Université de Nice - Sophia Antipolis

ÉCOLE DOCTORALE STIC

Sciences et Technologies de l'Information et de la Communication

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Mention : INFORMATIQUE

présentée par

Tommi TYKKÄLÄ

Équipe d'accueil : Autonomous Navigation and Mapping Systems, CNRS-I3S, UNS, France
et Machine Vision and Pattern Recognition Laboratory, LUT, Finland

---

REAL-TIME IMAGE-BASED RGB-D CAMERA MOTION TRACKING AND ENVIRONMENT MAPPING

Supervised by Dr. Andrew I. Comport[1] and Dr. Joni Kämäräinen[2]
[1] Autonomous Navigation and Mapping Systems, CNRS-I3S, University of Nice Sophia-Antipolis
[2] Machine Vision and Pattern Recognition Laboratory, Lappeenranta University of Technology

Soutenue publiquement le 4 septembre 2013 devant le jury composé de

Andrew I. COMPORT	Chargé de Recherche CR1 (CNRS)	Directeur de thèse
Joni K. KÄMÄRÄINEN	Lappeenranta University of Technology	Directeur de thèse
Shahram IZADI	Microsoft Research, Cambridge	Rapporteur
Eric MARCHAND	INRIA, Rennes	Rapporteur
José Maria Martinez MONTIEL	Universidad de Zaragoza	Rapporteur
Christan BARAT	Rechercher à I3S, Sophia-Antipolis	Examinateur
Sebastien CLERC	Thales-Alenia Space	Membre invité

Thesis is a joint-project between



The work has been supported by





European Union
European Regional Development Fund





"Nothing will work unless you do."
Maya ANGELOU

CONTENTS

Acknowledgements viii

Acronyms ix

1 Introduction 1
- 1.1 Objectives . . . 2
- 1.2 Contribution and publications . . . 3
- 1.3 Outline of the thesis . . . 5

I Background 7

2 Multi-view Geometry and Estimation 9
- 2.1 Perspective camera model . . . 9
  - 2.1.1 Extrinsic matrix structure . . . 10
  - 2.1.2 Perspective projection . . . 11
  - 2.1.3 Lens distortion model . . . 11
  - 2.1.4 Intrinsic matrix structure . . . 12
  - 2.1.5 The inverse model . . . 13
- 2.2 3D point initialization . . . 14
  - 2.2.1 Midpoint triangulation . . . 14
  - 2.2.2 Hartley-Sturm triangulation . . . 15
  - 2.2.3 Rectified stereo triangulation . . . 16
- 2.3 Estimation using Nonlinear Least Squares Minimization . . . 17
  - 2.3.1 Levenberg-Marquardt . . . 18
  - 2.3.2 Gauss-Newton . . . 18
  - 2.3.3 Random sample consensus . . . 19
  - 2.3.4 M-estimators . . . 20
  - 2.3.5 Iteratively re-weighted least squares . . . 21
  - 2.3.6 Linear system solvers . . . 21
    - 2.3.6.1 Conjugate gradient method . . . 22
    - 2.3.6.2 Cholesky factorization . . . 24

2.4 3D point refinement using multiple views 24
2.5 Geometrical pose estimation methods 25
2.5.1 Generating small motion 26
2.5.2 Matrix normalization 29
2.5.3 3D-to-2D 29
2.5.4 3D-to-3D 31
2.5.5 Statistical matching 32
2.6 Bundle adjustment for simultaneous estimation 33
2.6.1 Sparse structure 34
2.6.2 Marginalization 35
2.6.3 Extended Kalman Filter 36
2.7 Feature-based simultaneous localization and mapping . . . 37
2.7.1 Feature points as measurements 37
2.7.2 Loop-closure 38
2.7.3 PTAM 38
2.7.4 FrameSLAM 39
2.8 Dense tracking and mapping 40
2.8.1 Microsoft Kinect 40
2.8.1.1 Extended Caltech calibration 41
2.8.1.2 Oulu calibration 41
2.8.1.3 Depth noise removal by bilateral filtering . 42
2.8.2 KinectFusion 43
2.8.3 Multi-resolution surfels 44

3 Direct Image-based Estimation 45
3.1 Image registration techniques 46
3.1.1 Lukas-Kanade optical flow 46
3.1.2 Remarks on image gradient computation 47
3.1.3 Image registration using homography mapping . . . 48
3.1.4 Image registration using a rigid 3D structure 49
3.1.5 Inverse compositional image alignment 50
3.1.6 Multi-resolution pyramid for better convergence . . . 51
3.2 Direct stereo matching methods 53
3.2.1 Semi-global block matching 55
3.2.2 Stereo camera and calibration 55
3.2.3 Bayer filtering 56
3.3 Quadrifocal stereo tracking 57
3.3.1 Stereo cost function 58
3.4 Pixel selection 59
3.5 Lighting variations 60
3.6 Efficient Second-order Minimization 60
3.7 Photometrical structure refinement 61
3.8 Direct localization and mapping 63
3.8.1 From outdoor stereo systems to indoor environments 64

3.8.2 DTAM 64

II Contributions **66**

4 Efficient stereo tracking by variance bounded disparities **68**
4.1 Trifocal tensor warping 69
4.2 Estimation in two phases 70
4.3 Disparity initialization and propagation 71
4.4 Experiments 72
4.5 Analysis and limitations 74

5 Robust tracking by concurrent pixel and depth matching **81**
5.1 Combining appearance and structure in cost function 82
5.1.1 Bi-objective minimization 82
5.1.2 Balancing the cost by λ 83
5.1.3 Multi-resolution and depth filtering 84
5.1.4 Hybrid pixel selection 84
5.2 Simulation experiments 85
5.3 Results on PRoVisG MARS 3D Challenge 86
5.3.1 Depth map generation 87
5.3.2 Pose estimation 87
5.3.3 Results 88
5.4 Analysis and limitations 88

6 Real-time RGB-D tracking for a low-end GPU **92**
6.1 Tracking modes 93
6.1.1 Incremental dense tracking 93
6.1.2 Keyframe-based dense tracking 93
6.1.3 SLAM mode 95
6.2 Features 96
6.2.1 Embedding distortions in warping function 96
6.2.2 Tolerating dynamic foreground 97
6.3 Scalable GPU tracking 97
6.3.1 Warping 98
6.3.2 M-estimator 98
6.3.3 Linear system reduction 99
6.3.4 Evaluating matrix exponential 101
6.3.5 Selecting points on GPU 101
6.3.6 Vertex attributes at different stages 103
6.3.7 Preprocessing RGB images 103
6.3.8 Point cloud from raw disparity map 104
6.3.9 Online visualization issues 104
6.4 Accuracy 105

6.5 Results 106

7 Watertight and textured 3D reconstructions by RGB-D tracking 109
7.1 Depth map fusion using RGB data 110
7.2 Optional bundle adjustment for a 3D model 113
7.2.1 Interactive editor for bundle adjustment 113
7.3 Watertight polygonization 114
7.4 Mesh texturing 117
7.5 Memory consumption 118

8 Augmented Reality in Live Television Broadcasting 121
8.1 System 122
8.1.1 AR graphics using Panda3D 123
8.1.2 Motion capture system for live character animation . 124
8.1.3 AR composition using depth maps 124
8.2 Studio lighting and Microsoft Kinect 126
8.3 Tracking configurations 127
8.3.1 RGB-D sensor towards the scene 128
8.3.2 RGB-D sensor towards the floor 129
8.4 3D modeling of a studio environment 131
8.4.1 Depth map noise in a studio environment 132
8.4.2 Depth map filtering for studio model 132
8.5 Experiment: Tracking accuracy 133
8.6 Constraints 136
8.7 Summary 136

9 Conclusions 139
9.1 Perspectives 140

10 Résumé en français 142
10.1 Contributions et publications 145
10.1.1 Publication I 145
10.1.2 Publication II 146
10.1.3 Publication III 146
10.1.4 Publication IV 147
10.2 Conclusions et perspectives 147
10.2.1 Perspectives 148

A Appendix: Simulated Asteroid datasets

References

ACKNOWLEDGEMENTS

I thank everyone who have participated in this project and provided valuable feedback and support. Foremost, I have received tireless feedback and improvement ideas from my supervisors Andrew I. Comport and Joni-Kristian Kämäräinen, who have helped me in polishing the publications and this manuscript into a clear and concise form.

As a double-degree project, both I3S laboratory in Sophia-Antipolis and MVPR/LUT in Kouvola provided research environment and facilities to carry out research. I would like to thank Sebastien Clerc in Thales-Alenia Space for funding the project and introducing related industrial research methods. The first experiment was carried out in INRIA using a turn-table and a stereo camera, which were kindly provided by Patrick Rives and A.I. Comport. Maxime Meilland showed a reference result for the sequence and Mathieu Seiler was there for general assistance and support. With the second publication, I'm happy for inspiring collaboration with Cédric Audras. Based on Cédric's preliminary results with the Microsoft Kinect sensor, it was easy to see that RGB-D tracking has potential in indoor applications. During the first half of the project, Vesa Jääskeläinen arranged professional svn maintenance and support for me, which really made software development possible.

In MVPR/LUT, I'm glad that Joni-Kristian Kämäräinen persuaded me to switch to Ubuntu Linux and offered a modern laptop. With kind help of Jukka Lankinen, it was quick and easy to learn the best Linux tools and practises. Developing a real-time GPU implementation from scratch took a lot of time and effort, but it was also a great learning experience. Many experiments were carried out in MedusaTV studio in Kouvola, where camera tracking was benchmarked in live AR use. Marko Siitonen, Heikki Ortamo, Sauli Simola, Taija Lumitähti and Jori Pölkki consulted me with television studio practises, offered support with motion capture system and helped with the relevant tests. Big thanks to Hannu Hartikainen at Aalto University for helping out with Kinfu reference measurements.

Last but not the least, I thank all my lab mates for entertaining discussions during the project!

ACRONYMS

SLAM	Simultaneous Localization And Mapping
RANSAC	Random Sample Consensus
EKF	Extended Kalman Filter
IRLS	Iteratively Re-weighted Least-Squares
NLSQ	Non-Linear Least SQuares
BA	Bundle Adjustment
BRDF	Bidirectional Reflectance Distribution Function
IR	Infra-Red
RGB	Red-Green-Blue
RGB-D	Red-Green-Blue-Depth
GPU	Graphics Processing Unit
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
AR	Augmented Reality
3D	Three-dimensional
2.5D	Two-dimensional + Depth
2D	Two-dimensional
1D	One-dimensional
TV	Television
SVD	Singular Value Decomposition
TSDF	Truncated Signed Distance Function
IIR	Infinite Impulse Response
CCD	Charge-Coupled Device
FOV	Field-Of-View
ICP	Iterative Closest Point
MAD	Median Absolute Deviation

- CHAPTER 1 -

INTRODUCTION

In computer vision, camera pose estimation and 3D structure refinement are approached by defining cost functions whose minimum corresponds to desired model configuration. Because these two problems are by nature geometrical, original image measurements are often replaced by 2D feature points, which are extracted from the images. Feature points are distinctive local regions, such as corners and edges, which can be matched across multiple images. Finally, a geometrical cost function determines which parameters are ideal in relation to all 2D feature point observations. Despite that geometrical cost metric can provide precise estimates, the estimation process paradoxally utilizes only indirectly the original image data. 2D points must be extracted from images and matched in multiple views. This is the weak link of the estimation process, because extraction and matching errors can not be avoided. Especially matching errors must be addressed by techniques such as RANSAC [18, 55] and M-estimators [27] to increase tolerance to gross outliers. Feature extraction, also, is imprecise because the points are likely to have an offset to the ideal projection of a 3D point. For example, viewing angle and lighting conditions affect on the feature extraction process.

In this work, image based estimation methods, also known as *direct methods*, are studied which avoid feature extraction and matching completely. Cost functions are directly defined using raw pixel measurements and the goal is to produce precise pose and structure estimates. The presented cost functions minimize the sensor error, because measurements are not transformed or modified. In photometric camera pose estimation, 3D rotation and translation parameters are estimated by minimizing a sequence of image based cost functions. These cost functions are non-linear due to perspective projection and lens distortion. In image based structure refinement, 3D structure is refined by minimizing appearance variance to

the reference view using a number of additional views. Image based estimation methods are usable whenever the Lambertian illumination assumption holds, where 3D points have constant color despite viewing angle. The main application domains considered in this work are indoor reconstructions, robotics and augmented reality, which all benefit from more precise 3D camera tracking and environment mapping.

§ 1.1 OBJECTIVES

The overall project goal is to improve image based estimation methods, and to enable their use in real-time applications. The main questions for this work are

- *"What is an efficient formulation for an image based 3D pose estimation and structure refinement task?"*
- *"How to organize computation to enable an efficient real-time implementation?"*
- *"What are the practical considerations of using image based estimation methods in applications such as augmented reality and 3D reconstruction?"*

Two application cases are considered. In the first one, an asteroid reconstruction mission is of interest, where multi-view inference is used for autonomous pose estimation and 3D reconstruction. The first two publications describe developments carried out in I3S laboratory in Sophia-Antipolis in an aim to bring state-of-the-art computer vision to space applications in collaboration with Thales-Alenia Space. Particularly computational efficiency and tracking robustness were considered. Since standard benchmarking for visual odometry was not available, it was developed to monitor the development in accuracy and robustness. Blender software was used to generate synthetic datasets along with a ground truth trajectories. Also turn-table was used to test visual odometry with more realistic input.

During the project, the Microsoft Kinect sensor was released, which meant that relatively accurate RGB-D measurements could be captured in textureless indoor environments at 30Hz using a structured IR light pattern. Compared to depth maps generated from stereo camera input, structural accuracy became much more consistent and image content independent. I made a request to transform this PhD project into a double-degree with Lappeenranta University of Technology to be able focus on television production studio environments. This change simplified data acquisition, since real data could be easily captured in the actual application environment.

The goal in the second part of this thesis was to develop a 3D camera tracking solution to television production studios, which enables rendering AR graphics to live broadcasts. The film industry knows this technique as *matchmoving*, which is traditionally done in post-processing using semi-automatic trackers [17]. A tracker is a tool which estimates a 3D camera motion trajectory based on observed 2D point tracks [85, 86]. In studios, online matchmoving is typically done using an external motion capture system [49], which tracks a camera in real-time based on passive markers attached to the camera. For example, Brainstorm Multimedia software is then used to render view-dependent graphics in real-time [7]. Although motion capture is precise and enables a large operating volume, the total price of a professional system is currently $200 - 500$k€. Also cheaper systems exists, such as NaturalPoint Optitrak [51], which cost around 25k€, but their precision and capture rate are lower.

A real-time image based tracker was developed to allow affordable and automatic camera tracking, which KyAMK University of Applied Sciences in Kouvola could use for educational purposes. The project was funded by European Regional Development Fund and The Federation of Finnish Technology Industries. Particularly, a real-time implementation was necessary, and drift was not an option. Therefore also 3D model acquisition method must be developed to obtain a fixed reference. Some tolerance with foreground actors was required in a setup where the program content is directly used to camera pose estimation. As alternative approach a RGB-D sensor was pointed toward a textured carpet to avoid scene manipulation. A lot of experimenting was carried out in the studio environment along with studio people to obtain a usable system. The toolset was tested using a wide range of input videos, both synthetic and real. Finally, the tracking solution was experimented both in television programs with live augmented reality content and in 3D reconstruction of indoors.

§ 1.2 CONTRIBUTION AND PUBLICATIONS

The following publications were written during the project:

> Publication (i)
> "A Dense Structure Model for Image Based Stereo SLAM", T. M. Tykkälä, and A.I. Comport, IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, May 2011,
>
> Publication (ii)
> "Direct Iterative Closest Point for Real-time Visual Odometry", T. M. Tykkälä, C. Audras, and A. I. Comport, Workshop on Computer Vision in Vehicle Technology: From Earth to Mars in conjunction with the International Conference on Computer

Vision (CVVT/ICCV), Barcelona, Spain, Nov 2011,

Publication (iii)
"RGB-D Tracking and Reconstruction for TV Broadcasts", T.M. Tykkälä, H. Hartikainen, A.I. Comport, and J-K. Kämäräinen, 8th International Conference on Computer Vision Theory and Applications (VISAPP), Barcelona, Spain, Feb 2013,

Publication (iv)
"Live RGB-D Camera Tracking for Television Production Studios", T.M. Tykkälä, A.I. Comport, J-K. Kämäräinen and H. Hartikainen, Journal of Visual Communication and Image Representation, Elsevier, Apr 2013.

In publication (i), simultaneous estimation of 3D pose and structure parameters is formulated using an image-based SLAM cost function [82]. One use case for such cost function is in autonomous navigation and mapping of space environments. In a recent Itokawa asteroid sampling mission, also asteroid images were collected using an autonomous robot. These image were then used in manual 3D reconstruction of Itokawa asteroid. In our work, the disparity values are treated as free parameters whose joint covariance with pose parameters is estimated. Image based disparity variances are obtained by marginalizing the motion parameters out. These variances are then used to bound the estimation of the next disparity map. This process boosts disparity map computation, which is commonly a bottleneck. The topic was studied in collaboration with Thales-Alenia Space, who funded a part of this work and provided a realistic orbiting trajectory around Itokawa. The trajectories and available asteroid 3D models were used to evaluate the method.

In publication (ii), 2.5D maps are represented as general RGB-D measurements directly without depending on a specific measuring technique [81]. Maintaining as small drift as possible is essential for any visual odometry problem. For reducing the drift, both RGB image discrepancy and depth map discrepancies are concurrently minimized. When the error residual consists of two components with different units, the numerical uncertainty (standard deviations) can also vary between the components. This means that an additional adjustment parameter λ is required to balance the uncertainties in such a way that drift is minimized. To gain computational efficiency, it is proposed that salient image regions are selected using a gradient magnitude histogram. The histogram method reduces computational requirements of pixel selection from $O(n\log n)$ to $O(n)$, because sorting can be fully avoided. The experiments show that pose estimation can be made more precise, especially in cases where Lambertian assumption is only partially valid. The method is also demonstrated with using a real space sequence in PRoVisG MARS 3D Challenge [29].

In publication (iii), a system is presented which uses Microsoft Kinect sensor for augmenting graphics to a live TV broadcast. Live augmented reality is useful when designing TV and film scenes with interactive virtual components. AR graphics are traditionally added in post-processing. In case multiple shots are required, the process easily becomes expensive. Using a live AR system, the scenes can be practised in real-time using sufficient AR quality. When the scenes are captured in their final form, only one post-processing will be required. RGB-D camera tracking is proposed for live AR use in television studios to reduce post-processing needs and lower the costs [84]. During a live AR broadcast, drift is not an option and frequent loop-closure will be required. One method to implement loop-closure is through keyframes. When tracking is defined relative to a set of keyframes, the drift will not cumulate in time. A pre-defined keyframe model is also necessary when avoiding moving actors in the scene. The static model provides distance values and intensity values which are compared with the current RGB-D measurement. By filtering out image regions with too large discrepancy to the model, foreground motion does not bias the tracking process. The keyframes are not allowed to contain dynamic regions, because the appearance changes are modeled only by mathematical camera model.

In publication (iv), a real-time RGB-D tracker is developed which benefits from computational capacity and scalability of a GPU [80]. The implementation is one of the first ones for GPU. Several GPU optimizations are presented and discussed which enable real-time performance even on a low-end GPU. The system is designed to be low-cost and it only requires a RGB-D sensor and a laptop. RGB-D tracking is experimented in a TV production studio with and without foreground actors. Novel tools are built to pre-filter and refine 3D models generated by RGB-D tracking. The accuracy is compared with a recent KinectFusion system which concurrently tracks 3D camera pose and builds a voxel-based 3D model [53]. The comparison shows how the proposed method outperforms KinectFusion in a larger operating volume. Using a keyframe based 3D model as reference requires a nearest neighbor metric which measures pose similarity using a single metric. A metric is proposed which unifies angular and translational units. This thesis also provides examples how photorealistic and watertight appartment models are generated by a RGB-D mapping process (Chapter 7). The models are stored in a standard format to allow online visualization and 3D printing.

§ 1.3 Outline of the thesis

The thesis is structured as follows. In Chapter 2, the basic methods of geometrical, vision-based pose and structure estimation methods are sum-

marized. In Chapter 3, the image-based estimation methods are presented which give context to this thesis. The following chapters describe the contributions. In Chapter 4, disparity maps computation becomes more efficient by a re-localization scheme, where a part of disparity map is estimated within temporally propagated bounds. In Chapter 5 camera tracking dependency on image quality is reduced by simultaneous minimization of depth map error. In Chapter 6, a computationally efficient real-time GPU implementation is developed which minimizes a image based cost function, and produces robust and precise pose estimates. In Chapter 7 the process is described how watertight, textured 3D models can be produced by RGB-D tracking process. In Chapter 8 the real-time tracking and mapping technology is demonstrated in a real application case in a television production studio. The camera tracking is used to render interactive 3D graphics into a live television broadcast. The conclusions are made in Chapter 9.

Background

- CHAPTER 2 -

MULTI-VIEW GEOMETRY AND ESTIMATION

§ 2.1 PERSPECTIVE CAMERA MODEL

To understand the relationship between a 3D scene and it's 2D projection images, a mathematical camera model needs to be described. The model is fundamental for computer graphics, computer vision and augmented reality, because it enables rendering 2D images from 3D models (*the forward problem*), reconstructing 3D models based on 2D images (*the inverse problem*), and rendering graphics using the estimated camera poses. The computer vision field has traditionally been focused on the inverse problem. However, due to rapid development of RGB-D sensors, the forward process has also become increasingly important, because it enables generating synthetic but photo-realistic 2.5D images. 2.5D images are color images associated with a depth map. In this work, image-based modeling enables precise photometric cost functions for camera pose estimation and structure refinement. The notations given in this section are at times less general than the ones defined by Hartley [21] to focus only on the transformations required in this project. Namely, projective 3-spaces ($\mathbb{P}^3$) are not considered, because the 3D points are assumed to have finite coordinates.

Essentially a camera model is a geometrical mapping $\mathcal{P} : \mathbb{R}^3 \Rightarrow \mathbb{R}^2$, which transforms 3D points $\mathbf{P}_k$ into 2D image points $\mathbf{p}_k$. The components of the model are:

1. *Extrinsic* part, the 3D transformation of the geometry into a camera reference frame. The transformation is defined by rotation matrix $\mathbf{R} \in \mathbb{SO}(3)$ and translation vector $\mathbf{t} \in \mathbb{R}^3$, which are often embedded in a 4×4 matrix to allow better manipulation properties.

2. Perspective projection from 3D to 2D, by normalizing a homogeneous 3D coordinates by the depth (3rd component). The normalization function $\mathcal{N}(\mathbf{P})$ is defined explicitly to allow differentiation.

3. Lens distortion modeling using radial and tangential offsets using the function $D(\mathbf{p}, \boldsymbol{\alpha})$ whose coefficients $\boldsymbol{\alpha} \in \mathbb{R}^5$ are separately calibrated for every physical lens.

4. *Intrinsic* part, 2D transformation to the image points by 3×3 intrinsic matrix $\mathbf{K}$, which is the result of camera calibration. This transformation models image resolution, image aspect ratio, pixel skew and projection center on the image plane. A projection matrix Π is used to extract the final 2D coordinates. Π is used because 3-component homogeneous form is redundant.

The full camera model is

$$\mathbf{p} = \Pi \mathbf{K} D(N(\mathbf{R}\mathbf{P} + \mathbf{t}), \boldsymbol{\alpha}), \text{ where } \Pi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{2.1}$$

The color generation is typically modeled using a Bidirectional Reflectance Distribution Function (BRDF). BRDF defines how much an infinitesimally small surface patch at 3D point $\mathbf{P}$ reflects radiance into projected point $\mathbf{p}$ using various lighting parameters. In computer vision, Lambertian BRDF is often assumed for simplicity. Lambertian surfaces reflect the same color uniformly into all directions. Thus, images captured from multiple viewpoints will be directly color-wise comparable.

2.1.1 Extrinsic matrix structure

The world coordinate system W_C is the fixed coordinate system where all 3D geometry is finally transformed into. One or several camera coordinate systems C_C can be defined relative to W_C. To map 3D points $W_C \Rightarrow C_C$, a 4×4 extrinsic matrix $\mathbf{T}$ is required, which is composed of 3D rotation and translation.

The extrinsic matrix structure is

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} r_{x1} & r_{y1} & r_{z1} & t_1 \\ r_{x2} & r_{y2} & r_{z2} & t_2 \\ r_{x3} & r_{y3} & r_{z3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \in \mathbb{SE}(3), \tag{2.2}$$

where $\mathbf{R}$ is 3×3 rotation matrix and $\mathbf{t}$ is translation vector. The rows $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ are unit vectors which represent the X, Y, and Z-axes of the camera in W_C.

The camera pose matrix $\mathbf{T}^{-1}$ maps points into the opposite direction $C_C \Rightarrow W_C$, and it is defined by

$$\mathbf{T}^{-1} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R}^T & \mathbf{c} \\ \mathbf{0} & 1 \end{pmatrix}, \tag{2.3}$$

where $\mathbf{c}$ represents the camera origin in W_C. Thus, general matrix inversion procedure is not required.

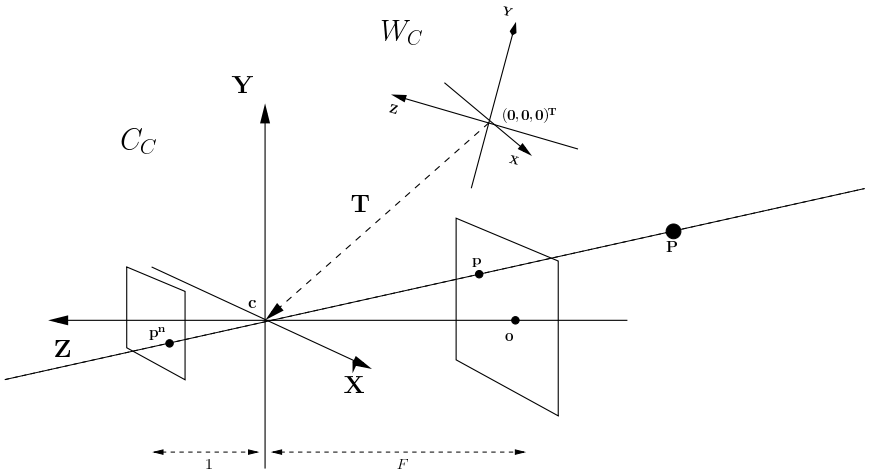


Figure 2.1: *A perspective camera model*

2.1.2 Perspective projection

3D points are converted into projective coordinates (in group $\mathbb{P}^2$) by defining equivalence $(x, y, z) \Leftrightarrow (x/z, y/z, 1)$. This definition performs implicit perspective projection, because 3D points move along 3D ray onto normalized image plane at $z = 1$. However, explicit function $\mathcal{N}(\mathbf{P}) : \mathbb{R}^3 \Rightarrow \mathbb{R}^3$ is necessary to allow differentiation.

The 3D points $\mathbf{P}_k$ in C_C are projected into normalized image points $\mathbf{p}_k^n$ by

$$\mathbf{p}^n = \mathcal{N}(\mathbf{P}) = \begin{pmatrix} p_1/p_3 \\ p_2/p_3 \\ 1 \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}. \tag{2.4}$$

$(u, v, 1)^T$ are general image coordinates which do not depend on the physical camera properties.

2.1.3 Lens distortion model

Real lenses, especially low-cost, produce images which can not be modeled with simple perspective projection. The imaging process contains an

additional non-linear component which performs lens distortion. The lens distortions are typically radial and tangential displacements around the optical axis [6]. The intersection point between the optical axis and the image plane is called *the principal point*. The principal point does not have any distortion and and is often very close to the center of the image.

We use standard Caltech distortion model to obtain distorted image points $\mathbf{p}^d$ by

$$\mathbf{p}^d = D(\mathbf{p}, \boldsymbol{\alpha}) = \begin{bmatrix} \mathbf{p}(1 + \alpha_1 r^2 + \alpha_2 r^4 + \alpha_5 r^6) + \mathbf{dx} \\ 1 \end{bmatrix}, \tag{2.5}$$

where $r^2 = p_1^2 + p_2^2$ and $\mathbf{dx}$ denotes the tangential distortion

$$\mathbf{dx} = \begin{bmatrix} 2\alpha_3 p_1 p_2 + \alpha_4(r^2 + 2p_1^2) \\ \alpha_3(r^2 + 2p_2^2) + 2\alpha_4 p_1 p_2 \end{bmatrix}. \tag{2.6}$$

$\alpha_1, \alpha_2, \alpha_5$ are the radial coefficients, α_3 and α_4 are the tangential coefficients, Since lens distortions are mostly radial distortions, $\mathbf{dx}$ can often be ignored.

This distortion model produces both *pincushion* and *barrel* distortions depending on the parameters. In *pincushion* distortion, the focal length increases with the radius of a lens. *Barrel* distortion, on the other hand, is the opposite, and the focal length decreases with the radius. Both distortion types are illustrated in Figure 2.2.

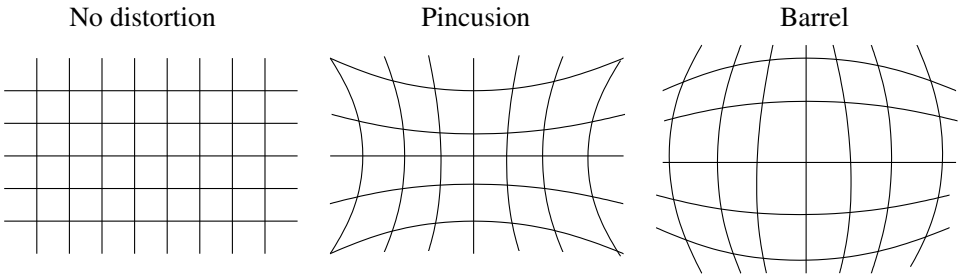


Figure 2.2: *Pincushion and barrel distortion illustrated*

2.1.4 Intrinsic matrix structure

The intrinsic matrix $\mathbf{K}$ scales and translates the normalized image coordinates $\mathbf{p}_k^n$ into pixel coordinates $\mathbf{p}_k$. The image resolution is $(\text{width}, \text{height})$.

$$\mathbf{p} = \mathbf{K}\mathbf{p}^n = \begin{pmatrix} -f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} -f_x u + o_x \\ f_y v + o_y \\ 1 \end{pmatrix}, \tag{2.7}$$

and based on similar triangles f_x and f_y scale the normalized units into pixel units, where

$$f_x = F * s = F * \frac{\text{width}/2}{F \tan \frac{\alpha_x}{2}} = \frac{\text{width}/2}{\tan \frac{\alpha_x}{2}} \quad f_y = a f_x. \tag{2.8}$$

F is the focal length in metric units, s converts the metric units into pixel units, and $a = \frac{\text{height}}{\text{width}}$ is the aspect ratio of the image. α_x is the viewing angle (radian units) in the X-axis direction. Square pixels are assumed when $a = 1$. $\mathbf{o} = (o_x, o_y)^T$ is *the principal point*, which is the projection of $\mathbf{c}$ in pixel coordinates. Here a convention from standard computer graphics libraries is adopted as negative z-axis represents the viewing direction (Fig. 2.1) and the origin of the image plane is in the upper left corner.

2.1.5 The inverse model

The inverse camera model maps image points $\mathbf{p}_k$ into 3d rays $\mathbf{r}_k(t)$ in W_C. The scale t_0 which matches with the first intersection is not known, and requires additional measurements. t_0 can be recovered for example by using multiple views and triangulation, or by direct measurement.

The rays are defined by

$$\mathbf{r}(t) = t\mathbf{R}^T \mathcal{D}^{-1}(\mathbf{K}^{-1} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}, \boldsymbol{\alpha}) + \mathbf{c}. \tag{2.9}$$

The inverse lens distortion model $\mathcal{D}^{-1}(\mathbf{p}, \boldsymbol{\alpha})$ can be formulated either as an image re-sampling problem or a point warping problem. When a distorted source image $\mathcal{I}_d$ is re-sampled by

$$\mathcal{I}(\mathbf{p}_k) = \mathcal{I}_d(\mathbf{K}\mathcal{D}(\mathbf{K}^{-1} \begin{bmatrix} \mathbf{p}_k \\ 1 \end{bmatrix}, \boldsymbol{\alpha})), \tag{2.10}$$

the image $\mathcal{I}$ will be undistorted. After replacing the original input images $\mathcal{I}_d$ by undistorted images $\mathcal{I}$, lens distortion will not require further addressing. The inverse model will be

$$\mathbf{r}(t) = t\mathbf{R}^T \mathbf{K}^{-1} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} + \mathbf{c}. \tag{2.11}$$

The problem is, however, that image undistortion degrades image quality, because interpolation function must be used to obtain intensity values in

between the pixels. $\mathcal{I}$ will also contain regions that do not map into any intensity value in $\mathcal{I}_d$.

The undistorted points $\mathbf{p}_k$ can also be directly estimated. Since $\mathcal{D}(\mathbf{p}, \boldsymbol{\alpha})$ can not be analytically inverted, the first-order Taylor approximation can be used [41]. The approximation works well especially for minor distortions, because the mapping is relatively smooth.

$$\mathcal{D}^{-1}(\mathbf{p}_n, \boldsymbol{\alpha}) = \begin{pmatrix} \mathbf{p}_n + \frac{-\mathbf{p}_n(\alpha_1 r^2 + \alpha_2 r^4 + \alpha_1^2 r^4 + \alpha_2^2 r^8 + 2\alpha_1\alpha_2 r^6)}{(1+4\alpha_1 r^2 + 6\alpha_2 r^4)} \\ 1 \end{pmatrix}, \tag{2.12}$$

where $\mathbf{p}^n$ is the distorted point, $r = p_1^2 + p_2^2$, and $\boldsymbol{\alpha}$ are the distortion coefficients. In this model only radial components α_1 and α_2 are supported and tangential distortions are ignored.

§ 2.2 3D POINT INITIALIZATION

In *triangulation*, a 3D point is produced by intersecting two or more 3D rays. Stereo triangulation methods are relevant when generating dense and sparse depth maps using two views. In this work, stereo triangulation will be used mostly to generate a 3D point cloud from a disparity map. Stereo triangulation methods are, however, also studied in unrectified case, because the asteroid mission two satellites require initial guess of 3D structure (see Sec. A in Appendix).

Calculating an intersection point is purely geometrical problem when noise is not present. However, when the projection points are extracted from images, and the rays are intersected, the 3D point will be contaminated by noise or may not even exist if the rays did not intersect. Two common approaches for triangulation will be described which can tolerate small amount of noise.

2.2.1 Midpoint triangulation

Let there be two rays $\mathbf{r}_1(s) = \mathbf{a} + s\mathbf{b}$ and $\mathbf{r}_2(t) = \mathbf{c} + t\mathbf{d}$. $\mathbf{a}, \mathbf{c} \in \mathbb{R}^3$ are the ray origins, and $\mathbf{b}, \mathbf{d} \in \mathbb{R}^3$ the direction vectors respectively. $s, t \in \mathbb{R}_+$ are the scales of the rays, which are free parameters. The problem is to estimate such s and t which satisfy $\mathbf{r}_1(s) = \mathbf{r}_2(t)$ and thus define the intersection point. This point can be found by minimizing the Euclidean distance between the rays. Euclidean squared distance is defined by

$$D(s,t) = \|\mathbf{r}_1(s) - \mathbf{r}_2(t)\|^2 = \|s\mathbf{b} - t\mathbf{d} + \mathbf{a} - \mathbf{c}\|^2. \tag{2.13}$$

At the intersection point, the derivatives respect to s and t are both zero

$$\frac{\partial D(s,t)}{\partial s} = 2\mathbf{b} \cdot (s\mathbf{b} - t\mathbf{d} + \mathbf{a} - \mathbf{c}) = 0 \tag{2.14}$$

$$\frac{\partial D(s,t)}{\partial t} = 2\mathbf{d} \cdot (s\mathbf{b} - t\mathbf{d} + \mathbf{a} - \mathbf{c}) = 0. \tag{2.15}$$

s_0 and t_0 become

$$t_0 = \frac{(-\mathbf{a}\cdot\mathbf{b} + \mathbf{b}\cdot\mathbf{c})(\mathbf{b}\cdot\mathbf{d})/\|\mathbf{b}\|^2 + \mathbf{a}\cdot\mathbf{d} - \mathbf{c}\cdot\mathbf{d}}{\|\mathbf{d}\|^2 - (\mathbf{b}\cdot\mathbf{d})(\mathbf{b}\cdot\mathbf{d})/\|\mathbf{b}\|^2} \tag{2.16}$$

$$s_0 = \frac{(\mathbf{b}\cdot\mathbf{d})t_0 - (\mathbf{a}\cdot\mathbf{b}) + (\mathbf{b}\cdot\mathbf{c})}{\|\mathbf{b}\|^2}. \tag{2.17}$$

In case the the rays do not intersect, s_0 and t_0 will match with the closest points along the rays. The midpoint is trivially calculated $\mathbf{P} = \frac{\mathbf{r}_1(s_0)+\mathbf{r}_2(t_0)}{2}$ (Figure 2.3a). The intersection may also have infinite solutions, when the projection point coordinates are equal.

The midpoint method does not minimize optimal quantity and can even increase error when compared to selecting either one of the points. For example, when using the midpoint method in a situation where $\mathbf{r}_1(s)$ is a noiseless ray and $\mathbf{r}_2(t)$ is not, the midpoint can not be more precise than $\mathbf{r}_1(s_0)$. This case may occur when the other camera is much closer to the target. A better approach is to weight $\mathbf{r}_1(s_0)$ and $\mathbf{r}_2(t_0)$ with camera distance and angle based weights.

2.2.2 Hartley-Sturm triangulation

Hartley and Sturm presented an optimal triangulation method for projections with Gaussian error distribution [21]. Instead of minimizing 3D distance, a cost function can be directly formulated for the 2D projection points. When $\mathbf{r}_1(s)$ is projected into view 2 and $\mathbf{r}_2(t)$ is projected into view 1, 2D epipolar lines $\mathbf{e}_1$ and $\mathbf{e}_2$ are produced. The problem is then to estimate such image points $\dot{\mathbf{p}}_1$ and $\dot{\mathbf{p}}_2$ which satisfy epipolar constraint $\dot{\mathbf{p}}_1^T\mathbf{F}\dot{\mathbf{p}}_2 = 0$ and minimize distance to the measured points $\mathbf{p}_1$ and $\mathbf{p}_2$ (Figure 2.3b). The fundamental matrix $\mathbf{F}$ is 3×3 matrix which can be estimated using a set of corresponding 2D points or derived from the camera parameters [21].

The cost function is thus

$$c(\dot{\mathbf{p}}_1, \dot{\mathbf{p}}_2) = (\mathbf{p}_1 - \dot{\mathbf{p}}_1)^T\mathbf{W}_1(\mathbf{p}_1 - \dot{\mathbf{p}}_1) + (\mathbf{p}_2 - \dot{\mathbf{p}}_2)^T\mathbf{W}_2(\mathbf{p}_2 - \dot{\mathbf{p}}_2) \tag{2.18}$$

$$\text{subject to } \dot{\mathbf{p}}_1^T\mathbf{F}\dot{\mathbf{p}}_2 = 0, \tag{2.19}$$

where $\mathbf{W}_k$ are the 2D covariances for 2D measurements.

The minimization finally requires estimating the roots of 6th order polynomial [21]. Tossavainen presents a computationally faster approximation [78]. The final 3D point is triangulated from corrected 2D projections

by using the midpoint method which is now guaranteed to produce the exact result.

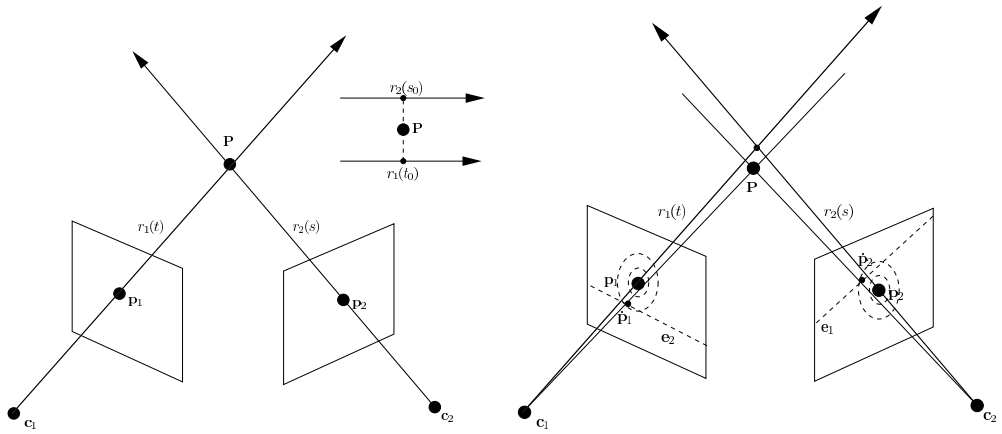


Figure 2.3: *a) Midpoint triangulation, b) Hartley-Sturm triangulation.*

2.2.3 Rectified stereo triangulation

Two views are *rectified*, when the epipolar lines are strictly horizontal. If this is not the case, it is possible to rectify the stereo pair by re-sampling one or both images. For a rectified stereo view, the rays will always intersect and triangulation becomes simpler. The corresponding projection points $\mathbf{p}_1 = (\ u,\ v,\ 1\ )^T$ and $\mathbf{p}_2(d) = \mathbf{p}_1 + (\ d,\ 0,\ 0\ )^T$ can be encoded by 1D disparity parameter d.

Based on two similar triangles

$$1)\ \frac{X}{X - x_1} = \frac{Z}{F + Z},\ \ 2)\ \frac{b - X}{b - X + x_2} = \frac{Z}{F + Z}, \tag{2.20}$$

where (X, Y, Z) is a 3D point in the view 1, F is the focal length in metric units, b is the distance between the focal points, and x_k are the projection x-coordinates in different views. Figure 2.4 illustrates the case geometrically.

The first equation implies $X = -Zx_1/F$. By substituting X in the second equation

$$Z = \frac{Fb}{x_2 - x_1} \Rightarrow z(d) = \frac{Fb}{d} \tag{2.21}$$

follows. Thus, the depth depends on the disparity value d, the focal length F and the baseline b.

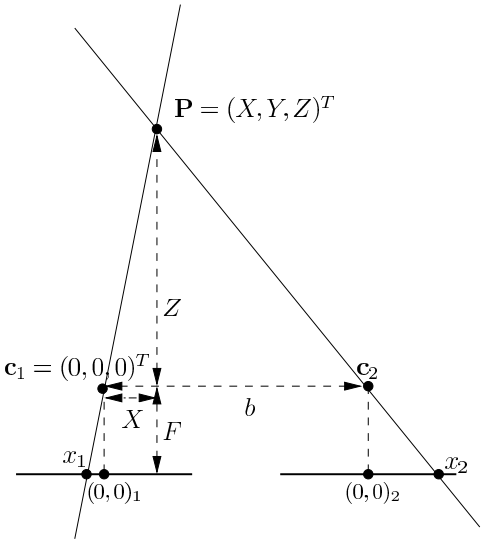


Figure 2.4: *Rectified stereo triangulation.* **P**, *optical center* c_k *and the projection point* x_k *are on the same line based on perspective projection model. Relation between the disparity and depth value is determined by similar triangles. Right triangles are formed from point* **P** *at distances* Z *and* $Z + F$*, where* F *is the focal length.*

§ 2.3 Estimation using Nonlinear Least Squares Minimization

Non-linear least squares minimization is widely used in computer vision problems to estimate parameters when a cost function has been defined. When a mathematical model exists, which generates data similar to observed measurements, its parameters can be estimated using NLSQ. Typically the cost function is defined to be a sum of squared errors, because NLSQ minimization methods can be used. When an initial guess $\mathbf{x}_0$ exists, NLSQ produces an estimate which has the minimal cost. Due to limited number of iterations, cost smoothness and numerical inaccuracy, the estimate will still contain a small error. Let a vector function $f : \mathbb{R}^m \Rightarrow \mathbb{R}^n$ be the model, whose initial $\mathbf{x}_0 \in \mathbb{R}^m$. The problem is to estimate $\mathbf{x} \in \mathbb{R}^m$ minimizing scalar error $e(\mathbf{x}) = \mathbf{e}(\mathbf{x})^T\mathbf{e}(\mathbf{x}) = \big(\mathbf{m} - f(\mathbf{x})\big)^T\big(\mathbf{m} - f(\mathbf{x})\big) = \|\mathbf{m} - f(\mathbf{x})\|^2$, where $\mathbf{m} \in \mathbb{R}^n$ is the measurement vector.

In 3D computer vision problems, the model f is non-linear due to perspective projection, lighting effects, shadows, occlusions and complicated surface models. However, if f is a piece-wise smooth mapping, the cost function can be locally approximated by the first-order Taylor expansion

$\mathbf{e}(\delta_\mathbf{x}) = \mathbf{e}_0 + \mathbf{J}\delta_\mathbf{x}$, where $\mathbf{e}_0 = \mathbf{e}(\mathbf{x}_0)$ is the current residual, $\mathbf{J} = \frac{\partial \mathbf{e}(\mathbf{x}_0)}{\partial \mathbf{x}}$ is the Jacobian and $\delta_\mathbf{x}$ is a parameter increment to be estimated.

The scalar error function becomes

$$e(\delta_\mathbf{x}) = \frac{1}{2}\mathbf{e}(\delta_\mathbf{x})^T\mathbf{e}(\delta_\mathbf{x}) = \frac{1}{2}\mathbf{e}_0^T\mathbf{e}_0 + \delta_\mathbf{x}^T\mathbf{J}^T\mathbf{e}_0 + \frac{1}{2}\delta_\mathbf{x}^T\mathbf{J}^T\mathbf{J}\delta_\mathbf{x} \tag{2.22}$$

where the derivative is zero when

$$\mathbf{J}^T\mathbf{J}\delta_\mathbf{x} = -\mathbf{J}^T\mathbf{e}_0 \ . \tag{2.23}$$

This linear system is also called *normal equation*, and it is solved by a method which fits the exact matrix structure involved. Popular choices are Cholesky decomposition and conjugate gradient method (Section 2.3.6). If the cost function assumptions are valid, the estimated increment satisfies $e(\mathbf{x}_0 + \delta_\mathbf{x}) < e(\mathbf{x}_0)$. However, increment $\mathbf{x}_0 + \delta_\mathbf{x}$ must be estimated multiple times in cases where the mapping is not linear. The following sections outlines few strategies to estimate the local minimum.

2.3.1 Levenberg-Marquardt

Levenberg-Marquardt (LM) optimization is typically used as a local optimization method for non-linear least squares minimization, because it has good convergence properties even when the initial guess is relatively far away from the optimum [37]. LM has adaptive step control which adjusts the rate of descend to take bigger steps when the function is flat. LM can, to some extent, cope with ill-posed problems where the measurements do not imply sufficient constraints to solve a unique estimate. In computer vision problems, insufficient texturing may lead into an ambiguous problem. The steps are computed by using Tikhonov regularized normal equation

$$\delta_\mathbf{x} = -(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})^{-1}\mathbf{J}^T\mathbf{e}, \tag{2.24}$$

where $\delta_\mathbf{x}$ is the parameter increment which minimizes the error, $\mathbf{I}$ is identity matrix and $\mathbf{e}$ is the residual vector. μ is Tikhonov regularizer which is used to guarantee a safe iteration step size. $\delta_\mathbf{x}$ is accepted only when the increment actually minimizes the error. Otherwise, μ is increased until the step size becomes so small that error is inevitably minimized. With acceptable step, μ is decreased and the next iteration will try a larger step. The algorithm listing is given in Algorithm 2.1. The maximum norm is defined $\|\mathbf{g}\|_\infty = \max(|g_1|, \ldots, |g_m|)$.

2.3.2 Gauss-Newton

Gauss-Newton minimization is a special case of LM, where $\mu = 0$. Thus, the step size is not adjusted, and each increment is estimated directly from

Algorithm 2.1 Levenberg-Marquardt minimization algorithm.

Input: A function $f : \mathbb{R}^m \Rightarrow \mathbb{R}^n$, a measurement $\mathbf{m} \in \mathbb{R}^n$ and initial parameters $\mathbf{x}_0 \in \mathbb{R}^m$.
Output: A vector $\mathbf{x}^+ \in \mathbb{R}^m$ minimizing $\|\mathbf{m} - f(\mathbf{x})\|^2$.

$\mathbf{x} = \mathbf{x}_0$, $\nu = 2$, $\varepsilon_1 = \varepsilon_2 = 10^{-15}$, $\tau = 10^{-3}$, $k_{\max} = 100$
$\mathbf{A} = \mathbf{J}^T\mathbf{J}$; $\mu = \tau * \max_{i=1\ldots m}(A_{ii})$
for all iterations $k \in [0, k_{\max}]$ **do**
  repeat
    $\mathbf{A} = \mathbf{J}^T\mathbf{J}$, $\mathbf{e}_\mathbf{x} = \mathbf{m} - f(\mathbf{x})$, $\mathbf{g} = \mathbf{J}^T\mathbf{e}_\mathbf{x}$
    Solve $(\mathbf{A} + \mu\mathbf{I})\delta_\mathbf{x} = \mathbf{g}$
    if $\|\delta_\mathbf{x}\| \leq \varepsilon_1\|\mathbf{x}\|$ **or** $\|\mathbf{g}\|_\infty \leq \varepsilon_2$ **then**
      return $\mathbf{x}$ # exit condition is met: increment or residual is sufficiently small
    else
      $\mathbf{x}_{\text{new}} = \mathbf{x} + \delta_\mathbf{x}$
      $\rho = (\|\mathbf{e}_\mathbf{x}\|^2 - \|\mathbf{m} - f(\mathbf{x}_{\text{new}})\|^2) / (\delta_\mathbf{x}^T(\mu\delta_\mathbf{x} + \mathbf{g}))$ # compute step condition variable
      if $(\ \rho \leq 0\ )$ **then**
        $\mu = \mu * \nu$; $\nu = 2 * \nu$ # decrease step size
      end if
    end if
  until $(\ \rho > 0\ )$
  $\mathbf{x} = \mathbf{x}_{\text{new}}$ # apply parameter increment
  $\mu = \mu * \max(\frac{1}{3}, 1 - (2\rho - 1)^3)$; $\nu = 2$ # increase step size
end for
return $\mathbf{x}$

the normal equation (eq. 2.23). The underlying assumption is that the initial guess is near the minimum of a smooth and convex cost function. In this case, step control can be avoided completely and the minimization becomes very efficient (Algorithm 2.2).

2.3.3 Random sample consensus

False 2D correspondences lead into unrobust estimation. Traditional sparse, feature-based pose estimation uses the Random Sample Consensus (RANSAC) for focusing on a few noiseless 3D points [55]. It works by finding subsets of points randomly until a consensus is found. The consensus set yields to the motion parameter hypothesis, which is supported by a sufficient number of inliers. Three 3D points or five 2D projection pairs are required to define 3D rotation and translation [20, 54]. The problem of this approach it's computational requirement which can be in the worst case $O(n^3)$ for n 3D points, if the last subset spans acceptable parameters. Also the inlier threshold is very problem-specific.

Algorithm 2.2 Gauss-Newton minimization algorithm.

Input: A function $f : \mathbb{R}^m \Rightarrow \mathbb{R}^n$, a measurement $\mathbf{m} \in \mathbb{R}^n$ and initial parameters $\mathbf{x}_0 \in \mathbb{R}^m$.
Output: A vector $\mathbf{x}^+ \in \mathbb{R}^m$ minimizing $\|\mathbf{m} - f(\mathbf{x})\|^2$.

$\mathbf{x} = \mathbf{x}_0$, $\varepsilon_1 = \varepsilon_2 = 10^{-15}$, $k_{\max} = 100$
for all iterations $k \in [0, k_{\max}]$ **do**
  $\mathbf{A} = \mathbf{J}^T\mathbf{J}$, $\mathbf{e}_\mathbf{x} = \mathbf{m} - f(\mathbf{x})$, $\mathbf{g} = \mathbf{J}^T\mathbf{e}_\mathbf{x}$
  Solve $\mathbf{A}\delta_\mathbf{x} = \mathbf{g}$
  if $\|\delta_\mathbf{x}\| \leq \varepsilon_1\|\mathbf{x}\|$ **or** $\|\mathbf{g}\|_\infty \leq \varepsilon_2$ **then**
    return $\mathbf{x}$ # exit condition is met: increment or residual is sufficiently small
  end if
  $\mathbf{x} \Leftarrow \mathbf{x} + \delta_\mathbf{x}$
end for
return $\mathbf{x}$

2.3.4 M-estimators

Due to the various noise sources in the estimation process, a mechanism to prevent noise to alter estimation process is very useful. Such measurements, which are not predictable by the generative model, are considered outliers or noise. The M-estimator can be used for managing outliers when the residual vector is of sufficient length for statistical purposes [27]. Essentially a M-estimator is a function for producing uncertainty based weights for residual elements. The main idea is to generate small weights for residual elements which are considered outliers. Spurious residual values are detected by analyzing the distribution of residual values. Typically in model fitting problems, the distribution of the residual has two components. The first component resembles a Gaussian distribution and represents the elements which are in accordance with the assumed model (e.g. values are close to zero). The second component consists of all elements which do not follow the model, and thus can be considered to be outliers. This rough division can be used to find such a threshold which optimally divides the distribution into inliers and outliers. Inliers always have small error whereas outliers may have any error value. Increased robustness is obtained by damping the high error values out from the estimation (Figure 2.5). Instead of using a fixed threshold, M-estimators provide a certain adaptation level to error profile variations. One method to determine the damping weights is by the Tukey weighting function

$$u_k = \frac{|e_k|}{c * median(\mathbf{e}_a)}, \tag{2.25}$$

$$\mathrm{w_k} = \begin{cases} (1 - (\frac{u_k}{b})^2)^2 & \text{if } |u_k| <= b \\ 0 & \text{if } |u_k| > b \end{cases}, \tag{2.26}$$

where $\mathbf{e}_a = \{|e_1|, |e_2|, \ldots, |e_n|\}$ is a set of absolute residual values, $c = 1.4826$ is the robust standard deviation, and $b = 4.6851$ is the Tukey specific constant. Thus, Tukey weighting generates adaptive weighting based on the statistical distribution of the residual. The weights w_k are produced per each residual element and they are roughly proportional to the inverse variances $1/\sigma_k^2$.

The weighted step is obtained by rewriting eq. 2.23 as follows

$$\mathbf{J}^T\mathbf{W}\mathbf{J}\mathbf{x} = -\mathbf{J}^T\mathbf{W}\mathbf{e}, \tag{2.27}$$

where $\mathbf{W}$ is a diagonal matrix with $\mathrm{diag}(\mathbf{W})_k = \mathrm{w_k}$. The robust step is obtained by $\mathbf{J} \Leftarrow \sqrt{\mathbf{W}}\mathbf{J}$ and $\mathbf{e} \Leftarrow \sqrt{\mathbf{W}}\mathbf{e}$ and therefore both LM and Gauss-Newton estimation can be trivially weighted. When the weights are quadratic, the square roots do not need to be evaluated in practice.

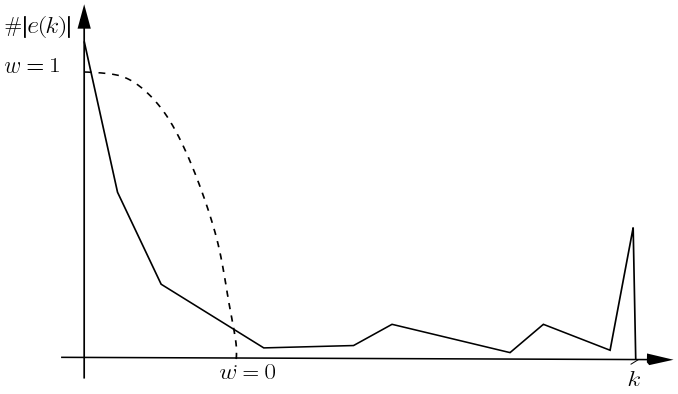


Figure 2.5: *Tukey M-estimator effect illustrated. The points with high error values are given small or zero weight, because their appearance change is not explained by the motion model.*

2.3.5 Iteratively re-weighted least squares

In Iterative Re-weighted Least-Squares (IRLS) [27] minimization, the weights are updated in each iteration using an element-wise weighting function $\rho : \mathbb{R}^m \Rightarrow \mathbb{R}^n$. ρ usually corresponds to an M-estimator, but also other weighting strategies can be used. IRLS algorithm is listed in Algorithm 2.3. IRLS minimization may require more iterations than Gauss-Newton, because often some inliers will also be given a small weight.

2.3.6 Linear system solvers

In this section, conjugate gradient method and Cholesky decomposition are described, which can both efficiently solve a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$,

Algorithm 2.3 IRLS

Input: A function $f : \mathbb{R}^m \Rightarrow \mathbb{R}^n$, a measurement $\mathbf{m} \in \mathbb{R}^n$, initial parameters $\mathbf{x}_0 \in \mathbb{R}^m$, and element-wise weighting function $w : \mathbb{R}^n \Rightarrow \mathbb{R}^n$.
Output: A vector $\mathbf{x}^+ \in \mathbb{R}^m$ minimizing $\|w^T(\mathbf{x})(\mathbf{m} - f(\mathbf{x}))\|^2$.

$\mathbf{x} = \mathbf{x}_0$, $\varepsilon_1 = \varepsilon_2 = 10^{-15}$, $k_{\max} = 100$
for all iterations $k \in [0, k_{\max}]$ **do**
  $\mathbf{w} = \rho(\mathbf{x})$, $\mathbf{W} = \text{zeros}(n, n)$, $W_{ii} = w_i$
  $\mathbf{J} \Leftarrow \sqrt{\mathbf{W}}\mathbf{J}$, $\mathbf{e} \Leftarrow \sqrt{\mathbf{W}}\mathbf{e}$
  $\mathbf{A} = \mathbf{J}^T\mathbf{J}$, $\mathbf{e}_\mathbf{x} = \mathbf{m} - f(\mathbf{x})$, $\mathbf{g} = \mathbf{J}^T\mathbf{e}_\mathbf{x}$
  Solve $\mathbf{A}\delta_\mathbf{x} = \mathbf{g}$
  if $\|\delta_\mathbf{x}\| \leq \varepsilon_1\|\mathbf{x}\|$ **or** $\|\mathbf{g}\|_\infty \leq \varepsilon_2$ **then**
    return $\mathbf{x}$ # exit condition is met: increment or residual is sufficiently small
  end if
  $\mathbf{x} \Leftarrow \mathbf{x} + \delta_\mathbf{x}$
end for
return $\mathbf{x}$

where $\mathbf{A}$ is a positive definite matrix. Linear system must be solved in every iteration of the estimation process. Various other methods exist, such as Singular Value Decomposition (SVD), but here we focus on the methods most fit for real-time estimation.

2.3.6.1 Conjugate gradient method

Krylov subspace methods form an important class of iterative methods. The Krylov subspace of order k generated by a $d \times d$ matrix $\mathbf{A}$ and a vector $\mathbf{b}$ of dimension d is the linear subspace.

$$\mathcal{B}(\mathbf{A}, \mathbf{b}) = \text{span}\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\}. \tag{2.28}$$

Krylov subspace algorithms construct an orthogonal basis of subspace $\mathcal{B}(\mathbf{A}, \mathbf{b})$ and use the basis vectors as movement directions, going along each one of them only once. Thus, at each iteration a method of this type eliminates one of the possible directions of the motion and after d iterations the solution is known. Thus, with exact arithmetic assumed, these methods can be regarded as direct methods. However, due to the roundoff errors in computations performed by computers, these methods can require more than d iterations to converge. Moreover, usually the dimension of the problem is very large and the algorithms are stopped well before d steps are done. An efficient method gives a good approximation in much less than d iterations. In Krylov methods the iterates $\mathbf{x}_k$ are constructed by minimizing an error-function.

The Conjugate Gradient (CG) algorithm is a very effective and popular method for the optimization of quadratic functions and the solution

of symmetric positive definite systems of equations. CG generates d conjugate search directions $\{\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots \boldsymbol{\delta}_{d-1}\}$, where conjugancy is defined by $\boldsymbol{\delta}_k^T \mathbf{A} \boldsymbol{\delta}_j = 0$, when $k \neq i$. The CG directions $\boldsymbol{\delta}_k$ form an orthogonal basis in the Krylov subspace $\mathcal{B}(\mathbf{A}, \mathbf{b})$. The iteration starts from $\mathbf{x}_0 = \mathbf{0}$. The initial residual is then $\mathbf{e}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b}$, and the first search direction is chosen to be $\boldsymbol{\delta}_1 = \mathbf{e}_0$. Now assuming the solution can be written as a linear combination

$$\mathbf{x} = \sum_{k=1}^{d} \gamma_k \boldsymbol{\delta}_k, \tag{2.29}$$

iteration update for parameters and residual will be

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \gamma_{k+1} \boldsymbol{\delta}_{k+1}, \quad \mathbf{e}_{k+1} = \mathbf{e}_k - \gamma_{k+1} \mathbf{A} \boldsymbol{\delta}_{k+1} \tag{2.30}$$

To solve step lengths γ_k, the final residual is first written as

$$\mathbf{e}_d = \mathbf{b} - \sum_{j=1}^{d} \gamma_j \mathbf{A} \boldsymbol{\delta}_j = \mathbf{0}. \tag{2.31}$$

The optimal step length for iteration k is obtained when multiplying both sides of the equation 2.31 by $\boldsymbol{\delta}_k^T$

$$\boldsymbol{\delta}_k^T \Big(\mathbf{b} - \sum_{j=1}^{d} \gamma_j \mathbf{A} \boldsymbol{\delta}_j \Big) = \mathbf{0} \Rightarrow \gamma_k = \frac{\boldsymbol{\delta}_k^T \mathbf{b}}{\boldsymbol{\delta}_k^T \mathbf{A} \boldsymbol{\delta}_k} = \frac{\boldsymbol{\delta}_k^T \mathbf{e}_{k-1}}{\boldsymbol{\delta}_k^T \mathbf{A} \boldsymbol{\delta}_k}. \tag{2.32}$$

All sum terms equal zero, except the k-th term due to conjugancy rule. The same rule is also used in the last $\mathbf{e}_{k-1} \Leftarrow \mathbf{b}$ substitution. Next an update is of interest, which can produce conjugate directions as a linear combination of $\mathbf{e}_k$ and $\boldsymbol{\delta}_k$.

Let

$$\boldsymbol{\delta}_{k+1} = \mathbf{e}_k + \beta_k \boldsymbol{\delta}_k, \tag{2.33}$$

where β_k must be

$$\boldsymbol{\delta}_{k+1}^T \mathbf{A} \boldsymbol{\delta}_k = (\mathbf{e}_k + \beta_k \boldsymbol{\delta}_k)^T \mathbf{A} \boldsymbol{\delta}_k = 0 \Rightarrow \beta_k = -\frac{\mathbf{e}_k^T \mathbf{A} \boldsymbol{\delta}_k}{\boldsymbol{\delta}_k^T \mathbf{A} \boldsymbol{\delta}_k} \tag{2.34}$$

Computational requirements of the terms γ_k and β_k can be reduced using the recursion formulas and conjugancy rule. The optimized conjugate gradient method is listed in Algorithm 2.4.

Algorithm 2.4 Optimized conjugate gradient method

Input: a $d \times d$ positive definite matrix $\mathbf{A}$, a $d \times 1$ vector $\mathbf{b}$.
Output: A $d \times 1$ vector $\mathbf{x}$ minimizing $\|\mathbf{Ax} - \mathbf{b}\|$.

$\mathbf{x}_0 = \mathbf{0}$, $\boldsymbol{\delta}_1 = \mathbf{b}$, $\mathbf{e}_0 = \mathbf{b}$
for all iterations $k \in [1, d]$ **do**
  $\gamma_k = \frac{\mathbf{e}_k^T \mathbf{e}_k}{\boldsymbol{\delta}_k^T \mathbf{A} \boldsymbol{\delta}_k}$
  $\mathbf{x}_k = \mathbf{x}_{k-1} + \gamma_k \boldsymbol{\delta}_k$, $\mathbf{e}_k = \mathbf{e}_{k-1} - \gamma_k \mathbf{A} \boldsymbol{\delta}_k$
  $\beta_k = \frac{\mathbf{e}_k^T \mathbf{e}_k}{\mathbf{e}_{k-1}^T \mathbf{e}_{k-1}}$
  $\boldsymbol{\delta}_{k+1} = \mathbf{e}_k + \beta_k \boldsymbol{\delta}_k$
end for
return $\mathbf{x}$

2.3.6.2 Cholesky factorization

A $n \times n$ real and positive definite matrix $\mathbf{A}$ can be factorized into $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is a lower triangular matrix. A linear system $\mathbf{Ax} = \mathbf{b}$, becomes easier to solve when substituting

$$\mathbf{L}(\mathbf{L}^T \mathbf{x}) = \mathbf{b} \Rightarrow \mathbf{Lz} = \mathbf{b}, \;\; \mathbf{L}^T \mathbf{x} = \mathbf{z}, \tag{2.35}$$

because inverting a system with a lower or upper triangular matrix requires merely back-substitution. In back-substitution, the rows can be solved in an incremental order.

Thus the only problem is how to factorize $\mathbf{A} = \mathbf{L}\mathbf{L}^T$. $\mathbf{A}$ can be represented in a block matrix form

$$\mathbf{A} = \mathbf{L}\mathbf{L}^L = \begin{pmatrix} l_{11} & \mathbf{0} \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix} \begin{pmatrix} l_{11} & \mathbf{L}_{21}^T \\ \mathbf{0} & \mathbf{L}_{22}^T \end{pmatrix} = \begin{pmatrix} l_{11}^2 & l_{11}\mathbf{L}_{21}^T \\ l_{11}\mathbf{L}_{21} & \mathbf{L}_{21}\mathbf{L}_{21}^T + \mathbf{L}_{22}\mathbf{L}_{22}^T \end{pmatrix}, \tag{2.36}$$

and thus $l_{11} = \sqrt{a_{11}}$ and $\mathbf{L}_{21} = \frac{1}{l_{11}} \mathbf{A}_{21}$ trivially, where a_{11} is the upper-left element in matrix $\mathbf{A}$ and $\mathbf{A}_{21}$ is the lower left sub-matrix.

$\mathbf{L}_{22}$ must be computed from

$$\mathbf{A}_{22} - \mathbf{L}_{21}\mathbf{L}_{21}^T = \mathbf{L}_{22}\mathbf{L}_{22}^T. \tag{2.37}$$

The problem is identical to original factorization problem but the matrix size is $(n-1) \times (n-1)$. Thus full factorization can be found recursively.

§ 2.4 3D POINT REFINEMENT USING MULTIPLE VIEWS

When a 3D point $\mathbf{P}$ has been triangulated using two views, but more 2D observations are available, it is possible to refine the point to be in consensus with all measurements. The cost function measures total re-projection

error by

$$e(\mathbf{P}) = \sum_{k=0}^{n} \rho\Big(\big\| \mathbf{K}_k D_k\big(N(\mathbf{R}_k\mathbf{P} + \mathbf{t}_k), \boldsymbol{\alpha}_k\big) - \mathbf{p}_k \big\|^2 \Big) = \mathbf{e}^T\mathbf{W}\mathbf{e}. \tag{2.38}$$

The residual elements are

$$e_k = \big\| \mathbf{K}_k D_k\big(N(\mathbf{R}_k\mathbf{P} + \mathbf{t}_k), \boldsymbol{\alpha}_k\big) - \mathbf{p}_k \big\|^2 = \mathbf{d}^T\mathbf{d}, \tag{2.39}$$

and the Jacobian elements are

$$J_{kj} = \frac{\partial e_k}{\partial P_j} = 2\mathbf{d}_k^T \frac{\partial \mathbf{d}_k}{\partial P_j} = 2\mathbf{d}_k^T \Pi \mathbf{K}_k \frac{\partial D_k(\mathbf{p}, \boldsymbol{\alpha})}{\partial \mathbf{p}} \frac{\partial N(\mathbf{P})}{\partial \mathbf{P}} \widehat{\mathbf{T}}_k \partial P_j. \tag{2.40}$$

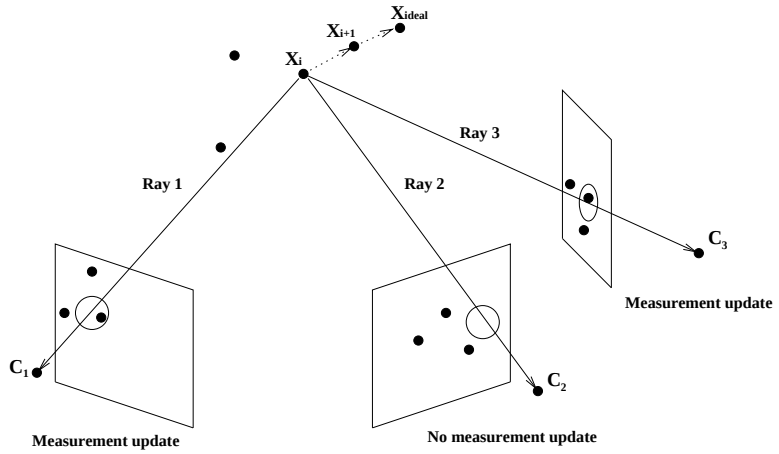


Figure 2.6: *3D point refinement using multiple views.* $\mathbf{X}_k$ *are 3D point estimates at iteration k. In every iteration, the measurements which are near the projection points are used in the update.*

The cost function is the same that is used in *bundle adjustment* [79], but here the viewing parameters remain fixed. 3D point estimation can be done by minimizing the re-projection error in all views concurrently but also sequentially by applying measurement updates to the Extended Kalman Filter (EKF), which is initialized at $\mathbf{P}$ (Figure 2.6). EKF will be discussed in more detail in Section 2.6.3.

§ 2.5 GEOMETRICAL POSE ESTIMATION METHODS

When points can be matched between the reference view and the current view, the relative pose can be estimated by minimizing a geometrical cost function. 2D or 3D Euclidean distances can be used as the error metric,

depending on the dimensionality of the points in both views. Cost formulations exist for 2D-to-2D, 3D-to-3D or 3D-to-2D cases, which are discussed in this section.

2.5.1 Generating small motion

It is often sufficient to parameterize only small motion, assuming that an initial guess exists. Especially when focusing on small angles, many sources of ambiguity can be avoided. According to *the Euler theorem*, each rotation sequence can be compactly expresses by rotation around a single axis. However, rotations can be expressed using various mathematical representations, such as quaternions, Euler angles, and axis-angle. Parameterized rotations are problematic for estimation because 1) they are non-linear mappings and 2) they are not unique. Besides natural ambiguities ($\alpha \sim \alpha \pm 2\pi$), some rotation parameterizations suffer from additional implicit ambiguities. For example, Euler angle parameterization allows parameterizing the same orientation in many ways. Quaternions contain an implicit vector normalization, which generates ambiguity for estimation, because axis-length can be arbitrary. Also other peculiarities exist such as *the gimbal lock*, where certain Euler rotation sequences lose degrees of freedom and numerical stability/accuracy can vary in the parameter space. Thus careful design is required when minimizing cost functions with rotation parameters.

Euler parameterization is given by

$$\mathbf{T}(\alpha, \beta, \gamma, t_x, t_y, t_z) = \begin{pmatrix} \mathbf{R}(\alpha, \beta, \gamma) & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \in \mathbb{SE}(3), \tag{2.41}$$

where $\mathbf{R}(\alpha, \beta, \gamma) = \mathbf{R}_\gamma \mathbf{R}_\beta \mathbf{R}_\alpha$, where α, β, and γ are the rotations around x-, y-, and z-axises, and $\mathbf{t} = (t_x, t_y, t_z)$ is the translation vector.

The explicit matrix components are

$$\mathbf{R}_\alpha = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}, \mathbf{R}_\beta = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, \tag{2.42}$$

$$\mathbf{R}_\gamma = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.43}$$

and the product becomes

$$\mathbf{R} = \begin{pmatrix} \cos\beta\cos\gamma & -\cos\alpha\sin\gamma + \sin\alpha\sin\beta\cos\gamma & \sin\alpha\sin\gamma + \cos\alpha\sin\beta\cos\gamma \\ \cos\beta\sin\gamma & \cos\alpha\cos\gamma + \sin\alpha\sin\beta\cos\gamma & -\sin\alpha\cos\gamma + \cos\alpha\sin\beta\cos\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{pmatrix} \tag{2.44}$$

A common linearization strategy with small angles is to replace the non-linear components by

$$\mathbf{R}_{\alpha}^{'} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -\alpha \\ 0 & \alpha & 1 \end{pmatrix}, \mathbf{R}_{\beta}^{'} = \begin{pmatrix} 1 & 0 & \beta \\ 0 & 1 & 0 \\ -\beta & 0 & 1 \end{pmatrix}, \mathbf{R}_{\gamma}^{'} = \begin{pmatrix} 1 & -\gamma & 0 \\ \gamma & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.45}$$

and the product becomes

$$\mathbf{R}^{'}(\alpha,\beta,\gamma) = \begin{pmatrix} 1 & \alpha\beta-\gamma & \beta+\alpha\gamma \\ \gamma & \alpha\beta\gamma+1 & \beta\gamma-\alpha \\ -\beta & \alpha & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix} \tag{2.46}$$

The final form follows from applying the small angle approximations $\sin\theta = \theta$, $\cos\theta = 1$, and neglecting the higher than linear terms ($\alpha\beta, \alpha\beta\gamma, \alpha\gamma, \beta\gamma \Leftarrow 0$). The linearized form can be used in small angle estimation, but it is not elegant, because the $\mathbf{R}^{'}(\alpha,\beta,\gamma) \notin \mathbf{SO}(3)$. This means that the column vectors in $\mathbf{R}^{'}(\alpha,\beta,\gamma)$ must be normalized to obtain a valid rotation matrix.

An alternative representation $\mathbf{T}(\boldsymbol{\omega},\boldsymbol{v}) \in \mathbb{SE}(3)$ forms a Lie group by exponential mapping [40]:

$$\mathbf{T}(\boldsymbol{\omega},\boldsymbol{v}) = e^{\mathbf{A}(\boldsymbol{\omega},\boldsymbol{v})} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}, \quad \mathbf{A}(\boldsymbol{\omega},\boldsymbol{v}) = \begin{bmatrix} [\boldsymbol{\omega}]_{\times} & \boldsymbol{v} \\ \mathbf{0} & 0 \end{bmatrix}, \tag{2.47}$$

where $(\boldsymbol{\omega},\boldsymbol{v}) \in \mathbb{R}^6$ encodes relative 3D rotation and translation between two camera poses. $\boldsymbol{\omega}$ is the rotation axis and $\|\boldsymbol{\omega}\|$ is the rotation angle in radians, and $\boldsymbol{v}$ is the velocity twist.

$$[\boldsymbol{\omega}]_{\times} = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} \tag{2.48}$$

produces a skew-symmetric matrix which performs a cross product.

The matrix exponential produces a 3×3 rotation matrix $\mathbf{R}$ and a 3×1 translation vector $\mathbf{t}$ embedded in a 4×4 matrix. This form suits estimation well, because linearization can be implemented without additional approximations. When estimating small motion using an iterative method, previous increments can be concatenated into a fixed base transform $\widehat{\mathbf{T}}$, because the transformations form a group $\mathbb{SE}(3)$. The next increment is then parameterized by $\widehat{\mathbf{T}}\mathbf{T}(\mathbf{x})$. Linearization of the iteration is only required at $\mathbf{x} = \mathbf{0}$, and the Jacobian becomes

$$\mathbf{J} = \widehat{\mathbf{T}}\frac{\partial \mathbf{T}(\mathbf{0})}{\partial \mathbf{x}} = \widehat{\mathbf{T}}\frac{\partial e^{\mathbf{A}(\mathbf{0})}}{\partial \mathbf{A}(\mathbf{0})}\frac{\partial \mathbf{A}(\mathbf{0})}{\partial \mathbf{x}} = \widehat{\mathbf{T}}\frac{\partial \mathbf{A}(\mathbf{0})}{\partial \mathbf{x}}. \tag{2.49}$$

Now $\frac{\partial \mathbf{A}}{\partial x_k}$ are constant matrices for each degree of freedom $k = 1 \ldots 6$ independently to parameterization $(\boldsymbol{\omega},\boldsymbol{v})$.

$$\frac{\partial \mathbf{A}}{\partial \omega_1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \frac{\partial \mathbf{A}}{\partial \omega_2} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \frac{\partial \mathbf{A}}{\partial \omega_3} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\frac{\partial \mathbf{A}}{\partial v_1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \frac{\partial \mathbf{A}}{\partial v_2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \frac{\partial \mathbf{A}}{\partial v_3} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

When adding a motion increment to the base transform $\widehat{\mathbf{T}}$, normalizations are not, *in theory*, required since $\widehat{\mathbf{T}}\mathbf{T}(\widehat{\boldsymbol{\omega}}, \widehat{\boldsymbol{v}}) \in \mathbb{SE}(3)$ applies without special normalization tricks[1].

A closed form solution exist for evaluating the matrix exponential

$$e^{\mathbf{A}(\boldsymbol{\omega}, \boldsymbol{v})} = \begin{pmatrix} \mathbf{R}(\boldsymbol{\omega}) & \frac{\left(\mathbf{I} - \mathbf{R}(\boldsymbol{\omega})\right)[\boldsymbol{\omega}]_\times \boldsymbol{v} + \boldsymbol{\omega}\boldsymbol{\omega}^T \boldsymbol{v}}{\|\boldsymbol{\omega}\|} \\ 0 & 1 \end{pmatrix}, \quad \text{if} \quad \boldsymbol{\omega} \neq \mathbf{0} \tag{2.50}$$

$$e^{\mathbf{A}(\boldsymbol{\omega}, \boldsymbol{v})} = \begin{pmatrix} \mathbf{I} & \boldsymbol{v} \\ \mathbf{0} & 1 \end{pmatrix}, \quad \text{if} \quad \boldsymbol{\omega} = \mathbf{0} \tag{2.51}$$

where

$$\mathbf{R}(\boldsymbol{\omega}) = \mathbf{I} + \frac{[\boldsymbol{\omega}]_\times}{\|\boldsymbol{\omega}\|} \sin(\|\boldsymbol{\omega}\|) + \frac{[\boldsymbol{\omega}]_\times^2}{\|\boldsymbol{\omega}\|^2} \left(1 - \cos(\|\boldsymbol{\omega}\|)\right) \tag{2.52}$$

is *the Rodriguez* formula.

Unfortunately, there is an important degenerate case at $\boldsymbol{\omega} = \mathbf{0}$, which is the important small angle case. This is why the closed form solution is often replaced by the numerical Padé approximant. Calculating the matrix exponential using Padé Approximation is defined by

$$e^{\mathbf{A}(\boldsymbol{\omega}, \boldsymbol{v})} \approx [D_{pq}(\mathbf{A})]^{-1} N_{pq}(\mathbf{A}), \tag{2.53}$$

where

$$N_{pq}(\mathbf{A}) = \sum_{j=0}^{p} \frac{(p+q-j)!p!}{(p+q)!j!(q-j)!} \mathbf{A}^j, \text{ and } D_{pq}(\mathbf{A}) = \sum_{j=0}^{p} \frac{(p+q-j)!q!}{(p+q)!j!(q-j)!} (-\mathbf{A}^j). \tag{2.54}$$

Notice the similarity with the Taylor series. If $q = 0$ then Eq. 2.53 would be a Taylor series. The Padé algorithm has some advantages over the Taylor series. In particular, Padé can obtain the same accuracy as Taylor in significantly less time. A drawback of Padé is that the algorithm performs poorly

[1] In practise, all real valued matrix operations on a computer introduce numerical errors which can be reduced by matrix normalization.

as $\|\mathbf{A}\|$ increases. From Higham [24] it was seen that the Padé approximation is more efficient when $p = q$. Like the Taylor series there is the question of where to terminate the series, and what are the appropriate values of p. Most articles say that 8 or 6 terms will give the best approximation, however Higham suggests that 13 terms gives the best approximation [24].

The main problem with the Padé approximation is that the accuracy decreases and computational requirements increases as $\|\mathbf{A}\|$ increases. To overcome this problem, the following identity

$$e^{\mathbf{A}} = (e^{\mathbf{A}/n})^n \tag{2.55}$$

is used. This technique is called *scaling-and-squaring* and it works by solving $e^{\mathbf{A}/n}$ using Padé and squaring the result to the power n. Typically, n is chosen to be the smallest power of two such that $\|\mathbf{A}/n\| < 1$.

2.5.2 Matrix normalization

By definition, a transformation matrix belongs to $\mathbb{SE}3$ algebraic group, where 3×3 sub-matrix must be $\mathbb{SO}3$ rotation matrix $\mathbf{R}$. 3D rotation matrices are *orthonormal*, which means that the column and row vectors have unit length and $\mathbf{u}_1 \times \mathbf{u}_2 = \mathbf{u}_3$, $\mathbf{u}_2 \times \mathbf{u}_3 = \mathbf{u}_1$, and $\mathbf{u}_3 \times \mathbf{u}_1 = \mathbf{u}_2$. When small motion is generated, small amount of numerical error will be introduced to 4×4 matrix $\mathbf{T}$. Numerical errors cumulate in time especially when extending a keyframe map. This is because additional real valued matrix operations are required to estimate the relative transformation to the nearest keyframe. When numerical bias cumulates in time, $\mathbf{T} \notin \mathbb{SE}3$, due to arbitrary scaling. Scaling increases exponentially and results in a non-recoverable tracking state.

To prevent numerical errors to cumulate, matrix normalization is required. Gram-schmidt procedure is well known method to guarantee an orthonormal base. The algorithm works by removing previously introduced directions from the remaining directions (see Algorithm 2.5). Despite that the algorithm looks simple, it contains more operations than is required. Because orthogonalization is convenient after every iteration, even small performance improvements are valuable. Algorithm 2.6 sketches computationally efficient orthogonalization. In effect, orthogonalization procedure removes scaling anomalies from $\mathbb{SO}3$ matrices.

2.5.3 3D-to-2D

When the baseline between two views can be assumed to be small, and a sufficient number of reliable feature points can be extracted from the images, it is possible to use a local minimization strategy for finding the relative pose between the views.

Algorithm 2.5 Gram-schmidt orthogonalization procedure

Input: 3×3 matrix $\mathbf{R}$.
Output: Orthonormal 3×3 matrix $\mathbf{R}'$.
1: Extract column vectors from $\mathbf{R} \Rightarrow (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$
2: **for all** iterations $k \in [1,3]$ **do**
3: $\quad \mathbf{u}_k \Leftarrow \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}$
4: $\quad$ **for all** iterations $j \in [k+1,3]$ **do**
5: $\quad\quad \mathbf{u}_j \Leftarrow \mathbf{u}_j - \text{proj}_{\mathbf{u}_k}(\mathbf{u}_j)$
6: $\quad$ **end for**
7: **end for**
8: Store column vectors back into $\mathbf{R}$

Algorithm 2.6 Optimized orthogonalization procedure

Input: 3×3 matrix $\mathbf{R}$.
Output: Orthonormal 3×3 matrix $\mathbf{R}'$.
1: Extract column vectors from $\mathbf{R} \Rightarrow (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$
2: $\mathbf{u}_3 \Leftarrow \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}$
3: $\mathbf{u}_2 \Leftarrow \mathbf{u}_3 \times \mathbf{u}_1$, $\mathbf{u}_2 \Leftarrow \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}$
4: $\mathbf{u}_1 \Leftarrow \mathbf{u}_2 \times \mathbf{u}_3$
5: Store column vectors back into $\mathbf{R}$

Let a set of 3D points $\mathbf{P}_k \in \mathcal{P}$ exists in a reference view with associated image descriptors $\mathbf{f}_k \in \mathbb{R}^n$, whose length is n. $\mathbf{p}_j \in \mathbb{R}^2$ are extracted from the current image and the associated image descriptors $\mathbf{f}_j^c$ are stored. Now m 2D-3D point pairs $(\mathbf{p}, \mathbf{P})_k$ are generated by finding the most similar $\mathbf{f}_j^c$ for each $\mathbf{f}_k$. Note that this may produce false matches which have to be eliminated by an outlier rejection mechanism (2.3.3,2.3.4).

The pose parameters $(\mathbf{R}, \mathbf{t})$ are estimated by minimizing the following cost function

$$e(\mathbf{R}, \mathbf{t}) = \sum_{k=1}^{m} \rho\Big(\|\mathbf{K}D\big(N(\mathbf{R}\mathbf{P}_k + \mathbf{t}), \boldsymbol{\alpha}\big) - \mathbf{p}_k\|^2 \Big) = \mathbf{e}^T\mathbf{W}\mathbf{e}, \tag{2.56}$$

where m is the amount of matching points, and $\rho(x) : \mathbb{R} \Rightarrow \mathbb{R}$ neglects statistical outliers by damping them to zero (Sec. 2.3.4).

The residual elements are

$$e_k = \|\mathbf{K}D\big(N(\mathbf{R}\mathbf{P}_k + \mathbf{t}), \boldsymbol{\alpha}\big) - \mathbf{p}_k\|^2 = \mathbf{d}_k^T\mathbf{d}_k, \tag{2.57}$$

and the Jacobian elements become

$$J_{kj} = \frac{\partial e_k}{\partial x_j} = 2\mathbf{d}_k^T \frac{\partial \mathbf{d}_k}{\partial x_j} = 2\mathbf{d}_k^T \Pi\mathbf{K} \frac{\partial D(\mathbf{p}, \boldsymbol{\alpha})}{\partial \mathbf{p}} \frac{\partial N(\mathbf{P})}{\partial \mathbf{P}} \widehat{\mathbf{T}} \frac{\partial \mathbf{A}(\mathbf{0})}{\partial x_j} \mathbf{P}_k, \tag{2.58}$$

With a decent initial guess and sufficient number of points, Gauss-Newton method produces accurate estimates (Sec. 2.3.2).

2.5.4 3D-to-3D

The traditional method for surface registration is the Iterative Closest Point algorithm (ICP) which alternates between finding temporary point correspondences and updating motion parameters until the system converges [38]. The algorithm estimates pose parameters by minimizing a point-to-plane distance in 3D. The computational efficiency depends on the amount of points to be matched. A naive ICP implementation which does not utilize parallel computation does not scale well when the number of points increases. ICP can be made more noise tolerant when combining it with M-estimation or RANSAC. ICP accuracy can improve when it is initialized by matching a set of 3D points by an effective 2D descriptor such as SIFT [39]. There are various way of finding temporary point correspondences during ICP iteration [59, 61] A common performance improvement is to use kd-tree for nearest neighbor searches, when aiming at real-time applications. A more efficient matching, however, uses projective association, in a case where the baseline is small [53].

The reference point cloud $\mathcal{P}^*$ must be associated with normals $\mathcal{N}^* = (\mathbf{n_1^k}, \ldots, \mathbf{n}_m^k)$. The point normals allow measuring a point set distance only in normal direction. This way tangential displacements are free in the cost function and surface alignment becomes more efficient. This cost metric is called *point-to-plane distance*, despite that plane is not defined outside a single point. Assuming 3D points are reconstructed from a depth map, the normals are trivially estimated by

$$\mathbf{n}(u,v) = \Big(\mathbf{P}(u+1,v) - \mathbf{P}(u,v)\Big) \times \Big(\mathbf{P}(u,v+1) - \mathbf{P}(u,v)\Big). \tag{2.59}$$

Unit normals are generated by $\mathbf{n}(u,v) \Leftarrow \mathbf{n}(u,v)/\|\mathbf{n}(u,v)\|$.

The pose parameters $(\mathbf{R}, \mathbf{t})$ are estimated by minimizing the following cost function

$$e(\mathbf{R},\mathbf{t}) = \sum_{k=1}^{m} \rho\Big(\big(\mathbf{R}\mathbf{P}_k + \mathbf{t} - \mathbf{Q}_k\big)\cdot \mathbf{n}_k\big)^2\Big) = \mathbf{e}^T\mathbf{W}\mathbf{e}, \tag{2.60}$$

where m is the amount of matching points, and $\rho(x) : \mathbb{R} \Rightarrow \mathbb{R}$ neglects statistically large displacements by damping them to zero, and $\mathbf{n}_k = (n_x, n_y, n_z)^T$ are the point normals.

The point associations $\mathbf{P}_k \longleftrightarrow \mathbf{Q}_k$ are generated automatically for each iteration of the minimization. The nearest $\mathbf{Q}_k$ are updated using Euclidean or projective distance. In projective association, the source points are projected onto the destination mesh from the point of view of the destination mesh's range camera. Kd-trees have been used to speed-up nearest neighbor searches to $O(logN)$ time, but projective association can be done even more efficiently in constant time [61].

Point-to-plane distance does not vary when sliding along the reference surface. This means that surface registration can often have larger convergence domain than a one using point-to-point based Euclidean distance. However, in mostly planar scenes, the motion will not be fully constrained by the cost function. To reduce the problem, the points $\mathbf{P}_k$ can be selected in such a way that their normals $\mathbf{n}_k$ have as uniform distribution as possible to all directions [61].

To minimize (2.60), it is useful to use the small angle approximation (eq. 2.46), and re-organize the residual elements into

$$e_k = (\mathbf{P}_k - \mathbf{Q}_k) \cdot \mathbf{n}_k + \mathbf{r} \cdot \mathbf{c}_k + \mathbf{t} \cdot \mathbf{n}_k, \tag{2.61}$$

where $\mathbf{c}_k = \mathbf{P}_k \times \mathbf{n}_k$, and $\mathbf{r} = (\alpha, \beta, \gamma)$.

The Jacobian will be simply

$$\mathbf{J} = \begin{pmatrix} c_0(0) & c_0(1) & c_0(2) & n_0(0) & n_0(1) & n_0(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_m(0) & c_m(1) & c_m(2) & n_m(0) & n_m(1) & n_m(2) \end{pmatrix}. \tag{2.62}$$

2.5.5 Statistical matching

Besides using raw measurements for pose estimation, various other approaches exist which rely on matching abstract primitives derived from the raw measurements. For example, statistical matching of point clouds can be justified in case raw measurements contain noise. It is possible to construct a mixture of Gaussians model from a raw point cloud and use it directly to align a relative pose [72]. Assuming Gaussians $\mathcal{N}_k^*(\mu^*, \mathbf{\Sigma}^*)$ in the reference coordinate system are associated with $\mathcal{N}_k(\mu, \mathbf{\Sigma})$ in the current coordinate system, the following likelihood function can be formulated

$$p(\mathbf{x}) = \prod_k \frac{1}{(2\pi)^3 \|\mathbf{\Sigma}_k\|^{\frac{1}{2}}} \exp\Big(- d_k^T(\mathbf{x}) \mathbf{\Sigma}_k^{-1}(\mathbf{x}) d_k(\mathbf{x}) \Big), \tag{2.63}$$

where

$$d_k(\mathbf{x}) = \mu_k - \mathbf{T}(\mathbf{x}) \mu_k^* \tag{2.64}$$

$$\mathbf{\Sigma}_k(\mathbf{x}) = \mathbf{\Sigma}_k + \mathbf{R}(\mathbf{x}) \mathbf{\Sigma}_k^* \mathbf{R}(\mathbf{x})^T, \tag{2.65}$$

where $\mathbf{R}(\mathbf{x})$ is 3×3 rotation matrix embedded in 4×4 rigid transformation matrix $\mathbf{T}(\mathbf{x})$. The likelihood measures similarity between paired Gaussians. d_k are the 3D differences between the means in the current coordinates system, and $\mathbf{\Sigma}_k$ are covariances which are summed together from the current and projected reference covariances.

Instead of maximizing likelihood, its possible to minimize the negative logarithm

$$e(\mathbf{x}) = -\sum_k -3\ln(2\pi) - \frac{1}{2}\ln|\mathbf{\Sigma}_k| - d_k^T(\mathbf{x})\mathbf{\Sigma}_k^{-1}(\mathbf{x})d_k(\mathbf{x}) \quad (2.66)$$

$$\sim \sum_k \frac{1}{2}\ln|\mathbf{\Sigma}_k(\mathbf{x})| + d_k^T(\mathbf{x})\mathbf{\Sigma}_k^{-1}(\mathbf{x})d_k(\mathbf{x}), \quad (2.67)$$

where $|\mathbf{\Sigma}|$ is the determinant of the covariance.

The benefit of this approach is that it is more invariant to measurement noise, but the challenge is to construct and temporally match a mixture of Gaussians which does not regularize the original data too much.

§ 2.6 BUNDLE ADJUSTMENT FOR SIMULTANEOUS ESTIMATION

In *bundle adjustment*, multiple camera poses and a 3D point set are concurrently optimized [8, 37, 79]. Bundle adjustment provides the best accuracy possible and should be used whenever a decent initial guess and measurement association can be provided. The name refers to the *bundles* of light rays originating from each 3D point and converging on each camera centre, which are adjusted optimally with respect to both structure and viewing parameters. The problem is formulated by a cost function which measures 2D re-projection error of the current pose and the structure configuration. The minimization requires an initial guess, which is iteratively improved until the cost decreases below a threshold. The relative pose between two views are often inferred from *the essential matrix*, which can be estimated from 2D point correspondences. The structure can be initialized using direct depth measurements or by triangulation from 2D projection points. The pose estimation process suffers from some drawbacks. Feature point extraction and matching is an error-prone process and requires an outlier rejection mechanism such as RANSAC. Extraction of high quality features such as SIFT can be expensive for real-time systems and still mismatches can not be fully prevented. For example, repetitive patterns and homogeneous textureless regions easily produce mismatches.

The minimization often relies on Levenberg-Marquardt, because the mapping from the parameters into 2D observations is non-linear. The cost function is

$$e(\mathbf{a},\mathbf{b}) = \sum_{i=1}^{m}\sum_{j=1}^{n}\rho\Big(\big\|\mathbf{K}_j D_j\big(N(\mathbf{R}_j\mathbf{P}_i + \mathbf{t}_j), \boldsymbol{\alpha}_j\big) - \mathbf{p}_{ij}\big\|^2\Big) = \mathbf{e}^T\mathbf{W}\mathbf{e}, \quad (2.68)$$

where $\mathbf{a} = (\mathbf{R}_1,\ldots,\mathbf{R}_n,\mathbf{t}_1,\ldots,\mathbf{t}_n,\mathbf{K}_1,\ldots,\mathbf{K}_n,\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_n)^T$ contain the extrinsic, and intrinsic parameters of n cameras, $\mathbf{b} = (\mathbf{P}_1,\ldots,\mathbf{P}_m)^T$ is a vector of

m 3D points which are associated with the measured projections $\mathbf{p}_{ij}$, and ρ is the robust weighting function.

Concurrent minimization of all free parameters is the key to coherent 3D reconstructions and SLAM. Despite being theoretically sound method, many practical problems arise when applying it to computer vision problems. One major problem is 2D feature extraction and matching between 2D views. General unrestricted images simply do not contain enough information to guarantee correct matches. As an example, repetitive patterns and textureless images are impossible to match automatically without additional information. Another problem is the large number of free parameters involved. n camera view introduces $6n$ extrinsic parameters, $3n$ intrinsic parameters (square pixels without skew), $5n$ distortion parameters, and m points introduces $3m$ structure parameters. Often the intrinsic parameters and lens distortions can be estimated for each camera view prior to bundle adjustment which reduces the number of free parameters from $14n + 3m$ to $6n + 3m$. The cost function minimization requires careful design to enable as many views and points as possible. Traditional `sba` library [37] and recently introduced g2o library [36] implement various techniques to reduce computational requirements. These libraries can provide convergent estimates automatically, when close enough initial guess has been provided. Initialization is finally the most critical phase, which can be extremely expensive using exhaustive trial-and-error approach. Thus manual or semi-automatic tools are considerable option to obtain correct initialization in a limited time frame.

2.6.1 Sparse structure

Taking into account sparse structure of bundle adjustment problem, a significant performance optimization is possible [37]. Let $\mathbf{A}_{ij}$ be a 1×6 matrix which encodes linear relationship from camera parameters j to the projection distance of $\mathbf{P}_i$ in view j (eq. 2.58). Also let $\mathbf{B}_{ij}$ be 1×3 matrix which encodes linear relationship from point $\mathbf{P}_i$ to projection distance in view j (eq. 2.40).

The full Jacobian for $n = 3$ views and $m = 4$ points becomes

$$\mathbf{J}_{kj} = \frac{\partial e_k}{\partial x_j} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{12} & \mathbf{0} & \mathbf{B}_{12} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{13} & \mathbf{B}_{13} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{21} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{21} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{22} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{23} & \mathbf{0} & \mathbf{B}_{23} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{31} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{31} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{32} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{32} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{33} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{33} & \mathbf{0} \\ \mathbf{A}_{41} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{41} \\ \mathbf{0} & \mathbf{A}_{42} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{42} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{43} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{43} \end{pmatrix}. \tag{2.69}$$

Sparse Jacobian implies sparse normal equations

$$\mathbf{J}^T\mathbf{W}\mathbf{J} = \begin{pmatrix} \mathbf{U}_1 & \mathbf{0} & \mathbf{0} & \mathbf{Q}_{11} & \mathbf{Q}_{21} & \mathbf{Q}_{31} & \mathbf{Q}_{41} \\ \mathbf{0} & \mathbf{U}_2 & \mathbf{0} & \mathbf{Q}_{12} & \mathbf{Q}_{22} & \mathbf{Q}_{32} & \mathbf{Q}_{42} \\ \mathbf{0} & \mathbf{0} & \mathbf{U}_3 & \mathbf{Q}_{13} & \mathbf{Q}_{23} & \mathbf{Q}_{33} & \mathbf{Q}_{43} \\ \mathbf{Q}_{11}^T & \mathbf{Q}_{12}^T & \mathbf{Q}_{13}^T & \mathbf{V}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{Q}_{21}^T & \mathbf{Q}_{22}^T & \mathbf{Q}_{23}^T & \mathbf{0} & \mathbf{V}_2 & \mathbf{0} & \mathbf{0} \\ \mathbf{Q}_{31}^T & \mathbf{Q}_{32}^T & \mathbf{Q}_{33}^T & \mathbf{0} & \mathbf{0} & \mathbf{V}_3 & \mathbf{0} \\ \mathbf{Q}_{41}^T & \mathbf{Q}_{42}^T & \mathbf{Q}_{43}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{V}_4 \end{pmatrix} = \begin{pmatrix} \mathbf{U} & \mathbf{Q} \\ \mathbf{Q}^T & \mathbf{V} \end{pmatrix}, \tag{2.70}$$

where $\mathbf{U}_k = \sum_{i=1}^{4} \mathbf{A}_{ij}^T W_{ij} \mathbf{A}_{ij}$, $\mathbf{V}_k = \sum_{j=1}^{3} \mathbf{B}_{ij}^T W_{ij} \mathbf{B}_{ij}$, $\mathbf{Q}_{ij} = \mathbf{A}_{ij}^T W_{ij} \mathbf{B}_{ij}$, and W_{ij} is the weight for measurement $\mathbf{p}_i$ in view j.

Finally, the normal equation becomes

$$\begin{pmatrix} \mathbf{U} & \mathbf{Q} \\ \mathbf{Q}^T & \mathbf{V} \end{pmatrix} \begin{pmatrix} \delta_{\mathbf{a}} \\ \delta_{\mathbf{b}} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_a \\ \mathbf{r}_b \end{pmatrix}, \tag{2.71}$$

where the camera parameters are denoted $\delta_{\mathbf{a}}$, the structural parameters are $\delta_{\mathbf{b}}$, and $\mathbf{r} = (\mathbf{r}_a, \mathbf{r}_b)^T = \mathbf{J}^T\mathbf{e}$.

2.6.2 Marginalization

A key technique for efficient bundle adjustment is marginalization of the cost function. In practise, the error function 2.68 is more sensitive to camera parameters than point parameters. This information can be used by

solving the system in two phases: first $\delta_{\mathbf{a}}$, and then $\delta_{\mathbf{b}}$. Left multiplication of eq. 2.71 by the block matrix

$$\begin{pmatrix} \mathbf{I} & -\mathbf{Q}\mathbf{V}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{2.72}$$

results in

$$\begin{pmatrix} \mathbf{U} - \mathbf{Q}\mathbf{V}^{-1}\mathbf{Q}^T & \mathbf{0} \\ \mathbf{Q}^T & \mathbf{V} \end{pmatrix} \begin{pmatrix} \delta_{\mathbf{a}} \\ \delta_{\mathbf{b}} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_a - \mathbf{Q}\mathbf{V}^{-1}\mathbf{r}_b \\ \mathbf{r}_b \end{pmatrix}, \tag{2.73}$$

Now estimation of $\delta_{\mathbf{a}}$ is possible without $\delta_{\mathbf{b}}$ by

$$(\mathbf{U} - \mathbf{Q}\mathbf{V}^{-1}\mathbf{Q}^T)\delta_{\mathbf{a}} = \mathbf{r}_a - \mathbf{Q}\mathbf{V}^{-1}\mathbf{r}_b \tag{2.74}$$

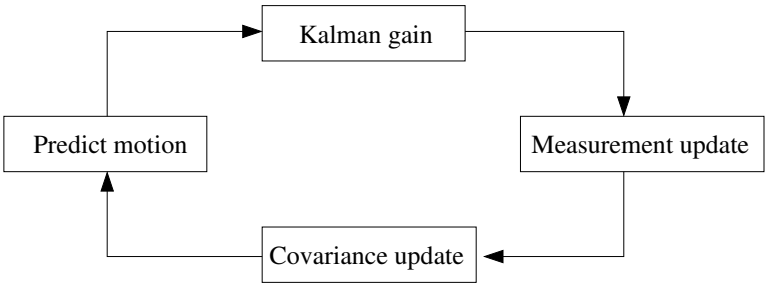
followed by the estimation of $\delta_{\mathbf{b}}$

$$\mathbf{V}\delta_{\mathbf{b}} = \mathbf{r}_b - \mathbf{Q}^T\delta_{\mathbf{a}}. \tag{2.75}$$

These equations are also known as *Schur complements*.

2.6.3 Extended Kalman Filter

The bundle adjustment problem can also be formulated as a stochastic process using *the Extended Kalman Filter* [87]. The non-linear cost function is first linearized and Jacobian $\mathbf{J}$ is obtained which maps a parameter increment into linear displacement in the measurement space. The EKF estimates parameters by an alternating sequence of linear predictions and corrections. For every discrete time step, a motion model is used to predict the current parameters $\mathbf{x}_k^{'}$, which are then refined into $\mathbf{x}_k$ using the current measurement $\mathbf{z}_k$. $\mathbf{A}$ is the matrix which linearizes continuous motion model, but it can also be set to identity which means that initial guess is directly the previous estimate. The main difference to bundle adjustment is that the EKF is an incremental method which aims at estimating only the most recent state (current camera pose along with the 3D points). The problem could be solved by marginalizing the previous pose parameters out from the full linear system (Section 2.6.2). Because full marginalization is not a computationally scalable process, EKF simplifies the estimation by forward propagating a Gaussian parameter distribution in time ($\mathbf{x}_{k+1} \Leftarrow \mathbf{x}_k$, $\mathbf{P}_{k+1} \Leftarrow \mathbf{P}_k$). The propagated Gaussian can correspond to the marginal distribution after *a system identification* phase, where the noise terms $(\mathbf{Q}, \mathbf{R})$ are tuned based on the application. The EKF essentially *filters* parameters from noisy measurements, where the noise must comply with Gaussian assumptions. The EKF does not recover if *the state* (parameter estimates and covariance) is corrupted due to outlier measurements. In practise, linearization and the process inaccuracies can result into non-positive definite covariance $\mathbf{P}$. One method for avoiding the problem is regularizing $\mathbf{P} \Leftarrow \mathbf{P}^T\mathbf{P}$ when necessary.



Phase	Formula
Kalman gain	$\mathbf{K}_k = \mathbf{P}_k^{'}\mathbf{J}^T(\mathbf{J}\mathbf{P}_k^{'}\mathbf{J}^T + \mathbf{R})^{-1}$
Measurement update	$\mathbf{x}_k = \mathbf{x}_k^{'} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{J}\mathbf{x}_k^{'})$
Covariance update	$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{J})\mathbf{P}_k^{'}$
Predict motion	$\mathbf{x}_{k+1}^{'} = \mathbf{A}\mathbf{x}_k, \;\; \mathbf{P}_{k+1}^{'} = \mathbf{A}\mathbf{P}_k\mathbf{A}^T + \mathbf{Q}$

Figure 2.7: *Kalman filtering process.*

§ 2.7 FEATURE-BASED SIMULTANEOUS LOCALIZATION AND MAPPING

When explicit an 3D model is not specified, camera pose tracking essentially becomes Simultaneous Localization And Mapping (SLAM) where both structure and motion are estimated concurrently. Several sparse, feature-based systems have been presented, which extract feature points from images, match them temporally and after tracking them during multiple frames, can generate a sparse set of 3D points. The sparse point cloud is used as a model with which 2D point locations can be predicted in the new images. Camera pose and 3D points can then be simultaneously estimated by minimizing the prediction error between the model points and 2D point measurements. As tool bundle adjustment and EKF have been proven useful. Davison's MonoSLAM is the first real-time system to execute visual SLAM [15]. It is based on Extended Kalman Filter. After MonoSLAM, various other systems have been built, such as Royer's system [60], FrameSLAM [34] and PTAM [33], which do not use EKF but are based on bundle adjustment.

2.7.1 Feature points as measurements

Features can be extracted by various feature detectors such as SIFT [39] and SURF [4]. Features have 2D image coordinates and a descriptor which describes the local appearance of a point. The descriptors are often scale-

and rotation invariant, because the features have to be matched in different views. Feature points simplify image data and allow defining smooth distance based cost functions. The downside, however, is that compression loses information and feature point matching often fails which may cost in the robustness and accuracy of the estimation. Although a relative camera pose can be estimated from a sufficient amount of 2D feature matches [26, 37], the problem can be simplified by introducing depth measurements. A RGB-D sensor such as a stereo camera or the Microsoft Kinect sensor can provide depth measurements densely over the full image. There are also methods for estimating depths by using a sequence of monocular images [15]. The depth values can be used to measure camera pose error in terms of 2D re-projection error or 3D point distance. In controlled environments, feature points can be efficiently extracted by using markers. Markers may even carry identification information such as color or a blinking pattern. The downside of markers is the manual effort in setting up the system.

2.7.2 Loop-closure

Typically the estimation process suffers from time evolving drift, which is often corrected by a loop closure mechanism. In a loop closure, the current viewpoint is identified using a previously stored map. The cumulated pose error can then be divided evenly along the loop for removing the drift. Loop closure is illustrated in Figure 2.8. Loop closure mechanisms are problematic in real-time AR applications unless they remove drift nearly in every frame, because bigger corrections are often visually disturbing. No loop closure can also be visually disturbing due to drift and false geometry. Royer divides the SLAM problem into separate learning phase and online phase which utilizes the pre-recorded visual learning path to guide a robot [60].

2.7.3 PTAM

PTAM is a monocular system which estimates camera pose in an unknown scene [33]. It is specifically designed to track a hand-held camera in a small AR workspace. In PTAM tracking and mapping are split into two separate tasks, processed in parallel threads on a dual-core computer: one thread deals with the task of robustly tracking erratic hand-held motion, while the other produces a 3D map of point features from previously observed video frames. This allows the use of computationally expensive bundle adjustment (sec.2.6) which is not usually associated with real-time operation: The result is a system that produces detailed maps with hundreds of landmarks which can be tracked at 30Hz frame-rate, at best with an accuracy and robustness competing with state-of-the-art model-based systems.

PTAM system is able to do loop closure via bundle adjustment for each frame, but suffers from error-prone feature extraction and matching process. It is noteable that the sparse 3D points must be initialized by a special procedure which imitates a stereo camera using a horizonally moved monocular camera. This has proven to be more practical than executing EKF with initially unknown depth parameters.

2.7.4 FrameSLAM

FrameSLAM initializes 3D points using stereo triangulation method and uses *bundle adjustment* (sec.2.6) for estimating camera poses and 3D points in multiple frames simultaneously [34]. The pose configuration is estimated within a sliding window of n recent frames, because the same scene geometry may be visible only for short time when the camera is moving. Despite that bundle adjustment is used to estimation, only a relative frame pose information (a skeleton) is stored. The skeleton is a reduced nonlinear system that is a faithful approximation of the larger system, and is used to solve large loop closures quickly, as well as forming a backbone for data association and local registration. Konolige illustrates the working of the system with large outdoor datasets (10 km), showing largescale loop closure and precise localization in real-time.

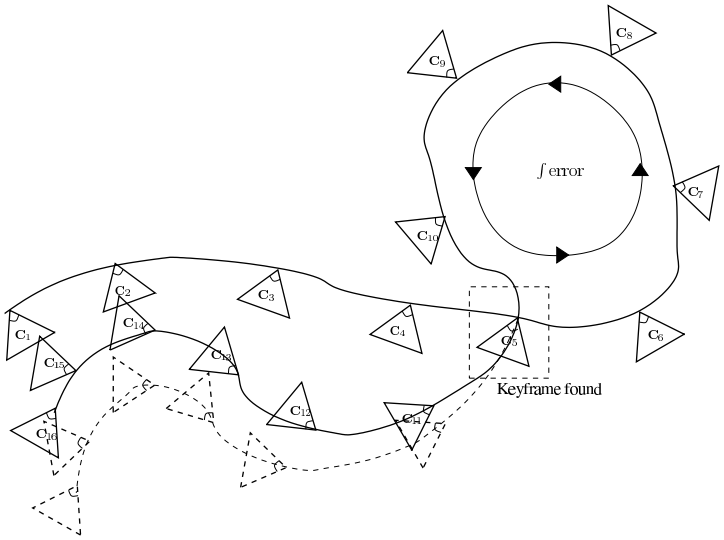


Figure 2.8: *Loop closure when re-entering same viewpoints. Loop-closure detection effectively removes drift cumulated during the loop. Dashed trajectory is contaminated by the errors cumulated during the loop.*

§ 2.8 DENSE TRACKING AND MAPPING

When the Kinect sensor was released, several emerging systems have been presented for camera tracking and environment mapping, which take advantage of dense and accurate depth maps. Henry *et al.* describe an approach for indoor mapping, where traditional feature-point based RANSAC pose estimate is refined using ICP and the final 3D maps are bundle adjusted [22]. A recent KinectFusion system does effort to avoid bundle adjustment completely [53]. It integrates dense depth maps into a voxel grid and rasterizes smooth reference depth maps using a ray caster, which are used with ICP method for camera tracking. Due to KinectFusion's memory-scalability problem, Kintinuous system has been introduced for mapping larger indoor environments [89]. It simply uses moving reconstruction volume and replaces the past geometry with a triangulated surface. The problem is that triangulated surfaces can not be easily used for loop-closure because triangulated surfaces have degraded texturing and geometry.

2.8.1 Microsoft Kinect

A RGB-D sensor such as the Microsoft Kinect sensor is well-suited for camera pose estimation because it produces a real-time feed of RGB-D measurements. The sensor consists of IR projector, IR sensor and a RGB camera. Internal hardware is used to generate a disparity map by matching IR light pattern and projected IR image. The RGB images are in 640×480 resolution at $30Hz$, but due to Bayer filtering they are redundant. The depth maps are also stored in 640×480 resolution. The sensor depth range is $\approx 1 - 5m$, and the accuracy decreases in distance [12].

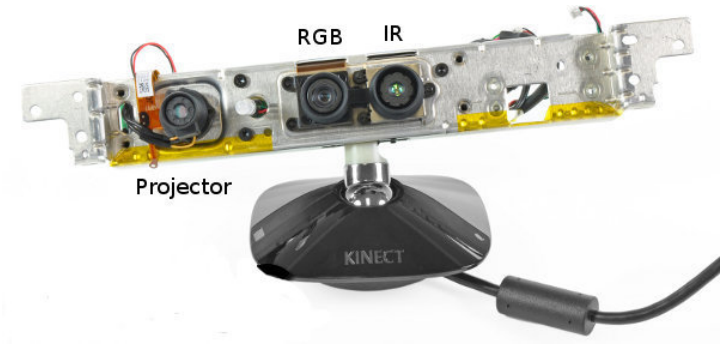


Figure 2.9: *Microsoft Kinect the plastic cover removed. Image is courtesy of* `www.ros.org`.

2.8.1.1 Extended Caltech calibration

Since the Microsoft Kinect sensor can be set into a special mode where the raw IR images can be stored, it is possible to use standard stereo calibration procedure for obtaining the calibration parameters for the IR and the RGB view [6] (Fig. 2.10). IR view and the depth view are trivially associated with image offset $(-4, -3)$.

A single camera calibration is used to initialize IR and RGB camera parameters. Then calibration is followed by a stereo procedure which re-estimates the Caltech parameters $\mathbf{K}_{IR}$, $\mathbf{K}_{RGB}$, $\mathbf{kc}_{RGB}$, and $\mathbf{T}_b$. The IR lens distortion parameters ($\mathbf{kc}_{IR}$) are forced to zero, because data has already been used to generate the raw disparity map. This means that the IR lens distortion is compensated by tweaking the enumerated Caltech parameters. The RGB camera lens distortion parameters $\mathbf{kc}_{RGB}$ are estimated without any special concerns. In practice, the distortion seems to be minor and the first two radial coefficients are sufficient. In our calibration $\mathbf{kc}_{RGB} \approx (0.2370, -0.4508, 0, 0, 0)$, and the stereo baseline is $b \approx 25mm$. $\mathbf{T}_b$ stores the baseline transform as 4×4 matrix. The conversion from raw disparities into depth values can be done by $z = \frac{8pf}{B-d}$, where p is the baseline between the projector and the IR camera, B is a device specific constant and f is IR camera focal length in pixel units. p and B are estimated by solving the linear equation $\begin{bmatrix} -\mathbf{1} & \mathbf{Z} \end{bmatrix} \begin{bmatrix} A & B \end{bmatrix}^T = \mathbf{D}$, where $A = 8pf$, $\mathbf{Z}$ is $n \times 1$ matrix of reference depth values z_k from the chessboard pattern, and $\mathbf{D}$ is a $n \times 1$ matrix whose elements are $d_k z_k$. Typical values are $p \approx 75mm$ and $B \approx 1090$. Note, that this reconstruction method is merely an approximation which precludes measurements at long ranges [12].



Figure 2.10: *RGB and IR images of the calibration pattern.*

2.8.1.2 Oulu calibration

The Microsoft Kinect sensor is conveniently calibrated using a specialized toolbox by Herrera *et al.*, which jointly estimates all calibration parameters [23]. A chessboard pattern is printed and a set of matching raw dis-

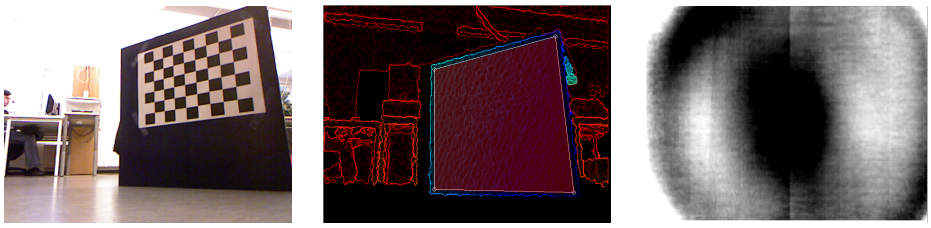
parity images and RGB images are captured and loaded into the toolbox. The toolbox then semi-automatically provides the intrinsic matrices $\mathbf{K}_{IR}$, $\mathbf{K}_{RGB}$, the baseline transform $\mathbf{T}_b$ between the views, reconstruction parameters $(c_0, c_1) \in \mathbb{R}^2$ and distortion coefficients $\boldsymbol{\alpha}_{RGB}$, α_0, α_1 and $\boldsymbol{\beta}$. The raw disparity map $\mathcal{D}(u, v)$ is then undistorted by

$$u(d, \alpha_0, \alpha_1, \boldsymbol{\beta}) = d + \boldsymbol{\beta}(u, v) \exp(\alpha_0 - \alpha_1 d), \tag{2.76}$$

and converted into depth values by

$$z(d) = \frac{1}{c_0 + c_1 u(d)}. \tag{2.77}$$

The disparity undistortion model corrects bias which increases with distance [12], and also models per pixel distortions using a map $\boldsymbol{\beta}(u, v)$. The RGB image lens distortion is modeled using the standard Caltech parameters $\boldsymbol{\alpha}_{RGB} \in \mathbb{R}^5$, which models radial distortions using three coefficients and tangential distortions using two coefficients [6]. Typically only the first two radial components are necessary, and the rest can be fixed to zero to speed up computation. The toolbox allows fixing distortion parameters prior to calibration.



(a) (b) (c)

Figure 2.11: *a) RGB b) planar board in disparity image c) disparity distortion pattern $\boldsymbol{\beta}$ [23]*

2.8.1.3 Depth noise removal by bilateral filtering

Unfortunately, the captured disparity maps can be contaminated by noise. An amount of noise can be reduced by filtering the maps prior to use. *Bilateral filter* smooths surfaces without mixing data from different depth layers [77]. A bilateral filter is defined by

$$\mathcal{D}_b(\mathbf{p}) = \frac{1}{n} \sum_{\mathbf{q} \in \Omega} \mathcal{D}(\mathbf{q}) * f(\|\mathbf{q} - \mathbf{p}\|, \sigma_f) * g(\|\mathcal{D}(\mathbf{q}) - \mathcal{D}(\mathbf{p})\|, \sigma_g), \tag{2.78}$$

where $\mathcal{D}(\mathbf{p})$ is the disparity value at 2d point $\mathbf{p}$, Ω is a spatial neighborhood around $\mathbf{p}$, $\mathcal{D}_b(\mathbf{p})$ is the filtered disparity value, and n is the amount

of pixels in neighborhood Ω. The weights are calculated as a product of two kernels f and g. f is the spatial damping which enforces weights to decay when the distance to $\mathbf{p}$ increases. g is the data-dependent weighting term which compares the value at $\mathbf{p}$ with a neighborhood value and damps weight when the difference increases. g essentially prevents mixing values at different depth levels and thus maintains the original edges. Generally, f and g are taken to be Gaussian functions with standard deviations σ_f and σ_g. Real-time bilateral filtering is possible when the filter is implemented on the GPU. Figure 2.12 shows the benefit of a bilateral filter over a Gaussian filter in depth map filtering.

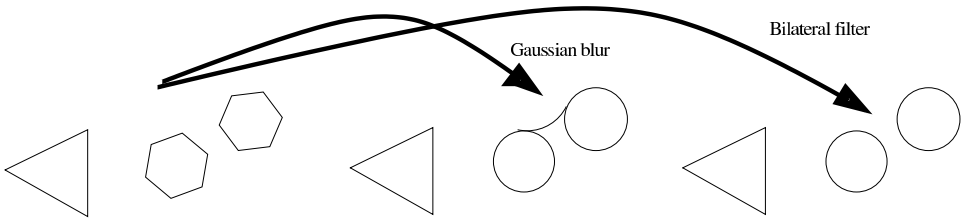


Figure 2.12: *Bilateral filter removes noise without altering range discontinuities.*

2.8.2 KinectFusion

The KinectFusion system [53] is a recent technique for computationally feasible camera tracking and reconstruction. It incrementally reconstructs a voxel-based 3D model of the scene which is used as motion reference. ICP is used to estimate camera pose (Sec. 2.5.4), because it can utilize dense depthmap measurements directly. However, ICP algorithms are not robust in planar environments. This is why problems can be expected for example at offices with flat walls. The voxel volume is improved in time by fusing current depth maps into it which are associated with ICP pose estimate. The 3D points are reconstructed using the current depth map and replaced by Truncated Signed Distance Functions (TSDF). TSDFs are 1D functionals, which describe a scalar density in the direction of a ray through a pixel. TSDF starts from $+1$ value, has 0-value at the expected surface and changes sign to -1 after the surface. The negative value is truncated shortly after the intersection point. Truncation is useful, because further intersections are not known due to occlusion. A smooth surface is defined as zero level set of the superposition of TSDF functions, and can it be reconstructed, for example, using a ray caster. To obtain a degree of adaptation to environment changes, the density values of the voxel grid are IIR filtered in time. This means that the influence of old depth maps decreases in time. Also normal directions of 3D points can be optionally used in filtering to favor measurements with similar normal. The KinectFusion system works well

in smaller than $3m \times 3m \times 3m$ operation volumes where sufficient grid density can be guaranteed. KinectFusion executes in real-time on a high-end GPU and consumes memory cubically as a function of voxel grid dimension.

2.8.3 Multi-resolution surfels

RGB-D measurements can also be converted into 6D Gaussians in 3D space, and treated as density functions. Stückler *et al.* describe a 3D environment as a multi-resolution surfel map [72]. Surfels are 2D elliptical surface patches whose coordinate system is determined by the two most significant principal components. New measurements are converted into surfels and associated with the model. The relative pose is estimated using statistical pose inference (Sec. 2.5.5). The benefit of surfel representation is that it allows fusing RGB-D measurements directly into the model by a simple Gaussian update. The surfels are stored in an octree whose resolution is adapted to measurement data distribution. Because surfels are oriented surfaces, 6 surfels are required per each octree node to take into account all viewing directions. The fused surfel maps are globally optimized using `g2o` framework [36].

- CHAPTER 3 -

DIRECT IMAGE-BASED ESTIMATION

2D feature point extraction and matching is an error-prone process which produces bias to estimated parameters. 3D pose and structure estimation can become more precise when using image-based cost functions to refine a good initial guess [59]. Direct image based cost functions *warp* image data into a reference view and measure the cost as a per-pixel difference [28, 2, 14]. The warping typically requires a 3D model or a relatively accurate 3D estimate of the environment geometry. When tracking a planar image region, homography mapping itself is sufficient for warping and explicit 3D model is not required [31]. Direct methods benefit of using more points to estimate parameters than sparse feature-based methods. Denser point clouds allow compensating with pixel noise, lighting variations and occlusions. However, sometimes the number of salient points in an image can be too small to allow parameter inference. Direct methods are thus often also directly dependent on the amount of texture in the environment. The direct estimation produces, at best, very precise results. In many applications, such as industrial robotics, precise 3D models exist and can be used. However, when estimating 3D structure also using image measurements, the pose estimation accuracy directly depends on the reconstruction accuracy, which can vary a lot especially when using dense stereo matching methods. Lighting variations can be problematic to model at local pixel level. The direct methods work well in environments with Lambertian surfaces, where 3D surface colors can be assumed to remain nearly constant independently of the viewing angle. Lighting variations have also been modeled at pixel level [46]. Due to view-dependencies in the appearance, direct approaches are at their best in local optimization when initial guess exists. In real-time applications, high frame rate is beneficial, because it reduces appearance changes at pixel level. The convergence domain can be extended by using multiple image layers.

§ 3.1 IMAGE REGISTRATION TECHNIQUES

Image registration techniques are rooted to the Lukas-Kanade method which aims at minimizing photometrical difference between two image patches [2]. The relation between a reference image $\mathcal{I}^*$ and the current image $\mathcal{I}$ is defined by the *warping function* $w(\mathbf{x})$, which generates re-sampling points using parameters $\mathbf{x}$. The warping function usually perform a 2D or 3D transformation to the reference points.

3.1.1 Lukas-Kanade optical flow

Optical flow models the image appearance change due to 2D fronto-parallel translation during a time interval dt [2]. The optical flow field is the velocity field representing how much a single pixel at (u, v) moves between consecutive frames. Optical flow methods are useful when tracking 2D points over a time interval.

An image $\mathcal{I}^*$ is captured at time instant t. When the camera moves a few frames in side-ways, perpendicular to the scene, the problem is to estimate the 2D translation parameters $\mathbf{x} = (du, dv)^T \in \mathbb{R}^2$ which satisfy Lambertian assumption

$$\mathcal{I}(\mathbf{p}, t) = \mathcal{I}(\mathbf{p} + \mathbf{x}, t + dt) \;\; \forall \mathbf{p} \in \Omega, \tag{3.1}$$

which implies that the only difference between two image regions captured at time instants t and $t + dt$ is a 2D displacement. $\Omega \in \mathbb{R}^2$ is the image region around a point of interest.

The scalar error function encoding the problem is

$$e(\mathbf{x}) = \sum_{\mathbf{p} \in \Omega} \Big(\mathcal{I}(w(\mathbf{x}, \mathbf{p})) - \mathcal{I}^*(\mathbf{p}) \Big)^2 = \mathbf{e}^T \mathbf{e}, \tag{3.2}$$

where $\mathcal{I}$ is the current image captured at time instant $t + dt$, $\mathcal{I}^*$ is the reference image captured at time instant t, the warping function $w(\mathbf{x}, \mathbf{p}) = \mathbf{p} + \mathbf{x}$, and the $n \times 1$ residual vector $\mathbf{e}$ contains error elements for every $\mathbf{p} \in \Omega$.

The Jacobian becomes

$$\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{x}} = \frac{\partial \mathcal{I}}{\partial \mathbf{p}} \frac{\partial w(\mathbf{x}, \mathbf{p})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathcal{I}}{\partial u} & \frac{\partial \mathcal{I}}{\partial v} \end{pmatrix}, \tag{3.3}$$

and the estimate becomes

$$\begin{pmatrix} du \\ dv \end{pmatrix} = (\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T \mathbf{e}. \tag{3.4}$$

This estimation works only for small displacements, because the pixel values change linearly only within 1 grid unit (see Fig. 3.1). Larger displacements can be estimated by bootstrapping the estimation using a flow field estimated at lower resolution.

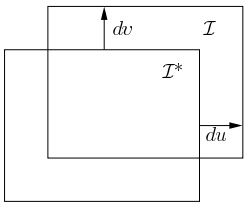


Figure 3.1: *Lukas-Kanade optical flow. A 2D displacement (du, dv) is estimated which minimizes image difference between $\mathcal{I}^*$ and $\mathcal{I}$.*

3.1.2 Remarks on image gradient computation

The image gradient can be evaluated exactly at the warped points by differentiating the interpolation function. Bilinear interpolation is defined by

$$\mathcal{I}(s,t) = (1-s)(1-t)I_1 + s(1-t)I_2 + stI_3 + (1-s)tI_4 \tag{3.5}$$

where I_1, I_2, I_3, and I_4 are the nearest pixel values, in clockwise-order starting from upper-left corner, and (s,t) is the sub-pixel coordinate of the warped sample point.

The differential becomes

$$\frac{\partial \mathcal{I}(s,t)}{\partial s} = -(1-t)I_1 + (1-t)I_2 + tI_3 - tI_4 \tag{3.6}$$

$$\frac{\partial \mathcal{I}(s,t)}{\partial t} = -(1-s)I_1 - sI_2 + sI_3 + (1-s)I_4. \tag{3.7}$$

Notice that bilinear interpolation is continuously defined only inside its one pixel domain. Discontinuities can be expected at the borders of a pixel. The Gaussian PSF would guarantee smooth derivatives over a larger domain, but it is computationally expensive for real-time applications. Also interpolation functions generally produce only approximate surfaces, which may not correspond to reality.

To obtain continuity over a larger image region, it is better to compute numerical gradients based on measurements

$$\begin{aligned} G_x &= \Big((I_{x+1} - I) + (I - I_{x-1})\Big)/2 = (I_{x+1} - I_{x-1})/2 \\ G_y &= \Big((I_{y+1} - I) + (I - I_{y-1})\Big)/2 = (I_{y+1} - I_{y-1})/2. \end{aligned}$$

This does not produce exactly correct intensity derivatives, but the benefit is that 3x3 regions provide larger convergence domain compared to operating only inside one pixel.

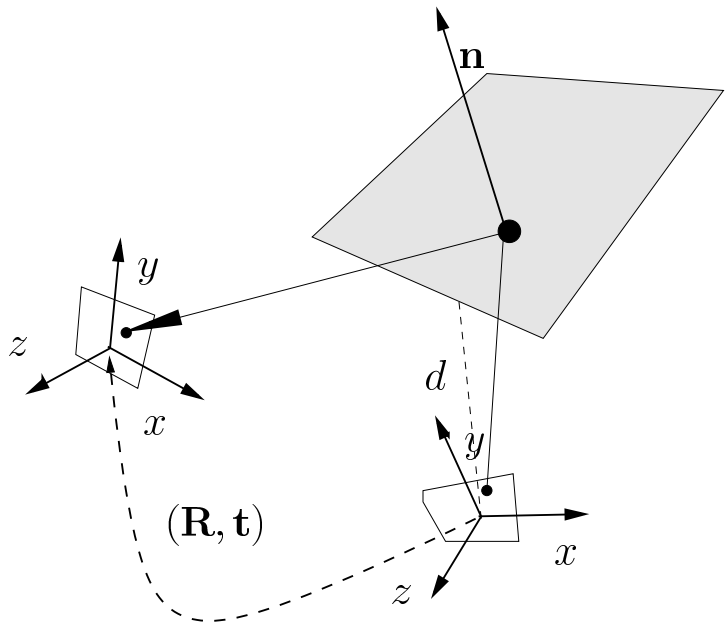


Figure 3.2: *A normal* $\mathbf{n}$*, plane distance to origin* d *and relative pose* $(\mathbf{R}, \mathbf{t})$ *define a homography mapping* $h : \mathbb{R}^2 \Rightarrow \mathbb{R}^2$ *between the camera image and the plane.*

3.1.3 Image registration using homography mapping

Camera tracking in 3D space becomes possible by introducing *a priori* information of surrounding 3D geometry into the estimation. If a planar region can be detected in an image sequence, a homography mapping describes the optical flow analytically. Planar homography mapping is essentially a 3×3 matrix

$$\mathbf{H}(\mathbf{x}, \mathbf{n}, d) = \mathbf{R}(\mathbf{x}) - \frac{\mathbf{t}(\mathbf{x})\mathbf{n}^T}{d}, \tag{3.8}$$

where $\mathbf{x} \in \mathbb{R}^6$ are the motion parameters, $\mathbf{n}$ is the plane normal and d is the plane distance to origin (Fig 3.2).

The warping function for a homography is then

$$w(\mathbf{x}, \mathbf{n}, d, \mathbf{p}) = \mathbf{K}\mathbf{H}(\mathbf{x}, \mathbf{n}, d)\mathbf{K}^{-1}\mathbf{p}, \tag{3.9}$$

where $\mathbf{p} = (u, v, 1)^T$ is a 2D point.

Using a planar homography for camera pose estimation is accurate and works even with a monocular view. However, during motion the reference patch must be updated before it moves outside the camera view. Reference update systematically introduces cumulating drift. Also if the reference patches are automatically initialized, the physical geometry under a patch may not always be planar and tracking bias will be introduced.

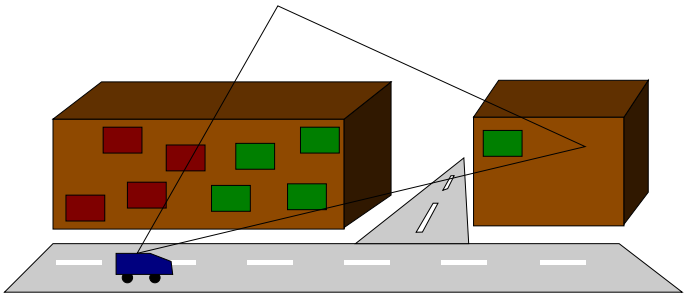


Figure 3.3: *Planar patch based pose estimation by Silveira* et al *[68]. Planar patches are automatically detected from image and tracked. When patches go out of view, new patches are initialized.*

Several algorithms have been developed to track a camera based on a set of known 3D planar surfaces [68]. Silveira *et al.* describe an image-based pose estimation approach using multiple planar patches extracted from an image (Figure 3.3). In their work, an effort is made also to model lighting variation by introducing a local gain parameter α for each patch and a global brightness parameter β. The residual elements to be minimized for a patch are

$$e_k = \Big(\alpha\mathcal{I}\big(w(\mathbf{x},\mathbf{n},d,\mathbf{p}_k)\big) + \beta\Big) - \mathcal{I}^*(\mathbf{p}_k), \tag{3.10}$$

where $\mathcal{I}^*$ is the reference image, $\mathcal{I}$ is the current image, and $\mathbf{p}_k$ are the points in 2D patch region $\mathbf{R}$. As can be seen, the cost function is direct pixel based error which does not rely on feature extraction.

In practise, the translations, rotations and normals must be initialized with a closed-form solution before the minimization can take place. The initialization is tricky because image noise may be more dominant than motion during the first frames. Specifically a sufficient translational motion is required. The problem is solved by tracking the patch region using a 8 degree of freedom homography matrix $\mathbf{H}$ without decomposition into $(\mathbf{R},\mathbf{t})$. After some frames, decomposition $\widehat{\mathbf{H}} \Rightarrow (\widehat{\mathbf{R}},\widehat{\mathbf{t}})$ would be possible, but it has two solutions [40]. Silveira *et al.* choose to evaluate the residual (eq. 3.10) using $\widehat{\mathbf{H}}$ and estimate subset $(\widehat{\mathbf{R}},\widehat{\mathbf{t}},\widehat{\alpha},\widehat{\beta})$ using approximate $\frac{\partial w}{\partial \mathbf{x}}$, without precise normals. Then $(\widehat{\mathbf{R}},\widehat{\mathbf{t}})$ are used to initialize the normals. After initialization phase, the estimation can be executed in SLAM mode, by updating all parameters simultaneously including the normals.

3.1.4 Image registration using a rigid 3D structure

Lukas-Kanade style minimization can be extended to minimize 3D transformation parameters of a fixed rigid 3D structure [14, 30, 59]. The projective pose estimation described in Section 2.5.3 can be modified to minimize

appearance-based cost. In this case, the warping function is defined by the camera transformation

$$w(\mathbf{x}, \mathbf{P}) = \mathbf{K}D\big(N(\mathbf{R}(\mathbf{x})\mathbf{P} + \mathbf{t}(\mathbf{x})), \boldsymbol{\alpha}\big), \tag{3.11}$$

and the pose parameters $(\mathbf{R}, \mathbf{t})$ are estimated by minimizing the following cost function

$$e(\mathbf{x}) = \sum_{k=1}^{m} \rho\Big(\Big(\mathcal{I}\big(w(\mathbf{x}, \mathbf{P}_k)\big) - \mathcal{I}^*\big(w(\mathbf{0}, \mathbf{P}_k)\big)\Big)^2\Big) = \mathbf{e}^T\mathbf{W}\mathbf{e}, \tag{3.12}$$

where m is the amount of 3D points $\mathbf{P}_k$, and $\rho(x) : \mathbb{R} \Rightarrow \mathbb{R}$ neglects statistically large displacements by damping them to zero (Sec. 2.3.4).

The residual elements are

$$e_k = \mathcal{I}\big(w(\mathbf{x}, \mathbf{P}_k)\big) - \mathcal{I}^*\big(w(\mathbf{0}, \mathbf{P}_k)\big), \tag{3.13}$$

and the Jacobian elements become

$$J_{kj} = \frac{\partial e_k}{\partial x_j} = \frac{\partial \mathcal{I}}{\partial \mathbf{p}} \Pi\mathbf{K} \frac{\partial D(\mathbf{p}, \boldsymbol{\alpha})}{\partial \mathbf{p}} \frac{\partial N(\mathbf{P})}{\partial \mathbf{P}} \frac{\partial \mathbf{A}(\mathbf{0})}{\partial x_j} \mathbf{P}_k. \tag{3.14}$$

The minimization process is illustrated in Figure 3.4. In *forward compositional* minimization, the pose increments are concatenated into the right side of the base transform $\widehat{\mathbf{T}}$ (Sec. 2.5.1). When concentrating on the image difference between $\mathcal{I}^w$ and $\mathcal{I}^*$, the base transform $\widehat{\mathbf{T}}$ becomes irrelevant to linearization and the geometrical optical flow $\frac{\partial w(\mathbf{0}, \mathcal{P})}{\partial \mathbf{x}}$ can be computed only once for all iterations. The gradient of the warped image $\mathcal{I}^w$ is computed in every iteration. Gradient computation is problematic especially when the reference point cloud is sparsified by saliency selection. With sparsified points, it is not certain whether a neighborhood of a point $\mathbf{p} \in \mathbb{R}^2$ is fully defined to compute the numerical gradient at $\Delta\mathcal{I}^w(\mathbf{p})$.

3.1.5 Inverse compositional image alignment

The inverse compositional approach is adopted for efficient minimization of the cost function [2]. The cost is reformulated as

$$\begin{aligned} \mathbf{c}^*(\mathbf{x}) &= \mathcal{I}^*\left(w\left(\mathcal{P}^*; e^{-\mathbf{A}(\mathbf{x})}\right)\right), \ \mathbf{c}^w = \mathcal{I}\left(w\left(\mathcal{P}^*; \widehat{\mathbf{T}}\right)\right) \\ \mathbf{e} &= \mathbf{c}^*(\mathbf{x}) - \mathbf{c}^w, \end{aligned}$$

where reference point colors in $\mathbf{c}^*$ are now a function of the inverse motion increment (Figure 3.5). When the current camera pose estimate $\mathbf{T}_g$ is expressed relative to the nearest reference $\mathbf{T}^*$, the initial base transform becomes $\widehat{\mathbf{T}} = \mathbf{T}_g(\mathbf{T}^*)^{-1}$. The current warped intensities $\mathbf{c}^w$ are produced by re-sampling $\mathcal{I}$ using $\widehat{\mathbf{T}}$. The residual $\mathbf{e}$ is then minimized by re-sampling

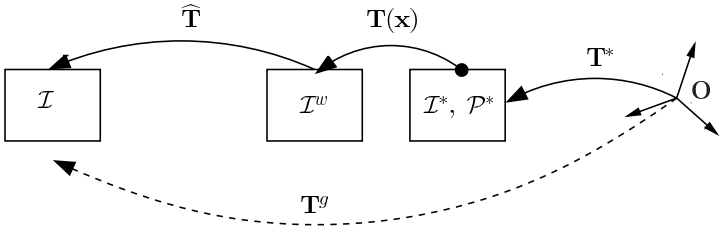


Figure 3.4: *Forward compositional image alignment. The reference points are warped from the reference to current image and the interpolated intensities generate warped image $\mathcal{I}^w$. The increment $\mathbf{T}(\mathbf{x})$ is estimated which minimizes the difference between $\mathcal{I}^w$ and $\mathcal{I}^*$. $\Delta\mathcal{I}^w$ must be updated in every iteration. $\mathbf{T}^g$ is the initial guess of the current pose and $\widehat{\mathbf{T}}$ is the base transformation.*

the reference image $\mathcal{I}^*$. The benefit can be observed from the form of the Jacobian

$$\mathbf{J}_{ij} = \frac{\partial \mathbf{c}(\mathbf{0})}{\partial \mathbf{x}} = \nabla \mathcal{I}^* \left(w(\mathbf{P}_i; \mathbf{I}) \right) \frac{\partial w(\mathbf{P}_i; \mathbf{I})}{\partial x_j}, \tag{3.15}$$

where $\nabla \mathcal{I}(\mathbf{p}) = \left[\frac{\partial \mathcal{I}(\mathbf{p})}{\partial x} \ \frac{\partial \mathcal{I}(\mathbf{p})}{\partial y} \right]$. Now $\mathbf{J}$ does not depend on $\widehat{\mathbf{T}}$ anymore and it can be computed only once for each $\mathcal{K}^*$ for better computational performance. The $\mathbb{SE}(3)$ group associativity enables collecting the estimated increment into the base transform by $\widehat{\mathbf{T}}^k e^{\mathbf{A}(\hat{\mathbf{x}})} \Rightarrow \widehat{\mathbf{T}}^{k+1}$.

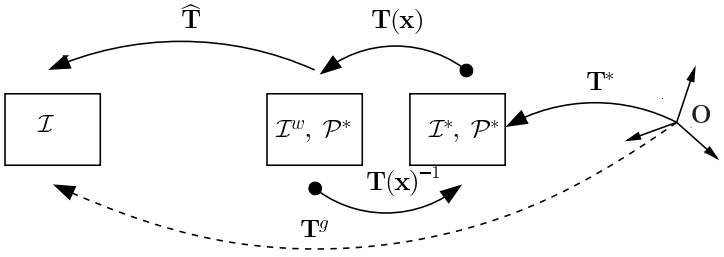


Figure 3.5: *Inverse compositional image alignment. The inverse increment is estimated which minimizes the difference between $\mathcal{I}^w$ and $\mathcal{I}^*$. The full Jacobian is computed only once using $\mathcal{I}^*$ and $\mathcal{P}^*$. $\mathbf{T}^g$ is the initial guess of the current pose.*

3.1.6 Multi-resolution pyramid for better convergence

Considering that image gradient can change its direction between every pixel, linearization from the motion parameters to pixel values is guaranteed to hold only within one pixel. This means that convergence domain depends on the image resolution and may remain small with high resolution images. Convergence domain can be improved by utilizing a

multi-resolution image pyramid during the estimation. In *coarse-to-fine* estimation, the motion parameters are first estimated using a low-resolution image and refined layer-by-layer into precise estimate.

According to *Nyquist sampling theorem*, *"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart"*.

In image processing, the function $x(t)$ is a 2D image. The statement means that a continuous image whose maximum frequency in Fourier domain is B sinusoidal cycles per a pixel, can be replaced without loss, by a discrete image whose sampling points are $1/(2B)$ pixels apart in the original image. Thus, depending on the image content, the multi-resolution pyramid may represent the original image or may not. When the image contains rapid intensity variation, low resolution layers are contaminated due to *aliasing*. In aliasing high frequency data is falsely converted into low frequency noise. Despite that a multi-resolution pyramid can not represent all images without loss, a larger convergence domain is typically obtained by using only few additional layers which are not too much contaminated by aliasing. In practise, the convergence domain using 320×240 images can be extended by a multi-resolution pyramid with layers in 320×240, 160×120, and 80×60 resolutions. When a multi-resolution pyramid is used in photometric camera tracking, discontinuities in the cost function may occur when switching between different resolutions. This is why it is better to use the highest available depth map resolution during tracking with all pyramid layers.

Figure 3.6 illustrates aliasing effect after 80×60 layer. To reduce aliasing effects, high frequencies are normally removed prior to downsampling using a low-pass filter (Figure 3.7). Then a discrete image is converted into a continuous form using an interpolation filter so that image can be resampled.

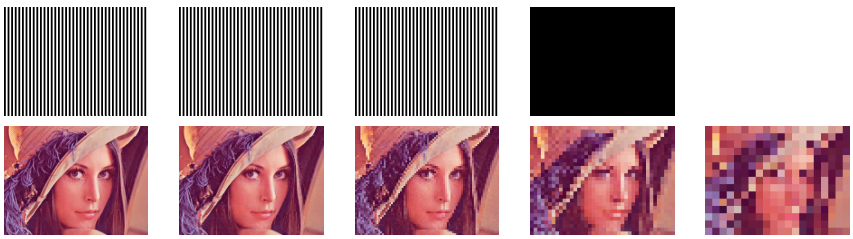


Figure 3.6: *A cycle of $B = 1/8px$ in 320×240 resolution downsampled into lower resolutions without filtering. Aliasing effects will occur after sampling points are more than 4 pixels apart (80×60). In 40×30 and 20×15 layers nothing is left of the original content. Lenna images also illustrated as reference.*

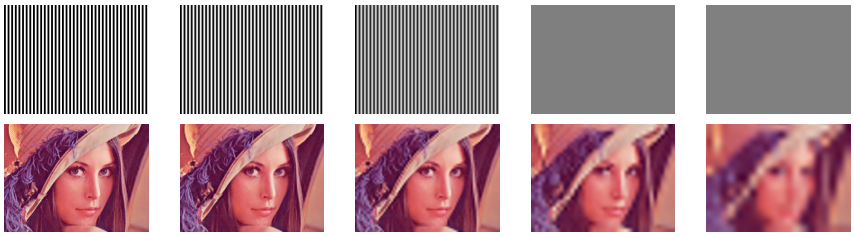


Figure 3.7: *A low-pass filter prior to downsampling removes high frequencies and maintains better image quality. The cycle images are reconstructed using bi-linear interpolation and Lenna using cubic interpolation.*

A rapid method to generate a multi-resolution pyramid is to downsampling layers recursively using a 2×2 box filter as anti-aliasing filter. This means that box filter size becomes $2^L \times 2^L$, where L is the layer index. The drawback of this approach is a small displacement in the layer images. The displacement occurs, because the origin of the downsampled image is averaged from 4 nearest neighbors, whose coordinates are $(0,0)$, $(1,0)$, $(0,1)$ and $(1,1)$. This displacement is taken into account by using the formula $\mathbf{p}_L = a\mathbf{p} + b$, when moving from high resolution coordinates to low resolution, where $a = \frac{1}{2^L}$ and $b = 0.5 * (a - 1.0)$. Figure 3.8 shows how the displacement is formed. In each layer, the units are scaled by $\frac{1}{2}$ and -0.25 displacement is applied.

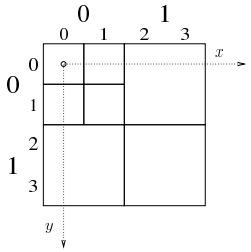


Figure 3.8: *The multi-resolution pyramid layers are displaced when a rapid* 2×2 *box filtering is used as anti-aliasing filter.*

§ 3.2 DIRECT STEREO MATCHING METHODS

A slightly out-dated survey of dense matching methods is provided by Scharstein *et al* [63]. The survey divides dense matching into following subtasks:

1. matching cost computation;
2. cost (support) aggregation;
3. disparity computation / optimization; and
4. disparity refinement.

Then disparity parameters are estimated either locally or globally. Estimating general dense structure is computationally expensive, but typically local methods have the advantage of being more easily parallelizable, which makes them more attractive. Global methods try to find optimal dense matching of full images instead of focusing on individual points. According to Middlebury online stereo benchmark the global methods are generally more successful as they are capable of producing correct matches also in regions with homogeneous texturing or discontinuities. Global methods aim to estimate a smooth map where the disparity map may have complex inter-dependencies through smoothness constraints.

Both local and global methods first choose a comparison metric between stereo images. Then aggregation strategy differs between the approaches.

Local methods aggregate cost values from a templates, which are fixed around both projection points. The costs are typically evaluated using SSD (Sum-of-Squared-Difference), SAD (Sum-of-Absolute-Diffecence) or Census, which uses Hamming distance for comparing internal intensity orders within a template. Global methods, on the other hand, aggregate cost terms for the full disparity map configuration, without specific rules. Global method may need to visit all image pixels many times to produce a global score.

For local methods, the disparity computation typically follows Winner-Takes-All (WTA) principle, where the disparity value with lowest aggregated cost is chosen. After rectification, the solution space is 1D horizontal line, where all candidates are evaluated by exhaustive search. With global methods, dynamic programming and graph-cut methods are popular choices for discrete optimization over all map configurations.

Finally, the parameter estimates are refined by a sub-pixel refinement phase. The locally aggregated matching scores are enumerated in a discrete pixel neighborhood, and then based on a parabol fit, provide an estimate between the pixels.

The problem of local methods is the template size selection. Small template sizes produce unreliable matches and bigger ones inaccurate matches. This is why multi-resolution coarse-to-fine approach improves robustness and precision in template matching [90]. Global methods mainly suffer from computational complexity, but for example dynamic programming has proven to be efficient also for real-time tasks. Selecting smoothness parameters is also problematic, because universal values do not exist.

3.2.1 Semi-global block matching

Global methods are often computationally too demanding for real-time use, but a few methods, such as Semi-Global Block Matching (SGBM), have proven to be sufficiently efficient. An example of real-time dense matching method, semi-global block matching (SGBM) is described.

SGBM uses dynamic programming over a discrete cost volume which can be done efficiently by using FPGA, GPU or multi-core CPU [25]. It has been implemented into OpenCV and FPGA-based real-time implementation have also been developed [3]. The SGBM finds smooth disparity maps by minimizing the following energy functional

$$E(\mathbf{d}) = \sum_{\mathbf{p}} C(\mathbf{p}, d_{\mathbf{p}}) + \sum_{\mathbf{q}\in\mathcal{N}_{\mathbf{p}}} P_1 \delta(|d_{\mathbf{p}} - d_{\mathbf{q}}| - 1) \tag{3.16}$$

$$+ \sum_{\mathbf{q}\in\mathcal{N}_{\mathbf{p}}} P_2 \delta(|d_{\mathbf{p}} - d_{\mathbf{q}}| - k), \tag{3.17}$$

where $\mathbf{p} \in \mathbb{R}^2$ are center points of the pixels in the image grid and $d_{\mathbf{p}}$ the disparity values associated with them. $\mathcal{N}_{\mathbf{p}}$ is a set of points containing the 8 neighbor points of $\mathbf{p}$. $C(\mathbf{p}, d)$ is the block matching cost for $\mathbf{p}$ and P_1 and P_2 are the constant discontinuity penalties for unit jumps and greater jumps ($k > 1, k \in \mathbb{Z}$) [25]. In sequences with little lighting variations, the Birchfield-Tomasi metric is the recommended metric for block matching due to it's computational efficiency [76]. In block matching, single pixels scores in the block are locally aggregated to obtain more reliable score. In environments with lighting variations, lighting invariant block matching such as the BRIEF must be used instead [10]. The minimum of E is obtained by dynamic programming where the independent costs of 8 incoming 1D directions are maintained per each point. The global aggregation of the directional costs is done in two passes where, the first pass aggregates the costs propagated from upper and left sides and the second pass aggregates the lower and right side directions. Finally the disparities which have the smallest cost sum of the eight directions are selected. Penalty constants P_1 and P_2 need to satisfy $P_1 < P_2$ for penalizing slanted surfaces and discontinuities appropriately. Disparity values are refined into sub-pixel accuracy as a post-process. Occlusions are handled by *a consistency check*, which means computing disparity map from left to right and right to left and then removing the inconsistencies as occlusions. This is obviously an expensive approach as the disparity map will be computed twice. Finally the disparity values are refined to sub-pixel accuracy.

3.2.2 Stereo camera and calibration

A stereo camera consists of two cameras which have fixed baseline. The standard Caltech calibration kit is often used to obtain intrinsic and extrinsic parameters [6]. Since Caltech calibration decides to use only extracted

corner points in calibration, it could be further improved by refining the parameters using a direct method. However, due to wide use of the kit, the parameters are compatible with OpenCV and various other tools. The calibration procedure requires a stereo sequence, which contains a planar checkerboard pattern (Figure 3.9). The kit then finds the board corners and gives them as input to Zhang's method [91], which estimates the parameters. After stereo calibration procedure, $\mathbf{K}_1, \mathbf{K}_2$ and $\mathbf{T}_b$ are known and 3D point triangulation is well defined. Triangulation methods were discussed in Section 2.2. Depending on the optics of the camera, lens distortion may add radial and tangential displacement into the image which must be modeled and corrected (Section 2.1.3). Usually the parameters of the distortion are estimated as a part of calibration procedure. Although stereo baseline is often roughly a horizontal displacement, rectification is still performed for both views to ensure that epipolar lines are strictly horizontal. Then the baseline transformation simplifies into a horizontal displacement, and the disparity map is generated by storing per-pixel displacement d_k as a result of a dense matching method. Typically disparity maps are defined for the right stereo view and $d_k > 0$.



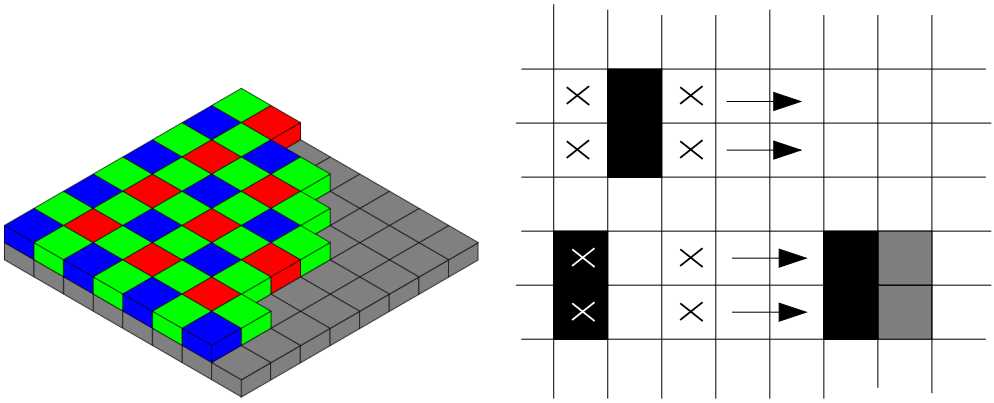
(a) (b)

Figure 3.9: *a) RGB image 1 b) RGB image 2*

3.2.3 Bayer filtering

In low-cost RGB cameras, such as Microsoft Kinect, a RGB measurement is not done per each pixel. Instead, each pixel receives either R, G, or B measurement, which are interpolated across the full image. The bayer filtering pattern is illustrated in Figure 3.10. Due to interpolation, the image edges can spread or disappear artificially and may not correspond with the true scene edges anymore. If de-Bayered RGB image is bi-linearly interpolated, two interpolations will be done. The other one can be eliminated by always

interpolated directly using original Bayer data [19]. The effect can be noticed especially when observing sharp edges (Figure 3.10). In this work, 640×480 images are Gaussian filtered by 5×5 kernel and sub-sampled into 320×240 resolution to filter out potentially faulty high-frequencies. When using image-based registration approach, the image quality should be as good as possible. With professional HD cameras, Bayer filtering is not used, but each pixel receives a unique RGB value.



(a) (b)

Figure 3.10: *a) Bayer pattern in Kinect sensor. b) Examples how interpolation can remove edges and spread them.*

§ 3.3 QUADRIFOCAL STEREO TRACKING

Via a disparity map based dense 3D point cloud, it is possible track the camera pose by minimized image difference at projected 3D points. Considering a 3D reference point $\mathbf{P}$ and the 2D projections $\mathbf{p}_1 \dots \mathbf{p}_n$ into n views, the correct motion $\mathbf{x} \in \mathbb{R}^6$ has the property that $c(\mathbf{P}) = c(\mathbf{p}_i(\mathbf{x}))$, for all $i \in [1, n]$ where the function c returns the color of a point. This property assumes Lambertian reflection for the surfaces of the environment, but the assumption holds particularly well for the views which have a small baseline to the reference view.

A quadrifocal relationship between all projection points $\mathbf{p}_i$ means that they are constrained by a common 3D point [21]. The constraint can be expressed as $3 \times 3 \times 3 \times 3$ array of scalar parameters called *the quadrifocal tensor*. In an application to dense vSLAM, the constraint can be parameterized by 3D motion parameters $\mathbf{x}$ which define the pose between two stereo views. Assuming a fixed reference view for which dense matching has been solved, 3D points can be expressed either by projection pairs

$(\mathbf{p}_1^*, \mathbf{p}_2^*)$ but also by $(\mathbf{p}_1^*, \mathbf{l}_2^*)$ where 3D points are generated at the intersection point of a ray and a back-projected plane through $\mathbf{l}_2^*$. $\mathbf{l}_2^*$ can be arbitrary 2D line through projection point $\mathbf{p}_2^*$. By varying the motion parameters $\mathbf{x}$, the quadrifocal constraint fixes the matching projection points in the current stereo view. Thus it can be used as a model to warp points from the reference view into the current view.

In the work of Comport *et al*, the quadrifocal warping of reference points into the current view is formulated in terms of two *trifocal* tensors in a stereo camera setting [14]. Trifocal tensor is a $3 \times 3 \times 3$ array of scalar parameters which defines the relative pose between three views. The first trifocal tensor warps the 3D point defined by projection pair $(\mathbf{p}_1^*, \mathbf{l}_2^*)$ into $\mathbf{p}_3(\mathbf{x})$ in the right view of the current camera. The second trifocal tensor, correspondingly, warps the point into $\mathbf{p}_4(\mathbf{x})$ in the left view of the current camera. Interestingly, the trifocal warping of 3D points from the reference view into the current view this way is equivalent to homography mapping, where homographies are defined per-point. The motion fitness is evaluated by color consistency $c(\mathbf{P}) = c(\mathbf{p}_i(\mathbf{x}))$. When the two trifocal warps are applied simultaneously using the relative pose in the adjoint map, the quadrifocal relationship holds between all four projection points $(\mathbf{p}_1^*, \mathbf{p}_2^*, \mathbf{p}_3(\mathbf{x}), \mathbf{p}_4(\mathbf{x}))$.

3.3.1 Stereo cost function

The cost function measures the intensity difference between the reference stereo image and the current stereo image. Bilinear interpolation is used to interpolate a color value per each projection point which is then stored in the *warped image* in the reference view. The warped image and the reference image are re-organized into 1D vectors, which are then directly compared pixel-by-pixel with robust least-squares metric. The cost function is

$$C(\mathbf{x}) = \mathbf{e}_R^T(\mathbf{x})\mathbf{W}_R\mathbf{e}_R(\mathbf{x}) + \mathbf{e}_L^T(\mathbf{x})\mathbf{W}_L\mathbf{e}_L(\mathbf{x}), \tag{3.18}$$

where $\mathbf{e}_R$ and $\mathbf{e}_L$ are the intensity residuals between right and left views. $\mathbf{W}_R$ and $\mathbf{W}_L$ are the robust weight matrices obtained from a M-estimator (Sec. 2.3.4). This cost function form does not require any predefined temporal correspondences as they are solved iteratively via estimating the camera pose increments. Figure 3.11 illustrates the components of the cost function. The minimization can be done using *coarse-to-fine* approach for improving the convergence domain. An image pyramid is generated from each original image by warping. The minimization starts from the lowest resolution images and, after convergence, proceeds to higher resolutions until the original resolution is met.

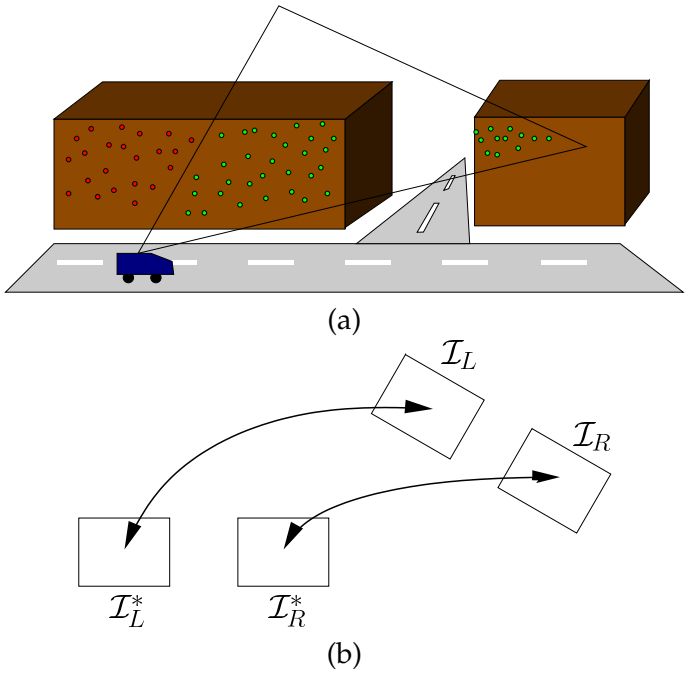


Figure 3.11: *a) In contrary to Silveira, Comport* et al. *use dense matched 3D points to direct visual odometry. b) Left and right stereo images are matched from the available 4 combinations.*

§ 3.4 PIXEL SELECTION

The Jacobian of the cost function is composed of image gradient and geometric warp gradient $\mathbf{J} = \mathbf{J}_{\mathcal{I}} * \mathbf{J}_{\mathcal{G}}$. The elements of $\mathbf{J}$ will be zero for the points which do not have any image gradient ($\mathbf{J}_{\mathcal{I}} = 0$) or whose $\mathbf{J}_{\mathcal{G}}$ is invariant to a given motion. For example, points at infinity constrain only rotation but not translation. $\mathbf{J} = [\mathbf{J}^1, \mathbf{J}^2, \mathbf{J}^3, \mathbf{J}^4, \mathbf{J}^5, \mathbf{J}^6]$, where the columns $[\mathbf{J}^4, \mathbf{J}^5, \mathbf{J}^6]$ contain the differentials for the rotational movement. The elements of these columns corresponding to points that are far away will be close to zero. Meilland *et al* has taken into account these observations when compressing large outdoor environment maps, consisting of a set of spherical image and depth measurements [45]. An even amount of points are selected from each column based on their magnitude. The points which are already selected once are skipped. The sorting of columns is computationally expensive for real-time purposes which is why the processing is done offline for the map.

§ 3.5 LIGHTING VARIATIONS

Lighting variations can cause tracking problems when Δt between image measurements increases. Due to the Lambertian reflection assumption, the colors of 3D points should stay constant independently of the point of view. As the Δt increases, the colors can change either due to non-Lambertian reflection or by lighting variation and shadows. One approach to solve the problem is simply to maintain a sufficiently high frame rate for the assumption to hold. Explicit modeling of lighting variation is required in applications where high frame can not be maintained. Lighting variation can be divided into global and local variation, where global variation means overall image brightness changes and local variation are more complicated phenomena such as non-Lambertian surfaces, spotlights and ramps. Although some lighting effects, such as spotlights, can be tolerated by using a M-estimator, an explicit model for local variations is useful, because it allows using more data for the estimation. Meilland *et al.* have modeled lighting variations [46] in similar way as Silveira *et al.* [68]. However Silveira combines a global affine transformation with a robust model which is not possible in the former work.

The lighting tolerant residual is

$$e_k(\mathbf{x}) = (\alpha_k \mathcal{I}\left(w(\mathbf{x}, d_k)\right) + \beta) - \mathcal{I}^*\left(\mathbf{p}_k\right) \tag{3.19}$$

where $\mathcal{I}^*$ is the reference image, $\mathcal{I}$ is the current image, $w(\mathbf{x}, d_k)$ is the trifocal mapping which depends on 3D camera motion $\mathbf{x} \in \mathbb{R}^6$ and $\mathbf{p}_k$ are the points selected using *pixel selection* described in the previous section. The cost function is robust as residuals are weighted by $\mathbf{W}$. The lighting parameters are α and β, where α is the gain per each pixel and β is the global brightness. These parameters are updated during run-time based on current images. Fixed 3D model is assumed throughout tracking, but an approach is suggested for automatic 3D model acquisition.

§ 3.6 EFFICIENT SECOND-ORDER MINIMIZATION

Efficient Second-order Minimization (ESM) is a technique which can increase convergence speed in image registration tasks [68]. The idea is to replace

$$\mathbf{J} \Leftarrow \frac{1}{2}\Big(\mathbf{J}(\mathbf{0}) + \mathbf{J}(\mathbf{x})\Big), \tag{3.20}$$

where $\mathbf{J}(\mathbf{0})$ is the Jacobian of inverse compositional cost function and $\mathbf{J}(\mathbf{x})$ is the current Jacobian which can be computed by warping the current image data into reference coordinate system and using the same formula as with $\mathbf{J}(\mathbf{0})$. By this procedure $\mathbf{x}$ will be defined in the same coordinate system

without problems. ESM has been originally developed for homography based warping function which can continuously map image data. A numerical image gradient requires pixel samples both horizontally, and vertically at ± 1 units away from the reference pixel. When dealing with rigid 3D structures and a photometrical cost function, a problem arises for due to pixel selection and missing depth measurements, since the image gradient can not always be computed. There are two alternative methods to circumvent this limitation. One method to utilize ESM is to generate *support pixels* which are dilated from pixel selection mask by one unit. The support pixels are not used in the cost function, but merely enable obtaining the necessary pixels for gradient computation. If all pixel neighbors have a valid depth value, it is possible to warp a template region from the current image and compute the current gradient. Another method is to use the original pixel selection mask, sample the gradients from current image and then rotate the image gradients to match reference image orientation. Because the gradients can only be rotated in a 2D plane, only an approximation is possible by projecting the current motion estimate into a simple z-axis rotation.

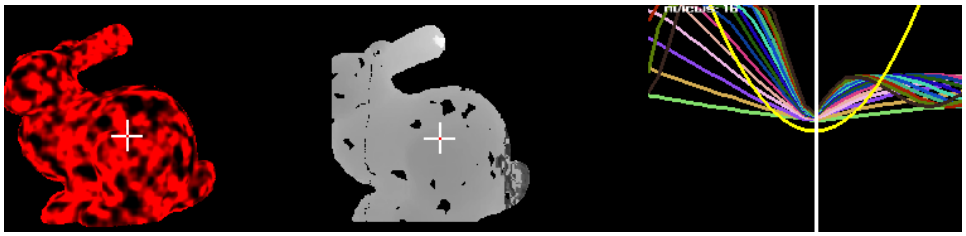
§ 3.7 PHOTOMETRICAL STRUCTURE REFINEMENT

Despite that the Kinect sensor is fairly accurate, the depth measurements still contain an amount of noise which can be reduced by combining information from multiple maps. In the context of this work, photometrical optimizations are interesting. The necessary pre-condition is that the precise depth values reside within bounded ranges, and image measurements with relatively accurate camera poses are available.

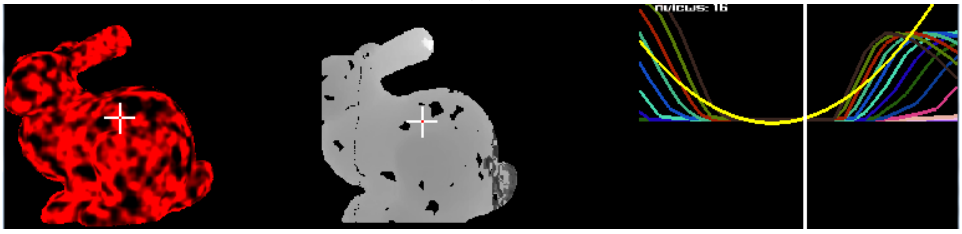
Plane sweeping method use a photometrical cost function for discrete depth optimization and can also operate in real-time [56]. The only difference to eq. 3.21 is that cost evaluations are grouped into planes with same depth value (Figure 3.13b). The space before the reference camera is discretized in planes. For every depth candidate z_k, every pixel of the reference view is back-projected onto this plane and re-projected into every additional view. Using these color values, a cost error can be calculated, allowing to derive the optimal depth for that reference pixel. In case lighting variations exist, normalized cross correlation can be used as as robust similarity measure for plane-sweep [88]. Due to independent point optimizations, the algorithm can operate in parallel. The precision can be increased, besides increasing the number of views, by taking into account a larger support region in the reference image. When the support region size is above 1 pixel unit, a fronto-parallel surface assumption is also introduced.

Figure 3.12 illustrates the cost curve within search bounds with and without image gradient. The depth map estimate is generated from a stereo

view using SGBM and the reference depth maps are produced by ray tracing.



(a)



(b)

Figure 3.12: *a) Textured Stanford bunny edges are associated with cost curves from 16 views. The vertical white line represents the correct depth value within the bounds. Yellow parabola's minimum represents the final image based estimate. b) Without edge information, image based estimation is not precise.*

When total variation prior is neglected, the photometrical optimization for individual 3D points is done by by defining a discrete number of solution candidates in domain $[z_m - \delta, z_m + \delta]$ around initial guess depth $\mathbf{z}_m$. A set of points $\mathbf{P}_k$ matching with the depth candidates z_k is then generated, and projected into the surrounding images (Figure 3.13). The optimal depth z_o is selected using a photometrical error function

$$z_o = \mathbf{e}_3^T \Big(\underset{k}{\operatorname{argmin}} \sum_{j=1}^{n} \left(\mathcal{I}^j(w(\mathbf{P}_k; \mathbf{T}_j)) - \mathcal{I}^*(w(\mathbf{P}_k; \mathbf{I})) \right)^2 \Big), \tag{3.21}$$

where $\mathbf{e}_3^T = (0, 0, 1)^T$ selects the z-coordinate of solution point $\mathbf{P}_o$. It is noted that this cost function will provide arbitrary results for points which are on homogeneous image regions without any gradient. Therefore photometrical refinement can only be done for the points with sufficient image gradient magnitude. In DTAM, this problem was addressed by assuming smoothness when the image gradient has small magnitude [52]. The benefit with discrete optimization is that it works even when the amount of image measurements is small. In our implementation, the refinement is not done in case the optimal depth is found at the search range boundary. This is to ensure that a local minimum is within the range.

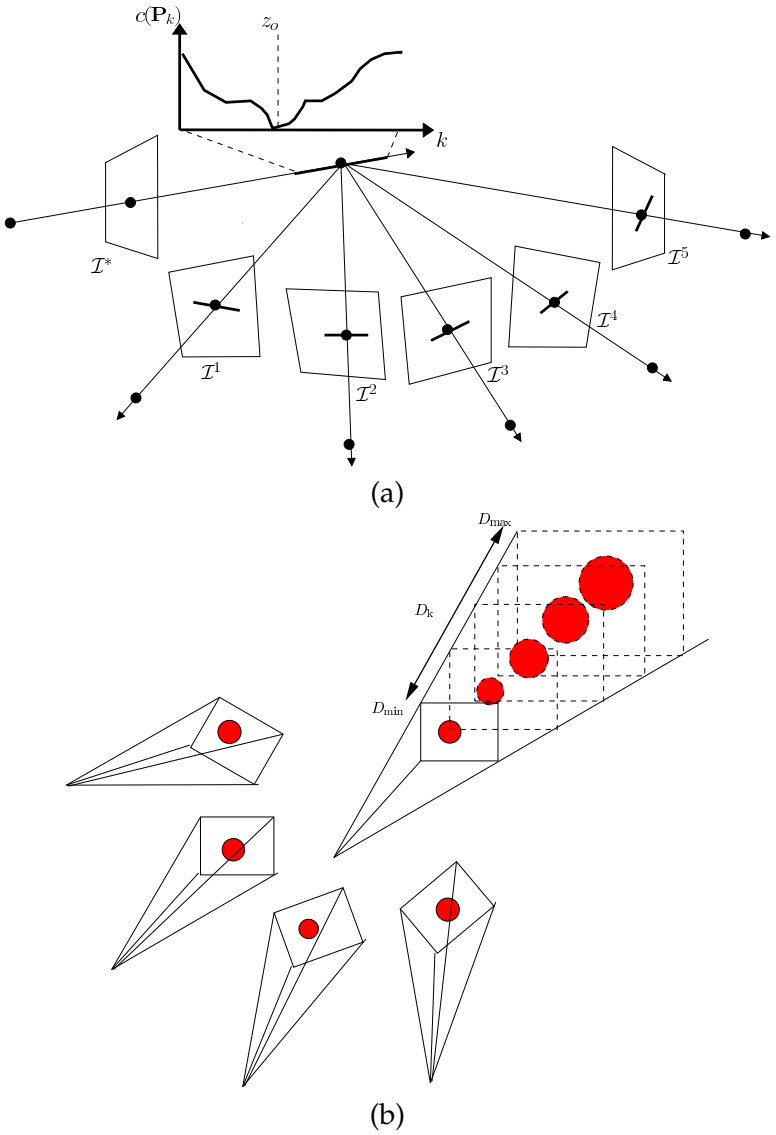


Figure 3.13: *a) Photometrical refinement using bounded depth range. The optimal depth has the most similar color in all images. The pose configuration is assumed to be precise. b) Traditional plane sweeping algorithm illustrated. A depth value at reference image pixel is determined by the best color consistency in multiple views.*

§ 3.8 Direct localization and mapping

A direct localization and mapping task is typically divided into a separate camera tracking and structure refinement problems, because simultaneous problem is difficult to solve. The main difficulties are 1) unknown noise

distribution with the point clouds, 2) maintaining low computational requirements, and 3) view-dependent measurements.

3.8.1 From outdoor stereo systems to indoor environments

The point clouds generated from an image sequence with varying viewpoint are contaminated by noise whose distribution depends on the image content and the reconstruction method. To be able to use the first-order Taylor approximation in photometrical pose estimation, the cost function must be assumed to be locally smooth. In stereo systems, especially local dense matching methods may produce noise in arbitrary magnitude, which may sometimes prevent further mathematical inference. Global matching methods work better in continuous and smooth environments, but easily produce false surfaces at discontinuities with arbitrary error magnitude. In textured outdoor environments, visual 3D maps have been experimented for urban navigation. The maps consist of spherical keyframe images with depth maps and GPS-based pose [45]. These maps have limited lifetime due to changing weather and lighting conditions. However, with a valid map, a robot which has cameras oriented to all directions can navigate in outdoor environments despite that some of the stereo-based depth maps may be contaminated by noise.

In indoor settings, more controlled lighting environment can be obtained. Due to recent developments with RGB-D sensors, the noise in depth maps has greatly reduced. Only a small bias is introduced when the distance to sensor increases to several meters [12]. RGB-D sensors can thus produce depth maps whose noise is local and almost Gaussian. The remaining noise can be removed almost fully by a simple depth fusion mechanism (one method described in sec. 7.1). Also image content is contaminated by pixel noise, which is due to CCD sensor noise, bi-linear interpolation, Bayer filtering, and complicated non-Lambertian surfaces. Compared to depth noise, pixel noise sources are less problematic, because their magnitude is relatively small. Thus, novel RGB-D sensors allow utilizing direct localization and mapping approaches indoors with accuracy never seen before. Audras *et al.* presented a CPU desktop system performing photometrical cost function minimization using a RGB-D sensor [1]. As follow-up work, Chapter 6 presents a GPU implementation, which is efficient and scalable with hardware development [80].

3.8.2 DTAM

DTAM is a system for real-time camera tracking and reconstruction which relies not on feature extraction but dense, every pixel methods [52]. It is a monocular system which aims at estimating dense textured depth maps at

selected keyframes. For some reason it does extra effort in depth map estimation by not utilizing a RGB-D sensor. Instead, it uses a non-convex optimization framework to minimize photometric cost and smoothness cost over the full depth map domain. Particularly total variation smoothness prior, imposed through *primal-dual* formulation minimization, is used to de-noise depth maps [73]. In primal-dual minimization, iteration becomes computationally more efficient by decoupling data and prior minimization into separate phases. Total variation prior is used to guess a smooth surface at image regions whose photometrical cost terms are small. Photometric RGB-D camera tracking is used to update camera pose for each incoming frame whose data can then be fused into depth map optimization. The algorithm is parallelisable throughout and DTAM achieves real-time performance using current high-end GPU hardware. Due to dense measurements DTAM becomes more robust than PTAM under heavy motion. The drawbacks of the system are a special initialization procedure, discretization of the depth range, and incompatibility with dynamic scenes. Smoothness terms also typically require parameter tuning.

CONTRIBUTIONS

- CHAPTER 4 -

EFFICIENT STEREO TRACKING BY VARIANCE BOUNDED DISPARITIES

In this chapter, bundle adjustment spirited joint cost function is presented for direct simultaneous localization and mapping. Bundle adjustment was discussed in Section 2.6. Direct estimation is often simplified into separate structure and pose estimation phases, due to its computational complexity. Here this simplification is not made and a Gaussian distribution for both pose parameters and disparity parameters [82]. Stereo camera is used as a RGB-D sensor and the disparity maps are generated from the stereo images by using a local dense matching method (Sec. 3.2). The initial matching is assumed to produce disparities whose error is local and Gaussian. The motion is initialized to $\mathbf{x} = \mathbf{0}$ assuming that previous increments have been concatenated to the base transform $\widehat{\mathbf{T}}$.

The joint cost function is

$$C(\mathbf{x}) = \mathbf{e}_x^T \mathbf{W}_x \mathbf{e}_x + \mathbf{e}_d^T \mathbf{W}_d \mathbf{e}_d, \tag{4.1}$$

where

$$\mathbf{e}_x = \mathcal{I}_R \left(w^R(\mathbf{x}, d_k) \right) - \mathcal{I}_R^* \left(\mathbf{p}_k^R \right) \tag{4.2}$$

$$\mathbf{e}_d = \mathcal{I}_L^* \left(w^L(d_k) \right) - \mathcal{I}_R^* \left(\mathbf{p}_k^R \right), \tag{4.3}$$

where $\mathbf{e}_x$ is the temporal residual between the subsequent images and $\mathbf{e}_d$ is the spatial residual of the direct and dense matching. $\mathbf{W}_x$ and $\mathbf{W}_d$ are the corresponding robust weight matrices. Section 4.1 presents the associated warping functions $w^*(\mathbf{x}, d_k)$ and $w^L(d_k)$ in detail. The task is to estimate the motion parameters $\mathbf{x} \in \mathbb{R}^6$ and the disparity bounds.

The cost function is direct and image-based as both structure and pose depend purely on a photometric cost metric. The estimation process in done soundly in two phases through marginalization and the conditional

disparity variances are calculated. These bounds are then propagated into the current stereo frame and they are used to boost disparity map computation.

§ 4.1 TRIFOCAL TENSOR WARPING

The warping from a reference stereo view to the current view can be modeled by a trifocal tensor. The trifocal tensor requires fixed projection points in the reference view and the motion parameters. As an example, the process is described from the right reference image into the right current image. A fixed reference pose of a stereo camera is expressed as $(\mathbf{I}, \mathbf{T}^{RL})$, where right camera view is chosen as the reference coordinate system which as identity transformation, and $\mathbf{T}^{RL}$ is the baseline transformation from the right view into the left view. The motion of the current right view is expressed relative to the reference as $e^{\mathbf{A}(\mathbf{x})}$.

The dynamical configuration is expressed as trifocal tensor by (s_1, s_2, s_3), where each matrix slice of the tensor is $s_j(\mathbf{x}) = \mathbf{a}_j \mathbf{b}_4^T(\mathbf{x}) - \mathbf{a}_4 \mathbf{b}_j^T(\mathbf{x})$, where $\mathbf{a}_j$ are the columns of $[\mathbf{T}^{RL}]_{3\times 4}$ and $\mathbf{b}_j(\mathbf{x})$ are the columns of $[(e^{\mathbf{A}(\mathbf{x})})^{-1}\mathbf{T}^{RL}]_{3\times 4}$. The homography mapping for a point k is denoted $h_k(\mathbf{x})$, whose columns j are defined as

$$h_k^j(\mathbf{x}) = s_j^T(\mathbf{x})(\mathbf{K}^R)^T \mathbf{l}_k^R, \tag{4.4}$$

where $\mathbf{K}^R$ is the intrinsic matrix of the right camera and $\mathbf{l}_k^R = [1, 0, -x_k^R]^T$ defines a vertical line through point $\mathbf{p}_k^R$.

For mapping all image points from the reference view to the current right view, the following two warping functions are required:

$$w^L(d_k) = \mathbf{p}_k^R + (d_k, 0)^T \tag{4.5}$$

$$w^R(\mathbf{x}, d_k) = N\left(\mathbf{K}^R h_k(\mathbf{x})(\mathbf{K}^L)^{-1}[w^L(d_k)\ 1]^T\right), \tag{4.6}$$

where $\mathbf{K}^L$ is the intrinsic matrix of left camera, $\mathbf{K}^R$ the intrinsic matrix of target view and $N(\mathbf{p}) = (u/w, v/w)^T$ for $\mathbf{p} = (u, v, w)^T$. $w^L(d_k)$ warps points to the left stereo view based on the given disparity map. $w^R(\mathbf{x}, d_k)$ and $w^L(\mathbf{x}, d_k)$ warp points under motion $\mathbf{x}$ into current right and left view. The process illustrated in Figure 4.1.

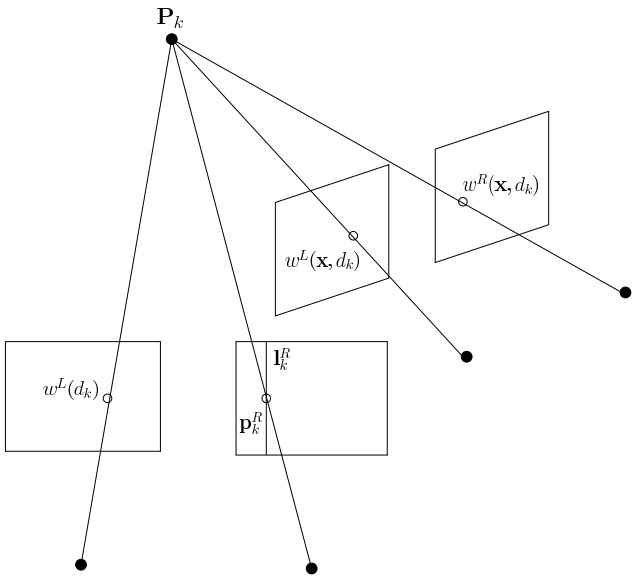


Figure 4.1: *The warping of a reference point pair into the current stereo view. The intersection point $\mathbf{P}_k$ of a ray through $w^L(d_k)$ and a plane through $\mathbf{l}_k^R$ is thus projected into next view using trifocal tensor.*

§ 4.2 ESTIMATION IN TWO PHASES

The weighted normal equations show the linear relationship between $(\Delta\mathbf{x}, \Delta\mathbf{d})$ and $(\mathbf{e}_x, \mathbf{e}_d)$

$$\mathbf{J}^T\mathbf{W}\mathbf{J}\begin{bmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{d}\end{bmatrix} = -\mathbf{J}^T\mathbf{W}\begin{bmatrix}\mathbf{e}_x\\ \mathbf{e}_d\end{bmatrix}, \tag{4.7}$$

where $\mathbf{J}^T\mathbf{W}\mathbf{J}$ is the inverse covariance (or Hessian). The equation can be decomposed into motion and structure specific blocks

$$\begin{bmatrix}\mathbf{J}_x^T\mathbf{W}_x\mathbf{J}_x & \mathbf{J}_x^T\mathbf{W}_x\mathbf{J}_{xd}\\ \mathbf{J}_{xd}^T\mathbf{W}_x\mathbf{J}_x & \mathbf{J}_{xd}^T\mathbf{W}_x\mathbf{J}_{xd} + \mathbf{J}_d^T\mathbf{W}_d\mathbf{J}_d\end{bmatrix}\begin{bmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{d}\end{bmatrix} = -\begin{bmatrix}\mathbf{J}_x^T\mathbf{W}_x\mathbf{r}_x\\ \mathbf{J}_{xd}^T\mathbf{W}_x\mathbf{r}_x + \mathbf{J}_d^T\mathbf{W}_d\mathbf{r}_d\end{bmatrix}, \tag{4.8}$$

where $\mathbf{J}_x = \frac{\partial\mathbf{e}_x}{\partial\mathbf{x}}$, $\mathbf{J}_{xd} = \frac{\partial\mathbf{e}_x}{\partial\mathbf{d}}$ and $\mathbf{J}_d = \frac{\partial\mathbf{e}_d}{\partial\mathbf{d}}$.

The equation 4.8 can be simplified into form

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{d}\end{bmatrix} = \begin{bmatrix}\mathbf{E}\\ \mathbf{F}\end{bmatrix}, \tag{4.9}$$

from which marginalized problems are obtained by Gaussian elimination

$$(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})\Delta\mathbf{x} = \mathbf{E} - \mathbf{B}\mathbf{D}^{-1}\mathbf{F} \tag{4.10}$$

$$(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})\Delta\mathbf{d} = \mathbf{F} - \mathbf{C}\mathbf{A}^{-1}\mathbf{E}. \tag{4.11}$$

As in Section 2.6.2, the marginalized covariances are $\mathbf{C}_x = \left(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1}$ and $\mathbf{C}_d = \left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}$. This pose estimation uses IRLS optimization to estimate the marginalized motion from the linear system

$$(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})\mathbf{x} = \mathbf{E} - \mathbf{B}\mathbf{D}^{-1}\mathbf{F}. \tag{4.12}$$

Locally, disparity estimation problem involves estimating a Lukas-Kanade problem (Section 3.1.1), whose Jacobian $\mathbf{J}_d = \frac{\partial \mathbf{e}_d}{\partial \mathbf{d}}$ depends on intensities in the template region. However, due to template size problem, these variances are valid only for fronto-parallel surfaces with sufficient texturing. On the other hand, if the disparities are not associated with a template and are treated as point offsets, the disparity cost function becomes a single pixel intensity comparison, which is vulnerable to noise. In this work, the latter cost function was chosen, despite that sufficient accuracy must be assumed with stereo images and dense matching.

§ 4.3 DISPARITY INITIALIZATION AND PROPAGATION

An important concern in selecting the dense matcher (Section 3.2) is that besides initialization, it can also be used for local refinement in a bounded domain. This is why local dense matching methods are of interest. Performance-wise they benefit of parallel computing and it is straightforward to re-localize a subset of points withing tight bounds while initializing the rest of the points from scratch.

Various local dense matchers (SSD, SAD, NCC, Daisy, and multi-resolution SAD) were experimented for producing initial disparity maps. Template matching methods perform the best when the environment consists of fronto-parallel surfaces with distinctive texturing. These two conditions are not always met in general environments and large mismatches can occur which produce structural noise of arbitrary magnitude. Disparity map noise can be reduced by multi-resolution approaches that regularize the 3D surfaces sufficiently. Therefore the estimates become for the majority part, only locally deviated. Three multi-resolution layers are selected in the experiments whose costs were summed up together. Yang *et al.* have shown that this approach can be implemented efficiently for real-time applications [90].

Because template matching is prone to gross outliers. A local smoothness constraint is valuable in filtering out bad matches. Disparity map smoothness are measured trivially by

$$S(\mathbf{p}) = \|\frac{\partial \mathcal{D}}{\partial u}(\mathbf{p})\| + \|\frac{\partial \mathcal{D}}{\partial v}(\mathbf{p})\|. \tag{4.13}$$

During tracking, the disparity maps become unions of 3D points which are measured at different time instants. The main idea is to gain computational efficiency by performing an exhaustive search only for the new points whose previous observations are not available. Initially for the first frame, the full 3D structure is determined by the multi-resolution SAD [90]. After previous motion has been estimated, the disparity values are searching withing bounded domains. The diagonal elements in the disparity covariance $\mathbf{C}_d$ matrix define the 1D variances for each d_k in the reference image. The intervals are solved as $[-p\sigma, p\sigma]$, whose end points are warped into the next reference frame for re-localization. σ_k are relative uncertainties up-to-scale, because the linear system can be multiplied on both sides by any non-zero scalar value. This is why reasonable scale p must be found experimentally. A sufficiently large p is selected to model also process noise. The disparity bound propagation is illustrated in Figure 4.2. The propagated bounds are projected into horizontal bounds, because their direction in the new reference image can be arbitrary. The re-localized disparities at discrete pixel coordinates are refined into sub-pixel accuracy by fixing the bound mid point in the left image, and by estimating the exact corresponding point in the right image based on the numerical cost gradient. The sampling points are filtered using 3×1 Gaussian kernel to reduce pixel noise. Re-localization is not done for the full disparity map, because pixel selection, M-estimator and variance thresholding sparsify propagated data. The combination map is illustrated in Figure 4.3.

§ 4.4 EXPERIMENTS

The Mars sequence using a real texture [50] was rendered in 640×480 resolution with Blender along with a ground truth trajectory. No lighting conditions were simulated and thus the model emitted texture colors directly. Three multi-resolution layers were used for motion estimation (see video[1]). A significant performance improvement is obtained by bounded search directly after the first frame (Fig. 4.5). Estimated trajectories for bounded and non-bounded cases are illustrated in Fig. 4.4. As can be seen, using a bounded search does not degrade pose estimation quality. The disparity map accuracy is improved by bounding in the problematic cases where the camera is closer to homogeneously textured Mars (Fig. 4.5). Structural accuracy is measured by $\frac{1}{n}\Sigma(d_k - d_k^o)^2$, where d_k^o are the ground truth disparities. In practice, they are obtained by projecting the intersection point between a ray and the object into the second view.

In another simulation, the method was evaluated using the Mars and Stanford bunny sequences, where stereo camera was rotated 360° around

[1] http://youtu.be/T7skSaNTvQo

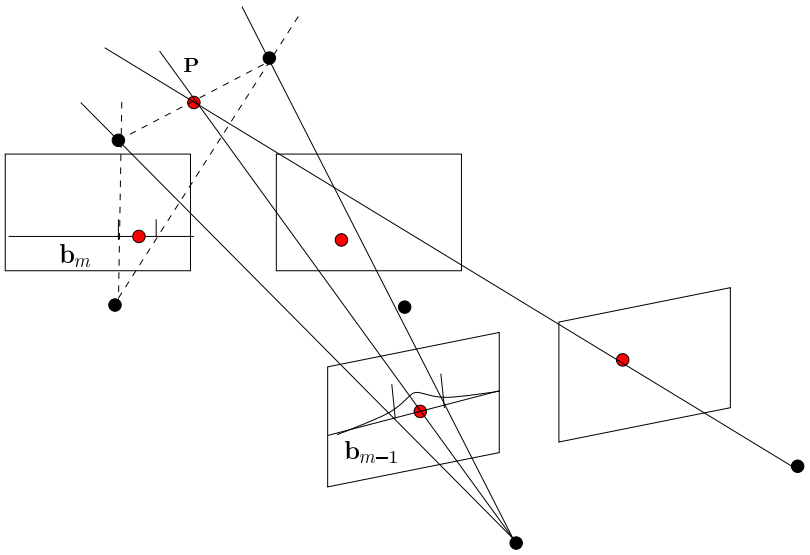


Figure 4.2: *The transfer of a bounds into next stereo view. A bound is defined as* $[-p\sigma_k, p\sigma_k]$ *interval using the disparity variance* σ_k*. The appropriate bound scale* p *is selected which includes also warping process noise.*

the object. This simulation experiment was done as a preparation for using a real turn-table sequence. Once again the amount of required template operations drops dramatically when bounding is used (Fig. 4.6). The curves show, how trajectory accuracy (Fig. 4.7) and structural accuracy (Fig. 4.8) are slightly improved due to bounding.

The proposed method is also tested using a real rock sequence (Fig. 4.5). A rock sequence was recorded using a calibrated stereo camera and a turn-table (Figure A.2). The original stereo images at 1280×960 resolution were first downsampled into 320×240, then background subtraction and rectification were performed to simulate a video recorded in space (see input videos[2][3]). A performance improvement is obtained by a bounded search directly after the first frame (Fig. 4.9). The camera pose trajectory is marginally better for the bounded version. Two multi-resolution layers were used for motion estimation (see video[4]).

An implementation of the proposed method was built with C++. Initially simple template matching was used for generating disparity maps, but as good template size was not found for real sequences, the exhaustive search had to be done using multi-resolution SAD, which also minimizes direct image error.

[2] `http://youtu.be/F0TkNRzG2aI`
[3] `http://youtu.be/TPbqk4bxkhE`
[4] `http://youtu.be/U7kE-bemLsw`

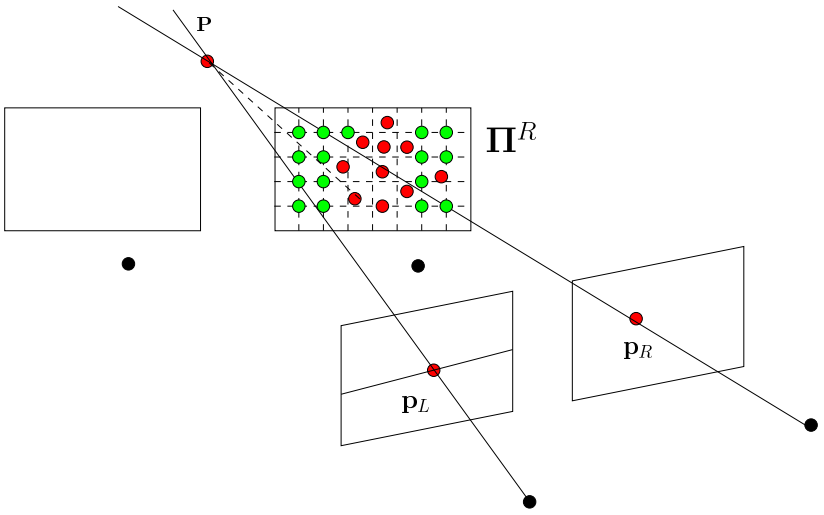


Figure 4.3: *The disparity map as a combination of old and new points. New points are initialized by using exhaustive search over the full epipolar line where as the old points are re-localized efficiently within tight bounds.*

§ 4.5 Analysis and limitations

The proposed approach models current motion and structure estimate using a single parameter vector, and propagates disparity bounds over time. The results show that pose estimation accuracy does not degrade while computational requirements with disparity map generation are dramatically reduced. Because motion error is larger than local structural error, the minimization uses Schur complement (eq. 4.1). The 6D pose estimation problem has shown to have good convergence properties, because the number of point measurements is large compared to number of parameters to be estimated [14]. The distribution of structural residual ultimately depends on the measurement hardware. In case, the distribution variance is large, global optimization method must be used. The marginalization works only when an initial guess exists and disparity parameters are only locally biased. Multi-resolution dense matching methods can provide relatively accurate disparity maps, but the results vary a lot depending on the image content. Template matching can, in worst case, produce errors with arbitrary magnitude. The obtained results are therefore optimistic when considering real application context. Semi-global block matching (SGBM) has the clear advantage over simple template matching based methods, because it abuses continuity constraint by penalizing discontinuities (Figure 4.10). Thus, relying on naive template matching produces significantly worse depth maps than global methods with compatible smoothness con-

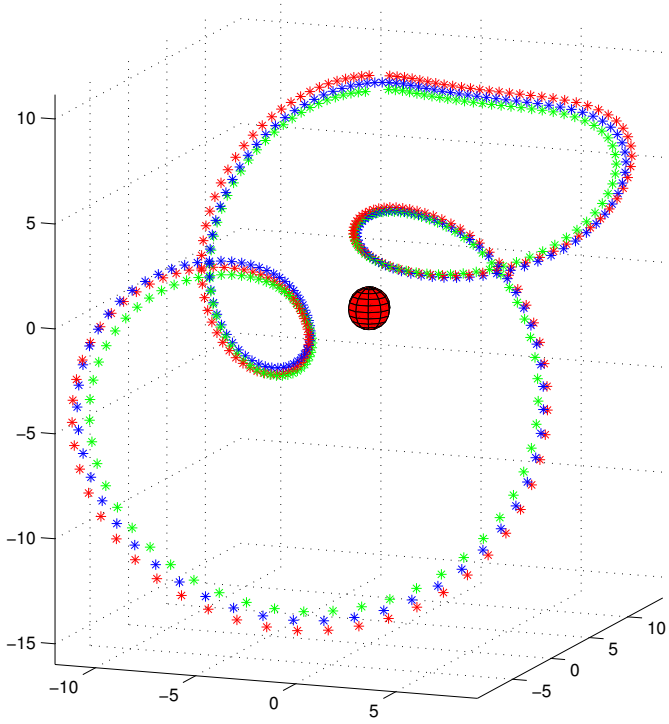


Figure 4.4: *Simulated Mars dataset, ground truth trajectory (blue) around Mars with estimated camera trajectories. Non-bounded in red ($\times$1.03) and bounded in green ($\times$0.97). As can be seen, the method can robustly track complicated 3D motion with low drift.*

straint. Global/semi-global matching methods on the other hand are computationally demanding. Real-time implementations of SGBM method exist for GPU and FPGA hardware [3]. When combining disparity measurements from multiple time instants, the disparity maps can be refined over time (see sections 7.1 and 3.7).

When a camera is moving rapidly and a photometric cost function is minimized, the reference image must be changed frequently to avoid image interpolation based inaccuracy. The Lambertian assumption holds well only within a short temporal window. If the camera is known to move slowly compared to the frame rate, another Schur complement (eq. 4.11) could be used to improve the reference disparity map accuracy. This, how-

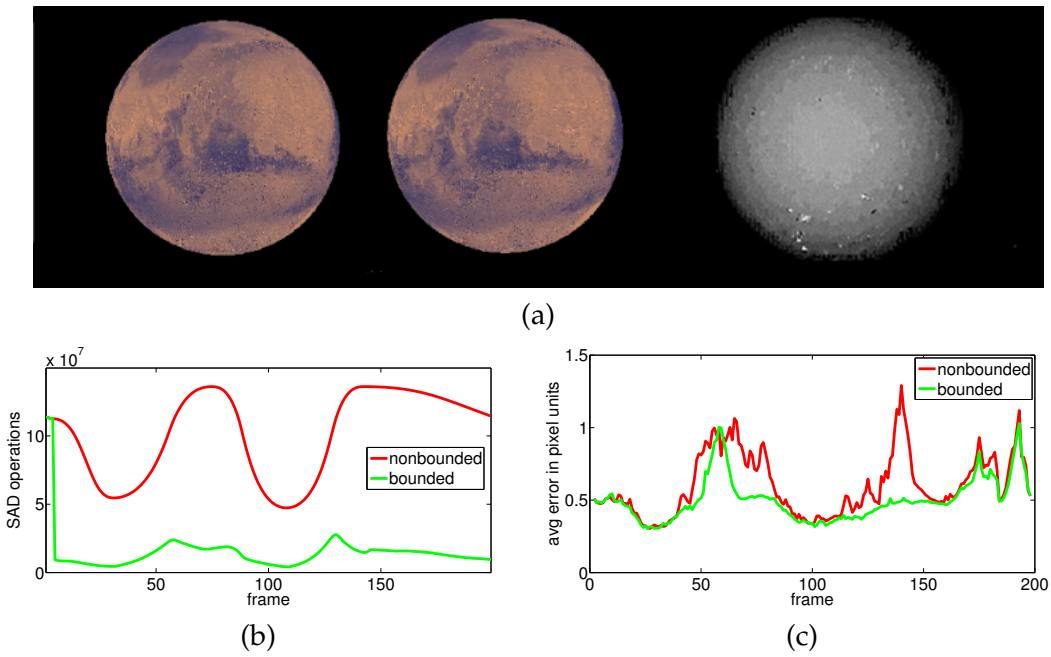
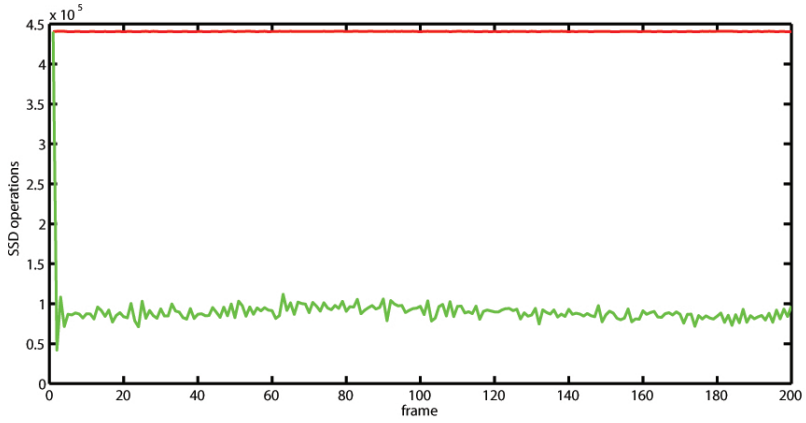


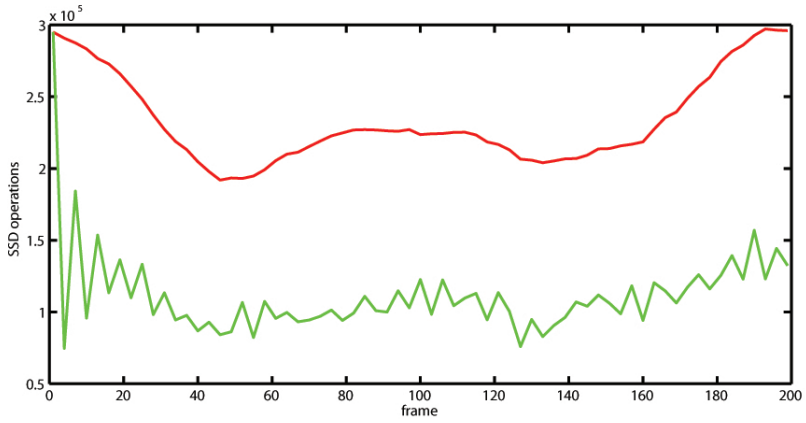
Figure 4.5: *a) Mars stereo sequence with the corresponding disparity image. Computational requirements of SAD dense matching are reduced without loss in tracking quality. b) The difference in amount of SAD operations with (green) and without disparity (red) bounding. The Mars silhouette area varies during the sequence causing fluctuations to the required computational requirement. c) The difference in disparity map accuracy with (green) and without (red) disparity bounding. In the problematic frames 60 and 140 camera is closer to Mars which, in general, has homogeneous texturing.*

ever, requires higher dense matching accuracy than what the current multi-resolution SAD approach produces.

The initial disparity parameters were refined into sub-pixel accuracy. By experiment, sub-pixel refinement improves tracking significantly. The problem, however, is that the disparity covariances depends on a very local image region, which becomes vulnerable to image noise. More stable disparity covariances are obtained from the Lukas-Kanade template matcher. Unfortunately larger template region implies degrades disparity precision. In this work, a design choice was made to maintain as accurate initial disparity values as possible, despite that covariance robustness decreases. As future work, a hybrid approach could be experimented where disparity values are kept accurate, but the covariances are computed using a larger image patch.

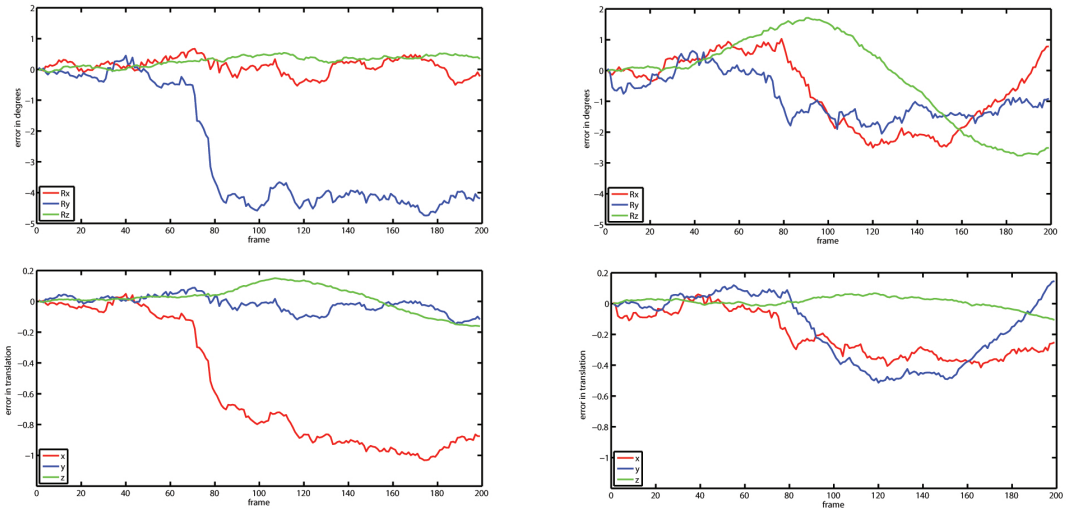


(a)

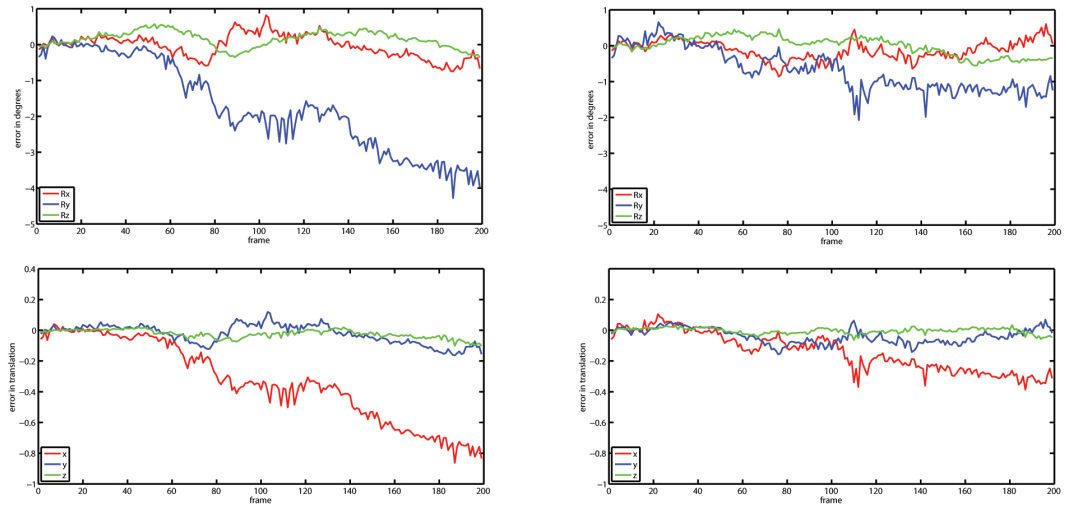


(b)

Figure 4.6: *The difference in the amount of SSD operations with (green) and without (red) bounding for a) 360° Mars sequence, b) 360° Stanford bunny sequence*

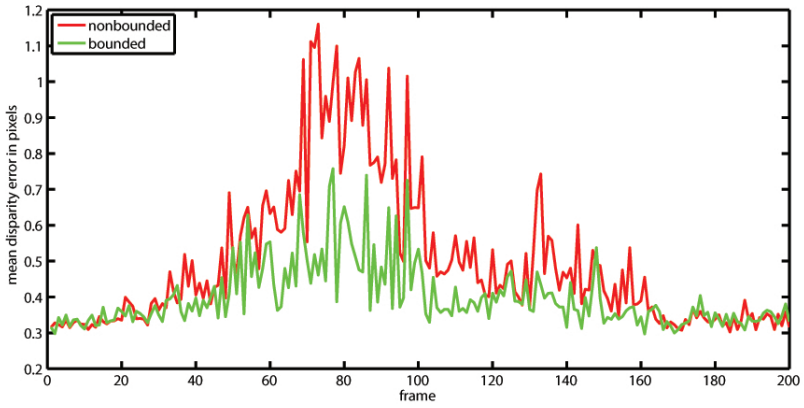


(a)

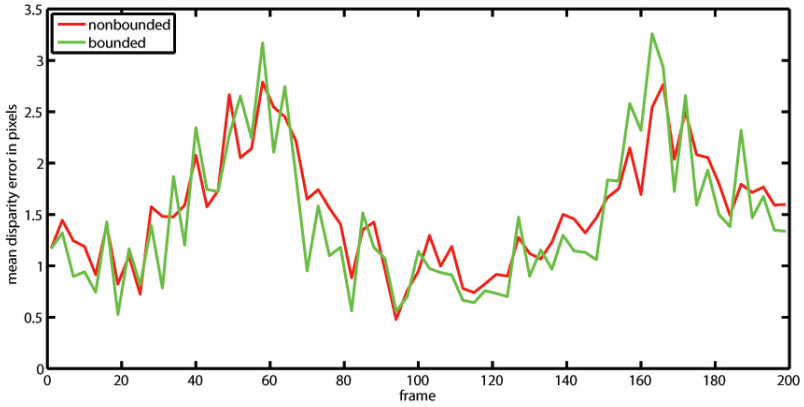


(b)

Figure 4.7: *The difference in the cumulative rotation and translation error without and with bounding for a) 360° Mars sequence, b) 360° Stanford bunny sequence. Rotation and translation errors are slightly reduced when bounding is enabled.*



(a)



(b)

Figure 4.8: *The difference in the disparity map error without and with bounding for a)* 360° *Mars sequence, b)* 360° *Stanford bunny sequence. The structural accuracy improves in case a) due to bounding. With Stanford bunny occlusions exist, which interfere bounding, but structural accuracy is still maintained.*

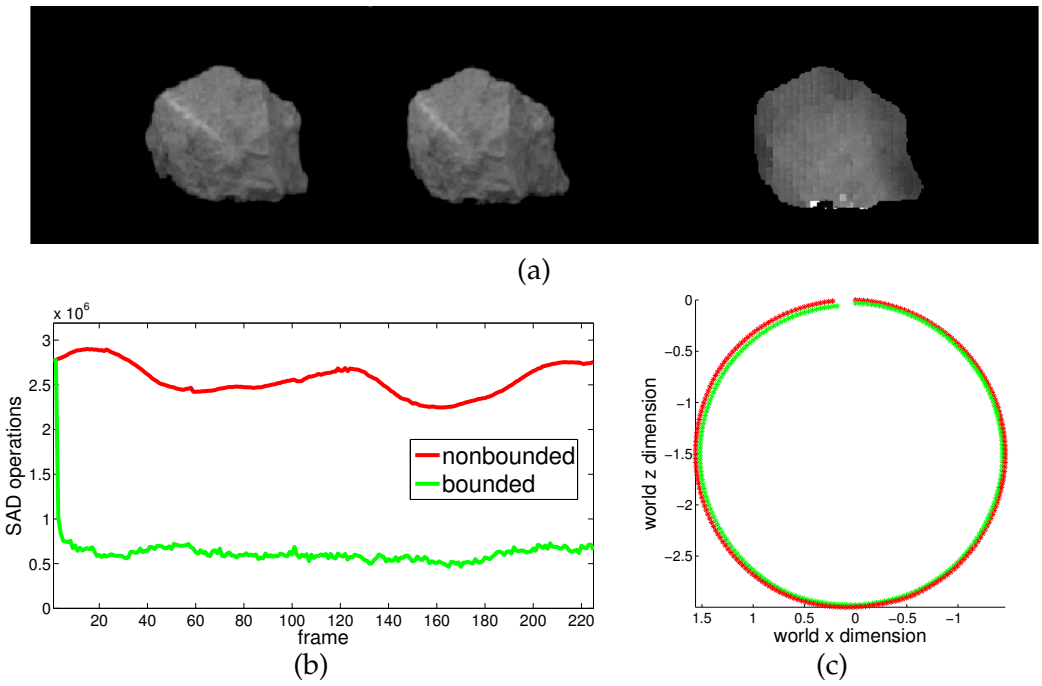


Figure 4.9: *a) Rock with the corresponding disparity image. b) The difference in amount of SAD operations with (green) and without disparity (red) bounding. c) The difference in bounded trajectory (green) and non-bounded trajectory (red). Bounded trajectory intentionally scaled by 0.98 for separation.*



Figure 4.10: *The difference in rock reconstruction quality between multi-resolution SAD (on left) and SGBM (on right). The rock image is illustrated in Fig. 4.9.*

- CHAPTER 5 -

ROBUST TRACKING BY CONCURRENT PIXEL AND DEPTH MATCHING

Already known research confirms, that using local bundle adjustment produces more accurate estimates than filtering in the context of localization and mapping [71]. Strasdat *et al*. conclude that : *"In order to increase the accuracy of visual SLAM it is usually more profitable to increase the number of features than the number of frames. This is the key reason why BA is more efficient than filtering for visual SLAM"*. When considering a visual odometry problem, this is a statement also in favor of using simply more data than propagating covariances in frame-to-frame basis. In this chapter, the measurements data is increased in frame-to-frame pose estimation by introducing depth residuals into estimation. The drift is reduced by also aligning the current depth map values with the reference points [81]. The combination produces an image-based method which is comparable with the standard ICP technique, but has advantages in computational requirements, precision and robustness. On the contrary to ICP, all data is stored in images which makes it possible to avoid expensive nearest neighbor searches in 3D space. As the cost function is directly image-based, it is therefore robust and precise. Instead of using KD-tree for point association [30], projective point association is utilized to favor computational performance. Additional efficiency is gained by a rapid, histogram based point selection procedure and M-estimation. Similar bi-objective minimization has also been experimented with Inertial Measurement Unit (IMU) and a hand-held camera [47]. The dense depth measurements that can be obtained by using a RGB-D sensor such as a stereo camera with a state-of-the-art dense matching technique or alternatively with a LIDAR, a structured light patterns (infra-red or visible light), sonar, etc. For indoor settings, the Microsoft Kinect provides good results at low costs (Section 2.8.1).

§ 5.1 COMBINING APPEARANCE AND STRUCTURE IN COST FUNCTION

A bi-objective least squares cost function is proposed which measures pose fitness using the cost

$$C(\mathbf{x}) = \mathbf{e}_I^T \mathbf{W}_I \mathbf{e}_I + \lambda^2 \mathbf{e}_Z^T \mathbf{W}_Z \mathbf{e}_Z, \tag{5.1}$$

where

$$\mathbf{e}_I = \mathcal{I}\left(w(\mathcal{P}; \widehat{\mathbf{T}}\mathbf{T}(\mathbf{x}))\right) - \mathcal{I}^*\left(w(\mathcal{P}; \mathbf{I})\right) \tag{5.2}$$

$$\mathbf{e}_Z = \mathcal{Z}\left(w(\mathcal{P}; \widehat{\mathbf{T}}\mathbf{T}(\mathbf{x}))\right) - \mathbf{e}_Z \widehat{\mathbf{T}}\mathbf{T}(\mathbf{x})\mathcal{P}, \tag{5.3}$$

where $\mathbf{e}_Z = (0, 0, 1)$ and $\mathbf{W}_I$ and $\mathbf{W}_Z$ are the diagonal weight matrices obtained from a M-estimator. $\mathcal{I} : \mathbb{R}^2 \Rightarrow \mathbb{R}$ is a color brightness function and $\mathcal{Z} : \mathbb{R}^2 \Rightarrow \mathbb{R}$ is a depth function. Reference variables and functions are denoted by *. $\mathbf{e}_Z$ measures depth map discrepancy, but it can be further improved by neglecting penalization of tangential motion (sec.2.5.4).

5.1.1 Bi-objective minimization

Since $C(\mathbf{x})$ (eq. 5.1) is a non-linear function of the unknown pose parameters, it has to be linearized with $\mathbf{x}$ for iterative minimization. With linearization it is assumed that function is locally continuous, smooth and differentiable.

This results in a Jacobian of the form

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \mathbf{J}_1 \\ \lambda \mathbf{J}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{J}_\mathcal{I} \mathbf{J}_w \mathbf{J}_\mathbf{T} \\ \lambda(\mathbf{J}_\mathcal{Z} \mathbf{J}_w \mathbf{J}_\mathbf{T} - \mathbf{e}_Z \mathbf{J}_\mathbf{T}) \end{bmatrix}, \tag{5.4}$$

where $\mathbf{J}_\mathcal{I}$ and $\mathbf{J}_\mathcal{Z}$ are the image and depth gradients with respect to pixel coordinates of dimension $n \times 2n$, $\mathbf{J}_w$ is the derivative of perspective projection of dimension $2n \times 3n$, and $\mathbf{J}_\mathbf{T}$ represents motion of a 3D point respect to motion parameters $\mathbf{x}$ with a dimension of $3n \times 6$.

$\mathbf{x}$ is obtained using the pseudo-inverse by

$$\Delta\mathbf{x} = -(\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{W} \begin{bmatrix} \mathbf{e}_\mathcal{I} \\ \lambda \mathbf{e}_\mathcal{Z} \end{bmatrix} \tag{5.5}$$

The increments are updated into the base transformation by $\widehat{\mathbf{T}} \Leftarrow \widehat{\mathbf{T}}\mathbf{T}(\mathbf{x})$ as long as it is necessary until reaching convergent condition $\|\mathbf{x}\| < \epsilon$.

Now since the depth component depends directly on the pose parameters, forward compositional alignment is proposed for bi-objective minimization, because the Jacobian must be updated in every iteration.

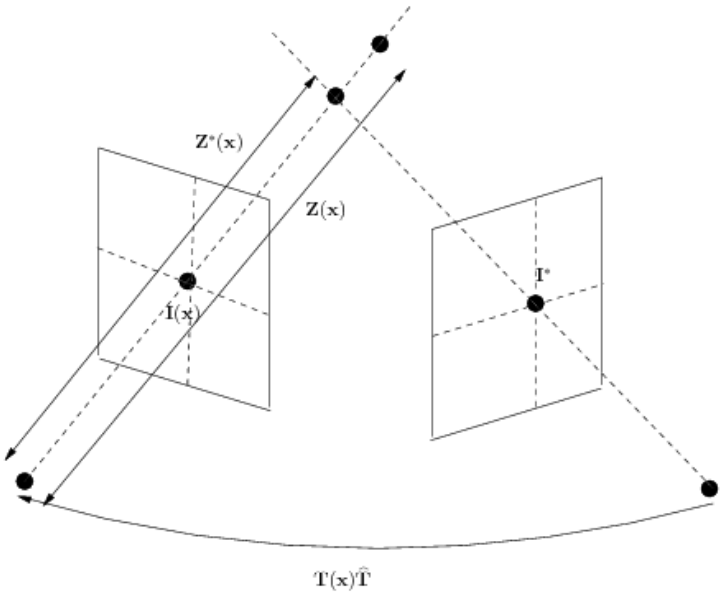


Figure 5.1: *Photometric and depth errors are minimized between subsequent image frames. In practise, the 3D point associated with color $\mathcal{I}^*$ is transformed and re-projected into the current coordinate system where it is possible to compute $\mathbf{e}_\mathcal{I} = \mathcal{I}(\mathbf{x}) - \mathcal{I}^*$ and $\mathbf{e}_\mathcal{Z} = \mathcal{Z}(\mathbf{x}) - \mathbf{Z}^*(\mathbf{x})$.*

5.1.2 Balancing the cost by λ

Looking at the Hessian matrix approximation,

$$\mathbf{H} = \mathbf{J}^T\mathbf{W}\mathbf{J} = \mathbf{J}_1^T\mathbf{W}_\mathcal{I}\mathbf{J}_1 + \lambda^2\mathbf{J}_2^T\mathbf{W}_\mathcal{Z}\mathbf{J}_2, \tag{5.6}$$

it can be seen how the local curvature of the cost function depends on both intensity and depth differentials. The benefit of incorporating depth measurement shows in this form where the Hessian is less likely to be singular, because the curvature is gathered from two sources. $\mathbf{H}$ can be singular in the cases where motion does not infer any appearance changes (e.g. homogeneous regions) or motion infers arbitrary appearance changes (e.g. non-Lambertian surfaces, occlusions). λ acts as a gain for the depth component and it is necessary to adjust it for proper balancing of the error function.

Depending on the metric unit of the 3D coordinate system and the local covariances of the components, one of the components of the residual may be negligible or fully dominate the error. This happens for example when pixel noise is numerically comparable to depth variations. Mathematically the wrong choice of λ results in cases $\mathbf{H} \approx \mathbf{J}_1^T\mathbf{W}_\mathcal{I}\mathbf{J}_1$ and $\mathbf{H} \approx \mathbf{J}_2^T\mathbf{W}_\mathcal{Z}\mathbf{J}_2$, where the fusion does not bring any benefits.

The optimal λ is the one which improves the estimated camera trajectory the most when compared to a purely intensity based cost function

(Section 3.1.4) or plain ICP (Section 2.5.4). There are mathematical methods such as L-curve based metrics and cross validation for automatic selection [47]. L-curve based selection finds such *Pareto optimal* λ which minimizes both cost components simultaneously by finding the optimal point in the curve whose axices represent the costs of the separate components. In cross validation, on the other hand, a geometry independent λ is learned by finding such parameter which minimizes projection error for an average cost over all *leave-one-out* combinations of the point data.

The manual selection of λ can be done by simulating the real application by an image sequence with depth maps and experimenting with different λ values. In many application cases the optimal λ can be fixed only once at the beginning if the depth range is known a priori. To automatically determine λ in real-time, a robust ratio between the centers of the $\mathcal{I}$ and $\mathcal{Z}$ distributions is proposed such that $\lambda = |(\sigma_I/\sigma_Z) * \text{median}(\mathcal{I})/\text{median}(\mathcal{Z})| \approx |\text{median}(\mathcal{I})/\text{median}(\mathcal{Z})|$. By experiments it was noticed that this produces close to manually chosen values for the test sequences used. This formula finds a robust scale factor which balances between the relative uncertainty of the components assuming unit standard deviations σ_I and σ_Z. After balacing both depth and intensity distributions have similar numerical variance.

5.1.3 Multi-resolution and depth filtering

Commonly multiple resolutions are used for increasing the convergence domain. The minimization starts from a low resolution and the solution is refined using a sequence of higher resolution images. For generating a multi-resolution pyramid each layer has to be low-pass filtered and then sub-sampled to avoid aliasing effects. In the case of depth images, low-pass filtering is problematic as it alters the underlying 3D structure. For preventing this, depth values are always sampled at the highest resolution even though matching intensity values are low-pass filtered.

When downsampling images using a 2×2 box filter, the sub-samples are produced in the middle of each 2×2 region. This needs to be taken into account, when matching points between different layers. In a case of sampling the high resolution depth image, the bilinear filtering coordinates at higher resolution are $x_h = 2^L x + 0.5(2^L - 1)$, where L is the amount of layers in-between.

5.1.4 Hybrid pixel selection

It is proposed that pixel selection is done efficiently without sorting the columns $\mathbf{J}$ by neglecting the geometrical component of Jacobian and focusing only on image and depth image gradients. Thus reference points $\mathcal{P}$



Figure 5.2: *A. Stanford bunny image. B. The best* 15% *pixels selected by image gradients. C. The best* 15% *pixels selected by depth gradients. Combination selection is required for constraining the motion the most efficiently.*

are selected into motion estimation using a 2D selection mask where the selection fitness is the combination gradient magnitude

$$S(x,y) = |\nabla \mathcal{I}_x| + |\nabla \mathcal{I}_y| + \lambda'(|\nabla \mathcal{Z}_x| + |\nabla \mathcal{Z}_y|), \tag{5.7}$$

where λ' finds the balance between the depth and intensity gradient.

Selection scores $S(x,y)$ are accumulated into a histogram and a portion of pixels are selected from the end of the histogram. A histogram is useful, because it is trivial to compute such a threshold which selects n best pixels efficiently without explicit sorting. A naive pixel selection first sorts n points in $O(n\log n)$ and then selects k points with the greatest $S(x,y)$ score. Using a histogram the complexity is $O(n)$, because full image sorting is avoided. A comparison of intensity and depth gradient based pixel selections is illustrated in Figure 5.2. Note, that same method can be used in M-estimation phase to avoid median computation. These optimizations are used also in the developed GPU implementation (Chapter 6).

Traditional point-to-plane ICP (section 2.5.4) does not focus only on depth map gradient regions but takes the advantage of full 3D point set. It, however, suffers from a degenerate case when the scene is mostly planar, because the motion is not fully constrained. By focusing on gradient regions only, the computational requirement is reduced and the estimate does not easily drift, because the gradient regions often contain varying normals [61].

§ 5.2 Simulation experiments

Robustness and accuracy improvements were observed using two simulated sequences. In the first sequence, synthetic RGB-D sensor was set to rotate around Stanford bunny, whose image was toggled between textured

and plain white. Because intensity based pose estimation relies on image gradients, divergence occurs once in a while when the edge information of the silhouette is not sufficient for tracking (Figure 5.5). This problem is fixed by incorporated depth maps. The depth map matching keeps **H** non-singular during tracking and the intended trajectory of a full circle is obtained from camera pose estimation (see video[1]). In the second sequence, the accuracy improvement is obtained by manually finding an optimal λ for a rendered Itokawa sequence. A single gray image and depth image of the sequence are illustrated in Figure 5.3. The depth maps are generated using ray tracing and available asteroid model. The estimated trajectories with and without incorporated depth maps are compared with the ground truth trajectory in Figure 5.4. The figure shows how the camera trajectory is improved when using depth data. Non-Lambertian surface properties of the Itokawa sequence produces error in purely intensity based minimization. The same result is illustrated in a video with[2] and without[3] depth residual minimization.

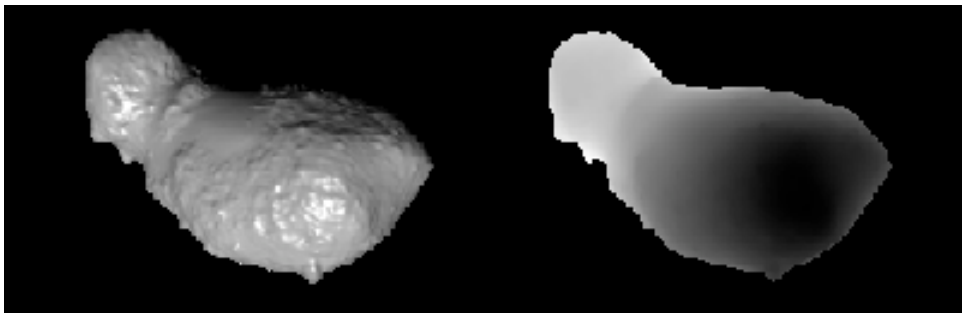


Figure 5.3: *Itokawa and the corresponding depth map. True Itokawa BRDF is the same as the Hapke model. Specularity will cause problems for intensity based estimation. The rendered picture has a general specular surface for simulating the problem.*

§ 5.3 Results on PRoVisG MARS 3D Challenge

The cost function was also experimented with a real stereo sequence provided by the PRoVisG MARS 3D Challenge [29]. In the task setting, the aim is to automatically determine the 3D trajectory and the reconstruction from a sequence of stereo images. The tasks are divided into dense matching, camera pose estimation and 3D reconstruction. The given sequence

[1] http://youtu.be/VVJkMpFliJw
[2] http://youtu.be/JWqu97sp1tM
[3] http://youtu.be/hIgfGhbusLY

consists of 35 stereo images at 1280×1024 resolution, which have been captured using a moving Mars rover. The cameras have a 64° FOV and the baseline of the stereo camera is 10 centimeters. The estimated length of the rover trajectory is 7.8 meters. The frame rate of the sequence is relatively low compared to the vehicle motion and the images do not contain a large quantity of texture detail as the sand covers the most part of the views. However 3–10 small rocks are visible in the view in all of the frames. There are no major lighting effects in the sequence. An example intensity image and a depth image of the right stereo view along with the corresponding residuals is illustrated in Figure 5.7.

5.3.1 Depth map generation

Depth maps were generated by converting SGBM disparity maps into depth format (Sections 3.2.1 and 2.2). The images are low-pass filtered and downsampled into 320×240 resolution using a multi-resolution pyramid because pixel noise has to be filtered out and the minimization of the proposed cost function works more efficiently with smoother gradients. The baseline of the stereo used is small and thus the matching was done using a discrete disparity range of $[0, 32]$. The disparity values were refined into sub-pixel accuracy in post-processing. Finally the obtained disparity map are converted into a depth map to evaluate the cost function.

Local stereo matching methods often produce false discontinuities (Figure 4.10), which are a potential problem for depth minimization as the greatest depth gradients have the greatest influence in the final pose estimate. This is why using a global matcher, such as SGBM, is important because smooth maps are produced by penalizing discontinuities. SGBM assigns discontinuities implicitly whenever image matching starts to fail. When the discontinuities are set locally in the wrong place, small deviations may occur which are filtered out as a post-process by using 5×5 median filter.

5.3.2 Pose estimation

The proposed cost function is minimized in order to estimate the pose parameters. In a typical application case the frame rate is high enough for using $\widehat{\mathbf{T}} = \mathbf{I}$ as the initial guess. For the given sequence the frame rate is low and an initial guess must be obtained by other means. $\widehat{\mathbf{T}}$ was generated by first matching temporally a set of 2D points and then minimizing the 2D distance between the warped points and the fixed target points $\mathcal{P}_{\text{sift}}$ in the current image. In this experiment, the 2D points were extracted and matched using SIFT. Initial pose parameters were estimated by minimizing the residual

$$\mathbf{e_G} = \rho(\mathcal{P}_{\text{sift}} - w(\mathcal{P}; \mathbf{T}(\mathbf{x})\widehat{\mathbf{T}})), \tag{5.8}$$

where ρ produces weighted distances and neglects statistically large displacements. Simple M-estimator $\rho(x) = \|x\|$ if $\|x\| < \tau$ and otherwise 0, was used in the experiment where τ is a fixed threshold. After minimization, the initial pose will have only small bias which can be corrected by minimizing the proposed cost function (eq. 5.1).

Figure 5.7 illustrates the intensity and the depth residuals after convergence. Real-time performance is reached by using the histogram based pixel selection. Figure 2.5 shows how the Tukey window based M-estimator rejects $20 - 40\%$ of all selected pixels. Pixel selection was set to 50% which produces 38400 points in 320×240 resolution.

5.3.3 Results

The algorithm was executed on Samsung R530 laptop with dual-core Intel i3 CPU (2.13GHz) of which only one core is used. The implementation was written in C++ and relies on Fortran based LAPACK and EXPOKIT routines for linear algebra and matrix exponentials. Figure 5.8 illustrates the optimization delay in milliseconds per frame with and without the depth component. By incorporating the depth component, roughly 50% more delay is added into computation. As a further optimization step, the algorithm could be run in parallel with both cores for halving the delays. It is unfortunate that the competition sequence has low frame rate and SIFT extraction and matching is required. The computational requirement of using SIFT was not evaluated because it was considered redundant phase in a real application where sufficient camera frame rate can be used. Three iterations with the initial cost function were required. Then three multi-resolution layers were used for convergence with the proposed cost function. In these sequences, the M-estimator rejects $20 - 30\%$ of the points by setting zero weights (Figure 5.8). The matrix sizes of the linear equation were kept the same, which is why the rejection does not show in Figure 5.8. In a real-time application FPGA hardware can be used for computing disparity maps [3]. The other additional phases such as pixel selection and 3D point extraction are $O(n)$ passes for the raw images which are very fast. The quality improvement when incorporating the depth component is difficult to evaluate as the ground truth trajectory for the PRoVisG MARS 3D Challenge has not been published. However by observing Figure 5.6 the camera trajectory and the 3D reconstruction are qualitatively good.

§ 5.4 Analysis and limitations

The results show that the drift can be reduced by incorporating depth map alignment into into pose the pose estimation. With simulated sequences, depth noise does not exist, and therefore an improvement is easy to obtain.

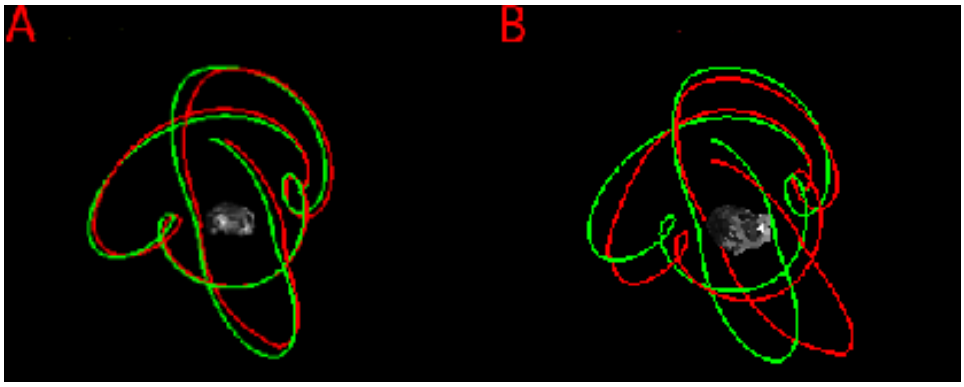


Figure 5.4: *Simulated Itokawa asteroid sequence. A) A complex camera trajectory is shown to be more accurate when also matching the depth maps. The green curve represents the ground truth camera trajectory and the red curve is the estimated one. B. trajectory of purely intensity-based minimization for comparison.*

The PRoVisG sequence shows that estimation works but no ground truth trajectory is available for measuring drift. A multi-resolution semi-global block matching was used to produce the initial disparity maps which were converted into depth maps. SGBM implements a smoothness constraint which is somewhat compatible with the sequence. Adjusting λ can be tedious when testing with different sequences. Without an automatic parameter optimization tool, cost function balancing may prevent wider use of this approach.

Two improvement ideas to this approach are a) using normal distance instead of Euclidean distance (Section 2.5.4), b) divide all input points into ones whose cost is measured by photometrical metric and the ones whose cost are measured by normal distance. The first improvement is obvious since the normal distance based metric has been proven beneficial over Euclidean distance. One example of its performance is the KinectFusion system [53]. The second improvement is based on the insight that one point may not have strong gradient in both image and depth, and on the other hand, full depth maps can be used with ICP. One interesting combination approach is to measure photometrical cost terms for image regions with strong gradient, and measure normal distance with the rest of the points.

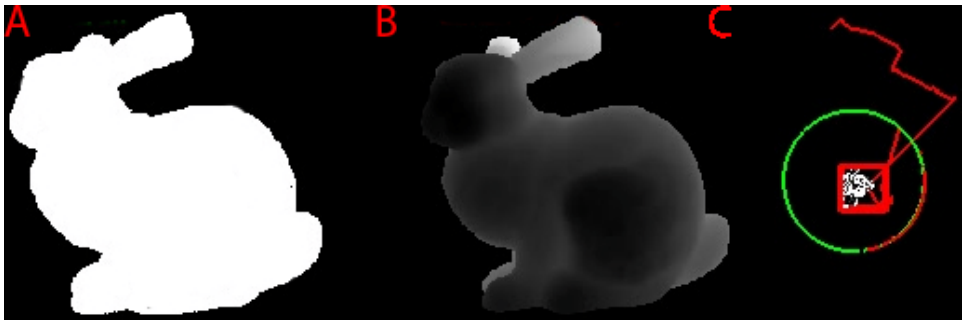


Figure 5.5: *Orbiting sequence around Stanford bunny. A. image from the sequence, B. the corresponding depth map, C. the estimated camera trajectory (red) along with the ground truth (green). Insufficient texturing produces a singular* **H** *which causes divergence during pose estimation. This problem is fixed by incorporating depth maps.*

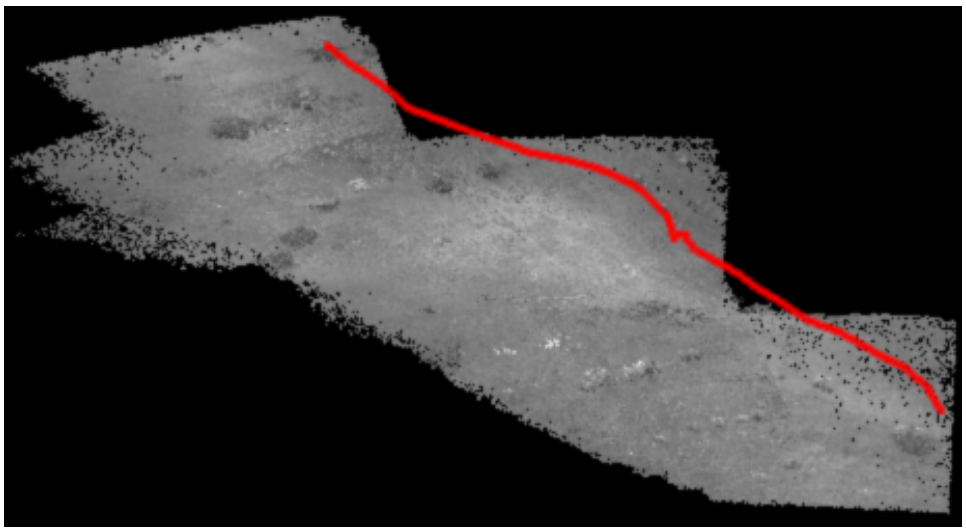


Figure 5.6: *PRoVisG MARS 3D Challenge sequence. The resulting camera trajectory and 3d reconstruction illustrated. The reconstruction shows the part of ground which is visible during the sequence.*

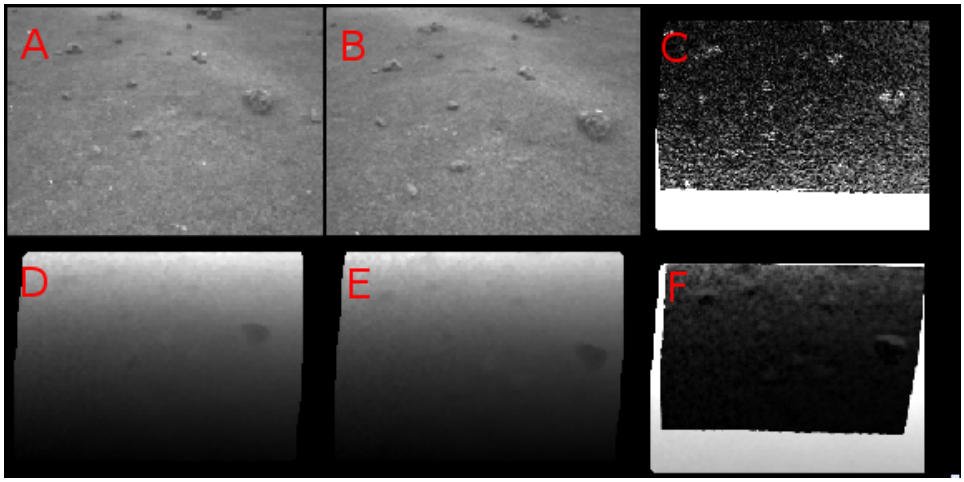
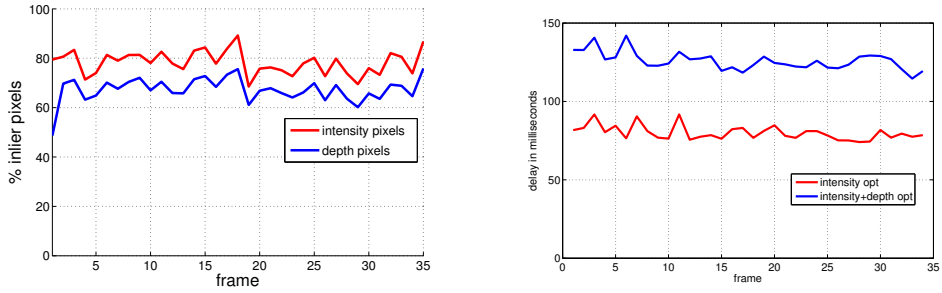


Figure 5.7: *PRoVisG MARS 3D Challenge sequence. The minimization of the proposed cost function is visualized. A) Image* 1 *B) Image* 2 *C) The intensity residual after minimization D) Depth map* 1 *E) Depth map* 2 *F) The depth residual after minimization. The remaining error is due to interpolation/filtering inaccuracy, occlusions and depth noise.*



(a) (b)

Figure 5.8: *PRoVisG MARS 3D Challenge sequence [29]. a) M-estimator functionality illustrated for the rover sequence. Total amount of selected pixels (inliers+outliers) is set to* 50% *which produces 38400 points in* 320×240 *resolution. b) Optimization delay per frame in milliseconds with (blue) and without (red) depth component. Incorporating the depth component increases computational requirement by* 50%*. For a full real-time system disparity maps must be generated by external hardware/GPU.*

- CHAPTER 6 -

REAL-TIME RGB-D TRACKING FOR A LOW-END GPU

The Microsoft Kinect sensor has sufficient RGB-D quality and frame rate for RGB-D tracking. One of the contributions in this work is a real-time GPU implementation for Microsoft Kinect which has been developed from scratch and can be easily accommodated into applications. The library depends on CUDA, which essentially performs all relevant computations in parallel on GPU threads. The library has been tested using various input RGB-D sequences. Our method was implemented in Ubuntu Linux environment using open software tools, Microsoft Kinect and a commodity PC laptop hardware on which the method runs real-time. The cost minimization requires special attention when aiming at an efficient real-time implementation. Because the computational phases benefit from parallel computing, we implement the full algorithm (6.4) on a commodity GPU. The minimization algorithm can be used for both dense tracking and keyframe tracking and it scales into multiple threads/cores. Only Cholesky inversion and matrix exponential computation are executed on a single GPU thread. Steinbrücker *et al.* have recently studied a similar minimization and suggested a real-time GPU implementation [70]. In the following GPU implementation also a M-estimator is supported which can increase robustness with negligible computational cost and minimization is performed efficiently using inverse compositional approach. The implementation accuracy and computational requirements are compared with Kinfu, which is an open-source implementation of KinectFusion [62].

The depth map registration proposed in Chapter 5 increases computational requirements linearly with the number of points selected for minimization. The least expensive way to implement it is to measure depth error with the same points which are already selected based on image gradient magnitude. Still Jacobian re-computation will be required in every iteration. In environments where texturing can be increased, depth registration does not bring benefits. The current implementation is designed to

be used in television broadcasting studios where real-time performance is crucial therefore the cost function does not minimize depth residual. This design choice simply favors computational efficiency in a textured environment. Also, when considering applications, where drift is not an option, bundle adjustment or other global techniques are used to produce a consistent model. It is important that the online tracker is merely locally precise to be able to switch between the keyframes.

§ 6.1 TRACKING MODES

RGB-D pose tracking can be executed in three different modes: incremental, keyframe and SLAM. These modes have been implemented and this section describes each mode. All modes minimize the photometric cost (eq. 3.13) using the inverse compositional image alignment.

6.1.1 Incremental dense tracking

In incremental tracking, a keyframe model is not present and previous RGB-D measurements are used as the motion reference frame. The benefit is that reference will typically be a very similar image to the current one, and photometrical minimization will produce good, smooth results. Incremental tracking has the reference update frequency as free parameter. Automatic reference update is possible by analyzing Median-Absolute-Deviation (MAD) during M-estimation. When the modeling error increases over a threshold, reference update can be signaled. However, signaling reference updates from a GPU side is a bad idea, since it introduces conditional statements and reading back variables into main memory which both are slow on GPU. Therefore a fixed update rate is used. High update frequency implies that Lambertian assumption holds better and tracking will be smooth and precise. The scene appearance can change quickly within few frames and therefore a common update frequency is in practise between $1-3$ frames. The downside with incremental tracking is that time-evolving drift will slowly cumulate in a long-term use. Reference update at lower frequency implies that the same geometry will be fixed for a longer period of time, with the cost of worse image similarity. In this case, computational requirements will naturally be lower on average, but the motion must be assumed to be limited.

6.1.2 Keyframe-based dense tracking

The incremental tracking suffers from time-evolving drift, which is not acceptable in studio use, because it causes virtual items to move away from their correct pose. A fixed 3D model is required to avoid drift (Figure 8.1).

Algorithm 6.1 Incremental dense pose tracking algorithm

Require: $\{\mathcal{P}^*, \mathbf{c}^*\}$ ⇐Select the best points from 1st RGB-D.
Input: $\mathbf{T}_{cur} = \mathbf{T}_{ref} = \mathbf{I}$
Output: Trajectory $\{\mathbf{I}, \mathbf{T}^1_{cur}, \ldots, \mathbf{T}^n_{cur}\}$.
1: **for** each RGB-D measurement **do**
2: $\widehat{\mathbf{T}} \Leftarrow$ Minimize$(\mathbf{I}, \mathcal{P}^*, \mathbf{c}^*, \mathcal{I}_{cur})$
3: $\mathbf{T}_{cur} \Leftarrow \widehat{\mathbf{T}}\mathbf{T}_{ref}$
4: **if** reference update signaled **then**
5: $\{\mathcal{P}^*, \mathbf{c}^*\} \Leftarrow$ Select the best points
6: Precompute Jacobian
7: $\mathbf{T}_{ref} = \mathbf{T}_{cur}$
8: **end if**
9: **end for**

Algorithm 6.2 Dense keyframe tracking algorithm.

Require: Keyframe database available
Input: $\mathbf{T}_{cur} = \mathbf{I}$
Output: Trajectory $\{\mathbf{I}, \mathbf{T}^1_{cur}, \ldots, \mathbf{T}^n_{cur}\}$.
1: **for** each RGB-D measurement **do**
2: $\{\mathcal{P}^*, \mathbf{c}^*, \mathbf{T}_{key}\} \Leftarrow$FindKeyframe$(\mathbf{T}_{cur})$
3: $\widehat{\mathbf{T}} \Leftarrow$Minimize$(\mathbf{T}_{cur}\mathbf{T}^{-1}_{key}, \mathcal{P}^*, \mathbf{c}^*, \mathcal{I}_{cur})$
4: $\mathbf{T}_{cur} \Leftarrow \widehat{\mathbf{T}}\mathbf{T}_{key}$
5: **end for**

In keyframe tracking, a set of fixed RGB-D keyframes exist, which can be used as a global reference for motion. The benefit is that time-evolving drift does not exist. The downside is that the content of the keyframes can easily change due to lighting variations and geometrical displacements in a studio setting. Thus, the keyframe model must be updated whenever lighting conditions or environment configuration changes. However, when keyframes are measured prior to online tracking, this problem can be avoided. Also work has been done to increases tolerance to varying lighting conditions (Section 3.5).

A keyframe model can be generated with various online/offline techniques prior to broadcasting [86, 53]. The online tracking is initialized at the first keyframe whose $\widehat{\mathbf{T}} = \mathbf{I}$. The cost function (eq.3.13) is then minimized in each frame to obtain 3D camera pose. The reference keyframe is switched when the current pose becomes closer to another keyframe. The keyframe tracking is sketched in Algorithm 6.2. Online tracking is driftless and very fast due to keyframe pre-computations and the utilization of GPU.

A similarity metric is required to find the nearest key pose $\mathbf{T}_k$. The challenge is to unify rotation and translation differences, because they are

expressed in different units. First, we define the relative transformation

$$\Delta \mathbf{T}_k = \mathbf{T}_{cur}\mathbf{T}_k^{-1} \Rightarrow (\theta_k, \mathbf{v}_k, \mathbf{d}_k), \tag{6.1}$$

where $\mathbf{T}_{cur}$ is the current camera pose. The relative rotation is decomposed into angle-axis representation $(\theta, \mathbf{v})$ based on the *Euler's rotation theorem*. The translation between the poses is expressed as the vector $\mathbf{d}$ between the origins. We define a potential keyframe index set as

$$\mathbf{\Omega} = \{\ |\theta_k| < \theta_{max}, ||\mathbf{d}_k|| < d_{max}\ ||\ k \in [1, n]\ \}, \tag{6.2}$$

where n is the number of keyframes in the database. Thus $\mathbf{\Omega}$ contains a subset of keyframe indices whose angle and distance are below user defined thresholds θ_{max} and d_{max}. This pre-selection prunes out distant poses efficiently. Thresholds are easy to set based on keyframe density in a 3D volume. The best keyframe is chosen by transforming the view frustum, represented by a set of 3D points, from the keyframe into the current frame and observing the 2D point discrepancy. This unifies rotation and translation errors into a single metric

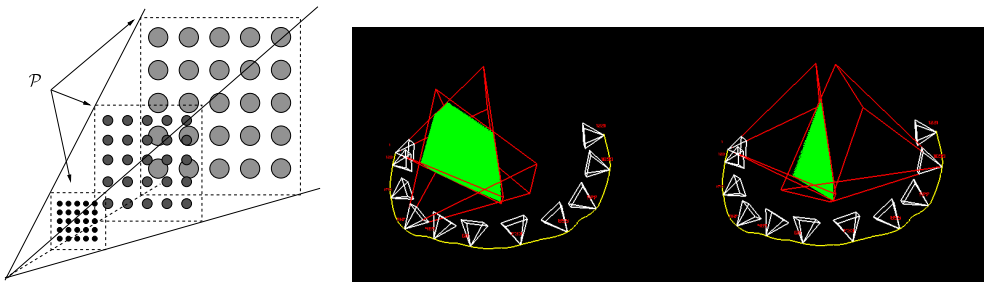
$$s = \underset{k\ \in\ \mathbf{\Omega}}{\operatorname{argmin}} ||w(\mathcal{P}; \Delta\mathbf{T}_k) - w(\mathcal{P}; \mathbf{I})||, \tag{6.3}$$

where $\mathcal{K}_s$ is the nearest keyframe. The idea is illustrated in Figure 6.2. In contrast to frustum intersection, this metric varies significantly also when the camera is rotating around z-axis. The test point set $\mathcal{P}$ is a sparse representation of the view frustum (Figure 6.1). In particular, the frustum is approximated by three sparse 2D grids, each having uniformly sampled depth coordinates in the overall depth range of the RGB-D sensor. The 3D points at different layers are generated by discrete steps along the viewing rays.

6.1.3 SLAM mode

In the SLAM mode, the keyframe database is built concurrently while tracking the camera. The algorithm is listed in Algorithm 6.3. First, an initial keyframe is generated with identity pose using the first RGB-D measurement. Two search ranges $\{\theta^1_{\text{max}}, \mathbf{d}^1_{\text{max}}\} \leq \{\theta^2_{\text{max}}, \mathbf{d}^2_{\text{max}}\}$ are defined, where the second one is required in case the nearest keyframe is out of domain and the map should be extended. When the pose estimate is no longer in the keyframe database domain, a new keyframe is generated. The domain can be checked using the pose distance criteria developed in section 6.1.2. In the current implementation, keyframes can be added until GPU memory runs out. SLAM mode is useful especially when the camera is known to re-enter scenes. For example, when building a 3D model of an appartment, it is sometimes necessary to add missing keyframes to remove holes in the model (see video[1]). Figure 2.8 showcases a loop-closure. The

[1] `Video:http://youtu.be/aFrVROLja38`



(a) (b)

Figure 6.1: *a) The 3D test point set $\mathcal{P}$ used to compare camera poses is illustrated. $\mathcal{P}$ approximates the view frustum by three 2D point grids. The projection error of $\mathcal{P}$ between the current view and each keyframe view is compared to find the nearest keyframe. b) As comparison, frustum intersection volume based metric. The metric varies little when rotating around z-axis and can behave un-predictable way when the camera angles are significantly different.*

downside of SLAM mode is that keyframes must have very small baseline to the current measurement, otherwise more bias will be generated compared to incremental tracking.

With a sequences that do not contain loop closures, the incremental tracking is a better option, because the Lambertian assumption holds well and large occlusions do not exist. In another example scene, the environment is scanned by hand in almost a direct 360° turn without a loop closure. The incremental tracking produces more accurate result (Fig. 7.11). More details are provided in Chapter 7.

§ 6.2 FEATURES

6.2.1 Embedding distortions in warping function

As discussed in Section 2.1.5, the lens distortions can be corrected by re-sampling the input images using the distortion model. It is better to model the distortions in the warping function, because image re-sampling degrades original image quality and part of the data is even lost. Image re-sampling requires interpolation, which will not produce exact image intensities between the samples. Also the number of distortion operations for a set of points will be smaller than full image resolution even though warping must be be done many times per frame.

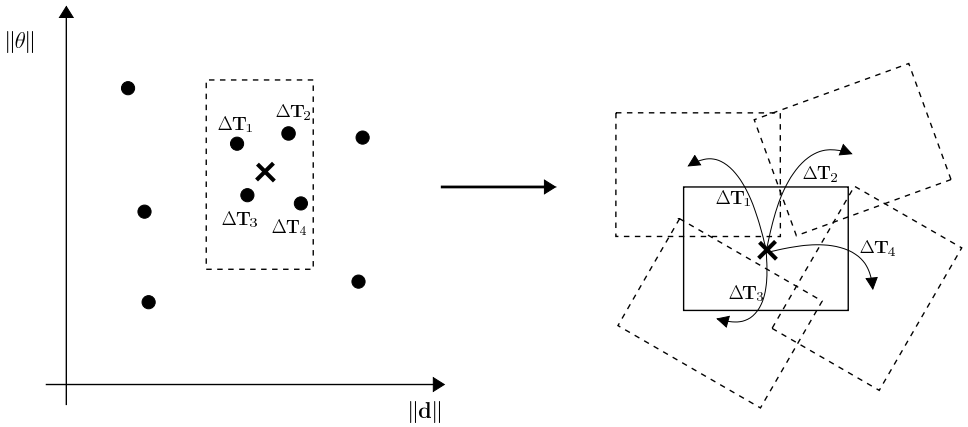


Figure 6.2: *The phases of finding the nearest pose. a) Pruning out distant poses, b) Finding the pose difference which minimizes optical flow of a 3D test point set.*

6.2.2 Tolerating dynamic foreground

Comport *et al.* proposes intensity-based M-estimation in a stereo setting [14], because disparity map computation is expensive for each frame. Due to recent development with RGB-D sensors, it is possible to measure both intensity and depth maps in real-time.

Now also depth correlation weights can be computed from the depth residual

$$\mathbf{e}_z = \mathcal{Z}\left(w(\mathcal{P};\widehat{\mathbf{T}})\right) - \mathbf{e}_3^T\widehat{\mathbf{T}}\begin{bmatrix}\mathcal{P}\\1\end{bmatrix}, \tag{6.4}$$

where $\mathcal{Z} : \mathbb{R}^2 \Rightarrow \mathbb{R}$ is the depth map function of the current RGB image and $\mathbf{e}_3^T = (0,0,1,0)$ is used to obtain the current depth value. When the standard deviation of depth measurements is τ, the warped points whose depth differs more than τ from the current depth map value can be interpreted as foreground objects. The depth weights become

$$\mathrm{w}_\mathrm{k}^\mathrm{z} = max\left(1 - \mathbf{e}_z^2(k)/\tau^2, 0\right)^2. \tag{6.5}$$

The weighting matrix $\mathbf{W}$ is now a product of intensity and depth based weighting $\mathrm{diag}(\mathbf{W})_k = \mathrm{w_k}^c\mathrm{w_k}^z$.

§ 6.3 SCALABLE GPU TRACKING

The pose estimation described in Section 3.1.4 is implemented on a low-end GPU with 48 CUDA cores. This implementation is important, because it is scalable. Scalability means that performance increases simply by adding new CUDA cores and the code itself does not require any modifications.

Algorithm 6.3 Keyframe SLAM algorithm.

Require: Keyframe database contains 1st RGB-D measurement
Input: $\mathbf{T}_{cur} = \mathbf{I}$, Search ranges $\{\theta^1_{\max}, \mathbf{d}^1_{\max}\} \leq \{\theta^2_{\max}, \mathbf{d}^2_{\max}\}$
Output: Trajectory $\{\mathbf{I}, \mathbf{T}^1_{cur}, \ldots, \mathbf{T}^n_{cur}\}$, Keyframe model $\mathbf{K}$.

for each RGB-D measurement **do**
  newKeyFlag = false
  $\mathbf{F} = \{\mathbf{P}^*, \mathbf{c}^*, \mathbf{T}_{key}\} \Leftarrow$FindKeyframe$(\mathbf{T}_{cur}, \theta^1_{\max}, \mathbf{d}^1_{\max})$
  if $\mathbf{F} = \emptyset$ **then**
    newKeyFlag = true
    $\{\mathbf{P}^*, \mathbf{c}^*, \mathbf{T}_{key}\} \Leftarrow$FindKeyframe$(\mathbf{T}_{cur}, \theta^2_{\max}, \mathbf{d}^2_{\max})$
  end if
  $\widehat{\mathbf{T}} \Leftarrow$Minimize$(\mathbf{T}_{cur}\mathbf{T}^{-1}_{key}, \mathbf{P}^*, \mathbf{c}^*, \mathbf{I}_{cur})$
  $\mathbf{T}_{cur} \Leftarrow \widehat{\mathbf{T}}\mathbf{T}_{key}$
  if newKeyFlag = true **then**
    $\{\mathbf{P}, \mathbf{c}\} \Leftarrow$ Select a subset of points with largest gradients
    Precompute Jacobian
    $\mathbf{K} = \{\mathbf{K}, \{\mathbf{P}, \mathbf{c}, \mathbf{T}_{cur}\}\}$
  end if
end for

CPU implementations have scalability problem, because the number of CPU cores increases very slowly compared to GPUs. Whereas the high-end CPUs may have 4-8 cores, the most powerful GPUs, such as the NVIDIA Tesla K10, have 3072 CUDA cores. This GPU implementation has been developed using NVS4200m GPU which is low-end GPU, but achieves real-time 30Hz frame rate. Considering that photometrical minimization benefits from texture details, HD resolution at 1920×1080 is interesting, because improved image gradient accuracy directly increases pose estimation precision. According to the Kinect 2.0 sensor specifications, it will support HD resolution and directly benefits from this scalable GPU implementation.

6.3.1 Warping

The warping function is applied in parallel to the selected $\mathbf{P}_k \in \mathcal{P}$ in the keyframe. The cost function (eq. 3.13) is evaluated using bilinear interpolation. Additionally the points are transformed into IR view for evaluating the nearest depth value (eq. 6.4). By evaluating the depth values directly in the IR view, re-sampling errors are avoided. The depths are used to generate weighting $\mathbf{W}_z$. The warping is illustrated in Figure 6.3.

6.3.2 M-estimator

Tukey based weighting using the median absolute deviation of the error distribution, easily becomes a bottleneck on GPU. Therefore, it is proposed that the histogram technique (eq.6.7) is used to find an approxi-

Algorithm 6.4 Minimization on GPU.

Input: $\mathcal{I}_{L=\{1,2,3\}}$ with 320×240, 160×120, and 80×60 sizes. $\mathcal{Z}$ in 320×240. Iteration counts $\{n_1, n_2, n_3\}$. $\widehat{\mathbf{T}} = \mathbf{T}_0$.
Output: Relative pose $\widehat{\mathbf{T}}$.

```
for all L = {3, 2, 1} do
    for all j = {1 ... n_L} do
        Compute residual e and W_z (6.3.1)
        Determine M-estimator weights W_c (6.2.2)
        W ⇐ W_c * W_z
        J ⇐ √W J, e ⇐ √W e
        Reduce linear system (6.3.3)
        Solving linear system for x̂ (6.3.3)
        T̂ ⇐ T̂ e^A(x̂) (6.3.4)
    end for
end for
```

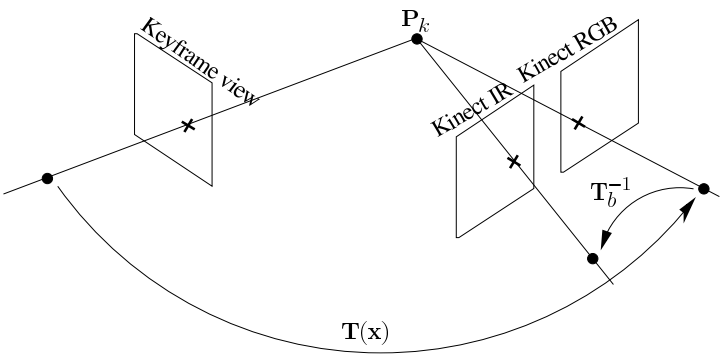


Figure 6.3: *The warping of points from keyframe into Microsoft Kinect RGB and IR image coordinate systems. The additional depth lookup from an IR image is required by Equations 5.3 and 6.4.*

mate median. The distribution of the residual is represented using a 64-bin histogram and n is set to half the residual length. The CUDA SDK `histogram64` routine fits this purpose, because it is fast enough to be executed at every iteration and is specifically designed for NVIDIA video cards [57]. By experiment, 64 bins seems to provide sufficient adaptation to different error distribution profiles.

6.3.3 Linear system reduction

The reduction of the linear system means computing $\mathbf{J}^T\mathbf{J}$ and $\mathbf{J}^T\mathbf{e}$. This phase compresses the linear equations from n-dimensional to six-dimensional space. Reduction is required for each iteration and, therefore it must be implemented efficiently. Matrix multiplication is often computed

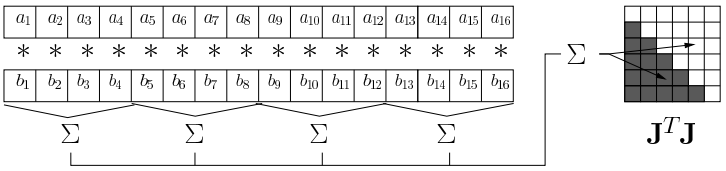


Figure 6.4: *Efficient reduction on GPU. Dot products are divided into sub-ranges which are reduced in parallel.* $\mathbf{J}^T\mathbf{J}$ *is symmetric* 6×6 *matrix and elements can be mirrored with respect to the diagonal. The Jacobian columns are transposed into rows to improve memory caching.*

in parallel by dividing k dot products into separate threads. In this case, we have only 36 dot products, but our video card can manage 1024 threads in parallel. To gain maximal efficiency, we parallelize the computation in the dot product direction instead. The process is illustrated in Figure 6.4. Each dot product has n elements, where n is usually 8192 or more. Dot products are reduced by dividing them into sub-blocks and summing them efficiently in parallel. The total sum is finally cumulated from the sub-sums. $\mathbf{J}^T\mathbf{J}$ is a positive semidefinite matrix which is also symmetric. This means it is sufficient to compute only the upper triangle of the values and mirror the results into the lower triangle. This property reduces the number of dot products from 36 to 21. $\mathbf{J}^T\mathbf{e}$ requires six dot products. The inversion can be efficiently calculated using Cholesky decomposition due to positive definite property (Section 2.3.6). The explicit inverse of $\mathbf{J}^T\mathbf{J}$ can also be avoided by using the conjugate gradient method (CGM) with six iterations. Both approaches are fast due to the small matrix size. The execution times were compared in practical use and CGM required on average $0.0119ms$ whereas Cholesky $0.0251ms$ on a single GPU thread. These numbers are sequence independent because the number of selected points remains fixed. CGM was chosen, because it is slightly faster.

The parameter vector $\mathbf{x}$ contains both rotational units (radians) and translational units, whose units are fixed in the calibration procedure. It should be noted that a big difference in parameter magnitudes may introduce numerical instability. For example, if the translation units are expressed in millimeters, their scale can be over $1000\times$ radian unit scale. To prevent any potential numerical problems, it makes sense to change the input point scale before computing the Jacobian $\mathbf{J}$ and change it back after final transformation increment has been estimated. In our implementation, we pre-scale the points by $\frac{1}{1000.0}$ and post-scale the translation increment by 1000.0. This trick produces parameter vectors whose maximum magnitudes in our test sequences are uniformly order of ≈ 0.01.

6.3.4 Evaluating matrix exponential

Matrix exponentials can be computed in closed form using the translation extended *Rodriquez formula* [40]. Unfortunately is not numerically stable in the most important small angle case. Expokit, however, provides various numerical approaches which are guaranteed to produce smooth mapping [67]. By comparing them with the MATLAB default implementation using Padé approximation, it seems that complex matrix exponential (`zgpadm`) is the most accurate. Thus, we choose to evaluate $\mathbf{T}(\mathbf{x}) = e^{\mathbf{A}(\mathbf{x})}$ by the matrix exponential of a general complex matrix in full, using the irreducible rational Padé approximation combined with scaling-and-squaring. The exponential is computed in a single GPU thread due to sequential nature of the operation. One evaluation takes $0.2613ms$ on average. The evaluation is rather slow on GPU, because a single thread must manouver many sequential operations. Despite using the Lie generators to produce $\mathbb{SE}3$ group transformations, normalization is still required (Section 2.5.2).

6.3.5 Selecting points on GPU

The points used in the motion estimation can be freely selected from a reference RGB-D keyframe. The points which do not contribute to the residual vector through linearization are not useful. A subset of points should be selected, which are associated with the greatest absolute values of the Jacobian [45]. The selected points will then have strong geometric flow $\|\frac{\partial w}{\partial \mathbf{x}}\| > \theta_w$ and/or strong image gradient $\|\frac{\partial \mathcal{I}}{\partial \mathbf{p}}\| > \theta_g$. We use a simpler selection criteria and focus on the set of points $\mathcal{P}_s^*$ which only have strong image gradient. This method fits GPUs better, because it does not require sorting of the Jacobian elements.

$$\mathcal{P}_s^* = \{\ \mathbf{P}_k \mid \|\nabla \mathcal{I}^*(w(\mathbf{P}_k; \mathbf{I}))\| \ > \ \theta_g\ ,\ \mathbf{P}_k \in \mathcal{P}^*\}, \tag{6.6}$$

where θ_g is the threshold which selects p percent of all points. p depends on the computational capacity available and the amount of image edges in the application context.

Even though keyframe reference points can be selected in a pre-process, a fast implementation is useful when executing dense tracking incrementally. We adopt the histogram technique to find the best points efficiently [81]. We seek such histogram bin B_t for which

$$\Sigma_{k=B_t-1}^{255}\, \mathbf{h}(k) < n <= \Sigma_{k=B_t}^{255}\, \mathbf{h}(k), \tag{6.7}$$

where n is the number of points to be selected and

$$\begin{aligned} \mathcal{G} &= |\nabla_u \mathcal{I}^* \left(w\left(\mathcal{P}^*; \mathbf{I}\right)\right)| + |\nabla_v \mathcal{I}^* \left(w\left(\mathcal{P}^*; \mathbf{I}\right)\right)| \\ \mathbf{h} &= \text{histogram}(\mathcal{G}/2.0) \end{aligned}$$

Computing histograms on GPU is tricky, because simultaneous writing into the histogram bins is not possible. CUDA SDK presents a method [57]

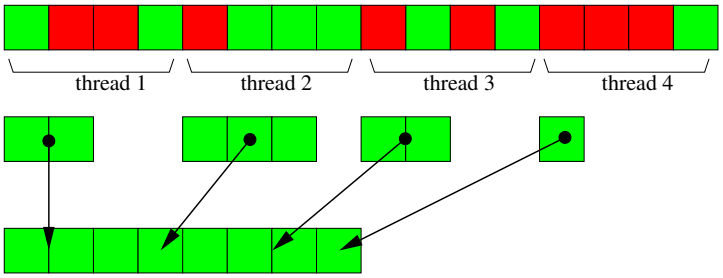


Figure 6.5: *Index buffer packing illustrated. Each thread selects points from a sub-range. The sub-counts are shared between the threads and the output buffer written.*

(`histogram256`) which reduces the problem by creating partial histograms, which are finally summed together. The data is processed in *warp* sized blocks, where warp determines the amount of concurrent threads. Write collisions can occur inside a warp, but they are avoided by *tagging*. In tagging, a memory slot is tested after a write and if the result is not what is expected, the operation is done again. This guarantees that at least one thread in a warp will succeed. The gradient value range is pre-scaled into $[0, 255]$ which fits `histogram256` method exactly. Shams *et al.* developed a similar implementation independently [64]. The downside of the method is that uncoalesced memory writes to global GPU memory are relatively slow. The final n indices are collected into a packed array. Packing is important because it allows eliminating all non-interesting points from further processing in the pipeline. Packing is implemented on a GPU by assigning each thread a slice of the original index range to compress (Figure 6.5). Each thread stores the interesting points into a temporary packed buffer with the count. The counts must be collected from all threads to determine the final output range of each thread. This allows parallel point selection which scales into multiple threads. Due to the discretization into bins, n pixels may not match the bin boundary automatically. n must be divisible by $T_{\max}$ to match the maximum amount of threads on a GPU. On our GPU $T_{\max} = 1024$. This is why it is useful to classify points into

$$\begin{aligned} \mathcal{P}_{semi} &= \{\, |\nabla c_k| = B_t \mid \mathbf{P}_k \in \mathcal{P} \,\} \\ \mathcal{P}_{good} &= \{\, |\nabla c_k| > B_t \mid \mathbf{P}_k \in \mathcal{P} \,\}, \end{aligned}$$

where $\mathcal{P}_{good}$ are all selected and the remaining points are chosen from $\mathcal{P}_{semi}$ to obtain exactly n selected points. B_t is determined by Equation 6.7.

6.3.6 Vertex attributes at different stages

The vertices extracted from RGB-D images originally are associated with 21 float attributes which are used in pre-processing phases. After points have been selected, only 9 float attributes are useful during minimization. Normals are estimated to take into account normal distance based cost evaluation (see Section 2.5.4). With fully image intensity based residuals the compression could be taken further by removing also vertex normals. The table of vertex attributes are illustrated in Table 6.1. Assuming that 320×240 points are pre-processed using point selection procedure and only 8192 points are selected and compressed, memory consumption per one point cloud is $6.15MB \Rightarrow 0.28MB$. This is very efficient compression since only 4.6% of original memory is required.

Table 6.1: *Vertex attributes. During minimization only fraction of the attributes are required. GPU memory is optimized as soon as preprocessing phase is finished.*

Uncompressed vertex attribute		Compressed vertex attribute	
X-coordinate	x	X-coordinate	x
Y-coordinate	y	Y-coordinate	y
Z-coordinate	z	Z-coordinate	z
Normal X-direction	n_x	Normal X-coordinate	n_x
Normal Y-direction	n_y	Normal Y-coordinate	n_y
Normal Z-direction	n_z	Normal Z-coordinate	n_z
Image U-coordinate	u	Image intensity (layer 1)	c_1
Image V-coordinate	v	Image intensity (layer 2)	c_2
Color R	r	Image intensity (layer 3)	c_3
Color G	g		
Color B	b		
Image gradient-U (layer 1)	$\Delta_u \mathcal{I}_1$		
Image gradient-V (layer 1)	$\Delta_v \mathcal{I}_1$		
Image gradient magnitude (layer 1)	$\|\Delta_u \mathcal{I}_1\| + \|\Delta_v \mathcal{I}_1\|$		
Image intensity (layer 1)	c_1		
Image gradient-U (layer 2)	$\Delta_u \mathcal{I}_2$		
Image gradient-V (layer 2)	$\Delta_v \mathcal{I}_2$		
Image intensity (layer 2)	c_2		
Image gradient-U (layer 3)	$\Delta_u \mathcal{I}_3$		
Image gradient-V (layer 3)	$\Delta_v \mathcal{I}_3$		
Image intensity (layer 3)	c_3		

6.3.7 Preprocessing RGB images

The preprocessor converts 640×480 Microsoft Kinect Bayer images into a pyramid of 320×240, 160×120, and 80×60. A 5×5 Gaussian filter is used with downsampling of the high resolution images. Downsampling is almost lossless, because the Bayer images are redundant. 2×2 block averaging is used to produce the rest of the layers. The lower resolution coordinates are obtained by $x_L = \frac{1}{2^L} x + \frac{1}{2^{L+1}} - \frac{1}{2}$, where L is the amount of

layers in-between. The RGB pre-processing steps are per-pixel operations which are executed in separate threads on the GPU.

6.3.8 Point cloud from raw disparity map

The 3D points are generated in parallel by a baseline transform

$$\mathbf{P}_k = \mathbf{T}_b z(d_k) \mathbf{K}_{IR}^{-1} \begin{bmatrix} \mathbf{p}_k \\ 1 \end{bmatrix}, \tag{6.8}$$

where $\mathbf{p}_k = (u_k, v_k)^T$ are the pixel coordinates in the IR view, $\mathbf{K}$ is the intrinsic matrix of the IR view, and the 4×4 baseline matrix $\mathbf{T}_b$ maps points from the IR view into the RGB view. Each cloud $\mathcal{P}$ has storage for 320×240 points which are processed in parallel and stored in a linear array. The intensity vector $\mathbf{c}^*$ corresponding to the 3D points $\mathbf{P}_k \in \mathcal{P}$ is produced by bi-linear interpolation, because the points do not match with RGB image pixels.

6.3.9 Online visualization issues

When designing a real-time implementation, visualization needs and performance optimizations are contradictory. On one hand, visualization is necessary for debugging purposes, but on the other hand, with CUDA it can become expensive. CUDA interop is a technology which allows locking OpenGL pixel and vertex buffers for CUDA use. The buffers can be manipulated by CUDA programs and finally rendered using standard OpenGL API. There is no need to copy any data from/to main memory. The buffer locking delay with CUDA interop can vary from negligible delay to tens of milliseconds depending on how it is done. If a locking call is done immediately after graphics rendering has started, huge delay can occur because the call will wait until the rendering has finished. Therefore CUDA manipulations and graphics rendering must be fully separated into different phases. A normal delay with the NVS4200m graphics card is few milliseconds in the worst case. If there are many buffers to be visualized, the runtime performance will be significantly degraded. Therefore it is necessary to consider which data is the most valuable for debugging purposes. The current GPU implementation allows setting a flag per each buffer to determine whether it is allowed to be rendered using CUDA interop or not. The buffers which are withheld from rendering, are simply allocated from GPU memory without a CUDA interop context. This small optimization saves several milliseconds in the GPU implementation but still allows efficient online visualization when necessary.

§ 6.4 ACCURACY

The incremental dense tracking is sketched in Algorithm 6.1. In this section the dense tracking accuracy is evaluated using the RGB-D SLAM benchmark provided by Technical University of Münich [74]. Kinfu is the open source implementation of KinectFusion [62] (Sec. 3.8). The proposed dense RGB-D tracking accuracy is also compared with Kinfu (Figure 6.6). Kinfu experiments were executed on a workstation with the NVIDIA Quadro 2000 GPU with 1024MB RAM and 192 CUDA cores, because our development laptop could not provide real-time computations. In this comparison, a fixed 3D model is not used and time-evolving drift is present in both systems. Our real-time tracker is executed in incremental mode without depth fusion. The depth maps were also re-sampled back into disparity maps to fit our implementation.

Table 6.2: *Drift and delay compared to Kinfu with* $(3m)^3$ *and* $(8m)^3$ *voxel grids.*

Dataset	Incremental dense	Kinfu(3)	Kinfu(8)	Motion
`freiburg1/desk`	$2.60cm/s$	$8.40cm/s$	$3.97cm/s$	$41.3cm/s$
	$52.2ms$	$135ms$	$135ms$	
`freiburg2/desk`	$1.08cm/s$	$0.64cm/s$	$1.30cm/s$	$19.3cm/s$
	$35.5ms$	$135ms$	$135ms$	

Table 6.2 shows the comparison between our method and Kinfu numerically using two Freiburg sequences with known ground truth trajectory. Our method uses 320×240 resolution where as Kinfu uses 640×480 resolution. The dense tracking drifts $1.08cm/s$ with the slower `freiburg2/desk` sequence and $2.60cm/s$ with the faster `freiburg1/desk` sequence. Kinfu has smaller drift with small voxel volumes (such as $(3m)^3$), but seems to suffer from gross tracking failures with bigger volumes such as $(5m)^3$ and $(8m)^3$. The operation volume is limited because a 512^3 voxel grid becomes too coarse with increasing size and bigger grids not not fit into GPU memory. Also ICP breaks down easily when the scene contains mostly planar surface (e.g. floor) [2]. Thus the scene must contain sufficient geometrical variations. Drift was measured by dividing the input frames into subsegments of several seconds ($10s$ and $2s$ correspondingly) whose median error was measured against the ground truth. 1 second average error was computed from the median subsegment. The error values were computed from bigger windows to average out random perturbations and neglect Kinfu tracking failures, which occurred in all cases except on `freiburg2/desk` using $(3m)^3$ volume. Our dense tracking has generally smaller drift with

[2] Video: `http://youtu.be/tNz1p1sdTrE`

the faster `freiburg1/desk` sequence, but both Kinfu (with the most compatible grid) and our method lose tracking once during the sequence. In this case $(3m)^3$ grid does not contain constraining geometry and operates worse than a bigger volume. Re-localization issues are discussed later in this chapter. Kinfu's dependency on careful setting of volume size, and dependency on geometrical variations makes it unsuitable to be used in our application. In many applications, the scenes can easily be larger than $(3m)^3$ and sufficient geometrical variation is more difficult to guarantee than sufficient texturing. Our dense tracking operates without failures even when planar surfaces are present, because our cost function matches also texturing. Memory consumption can be low even in larger operating volumes, because the keyframe placement can be optimized based on the camera motion zones.

§ 6.5 RESULTS

The minimization algorithm is implemented on a low-end NVIDIA Nvs4200m GPU using CUDA. Our laptop GPU has 48 CUDA cores and it allows executing 1024 parallel threads on a single multi-streaming processor. Because our implementation divides the computational task into n threads, it is scalable and benefits from GPU hardware development. However, despite that the system can be executed with a fraction of GPU capacity compared to KinectFusion, the accuracy is not essentially different. In effect, the system operates at 30Hz and the computation takes 23ms which leaves 10ms for rendering the augmented graphics. The computation time of the processing phases is illustrated in Figure 6.7. The green bars represent the minimization phases which directly scale into n threads and therefore become faster with more powerful GPU. 8192 points are selected and the minimization uses three multi-resolution layers 80×60, 160×120 and 320×240. The corresponding iteration counts are 2, 3, and 10 for each level of the pyramid. These numbers are represent the current performance at the time of writing this text. The drift removal is possible by tracking with respect to a static keyframe model, which can has been globally optimized.

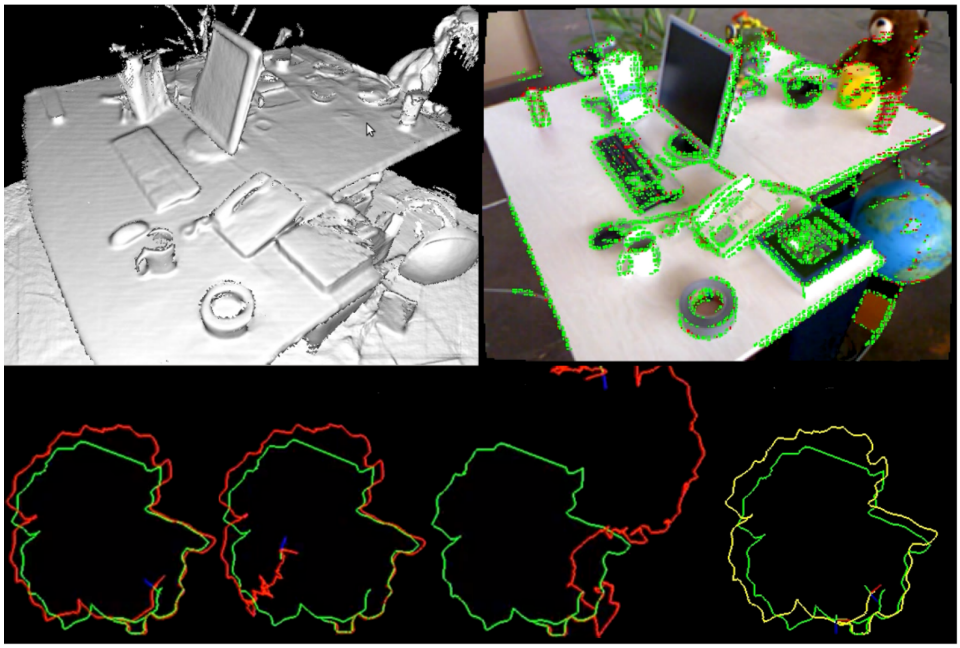


Figure 6.6: *Kinfu performance compared with incremental dense tracking using* `freiburg_desk2` *sequence with motion capture ground truth (green trajectory). Kinfu is executed with* $(3m)^3$*,* $(5m)^3$ *and* $(8m)^3$ *voxel volumes. The red trajectories on the left are output from from Kinfu. Based on the trajectories, Kinfu gains lower drift due to structure integration, but planar surfaces cause tracking failures.* $(3m)^3$ *volume does not contain the floor and therefore Kinfu works well. On the right, the yellow trajectory is the proposed incremental RGB-D tracking result. Problems with planar surfaces do not exist, and the method allows larger operating volumes due to lower memory consumption. By using keyframes, the drift can be completely eliminated. The green dots reveal the selected points.*

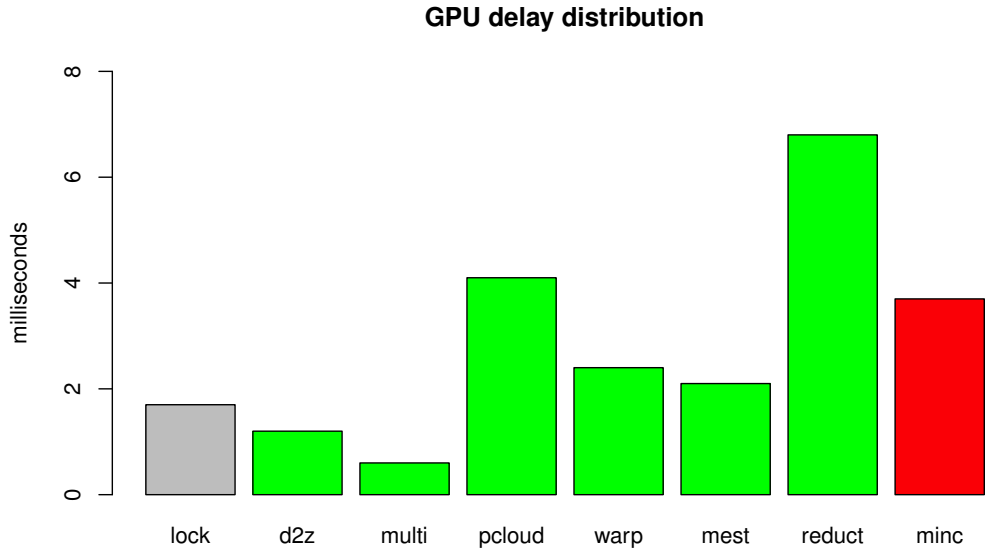


Figure 6.7: *GPU tracking time distribution* (23*ms*) *for one frame. Green bars represent phases scalable to n threads. The red bar is executed by a single thread. The gray bar (*`lock`*) contains CUDA interop delays which are only required for visualization.* `d2z` *is disparity to depth conversion,* `multi` *is multi-resolution pyramid generation,* `pcloud` *is the 3D reconstruction,* `warp` *contains the cost function evaluations,* `mest` *is the M-estimation,* `reduct` *is linear system reduction, and* `minc` *motion estimation and update.*

- CHAPTER 7 -

WATERTIGHT AND TEXTURED 3D RECONSTRUCTIONS BY RGB-D TRACKING

The keyframe SLAM algorithm (alg. 6.3) is tested in an appartment. The goal is to generate a 3D model without holes in the geometry. After recording a video, GPU-boosted RGB-D tracking is executed which produces 3D trajectory. Whether or not RGB-D tracking uses keyframes, only the trajectory is stored. The model keyframes are selected by looping the trajectory and storing a keyframe whenever user-specified angular or translational distance to the existing model is exceeded. The neighboring RGB-D measurements to the keyframes are efficiently localized (timestamp or frame index) and depth map fusion is executed. In depth map fusion, keyframe depth maps are filtered using all RGB-D measurements available. Then, optionally, bundle adjustment is possible for the full model. The results presented in this paper do not use bundle adjustment at all, because the sequences are relatively short and pose error remains small. Finally watertight polygon model is generated from the RGB point cloud using Poisson reconstruction method. Keyframe images are stored into a single texture and UV coordinates are generated for each polygon. The textured mesh is then stored in a simple Wavefront format which can be loaded into various standard 3D modeling programs for further refinement. The video[3] illustrates the full process for sequence `Room A`.

The phases in our process are thus

1. Record RGB-D video (manual)
2. Generate 3D trajectory by RGB-D tracking (automatic) (chap. 6)
3. Select keyframes (automatic)
4. Depth map fusion (automatic) (sec. 7.1)
5. Optional : bundle adjustment (semi-automatic) (sec. 7.2)

[3] `Video:http://youtu.be/tD3lFxrCHaw`

6. Watertight polygonization (automatic) (sec. 7.3)
7. Texture map generation (automatic) (sec. 7.4)
8. UV coordinate generation (automatic) (sec. 7.4)
9. Store Wavefront mesh (automatic)

§ 7.1 DEPTH MAP FUSION USING RGB DATA

By noting that local tracking is always accurate, depth map accuracy can be increased by local data fusion. Because the keyframes are sparsely selected from the stream of RGB-D measurements, they do not automatically utilize all information available. The intermediate point clouds, which are not selected as keyframes, are warped into a nearest keyframe and the final maps are filtered in post-processing. In effect, this improves depth map accuracy and fills holes. If larger resolution is used during depth fusion, more precise depth maps can be obtained [44]. In general environments with occlusions and complex geometries, the warped depths may form multi-peaked depth distributions in the reference view, and a method is required for determining a local depth range which contains *the best* depth value. The depth maps are corrupted by measurement noise, whose magnitude and distribution depends on the measurement device, the scene content and the distance. Noise is often managed by imposing smoothness constraints on the environment geometry. Such constraints can also cause problems as they might connect sparse geometries and add surfaces where they do not actually exist. One method to avoid smoothness constraints is to use median filtering for each points independently. This approach has been taken, for example, by Hirschmüller [25]. When the depths are transformed, only one point is allowed per each target pixel. In case of multiple points, the nearest one to camera is selected. Median filtering is rapid, robust and can be executed in parallel per each pixel using, for example, Quickselect algorithm [58]. Median filtering is especially useful when the depth values may have large variances (for example stereo camera + template matching). Depth maps with small variance can be used as initial guess directly. When the initial guess exists, bounds can be generated within which local averaging can take place. The inverse depth samples within $z_m \pm \delta$ are averaged to find the robust estimate (Figure 7.1). Inverse depths are more likely to have Gaussian distribution in stereo based settings [13, 48]. δ is the depth sample window, which is user-specified. δ depends on the RGB-D sensor depth noise level.

As can be seen from Figure 7.2, the area covered by a pixel increases with depth. If the median is computed per each pixel, care should be taken that $(\Delta x, \Delta y)$ will be bounded for the samples. Ray distance based uncertainty is roughly proportional to $\sigma_r \approx \frac{z}{f}\sigma_{\text{pix}}$, where z is the depth value, f

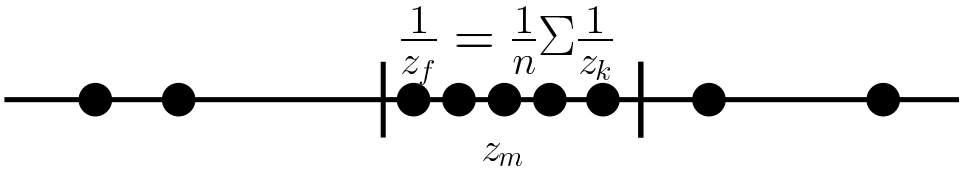


Figure 7.1: *Robust depth estimation. A local average is computed within bounds near the median depth. Inverse depths are more likely to have Gaussian distribution in stereo camera settings. In case of small depth error, the bounds can be directly initialized near a depth map measurement.*

is the focal length and σ_{pix} is the standard deviation of an image coordinate [13].

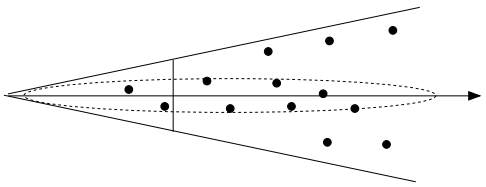
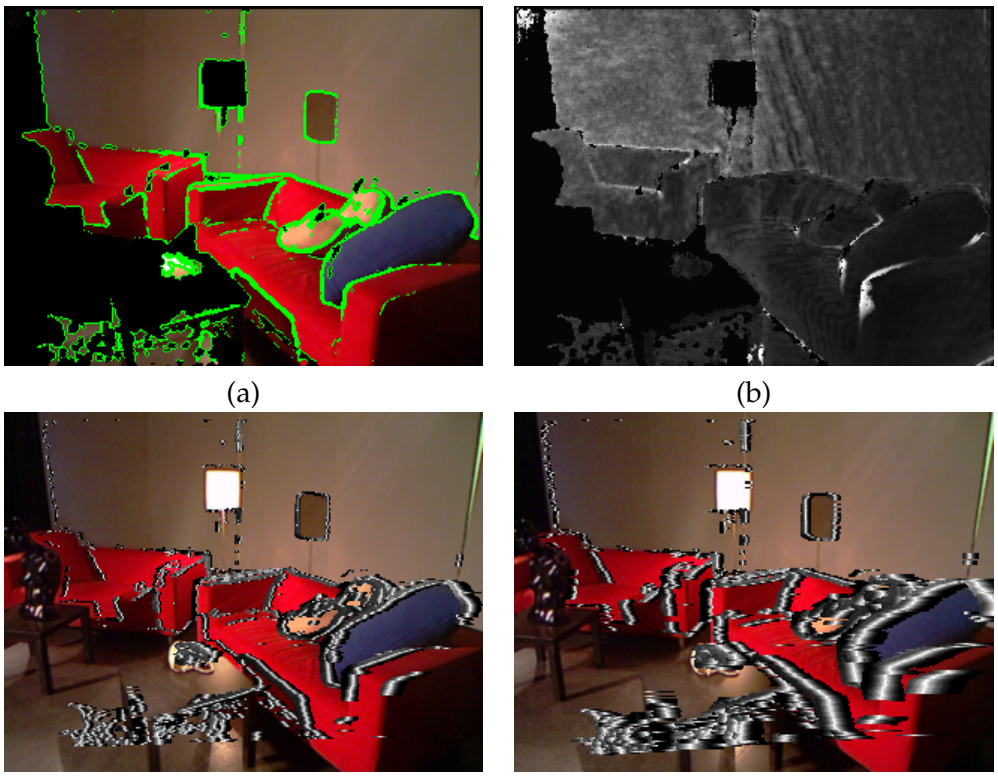


Figure 7.2: *The area covered by a pixel becomes larger at longer distances. The pixel median can be improved by focusing on samples near a ray which is defined through the pixel center.*

Now the depth values are defined by local averages near the median depth. It is expected that depth values are close to correct value. Thus photometrical adjustment is possible within uncertainty bounds. The bounds are generated by computing Gaussian variances from the depth samples near the optimum. Raw disparity maps are provided without any confidence measure. The Microsoft Kinect's benefit over a stereo camera is that structural error distribution is often local, where dense matching methods may produce any distribution depending on the scene texturing. Local error distribution in structure is crucial both for pose estimation and structural estimation. Locality reduces computational requirements and allows searching the best estimate within bounds. A large portion of the outlier points can be neglected based on color deviation to the keyframe pixels. Thus, prior to any filtering, it is useful to discard the points whose color difference to the reference color is larger than a threshold. The basic algorithm is listed in Algorithm A.1.

The filtered depth map can be photometric refined further using the standard deviations with Algorithm A.2. In practise, OpenMP is used to parallelize computations on CPU. The benefit with discrete optimization is that it works even when the amount of image measurements is small.

An example standard deviation image is illustrated in Figure 7.3b. As can be seen, the standard deviations increase monotonically as function of distance. The standard deviations are used to bound a photometrical refinement phase, which seeks the best depth value in terms of a re-projection error into multiple views (Sec. 3.7). The candidate points are generated by dividing the bound region evenly into n points.



(a) (b) (c) (d)

Figure 7.3: *Photometrical refinement for a 3D model. a) The green points are selected for photometrical refinement. Only the regions with strong gradient can be photometrically refined. b) Depth standard deviation image after depth map fusion phase. The photometrical search bounds are set based on this image. c,d) The cost values visualized for n depth values within the bounds. White color denotes low cost and black high cost.*

§ 7.2 Optional bundle adjustment for a 3D model

In the case that pose errors at the boundary keyframes contain too much drift, a global pose refinement must be done using bundle adjustment or a graph optimization framework [36, 37]. In smaller spaces, the initial pose estimates are, however, sufficiently accurate. Sparse bundle adjustment (SBA) can be used to remove the small global error at the end of estimated camera trajectory [37]. SBA does not converge without a good initial guess. Sometimes an initial guess can be successfully extracted from an unordered image set by extracting and matching feature points and initializing camera configurations using their geometrical relations. The process is error-prone because image content such as homogeneous texturing and repetitive patterns prevents reliable extraction/matching. SBA is then used to optimize global configuration. As a result the camera trajectory is improved and multiple geometries disappear (Figure 7.4).

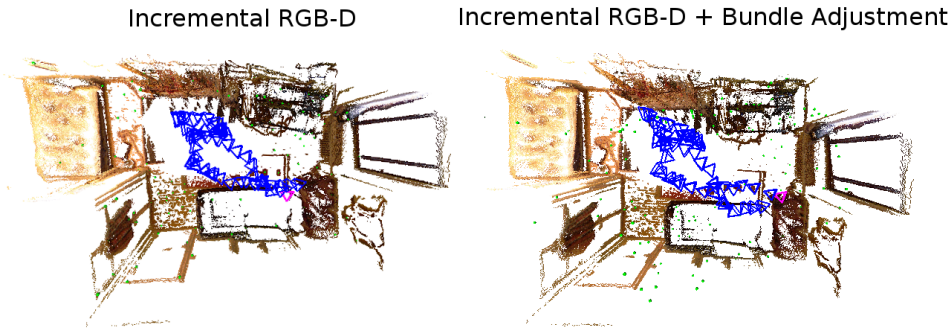


Figure 7.4: *The effect of sparse bundle adjustment illustrated. Minor error at the end of the trajectory is corrected.*

7.2.1 Interactive editor for bundle adjustment

An interactive tool was built to generate annotations for projection points (Figures 7.5,7.6). The editor produces a list of 3D points and their projections in all keyframe views in a small text file. A mouse is used for 2D point selection and linking across the views. The editor visualizes a 3D line segment for each 2D point. The line segment starts from the camera origin and ends at the 3D point determined by the depth map. Therefore its easy to verify that 3D points are associated with precise depth measurement. Finally, the output file is directly used to execute bundle adjustment function in SBA library. SBA library has depth axis in reverse direction compared

to OpenGL, which must be taken into account by mirroring camera depth axices and all depth coordinates.

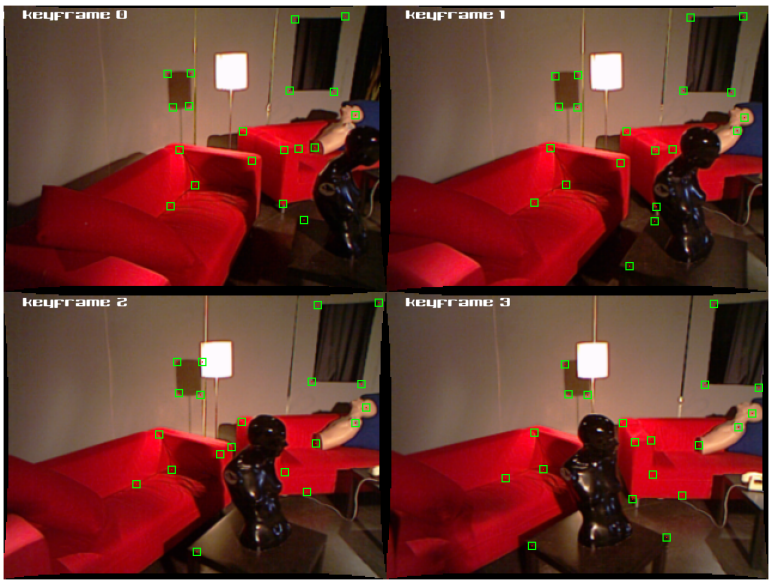


Figure 7.5: *The interactive tool for semi-automatic annotation of projection points illustrated. Annotated projection points can be added/removed using mouse. The projections are linked into a unique 3D point.The tool also shows corresponding 3D points (green spheres) along with 3D ray which can be moved over a keyframe image.*

§ 7.3 WATERTIGHT POLYGONIZATION

Polygon models are compact in their memory consumption and are better supported by standard 3D modeling programs than point clouds. A polygonization phase generates a polygon mesh from a point cloud. There are various methods available which generate polygon models from points clouds[5], but the main criteria in our applications is tolerance to noise and missing data. This rules out basic Delaunay based triangulation methods. Marron *et al.* acknowledge noise in the point clouds and propose a greedy method for rapid polygonization [43]. The method does not fit implicit surface to a point cloud, but incrementally grows surfaces by inspecting the near vertices with oriented normals. Triangles are generated based on local planar regression, to take into account noise in the raw point data, and also hole filling is supported. Marron's method has been used in Kintinous system which documents the results [89]. In the presence of sampling noise,

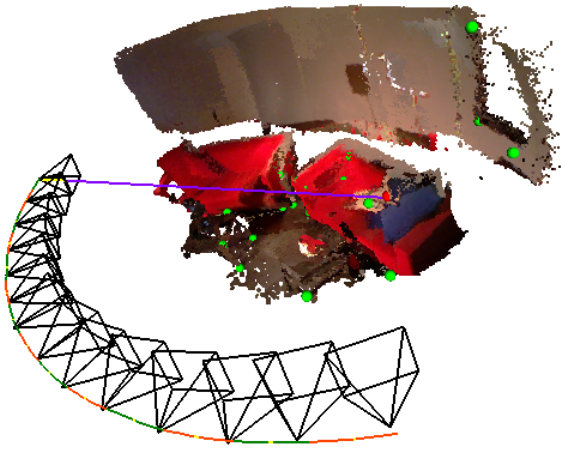


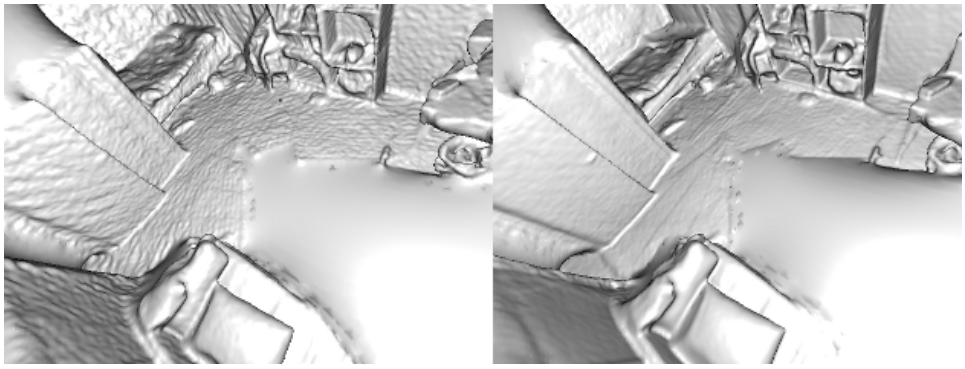
Figure 7.6: *The depth values for 3D points can be guaranteed to be correct by 3D ray vs. surface comparison.*

an alternative and common approach is to fit the points using the a zero level-set of an implicit function, such as a sum of radial base or piecewise polynomial functions [30].

In this work, the Poisson method is selected, because it produces a watertight surface based on a photometrically refined, oriented point cloud [83, 32]. The point normals are derived from the depth fused maps (eq. 2.59). Oriented points are transformed into a reference coordinate system. The Poisson method finds a scalar function whose gradients best match the vector field, and extracts the appropriate isosurface. The algorithm uses OpenMP for multi-threaded parallelization and octree data structure to reduce memory consumption [32]. To further avoid memory limitations, the mesh could also be done piece-by-piece using a single, moving reconstruction volume. The most interesting parameters are octree grid resolution, point weights and minimum point count in an octree node. Because octree resolution is limited, the resulting mesh may become oversmooth at complex regions. The reconstruction accuracy depends mostly on the precision of the oriented point cloud. With Microsoft Kinect sensor, the depth noise increases with distance. In our tests, we measure distance to the camera and set quadratically decaying point weights. Poisson reconstruction result without and with depth fusion can be observed in Figure 7.7b. Notice how Poisson method generates the floor and fills holes despite that measurements do not exist (compare with Fig. 7.7a).



(a)



(b)

Figure 7.7: *`Room B` sequence. a) A keyframe model obtained by executing keyframe SLAM in an appartment. b) Watertight Poisson reconstruction without and with depth fusion. Notice how holes are filled and missing regions such as the floor appears.*

§ 7.4 MESH TEXTURING

The example sweep in Figure 7.7a created 23 RGB-D keyframes whose textures are redundant in color due to Bayer filtering. The images are downsampled into 320×240 resolution and stored into a 2048×2048 texture (Figure 7.8). The size is good for testing purposes and it allows 6×8 keyframes to be used.

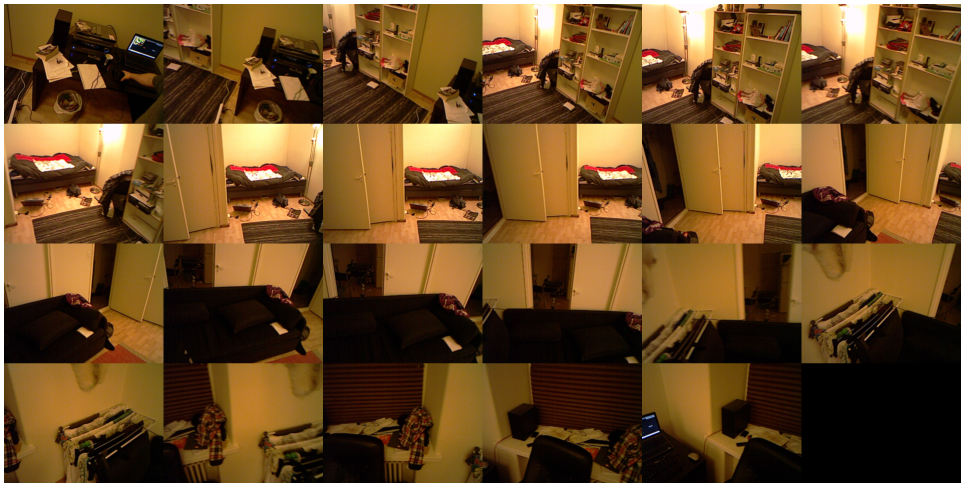


Figure 7.8: *Keyframe images are stored into a single big texture.*

The Poisson polygons which are in the visible range of the RGB-D sensor are projected onto all keyframe views and UV-coordinates are generated. Poisson reconstruction module outputs polygons with n corner points. 2D area of a polygon $\triangle$ is evaluated using the following formula

$$A(\triangle) = \frac{1}{2}\sum_{k=0}^{n} \det\Big(\begin{bmatrix} \mathbf{p}_k^T \\ \mathbf{p}_{\mathrm{mod}(k+1,n)}^T \end{bmatrix} \Big), \tag{7.1}$$

where points $\mathbf{p}_k = (u, v)^T$ are the corner points of a 2D polygon. The formula applies to convex and concave polygons as long as they are not self-intersecting. When a polygon is visible in more than one view, we choose to favor largest spatial resolution with the formula

$$\triangle_{\mathrm{uvkey}} = \underset{j}{\mathrm{argmax}}\ A_j(\triangle) \in \mathbb{N}, \tag{7.2}$$

where $\triangle_{\mathrm{uvkey}}$ is the index of the best UV-mapping keyframe (Fig. 7.9). Because frequent switches in UV-mapping directions can cause visually disturbing seams, mapping can be improved by enforcing the locally dominant keyframe. One method to do so is to recursively enumerate con-

nected polygon neighbors in n passes, and then prefer the mapping direction which has the largest number of votes. Finally the selected UV-coordinates are converted into global texture coordinates and stored. The keyframe images are not undistorted to better maintain maximum texture quality. The resulting meshes can be observed in Figure 7.12.

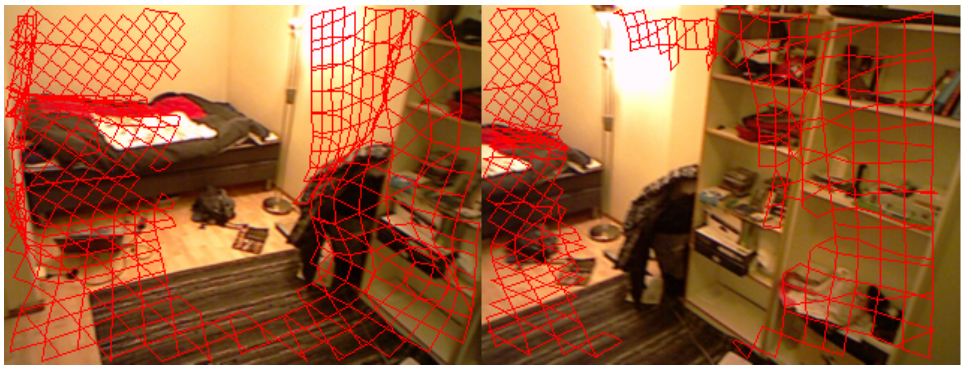


Figure 7.9: *Poisson(7) polygons are projected onto keyframe images. UV-coordinates can be chosen from the view which has the best spatial resolution. The corner of the shelf is visible in two different keyframes, but the selection favors the left image, because the camera is closer.*

When final texturing is patched together from keyframes, some color banding effects may occur if brightness varies across the images. Manual camera settings reduce global lighting variation across images. To reduce the problem further, averaging or median filtering could be attempted. Also more sophisticated multi-texture blending methods exist [59, 75].

§ 7.5 MEMORY CONSUMPTION

The memory consumption is shown in Table 7.1. The consumption is separated into geometry and texture consumption for Poisson meshes. The datasets `Room A`, `Room B` and `Kitchen` have 47, 23 and 14 keyframes. After the trajectory has been estimated, a raw point cloud is generated. The minimum requirement per vertex is 9 attributes (position, normal and color). After Poisson reconstruction and texture mapping, the vertices require only 3 floats (x,y,z) and n triangles have in total $9 * n$ attributes ($3 * 2$ UV coordinates + $3 * 1$ index). In addition texture map requirement is computed directly as $k * 320 * 240 * 3$, where k is the number of keyframes. Table 7.1 shows memory footprint for $2^7, 2^8$, and 2^9 octree grids. The corresponding geometric quality is illustrated in Figure 7.10. The reconstructions in Figure 7.12 are generated using 2^9 grid.

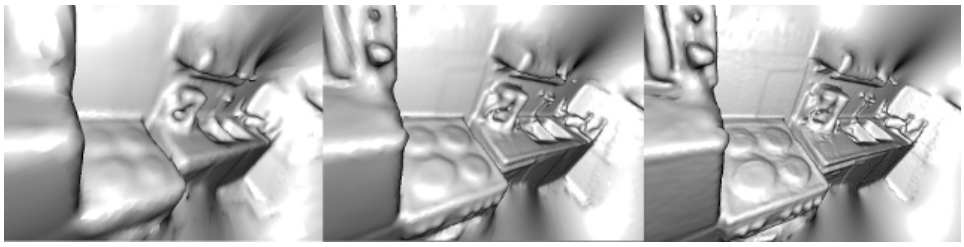
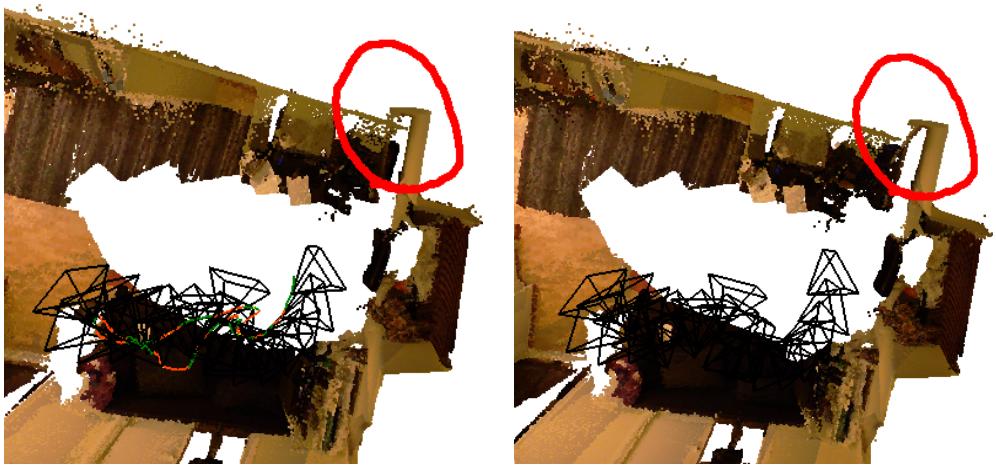


Figure 7.10: *Kitchen scene reconstructed with 2^7, 2^8 and 2^9 octree resolution. Phong shading reveals the level of geometrical details at each resolution.*

Table 7.1: *Memory consumption of the test sequences.*

Dataset	Raw	Poisson(9)	Poisson(8)	Poisson(7)
`Room A`	$124MB$	$(32+12)MB$	$(7.8+12)MB$	$(2.0+12)MB$
`Room B`	$60.6MB$	$(21+5)MB$	$(5.8+5)MB$	$(1.6+5)MB$
`Kitchen`	$36.9MB$	$(14+3)MB$	$(3.8+3)MB$	$(1.1+3)MB$



(a) (b)

Figure 7.11: *`Room B` sequence. Reconstruction bias in when operating in a) incremental mode, b) SLAM mode. After 360° turn the first and last keyframe should map points consistently into a single 90° corner. In this case a) is slightly more precise, because subsequent images are photometrically the most comparable with negligible interpolation errors.*

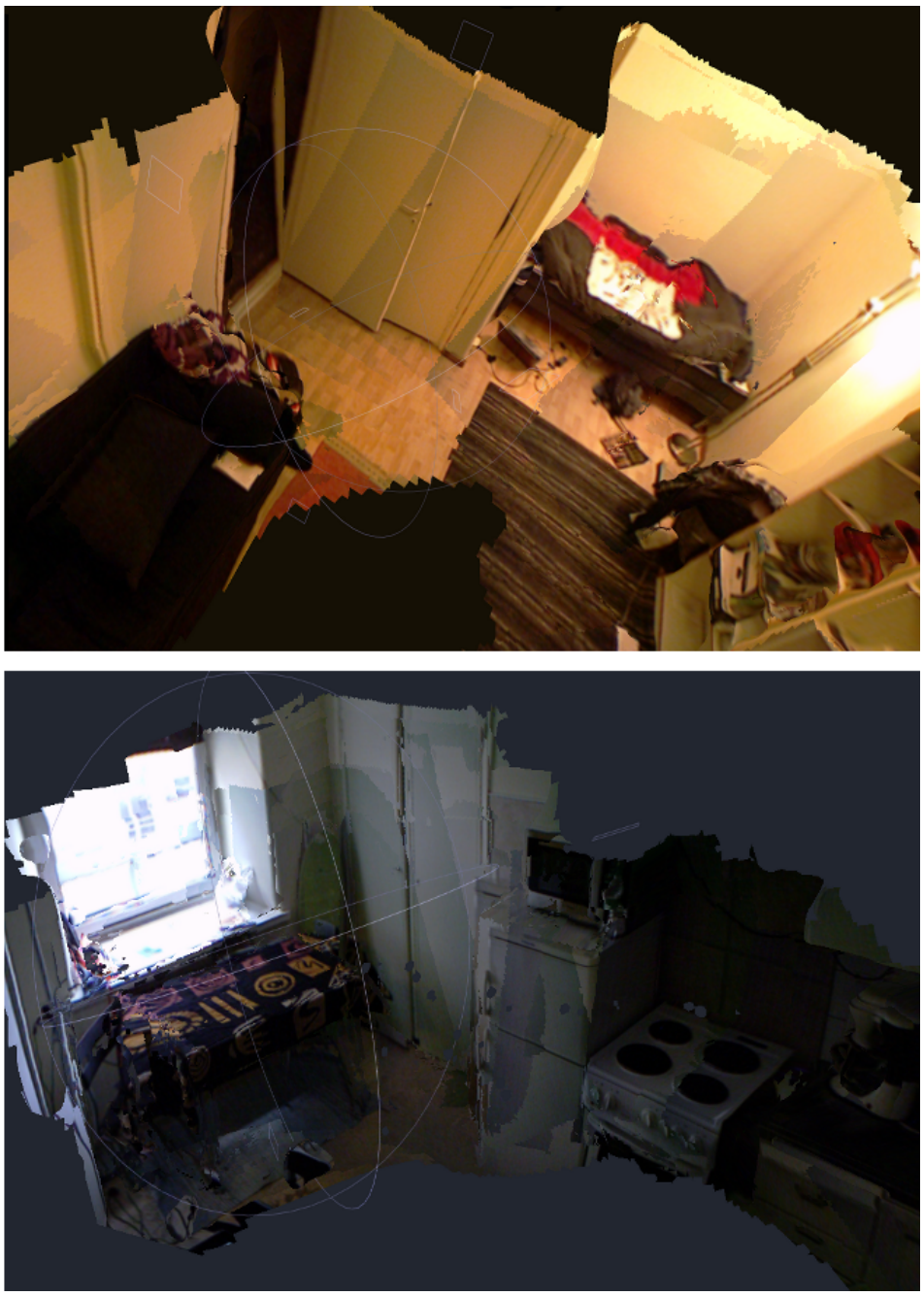


Figure 7.12: *Final textured Poisson meshes loaded into Meshlab for inspection: a) `Room B`, b) `Kitchen`. Poisson reconstruction produces watertight mesh, whose texturing is photorealistic as it is directly mapped from the keyframe images. The cost of reduced memory footprint is over-smoothing, which may occur at thin surfaces such as the shelf in 7.12a. Also lighting changes can be detected at seams where texture data source switches from one keyframe to another. Otherwise the models are photorealistic and in metric units.*

- CHAPTER 8 -

AUGMENTED REALITY IN LIVE TELEVISION BROADCASTING

In this chapter, a real-time camera tracking system is developed for television production studios based on our GPU implementation. The aim is to reduce the costs of matchmoving in studio environments by introducing an affordable RGB-D sensor based camera tracking tool which operates in real-time, fits studio use, and only requires a low-end GPU. This system has the following novelties in comparison to other low-cost camera tracking solutions [1, 53]. 1) A RGB-D keyframe-based tracking method is proposed, which does not suffer from time-evolving drift. A static studio scene is first modeled as a database of RGB-D keyframes, which are obtained by using incremental dense tracking approach and fine-tuned using bundle adjustment. The database is then used as a reference for real-time pose estimation (Figure 8.1)[1]. By defining the camera tracking problem relative to the nearest keyframes, drift is avoided. It will be shown how the keyframe tracking eventually outperforms incremental tracking. The estimation is robust due to gradient-based pixel selection and an M-estimator.

When aiming at processing dense RGB-D data in real-time, the algorithms must be parallelized to obtain sufficient performance and scalability properties. 2) The computational scalability of the camera tracking is improved by designing the full algorithm for a low-end GPU. A detailed description from a cost function definition into an efficient GPU implementation is given.

3) With a static keyframe-based 3D model available, the dynamic foreground points are rejected from the camera pose estimation by observing discrepancies in intensity and depth simultaneously. Generally in RGB-D tracking, outliers both in color and depth can exist (occlusions, foreground

[1] http://youtu.be/L_0LnFc7QxU

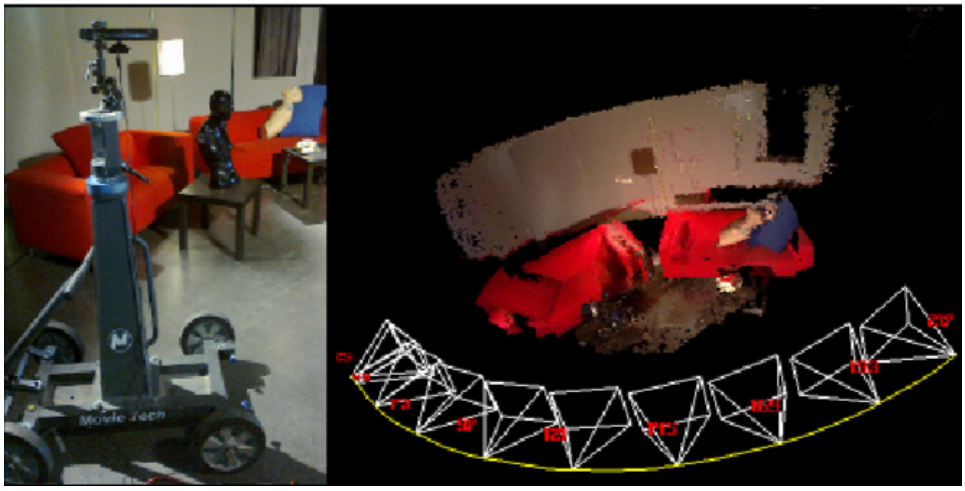


Figure 8.1: *A RGB sensor is used to build a keyframe database which contains the views in the camera motion zone. The TV camera pose is estimated by registering the image content with the nearest keyframe.*

objects, lighting effects etc) and must be taken care of. Our results are verified in a real television broadcasting studio with and without foreground dynamics. By these steps, RGB-D based tracking is evaluated in actual application use [84].

§ 8.1 SYSTEM

An overall sketch of the full system is illustrated in Figure 8.2. The system consists of a motion capture studio (NaturalPoint Optitrack [51]) which sends motion capture stream in real-time to a Linux client which uses Panda3D engine to render an interactive 3D character into live TV broadcast. The camera pose is tracked using the developed GPU implementation (Chapter 6). First a static keyframe model is recorded using keyframe SLAM mode prior to broadcasting. Then live tracking utilizes the keyframe model to obtain driftless tracking which is robust to foreground actors when sufficient portion of background remains visible. A professional HD camera is calibrated with the Microsoft Kinect sensor and the depth maps are re-sampled to match HD camera point of view. The depth maps are available to AR composition, where the final pixels are selected based on the distance.

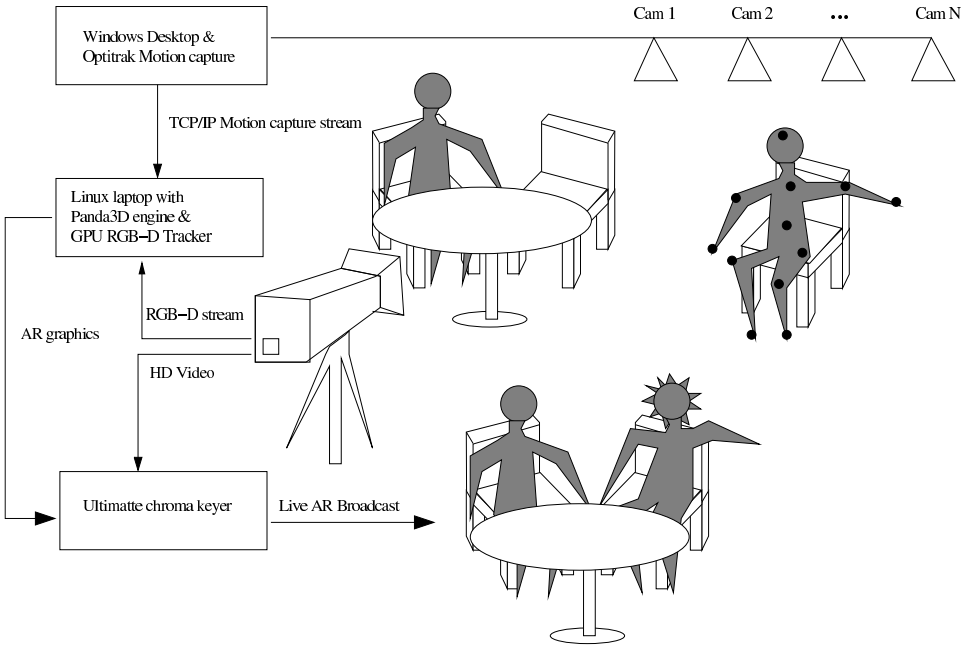


Figure 8.2: *A schematic view of the overall broadcasting system. The RGB-D sensor is attached to the TV camera and oriented either towards the scene or toward the floor. The former configuration uses background scene texturing and the latter textured carpet for 3D tracking.*

8.1.1 AR graphics using Panda3D

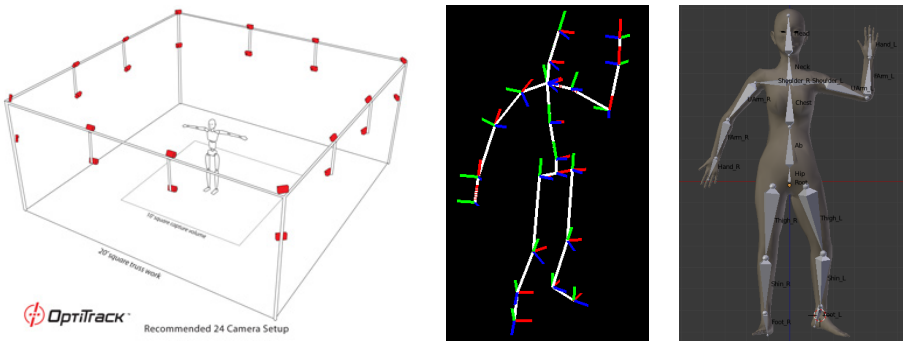
AR graphics are rendered in real-time using Panda3D engine. Panda3D is a library of subroutines for 3D rendering and game development [11]. The library is C++ with a set of Python bindings, but in this project only C++ components were utilized. Panda3D was created for commercial game development, and its primary users are still commercial game developers. It was developed by Disney for their massively multi-player online game Toontown, and has also been used in the Pirates of the Caribean game. It was released as free software in 2002. Since version 1.5.3, Panda3D has been released under the modified BSD license, which is a free software license with very few restrictions on usage. Panda3D is currently developed jointly by Disney and Carnegie Mellon University's Entertainment Technology Center. In this project, Panda3D is used to render AR graphics into live video stream from the current camera angle (Figure 8.4). Panda3D animates a character using the skinning technique, where a discrete set of rigid bodies deform skin vertices based on their bone weights.

8.1.2 Motion capture system for live character animation

In our studio application, we experiment the Optitrak system for live 3D character animation. The Microsoft Kinect SDK has a built-in skeleton tracker, but it is only useful for games which do not require maximal precision. The tracking uses a single point of view and therefore can not always track the full body configuration. Tracking failures happen with complicated/partially occluded motion. When the pose is too complicated for the tracker, it is enforced to match its training data [65]. Despite the shortcomings, the Kinect SDK pose estimator is an alternative for low-cost live motion capture. The NaturalPoint Optitrack system, however, is more precise, because it uses tens of cameras for skeleton tracking. The system is bundled with Arena software which supports real-time 3D character pose streaming. 24 cameras in a truss setup track character pose in 3D based on retro-reflective marker motion (Fig. 8.3). The markers are illuminated by IR light sources which are attached to the cameras. In this case, the IR light will not interfere with the RGB-D sensor, because the motion capture system is not setup in the same space with the scene (Figure 8.2). An actor wears a special full-body costume where the markers are attached. The tracking operates fully in IR zone. Arena software receives a synchronized set of IR images, extracts the marker 2D coordinates and reconstructs a sparse 3D point cloud. The software fits a fixed skeleton structure into point cloud and converts the data into a set of rigid bodies. The software uses NatNet protocol for real-time data broadcasting over a network. A custom client software was implemented which connects to Arena and receives the data. The motion parameterization is mapped to a Panda3D 3D character, which is created in Blender. The skeleton armature was made compatible by inspecting and replicating Arena software output. Because the local coordinate systems did not automatically match, fixed conversions were required from the Arena definition into the Panda3D format. Also the bone scales vary depending on the motion capture actor. Therefore local scale adjustment was made possible, which functions by scaling the relative translations between the bones.

8.1.3 AR composition using depth maps

AR composition was experimented on the GPU using a custom-made `cg` shader [42], which compares Kinect depth value and AR character depth value and renders the source which is closer to the camera (Figure 8.4). Depth map up-sampling is necessary when registering range images with professional HD camera images. When a point cloud is transformed and projected onto a RGB image, it will produce an irregular set of 2D sample points whose data should be interpolated over the full image. If the point set is relatively accurate, a connected mesh (based on the original image



(a) (b) (c)

Figure 8.3: *a) 24 cameras in truss setup are used to track live 3D character motion. b) Optitrack skeleton structure c) Blender skeleton structure*

grid) can be ray casted to obtain a depth value for each pixel. In a trivial approximation, the source depth map is up-sampled and the points are directly transformed into a target view. All holes are not filled, but computational requirements will be low when using a parallel implementation.

Also novel methods are available, but have not been tested in our application. *Joint bilateral up-sampling* is a method which uses a high resolution RGB image to guide depth map filtering [35]. A depth data is diffused within precise edges in the HD image using a bilateral filter. The process is continued by a sequence of post-processing passes which improve depth map continuity locally until the quality is sufficient [69]. One pass discretely optimizes such depth variants which produce smoothest neighborhoods. Local discontinuity is used as the error function, and pixel domain is the same as with the initial bi-lateral pass. The subregions which are already smooth are not altered. This approach is GPU-friendly because each subregion can be processed independently. Depth discontinuity weights are not used, and therefore problems can be expected on depth discontinuity regions where color does not significantly change.

Capturing HD images directly into GPU memory is possible using NVIDIA SDI capture card which is placed to the side of a graphics card. Then data transfer is possible transparently and directly into the GPU memory. This setup is particularly useful because transfer of full HD image resolution (1920×1080) easily becomes a bottleneck. Typical graphics cards store images first into RAM and then upload them into the GPU. In this applications, the GPU is heavily used already and memory transfers must also be optimized.

In practise, the depth maps obtained using a RGB-D sensor have an amount of measurement noise, which produces noise to composition. The depth values are captured correctly at sofa regions, but especially floor and

background wall contain noise. Visual artifacts are possible only in preview use, such as when practising scenes which have virtual 3D characters. For noiseless composition, Ultimatte chroma keyer[2] performs purely key color based composition. Ultimatte was experimented in a live television broadcast without problems.



Figure 8.4: *A real-time AR broadcast with virtual 3D characters which are rendered from the estimated pose using Panda3D. See example video*[1].

§ 8.2 Studio Lighting and Microsoft Kinect

In studio environments, the color constancy assumption works well, because lighting can be fixed. The keyframe database will be valid as long as the lighting conditions do not change. However, the Microsoft Kinect sensor adapts to environment lighting changes by default. This degrades the tracking algorithm performance and prevents total lighting control during broadcasting. Due to Ubuntu Linux operating system, the `freenect` API was used in the project as it is compiles easily on Linux and is customizable and open source. The API operates by sending and receiving messages

[2] `http://www.ultimatte.com`

using the USB channel. According to reverse engineered protocol documentation, it is possible to fix white balance and exposure parameters. By experiment, the initial values could not be set however. The initial values are determined by the current lighting conditions. This is slightly restrictive since, for example, the frame rate easily switches into 15Hz if the initial conditions are low light. Also OpenNI drivers are available for Linux, but were not experimented. The Microsoft Kinect RGB camera is of low quality and in standard studio lighting conditions the colors can saturate (see Fig. 8.5). Also specular surfaces must be avoided in the scene and external IR light must not exist.

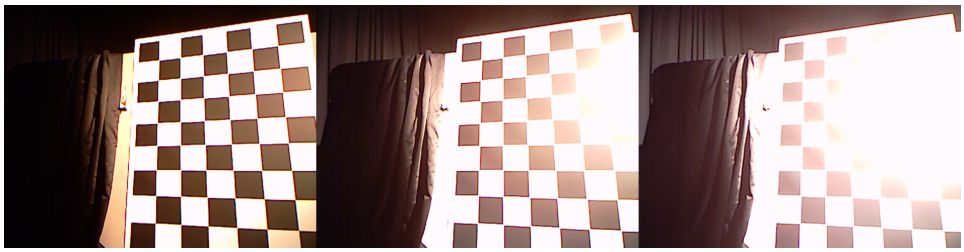


Figure 8.5: *Brightness saturation effect with Microsoft Kinect sensor when increasing studio lighting.*

With professional HD cameras this is not the case since they have remarkably better dynamic range (Figure 8.6). To circumvent this limitation, the RGB-D sensor depth maps can be directly calibrated with a HD camera [23] (Section 8.1.3).



Figure 8.6: *HD camera (left) and Microsoft Kinect RGB image (right) in a typical studio lighting conditions.*

§ 8.3 TRACKING CONFIGURATIONS

Two different tracking configurations are considered. The RGB-D sensor is oriented either towards the scene or towards a textured carpet on the floor

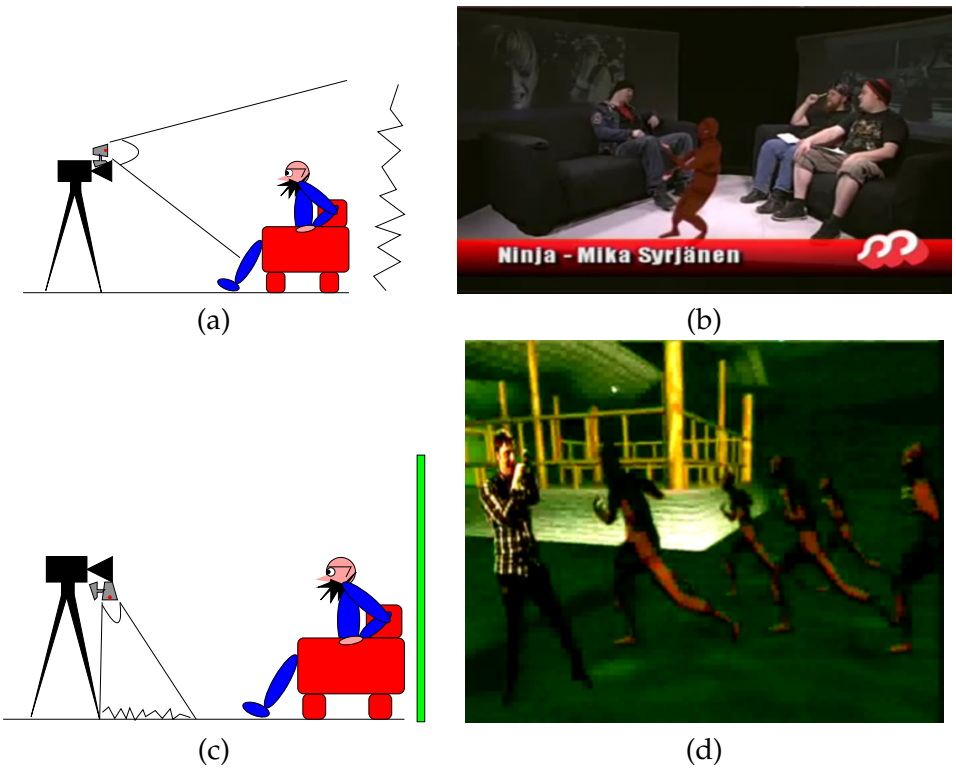


Figure 8.7: *The camera tracking was experimented in two different configurations. a) The RGB-D sensor oriented towards the scene where and tracking uses the background pixels. b) AR character added to a physical scene c) The RGB-D sensor oriented toward the floor with a textured carpet. d) A virtual scene with a real character extracted from video.*

(Figure 8.7).

8.3.1 RGB-D sensor towards the scene

This is a novel approach, which allows using high quality gradient information from HD images to increase RGB-D tracking accuracy. Professional HD cameras are also compatible with studio lighting conditions (Section 8.2). The challenge is, however, to avoid the foreground actors during the estimation. When HD images are directly stored to GPU memory, AR composition is possible with a customized technique. By implementing chroma keying directly on GPU, the system costs are significantly reduced. The *de facto* standard is the Ultimatte keyer, which performs chroma keying in many production studios and costs 25k€.

During broadcasting, the scene will contain moving actors. The pose estimation should only use the static background in order to avoid estimation bias. After the static keyframe model has been captured, the scene can be segmented into static and dynamic components. The segmentation is useful for weighting the dynamic foreground out from camera estimation. The Tukey weighting is illustrated in Figure 8.8. The estimation requires a sufficient amount of visible points and, therefore, foreground actors are not allowed to occlude more than a fraction of the selected points. In the Figure 8.8, rougly 1/3 of the keyframe points are occluded and the tracking still works. See an example video of M-estimator performance in a scene with extreme foreground motion [4]. The moving actor does not disturb camera tracking when relying on the keyframes.

In case multiple cameras are used during the broadcast, RGB-D sensors interfere each other, because each sensor projects an IR pattern towards the scene. Reconstructing a precise depth map from a mixed IR pattern is difficult. One solution has been proposed by Microsoft Research to overcome this problem [9]. The idea is to literally shake the RGB-D sensors to blur the interfering patterns out from the disparity map estimation. In practise, attaching a shaking device into TV camera requires special attachment method.

8.3.2 RGB-D sensor towards the floor

This is a typical industrial solution, because outliers do not exist and a simply homography mapping models image changes (Section 3.1.3). In the experiment, A flower pattern was printed on a $2.5m \times 2.5m$ carpet and set below a TV camera. The flower pattern provides excellent image gradients in all directions and does not have repetitive texturing which could confuse the tracker. The RGB-D sensor was attached as close to the TV camera objective as possible and oriented towards the carpet. The RGB-D tracker was executed in the SLAM mode (Section 6.1.3) and thus the pattern is learned concurrently to tracking (Figure 8.9). Multiple AR characters were positioned on the carpet and animated using a single 3D pose stream sent by the Arena software. The floor was estimated from the first RGB-D measurement by plane regression. Plane parameters are used to position AR characters exactly on the floor level. Then camera is rotated and translated in 3D space above the floor. AR graphics remain at their correct floor coordinates through-out the experiment[5].

This configuration allows using a green-screen for background subtraction and allows using multiple cameras, because the RGB-D sensors do not interfere each other. Now the background scene is fully computer generated, and rotates and translates based on the camera pose (Figure 8.7d).

[4] http://youtu.be/iVdYPbHY2ro

[5] http://youtu.be/CpXXWeCDD5o

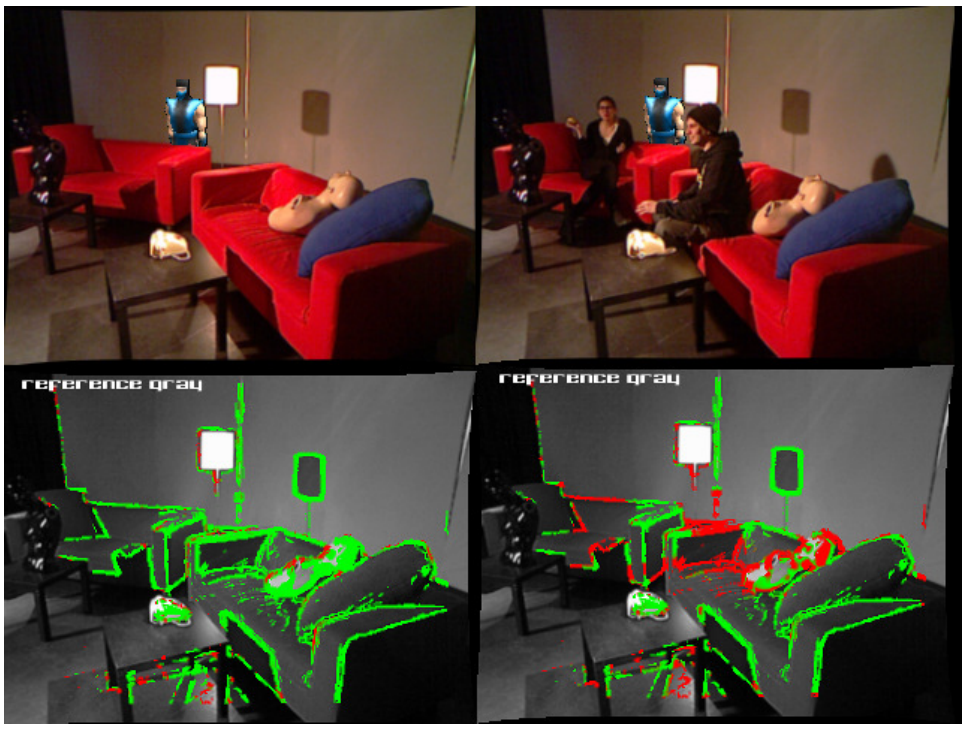


Figure 8.8: *The points which currently participate in pose estimation are selected based on the image gradient magnitude. M-estimator weights are damped by the depth residual based weights to neglect the dynamic foreground. The weight 1.0 is assigned to the green points and 0.0 to the red points. The foreground actors can cause a tracking failure in KinectFusion system [53].*

Because TV camera and RGB-D sensor are pointed to different directions without any overlap, the estimation of the relative transformation is trickier. Rotational motion is the same for both cameras, because they belong to the same physical body. The RGB-D sensor was placed as close to the TV camera objective as possible to minimize translation bias.

The angle and translation range is limited when using a textured carpet and a green-screen. In our studio, the green-screen was set only on one wall and it was difficult to rotate the camera and maintain fully green images. The rotation range was only $\sim 45°$. The carpet also limits translational motion to one meter in all directions. In many TV studios, fancy camera motion does not exist, and cameras are only rotated, translated and zoomed little during the broadcast. Thus carpet solution is practical in studio use and can replace pan/tilt heads, which mechanically measure limited camera motion in a similar use case.

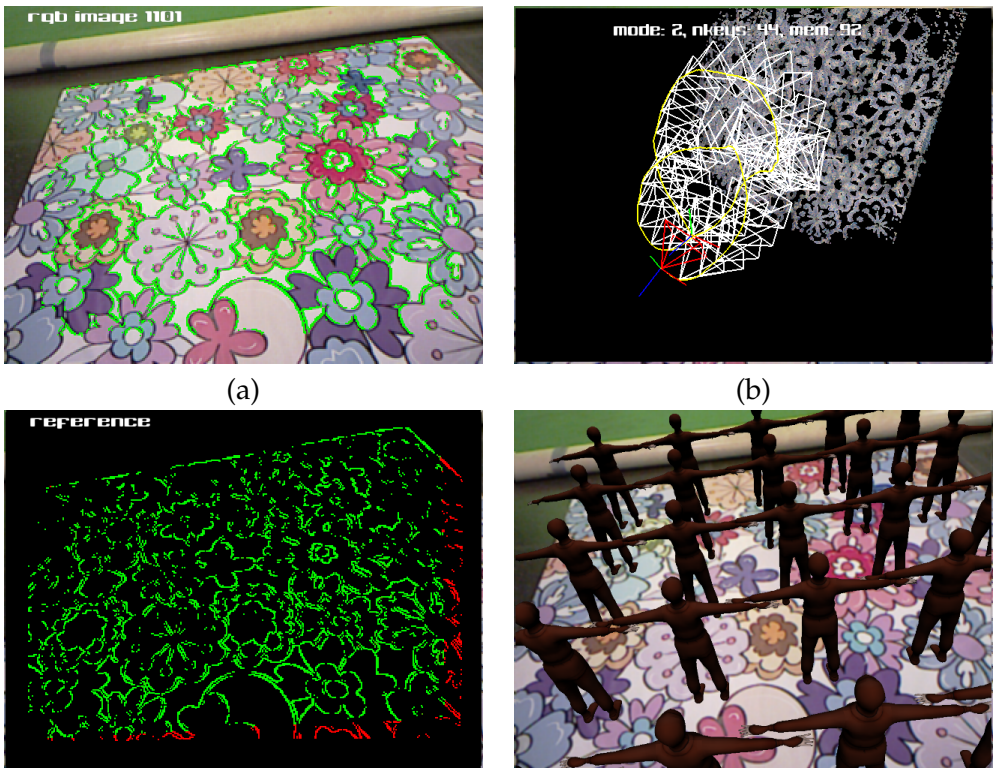


Figure 8.9: *a) A carpet with a flower pattern used for 3D camera tracking. Selected points on green. b) 3D map of the carpet built concurrently. c) Nearest keyframe points illustrated on green. d) AR graphics positioned on the carpet coordinate system.*

§ 8.4 3D MODELING OF A STUDIO ENVIRONMENT

The same pipeline described in Section 7 is useful for 3D modeling of a studio environment, but in addition a keyframe consistency check turned out to be useful (sec. 8.4.1). The initial setup contained an IR emitting light-source and a specular statue which corrupted regions in the depth maps.

A polygon model is useful when designing AR interaction in a reference coordinate system. It allows defining AR placement in the scene and enables motion with collisions to real world objects. Despite the fact that the algorithms formulated in this project all utilize point based reconstructions, polygon models are more compact in their memory consumption and are better supported by standard 3D modeling programs such as Autodesk's 3DSMax and Meshlab. A Poisson polygonization was discussed in Section 7 and an example in studio environment is illustrated in Figure 8.10

with video[3].

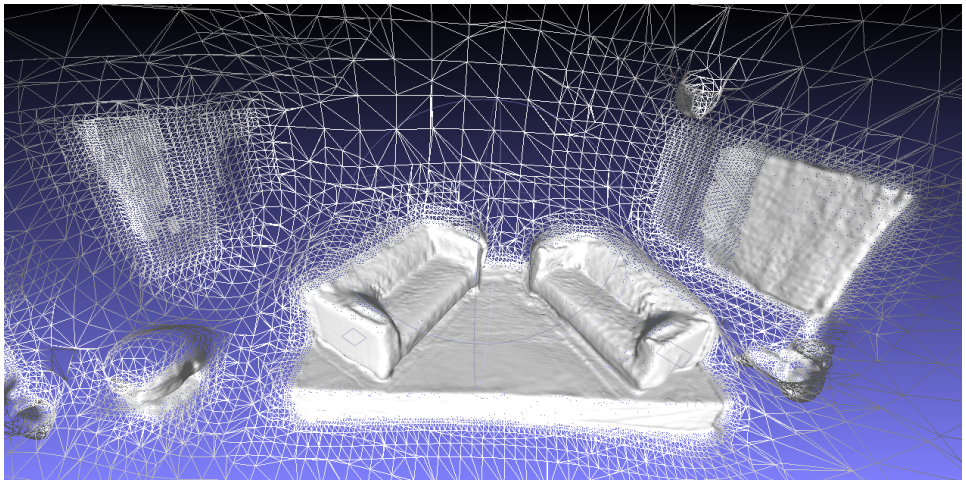


Figure 8.10: *Wireframe model of TV studio scene via Poisson method.*

8.4.1 Depth map noise in a studio environment

Image-based pose estimation depends on the depth map quality and the amount of static edges. An experiment was carried out to check how consistent 3D structure is produced when combining the selected keyframe points from multiple keyframes into one reference coordinate system. An example studio scene is illustrated in Figures 8.1 and 8.4. When four keyframes are combined in Figure 8.11, it can be noticed how the IR emitting light source and the black specular statue in the scene produce inconsistent edge information. False depth coordinates bias pose estimation, because motion across images is not fully explained by camera motion. M-estimator reduces outlier problems, but does not prevent bias, because also outlier points with small residual are accepted. IR emitting or specular surfaces should not be placed into the studio scene, but the problem can reduced by performing filtering described in section 8.4.2.

8.4.2 Depth map filtering for studio model

To increase keyframe quality for online 3D tracking, the depth maps are fused from several RGB-D frames. As described in Section 7.1, it is also useful to neglect points which are statistically rare. When a certain (x, y) coordinate in the reference depth map does not collect a sufficient amount of support with similar RGB color, it should be discarded. This mechanism essentially filters out all geometry whose appearance is not directly

[3] Video: `http://youtu.be/Xnn_06r7tFE`



Figure 8.11: *4 keyframes transformed into same reference coordinate system. Kinect reconstruction problems can be noticed from multiple edges. The blue circle reveals how IR emitting light source does not have consistent edges. The purple circle reveals how the black specular statue does not have consistent edges.*

dependent on camera motion. Also the reconstruction problems illustrated in Fig. 8.11 can be removed from the final reconstruction. Observe how the incorrectly reconstructed specular statue and IR light source disappear when increasing the minimum amount of support points N (Figure 8.12). The result after geometric depth fusion illustrated in Figure 8.12f.

§ 8.5 Experiment: Tracking accuracy

In Figure 8.13, it is shown how the drift increases with dense tracking when moving front and back along a fixed rail in a studio environment. The pose estimation problem could be replaced by a simpler one, since the motion is restricted. The experiments are, however, carried out in the most general 6DOF, because scene specific tuning degrades usability. Distance to the ground truth is measured in every frame as an error metric. For this sequence, the ground truth is generated using dense tracking but without the bundle adjustment phase. The drift problem is solved by tracking relative to corrected keyframes. In a long-term use, even a small number of keyframes eventually outperforms dense tracking due to drift. With shorter sequences drift is naturally negligible[6].

The demonstration video shows the difference between dense tracking and keyframe tracking in terms of AR graphics [7]. In this experiment, larger tracking bias can be expected. The RGB-D sequence is difficult, because it contains only few edges which can be used to track the camera. Some of

[6] http://youtu.be/wALQB3eDbUg
[7] http://youtu.be/zfKdZSkG4LU

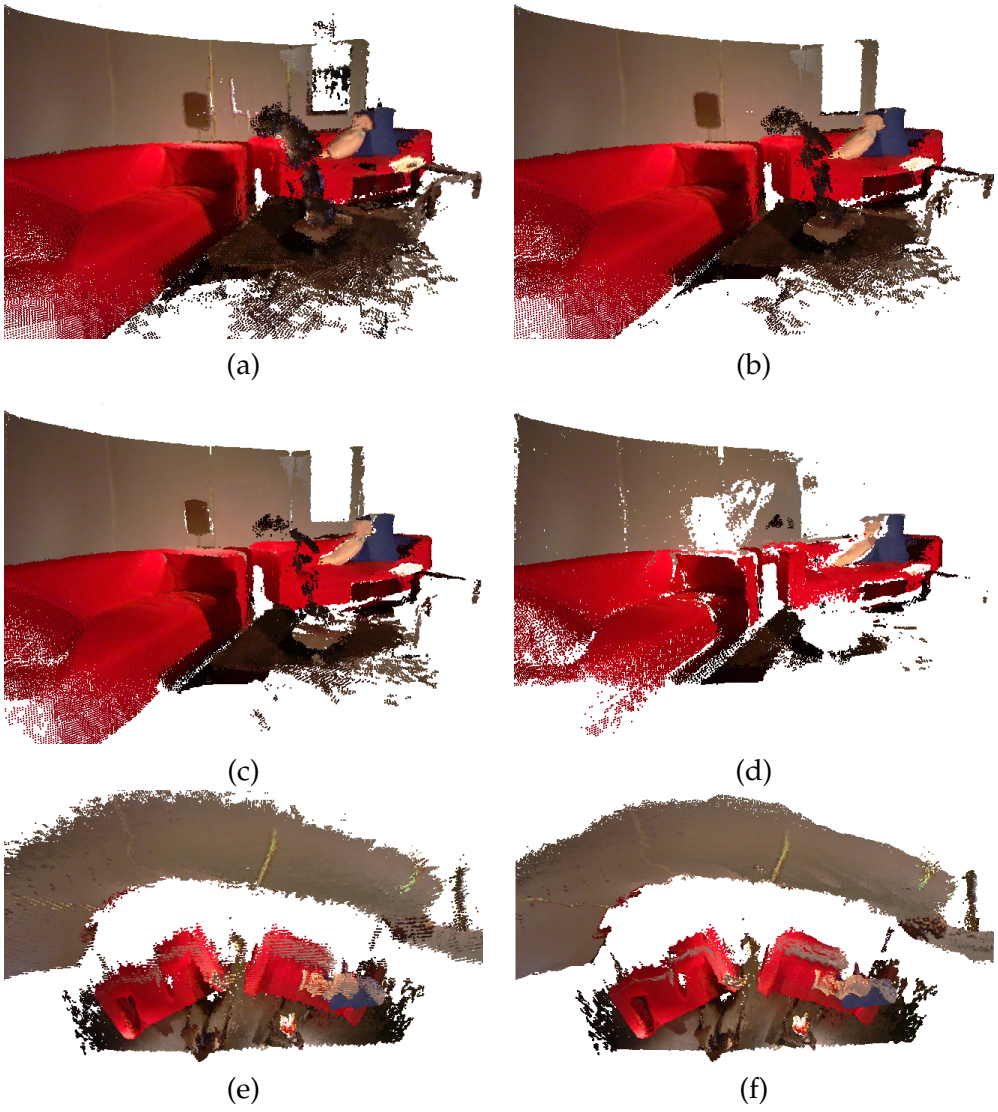


Figure 8.12: *Filtering out inconsistent 3D points. The images illustrate final keyframe model, when each 3D points must have at least N observations with same color. a) $N = 1$ b) $N = 32$ c) $N = 64$ d) $N = 128$. Notice how the black statue and IR emitting light disappear when N is increased. These object types are not well supported by Microsoft Kinect sensor. e) The raw keyframe point cloud, f) The keyframe points after depth map fusion. Random noise has disappeared.*

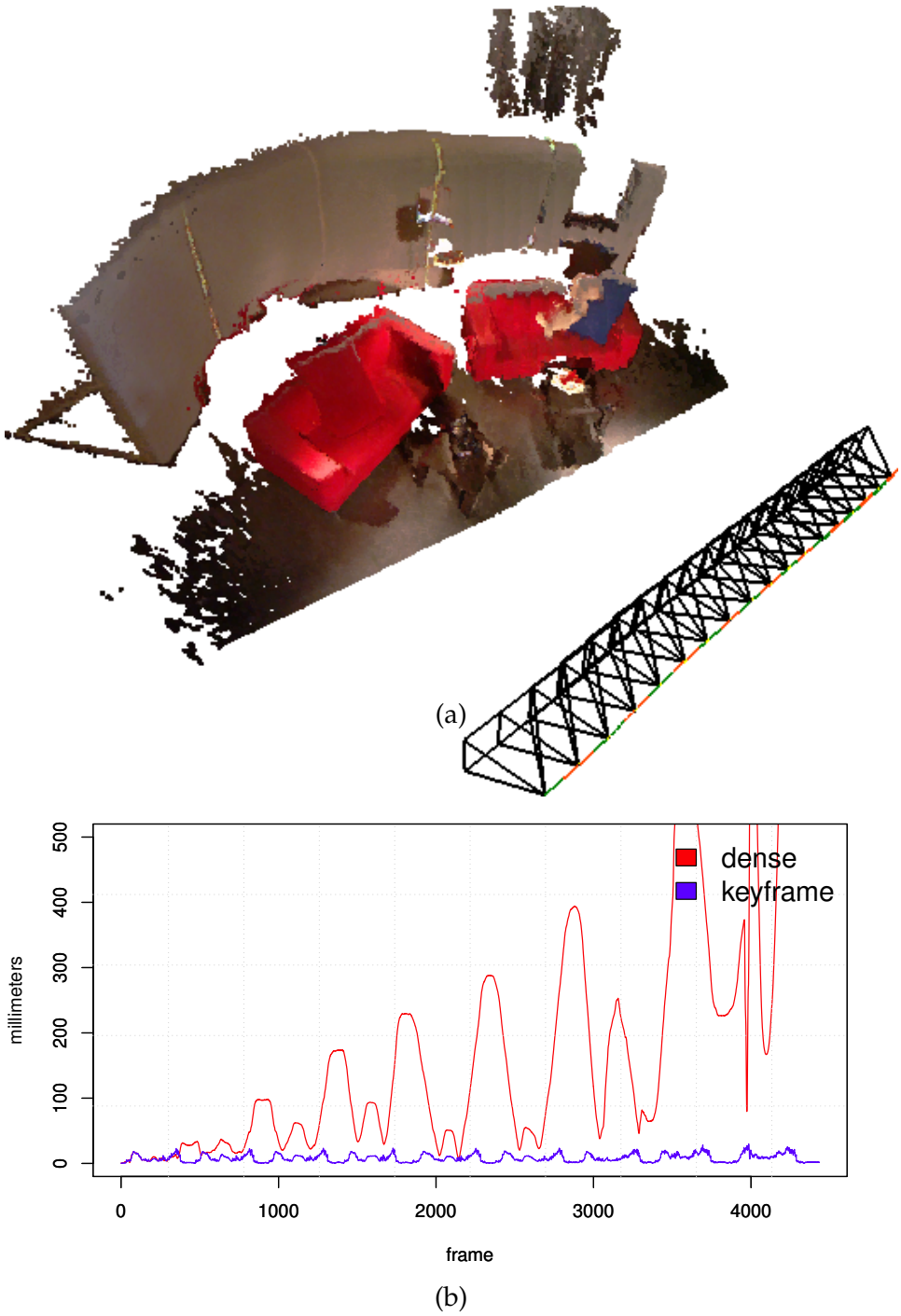


Figure 8.13: *a) A rapid 3D scene reconstruction is made by moving a camera along a fixed 3.30m studio rail. b) A comparison between dense tracking and keyframe tracking. In dense tracking drift increases in time, but keyframe tracking maintains small bounded error. A person is moving in the scene during the last cycles.*

these edges are also associated to false depth values and must be filtered out using the scheme proposed in Section 7.1. Depth map inaccuracy introduces false edge motion which can increase bias during estimation.

Figure 8.14 shows how the online tracking accuracy depends on the number of keyframes. The sequence is illustrated in Figure 8.1, but keyframe switching error are quantified in an empty scene. The ground truth is generated by applying bundle adjustment to 27 keyframes which are initialized by dense tracking. Keyframe tracking is then executed with a sparse number of ground truth keyframes (14, 9, 7, 5) by skipping a number of keyframes along the trajectory. The camera pose is compared in each frame to the corresponding ground truth pose. A small number of keyframes produce local drift which shows as error ramps between the keyframes. This can be visually disturbing. With a sufficient number of keyframes and sufficient scene texturing, the error remains small. In longer sequences, keyframe pose error finally depends on the bundle adjustment accuracy. Bundle adjustment removes global error but easily introduces local variance to the key poses if the 2D point correspondences are not precise. With the demonstration video, 14 keyframes produce small switching effects. Even more precise ground truth could be obtained by varying poses and evaluating image-based consistency globally. This phase is avoided, because the experiments are carried out with a laptop, whose computational capacity is limited.

§ 8.6 CONSTRAINTS

If the camera is moving too quickly, the minimization may not converge, because we use local optimization strategy. Global minimization strategies of the cost function are not discussed in this work because they are computationally too expensive to operate in real-time. Therefore, to avoid relocalization needs, we must assume limited camera speed, sufficient scene texturing, sufficient keyframe density, and that a sufficient amount of selected keyframe points are visible despite the occluding actors.

§ 8.7 SUMMARY

In this work, an affordable real-time matchmoving solution was developed, which can be used to produce broadcasts with interactive digital components, such as virtual characters and stage items. The solution performs in real-time using a RGB-D sensor, and a laptop with a low-end GPU (NVS4200m). This system was able to accurately and robustly track a camera in broadcasting studio sized operating volumes which are too big for voxel based approaches. Drift does not exist as the camera is tracked

relative to the nearest keyframe. The pose error without bundle adjustment is space-evolving, because the distant keyframes will contain more error. The keyframes were generated using GPU-enhanced dense tracking, and fine-tuned, when necessary, using sparse bundle adjustment. A M-estimator was enhanced by segmentation-based weights, which allows actors to move in the foreground while tracking the camera. When operating in the RGB-D sensor range, our pose estimation accuracy depends mostly on the texturing, which is trivial to increase in studio environments. On the other hand, when pointing RGB-D sensor towards the floor, a green-screen can be used. Experimentation was also done how textured carpet works when rendering fully virtual scene from the correct camera angle. Camera tracking has been demonstrated in a real broadcast studio with and without a dynamic foreground. Drift and keyframe switching errors have also been quantified. Future work will address the practical issues of how studio staff and cameramen can use this computer vision system in live broadcasts.

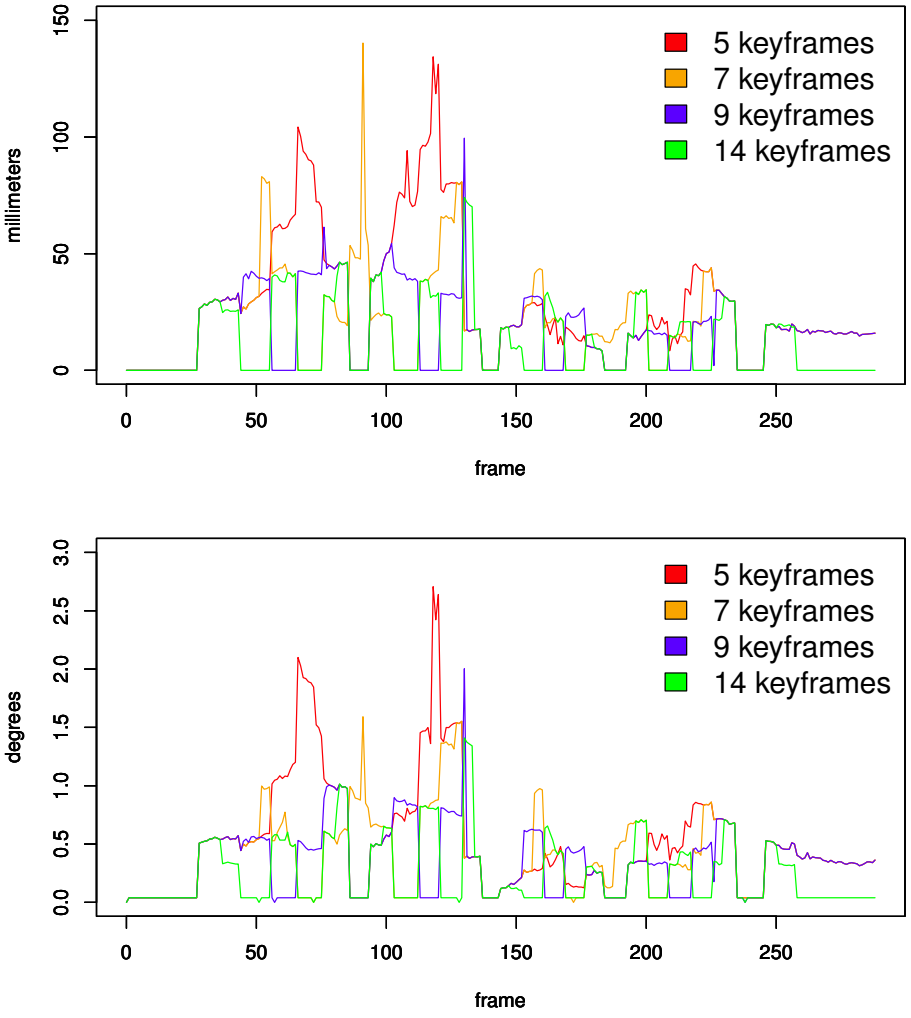


Figure 8.14: *The translation and rotation error as a function of keyframe count. The sequence is illustrated in Figure 8.1 and the ground truth is generated by applying bundle adjustment technique for maximal amount of keyframes. Keyframe switching error depends on the amount of keyframes. At least 14 keyframes are required to obtain small keyframe switching error. Inconsistent keyframe geometry and local errors in bundle adjustment produce ramps.*

- CHAPTER 9 -

CONCLUSIONS

The presented work covered many theoretical and practical aspects of RGB-D tracking and reconstruction. Photometric cost functions were formulated and efficiently minimized in different use cases. The accuracy and robustness of theses approaches was verified using real and simulated sequences with ground truth data. Temporal correspondence was elegantly expressed via a photometric cost function and 3D structure was also photometrically refined within the estimated depth bounds. Finally, an efficient GPU implementation was developed, which minimizes photometric pose error in real-time. Due to Lambertian assumption, photometric cost functions are effective within a short temporal window. The precision is good because they minimize true sensor error and the raw measurements need not to be simplified into feature points. Image based refinement methods can replace and enhance the initial estimates obtained using traditional geometrical estimation methods. Overall, this project became a path from photometric cost function definitions to real-time and online systems, whose accuracy and practical aspects were made visible using various sequences.

In publication (i) [82], a photometric cost function was presented, which measures the fitness of 3D pose and dense disparity map with given stereo images. Disparity variance based bounds were derived and propagated in frame-to-frame basis to speed-up disparity map generation. The formulation itself is mathematically sound, but problems arise in the practical application setting (Section 4.5). Generating dense depth maps with sufficient precision is difficult, because two satellites are required (see Appendix), and also target texturing and compatibility with the chosen dense matching method can not be verified without a real sequence. Also monocular SLAM techniques exist, but they require special initialization phase, and depend heavily on the optical flow quality. Based on the available asteroid imaginary, it is possible that dense and precise optical flow could be extracted from the sequences, but real sequences are necessary to test the

idea further. If the optical flow is not dense and the images contain insufficient texturing, the 3D surface must be guessed based on the smoothness assumption [52]. Despite these practical problems, the proposed cost function provides novel insight on how joint estimation of 3D pose and structure is possible using a photometric cost function and also how to avoid unnecessary disparity map computation.

In publication (ii) [81], 2.5D maps were represented as general RGB-D measurements without depending on a specific measuring technique. Both RGB and depth map discrepancies were concurrently minimized to reduce drift. This novel formulation is more robust than purely image based approach in environments with homogeneous texturing. Also novel histogram based saliency selection was proposed to reduce computational requirements in pixel selection from $O(n\log n)$ to $O(n)$,

Publications (iii) [84] and (iv) [80], focus on developing practical and real-time tools for indoor mapping and augmented reality using a RGB-D sensor. Camera tracking solution was developed for a television production studio, which uses photometric minimization for pose estimation. A procedure was proposed to build a 3D keyframe model of the static studio, which then provides a fixed reference for driftless camera tracking. Keyframe model is post-processed in offline to increase depth map and pose quality. An offline model is also useful when designing AR interaction prior to the broadcast. Many computational enhancements were required to be able to execute the system in real-time using the available low-end GPU. A scalable GPU implementation was developed, which implements various optimizations, but does not compromise tracking accuracy. Designing a real-time tracking system is by nature more reductive than incremental in terms of new functionalities, because care has to be taken that 30Hz implementation is actually obtained using the available hardware. In this project, the hardware has been more or less light-weight since all experiments have been carried out using a laptop with a low-end GPU. All tools used to experiment, validate, test, visualize and understand the characteristics of photometrical estimation methods have been developed from the scratch. The tracking and mapping methods were evaluated both in television production studio and in an apartment (Section 7). High tracking precision was maintained in indoor environments and even foreground actors were tolerated.

§ 9.1 PERSPECTIVES

This section discusses the remaining problems which could be addressed in future work. In larger environments, the incremental keyframe generation will produce spatially-evolving error, which finally must be corrected using a global technique. One correction solution is traditional SBA. SBA

uses projection point associations in multiple views, which are tedious to generate in environments with homogeneous texturing. More automatic pose correction methods should be developed. One option would be to initialize the configuration using the current approach and proceed with automatic but bounded feature extraction and matching [59]. That would eliminate gross feature matching problems. After traditional bundle adjustment, final refinement could be done by photometric bundle adjustment. Some novel techniques have been recently developed for global keyframe optimization. For example `g2o` framework developed for graph optimization tasks [36] has been used for automatic and global keyframe optimization [72].

The re-localization problem is still to be solved. Re-localization is necessary if the camera speed or occluded image region increases too much. Image-based re-localization is very difficult in general environments, because nothing guarantees a unique mapping between a current view and an available model. A feature vector histogram database (*bag-of-words*) has been used to identify the current image from tens of thousands of images [16]. In our studio applications case, natural re-localization scheme is to initialize the current camera pose using an external tracking system since NaturalPoint Optitrak is available. Recent developments in re-localization are reported by Shotton *et al.* [66].

In the current implementation, keyframes are stored uniformly in the angle and position spaces. In online use, keyframe switching may sometimes cause small ramps, which could be avoided if the local keyframe density would be selected appropriately. Comport *et al.* observe Median absolute deviation (MAD) of the error residual to determine when the reference should be changed. Appearance changes are not uniform when a keyframe is approached in different directions. Thus, a mechanism to optimize keyframe amount to its minimum would be useful, which takes into account various incoming directions.

The Microsoft Kinect is imprecise with specular surfaces and image regions with external IR light (Figure 8.11). Gross reconstruction errors may occur when the environment is not fully compatible with the device. This is a problem especially when storing keyframe points, because invalid geometry implies tracking problems. Currently a depth map fusion phase helps in filtering out points whose appearance does not vary solely as a function of the camera pose. This removes a large amount of outliers, but the model should be qualitatively verified in a model editor, which supports manual outlier removal.

In live AR application, which uses a textured carpet, a calibration tool should be developed which produces exact calibration between the RGB-D sensors oriented towards the floor and the TV camera oriented towards the scene.

- CHAPTER 10 -

RÉSUMÉ EN FRANÇAIS

En matière de vision artificielle, on approche l'estimation de la pose des caméras et l'affinement des structures 3D en définissant des fonctions de coût dont les minima correspondent à la configuration de modèle désirée. Ces deux problèmes étant par nature géométriques, les mesures de l'image originale sont souvent remplacées par des points caractéristiques 2D extraits des images. Les points caractéristiques sont des régions locales particulières, comme les coins et les arêtes, que l'on peut faire correspondre d'une image à l'autre. En fin de compte, une fonction de coût géométrique détermine les paramètres idéaux par rapport à toutes les observations de points caractéristiques 2D. Si les données de coût géométrique permettent d'obtenir des estimations précises, le processus d'estimation n'utilise paradoxalement qu'indirectement les données de l'image originale. Les points 2D doivent être extraits des images et une correspondance doit être établie entre eux dans des vues multiples. C'est là le maillon faible du processus d'estimation, car il est impossible d'éviter les erreurs d'extraction et de correspondance. Les erreurs de correspondance, en particulier, doivent être résolues par des techniques telles que RANSAC et les M-estimateurs, pour augmenter la tolérance aux aberrations évidentes. L'extraction de points caractéristiques est elle aussi imprécise, car les points sont probablement décalés par rapport à la projection idéale d'un point 3D. Par exemple, l'angle de vue et les conditions d'éclairage influencent le processus d'extraction des points caractéristiques.

Dans ce travail, les méthodes d'estimation basées sur l'image – aussi appelées *méthodes directes* – étudiées évitent complètement l'extraction de points caractéristiques et l'établissement de correspondance entre eux. Les fonctions de coût sont directement définies à l'aide de mesures brutes en pixels, et le but est d'obtenir des estimations précises de pose de caméra et

de structure. Les fonctions de coût présentées minimisent l'erreur de capteur, car les mesures ne sont pas transformées ni modifiées. En ce qui concerne l'estimation de la pose des caméras photométriques, les paramètres de rotation et de translation 3D sont estimés en minimisant une séquence de fonctions de coût basées sur l'image. Ces fonctions de coût sont non linéaires, à cause de la projection en perspective et de la distorsion de l'objectif. En ce qui concerne l'affinement de structure basé sur l'image, la structure 3D est affinée en minimisant l'écart d'apparence par rapport à la vue de référence, en utilisant des vues supplémentaires. Les méthodes d'estimation basées sur l'image peuvent être utilisées lorsque l'hypothèse lambertienne d'éclairage se vérifie, les points 3D ayant une couleur constante quel que soit l'angle de vue. Les principaux domaines d'application considérés dans ce travail sont la reconstruction d'intérieurs, la robotique et la réalité augmentée, pour lesquels plus de précision en matière de suivi de caméra 3D et de mappage d'environnement est un avantage.

L'objectif global du projet est d'améliorer les méthodes d'estimation basées sur l'image et de permettre leur utilisation dans le cadre d'applications en temps réel. Les principales questions relatives à ce travail sont :

- *« Qu'est-ce qu'une formulation efficace pour la pose et la structure 3D tâche d'estimation basée sur des images? ? »*
- *« Comment organiser le calcul pour permettre une implémentation en temps réel ? »*
- *« Quelles considérations pratiques s'appliquent à l'utilisation des méthodes d'estimation basées sur l'image dans des applications telles que la réalité augmentée et la reconstruction 3D ? »*

Deux cas d'application sont considérés. Le premier concerne une mission de reconstruction d'un astéroïde, avec l'utilisation d'une inférence multivue pour l'estimation autonome de la pose et la reconstruction 3D. Les deux premières publications décrivent les développements réalisés dans le laboratoire I3S de Sophia- Antipolis dans le but d'appliquer la vision artificielle de pointe aux applications spatiales, en collaboration avec Thales-Alenia Space. L'efficacité du calcul et la robustesse du suivi ont particulièrement été prises en considération. L'évaluation comparative standard pour l'odométrie visuelle n'étant pas disponible, elle a été développée, afin de suivre le développement en termes de précision et de robustesse. Le logiciel Blender a été utilisé pour générer des ensembles de données synthétiques avec des trajectoires vérité terrain. Une plaque tournante a également été utilisée pour tester l'odométrie visuelle avec des données plus réalistes.

En cours de projet, Microsoft a lancé son capteur Kinect, ce qui a permis de capturer des mesures RGB-D assez précises dans des environnements

intérieurs sans texture à 30 Hz, à l'aide d'un motif lumineux IR structuré. Par rapport aux cartes de profondeur générées à partir de caméras stéréo, la précision structurale est devenue beaucoup plus régulière et indépendante du contenu de l'image. J'ai fait la demande de transformer ce projet de doctorat en un double diplôme avec l'Université de technologie de Lappeenranta, afin de pouvoir me concentrer sur les environnements des studios de production télévisuelle. Ce changement a simplifié l'acquisition de données, puisqu'il était alors possible de capturer facilement des données réelles dans l'environnement d'application réel.

L'objectif de la deuxième partie de cette thèse consistait à développer une solution de suivi de caméra 3D pour les studios de production télévisuelle qui permette de calculer les rendus des images de réalité augmentée sur les diffusions en direct. L'industrie du cinéma connaît cette technique sous le nom de *matchmoving*. Elle s'effectue habituellement en postproduction à l'aide de trackers semi- automatiques [17]. Un tracker est un outil qui estime la trajectoire de mouvement d'une caméra 3D d'après des tracés de points 2D [85, 86]. Dans les studios, le matchmoving en temps réel s'effectue habituellement à l'aide d'un système externe de capture des mouvements [49], qui suit une caméra en temps réel en se basant sur des repères passifs fixés à la caméra. Le logiciel Brainstorm Multimedia, par exemple, est alors utilisé pour calculer en temps réel le rendu des images dépendantes de la vue [7]. Bien que la capture des mouvements soit précise et permette de travailler sur un grand volume, le prix total d'un système professionnel est actuellement de 200000 à 500000 €. Des systèmes moins chers existent, comme NaturalPoint Optitrak [51], qui coûte environ 25000 €, mais leur précision et leur cadence de capture sont plus faibles.

Un tracker en temps réel basé sur l'image a été développé pour offrir un suivi automatique de caméra économique, que l'université KyAMK de sciences appliquées de Kouvola pourrait utiliser à des fins éducatives. Le projet a été financé par le Fonds européen de développement régional et la Fédération des industries technologiques finlandaises. Une implémentation en temps réel était nécessaire et aucune dérive n'était tolérée. Pour cela, une méthode d'acquisition de modèle 3D devait être développée pour obtenir une référence fixe. Une certaine tolérance était nécessaire par rapport aux acteurs en premier plan, dans une configuration dans laquelle le contenu du programme était directement utilisé pour estimer la pose de caméra. Une approche alternative consistait à pointer un capteur RGB-D vers un tapis texturé pour éviter toute manipulation de la scène. Une expérimentation intense a eu lieu en environnement de studio, en collaboration avec le personnel du studio, afin d'obtenir un système utilisable. L'ensemble d'outils a été testé avec une grande variété de vidéos de synthèse et réelles comme sources. Enfin, la solution de suivi a été expérimentée sur des programmes télévisuels avec des contenus de réalité augmentée en direct et en reconstruction 3D d'intérieurs.

§ 10.1 CONTRIBUTIONS ET PUBLICATIONS

Les publications suivantes ont été écrites au cours du projet :

Publication (i)
"A Dense Structure Model for Image Based Stereo SLAM", T. M. Tykkälä, and A.I. Comport, IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, May 2011,

Publication (ii)
"Direct Iterative Closest Point for Real-time Visual Odometry", T. M. Tykkälä, C. Audras, and A. I. Comport, Workshop on Computer Vision in Vehicle Technology: From Earth to Mars in conjunction with the International Conference on Computer Vision (CVVT/ICCV), Barcelona, Spain, Nov 2011.

Publication (iii)
"RGB-D Tracking and Reconstruction for TV Broadcasts", T.M. Tykkälä, H. Hartikainen, A.I. Comport, and J-K. Kämäräinen. 8th International Conference on Computer Vision Theory and Applications (VISAPP), Barcelona, Spain, Feb 2013.

Publication (iv)
"Live RGB-D Camera Tracking for Television Production Studios", T.M. Tykkälä, A.I. Comport, J-K. Kämäräinen and H. Hartikainen. Journal of Visual Communication and Image Representation, Elsevier, Apr 2013.

10.1.1 Publication I

Dans la publication (i), l'estimation simultanée de la pose 3D et des paramètres de structure est formulée à l'aide d'une fonction de coût SLAM basée sur l'image [82]. Une telle fonction de coût peut être utilisée en navigation autonome et pour le mappage d'environnements spatiaux. Au cours d'une récente mission d'échantillonnage sur l'astéroïde Itokawa, des images de l'astéroïdes ont été collectées par un robot autonome. Ces images ont ensuite été utilisées dans le cadre de la reconstruction manuelle en 3D de l'astéroïde Itokawa. Dans notre travail, les valeurs de disparité sont traitées comme des paramètres libres dont la covariance liée aux paramètres de pose est estimée. Les écarts de disparité basés sur l'image sont obtenus en marginalisant les paramètres de mouvement. Ces écarts sont alors utilisés pour limiter l'estimation de la carte de disparité suivante. Ce processus accélère le calcul de la carte de disparité, qui constitue souvent un goulot d'étranglement. Le sujet a été étudié en collaboration avec

Thales-Alenia Space, qui a financé une partie de ce travail et fourni une trajectoire d'orbite réaliste autour d'Itokawa. Les trajectoires et des modèles 3D d'astéroïde disponibles ont été utilisés pour évaluer la méthode.

10.1.2 Publication II

Dans la publication (ii), des cartes 2,5D sont représentées directement comme des mesures RGB-D générales, indépendamment d'une technique de mesure spécifique [81]. Il est essentiel pour tout problème d'odométrie visuelle de maintenir une dérive aussi faible que possible. Pour réduire la dérive, les différences de l'image RVB et de la carte de profondeur sont simultanément minimisées. Lorsque l'erreur résiduelle consiste en deux composants avec des unités différentes, l'incertitude numérique (les écarts types) peut également varier d'un composant à l'autre. Cela signifie qu'un paramètre d'ajustement supplémentaire λ est nécessaire pour compenser les incertitudes de telle manière que la dérive soit minimisée. Pour améliorer l'efficacité de calcul, il est proposé de sélectionner les régions significatives de l'image à l'aide d'un histogramme de magnitude de gradient. La méthode avec histogramme réduit les exigences de calcul pour la sélection de pixels de $O(n\log n)$ à $O(n)$, car on peut éviter totalement le tri. Les expériences montrent que l'estimation de la pose peut être plus précise, surtout lorsque l'hypothèse lambertienne ne se vérifie que partiellement. La méthode est également démontrée à l'aide d'une séquence spatiale réelle du PRoVisG MARS 3D Challenge [29].

10.1.3 Publication III

Dans la publication (iii), un système utilisant le capteur Microsoft Kinect pour insérer des éléments de réalité augmentée dans une télédiffusion en direct est présenté. La réalité augmentée en direct est utile pour concevoir des scènes télévisuelles et cinématographiques qui contiennent des composants virtuels interactifs. Les éléments de réalité augmentée sont habituellement ajoutés en post-production. Si plusieurs prises sont nécessaires, le processus peut facilement revenir cher. Avec un système de réalité augmentée en direct, les scènes peuvent être répétées en temps réel avec une qualité de réalité augmentée suffisante. Au moment où les scènes sont enregistrées dans leur forme définitive, une seule étape de post-production est nécessaire. Le suivi de caméra RGB-D est proposé aux studios de télévision pour utiliser la réalité augmentée en direct, afin de réduire les besoins de post-production et d'abaisser les coûts [84]. Lors d'une diffusion en direct avec réalité augmentée, la dérive est interdite et une fermeture de boucle fréquente est nécessaire. Une méthode d'implémentation de fermeture de boucle consiste à utiliser des images clés. Lorsque le suivi est défini relativement à un ensemble d'images clés, la dérive ne se cumule pas

dans le temps. Un modèle d'image clé prédéfini est également nécessaire lorsqu'il faut éviter les acteurs en mouvement dans la scène. Le modèle statique offre des valeurs de distance et des valeurs d'intensité, qui sont comparées à la mesure RGB-D courante. En filtrant les régions d'image présentant une trop grande différence avec le modèle, le mouvement en premier plan n'influence pas le processus de suivi. Les images clés ne peuvent pas contenir de régions dynamiques, car les modifications d'apparence sont uniquement modélisées par modèle de caméra mathématique.

10.1.4 Publication IV

Dans la publication (iv), un tracker RGB-D en temps réel disposant de la capacité de calcul et de l'extensibilité d'une unité centrale graphique est développé [80]. Cette implémentation est l'une des premières pour une unité centrale graphique. Plusieurs optimisations d'unité centrale graphique permettant un fonctionnement en temps réel, même avec une unité centrale graphique bas de gamme, sont présentées et décrites. Le système est conçu pour ne pas coûter cher, et il ne nécessite qu'un capteur RGB-D et un ordinateur portable. Le suivi RGB-D est expérimenté dans un studio de production télévisuelle, avec et sans acteurs au premier plan. De nouveaux outils sont créés pour préfiltrer et affiner les modèles 3D générés par le suivi RGB-D. La précision est comparée à un système KinectFusion récent, qui assure simultanément le suivi de la pose de caméra 3D et génère un modèle 3D à base de voxels [53]. La comparaison montre en quoi la méthode proposée est plus performante que KinectFusion pour un grand volume de travail. L'utilisation comme référence d'un modèle 3D basé sur les images clés nécessite une métrique du plus proche voisin pour mesurer la similarité de pose à l'aide d'une seule métrique. Une métrique qui unifie les unités d'orientation et de translation est proposée. Cette thèse fournit également des exemples de la manière dont des modèles d'appartements photoréalistes et étanches sont générés par un processus de mappage RGB-D (Section 7). Ces modèles sont stockés dans un format standard pour permettre leur visualisation en ligne et leur impression 3D.

§ 10.2 CONCLUSIONS ET PERSPECTIVES

Le travail présenté a couvert de nombreux aspects théoriques et pratiques du suivi RGB-D et de la reconstruction. Des fonctions de coût photométrique ont été formulées et minimisées efficacement dans différents cas d'utilisation. La précision et la robustesse de ces approches ont été vérifiées à l'aide de séquences réelles et simulées avec des données de vérité terrain. La correspondance temporelle a été exprimée de manière élégante via une

fonction de coût photométrique et la structure 3D a aussi été affinée photométriquement dans les limites de profondeur estimées. Enfin, une implémentation efficace d'unité centrale graphique minimisant l'erreur de pose photométrique en temps réel a été développée. Étant donné l'hypothèse lambertienne, les fonctions de coût photométrique ne sont valides que dans un court intervalle de temps. La précision est bonne, car ces fonctions minimisent l'erreur réelle des capteurs et les mesures brutes ne doivent pas être simplifiées en points caractéristiques. Les méthodes d'affinement basées sur l'image peuvent remplacer et améliorer les estimations initiales obtenues à l'aide des méthodes d'estimation géométrique traditionnelles. Globalement, ce projet est devenu un lien entre les définitions de fonctions de coût photométrique et les systèmes en temps réel, dont la précision et les aspects pratiques ont été révélés à l'aide de diverses séquences.

10.2.1 Perspectives

Cette section traite des problèmes restants, qui doivent être abordés dans un travail futur. Dans des environnements plus grands, la génération incrémentale d'images clés produit une erreur évoluant dans l'espace, qui doit au bout du compte être corrigée à l'aide d'une technique globale. Une solution corrective traditionnelle est SBA. SBA utilise les associations de points de projection dans des vues multiples, qui sont fastidieuses à générer dans les environnements présentant une texture homogène. D'autres méthodes de correction automatique de pose devraient être développées. Une option consisterait à initialiser la configuration à l'aide de l'approche actuelle et de continuer avec l'extraction automatique mais limitée de caractéristiques et leur mise en correspondance [59]. Ceci éliminerait les problèmes flagrants de correspondance des caractéristiques. Après l'ajustement groupé habituel, un affinement définitif pourrait être effectué par ajustement photométrique groupé. De nouvelles techniques ont récemment été développées pour l'optimisation globale des images clés. Par exemple, l'infrastructure `g2o` développée pour les tâches d'optimisation graphique [36] a été utilisée pour l'optimisation automatique et globale des images clés [72].

Le problème de relocalisation reste à résoudre. La relocalisation est nécessaire si la vitesse de la caméra ou si la région cachée de l'image devient trop importante. La relocalisation basée sur l'image est très difficile dans les environnements généraux, car rien ne garantit un mappage unique entre une vue courante et un modèle disponible. Une base de données d'histogrammes de vecteurs de caractéristiques (*bag-of-words*) a été utilisée pour identifier l'image courante parmi des dizaines de milliers d'images [16]. Dans notre cas d'applications pour studio, le procédé naturel de relocalisation consiste à initialiser la pose courante de la caméra à l'aide d'un système de suivi externe, puisque NaturalPoint Optitrak est

disponible. Les développements récents en matière de relocalisation sont rapportés par Shotton *et al.* [66].

Dans l'implémentation courante, les images clés sont stockées uniformément dans les espaces d'orientation et de position. Pour une utilisation en temps réel, le changement d'image clé peut parfois provoquer de légers changements de vitesse, ce qui peut être évité en sélectionnant la densité d'image clé appropriée. Comport *et al.* observent l'écart médian absolu de l'erreur résiduelle pour déterminer le moment auquel la référence doit être changée. Lorsqu'une image clé est approchée dans différentes directions, les changements d'apparence ne sont pas uniformes. Pour cette raison, un mécanisme de minimisation de la quantité d'images clés prenant en compte différentes directions d'approche serait utile.

Microsoft Kinect n'est pas précis pour les surfaces spéculaires et les régions de l'image soumises à une lumière IR (Figure 8.11). De grossières erreurs de reconstruction peuvent se produire lorsque l'environnement n'est pas entièrement compatible avec le dispositif. Cela constitue un problème, surtout lors du stockage de points d'images clés, car une géométrie non valide implique des problèmes de suivi. À l'heure actuelle, une phase de fusion de la carte de profondeur aide à filtrer les points dont l'apparence ne varie pas uniquement en fonction de la pose de la caméra. Ceci supprime une grande quantité d'aberrations, mais le modèle doit être vérifié qualitativement dans un éditeur de modèle prenant en charge la suppression manuelle des aberrations.

Pour l'application de réalité augmentée en direct, qui utilise un tapis texturé, un outil d'étalonnage permettant un étalonnage exact entre les capteurs RGB-D orientés vers le sol et la caméra TV orientée vers la scène doit être développé.

- CHAPTER A -

APPENDIX: SIMULATED ASTEROID DATASETS

Synthetic datasets were rendered with the Blender along with ground-truth trajectory. The Eros and Itokawa 3D models were obtained from NASA and Thales-Alenia Space provided a real trajectory around Itokawa asteroid (Figure A.1). The material properties, however, could not be modeled using default Blender and precise simulation of an asteroid mission was not possible. Some softwares such as `PANGU` (Planet and Asteroid Natural Scene Generation Utility) exist for simulating Hapke BRDF with asteroid models. `PANGU` is designed by European Space Agency specifically for producing realistic imaginery of space environments, but unfortunately it was not available for this project. Therefore, general specular material was used instead as Hapke also contains some specular components. Despite the material simulation problem, the asteroid orbiting case was not natural for a stereo-based technology, because sufficient baseline would have required a very large baseline (two separate satellites). The severity of this problem was verified using Hartley-Sturm triangulation to reconstruct 3D points from noisy 2D projections whose noise distribution is Gaussian (stdev $1px$ and $0.1px$). The image resolution is assumed to be 800×600 and camera field-of-view is 49.13°. The resulting graphs illustrate the reconstruction error dependency on stereo baseline (Figure A.1). If the disparity stdev is merely $0.1px$, $200m$ baseline will be sufficient. These numbers imply that applying stereo-based tracking and mapping technology is not feasible for asteroid missions. Also monocular approaches exist [15], but producing precise dense reconstructions without explicit depth measurements is problematic. In a recent DTAM system, dense reconstruction is obtained with a monocular camera by assuming continuous surface where measurements do not exist [52].

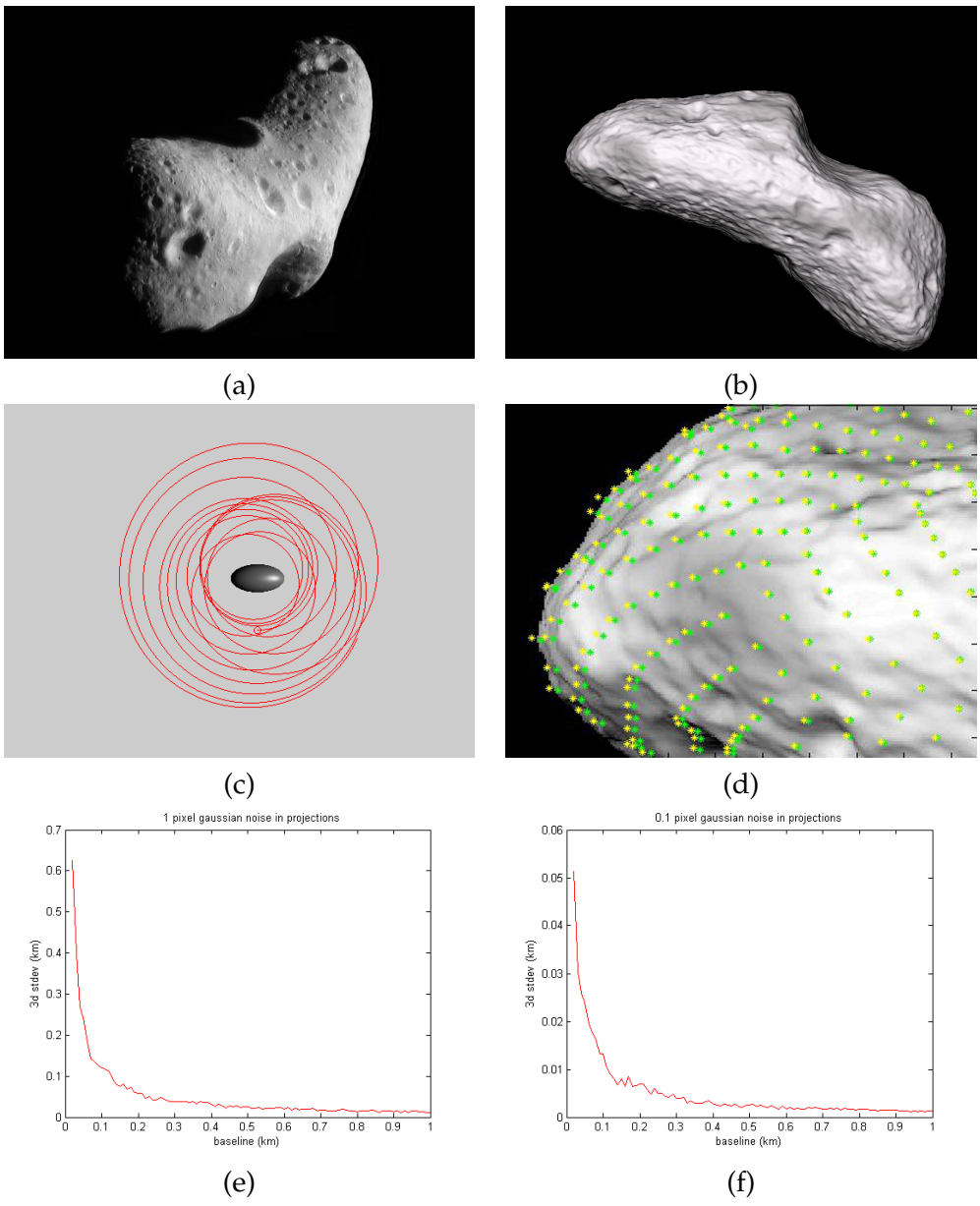


Figure A.1: *Blender generated dataset for asteroid reconstruction mission. a) Real Eros Asteroid. b) A rendered Eros using Blender, c) A real satellite trajectory around the Itokawa asteroid provided by Thales-Alenia Space, d) Convergence was verified by matching a precise target projection with an estimated projection points. The precise projections on green and photometrically estimated ones on yellow before convergence. e) Hartley-Sturm reconstruction error in kilometers as a function of stereo baseline in an asteroid orbiting mission. Assuming* $1px$ *Gaussian stdev in disparity values. Over 1km baseline is required to reduce reconstruction error into tens of meters. f) Assuming* $0.1px$ *Gaussian stdev in disparity values. 200meters is a sufficient baseline, which produces* $10m$ *standard deviation for 3D points. Calibration is exact without any noise.*

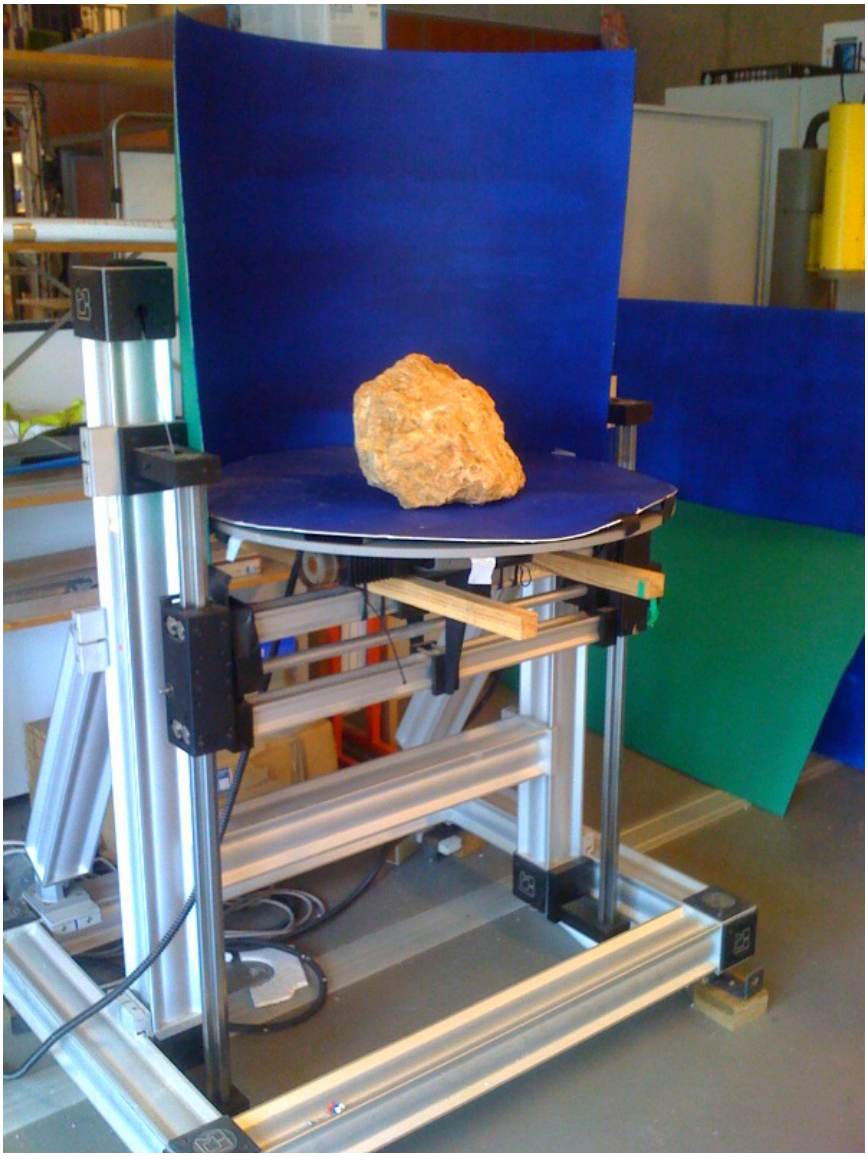


Figure A.2: *The turn-table at INRIA was used for recording the rock sequence.*

Algorithm A.1 RGB-D depth map fusion procedure.

Input: Reference RGB-D image $\{\mathcal{I}^*, \mathcal{Z}^*\}$ with depth map pose $\mathbf{T}^*$, RGB-D images $\{\mathcal{I}_k, \mathcal{Z}_k\}$, and depth map poses $\mathbf{T}_k$, where $k = \{1, 2, \ldots, n\}$. $\mathbf{T}_b$ is the baseline transform and $\mathbf{K}$ is the intrinsic matrix. B is initial mode bound, C is color threshold, G is maximum distance to pixel center ray, N is a threshold to control statistical reliability.

Output: Filtered $\mathcal{Z}_f$ and stdev image σ

$\mathbf{A} \Leftarrow \emptyset$
for all $k = \{1, 2, \ldots, n\}$ **do**
  for all $v = \{1 \ldots \text{height}\}$ **do**
    for all $u = \{1 \ldots \text{width}\}$ **do**
      $\mathbf{p} = (u, v, 1)^T$
      $\mathbf{P} \Leftarrow \mathcal{Z}_k(u, v)\mathbf{K}^{-1}\mathbf{p}$ # reconstruct a point
      $\mathbf{c} \Leftarrow \mathcal{I}_k(w(\mathbf{T}_b, \mathbf{P}))$ # sample color from associated RGB image
      $\mathbf{P}^* \Leftarrow \mathbf{T}^*\mathbf{T}_k^{-1}\mathbf{P}$ # warp point into reference frame
      if $\|\mathbf{c} - \mathcal{I}^*(w(\mathbf{T}_b, \mathbf{P}^*))\| \geq C$ **then**
        continue # Lambertian assumption does not hold, next point
      end if
      $\mathbf{p}^* \Leftarrow \text{round}(\mathbf{K}N(\mathbf{P}^*))$ # project and discretise into integers
      $\mathbf{r}^* \Leftarrow \text{normalize}(\mathbf{K}^{-1}\mathbf{p}^*)$ # generate a unit ray through pixel center
      if $\|\mathbf{P}^* - (\mathbf{P}^* \cdot \mathbf{r}^*)\mathbf{r}^*\| \geq G$ **then**
        continue # point too distant from ray
      end if
      $\mathbf{A}(\mathbf{p}^*) \Leftarrow \{\mathbf{A}(\mathbf{p}^*), (0, 0, 1) \cdot \mathbf{P}^*\}$ # collect z measurement
    end for
  end for
end for
for all $\mathbf{p} \in \{\text{width} \times \text{height}\}$ **do**
  if $\#\mathbf{A}(\mathbf{p}) < N$ **then**
    $\mathcal{Z}_f(\mathbf{p}) \Leftarrow 0$, $\sigma(\mathbf{p}) \Leftarrow 0$, continue; # not enough statistical support
  end if
  $\mathcal{Z}_m \Leftarrow \text{median}(\mathbf{A}(\mathbf{p}))$ # calculate median z
  $\Omega \Leftarrow \text{listIndices}(\|\mathbf{A}(\mathbf{p}) - \mathcal{Z}_m\| \leq B)$ # enumerate z samples near median
  $\mathcal{Z}_f(\mathbf{p}) \Leftarrow 1.0/(\ \frac{1}{N}\sum 1.0/\mathbf{A}_\Omega(\mathbf{p})\ )$ # local average using inverted values
  $\sigma(\mathbf{p}) \Leftarrow \sqrt{\frac{1}{N}(\mathcal{Z}_f(\mathbf{p}) - \mathbf{A}_\Omega(\mathbf{p}))^2}$ # determine stdev image
end for

Algorithm A.2 Depth refinement using photometric minimization.

Input: Reference RGB-D $\{\mathcal{I}^*, \mathcal{Z}^*\}$, stdev image $\boldsymbol{\sigma}^*$, reference depth map pose is $\mathbf{T}^*$. gradient mask $\mathbf{M}^*$, RGB images $\mathcal{I}_k$ with poses $\mathbf{T}_k$, where $k = \{1, 2, \ldots, n\}$. baseline matrix $\mathbf{T}_b$, intrinsic matrix $\mathbf{K}$, number of discrete candidates N.

Output: Photometrically refined $\mathcal{Z}_f$

for all $v = \{1 \ldots \text{height}\}$ **do**
  for all $u = \{1 \ldots \text{width}\}$ **do**
    if $\mathbf{M}(u, v) > 0$ **then**
      # enumarate discrete depth candidates in depth range
      $z_0 \Leftarrow \mathcal{Z}^*(u, v) - \boldsymbol{\sigma}^*(u, v)$, $z_1 \Leftarrow \mathcal{Z}^*(u, v) + \boldsymbol{\sigma}^*(u, v)$
      $\mathbf{S}(u, v, p) = z_0 + p * (z_1 - z_0)/N$, where $p = \{1, 2, \ldots, N\}$
    end if
  end for
end for
$\mathbf{C} = \emptyset$
for all $k = \{1, 2, \ldots, n\}$ **do**
  $\mathbf{T} = \mathbf{T}_k(\mathbf{T}^*)^{-1}$
  for all $v = \{1 \ldots \text{height}\}$ **do**
    for all $u = \{1 \ldots \text{width}\}$ **do**
      if $\mathbf{M}(u, v) > 0$ **then**
        # increment cost terms to the kth image
        for all $p = \{1 \ldots N\}$ **do**
          $\mathbf{P} \Leftarrow \mathbf{S}(u, v, p)\mathbf{K}^{-1}(u, v, 1)^T$ # reconstruct a point
          $\mathbf{C}(u, v, p) = \mathbf{C}(u, v, p) + \|\mathcal{I}_k(w(\mathbf{T}, \mathbf{P})) - \mathcal{I}^*(u, v)\|$
        end for
      end if
    end for
  end for
end for
for all $v = \{1 \ldots \text{height}\}$ **do**
  for all $u = \{1 \ldots \text{width}\}$ **do**
    if $(\mathbf{M}(u, v) > 0)$ **then**
      # avoid border samples (the minimum not within the bounds?)
      bestIndex $\Leftarrow \underset{p}{\text{argmin}}\ \mathbf{C}(u, v, p))$
      if $1 <$ bestIndex $< N$ **then**
        $\mathcal{Z}_f(u, v) \Leftarrow \mathbf{S}(u, v, \text{bestIndex})$ # refine depth
      end if
    end if
  end for
end for

BIBLIOGRAPHY

[1] AUDRAS, C., COMPORT, A. I., MEILLAND, M., AND RIVES, P. Real-time dense rgb-d localisation and mapping. *In Australian Conference on Robotics and Automation. Monash University, Australia, 2011* (2011).

[2] BAKER, S., AND MATTHEWS, I. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision 56*, 3 (Feb. 2004), 221–255.

[3] BANZ, C., HESSELBARTH, S., FLATT, H., BLUME, H., AND PIRSCH, P. Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga-implementation. In *Embedded Computer Systems (SAMOS), 2010 International Conference on* (2010), pp. 93–101.

[4] BAY, H., ESS, A., TUYTELAARS, T., AND VAN GOOL, L. Surf: Speeded up robust features", computer vision and image understanding. *Computer Vision and Image Understanding 110*, 3 (2008), 346–359.

[5] BOISSONNAT, J.-D., DEVILLERS, O., PION, S., TEILLAUD, M., AND YVINEC, M. Triangulations in CGAL. *Comput. Geom. Theory Appl. 22* (2002), 5–19.

[6] BOUGUET, J.-Y. Camera calibration toolbox for Matlab. Referred Nov 4th, 2010. `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`.

[7] BRAINSTORM MULTIMEDIA. Brainstorm products for managing ar graphics at tv studios. Referred Apr 28th, 2013. `http://www.brainstorm.es/live`.

[8] BROWN, D. The Bundle Adjustment—Progress and Prospects. *Int. Archives of Photogrammetry 21*, 3 (1976), 3–33.

[9] BUTLER, D. A., IZADI, S., HILLIGES, O., MOLYNEAUX, D., HODGES, S., AND KIM, D. Shake'n'sense: reducing interference for overlapping

structured light depth cameras. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems* (Austin, Texas, USA, 2012), CHI '12, ACM, pp. 1933–1936.

[10] CALONDER, M., LEPETIT, V., STRECHA, C., AND FUA, P. Brief: binary robust independent elementary features. In *Proceedings of the 11th European conference on Computer vision: Part IV* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 778–792.

[11] CARNEGIE MELLON UNIVERSITY. Panda 3D engine. Referred Apr 28th, 2013. `http://www.panda3d.org`.

[12] CHANG, C., CHATTERJEE, S., AND KUBE, P. A Quantization Error Analysis for Convergent Stereo. In *International Conference on Image Processing (ICIP)* (Austin, Texas, USA, 1994), vol. 2, pp. 735–739.

[13] CIVERA, J., DAVISON, A. J., AND MONTIEL, J. Inverse Depth Parametrization for Monocular SLAM. *Robotics, IEEE Transactions on 24*, 5 (Oct. 2008), 932–945.

[14] COMPORT, A. I., MALIS, E., AND RIVES, P. Accurate quadri-focal tracking for robust 3d visual odometry. In *IEEE International Conference on Robotics and Automation , ICRA'07* (Rome, Italy, April 2007).

[15] DAVISON, A., REID, I., MOLTON, N., AND STASSE, O. MonoSLAM: Real-time single camera SLAM. *PAMI 29* (2007), 1052–1067.

[16] DENG, J., BERG, A. C., LI, K., AND FEI-FEI, L. What does classifying more than 10,000 image categories tell us? In *Proceedings of the 11th European conference on Computer vision: Part V* (Heraklion, Crete, Greece, 2010), ECCV'10, Springer-Verlag, pp. 71–84.

[17] DOBBERT, T. *Matchmoving: The Invisible Art of Camera Tracking*. Sybex, 2005.

[18] FISCHLER, M., AND BOLLES, R. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *Workshop In DARPA Image Understanding* (University of Maryland, College Park, April 1980), pp. 71–88.

[19] GONÇALVES, T., AND COMPORT, A. I. Real-time direct tracking of color images in the presence of illumination variation. In *ICRA* (2011), pp. 4417–4422.

[20] HARALICK, R., LEE, C., OTTENBERG, K., AND NOLLE, M. Review and analysis of solutions of the three point perspective pose estimation problem. In *Int. Journal of Computer Vision* (1994), vol. 13,3, pp. 331–356.

[21] HARTLEY, R., AND ZISSERMAN, A. *Multiple View Geometry in computer vision*. Cambridge University Press, 2001.

[22] HENRY, P., KRAININ, M., HERBST, E., REN, X., AND FOX, D. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research 31*, 5 (Apr. 2012), 647–663.

[23] HERRERA, D., KANNALA, J., AND HEIKKILA, J. Joint depth and color camera calibration with distortion correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 10 (2012), 2058–2064.

[24] HIGHAM, N. J. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications 26*, 4 (2005), 1179–1193.

[25] HIRSCHMULLER, H. Stereo processing by semi-global matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 2 (2008), 328–341.

[26] HORN, B. K. P. Recovering baseline and orientation from essential matrix. *MIT AI Memo, Internal Report* (1990).

[27] HUBER, P.-J. *Robust Statistics*. Wiler, New York, 1981.

[28] IRANI, M., AND ANANDAN, P. About direct methods. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice* (London, UK, UK, 1999), ICCV '99, Springer-Verlag, pp. 267–277.

[29] JOANNEUM RESEARCH FORSCHUNGSGESELLSCHAFT, J. R. Instutute of Digital Image Processing, PRoVisG Mars Challenge, Austria. Referred Apr 4th, 2013. `http://provisg.eu/news/provisg-mars-3d-challenge`.

[30] JOHNSON, A. E., AND KANG, S. B. Registration and integration of textured 3-d data. *Image and Vision Computing 17*, 2 (1999), 135–147.

[31] KATO, H., AND BILLINGHURST, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99)* (San Francisco, USA, Oct. 1999).

[32] KAZHDAN, M., AND HOPPE, H. Screened poisson surface reconstruction. *ACM Transactions on Graphics* (2013). To Appear, Implementation: `http://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version4.51/`.

[33] KLEIN, G., AND MURRAY, D. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)* (Nara, Japan, November 2007), pp. 225–234.

[34] KONOLIGE, K., AND AGRAWAL, M. FrameSLAM: from bundle adjustment to realtime visual mappping. *IEEE Trans. on Robotics 24* (2008), 1066–1077.

[35] KOPF, J., COHEN, M. F., LISCHINSKI, D., AND UYTTENDAELE, M. Joint bilateral upsampling. In *ACM Transactions on Graphics, Proceedings of SIGGRAPH 2007* (New York, NY, USA, 2007), vol. 26, p. 96.

[36] KÜMMERLE, R., GRISETTI, G., STRASDAT, H., KONOLIGE, K., AND BURGARD, W. g2o: A General Framework for Graph Optimization. In *IEEE International Conference on Robotics and Automation , ICRA'11* (Shanghai, China, May 2011).

[37] LOURAKIS, M. A., AND ARGYROS, A. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Transactions on Mathematical Software 36*, 1 (2009), 1–30.

[38] LOW, K. Linear least-squares optimization for point-to-plane icp surface registration. In *Technical report TR04-004* (University of North Carolina, 2004).

[39] LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (November 2004), 91–110.

[40] MA, Y., SOATTO, S., KOSECKA, J., AND SASTRY, S. *An invitation to 3-D vision: from images to geometric models*, vol. 26 of *Interdisciplinary applied mathematics*. Springer, New York, 2004.

[41] MALLON, J., AND WHELAN, P. F. Precise Radial Un-distortion of Images. *ICPR2004 - 17th International Conference on Pattern Recognition Cambridge, UK, 23rd - 26th* (August 2004), 18–21.

[42] MARK, W. R., GLANVILLE, R. S., AKELEY, K., AND KILGARD, M. J. Cg: a system for programming graphics hardware in a c-like language. *ACM Trans. Graph. 22*, 3 (2003), 896–907.

[43] MARTON, Z., RUSU, R., AND BEETZ, M. On fast surface reconstruction methods for large and noisy point clouds. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on* (2009), pp. 3218–3223.

[44] MEILLAND, M., AND COMPORT, A. Super-resolution 3D Tracking and Mapping. In *IEEE International Conference on Robotics and Automation, ICRA'13* (Karlsruhe, Germany., May 6-10 2013).

[45] MEILLAND, M., COMPORT, A. I., AND RIVES, P. A spherical robot-centered representation for urban navigation. In *IEEE International Conference on Robotics and Automation, ICRA'10* (Taipei, Taiwan, 2010).

[46] MEILLAND, M., COMPORT, A. I., AND RIVES, P. Real-time dense visual tracking under large lighting variations. In *Proceedings of the British Machine Vision Conference, BMVC'11* (University of Dundee, Scotland, 2011), BMVA Press, pp. 45.1–45.11.

[47] MICHOT, J., BARTOLI, A., AND GASPARD, F. Bi-Objective Bundle Adjustment With Application to Multi-Sensor SLAM. In *International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'10)* (Paris,France, 2010).

[48] MONTIEL, J., CIVERA, J., AND DAVISON, A. Unified inverse depth parametrization for monocular slam. In *Proceedings of Robotics: Science and Systems* (Philadelphia, USA, August 2006).

[49] MOTIONANALYSIS CORPORATION. MotionAnalysis products for motion capture and live camera tracking. Referred Apr 28th, 2013. `http://www.motionanalysis.com`.

[50] NASA/USGS. High resolution color texture map of Mars, captured by Viking 26th October 2001. Referenced Apr 2013, `http://www.solarviews.com/cap/mars/marscyl1.htm`.

[51] NATURALPOINT. Optitrack motion capture system for low-cost tracking. Referred Apr 28th, 2013. `http://www.naturalpoint.com/optitrack`.

[52] NEWCOMBE, R., LOVEGROVE, S., AND DAVISON, A. DTAM: Dense Tracking and Mapping in Real-Time. In *13th International Conference on Computer Vision (ICCV'11)* (Barcelona, Spain, Nov. 2011).

[53] NEWCOMBE, R. A., IZADI, S., HILLIGES, O., MOLYNEAUX, D., KIM, D., DAVISON, A. J., KOHLI, P., SHOTTON, J., HODGES, S., AND FITZGIBBON, A. Kinectfusion: Real-time dense surface mapping and tracking. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality* (Washington, DC, USA, 2011), ISMAR '11, pp. 127–136.

[54] NISTÉR, D. An efficient solution to the five point relative pose problem. In *Proceedings in IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR 2003, 2003), vol. 2, pp. 195–202.

[55] NISTÉR, D. Preemptive ransac for live structure and motion estimation. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2* (2003), ICCV '03, pp. 199–.

[56] PAALANEN, P., AND KAMARAINEN, J.-K. Narrow baseline GLSL multiview stereo. In *3D Data Processing, Visualization and Transmission (3DPVT)* (Paris, France, 2010).

[57] PODLOZHNYUK, V. Histogram calculation in cuda. In *CUDA SDK, Nvidia Corporation* (2007).

[58] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA, 1992.

[59] PULLI, K. Surface modeling and display from range and color data. Ph.D. Thesis, 1997, Referred 29.8, 2013. `http://grail.cs.washington.edu/theses/PulliPhd.pdf`.

[60] ROYER, E., LHUILLIER, M., DHOME, M., AND LAVEST, J.-M. Monocular vision for mobile robot localization and autonomous navigation. *JOURNAL OF COMPUTER VISION 74*, 3 (2007), 237–260.

[61] RUSINKIEWICZ, S., AND LEVOY, M. Efficient variants of the ICP algorithm. In *Third International Conference on 3-D Digital Imaging and Modeling (3DIM2001)* (Quebec City, Canada, 2001), pp. 145–152.

[62] RUSU, R. B., AND COUSINS, S. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)* (Shanghai, China, May 9-13 2011).

[63] SCHARSTEIN, D., SZELISKI, R., AND ZABIH, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Proceedings of the IEEE Workshop on Stereo and Multi-Baseline Vision, Kauai, HI.* (December 2001).

[64] SHAMS, R., AND KENNEDY, R. A. Efficient histogram algorithms for NVIDIA CUDA compatible devices. In *Proceedings of the International Conference on Signal Processing and Communications Systems, IEEE* (Gold Coast, Australia, 2007), pp. 418–422.

[65] SHOTTON, J., FITZGIBBON, A., COOK, M., SHARP, T., FINOCCHIO, M., MOORE, R., KIPMAN, A., AND BLAKE, A. Real-time human pose recognition in parts from single depth images. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition* (Washington, DC, USA, 2011), CVPR '11, IEEE Computer Society, pp. 1297–1304.

[66] SHOTTON, J., GLOCKER, B., ZACH, C., IZADI, S., CRIMINISI, A., AND FITZGIBBON, A. Scene Coordinate Regression Forests for Camera Relocalization in RGB-D Images. In *Proceedings in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'13)* (2013).

[67] SIDJE, R. B. Expokit: Software package for computing matrix exponentials. *ACM - Transactions On Mathematical Software 24*, 1 (1998), 130–156.

[68] SILVEIRA, G., MALIS, E., AND RIVES, P. An Efficient Direct Approach to Visual SLAM. *IEEE Transactions on Robotics 24*, 5 (2008), 969–979.

[69] SMIRNOV, S., GOTCHEV, A. P., AND EGIAZARIAN, K. Methods for depth-map filtering in view-plus-depth 3d video representation. *EURASIP Journal on Advances in Signal Processing* (2012), 25.

[70] STEINBRÜCKER, F., STURM, J., AND CREMERS, D. Real-time visual odometry from dense rgb-d images. In *Workshop on Live Dense Reconstruction with Moving Cameras at the International Conference on Computer Vision (ICCV'11)* (Barcelona,Spain, 2011).

[71] STRASDAT, H., MONTIEL, J. M. M., AND DAVISON, A. J. Visual SLAM: Why filter? *Image and Vision Computing 30*, 2 (Feb. 2012), 65–77.

[72] STÜCKLER, J., AND BEHNKE, S. Integrating Depth and Color Cues for Dense Multi-Resolution Scene Mapping Using RGB-D Cameras. In *In Proceedings of IEEE International Conference on Multisensor Fusion and Information Integration (MFI)* (Hamburg, Germany, Sep 2012).

[73] STÜHMER, J., GUMHOLD, S., AND CREMERS, D. Real-time dense geometry from a handheld camera. In *Proceedings of the 32nd DAGM conference on Pattern recognition* (Berlin, Heidelberg, 2010), Springer-Verlag, pp. 11–20.

[74] STURM, J., MAGNENAT, S., ENGELHARD, N., POMERLEAU, F., COLAS, F., BURGARD, W., CREMERS, D., AND SIEGWART, R. Towards a benchmark for rgb-d slam evaluation. In *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS)* (Los Angeles, USA, June 2011).

[75] SZELISKI, R. Image alignment and stitching: A tutorial. Tech. rep., MSR-TR-2004-92, Microsoft Research, 2004, 2005.

[76] TOMASI, C., AND BIRCHFIELD, S. Depth discontinuities by pixel-to-pixel stereo. In *Sixth International Conference on Computer Vision, (ICCV'98)* (Bombay,India, 1998), pp. 1073–1080.

[77] TOMASI, C., AND MANDUCHI, R. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (ICCV'1998)* (Bombay,India, 1998), pp. 839–846.

[78] TOSSAVAINEN, T. Approximate and sqp two view triangulation. In *Computer Vision - ACCV 2010 - 10th Asian Conference on Computer Vision, Queenstown, New Zealand, November 8-12, 2010, Revised Selected Papers, Part III* (2010), R. Kimmel, R. Klette, and A. Sugimoto, Eds., vol. 6494 of *Lecture Notes in Computer Science*, Springer, pp. 1–14.

[79] TRIGGS, B., MCLAUCHLAN, P., HARTLEY, R., AND FITZGIBBON, A. Bundle adjustment - a modern synthesis. In *Vision Algorithms: Theory and Practice* (2000), B. Triggs, A. Zisserman, and R. Szeliski, Eds., vol. 1883 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 298–372.

[80] TYKKÄLÄ, T. M., , COMPORT, A. I., KÄMÄRÄINEN, J. K., AND HARTIKAINEN, H. Live RGB-D Camera Tracking for Television Production Studios. In *Journal of Visual Communication and Image Representation, Elsevier* (Jun 2013).

[81] TYKKÄLÄ, T. M., AUDRAS, C., AND COMPORT, A. I. Direct iterative closest point for real-time visual odometry. In *13th International Converence on Computer Vision, (ICCV'11), 2nd IEEE Workshop on Computer Vision in Vehicle Technology: From Earth to Mars* (Barcelona, Spain, Nov 2011).

[82] TYKKÄLÄ, T. M., AND COMPORT, A. I. A Dense Structure Model for Image Based Stereo SLAM. In *IEEE International Conference on Robotics and Automation (ICRA'11)* (Shanghai, China, May 9-13 2011).

[83] TYKKÄLÄ, T. M., COMPORT, A. I., AND KÄMÄRÄINEN, J.-K. Photorealistic 3D Mapping of Indoors by RGB-D Scanning Process. In *IEEE RSJ/International conference on Intelligent Robot and System, IROS* (Tokyo, Japan, Nov 2013).

[84] TYKKÄLÄ, T. M., HARTIKAINEN, H., COMPORT, A. I., AND KÄMÄRÄINEN, J. K. RGB-D Tracking and Reconstruction for TV Broadcasts. In *8th International Conference on Computer Vision Theory and Applications (VISAPP'13)* (Barcelona, Spain, Feb 2013).

[85] VICON. Boujou, professional matchmoving solution for film industry. Referenced Apr 2013, `http://www.vicon.com/boujou/`.

[86] VISCODA. Voodoo camera tracker: A tool for the integration of virtual and real scenes, Digital Laboratorium für Informationstechnologie, University of Hannover. Referenced Apr. 2013, `http://www.viscoda.com/index.php/en/voodoo-manual`.

[87] WELCH, G., AND BISHOP, G. An Introduction to the Kalman Filter. Tech. Rep. TR 95-041, University of North Carolina at Chapel Hill, Department of Computer Science, 2006.

[88] WENDEL, A., MAURER, M., GRABER, G., POCK, T., AND BISCHOF, H. Dense reconstruction on-the-fly. In *Proceedings in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'12)* (2012), pp. 1450–1457.

[89] WHELAN, T., KAESS, M., FALLON, M., JOHANNSSON, H., LEONARD, J., AND MCDONALD, J. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras* (Sydney, Australia, Jul 2012).

[90] YANG, R., AND POLLEFEYS, M. Multi-resolution real-time stereo on commodity graphics hardware. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'03) 1* (2003), 211–217.

[91] ZHANG, Z. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI'00) 22*, 11 (November 2000), 1330–1334.