Tampere University

Tatu Mäkinen

MODELING ENVIRONMENT USING MULTI-VIEW STEREO

Information Technology and Communication Sciences
Master of Science Thesis
May 2019

ABSTRACT

Tatu Mäkinen: Modeling environment using multi-view stereo
Master of Science Thesis
Tampere University
Master's Degree Programme in Information Technology
May 2019

In this work, we study the potential of a two-camera system in building an understanding of the environment. We investigate, if stereo camera as the sole sensor can be trusted in real time environment analysis and modeling to enable movement and interaction in a general setting.

We propose a complete pipeline from the sensor setup to the final environment model, evaluate currently available algorithms for each step, and make our own implementation of the pipeline. To assess real world performance, we record our own stereo dataset in a laboratory environment in good lighting conditions. The dataset contains stereo recordings using different camera angles concerning the movement, and ground truth for the environment model and the camera trajectory recorded with external sensors.

The steps of our proposed pipeline are as follows. 1) We calibrate two cameras using *de facto* method to form the stereo camera system. 2) We calculate depth from the stereo images by finding dense correspondences using semi global block matching and compare results to a recent data driven convolutional neural network algorithm. 3) We estimate camera trajectory using temporal feature tracking. 4) We form a global point cloud from the depth maps and the camera poses and analyze drivability in indoors and outdoors environments by fitting a plane or a spline model, respectively, to the global cloud. 5) We segment objects based on connectivity in the drivability model and mesh rough object models on top of the segmented clouds. 6) We refine the object models by picking keyframes containing the object, re-estimating camera poses using structure from motion, and building an accurate dense cloud using multi-view stereo. We use a patch-based algorithm that optimizes the photo consistency of the patches in the visible cameras.

We conclude that with current state of the art algorithms, a stereo camera system is capable of reliably estimating drivability in real time and can be used as the sole sensor to enable autonomous movement. Building accurate object models for interaction purposes is more challenging and requires substantial view coverage and computation with the current multi-view algorithms.

Our pipeline has limitations in long-term modeling: drift accumulates, which can be dealt with by implementing loop closure, and using external information such as GPS. Data wise, we inefficiently conserve complete information, while storing compressed presentations such as octrees or the built model can be considered. Finally, environments with insufficient texture and lighting are problematic for camera-based systems and require complementary solutions.

Keywords: computer vision, 3D modeling, stereo, multi-view

TIIVISTELMÄ

Tatu Mäkinen: Ympäristön mallinnus stereokameroilla
Diplomityö
Tampereen yliopisto
Tietotekniikan koulutusohjelma
Toukokuu 2019

Tässä työssä tutkimme kamerapohjaista ympäristön mallintamista. Tavoitteena on selvittää, soveltuuko stereokamera ainoana sensorina reaaliaikaiseen ympäristön analyysiin ja mallintamiseen, joka mahdollistaisi liikkumisen ja vuorovaikutuksen ympäristön kanssa yleisesti.

Muodostamme mallinnusprosessista kokonaisuuden alkaen kameroista ja päätyen 3D-malliin, vertailemme algoritmeja prosessin eri vaiheissa ja teemme oman implementaation. Vertaillaksemme ehdottamamme prosessin todellista toimivuutta, nauhoitamme stereovideoaineiston. Aineisto sisältää stereovideonauhoituksia käyttäen eri kamerakulmaa liikkeeseen nähden, ja vertailuaineiston ympäristömallista ja kameran liikkeestä, jotka on nauhoitettu eri sensorijärjestelmillä.

Ehdottamamme mallinnusprosessi koostuu seuraavista vaiheista. 1) Kalibroimme kamerat muodostaen stereokameramallin. 2) Laskemme syvyysinformaation stereokuvista etsimällä kuvista tiheästi vastaavuuksia käyttäen puoliglobaalia lohkovertailualgoritmia, ja vertailemme tuloksia viimeaikaisiin konvoluutioneuroverkkoalgoritmeihin. 3) Estimoimme kameran liikettä seuraamalla kuvan piirteiden liikettä ajan suhteen. 4) Muodostamme globaalin pistepilvimallin yhdistämällä syvyyskuvien ja kameran liikkeen informaation, ja analysoimme ajettavuutta sisätiloissa tasomallilla ja ulkotiloissa spline-mallilla. 5) Segmentoimme ympäristön objektit ajettavuusmallin perusteella, ja muodostamme karkeat objektimallit segmentoitujen objektipistepilvien ympärille. 6) Parannemme objektimallien laatua etsimällä datasta objektin sisältävät kuvat, estimoimalla lokaalisti kameran asennot objektin suhteen käyttäen rakenne liikkeestä -algoritmia ja rakentamalla uuden objektimallin moninäkymästereoalgoritmilla. Käytämme tilkkupohjaista algoritmia, joka optimoi tilkkujen projektioiden yhdenpitävyyden sen näkevissä kameroissa.

Työmme perusteella nykyisillä algoritmeilla stereokamerajärjestelmä soveltuu luotettavaan ajettavuuden analysointiin, ja sitä voidaan käyttää ainoana sensorina mahdollistamaan autonominen liikkuminen reaaliajassa. Tarkkojen objektimallien luonti interaktiota varten osoittautui haasteelliseksi: nykyiset algoritmit vaativat kattavasti näkymiä objektin ympäriltä ja merkittävästi laskentaa.

Mallinnusalgoritmi on vajavainen pidemmän aikavälin mallinnuksessa, kun virhe kasaantuu kameran liikkeen estimoinnissa johtaen ajautumiseen. Ongelma voitaisiin ratkaista silmukansulkemisalgoritmilla, ja käyttäen referenssi-informaatiota, kuten GPS:ää. Datanäkökulmasta tallennamme tehottomasti kaiken tiedon, kun informaatio voitaisiin pakata esimerkiksi octree-muotoon, tai ainoastaan muodostettu ympäristömalli voitaisiin säilyttää. Lopuksi, kamerapohjainen mallinnus vaatii toimiakseen riittävästi valoa ja tekstuuria, joiden puuttuessa vaaditaan täydentäviä ratkaisuja.

Avainsanat: konenäkö, 3D mallinnus, stereokamera

PREFACE

This work was done related to the Spatial Sensing for Machines project with the 3D Media Group in Tampere University.

I would like to thank Prof. Atanas Gotchev and Dr. Sari Peltonen for supervising the thesis, Dr. Jari Yli-Hietanen for the opportunity to work on this topic and guidance throughout the process, and Olli Suominen for managing the project. I would also like to thank Ihtisham Ali for helping me with the thesis and Ahmed Durmush for technical support.

Finally, I would like to thank everyone I worked with on the third floor of Tietotalo. Special thanks to the personnel of the 3D Media Group, it has been a pleasure.

Tampere, 26th May 2019

Tatu Mäkinen

CONTENTS

LIST OF FIGURES

NOMENCLATURE

bundle adjustment	Jointly optimizing constructed model and camera poses
calibration	Building a mathematical model of the camera system
disparity	Pixel difference between stereo images of the same object
epipolar geometry	Geometry defining relations between two cameras
feature	Distinguishable property in an image
homogeneous form	Additional dimension equal to one is added, marked with $\sim$ symbol
Lambertian surface	Ideally reflecting surface; follows Lambert's cosine law
loop closure	Correcting camera trajectory when previously seen areas are redetected
photo consistency	Similarity between images
point cloud	Set of points in 3D space
rectification	Aligning images to a common plane
spline	Piecewise defined polynomial function
stereo vision	Vision based on two cameras
triangulation	Determining 3D locations based on image correspondences and camera geometry
voxel	Pixel in 3D space

ACRONYMS

NCC	Normalized Cross Correlation
PCC	Pearson Correlation Coefficient
SSD	Sum of Squared Differences
SAD	Sum of Absolute Differences
MATLAB	Matrix Laboratory
BRIEF	Binary Robust Independent Elementary Features
BT	Birchfield and Tomasi sampling insensitive measure
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DoG	Difference of Gaussians
FAST	Features from Accelerated Segment Test
FoV	Field of View
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
FREAK	Fast Retina Keypoint
GPS	Global Positioning System
GPU	Graphical Processing Unit
IMU	Inertial Measurement Unit
KLT	Kanade Lucas Tomasi feature tracking algorithm
LIDAR	Light Detection and Ranging
MI	Mutual Information
Mpx	Mega pixel
MRI	Magnetic Resonance Imaging
MVS	Multi View Stereo; vision based on two or more cameras
OpenCV	Open Computer Vision
ORB	Oriented FAST and Rotated BRIEF
PC	Personal Computer
PCA	Principal Component Analysis
PSMN	Pyramid Stereo Matching Network

RADAR	Radio Detection and Ranging
RANSAC	Random Sample Consensus
RMSD	Root Mean Square Deviation
ROS	Robot Operating System
SfM	Structure from Motion
SGBM	Semi Global Block Matching
SLAM	Simultaneous Localization and Mapping
SoA	State of the Art
SONAR	Sound Navigation and Ranging
ToF	Time of Flight
VR	Virtual Reality

SYMBOLS

$*$	convolution
α	angle
a	tangential distortion coefficient
$\mathbf{B}$	deformation matrix
b	baseline
$\mathbf{C}$	camera location
$\mathcal{C}$	cost function
$\mathbf{c}$	principal point
Cov	covariance
$\mathbf{D}$	disparity map
d	disparity
d_c	distance
Δ	change
det	determinant
E	expected value
$\mathbf{e}$	epipole
$\mathbf{F}$	fundamental matrix
f	function
G	energy
$\mathbf{G}$	2D Gaussian
H	entropy
$\mathbf{I}$	identity matrix
$\mathbf{J}$	image
$\mathbf{K}$	intrinsic matrix
κ	constant
λ	eigenvalue
l	focal length
$\mathbf{M}$	structure tensor
μ	mean

$\mathcal{N}$	neighborhood
N	natural number; $n \in \mathbb{N}$
$\mathbf{n}$	normal vector
o	centre of distortion
P	probability
$\mathbf{P}$	projection matrix
$\tilde{\mathbf{p}}$	pixel in homogeneous coordinates
$\mathbf{p}$	pixel
p	probability density function
Q	photo consistency
q	radial distortion coefficient
$\mathbf{R}$	rotation matrix
r	radius
$\mathbf{r}$	direction vector
ρ	penalty
s	skew
S	space
σ	standard deviation
$\mathbf{T}$	transformation matrix
T	truth function
θ	threshold
$\mathbf{t}$	translation vector
tr	trace
$\mathbf{v}$	vector
ω	weight function
$\mathbf{w}$	world point
w_s	window size

1 INTRODUCTION

Human visual system is the pinnacle of billions of years of evolution. It is arguably our most important sense, as it provides a substantial amount of information of our environment.

For decades, researchers have dreamed of replicating this capability by making machines able to see. The research community and industry alike have put tremendous effort to advance the field. Capable computer vision would lead way to a completely new era of possibilities, as the machines could operate autonomously.

Though the analogy between human sight and cameras is often used, it should be noted that the human visual system is vastly different from a camera sensor system. Human vision is heavily based on assumptions ingrained in biology and individual life experiences. Numerous visual illusions demonstrate how these assumptions often override the actual incoming light [8].

Computer vision relies on mathematical or statistical analysis of the input sensor data. Traditional engineering approaches try to model this relationship between the sensor data and real world. Machine learning approaches on the other hand rely on sufficient data to define the relationship. In our work, we focus on the more traditional model-based approaches. Despite their fortes, current data driven algorithms lack explainability, and generalization to real world situations is uncertain in the absence of big data.

The fundamental goal in computer vision is to understand the physical world based on sensor data. In our work, this "understanding" is in the sense of building an accurate 3D model of the environment. Using two cameras, *i.e.*, stereo camera, as our sensor, we aim to process and refine the incoming pixel information into a substantial and compact presentation of the environment that can be used for navigation and interaction.

Besides cameras, there are other passive and active (energy emitting) sensors, which can be used to gather information of the environment. Each sensor has its pros and cons, and hence the most robust modeling can be achieved by combining multiple sensors, if the complexity in data processing is disregarded. Here we shortly discuss the different sensor alternatives.

Camera is inexpensive, efficient and able to acquire abundant information of the environment. This is largely due to the immense amount of research and industry effort that has been put into the development of camera optics and sensors. The main disadvantage of cameras for extracting 3D information is that they produce images in 2D. Thus, we

have the inverse problem of modeling the original shape based on the 2D projections, which requires a lot of computation. In our work, we use stereo camera, *i.e.*, two cameras for which we know their geometrical relationship, to constrain the problem to allow real time processing. The versatility, the price, and the resemblance to our own vision make cameras our sensor of choice.

Light Detection and Ranging (LIDAR) sensors emit light and model the environment by measuring the reflections. LIDAR sensors produce accurate depth information directly, and thus avoid the data processing hindrances. Generally, the method is based on Time of Flight (ToF) and only depth information is computed from the reflections. Nevertheless, more reflectance data can be captured, *e.g.*, full waveform analysis can be performed to identify material types. LIDAR technologies have recently progressed rapidly due to the high industry demand. Temporal resolution of LIDAR is currently good enough for obstacle avoidance and the accuracy of the model points is superior to other methods. However, increasing the spatial resolution makes LIDAR slow and a camera is needed to combine color information to the model. Reflective and non-reflective are problematic for LIDAR, and most importantly, the price of the sensor is at present comparatively high. [39]

Sound Navigation and Ranging (SONAR) and Radio Detection and Ranging (RADAR) use sound and radio waves respectively with a mindset similar to LIDAR to sense the environment. Nevertheless, for accurate environment modeling purposes, the current quality and deployability of these technologies is not on par with LIDAR and camera based depth sensing methods.

Traditionally camera based (photogrammetric) methods rely on finding correspondences between images to infer the underlying world in 3D. However, images contain many other cues that can be used in depth estimation. 1) Shading is useful if lighting can be modeled. 2) Irregularities in texture patterns can be used if patterns can be assumed, *e.g.*, hole in a wall. 3) Contours of the objects can be taken advantage of assuming regular shape, *e.g.*, an ellipse in the image can be assumed to be a projection of a circle in the 3D world. Moreover, combining contours with shading information, regular body of rotation assumptions can be made. 4) Defocusing effects due to the limited camera depth of field can be used as well to differentiate between near and far objects. Overall, generally applicable models based on these properties are difficult to derive, and thus, they should be considered as complementary methods to depth estimation at best. [43]

In this work we investigate the current State of the Art (SoA) in camera based modeling as presented in Figure 1.1. Starting with two streams of images from the stereo camera system, we aim to build an accurate model of the environment that would enable increased autonomy for machines.

The algorithmic steps towards the desired 3D model start from stereo camera calibration to build an accurate mathematical presentation of the camera system. Using the calibrated stereo cameras, we approach the depth estimation as disparity estimation between the stereo images, for which we compare the well-founded Semi Global Block Matching

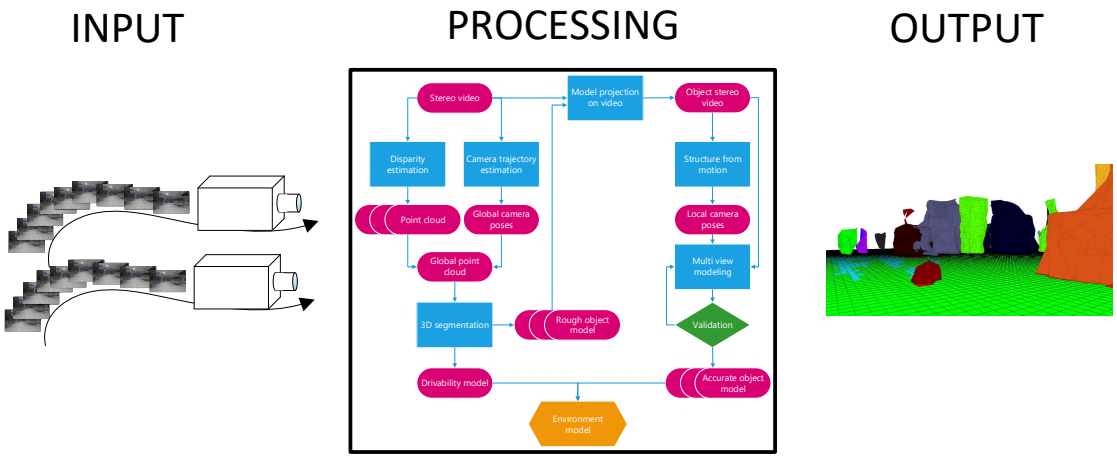


Figure 1.1. *Via processing the information gathered by two cameras, we aim to create an accurate 3D model of the environment.*

(SGBM) method [27] and a current SoA data driven approach [6]. Simultaneously we track the camera movement, for which we make our own adaptation based on Kanade Lucas Tomasi feature tracking algorithm (KLT) feature tracking [55], which we compare to the SoA trajectory estimation algorithm [44]. Combining the results to a global model, we then investigate the use of splines in outdoors drivability analysis [4], and simple ground plane model indoors. For refining object models, we examine the viability of multi-view modeling by combining Structure from Motion (SfM) and a patch based multi-view modeling approach [17].

The main contributions of this work are:

1. proposed complete stereo pipeline for environment modeling and its implementation;
2. real time drivability estimation using stereo vision;
3. analysis and evaluation of the currently available camera based modeling algorithms;
4. stereo dataset for assessing real world performance indoors.

The work is constructed as follows. In Chapter 2, we discuss the theory of camera based modeling and unfold the principal ideas behind the algorithms. In Chapter 3, we present the stereo dataset that we recorded to evaluate the real world performance of the algorithms. We continue in Chapter 4 by presenting our proposed pipeline for environment modeling, and discuss the details of our implementation. In Chapter 5, we compare our implementation to the current SoA algorithms and discuss the limitations of our pipeline. Finally, in Chapter 6 we present our conclusions.

2 THEORETICAL BACKGROUND

In this chapter, we discuss the theory behind stereo vision and multiple view modeling.

2.1 Camera

2.1.1 Pinhole model

Simple camera system can be thought of as a pinhole model, where all the light goes through a small hole to the sensor plane. Focal length l is the distance between the hole and the sensor plane, and image plane refers to the projection of the sensor plane image at the focal length distance in front of the pinhole. Pinhole camera model is presented in Figure 2.1.

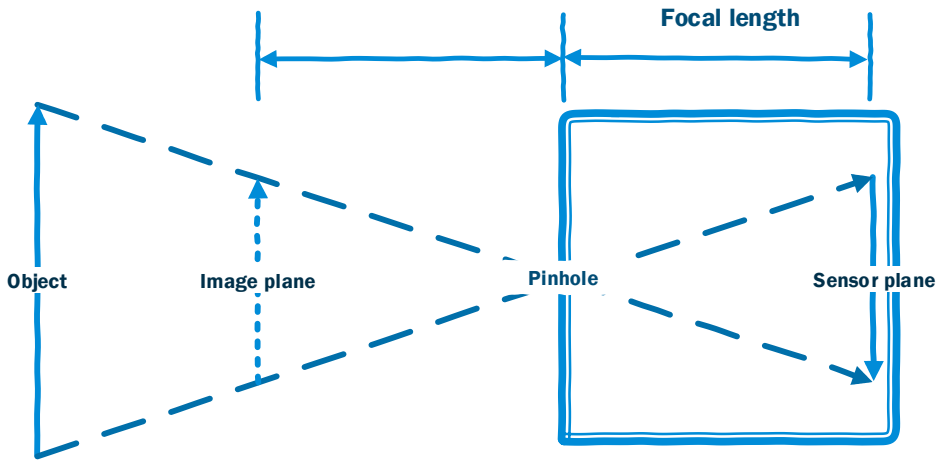


Figure 2.1. *Pinhole camera model.*

Real camera usually has multiple lenses in the place of the pinhole to increase the amount of incoming light and properly direct it to the sensor plane. However, mathematically a normal camera system can still be modeled as a pinhole camera. This model is generally divided in extrinsic and intrinsic elements. Extrinsic parameters determine how light from the world comes to the camera. Intrinsic parameters in turn model how the light is

transferred inside the camera to the hypothetical image plane. The system is called a perspective camera, as objects closer to the camera form bigger projections to the image plane than distant objects.

Intrinsic parameters are conventionally presented by a 3x3 intrinsic matrix $\mathbf{K}$ containing five parameters:

$$\mathbf{K} = \begin{bmatrix} l_x & s & \mathbf{c}_x \\ 0 & l_y & \mathbf{c}_y \\ 0 & 0 & 1 \end{bmatrix},$$

where

- l_x and l_y are the horizontal and vertical focal lengths. They can also be expressed with respect to the focal length l as $l_x = l\alpha_x$ and $l_y = l\alpha_y$, where α_x and α_y are horizontal and vertical pixel per length scale factors.
- s represents the skew of the sensor plane normal with regard to the camera's optical axis. Optical axis is defined by the camera lens system as the direction in which light does not bend. Skew is usually close to zero, meaning that the optical axis is perpendicular to the sensor plane.
- $(\mathbf{c}_x, \mathbf{c}_y)^T$ is the principal point, which is the intersection point between the optical axis and the image plane.

Extrinsic parameters are presented by a 3x4 matrix $[\mathbf{R}|\mathbf{t}]$ containing 3x3 rotation matrix $\mathbf{R}$ and 3x1 translation vector $\mathbf{t}$:

$$[\mathbf{R}|\mathbf{t}] = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} & \mathbf{R}_{13} & \mathbf{t}_x \\ \mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{23} & \mathbf{t}_y \\ \mathbf{R}_{31} & \mathbf{R}_{32} & \mathbf{R}_{33} & \mathbf{t}_z \end{bmatrix}.$$

Translation vector $\mathbf{t}$ is equal to the position of the world coordinate system's origin, and the rotation matrix $\mathbf{R}$ corresponds to the rotation of the origin. Both are determined in the camera's coordinate system.

The intrinsic and extrinsic transformation matrices can be combined to form a 3x4 matrix presentation for mapping world points to the camera's image plane. This is called the camera projection matrix defined as $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$. Thus, a world point $\mathbf{w} = (\mathbf{w}_x, \mathbf{w}_y, \mathbf{w}_z)^T$

is mapped to the image pixel $\mathbf{p} = (\mathbf{p}_x, \mathbf{p}_y)^T$ by

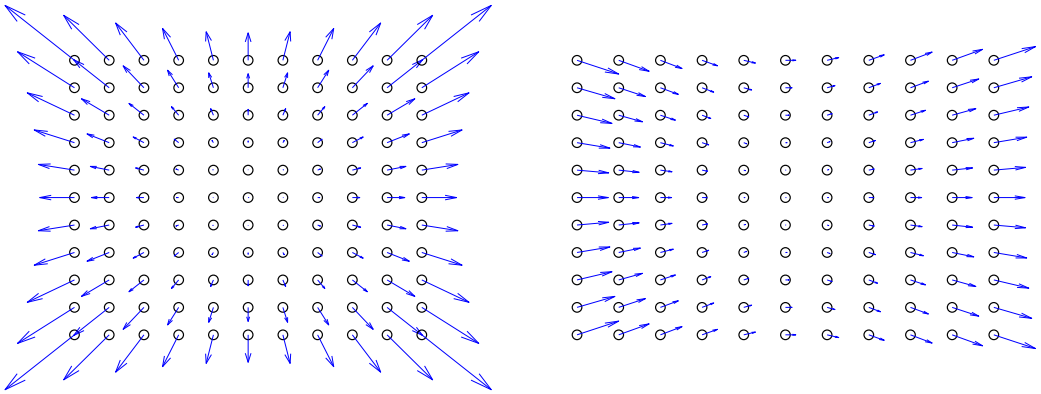
$$\mathbf{P}\begin{bmatrix}\mathbf{w}_x\\ \mathbf{w}_y\\ \mathbf{w}_z\\ 1\end{bmatrix} = \overbrace{\begin{bmatrix} l_x & s & \mathbf{c}_x\\ 0 & l_y & \mathbf{c}_y\\ 0 & 0 & 1\end{bmatrix}}^{\mathbf{K}} \overbrace{\begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12} & \mathbf{R}_{13} & \mathbf{t}_x\\ \mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{23} & \mathbf{t}_y\\ \mathbf{R}_{31} & \mathbf{R}_{32} & \mathbf{R}_{33} & \mathbf{t}_z\end{bmatrix}}^{\mathbf{R}|\mathbf{t}} \begin{bmatrix}\mathbf{w}_x\\ \mathbf{w}_y\\ \mathbf{w}_z\\ 1\end{bmatrix} = \kappa \begin{bmatrix}\mathbf{p}_x\\ \mathbf{p}_y\\ 1\end{bmatrix}, \kappa \neq 0, \tag{2.1}$$

where κ is the scale of the projection. World coordinates are presented in the homogeneous form, where fourth dimension equal to one is added to make projection by simple matrix multiplication possible.

2.1.2 Distortion

To make the perspective camera model more general, nonlinear lens distortions have to be taken into account.

In radial distortion, light bends differently with regard to the distance from the center of distortion o. Radial distortion is symmetric and is caused by imperfections in the lens shape. Tangential (decentering) distortion must also be considered if the lens is not parallel to the image plane. The effects of distortion are presented in Figure 2.2.



(a) radial distortion ***(b)*** tangential distortion

Figure 2.2. *Effects of radial and tangential distortion. The circles represent real points and arrows point to their distorted locations.*

Distortion can be modeled as

$$\mathbf{p}_r = \mathbf{p}_d + \overbrace{\begin{bmatrix} \tilde{x}(q_1 {d_c}^2 + q_2 {d_c}^4 + q_3 {d_c}^6 + \dots) \\ \tilde{y}(q_1 {d_c}^2 + q_2 {d_c}^4 + q_3 {d_c}^6 + \dots) \end{bmatrix}}^{\text{radial correction}} + \overbrace{\begin{bmatrix} \big(a_1({d_c}^2 + 2\tilde{x}^2) + 2a_2\tilde{x}\tilde{y}\big)(1 + a_3 {d_c}^3 + \dots) \\ \big(a_2({d_c}^2 + 2\tilde{y}^2) + 2a_1\tilde{x}\tilde{y}\big)(1 + a_3 {d_c}^3 + \dots) \end{bmatrix}}^{\text{tangential correction}}, \quad (2.2)$$

where
$\mathbf{p}_r = (\mathbf{p}_{rx}, \mathbf{p}_{ry})^T$ real pixel coordinates,
$\mathbf{p}_d = (\mathbf{p}_{dx}, \mathbf{p}_{dy})^T$ distorted pixel coordinates,
$\tilde{x} = o_x - \mathbf{p}_{dx}$ horizontal distance from center of distortion,
$\tilde{y} = o_y - \mathbf{p}_{dy}$ vertical distance from center of distortion,
$d_c = \sqrt{\tilde{x}^2 + \tilde{y}^2}$ distance from the center of distortion,
q_i radial distortion coefficients, and
a_i tangential distortion coefficients.

Center of distortion o is usually assumed equal to the principal point $\mathbf{c}$. Moreover, in the standard form only three parameters for radial and two parameters for tangential distortion are used. Another common practice is to use only one instead of three radial distortion parameters and compensate the error with modifications to the intrinsic matrix. [62]

After the distortion coefficients have been estimated in calibration, the polynomial undistortion model presented above can be used to undistort the images. Undistortion is a crucial step if the goal is to make associations between real world and the image.

2.1.3 Calibration

Calibration is done to calculate the parameters of the camera system. This includes the intrinsic parameters, distortion parameters and in the case of stereo camera calibration, the parameters describing the relationship between the two cameras, *i.e.*, translation and rotation between them.

There are two general approaches to calibration: self-calibration and photogrammetric calibration. In self-calibration, the parameters are calculated from a sequence of images with different camera poses. It is assumed that internal parameters and the environment stay constant, and thus the camera parameters can be estimated using correspondences between images. Photogrammetric calibration, on the other hand, uses a calibration object with known geometry to make the connection between the images and the world.

Zhang's calibration method

Zhang's calibration method [61] is the standard photogrammetric method used in practice, as it provides accurate calibration with relatively little effort. The calibration object used is a 2D plane with a repetitive pattern, for which the size of the pattern is known. An example of the calibration object can be seen in Figure 3.7.

The method is based on two different definitions of the relationship between the camera and calibration object. First, homography mapping is used to define the relationship between the two planes, *i.e.*, the calibration object and the image plane with eight parameters. Second, the geometric relationship between the camera's coordinate system and the object's coordinate system is defined with three rotational and three translational parameters. As there is a two-parameter difference between the definitions, we get two constraints for the intrinsic parameters with each observation (where the calibration object is in the image). Thus, as there are five parameters in the intrinsic matrix, unique solution can be defined with three or more observations. To be more concise, the steps of the algorithm are as follows.

1) The first step of the algorithm is to estimate the intrinsic parameters using a closed form solution established by the aforementioned constraints. The constraints are stacked to a matrix $\mathbf{A}$, and an equation $\mathbf{A}\mathbf{b} = 0$ is formed, where vector $\mathbf{b}$ is based on the intrinsic parameters. From this equation, $\mathbf{b}$ can be solved as the eigenvector of the smallest eigenvalue of $\mathbf{A}^T\mathbf{A}$, and hence we get an estimate of the intrinsic parameters.

2) Based on the estimated intrinsic matrix and the homographies, the extrinsic parameters are calculated. 3) Then, radial distortion is approximated by stacking equations based on the previously estimated parameters and the radial distortion model, from which linear least squares solution is calculated. 4) Finally, to refine the parameters received from the closed solution, maximum likelihood estimation is used to solve the nonlinear equation, initialized using the estimated parameters.

As the algorithm is based on the constraints received from the geometric and homographic relations, the movement of the calibration object should include all the six geometric degrees of freedom to provide enough restrictions on the camera parameters. In practice, this means that the calibration objects should not be coplanar in the images. The calibration accuracy can be increased by rising the amount of observations. However, this comes with diminishing returns, and an empirical upper limit of 70 observations is presented in the literature [45].

2.2 Stereopsis

2.2.1 Backprojection to optical rays

After the images have been undistorted, relationship between the images and the world can be geometrically modeled. In this model, image pixels correspond to 3D cones in world space. Everything inside the cone is projected to the corresponding pixel in the image. The diameter of the cone depends on the camera resolution. However, for moderate distances the diameter can be assumed to be close to zero. This means that the 3D cone becomes a 3D ray, *i.e.*, an optical ray in the world space. The relationship between image pixels and cones in the 3D world is illustrated in Figure 2.3

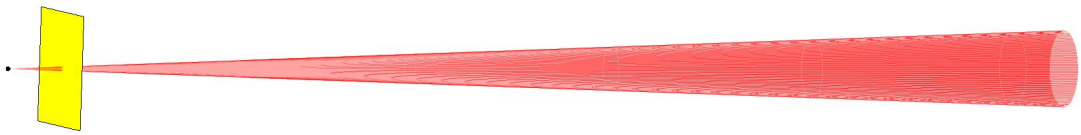


Figure 2.3. *Image pixel measures incoming light inside a 3D cone. The diameter of the cone depends directly on the resolution of the sensor plane, which is represented in the picture as the yellow imaginary image plane (see Figure 2.1).*

The optical ray can be calculated by picking two known points it passes through, *e.g.*,

- camera center in homogeneous world coordinates $\tilde{\mathbf{C}} = [-\mathbf{R}^T\mathbf{t}, 1]^T$
- intersection point with the plane at infinity $\tilde{\mathbf{D}} = [\mathbf{R}^T\tilde{\mathbf{p}}, 0]^T$

Optical ray is then defined as the join of these two points

$$\tilde{X}(d_c) = d_c\tilde{\mathbf{D}} + \tilde{\mathbf{C}} = [\mathbf{R}^T(d_c\tilde{\mathbf{p}} - \mathbf{t}), 1]^T \tag{2.3}$$

where
$\tilde{X}$ the ray model defined in homogeneous world coordinates,
$\tilde{\mathbf{p}} = [\tilde{\mathbf{p}}_x, \tilde{\mathbf{p}}_y, 1]^T$ homogeneous pixel coordinates, and
d_c distance from the camera center.

2.2.2 Epipolarity and Fundamental matrix

Besides relating pixels to world geometry and vice versa, associations between different camera sensors can also be made. This is essential in stereo vision and multi-view imaging, where information from two or more cameras, respectively, is combined.

As explained in the previous section, pixels correspond to optical rays, which can then be projected to other cameras. An optical ray defined by a pixel in one camera is represented by a line in the image planes of the other cameras that have the ray in their Field of View (FoV). The projections of an optical ray are referred to as epipolar lines. Leaving out

the intermediate definition of optical rays, there is thus a direct correspondence between pixels in one camera and the epipolar lines in other cameras if camera parameters are known. Epipolar lines can be exploited, *e.g.*, to limit the search space when finding correspondences between images.

The epipolar relationship between two cameras is presented in mathematical form by the fundamental matrix $\mathbf{F}$. Given pixel $\mathbf{p}$ in one camera, epipolar line in the other camera is defined as $\mathbf{F}\tilde{\mathbf{p}}$. In general, fundamental matrix satisfies

$$\forall(\tilde{\mathbf{p}}_1 \leftrightarrow \tilde{\mathbf{p}}_2) : \tilde{\mathbf{p}}_1^T \mathbf{F} \tilde{\mathbf{p}}_2 = 0,$$

where $(\tilde{\mathbf{p}}_1 \leftrightarrow \tilde{\mathbf{p}}_2)$ are corresponding pixels in the homogeneous form.

Fundamental matrix can be calculated from at least seven point correspondences between two images. Assuming Gaussian error in the point correspondences, maximum likelihood estimate can be found via cost minimization in the method referred as "The Gold Standard Method" [25]. Otherwise, if the camera parameters are known, fundamental matrix is defined as

$$\mathbf{F} = [\mathbf{P}_2 \tilde{\mathbf{C}}_1]_\times \mathbf{P}_2 \mathbf{P}_1^+, \tag{2.4}$$

where
- $\mathbf{P}_1, \mathbf{P}_2$ projection matrices,
- $\tilde{\mathbf{C}}_1$ center of the first camera, where $\mathbf{P}_1 \tilde{\mathbf{C}}_1 = \mathbf{0}$,
- $^+$ pseudo inverse, and
- $[]_\times$ skew symmetric matrix of the vector (see Appendix A.1).

2.2.3 Rectification

In the pinhole model, all the light coming to the camera goes through its optical center. Thus, when the optical center of one camera is projected to the image plane of another camera, all the epipolar lines go through the projected point. This common point between the epipolar lines is referred to as the epipole.

In an ideal stereo camera system, both cameras are parallel and the cameras' image planes are aligned with the baseline, which is the line that goes through the optical centers. This implies that both of the epipoles are at infinity, and the epipolar lines run horizontally parallel through the images. Hence, corresponding pixels are found on the same horizontal line in the images.

Rectification is the process of mathematically idealizing a stereo camera system, making it equivalent to the model described above. This is done via rotating the camera models around their optical centers so that 1) their image planes become coplanar and 2) the horizontal axes of the image planes are aligned with the baseline. New common intrinsic

parameters are also calculated to make the vertical correspondences proportionate.

For rectification, individual intrinsic and extrinsic parameters of the cameras need to be calculated first via calibration described in Section 2.1.3. If rectification is done in post processing where camera projection matrices $\mathbf{P}_1, \mathbf{P}_2$ are known, they can be factorized to the form $\mathbf{P}_i = \mathbf{K}_i[\mathbf{R}_i|\mathbf{t}_i]$ using, *e.g.*, QR-decomposition. New shared rotation matrix $\mathbf{R}$ can then be defined based on the individual camera parameters, *e.g.*, as follows:

- x-axis is the baseline $b = \mathbf{c}_1 - \mathbf{c}_2$,
- y-axis is orthogonal to the defined x-axis and to either of the old z-axis (usually from the left camera), and
- z-axis is orthogonal to the previous two.

Intrinsic matrix $\mathbf{K}$ is arbitrary and can be defined, *e.g.*, as the average of the old intrinsic matrices where skew is removed. New projection matrices $\mathbf{P}'_1, \mathbf{P}'_2$ and the rectifying image transformations $\mathbf{T}_1, \mathbf{T}_2$ are thus established as

$$\mathbf{P}'_i = \mathbf{K}[\mathbf{R}| - \mathbf{R}\mathbf{c}_i]$$

$$\mathbf{T}_i = \mathbf{P}'_{i[1,2,3;1,2,3]}\mathbf{P}^{-1}_{i[1,2,3;1,2,3]}.$$

Overall, rectification process produces camera models that differ only in their location on the baseline. [18]

2.2.4 Disparity estimation

Disparity estimation from stereo images is done by first finding corresponding points in the images, and then calculating disparities between the corresponding points. Disparity is directly proportional to the distance of the object from the cameras, *i.e.*, depth. Accordingly, if we know the parameters of the camera system, and are able to find correspondences between the images, we can calculate depth.

Occlusions due to the difference between the views are problematic, *i.e.*, when an object blocks the view of the world visible from the other camera's view. There are no corresponding points to be found and, hence, no disparity. Moreover, these areas of depth discontinuities also affect their neighboring regions in that, if correspondence matching is done using a local window, the neighboring pixels are vastly different depending on the camera location. Finally, global search for correspondences between the images has the complexity of $O(N^2)$. Thus, the search space is usually restricted based on the camera models, and disparities are searched only up to a defined limit. If the stereo camera model is accurate, the epipolar constraint reduces the search space to the corresponding 1D line. Rectification makes applying the epipolar constraint straightforward, as the matches can be searched from the same horizontal line in the common rectified image plane.

Dense disparity calculation generally includes the same four steps: 1) computing matching

costs, 2) aggregating costs, 3) computing disparities and 4) refining disparity maps. Depending on the algorithm, disparities can be calculated locally, globally as an optimization problem, or by combining both local and global aspects, *e.g.*, via semi global matching.

Semi global matching

Disparity estimation can be presented as a global energy minimization problem, where the goal is to find the disparity that minimizes the sum of pixel wise matching costs and two added penalty terms. The penalty terms compensate for errors of ambiguous pixel wise matching by penalizing changes in disparity. Global energy G for disparity map $\mathbf{D}$ is defined as

$$G(\mathbf{D}) = \sum_{i,j} \Big(\overbrace{\mathcal{C}(\mathbf{J}_{ij}, \mathbf{D}_{ij})}^{\text{pixelwise matching costs}} + \overbrace{\rho_1 \sum_{k,l \in \mathcal{N}_{i,j}} T\big(|\mathbf{D}_{ij} - \mathbf{D}_{kl}| \leq \theta\big)}^{\text{penalty for small disparity changes}} + \overbrace{\sum_{k,l \in \mathcal{N}_{i,j}} \max(\rho_1, \frac{\rho_2}{|\mathbf{J}_{ij} - \mathbf{J}_{kl}|}) T\big(|\mathbf{D}_{ij} - \mathbf{D}_{kl}| > \theta\big)}^{\text{penalty for large disparity changes}} \Big) \quad (2.5)$$

$$T(a) = \begin{cases} 1 \text{ if } a == \text{true}, \\ 0 \text{ else} \end{cases} \qquad k, l \in \mathcal{N}_{i,j} \iff \begin{cases} k \in & \left[i - \frac{w_s+1}{2}, i + \frac{w_s+1}{2} \right] \setminus i \\ l \in & \left[j - \frac{w_s+1}{2}, j + \frac{w_s+1}{2} \right] \setminus j \end{cases}$$

where
$\mathbf{J}$ image,
$\mathcal{C}(\mathbf{p}, d)$ pixelwise matching cost for pixel $\mathbf{p}$ at disparity d,
ρ_1, ρ_2 penalty coefficients,
θ threshold differentiating between small and large disparity changes, and
$w_s \in 2N + 1$ neighborhood window size.

The distinction between the penalty terms ensure that smooth disparity changes occurring, *e.g.*, at slanted surfaces are not penalized as much as larger disparity changes caused by depth discontinuities. The penalty for larger disparity changes ρ_2 is scaled inversely proportional to the intensity, down to ρ_1. The inverse scaling is on the grounds that in an accurate disparity map, there is generally a strong correlation between the intensity changes and disparity changes.

As the minimization of the global energy presented in Equation 2.5 is a NP-complete problem, it needs to be approximated. Semi global matching [27] estimates the minimum of the energy by aggregating path wise minima along different directions across the image.

Path wise matching cost is defined as

$$\mathcal{C}_{path}(\mathbf{r},\mathbf{p},d) = \mathcal{C}(\mathbf{p},d) + \min \begin{pmatrix} \mathcal{C}_{path}\Big(\mathbf{r},\mathbf{p}+\mathbf{r}_s, \quad d \quad \Big), \\ \mathcal{C}_{path}\Big(\mathbf{r},\mathbf{p}+\mathbf{r}_s, \quad \underset{m\in|m-d|\leq\theta}{\operatorname{argmin}}\big(\mathcal{C}_{path}(\mathbf{r},\mathbf{p}+\mathbf{r}_s,m)\big) \quad \Big) + \quad \rho_1, \\ \mathcal{C}_{path}\Big(\mathbf{r},\mathbf{p}+\mathbf{r}_s, \quad \underset{n\in|n-d|>\theta}{\operatorname{argmin}}\big(\mathcal{C}_{path}(\mathbf{r},\mathbf{p}+\mathbf{r}_s,n)\big) \quad \Big) + \quad \rho_2 \end{pmatrix}, \tag{2.6}$$

where $\begin{cases} \mathbf{r} & \text{direction of the path, and} \\ \mathbf{r}_s & \text{step in image coordinates in the direction } \mathbf{r}. \end{cases}$

Consequently, path wise cost is calculated recursively starting from the image boundary and summing up the minimum costs on the pathway to pixel $\mathbf{p}$. Changes in the disparity on the way are penalized to encourage smoothness. The pixel wise matching cost $\mathcal{C}$ assesses local photo consistency, for which different measures are discussed in Section 2.4.

Finally, the semi global matching cost $\mathcal{C}_{sg}$ is defined as the sum of the path wise matching costs as

$$\mathcal{C}_{sg}(\mathbf{p},d) = \sum_{\mathbf{r}} \mathcal{C}_{path}(\mathbf{r},\mathbf{p},d) \tag{2.7}$$

By increasing the amount of paths $\mathbf{r}$, a better estimate of the global minimum can be achieved. Already with diagonal, horizontal, and vertical paths, the aggregation gives a decent estimate of the validity of the disparity.

Disparity map $\mathbf{D}$ is built by finding the semi global matching cost minima for all the pixels $\mathbf{p}$ and combing the corresponding disparities d to form the map. To account for occlusions, the disparity estimation is done two times, swapping around the matched image and the base image. The two disparity maps are then compared in a consistency check, where too large differences in disparities are invalidated.

2.3 Features

Features are distinguishable image properties, *e.g.*, points, edges, or 3D objects in the image. The choice of which features to use is highly dependent on the problem at hand. Nevertheless, their robustness for rotation, scaling, occlusions, and lighting changes should be considered.

In stereo and multiple view vision, features are used to find corresponding points in the images taken temporally and/or spatially from different points, as visualized in Figure 2.4. Assumptions of constrained camera parameters and locally static environment alleviate the correspondence problem. However, it is not trivial to find features that stay consistent without profound understanding of the scene. Moreover, depending on the application the

trade-off between accuracy and performance has to be taken into account.

Figure 2.4. *Two images taken with small temporal and spatial difference.*

Generally, feature processing starts by detecting locations from images that contain applicable information, *e.g.*, gradients in multiple directions. Next, a feature vector is extracted to make an invariant descriptor of the feature. In the end, features are matched, often limiting the search space by exploiting available information of camera parameters and the environment. The general feature-processing pipeline is illustrated in Figure 2.5. [54]

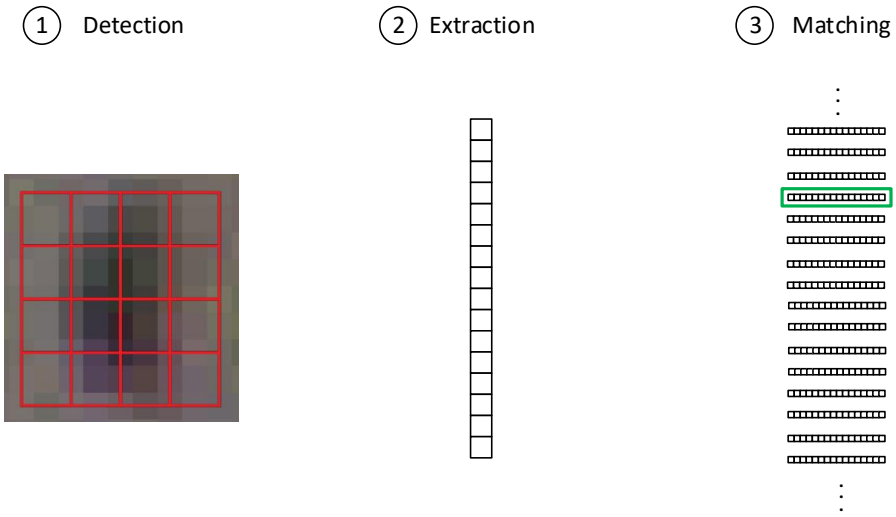


Figure 2.5. *General feature-processing pipeline includes detection, extraction and matching.*

2.3.1 Feature tracking

Instead of the individual matching and extraction process for each image, features can also be tracked from image to image. Tracking becomes a viable option especially when there is only a minor difference between the images, *e.g.*, in video data where temporally sequent frames have mostly similar content. After the initial detection and extraction of

features, they can be searched in the next image with the assumption that their location and content have stayed mostly the same.

Tracking can be done in a continuous manner, where the latest matched area is used in the next matching procedure. Other approach is to use the original feature area and apply a motion model to compensate for changes. The two approaches are illustrated in Figure 2.6. Regardless of the chosen tracking method, the tracking will eventually fail in a dynamic process. Consequently, the feature detection and extraction process have to be repeated ever so often. Overall, the most noteworthy advantages of tracking are the temporal smoothness it provides and its efficiency. [54]

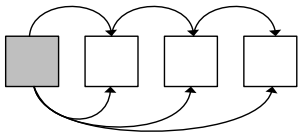


Figure 2.6. *Tracked feature can be continuously updated, or the original feature can be used with a transformation model.*

Kanade-Lucas-Tomasi feature tracking algorithm

KLT algorithm [55] tracks features between images based on the assumption that the differences between the images are small. The algorithm suggests that these minor changes can be approximated using a linear model, and this model is then used to guide the search for the feature match in the second image. Moreover, the accuracy of the calculated relationship model is estimated to assess the quality of the tracking.

The parameters for the relationship model are found via error minimization. The error term is the Sum of Squared Differences (SSD)

$$\text{SSD} = \sum_{\mathbf{p}} \left(\mathbf{J}_i[\mathbf{p}] - \mathbf{J}_{i-1}[f(\mathbf{p})]\right)^2 \omega(\mathbf{p}), \tag{2.8}$$

defined between the images transformed to the same space using the relationship model f. The weight function ω can be used to emphasize parts of the tracked area, *e.g.*, the center using a Gaussian.

In the simplest case, the relationship between the images is modeled using a displacement vector $\mathbf{v}$ so that $\mathbf{J}_i(\mathbf{p}) = \mathbf{J}_{i-1}(\mathbf{p} + \mathbf{v})$. A more complex relationship can be defined by adding an affine transformation matrix $\mathbf{T}$, so that $\mathbf{J}_i(\mathbf{p}) = \mathbf{J}_{i-1}(\mathbf{T}\mathbf{p} + \mathbf{v})$. Still, further additions to the model can be made. Nevertheless, this usually contradicts the general tracking use case, as increasing the number of model parameters makes it computationally heavy to optimize.

Assuming small difference between the images, the transformed image is approximated

using the first order Taylor approximation

$$\mathbf{J}(\mathbf{T}\mathbf{p}+\mathbf{v}) = \mathbf{J}(\mathbf{p}+\mathbf{B}\mathbf{p}+\mathbf{v}) \approx \mathbf{J}(\mathbf{p}) + (\mathbf{B}\mathbf{p}+\mathbf{v})\frac{\partial \mathbf{J}(\mathbf{p})}{\partial \mathbf{p}}, \tag{2.9}$$

where $\mathbf{B} = \mathbf{T} - \mathbf{I}$ is the deformation matrix and $\mathbf{I}$ is the identity matrix. Using this approximation in the error term, it is differentiated with regard to the model parameters to find such $\hat{\mathbf{T}}$ and $\hat{\mathbf{v}}$ that minimize the error. Due to the crudeness of the Taylor approximation, the minimization is done iteratively, starting from $\mathbf{B}_0 = [\mathbf{0}^T, \mathbf{0}^T], \mathbf{v} = \mathbf{0}^T$, and continuing until the error is below the set threshold.

KLT algorithm uses the simpler displacement vector model in the tracking process, assuming minimal deformation between sequent frames. However, in the KLT feature quality assessment, which is done between the current tracked region and the initial region, the affine relationship model is used, as deformation is likely to be present at this temporally longer scale. The complete KLT algorithm includes also the selection of good features to track, which is based on the eigenvalues of the gradient in the tracked region.

2.3.2 Harris

Harris algorithm [24] detects corner features in an image by analyzing the SSD between the image and its shifted version.

Similar to Equation 2.9 in the KLT algorithm, Harris algorithm uses Taylor expansion to approximate the transformed image as $\mathbf{J}(\mathbf{p}+\mathbf{v}) \approx \mathbf{J}(\mathbf{p}) + \mathbf{v}\frac{\partial \mathbf{J}(\mathbf{p})}{\partial \mathbf{p}}$. The SSD caused by a shift $\mathbf{v}$ in image coordinates can then be written in a matrix form

$$\mathrm{SSD}_{\mathbf{p}}(\mathbf{v}) = \mathbf{v}^T \begin{bmatrix} \left\langle \frac{\partial \mathbf{J}}{\partial x}\frac{\partial \mathbf{J}}{\partial x} \right\rangle_{\mathbf{p}} & \left\langle \frac{\partial \mathbf{J}}{\partial x}\frac{\partial \mathbf{J}}{\partial y} \right\rangle_{\mathbf{p}} \\ \left\langle \frac{\partial \mathbf{J}}{\partial x}\frac{\partial \mathbf{J}}{\partial y} \right\rangle_{\mathbf{p}} & \left\langle \frac{\partial \mathbf{J}}{\partial y}\frac{\partial \mathbf{J}}{\partial y} \right\rangle_{\mathbf{p}} \end{bmatrix} \mathbf{v} = \mathbf{v}^T \mathbf{M}_{\mathbf{p}} \mathbf{v}, \tag{2.10}$$

where the structure tensor $\mathbf{M}_{\mathbf{p}}$ is formed by the gradients $\langle\rangle_{\mathbf{p}}$, which are estimated using convolution with a gradient filter and summed up in the Gaussian window around $\mathbf{p}$.

Eigenvalue analysis of the structure tensor can be used to determine the nature of the intensity changes in the region. If either of the eigenvalues λ_1, λ_2 is large, the region is determined as an edge, since there is a strong gradient in one direction. If both eigenvalues are large, the region is determined as a corner having large gradient in all directions. The Harris corner measure H is thus determined as

$$H = \lambda_1\lambda_2 - \kappa(\lambda_1+\lambda_2)^2 = \mathrm{det}(\mathbf{M}) - \kappa \mathrm{tr}(\mathbf{M})^2, \tag{2.11}$$

where constant κ controls the balance between edge response and corner response. The second form of the equation uses trace tr and determinant det to avoid the inefficient eigenvalue computation.

2.3.3 Fast Retina Keypoints

Fast Retina Keypoint (FREAK) algorithm [1] is one of the frequently used feature extraction and matching methods. The algorithm is named after the analogies made in the original paper between the human retina and the used sampling pattern.

FREAK descriptor is formed by first sampling the point area using "retinal" sampling pattern, which consists of overlapping circles with higher density in the middle, density decreasing and radius increasing when moving further away from the center. The circles represent the receptive fields that are smoothed using Gaussians with σ equal to the circle radius.

The intensities between pairs of the receptive field are compared to form the FREAK descriptor, which is a string containing one-bit results of these comparisons. To make the algorithm efficient, the algorithm performs a predefined set of comparisons. This definition is data based: training data is formed by performing exhaustive comparison of the receptive fields for keypoints detected in variable image data. The effective pairs are then iteratively chosen by maximizing the variance in the training set, until 512 pairs are picked. To define orientation, 45 receptive field pairs symmetric with regard to the keypoint center are compared. Finally, FREAK descriptors can be matched effectively by first comparing the first bytes of the feature vector, ruling out most of the candidates.

2.3.4 Features from Accelerated Segment Test

Features from Accelerated Segment Test (FAST) [47] is a corner detection algorithm that compares pixel's intensity values to its surrounding pixels to decide whether it is a corner or not. FAST features are efficient to compute and can thus be used in real time applications.

The algorithm is based on the idea of comparing pixel's intensity to its surrounding pixels on the Bresenham's circle as depicted in Figure 2.7.

Surrounding pixels $\mathbf{p}_s \in \mathcal{N}_{\mathbf{p}_p}$ of the center pixel $\mathbf{p}_c$ are classified as

$$\underset{\mathbf{p}_s \in \mathcal{N}_{\mathbf{p}_c}}{\mathcal{C}(\mathbf{p}_s)} = \begin{cases} \text{darker} & \text{if} \quad \mathbf{J}(\mathbf{p}_s) < \mathbf{J}(\mathbf{p}_c) - \theta, \\ \text{similar} & \text{if} \quad \mathbf{J}(\mathbf{p}_c) - \theta < \mathbf{J}(\mathbf{p}_s) < \mathbf{J}(\mathbf{p}_c) + \theta, \text{ and} \\ \text{brighter} & \text{if} \quad \mathbf{J}(\mathbf{p}_s) > \mathbf{J}(\mathbf{p}_c) + \theta, \end{cases} \tag{2.12}$$

where θ is the similarity threshold. If there are more than N contiguous pixels in the circle that are all brighter or darker than the center pixel, the center pixel is classified as a corner. However, this evaluation is not efficient to calculate, especially for $N < 12$. Hence, FAST algorithm uses a machine learning approach to make a classifier simulating the comparison process.

Training data for the machine-learning algorithm is created by classifying arbitrary image

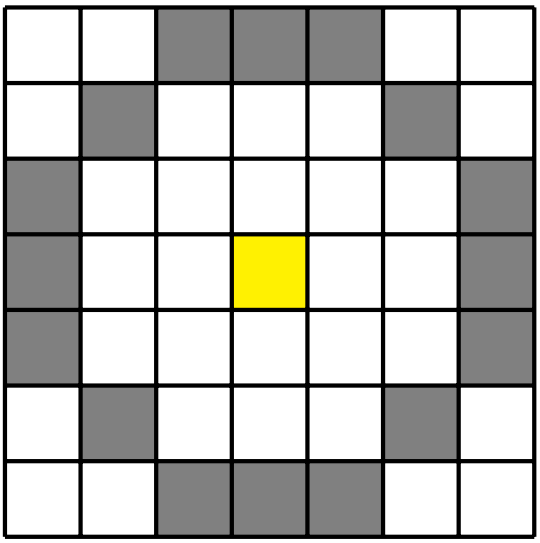


Figure 2.7. *Bresenham's circle used for FAST features. The 16 surrounding pixels of the yellow center pixel are drawn in gray.*

patches with binary {corner, not a corner} labels using the aforementioned algorithm. Then, the relative position on the Bresenham's circle that best separates the data is picked, and it is used to classify the pixels into the three subsets of darker, similar and brighter values. The separation and classification process is repeated for the subsets iteratively until every pixel is correctly classified. This effectively builds a decision tree, which is then used as the corner detector replacing the laborious calculation of the number of contiguous brighter or darker values. Additionally, a score function is used to receive only the maxima of the adjacent detected corners.

2.3.5 Binary Robust Independent Elementary Features

Binary Robust Independent Elementary Features (BRIEF) [5] is an efficient algorithm to calculate, match, and store features. The drawback of the basic form of the algorithm is its sensitivity to rotation and scaling.

BRIEF descriptor is calculated from a smoothed NxN image patch as a binary result of k pixel intensity comparisons. Smoothing is done using a Gaussian kernel to reduce sensitivity to noise. The arrangement of the k pixel comparisons is sampled from an isotropic Gaussian distribution $\mathbf{p}_1, \mathbf{p}_2 \sim \mathcal{N}([0,0]^T, \frac{N^2}{25}\mathbf{I})$, where $\mathbf{p}_1$ and $\mathbf{p}_2$ are pixel coordinates relative to the center pixel. The resulting descriptor is a k bit string, where the bits tell which of the compared intensity values were smaller. These BRIEF descriptors can be efficiently matched based on Hamming distance between the compared strings. Usually BRIEF is used as 16, 32, or 64 bytes version (k = 128, 256, or 512).

2.3.6 Oriented FAST and Rotated BRIEF features

Oriented FAST and Rotated BRIEF (ORB) algorithm [48] combines variants of FAST and BRIEF algorithm to form a robust descriptor that is efficient to compute and invariant to rotation.

ORB uses a rotation and scaling invariant version of the FAST algorithm described in Section 2.3.4. First, FAST keypoints are detected using a Bresenham's circle with a radius of nine. The detected keypoints are ranked using the Harris corner measure described in Equation 2.11, and N keypoints with the highest scores are picked. Second, scaling robustness is achieved by repeating the detection at different resolutions. Third, for rotational invariance, intensity centroid is calculated based on the moments of the keypoint area [46]. Rotation is estimated as the vector from keypoint center to the centroid.

BRIEF algorithm described in Section 2.3.5 is modified as well to make it robust to rotation. This is done by finding an uncorrelated set of binary tests based on training data, instead of the sampling from an isotropic Gaussian distribution that was done in the original version. These rotated BRIEF descriptors are calculated from the locations of the oriented FAST keypoints, forming the ORB features.

2.4 Photo Consistency

Photo consistency assesses uniformity of the compared image regions. Measures of photo consistency can be: 1) based on functional relations between images, 2) purely statistical, or 3) a combination of the two. In the context of our work, photo consistency is relevant especially in disparity estimation, camera trajectory estimation, and multi-view modeling.

Generally, photo consistency Q of a world point $\mathbf{w}$ within the field of view of two cameras j, k can be defined as

$$Q(\mathbf{w}) = \mathcal{C}\Big(\mathbf{J}_j\big(\mathcal{N}(\mathbf{P}_j\tilde{\mathbf{w}})\big), \mathbf{J}_k\big(\mathcal{N}(\mathbf{P}_k\tilde{\mathbf{w}})\big)\Big), \tag{2.13}$$

where
$\mathcal{C}$ photo consistency measure,
$\mathbf{P}_i\tilde{\mathbf{w}}$ projection of $\mathbf{w}$ into the image $\mathbf{J}_i$ (see Equation 2.1),
$\mathcal{N}(\mathbf{p})$ support domain around pixel $\mathbf{p}$, and
$\mathbf{J}_i(\mathcal{N})$ image intensities sampled within the domain.

Support domain $\mathcal{N}$ needs to be proportional to the resolution. Depending on the measure and the implementation, the compared domain may vary from subpixel sampling to global, whole image level.

In this section, we shortly discuss the photo consistency measures that we considered in developing our algorithm.

2.4.1 Intensity difference

In its simplest form, photo consistency can be calculated as SSD [23] or Sum of Absolute Differences (SAD). These measures assume that the world points manifest themselves as similar pixel values in the images. Namely, they assume that there are no bias or gain changes, and that the materials are approximately Lambertian: the surface has the same observed brightness regardless of the viewing angle, *i.e.*, its radiance is constant. SSD and SAD are applicable, *e.g.*, when using a stereo camera setup with two similar sensors and synchronized triggering.

Even if the similarity assumption holds, image sampling can produce problems in areas where intensity is changing rapidly. This can be compensated using subpixel resolution. Nevertheless, it comes at a high computational cost. Alternative solution to the sampling problem is Birchfield and Tomasi sampling insensitive measure (BT) that uses linearly interpolated intensity functions in the compared pixel areas to reduce the effect of sampling [2].

2.4.2 Normalized Cross Correlation

Normalized Cross Correlation (NCC) is a robust measure of photo consistency that can endure different relative scales and offsets, *e.g.*, in varying lighting conditions. It can be understood starting from the definition of covariance.

Covariance is defined as

$$\mathrm{Cov}(X,Y) = E\Big(\big(X - E(X)\big)\big(Y - E(Y)\big)\Big), \tag{2.14}$$

which is the expected value E of the product of the deviations between the variables X, Y and their expected values. Covariance measures linear dependence, however, its magnitude depends on the magnitudes of the variables. To address this issue, Pearson Correlation Coefficient (PCC) is a normalized version of the covariance, defined as

$$\mathrm{PCC}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E\big(X - E(X)\big)\big(Y - E(Y)\big)\big)}{\sqrt{E(X-EX)^2}\sqrt{E(Y-EY)^2}}, \tag{2.15}$$

where covariance is divided by the product of the standard deviations σ_X, σ_Y of the variables. PCC provides a proportionate measure of the linear dependence of the variables.

If the relationship cannot be assumed to be linear, correlation can be measured using, *e.g.*, Spearman's rank correlation coefficient. Spearman correlation uses the same formula as PCC. However, in Spearman correlation, the variables X, Y are converted to rank variables, *i.e.*, the variables are sorted in their value domain, and their place in the order defines the rank variable.

NCC in the photo consistency context is essentially equal to calculating the PCC for

discrete 2D signals X, Y, *i.e.*, two image patches in gray scale. NCC is defined as

$$\text{NCC}(X,Y) = \frac{\sum_m \sum_n (X_{mn} - \mu_X)(Y_{mn} - \mu_Y)}{\sqrt{\sum_m \sum_n (X_{mn} - \mu_X)^2} \sqrt{\sum_m \sum_n (Y_{mn} - \mu_Y)^2}}, \tag{2.16}$$

where μ_X, μ_Y are the mean patch intensities. It should be noted that NCC does not perform well in areas with low variance, and is undefined if either of the variances in the denominator are zero. Matching content near depth discontinuities is also problematic, as these areas have different content depending on the view, and thus do not correlate.

2.4.3 Mutual information

Mutual Information (MI) measures the amount of information one random variable contains about another. In the context of photo consistency, MI [27] tells how well information in one image can be predicted using another image. MI is insensitive to illumination changes, reflections, and other inconsistencies between the images. It can be used even between images coming from completely different types of sensors, *e.g.*, for matching Magnetic Resonance Imaging (MRI) images with regular color images.

MI between two image patches $\mathbf{J}_1, \mathbf{J}_2$ is the difference between their individual entropies and the joint entropy. Entropy H is defined as

$$H(X) = -\sum_{x \in S_X} p_X(x) \log p_X(x), \tag{2.17}$$

where S_X is the domain of of the variable X and p_X is its probability density function [9]. For discrete images, the probability density functions can be estimated using a joint probability histogram. The probabilities in this histogram are calculated as the sum of intensity value correspondences normalized by the number of pixels

$$P_{\mathbf{J}_1,\mathbf{J}_2}(i,j) = \frac{1}{|\mathcal{N}_{\mathbf{p}_c}|} \sum_{\mathbf{p} \in \mathcal{N}_{\mathbf{p}_c}} T\Big(\{\mathbf{J}_1(\mathbf{p}), \mathbf{J}_2(\mathbf{p})\} == \{i,j\}\Big), \tag{2.18}$$

$$T(\mathbf{p}) = \begin{cases} 1 & \text{if the intensities at pixel } \mathbf{p} \text{ correspond to } \{i,j\}, \\ 0 & \text{otherwise;} \end{cases}$$

$$\text{where} \begin{cases} P_{\mathbf{J}_1,\mathbf{J}_2} & \text{joint probability distribution} \\ i,j \in [\min(\mathbf{J}), \max(\mathbf{J})] & \text{possible intensity values in the images, and} \\ \mathcal{N}_{\mathbf{p}_c} & \text{compared area around pixel } \mathbf{p}_c. \end{cases}$$

An example of the formulation of the joint histogram is presented in Figure 2.8. For instance, calculating $P_{\mathbf{J}_1,\mathbf{J}_2}(1,2)$ for the presented image patches would be done as follows:

1. calculate the amount of pixels where there is the value '1' in the first image and the value '2' in the second image $\rightarrow 3$

2. divide by the patch size $\rightarrow 3/9 = 1/3$.

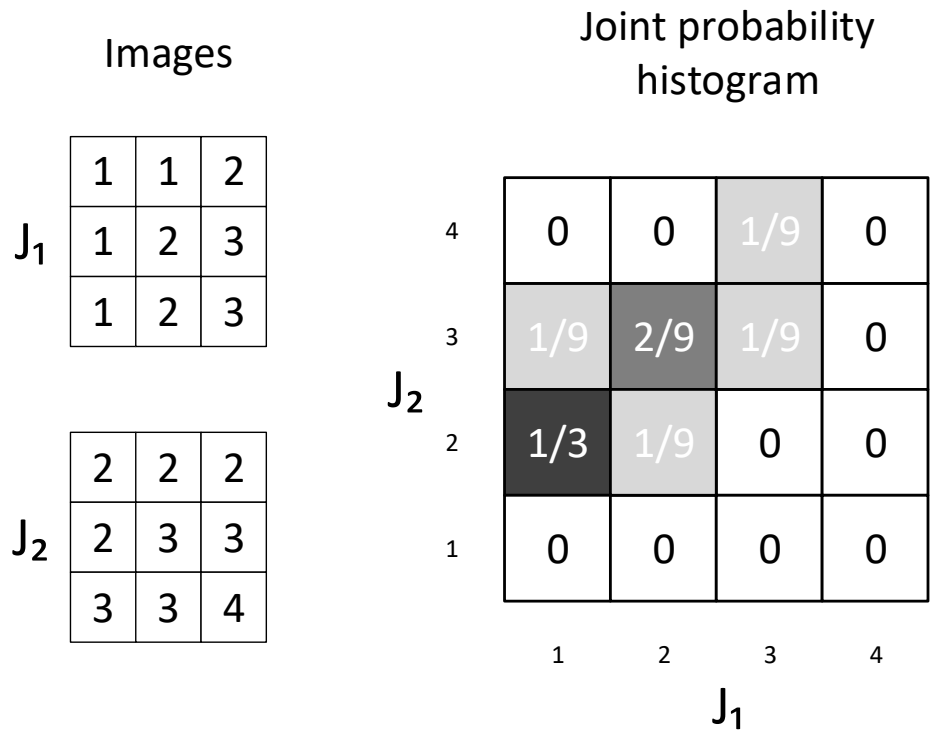


Figure 2.8. *Joint probability histogram for two $3x3$ images. Darker values indicate higher probability.*

Individual probability distributions $P_{\mathbf{J}_1}, P_{\mathbf{J}_2}$ can be calculated from the joint probability distribution by summing over the rows and the columns, respectively.

Probability density function is estimated from the probability distributions by convolution with a 2D Gaussian $\mathbf{G}$. Thus, entropy can now be calculated as

$$H = -\sum_{\mathbf{p}\in\mathcal{N}_{\mathbf{P}c}} \frac{\log(P * \mathbf{G}) * \mathbf{G}}{|\mathcal{N}_{\mathbf{p}_c}|}, \tag{2.19}$$

where $*$ denotes convolution. Plugging the individual probability distributions and the joint distribution in Equation 2.19, corresponding entropies are estimated. Finally, MI is defined as

$$MI = \overbrace{H_{\mathbf{J}_1} + H_{\mathbf{J}_2}}^{\text{individual entropies}} - \overbrace{H_{\mathbf{J}_1,\mathbf{J}_1}}^{\text{joint entropy}} \tag{2.20}$$

Low joint entropy indicates that there is a clear relationship between the images, *i.e*, $\mathbf{J}_1$ explains $\mathbf{J}_2$ well. The individual entropy terms in turn present complexity in the images. Thus, high MI score tells that the compared areas have high information content and are similar to each other.

MI assumes only statistical relationship between the images and thus can be considered to be related to Spearman correlation. To conclude, MI is robust to inconsistencies, efficient

to calculate, and works well with optimization functions.

2.4.4 Census transformation

Census transformation [57] is an ordering based transformation that, coupled with Hamming distance, can be used to measure photo consistency.

Census transformation is calculated by comparing the intensity values to the NxN neighborhood around them. A bit value is received for each comparison: the value is '1' if the compared neighborhood pixel has smaller intensity than the center pixel; otherwise, the bit is assigned to '0'. The bits are concatenated to form a string to complete the transformation. Photometric similarity between the Census transformed images can be assessed by calculating Hamming distance between the compared strings. The process is illustrated in Figure 2.9.

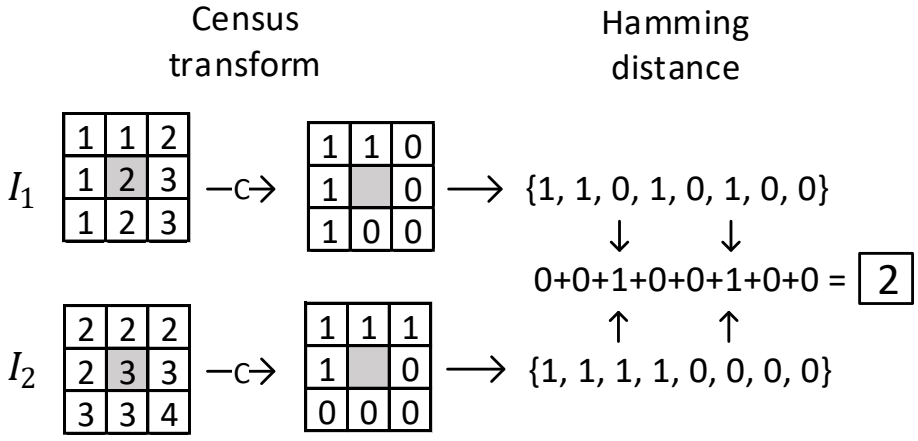


Figure 2.9. *An example of Census transformation and the calculation of Hamming distance.*

Census transformation is considered the most robust photo consistency measure for stereo vision [28]. As it depends on comparisons between local intensity values, Census is invariant under changes of gain and bias.

2.5 Point cloud reconstruction

2.5.1 Triangulation

When corresponding points are found in two images, the 3D location of the source of light $\mathbf{w}$ can be calculated as the intersection point between the two optical rays as presented in

Figure 2.10.

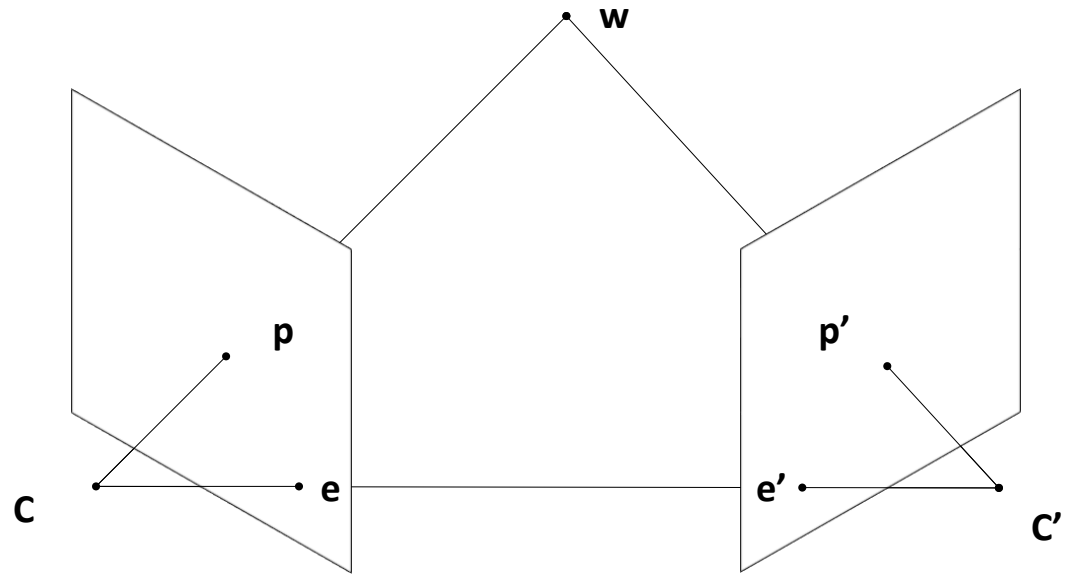


Figure 2.10. *In ideal case the 3D point $\mathbf{w}$ is found at the intersection point of the optical rays defined by the camera centers $\mathbf{C}$ and $\mathbf{C}'$ and corresponding pixels $\mathbf{p}$ and $\mathbf{p}'$ (see Equation 2.3); $\mathbf{e}$ and $\mathbf{e}'$ are the epipoles.*

However, due to inaccuracies in the found corresponding pixels and the errors of the camera parameters, the rays usually do not intersect. Thus, in triangulation we find the best estimate for $\mathbf{w}$ by minimizing its reprojection error in the images. We formulate the minimization problem as a search for pixels close to the found corresponding pixels $\mathbf{p}$ and $\mathbf{p}'$ that satisfy the epipolar constraint exactly. Under the assumption of Gaussian noise in the measurements, optimal solution to the problem can be reduced to finding real roots for a sixth degree polynomial [25].

2.5.2 Depth from disparity

For a calibrated stereo camera system, depth is calculated through disparity estimation, which is explained in Section 2.2. Knowing the parameters of the system, there is a direct correspondence between disparity d and depth $\mathbf{w}_z$. The resulting depth can then be used with the perspective projection model to calculate the x and y coordinates as

$$\mathbf{w}_z = \frac{lb}{d}, \qquad \mathbf{w}_x = \frac{\mathbf{w}_z(\mathbf{p}_x - \mathbf{c}_x)}{l_x}, \qquad \mathbf{w}_y = \frac{\mathbf{w}_z(\mathbf{p}_y - \mathbf{c}_y)}{l_y},$$

where b is the baseline of the stereo camera system.

2.6 Mesh reconstruction

In mesh reconstruction, the aim is to build continuous surfaces on top of the constructed point clouds. There exists a multitude of algorithms corresponding to different needs for input and output. In this section, we discuss the two algorithms we used to build a rough and a more detailed model based on the cloud.

2.6.1 Alpha shape surface estimation

One of the simplest ways to build a mesh around points is convex hull. It is defined as the smallest convex set that contains all the points and line segments connecting the points. Alpha shape surface estimation is based on a generalization of convex hull [12].

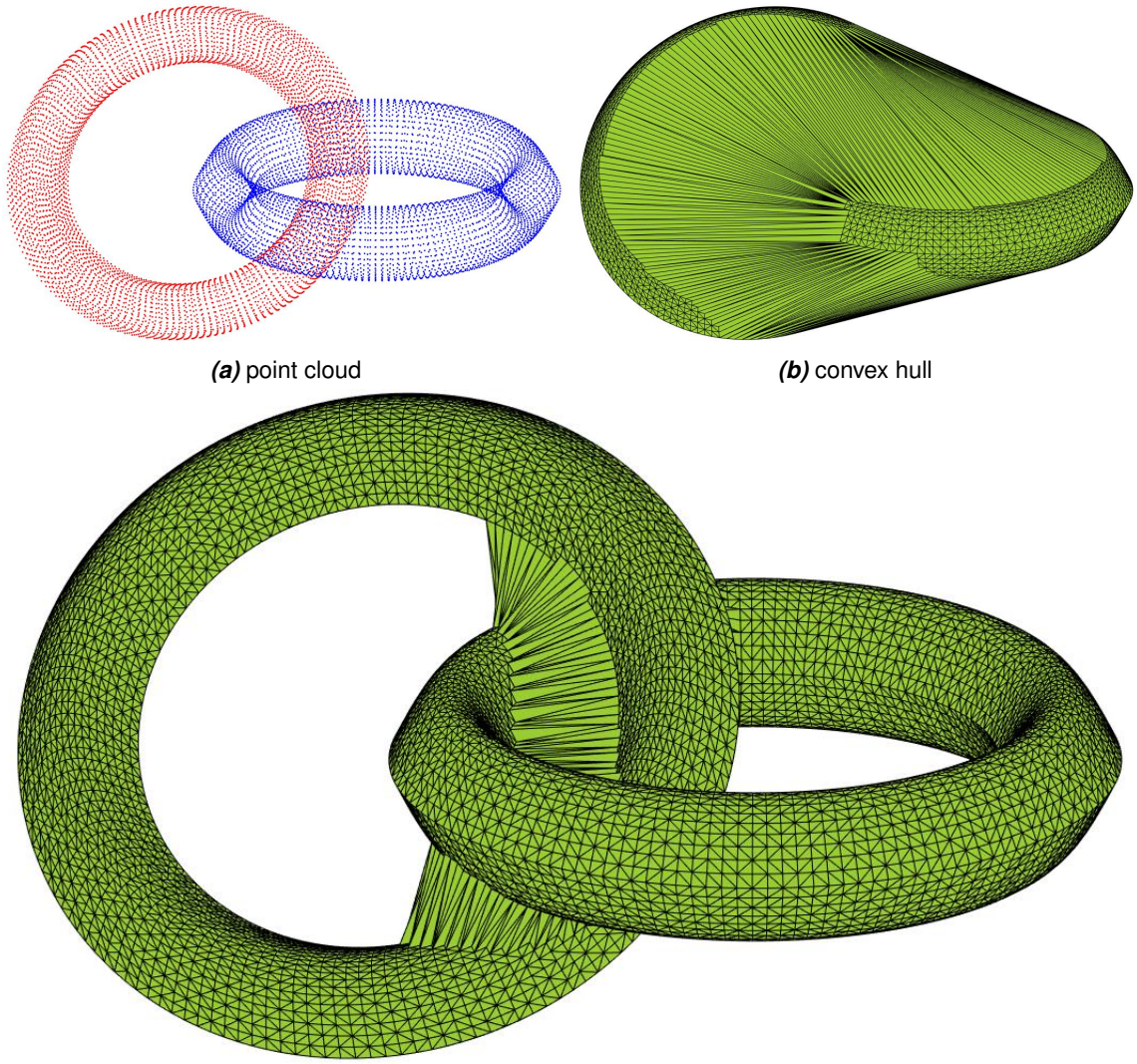
Alpha shape reconstruction introduces variable α to determine the detail of the reconstruction. Cavities where there are no points in the radius equal to α are left empty. Alpha shape reconstruction with $\alpha = \infty$ results in the convex hull. On the other end, reconstruction with $\alpha = 0$ produces the original point cloud. An example of alpha shape surface estimation is presented in Figure 2.11. Alpha shape is a simple reconstruction algorithm that through the variable α offers a convenient way to adjust the detail in the model based on the quality of the point cloud. [13]

2.6.2 Estimating point orientation

More refined mesh reconstruction methods usually require the point orientations as an additional input. There are point cloud creation methods that intrinsically assess the orientations in the measuring process, *e.g.*, some full waveform analysis LIDAR methods [39]. Nevertheless, generally, the raw point cloud data is unoriented, and only weak information related to orientation is provided, such as the camera poses in stereo vision.

The standard method for point normal estimation [29] starts by defining tangent plane for each point $\mathbf{w}$. Tangent plane is calculated from the N closest points to $\mathbf{w}$. Plane location is their centroid, *i.e.*, the mean of the point locations. Plane normal is defined as the third principal component of the data. Principal components are orthogonal vectors that maximize the variance in the data. The first component is pointing to the direction where the variance is the largest, and second component is orthogonal to the first, pointing to the largest left over variance. Thus, the third component is the normal of the plane formed by the first two components.

As Principal Component Analysis (PCA) leaves two possible directions for the point normals, propagation can be used to make their orientations consistent. The points are connected using Euclidean minimum spanning tree, which is extended to connect all the points in N neighbourhood. The graph is weighted based on the dot product between the



(a) point cloud ***(b)*** convex hull

Figure 2.11. *Alpha shape reconstruction represents the structure at the chosen level of detail. Convex hull corresponds to alpha shape reconstruction with $\alpha = \infty$. The limits of the algorithm can be seen in the large picture: since the two rings are close to each other, the algorithm is unable to separate them without breaking the rings' own structure.*

connected point normals $\mathbf{n}_i, \mathbf{n}_j$ as $\omega_{i,j} = 1 - |\mathbf{n}_i \cdot \mathbf{n}_j|$. Orientation is then propagated using the minimum weight-spanning tree, assigning consistent orientation from parent nodes to their children. [29]

2.6.3 Poisson surface estimation

Poisson surface estimation produces smooth surfaces without heuristics and tolerates a small amount of noise. The algorithm is based on finding a solution to Poisson's equation for a set of oriented points. [32]

First, a continuous vector field is formed from the oriented points. An octree of depth N is used to partition the cloud to nodes. This refers to recursively dividing the world

space into eight sub volumes until depth N. For each node, a node function is defined as an approximated Gaussian scaled and centered based on the node. To remove the discretization effect of the octree, trilinear interpolation is used to estimate the value of the vector field.

Indicator function f is defined as zero outside and one inside the model. The gradient of f is zero everywhere except on the surface of the model, where it is equal to the inward surface normal. Thus, the indicator function can be solved by minimizing the difference between the vector field and the gradient of the indicator function. The surface is then extracted from the indicator function globally adjusted to the point set. In the more recent screened Poisson version of the algorithm [33], the adjustment is done locally to reduce the over smoothing of the model. An example a model built using Poisson surface estimation is presented in Figure 4.10.

3 TUTLAB DATASET

TUTLAB dataset includes stereo video sequences recorded using a small robotic platform in static indoors environment, in good lighting conditions. The dataset was made for testing stereo algorithms and examination of the effect of different camera directions with regard to movement.

3.1 Sensor setup

The sensors that were used to record the dataset are presented in Figure 3.1.

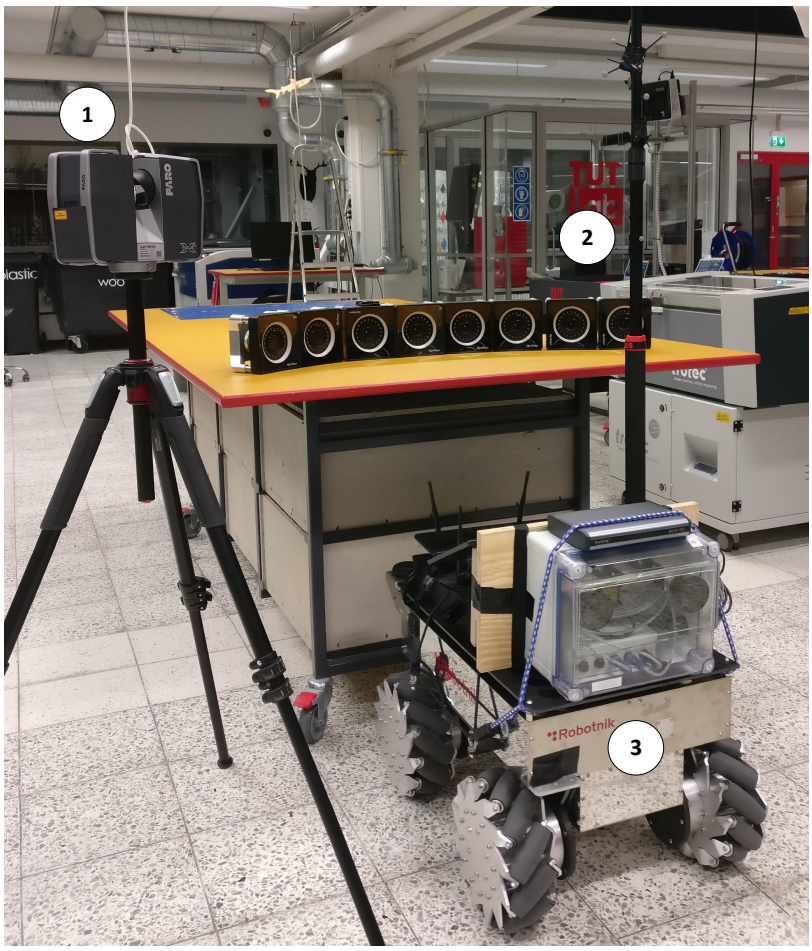


Figure 3.1. *The sensors used to record the dataset: 1) LIDAR; 2) motion capture cameras and a pole with reflective markers; 3) stereo cameras, IMU, and GPS.*

The robotic platform seen from all sides and more detailed measurements of the setup are shown in Figure 3.2.

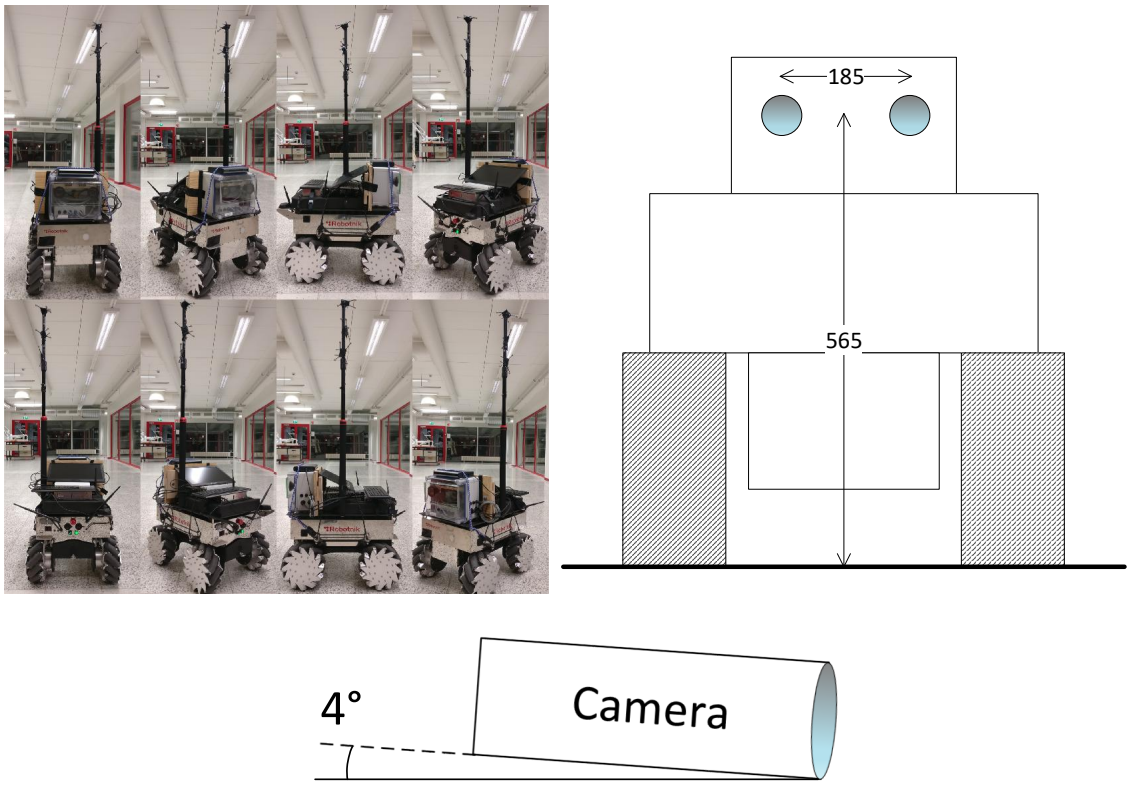


Figure 3.2. *Robot setup and measurements of the recording platform: 185 mm baseline, 565 mm distance from the ground, $4°$ tilt downwards.*

The technical details of our sensors are presented in Table 3.1.

Our main sensor was a 2.3 Mpx stereo camera setup with a baseline of 185 mm. The fairly large baseline provides greater disparity for the stereo images and thus increases the maximum depth detection range. We recorded at 50 FPS and with $\sim$ 20 ms exposure time to ensure minimal motion blur and abundant overlap in the images even at the full speed of the robot. We used lenses with a large FoV to get more information of the environment. To ensure simultaneous capture of the stereo cameras, we used a trigger signal.

To get the ground truth camera poses we used motion capture. As there were high obstacles in the recording area, we attached the reflective markers to the robot using a pole. Seven motion capture cameras were used to cover the recording area. However, the passive markers did not reflect enough light for the motion capture system for distances greater than five meters, which left some of the areas uncovered. An example of the ground truth camera poses is shown in Figure 3.3. Missing values can be seen as the gaps in the trajectory.

LIDAR was used to get the ground truth point cloud model of the environment. Our static setup on a tripod as seen in Figure 3.1 was placed in four central places of the laboratory to fully cover the area captured in the stereo video data. The resulting LIDAR point cloud is presented in Figure 3.4.

Table 3.1. *Details of the used equipment*

Equipment	Commercial name	Technical details
Camera x 2	Basler acA1920-50gc	2.3 Mega pixel (Mpx) 1/1.2" Sony IMX174 CMOS global shutter 50 Frames Per Second (FPS)
Camera lens x 2	VS Technology VC-0618H1	6 mm focal length 77.9° x 94.8° FoV
Motion tracking camera x 7	OptiTrack Prime 17W	1.7 Mpx 360 FPS 70° horizontal Mpx 2.8 ms latency
LIDAR	FARO Focus 3D x 130	0.6-130 m range 0.009° step size ± 2 mm ranging error
Robot	Robotnik SUMMIT-XL STEEL	Inertial Measurement Unit (IMU), Global Positioning System (GPS) 3 m/s omni directional wheels

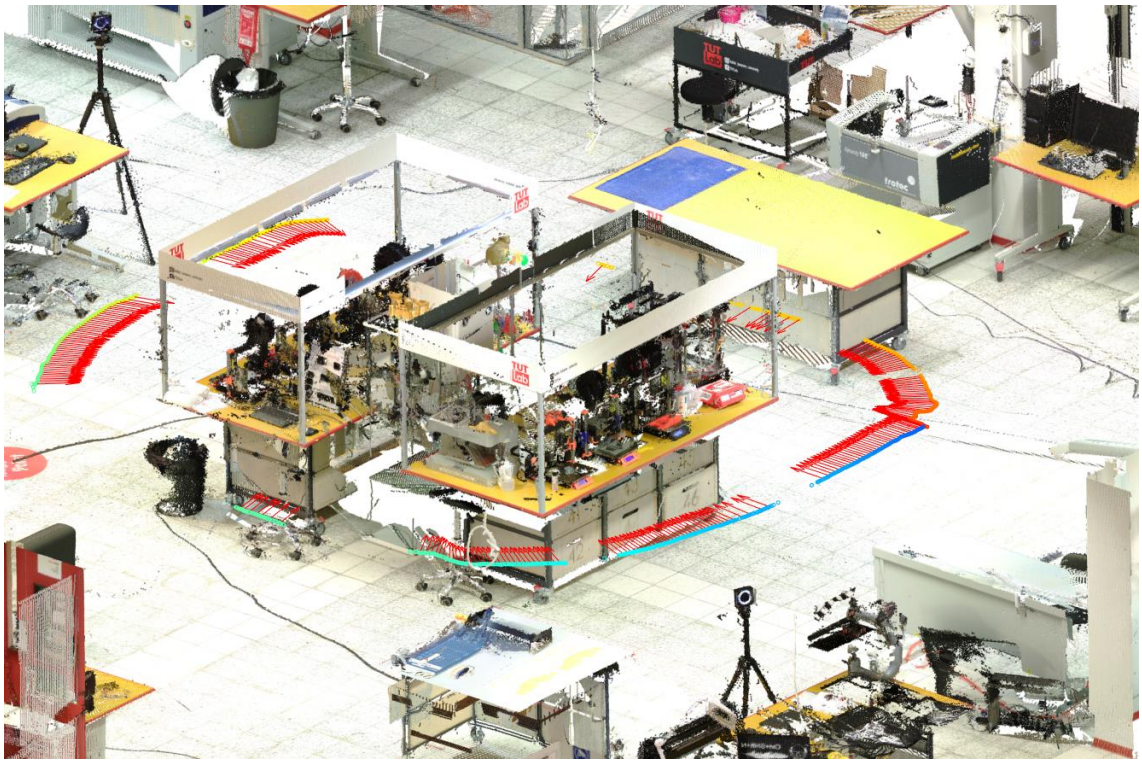


Figure 3.3. *Ground truth camera poses through motion capture manually aligned to the LIDAR point cloud. The trajectory is illustrated from start to end with the colored points from blue to red. The red arrows show the orientation of the camera.*

Figure 3.4. *The ground truth point cloud model received using LIDAR.*

The internal IMU and GPS data of the robot are also provided in the dataset. Nevertheless, especially the GPS data is inaccurate as the recording was done indoors. Example of the content is shown in Figure 3.5.

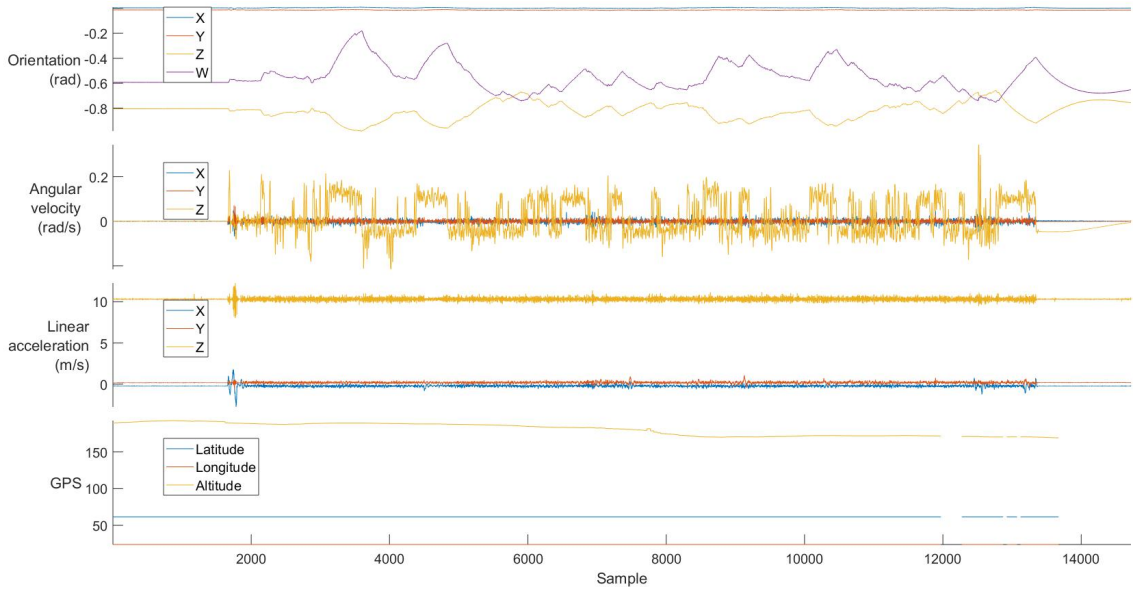


Figure 3.5. *Example of the collected IMU and GPS data.*

3.2 Dataset

3.2.1 Content

The dataset was recorded in TUTLAB, which is a fabrication laboratory in Tampere University of Technology. The environment was static except for the moving heads of the

3D printers. The area was well illuminated, which made the use of a low exposure time possible.

In the dataset, the robot goes around a large table using different types of movement. Using its omni directional wheels, the robot was driven with its cameras pointing in different directions with regard to the movement. The recordings include:

(a) a lap with the cameras pointing backwards,

(b) a lap with the cameras pointing towards the center of the table,

(c) a lap with the cameras pointing away from the center of the table.

(d) two laps around the table: first with the cameras pointing forward, and second with a small angle with regards to the moving direction, and

(e) two laps similarly in the reverse direction.

3.2.2 Data structure

The structure of the dataset is presented in Figure 3.6.

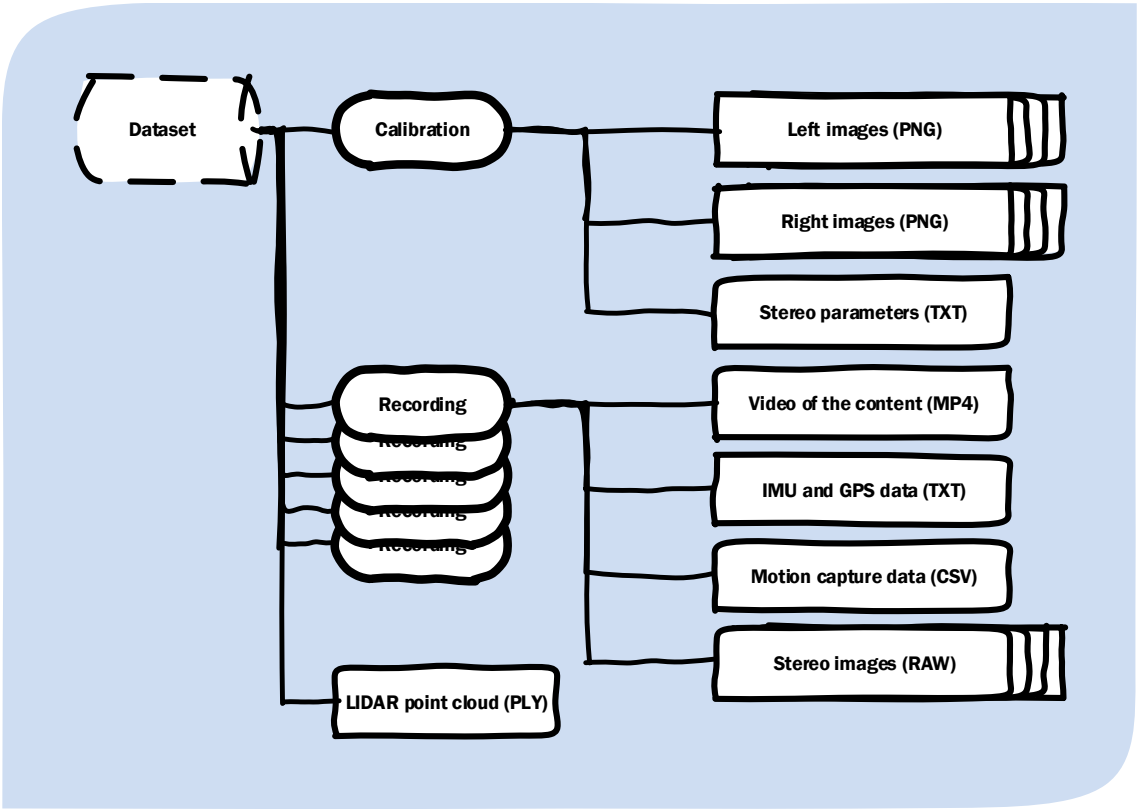


Figure 3.6. *Hierarchical structure of the dataset.*

The calibration data includes stereo images with a photogrammetric object, and the precomputed stereo parameters. The five recordings comprise the raw stereo data, the motion capture data, and the recorded IMU and GPS data. Moreover, a thumbnail video

is provided. Finally, there is the ground truth LIDAR point cloud, and example Matrix Laboratory (MATLAB) functions for handling the data.

3.3 Synchronization and calibration

3.3.1 Synchronization

We used four separate systems in the recording of the dataset.

1. **Stereo video** (50 FPS) capture was done with C code using Open Computer Vision (OpenCV) library on a Personal Computer (PC) on top of the robot;
2. **IMU and GPS** (50 FPS) were captured using a MATLAB script on the same PC connected to the robot's internal computer using Robot Operating System (ROS);
3. OptiTrack's Motive software was used for **motion capture** (120 FPS) on a separate PC;
4. **LIDAR** scanning was done using the scanners own software.

As there was no built-in synchronization in the recording process, we did it manually in the post processing of the data. Time stamps were recorded for all the data, and the clocks were assumed to have a constant rate. This allowed us to synchronize the data by finding the starting point of the robot movement from the raw data. Starting point can be found from the optical flow in the image data, the first peak in acceleration in the IMU data, and the first deviation larger than the maximum error of the motion capture system.

The timestamps of the image files were written at the time of writing on the disk rather than at the moment of capture. As there was considerable buffering in the writing process, we assumed constant 50 FPS and used the image indexing for the synchronization of the frames after finding the starting frame. This assumption was based on the fact that there were no frames dropped during the recording process and the variance in the frame rate was minimal.

3.3.2 Camera calibration

We recorded calibration data for the stereo cameras using the same system used to record the actual data. We picked around a hundred stereo frames where the calibration object thoroughly covers the imaging area of the cameras, and used MATLAB's application for stereo camera calibration to calculate the parameters. As the calibration object, we printed a chessboard pattern with a grid size of 10 cm x 10 cm. Examples of the calibration images are presented in Figure 3.7.

We positioned the chessboard in a way that there was no coplanarity, as the algorithm assumes six degrees of freedom in the movement. The used calibration algorithm is

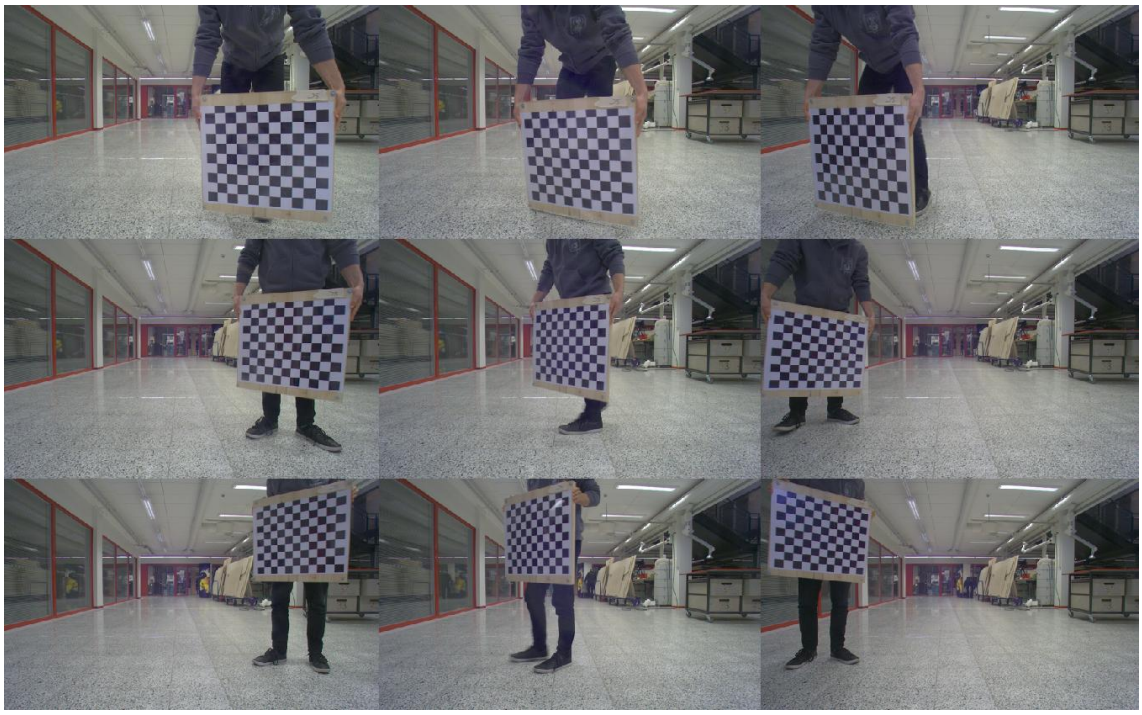


Figure 3.7. *Examples of the calibration images.*

discussed in more detail in Section 2.1.3.

4 PROPOSED APPROACH

In this chapter, we first give a brief overview of the proposed approach. We then discuss the components of the algorithm in detail in their respective sections.

4.1 Overview

The proposed system is summarized in Figure 4.1.

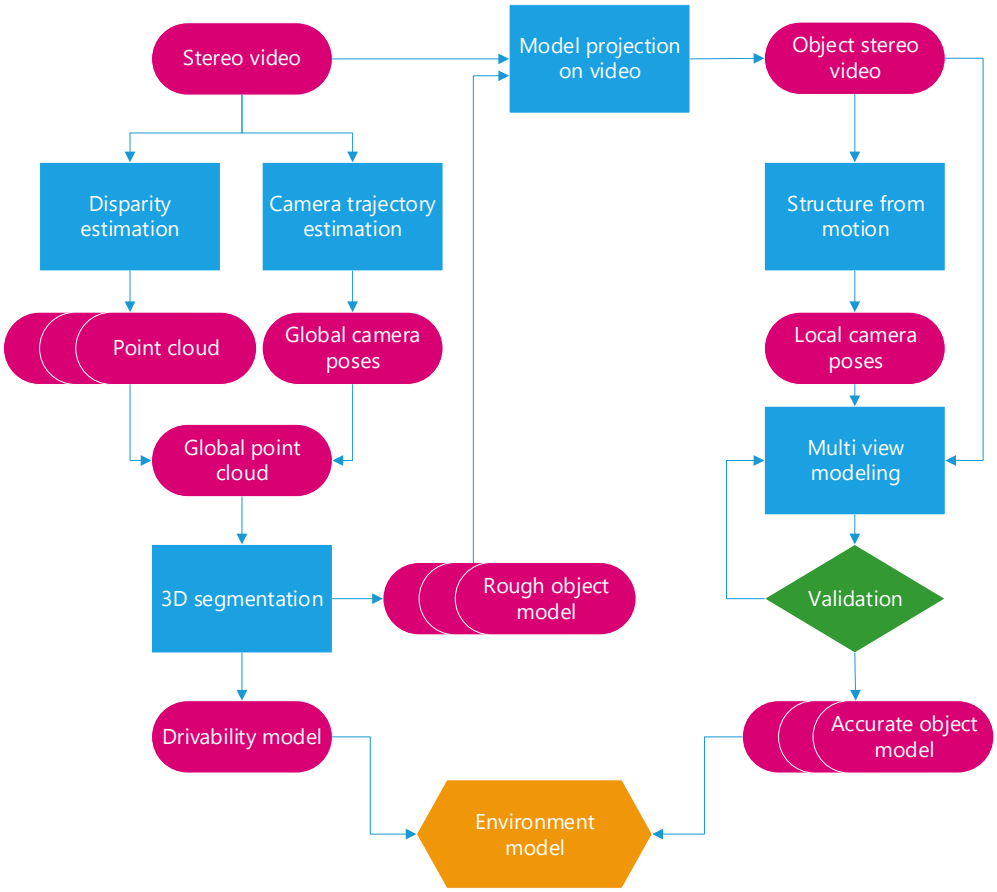


Figure 4.1. *Overview of the system*

Using a calibrated stereo camera system, we get stereo video data as input. We rectify the stereo image pairs and calculate corresponding disparity maps. We then build up 3D

point clouds based on the disparity maps. By tracking features, we estimate the camera movement, and merge the individual point clouds to a global cloud.

Starting from the global cloud, we build an environment model. We fit a spline model to the global cloud and segment drivable surface and obstacles based on the steepness of the model. We further section the point cloud in the obstacle area to separate individual object point clouds using available 3D information. Finally, we build rough object models based on the segmented point clouds.

To improve the raw 3D models, we project them back to the original stereo video and segment the image areas where the object is visible. Using SfM, we locally estimate camera movement from the object video. Based on the camera poses and the quality of the video, we choose representative keyframes to combine the available information of the object to form an accurate object model.

Essentially, we form a multi-view problem that we solve using a patch model approach. Patches are small slanted planes that we assume to represent the local structure of the object. We optimize the locations and orientations of these patches by minimizing projected photo consistencies in the image data. We then build a mesh around the acquired model, and add texture by projecting the model back to the image space. The final result of our algorithm is a rough environment model that can be used for movement, and more detailed object models for recognizing and handling the objects.

4.2 Stereo algorithm

4.2.1 Disparity estimation

Calibration

First step in the pipeline is the calibration of the cameras. We use MATLAB'S implementation of stereo camera calibration, which is based on the photogrammetric method proposed by Zhang [61]. We photograph the calibration object, which is in our case a chessboard pattern with known grid size, from multiple views to obtain the stereo camera parameters. As a result, we get the intrinsic parameters of the cameras and stereo parameters describing the relationship between them. The method is accurate and requires little effort to carry out.

Calibration provides us the distortion parameters that we can use to undistort the images as discussed in Chapter 2.1.2. Undistortion removes nonlinearities in the images, and thus enables the use of the pinhole model to relate pixels from one camera to the epipolar line in the image of the other camera.

Rectification

To simplify the matching process we rectify the stereo images, which means that we geometrically transform the images so that the epipolar lines run parallel between the images. The rectification algorithm is discussed in Section 2.2.3. Rectification is practical to do as it simplifies the correspondence search done in disparity estimation.

Disparity calculation

After rectification, we calculate the disparity map from the image pair via semi global matching. The basic idea of semi global matching is to form an approximation of the global, 2D smoothness constraint by combining many 1D constraints along paths from different directions. The details of the algorithm are discussed in Section 2.2.4.

We use MATLAB's block wise implementation, which compared to the pixel wise calculation provides less noise at the cost of lost matches in areas with depth discontinuities. The matching cost used is SAD which, as discussed in Section 2.4, is not robust. Nevertheless, it is efficient to calculate and accurate when there are minimal differences between the images [28], as is in our case.

SGBM provides good results and can be implemented to run in real time, especially if run on specific hardware (Field Programmable Gate Array (FPGA) or Graphical Processing Unit (GPU)). Performance of the algorithm depends on the number of disparities considered and the number of directions used in the path wise aggregation.

Disparity filtering

We calculate two disparity maps: first, using the left image as the base image and match right image to it, and then vice versa. This allows us to perform a consistency check to invalidate inconsistencies caused by occlusions and noise as discussed in Section 2.2.4. Moreover, to account for the limitations of our system due to the baseline and sampling, we invalidate disparities corresponding to distances larger than 12 meters. Finally, we calculate the point cloud corresponding to the disparity map as explained in Section 2.5.

4.2.2 Camera trajectory estimation

We estimate camera poses from tracked features by minimizing the reprojection error of points triangulated from the feature locations.

Feature detection, extraction, and tracking

In the beginning of our camera trajectory estimation algorithm, we detect corner features in the images using FAST algorithm, which is discussed in more detail in Section 2.3.4. We utilize FAST features, as they are efficient to compute without trading off accuracy. FAST algorithm can be run in real time, which is essential in our algorithm. At the keypoint locations found by FAST, we extract FREAK descriptors. They are efficient to compute and robust, as discussed in Section 2.3.3. The specific algorithms used for feature detection and extraction are not set in stone. For instance, ORB algorithm discussed in Section 2.3.6 could be considered to provide more robustness.

To find stable features for camera pose estimation, we filter the feature space using circular matching as illustrated in Figure 4.2. In our implementation of circular matching, we use

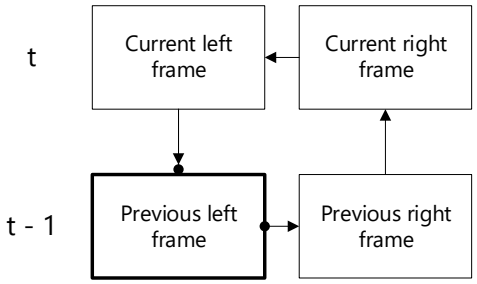


Figure 4.2. *Circular matching*

the KLT feature tracking discussed in Section 2.3.1. Starting from the feature candidates in the previous left frame, we track them to the previous right frame. We continue the tracking to the current right frame, and then to the current left frame. Finally, we complete the circle by returning to the previous left frame. Features that are tracked back close to their origin (< 0.5 pixel error) are accepted. [21]

We carry on filtering the features based on their age and strength. Along the tracking process, we weight the features based on the number of consecutive frames they are in, *i.e.*, their age. We increase the weights until N frames ($N = 15$ in our implementation), in which case we revert them back to the smallest value. We pick one representative feature in every 50x50 block; first criteria being the weight, and second the feature strength.

Our weighting system, based primarily on the feature age, ensures that old feature tracks, which are more likely to be rotationally correct, are taken into account. The old features generally correspond to objects geometrically further away, persisting in the FoV despite movement. This is also why they are not that good for estimating translation, for which features closer to the camera are needed. The eventual drop to the minimum weight for the oldest features guarantees that we take newer features into account for this purpose. As a whole, the weights offer a good balance between translation and rotation estimation.

Redetection of features

After the initial detection of features, we repeat the detection process when there are not enough tracked features left. Features are left out evidently when they go out of the FoV. Moreover, we also discard features in the circular matching process.

In our redetection implementation, we bucket the features to perform the detection only in areas where new features are needed. We divide the image in 50x50 blocks, and repeat the detection in blocks where there are no features left. To keep our algorithm simple, we carry out this step by doing the detection to the whole image and masking new features in blocks where we still have old features.

Pose computation

Ending up with filtered features uniformly distributed across the image, we calculate the pose based on these features. We triangulate the world locations of the features in previous view as explained in Section 2.5.1, using the known feature locations and camera parameters. We then estimate the new camera poses by minimizing the reprojection error of the triangulated points in the current view.

We use Random Sample Consensus (RANSAC) approach, where we iteratively sample the feature space and minimize the reprojection error for the subsamples of the data. We evaluate the resulting camera model using the whole data and preserve the best model. We end the iterative process when the probability of outliers for the model is small enough, or when we reach the maximum number of iterations.

For the reprojection error minimization, we use MATLAB's implementation of the Levenberg-Marquardt algorithm. We use quadratic loss function as the cost, and project to the left camera only. Thus, our cost function is defined as

$$\mathcal{C}([\mathbf{R}|\mathbf{t}]) = \left(\mathbf{F}_{\mathbf{J}_L} - \text{Cart}(\mathbf{K}_L[\mathbf{R}|\mathbf{t}]\mathbf{X})\right)^2,$$

where
- $[\mathbf{R}|\mathbf{t}]$ camera pose to optimize,
- $\mathbf{F}_{\mathbf{J}_L}$ matrix containing detected feature locations in the current left image,
- $\mathbf{K}_L$ is the intrinsic matrix of the left camera,
- $\mathbf{X}$ is the matrix of triangulated locations from the previous view, and
- Cart is the transformation from world coordinates to image (Cartesian) coordinates, where each column of the input matrix is divided by the corresponding entry of the third row.

Levenberg-Marquardt is the standard optimizer used for nonlinear least squares problems such as ours, and quadratic loss function is the most commonly used loss function for it. We also considered Huber loss, which is a combination of squared loss and mean absolute

error, where the Huber cost factor defines the boundary between the two. However, the cost factor varies depending on the data and thus, brings in redundant complexity to the algorithm.

We experimented also reprojection to both left and right image planes to have a more complete error assessment. This turned out to provide negligible improvement with regard to the performance loss.

4.2.3 Data segmentation

Input data for segmentation is the global point cloud, which we form by combining the individual clouds derived from the disparity maps. Before merging, we transfer the clouds to the global coordinate system based on the calculated camera movement.

Drivability model

We fit a surface model to the global point cloud data using cubic spline interpolation. The spline model consists of piecewise third degree polynomials, which helps to mitigate the oscillation phenomenon that emerges when using models that are more complex. We rotate the point cloud to have the gravity vector codirectional to the z-axis, and sample the spline model at the selected grid spacing in the xy-plane.

Depending on the input data, we carry out following filtering steps. 1) As the spline model cannot represent drivable cavities, we exclude points above height h from the fitting. 2) For noisy point cloud data, we estimate normals for the points by fitting local planes to the neighboring N points. We then exclude points whose estimated normals deviate more than α from the gravity plane. 3) We smooth the acquired model using sliding median filtering.

We label the model in the xy-grid as 1) drivable, 2) obstacle, 3) safety area, or 4) uncertain. We label model areas that do not have real points in radius r as uncertain, since they are based on interpolation. Using sliding window, we take the highest and lowest point in the neighborhood and determine the area drivable or obstacle based on the steepness angle α defined as depicted in Figure 4.3. Finally, we set the areas that are close to the obstacles as safety areas. An example of the spline model is represented in Figure 4.4.

If the drivable surface can be assumed to be planar, *e.g.*, in indoors areas, we use plane fitting as an alternative for the drivability estimation. We label the fitted plane based on the points projected to it. 1) We define the area as an obstacle if there are points in the drivable volume above the area, *i.e*, above the confidence interval of the plane and below the height of the vehicle. Moreover, areas that have projected points below the plane and and not enough points inside the plane confidence interval are labeled as obstacles. 2) The area is labeled drivable if there are enough points inside the plane confidence interval

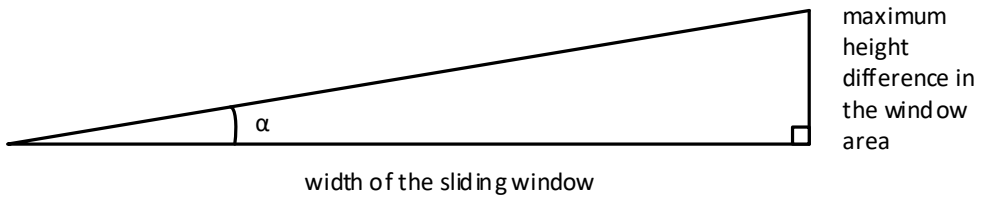


Figure 4.3. *Steepness of the model is determined using simple trigonometry.*

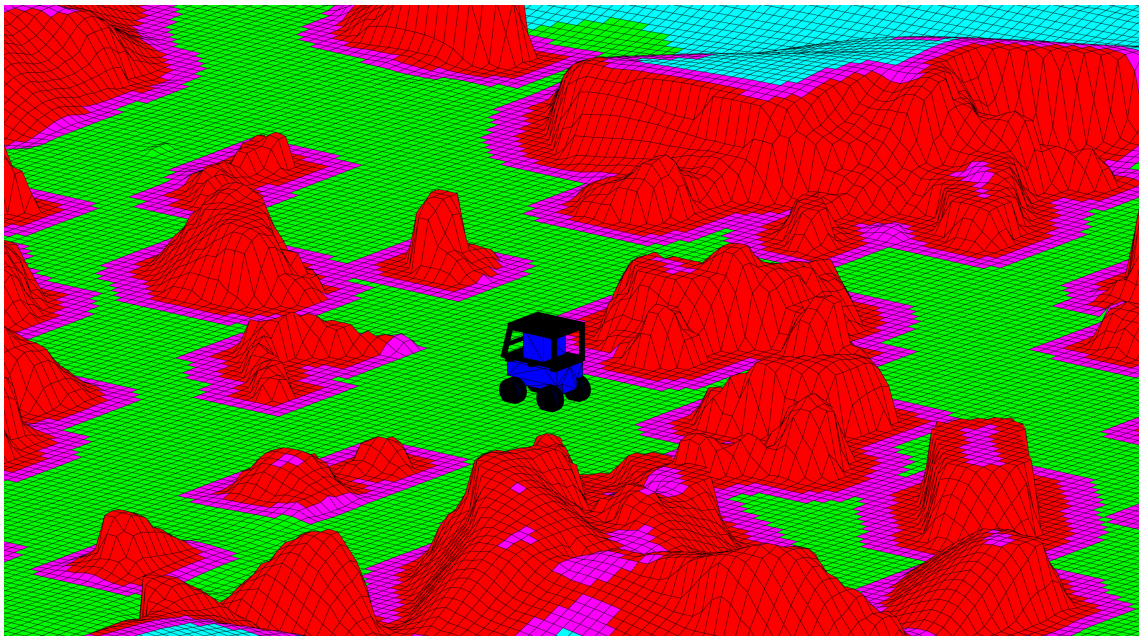


Figure 4.4. *We use spline model fit to the global cloud to estimate drivability for the robot. Drivable areas are drawn in green, obstacles in red, safety areas in purple, and areas with insufficient information are in light blue.*

and it was not labeled as an obstacle. 3) Safety areas and 4) uncertain areas are defined similar to the spline model.

Rough object models

After drivability estimation, we continue by refining the interesting areas of the model, *i.e.*, the obstacle areas. We segment the global cloud to object clouds based on connectivity of the obstacle areas in the xy drivability map. These object clouds include noise due to the limitations of stereo vision and drift in estimated camera movement. Examples of the rough models are presented in Figure 4.5.

The models are accurate enough for moving around in the environment without bumping to objects. We can also make rough estimates of their geometry, such as volume and orientation. Object recognition based on the rough models would be rather challenging

Figure 4.5. *Examples of the rough 3D object models segmented from the global point cloud.*

even with a specific library of possible objects. For interaction purposes that require delicate contact with the object, we need models that are more detailed. An example of the acquired rough environment model containing the estimated drivable surface and the object models is presented in Figure 4.6.

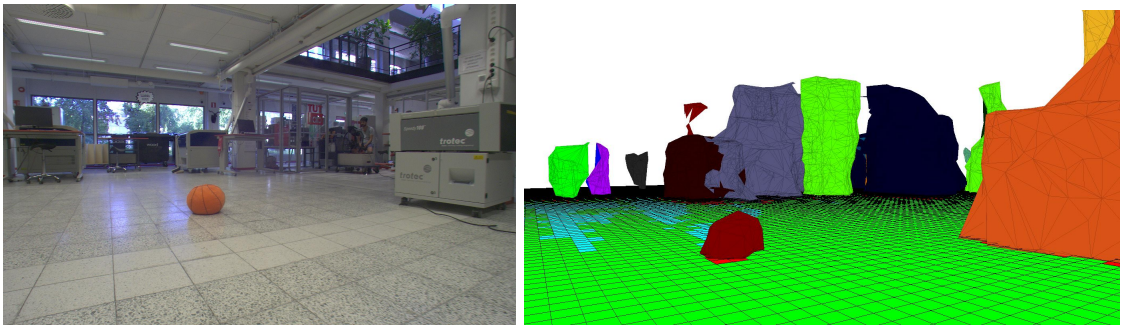


Figure 4.6. *Same view from the left camera and from the constructed rough environment model. Colors of the objects have been randomly assigned.*

4.3 Multi-view algorithm

The main criteria considered in this part of the algorithm is the accuracy of the model. Performance is less of an issue as we can wait for the model to be processed before interaction. Situation is different from the drivability model where real time processing is needed in order to move in a dynamic environment.

4.3.1 Keyframe selection

To improve the accuracy of the object models we return to the image data to combine all available information of the object. We form a rough mesh model on top of the object point cloud using MATLAB's implementation of alpha shape reconstruction discussed in Section 2.6.1. It provides a simple and efficient way to estimate the rough shape of the object.

We project the constructed mesh models to the stereo images, and pick and mask the frames based on the projections. The objective of this processing step is to remove abundant information that would slow down further processing and possibly lead to noisy results. We leave the models rough on purpose to ensure that the real object is fully covered.

To remove abundant data, we pick frames to represent the object called keyframes. Criteria that we consider in the keyframe selection are as follows:

- sufficient overlap - enough same content in the images,
- sufficient baseline - enough distance between the cameras (shorter baseline gives more uncertainty),
- degeneracy avoidance - motion degeneracy (camera rotating without translation) and structure degeneracy (coplanar points in physical space, e.g., wall),
- number of frames - diminishing returns after sufficient level of coverage is reached,
- motion blur - greatly reduces the amount of extracted features, and
- minimizing reprojection errors.

The keyframe selection could be done automatically based on the global camera trajectory and the analysis of the image data. If data is insufficient, more information could be ordered to be collected. As our implementation is only demonstrative, we pick the keyframes manually to ensure proper coverage.

4.3.2 Structure from motion

We use MATLAB's implementation of SfM to get camera parameters that are locally accurate with regard to the object. This is essential to do as we base our multi-view matching process on epipolar constraints with small margin for error. In SfM, we optimize the camera parameters by simultaneously forming a sparse model of the object, and minimizing the model's reprojection error in the images.

4.3.3 Multi-view modeling

We combine the obtained information of the object using multi-view stereopsis, following the approach of Furukawa and Ponce [17].

Features

We start by detecting and extracting features in the images. To get comprehensive coverage, we combine corner features detected using Harris corner detector discussed in Section 2.3.2, and blob features using a Difference of Gaussians (DoG) based algorithm.

DoG is defined as the difference between 2D convolutions ($*$) on the image $\mathbf{J}$ using two discrete 2D Gaussian filters $\mathbf{G}$ with standard deviations σ and $\kappa\sigma$. The ratio $\kappa = 1.6$ is chosen to provide an approximation of the Laplacian operator [41]. Filter size is defined proportional to the standard deviation as $s = 2\lceil 2\kappa\sigma \rceil + 1$ to contain the significant portion of the Gaussians. As convolution is distributive, the DoG operator can be noted as

$$D_i = |(\mathbf{G}_{\sigma_i} - \mathbf{G}_{\kappa\sigma_i}) * \mathbf{J}|.$$

We calculate the DoG response for three different deviations $\sigma_i \in \left\{2^{\frac{i}{2}}K, i = 0, 1, 2\right\}$, where K determines the detected blob size, and is chosen based on the image resolution. Using a $3x3$ sliding window, we find the local minima and maxima of the responses. Finally, we select as features only the points that have the strongest response to the middle-sized DoG filter D_1, and thus, correspond specifically to the Gaussian of this size. The process is illustrated in Figure 4.7

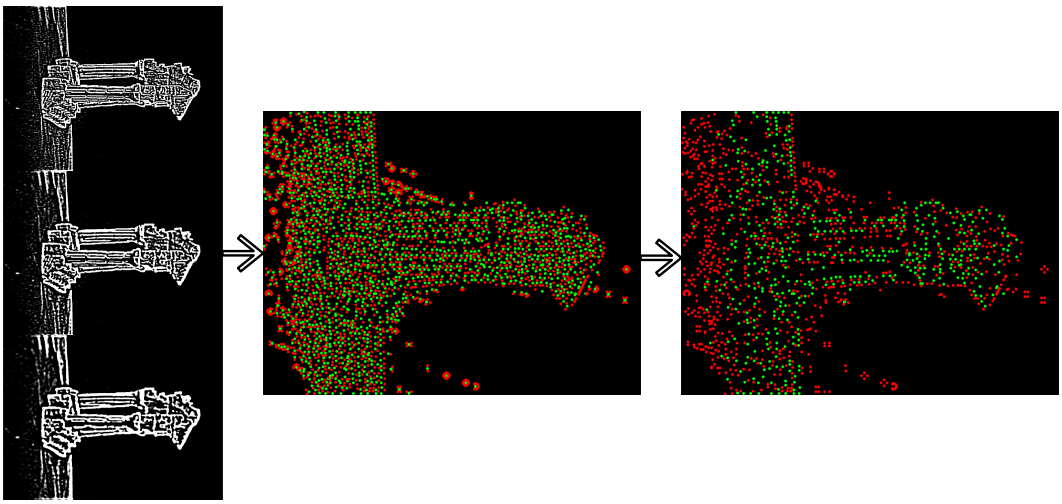


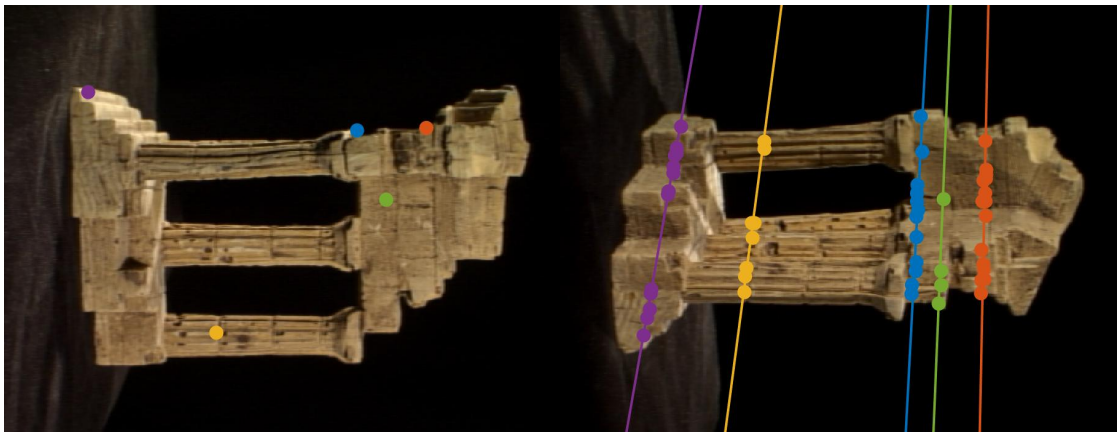
Figure 4.7. *Steps of our DoG algorithm. First, three responses are calculated for Gaussians with different deviations. Then local minima (red) and maxima (green) are selected using sliding window. In the end, those minima and maxima that are the strongest for the middle sized Gaussian are preserved.*

We assure an even distribution of features by dividing the image into a γ sized grid and

selecting the ρ strongest features of both types from each image block.

Patch initialization

Starting from the features in one image, we search epipolar matches in other images. This means that we pick all the features whose location differs less than N pixels from the epipolar line corresponding to the feature location in the reference image. We do the matching only between features of the same type. An illustration of the process is presented in Figure 4.8.



(a) reference image ***(b)*** matched image

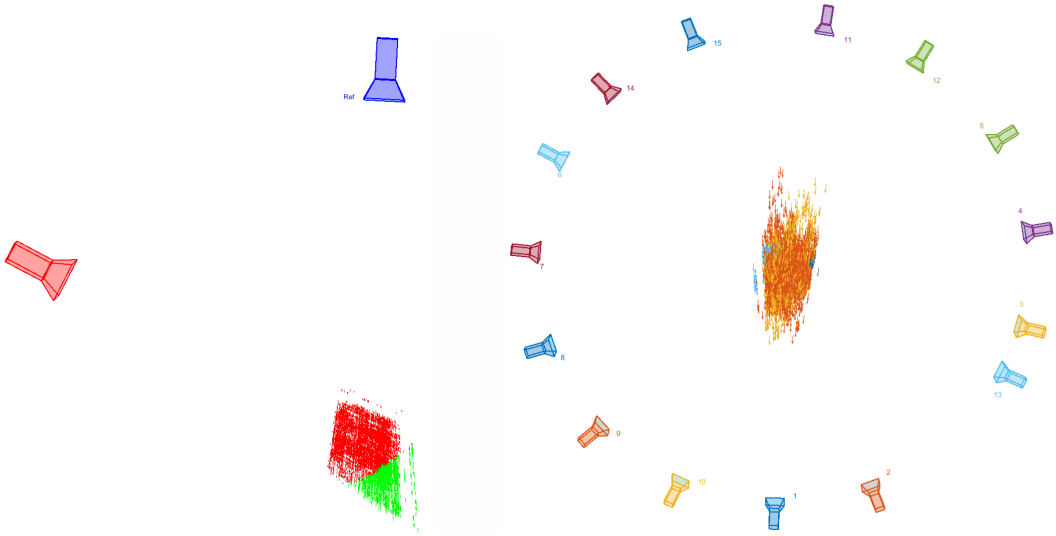
Figure 4.8. *Subset of features in reference image and corresponding matches in the other image. All features that are < N pixels from the epipolar line are considered as match candidates.*

Based on these candidate matches we form patches, which are small slanted planes. First, we triangulate the locations of the patches from the feature locations of the candidate pairs. Second, we initialize the patch normals to be pointing towards the reference camera. Finally, we define the corners of the patch so that its projection in the reference image is the kxk pixel area around the feature location.

Patch filtering

To filter out the unlikely matches from the candidate patches, we remove those whose normal differs more than 60° from the vector pointing from the patch toward the other camera. We calculate the visibility in other cameras in the same manner. Filtering process is demonstrated in Figure 4.9.

We estimate the photo consistency in visible cameras for the remaining patches. We project the patch model to the visible cameras and calculate NCC (discussed in Section 2.4.2) between the reference image and the projected image. If there are more than m cameras where photo consistency is estimated below κ, we accept the patch for the



(a) visibility filtering ***(b)*** photo consistency filtering

Figure 4.9. *(a) Patch normals are initialized to point towards the reference camera (blue). The patches for which the angle between the vector from patch center towards the matched camera (red) and the patch normal is greater than $60°$ are filtered out (the red portion). (b) Patches after photo consistency filtering.*

optimization procedure.

Patch optimization

Ending up with a set of potential candidates for the patches, we find the optimal location and normal orientation for them. We use sparse exhaustive search, going through the combinations of modifications for distance from the reference camera, normal pitch and normal yaw (roll is fixed by the reference camera x-axis). Patch definition with regard to the reference camera stays the same, and thus, the reference image does not change.

If there are more than n projections where the photo consistency is measured below $\kappa/2$, the patch is accepted, and patch candidates originating from the same features are removed from the optimization process. We start the optimization from the candidates closest to the reference camera, as they are better defined due to the limits of image sampling. The patch model is ready when there are no more patch candidates left.

Model reconstruction

The reference approach [17] continues from this step by expanding the patches to their neighborhood to fill the holes in the model. However, for our purposes of interaction and recognition of the object, the model has already enough detail.

To avoid unnecessary complexity in the implementation, we make a point cloud representation of the patch model using their center locations. We remove clear outliers by calculating

the average distance between points in the neighborhood and then thresholding based on the standard deviation. We then build a mesh model on top of the denoised cloud using Poisson surface reconstruction discussed in Section 2.6.3. As the algorithm requires an oriented point cloud, we estimate the point normals using approach discussed in Section 2.6.2.

Finally, we add texture to the model by projecting each triangle in the mesh to the camera with the best view. We determine the best view by first taking the cameras that are in the $60°$-visibility cone determined by the surface normal, and then picking the camera whose normal has the smallest dot product with the surface normal. An example of the built models is shown in Figure 4.10.

Figure 4.10. *Point cloud built from the patch model and the final textured model.*

5 EVALUATION

In this chapter we compare our implementation to the current SoA methods. We also discuss the limitations of our algorithm and the possible improvements that could be made.

5.1 Disparity estimation

5.1.1 State of the art

During recent years, the focus of the stereo vision research has been in applying machine-learning methods to the problem. Here we shortly discuss two of these approaches.

Matching cost computation using convolutional neural networks

In the seminal work of Zbontar and LeCun [59], neural networks replace elements of the traditional approach. Matching cost computation is done using a Convolutional Neural Network (CNN) trained to calculate similarity measures for small image patches. The general steps of dense disparity calculation are 1) cost aggregation, 2) disparity calculation, and 3) refinement.

1) Cost aggregation is done following the work of Zhang et al. [60], where support region is grown in four directions, combining similar pixel values up to a distance limit. Furthermore, semi global matching is carried out in four directions as explained in Section 2.2.4. 2) Disparity calculation is performed by selecting the disparities that minimize the calculated matching cost. 3) Disparities are refined: first using the left right consistency check, second using subpixel estimation by fitting a quadratic curve, and finally by filtering the disparity map using a median and a bilateral filter.

End to end solution using neural networks

The current leading algorithms in the stereo benchmarks have gone further, and propose end-to-end solutions using neural networks. One of the recent algorithms is Pyramid Stereo Matching Network (PSMN) proposed by Chang and Chen [6].

In PSMN, CNN is first used for feature extraction. Then, spatial pyramid pooling is done, which means that features are compressed in different scales to provide a more global context. The outputs of the pyramid pooling for the stereo images are concatenated to form a 4D cost volume, which is fed to a stacked hourglass architecture to learn more context information. Finally, disparity is calculated as the sum of disparities weighted by their predicted probability. The current leading algorithm [7] of the KITTI 2015 benchmark [20] is based on a modification of PSMN.

5.1.2 Benchmark: KITTI 2015 stereo

KITTI 2015 stereo benchmark consists of a set of 1392x512 pixel stereo images captured from a car driven in urban environment. The benchmark is evaluated based on the ground truth disparities received using a LIDAR. The current leading real time (> 5 FPS) implementations in the benchmark with related publications are presented in Table 5.1. The GPU implementation of the disparity estimation via semi global matching [27] (shown in bold) can be regarded as an optimal version of our algorithm. [20]

Algorithm	Disparity outliers (%)	Running time (s)	GPU
DispNet with correlation layer [42]	4.3	0.06	yes
Multi block matching [16]	6.0	0.13	no
Learning of cost volume aggregation [36]	6.3	0.03	yes
Semi global matching [27]	**6.3**	**0.11**	**yes**
Appearance aligned block matching stereo [15]	6.6	0.08	no
Summed normalized cross correlation [14]	7.1	0.08	no
Embedded stereo via semi global matching [26]	8.2	0.0064	yes
Prediction correction optical flow [10]	8.4	0.08	yes
Efficient large scale stereo [22]	9.6	0.19	no
OpenCV block matching [3]	25.2	0.1	no
Convolutional spatial propagation [7]	1.74	0.5	yes
OpenCV semi global block matching [27]	10.84	1.1	no

Table 5.1. *Real time algorithms (> 5 FPS) in KITTI 2015 Stereo benchmark (on 25.1.2019). Outliers are disparities with > 3 px or > 5 % error, and the percentage is calculated as an average over ground truth pixels. Current leading submission and the OpenCV SGBM implementation are provided for reference below the dashed line.*

Based on the benchmark results, semi global matching is a viable option for real time disparity estimation. The algorithm is robust, and as a model based approach, it has the advantage of being mathematically explainable compared to the data driven methods. Thus, its limitations are known and they can be taken into account.

If the criteria for real time is loosened, the benchmark is currently dominated by deep

learning based methods, leading submissions achieving under two percentage of disparity outliers. It is arguable if these algorithms perform equally well for different types of data. Nevertheless, the benchmark results are promising, and the learning based methods will likely be dominant in applications where transparency of the algorithm is not crucial and training data is available.

5.1.3 Evaluation: TUTLAB dataset

We calculated disparity maps for the TUTLAB dataset using our implementation of the SGBM algorithm, and the SoA PSMN algorithm, which is openly available at [6]. Examples of the results are presented in Figure 5.1.

The most significant difference between the two algorithms is that PSMN produces a smooth disparity map that covers the entire image, whereas the SGBM disparity map contains values only in the areas where it found correspondences. Geometry of the stereo camera system constrains the searched area to exclude areas not visible in both cameras, manifested as the missing values on the left and right side of the SGBM disparity map (5.1b). PSMN, on the other hand, is not limited by the hard constraints of the real world, and extrapolates disparities even in these areas that technically do not have a disparity. Similarly, SGBM does not calculate disparities in the occluded regions that are only visible to one of the cameras, while PSMN does. In addition to the legitimate missing values, SGBM fails to calculate disparities here and there, mainly in the homogeneous areas of the image.

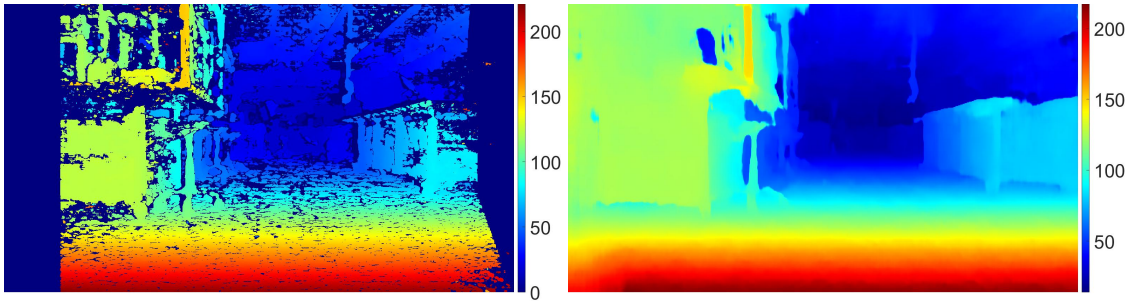
To assess the accuracy of the evaluated disparities, ground truth is needed. As there was no synchronization between the LIDAR and motion capture system, the alignment between the ground truth model and the trajectory had to be done manually. First, the trajectory was roughly scaled, rotated, and translated to the correct place as presented in Figure 3.3. For finer alignment, the raw data and the ground truth point cloud projections were visually compared to reduce the alignment error, as presented in Figure 5.2.

Evidently, the alignment is not perfect, and thus disparity ground truth is not established. Instead, we construct point clouds corresponding to the SGBM and PSMN disparity maps. These point clouds are placed inside the LIDAR point cloud using the manually aligned ground truth camera poses. As the accuracy metric, mean distance to the ground truth cloud is calculated. The results are presented in Table 5.2.

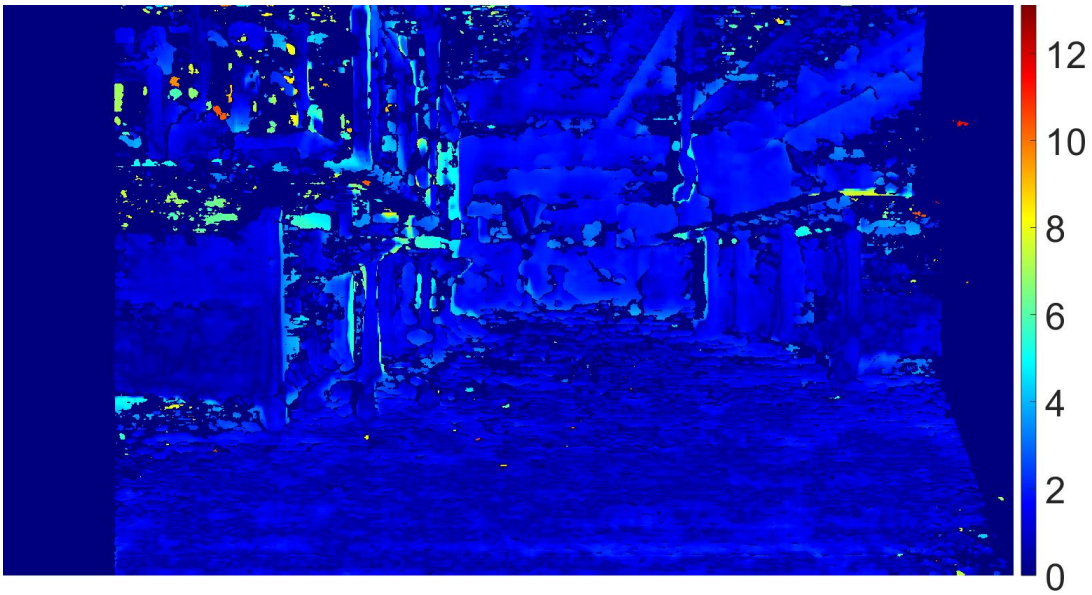
Table 5.2. *Mean and standard deviation of the distance to the ground truth model.*

Algorithm	μ (m)	σ
PSMN (full)	0.12	0.13
SGBM (full)	0.11	0.29
PSMN (< 5 m)	0.09	0.05
SGBM (< 5 m)	0.08	0.04

(a) Rectified left image



(b) $\mathbf{D}_{SGBM}$ ***(c)*** $\mathbf{D}_{PSMN}$



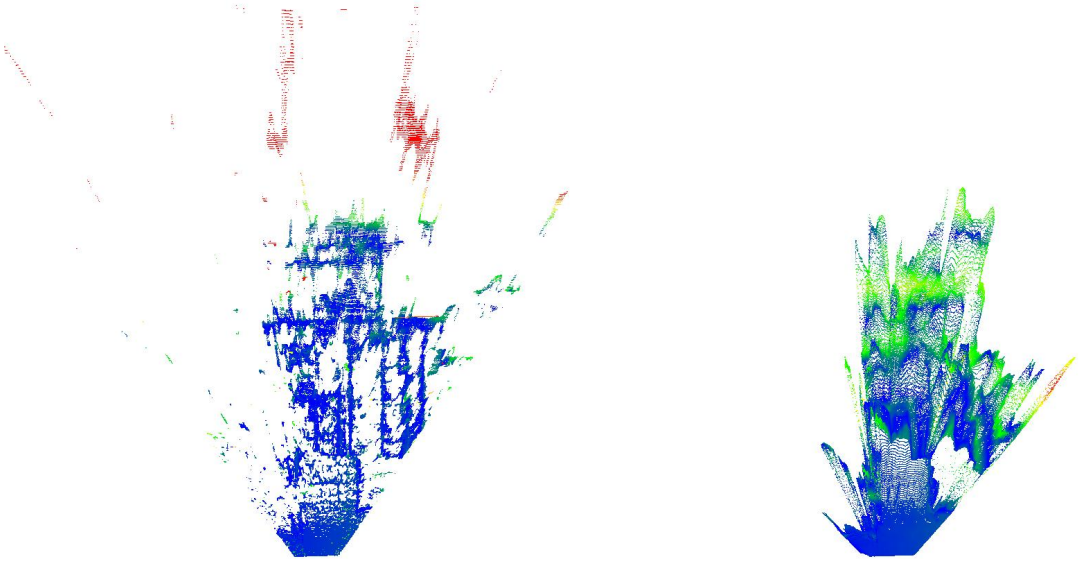
(d) $\sqrt{|\mathbf{D}_{SGBM} - \mathbf{D}_{PSMN}|}$

Figure 5.1. *Disparity maps calculated from the TUTLAB data: (a) original rectified image; (b) our implementation of SGBM; (c) PSMN; and, (d) squared disparity difference, disregarding areas where SGBM did not produce a result.*

When a lower limit to the disparity is not set, PSMN produces rather large errors in the farther regions, where the smooth disparity map results in erroneous smooth surfaces. The full point cloud reconstruction based on SGBM has lower mean distance to the ground truth, as disparity is only estimated in the areas where the algorithm finds correspondences. Different from the PSMN cloud, SGBM produces clusters of clear outliers here and there, whereas the erroneous points of the PSMN are systematically wrong. At a short distance, the distance to the ground truth is similar for both algorithms. An example of the distribution

Figure 5.2. *The same view from the raw image data and the ground truth camera pose manually aligned to the LIDAR point cloud.*

of the errors is shown in Figure 5.3.



(a) SGBM ***(b)*** PSMN

Figure 5.3. *Distribution of the error in the reconstructed point cloud model: (a) SGBM; and, (b) PSMN. Error is presented by a color scale from blue to red.*

5.1.4 Discussion

SGBM produces a cloud whose points are more exact, especially if an appropriate lower limit to the disparity is set. However, there are points missing in the areas where the disparity estimation failed, and some clearly erroneous points. PSMN produces a smooth, continuous point cloud without clear outliers. Nevertheless, there are larger errors in the points further away from the camera.

Point cloud produced from the PSMN result is more visually pleasing, and would be better suited for, *e.g.*, Virtual Reality (VR) applications where the deviations from the ground truth in further areas are not problematic. In our case of building an exact model of the environment, the systematic errors that PSMN produces would cause issues, as they

would be harder to remove via denoising than the clearly erroneous points of SGBM.

SGBM can be implemented to run in real time on specific hardware (GPU/FPGA) [26], and on Central Processing Unit (CPU) with modifications to the algorithm [16]. There are several implementations openly available: from the basic OpenCV implementation, to the hardware specific real time implementations. SGBM can be considered the standard method of the traditional disparity estimation algorithms, having a vast support in the literature. It provides an explainable workflow, where the desired properties such as the accuracy and the efficiency of the algorithm can be tuned to suit the application. For instance, the matching cost can be changed to a more robust Census or MI if there are large differences between the sensors, and the global context can be emphasized by increasing the number of paths in the smoothness constraint.

PSMN and other CNN algorithms need specific hardware (GPU) to run. Most of the current algorithms are optimized for accuracy rather than speed. However, real time operation has been achieved [36], [42]. There are several CNN implementations openly available for disparity estimation. Nevertheless, they are not on the standard library implementation level of SGBM. As an example, in the course of this work, considerable effort was put into finding an implementation that could be run for our data.

Generalization of the end-to-end neural network disparity estimation algorithms remains debatable. The PSMN model that we used in our evaluation was trained using the KITTI 2015 stereo dataset, which consists of outdoor driving scenes with a completely different camera setup from ours. For our TUTLAB dataset, the results were comparatively worse, as SGBM was able to beat PSMN in accuracy. Still, the results were promising for such a different dataset, indicating that CNN based solutions have potential to be generally superior to traditional methods if appropriate data is available for training.

The lack of appropriate data is the key issue in the application of neural networks in disparity estimation. Generating ground truth disparities currently requires a costly dynamic LIDAR, and the sensor still has its issues with resolution and reflections, which have to be taken into account. Data can be created via simulation. However, it is arguably impossible to build an accurate simulation of reality, and the algorithm is likely to learn traits specific to the simulation. Currently there are few datasets openly available, and they are for specific use cases.

Finally, the data driven algorithms lack explainability compared to SGBM. The architectures are built based on feature extraction via convolution, with different heuristics added on top, to produce as good result as possible. What exactly happens in each part of the network is difficult to quantify, and thus harder to optimize.

5.2 Camera trajectory estimation

5.2.1 State of the art

Camera trajectory estimation from video data is generally referred to as visual odometry. At present, the most accurate visual odometry algorithms estimate the structure of the environment at the same time to ensure consistency.

Visual SLAM using ORB features

The current SoA visual Simultaneous Localization and Mapping (SLAM) approach is called ORB SLAM2 [44].

ORB SLAM2 is based on ORB features that are discussed in Section 2.3.6. The algorithm tracks features locally, and uses bundle adjustment to minimize reprojection error of the tracked points. Bundle adjustment is also done for the keyframes that are chosen from the image sequence to acquire a consistent trajectory. Finally, when a loop is detected, pose graph optimization is used to close the loop, and the loop sequence is bundle adjusted to correct the trajectory.

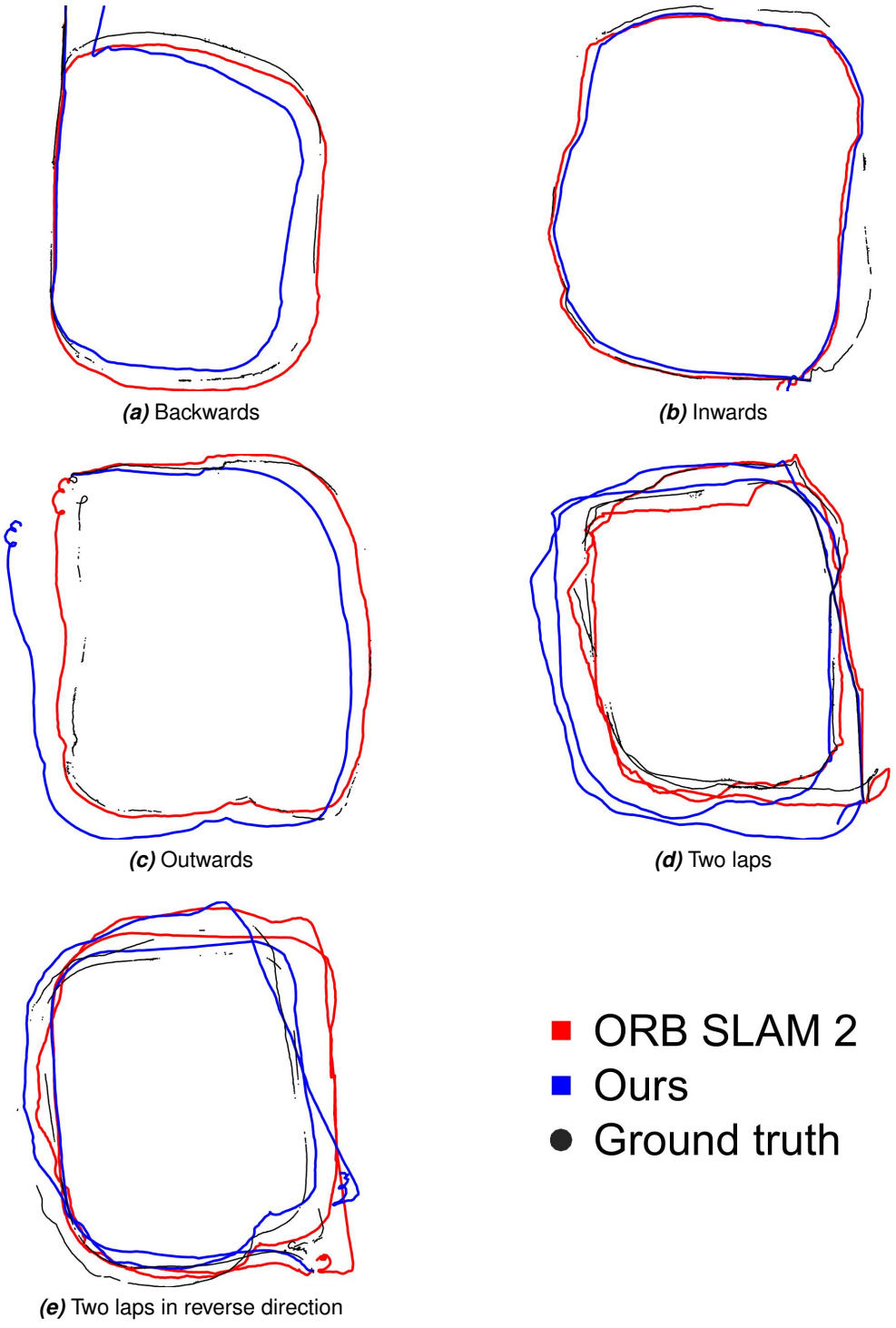
5.2.2 Evaluation: TUTLAB dataset

We calculated camera trajectories for the TUTLAB dataset using our feature based algorithm, and the SoA ORB SLAM 2, which is openly available under the GNU General Public License at [44]. The estimated trajectories are presented in Figure 5.4.

As explained in Chapter 3, our ground truth via motion capture was not synchronized with the other sensor systems. Thus, the presented ground truth trajectories were manually 1) scaled to the corresponding size and 2) rotated around the gravitational axis to have initial camera angle corresponding to the estimated trajectories. Moreover, the ground truth trajectory has values missing due to the limitations of the motion capture system. Nevertheless, it accurately represents the shape of the trajectory for the acquired parts.

Visual comparison to the manually aligned ground truth shows that the shape of the estimated trajectories is close to the real trajectory for both algorithms. The most evident difference between the two algorithms is the lack of loop closure in our version, which results in the prominent drift in the trajectories. Given the similarity between the shapes of the estimated trajectories, our algorithm combined with a loop closure implementation would arguably produce trajectories comparable to the SoA.

Even with its flaws, the ground truth data can be used to accurately model the ground plane, as it is invariant to the manual scaling and rotation around the gravitational axis.



(a) Backwards

(b) Inwards

(c) Outwards

(d) Two laps

(e) Two laps in reverse direction

Figure 5.4. *Calculated trajectories for the TUTLAB dataset (see Section 3.2 for detail on the sequences). Trajectories produced by our algorithm are drawn in blue; ORB SLAM 2 trajectories are drawn in red; and the ground truth trajectories are presented by the black dots.*

Thus, we can reliably quantify the error in the estimated trajectory height relative to the ground plane. First, we fit the ground plane model $ax + by + cz + d = 0$ to the ground truth

data by setting $b = -1$ and stacking the N data points to form equation

$$\mathbf{y} = \mathbf{X}\beta \iff \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} x_1 & z_1 & 1 \\ \vdots & \vdots & \vdots \\ x_N & z_N & 1 \end{bmatrix} \begin{bmatrix} a \\ c \\ d \end{bmatrix}, \tag{5.1}$$

from which the least squares plane model estimate can be calculated as $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. We then calculate the distance to the plane model for each estimated camera pose, and assess the error per one step of the algorithms as the change in the distance of the consecutive poses. The results are presented in Table 5.3

Table 5.3. *Mean and standard deviation of the error in the trajectory height estimation per one step ($\Delta t = 0.8$ s) of the algorithms for the five TUTLAB sequences.*

	ORB SLAM 2		Ours	
Seq.	μ (mm)	σ	μ (mm)	σ
(a)	**0.13**	0.89	0.31	1.84
(b)	**0.00**	1.46	–1.82	2.43
(c)	**0.14**	1.79	–0.18	2.25
(d)	**–0.01**	1.54	0.54	1.02
(e)	**–0.01**	1.50	–1.57	1.68

The mean trajectory height error per one step for ORB SLAM 2 is close to zero as a result of the loop closure, whereas there is upwards or downwards drift in our algorithm. For longer sequences, the error deviations for both algorithms confirm that complementary methods are indeed needed for drift avoidance: be it loop closure, planarity assumption, or something else. Normalized error histograms overlaid with the estimated normal probability density functions are presented in Appendix A.2.

5.2.3 Discussion

It should be noted that our dataset is mainly limited to planar camera motion apart from the trembling of the platform. Especially outdoors applications would likely contain all six degrees of freedom, as uneven surfaces would result in camera pitch and roll changes in addition to the variance in height. Given the errors in the TUTLAB evaluation, we presume that neither of the algorithms would work in a completely unconstrained setting as is. Nevertheless, usually assumptions about, *e.g.*, the planarity of the movement can be made.

The lack of loop closure is the evident defect of our system. Drift can only be reduced to a point, as the error accumulates over time, causing the trajectory to diverge. Loop closure boils down to forming a compressed model of the traversed environment, and efficient detection of an already visited location. For instance, ORB SLAM2 utilizes dynamic

keyframe memorization and comparison between the feature presentations to detect the loops. In addition to loop closure, GPS provides a good reference to correct the trajectory outdoors, while indoors the planarity assumption is convenient.

5.3 Environment model

5.3.1 State of the art

Point cloud segmentation methods can be roughly classified in model fitting, region growing, and feature clustering based subgroups, of which our focus is in the model fitting based segmentation.

Our algorithm is based on spline or plane model fitting accompanied with the object segmentation based on the model. Similar approaches can be found in the literature: 1) For drivability estimation, building a surface model by fitting splines to the point cloud has been proposed by Broggi et al.[4]; 2) Object segmentation based on surface continuity on the other hand has been proposed by Douillard et al. [11].

Octree based region growing

One of the current SoA approaches to the environment segmentation is octree based region growing proposed by Vo et al. [56]. Octree is a hierarchical structure which recursively divides the space into smaller subspaces until satisfactory detail is achieved. The use of octrees is especially beneficial for real time algorithms, as it makes the data handling more efficient.

The octree algorithm starts by dividing space into voxels (3D pixels) until they are representative of the underlying cloud. For each voxel, normal vector and residual are calculated as the voxel features. The normal vector is estimated using PCA, and the residual is calculated as the Root Mean Square Deviation (RMSD) of the cloud from the plane defined by the center of mass of the points and the estimated normal. The algorithm continues by grouping voxels with similar features to form smooth surface segments. Finally, region growing is used to extend the boundary voxels.

5.3.2 Evaluation: TUTLAB dataset and outdoors tests

The quality of the environment model is largely dependent on the accuracy of the disparity estimation and the camera pose estimation, as the merged global cloud is the input data for our algorithm. Especially drift and other inconsistencies in the trajectory estimation are problematic, as the whole model becomes deformed when the error accumulates.

Plane model

In an indoors setting, we cut corners by fitting a plane to the floor, and estimated obstacles with regard to deviations from the floor. In our tests, we found the algorithm to work well, given that 1) there was enough texture in the floor to correctly fit the plane, and 2) when the operation time was moderate so that the drift did not cause the floor to twist. The accuracy of the input point cloud from our pipeline allowed the spline model to differentiate objects of the size of a basketball as obstacles in the drivability model.

Object segmentation based on the continuity in the drivability model correctly segmented the clearly separate objects of the environment. However, as the segmentation was done in 2D, some volumetrically distinct objects were clumped together.

Spline model

As the more general solution, spline model is able to estimate drivability in more complex outdoors scenes with uneven topographies. The distinction between the drivable and obstacle areas was done based on the maximum model height difference in the sliding window area, which was adjusted based on the capability of the vehicle. Some detail was lost in the object boundaries compared to the plane model, and there was more tendency for adjacent objects to be clumped together in the object segmentation step when using the spline drivability model.

We tested the spline algorithm in outdoors and indoors settings, where it was able to distinguish the drivable areas in the environment. An example of the outdoor results is shown in Figure 5.5.

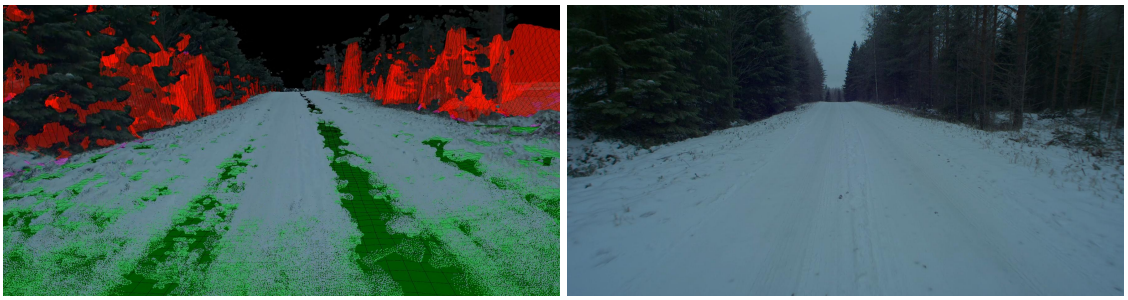


Figure 5.5. *Example of the spline model. Drivable surface is drawn in green, obstacles in red, and areas too close to the obstacles in purple. The point cloud where the spline model is fitted is also shown for reference.*

5.3.3 Discussion

Our modeling algorithm was able to correctly estimate drivability for the moderately noisy point clouds produced by our pipeline. Given that the direction of gravity is known or can be

estimated, the local height differences of the model accurately differentiate the obstacles from the drivable surfaces. The model forms a compact presentation of drivability: it is essentially a 2D matrix containing the heights in the environment. The model can be locally updated based on the input point clouds to allow real time dynamic drivability estimation.

For object segmentation, the connectivity in the drivability map roughly divides the obstacle areas into objects and clumps of adjacent objects. A true volumetric representation such as the octrees has a clear edge in this regard, as the spline model only has one value for height in each map location.

5.4 Multi-view modeling

5.4.1 State of the art

Photogrammetry is a heavily researched topic, and hence there is a plethora of commercial and open solutions available for multi-view modeling. The packages generally combine multiple algorithms together to provide a seamless workflow for the end user. Of the current open software, we mention OpenMVS, COLMAP [50], and AliceVision [31]. General steps in the photogrammetry pipeline are 1) feature and image matching to find confluences, 2) SfM to estimate camera parameters, 3) dense depth estimation, and finally 4) meshing and texturing the model. Our focus was in the dense depth estimation, as we already had the estimated camera parameters. We picked our algorithm to implement based on the results in the Middlebury multi-view stereo benchmark.

5.4.2 Benchmark: Middlebury multi-view stereo

Middlebury multi-view stereo benchmark [52] consists of two sets of 640x480 images captured around an object. The benchmark is evaluated based on a ground truth 3D model of the object acquired using a LIDAR. There are three categories based on the number of used views: "Full" with over 300 views, "Ring" with around 50 views and "Sparse" with 16 views.

Adding views to the multi-view problem provides diminishing returns in the quality of the model, as the information in the new views is mostly already provided in the sparser coverage. Meanwhile, the computation time increases, depending on the complexity of the algorithm, at least linearly. Thus, we focus on the sparse multi-view modeling case, as our aim is to increase the overall quality of the model without losing too much in performance.

In Table 5.4 are presented the ten most accurate algorithms for sparse multi-view modeling in the Middlebury benchmark. Results for the multi-view algorithm [17] that we used as the basis of our implementation are shown in bold.

Algorithm	Sparse completeness (%)	Total completeness (%)	Sparse running time (min)
Y. Furukawa et al. (v3)[17]	**99.25**	**99.55**	**180**
Y. Furukawa et al. (v2) [17]	**99.2**	**99.38**	**290**
N. Savinov et al. [49]	98.75	99.4	2600
Z. Li et al. [38]	98.65	98.88	100
S. Li et al. [37]	98.05	99.12	0.83
W. Huang et al. [30]	97.85	98.7	5.4
Y. Liu et al. [40]	97.8	97.8	20
S. Galliani et al. [19]	97.8	98.87	5.0
I. Kostrikov et al. [35]	97.55	98.45	91
A. Zaharescu et al. [58]	97.5	98.2	40

Table 5.4. *Ten most accurate algorithms for sparse multi-view modeling in the Middlebury multi-view stereo benchmark (on 31.1.2019). Completeness measures the fraction of the ground truth closer to the reconstruction than the threshold (1.25 mm). "Sparse" completeness is the mean of "Dino Sparse" and "Temple Sparse" results, and total completeness is the mean of the all three "Sparse", "Ring", and "Dense" categories. "Sparse" running time is the mean of the normalized "Sparse" running times, where normalization is done with regard to the test processor clock speed.*

According to the Middlebury benchmark, the algorithm we chose to implement is the most accurate for sparse multi-view modeling. It has also been proved to work in a more general scenario for outdoor scenes [53]. The running time for the algorithm is fairly high. However, we only implemented the ideas of the initial sparse model building and left out the tedious refining process, which is the most time consuming part of the algorithm.

It should be noted that the Middlebury benchmark provides a very limited test setting having only two scenes. The data is optimal in the sense that the scenes are seen from all angles, and accurate camera parameters are provided. The visual hull in the images is exploitable, as the background is easily extracted. Moreover, the images in the dataset are in low resolution. EPFL benchmark [53] contains more robust scenes for multi-view stereo evaluation. Nevertheless, an online platform for evaluation of the algorithms is not included. Of the more recent work, ETH3D benchmark [51] and "Tanks and Temples" [34] benchmark provide potent alternatives to the *de facto* Middlebury benchmark.

5.4.3 Evaluation: TUTLAB dataset

As described in Section 4.3, we implemented our multi-view algorithm from scratch based on the well-established approach of Furukawa et al. [17]. Our algorithm produced results similar to the ones presented in the original paper for the Middlebury dataset, as seen in Figure 4.10). However, we did not manage to achieve same kind of performance for our

TUTLAB test data.

The purpose of the multi-view algorithm in our pipeline was to improve the accuracy of the rough models received from the stereo algorithm. We intended to combine temporal information by picking keyframes that contain the object of interest, and run the multi-view algorithm. Nevertheless, compared to the "Sparse" multi-view case of the Middlebury benchmark, our data had considerably less angles covered as the robot was merely going past the objects. Moreover, our camera parameters had the error of the trajectory estimation via SfM, and the scenes were in general far from the optimal setting of the Middlebury data. By manually picking the keyframes used for the multi-view modeling, we were able to obtain results corresponding to basic SfM approaches. An example of the results is shown in Figure 5.6. There are erroneous points near depth discontinuities, where the background is homogeneous, and the cloud is not dense. The achieved level of accuracy is clearly insufficient for proper model building.

Figure 5.6. *Point cloud model produced by our multi-view algorithm and the original object in an image.*

For comparison, we run the open AliceVision software for the same data containing the chair. Similar error with the homogeneous background can be seen, and the model in general is deformed.

Figure 5.7. *Textured mesh model produced by open source AliceVision software.*

5.4.4 Discussion

Multi-view modeling has the clear potential to improve the model quality, especially if the object is seen from multiple angles. Our aim was to improve the object model to the accuracy that would enable interaction with the object. Our implementation was able to achieve this level of performance for the optimal Middlebury benchmark data. However, for our test data the results were not satisfactory.

The difficulties encountered when testing with the TUTLAB dataset are likely related to insufficient view coverage of the object. The robot passing by an object does not produce enough information for the multi-view algorithm to build an accurate model. Specific camera trajectory, *e.g.*, around the object is needed, and presumably, a refinement step to produce an object model for interaction purposes.

On another note, current multi-view algorithms require a lot of computation and are thus slow to run. Conclusions cannot be made based on our proof of concept MATLAB implementation. However, as can be seen in the Middlebury benchmark (Table 5.4), the runtimes are generally in the scale of minutes or hours even for sparse multi-view modeling.

6 CONCLUSION

In the course of this work, we studied and implemented the full pipeline of environment modeling from the viewpoint of autonomous machines. The main steps in the pipeline are disparity estimation, camera trajectory estimation, constructing an environment model for movement purposes, and building an accurate object model for interaction.

We implemented disparity estimation using traditional SGBM method. We evaluated our implementation against a SoA CNN based method using our test data, for which both provided equally accurate results. The largest advantage of the traditional method is its explainability and traceability; whereas the CNN based methods have the potential to produce more accurate results given the right training data.

For camera trajectory estimation, we implemented a feature-tracking algorithm combining elements found in the literature. The drift of our algorithm is at the level of current SoA for our test data. However, longer tracking periods would require loop closure to correct the accumulating error when revisiting previously seen areas, or accurate secondary information, *e.g.*, in the form of GPS.

We built our environment model, with the autonomous machine in mind, as a spline model describing the topology of the environment. We tested the algorithm in indoors and outdoors for the noisy input point clouds produced by our pipeline. Our model was able to accurately differentiate drivable areas and obstacles in the environment. Moreover, it provides a compact and comprehensible presentation of the environment for drivability estimation. As an alternative method to be used indoors, plane based model offered slight improvement in accuracy in the object boundaries. Finally, we found that rough object segmentation is feasible based on the connectivity in the topology map.

To build more accurate models for object handling, we implemented a multi-view algorithm based on a SoA approach. Our implementation worked for optimal benchmark data. However, for our test scenario, the camera movement with regard to the modeled object did not provide adequate views to build an accurate model. If the camera can be guided to provide proper coverage of the object, current SoA algorithms are able to build an accurate model if computation time is not an issue.

To achieve a functional visual system for machines using our pipeline, following improvements should be made. 1) Efficiency: SGBM could be implemented in hardware, and only necessary information (built model, information needed for loop closure) should be kept in memory. 2) Loop closure: we pondered using the built environment model to detect

previously visited areas. Current SoA detects loops on image level. 3) Accurate object models for interaction: this could be achieved with multi-view modeling if proper views can be captured.

To conclude, stereo camera system shows promising results for environment modeling. With the current SoA algorithms, it can be used as the sole sensor for autonomous movement, and, given appropriate conditions, for interaction with the environment.

REFERENCES

[1] A. Alahi, R. Ortiz and P. Vandergheynst. FREAK: Fast Retina Keypoint. *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, 510–517.

[2] S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, 1073–1080.

[3] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[4] A. Broggi, E. Cardarelli, S. Cattani and M. Sabbatelli. Terrain mapping for off-road Autonomous Ground Vehicles using rational B-Spline surfaces and stereo vision. *2013 IEEE Intelligent Vehicles Symposium (IV)*. 2013, 648–653.

[5] M. Calonder, V. Lepetit, C. Strecha and P. Fua. BRIEF: Binary Robust Independent Elementary Features. *Computer Vision – ECCV 2010*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 778–792.

[6] J. Chang and Y. Chen. Pyramid Stereo Matching Network. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. `https://github.com/JiaRenChang/PSMNet` (Accessed: 2019-05-13). 2018, 5410–5418.

[7] X. Cheng, P. Wang and R. Yang. Learning Depth with Convolutional Spatial Propagation Network. *CoRR* (2018). arXiv: `1810.02695`.

[8] S. Coren and J. S. Girgus. *Seeing is deceiving: The psychology of visual illusions.* Lawrence Erlbaum, 1978.

[9] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012, 12–23.

[10] M. Derome, A. Plyer, M. Sanfourche and G. Le Besnerais. A Prediction-Correction Approach for Real-Time Optical Flow Computation Using Stereo. *Pattern Recognition*. Springer International Publishing, 2016, 365–376.

[11] B. Douillard, J. Underwood, N. Melkumyan, S. Singh, S. Vasudevan, C. Brunner and A. Quadros. Hybrid elevation maps: 3D surface models for segmentation. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, 1532–1538.

[12] H. Edelsbrunner, D. Kirkpatrick and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory* 29.4 (1983), 551–559.

[13] H. Edelsbrunner and E. P. Mücke. Three-dimensional Alpha Shapes. *ACM Trans. Graph.* 13.1 (1994), 43–72.

[14] N. Einecke and J. Eggert. A Two-Stage Correlation Method for Stereoscopic Depth Estimation. *2010 International Conference on Digital Image Computing: Techniques and Applications*. 2010, 227–234.

[15] N. Einecke and J. Eggert. Stereo image warping for improved depth estimation of road surfaces. *2013 IEEE Intelligent Vehicles Symposium (IV)*. 2013, 189–194.

[16] N. Einecke and J. Eggert. A multi-block-matching approach for stereo. *2015 IEEE Intelligent Vehicles Symposium (IV)*. 2015, 585–592.

[17] Y. Furukawa and J. Ponce. Accurate, Dense, and Robust Multiview Stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.8 (2010), 1362–1376.

[18] A. Fusiello, E. Trucco and A. Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications* 12.1 (2000), 16–22.

[19] S. Galliani, K. Lasinger and K. Schindler. Massively Parallel Multiview Stereopsis by Surface Normal Diffusion. *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, 873–881.

[20] A. Geiger, P. Lenz and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, 3354–3361.

[21] A. Geiger, J. Ziegler and C. Stiller. StereoScan: Dense 3d reconstruction in real-time. *2011 IEEE Intelligent Vehicles Symposium (IV)*. 2011, 963–968.

[22] A. Geiger, M. Roser and R. Urtasun. Efficient Large-Scale Stereo Matching. *Computer Vision – ACCV 2010*. Springer Berlin Heidelberg, 2011, 25–38.

[23] M. J. Hannah. Computer matching of areas in stereo images. PhD thesis. Stanford University, 1974.

[24] C. Harris and M. Stephens. A combined corner and edge detector. *In Proc. of Fourth Alvey Vision Conference*. Citeseer. 1988, 147–151.

[25] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[26] D. Hernandez-Juarez, A. Chacón, A. Espinosa, D. Vázquez, J. Moure and A. López. Embedded Real-time Stereo Estimation via Semi-global Matching on the GPU. *Procedia Computer Science* 80 (2016). International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, 143–153.

[27] H. Hirschmuller. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008), 328–341.

[28] H. Hirschmuller and D. Scharstein. Evaluation of Stereo Matching Costs on Images with Radiometric Differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.9 (2009), 1582–1599.

[29] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle. Surface Reconstruction from Unorganized Points. *SIGGRAPH Comput. Graph.* 26.2 (1992), 71–78.

[30] W. Huang, K. Sun and W. Tao. Multi-view Stereo Combined with Space Propagation and Pixel-Level Refinement. *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*. 2017, 208–213.

[31] M. Jancosek and T. Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. *CVPR 2011*. 2011, 3121–3128.

[32] M. Kazhdan, M. Bolitho and H. Hoppe. Poisson Surface Reconstruction. *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP '06. Cagliari, Sardinia, Italy: Eurographics Association, 2006, 61–70.

[33] M. Kazhdan and H. Hoppe. Screened Poisson Surface Reconstruction. *ACM Trans. Graph.* 32.3 (2013), 29:1–29:13.

[34] A. Knapitsch, J. Park, Q.-Y. Zhou and V. Koltun. Tanks and Temples: Benchmarking Large-scale Scene Reconstruction. *ACM Trans. Graph.* 36.4 (2017), 78:1–78:13.

[35] I. Kostrikov, E. Horbert and B. Leibe. Probabilistic Labeling Cost for High-Accuracy Multi-view Reconstruction. *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, 1534–1541.

[36] A. Kuzmin, D. Mikushin and V. Lempitsky. End-to-End learning of cost-volume aggregation for real-time dense stereo. *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*. 2017, 1–6.

[37] S. Li, S. Y. Siu, T. Fang and L. Quan. Efficient Multi-view Surface Refinement with Adaptive Resolution Control. *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, 349–364.

[38] Z. Li, K. Wang, W. Zuo, D. Meng and L. Zhang. Detail-Preserving and Content-Aware Variational Multi-View Stereo Reconstruction. *IEEE Transactions on Image Processing* 25.2 (2016), 864–877.

[39] T. Lillesand, R. Kiefer and J. Chipman. Microwave and LIDAR Sensing. *Remote sensing and image interpretation*. John Wiley & Sons, 2014, 471–484.

[40] Y. Liu, X. Cao, Q. Dai and W. Xu. Continuous depth estimation for multi-view stereo. *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, 2121–2128.

[41] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences* 207.1167 (1980), 215–217.

[42] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy and T. Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, 4040–4048.

[43] T. Moons, L. V. Gool and M. Vergauwen. 3D Reconstruction from Multiple Images Part 1: Principles. *Foundations and Trends® in Computer Graphics and Vision* 4.4 (2010), 287–404.

[44] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics* 33.5 (2017). `https://github.com/raulmur/ORB_SLAM2`(Accessed: 2019-05-13), 1255–1262.

[45] C. Ricolfe-Viala and A.-J. Sanchez-Salmeron. Camera calibration under optimal conditions. *Opt. Express* 19.11 (2011), 10769–10775.

[46] P. L. Rosin. Measuring Corner Properties. *Computer Vision and Image Understanding* 73.2 (1999), 291–307.

[47] E. Rosten and T. Drummond. Machine Learning for High-Speed Corner Detection. *Computer Vision – ECCV 2006*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, 430–443.

[48] E. Rublee, V. Rabaud, K. Konolige and G. Bradski. ORB: An efficient alternative to SIFT or SURF. *2011 International Conference on Computer Vision*. 2011, 2564–2571.

[49] N. Savinov, C. Häne, L. Ladický and M. Pollefeys. Semantic 3D Reconstruction with Continuous Regularization and Ray Potentials Using a Visibility Consistency Constraint. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, 5460–5469.

[50] J. L. Schönberger, E. Zheng, J.-M. Frahm and M. Pollefeys. Pixelwise View Selection for Unstructured Multi-View Stereo. *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, 501–518.

[51] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys and A. Geiger. A Multi-view Stereo Benchmark with High-Resolution Images and Multi-camera Videos. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, 2538–2547.

[52] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein and R. Szeliski. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 1. 2006, 519–528.

[53] C. Strecha, W. von Hansen, L. Van Gool, P. Fua and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, 1–8.

[54] R. Szeliski. Feature Detection and Matching. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010, 205–255.

[55] C. Tomasi and T. Kanade. Detection and Tracking of Point Features. *International Journal of Computer Vision* 9 (Jan. 1991), 137–154.

[56] A.-V. Vo, L. Truong-Hong, D. F. Laefer and M. Bertolotto. Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing* 104 (2015), 88–100.

[57] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. *Computer Vision — ECCV '94*. Springer Berlin Heidelberg, 1994, 151–158.

[58] A. Zaharescu, E. Boyer and R. Horaud. Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multiview Reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.4 (2011), 823–837.

[59] J. Žbontar and Y. LeCun. Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches. *Journal of Machine Learning Research* 17.65 (2016), 1–32.

[60] K. Zhang, J. Lu and G. Lafruit. Cross-Based Local Stereo Matching Using Orthogonal Integral Images. *IEEE Transactions on Circuits and Systems for Video Technology* 19.7 (2009), 1073–1079.

[61] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), 1330–1334.

[62] Zhengyou Zhang. On the epipolar geometry between two images with lens distortion. *Proceedings of 13th International Conference on Pattern Recognition*. Vol. 1. 1996, 407–411 vol.1.

A APPENDIX

A.1 Skew symmetric matrix

Skew symmetric matrix corresponding to a vector $a = (a_1, a_2, a_3)^T$ is defined as

$$[a]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

.

A.2 Trajectory height estimation error