Camera Pose Estimation Using 2D-3D Line Pairs Acquired and Matched with a Robust Line Detector and Descriptor

PhD Thesis

Hichem Abdellali

Supervisor:
Prof. Zoltan Kato

Doctoral School of Computer Science
Institute of Informatics
University of Szeged

Szeged
2021

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

2D	Two-dimensional
3D	Three-dimensional
AI	Artificial Intelligence
GT	Ground Truth
LM	Levenberg–Marquardt
ML	Machine Learning
DoF	Degrees of Freedom
FOV	Field of View
CNN	Convolutional Neural Networks
ORB	Oriented FAST and Rotated BRIEF
L2D2	Learnable Line Detector and Descriptor
MVVD	Multi-view Vertical Direction
MSAC	M-estimator SAmple and Consensus
SLAM	Simultaneous Localization and Mapping
SIFT	Scale-Invariant Feature Transform
LIDAR	Light Detection and Ranging
VGG19	Convolutional Neural Network that is 19 Layers Deep
RANSAC	RANdom SAmple Consensus

Acknowledgments

First and foremost, I would like to praise Allah for his blessing during my Ph.D. and in completing this thesis. I would like to address the most of my acknowledgments to my advisor, Prof. Zoltán Kató, for the infinite support of my Ph.D. study as well in my personal life. He inspired me to become an independent researcher and helped me realize the power of critical reasoning. He also demonstrated what a brilliant and hard-working scientist could accomplish. My special thanks of gratitude are expressed to him. I would like to thank my lovely wife, Zahra, for all the unwavering support over the years, for all the consistent advice, she has stood next to me through all my obstacles, my travails, my absences, my fits of pique and impatience. She gave me support and help, discussed ideas, and prevented several wrong turns, and continually provided the necessary breaks from research. For my Father, and Mother's soul who deserve eternal thanks, gratitude and pray, for their infinite love, support, and encouragement. I am also thankful to my colleagues at the Institute of Informatics for their pieces of advice and help in times of need. I am very lucky and can never forget the cheer of my sisters, brothers, uncles, and their wives who have been great over the years.

Without such a big team behind me, I doubt that I would be in this place today. Indeed, I would like to dedicate this thesis to all who believed in me as a way of expressing my gratitude and appreciation. This research was supported by the Stipendium Hungaricum scholarship program. I am grateful for the opportunity they provided, which has led to the submission of this thesis.

This thesis work was partially supported by the NKFI-6 fund through project K120366 and project K135728; "Integrated program for training new generation of scientists in the fields of computer science", EFOP-3.6.3-VEKOP-16-2017-0002; the Research & Development Operational Programme for the project "Modernization and Improvement of Technical Infrastructure for Research and Development of J. Selye University in the Fields of Nanotechnology and Intelligent Space", ITMS 26210120042, co-funded by the European Regional Development Fund. Research & Innovation Operational Programme for the Project: "Support of research and development activities of J. Selye University in the field of Digital Slovakia and creative industry", ITMS code: NFP313010T504, co-funded by the European Regional Development Fund.

Hichem Abdellali, 2021.

Abstract

Camera pose estimation refers to estimating the camera pose, which is composed of the rotation $\mathbf{R}$ and translation $\mathbf{t}$ parameters with respect to the world coordinate system. Estimating the projective mapping and thereby extracting the camera parameters is the goal of camera pose estimation. However, the pose estimation process requires input parameters, like points, planes, or lines. In this thesis, we work with 2D-3D line pairs; therefore, we focused on finding a solution for 2D line detection and matching through fully automatic algorithms and CNN.

This thesis proposes novel solutions for pose estimation using 2D-3D line pairs and a novel line segment detector and descriptor based on convolutional neural networks. The pose solvers can estimate the absolute and relative pose of a camera system of a general central projection camera such as perspective or omnidirectional cameras. They work both for the minimal case and the general case using 2D-3D line pairs in presence of noise, or outliers. The algorithms have been validated on a large synthetic dataset as well as on real data. Experimental results confirm the stable and real-time performance under realistic conditions. Comparative tests show that our method compares favorably to the latest State-of-the-Art algorithms. Regarding the learnable line segment detector and descriptor, it allows efficient extraction and matching of 2D lines on perspective images. While many hand-crafted and deep features have been proposed for key points, only a few methods exist for line segments. However, line segments are commonly found in structured environments, in particular urban scenes. Moreover, lines are more stable than points and robust to partial occlusions. Our method relies on a 2-stage deep convolutional neural network architecture: In stage 1, candidate 2D line segments are detected, and in stage 2, a descriptor is generated for the extracted lines. The network is trained in a self-supervised way using an automatically collected dataset. Experimental results confirm the State-of-the-Art performance of the proposed L2D2 network on two well-known datasets for autonomous driving both in terms of detected line matches as well as when used for line-based camera pose estimation and tracking.

Abstract In Hungarian

A kamera-pozíció becslése a kamera pózának becslésére vonatkozik, amely az $\mathbf{R}$ forgatási és a $\mathbf{t}$ transzlációs paraméterekből áll a világkoordináta-rendszerhez képest. A kamerapóz becslés célja a projektív leképezés becslése és ezáltal a kamera paramétereinek kinyerése. A pózbecslési folyamat azonban bemeneti adatokat igényel, melyek lehetnek például pontok, síkok vagy vonalak. Ebben a dolgozatban 2D-3D egyenespárokkal dolgozunk, ezeket használó megoldásokat javasoltunk, ezért megoldást kerestünk a 2D egyenes-detekcióra és -illesztésre is egy teljesen automatikus algoritmus és egy CNN segítségével.

A disszertáció új megoldásokat javasol a pózbecsléshez 2D-3D egyenespárok és egy új, konvolúciós neurális hálón alapuló egyenes-szakasz detektor és leíró segítségével. A pózbecslők meg tudják becsülni egy általános középpontos kamerarendszer abszolút és relatív pozícióját, ilyen kamerarendszer állhat például perspektivikus vagy omnidirekcionális kamerákból is. A megoldók 2D-3D vonalpárok használatával mind a minimális esetekre, mind az általános esetekre működnek, zaj vagy kiugró értékek jelenlétében is. Az algoritmusokat nagyméretű szintetikus adatkészleten és valós adatokon is validáltuk. A kísérleti eredmények alátámasztják a stabil és valós idejű teljesítményt akár valós körülmények között is. Az összehasonlító tesztek azt mutatják, hogy módszerünk jól teljesít a legmodernebb algoritmusokkal szemben. Ami a tanulható egyenes-szakasz detektort és leírót illeti, lehetővé teszi a 2D szakaszok hatékony kinyerését és illesztését perspektivikus képeken. Míg számos kézzel készített és mély-háló alapú jellemzőt javasoltak már kulcspontokhoz, a vonalszakaszokhoz csak néhány módszer létezik. A vonalszakaszok azonban gyakran megtalálhatók strukturált környezetekben, különösen városi jelenetekben. Ezenkívül a vonalak stabilabbak, mint a pontok, és robusztusak a részleges takarásokra. Ezért fontosak az olyan alkalmazásokhoz, mint a pózbecslés, a vizuális odometria vagy a 3D-s rekonstrukció. Módszerünk egy 2 lépéses mélykonvolúciós neurális hálózati architektúrán alapul: az 1. szakaszban lehetséges 2D vonalszakaszokat detektálunk, a 2. szakaszban pedig leírót generálunk a kinyert vonalakhoz. A hálózat tanítása önfelügyelt módon történik egy automatikusan gyűjtött adatkészlet segítségével. A kísérleti eredmények megerősítik a javasolt L2D2 hálózat legkorszerűbb teljesítményét két jól ismert autonóm vezetési adathalmazon, mind a detektált vonalak párosítása, mind pedig az egyenes alapú kamerapózbecslési és -követési alkalmazások tekintetében.

Chapter 1

Introduction

Computer vision is understood as the host of techniques to acquire, process, analyze, and understand complex higher-dimensional data from our environment for scientific and technical exploration [70]. Simply computer vision aims to create a model of the real world using cameras for analysing or understanding. This discipline became a key technology in many areas, from industrial usage to simple end users [29, 101, 146].

In recent years, perceptual interfaces [167] have emerged to motivate an increasingly large amount of research within the machine vision community; some of the areas are structure from motion, stereo matching, person tracking algorithms, face detection algorithms, bundle adjustment based approaches to scene reconstruction, and camera calibration, stereo vision, point cloud segmentation, and pose estimation of rigid, articulated, and flexible objects [159, 184]. despite all this advancement, the computer vision problem is still difficult. Why is vision so difficult? In part, it is because vision is an inverse problem, in which we seek to recover some unknowns given insufficient information to specify the solution fully. In this thesis, novel solutions are presented for the camera pose estimation problem. The proposed technique provides solutions for estimating the camera pose using lines with and without the vertical direction and how to acquire the 2D-3D lines.

In the literature, the camera pose estimation is an essential step in many applications, including robotics, 3D reconstruction. The problem is also known as extrinsic camera calibration. It addresses the issue of determining the position and orientation of a camera with respect to a world coordinate frame. Mainly the pose estimation refers to two cases, absolute and relative camera poses. The camera pose estimation has been extensively studied, yielding various formulations and solutions. Most of the approaches focus on a single perspective camera pose estimation using many 2D–3D point correspondences, known as the Perspective-n-Point (PnP) problem. In this thesis we explore the pose estimation problem using 2D-3D line correspondences, which is known as the Perspective-n-Line (PnL) problem. This problem can be solved with at least a minimum number of 3 2D-3D line correspondences. In fact, using line segments is not new since it preserves the structural information of the scene much better than points. We also proposed a solution to acquire 2D-3D repeatable lines, which are good for pose estimation through a fully automatic process, which relies on a convolutional neural network to detect and match 2D lines.

For a clear overview, the thesis is split into 4 chapters, as follows:

Chapter 2 introduces the reader to the basic aspects of the pose estimation problem with and without prior information provided by IMU sensors using 2D-3D line pairs, in a single view camera or a multi-view camera system; also we described a model which enables us to deal with both traditional perspective and omnidirectional cameras.

Chapter 3 explores first the pose estimation State-of-the-Art, then we explain in detail the 3 proposed solutions for the pose estimation problem using 2D-3D lines, the 3 solutions are based on the same geometrical observation, which uses the relation between a line in 3D and a 2D line observed from a camera, each of the solutions uses a different solving derivation to create robust solvers. The first solution uses known vertical direction, which can be acquired from an IMU sensor and is suitable for multi-view perspective cameras. The second solution is suitable for a single or multi-view perspective camera system. The third solution relies on the Cayley rotation representation, the solution was designed robustly to resist to noise on the input data and compatible with single or multi-view omnidirectional or perspective camera system. Finally, results on synthetic data are shown, proving the efficiency of the methods, of course, each of the methods presents an example of applications on real data like drone localisation, or data fusion, which are useful in all the pose estimation related topics.

Chapter 4 addresses the problem of extracting the 2D-3D line correspondences; this topic is essential for our proposed pose estimation solution. After mentioning the background of the different techniques used in this research for line detection or line descriptor, a complete system design is presented, with two different stages. The first stage focuses on line segment detection and how we achieved to extract lines on images through a deep neural network-inspired solution with the concept of repeatable lines, good for pose estimation. In the second stage, we address the solution for 2D line segment matching, with a designed deep neural network to get a robust descriptor using a training data that is fully automatically acquired using our proposed algorithm. The algorithm provides the best 2D-3D line correspondences. In the end, results on real data are shown and proving the efficiency of our line detector/descriptor and experimented with our robust pose estimation solver.

Chapter 5 draws the conclusions on the different experiments.

Chapter 2

Fundamentals

2.1 Camera Geometry

The success of the computer vision field relies on the use of an important key component (besides the lighting and the vision processing) which is the camera! The camera allows us to see and capture the world through the lens and directed to the sensor in the form of light [58, 146]. The sensor will convert this light into digital information, which is transferred to the processor for analysis and treatment to produce the final image, whereas the captured image depends mainly on the type of lens because it will define the amount of information (field of view) that we are capturing [141]. Since the human likes to observe and inspired by nature to achieve inventions or build new technology, eyes in nature are defined differently compared to human; some of the insects or animals see over a larger field of view which let them hunt or escape in critical situation (thanks to their unique eye geometries) [63]. Considering these fact, human has derived a new type of lens called omnidirectional camera, which is mainly known as fisheye lens, that are capable of amazingly wide warp the scene interestingly, and let us get a larger field of view compared to the widely used classical lenses (perspective). In this section, we introduce the fundamentals of the perspective and the omnidirectional model used in the thesis, which can be seen as a framework for perspective, omnidirectional, or a hybrid vision system that uses both types. The proposed solutions, which use the two representations not only simplifies the usage of different camera model, but also facilitates to build hybrid heterogeneous central camera system.

2.1.1 Perspective Camera Projection and Rigid Body Transformations

The perspective model is also known as the pinhole camera since it describes the image formation process of a simple optical device with a small hole. This model represents an accurate description of modern cameras where light is focused in a single point (through the virtual pinhole at the focal point). The 3D world information is mapped into a 2D image plane by a projective transformation [58].

The perspective camera model is formulated by an equation that describes how a point in space is mapped into an image, where the center of projection is behind the image plane. This formulation is described using matrix representations. In this form, points can be represented in the euclidian space, yet a simpler notation is developed using homogeneous coordinates (homogeneous coordinates simplify the formulation as matrix multiplications). Indeed Its main advantage is that the image transformations like rotations, translation, change of scale become a simple matrix multiplications and to be represented as linear transformations using 4x4 matrices.

Let's assume that a 3D point $\mathbf{X}_c$ is expressed in the Cartesian coordinate system by

$[x \quad y \quad z]^T$, then it is possible to map any such point $\mathbf{X}_c$ into the homogeneous coordinates $\mathbf{X}$ by adding an extra dimension [58]:

$$\mathbf{X}_c = \begin{bmatrix} x & y & z \end{bmatrix}^T \leftrightarrow \mathbf{X} = \begin{bmatrix} hx & hy & hz & h \end{bmatrix}^T \tag{2.1}$$

Cartesian coordinates define the euclidean space, and the points in homogeneous coordinates define the projective space [58].

The perspective camera model can be defined by providing two sets of parameters [58]:

- Intrinsic camera parameters: the parameters necessary to link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame (internal structure of the camera).

$$\mathbf{K} = \underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} 1 & s/f_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}} \tag{2.2}$$

  where f_x and f_y are the focal lengths, x_0 and y_0 is the principal point offset, and s is the axis skew.

- Extrinsic camera parameters: the parameters that define the translation and orientation of the reference camera with respect to a known world reference frame. The extrinsic matrix takes the form of a rigid transformation matrix: a 3x3 rotation matrix $\mathbf{R}$ in the left side, and 3x1 translation column vector $\boldsymbol{t}$ in the right side:

$$[\mathbf{R} \mid \boldsymbol{t}] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{array}\right] \tag{2.3}$$

  where $\mathbf{R}$ is composed by rotations along each axis. If α, β, and γ are the rotation angles, then:

$$\mathbf{R} = \mathbf{R}_Z(\gamma)\mathbf{R}_Y(\beta)\mathbf{R}_X(\alpha) \tag{2.4}$$

$$\mathbf{R} = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{2.5}$$

The full projection matrix $\mathbf{P}$ which describes how to transform a points in the world coordinate system into the camera coordinate system is written below:

$$\mathbf{P} = \overbrace{\mathbf{K}}^{\text{Intrinsic Matrix}} \quad \overbrace{[\mathbf{R} \mid \mathbf{t}]}^{\text{Extrinsic Matrix}} \tag{2.6}$$

where the vector $\mathbf{t}$ is the translation of the world coordinate system into the camera coordinates, and the columns of $\mathbf{R}$ represent the directions of the three world-axes in the camera coordinates. Some of the camera parameters can be estimated prior to the proposed solution, thus we assume that the cameras are fully calibrated. In other words, their intrinsic parameters are known prior to the experiments, they can be obtained by standard camera calibration methods (e.g. MATLAB Calibration Toolbox [163]). As a result we get the camera

calibration matrix, $\mathbf{K}$. Remember that the Intrinsic matrix $\mathbf{K}$ is simply mapping the camera coordinate system into the image coordinate system

$$\mathbf{P} = \overbrace{\begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}}^{\text{Intrinsic Matrix}} \times \underbrace{\left(\begin{array}{c|c} I & \mathbf{t} \end{array} \right)}_{\text{3D Translation}} \times \overbrace{\underbrace{\left(\begin{array}{c|c} \mathbf{R} & 0 \\ \hline 0 & 1 \end{array} \right)}_{\text{3D Rotation}}}^{\text{Extrinsic Matrix}} \tag{2.7}$$

Unambiguously, a 3D point is mapped into the perspective image by:

$$\mathbf{x}' \cong \mathbf{P}\mathbf{X} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}, \tag{2.8}$$

where '$\cong$' denotes the equivalence of homogeneous coordinates, *i.e.* equality up to a non-zero scale factor. Since we assume a calibrated camera, we can multiply both sides of (2.8) by $\mathbf{K}^{-1}$ and work with the equivalent normalized image

$$\mathbf{x} = \mathbf{K}^{-1}\mathbf{x}' \cong \mathbf{K}^{-1}\mathbf{P}\mathbf{X} = [\mathbf{R}|\mathbf{t}]\mathbf{X}. \tag{2.9}$$

The above equation is the starting point for the perspective estimation problem [83, 89, 103, 107] addressed in Chapter 3.

2.1.2 Omnidirectional Cameras

Since conventional camera (perspective) has a very limited field of view (FOV) that make them restrictive in a variety of vision applications, one effective way to enhance the field of view of a camera is to combine multiple cameras into a one single camera framework, another way to increase the field of view is to combine the camera with a shaped lens (dioptric) or with mirrors, which is refereed as catadioptric image formation. Cameras with shaped lens or mirrors are known as omnidirectional cameras, they have been used in many computer vision and robotics areas. There exist several types of omnidirectional cameras which can be classified as central and noncentral [15, 48, 132] (Note that these types cannot be described using the conventional pinhole model because of the high distortion). Among the noncentral cameras we can find the rotating camera and polycameras. On the other hand, central omnidirectional cameras are those which satisfy the single-viewpoint property. This is an important property since it allows to easily calculate the directions of light rays coming into the camera [15, 48, 132]. All central catadioptric cameras can be modeled by a unit sphere and a perspective projection, such that the projection of 3D points can be mapped within the two type of cameras. The spherical camera model used to explain the central catadioptric systems was proposed originally by Geyer and Daniilidis [47], which represents central omnidirectional cameras as well as perspective cameras as a projection onto the surface of a unit sphere $\mathcal{S}$. The camera coordinate system is in the center of $\mathcal{S}$, and the Z axis is the optical axis of the camera which intersects the image plane in the *principal point*. In the following, we describe the recent representation of the spherical camera model used in the thesis [151, 152].

Scaramuzza's Omnidirectional Camera Model

In 2006, Scaramuzza model the omnidirectional images using parameters which describe the distortion, these parameters describe the image projection function by a polynomial based on Taylor series expansion [151, 152]. It does not require either any priori knowledge of the motion or a specific model of the omnidirectional sensor. Hence, the general polynomial form proposed by Scaramuzza *et al.* [151, 152] is easy to apply for different types of cameras.

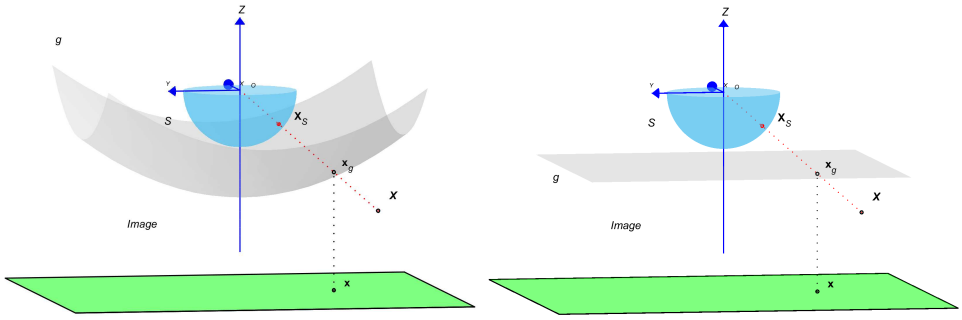


Figure 2.1. *Omnidirectional camera model (left) and perspective camera model (right) using Scaramuzza's spherical representation [64, 151, 152].*

Following [151, 152], we assume that the camera coordinate system is in the unit sphere $\mathcal{S}$, the origin is the effective projection center of the omnidirectional camera. Let us first see the relationship between a point $\mathbf{x} = [x_1, x_2]^\top \in \mathbb{R}^2$ in the image $\mathcal{I}$ and its representation $\mathbf{X}_\mathcal{S} = [X_{\mathcal{S}1}, X_{\mathcal{S}2}, X_{\mathcal{S}3}]^\top \in \mathbb{R}^3$ on the unit sphere $\mathcal{S}$ (see Fig. 2.1). Note that only the half sphere on the image plane side is actually used, as the other half is not visible from image points. [151, 152] places a surface g between the image plane and the unit sphere $\mathcal{S}$, which is rotationally symmetric around z (see Fig. 2.1). As shown by [151, 152] polynomials of order three or four are suitable for accurately modeling all commercially available catadioptric and many types of fisheye cameras as well, thus we used a fourth order polynomial:

$$g(\|\mathbf{x}\|) = a_0 + a_2\|\mathbf{x}\|^2 + a_3\|\mathbf{x}\|^3 + a_4\|\mathbf{x}\|^4, \tag{2.10}$$

The 4 parameters (a_0, a_2, a_3, a_4) represent the internal parameters of the camera (only 4 parameters as a_1 is always 0 according to [152]). But, if we set all the parameters of g to be zero, except the constant a_0 we would get a perspective camera (g is a planar surface parallel to the image plane) as seen in Fig. 2.1. The consequent bijective mapping $\Phi : \mathcal{I} \to \mathcal{S}$ is given by

1. lifting the image point $\mathbf{x} \in \mathcal{I}$ onto the g surface by an orthographic projection

$$\mathbf{x}_g = \begin{bmatrix} \mathbf{x} \\ a_0 + a_2\|\mathbf{x}\|^2 + a_3\|\mathbf{x}\|^3 + a_4\|\mathbf{x}\|^4 \end{bmatrix} \tag{2.11}$$

2. then centrally projecting the lifted point $\mathbf{x}_g$ onto the surface of the unit sphere $\mathcal{S}$:

$$\mathbf{X}_\mathcal{S} = \Phi(\mathbf{x}) = \frac{\mathbf{x}_g}{\|\mathbf{x}_g\|} \tag{2.12}$$

Thus the omnidirectional camera projection is fully described by means of unit vectors $\mathbf{X}_\mathcal{S}$ in the half space of $\mathbb{R}^3$ and these points correspond to the unit vectors of the projection rays.

From the Spherical Model to the Perspective Camera

Similarly, the image points of a perspective camera can be represented on the unit sphere $\mathcal{S}$ by the bijective mapping $\mathbf{x} \mapsto \mathbf{x}_\mathcal{S}$: $\mathbf{x}_K = \mathbf{K}^{-1}\mathbf{x}$ and $\mathbf{x}_\mathcal{S} = \mathbf{x}_K / \|\mathbf{x}_K\|$ (see Fig. 2.1). Thus the projection of a calibrated central camera is fully described by means of unit vectors $\mathbf{x}_\mathcal{S}$ in the half space of $\mathbb{R}^3$. A 3D world point $\mathbf{X}$ is projected into $\mathbf{x}_\mathcal{S} \in \mathcal{S}$ by a simple central projection taking into account the pose:

$$\mathbf{x}_\mathcal{S} = \frac{\mathbf{R}\mathbf{X} + \mathbf{t}}{\|\mathbf{R}\mathbf{X} + \mathbf{t}\|} \tag{2.13}$$

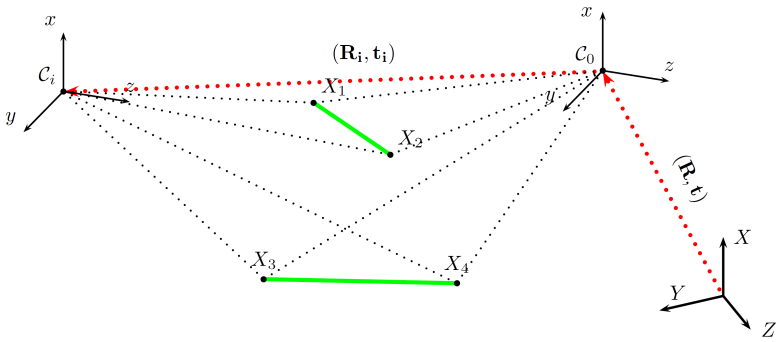


Figure 2.2. *The absolute and the relative pose visualization*

2.2 Absolute and Relative Pose

Most of the time, a single camera can't see and cover a bigger area compared to a system of multiple cameras. Defining the camera pose individually might not be always the best solution in many computer vision applications. In that case we have to deal with the absolute pose and relative poses. Absolute pose estimation of a camera consists in determining its position and orientation with respect to a reference 3D world coordinate frame, while relative pose estimation also aims to compute a rigid body transformation but with respect to another device (*e.g.* a reference camera). Relative poses provide crucial information about the actual local state of the system, while absolute pose provides a global information. Such configuration is mostly known as a multi-view camera system. So, given a set of 2D-3D line correspondences $(l_i \leftrightarrow L_i)$, that can be represented simply by the 2D-3D line endpoints, one can recover the 3D rigid body transformation noted as $\mathbf{T} = (\mathbf{R}, \mathbf{t}) : \mathcal{W} \rightarrow \mathcal{C}$ acting between the world coordinate frame $\mathcal{W}$ and the camera coordinate frame $\mathcal{C}_0$. Technically, by relative pose we refer to the rigid body transformation $\mathbf{T_i}$ that acts between the coordinate systems of the reference frame $\mathcal{C}_0$, and any second camera $\mathcal{C}_i$ and expressed relative to the first one $\mathcal{C}_0 \rightarrow \mathcal{C}_i$ (please see Fig. 2.2). By definition $\mathbf{T_i}$ acts on the line end-points X_i expressed in the reference coordinate system. The relative pose brings 3D points expressed in $\mathcal{C}_0$ into $\mathcal{C}_i$, thus the $\mathbf{T_i}$ notation can also be used. The absolute pose $\mathbf{T}$ of $\mathcal{C}_0$ and the relative pose $\mathbf{T_i}$ of $\mathcal{C}_i$ satisfy the following relation:

$$\mathbf{T}_r = \mathbf{T_i}\mathbf{T} \tag{2.14}$$

where $\mathbf{T}_r$ is the absolute pose for camera $\mathcal{C}_i$. Exactly the same transformation applies if we consider the spherical camera model, as shown previously in Section 2.1.2.

2.3 Line Representation

In an urban environment, points appear to be relatively easy to localize and provide good discrimination. However, they are known to be view-dependent; their appearance changes substantially with the camera's location, especially in scenarios involving low-textured objects, which are relatively common in man-made space. On the other hand, line segments are more robust to lighting and are present even in low-textured objects. Indeed, straight lines are standard in man-made environments and are arguably better features to track than points, they are trivially stable under a wide range of viewing angles, and a number of measurements can be made along their length to localize them accurately [157]. As a result, line features are explored here instead of points.

A 3D line may have various representations in the projective space [17, 130]. For ex-

ample, Plücker line coordinates are popular as they are *complete* (*i.e.* every 3D line can be represented) and allow for a linear projection model similar to (2.9) [17, 58, 98, 131, 199]. However, Plücker coordinates are not *minimal* because a 3D line has 4 degrees of freedom but its Plücker coordinate is a homogeneous 6-vector. However, in this thesis, the 3D lines are formulated differently and represented as:

$$L = (\mathbf{V}, \mathbf{X}), \tag{2.15}$$

As in Fig. 3.1 a 3D line can be represented in terms of a unit vector $\mathbf{V}$ which indicates the direction of the line, and $\mathbf{X}$ which is a point on the 3D line [65, 162]. This line representation will let us to set the basic equations which will be used to construct the proposed solutions in Chapter 3. Practically, this line representation allow us to describe the geometric configuration of the lines and the projection plane. In other words, we can express the scalar product between the projection plane normal and the unit direction vector. When the algebraic error is minimized (that is close to zero), it means that the cosines of the angle between these two vectors will be close to orthogonal which means that the geometric error is also minimized.

2.4 Known Vertical Direction

Inertial navigation may be defined as the indication, in real time, of position and velocity of a moving vehicle using sensors [76]. The sensors or instruments of an inertial navigation system generally are called inertial measurement units (IMU's). Conventionally, two types of IMU exist: the accelerometer that senses linear accelerations, and the gyroscope that senses angular rate [73]. Angular accelerometers also exist; however, for purposes of navigation, gyroscopes have proved to be more accurate. Each device, whether accelerometer or gyroscope, has three axes associated with it: the input axis (1-axis), the output axis (2-axis), and a third axis particular to the instrument in some way. These three axes are mutually perpendicular and they form the instrument frame [72].

Nowadays, a lot of cheap available IMUs provide very accurate roll and pitch angle, i.e. the vertical direction. Modern camera systems are often coupled with these sensors to distinguish whether the image orientation is landscape or portrait, here we show that knowing the camera vertical direction can reduce the complexity of the rotation matrix and simplify the camera pose problem [89].

Assuming that the camera coordinate system is a standard right-handed system with the X axis pointing up, the coordinates of the world vector $(1, 0, 0)^\top$ are known in the camera coordinate frame $\mathcal{C}$. Given this *up-vector*, we can compute the rotation $\mathbf{R}_v = \mathbf{R}_Z(\gamma)\mathbf{R}_Y(\beta)$ around Y and Z axes,

$$\mathbf{R}_v = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{2.16}$$

which aligns the world X axis with the camera X axis, thus the only unknown parameter in the rotation matrix $\mathbf{R} = \mathbf{R}_v\mathbf{R}_X(\alpha)$ is the rotation $\mathbf{R}_X(\alpha)$ around the vertical X axis:

$$\mathbf{R}_X(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{2.17}$$

2.5 Angular Distance and Translation Error

To measure the rotation error between an estimated pose and a Ground Truth (GT) pose, we can calculate the difference between each of the three-axis in the range of $180°$. But, this measurement doesn't provide a global view of the rotation error. Herein, we use a more sophisticated measure to have a full view of the rotation error. According to Euler theorem, a rotation can be represented by an angle-axis pair, expressing a rotation with angle θ around an axis ($0° \leq \theta \leq \pi$). Given the true rotation matrix (ground truth) $\mathbf{R}$ and the estimated one $\hat{\mathbf{R}}$, we can represent the error between the two rotations in terms of an angular distance (i.e. the rotational error), that is defined [59] as the rotation angle in the range of $[0°, \pi]$, and can be expressed with:

$$\epsilon = \hat{\mathbf{R}}\mathbf{R}^\top = \hat{\mathbf{R}}^\top\mathbf{R} = \mathbf{R}\hat{\mathbf{R}}^\top = \mathbf{R}^\top\hat{\mathbf{R}} \tag{2.18}$$

All the previous expression represent a rotation through the same angle. Note that $\hat{\mathbf{R}}\mathbf{R}^\top = \mathbf{I} = 0°$ in the case that the estimated rotation $\hat{\mathbf{R}}$ is perfectly equal to the ground truth rotation $\mathbf{R}$, and as in [59] if r and $\hat{r}$ are quaternion representations of $\hat{\mathbf{R}}$ and $\mathbf{R}$ respectively, then the angular distance ϵ can be easily calculated by:

$$\epsilon = 2\arccos(|c|) \text{where } (c, a) = \hat{r}^{-1}r \tag{2.19}$$

Note that $c = \cos(\epsilon/2)$ and a is a vector of length $\sin(\epsilon/2)$ representing the rotation axis. The absolute value sign in $|c|$ is required for the sign ambiguity in the quaternion representation of the rotation $\hat{\mathbf{R}}^\top\mathbf{R}$ which guarantee that the angle ϵ lies in the range $[0, \pi]$ as required by the definition of the angular distance [59].

For the translation error, we simply use the L2 norm between the estimated translation and the GT translation. Note that the translation error can be also measured in terms of the percentage translation errors relative to the ground truth translation.

2.6 Normalization of the 3D Data

In Most computer vision applications, the 3D data is given in an arbitrary world coordinate system $\mathcal{W}$ as it is captured by the 3D sensors. But for numerical stability of our solutions (proposed in Section 3.4 and Section 3.5), we will transform our 3D line segments into a unit cube around the centroid of the 3D scene: First a uniform scaling factor κ is calculated using the maximum domain of the coordinates along all three axes, practically the height h, width w and depth d of the data, then choosing the maximum of these as the uniform scaling measure $\kappa = \frac{1}{max(|h|,|w|,|d|)}$. After we calculate the centroid $[u, v, z]^T$ of the 3D scene points, the normalization matrix $\mathbf{N}$ is then composed of the translation $-[u, v, z]^T$ followed by a uniform scaling by κ.

$$\mathbf{N_3} = \begin{bmatrix} \kappa & 0 & 0 & 0 \\ 0 & \kappa & 0 & 0 \\ 0 & 0 & \kappa & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -u \\ 0 & 1 & 0 & -v \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

We used uniform scaling to avoid changing the direction vector of the 3D line. This way, the rotational part of the pose is not affected by this normalization. Normalization is then applied to the 3D line points $\mathbf{N}_3\mathbf{X}$ used in (3.4) and (3.6). The solution is then obtained in this normalized space, hence the result $(\tilde{\mathbf{R}}, \tilde{\mathbf{t}})$ need to be denormalized. Since the equations used to solve for the rotation are unaffected by this normalization (thanks to uniform scaling!), $\tilde{\mathbf{R}}$ is the final rotation, while the translation $\tilde{\mathbf{t}}$ needs to be corrected by

applying $\begin{bmatrix} \mathbf{I} & \tilde{\mathbf{t}} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{N}_3$.

Handling Relative and Absolute Pose

When we consider multiple cameras, the absolute and relative poses are applied in a chain of transformations, such as for a 3D homogeneous world point $\mathbf{X}$ we apply $\begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_0 & \mathbf{t}_0 \\ 0 & 1 \end{bmatrix} \mathbf{X}$, where $(\mathbf{R}_0, \mathbf{t}_0)$ is the absolute pose, and $(\mathbf{R}_i, \mathbf{t}_i)$ the relative. This way when applying data normalization with $\mathbf{N}_3$, the relative pose $(\mathbf{R}_i, \mathbf{t}_i)$ cannot be handled separately as shown before, instead, we can denormalize the estimated full pose of the relative camera(example second camera) at once as $\begin{bmatrix} \mathbf{R_f} & \mathbf{t_f} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{R}_{i=1}} & \tilde{\mathbf{t}_{i=1}} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{R}_0} & \tilde{\mathbf{t}_0} \\ 0 & 1 \end{bmatrix} \mathbf{N}_3$. Finding the denormalized relative pose is done by denormalizing the absolute pose of the reference camera and the full pose also, then taking the relative pose between these two, as $\begin{bmatrix} \mathbf{R_r} & \mathbf{t_r} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R_f} & \mathbf{t_f} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R_a} & \mathbf{t_a} \\ 0 & 1 \end{bmatrix}^{-1}$ The translation $\mathbf{t_r}$ of the denormalized relative pose has to be scaled back to original scale using the same division with the norm of $\|\mathbf{R_a}\|$.

2.7 Convolutional Neural Network

CNN are deep learning computer vision networks that can recognize and classify picture features. CNN's architecture are inspired by the organization and functions of the visual cortex in our brain. It is designed to mimic the connections between neurons in the human brain [139]. CNN are used for computer vision tasks, which solve image processing and Machine learning-based problems, such as object identification, image recognition, image classification, image segmentation [39]. It is viewed as a refined and a boosted machine learning algorithm due to the high performance in learning and perception that can even surpass human performance [60]. However, for training, CNN requires a huge amount of data. CNN has mostly demonstrated remarkable success in many computer vision applications like traffic sign identification, medical picture segmentation, face detection [20, 198], and object identification in natural photos, where there is sufficient labeled data available for training [20]. But, image recognition is a task that humans have been performing since their existence. Like children learn to recognize objects since their birth, so is it possible to teach computers to perform the same thing? And is it feasible for a human to build a machine that can perceive just like humans [11, 14]? Thanks to the advancement of AI and neural networks, nowadays, the answer to this question has become a yes [21, 54]. Just as humans, a computer must see several examples of images before it can generalize the input and make predictions for images that it has never seen before [54, 147]. So, how does the image appear to computers? For a human, our brain does the work but still impossible for us to answer this question completely [41, 86]; on the other hand, computers see an object as 0 and 1 or as numerical data at a higher level [137]. As a result, developing a computer that can analyze and recognize images is quite a complex task [43]. With the advancement of technology, almost everyone knows that pixels are the building blocks of an image taken by a camera. Each pixel has value information of the brightness and hue, in the range of 0-255 [90]. When humans see an image, their brains immediately process the higher level information intuitively [36]. Neurologists and scientists proved that this happens through each receptive in neurons located in the human brain that are connected to other neurons in order to cover the full visual field. The receptive field is a small proportion of the visual field, where each neuron in the biological vision system responds to stimuli [12]. In the same way, each neuron in CNN analyzes data in only its receptive area. The CNN layers are programmed to identify simpler patterns first, such as lines and curves, before progressing to more complex patterns, such as faces and objects. As a result, it is plausible to claim that

using a CNN may provide vision to computers[20].

2.8 From Perceptron to Convolutional Neural network

The perceptron (Fig. 2.3) is a representation of the biological neuron in the human brain that performs human-like brain functions [31]. In Machine Learning, it is a supervised learning algorithm of binary classifiers. The perceptron conducts binary classification (linear classifier) or two-class categorization and enables neurons to learn and register information procured from the inputs [31, 87, 136, 142]. Invented by Frank Rosenblatt in 1957 is known as the most basic form of a neural network [23, 31, 142]. The perceptron constituents of 4 models: Input values, Weights and bias, the sum, and the Activation function [8]. furthermore, perceptrons only have one input and one output layer, so they are mathematically not very complex to understand:

$$\widehat{z} = \widehat{b} + \sum_i \widehat{w}_i \widehat{x}_i. \tag{2.21}$$

where $\widehat{w}_i$ are the inputs, $\widehat{w}_i$ are the weights, $\widehat{b}$ is the bias and $\widehat{z}$ is the resultant logit.

$$\widehat{y} = \begin{cases} 1 & , \widehat{z} \geq 0, \\ 0 & \text{otherwise}. \end{cases} \tag{2.22}$$

The second equation (2.22) defines the decision. Note that the output can be represented as 1 or 0. It can also be represented as 1 or -1 depending on which activation function is used, the activation function can be linear, non-linear, sigmoid, ReLu, ... etc[22, 50]. In the case of perceptrons, the activation function is a simple function which returns 1 if the whole sum is over 0 and returns 0 otherwise. This is called the step function because it jumps from 0 to 1 when the input is over 0 [8, 23, 31]. Perceptrons have as many input and output neurons as needed, but they will always have an additional input neuron called bias. The reason why a bias is a need is that because without it, the perceptron would not learn about a non-zero input and a zero output or vice versa. When the sum is computed, if all inputs are zero, the result will be therefore zero, and the output will inevitably be zero as well [8, 23]. Hence, the weights can never learn because the result will always lead to zero. An artificial neural network is created by connecting one or more neurons to the input Fig. 2.4. Each pair of neurons may or may not have a connection between them[7] Depending on how neurons are connected, a network act differently.

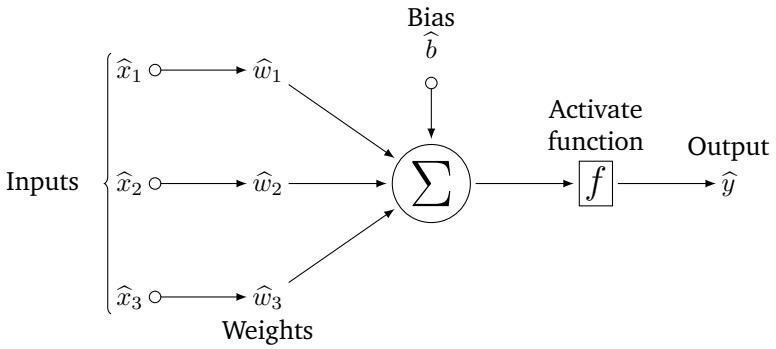


Figure 2.3. *Illustration of a single perceptron [8]*

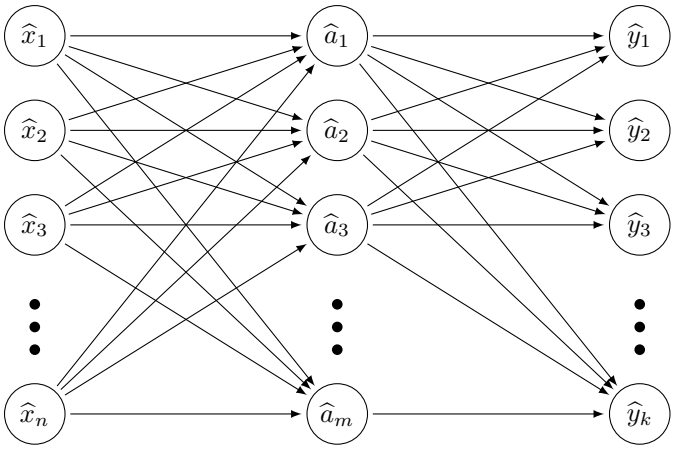


Figure 2.4. *Illustration of a multilayer perceptron [28, 139]*

In the 1980s, Thanks to Yann LeCun, a postdoctoral computer science researcher, who introduces first the notion of convolutional neural networks [96], CNN was built on the work done by Kunihiko Fukushima, a Japanese scientist who, a few years earlier, had invented the neocognitron, a very basic image recognition neural network [52]. The early version of CNN, called LeNet, could recognize handwritten digits, and was used by banking and postal services to read zip codes on envelopes and digits on checks [52, 96]. Nevertheless, due to the technology limitation, it remained on the sidelines of computer vision and artificial intelligence. CNN need many data and computed resources to work efficiently for large images. At the time, the technique was only applicable to images with low resolutions. Overall, convolutional neural networks played an essential role in the recent success of deep learning in computer vision [20, 138]. The Modern CNN is proposed by [97] and the building blocks have been similar since 1998 (see Fig. 2.5). CNN always contains two basic operations, namely convolution, and pooling [208].

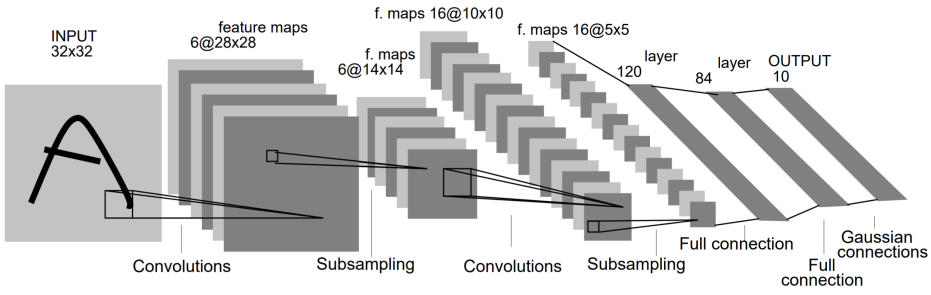


Figure 2.5. *Architecture of LeNet-5 a Convolutional Neural Network for digits recognition, proposed by [96, 97]*

Convolution operation:

The Convolutional Neural Network (CNN) is a type of feed-forward neural network with sparsely connected layers, unlike traditional neural networks, where each layer is fully connected. This is done by connecting a local region of the input layer to the neuron in the next layer[114]. CNN assumes by definition that the inputs are images; this enables us to encode

specific properties in the architecture to recognize patterns in the images. The purpose of convolutions here is to extract features from the input image, in the sense that it perceives different objects similar to human perception based on shapes, size, and colors [7, 20]. CNN can use a variety of detectors (such as edge detectors, corner detectors) to interpret images. Convolutions preserve the spatial relationship between pixels by learning the high-level image features using small squares of input data. Since every image can be considered as a matrix of pixel values. The convolution (see Fig. 2.6) is executed through a sliding matrix over the original image by the stride value; for every pixel position, we compute the element-wise multiplication between the two matrices, and then we add the multiplication outputs to get the final value which forms a single element of the output matrix [7]. In CNN terminology, the sliding matrix is called filter or kernel, and the resulting matrix is called the activation map or the feature map. In practice, a CNN learns the values of these filters on its own during the training process. The feature map size depends on three parameters before the convolution step is performed [7, 20, 142].

- Depth(channels): corresponds to the number of filters we use for the convolution operation. Each filter produces a feature map of the resulting tensor.
- Stride: is the size of the step we move the convolution filter at each step in pixels over the input matrix. Having a larger stride will produce smaller feature maps, while a smaller stride will use all the information available from the input features.
- Zero-padding: sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix and preserve the image size.

Since we use PyTorch in the experimental part [126], the output value of the layer with input size $\left(\widehat{N}, C_{\text{in}}, H, W\right)$ and output $\left(\widehat{N}, C_{\text{out}_j}, H_{\text{out}}, W_{\text{out}}\right)$ can be precisely described as:

$$\text{out}\left(\widehat{N}_i, C_{\text{out}_j}\right) = \text{bias}\left(C_{\text{out}_j}\right) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}\left(C_{\text{out}_j}, k\right) \star \text{input}\left(\widehat{N}_i, k\right)$$

where $\star$ is the valid 2D cross-correlation operator, $\widehat{N}$ is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels. Note that The only difference between the convolution and cross-correlation operations is that the kernel is flipped (along all spatial dimensions) before being applied, which reduces the function call and the computation time. The advantage of cross-correlation is that it avoids the additional step of flipping the filters to perform the convolutions [75]

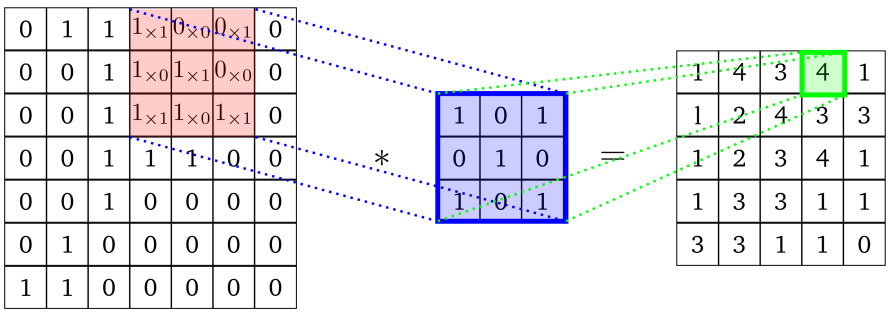


Figure 2.6. *Example of a convolution operation with a* $\mathbf{3} \times \mathbf{3}$ *kernal*

Pooling operation: Pooling can be used to downscale the dimensionality of the feature maps, reducing their width and height while maintaining the depth intact and allowing to reduce the number of parameters which shortens the training time and prevents overfitting

[133]. Pooling layers downsample each feature map independently. Pooling can be different types: Max, Average, Sum, etc.. Max pooling (Fig. 2.7) is the most common type of pooling, which takes the maximum value in each window. Pooling does not have any parameters.

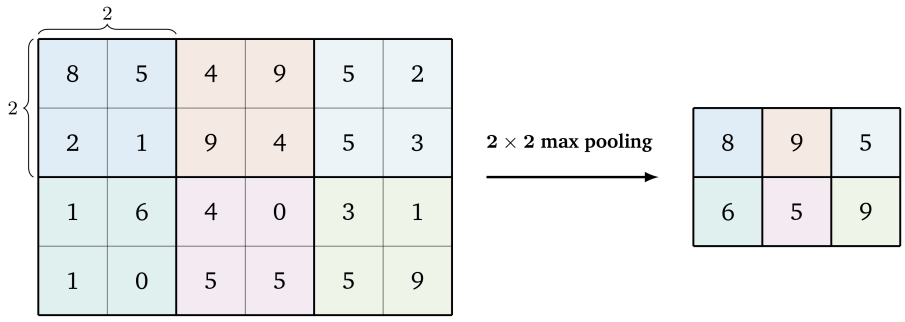


Figure 2.7. *Example of a max pooling operation using a* $\mathbf{2 \times 2}$ *kernal*

Chapter 3

Camera Pose Estimation with 2D-3D Line Pairs

3.1 State-of-the-Art Overview

Camera pose estimation is a typical task in the computer vision and robotics field; it is essential to allow moving devices like a car, drone, or robot with one or multiple cameras to navigate and avoid obstacles [44, 186]. In general, we have to specify the type of the camera pose that we are manipulating or solving, thus absolute or relative, which is usually needed when a system of two or more sensors is considered [6, 85]. Absolute and relative poses are fundamental in various computer vision applications, such as visual odometry [104, 175], simultaneous localisation and mapping (SLAM) [105, 134], image-based localisation and navigation [46, 91], augmented reality [74, 195]. The problem has been extensively studied, yielding various formulations and solutions. Most of the approaches focus on a single perspective camera pose estimation using point correspondences [32, 61, 81, 84, 103, 125]. However, in modern applications, especially in vision-based localisation and navigation for robotics and autonomous vehicles, it is often desirable to use multi-camera systems which cover large fields of views and provide direct 3D measurements [6]. This problem is known as non-perspective absolute pose estimation such as a camera system may be modeled as a virtual camera with many projection centers. Furthermore multi-camera systems containing a mixture of perspective and omnidirectional cameras are becoming more and more desirable by many modern applications since they fit well the need of special vision-based robotics and autonomous vehicle localization and navigation applications [2, 42, 51, 64, 71, 111, 160], where a higher field of view is often necessary for a robust interpretation of highly complex urban scenes. This topic is in the center of interest for both the academic and industrial areas, especially for autonomous driving in urban environments where pose estimation is needed to navigate in complex environments such as [49, 66, 109] which provide access to precise sensors and different types of cameras.

The topic of the camera pose estimation has been widely studied in the last few decades, which can be established using the image content and visual information with different complexity (*e.g.* points, lines, regions, or even higher level semantic objects). Many studies and research have been conducted about the various solutions; one is the absolute pose estimation of a perspective camera from n 2D-3D point correspondences which is known as the *Perspective n Point* (PnP) problem [40, 78, 83, 103, 107], as well as the P3P which is the most common absolute pose estimation approach, that can be solved with a minimum number of 3 correspondences. A first solution of the P3P problem was introduced by Grunert [56] for a geodesic application. Later one several methods have been proposed like [40, 110] where they employ the cosines functions to formulate a system of three quadratic

equations in the features distances from the camera; other solutions are using Sylvester resultant [35] or and Wu-Ritz's zero-decomposition method [183], the problem is that those methods are time-consuming and numerical unstable which involve a less accuracy on the estimated pose. Some of the recently proposed method use a geometric approach and directly solving the camera pose by avoiding the calculations of the features distances or formulating the problem directly as quartic polynomial [78]. Various solutions have been developed proving that this problem can be solved accurately in linear time with respect to the number of correspondences [83] for both large n as well as for the $n = 3$ minimal case, like [27, 61, 83, 84, 103, 205], moreover [32, 81, 125] presented minimal solvers for the generalized PnP problem and for a multi-camera system [82].

As we have seen, while several point-based methods exist, the use of line correspondences, known as the *Perspective n-Line* (PnL) problem, has also been investigated in the last two decades. Using line features instead of points is an attractive alternative in urban environments and road scenes since point correspondences are less reliable in an urban environment due to the repetitive structures. Using line features instead of points is an attractive alternative in such scenarios. The minimal case of $n = 3$ line correspondences is particularly important as its solution is the basis for dealing with the general PnL problem. It has been shown in [33], that P3L leads to an 8^{th} order polynomial, which is higher than the 4^{th} polynomial of a P3P problem. The first globally optimal non-iterative solution for the absolute pose of a single camera (AlgLS), proposed by [118], formulates the problem as a multi-variate polynomial system with rotation parametrized by the Cayley-Gibbs-Rodriguez (CGR) representation. Zhang *et al.* proposed RPnL [203] which was further modified into the Accurate Subset-based PnL (ASPnL) method [188], which is one of the most accurate non-iterative methods. Another recent work from Wang *et al.* deals with the P3L [178] as well as with the PnL problem [179] which is a fast and robust solution (called SRPnL) [179] confirmed its superior performance by a comprehensive experimental comparison with many States-of-the-Art PnL solvers, like AlgLS [118], ASPnL [188]. For multi-view camera systems, one notable work is the minimal NP3L solver of Lee [98], which deals with the 6 pose parameter estimation for a fully calibrated multi-view perspective camera system. More details are presented in [188]. Some of the solutions are proposed with known vertical direction [64, 65, 95], in [65] leads to two fast and robust solvers. Most recently, a solution using the hidden variable method has been proposed by [177] that parametrize the rotation using the Cayley–Gibbs–Rodriguez (CGR) parameterization and formulate the PnL problem into a polynomial system which are solved with the hidden variable method and polished via the Gauss–Newton method. Additionally, [194] proposed a high speed and global optimality solution which is based on a nonlinear least squares (non-LLS) formulated by parameterizing the rotation matrix with the Cayley representation. Moreover, Liu proposed a globally optimal camera orientation estimation algorithms by decoupling the rotation and translation estimation and applying the Branch-and-Bound (BnB) algorithm which search globally the entire rotation space to obtain the optimal camera orientation [112]. Another solution proposed by [170] focus on PnP(L) solvers for the uncertainty-aware pose estimation.

While the use of point and line correspondences are widespread, there are pose estimation methods relying on the combination of points and lines [37, 145, 171, 204]. Furthermore, there are pose estimation methods relying on matching sets of 2D-3D regions [160, 161] or silhouettes [201]. More recently [127] proposed a more complex pipeline relying on point-to-plane mismatches, where image sets are registered to a 3D pointcloud through the use of structured scenes and polynomial sum-of-squares optimization framework. However, such approaches cannot be used in real-time driving and navigation applications due to their increased complexity. As for the relative pose, lines in two images do not provide any constraints on the camera pose [58]. However, if the information is available about some special 3D geometric configuration of the lines, then relative pose can also be estimated [38]. In [175], relative pose estimation is extended with a 3D

line direction estimation step applied for monocular visual odometry, while [108] used a line-based space resection approach for UAV navigation. Recently, in [206], lines of building structures were used for trajectory estimation in a SLAM approach, while in [134], a hybrid point and line-based SLAM technique was proposed. Recently, deep neural network has been also used for pose estimation like [123] a robust camera pose estimation using multiple feature descriptor databases, [30, 140, 192] solve the relative camera pose on a multi-view environment and autonomous navigation of UAVs. Mainly in these methods feature need to be extracted from a large datasets, for example ORB algorithm to extract the feature points [135] or VGG19 model [182].

None of the methods mentioned previously estimate full absolute and relative poses simultaneously in a multi-view camera system using 2D-3D line correspondences with realistic conditions that include noise or outliers, either on perspective cameras or including both omnidirectional and perspective cameras. Moreover, while there are efficient solutions to line detection/matching in hybrid perspective and/or omnidirectional camera systems are by far less researched in the context of line-based pose estimation.

3.2 Starting Point of the Pose Estimation with Lines

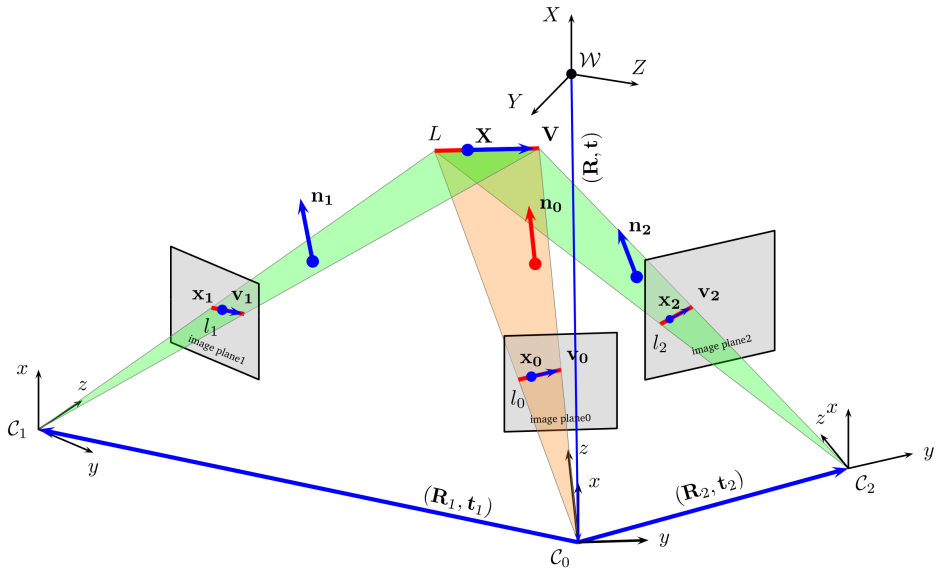


Figure 3.1. *Projection of a 3D line in a 3 cameras system [1, 4].*

Pose estimation consists of computing the position and the orientation of a camera with respect to a 3D world coordinate system, this problem has 6 degree of freedom 3 on the rotation and 3 on the translation. In this section we explain the basis part of the three proposed solvers, which is the derived equations based on a geometrical observation. As mentioned previously in Section 2.3, we are using line segments which are represented by a unit direction vector $\mathbf{V}$ and a point on the line $\mathbf{X}$. Let us imagine that we have multiple cameras N. So, the projection of a 3D line L in a multi-view camera system produces one 2D line $l_i, i = 0, 1 \dots N$ in each image plane, which can also be represented as:

$$l_i = (\mathbf{v}_i, \mathbf{x}_i), i = 0, 1 \dots N. \tag{3.1}$$

Note that the point $\mathbf{x}_i$ is *not* necessarily the image of the 3D point $\mathbf{X}$! Intuitively, for each camera i (i = 0,1$\dots N$), both L and l_i lie on a projection plane π_i passing through the camera projection center $\mathbf{C}_i$. The unit normal to the plane π_i in the camera coordinate system $\mathcal{C}_i$ is denoted by $\mathbf{n}_i$, which can be computed from the image line l_i as:

$$\mathbf{n}_i = \frac{(\mathbf{v}_i \times \mathbf{x}_i)}{\|\mathbf{v}_i \times \mathbf{x}_i\|} \tag{3.2}$$

For the reference camera $\mathcal{C}_0$, Since L lies also on π_i, its direction vector $\mathbf{V}$ is perpendicular to $\mathbf{n}_i$. Hence, we get the following equation which involves only the absolute pose $(\mathbf{R}, \mathbf{t})$ [65]

$$\mathbf{n}_0^\top \mathbf{R}\mathbf{V} = \mathbf{n}_0^\top \mathbf{V}^{\mathcal{C}_0} = 0, \tag{3.3}$$

where $\mathbf{R}$ is the rotation matrix from the world $\mathcal{W}$ to the reference camera $\mathcal{C}_0$ frame and $\mathbf{V}^{\mathcal{C}_0}$ denotes the unit direction vector of L in the reference camera coordinate frame. Furthermore, the vector from the camera center $\mathbf{C}_0$ to the point $\mathbf{X}$ on line L is also lying on π_0, thus it is also perpendicular to $\mathbf{n}_0$:

$$\mathbf{n}_0^\top (\mathbf{R}\mathbf{X} + \mathbf{t}) = \mathbf{n}_0^\top \mathbf{X}^{\mathcal{C}_0} = 0, \tag{3.4}$$

where $\mathbf{t}$ is the translation from the world $\mathcal{W}$ to the reference camera $\mathcal{C}_0$ frame and $\mathbf{X}^{\mathcal{C}_0}$ denotes the point $\mathbf{X}$ on L in the reference camera coordinate frame.

In the case of N cameras, a 3D line L has up to N images, one in each camera $\mathcal{C}_{i=1\ldots N}$. These cameras may be assembled into an ad-hoc multi-camera system or they might originate from a single camera moving along a trajectory [13, 27, 99, 129] – in either case, they form a camera system with unknown relative poses $(\mathbf{R}_i, \mathbf{t}_i) : \mathcal{C} \rightarrow \mathcal{C}_i$ with respect to the reference camera coordinate frame $\mathcal{C}$.

The projection of L yields similar equations but the unknown relative pose $(\mathbf{R}_i, \mathbf{t}_i)$ will also be involved. Hence (3.3) becomes:

$$\mathbf{n}_i^\top \mathbf{R}_i \mathbf{V}^{\mathcal{C}_0} = \mathbf{n}_i^\top \mathbf{R}_i \mathbf{R} \mathbf{V} = 0 \tag{3.5}$$

Similarly, (3.4) can be written for the other camera as:

$$\mathbf{n}_i^\top (\mathbf{R}_i \mathbf{X}^{\mathcal{C}_0} + \mathbf{t}_i) = \mathbf{n}_i^\top (\mathbf{R}_i (\mathbf{R}\mathbf{X} + \mathbf{t}) + \mathbf{t}_i) = 0 \tag{3.6}$$

Equation (3.5) involves only the rotations whereas (3.6) involves both rotation and translation.

For omnidirectional cameras (see Fig. 3.2), a 3D line L is centrally projected by a projection plane $\pi_L = (\mathbf{n}, d)$ onto the surface of the unit sphere $\mathcal{S}$ as explained in Section 2.1.2. Since the camera projection center is also on π_L, d becomes zero and thus π_L is uniquely determined by its unit normal $\mathbf{n}$. The image of L is the intersection of the *ray surface* $\mathcal{S}$ and π_L, which is a *great circle*, while a particular line segment becomes a *great circle segment* on the unit sphere $\mathcal{S}$ with endpoints $\widetilde{\mathbf{a}}$ and $\widetilde{\mathbf{b}}$ (both are on $\mathcal{S}$, hence they are unit length!). The unit normal $\mathbf{n}$ to the projection plane π_L in the camera coordinate frame $\mathcal{C}$ is then given by:

$$\mathbf{n} = \widetilde{\mathbf{a}} \times \widetilde{\mathbf{b}}. \tag{3.7}$$

Since L lies also on π_L, its direction vector $\mathbf{V}$ is perpendicular to $\mathbf{n}$, the same equations as (3.3) - (3.6) are used. Until this point, we mentioned the core equations, which will let us build the proposed solvers that are presented in the following three sections.

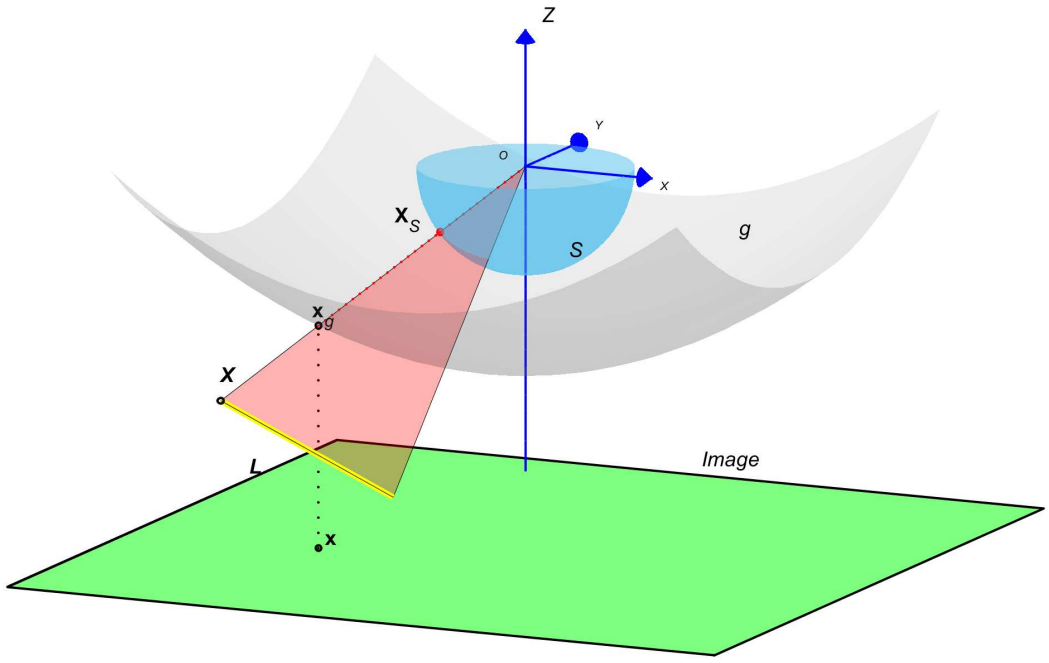


Figure 3.2. *Projection plane of a line (yellow) in the spherical camera model [2].*

3.3 Camera Pose Estimation with Known Vertical Direction

Many modern devices like cameras, smartphones and UAVs are equipped with a cheap and precise IMU. Such devices provide the vertical direction from which one can calculate 2 rotation angles, thus reducing the free parameters from 6 to 4. Hence, we formulate the solution based on the geometric observation (from Section 3.2) in a way that the vertical direction is known and provided as an input to the proposed solution. Starting from Section 2.4, we aim to compute $\mathbf{R}_X(\alpha)$ and $\mathbf{t}$, *i.e.* 4 unknowns: the rotation angle α and the translation components t_x, t_y, t_z of each camera using the equations (3.3) - (3.6). Although each 2D-3D line correspondence $L \leftrightarrow l$ provides 2 equations, only one contains $\mathbf{t}$. Therefore we need at least 3 line correspondences for each camera, *i.e.* in the minimal case, we need 6 2D–3D line pairs to solve for the absolute and relative pose of the stereo camera system. In order to eliminate $\cos(\alpha)$ and $\sin(\alpha)$ from $\mathbf{R}_X(\alpha)$, let us substitute $q = \tan(\alpha/2)$ [10, 65, 89], for which $\cos(\alpha) = (1-q^2)/(1+q^2)$ and $\sin(\alpha) = 2q/(1+q^2)$, yielding to a new form for $\mathbf{R}_X$

$$(1+q^2)\mathbf{R}_X(q) = \begin{bmatrix} 1+q^2 & 0 & 0 \\ 0 & 1-q^2 & -2q \\ 0 & 2q & 1-q^2 \end{bmatrix}. \tag{3.8}$$

3.3.1 Minimal Case

The minimal case of a multi-view camera system is a stereo camera pair, represented by their camera coordinate frames $\mathcal{C}_i, i = 0, 1$ with $\mathcal{C}_0$ being the reference camera for which we want to estimate the absolute pose $(\mathbf{R}, \mathbf{t}) : \mathcal{W} \rightarrow \mathcal{C}_0$ of the camera system, and the relative pose of $\mathcal{C}_1$ is denoted by $(\mathbf{R}_1, \mathbf{t}_1) : \mathcal{C}_0 \rightarrow \mathcal{C}_1$. Herein, we will derive a system of equations to estimate the absolute and relative poses of such a minimal camera system. As mentioned in Section 3.2 the 3D lines are represented as (2.15) and the corresponding 2D lines are presented as (3.1). Thus, the system of equation has equations which involves only the absolute pose (3.3) and (3.4), plus equations (3.5) and (3.6) which involve only one relative pose. Equation (3.5) involve only the rotations which means it will be used to solve the rotation, whereas (3.6) involve both rotation and translation. Substituting the new form of $\mathbf{R}$ (3.8) into (3.3), we get a quadratic equation in terms of q:

$$\mathbf{n}_0^\top \mathbf{R}_v \mathbf{R}_X(q) \mathbf{V} = q^2 A_1 + q B_1 + C_1 = 0 \tag{3.9}$$

where A_1, B_1, C_1 are coefficients in terms of $\mathbf{n}_0$, $\mathbf{R}_v$, and $\mathbf{V}$.

Regarding (3.5), we backsubstitute both $\mathbf{R} = \mathbf{R}_v \mathbf{R}_X(q)$ as well as $\mathbf{R}_1 = \mathbf{R}_{v,1} \mathbf{R}_X(q_1)$ yielding:

$$\begin{aligned} \mathbf{n}_1^\top \mathbf{R}_{v,1} \mathbf{R}_X(q_1) \mathbf{R}_v \mathbf{R}_X(q) \mathbf{V} = \\ q^2 q_1^2 A + q^2 q_1 B + q^2 C + q q_1 D + q E + q q_1^2 F + q_1^2 G + q_1 H + I = 0 \end{aligned} \tag{3.10}$$

where the coefficients A through I are expressed in terms of $\mathbf{n}_1$, $\mathbf{R}_v$, $\mathbf{R}_{v,1}$, and $\mathbf{V}$. Equations (3.9) and (3.10) provide a system of quadratic equations for the unknown absolute rotation q and relative rotation q_1. This system can be efficiently solved using a direct solver generated by [88], which gives 2 solutions for q and 2 for q_1. For each possible $(\mathbf{R}, \mathbf{R}_1)$ pair, the absolute and relative translations $\mathbf{t}$ and $\mathbf{t}_1$ are obtained by backsubstituting the rotation matrices into Equations (3.4) and (3.6) yielding a system of linear equations in terms of $\mathbf{t}$ and $\mathbf{t}_1$, which can be solved by an SVD decomposition.

3.3.2 Multi-view Case

When the 3D lines are viewed by a system of N calibrated perspective cameras, each 3D line L has up to N images $l_i, i = 1 \dots N$. One of the cameras (*e.g.* the left camera in a stereo

setup) is the reference camera $\mathcal{C}_0$ therefore the absolute pose of the camera system $(\mathbf{R}, \mathbf{t})$ is defined as the rigid transformation acting between $\mathcal{W}$ and $\mathcal{C}_0$, while individual camera frames $\mathcal{C}_i$ are related to the reference camera frame via the relative poses $(\mathbf{R}_i, \mathbf{t}_i) : \mathcal{C}_0 \rightarrow \mathcal{C}_i, i = 1, \ldots, N$. Of course, we assume that the vertical direction (*i.e.* $\mathbf{R}_{v,i}$) is available for each camera, hence the equations (3.3) - (3.6) remain valid for each camera pair $(\mathcal{C}_0, \mathcal{C}_i), i = 1, \ldots, N$. We will now formulate a least-squares solution for the multi-view case, based on the equations (3.9) and (3.10) by simply stacking (3.9) with all equations (3.10) from each camera $i = 1, \ldots, N$, the full system of equation will contain the absolute rotation q and the relative rotation q_i. A least-squares solution is obtained by finding a solution $(q, q_1, \ldots, q_N)$ which minimizes the squared error of the system. This would yield an 8^{th} order polynomial system of equation in terms of $(q, q_1, \ldots, q_N)$

$$\begin{aligned} (q^2 A_1 + q B_1 + C_1)^2 = 0 \\ \forall i = 1, \ldots, N: \quad (q^2 q_i^2 A + q^2 q_i B + q^2 C + q q_i D + q E + q q_i^2 F + q_i^2 G + q_i H + I)^2 = 0 \end{aligned} \tag{3.11}$$

Therefore, in order to reduce the order of the system, let us look for the vanishing point of the derivative, yielding a 7^{th} order polynomial system in the same variables. The first equation of (3.11) contains only q, hence its derivative is

$$A_3 q^3 + B_3 q^2 + C_3 q + D_3 = 0, \tag{3.12}$$

while the second equation of (3.11) contains both q and q_i yielding 2 equations, which are the partial derivatives with respect to q

$$\begin{aligned} A_4 q^3 q_i^4 + B_4 q^3 q_i^3 + C_4 q^3 q_i^2 + D_4 q^3 q_i + E_4 q^3 \\ + F_4 q^2 q_i^4 + G_4 q^2 q_i^3 + H_4 q^2 q_i^2 + I_4 q^2 q_i + J_4 q^2 \\ + K_4 q q_i^4 + L_4 q q_i^3 + M_4 q q_i^2 + N_4 q q_i + O_4 q \\ + P_4 q_i^4 + Q_4 q_i^3 + R_4 q_i^2 + S_4 q_i + T_4 = 0, \end{aligned} \tag{3.13}$$

and q_i:

$$\begin{aligned} A_5 q^4 q_i^3 + B_5 q^3 q_i^3 + C_5 q^2 q_i^3 + D_5 q q_i^3 + E_5 q_i^3 \\ + F_5 q^4 q_i^2 + G_5 q^3 q_i^2 + H_5 q^2 q_i^2 + I_5 q q_i^2 + J_5 q_i^2 \\ + K_5 q^4 q_i + L_5 q^3 q_i + M_5 q^2 q_i + N_5 q q_i \\ + O_5 q_i + P_5 q^4 + Q_5 q^3 + R_5 q^2 + S_5 q + T_5 = 0, \end{aligned} \tag{3.14}$$

where the coefficients A_k through T_k are expressed in terms of $\mathbf{n}_i$, $\mathbf{R}_v$, $\mathbf{R}_{v,i}$, and $\mathbf{V}$. Thus for a camera system with N cameras, we have (3.12) for the reference camera and two equations (3.13)–(3.14) for each additional camera, a total of $2 * (N-1) + 1$ equations. The initialization of the absolute and relative pose parameters is obtained through the pairwise direct solver discussed in the previous section.

This provides us with one initial relative pose for each camera and $N - 1$ possible value for the absolute pose, which are averaged to have one initial value for the absolute pose. Starting from this initialization, the system of the polynomial equations is efficiently solved in MATLAB via `fsolve`. The summary of the proposed algorithm is shown in Algorithm 1. which has 2 main steps:

1. Direct Solver using 6 2D-3D line pairs and one camera pair. In the experiments, we run the Direct Solver 50 times with a different randomly selected set of 6 2D-3D line pairs. Since the direct solver provides more than one possible solution, we first

check if an obtained pose is geometrically valid meaning that the cameras are in front of the visible face of the lines and the lines are not behind the camera. Out of the geometrically correct poses, we select the pose based on the line back-projection error.

2. Least-Squares Solver using all the available 2D-3D line pairs and all cameras. The initialization is important here, for which we use the pose obtained in the previous step. Then the polynomial system is solved first using `fsolve` in MATLAB, then the rotations are backsubstituted into the linear system of equation which is solved via SVD.

Algorithm 1 Summary of the proposed Algorithm for N cameras

Input: N: the number of the cameras ≥ 2
n: 2D-3D lines pairs ($\mathbf{n}_i$, **X**, and **V**)
Vertical direction rotation $\mathbf{R}_v$ for the reference camera as well as $\mathbf{R}_{v,i}$ for the other cameras

Output: The absolute pose $(\mathbf{R}, \mathbf{t}) : \mathcal{W} \rightarrow \mathcal{C}_0$ and the relative poses $(\mathbf{R}_i, \mathbf{t}_i) : \mathcal{C}_0 \rightarrow \mathcal{C}_i$

1: one camera from the N cameras is set as the reference camera, the remaining $N-1$ cameras are considered as relative to the reference one.
2: Calculation of the initial rotation and translation for each camera pair with the direct solver using 6 2D-3D line pairs as discussed in Section 3.3.1.
3: In a RANSAC-like environment, the direct solver is executed ρ times ($\rho = 50$ was in our experiments)
4: The best solution is then selected from each $(N-1)$ camera pairs based on *e.g.* the line back-projection error
5: Estimation of the rotation and the translation for the N cameras with the least squares solver presented in Section 3.3.2 in 2 steps:
6: Rotation: Calculation of the coefficients and estimation of the rotation using all the inlier line pairs by solving the polynomial system of equations (3.12)–(3.14) with `fsolve` in MATLAB, initialized by the best solution selected in Step. 4 (regarding the reference camera we took the average rotation from the $N-1$ camera pairs).
7: Translation: By back-substituting the absolute and relative rotations, solve the (overdetermined) linear system of equations (3.4) and (3.6) via SVD
8: build the final absolute and relative poses obtained in Step. 6 and Step. 7

3.3.3 Synthetic Experiments

We conducted our experiments on synthetic and real data. For the synthetic experiments, we generated a benchmark dataset containing 1000 sets of 60 2D-3D line pairs seen by 5 cameras with:

- Same intrinsic parameters Focal length equal to 800
- The 3D lines were arranged on 3 planes placed in the 3D world space. With respect to one of these planes, an additional random transformation was applied on the other two planes with
  - Plane1 X axis: rotation with 10 to 20 degrees
  - Plane1 Y axis: translation -1 to -2 units
  - Plane1 Z axis: translation 2 to 3 units
  - Plane2 X axis: rotation 10 to 20 degrees
  - Plane2 Y axis: translation between -1 to -2 units
  - Plane2 Z axis: translation 2 to 3 units

2D Noise	Pixel Shift(px)		
	Variance	Mean	Median
3%	22	11	11
5%	65	18	18

Table 3.1. *2D noise interpretation in term of pixel shift(px) [4].*

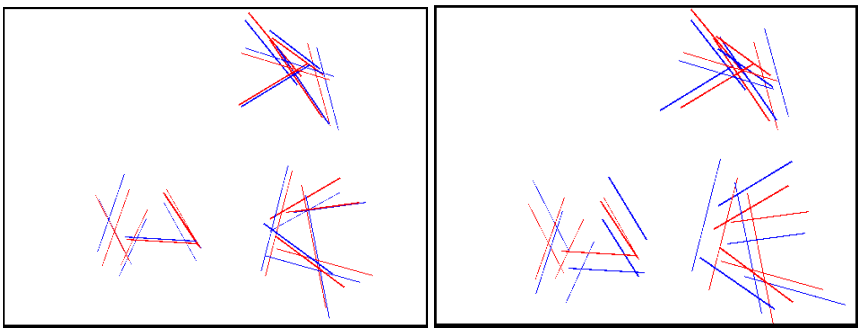


Figure 3.3. *The effect of the 2D noise: the red lines are the correct ones and the blue lines are the noisy ones, 3% noise on the left image and 5% noise on the right image [4].*

This transformation aims to produce 3D lines similar to those typically found in a usual urban environment. The 2D image of the lines were generated using the camera projection $P = K[R|t]$. generated randomly for 5 cameras with 1024×768 pixels, each camera has been rotated randomly between -10 to 10 degrees around the 3 axis, and random translation of 4 to 5 units along Z for the absolute camera and between -0.5 to 0.5 units for the 4 relative cameras over the 3 axis. All the experiment were done on an $i7$ computer with 3.40GHz CPU and 16 GB of RAM.

The synthetic experiments involve both perfect (noiseless) cases as well as noisy ones where random Gaussian noise is added to the line parameters in order to evaluate the sensitivity of our algorithm. We have generated noisy datasets with 3% and 5% random noise added to one endpoint of the line as well as the unit direction vector, both in 3D and 2D, on 2D we added 3% and 5% noise which is a quite high level of noise!. Table 3.1 and Fig. 3.3 shows the effect of the 2D noise.

- The 2D 3% noise corresponds to $[-31, +31]$ pixel error
- The 2D 5% noise corresponds to $[-54, +54]$ pixel error

Furthermore, we have also added a random noise to the vertical direction of $\pm 0.2^\circ$ along both axes, which is equivalent to an overall vertical direction error up to 0.27°.

In the first experiment using the noiseless dataset, we wanted to see how our solvers perform and to demonstrate, that the direct solver provides sufficiently precise initialization for the least-squares solver. For more than 2 cameras, we solve the pose by camera pairs

proposed	Execution Time (s)
Direct Solver	0.017
Least Squares solver	4.01
Full Algorithm	4.24

Table 3.2. *Execution time for the proposed algorithms implemented in MATLAB using 5 cameras with 60 lines [4].*

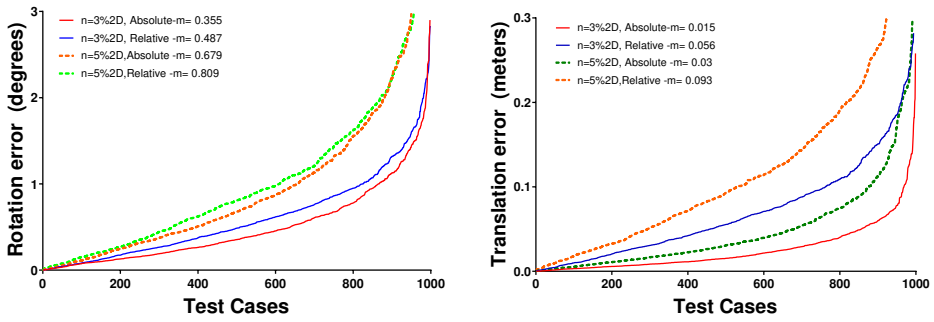


Figure 3.4. *2D and 3D noise influence on the proposed algorithm for 2 cameras when there is up to 3% and 5% noise. Left plot shows the rotation error in degrees, the right plot shows the translation error in meters, n is the level of noise. Absolute is referring to the absolute pose error and Relative to the relative pose error, respectively. m indicates the median error [4].*

(which always involves the reference camera!) and then we initialize the least-squares solver based on these solutions for whatever the number of cameras is. In Table 3.3, we can see that the error is up to 1e-15. It is thus clear that the proposed algorithm performs perfectly when there is no noise.

Error	Absolute pose	Relative pose 1
Rotation(degree)	1.43e-14	2.23e-14
Translation(meter)	2.84e-15	2.97e-15

Table 3.3. *Median error with the perfect synthetic data using 2 cameras [4].*

Of course, in a real application we cannot expect perfect data. Therefore we quantitatively evaluated our algorithm using datasets with various noise levels as well as with noisy vertical direction. First, we tested the robustness of the proposed algorithm by including only 2D and 3D noise up to 3% and 5%. In Fig. 3.4, the experiment shows that the algorithm is robust up to 5% noise. Half of the cases have an error lower than $0.85°$, and lower than $9.5cm$ in translation. We can also see that the absolute pose has much lower error – this is not surprising as we have put more constraints on it in our system of equations.

For an even more realistic configuration, we tested the proposed solution when we have noise on the vertical direction. IMU sensors have a typical error of $0.2°$, hence our dataset simulated this noise. In this benchmark, we compared this baseline with 3% 2D noise with vertical direction noise and 5% 2D noise with vertical direction noise. Clearly, the proposed solution is robust to up to 3% 2D noise with $0.2°$ noise on the vertical direction, and up to 5% 2D noise with $0.2°$ noise on the vertical direction. Fig. 3.5 and Fig. 3.6 show the results obtain over this test with 2 cameras. We run also this experiment on 5 cameras shown in Fig. 3.7, these results are quite similar to the 2 camera case. Note also, that for this level of noise, the absolute pose error is always lower than the max-min range of the relative cameras. The execution time is also promising for future work for a real time application, the proposed algorithm estimates poses for 5 cameras with 60 lines in less then 5 seconds (see table Table 3.2)

3.3.4 Real Data Experiments

Concerning the real data, Fig. 3.8 shows the 5 images extracted from a $4K$ resolution drone video corresponding to 5 camera poses. The middle camera was set as the reference and

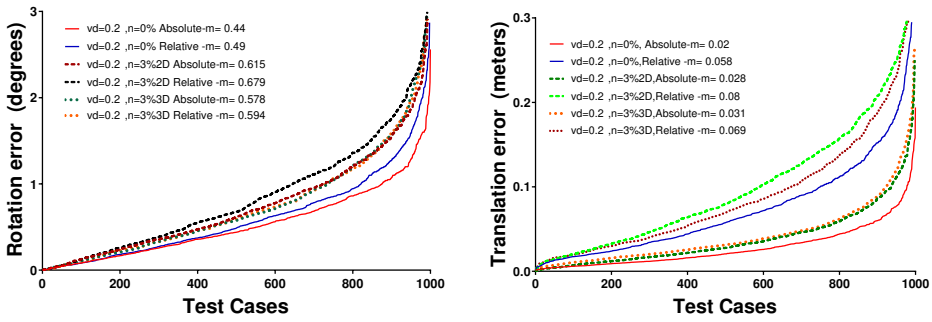


Figure 3.5. *Comparison of the outcome of different 2D and 3D error types with 2 cameras when we have up to 3% noise and $0.2°$ vertical direction error. The left plot is showing the rotation error, the right one is for the translation error. vd means the vertical direction noise, n is the level of noise, Absolute is referring to the absolute pose error and Relative to the relative pose error. m indicates the median error [4].*

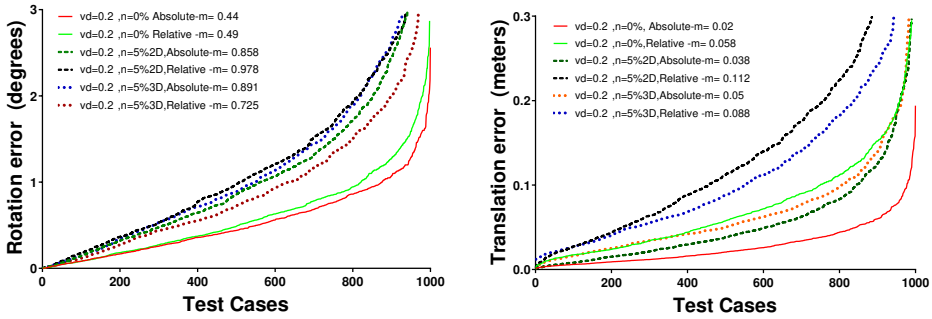


Figure 3.6. *Comparison of the outcome of different 2D and 3D error types with 2 cameras when we have up to 5% noise and $0.2°$ vertical direction error. The left plot is showing the rotation error, the right one is for the translation error. vd means the vertical direction noise, n is the level of noise, Absolute is referring to the absolute pose error and Relative to the relative pose error. m indicates the median error [4].*

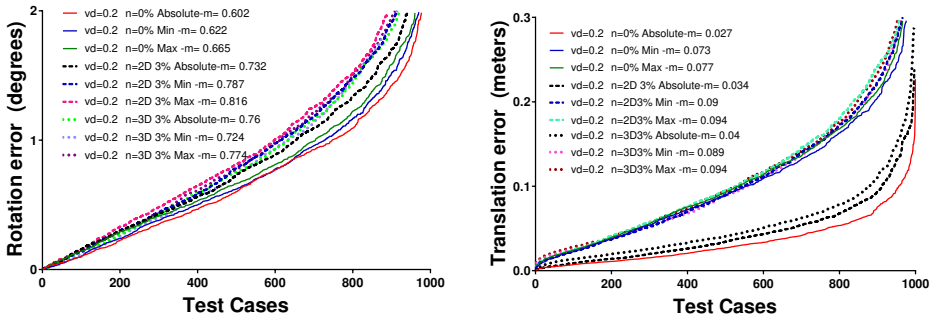


Figure 3.7. *Comparison of the outcome of different 2D and 3D error types with 5 cameras when we have up to 3% noise and $0.2°$ vertical direction error. The left plot is showing the rotation error, the right one is for the translation error.vd means the vertical direction noise, n is the level of noise, Absolute is referring to the absolute pose error and Relative min and max refer to the smallest and highest noise over the relative pose errors, respectively. m indicates the median error [4].*

the 4 others as relative. Here, the 3D data is a dense point cloud generated by Riegl VZ400 Lidar scanner. We have extracted 30 2D-3D line pairs. For the ground truth pose, we placed

Figure 3.8. *The extracted frames and the lines used for the real data experiment. The upper right frame is the reference frame, while the subsequent frames represent the images captures by the relative cameras from (absolute camera: top right, relative camera 1: middle right, relative camera 2: bottom right, relative camera 3: bottom middle, relative camera 4: bottom left). The bigger image shows the 3D point cloud including the used lines (colored lines), the ground truth camera poses (green) and the estimated ones (red). The middle camera (green with blue edges) was set as the reference camera, the relative camera position was set from right to left [4].*

special Lidar markers on the facade of the buildings which was captured with an angular resolution of $0.05°$. The 3D location of these markers are read by the Lidar scanner, while the markers on the 2D images were manually selected. Based on the 2D-3D marker point correspondences, the absolute pose of each camera was estimated using the point based UPnP method [83]. On the other hand, these marker points let us to calculate the metric error in 3D space while we computed the forward projection of the 2D marker points onto the 3D surfaces. From each image, we extracted lines using OpenCV [24] EDLines detector and LBD descriptors to select the appropriate line matches automatically. In our camera system we used 5 perspective cameras and between each image pairs, we generated line matches based on the LBD descriptors of the lines by selecting the best matches for each of them. After we finished this process for each image pairs we only used those detected lines which had corresponding pairs in the other 4 images. The result is shown in Table 3.4 and Fig. 3.8. The proposed algorithm performs well on the real-data. The estimated poses are shown in the point cloud images with red color (the reference camera is marked with blue edges). 2 poses out of 5 have a rotation error lower than $0.07°$ and on average it is lower than $0.15°$ which is quite promising. Furthermore, the translation error is lower than $0.15m$ in average. Additionally, the forward projection error of the markers with the estimated poses show in Table 3.5 that the median distance from the ground truth markers is less than 5 cm for 4 out of 5 cameras.

3.3.5 Conclusion

In this section a novel solution to estimate the absolute and relative pose of a multi-view camera system from line correspondences with known vertical direction was proposed. the

Error	Absolute	R1	R2	R3	R4
Rotation(degree)	0.062	0.287	0.095	0.034	0.336
Translation(meter)	0.218	0.105	0.042	0.025	0.237

Table 3.4. *Efficiency of the proposed algorithm on the real data using 5 cameras. The table shows the median error, absolute column is referring to the absolute pose, the enumerated R refer to the relative poses [4].*

Error(meter)	Absolute	R1	R2	R3	R4
min	0.014	0.0009	0.01	0.009	0.029
max	0.021	0.073	0.103	0.17	0.34
mean	0.06	0.033	0.34	0.068	0.14
median	0.035	0.027	0.027	0.049	0.127

Table 3.5. *Forward projection error for the real data experiment. Each column represent the error for each camera. Absolute refers to the absolute pose, R1 through R4 refers to the relative poses [4].*

solution consists of a minimal solver (2 cameras and 6 line-pairs) which can be used efficiently within a hypothesis-testing framework like RANSAC [40]. Using such solutions, one can solve for a multi-view system with an arbitrary number of cameras based on a least squares solution for which the initialization is provided by the direct solutions. The proposed method has been evaluated on synthetic and real datasets, these tests confirm the robustness and near real-time computational complexity of the proposed approach.

3.4 Pose Estimation for A Perspective Camera System

In Section 3.3 we showed a solution wich exploit IMU sensors. Thus, here we deal with the case that such sensor or provided information is not accessible, and we use the full rotation matrix. The absolute pose $(\mathbf{R}, \mathbf{t})$ has 6 degrees of freedom: 3 rotation angles (α, β, γ) and 3 translation components (t_x, t_y, t_z) along the 3 coordinate axes. Thus to solve for the pose using (3.3) and (3.4), we need a minimum of 3 line correspondences $\{(L_i, l_i)\}_{i=1}^{3}$. The solution is obtained in two steps: first the rotation $\mathbf{R}$ is solved using (3.3), which in general involves solving a system of 8-th order polynomials. Then translation is obtained from (3.4) by backsubstituting $\mathbf{R}$, which yields a linear system of equations in terms of $\mathbf{t}$ [98, 178, 188, 203]. Clearly, the main challange is the solution for $\mathbf{R}$ due to the nonlinearity of the equations as well as the additional constraints on $\mathbf{R}$ to be a valid rotation (*i.e.* orthonormal) matrix. Although for special line configurations (*e.g.* orthogonal, parallel or intersecting lines) [188] or with additional knowledge of *e.g.* the vertical direction [4, 65], a lower order polynomial may be achieved, usually the P3L polynomial will not be lower than 8 for general line configurations [33].

Let us have a closer look at the parametrization of rotations as the final form of (3.3) depends on this. It is well known, that the rotation group $SO(3)$ has 3 degrees of freedom. The most popular parametrization is *Euler angles*, which represents $\mathbf{R}$ via 3 rotation angles around the coordinate axes. Unfortunately, this representation involves trigonometric functions which would yield trigonometric equations in (3.3). One approach is to letting these trigonometric functions of one angle α to be two separate unknowns [65, 107, 188, 203], which –together with the trigonometric constraints– lead to polynomial equations. Alternatively, one can solve directly for the 9 elements of $\mathbf{R}$ in (3.3) –as a linear system–

and then enforce orthonormality on the solution yielding again to (different) polynomial equations [178, 188]. Herein, in order to eliminate trigonometric functions, we substitute $s = \tan(\alpha/2)$ as in [10, 65, 89], for which $\cos(\alpha) = (1-s^2)/(1+s^2)$ and $\sin(\alpha) = 2s/(1+s^2)$, yielding a second order polynomial in s for one rotation around a coordinate axis.

3.4.1 Direct Least Squares Solver

In order to reduce the number of unknowns to 2 in (3.3), we eliminate one rotation in $\mathbf{R}$ by defining an intermediate coordinate system $\mathbb{N}$ [178, 188, 203] in which the rotation angle around the X axis can be easily obtained. First, let us select a line pair (L_0, l_0) with the longest projection length as longer edges are less affected by noise on their endpoints [178, 188, 203]. Both L_0 and l_0 lie on a projection plane π_0 passing through the camera projection center $\mathbf{C}_0$. The unit normal to the plane π_0 in the camera coordinate system $\mathcal{C}_0$ is denoted by $\mathbf{n}_0$. The origin of $\mathbb{N}$ is located at the origin of $\mathcal{W}$ and its axes $(\mathbf{X}_\mathcal{M}, \mathbf{Y}_\mathcal{M}, \mathbf{Z}_\mathcal{M})$ are

$$\mathbf{Y}_\mathcal{M} = \frac{\mathbf{n}_0^\mathcal{C}}{\|\mathbf{n}_0^\mathcal{C}\|} \tag{3.15}$$

$$\mathbf{X}_\mathcal{M} = \frac{\mathbf{n}_0^\mathcal{C} \times \mathbf{V}_0^\mathcal{W}}{\|\mathbf{n}_0^\mathcal{C} \times \mathbf{V}_0^\mathcal{W}\|} \tag{3.16}$$

$$\mathbf{Z}_\mathcal{M} = \frac{\mathbf{X}_\mathcal{M} \times \mathbf{Y}_\mathcal{M}}{\|\mathbf{X}_\mathcal{M} \times \mathbf{Y}_\mathcal{M}\|} \tag{3.17}$$

where the Y axis of $\mathbb{N}$ aligns with $\mathbf{n}_0^\mathcal{C}$, and $\mathbf{V}^\mathcal{W}$ denotes the unit direction vector of L_0 in the world coordinate frame.

The rotation $\mathbf{R}_\mathcal{M} = [\mathbf{X}_\mathcal{M}, \mathbf{Y}_\mathcal{M}, \mathbf{Z}_\mathcal{M}]^\top$ rotates the normals and direction vectors into the intermediate frame $\mathbb{N}$. The rotation $\mathbf{R}_x^\mathcal{M}$ around X axis within $\mathbb{N}$ is then easily calculated because it is the angle between the Z axis and $\mathbf{V}_0^\mathcal{M}$, hence the rotation matrix acting within the intermediate coordinate frame $\mathbb{N}$ is composed of the rotations around the remaining two axes as:

$$(1+s^2)(1+r^2)\mathbf{R}^\mathcal{M} = \mathbf{R}_y^\mathcal{M}(s)\mathbf{R}_z^\mathcal{M}(r) = \begin{bmatrix} (1-s^2)(1-r^2) & -2r(1-s^2) & 2s(r^2+1) \\ 2r(s^2+1) & (s^2+1)(1-r^2) & 0 \\ -2s(1-r^2) & 4sr & (1-s^2)(r^2+1) \end{bmatrix} \tag{3.18}$$

and we obtain the new form of (3.3) as:

$$(\mathbf{R}_\mathcal{M}\mathbf{n})^\top \mathbf{R}^\mathcal{M}(\mathbf{R}_x^\mathcal{M}\mathbf{R}_\mathcal{M}\mathbf{V}) = \mathbf{n}^{\mathcal{M}\top}\mathbf{R}^\mathcal{M}\mathbf{V}^\mathcal{M} = 0. \tag{3.19}$$

Expanding the above equation gives a 4-th order polynomial of (s, r) with coefficients in terms of $\mathbf{V}^\mathbb{N}$ and $\mathbf{n}^\mathbb{N}$:

$$\mathbf{a}^\top\mathbf{u} = \begin{bmatrix} V_1\,n_1 + V_2\,n_2 + V_3\,n_3 \\ 2\,V_1\,n_2 - 2\,V_2\,n_1 \\ 2\,V_3\,n_1 - 2\,V_1\,n_3 \\ -V_1\,n_1 - V_2\,n_2 + V_3\,n_3 \\ -V_1\,n_1 + V_2\,n_2 - V_3\,n_3 \\ V_1\,n_1 - V_2\,n_2 - V_3\,n_3 \\ 2\,V_1\,n_2 + 2\,V_2\,n_1 \\ 2\,V_3\,n_1 + 2\,V_1\,n_3 \\ 4\,V_2\,n_3 \end{bmatrix}^\top \cdot \begin{bmatrix} 1 \\ r \\ s \\ r^2 \\ s^2 \\ s^2r^2 \\ s^2r \\ sr^2 \\ sr \end{bmatrix} = 0 \tag{3.20}$$

where $\mathbf{V}^\mathbb{N} = [V_1, V_2, V_3]^\top$ and $\mathbf{n}^\mathbb{N} = [n_1, n_2, n_3]^\top$ are the 3D line unit direction vector and the projection plane unit normal, respectively, defined in the intermediate coordinate system

$\mathbb{N}$. Each line pair generates one such equation, yielding a system of M equations, which is solved in the least squares sense. For this purpose, let's take the sum of squares of the above system

$$e(s,r) = \sum_{i=1}^{M} (\mathbf{a}_i^\top \mathbf{u})^2 \tag{3.21}$$

and then find $\arg\min_{(s,r)} e(s,r)$. The first order optimality condition for (3.21) is

$$\nabla e(s,r) = \begin{bmatrix} \frac{\partial e(s,r)}{\partial s} \\ \frac{\partial e(s,r)}{\partial r} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{M} \mathbf{b_s}_i^\top \mathbf{u_s} \\ \sum_{i=1}^{M} \mathbf{b_r}_i^\top \mathbf{u_r} \end{bmatrix} = \mathbf{0} \tag{3.22}$$

where for each line pair

$$\mathbf{b_r}^\top \mathbf{u_r} = \begin{bmatrix} 2\,a_1a_4 \\ 6\,a_2a_4 \\ 2\,a_1a_7 + 2\,a_3a_4 + 2\,a_5a_9 \\ 4\,a_1a_2 + 2\,{a_4}^2 \\ 2\,a_1a_9 + 2\,a_4a_5 \\ 6\,a_2a_7 + 6\,a_4a_6 + 6\,a_8a_9 \\ 4\,a_1a_6 + 4\,a_2a_3 + 4\,a_4a_7 + 4\,a_5a_8 + 2\,{a_9}^2 \\ 6\,a_2a_9 + 6\,a_4a_8 \\ 4\,{a_2}^2 \\ 2\,a_3a_7 \\ 2\,a_3a_9 + 2\,a_5a_7 \\ 4\,a_1a_8 + 4\,a_2a_5 + 4\,a_4a_9 \\ 4\,{a_6}^2 \\ 8\,a_6a_8 \\ 8\,a_2a_6 + 4\,{a_8}^2 \\ 8\,a_2a_8 \\ 6\,a_6a_7 \\ 6\,a_6a_9 + 6\,a_7a_8 \\ 4\,a_3a_6 + 2\,{a_7}^2 \\ 4\,a_3a_8 + 4\,a_5a_6 + 4\,a_7a_9 \end{bmatrix}^\top \begin{bmatrix} 1 \\ r^2 \\ s^2 \\ r \\ s \\ s^2r^2 \\ s^2r \\ sr^2 \\ r^3 \\ s^4 \\ s^3 \\ sr \\ r^3s^4 \\ r^3s^3 \\ r^3s^2 \\ r^3s \\ r^2s^4 \\ r^2s^3 \\ rs^4 \\ rs^3 \end{bmatrix} \tag{3.23}$$

and similarly $\mathbf{b_s}^\top \mathbf{u_s}$ can also be expressed in terms of the coefficients $\mathbf{a}$ of each line pair. Thus the solution of the system of 2 polynomial equations (each of them is 7-th order) in (3.22) provides the rotation parameters (s,r). Herein, we use the automatic generator of Kukelova *et al.* [88] to generate a solver using Grobner basis[88, 94] for the system in (3.22). Once the solution(s) are obtained, the complete $\mathbf{R}$, acting between the world $\mathcal{W}$ and camera $\mathcal{C}$ frame, is obtained as $\mathbf{R} = \mathbf{R}_{\mathcal{M}}^\top (\mathbf{R}^{\mathcal{M}} \mathbf{R}_x^{\mathcal{M}}) \mathbf{R}_{\mathbb{N}}$.

The translation $\mathbf{t}$ is then obtained by backsubstituting $\mathbf{R}$ into (3.4) yielding a system of linear equations, which can be solved by SVD decomposition. While in (3.3), all quantities are already normalized ($\mathbf{n}$ and $\mathbf{V}$ are of unit length), (3.4) contains the 3D point $\mathbf{X}$ given in an arbitrary world coordinate system $\mathcal{W}$, which needs to be normalized for numerical stability [58]. Therefore, we transform our 3D line segments into a unit cube around the centroid of the 3D scene, by a normalizing transformation $\mathbf{N_3}$ as detailed in Section 2.6

Although (3.22) might have several solutions, the solver will only return the real ones and then one has to select the geometrically valid $(\mathbf{R}, \mathbf{t})$ based on the visibility of the lines and the line back-projection error (3.25) [4, 65, 98, 178, 188].

3.4.2 Multi-view Case and Pose Refinement

Equations (3.19) and (3.4) remain valid for $\mathcal{C}_0$, while for the other cameras $\mathcal{C}_i$, the projection of L yields similar equations but the unknown relative pose $(\mathbf{R}_i, \mathbf{t}_i)$ will also be involved,

and from (3.5) we get:

$$(\mathbf{R}_{\mathcal{M}_i}\mathbf{n}_i)^\top \mathbf{R}^{\mathcal{M}_i}(\mathbf{R}_x^{\mathcal{M}_i}\mathbf{R}_{\mathcal{M}_i}\mathbf{R}\mathbf{V}) = \mathbf{n}^{\mathcal{M}_i\top}\mathbf{R}^{\mathcal{M}_i}\mathbf{V}^{\mathcal{M}_i} = 0 \tag{3.24}$$

which –after a similar derivation as in the single camera case– yields also a system of polynomial equations of the same form as in (3.22), hence the same solver can be used to solve for each camera $\mathcal{C}_i, i = 1, \ldots, N-1$. Once the solutions are obtained, each $\mathbf{R}_i$ is backsubstituted into the corresponding linear system similar to (3.6), which is solved for $\mathbf{t}_i$ by SVD.

Robust Outlier Filtering

In real applications, putative line correspondences are extracted that are corrupted by outliers as well as noise. While noise is handled well by our least squares formulation of the equations, outliers must be removed via RANSAC [40] or the M-estimator sample consensus (MSAC) algorithm [166], which relies on a minimal solver and a back-projection error. Note that the minimal set consists of 3 line-pairs. Herein, we used the error measure proposed in [98] that practically calculates the mean of the shortest distances $d_{\mathbf{x}_s}$ and $d_{\mathbf{x}_e}$ from the 2D line segment endpoints $\mathbf{x}_s$ and $\mathbf{x}_e$ to the corresponding infinite line determined by the backprojected 3D line onto the normalized plane:

$$\frac{d_{\mathbf{x}_s} + d_{\mathbf{x}_e}}{2(\|\mathbf{x}_e - \mathbf{x}_s\|)} \tag{3.25}$$

Note, that the error is normalized with the length of the 2D line segment, hence making the measure independent of the length of the detected 2D line segment.

Pose Refinement

We will now formulate a least-squares refinement for the multi-view case, based on equations (3.3)-(3.6) by simply stacking for each line pair in $\mathcal{C}_0$ (3.3), (3.4) and for each camera $i = 1, \ldots, N-1$ and each line pair in $\mathcal{C}_i$ (3.5) and (3.6) containing the absolute pose $(\mathbf{R}, \mathbf{t})$ and the relative poses $(\mathbf{R}_i, \mathbf{t}_i)$:

$$\left.\begin{aligned} \forall j = 1, \ldots, n_{\mathcal{C}_0} : \\ \mathbf{n}_j^{\mathcal{C}_0\top}\mathbf{R}\mathbf{V}_j = 0 \\ \mathbf{n}_j^{\mathcal{C}_0\top}(\mathbf{R}\mathbf{X}_j + \mathbf{t}) = 0 \\ \forall i = 1, \ldots, N-1; \forall j = 1, \ldots, n_{\mathcal{C}_i} : \\ \mathbf{n}_j^{\mathcal{C}_i\top}\mathbf{R}_i\mathbf{R}\mathbf{V}_j = 0 \\ \mathbf{n}_j^{\mathcal{C}_i\top}(\mathbf{R}_i(\mathbf{R}\mathbf{X}_j + \mathbf{t}) + \mathbf{t}_i) = 0 \end{aligned}\right\} \tag{3.26}$$

A least-squares solution is then obtained by minimizing the squared error of the system, which can be solved via standard algorithms like *Levenberg-Marquardt* [120] with the initialization obtained from the direct solver. Note, that this step is optional, and only executed for the overdetermined $n > 3$ case if the line parameters are noisy. Experiments show, that the initialization given by the direct solver is sufficiently stable, hence only a few iterations are needed to reach the optimum. The proposed algorithm, that we call *Multi-view RPnL* (MRPnL) is summarized in Algorithm 2, Note that the proposed algorithm has 2 main steps:

1. MRPnL direct solver using all 2D-3D line pairs for each camera. Since the direct solver may provide more than one possible solution, we first check if an obtained pose is geometrically valid meaning that the cameras are in front of the visible face of the lines and the lines are not behind the camera. Out of the geometrically correct poses, we select the best pose based on the line back-projection error.
2. Pose refinement using all the available 2D-3D line pairs and all cameras simultaneously. The initialization is important here, for which we use the pose obtained in the previous step. Then the system of (3.26) is solved using `fsolve` in MATLAB.

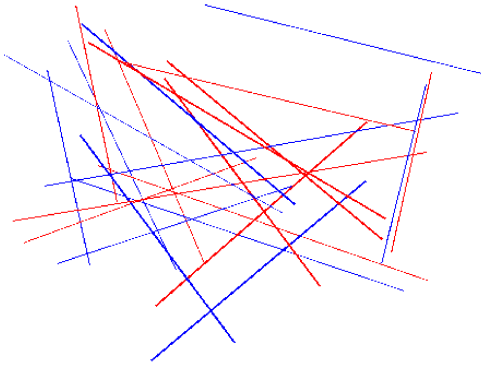


Figure 3.9. *Illustration of* 10% *2D noise on* 10 *random lines placed on* 1 *plane. Red is original, blue one is the noisy [1].*

Algorithm 2 The proposed MRPnL algorithm.

Input: N: the number of the cameras ≥ 1 and the reference camera $\mathcal{C}_0$
$n_{\mathcal{C}_i}$ 2D-3D lines pairs ($\mathbf{n}_j^{\mathcal{C}_i}$, $\mathbf{X}_j$, and $\mathbf{V}_j$) for each camera $\mathcal{C}_i$
Output: The absolute pose $(\mathbf{R}, \mathbf{t}) : \mathcal{W} \rightarrow \mathcal{C}_0$ and the relative poses $(\mathbf{R}_i, \mathbf{t}_i) : \mathcal{C}_0 \rightarrow \mathcal{C}_i$

1: Normalize the 3D line points $\mathbf{X}_j$ by $\mathbf{N}_3$.
2: Rotation: Determine the intermediate rotation $\mathbf{R}_{\mathbb{N}}$ and apply it to the input line pairs. Then $\mathbf{R}_x^{\mathcal{M}}$ is calculated and the two remaining rotation $\mathbf{R}^{\mathbb{N}}$ is obtained by solving the polynomial system of equations (3.22), which together provides $\mathbf{R}$.
3: Translation: By back-substituting the rotation, solve the linear system of equations (3.4) for $\mathcal{C}_0$ or (3.6) for the other cameras via SVD.
4: Optional refinement of the absolute and relative poses for all cameras and lines simultaneously by solving the system (3.26) in the lest squares sense (see Section 3.4.2).
5: Denormalize to get the final pose estimates.

3.4.3 Comparison with State-of-the-Art

For the quantitative evaluation of the proposed method, synthetic datasets were generated using the calibration parameters of a real perspective camera, with available physical parameters (sensor size), enabling us to represent our 3D scene in an equivalent metric space. Multiple sets of 1000 samples were generated containing 2D-3D line pairs. The 3D scene was created with a typical urban or indoor environment in mind, where only few planar surfaces are usually visible in a camera at once, thus we created 3 planes, placing them randomly in the 3D space, each containing 20 random line segments, with a minimum length of 0.5 m

- X axis : Rotation= $\pm 30^\circ$ around all 3, $\pm[1-2]$ meters translation
- Y axis : Rotation= $\pm 30^\circ$ around all 3, $\pm[1-2]$ meters translation
- Z axis : Rotation= $\pm 30^\circ$ around all 3, $\pm[0.5-1.5]$ meters translation

The 2D data was then generated by capturing images of the scene, practically projecting the 3D lines with a virtual camera using the parameters of a standard commercial camera with APS-C size sensor, 2378x1580 pixel resolution and 16 mm focal length. In each scene we placed 5 cameras with a random rotation of $\pm 50^\circ$ around all 3 axes, and random translation of ± 1 m in the horizontal and vertical direction, while in the optical axis' direction at $[4-6]$ m from the scene.

The estimated pose was evaluated in terms of the angular distance ϵ, that represents the overall rotation error, and also in terms of translation error as the norm of the difference between the ground truth and estimated translation (see Section 2.5). To evaluate the

robustness of the methods against noisy line detections, random noise was added to the line parameters. Practically we simulated noise by corrupting one endpoint of the line (similarly in 2D and 3D) as adding a random number to each coordinate of the point up to the specified percentage of the actual coordinate value. The unit direction vector was also modified in the same manner. We show results for 3%, 10% and 15% 2D and 3D noise levels, that translate to an average shift on the image of 22px with 3% noise up to 110px with 15% noise (example of 10% noise can be seen in Fig. 3.9).

First, the performance of the proposed MRPnL direct solver and MRPnL-LM (LM refers to the refinement step in Section 3.4.2) is evaluated for single camera case. Wang *et al.* published a recent comparison [178] of 6 State-of-the-Art methods with their SRPnL method, which proved to be dominating both in terms of CPU time as well as efficiency and robustness. Therefore herein, we only focus on comparing the proposed MRPnL algorithm with the most competitive methods from [178]: SRPnL and AlgLS [118].

The MATLAB implementation of the competing methods are available from [178]. For a fair comparison, we used the automatic generator of Kukelova *et al.* [88], that provides a MATLAB-based solver, but we remark that we also successfully used Kneip's generator [80], which produces a solver in C++ that is much faster. All experiments were done on an $i7$ computer with 16 GB of RAM.

Comparisons were performed in two different setups, first using the minimum number of line matches that each algorithm requires, then using all 60 line pairs of the scene, with only a single camera, since the formulation of SRPnL and AlgLS doesn't cover multi-view setups.

For the first setup, using $n = 4$ lines for AlgLS and $n = 3$ for the other methods, the obtained results are shown in Fig. 3.10. Results on plots are always sorted based on the error from best to worst. In this setup all the methods perform very similar in terms of median errors of the pose parameters, only AlgLS produces lower errors due to the higher number of line-pairs it is using ($n = 4$). The methods are robust for up to 3% 2D noise, where the median ϵ already reaches above 2°, only exception is AlgLS with 1.52°. In terms of runtime, MRPnL is the fastest with 2.4ms, then SRPnL follows with 3.5ms as the highest median runtime among multiple data sets. MRPnL LM takes more time (8.3ms), but is still much faster than AlgLS (53ms).

Using $n = 60$ line-matches, AlgLS and MRPnL LM have the best results with the lowest median rotation and translation errors, with 15% noise MRPnL LM handles better the 3D noise, while AlgLS favors the noise in 2D domain, both of them having the median ϵ below 1.5° with 2D noise and 1.05° with 3D noise (see Fig. 3.11 for results). The only other method that handles well 15% noise both in 2D and 3D domain is MRPnL (median ϵ 2.5° and 2.17° respectively, and translation error of 0.21m and 0.1m), while SRPnL can only obtain similar errors with up to 5% noise. Since MRPnL robustly provides a good initialization, LM refinement usually performs only 5 iterations, keeping the runtime comparable with SRPnL and MRPnL.

Based on the data presented in [178], we can confirm that the CPU time of the methods does not change significantly for $n < 200$ lines. Since in a realistic dataset of an urban environment we shouldn't expect to have such many inlier pairs (*e.g.* the large scale dataset presented by [116] also uses an average of 130 lines per image), we did not find an evaluation with hundreds of lines relevant.

3.4.4 Multi-view Case

The multi-view configuration presented in Section 3.4.2 with LM refinement, was tested on data with 10% 2D noise using 5 cameras. Results are shown in Fig. 3.12, where –as a baseline– we also show the results achieved with a single camera. Clearly, the accuracy of pose estimates are consistent over all cameras. As for the CPU time, one can see that it scales with the number of cameras but still remains under 56 ms for 5 cameras, which is

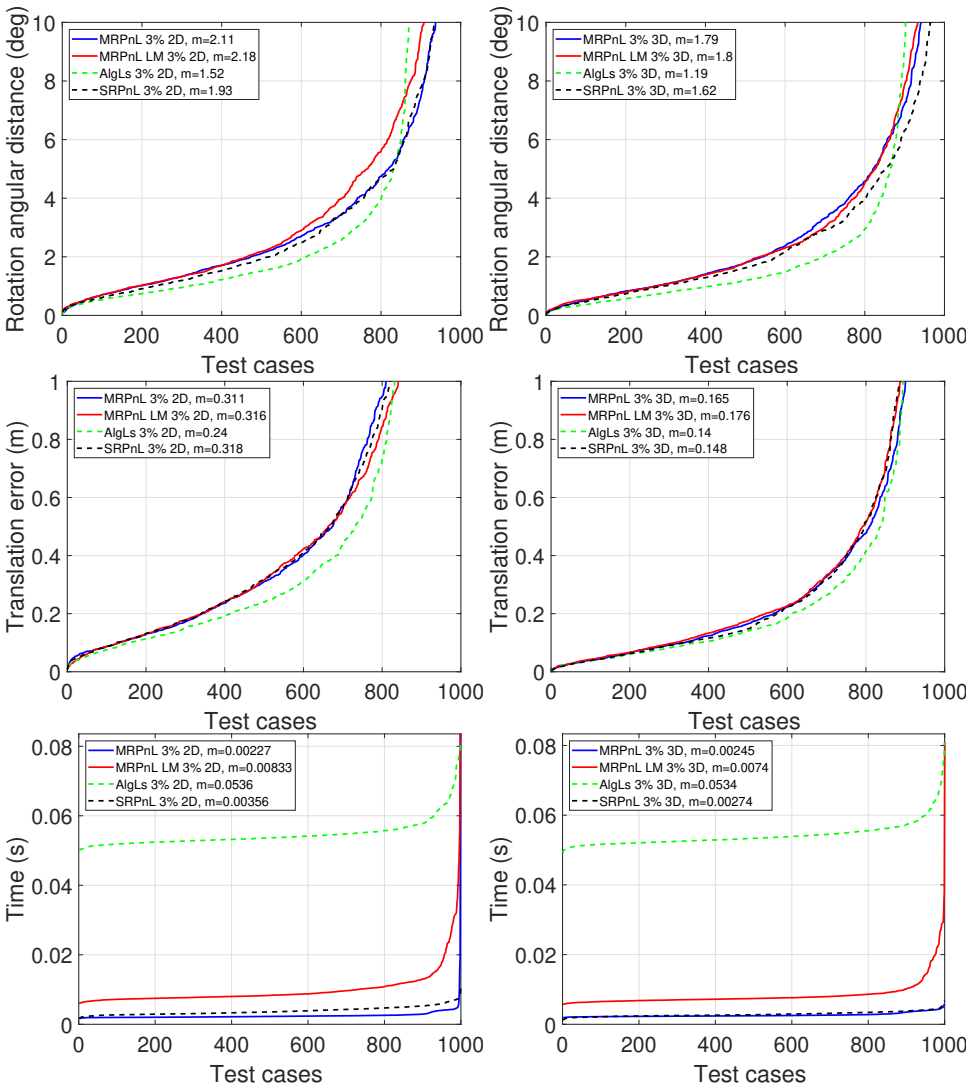


Figure 3.10. *Rotation error, translation error, and CPU time for MRPnL with and without refinement, SRPnL, and the globally optimal AlgLS with 3% noise and $n = 3$ lines, except for AlgLs $n = 4$. First column is with 2D noise, second column with 3D noise [1].*

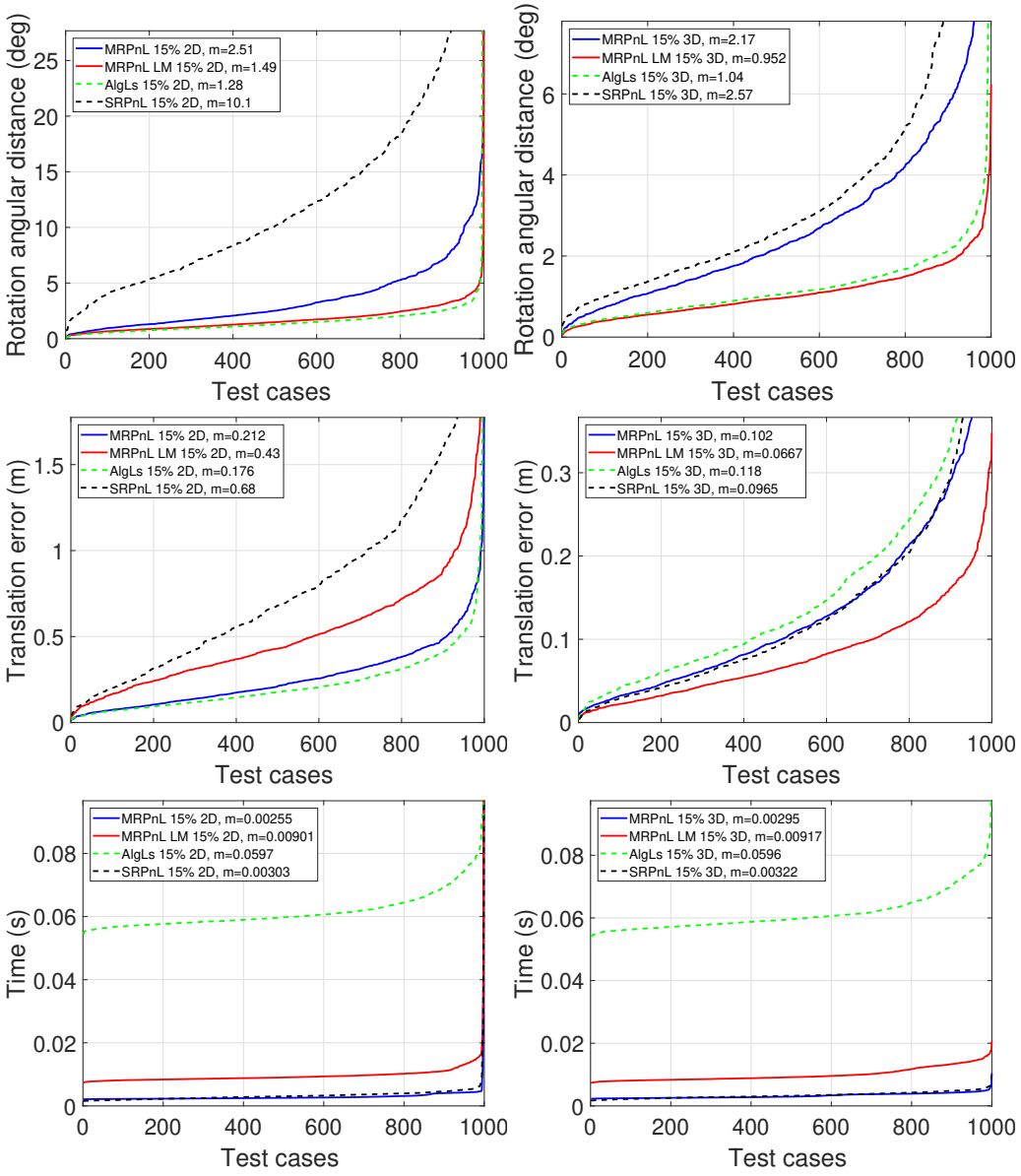


Figure 3.11. *Rotation error, translation error, and CPU time for MRPnL with and without refinement, SRPnL and the globally optimal AlgLS with* 15% *noise and all* 60 *lines used, first row with 2D noise, second row with 3D noise [1].*

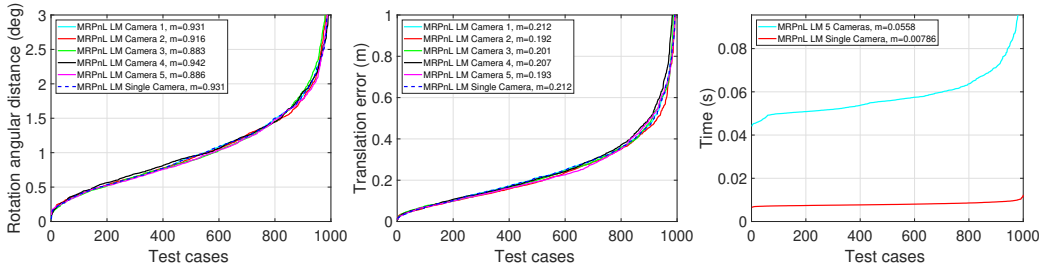


Figure 3.12. *Overall rotation error, translation error, and CPU time for MRPnL-LM in a multi-view setup with* 5 *cameras, using* 10% *2D noise [1].*

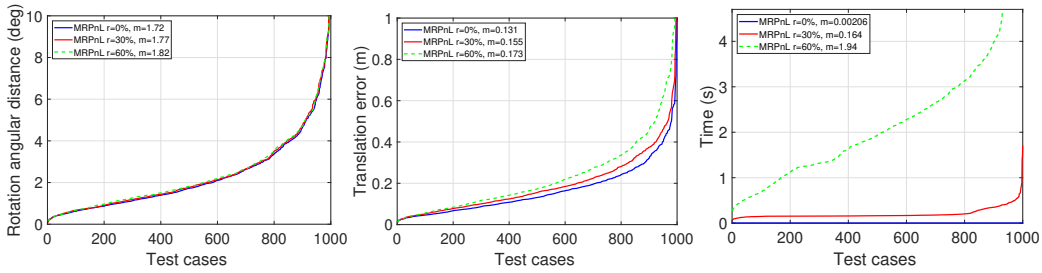


Figure 3.13. *MRPnL-LM pose estimation results on the inlier line-pairs obtained by RANSAC with* 10% *noise and* $r = 30\%$ $r = 60\%$ *outlier ratio, compared to the baseline results without RANSAC on the inlier set (*$r = 0\%$*) [1].*

slightly faster than AlgLS for a single camera. It is thus clear that the proposed MRPnL LM algorithm performs well in a multi-view setup, median rotation errors remain below 1° and the translation below 22 cm.

3.4.5 Robustness to Outliers

Since in a multi-view system, filtering the outliers has to be performed for each view independently from the others, we evaluate the robustness to outliers on a single camera only. The proposed MRPnL direct solver proved to be the fastest and more robust to noise than SRPnL, thus it is well suited for outlier detection in a RANSAC algorithm (see Section 3.4.2). In our experiments, we used the built in M-estimator sample consensus (MSAC) algorithm function of MATLAB [166] together with the line back-projection error presented in (3.25).

The synthetic dataset previously defined was extended by adding a specific number of outlier 2D-3D line-pairs with randomly generated coordinates, to obtain the outlier ratio of: 30% and 60% (26, 90 outliers respectively). The threshold for RANSAC was experimentally determined as the average between the maximum of the inliers' and minimum of the outliers' back-projection error calculated with the reference pose. In our tests, RANSAC with MRPnL was able to robustly filter out all outliers in most test cases, since there was a clear separation between the inliers and outliers, but we found that only a smaller inlier set can be obtained if the outlier lines are taken randomly from the same planes as the inliers, thus they are not different enough from the correct lines. Pose estimation errors of MRPnL on the obtained inliers with 10% noise and 60% outlier ratio are very similar to the baseline results obtained using only the inliers, we can observe an increase only in median translation errors, as shown in Fig. 3.13. The expense of such filtering is visible in the runtime plot in Fig. 3.13, where 30% outliers can be filtered relatively fast, but 60% outliers in almost two seconds.

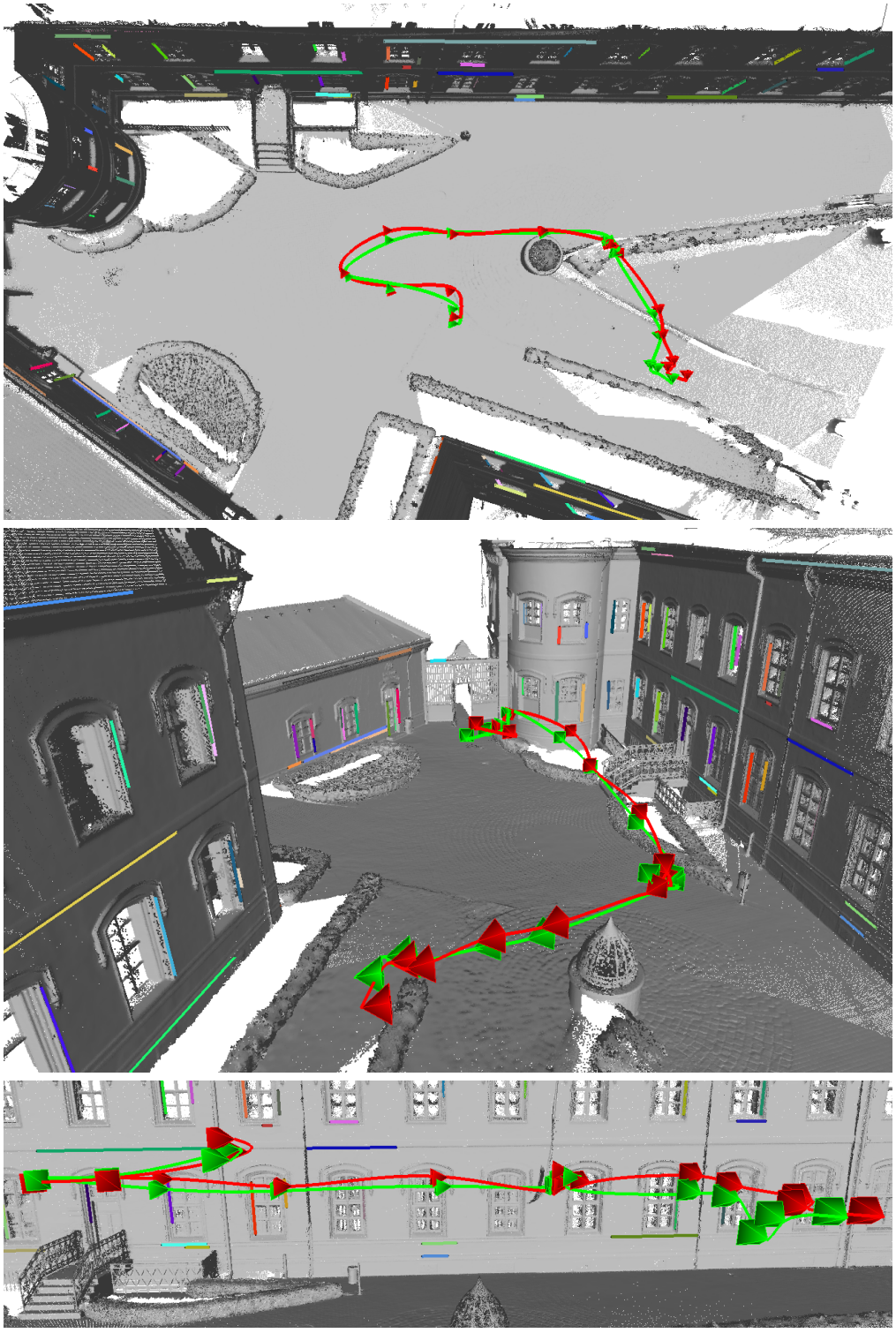


Figure 3.14. *MRPnL-LM trajectory estimation results on* 16 *frames of a longer drone sequence. Ground truth camera poses and the trajectory are shown in green, the estimated ones in red, while the used 3D lines (*81 *in total) are also visible (better viewed in color) [1].*

3.4.6 Real Data

To evaluate the proposed algorithm on real data, we used a 2D-3D dataset captured in an outdoor urban environment, containing a dense 3D pointcloud of the scene, captured with a Riegl VZ400 Lidar (with an attached Nikon DSLR providing reference RGB images), and 2D 4K resolution video sequences captured by a flying UAV along different trajectories. We extracted intermittent frames from the video sequence (16 frames from a sequence of 1800), to better evaluate the robustness to changing scene and lighting conditions. The ground truth pose of each camera was estimated with UPnP [83] using reflective markers placed on the scene, automatically detected by the scanner in 3D, and manually selected and matched in 2D. For many multi-view localization and visual odometry applications the most important criteria of a good result is the correct projection between 2D and 3D domain. This can be evaluated easily by forward projection error measured in the marker points (for the ground truth poses the maximum error was 12 cm, and the median 4 cm). 2D lines on the frames were extracted using OpenCV LSD detector [174]. In order to have a known inlier set, 2D-3D matching is done by manually matching 2D lines to lines detected on the LIDAR reference RGB images, that directly provides the corresponding 3D lines. Other methods (*e.g.* [116]) rely on VisualSFM, and use the estimated camera poses to project 2D lines into 3D, while with our approach, learnable line segment descriptors [172] could also be used for automatic 2D-2D matching.

Despite the fact that only a relatively small number of lines were used (an average of 13 lines and maximum 21 lines per image, compared to *e.g.* [116], where they used 130 lines and 50 points per image), the proposed MRPnL solver can estimate the absolute and relative poses quite robustly, 15 out of the 16 frames have a maximum forward projection error of less than 30 cm, the median forward projection error being 8 cm. Applying the LM refinement to the whole system increased the algorithm runtime from 80ms to 480ms, but all error measures show improvement, median forward projection error is reduced to 7.4 cm. Results of the camera path estimation can be seen in Fig. 3.14, where green marks the ground truth trajectory and camera positions and red the estimated one. All the used 81 3D lines are also shown in random colors.

Up to 30% outlier ratio is well tolerated in the real case too, showing similar results as above, even if the outliers are quite similar to the inliers, since they are randomly selected from the same visible scene planes. Results show similar performance to the synthetic experiments, errors increase only when the number of inliers gets drastically reduced by the outlier filtering due to no clear separation between inliers and outliers. We have shown, that the proposed MRPnL method is able to handle the challenging path estimation of 4DoF quadrotor UAVs that can have an unsystematic movement, making sudden turns, floating around in any direction.

3.4.7 Conclusion

In this section, a novel algebraic approach has been presented for computing the absolute and relative poses of a multi-view perspective camera system using line correspondences, which works without reformulation both for minimal problems (3 line pairs per camera) as well as for the general $n > 3$. Unlike previous approaches [178, 188, 203], rotation is solved first through a two-variate 7-th order system of polynomial equations using a Grobner basis solver, which reduces numerical error propagation and works both for minimal and general line sets. Then the translation is solved via a linear system. Experimental tests on large synthetic as well as real datasets confirm the State-of-the-Art performance of the proposed algorithm. Comparative results show that our method outperforms recent alternative methods (AlgLS [118], ASPnL [188], SRPnL [178]) in terms of speed, accuracy, and robustness. Furthermore, unlike these methods, our algorithm works for multi-view scenarios and is robust up to 60% outlier ratio when combined with a RANSAC-like method.

3.5 Pose Estimation using General Central Projection Cameras

In this section, we will propose a universal solution for central camera setups with omnidirectional and perspective cameras (see Fig. 3.2 and Fig. 2.1). Systems with a mixture of perspective and omnidirectional cameras allow us to cover a large field of view, that it is often necessary for a robust interpretation of highly complex urban scenes. Furthermore, camera combinations have been extremely useful, because they fit well the need of special vision-based robotics and autonomous vehicle localization and navigation applications [2, 42, 51, 64, 71, 111, 160], thanks to the bijective mapping in Section 2.1.2. The solution has a direct least squares solver to the absolute and relative pose problem. First, a minimal direct solver using Grobner basis which works with 3 line pairs, suitable for hypothesis testing in RANSAC [40]. Then a direct least squares solver which works for $n \geq 3$ 2D-3D line pairs. Both solvers run efficiently due to the low-order polynomial system of equations obtained via Cayley parametrization of the rotation matrix.

3.5.1 Cayley Parametrization of 3D Rotations

Beside the possible ways to represent the rotation matrix mentioned in Section 3.4. Here, we will use the Cayley transform to obtain a parametrization of the rotation matrix $\mathbf{R}$ in terms of 3 parameters $\mathbf{b} = [b_1, b_2, b_3]^\top$. Following [59, 185], The Cayley transform of a rotation matrix is a skew-symmetric matrix and vice versa. Therefore the correspondence $\mathbf{R} \leftrightarrow [\mathbf{b}]_\times$ is a one-to-one map between skew-symmetric matrices (represented as 3-vectors) and 3D rotations, excluding rotation angles $\pm 180^\circ$. Thus we have

$$\overline{\mathbf{R}} = (1 + \mathbf{b}^\top \mathbf{b})\mathbf{R} = (1 - \mathbf{b}^\top \mathbf{b})\mathbf{I} + 2[\mathbf{b}]_\times + 2\mathbf{b}\mathbf{b}^\top = \begin{bmatrix} 1 + b_1^2 - b_2^2 - b_3^2 & 2b_1b_2 - 2b_3 & 2b_1b_3 + 2b_2 \\ 2b_1b_2 + 2b_3 & 1 - b_1^2 + b_2^2 - b_3^2 & 2b_2b_3 - 2b_1 \\ 2b_1b_3 - 2b_2 & 2b_2b_3 + 2b_1 & 1 - b_1^2 - b_2^2 + b_3^2 \end{bmatrix} \tag{3.27}$$

Note that in the equations (3.3) and (3.5), we only use the above matrix, but to get the proper rotation matrix $\mathbf{R}$ from $\mathbf{b}$, the scale factor $(1 + \mathbf{b}^\top \mathbf{b})$ has to be used too:

$$\mathbf{R} = \left(\frac{1}{(1 + \mathbf{b}^\top \mathbf{b})} \right) \overline{\mathbf{R}} \tag{3.28}$$

3.5.2 Minimal Solver

Given a set of 2D-3D putative line correspondences, first an inlier set has to be determined in order to obtain a robust pose estimate. This can be done via RANSAC [40] or the M-estimator sample consensus (MSAC) algorithm [166], which relies on a minimal solver and a back-projection error metric. In our case, the minimal set consists of 3 line-pairs, providing 3 equations for the rotation only as in (3.3) and 3 equations for the translation as in (3.4). Using the Cayley parametrization of the rotation matrix $\mathbf{R}$, we get the following second

order polynomial equation from (3.3):

$$\mathbf{c}^\top \mathbf{x} = \begin{bmatrix} n_1v_1 + n_2v_2 + n_3v_3 \\ 2n_1v_2 + 2n_2v_1 \\ 2n_1v_3 + 2n_3v_1 \\ 2n_2v_3 + 2n_3v_2 \\ n_1v_1 - n_2v_2 - n_3v_3 \\ -n_1v_1 + n_2v_2 - n_3v_3 \\ -n_1v_1 - n_2v_2 + n_3v_3 \\ -2n_2v_3 + 2n_3v_2 \\ 2n_1v_3 - 2n_3v_1 \\ -2n_1v_2 + 2n_2v_1 \end{bmatrix}^\top \begin{bmatrix} 1 \\ b_1b_2 \\ b_1b_3 \\ b_2b_3 \\ {b_1}^2 \\ {b_2}^2 \\ {b_3}^2 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \mathbf{0} \tag{3.29}$$

where $\mathbf{n} = [n_1, n_2, n_3]^\top$ and $\mathbf{V} = [v_1, v_2, v_3]^\top$ are the projection plane unit normal and the 3D line unit direction vector, respectively. Given 3 such line-pairs, we obtain a system of 3 equations of the form (3.29) in the 3 unknown rotation parameters $\mathbf{b} = [b_1, b_2, b_3]^\top$, which can be easily solved by a solver using Grobner basis [80, 88, 93]. In our experiments, we used the automatic generator of Kukelova *et al.* [88] for a fair comparison in MATLAB with competing methods, but we remark that we also successfully used Kneip's generator [80] which produces a solver in C++, that is an order of magnitude faster! The translation $\mathbf{t}$ is then obtained by backsubstituting $\mathbf{R}$ into (3.4) yielding a system of linear equations, which can be solved by SVD decomposition. Although (3.29) might have several solutions, the solver will only return the real ones and then one has to select the geometrically valid $(\mathbf{R}, \mathbf{t})$ based on the visibility of the lines and the line back-projection error (see Section 3.5.3). Note that the relative camera poses $(\mathbf{R}_i, \mathbf{t}_i)$ are also obtained in a similar way once the absolute pose $(\mathbf{R}, \mathbf{t})$ is computed and backsubstituted into (3.5) and (3.6).

3.5.3 Line Back-Projection Error on the Unit Sphere

RANSAC will iteratively sample a minimal line-set, solve it via the minimal solver outlined above, and then classify the line-pairs into inliers and outliers based on the line back-projection error. Therefore the second component for our robust pose estimation is the line back-projection error. While solutions exists for perspective cameras [98, 162], these metrics are not usable in our case as we are working on the unit sphere $\mathcal{S}$. Therefore, given an observed image line segment with its endpoint spherical coordinates $(\widetilde{\mathbf{a}}, \widetilde{\mathbf{b}})$ and the corresponding 3D line backprojected to the unit sphere as $(\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}})$, let us define the line back-projection error of the 3D line w.r.t. its observed image line directly on $\mathcal{S}$. First of all, lines become *great circles* and a particular line segment becomes a *great circle segment* on the unit sphere. We will derive an error function to characterize the line projection error as the distance between $(\widetilde{\mathbf{a}}, \widetilde{\mathbf{b}})$ and $(\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}})$. Points on the observed spherical line $(\widetilde{\mathbf{a}}, \widetilde{\mathbf{b}})$ can be parametrized by a scalar $\phi = 0, \ldots, \lambda$ with λ being the geodesic length (or *great-circle length* or *orthodromic length* of the observed segment:

$$\lambda = \arctan\left(\frac{\|\widetilde{\mathbf{a}} \times \widetilde{\mathbf{b}}\|}{|\widetilde{\mathbf{a}} \cdot \widetilde{\mathbf{b}}|}\right). \tag{3.30}$$

A simple metric is to compute the shortest *orthodromic distance* of the endpoints of the observed segment to the backprojected line $(\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}})$. Let $\delta(\mathbf{p})$ be the shortest *orthodromic distance* of a point $\mathbf{p}$ on the observed segment to the backprojected line $(\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}})$:

$$\begin{aligned} \delta(\mathbf{p}) &= \arctan\left(\frac{|\mathbf{n} \cdot \mathbf{p}|}{\|\mathbf{n} \times \mathbf{p}\|}\right) \\ &= \arctan\left(\frac{\|\mathbf{n}\|\|\mathbf{p}\||\sin\theta|}{\|\mathbf{n}\|\|\mathbf{p}\||\cos\theta|}\right) = \theta \end{aligned} \tag{3.31}$$

where $\mathbf{n}$ is the unit normal vector of the projection plane of $(\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}})$, *i.e.*

$$\mathbf{n} = \frac{\widetilde{\mathbf{A}} \times \widetilde{\mathbf{B}}}{\|\widetilde{\mathbf{A}} \times \widetilde{\mathbf{B}}\|}, \tag{3.32}$$

θ is the angle (in radian) between $\mathbf{p}$ and the plane with normal $\mathbf{n}$, *i.e.* the plane passing through $\widetilde{\mathbf{A}}$, $\widetilde{\mathbf{B}}$, and the center of the sphere. Thus the shortest distance of $\widetilde{\mathbf{a}}$ and $\widetilde{\mathbf{b}}$ to the backprojected line are given by $\delta(\widetilde{\mathbf{a}})$ and $\delta(\widetilde{\mathbf{b}})$. Since the error represented by these distances is inversely proportional to the length of the line segment (same δ distance on a longer line segment means a smaller back-projection error), we used sum of squared distances weighted with the inverse of the length λ of the line segment:

$$\frac{1}{\lambda}(\delta^2(\widetilde{\mathbf{a}}) + \delta^2(\widetilde{\mathbf{b}})). \tag{3.33}$$

Note that The 2D image data is normalized by definition as we work on the unit sphere. However, the 3D lines are normalized for numerical stability [58] as explained in Section 2.6

3.5.4 Direct Least Squares Solver

Let us now focus on the general case, when we have $n > 3$ inlier but noisy 2D-3D line pairs. We also start from (3.3) and (3.4). Each line pair generates one such pair of equations, yielding a system of $n > 3$ equations, which is solved in the least squares sense. For this purpose, let's take the sum of squares of the nonlinear system constructed from (3.29):

$$E(\mathbf{b}) = \sum_{i=1}^{n} (\mathbf{c}_i^\top \mathbf{x})^2 \tag{3.34}$$

and then find $\arg\min_{\mathbf{b}} E(\mathbf{b})$. The first order optimality condition is

$$\nabla E(\mathbf{b}) = \begin{bmatrix} \frac{\partial E(\mathbf{b})}{\partial b_1} \\ \frac{\partial E(\mathbf{b})}{\partial b_2} \\ \frac{\partial E(\mathbf{b})}{\partial b_3} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \mathbf{d_{b_1}}_i^\top \mathbf{x_{b_1}} \\ \sum_{i=1}^{n} \mathbf{d_{b_2}}_i^\top \mathbf{x_{b_2}} \\ \sum_{i=1}^{n} \mathbf{d_{b_3}}_i^\top \mathbf{x_{b_3}} \end{bmatrix} = \mathbf{0} \tag{3.35}$$

where for each line pair $\mathbf{d_{b_1}}$, $\mathbf{d_{b_2}}$, and $\mathbf{d_{b_3}}$ can be expressed in terms of the coefficients $\mathbf{c}$ of each line pair. Thus the solution of the system of 3 polynomial equations (each of them is third order) in (3.35) provides the rotation parameters $\mathbf{b}$. We successfully used the solver generators of [80, 88] to generate MATLAB and C++ solvers for the above polynomial system. The translation $\mathbf{t}$ is then obtained by backsubstituting $\mathbf{R}$ into (3.4) yielding a system of linear equations, which can be solved by SVD decomposition. Multiple solutions are eliminated in the same way as for the minimal solver. Relative camera poses $(\mathbf{R}_i, \mathbf{t}_i)$ are also obtained in a similar way once the absolute pose $(\mathbf{R}, \mathbf{t})$ is computed and backsubstituted into (3.5) and (3.6).

Algorithm 3 Summary of the proposed robust pose estimation algorithm for N central cameras

Input: 2D-3D putative line matches from N cameras
Output: The absolute pose $(\mathbf{R}, \mathbf{t}) : \mathcal{W} \to \mathcal{C}$ and the relative poses $(\mathbf{R}_i, \mathbf{t}_i) : \mathcal{C} \to \mathcal{C}_i$

1: Calculate $\mathbf{N_3}$ as in Section 2.6 and normalize the 3D line endpoints $\mathbf{X}$.
2: Calculate the normal $\mathbf{n}$ of the projection plane for each 2D line and the unit direction vector $\mathbf{V}$ for each 3D line $\mathbf{L}$ as described in Section 3.2.
3: Filter outliers using the Cayley minimal solver proposed in Section 3.5.2 with MSAC.
4: Using the obtained inlier set of 2D-3D line pairs estimate the absolute $(\tilde{\mathbf{R}}, \tilde{\mathbf{t}})$ and relative $(\tilde{\mathbf{R}}_i, \tilde{\mathbf{t}}_i)$ poses with the Cayley-LS solver presented in Section 3.5.4.
5: Return the denormalized $(\mathbf{R}, \mathbf{t})$ and $(\mathbf{R}_i, \mathbf{t}_i)$ poses.

3.5.5 Experimental Results

Quantitative evaluation was performed on synthetically generated datasets. Since for both the perspective and omnidirectional cameras we used the calibration parameters of real cameras, with available physical parameters such as the sensor size, we calculated an estimated pixel-to-meter ratio, thus being able to represent our 3D scene in an equivalent metric space. Multiple sets of 1000 samples were generated containing 2D-3D line pairs. The 3D scene was created with a typical road scene in mind, where only a few planar surfaces are usually visible in a camera, thus we created 3 planes randomly placed (with a rotation of $\pm 30^\circ$ around all 3 axes, $\pm[1-2]$ m horizontal and vertical translation, and $\pm[0.5-1.5]$ m in depth) in the 3D space, each containing 20 random line segments, with a minimum length of 0.5 m.

For the 2D side we generated images of the scene by projecting the lines with perspective and omnidirectional cameras as well, using the parameters of a standard commercial camera with APS-C size sensor, 2378x1580 pixel resolution and 16 mm normal lens, respectively an 8 mm fisheye lens. Each camera was placed in the scene with a random rotation of $\pm 50^\circ$ around all 3 axes, and random translation of ± 1 m in the horizontal and vertical direction, while in the optical axis direction, the perspective camera was placed at $[4-6]$ m from the scene, the omnidirectional camera at $[2-3]$ m. Each of the 1000 test cases contains $2+1$ cameras, both in the omni-perspective and perspective-omni configuration to cover all possible variations of reference and relative cameras.

To evaluate the robustness of the proposed Algorithm 3 for noisy line detections, we simulated noise by corrupting one endpoint of the line (similarly in 2D and 3D), essentially adding a random number to each coordinate of the point up to the specified percentage of the actual coordinate value. The unit direction vector was also modified in the same manner. We show results for 7% and 15% 2D and 3D noise levels, with the only exception that on the omnidirectional images the 2D noise was limited to 10% instead of 15% because of the high nonlinear distortion of the camera that accentuates these errors. These error levels translate to a shift on the image with an average of $55-51$px (with 7% omni and perspective noise), 79px (with 10% omni noise) and 110px (with 15% perspective noise) respectively.

3.5.6 Comparison with State-of-the-Art

The proposed Cayley minimal solver and Cayley least squares solver (Cayley-LS), were compared to two State-of-the-Art methods: AlgLS, one of the most accurate non-iterative methods, which estimates the camera's pose by directly solving the corresponding least-squares problem algebraically, and SRPnL, a novel closed-form solution to the PnL pose problem that solves univariate polynomials and includes a Gauss Newton refinement. Comparisons were performed in two different setups, first using the minimum number of line matches that each algorithm requires, then using all 60 line pairs of the scene, only on a single perspective camera, since the formulation of the other methods doesn't support omnidirectional cameras, neither a multi-view setup was presented with them.

The minimum number of lines required is n=3, except for AlgLS, that uses n=4. Based on Fig. 3.15 we can conclude that all the methods perform very similar in terms of median errors of the pose parameters, only AlgLS produces slightly lower median errors due to the higher number of line-pairs it is using (n=4), but it is also the least robust of all tested methods, producing higher than 20° rotation error in 25% of the cases, compared to 15% with Cayley Minimal Solver, and 5% with SRPnL and Cayley-LS. None of the methods are handling well this level of noise, median angular distance is above 3.5° and 4.2° and translation error above 35 cm and 60 cm for 3D and 2D noise respectively. In terms of runtime, the proposed Cayley minimal solver is the fastest with 2 ms, followed by SRPnL with 3 ms,

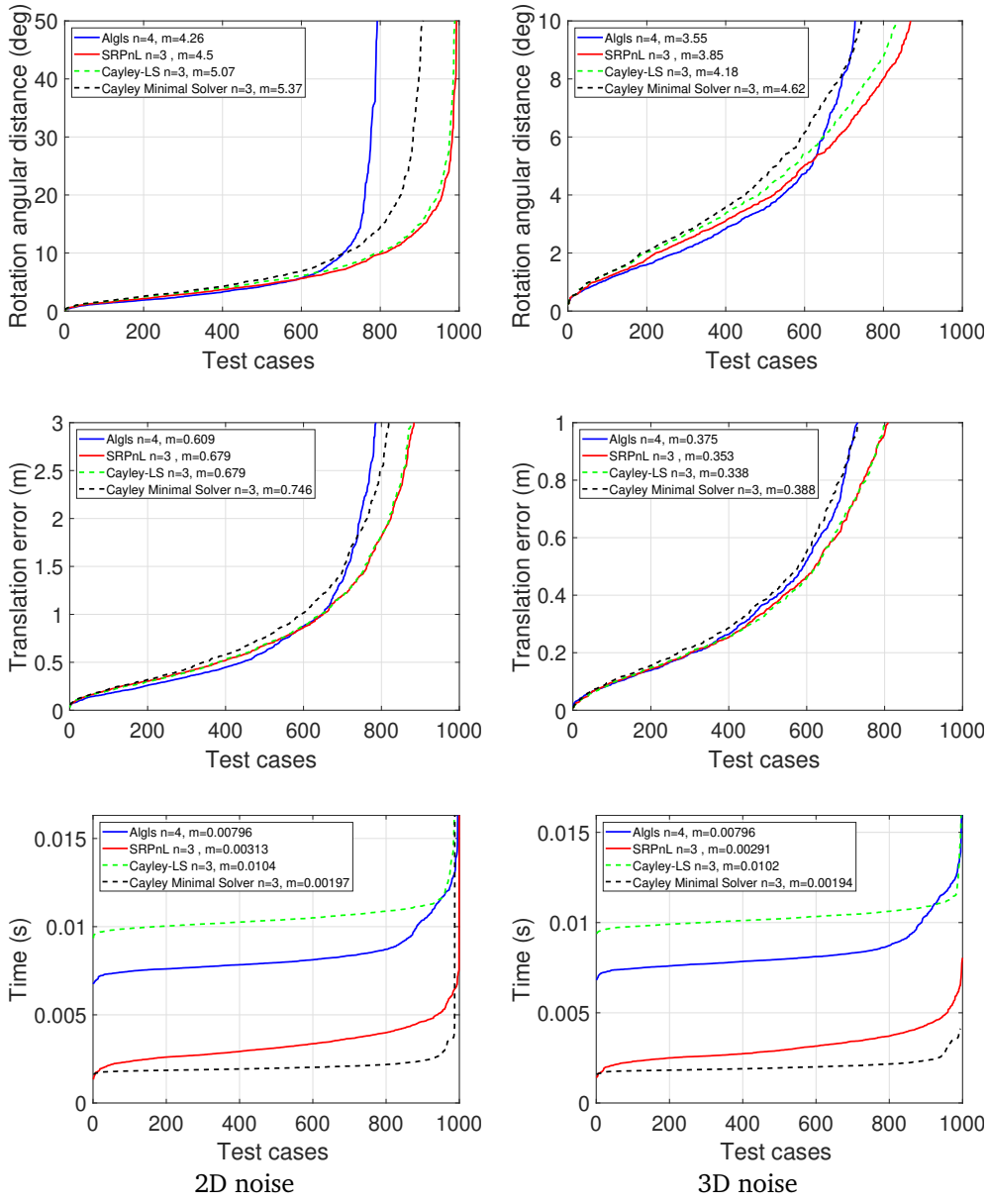


Figure 3.15. *Comparison to the State-of-the-Art methods with 7% 2D (left) and 7% 3D noise (right) using the minimum number of line matches with each method [2].*

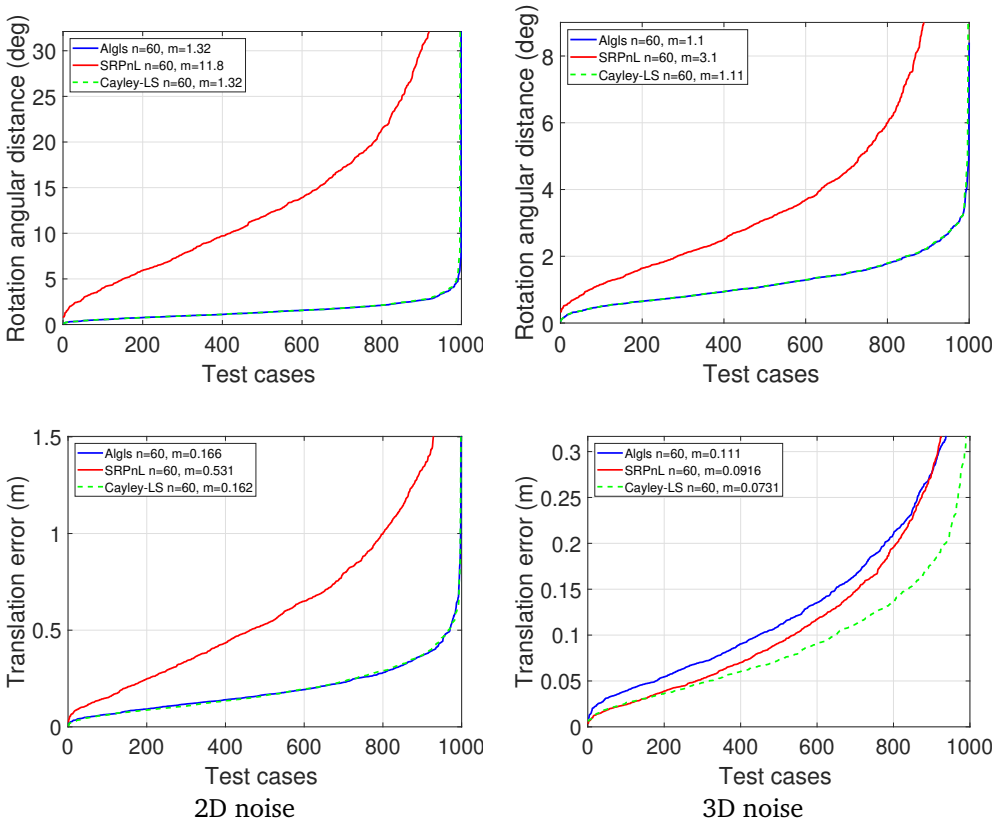


Figure 3.16. *Comparison to State-of-the-Art methods with* 15% *2D noise on the left and* 15% *3D noise on the right, using* $n = 60$ *line pairs [2].*

then AlgLS with 8 ms and Cayley LS 10 ms.

We also performed comparisons in case of $n = 60$ line pairs, where we also obviously excluded from the comparisons shown in Fig. 3.16 the Cayley Minimal Solver, since we are using $n = 60$ lines. The error plots in Fig. 3.16 show that AlgLS and Cayley-LS have the best results with lowest median rotation and translation errors, robust for up to 15% noise with median angular distance below $1.5°$, but AlgLS favors the noise in 2D domain. SRPnL shows lack of precision, producing much higher pose errors, already with 7% noise. Cayley-LS has the same characteristic as AlgLS in every test setup, the only difference noticeable is in case of 15% 3D noise where Cayley-LS is more robust in terms of translation error (see Fig. 3.16), while practically having the same execution time of 9 ms as AlgLS. With 60 lines the runtime of the algorithms keep the same characteristic and very similar median value, the number of lines not affecting significantly the runtime. We remark, that for the purpose of a fair evaluations we used the MATLAB solver generated by [88], while also a C++ solver generated by the automatic tool from the Polyjam software [80] is available, which is much faster.

3.5.7 Multi-view Setup Composition

Since we are dealing with perspective and omnidirectional cameras in the same framework, we have to test the sensibility of the proposed algorithms to different compositions of the camera setup. Due to the way our equations are formalized, we always have a selected reference camera that can propagate errors to the other cameras due to the absolute and relative pose formulation. In Fig. 3.17 we can see the two relevant configurations with 7% noise, *setup 1* referring to 2 perspective and one omni (2p+1o) cameras while *setup 2*

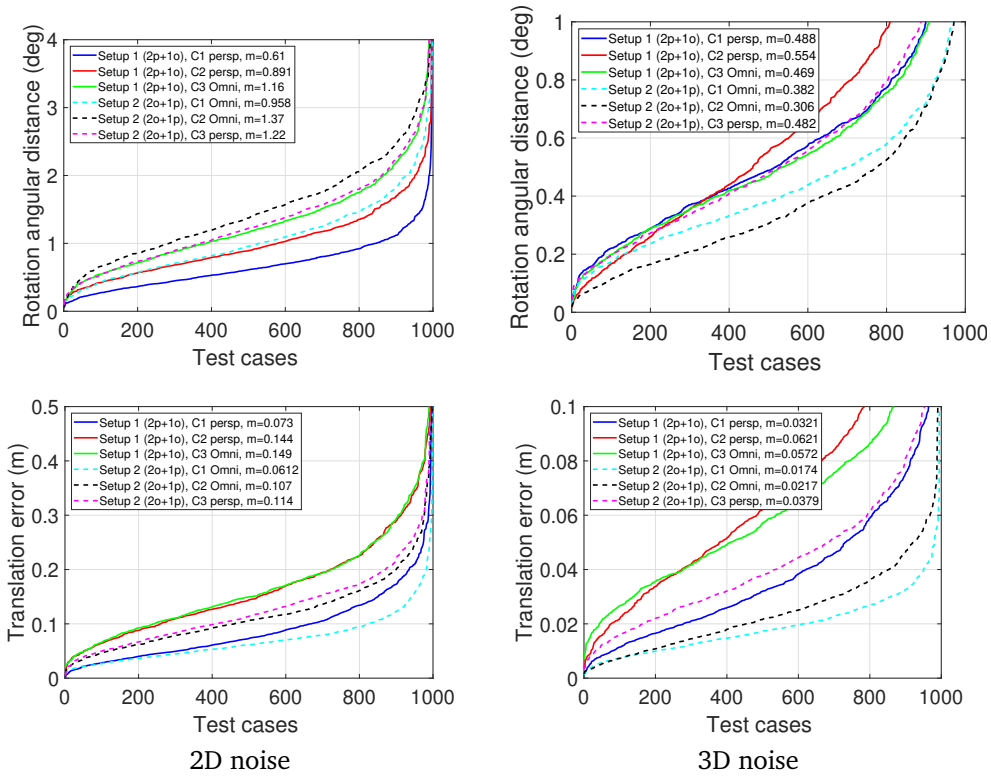
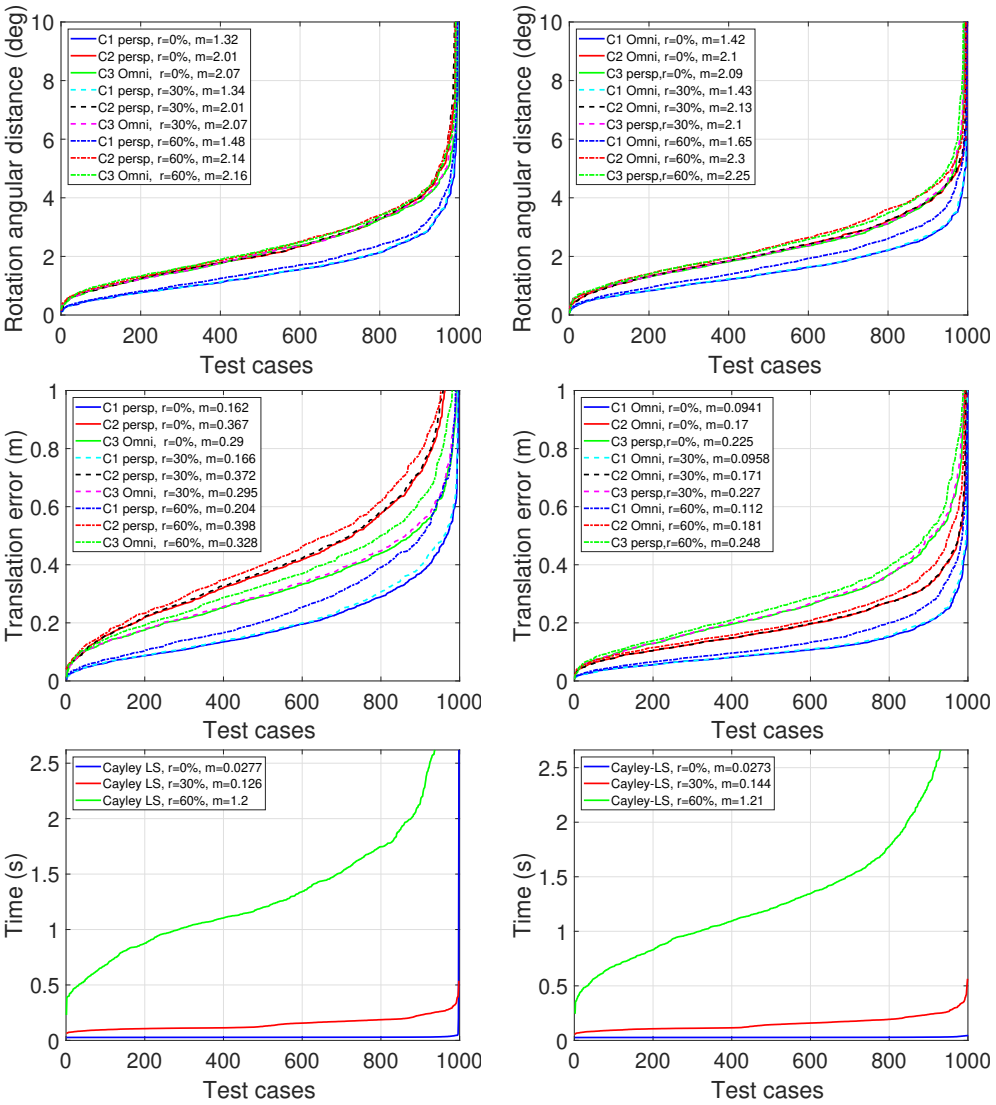


Figure 3.17. *Comparison of Cayley-LS results between different camera compositions in terms of median pose errors with* 7% *2D noise on the left and* 7% *3D noise on the right [2].*

referring to 2 omni and 1 perspective (2o+1p) cameras. These setups contain all 4 possible combinations of reference and relative camera. Based on Fig. 3.17 we can conclude, that there is no clear advantage of using one type of camera or the other as a reference, since *setup 1* has slightly better results with 2D noise, while *setup 2* is better with 3D noise.

3.5.8 Robustness to Outliers

Since the proposed Cayley minimal solver proved to be the fastest of the tested methods and robust to noise, it is well suited for outlier detection in a RANSAC algorithm. In our experiments we used the built in M-estimator sample consensus (MSAC) algorithm function of MATLAB [166] together with the line back-projection error presented in Section 3.5.3. The synthetic dataset previously defined was extended by adding a specific number of outlier 2D-3D line-pairs with randomly generated coordinates, to obtain the outlier ratio of: 30% and 60% (26, 90 outliers respectively). The threshold for RANSAC was experimentally determined as the average between the maximum of the inliers' and minimum of the outliers' back-projection error calculated with the reference pose. In our tests RANSAC with the Cayley minimal solver was able to robustly filter out all outliers, since there was a clear separation between the inliers and outliers, but we found that a smaller inlier set can only be obtained if the outlier lines are taken from the same planes as the inliers, thus they are not different enough from the correct lines. Pose estimation errors of the Cayley-LS solver on the inlier sets, using the two camera configurations presented previously with 15% noise and 30%, 60% outlier ratio, are shown in Fig. 3.18. We can see that the algorithm is robust up to 15% noise with 60% outliers where the angular distance or the translation doesn't change too much compared to the Cayley-LS solver run only on inliers. The runtime plots in the last row of Fig. 3.18 show that, while a higher than 50% outlier ratio can be filtered out robustly, it drastically increases the execution time of the algorithm. If a reduced number of outliers can be assumed ($< 30\%$ outlier ratio) a 5 folds increase in runtime is to be expected,



setup 1 with (2p+1o) *setup 2* with (2o+1p)

Figure 3.18. *Cayley-LS pose estimation results on the inlier line-pairs provided by RANSAC with* 15% *noise and* $r = 30\%$*,* $r = 60\%$ *outlier ratio, compared to the baseline results without RANSAC on the inlier set* $r = 0\%$ *[2].*

that could still fit in many applications' requirements.

3.5.9 Real Data

To evaluate the proposed algorithm on real data, we have used a set of 17 2D perspective and omnidirectional images captured in an outdoor urban environment (see Fig. 3.19), where the dense 3D point cloud of the scene was captured with a Riegl VZ400 Lidar scanner with an angular resolution of $0.05°$. The perspective images were captured by a flying drone in 4K resolution, while the omnidirectional images were taken with a Canon DSLR camera with a 8mm fisheye lens. The ground truth pose if each camera images was estimated with UPnP [83] using highly reflective markers placed on the building surface, that were automatically scanned by the scanner, and detected and matched with 2D manually. Relying on these markers we can compute a metric forward projection error to evaluate the precision of the estimated camera pose. For the reference poses the maximum forward projection error was 10 cm, and the median was 3 cm. To provide the necessary input to our algorithm, we

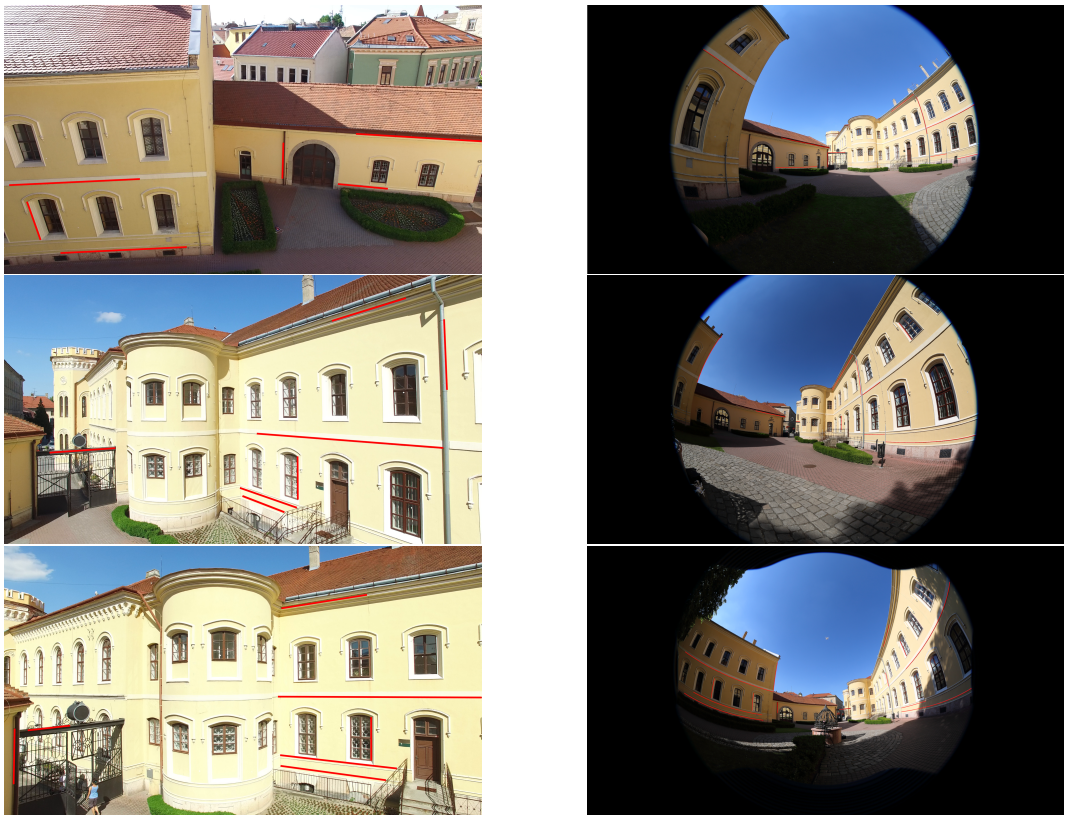


Figure 3.19. *Some of the images (perspective and Omnidirectional) and Lines (in red) used in the real experiment*

detected 2D lines on the perspective images using the OpenCV LSD detector [174], while on the omnidirectional images we used the automatic line extraction toolbox of Bermudez [19]. The corresponding 3D lines were produced by relying on the images captured with the camera attached to the scanner, that has a very precise pose calibrated in laboratory environment. We used these to project 2D lines detected on the reference images into the 3D pointcloud, then manually matching the 2D lines from our evaluation set of 17 images with the lines on the reference images, we directly obtained the 2D-3D matches. This could also be done by calculating line segment descriptors like [172] and using them to automatically match lines.

We evaluated the Cayley-LS solver on the 17 images of the real dataset by choosing randomly a reference camera. The results show, that despite the fact that we used only a relatively small number of lines (an average of 15 lines and maximum 22 lines per image, compared to *e.g.* [117], where they used 130 lines and 50 points per image) the proposed Cayley-LS solver can estimate the absolute and relative poses quite robustly, even independently of the selection of the reference camera. In 16 out of the 17 configurations all cameras have a correct pose estimated with a maximum forward projection error of less than 30 cm, except one camera, that might have had too much noise on the 3D lines. Obviously in the 17^{th} configuration, when this camera is chosen as reference, the errors propagate over to multiple other cameras, thus only a total of 12 cameras have lower than 30 cm maximum forward projection error. In the other 16 cases median rotation errors are below 1° along all three axes, while median translation errors below 40 cm. Considering the distance of the cameras from the scene walls is between $10-25$ meters, these results prove the precision and robustness of the proposed method, even for low number of lines in different multi-view setups.

Fusion Based on the calculated camera poses the 2D-3D fusion can be performed by colorizing the scene pointcloud from all the available cameras. An important step of this is checking the visibility of the surfaces from each given viewpoint, since in a complex urban environment occlusions can easily happen. For this purpose we used the hidden point remove tool of [77] to obtain only the visible scene-points from each camera, then we averaged out the color values proposed by multiple cameras for each vertex. The resulting colorized pointcloud for the dataset can be seen in Fig. 3.20, including the estimated camera poses.

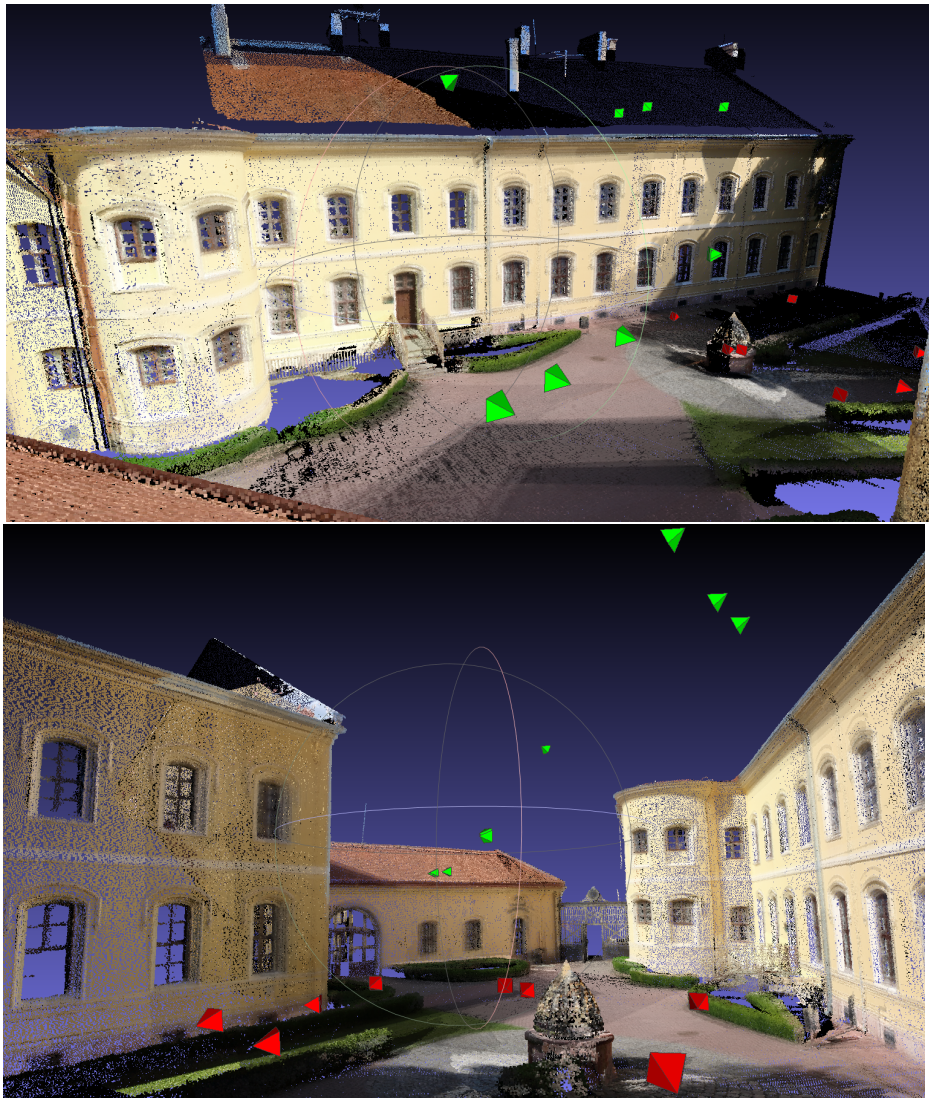


Figure 3.20. *Fusion result shown as colorized pointcloud with estimated omni (red) and perspective (green) camera positions illustrated [2].*

3.5.10 Conclusion

In this section, a novel robust pose estimation method for central perspective and omnidirectional cameras using line correspondences was presented. Due to the Cayley representation of the rotation, the proposed approach yields a low-order polynomial system both for the minimal as well as for the general n-line case, which can be efficiently solved using Grobner basis solvers. The proposed method is able to deal with outliers as well as noise on the line parameters. It compares favorably to the State-of-the-Art methods, being more robust to 3D noise than AlgLS and SRPnL. The efficiency of the proposed solution was validated both on

synthetic and real data with omnidirectional and perspective camera setups.

3.6 Solver Analysis

In this section, the three proposed solutions are compared in terms of performance and time execution. This study will make a clear conclusion about the capability of each solver. The study will show the behavior of the proposed solutions with noisy data as well as the perfect data. Moreover, the study will explore the number of solutions returned by the solvers and the usability of the line back-projection functions presented in Section 3.4.2 for the perspective cameras; and the line back-projection error which works on the unit sphere presented in Section 3.5.3. All the solvers have been set to return all solutions (real solutions and real part of the complex solutions). All the experiments were done on an i7 computer with a 3.40GHz CPU and 16 GB of RAM on MATLAB R2018b. Also, the solvers have been optimized and shortened in terms of code.

- The first solution [4] presented in Section 3.3 which uses the vertical direction is referred as MVVD (multi view vertical direction)
- The second solution [1] presented in Section 3.4 is referred as MRPnL and MRPnL-LM when including the refinement step
- The third solution [2] presented in Section 3.5 is referred as Cayley for the minimal solver and Cayley-LS for the least squares solver

3.6.1 Unified Comparison of the Solvers

The three proposed solution are compared in terms of performance with up to 3% 2D and 3D noise using 3(minimal case)/30/60 2D-3D line pairs per camera (3% noise on 2D error corresponds to [-22,+22] pixel error), and in term of time execution. We have used two cameras for a fair comparison between the solvers. The error in the following plots shows the relative pose error for the second camera. Note that in the minimal case, the same 3 2D-3D line pairs have been used for all the solvers.

Because of the vertical direction ($\mathbf{R}_v = \mathbf{R}_Z(\gamma)\mathbf{R}_Y(\beta)$ around Y and Z axes) is provided for MVVD, we are going to show the error on the X rotation angle and the global angular distance. Obviously, the performance of the MVVD is the best in the minimal cases because of the known vertical direction. However, we observe in Fig. 3.21,Fig. 3.22 and Fig. 3.23 that the MVVD is similar in rotation error with MRPnL-LM on the case of 30 2D-3D line pairs, but slightly better than the two other solvers in the case of 60 2D-3D line pairs, which is due to the noise level on the line. No big difference is observed on the translation error between the solvers, just a tiny small difference in 30 2D-3D line pairs case and almost identical with 60 2D-3D line pairs. Additionally, the MVVD is seen to be faster in the minimal case than the Cayley-LS due to the one iteration setting. However, the MVVD seems to be $\approx$8 times slower than MRPnL-LM and Cayley-LS in the genral case with n>3.

Interestingly, MRPnL outperforms the Cayley-LS in the minimal case in terms of pose error and time. But, compared to the Cayley minimal solver, they perform almost identical in terms of pose error and time. The same thing is observed in the case of 30 2D-3D line pairs. However in the case of 60 2D-3D line pairs, both MRPnL-LM and Cayley-LS have similar performance in terms of pose error; there is only a 3ms median difference in time.

Increasing the noise to 15% we can see in Fig. 3.24 the robustness of the Cayley-Ls . For such case, Cayley-LS outperform MRPnL in terms of pose error and ranked the best in terms of absolute translation compared to MVVD.

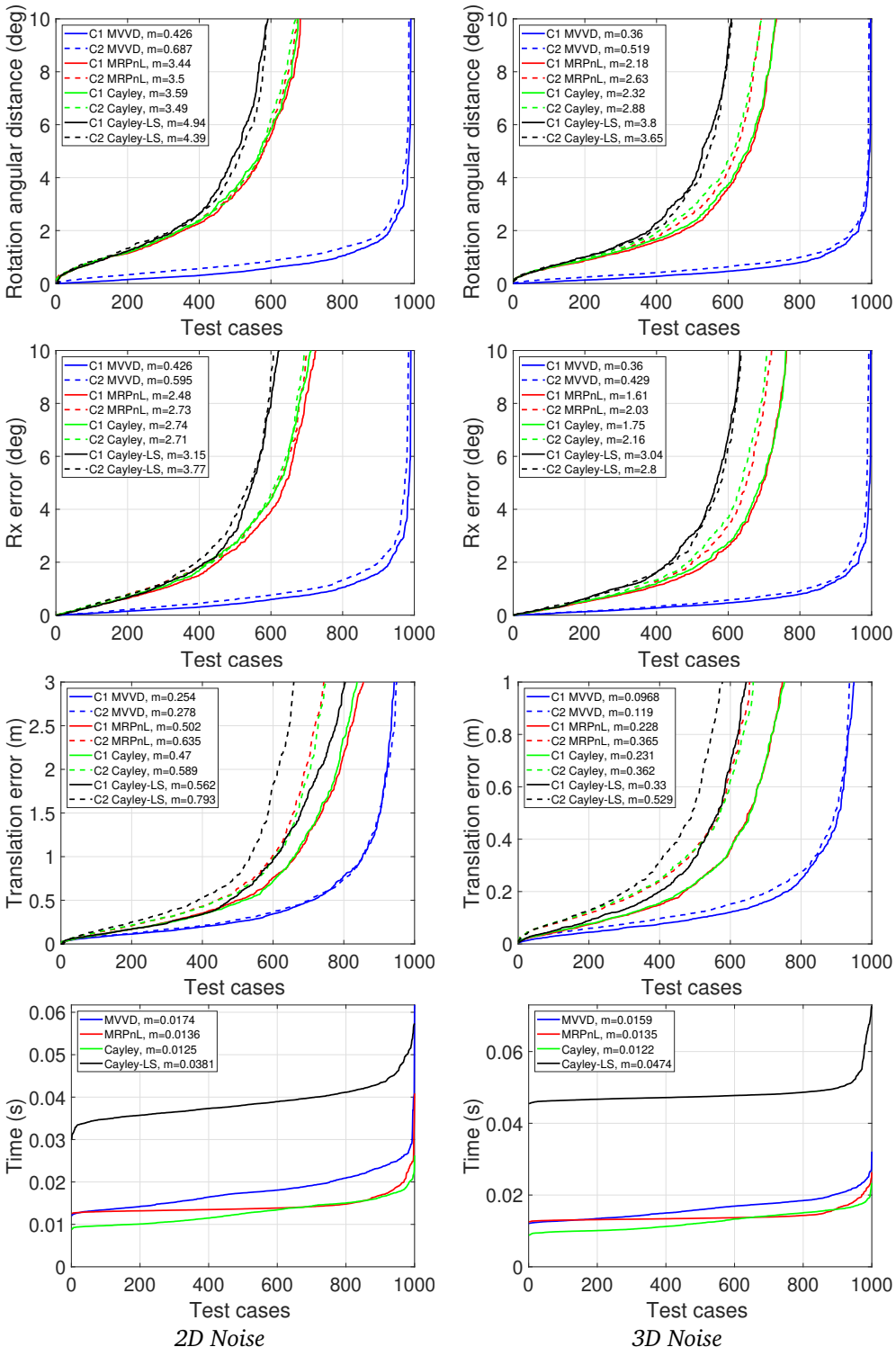


Figure 3.21. *Comparisons between the different solvers with the minimal case using 2 cameras in terms of median pose error with 3% 2D noise on the left and 3% 3D noise on the right.*

3.6.2 All Solutions Returned VS Only Real Solutions

We explore here the number of solutions returned by the solvers in the case of the perfect data and the type of solutions returned: real solutions or complex solution. We observe also the performance and the difference of the solvers when we set to return all solutions which

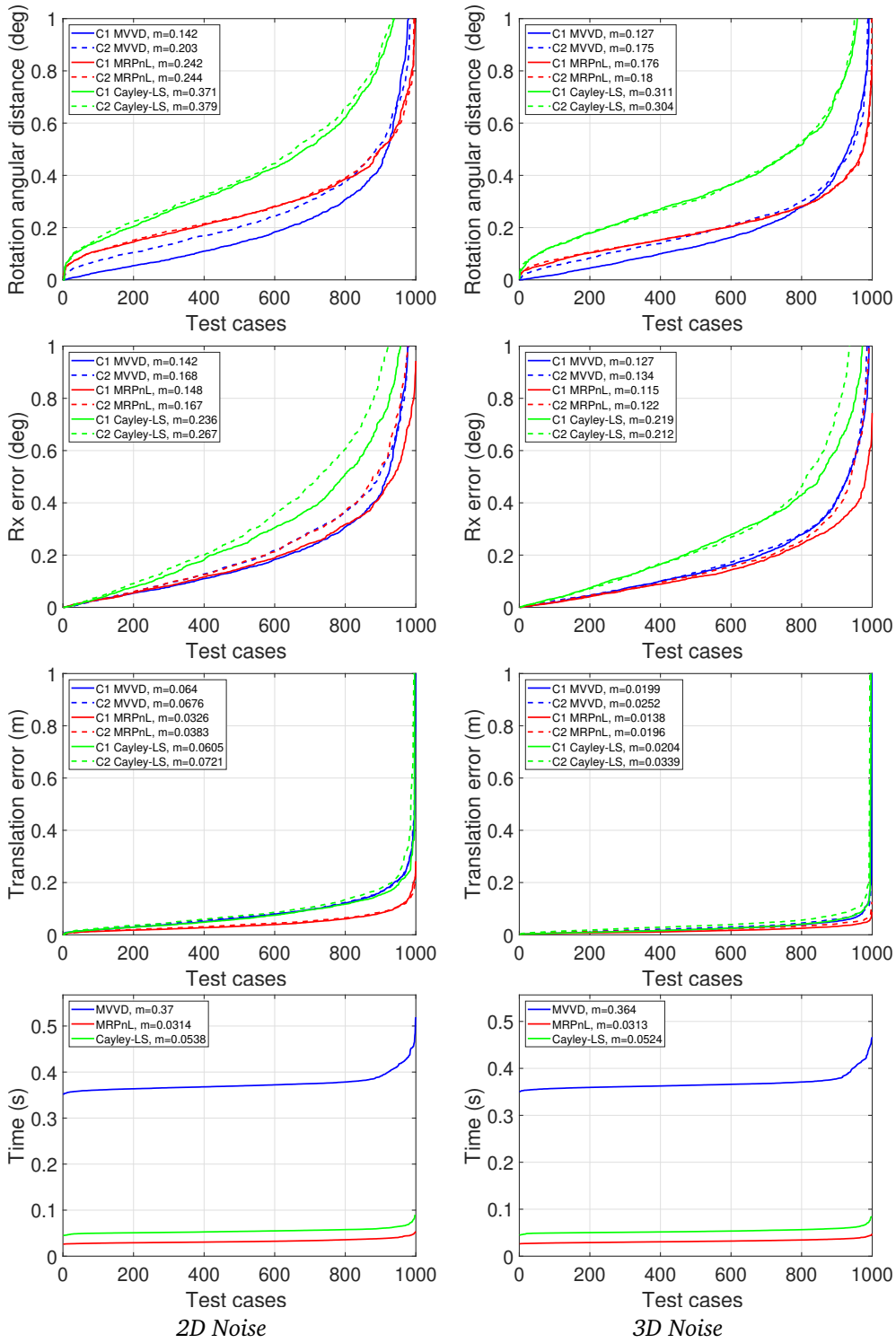


Figure 3.22. *Comparisons between the different solvers with 30 2D-3D line pairs using 2 cameras in terms of median pose error with 3% 2D noise on the left and 3% 3D noise on the right.*

include the real ones and the real part of the complex solutions compared to when we return the real solutions only.

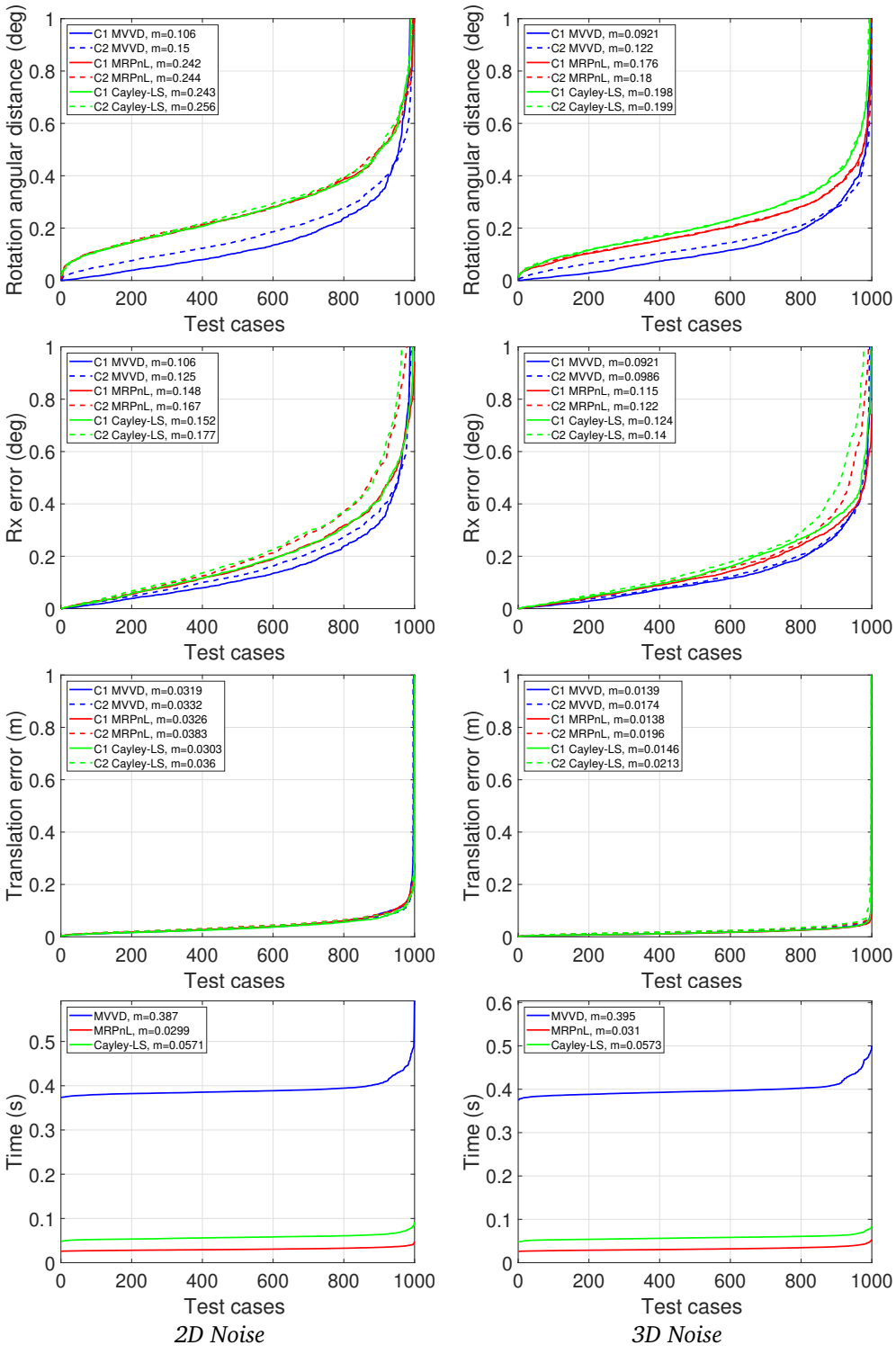


Figure 3.23. *Comparisons between the different solvers with 60 2D-3D line pairs using 2 cameras in terms of median pose error with 3% 2D noise on the left and 3% 3D noise on the right.*

Pose Error

MRPnL does not show any difference in the pose error for the minimal case Fig. 3.25 and the general case of 30/60 2D-3D line pairs Fig. 3.26, of course, the time is slightly smaller

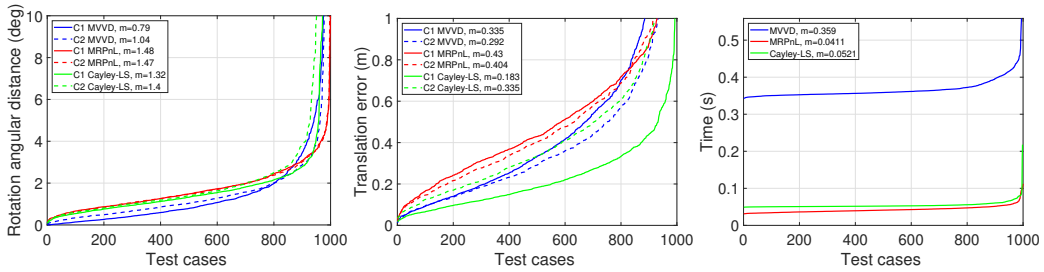


Figure 3.24. *Extra case: Robustness of the Cayley Solver. Comparisons between the different solvers with 60 2D-3D line pairs using 2 cameras in terms of median pose error with 15% 2D noise.*

for the case when we return only real solutions, the curves are exactly identical for both cases. For the Cayley minimal solver and Cayley-LS, we observe a slight difference in the error curve in the minimal case Fig. 3.27, and almost unnoticeable difference in the 30/60 2D-3D line pairs case Fig. 3.28, especially in the translation error. It is interesting that in the minimal case, the median translation error for the Cayley minimal solver and Cayley-LS is better when we include the real part of the complex solutions, but not for Cayley-LS with the median angular distance error, which prefers the real solutions only. Of course, the time is affected when we return more solutions!

Number of Solution

MVVD: In the minimal case, MVVD returns 4 solutions for each camera, from which we do a combination of 16 solutions (between each pairs of reference and other camera), the solutions are checked using the line back-projection error presented in Section 3.4.2. For the perfect data, the solver returns most of the time only real solutions. Over the 1000 solved cases, we have 889 cases where a real solution was picked, 60 cases with complex solutions from which we take the real part, and 51 cases where we have a mixture of real and complex solutions over the camera pair.

MRPnL minimal case (Table 3.6): In the minimal case, MRPnL returns 25 solutions (real + complex).

- In the perfect data: 3/1000 cases where the solver selected a solution from the complex ones
- In the 3%2D noisy data: 7/1000 cases where the solver selected a solution from the complex ones

solutions	Perfect data	3%2D Noise	solutions	Perfect data	3%2D Noise
1	0	3	8	1	0
3	5	19	10	53	44
5	40	43	12	42	40
7	601	584	14	126	105
9	132	162	16	132	162
11	126	105	18	601	584
13	42	40	20	40	43
15	53	44	22	5	19
17	1	0	24	0	3

Table 3.6. *Overview about the occurrence number of the real solutions (Left) and only complex solutions (Right) when we use the minimal number of line for MRPnL.*

MRPnL general case (Table 3.7): In the general case of 60 2D-2D line pairs, MRPnL returns also 25 solutions (real + complex) for each of the 1000 test cases

- In the perfect data: 1/1000 cases where the solver selected a solution from the complex ones
- In the 3%2D noisy data: 0/1000 cases where the solver selected a solution from the complex ones

solutions	Perfect data	3%2D Noise	solutions	Perfect data	3%2D Noise
1	904	908	20	2	2
3	94	90	22	94	90
5	2	2	24	904	908

Table 3.7. *Overview about the occurrence number of the real solutions (Left) and only complex solutions (Right) when we use 60 lines for MRPnL.*

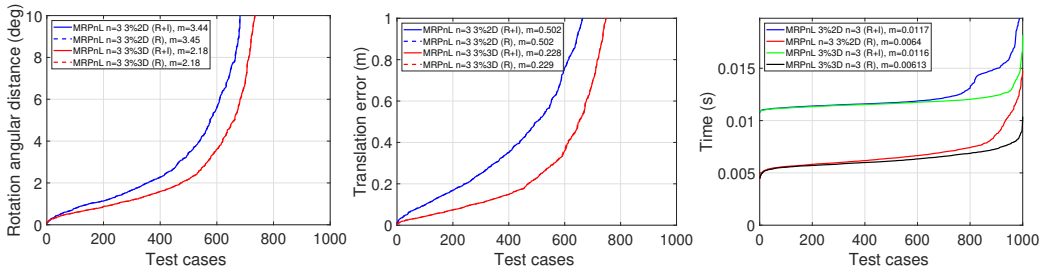


Figure 3.25. *Study of the MRPnL solver when All solutions are returned (R+I: real part of complex solution and real ones) and when only real solutions (R) are returned with 3 2D-3D line pairs using 1 camera in terms of median pose error with 3% 2D noise.*

Cayley Minimal solver (Table 3.8): In the minimal case, the solver returns always 8 solutions (real + complex) for each of the 1000 test cases.

- In the perfect data: 0/1000 cases where the solver selected a solution from the complex ones
- In the 3%2D noisy data: 43/1000 cases where the solver selected a solution from the complex ones

solutions	Perfect data	3%2D Noise	solutions	Perfect data	3%2D Noise
2	56	60	2	147	118
4	748	761	4	748	761
6	147	118	6	56	60
8	49	43	8	0	18

Table 3.8. *Overview about the occurrence number of the real solutions (Left) and only complex solutions (Right) when we use 3 lines for the Cayley minimal solver.*

Cayley-LS Minimal Case (Table 3.9): In the minimal case, the solver returns always 27 solutions (real + complex) for each of the 1000 test cases we could solve 993 cases in the perfect case.

- In the perfect data: 4/993 cases where the solver selected a solution from the complex ones

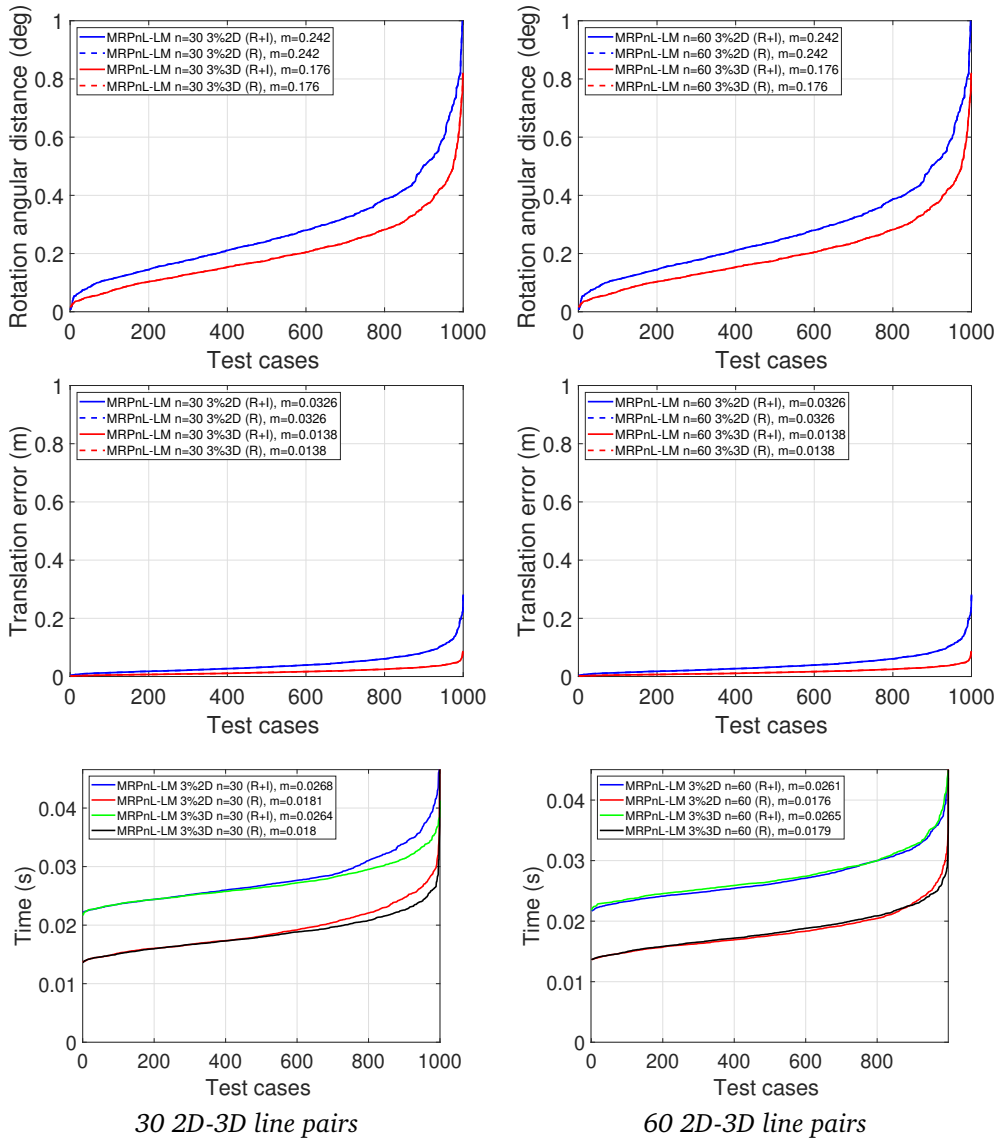


Figure 3.26. *Study of the MRPnL-LM solver when all solutions are returned (R+I: real part of complex solution and real ones) and when only real solutions (R) are returned with 30/60 2D-3D line pairs using 1 camera in terms of median pose error with 3% 2D noise.*

- In the 3%2D noisy data: 4/1000 cases where the solver selected a solution from the complex ones

Cayley-LS General Case (Table 3.10): In the general case of 60 2D-2D line pairs returns also 27 solutions (real + complex) for each of the 1000 test cases

- In the perfect data: 3/1000 cases where the solver selected a solution from the complex ones
- In the 3%2D noisy data: 1/997 cases where the solver selected a solution from the complex ones

3.6.3 Line Back-Projection

To show the usability and the limitation of the two line back-projection functions Section 3.4.2 and Section 3.5.3. We analyzed the line back-projection values obtained using

solutions	Perfect data	3%2D Noise	solutions	Perfect data	3%2D Noise
3	0	7	8	2	1
5	29	36	10	21	13
7	451	462	12	47	50
9	237	237	14	80	71
11	126	123	16	126	123
13	80	71	18	237	237
15	47	50	20	451	462
17	21	13	22	29	36
19	2	1	24	451	7

Table 3.9. *Overview about the occurrence number of the real solutions (Left) and only complex solutions (Right) when we use 3 lines for the Cayley-LS.*

solutions	Perfect data	3%2D Noise	solutions	Perfect data	3%2D Noise
1	631	625	18	2	2
3	312	311	20	11	12
5	44	47	22	44	47
7	11	12	24	312	311
9	2	2	26	631	625

Table 3.10. *Overview about the occurrence number of the real solutions (Left) and only complex solutions (Right) when we use 60 lines for the Cayley-LS.*

the selected pose with the line back-projection function, and the selected pose using the GT pose, with one camera and 3 line pairs using perfect and noisy data (3%2D) over the 1000 test cases Fig. 3.29. The line back-projection error from both functions on the perfect data are very small at the order of 1e-15, and 1e-23 (almost absolute zero value). This means that the projected 3D lines into the normalized image plane, or in the unit sphere, are perfectly overlapping the corresponding 2D lines, with the selected pose using both line back-projection functions, or using the GT pose. Obviously, the same behavior is occurring with the noisy data because the line back-projection function guarantees the selection of a pose that project the 3D lines correctly into 2D. Concerning the difference between the pose error when we select the pose with the line back-projection function or the GT pose we can see in Fig. 3.30 and Fig. 3.31 that in the minimal case there is up to 1° and 2° in median angular distance difference in rotation for MRPnL and the Cayley-LS, respectively. And up to $20cm$ in median translation for both MRPnL and the Cayley-LS. It is really hard to make sure that the line back-projection function finds the optimal solution in the minimal case because of the large number of the return solutions and due to the configuration of the 3 line pairs, which could be exactly aligned as the observed segments in 2D after being projected. However, no difference is observed in the general case of 60 2D-3D line pairs, which means that selecting the correct pose is the same with the line back-projection function or with the GT pose.

3.6.4 Kukelova's Solver Generator VS Kneip's Solver Generator

Lastly, we investigate the difference between the solver generated with two solvers generator, Kukelova's automatic solver generator [88] and Kneip's automatic solver generator [80].

Kukelova's automatic solver generator: The automatic generator of Kukelova uses a Grobner basis and consists of three key steps. First: the input problem is solved using a

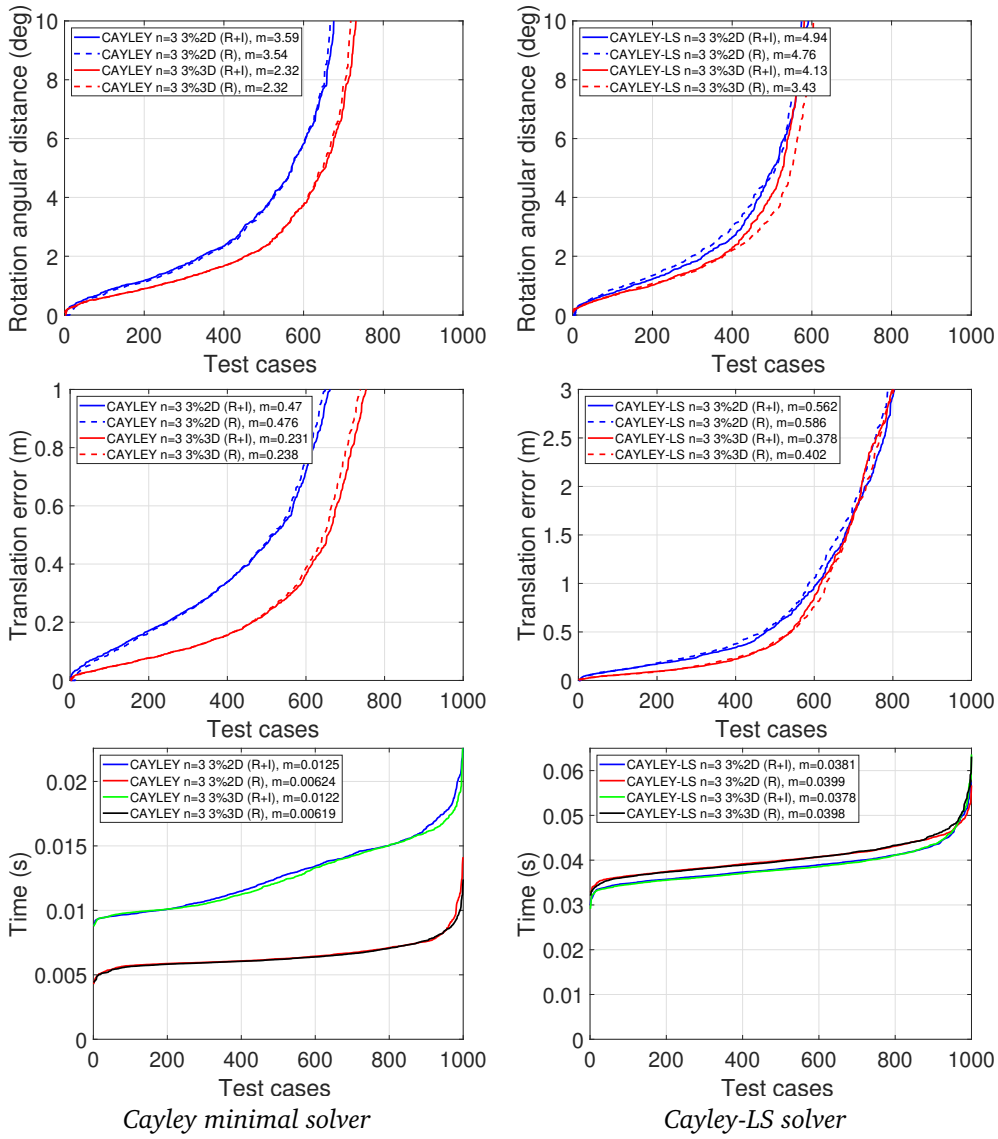


Figure 3.27. *Study of the Cayley minimal solver and the Cayley-LS when all solutions are returned (R+I: real part of complex solution and real ones) and when only real solutions (R) are returned with 3 2D-3D line pairs using 1 camera in terms of median pose error with 3% 2D noise.*

computer algebra system like Macaulay2 [53] by finding a Grobner basis for the equations on finite field in order to avoid expensive growth of coefficients and to avoid numerical instability. Second: s step is called elimination template, which is needed to obtain a computationally efficient and numerically robust procedure through selecting the set of polynomial equations to obtain a Grobner basis for constructing a special matrix. the action matrix is then constructed in the last step from the resulting equations, and the solutions are obtained numerically as the eigenvalues or eigenvectors of the active matrix and return in the form of equations [88]. Note that the solvers generated has been set to return all solution as explained at the beginning of the section.

Kneip's automatic solver generator: Polyjam is a C++ library for setting up algebraic geometry problems and generating efficient C++ code that solves polynomial systems of equations. Polyjam inspired from the method presented in Kukelova's solver generator [88],

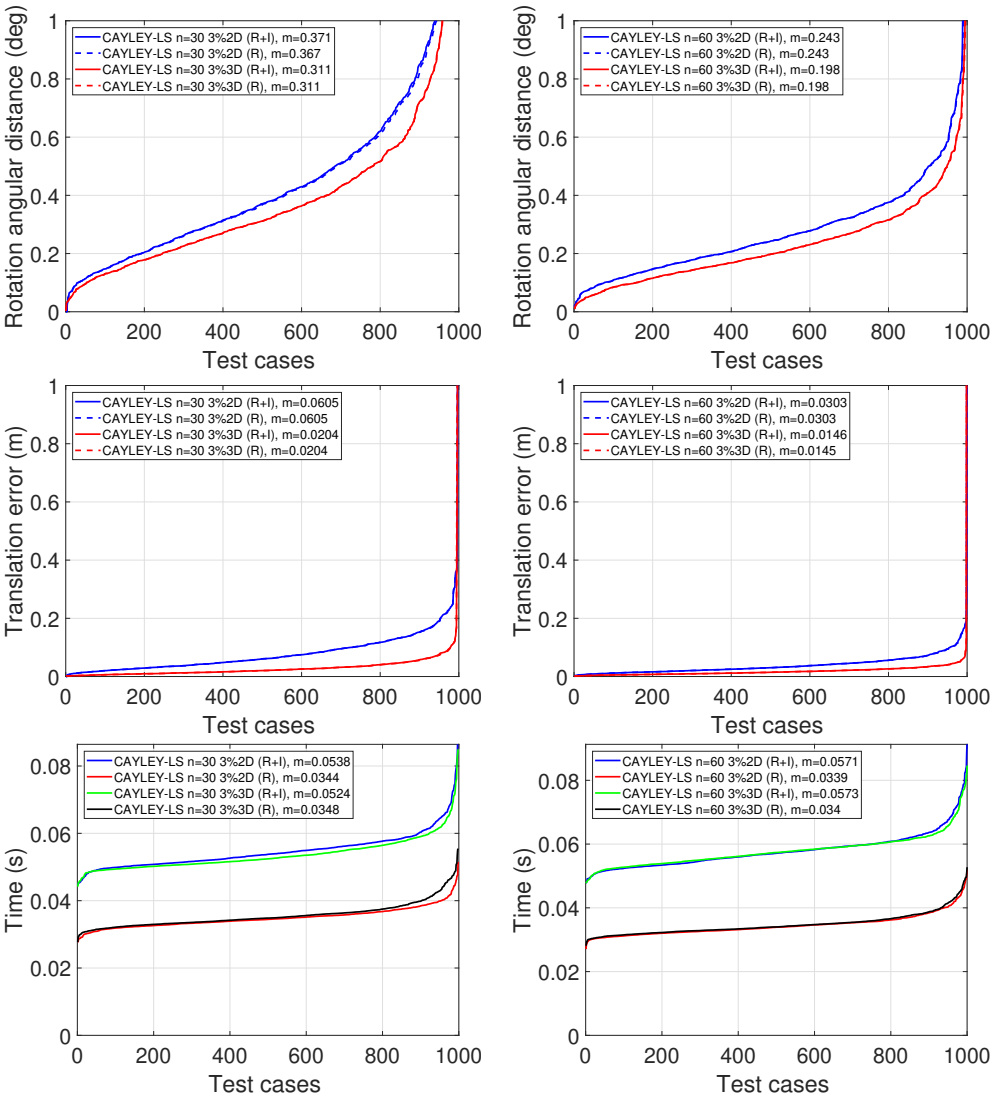


Figure 3.28. *Study of the Cayley-LS when all solutions are returned (R+I: real part of complex solution and real ones) and when only real solutions (R) are returned with 30/60 2D-3D line pairs using 1 camera in terms of median pose error with 3% 2D noise.*

which was mentioned briefly in the previous point. As explained in [80] polyjam (1) Create an instance of a problem with integer coefficients. (2) Call Macaulay2 [53] to compute the Groebner basis for this problem instance, and communicate the form of this basis to polyjam. (3) Multiply the original equations by all monomials up to a certain degree, store the coefficients in a matrix (where each column represents a monomial), and perform Gauss elimination. Repeat this procedure with increasing degree until all required polynomials are found.(4) Remove unnecessary lines in the matrix one by one. (5) Translate the obtained elimination template into C++ code. The Gauss elimination on the large matrix sort of replaces the polynomial combinations in Buchberger's method. The outlined procedure results again in too many polynomials, so the recipe is completed by eliminating unnecessary polynomials one-by-one, each time checking whether Gauss elimination on the remaining matrix still leads to all required final polynomials. The advantage of Kukelova's method is that the recipe simply translates into copying the coefficients into a large matrix and performing Gauss-elimination on the latter, for which there exist numerically stable solvers [80]. The

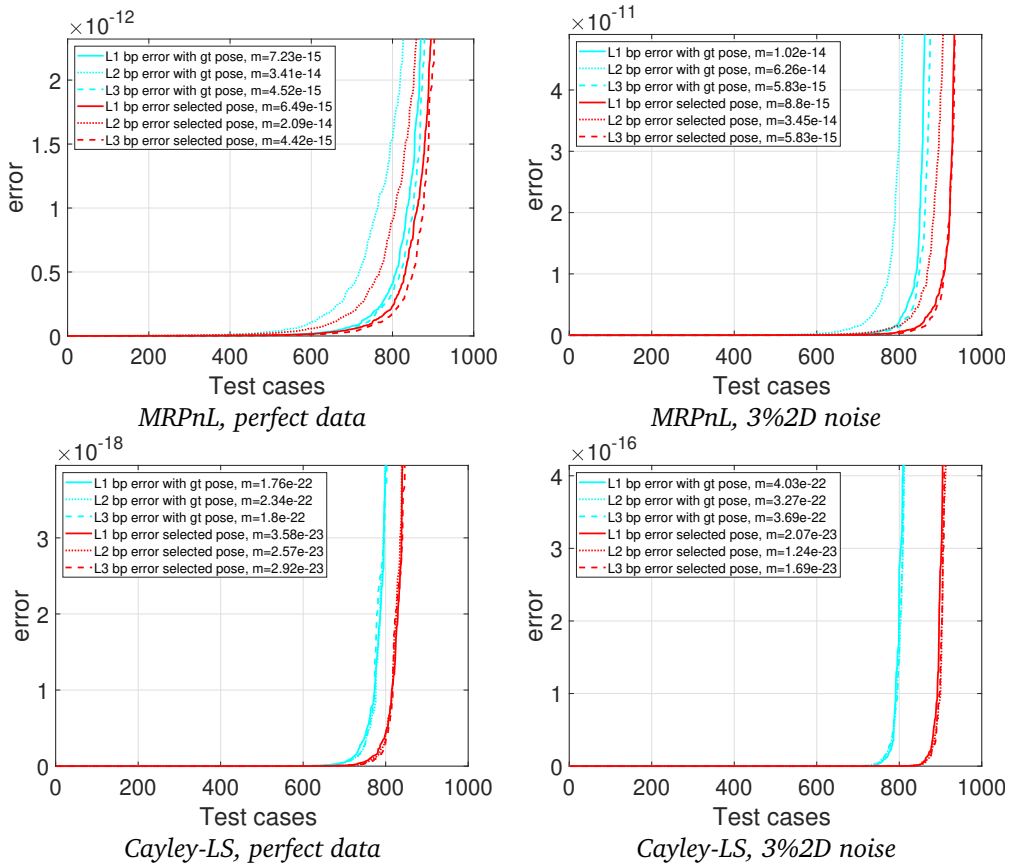


Figure 3.29. *Comparisons of the line back-projection values using the same 3 lines with the selected pose using the line back-projection function (RED) and selected with the GT pose (Turquoise), with perfect**(Left)** and 3%2D noisy data **(Right)**, using MRPnL and Cayley-LS. In the **top** row, the line back-projection presented in Section 3.4.2 is used, and in the **bottom** row, the line back-projection presented in se.BP is used.*

solvers generated with Kneip's solver generator returns real solutions and the real part of the complex solution only if the imaginary part of the eigenvalue decomposition for each solution is below 0.0001 (this is integrated in the C++ code).

Interestingly the difference is only seen in the minimal case where it seems that Kukelova's solver generator produces more robust solvers. No difference is observed in the median of pose in the general case with 60 2D-3D line pairs (see Fig. 3.32 and Fig. 3.33). However, Kneip's solver generator produces up to $10\times$ faster solvers. Note that the testing has been done through C++ MEX(framework to create MATLAB executable for C++ code) on the same MATLAB environment, which means that the C++ solvers generated with Kneip's solver generator are much faster if they are run directly in C++ without any intermediate step.

3.6.5 Solver Analysis Conclusion

In this section, the three proposed solutions have been compared and analyzed to each other, in terms of pose error, number of solutions, line back-projection function, and type of solver generator used. The three solutions can be easily used and suitable within a system of different types of cameras. The two line back-projection functions proved their efficiency. Moreover, the solvers can be integrated within MATLAB or C++ applications. The three solutions have different strengths and weaknesses. Solution presented in Section 3.3 is

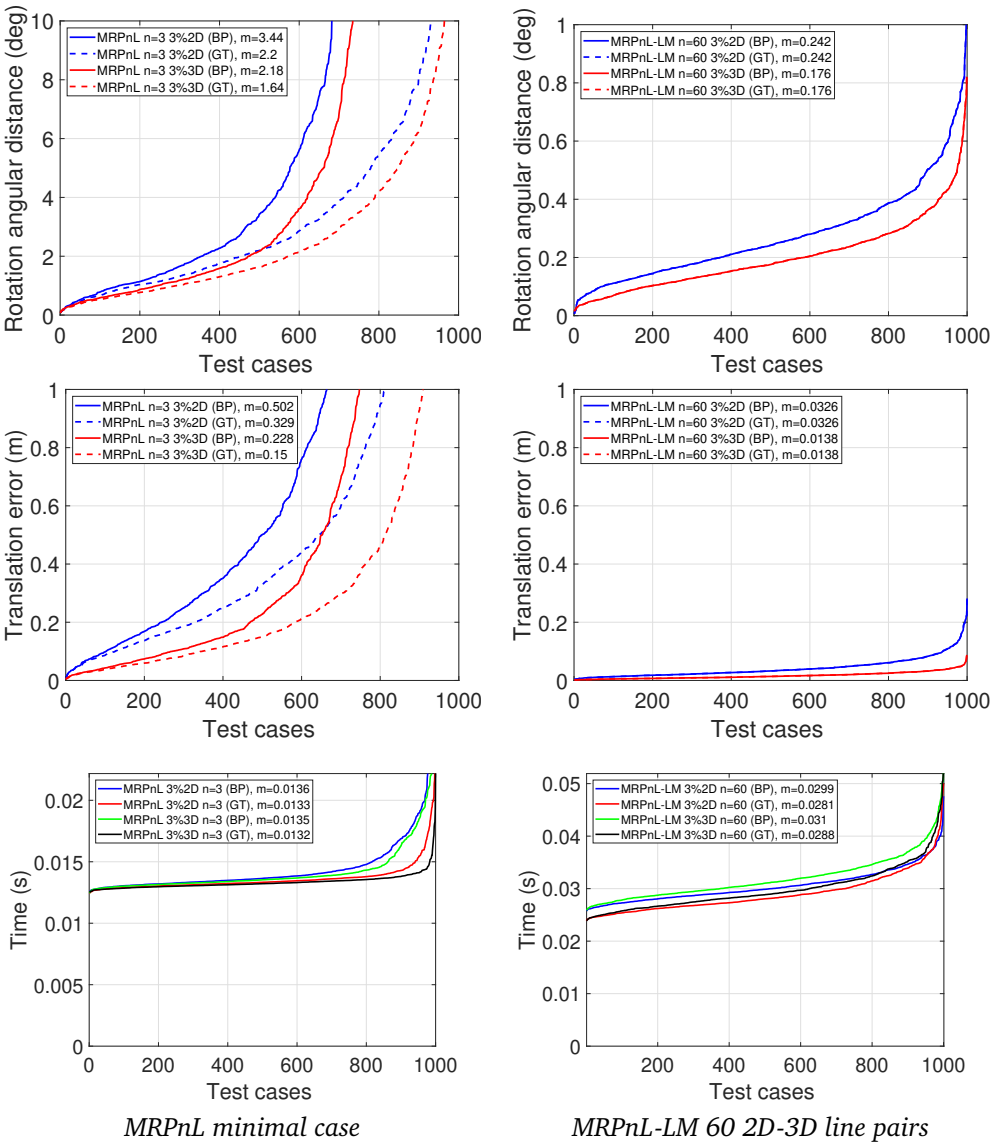


Figure 3.30. *Study of the line back-projection function which works on the normalized image plane using 3/60 2D-3D line pairs. BP: when we select the pose with the line back-projection function; GT: when we select the pose with the GT pose.*

good for a perspective camera system when the vertical direction is known, but it might become slow if more cameras and lines are provided. Solution presented in Section 3.4 is perfectly good for a perspective camera system when no vertical direction is provided, this solution can be used with noisy data and filter outliers in real time, and numerically stable due to the 3D data normalization. Unfortunately, it is less robust when the noise level is high. Finally, the solution presented in Section 3.5 is a robust real time solution which can be used for perspective and omnidirectional cameras, the solution is numerically stable with high noise level and can filter outliers.

3.7 Summary

In this chapter, three novel solutions for the camera pose estimation using 2D-3D line pairs are proposed. The first solution estimates the absolute and relative pose of a multi-view perspective camera system with known vertical direction simultaneously. This first solution

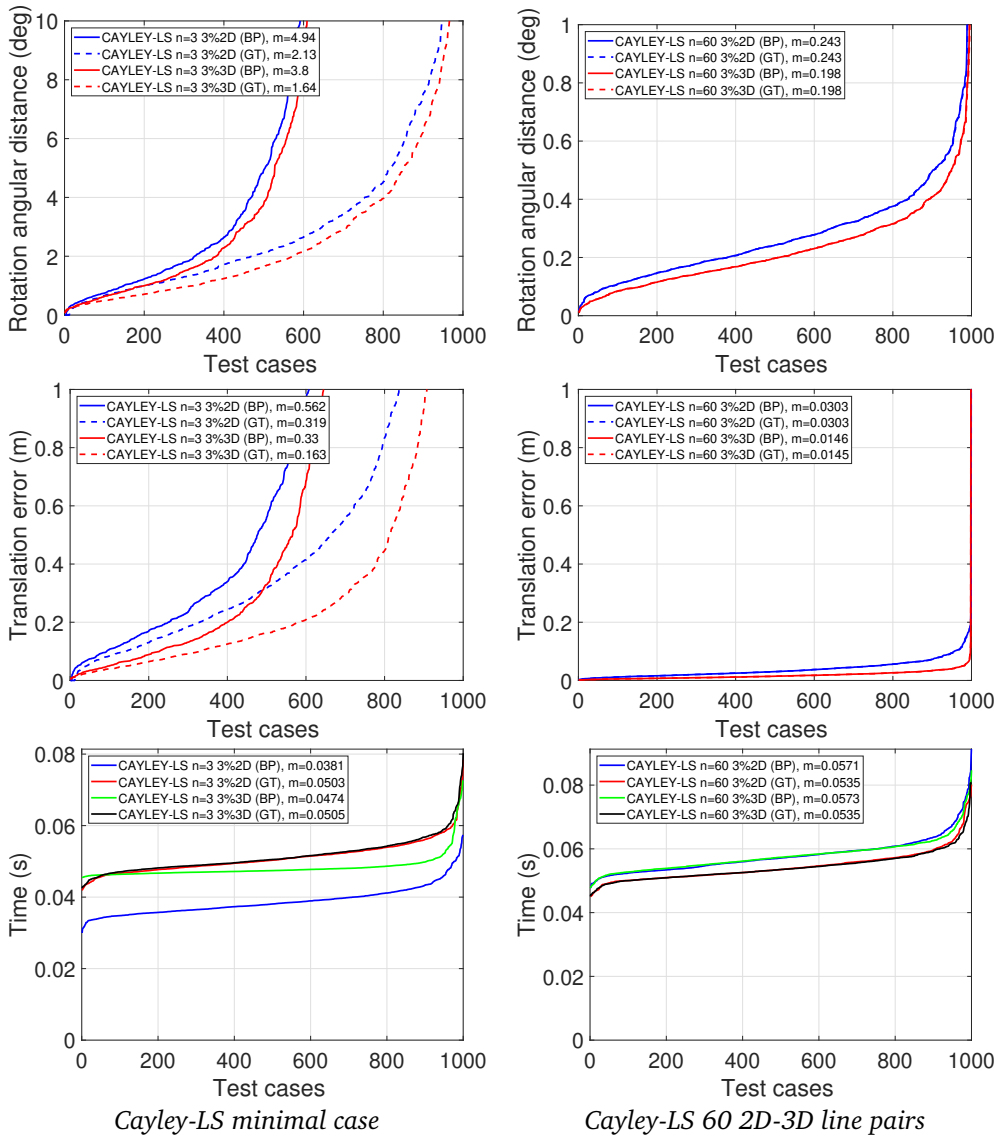


Figure 3.31. *Study of the line back-projection function which works on the unite sphere using 3/60 2D-3D line pairs. BP: when we select the pose with the line back-projection function; GT: when we select the pose with the GT pose.*

consists of a minimal solver (2 cameras and 6 line-pairs), which can be used efficiently within a hypothesis-testing framework like RANSAC [40]. Using such solutions, one can solve for a multi-view system with an arbitrary number of cameras based on a least squares solution for which the initialization is provided by the direct solutions.
Secondly: a novel algebraic approach has been proposed for computing the absolute and relative poses of a multi-view perspective camera system using line correspondences, which works without reformulation both for minimal problems (3 line pairs per camera) as well as for the general $n > 3$ and multiple camera cases. The rotation is solved first through a two-variate 7-th order system of polynomial equations using a Grobner basis solver. Then the translation is solved via a linear system. Experimental tests on large synthetic as well as real datasets confirm the State-of-the-Art performance of the proposed algorithm. Comparative results show that our method outperforms recent alternative methods (AlgLS [118], ASPnL [188], SRPnL [178]) in terms of speed, accuracy, and robustness. Furthermore, unlike these methods, our algorithm works for multi-view scenarios and is robust up to 60%

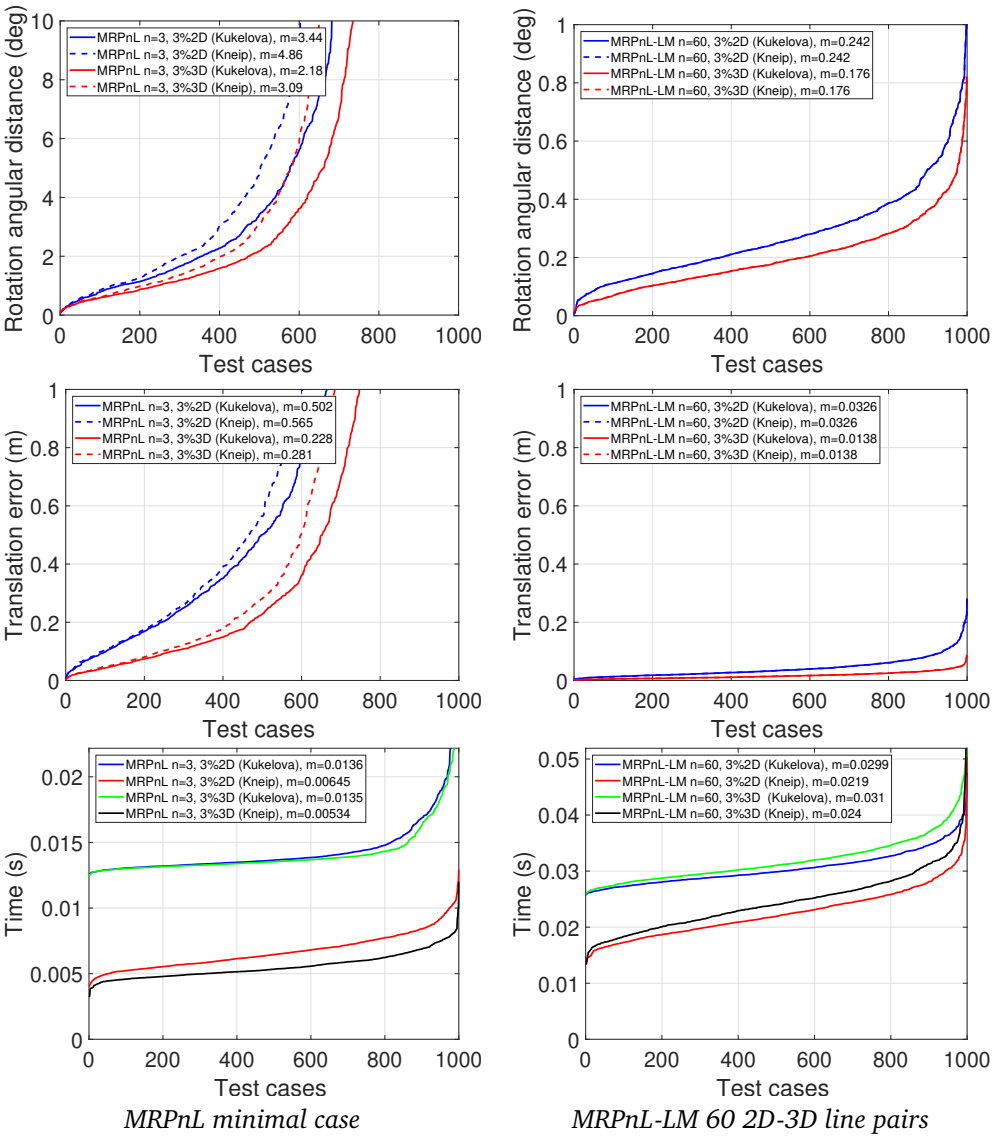


Figure 3.32. *Study of the two different solver generators through the MRPnL solver.*

outlier ratio when combined with a RANSAC-like method.
Lastly: another novel robust pose estimation method for central perspective and omnidirectional cameras using line correspondences. This novel solution uses the Cayley representation of the rotation, which yields a low-order polynomial system both for the minimal as well as for the general n-line case, which can be efficiently solved using Grobner basis solvers. The proposed method is able to deal with outliers as well as noise on the line parameters. It compares favorably to State-of-the-Art methods, being more robust to 3D noise than AlgLS and SRPnL. The efficiency of the proposed solution was validated both on synthetic and real data with omnidirectional and perspective camera setups.
Moreover, these three solvers have been explored and analysed within a unified comparison to identify the various strengths and weaknesses of the methods.

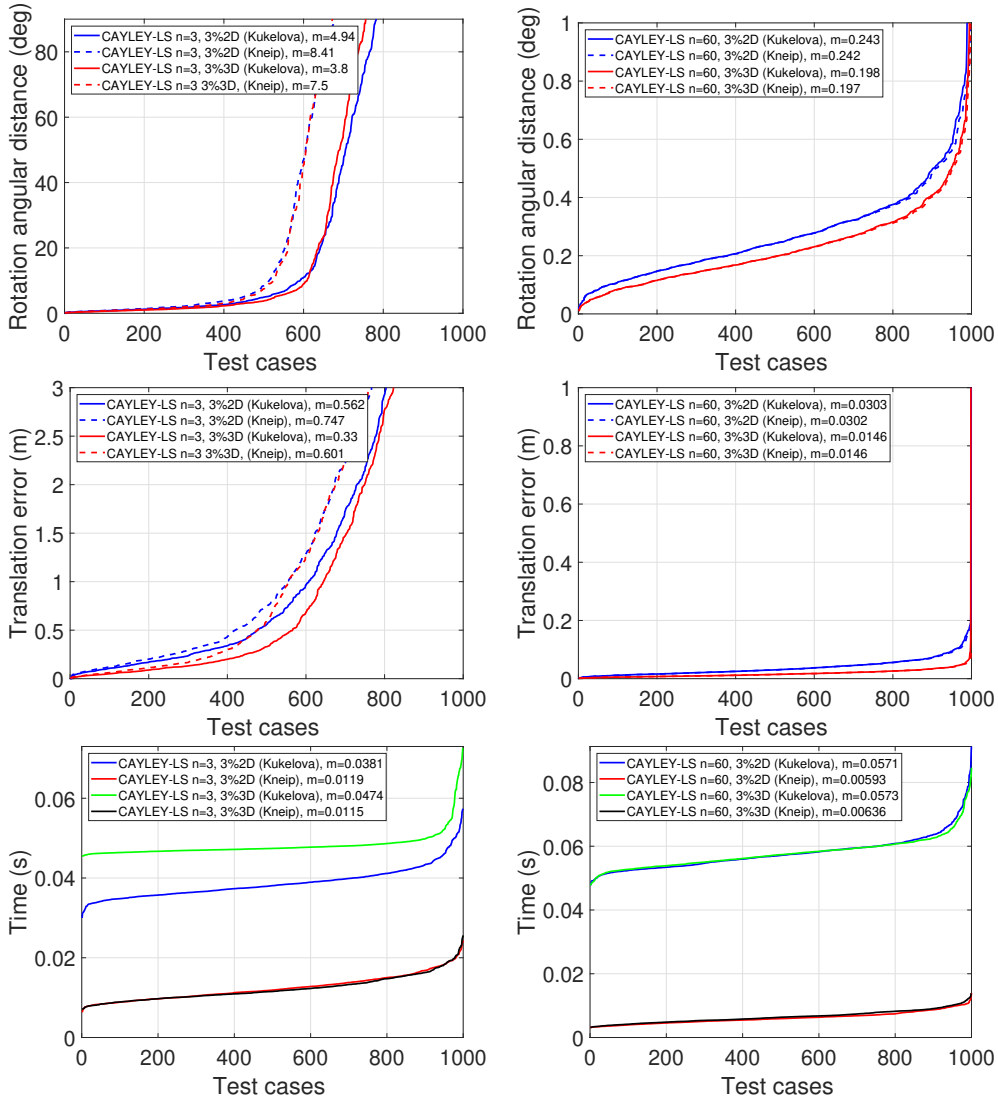


Figure 3.33. *Study of the two different solver generators through the Cayley-LS solver.*

Chapter 4

Learnable Line Detector and Descriptor

4.1 State-of-the-Art Overview

Local features [34, 168] play a fundamental role in almost all fields of computer vision, where matching between images is needed, *e.g.* pose estimation [78, 83, 103, 107], registration, 3D reconstruction, structure from motion, visual localization [148, 149, 150], object recognition, visual odometry and simultaneous localization and mapping, *etc*. Handcrafted keypoint extractors like SIFT [113], BRIEF [26], or ORB [144] are still widely used in spite of numerous alternative end-to-end learning based approaches like R2D2 [143], LIFT [193], MatchNet [57], or DeepCompare [196]. Although learnable features have been a rather active research topic recently, their advantages over handcrafted ones are still not evident [16, 34, 154]. For example, according to [34], extraction of handcrafted descriptors on a CPU is often much faster than extraction of learned descriptors on a GPU; and recent benchmarks targeting their application in image-based reconstruction and localization pipelines suggest that handcrafted features still perform just as well or even better than deep-learned features on such tasks. Among learned keypoint descriptors, L2-Net [164] and its variants became particularly popular. In [119], it was shown that a more powerful descriptor (called HardNet) with significantly simpler learning objective can be learned via an efficient hard negative mining strategy. In [189], a modified robust angular loss function (RAL-Net) was proposed which, coupled with HardNet's negative mining strategy, yield State-of-the-Art performance. A significant amount of work on local features and feature matching (surveys in [16, 34, 154]) was presented in the past 20 years to overcome common challenges like texture-less areas or repetitive patterns typical to a man-made environment found *e.g.* on planar building facades, traffic signs on roads, printed circuits and various other mass-produced objects.

As opposed to the wast amount of work done on extracting and matching keypoints, the field of line descriptor extraction is less active. The clear advantage of line features is a higher robustness to repetitive structures and occlusion. 2D lines are typically detected using the LSD [55] or the faster EDLines detector [9]. Hough transform was also used in different ways trying to extend it to line segment detection [190], but without a significant breakthrough. Recently, promising deep learning based solutions started emerging with the advent of the wireframe parsing [67, 191, 207] approach, where two parallel branches are predicting junction maps and line heatmaps, finally merged into line segments. Wireframes are constructed from the salient structural lines in the scene and their junction points, usually manually labeled for ground truth data [67], and can be parametrized in a multitude of ways; [191] uses a holistic 4D attraction field map, while in contrast, line junctions are

validated by a line verification module in [67]. Most recently [128] proposed SOLD2, a self-supervised solution using synthetic training data in the first stage, then adding homographic augmentation in their training pipeline, outperforming the current State-of-the-Art. Besides the two-step approach of most wireframe related solutions, [68] proposed a compact and fast one-step model using a tri-point line-representation. Recently, semantic line attributes are also used to gain a higher-level representation of lines [100, 155, 165].

Like hand-crafted keypoint descriptors, line segment descriptors can be constructed from the neighborhood appearance of the detected line, without resorting to any other constraints or a priori knowledge. hand-crafted line descriptors include the *Mean Standard deviation Line Descriptor* (MSLD) proposed in [181],which is constructed by the following three steps: (1) For each pixel on the line segment, its pixel support region (PSR) is defined and then the PSR is divided into non-overlapped sub-regions. (2) Line gradient description matrix (GDM) is formed by characterizing each sub-region into a vector. (3) MSLD is built by computing the mean and standard deviation of GDM column vectors. the SIFT-like method, LEHF (*Line-based Eight-directional Histogram Feature*) [62]. A wide baseline stereo line matching method is proposed in [18], which compares the histograms of neighboring color profiles and iteratively eliminates mismatches by a topological filter. In [200], a multi-scale line detection and matching strategy coupled with the Line Band Descriptor (LBD) is proposed which relies on both the local appearance of lines and their geometric attributes, the matching is done by building a relational graph for candidate line matches and then a spectral technique is used to solve this matching problem efficiently. Another approach is SMSLD [173] to add scale-invariance to line segment descriptors using 5 basic rules. In [173], these rules are applied to enhance both the line descriptor of [18] as well as MSLD [181].

Other methods typically rely on some kind of partial 3D information or structural organization of a set of lines to solve matching: Putative line correspondences in [153] relies on both cross-correlation based on gray-level information and the multiple view geometric relations (epipolar geometry and trifocal tensor) between the images. [116] rely on VisualSFM, and use the estimated camera poses to project 2D lines into 3D. Visual localization is cast as an alignment problem of the edges of the query image to a 3D model consisting of line segments in [115] thus avoiding descriptor extraction and matching. They define an efficient Chamfer distance-based aligning cost, incorporated into a first-best-search strategy to register image lines with the model lines. A line matching method in an affine projection space is proposed in [180] to compensate for viewing angle changes in aerial oblique images. First the monocular image orientation is obtained through geometric structures of buildings, then according to the pose information of the camera, an affine projection matrix is obtained. The original image can be rectified as a conformal image based on this projection matrix, thereby reducing the difference in the viewing angle between images. Then, line matching is performed on the rectified images and the inverse affine projection is used to back-project the matched line pairs into the original images. A hierarchical method is proposed in [106], where line segments are matched first in groups and then individually. Group matching relies on Line–Junction–Line (LJL), which consists of two adjacent line segments and their intersecting junction. LJLs are then matched based on a robust descriptor, and unmatched lines are subsequently matched using local homographies of neighboring matched LJLs.

To the best of our knowledge, the first learnable line segment detector and descriptor (SOLD2) has been proposed in [128], which combines line segment detection and then descriptor generation in one single pipeline, although the detector and descriptor branch of the proposed deep network is separated: the detection branch is inspired by the line detection architecture of wireframe detectors while the descriptor is essentially a keypoint descriptor

sampled along the detected line segments. A similar descriptor (but without detection), called LLD has been proposed in [169], however it is specifically designed for automatic 2D-2D matching in a SLAM problem, where image pairs have a short baseline thus avoiding drastic changes in viewpoint and photometric properties. LLD descriptors are constructed on top of a deep yet lightweight fully convolutional neural network inspired by L2Net [164] with triplet loss where triplets are mined from subsequent frames of the KITTI [45] and EuroC datasets. A most recent Line Segment Detector and Descriptor called ELSD proposed by [197] which represent a line segment by 3 parameters, the Center, the Angle, and the Length, and extract their descriptors through a shared feature that is integrated with the detector branch, and the two branches are jointly trained in an end-to-end manner.

4.1.1 Contributions

In Chapter 4 a new *Learnable Line Detector and Descriptor* (L2D2) is developed that is robust enough to detect and match 2D lines across wide view-point changes. The clear advantage of line features over keypoints is that they are commonly found in man-made environments (*e.g.* buildings), have less issues with repetitive structures and -most importantly- do not require a point-wise match, hence potentially they can be used when cameras have very little overlapping views. Our deep convolutional net's architecture consists of two phases:

1. A line segment detector with a lightweight residual network architecture inspired by wireframe networks [67]

2. A patch-based descriptor network inspired by RAL-Net [189] with a rectangular patch size adapted to line-based orientation normalization, yielding 128-dimensional unit feature vectors that can be matched via an angular distance.

For efficient training, we adopt the hard negative mining strategy from [119] combined with the robust angular loss function of RAL-Net [189]. The training data is generated automatically from the Lyft dataset [79], which contains high quality Lidar point clouds and precise ground truth poses for RGB camera images. Experimental results obtained on KITTI [45], KITTI360 [187] and Lyft [79] show State-of-the-Art performance compared to SOLD2 [128], LLD [169], and SMSLD [173]. Our semi-supervised training allows for further improvement using other datasets, but the performance of our descriptor on the KITTI and KITTI360 datasets shows that it already generalizes well to rather different road scenes.

4.2 Learning to Detect Matchable Line Segments

Feature *repeatability* and *reliability* is the key for establishing good matches between image pairs. For keypoints, such an end-to-end network is R2D2 proposed in [143]. Repeatability simply means, that the same keypoint can be detected in all views regardless of the geometric/photometric changes, while reliability (of a descriptor) means, that such keypoints can be reliably matched. For lines, repeatability has a slightly different meaning: any detector (including the proposed one) will detect line segments, but a line correspondence is always interpreted as a matching *infinite* line pair. Therefore the lines defined by the actually detected line segments should be considered as *e.g.* any line-based pose estimation method rely on infinite line pairs [2, 4, 65]. Thus a *repeatable* line is an infinite line, parts of which can be detected on multiple images. This is an important difference w.r.t keypoints or wireframes, where exact position of the point feature or line segment is critical. This higher level aspect is rarely considered in current line detectors, but considering a pose estimation or visual odometry application, detecting multiple line segments along the same infinite line are useless. Thus we argue that detecting less but more relevant line segments is better as this way the descriptor could also be more reliable than matching too many irrelevant lines.

4.2.1 Training Data

An important part of our method is the automatic construction of a large training dataset of matching 2D line segment pairs. For this purpose, a dataset with ground truth camera poses and corresponding 3D point cloud is needed. Herein, the Lyft dataset [79] has been used, which contains several *sequences* of images captured by a fully calibrated multi-view RGB camera system and a dense, high precision metric Lidar. Each of the 150 sequences consists of exactly 126 frames, that were captured in one single burst by the system in 25 seconds. The extraction of matching lines is done automatically using the 3D data, hence our model can be trained in a self supervised way, including other future datasets, when available. The total size of the dataset is 741706 line pairs, which is divided into a training ($\approx 60\%$) and test ($\approx 40\%$) set.

The training data consists of 426496 line pairs. A sequence of non-overlapping images is called a *cluster*. Thus images in a cluster cover the full scene of the dataset without redundancy in the image views. Since the Lyft dataset contains multiple drives along the same trajectories, multiple clusters can be constructed (22 in total), which can be considered as separate scans of the full scene captured at different times. In the process of creating a cluster, overlaps were avoided by calculating the Euclidean distance of each frame's absolute position in a sequence to all the other cameras from different sequences, and keeping images with a distance larger than 27m (the maximum distance between the lanes on the same road).

Figure 4.1. *Example of a 3D fitted line (red) to the merged 3D points coming from two 2D lines on a building edge where we have a discontinuity in the 3D, and only one side is visible from the left view (these 3D points are shown in green), while the right view sees the yellow 3D points corresponding to the 2D detected line on that image. The rest of the point cloud is shown in a gray color. In our method, even if the 3D points are from both sides of a discontinuity, we are still able to robustly fit a 3D line to the* merged *set of 3D points (*i.e. *green and yellow points). This fitted 3D line is shown in red, which is then kept since it fulfills the validation conditions in the proposed dataset construction steps [3].*

To create 2D line pairs, first 2D-3D line correspondences were determined by detecting lines on the images and projecting them into the 3D point cloud (see Fig. 4.1). Using the algorithm outlined below, all the relevant information can be extracted from the data structure, such as what are the 2D views of each 3D line (*i.e.* which camera sees the 3D line), or which 3D lines are visible on a given 2D image. These 2D line pairs are then used for the detector training, while the line support regions (patches) are extracted for all 2D lines corresponding to the same 3D line, yielding a list of all possible patch-correspondences of the 3D line used for the descriptor training. This list is then used to create the batches for training. Note that a batch can contain only one randomly chosen patch pair for each 3D line in order to ensure a correct hard-negative mining within the batch. The dataset construction consists of the following main steps:

1. Detect lines on all 2D images using any line detector and keep only the lines that are at least 48 pixels long (height h of the line support region).

2. For each 2D image, we project 2D lines onto the 3D point cloud using the ground truth pose and filter the 3D points that backproject to a 2D line within a 2 pixels distance (point-to-line distance). A 3D line is then fitted to these points. That gives us the candidate 3D lines for the dataset.

3. Then we cluster 3D line segments that are assumed to be part of the same 3D line using the following procedure for each candidate:

   (a) we calculate the bounding box with ± 20 cm in all directions of the 3D line segment

   (b) for subsequent filtering, consider only 3D lines whose mid point is closer than 1.5m from the mid point of the selected 3D line

   (c) keep lines whose distance is less than 25cm from the mid point of the selected 3D line

   (d) keep lines whose angle is less than 5 degrees w.r.t. the selected 3D line

   (e) the kept lines endpoints are checked if at least one endpoint is inside the bounding box or there is an intersection with the bounding box of the selected 3D line

   (f) the lines that fulfilled the previous constraints are marked as *processed* and we merge their 3D points and do a robust 3D line fitting with RANSAC by using a point to line distance threshold of 3cm.

   (g) the fitted 3D line is backprojected to all the corresponding 2D lines and those are discarded whose line back-projection error [64] is above 0.01. These 2D lines form the visibility group of the fitted 3D line, hence they are all corresponding.

4. After processing all the 3D lines, we obtain all the views with at least two 2D correspondences, for which the 48×32 line support regions are extracted.

4.3 Line Detector Network

The network architecture is inspired by the line detection branch of the Wireframe Parsing [67] method, which in turn was inspired by the Stacked Hourglass network [124]. Based on our observations, the stacked module in the line detection of [67], also used as the backbone of the most recent State-of-the-Art SOLD2 [128] method, is not necessary for *repeatable* line detection, thus the architecture's complexity can be greatly reduced from 20.77M to only 1M parameters, while the line detection performance is similar (see Section 4.6). As shown on Fig. 4.2, its main components are the three Residual Modules R1,R2,R3, followed by deconvolutions to produce a line segment heat map at the input resolution.

For training the line detector network, a mean squared difference loss is applied on each output of the batches of $b = 20$ images:

$$\mathcal{L}_{MSE} = \frac{1}{b} \sum_{i=1}^{b} (\frac{1}{N} \sum_{j=1}^{N} (h_j(I_i) - GT_j(I_i))^2) \tag{4.1}$$

where $h_j(I_i)$ is the j^{th} pixel of the detection heatmap output of the network for image I_i and $GT_j(I_i)$ is the corresponding binary image pixel of the ground truth lines on the image, while N is the number of pixels on the image. Each batch is constructed by starting from a randomly selected 3D line, collecting the camera views that see that line, then adding new lines from the field of view of the selected cameras, and repeating the process until the desired batch size is reached. This way, in each batch seen by the network in one iteration we have different views of the same 3D line, *i.e.* it is guaranteed that the 2D lines used for training are *repeatable*. We used the Stochastic Gradient Descent (SGD) optimizer for 100 epochs, and we set a fixed learning rate of 0.025, and the momentum to be 0.9, weight decay equals to 0.0001.

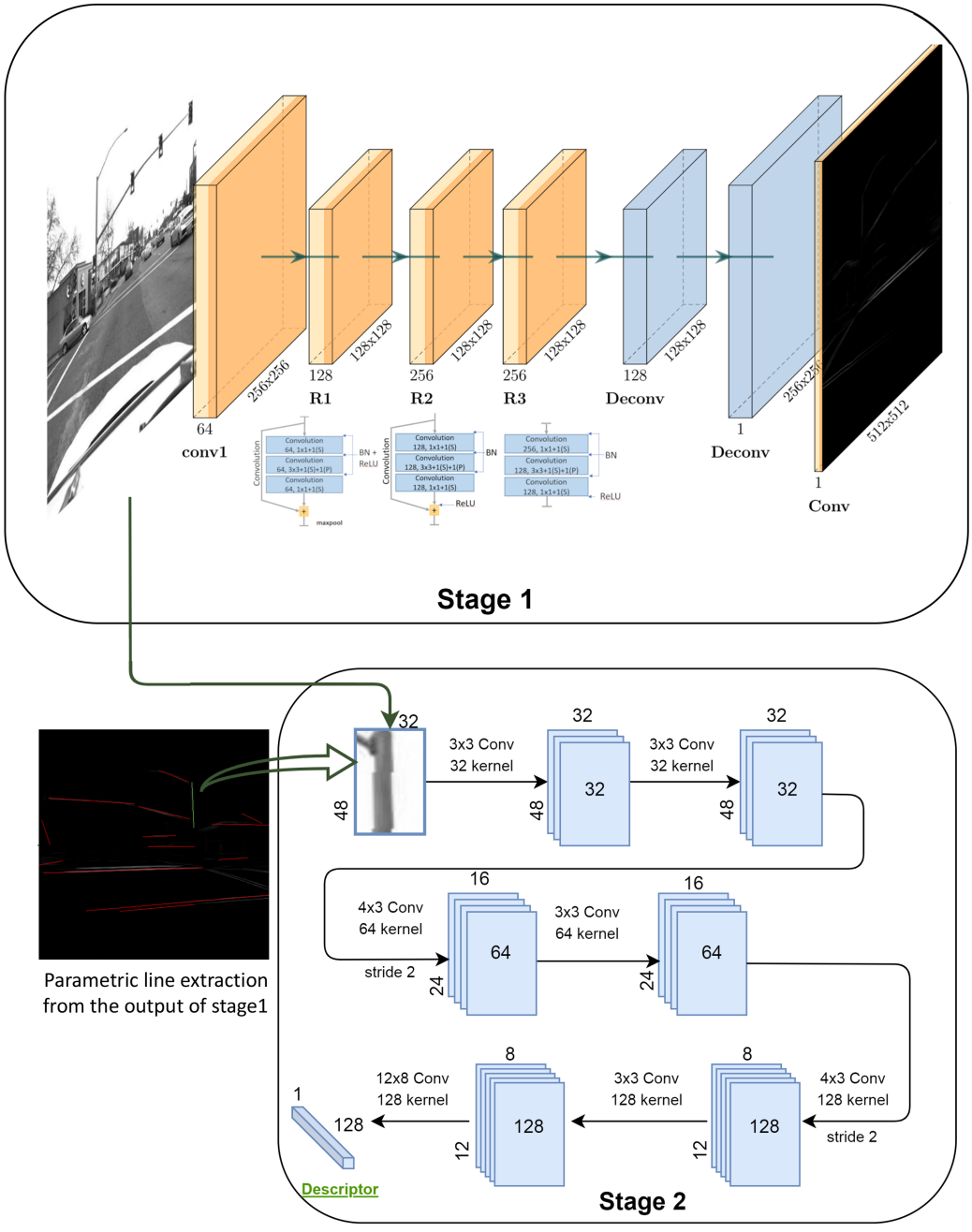


Figure 4.2. *The proposed L2D2 network.* **Stage1***: Detector,* **Stage 2***: Descriptor [3].*

The detection network provides a heatmap, where each pixels intensity represents the probability that it belongs to a line segment. In most applications a parametric representation of the lines is needed, thus in a postprocessing step we find connected components using Hough transform. False detections of multiple versions of the same line due to noisy network output are filtered based on the angle of lines. Note that SOLD2 also uses postprocessing to get the lines, but it is more complex as it has distinct heat maps for line segments and junctions which need to be combined involving a regular sampling of points along each line (between 2 junctions), adaptive local-maximum search, and accepting the lines verifying a minimum average score and inlier ratio.

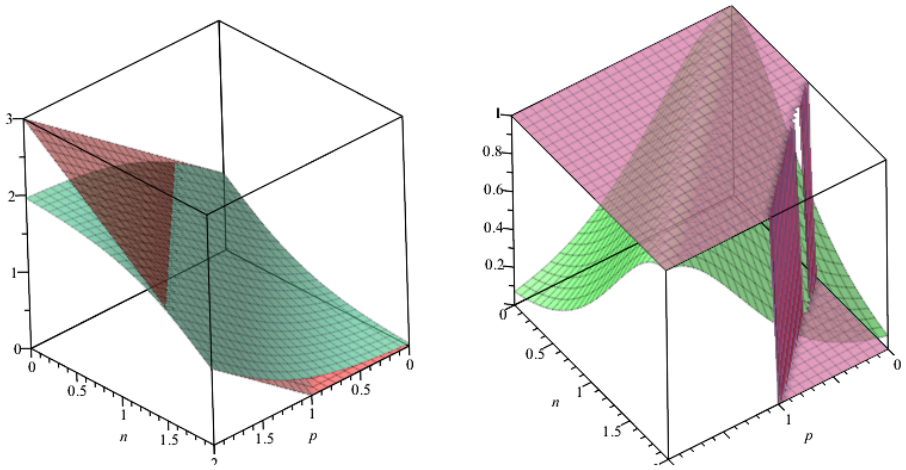
4.4 Line Descriptor Network

Given a set of 2D lines $\{l_i\}_{i=1}^{\ell}$, how to extract descriptors that are sufficiently discriminative and reliable across wide viewpoint changes? Our approach consists of a support region selection mechanism which guarantees a normalized orientation with respect to each line l_i and a deep neural network architecture based on RAL-Net [189] (which adopts the HardNet [119] architecture, which is identical to L2Net [164]). Fig. 4.2 summarizes the layers. The input is the 32×48 pixels line support region with normalized grayscale values (subtracting the mean and dividing by the standard deviation) and the output is an L2 normalized 128D unit length descriptor. The whole feature extraction is built of full convolution layers, downsampling by two-stride convolution. There is a Batch Normalization (BN) [69] layer and a ReLu [122] activation layer in every layer except the last one. To prevent overfitting, there is a 0.3 Dropout layer [158] above the bottom layer as in RAL-Net [189].

4.4.1 Support Region of a Line

The standard strategy for keypoint descriptors is to use a squared window around the detected keypoint and construct (for hand-crafted features like SIFT [113]) or learn (for learned descriptors like L2Net [164]) a feature vector based on the visual pattern within that window. A critical step in these approaches is the coordinate transformation applied to the window before extracting the descriptor such that it will be invariant to local deformations (mainly scale and orientation) of corresponding visual patterns across matched views. A typical solution for hand-crafted features, proposed originally for SIFT [113], is to take the dominant gradient direction of the pattern as the orientation and the keypoint's detection scale as the scale of the local coordinate system. For learned keypoint detectors, *e.g.* in [102, 202], the covariant constraint of the local feature detector is adopted via training of a transformation predictor which also provides a local coordinate frame in which a canonical feature can be extracted.

Since, unlike keypoints, each line l_i has its own length $|l_i|$ and orientation φ_i, a normalization is needed to fix a local coordinate frame for generating the descriptor. In MSLD [181], LBD [200], and DLD [92], a rectangular region centered at the line is used as the local region (hence orientation is normalized w.r.t. φ_i), which is divided into several sub-regions and a SIFT-like descriptor is calculated for each sub-region. The normalization of length $|l_i|$ is achieved by constructing the final descriptor with the mean and standard deviation of these sub-region descriptors. In contrast, LLD [169] samples a few points along the line l_i and for each of these points an L2Net-like keypoint descriptor is extracted and then averaged into a descriptor. Hence for LLD, the orientation normalization is relying on the individual keypoint's orientation, which is acceptable for short baseline images typical in SLAM, but for wide baseline matching, we propose another approach: Since lines have a natural orientation we simply rotate each line by its φ_i into a *vertical* orientation and then we extract a $w \times h$, $w < h$ patch centered at the vertical line. Of course, only lines with $|l_i| > h$ are considered, but the detection accuracy of short line segments is unstable anyway. Note that we do not normalize the length, instead we simply take a sample of length



Loss functions Derivative of loss functions

Figure 4.3. *Loss functions: LLD (4.2) in red, L2D2 (4.3) in green [3].*

h from the beginning of the vertical line (see Fig. 4.2). The reason is that intensity patterns around lines are typically repetitive, or homogeneous with a characteristic change across the line, hence scaling down will not help in characterizing corresponding lines across views as their extracted line segments length might be drastically different due to occlusion and other visibility constraints, hence an equally sampled area along the line tells more then a downscaled version with an uncorrelated scale across views!

4.4.2 Loss function

Triplet loss has been successfully applied in many descriptor learning tasks [119, 169, 189]:

$$\mathcal{L}_3 = [d(\mathbf{a}, \mathbf{p}) - d(\mathbf{a}, \mathbf{n}) + m]_+ \tag{4.2}$$

with $\mathbf{a}, \mathbf{p}, \mathbf{n}$ being an anchor, positive (w.r.t. $\mathbf{a}$), and negative (w.r.t. $\mathbf{a}$) descriptor of the triplet, respectively. $[x]_+$ means a clamping at 0, *i.e.* $\max(x, 0)$, while d is the distance of a descriptor pair. The common difficulty for triplet loss (4.2) is that the choice of the margin m (which is typically set to $m = 1$ [119, 156, 169]) has a great impact on the result [176, 189], since there is always a part of the selected triplets with 0 derivative (see Fig. 4.3). However, as also argued in [189], we can relax the clipping effect of (4.2) by choosing a nonlinear loss function [189]

$$\mathcal{L} = 1 + \tanh(d(\mathbf{a}, \mathbf{p}) - d(\mathbf{a}, \mathbf{n})) \tag{4.3}$$

which has a smooth derivative (see Fig. 4.3)

$$\mathcal{L}' = 1 - \tanh^2(d(\mathbf{a}, \mathbf{p}) - d(\mathbf{a}, \mathbf{n})) \tag{4.4}$$

In this way, positive $(\mathbf{a}, \mathbf{p})$ and $(\mathbf{a}, \mathbf{n})$ negative pairs should be more important when their similarity is high and vice versa giving less weight rather than letting their derivative to be 0. This is important when a considerable amount of false negative matches exists (since we are training in a self-supervised way, this is inevitable). Therefore, our loss puts less weight on triplets when the cosine distance between the negative pair $(\mathbf{a}, \mathbf{n})$ is much higher than for the positive pair $(\mathbf{a}, \mathbf{p})$.

Herein, we follow the HardNet [119] strategy to construct batches: First a matching set $M = \{l_{\mathbf{a}_i}, l^+_{\mathbf{a}_i}\}_{i=1}^N$ of N line pairs is generated, where $l_{\mathbf{a}_i}$ stands for an anchor line and $l^+_{\mathbf{a}_i}$ for its positive pair (*i.e.* they correspond to the same 3D line). M must contain exactly one pair originating from a given 3D line! Then the line support regions are extracted (see

Section 4.4.1) and passed through our L2D2 network. That provides the descriptors $(\mathbf{a}_i, \mathbf{p}_i)$, from which a pairwise $N \times N$ distance matrix $\mathbf{D}$ is calculated such that $\mathbf{D}_{i,j} = d(\mathbf{a}_i, \mathbf{p}_j), i = 1..N, j = 1..N$. While LLD [169] uses L2 norm $d(\mathbf{a}_i, \mathbf{p}_j) = \|\mathbf{a}_i - \mathbf{p}_j\|$ of the descriptor vector differences, following [176, 189], we will use cosine similarity for our metric learning, since our descriptors are unit vectors:

$$d(\mathbf{a}_i, \mathbf{p}_j) = (1 - \mathbf{a}_i \cdot \mathbf{p}_j) = (1 - \cos(\sphericalangle(\mathbf{a}_i, \mathbf{p}_j))) \tag{4.5}$$

Using $\mathbf{D}$, for each matching pair $\mathbf{a}_i$ and $\mathbf{p}_i$, the closest non-matching descriptor $\mathbf{n}_i$ is found by searching the minimum over the off-diagonal elements of the ith row and ith column of $\mathbf{D}$. The following loss is then minimized for each batch:

$$\frac{1}{N}\sum_{i=1}^{N}(1 + \tanh(d(\mathbf{a}_i, \mathbf{p}_i) - d(\mathbf{a}_i, \mathbf{n}_i))) \tag{4.6}$$

The training data is partitioned into 3332 batches of 128 corresponding patch pairs, each batch being created from a cluster. We applied the strategy which trains for 200 epochs with learning rate linearly decreasing to 0 in the end, as in RAL-Net [189]. We choose Stochastic Gradient Descent (SGD) as our optimizer and we set the initial learning rate to be 0.1, and the momentum to be 0.9, dampening equal to 0.9 and weight decay equal to 0.0001.

4.5 Experimental Results

In this section, the different usability of the proposed method are explored. Experimental validation of the proposed L2D2 method was performed on the separate testing data from Lyft [79] (see Section 4.2.1), and to show the generalization capability of the method, the KITTI Visual Odometry [45], and the KITTI360 [187] datasets were used. More precise, the line detector is compared to the State-of-the-Art in terms of *repeatability* of the extracted 2D lines and the matching performance of the descriptor is evaluated in two different settings using cosine similarity metric (4.5): First, a line is matched against a large set of lines, this will show the ability of the descriptors to retrieve a requested information from a large dataset. Second, lines are matched across image pairs, which is a typical scenario for relative pose estimation or 3D reconstruction. Furthermore, the workability of the poposed descriptor is also explored in a pose estimation as well as tracking scenario [1] which is presented at the end of this section. Moreover, The full combination of the detector and descriptor is tested as a full pipeline and compared to State-of-the-Art.

We have to highlight, that LLD was trained on a combined dataset that contained the KITTI sequence 00, and it is mostly optimized for short baseline SLAM-like applications, while the hand-crafted multiscale SMSLD was proposed for wide baseline stereo matching. Both LLD and the proposed method uses the cosine similarity metric (4.5), while SMSLD uses the $L2$ norm of the descriptors difference for matching. Both L2D2 and SOLD2 uses postprocessing to get the lines, but SOLD2's postprocess is more complex as it has distinct heat maps for line segments and junctions which need to be combined involving a regular sampling of points along each line (between 2 junctions), adaptive local-maximum search, and accepting the lines verifying a minimum average score and inlier ratio. SOLD2 uses a syntactically generated dataset and in a second stage the manually annotated wireframe dataset (5000 real images). Our data is parsed from a much larger real dataset ($>$ 10000 images) with multi-view line pairs. Thus our training data is automatically generated from real images whereas SOLD2 relies on synthetic images and manually annotated real images. Note that our validation procedure (also used within the training data generation) is more strict than merely checking the epipolar geometry of 2D line pairs because even if the pro-

[1] the tracking experiment was produced by the BSc co-author Viktor Vilagos as in the published paper [3]

jection planes are the same for a 2D line pair, they can be still images of different 3D lines within the common projection plane, while our procedure strictly checks whether they are the image of the same 3D line.

Navigation in an urban environment, requires repeatable features, hence the proposed line detector is evaluated and compared in terms of *repeatability* of the extracted 2D lines as discussed in Section 4.2. From the KITTI dataset, we randomly select 1257 image-pairs where -according to our ground-truth (GT) generation procedure outlined in Section 4.2.1- at least 1 line pair exists. Note, that EDLines [9] has been used for the GT line detection, hence the selection of this image set is somewhat biased towards EDLines. Then lines are detected on each image using SOLD2 [128] and our L2D2. Each detected line returned by each method is validated using the pointcloud of the test data. Thus detected lines are accepted as correct if it has 2 views (*i.e.* repeatable) and the pointcloud supports it (*i.e.* it corresponds to a true 3D line). Out of the image pairs, $1091/1060/1170$ contains at least 1 such validated line in case of L2D2/SOLD2/EDLines detectors. The average/maximum length of the detected line segments are L2D2:$94/484$, SOLD2:$89/401$ and EDLines:$81/393$. Analyzing this data together with the number of detected and validated line segments as well as the number of distinct infinite lines they correspond to, shown in Table 4.1. We can conclude, that EDLines detects the most repeatable lines across image pairs, but the proposed method tends to detect the longest line segments and has the highest ratio of detected individual *repeatable* infinite lines, that are more useful for real applications like pose estimation (see Section 4.7)

Statistics/Detector	L2D2	SOLD2	EDLines
detected line segments	**73,063**	70,836	66,887
unique infinite detected lines	**59,395**	53,381	44,485
percentage	**81.29%**	75.36%	66.51%
validated line segments	10,685	13,552	**17,771**
unique infinite validated lines	9,785	11,771	**13,762**
percentage	**91.58%**	86.86%	77.44%

Table 4.1. *Detector performance comparison [3].*

Global matching is when a line is matched against a large set of lines corresponding to an entire cluster in Lyft and a whole sequence in KITTI. This case characterizes the performance of the descriptors in applications like localization or SLAM, where matching lines have to be retrieved from a large image dataset. Fig. 4.4 shows comparative results with LLD [169] and SMSLD [173] descriptors. The global matching was performed on a data that contain min:624/avg:4499/max:18947 number of lines. Our method clearly outperforms them: 85% of the true line-pairs ranked in the top 10 out of 4499 possible line matches in average on Lyft, compared to 68%(SMSLD) and 71%(LLD). While on KITTI these percentages are 71%(L2D2), 56%(SMSLD), and 60%(LLD). In spite of the quite different scenes in KITTI compared to our training dataset, our method outperforms the others, which clearly demonstrates its generalization capability. In spite of the quite different scenes in KITTI compared to our training dataset. A clear advantage of the proposed method can be observed over both the hand-crafted and the learned descriptors, which clearly demonstrates its generalization capability. One can also observe that the percentage of correct line matches found in the 2^{nd} to 10^{th} positions is also decreased due to the increased number of lines, since in the KITTI test data matching is performed between 24607 lines, that is 5 times more than the average number in the Lyft clusters' case.

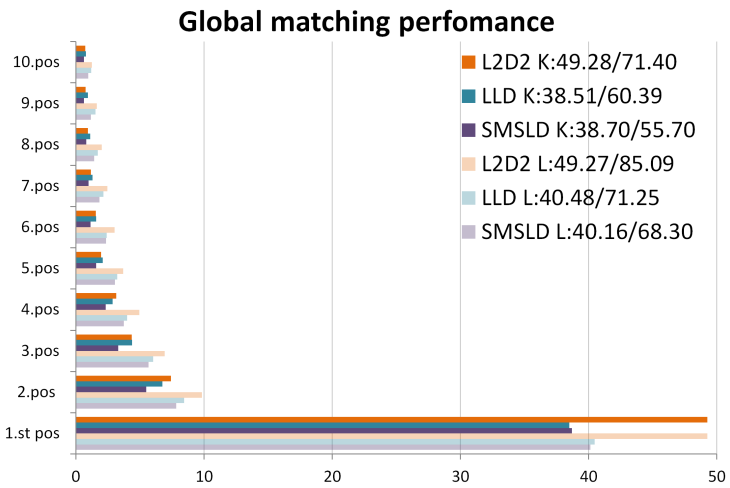


Figure 4.4. *Global matching performance on the Lyft and KITTI testing data. Percentage of pairs that were ranked in the top* 10 *positions are shown. Legend contains for each method the % of lines matched correctly / % of lines found in positions* $(1-10)$ *[3].*

Matching Between Image Pairs is a typical scenario for relative pose estimation or 3D reconstruction. Here, the aim to find all of the true line correspondences (*i.e.* inliers) and avoid wrong matches (*i.e.* outliers). Thus an appropriate filtering of the putative matches is needed, relying on the discriminative power of the descriptor, in order to maximize the inlier/outlier ratio of the returned set of putative line-pairs. On the KITTI dataset, we match a line with the candidate lines detected on another image (in average 44 lines per image) for which a ground truth match is located. The matching is evaluated in terms of the matching score, which is the ratio of the descriptor distances of the best ranked match over the second best match, which is expected to be low for a highly discriminative descriptor and close to 1 for a weakly discriminative matching. Fig. 4.5 shows comparative results on 45000 image pairs. The matching ratio (number of correct matches found over the number of true line-pairs in an image pair) is 79.92% for our method, 74.40% for SMSLD, 73.75% for LLD and 73.05% for SOLD2. The most significant difference between the discriminative power of the methods can be highlighted when the matching ratio is checked at a τ threshold value set to drop maximum 10% of the correct matches. At the expense of losing 10% of the correct matches, as shown in Fig. 4.5 with the horizontal line corresponding to this particular τ threshold, the above mentioned ratios can be improved to 93.17% (L2D2), 87.69% (SMSLD) and 78.27 (LLD). The last plot in Fig. 4.5 shows that almost $1/3$ of the test cases contain more than 30% outliers with LLD, while SMSLD and the proposed method improves this ratio, with L2D2 the resulting test set only contains 16% of the image pairs where there are outliers in the matches. Examples of the detected and matched lines between two image frames are shown in Fig. 4.6, where we can observe how LLD and SMSLD tend to introduce some incorrect matches between similar line segments, while the proposed method matches them correctly. Note that SOLD2 has its own selection mechanism (see Fig. 4.7), hence it returns only the correct matches according to its selection metric [128].

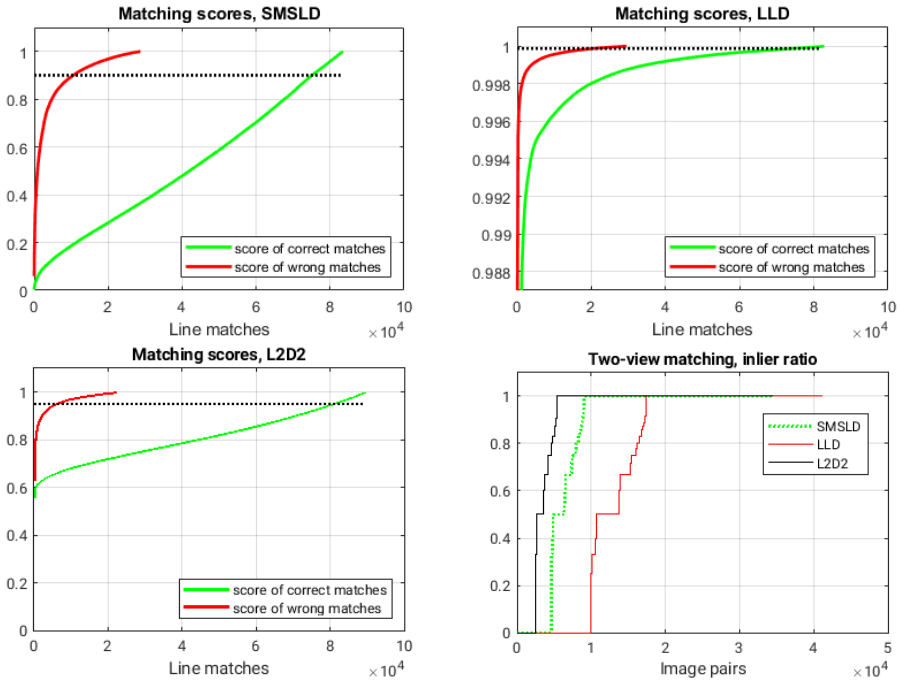


Figure 4.5. *Matching scores of the correct and wrong line pairs on KITTI. The threshold τ is also visualized with a dotted line, at which only 10% of the correct matches are lost. Lastly, the inlier ratio of the image pairs are shown after applying this τ threshold [3].*

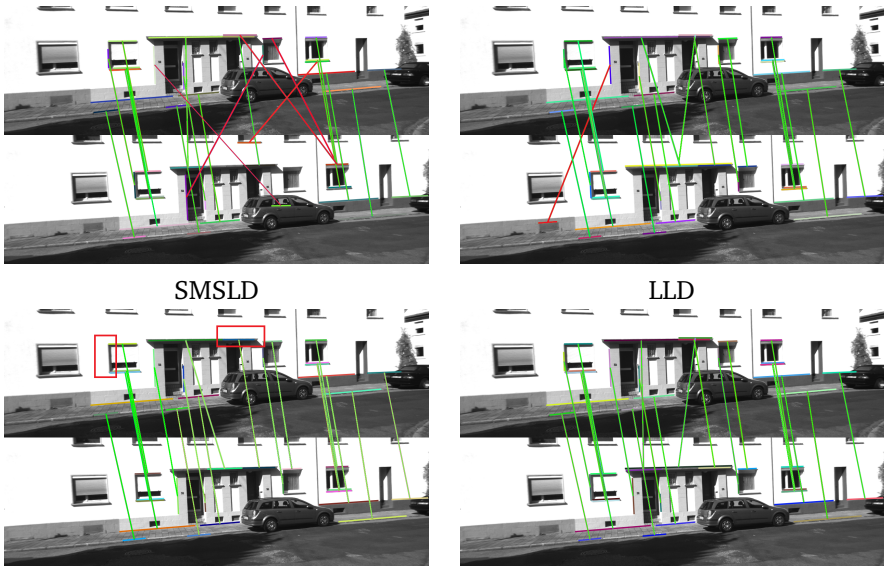
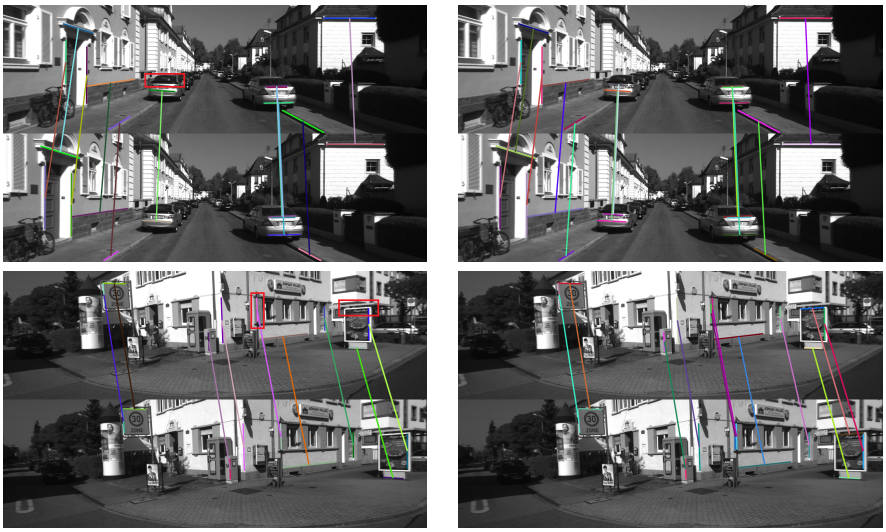


Figure 4.6. *Matching examples on a KITTI image pair. Correct matches are shown with green, wrong matches with red lines. Note: SOLD2 automatically filters bad matches (shown in red boxes) based on its own metrics, while for the other methods we show all putative matches [3].*

Full Detector-Descriptor Pipeline is the real scenario for working with line-correspondences. The detector outputs of SOLD2 [128], EDLines [9], and L2D2 were analyzes previously. Using these 2D lines, we match lines between the image pairs using SOLD2 and L2D2 deep descriptors as well as the hand-crafted SMSLD. Results are summarized in Table 4.2,



SOLD2 **L2D2** (proposed)

Figure 4.7. *Matching 2D line examples(colored) on two KITTI image pairs. where SOLD2 matching algorithm automatically filters out the matches that are considered bad (marked with red rectangle)*

where in the diagonal we can see the performance of the SOLD2, L2D2, and EDLines-SMSLD pipelines, while off-diagonal numbers show the performance of the various detector/descriptor combinations. In each case, the numbers in parentheses represent the number of matched line pairs out of all validated GT line pairs. Our pipeline is the most efficient by successfully matching 84% of detected GT line pairs, while the EDLines/SMSLD pipeline detects the highest number of GT line pairs between image pairs (but note that the random selection is based on EDLines-detected GT line pairs as discussed earlier this section, thus being biased). It is also interesting to note that the lines detected by SOLD2 are the hardest to match for any of the descriptors, it achieves better matching ratio with both EDLines and L2D2 detected lines. Overall, all CNN based descriptors performed better than the hand-crafted SMSLD descriptor, on all sets, but DLD underperformed SOLD2 and the proposed L2D2 network.

Descriptor/Detector	L2D2	SOLD2	EDLines
L2D2	**84.08% (5398/6420)**	**72.11% (5843/8103)**	**84.21% (9760/11590)**
SOLD2	82.49% (5296/6420)	70.15% (5685/8103)	78.04% (9045/11590)
SMSLD	72.35% (4645/6420)	65.36% (5296/8103)	74.78% (8667/11590)
DLD	77.71% (4989/6420)	66.74% (5408/8103)	76.20% (8832/11590)

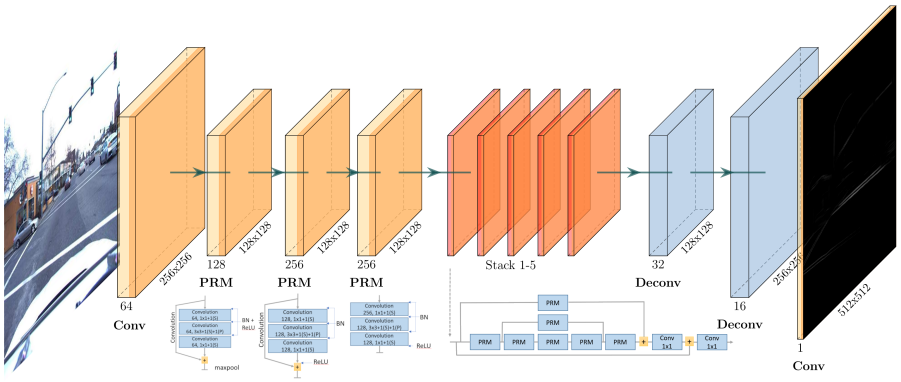
Table 4.2. *Descriptor performance comparison on validated line segments, using the proposed L2D2, SOLD2, EDLines detectors, and L2D2, SOLD2, SMSLD and DLD descriptors [3].*

4.6 Ablation Study

As stated previously, the line detection stage of our L2D2 network is inspired by the Stacked Hourglass network [124], but greatly reducing its complexity by essentially removing the 5 Hourglass blocks. In order to quantitatively demonstrate the effect of this, a training was performed with the same training data using the detection branch of the wireframe network proposed in [67], including the 5 stacks, as shown in Fig. 4.8. The main difference between the architectures is the inclusion of the 5 stacked Hourglass modules in Fig. 4.8, that are each constructed of 7 pyramidal residual modules (PRM) and 2 convolutions (for

more details see [67]), while L2D2 only uses the first 3 PRMs that come before the stack of Hourglass modules.

The training data of 10630 images originally used had to be reduced to 4699 images as training the Hourglass network with our full training data would take 2+ weeks for 100 epochs on an Nvidia GV100 32GB GPU. For a fair comparison, we also reduced the batch size to 1, as the Hourglass network's memory complexity did not allow a higher value. With these settings, we ran two training: 1) with all GT lines on the training images and 2) with the GT lines that has matching pair on another image (*i.e.* using only the *repeatable lines*). Then the evaluation is done on the same dataset presented in the experimental section by checking the number of detected *repeatable* line segments (*i.e.* those lines that have a pair detected in another image). In both cases, L2D2 detects more lines at a much lower computational complexity and a much lower number of parameters (see the table in Fig. 4.8). This clearly shows that reducing the architecture's complexity not only reduces the training and inference time, but helps to learn the concept of repeatable lines in less epochs. We also remark, that increasing the training dataset and the batch size to 20, L2D2's performance becomes higher. Comparing the complexity of the networks, the total number of parameters is reduced from 20.77M to only 1M with L2D2's simplified architecture.



Statistics/Network(trained with)	4699 images, all 285076 lines, b=1		4699 images, 56866 repeatable lines, b=1	
	HG	L2D2	HG	L2D2
validated line segments	10382	11216	1872	5455
unique infinite validated lines	9231	10154	1775	5183
percentage	88.9%	90.53%	94.82%	95.01%
detected line segments	51350	55297	7965	73691
unique infinite detected lines	36308	43721	6994	59099
percentage	70.71%	79.07%	87.81%	80.2%

Figure 4.8. *Ablation study. Top: line detection branch of [67] with 5 Hourglass stacks. Bottom: detection comparison of the Stacked Hourglass network with L2D2 trained on the same data, results are shown as validated/detected line segments [3].*

4.7 Pose Estimation and Tracking

4.7.1 Pose Estimation

As we have seen in the previous section, L2D2 enables the matching of 2D lines between two views with a high inlier ratio. Herein, we will use such matches to establish 2D-3D line correspondences by taking one of the images as a reference image that has 3D lines for each 2D lines (this can be achieved by knowing the camera pose w.r.t. the Lidar sensor as it is a common practice for Lidar scanners). To validate the line matches in a RANSAC absolute pose estimation application, we used the robust method explained in Chapter 3 [2]. Since pose estimation needs a minimum of 3 line-correspondences and lines should not be *e.g.* parallel, we selected image pairs from the 1257 image-pairs described in the matching evaluation step, where at least 6 ground truth line-pairs are available, resulting in 442, 222

and 695 image-pairs for L2D2, SOLD2 and hand-crafted pipelines respectively. Note that our method provides 2 times more image pairs with at least 6 image pairs than SOLD2, while the (somewhat biased) number of image pairs from the EDLines+SMSLD pipeline is the highest. Using L2D2 and SMSLD, detected lines on the reference image are matched with all detected lines on the other image, a unique one-to-one matching is obtained by relying on the matching score to keep the best match for each line, and discard matches with scores above a $\tau = 0.92$ threshold. SOLD2 manages the matching and filtering of bad matches internally. These putative line-correspondences are then fed into the robust pose estimation. The estimated pose has been evaluated in terms of angular distance of the rotation error and percentage of the translation error's relative to the ground truth translation. Fig. 4.9 shows these pose errors for each pipeline on the 85 image pairs on which all of them had at least 6 validated GT line-pairs. We can observe that L2D2 outperforms both methods, solving 94% of the cases with an error of less than $5°$ *and* 5%, which clearly shows that our method detects *repeatable* lines which can be *reliably* matched.

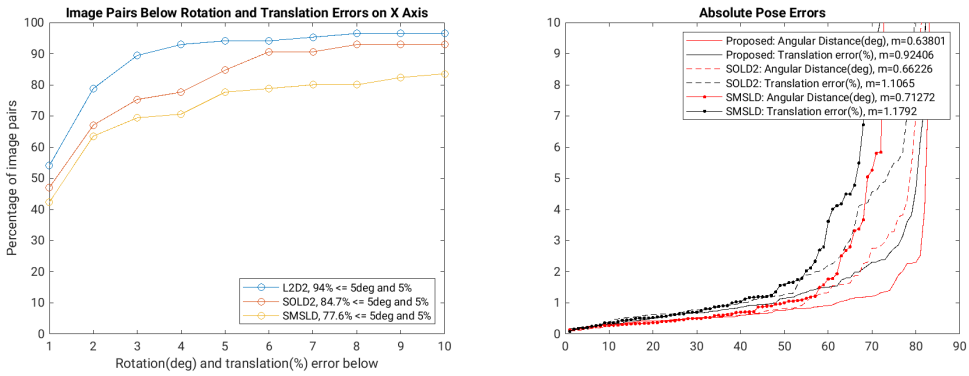


Figure 4.9. *Pose estimation errors using [2] on KITTI image pairs (m stands for median value) [3].*

4.7.2 Pose Tracking

Pose tracking is fundamental for visual odometry and navigation. Herein, the quantitative results obtained using the proposed L2D2 method within a minimalistic Kalman-filter is presented, designed to track a camera's extrinsic parameters. The filter gives a pose prediction for each frame, and receives measurement from the pose estimator presented in Section 3 to update the state of the tracked pose. The line correspondences come from matching 3D lines from previous frames with the detected 2D lines from the current image. Again, repeatability and reliability is critical, since for tracking we need to detect the same 3D line on subsequent frames, which should be reliably matched across neighboring frames. While we make use of the prediction and covariance provided by the Kalman filter to define a bounding box where the corresponding 2D line should be searched, which helps reliable matching once the line is detected. but due to repetitive structures, nearby detected 2D lines typically have similar visual appearance, thus challenging descriptor reliability. The pose tracking application was tested on the KITTI360 [187] dataset, which is recorded in rural areas with challenging narrow, winding streets.

Kalman Filter Tracker

The pose tracking algorithm has 2 main parts. The first is a Kalman filter [25], which provides pose *predictions* and handle inaccurate measurements to provide optimal poses. The second part is the 2D-3D line-based pose estimation using the solver presented in Section 3, which provides pose estimates as *measurements* for the Kalman filter. For each measurement,

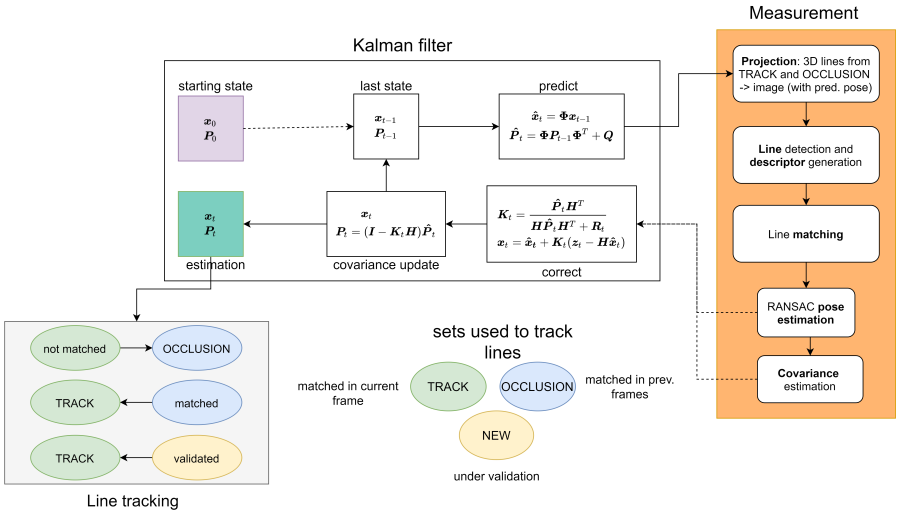


Figure 4.10. *Pose tracking algorithm using a standard linear Kalman filter [3, 25].*

we need to estimate the error of the pose, and use it to provide a *measurement covariance matrix* for the Kalman filter. For every frame, the Kalman filter gives a pose prediction with a covariance based on the previous state of the filter and the defined state-transition model. Using these predictions, 3D lines are matched on the next frame, and a pose is estimated from these line correspondences. The flowchart of the algorithm is shown in Fig. 4.10 where $\boldsymbol{x}$ is the state (translation, rotation), $\boldsymbol{P}$ is the state covariance matrix, $\hat{\boldsymbol{x}}$ is the predicted state, and $\hat{\boldsymbol{P}}$ is the corresponding covariance matrix. $\boldsymbol{\Phi}$ is the state-transition model, and $\boldsymbol{Q}$ is the covariance of the process noise. $\boldsymbol{K}$ is the Kalman gain, which regulates how much we trust in the measurement and the prediction, $\boldsymbol{H}$ is the measurement model which defines the connection between the $\boldsymbol{z}$ measurements and $\boldsymbol{x}$ states. $\boldsymbol{R}$ is the covariance of the measurement noise (which we estimate along with the line-based pose). t is the current time step and $t-1$ is the previous one, $\boldsymbol{x}_0$ and $\boldsymbol{P}_0$ are the starting state and covariance. For further setails see [25]. MATLAB implementation (`trackingKF`) of the Kalman filter is used in the experiments.

The Kalman filter tracks the translation using velocity and constant acceleration, while the orientation is tracked using a constant angular velocity model. Constant in this context means that it's not changing within a prediction, only within a correction step. The measurement vector contains 6 elements, 3 translation and 3 rotation for each axis, where the rotation is parametrized using Euler angles. To find correspondences on new frames, the tracked 3D lines are projected via the predicted pose onto the image, and then they are matched with nearby detected 2D line segments. To find close line segments, bounding boxes are used around the projected 3D lines whose size depends on the covariance of the prediction. To match a 3D line with candidate 2D segments we use use the descriptor of the last matched 2D line segment for each 3D line. These putative line-correspondences are then used to estimate the pose of the new camera image using the robust pose estimator.

To estimate the error of the obtained camera pose, the weighted sum of the following metrics error is used:

1. The *inlier error*

$$e_{inl} = \frac{w^i \cdot \max(\frac{m^i - (\|I\| - 3)}{m^i}, 0) + w^c \cdot \max(\frac{m^c - (c-1)}{m^c}, 0) + w^l \cdot \max(\frac{m^l - (l - 0{,}4)}{m^l}, 0)}{w^i + w^c + w^l} \tag{4.7}$$

where $\|I\|$ is the number of inlier line pairs provided by Section 3, c is the number

Name	weight (w)	max. value	min. value	corr. max. value (m)
num. of inliers	$w^i = 0.3$	25	3	$m^i = 22$
num. of clusters	$w^c = 0.4$	8	1	$m^c = 7$
image coverage	$w^l = 0.3$	25%	0.4%	$m^l = 24.6\%$

Table 4.3. *Metrics used to calculate inlier error. The max. values in were set by analyzing the data, and deciding what values are needed for an accurate pose, while the min. values are the theoretical minimums calculated [3].*

of orientation clusters, and l is the image coverage in %. c is obtained by clustering the line segments based on their orientation, such that lines in a cluster have orientation within $\pm 5^\circ$ of the cluster mean. The image coverage l is caluclated by fitting a bounding boxes to each line and divide the area of their union with the area of the image. $w_{i,c,l}$ and $m_{i,c,l}$ are defined in Table 4.3. The inlier error, e_{inl} is normalized into $(0, 1)$. Obviously, The more inlier, the most varying line orientations and better coverage yields more reliable pose estimates.

2. *line back projection error* is characterized as the median of the inlier line pair errors from [121]:

$$b = \text{median}\left(\frac{d_a + d_b}{2(\|\boldsymbol{p}_b - \boldsymbol{p}_a\|)}\right) \tag{4.8}$$

where d_a and d_b are point-to-line distances, and $\boldsymbol{p}_b$ and $\boldsymbol{p}_a$ are the endpoints of the 2D line segment. We also calculate the ratio h of the high (> 0.015) individual line pair errors in (4.8), which should be low for good poses. The 0.015 threshold was set by analyzing the back projection errors on the dataset. The final back-projection error e_{bp} is then calculated as

$$e_{bp} = \frac{b(1+h)}{m_{bp}} \tag{4.9}$$

e_{bp} is normalized into $(0, 1)$ using m_{bp}, which is a constant determined to make e_{bp} 1 for poses with 10° and 5m of error, which is accurate enough to still be usable. The error, e_{bp} is not strictly under 1, with extremely bad poses, it can be higher. To set an upper limit to the error would not be optimal for the Kalman filter, because the covariance has to scale with the error.

Accurate covariances are important for the Kalman filter in order to handle bad measurements which should be characterized with a high covariance, so that the tracked pose doesn't get influenced by them. To make sure the covariance estimation is good enough, the real error of the pose for a GT dataset is measured, and thoroughly tuned the weight parameters shown in Table 4.3 to make the covariance estimates as close as possible to the true values.

To construct the measurement covariance matrix $\hat{\boldsymbol{R}}$, first the weighted sum of e_{inl} and e_{bp} is calculated

$$\hat{r} = \frac{w_{inl} \cdot e_{inl} + w_{bp} \cdot e_{bp}}{w_{inl} + w_{bp}} \tag{4.10}$$

where $\hat{r} \in (0, 1)$, but the upper limit is also not strict here, like in (4.9). $\hat{\boldsymbol{R}}$ is then constructed as a diagonal covariance matrix:

$$\hat{\boldsymbol{R}} = \hat{r} \cdot \begin{bmatrix} s_t & 0 & 0 & 0 & 0 & 0 \\ 0 & s_t & 0 & 0 & 0 & 0 \\ 0 & 0 & s_t & 0 & 0 & 0 \\ 0 & 0 & 0 & s_r & 0 & 0 \\ 0 & 0 & 0 & 0 & s_r & 0 \\ 0 & 0 & 0 & 0 & 0 & s_r \end{bmatrix} \tag{4.11}$$

where s_t and s_r are scaling constants, that depend on the data. The parameters and con-

detector	EDLines	L2D2
num. of seq.	229	6
num. of frames	6903	114
num. of trackable lines (mean)	6.23	7.39
num. of cluster (mean)	3.85	3.42
num. of seq. on winding streets	88	2
mean length of seq.	30.15 frames	19 frames
mean dist. trav. on seq.	22.52m	12.30m
max length of seq.	82 frames	23 frames
max dist. trav. on seq.	86.91m	19.72m

Table 4.4. *KITTI360 input sequences for tracking [3].*

stant values used in the experiments where all tuned for the KITTI360 data set.

It is important to keep count of the previously matched (*i.e.* trackable) 3D lines, so they can be reused even if they are not detected on the next frame. For this purpose, 3 sets are used, each corresponding to a well defined category: The lines that were matched successfully go into the TRACK set, the ones that were not matched on the current frame, but were matched on one of the previous frames are in the OCCLUSION set, if they are not matched for certain number of frames, they get removed. New 2D-3D line correspondences are validated at the end of every frame if the covariance of the estimated pose is low enough, these go to the NEW set, and get moved to TRACK if they are matched on a subsequent frame, and the back projection error is lower than a threshold. Lines from NEW are also removed if the conditions are not met for a couple frames.

Pose Tracking Results

For the testing, short sequences from the full dataset were selected, on which at least 3 trackable lines on each consecutive frame are validated. The test data consists of 6 such sequences with a total of 114 frames, and 7.39 trackable lines per frame. The lines detected by L2D2, are compared with SMSLD using the proposed line descriptor L2D2. The errors of the estimated poses and the statistics are summarized in Fig. 4.11. We can see, that in spite of matching slightly less lines and having approximately the same number of inliers after RANSAC, L2D2 line correspondences yield consistently better pose estimates with the Kalman filter. Moreover, additional experiment has been done on a larger set of sequences with the hand-crafted EDLines-SMSLD pipeline and the L2D2 descriptor (using the EDLines detected lines for a fair compairoson with SMSLD). The statistics of the input data for both experiments are shown in Table 4.4. The referred clusters in Table 4.4 are describing the variety of orientation of the trackable lines on each frame (see the previous section for more details about clusters). The pose tracking results with EDLines are in Fig. 4.11. Good poses are poses with less than 2° and 2m of errors, bad poses have more than 20° or 20m of errors.

A couple of representative examples with the L2D2 detector and descriptor can be seen in Fig. 4.12 (showing reliable matching of dense, similar repetitive line segments), Fig. 4.13 (showing how projecting an infinite 3D line helps matching long 3D lines across frames), and Fig. 4.14 (showing that a bad prediction is still correctly handled with successfull line matches). Each example has 4 images, all images contain the inlier line pairs. Red line segments are the 2D detected, the green ones are the 3D segments projected onto the image. For each example, the 3D lines are projected with 4 different poses: the predicted, estimated from line correspondences (*i.e.* the *measurement* for the Kalman filter), the Kalman filter output (which is the tracked optimal pose), and the GT pose. The errors of the used poses are also on the titles of the figures.

	L2D2 descriptor	SMSLD
% of good poses	69.3%	59.65%
% of bad poses	3.3%	10.53%
mean of good matches	8.87	10.49
mean of bad matches	3.3	7.17
mean of good RANSAC inliers	5.96	6.93
mean of bad RANSAC inliers	0.92	1.08

	L2D2 descriptor.	SMSLD
% of good poses	48.25%	42.81%
% of bad poses	33.38%	39.89%
mean of good matches	12.02	11.97
mean of bad matches	2.63	4.58
mean of good RANSAC inliers	7.87	7.6
mean of bad RANSAC inliers	0.85	0.75

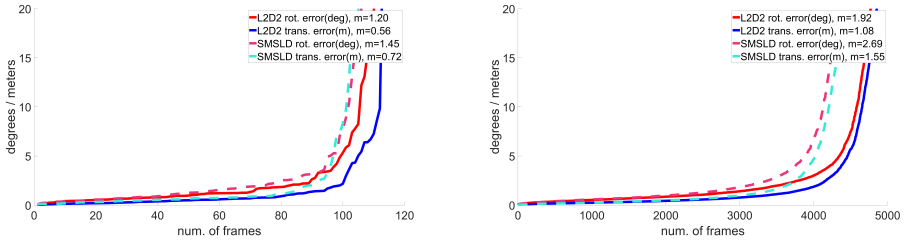


Figure 4.11. *Pose tracking results of the L2D2 vs. SMSLD.(Left) and with EDLines: using L2D2 descriptor vs. SMSLD (right) [3].*

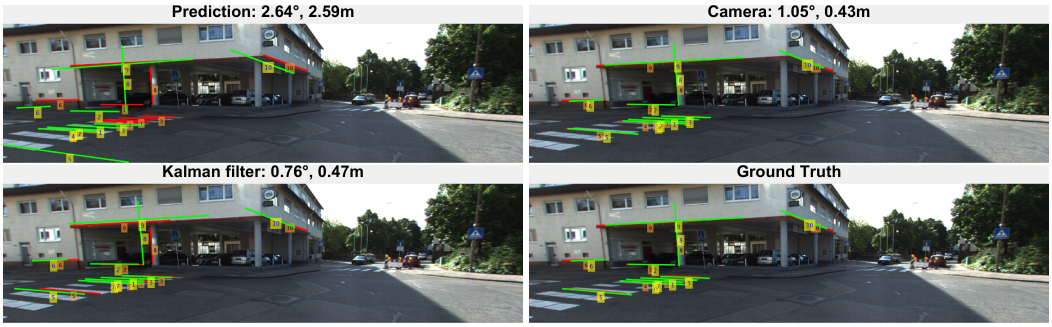


Figure 4.12. *Tracking example 1, the good lines are chosen from very similar 2D lines [3].*

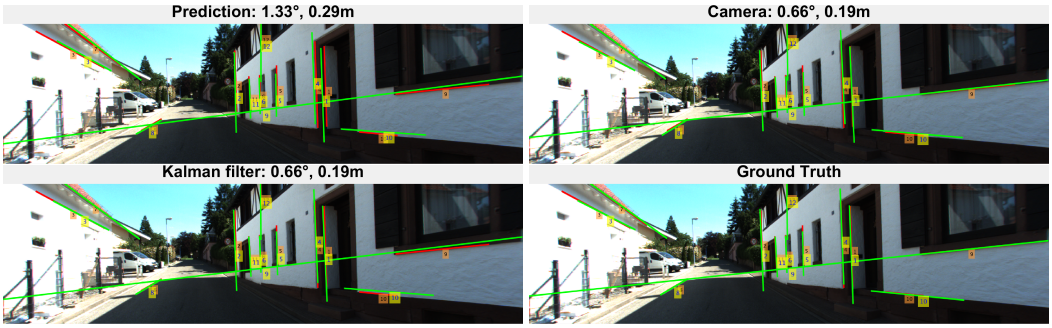


Figure 4.13. *Tracking example 2, 3D line extension yields a correct match on the 3D line num. 9. As a 3D line segment, it would be projected outside of the image [3].*

Figure 4.14. *Tracking example 3, the prediction is very inaccurate, but the matching still works because the bounding box around the lines are resized based on the covariance of the predicted pose [3].*

4.8 Summary

Line detection and line matching is a fundamental problem in image processing. Line detection is an approach to find whether sets of pixels lie on a segment. However, line matching relies on descriptive information within the space of the 2D line that can be used to distinguish similar lines on different images. In this chapter, a full solution for line detection and line matching was presented, composed of a line detector to extract repeatable lines and a robust descriptor to match wide baseline line segments. The descriptor network provides a 128D unit descriptor vector which can be easily matched via a cosine similarity. The training data preparation was fully automatic and can be adapted to other datasets as well. Experimental results confirm the State-of-the-Art performance of the proposed method on three different datasets for autonomous driving both in terms of matching detected lines as well as in terms of inlier/outlier ratio. Furthermore, the detected line pairs were successfully used for line-based camera pose estimation and pose tracking using the robust solver proposed in the previous chapter.

Chapter 5

Conclusions

The research presented in this thesis aims to develop novel solutions for the camera pose estimation problem using 2D-3D line correspondences in an urban environment. These scenarios are rising unlimited challenges that require new robust, and fast solutions. In this thesis, similar complex scenarios are addressed, including camera pose estimation, 2D line detection, and matching. Overall, the experimental results indicate that the proposed solutions are particularly effective for these tasks. In the following, the content of each chapter is summarized and concluded.

In ***Chapter 2***, a presentation about the pose estimation and line detection methodology and all necessary definitions to understand the scope of the thesis are provided. The objective is to introduce, assist, simplify the necessary notions with regards to the studied subject in the thesis. This includes the fundamentals of the camera geometry for perspective and omnidirectional cameras, exploring the projection of a 3D line to an absolute or a relative camera, and camera pose estimation. Moreover, the keys of convolutional neural networks are mentioned.

In ***Chapter 3***, The proposed approaches for the camera pose estimation using 2D-3D line pairs are addressed; the solutions are based on a geometrical observation. This includes, first: a solution to estimate the absolute and relative pose of a multi-view perspective camera system from line correspondences with known vertical direction simultaneously. Second: A novel algebraic approach for computing the absolute and relative poses of a multi-view perspective camera system using line correspondences. Third: a novel robust pose estimation method built with the Cayley representation of the rotation working for central perspective and omnidirectional cameras using line correspondences. Experimental tests on large synthetic as well as real datasets confirm the State-of-the-Art performance. Comparative results show that our methods outperform recent alternative methods and works for multi-view scenarios, and are robust to outliers when combined with a RANSAC-like method.

In ***Chapter 4***, a solution for the following problems is proposed: 2D line segment detection, 2D line matching, and 2D-3D line acquiring. These solutions translate into a robust learnable line detector and descriptor named L2D2, which is intended for wide baseline line matching, and a fully automatic algorithm to build and collect 2D-3D line pairs, which can be used for training. The learnable line detector and descriptor solution is composed of two CNNs. First, a lightweight line detector that detects repeatable lines. Second, a robust descriptor network provides a 128D unit descriptor vector which can be easily matched via cosine similarity. The training data preparation is fully automatic and can be adapted to other datasets as well. The experimental results confirm the State-of-the-Art performance of the proposed method.

Summary in English

Computer vision is understood as the host of techniques to acquire, process, analyze, and understand complex higher-dimensional data from our environment for scientific and technical exploration [70]. Simply computer vision aims to create a model of the real world using cameras for analyzing and understanding it. This discipline became a key technology in many areas, from industrial usage to simple end customers. In recent years, perceptual interfaces [167] have emerged to motivate an increasingly large amount of research within the machine vision community; some of the areas are structure from motion, stereo matching, text recognition, person tracking algorithms, camera calibration, stereo vision, point cloud segmentation, and pose estimation of rigid, articulated, and flexible objects. [159, 184]. Nowadays, the topic of pose estimation in real-time performance is in the center of interest for both the academic and industrial side, especially for autonomous driving in urban environments where pose estimation is needed to navigate in such complex space, it is essential to allow moving devices like a car, drone or a robot with one or multiple cameras to navigate and avoid obstacles. The pose estimation topic is fundamental in various computer vision applications, such as simultaneous localisation, and mapping (SLAM), image-based localization and navigation, augmented reality. This work presents my research on developing solutions for the problem of pose estimation using 2D-3D line pairs and also for the detection/acquisition and matching of such lines in a semi-supervised manner through CNN.

A.1 Key Points of the Thesis

In the following, I summarized my results and highlighted key findings in two main thesis groups. In the first one, I present my findings on the topic of pose estimation using 2D-3D line pairs known as the PnL problem, while in the second one, I present my results on the 2D line detection and 2D line matching topic. In Table A.1, the connections between the thesis points and the corresponding publications are displayed.

I Pose Estimation Using 2D-3D Line Pairs

The basic idea here, is to build up a system of polynomial equations whose solution directly provides the pose. This idea can be extended into a general framework for the pose estimation of a central spherical camera system composed of perspective and omnidirectional cameras. The following points summarize my contribution on the absolute pose estimation topic:

(a) I have proposed a new solution inspiring from [65], on which we don't assume a known relative pose. In the proposed solution [4], I derived the concrete equations, I have constructed the minimal solver and the least squares formulation of the equations for the general case. I have generated the synthetic noisy data, and performed the evaluation both on synthetic and real data. In the second solution called MRPnL [1]. I derived the concrete equations for both minimal and the general case. I experimentally tested, validated and plotted the performance of the two proposed solutions through quantitative and qualitative evaluation on synthetic and real data, respectively, and compared it with the State-of-the-Art methods.

(b) Given a Cayley representation of the rotation, I derived the concrete equations that involve the absolute and relative poses [2]. The solution works on a system of perspective and omnidirectional cameras. I have constructed the direct solvers using the automatic generator of [88] and [80]. I ran and demonstrated the performance of the solution through quantitative evaluations and proved its robustness against noise on the synthetic data and real data. I have also compared the results to the State-of-the-Art methods.

II 2D Line Detection and Matching

This thesis summarizes my contributions with a new line detector and descriptor [3], that is a crucial ingredient of any pose estimation application. First, inspiring from the architecture of the line detection branch of the wireframe parsing [67, 207], I have proposed a lightweight line detection architecture. Second, I have adapted [189] to take a centered rectangular patch from the rotated detected line into a vertical orientation. I have implemented a fully automatic algorithm that can be applied to work with any new dataset to create high-quality training. I trained the full network using training data built by the proposed data generator. I measured the efficiency of the proposed network for line detection and matching performance. I demonstrated the performance of the proposed method against the State-of-the-Art methods.

	I		II
	a	b	
[4]	•		
[1]	•		
[2]	•	•	
[3]			•

Table A.1. *The connection between the thesis points and publications.*

Publications

Refereed Conference Papers (publications used in the thesis)

[4] Hichem Abdellali and Zoltan Kato. Absolute and Relative Pose Estimation of a Multi-View Camera System using 2D-3D Line Pairs and Vertical Direction. In *Proceedings of International Conference on Digital Image Computing: Techniques and Applications*, pages 1–8, Canberra, Australia, December 2018. IEEE

[1] Hichem Abdellali, Robert Frohlich, and Zoltan Kato. A Direct Least-Squares Solution to Multi-View Absolute and Relative Pose from 2D-3D Perspective Line Pairs. In *Proceedings of ICCV Workshop on 3D Reconstruction in the Wild*, pages 2119–2128, Seoul, Korea, October 2019. IEEE

[2] Hichem Abdellali, Robert Frohlich, and Zoltan Kato. Robust Absolute and Relative Pose Estimation of a Central Camera System from 2D-3D Line Correspondences. In *Proceedings of ICCV Workshop on Computer Vision for Road Scene Understanding and Autonomous Driving*, pages 895–904, Seoul, Korea, October 2019. IEEE

[3] Hichem Abdellali, Robert Frohlich, Viktor Vilagos, and Zoltan Kato. L2D2: Learnable Line Detector and Descriptor. In *Proceedings of International Conference on 3D Vision*. IEEE, November 2021, Accepted.

Journal publications

[5] Hichem Abdellali and Zoltan Kato. 3D reconstruction with depth prior using graph-cut. *Central European Journal of Operations Research*, 29(2):387–402, jul 2020

Bibliography

[1] Hichem Abdellali, Robert Frohlich, and Zoltan Kato. A Direct Least-Squares Solution to Multi-View Absolute and Relative Pose from 2D-3D Perspective Line Pairs. In *Proceedings of ICCV Workshop on 3D Reconstruction in the Wild*, pages 2119–2128, Seoul, Korea, October 2019. IEEE.

[2] Hichem Abdellali, Robert Frohlich, and Zoltan Kato. Robust Absolute and Relative Pose Estimation of a Central Camera System from 2D-3D Line Correspondences. In *Proceedings of ICCV Workshop on Computer Vision for Road Scene Understanding and Autonomous Driving*, pages 895–904, Seoul, Korea, October 2019. IEEE.

[3] Hichem Abdellali, Robert Frohlich, Viktor Vilagos, and Zoltan Kato. L2D2: Learnable Line Detector and Descriptor. In *Proceedings of International Conference on 3D Vision*. IEEE, November 2021,.

[4] Hichem Abdellali and Zoltan Kato. Absolute and Relative Pose Estimation of a Multi-View Camera System using 2D-3D Line Pairs and Vertical Direction. In *Proceedings of International Conference on Digital Image Computing: Techniques and Applications*, pages 1–8, Canberra, Australia, December 2018. IEEE.

[5] Hichem Abdellali and Zoltan Kato. 3D reconstruction with depth prior using graph-cut. *Central European Journal of Operations Research*, 29(2):387–402, jul 2020.

[6] Hamid Aghajan and Andrea Cavallaro. *Multi-Camera Networks: Principles and Applications*. Elsevier Science & Techn., 2009.

[7] Hamed Habibi Aghdam and Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks*. Springer International Publishing, 2017.

[8] Mohd Vasim Ahamad, Rashid Ali, Falak Naz, and Sabih Fatima. Simulation of learning logical functions using single-layer perceptron. In *Advances in Intelligent Systems and Computing*, pages 121–133. Springer Singapore, 2020.

[9] Cuneyt Akinlar and Cihan Topal. EDLines: A real-time line segment detector with a false detection control. *Pattern Recognition Letters*, 32(13):1633 – 1642, 2011.

[10] Cenek Albl, Zuzana Kukelova, and Tomás Pajdla. Rolling shutter absolute pose problem with known vertical direction. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 3355–3363, Las Vegas, NV, USA, June 2016.

[11] Editors Of Scientific American. *Understanding Artificial Intelligence*. Grand Central Publishing, 2002.

[12] Sean Armin. The definitive neurological surgery board review. *Journal of Neurosurgery*, 104(2):334–335, feb 2006.

[13] Clemens Arth, Manfred Klopschitz, Gerhard Reitmayr, and Dieter Schmalstieg. Real-time self-localization from panoramic images on mobile devices. In *Proceedings of International Symposium on Mixed and Augmented Reality*, pages 37–46, Basel, Switzerland, October 2011. IEEE Computer Society.

[14] Donald G. Bailey. *Design for Embedded Image Processing on FPGAs*. John Wiley & Sons, 2011.

[15] Simon Baker and Shree K. Nayar. *International Journal of Computer Vision*, 35(2):175–196, 1999.

[16] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 3852–3861, July 2017.

[17] Adrien Bartoli and Peter Sturm. The 3D line motion matrix and alignment of line reconstructions. *International Journal of Computer Vision*, 57(3):159–178, 2004.

[18] H. Bay, V. Ferraris, and L. Van Gool. Wide-baseline stereo matching with line segments. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, volume 1, pages 329–336. IEEE, June 2005.

[19] J. Bermudez-Cameo, G. Lopez-Nicolas, and J. J. Guerrero. Automatic line extraction in uncalibrated omnidirectional cameras with revolution symmetry. *International Journal of Computer Vision*, 114(1):16–37, August 2015.

[20] Dulari Bhatt, Chirag Patel, Hardik Talsania, Jigar Patel, Rasmika Vaghela, Sharnil Pandya, Kirit Modi, and Hemant Ghayvat. CNN variants for computer vision: History, architecture, application, challenges and future scope. 10(20):2470, oct 2021.

[21] Simon Elias Bibri. *The Human Face of Ambient Intelligence*. Springer-Verlag GmbH, 2015.

[22] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[23] J. David Irwin Bogdan M. Wilamowski, editor. *Intelligent Systems*. Taylor & Francis Ltd., 2018.

[24] Adrian Bradski. *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]*. O'Reilly Media, 1. ed. edition, 2008. Gary Bradski and Adrian Kaehler.

[25] Richard A. Brown and Phil Hwang. *Introduction to random signals and applied Kalman filtering (3rd ed.)*. John Wiley and Sons, 1997.

[26] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary robust independent elementary features. In *Proceedings of European Conference on Computer Vision*, pages 778–792. Springer, 2010.

[27] Federico Camposeco, Torsten Sattler, and Marc Pollefeys. Minimal solvers for generalized pose and scale estimation from two rays and one point. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Proceedings of European Conference Computer Vision*, volume 9909 of *Lecture Notes in Computer Science*, pages 202–218, Amsterdam, The Netherlands, October 2016. Springer.

[28] Lucia Cavallaro, Ovidiu Bagdasar, Pasquale De Meo, Giacomo Fiumara, and Antonio Liotta. Artificial neural networks training acceleration through network science strategies. *Soft Computing*, 24(23):17787–17795, sep 2020.

[29] Shouvik Chakraborty and Kalyani Mali, editors. *Applications of Advanced Machine Intelligence in Computer Vision and Object Recognition: Emerging Research and Opportunities*. IGI Global, 2020.

[30] Jorge L. Charco, Boris X. Vintimilla, and Angel D. Sappa. Deep learning based camera pose estimation in multi-view environment. In *nternational Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 224–228. IEEE, nov 2018.

[31] Chiheb Chebbi. *Mastering Machine Learning for Penetration Testing*. Packt Publishing, 2018.

[32] Chu-Song Chen and Wen-Yan Chang. On pose recovery for generalized visual sensors. 26(7):848–861, jul 2004.

[33] Homer Chen. Pose determination from line-to-plane correspondences: Existence condition and closed-form solutions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):530–541, 1991.

[34] Gabriela Csurka, Christopher R Dance, and Martin Humenberger. From handcrafted to deep local features. *arXiv preprint arXiv:1807.10254*, 2018.

[35] Donal O'Shea David A. Cox, John Little. *Using Algebraic Geometry*. Springer-Verlag, 2005.

[36] James J. DiCarlo, Davide Zoccolan, and Nicole C. Rust. How does the brain solve visual object recognition? *Neuron*, 73(3):415–434, feb 2012.

[37] Timothy Duff, Kathlen Kohn, Anton Leykin, and Tomas Pajdla. PLMP - point-line minimal problems in complete multi-view visibility. In *International Conference on Computer Vision (ICCV)*, pages 1675–1684. IEEE, oct 2019.

[38] Ali Elqursh and Ahmed Elgammal. Line-based relative pose estimation. In *Conference on Computer Vision and Pattern Recognition CVPR 2011*, pages 3049–3056. IEEE, jun 2011.

[39] Sachin Sudhakar Farfade, Mohammad J. Saberian, and Li-Jia Li. Multi-view face detection using deep convolutional neural networks. In *In Proceedings of the 5th ACM on International Conference on Multimedia Retrieval—ICMR '15*. ACM, jun 2015.

[40] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[41] Ford Martin Ford. *Architects of Intelligence*. Packt Publishing, 2018.

[42] Robert Frohlich, Levente Tamás, and Zoltan Kato. *Handling Uncertainty and Networked Structure in Robot Control*, volume 42 of *Studies in Systems, Decision and Control*, chapter Homography Estimation Between Omnidirectional Cameras Without Point Correspondences, pages 129–151. Springer, February 2016. Chapter 6.

[43] Ahmed Fawzy Gad. *Practical Computer Vision Applications Using Deep Learning with CNNs*. Springer-Verlag GmbH, 2018.

[44] Jose Garcia-Rodriguez and Miguel A. Cazorla Quevedo, editors. *Robotic Vision: Technologies for Machine Learning and Vision Applications*. IGI Global, 2013.

[45] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2012.

[46] Marcel Geppert, Peidong Liu, Zhaopeng Cui, Marc Pollefeys, and Torsten Sattler. Efficient 2d-3d matching for multi-camera visual localization. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, may 2019.

[47] C. Geyer and K. Daniilidis. A unifying theory for central panoramic systems. In *European Conference on Computer Vision*, pages 445–462, Dublin, Ireland, June 2000.

[48] Christopher Geyer and Kostas Daniilidis. Catadioptric projective geometry. *International Journal of Computer Vision*, 45(3):223–243, 2001.

[49] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Ganapati Mirashi, Chiragkumar Savani, M. Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2d2: Audi autonomous driving dataset. *ArXiv*, abs/2004.06320, 2020.

[50] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323, 2011.

[51] Xuanrui Gong, Yaowen Lv, Xiping Xu, Yuxuan Wang, and Mengdi Li. Pose estimation of omnidirectional camera with improved EPnP algorithm. *Sensors*, 21(12):4008, jun 2021.

[52] Daniel Graupe. *Deep Learning Neural Networks*. WORLD SCIENTIFIC, jun 2016.

[53] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at `http://www.math.uiuc.edu/Macaulay2/`.

[54] Josh Gregory. *Artificial Intelligence*. CHERRY LAKE PUB, 2017.

[55] R. Grompone von Gioi, J. Jakubowicz, J. Morel, and G. Randall. LSD: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, April 2010.

[56] J. A. GRUNERT. Das pothenotische problem in erweiterter gestalt nebst bber seine anwendungen in der geodasie. *Grunerts Archiv fur Mathematik und Physik*, pages 238–248, 1841.

[57] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C. Berg. MatchNet: Unifying feature and metric learning for patch-based matching. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 3279–3286, 2015.

[58] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, 2004.

[59] Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. Rotation averaging. *International Journal of Computer Vision*, 103(3):267–305, July 2013.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, jun 2016.

[61] Joel A. Hesch and Stergios I. Roumeliotis. A direct least-squares (dls) method for pnp. In *2011 International Conference on Computer Vision, ICCV 2011*, Proceedings of the IEEE International Conference on Computer Vision, pages 383–390, December 2011. 2011 IEEE International Conference on Computer Vision, ICCV 2011 ; Conference date: 06-11-2011 Through 13-11-2011.

[62] Keisuke Hirose and Hideo Saito. Fast line description for line-based SLAM. In *Proceedings of the British Machine Vision Conference*. BMVA, 2012.

[63] Craig Hoffman and Ronald Driggers, editors. *Encyclopedia of Optical and Photonic Engineering, Second Edition*. CRC Press, sep 2015.

[64] Nora Horanyi and Zoltan Kato. Generalized pose estimation from line correspondences with known vertical direction. In *Proceedings of International Conference on 3D Vision*, pages 1–10, Qingdao, China, October 2017. IEEE.

[65] Nora Horanyi and Zoltan Kato. Multiview absolute pose using 3D - 2D perspective line correspondences and vertical direction. In *Proceedings of ICCV Workshop on Multiview Relationships in 3D Data*, pages 1–9, Venice, Italy, October 2017. IEEE.

[66] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset. `https://level-5.global/level5/data/`, 2020.

[67] Kun Huang, Yifan Wang, Zihan Zhou, Tianjiao Ding, Shenghua Gao, and Yi Ma. Learning to parse wireframes in images of man-made environments. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 626–635, 2018.

[68] Siyu Huang, Fangbo Qin, Pengfei Xiong, Ning Ding, Yijia He, and Xiao Liu. Tp-lsd: Tri-points based line segment detector. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Proceedings of European Conference Computer Vision*, pages 770–785. Springer International Publishing, 2020.

[69] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of International Conference on Machine Learning*, page 448–456. JMLR.org, 2015.

[70] Bernd Jähne. *Handbook of computer vision and applications*. Academic Press, San Diego, 1999.

[71] Carlos Jaramillo, Liang Yang, J. Pablo Muñoz, Yuichi Taguchi, and Jizhong Xiao. Visual odometry with a single-camera stereo omnidirectional system. *Machine Vision and Applications*, 30(7-8):1145–1155, jul 2019.

[72] Christopher Jekeli. *Inertial Navigation Systems with Geodetic Applications*. DE GRUYTER, dec 2001.

[73] Christopher Jekeli. *Inertial Navigation Systems with Geodetic Applications*. Gruyter, Walter de GmbH, 2012.

[74] Li Jinyu, Yang Bangbang, Chen Danpeng, Wang Nan, Zhang Guofeng, and Bao Hujun. Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality. *Virtual Reality & Intelligent Hardware*, 1(4):386–410, aug 2019.

[75] Marc Jorda, Pedro Valero-Lara, and Antonio J. Pena. Performance evaluation of cuDNN convolution algorithms on NVIDIA volta GPUs. 7:70461–70473, 2019.

[76] Mahzad Kalantari, Amir Hashemi, Franck Jung, and Jean-Pierre Guedon. A new solution to the relative orientation problem using only 3 points and the vertical direction. *Journal of Mathematical Imaging and Vision*, 39(3):259–268, nov 2010.

[77] Sagi Katz, Ayellet Tal, and Ronen Basri. Direct visibility of point sets. In *ACM Transactions on Graphics*, volume 26 of *SIGGRAPH '07*, pages 24–es, New York, NY, USA, July 2007. ACM.

[78] Tong Ke and Stergios I. Roumeliotis. An efficient algebraic solution to the perspective-three-point problem. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4618–4626. IEEE, jul 2017.

[79] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft level 5 av dataset 2019. urlhttps://level5.lyft.com/dataset/, 2019.

[80] Laurent Kneip. *Polyjam*, 2015 [online]. https://github.com/laurentkneip/polyjam.

[81] Laurent Kneip and Paul Furgale. OpenGV: A unified and generalized approach to real-time calibrated geometric vision. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2014.

[82] Laurent Kneip, Paul Timothy Furgale, and Roland Y. Siegwart. Using multi-camera systems in robotics: Efficient solutions to the npnp problem. *2013 IEEE International Conference on Robotics and Automation*, pages 3770–3776, 2013.

[83] Laurent Kneip, Hongdong Li, and Yongduek Seo. UPnP: an optimal O(n) solution to the absolute pose problem with universal applicability. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Proceedings of European Conference Computer Vision, Part I*, volume 8689 of *Lecture Notes in Computer Science*, pages 127–142, Zurich, Switzerland, September 2014. Springer.

[84] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *CVPR 2011*, pages 2969–2976, 2011.

[85] Moritz Knorr. *Self-Calibration of Multi-Camera Systems for Vehicle Surround Sensing*. Saint Philip Street Press, 2020.

[86] David Kopec. *Classic Computer Science Problems in Java*. Simon + Schuster Inc., 2020.

[87] Miroslav Kubat. *An Introduction to Machine Learning*. Springer International Publishing, 2021.

[88] Zuzana Kukelova, Martin Bujnak, and Tomas Pajdla. Automatic generator of minimal problem solvers. In *Proceedings of European Conference on Computer Vision*, pages 302–315. Springer Berlin Heidelberg, 2008.

[89] Zuzana Kukelova, Martin Bujnak, and Tomáš Pajdla. Closed-form solutions to minimal absolute pose problems with known vertical direction. In Ron Kimmel, Reinhard Klette, and Akihiro Sugimoto, editors, *Proceedings of Asian Conference on Computer Vision, Part II*, volume 6493 of *LNCS*, pages 216–229, Queenstown, New Zealand, November 2010. Springer.

[90] Kye-Si Kwon and Steven Ready, editors. *Practical Guide to Machine Vision Software*. Wiley-VCH Verlag GmbH & Co. KGaA, dec 2014.

[91] Woong Kwon, Jun Ho Park, Minsu Lee, Jongbeom Her, Sang-Hyeon Kim, and Ja-Won Seo. Robust autonomous navigation of unmanned aerial vehicles (UAVs) for warehouses' inventory application. *IEEE Robotics and Automation Letters*, 5(1):243–249, jan 2020.

[92] Manuel Lange, Fabian Schweinfurth, and Andreas Schilling. Dld: A deep learning based line descriptor for line feature matching. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 5910–5915, 2019.

[93] Viktor Larsson, Kalle Astrom, and Magnus Oskarsson. Efficient solvers for minimal problems by syzygy-based reduction. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2383–2392. IEEE, jul 2017.

[94] Viktor Larsson, Magnus Oskarsson, Kalle Åström, Alge Wallis, Zuzana Kukelova, and Tomás Pajdla. Beyond grobner bases: Basis selection for minimal solvers. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 3945–3954, Salt Lake City, UT, USA, June 2018. IEEE Computer Society.

[95] Louis Lecrosnier, Remi Boutteau, Pascal Vasseur, Xavier Savatier, and Friedrich Fraundorfer. Camera pose estimation based on PnL with a known vertical direction. *Robotics and Automation Letters*, 4(4):3852–3859, oct 2019.

[96] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. 1(4):541–551, dec 1989.

[97] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[98] Gim Hee Lee. A minimal solution for non-perspective pose estimation from line correspondences. In *Proceedings of European Conference on Computer Vision*, pages 170–185, Amsterdam, The Netherlands, October 2016. Springer.

[99] Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Motion estimation for self-driving cars with a generalized camera. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 2746–2753, Portland, OR, USA, June 2013.

[100] Jun-Tae Lee, Han-Ul Kim, Chul Lee, and Chang-Su Kim. Semantic line detection and its applications. In *Proceedings of International Conference on Computer Vision*, pages 3249–3257, 2017.

[101] Raymond S. T. Lee. *Artificial Intelligence in Daily Life*. Springer Singapore, 2020.

[102] Karel Lenc and Andrea Vedaldi. Learning covariant feature detectors. In Gang Hua and Hervé Jégou, editors, *Proceedings of ECCV Workshops*, pages 100–117, Amsterdam, Netherlands, 2016. Springer.

[103] Vincent Lepetit, Francesc Moreno-Noguer, and P. Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81:155–166, 2009.

[104] Haoang Li, Wen Chen, Ji Zhao, Jean-Charles Bazin, Lei Luo, Zhe Liu, and Yun-Hui Liu. Robust and efficient estimation of absolute camera pose for monocular visual odometry. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2675–2681. IEEE, may 2020.

[105] Haoang Li, Jian Yao, Jean-Charles Bazin, Xiaohu Lu, Yazhou Xing, and Kang Liu. A monocular SLAM system leveraging structural regularity in manhattan world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2518–2525. IEEE, may 2018.

[106] Kai Li, Jian Yao, Xiaohu Lu, Li Li, and Zhichao Zhang. Hierarchical line matching based on line–junction–line structure descriptor and local homography estimation. *Neurocomputing*, 184:207 – 220, 2016. RoLoD: Robust Local Descriptors for Computer Vision 2014.

[107] S. Li, C. Xu, and M. Xie. A robust O(n) solution to the perspective-n-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1444–1450, 2012.

[108] Julien Li-Chee-Ming and Costas Armenakis. UAV navigation system using line-based sensor pose estimation. 21(1):2–11, jan 2018.

[109] Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *arXiv.org*, 2109.13410, 2021.

[110] S. Linnainmaa, D. Harwood, and L.S. Davis. Pose determination of a three-dimensional object using triangle pairs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):634–647, 1988.

[111] Ruyu Liu, Jianhua Zhang, Kejie Yin, Zhiyin Pan, Ruihao Lin, and Shengyong Chen. Absolute orientation and localization estimation from an omnidirectional image. In *Lecture Notes in Computer Science*, pages 309–316. Springer International Publishing, 2018.

[112] Yinlong Liu, Guang Chen, and Alois Knoll. Globally optimal camera orientation estimation from line correspondences by BnB algorithm. 6(1):215–222, jan 2021.

[113] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[114] I.R. Management Association. *Deep Learning and Neural Networks: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2020.

[115] B. Micusik and H. Wildenauer. Descriptor free visual indoor localization with line segments. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 3165–3173. IEEE, June 2015.

[116] Pedro Miraldo, Tiago Dias, and Srikumar Ramalingam. A minimal closed-form solution for multi-perspective pose estimation using points and lines. In *The European Conference on Computer Vision*, pages 1–17, Munich, Germany, September 2018. Springer.

[117] Pedro Miraldo, Tiago Dias, and Srikumar Ramalingam. A minimal closed-form solution for multi-perspective pose estimation using points and lines. In *Computer Vision – ECCV 2018*, pages 490–507. Springer International Publishing, 2018.

[118] Faraz M. Mirzaei and Stergios I. Roumeliotis. Globally optimal pose estimation from line correspondences. In *2011 IEEE International Conference on Robotics and Automation*, pages 5581–5588. IEEE, may 2011.

[119] Anastasiia Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 4826–4837. Curran Associates, Inc., 2017.

[120] Jorge J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In *Lecture Notes in Mathematics*, pages 105–116. Springer Berlin Heidelberg, 1978.

[121] Francesc Moreno-Noguer, Vincent Lepetit, and Pascal Fua. Accurate non-iterative o(n) solution to the pnp problem. In *Proceedings of International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[122] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of International Conference on Machine Learning*, page 807–814, Madison, WI, USA, 2010. Omnipress.

[123] Yoshikatsu Nakajima and Hideo Saito. Robust camera pose estimation by viewpoint classification using deep learning. 3(2):189–198, dec 2016.

[124] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Proceedings of European Conference Computer Vision*, volume 9912, pages 483–499. Springer International Publishing, 2016.

[125] David Nistér and Henrik Stewénius. A minimal solution to the generalised 3-point pose problem. *Journal of Mathematical Imaging and Vision*, 27:67–79, 2004.

[126] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[127] Danda Pani Paudel, Adlane Habed, Cédric Demonceaux, and Pascal Vasseur. Robust and optimal registration of image sets and structured scenes via sum-of-squares polynomials. *International Journal of Computer Vision*, 127(5):415–436, May 2019.

[128] Rémi Pautrat, Lin Juan-Ting, Viktor Larsson, Martin R. Oswald, and Marc Pollefeys. SOLD2: Self-supervised occlusion-aware line description and detection. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2021.

[129] Robert Pless. Using many cameras as one. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–587, 2003.

[130] H. Pottmann and J. Wallner. *Computational Line Geometry*. Mathematics and Visualization. Springer, 2009.

[131] Bronislav Pribyl, Pavel Zemcík, and Martin Cadík. Camera pose estimation from lines using Plücker coordinates. In Xianghua Xie, Mark W. Jones, and Gary K. L. Tam, editors, *Proceedings of the British Machine Vision Conference*, pages 45.1–45.12, Swansea, UK, September 2015. BMVA Press.

[132] Luis Puig and J.J. Guerrero. *Omnidirectional Vision Systems*. Springer London, 2013.

[133] Pradeep Pujari, Mohit Sewak, and Md. Rezaul Karim. *Practical Convolutional Neural Network Models*. Packt Publishing, 2018.

[134] Albert Pumarola, Alexander Vakhitov, Antonio Agudo, Alberto Sanfeliu, and Francese Moreno-Noguer. PL-SLAM: Real-time monocular visual SLAM with points and lines. In *International Conference on Robotics and Automation (ICRA)*, pages 4503–4508. IEEE, may 2017.

[135] Tang Xia Qing, Wu Fan, and Zong Yan Tao. Camera pose estimation method based on deep neural network. In *ICDLT : Proceedings of the 3rd International Conference on Deep Learning Technologies*, page 85–90. ACM Press, 2019.

[136] R. Vettriselvan R. Sujatha, S. L. Aarthy, editor. *Integrating Deep Learning Algorithms to Overcome Challenges in Big Data Analytics*. Taylor & Francis Ltd., 2021.

[137] M. Rafiquzzaman. *Microcontroller Theory and Applications with the PIC18F*. John Wiley & Sons, 2017.

[138] Baoxin Li Ragav Venkatesan. *Convolutional Neural Networks in Visual Computing*. Taylor & Francis Ltd., 2017.

[139] D. Binu B.R. Rajakumar, editor. *Artificial Intelligence in Data Mining: Theories and Applications*. Elsevier Science, 2021.

[140] Jason R. Rambach, Aditya Tewari, Alain Pagani, and Didier Stricker. Learning to fuse: A deep learning approach to visual-inertial camera pose estimation. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 71–76. IEEE, sep 2016.

[141] Sidney Ray. *Applied Photographic Optics*. Taylor & Francis Ltd., 2002.

[142] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *An Introduction to Machine Learning*. Springer-Verlag GmbH, 2019.

[143] Jerome Revaud, Philippe Weinzaepfel, César Roberto de Souza, and Martin Humenberger. R2D2: repeatable and reliable detector and descriptor. In *NeurIPS*, 2019.

[144] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of International Conference on Computer Vision*, pages 2564–2571. IEEE, November 2011.

[145] Yohann Salaün, Renaud Marlet, and Pascal Monasse. Robust and accurate line- and/or point-based pose estimation without manhattan assumptions. pages 801–818. Springer International Publishing, 2016.

[146] Dominik Sankowski and Jacek Nowakowski. *Computer Vision in Robotics and Industrial Applications*, volume Series in Computer Vision: Volume 3. WORLD SCIENTIFIC, aug 2014.

[147] Dipanjan Sarkar, Raghav Bali, and Tushar Sharma. *Practical Machine Learning with Python*. Springer-Verlag GmbH, 2017.

[148] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12716–12725, June 2019.

[149] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Efficient & effective prioritized matching for large-scale image-based localization. 39(9):1744–1756, 2016.

[150] Torsten Sattler, Akihiko Torii, Josef Sivic, Marc Pollefeys, Hajime Taira, Masatoshi Okutomi, and Tomas Pajdla. Are large-scale 3D models really necessary for accurate visual localization? In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, page 10, July 2017.

[151] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion. In *International Conference on Computer Vision Systems*, pages 45–51, Washington, USA, January 2006.

[152] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. A toolbox for easily calibrating omnidirectional cameras. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5695–5701. IEEE, oct 2006.

[153] C. Schmid and A. Zisserman. Automatic line matching across views. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 666–671. IEEE, June 1997.

[154] J. L. Schönberger, H. Hardmeier, T. Sattler, and M. Pollefeys. Comparative evaluation of hand-crafted and learned local features. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 6959–6968, July 2017.

[155] Johannes L. Schönberger, Marc Pollefeys, Andreas Geiger, and Torsten Sattler. Semantic visual localization. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 6896–6906, June 2018.

[156] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A unified embedding for face recognition and clustering. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 815–823, June 2015.

[157] P. Smith, I. Reid, and A. J. Davison. Real-time monocular SLAM with straight lines. In *British Machine Vision Association (BMVC)*. British Machine Vision Association, 2006.

[158] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[159] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer London, 2011.

[160] Levente Tamas, Robert Frohlich, and Zoltan Kato. Relative pose estimation and fusion of omnidirectional and lidar cameras. In Lourdes de Agapito, Michael M. Bronstein, and Carsten Rother, editors, *Proceedings of the ECCV Workshop on Computer Vision for Road Scene Understanding and Autonomous Driving*, volume 8926 of *Lecture Notes in Computer Science*, pages 640–651, Zurich, Switzerland, September 2014. Springer.

[161] Levente Tamas and Zoltan Kato. Targetless calibration of a lidar - perspective camera pair. In *Proceedings of ICCV Workshop on Big Data in 3D Computer Vision*, pages 668–675, Sydney, Australia, December 2013. IEEE, IEEE.

[162] Camillo J. Taylor and David J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1021–1032, November 1995.

[163] Inc. The MathWorks. *Computer Vision Toolbox*. Natick, Massachusetts, United State, 2018.

[164] Yurun Tian, Bin Fan, and Fuchao Wu. L2-Net: Deep Learning of Discriminative Patch Descriptor in Euclidean Space. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6128–6136. IEEE, jul 2017.

[165] Carl Toft, Erik Stenborg, Lars Hammarstrand, Lucas Brynte, Marc Pollefeys, Torsten Sattler, and Fredrik Kahl. Semantic match consistency for long-term visual localization. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Proceedings of European Conference Computer Vision*, pages 391–408. Springer International Publishing, 2018.

[166] P.H.S. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, apr 2000.

[167] Matthew Turk. Computer vision in the interface. *Communications of the ACM*, 47(1):60–67, jan 2004.

[168] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: A survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280, July 2008.

[169] A. Vakhitov and V. Lempitsky. Learnable line segment descriptor for visual SLAM. *IEEE Access*, 7:39923–39934, 2019.

[170] Alexander Vakhitov, Luis Ferraz Colomina, Antonio Agudo, and Francesc Moreno-Noguer. Uncertainty-aware camera pose estimation from points and lines. IEEE, jun 2021.

[171] Alexander Vakhitov, Jan Funke, and Francesc Moreno-Noguer. Accurate and linear time pose estimation from points and lines. pages 583–599. Springer International Publishing, 2016.

[172] Alexander Vakhitov, Victor Lempitsky, and Yinqiang Zheng. Stereo relative pose from line and point feature triplets. In *Computer Vision – ECCV 2018*, pages 662–677. Springer International Publishing, 2018.

[173] B. Verhagen, R. Timofte, and L. Van Gool. Scale-invariant line descriptors for wide baseline matching. In *Proceedings of Winter Conference on Applications of Computer Vision*, pages 493–500. IEEE, March 2014.

[174] R.G. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. LSD: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, apr 2010.

[175] Naja von Schmude, Pierre Lothe, and Bernd Jähne. Relative pose estimation from straight lines using parallel line clustering and its application to monocular visual odometry. SCITEPRESS - Science and Technology Publications, 2016.

[176] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 5265–5274. IEEE, June 2018.

[177] Ping Wang, Yongxin Chou, Aimin An, and Guili Xu. Solving the PnL problem using the hidden variable method: an accurate and efficient solution. *The Visual Computer*, pages 1–12, nov 2020.

[178] Ping Wang, Guili Xu, and Yuehua Cheng. A novel algebraic solution to the perspective-three-line problem. *Computer Vision and Image Understanding (CVIU)*, 191:102711, 2020. in press.

[179] Ping Wang, Guili Xu, Yuehua Cheng, and Qida Yu. Camera pose estimation from lines: a fast, robust and general method. *Machine Vision and Applications*, 30(4):603–614, feb 2019.

[180] Qiang Wang, Wei Zhang, Xiaolong Liu, Zhenxin Zhang, Muhammad Hasan Ali Baig, Guo Wang, Long He, and Tiejun Cui. Line matching of wide baseline images in an affine projection space. *International Journal of Remote Sensing*, pages 1–23, July 2019.

[181] Zhiheng Wang, Fuchao Wu, and Zhanyi Hu. MSLD: A robust descriptor for line matching. *Pattern Recognition*, 42(5):941 – 953, 2009.

[182] Bhattarabhorn Wattanacheep and Orachat Chitsobhuk. Camera pose estimation using CNN. ACM, aug 2020.

[183] Wu Wen-Tsun. Basic principles of mechanical theorem proving in elementary geometries. 2(3):221–252, sep 1986.

[184] Christian Wöhler. *3D Computer Vision: Efficient Methods and Applications*. Springer London, 2 edition, 2013.

[185] F. C. Wu, Z. H. Wang, and Z. Y. Hu. Cayley transformation and numerical stability of calibration equation. *International Journal of Computer Vision*, 82(2):156–184, dec 2008.

[186] Yihong Wu, Fulin Tang, and Heping Li. Image-based camera localization: an overview. *Visual Computing for Industry, Biomedicine, and Art*, 1(1), sep 2018.

[187] Jun Xie, Martin Kiefel, Ming-Ting Sun, and Andreas Geiger. Semantic instance annotation of street scenes by 3d to 2d label transfer. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2016.

[188] C. Xu, L. Zhang, L. Cheng, and R. Koch. Pose estimation from line correspondences: A complete analysis and a series of solutions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1209–1222, 2016.

[189] Yanwu Xu, Mingming Gong, Tongliang Liu, Kayhan Batmanghelich, and Chaohui Wang. Robust angular local descriptor learning. In C.V. Jawahar, Hongdong Li, Greg Mori, and Konrad Schindler, editors, *Proceedings of Asian Conference on Computer Vision*, pages 420–435, Perth, Australia, 2019. Springer.

[190] Zezhong Xu, Bok-Suk Shin, and Reinhard Klette. Accurate and robust line segment extraction using minimum entropy with hough transform. *IEEE Transactions on Image Processing*, 24(3):813–822, 2015.

[191] Nan Xue, Tianfu Wu, Song Bai, Fudong Wang, Gui-Song Xia, Liangpei Zhang, and Philip H.S. Torr. Holistically-attracted wireframe parsing. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 2785–2794, 2020.

[192] Chenhao Yang, Yuyi Liu, and Andreas Zell. RCPNet: Deep-learning based relative camera pose estimation for UAVs. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1085–1092. IEEE, sep 2020.

[193] Kwang Moo Yi, Eduard Trulls Fortuny, Vincent Lepetit, and Pascal Fua. LIFT: Learned invariant feature transform. In *Proceedings of European Conference on Computer Vision*, volume 9910 of *Lecture Notes in Computer Science*, pages 467–483, Amsterdam, Nederlands, October 2016. Springer.

[194] Qida Yu, Guili Xu, and Yuehua Cheng. An efficient and globally optimal method for camera pose estimation using line features. 31(6), jul 2020.

[195] Zikang Yuan, Dongfu Zhu, Cheng Chi, Jinhui Tang, Chunyuan Liao, and Xin Yang. Visual-inertial state estimation with pre-integration correction for robust mobile augmented reality. In *Proceedings of the 27th ACM International Conference on Multimedia*, page 1410–1418. ACM, oct 2019.

[196] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 4353–4361, June 2015.

[197] Haotian Zhang, Yicheng Luo, Fangbo Qin, Yijia He, and Xiao Liu. Elsd: Efficient line segment detector and descriptor. *ArXiv*, abs/2104.14205, 2021.

[198] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, oct 2016.

[199] Lilian Zhang and Reinhard Koch. Hand-held monocular SLAM based on line segments. In *Proceedings of the Irish Machine Vision and Image Processing Conference*, pages 7–14, Dublin, Ireland, 2011. IEEE Computer Society.

[200] Lilian Zhang and Reinhard Koch. An efficient and robust line segment matching approach based on LBD descriptor and pairwise geometric consistency. *Journal of Visual Communication and Image Representation*, 24(7):794 – 805, 2013.

[201] M. Zhang, Y. Chen, and G. Wang. Hybrid silhouette and key-point driven 3D-2D registration. In *Image and Graphics (ICIG), 2013 Seventh International Conference on*, pages 585–590, July 2013.

[202] X. Zhang, F. X. Yu, S. Karaman, and S. Chang. Learning discriminative and transformation covariant local feature detectors. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 4923–4931. IEEE, July 2017.

[203] Xiaohu Zhang, Zheng Zhang, You Li, Xianwei Zhu, Qifeng Yu, and Jianliang Ou. Robust camera pose estimation from unknown or known line correspondences. *Applied Optics*, 51(7):936, feb 2012.

[204] Ji Zhao, Laurent Kneip, Yijia He, and Jiayi Ma. Minimal case relative pose computation using ray-point-ray features. pages 1–1, 2019.

[205] Yinqiang Zheng, Yubin Kuang, Shigeki Sugimoto, Kalle Åström, and Masatoshi Okutomi. Revisiting the pnp problem: A fast, general and optimal solution. In *2013 IEEE International Conference on Computer Vision*, pages 2344–2351, 2013.

[206] Huizhong Zhou, Danping Zou, Ling Pei, Rendong Ying, Peilin Liu, and Wenxian Yu. StructSLAM: Visual SLAM with building structure lines. 64(4):1364–1375, apr 2015.

[207] Yichao Zhou, Haozhi Qi, and Yi Ma. End-to-end wireframe parsing. In *Proceedings of International Conference on Computer Vision*, pages 962–971. IEEE, October 2019.

[208] Wenbo Zhu, Yan Ma, Yizhong Zhou, Michael Benton, and Jose Romagnoli. Deep learning based soft sensor and its application on a pyrolysis reactor for compositions predictions of gas phase components. In *13th International Symposium on Process Systems Engineering (PSE 2018)*, pages 2245–2250. Elsevier, 2018.