al.[72] and Stuehmer et al.[54]. In both works, a 3D representation of the scene is obtained by estimating depth maps from multiple views and converting them to triangle meshes based on the respective neighborhood connectivity. Even though these techniques cover our context, they are designed for high-end computers and are not functional on mobile devices due to some time-consuming optimization operations. Another approach for live video-based 3D reconstruction, which is conceptually similar to ours, was proposed by Vogiatzis and Hernandez [39]. Here, the captured scene is represented by a point cloud where each generated 3D point is obtained as a probabilistic depth estimate by fusing measurements from different views. Similar to the already discussed methods, this one also requires substantial computational resources. Another key difference to our framework is the utilization of a marker to estimate camera poses, which entails considerable limitations in terms of usability. Re-
cently, the work of Pradeep et al.[33] appeared. It presents another pipeline for real-time 3D reconstruction from monocular video input based on volumetric depth-map fusion. Again, those techniques are developed for high-end computers and have never been demonstrated on embedded systems.

Probably the most similar method to ours was proposed in [85] and subsequently generalized in [86, 87]. Therein, a system for interactive in-hand scanning of objects was demonstrated. Similar to the approach, presented in this chapter, it relies on a surfel representation of the modeled 3D object. However, the developed fusion scheme is designed for measurements stemming from active sensors, which are considerably more accurate than stereo-based ones. Therefore, the employed confidence estimation is quite different from this proposed in this chapter.

Recently, the first works on live 3D reconstruction on mobile devices appeared. Wendel et al.[55] rely on a distributed framework with a variant of [49] on a micro air vehicle. A tablet computer is solely used for visualization while all demanding computations are performed on a separate server machine. Sankar et al.[88] proposed a system for interactively creating and navigating through visual tours. Thereby, an approximate geometry of indoor environments is generated based on strong planar priors and some user interaction. Pan et al.[41] demonstrated an automatic system for 3D reconstruction capable of operating entirely on a mobile phone. However, the generated 3D models are not very precise due to the sparse nature of the approach. Prisacariu et al.[42] presented a shape-from-silhouette framework running in real time on a mobile phone. Despite the impressive performance, the method suffers from the known weaknesses of silhouette-based techniques, e. g. the inability to capture concavities.
5.2 Multi-Resolution Depth Map Computation

In the first stage of the 3D modeling pipeline depth maps are created from a set of keyframes, and corresponding calibration information and camera poses. Here, the same basic setup that is proposed in Chapter 3 is adopted. Apart from being efficient and accurate, it is particularly appealing due to the potential of the utilized multi-resolution depth map computation scheme for implementation on mobile GPUs. In short summary, the process of acquiring depth maps with the adaptions is as follows:

The camera motion tracking system produces a series of keyframes and associated camera poses, which are provided to a dense modeling module. As abrupt jumps in the camera motion cannot be expected in this scenario as the tracking system guarantees a smooth keyframing sequence, a straightforward strategy is to maintain a sliding window containing the most recent keyframes and use them for stereo matching but also to check consistency between different depth maps. As in Chapter 3, to pursue an interactive framework on mobile devices, binocular stereo instead of multi-view stereo is applied to minimize the memory access overhead. In particular, a newly arrived keyframe is used as a reference image and is matched to an appropriate image in the current buffer. Thereby, a multi-resolution scheme for the depth map computation is employed to reduce the computational time and to avoid local maxima of the photoconsistency score along the considered epipolar segments. When moving from one resolution level to the next, the depth range is restricted based on the depth estimates at neighboring pixels. Additionally, computations are limited to pixels exhibiting sufficient local image texturedness within regions where the current 3D model has not reached the desired degree of maturity. The result is a depth map possibly corrupted by noise due to motion blur, occlusions, lack of texture, presence of slanted surfaces etc. A very efficient and effective filtering procedure is applied to remove the outliers. Thereby, the consistency of each depth measurement is tested
5.3 Confidence-Based Depth Map Fusion

A central issue in the design of a depth map fusion approach is the representation of the modeled scene. While triangle meshes exhibit a common geometric representation, they do not seem well-suited for interactive applications running in real time since considerable efforts are needed to guarantee the integrity and consistency of the mesh topology after adding, updating or removing any vertices. Note that the user is expected to make use of the live visual feedback and re-capture certain parts of the scene until the desired surface quality is reached. For that reason, we rely on a *surfel* representation [89]. A surfel $s_j$ consists of a position $p_j$, normal vector $N_j$, color $C_j$ and a confidence score $c_j$ which is defined as the difference between a cumulative inlier and outlier weight, i.e. $c_j = W_j^{(in)} - W_j^{(out)}$. Additional attributes like local patch radius or visibility information could be maintained if needed. The utilized surfel representation offers the required resilience since the unstructured set of surfels can easily be kept consistent throughout any modifications.

The proposed depth map fusion approach relies on the following scheme: When a new depth map becomes available, a weight is assigned to each pixel measurement reflecting its expected accuracy. Based on this input, the surfel model is modified by adding new sur-
fels, updating or removing existing ones. In the following, these steps are explained in more detail.

### 5.3.1 Confidence-Based Weighting

The accuracy of a depth measurement, obtained from stereo matching, depends on many factors, e.g. inherent scene texture, geometry orientation, camera noise, distance between the scene and the camera device etc. In an effort to capture all those aspects we assign different weights to each estimate and combine them subsequently to obtain a final weighting score that expresses our confidence in the particular depth value.

**Geometry-Based Weights.** The accuracy of a depth measurement depends on the local surface orientation at that point. The depth measurement is more accurate when the observed geometry is fronto-parallel and less accurate at grazing viewing angles. As a local normal vector is computed to each depth estimate, those cases can be identified by considering the scalar product between the normal and the respective viewing direction of the camera. If \( n_x \in S^2 \) denotes the normal vector and \( v_x \in S^2 \) stands for the normalized reversed viewing direction of the camera for a pixel \( x \in \Omega \subset \mathbb{Z}^2 \) within the image domain, we define a geometry-based weight to \( x \) as

\[
w_g(x) = \begin{cases} 
\frac{\langle n_x, v_x \rangle - \cos(\alpha_{\text{max}})}{1 - \cos(\alpha_{\text{max}})}, & \text{if } \phi(n_x, v_x) \leq \alpha_{\text{max}} \\
0, & \text{otherwise},
\end{cases}
\]

(5.1)

where \( \alpha_{\text{max}} \) is a critical angle at which the measurements are considered unreliable and is set to 60° throughout all experiments. The weight defined in (5.1) takes on values within \([0, 1]\). Note that it does not directly depend on the depth estimates. However, there is an indirect relation as the computation of the normal vectors relies on them.

**Camera-Based Weights.** The accuracy of a depth measurement,
obtained from binocular stereo, depends on the utilized camera setting. For example, short baselines implicate high depth imprecision as larger changes of the depth along the visual rays result in small projection footprints on the image plane of the non-reference camera. Analogously, increasing the image resolution or moving the camera closer to the scene leads to more accurate depth estimates. Based on these observations, a camera-based weight could be defined by measuring the depth deviation corresponding to a certain shift (for example one pixel) along the respective epipolar line. Yet, this cannot be realized efficiently since it involves an additional triangulation operation. Further complications pose the discrepancy between viewing ray traversal and pixel sampling. Instead, we revert the inference and measure the pixel shift $\delta$ that a certain offset along the ray produces. More concretely, the offset along the visual rays is set to 1/600 of the depth range. Then, a camera-based weight to a pixel $x$ is defined as

$$w_c(x) = 1 - e^{-\lambda\delta},$$

(5.2)

where $\lambda \in \mathbb{R}$ is a parameter specifying the penalizing behavior of the term and is set to 5.0 throughout all experiments, and $\delta$ is measured in pixel coordinates. Note that $w_c \in [0, 1]$ is inversely proportional to the estimated depths, i.e. larger depths get lower weights and smaller depths get higher weights. This corresponds to the intuition that parts of the scene closer to the camera are expected to be reconstructed more accurately than parts further away from the camera. Moreover, the length of the baseline is also taken into account by the formulation in (5.2). In particular, depth maps, obtained from short baselines, will generally be weighted lower.

**Photoconsistency-Based Weights.** Probably the most straightforward criterion to judge the accuracy of a depth measurement is its photoconsistency score. However, this is also the least discriminative criterion since the provided depth maps are already checked for consistency and filtered, thus, the respective matching scores are expected to be high. The easiest way to obtain the photoconsistency value to a depth estimate is to use the one delivered by the stereo module. Yet,
as normal information is available at that point, a more accurate measure can be employed. Here, we adopt Normalized Cross-Correlations (NCC) over $5 \times 5$ patches where the provided normal vectors are leveraged to warp the patches from the reference image to the second view. Then, for a pixel $x$ we specify

$$w_{ph}(x) = \begin{cases} NCC(x), & \text{if } NCC(x) \geq thr \\ 0, & \text{otherwise} \end{cases}$$

(5.3)

as the photoconsistency-based weight. Thereby, $thr$ is a threshold parameter set to 0.65 throughout all experiments, and $NCC(x)$ denotes the NCC score for the depth and the normal at $x$. Again, we have $w_{ph} \in [0, 1]$. It should be noted that the computation of the photoconsistency-based weights is more time-consuming than that of the geometry-based and the camera-based ones while having the least contribution to the final weighting values. For this reason, it could be omitted when more efficiency is required.

The last step is to combine all weight estimates and to provide a final overall weight to each depth measurement in the provided depth map. To this end, for each $x$ we set

$$w(x) = w_g(x) \cdot w_c(x) \cdot w_{ph}(x).$$

(5.4)

The overall weight lies in $[0, 1]$ and will be high only when all three weights, the geometry-based one, the camera-based one and the photoconsistency-based one, are high. In other words, a measurement is considered as accurate if it is accurate from geometric, stereoscopic and photometric point of view.

Fig. 5.1 shows an example of the estimated weighting for a depth map capturing a small church figurine. For all depth measurements the corresponding weights are computed according to (5.4). Note that the effects from applying the geometry and the camera term are clearly visible. Indeed, pixels, where the local normal vector points away from the camera, get small weights. Also, more distant measurements tend
Figure 5.1: Confidence-based weighting of depth measurements. The reference image of a stereo pair and corresponding color-coded weights to the computed depth estimates. Green represents high weighting, red represents low weighting. Note that pixels, where the local normal vector points away from the camera, get small weights. Also, more distant measurements tend to be weighted low.

to be weighted low. The effect from applying the photoconsistency term is less noticeable.
5.3.2 Measurement Integration

When a new depth map becomes available and confidence weights are assigned to all measurements, the provided data is used to update the current surfel cloud. This is done using three basic operations: surfel addition, surfel update and surfel removal. New surfels are created for parts of the depth map that are not explained by the current model. Surfels that are in correspondence with the input depth map are updated by integrating the respective depth and normal estimates. Surfels with confidence value below a certain threshold are removed from the cloud. In the following, these operations are explained in more detail.

Surfel addition. Surfels are added in those parts where the depth map is not covered by model surfels. Of course, for the initial depth map all measurements will create new surfels. For each newly created surfel the position and normal vector are set according to the depth and normal estimate of the measurement. The color is set to the color of the respective image pixel. The cumulative inlier weight is initialized with the weight of the depth measurement and the cumulative outlier weight - with zero.

Surfel update. If the projection of a surfel coincides with a provided depth measurement, the surfel is updated. Let \( s_j = (p_j, N_j, C_j, W_j^{(in)}, W_j^{(out)}, c_j) \) be the surfel of interest with the surfel location \( p_j \), the surfel normal \( N_j \), the surfel color \( C_j \), the current inlier weight \( W_j^{(in)} \), the current outlier weight \( W_j^{(out)} \) and current consistency weight \( c_j \). If there are multiple surfels along the same visual ray, we take the one closest to the camera center that is expected to be visible. Additionally, we maintain a state vector \( X_j = (p_1, p_2, p_3, \theta, \phi) \in \mathbb{R}^5 \) encoding its current position and normal. Thereby, the normal is represented by means of a polar angle \( \theta \) and an azimuth angle \( \phi \). When a new surfel is created, a spherical coordinate system is generated with the provided normal estimate as the first base vector. Let \( x = \Pi(p_j) \) be the projection of the surfel onto
the image plane of the current frame and let \(d(p_j)\) be its depth with respect to the camera center. At \(x\) the given depth map provides a depth measurement \(d_x\) and a normal measurement \(n_x\). In addition to that, we get a weight \(w(x)\) reflecting the accuracy of the estimates.

Now, we have to update the surfel based on this input. There are four different update cases (see Fig. 5.2):

1. \(d(p_j) \gg d_x\): The depth measurement occludes the model surfel. By itself this is not a visibility conflict since the depth map could capture a different part of the surface. The dashed line in Fig. 5.2(a) shows a potential visibility configuration. In fact, this is the most delicate case as both the surfel and the measurement could be outliers. Here, we just ignore the depth measurement and do not perform any surfel update. Note that this could cause problems when parts of the surface are acquired which are in the line of sight of already reconstructed ones (with the same orientation). However, this is unlikely to occur in practice as the user usually captures more accessible parts first before moving to locations that are more difficult to reach.

2. \(d(p_j) \ll d_x\): The depth measurement is behind the model surfel. This is a clear visibility conflict. In this case we add the measurement’s weight to the cumulative outlier weight of the surfel, i.e.

\[
W_j^{(out)} \leftarrow W_j^{(out)} + w(x). \tag{5.5}
\]

3. \(|\frac{d(p_j) - d_x}{d(p_j)}| < \epsilon\) and \(\mathbf{x}(N_j, n_x) \leq 45^\circ\): The measurement and the model surfel match, both in terms of depth and normal orientation. Then, the surfel position and normal are updated accordingly. In particular, we compute a running weighted average

\[
X_j \leftarrow \frac{W_j^{(in)} X_j + w(x)X_x}{W_j^{(in)} + w(x)} \tag{5.6}
\]

\[
W_j^{(in)} \leftarrow W_j^{(in)} + w(x),
\]

where the pixel’s depth \(d_x\) and normal \(n_x\) are converted into a state vector \(X_x\).
(4) \[
\frac{|d(p_j) - d_x|}{d(p_j)} < \epsilon \quad \text{and} \quad \gamma(N_j, n_x) > 45^\circ \]  
: The measurement and the model surfel match in terms of depth but the orientations of their normals deviate from each other. We consider this as a visibility conflict and increment the cumulative outlier weight according to (5.5).

Recall that there are two additional attributes to each surfel - a color \( C_j \) and a confidence score \( c_j \). The color is set to the color of the pixel with the largest weight \( w(x) \) used in the fusion process for the surfel. The confidence measure is defined as the difference between cumulative inlier weight and cumulative outlier weight, i. e. \( c_j = W_{j}^{(\text{in})} - W_{j}^{(\text{out})} \), and has to be updated each time one of those values is modified.

**Surfel removal.** Surfels are removed from the cloud during the acquisition process if their confidence falls below a threshold. We set this threshold to \(-0.5\) throughout all conducted experiments. Note that the removal of surfels opens up gaps that can be filled by new more accurate surfels.

One could wonder why the normals are integrated in the proposed depth map fusion scheme. In fact, they can be obtained in a post-processing step by considering the neighborhood of each point within the point cloud. There are two main reasons for this design decision. First, the normal information is useful as it captures the local geometric structure of each depth measurement and enables the identification of accidental matches like in the case depicted in Fig. 5.2(d). Second, the proposed scheme allows to leverage the neighborhood relation between different measurements, provided by the camera sensor. Moreover, note that the proposed depth map fusion procedure is incremental and lends itself to online applications. Also, it allows reconstructed parts of the scene to be recaptured by providing additional depth data and improving the accuracy of the respective subset of the surfel cloud.

Fig. 5.3 depicts the evolution of the confidence scores of the generated surfels for consecutive frames of a real-world sequence. Note that
the confidence values are small for newly created surfels but increase in the course of the acquisition process if they are observed from other viewpoints.
Figure 5.2: Different cases for a surfel update. Red denotes the incoming measurement and blue - the surfel. (a) Measurement is in front of the observed surfel. There is no visibility conflict. (b) Measurement is behind the observed surfel. There is a visibility conflict. (c) Measurement and observed surfel match. (d) Depths of the measurement and the observed surfel match but not their normals. There is a visibility conflict. See text for more details.
Figure 5.3: Confidence evolution during reconstruction. Visualized are the color-coded confidence scores of the generated surfels for consecutive frames of a real-world sequence. Green represents high confidence, red represents low confidence. An input image from the same viewpoint can be seen in Fig. 5.1. Note how the confidence values of surfels, seen from different directions, increase in the course of reconstruction.
Chapter 5 Point Cloud Fusion

5.4 Experimental Results

We validate the proposed confidence-based depth map fusion scheme by comparing it to two state-of-the-art real-time capable alternatives. Furthermore, we demonstrate its performance by integrating it into a system for live 3D reconstruction running on a mobile phone.

5.4.1 Comparison to Alternative Techniques

For the sake of comparison we implemented two alternative techniques meeting the efficiency requirements of the application at hand.

The first one is the merging method used in Chapter 3. Thereby, the interconnection between the different input depth maps is exploited barely to identify inconsistencies and to filter out outliers. All consistent depth measurements are back-projected to 3D and merged into a unified point cloud. Moreover, a coverage mask based on photometric criteria is estimated in each step to reduce the generation of redundant points.

To evaluate the viability of the confidence-based weighting approach, we combined the developed fusion scheme with the weight computation proposed in [84]. The basic idea of this strategy is to judge the accuracy of each depth measurement by analyzing the photoconsistency distribution along the respective visual rays. Rays with a single sharp maximum are expected to provide more accurate estimates than those exhibiting a shallow maximum or several local maxima. More details can be found in [84].

Figure 5.4 shows the reconstructions generated by applying all three techniques on a real-world image sequence. One of the input images can be seen in Figure 5.1. Camera poses were obtained by applying a version of [49]. Note that the approach in Chapter 3 does not explicitly estimate normals to the generated point cloud. Therefore, for the purpose of rendering we assigned to each point a normal vector based on the depth map that was used to create it. For the other two approaches we used the normal estimates obtained online from the fusion process. It is evident that while all three methods achieve a
5.4 Experimental Results

Figure 5.4: Comparison to alternative techniques. From left to right: Reconstructions with the depth map merging technique in Chapter 3, the developed fusion scheme with the weighting suggested in [84] and the complete approach proposed in this chapter. One of the images in the input sequence can be seen in Figure 5.1. The reconstructions contain 311135, 161647 and 181077 points, respectively. While all three methods achieve a high degree of completeness, the proposed approach with confidence-based weighting outperforms the other two in terms of accuracy.

High degree of completeness, the proposed one with confidence-based weighting outperforms the others in terms of accuracy. The technique in Chapter 3 produces an oversampling of the scene and is more sensitive to noise than the other two as each 3D point is based on a single depth measurement. This proves the importance of a depth map fusion scheme. Moreover, the reconstruction obtained with the proposed confidence-based weighting is significantly more accurate than the one relying on the weighting of [84], which validates the deployment of geometric and camera-based criteria in the depth integration process.
5.4.2 Live 3D Reconstruction on a Mobile Phone

Pursuing a system for live 3D reconstruction running on mobile phones as a primary goal, we integrated the proposed method into the framework described in Chapter 3. This substantially improved its accuracy while adding a negligible overhead of less than a second per processed image. In the following, multiple reconstructions of real-world objects, generated interactively on a Samsung Galaxy SIII and a Samsung Galaxy Note 3, are depicted.

Figure 5.5 depicts the reconstruction of a fabric toy of a hippopotamus. Expectedly, homogeneous regions (e.g., on the ball) lead to holes in the 3D model. However, the well-textured head of the hippopotamus is reconstructed at high geometric precision.

Figure 5.6 shows the reconstruction of a relief on a decoration vase. The model was captured outdoors under sunlight conditions. Note that this is a known failure case for many active sensors.

The capabilities of current mobile devices for in-hand scanning are
5.4 Experimental Results

Figure 5.6: Relief. Rendering of the reconstructed surfel cloud with colors and shading, and a reference image of the object. The model was captured outdoors.

Further demonstrated in Figure 5.7. The reconstruction of a Buddha statue in a museum is visualized. Even though the generated point cloud exhibits a substantial amount of high-frequency noise, many small-scale details like the wrinkles of the clothing or the face features are captured in the reconstruction.
Figure 5.7: Buddha statue. Rendering of the reconstructed surfel cloud with colors and shading, and a reference image of the object. Note the accurately captured small-scale details.

5.5 Conclusion

In this chapter we presented an accurate method for confidence-based depth map fusion that is efficient enough to run at interactive speed on current mobile phone hardware. At its core is a two-stage approach where confidence-based weights that reflect the expected accuracy are first assigned to each depth measurement and subsequently integrated into a unified and consistent 3D model. Thereby, the maintained 3D representation in form of a surfel cloud is updated dynamically so as to resolve visibility conflicts and ensure the integrity of the re-
construction. The advantages of the proposed approach in terms of accuracy improvements are highlighted by a comparison to alternative techniques which meet the underlying efficiency requirements of a smartphone platform. Additionally, the practical potential of the developed method is emphasized by integrating it into the live 3D reconstruction system described in chapter 3 running on a mobile phone and demonstrating its performance on multiple real-world objects.
Chapter 6

Applications

This chapter sketches some applications that can be built on top of (parts of) the system described in the chapters before. There are several modules that can be integrated into the existing system to extend its possibilities even further.

6.1 Face Modeling

Human faces constitute a good object to be scanned with image based modeling, they have enough texturedness to allow for dense reconstruction of the whole face. In addition, a face is unique to every human, there is no such thing as a database of all human faces. If someone wants to have their own face as 3D model, it needs to be scanned first. Further, it turns out that warping face template meshes to a face model requires dense enough sampling of the surface to offer enough geometry information to look similar enough to the person that was scanned. Humans are very good at detecting inconsistencies in human faces they know, if the model does not fit the real geometry people will find the model strange, even if the texture is nicely mapped on the model. In this section a slightly tuned version of the
Chapter 6 Applications

Figure 6.1: The point cloud projected on to a cylinder around the face. The left images shows the colors, in the middle the depths and on the right the point normals are displayed. As can be seen the area is not completely closed, but thanks to the representation of the data 2D interpolation algorithms can be used to close the holes.

Presented methods in the previous chapters is used to create dense face models.

One way of creating a mesh from the generated points is to choose a proxy geometry to store the point cloud as depth map. The simplest method would be to use a cylinder or sphere around the head and project the point cloud onto that surface, see Figure 6.1. This depth map based approach also allows to apply additional depth map fusion methods and the resolution of the depth map limits the number of points in the model. One big benefit of having a simple 2D representation of the point cloud is that smaller holes in the point cloud can be easily filled with efficient 2D interpolation approaches like Moving Least Squares that can smoothly fill the missing pixels in the depth map. From that depth map the mesh can directly be generated by connecting all neighboring pixels with triangle edges and project back the pixels as mesh vertices. The texture mapping is also already defined by projecting the camera images on this surface as well. Figure 6.2 illustrates this process, Figure 6.3 shows another result from the mobile phone app with the textured mesh.
Figure 6.2: Left: Texture from one image mapped on the cylinder by using the depth map in the middle. The depth map and the normal map were smoothed by a median filter. One can clearly see the typical artifacts, another approach would be using Moving Least Squares or Total Variation based methods.

Instead of just using a very simple geometry for the depth map, a template face mesh could be used to store the depth map. This has the advantage that the mapped face area is better distributed in the 2D map. For example, steep angles along the bottom side of the nose are badly mapped on a cylinder leading to stretched texture, however, on a face mesh with existing texture map these cases can be avoided. The disadvantage of more specific meshes are that they need to be well aligned and should follow the actual geometry.

One way of aligning such a mesh would be to use detected face features in the 2D images and estimate their positions in 3D by using the tracked camera poses and then run an ICP algorithm to align these points with the same ones on the face mesh, which gives a good initial guess for a non-rigid optimization and defines a useful bounding box of the face for the stereo computation.
Figure 6.3: Results from the mobile phone app showing the point cloud with color, shaded with the normals and the textured mesh of a face.
6.1 Face Modeling
6.2 Generic Modeling with Cloud Support

There are many application fields where it is very valuable to get a preview of the scanned objects while executing the scan even though the quality that can be computed on the device is not as good as needed for the final result. The final reconstruction would be processed offline on a powerful computer or in the cloud, but sometimes there is no access to the offline processing at the place where the scan happens due to missing cell connection or high roaming costs. In these cases it is important to verify that the scan is complete before leaving the place. Figure 6.4 shows an example for the preview that is possible with the presented system in this thesis and a comparison to a state of the art reconstruction from an offline system. As can be seen, the user can get a good idea of the final result during the scanning process.

There are also other situations where a mobile device can be useful. At an excavation site it is typically necessary to regularly document the current layer with the findings before continuing to dig deeper. Often laser scanners are used for this task, but since they need skilled personnel to be used the interval in scanning is limited. Here a simple mobile solution can be of great use if small parts of the site need to be updated. This can be done with a mobile phone and the updated part can then later be integrated into the complete reconstruction.
Figure 6.4: Left: Live on-device preview of the scanned object consisting of a point cloud. Right: State of the art textured mesh reconstruction of the same images from an offline system after 15 min of processing time on a high class desktop GPU.
6.3 Sparse Scene Reconstruction

Until now, the applications mostly focused on single objects where it was feasible to move the camera around and to take images from all sides. In these use cases the stereo point cloud approach works well, however, other application opportunities require a more exploratory motion. For example, an user walking outdoors wants to scan the geometry of the environment with the mobile phone. This could be the case for augmented reality applications or computer games where for example enemies should jump around building corners and charge towards the player. To be able to realistically display that, the building walls and the ground plane need to be estimated. One could use the same stereo approach in this case as well, but since the depth ranges are quite large in case of outdoor scenes the computations would become expensive. There is a different set of reconstruction methods that try to solve this case by only using the sparse 3D points from the structure from motion map [90, 91, 92]. The basic idea is to run a Delaunay triangulation in 3D of all map points and then carve all tetrahedron that are intersected by observation rays from the cameras to the 3D points. The interface between the non-carved solid and the intersected tetrahedrons represent the surface of the environment. This approach can also be used to cover areas where stereo is not able to recover the surface to approximate the scene geometry, for example in indoor scenes with white walls or uniformly colored floor. Figure 6.5 shows an implementation of the surface reconstruction by Ioannis Mariggis based on the sparse mapping system presented in this thesis.

This approach is efficient enough to run in real time in parallel to the tracking and mapping. It allows for immediate feedback which is important in dynamic applications like games and helps to improve augmented reality applications by allowing for smooth handling of occlusions. A drawback is the low resolution of the resulting mesh and the strong approximations of the surface geometry due to the sparseness of the map points, which can partly be compensated with the high resolution of the texture. However, texturing will be suboptimal
6.3 Sparse Scene Reconstruction

Figure 6.5: Sparse mesh reconstruction of a workshop room, left with texture, right shows the surface normals. The algorithm can run live on the phone, the surface mesh appears with a delay of few hundreds of milliseconds after the map points were created. These results are from an early state of the algorithm, not all triangles were textured and no smoothing was applied. The reconstruction can be improved by densifying the sparse point cloud for example with the computed depth maps. However, the biggest advantage of this approach is already visible; surfaces that are difficult to reconstruct with stereo-based approaches, are closed and can be mapped with a texture. The accuracy is not yet high, however, for use cases where high accuracy is not needed, the surface from this kind of reconstruction approach can be good enough. These results are created by the implementation by Ioannis Mariggis based on the sparse mapping presented in this thesis.
if the reconstructed model deviate too much from the true surface. In these cases, the model can be refined by densifying the map with additional points by subdividing the surface triangles and estimating the depth of the additional points, or by using the depth maps to increase the mesh surface resolution. This can happen in a background process while the user continues to explore the scene to finally result in a high quality reconstruction.
Chapter 7

Conclusion

In this thesis, the first dense stereo-based system for live interactive 3D reconstruction on mobile phones was proposed. We leveraged the resources offered by current smartphones and addressed the challenges posed by the underlying hardware limitations. To this end, multiple novel solutions covering usage of inertial measurement units to estimate absolute scale and robustify the camera tracking and proposing a dense reconstruction and fusion approach for accurate point clouds were proposed.

In Chapter 3 we described the fully automatic system that does not require markers or any other specific settings for initialization. The system performs feature-based tracking and mapping in real time. It leverages full inertial sensing in position and orientation to estimate the metric scale of the reconstructed 3D models and to make the process more resilient to sudden motions. The system offers an interactive interface for casual capture of scaled 3D models of real-world objects by non-experts. The approach leverages the inertial sensors to automatically select suitable keyframes when the phone is held still and uses the intermediate motion to calculate scale. Visual and auditory feedback is provided to enable intuitive and fool-proof operation. We
Chapter 7 Conclusion

propose an efficient and accurate multi-resolution scheme for dense stereo matching which makes use of the capabilities of the GPU and allows to reduce the computational time for each processed image to about 2-3 seconds on a Samsung Galaxy S3.

In Chapter 4, we propose a method that is based on sparse, very small patches and incorporates the minimization of photometric error directly into the EKF measurement model so that inertial data and vision-based surface measurements are used simultaneously during camera pose estimation. The tight integration of direct surface measurements and inertial data allows to track image regions that are difficult to tackle with approaches that rely on feature trackers like KLT for example line-like structures in images. We show optimizations to the formulation that allow for an efficient implementation that runs in real-time on a standard CPU and can be implemented on mobile platforms as well.

In Chapter 5, we present an efficient and accurate scheme for integrating multiple stereo-based depth hypotheses into a compact and consistent 3D model. Thereby, various criteria based on local geometry orientation, underlying camera setting and photometric evidence are evaluated to judge the reliability of each measurement. Based on that, the proposed fusion technique justifies the integrity of the depth estimates and resolves visibility conflicts.

7.1 Future Work

This thesis demonstrated that todays smartphones can be used to create 3D models directly on the device itself. This opens up possibilities that were only possible with specialized hardware equipped additional sensors like Google Tango. There are many open problems left to be solved in the presented system. The most obvious is the missing closed surface reconstruction from the dense point cloud on device. Chapter 6 sketched two methods. Additionally, there are many different approaches that have their specific strengths in certain use cases. Volumetric fusion approaches are a popular research topic. Such methods
were used in some impressive systems demonstrating their strengths in creating watertight models. Additionally, it is possible to efficiently track the modeled surface in a dense manner, which can reduce the tracking drift to a large extent. However, these methods require special care to be efficient enough for an implementation on a mobile device. Fast processing of the incoming new data is crucial in these algorithms because the model can only be tracked reliably where it was correctly reconstructed. If the reconstruction takes too long, the whole pipeline will fail. This fact limits the possible resolution of the volume. Therefore, it would be interesting to come up with methods that combine the advantages of systems that can sparsely track a scene without the need of a finished volumetric fusion. This would enable the quality of higher resolution volumes without reducing the tracking processing time. As a first step, the model tracking concept can be included in the presented system in a straightforward manner by running a photometric minimization for every new keyframe using the point cloud reconstructed up to that moment.

The presented system heavily focuses on stereo-based point cloud reconstruction but there are many other surface reconstruction methods (see Chapter 2) that could be used for objects that are difficult or impossible to fully reconstruct with the current approach. Specific adaptations to certain objects by using shape priors can allow for easier scanning of pre-known objects that are difficult to scan otherwise [93]. Examples for this would be objects with difficult surfaces like glossy cars or use cases where it is hard to take images from all necessary view points like when scanning the own heads including the back side. Here shape priors can improve the result by roughly knowing how the probable surface should look like.

The camera tracking can still be improved with a more advanced system like in [19] and due to the rolling shutter cameras used in mobile phones the resulting effects of these sensors should be compensated. This requires changes for the tracking, mapping and dense reconstruction modules of the pipeline.

To improve the user interface, the system could be extended with automatic detection of the object to be scanned and according adap-
tions to the algorithms. Guidance for unexperienced users should be integrated to help them efficiently create 3D models by for example showing the required camera poses for the scanning process or propose additional positions for the camera to further improve the result.
Bibliography


Bibliography


the bayes tree. The International Journal of Robotics Research (2011) 0278364911430419


[91] Yu, S., Lhuillier, M.: Incremental reconstruction of manifold surface from sparse visual mapping. In: 3D Imaging, Modeling,
Bibliography

Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on, IEEE (2012) 293–300
