TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

ACIN
AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Autonomous Extrinsic Calibration of a Depth Sensing Camera on Mobile Robots

MASTER'S THESIS

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Ao.Univ.-Prof. Dipl.-Ing. Dr. techn. M. Vincze
Dipl.-Ing. G. Halmetschlager-Funek

submitted at the

Vienna University of Technology
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Farhoud Malekghasemi
Laudongasse 36/515
A-1080 Vienna
Austria

Vienna, 15th June, 2018

Preamble

First I would like to thank my supervisor Dipl.-Ing. Georg Halmetschlager-Funek for his invaluable guidance during my master's thesis, and special thanks to my professor Dr. Markus Vincze for supporting and encouraging me during my master's studies.

I would also like to thank my family specially my parents Leyli Masoud and Farhad Malekghasemi who made this possible for me and were always supportive and gave me motivation to develop myself.

Special thanks to my dear wife Sara Rezaei for her support, whose understanding and help made hard days much easier.

Farhoud Malekghasemi
Vienna, 15th June, 2018

Abstract

This study presents a fast and autonomous system to find the rigid transformation between the RGB-D camera and a local reference frame on a mobile robot. The major advantages of the method over the conventional methods of calibration is that there is no need for a special setup or any known object in the scene. This is achieved by taking advantage of robot's motion combined with camera tracking method. It is shown that two circular motion and one plane detection are sufficient to autonomously calibrate the robot in different environments with some minimal texture. The presented method is evaluated with both, computer simulation and in real-life scenarios.

Kurzzusammenfassung

Diese Arbeit präsentiert ein schnelles und autonomes System, um die starre Transformation zwischen der RGB-D-Kamera und einem lokalen Koordinatensystem auf einem mobilen Roboter zu finden. Die Hauptvorteile des Verfahrens gegenüber den herkömmlichen Kalibrierungsverfahren bestehen darin, dass keine spezielles Setup oder irgendein bekanntes Objekt in der Szene benötigt werden. Dies wird erreicht, indem die Roboterbewegung mit einer verfolgung des Kamerapose kombiniert wird. Es wird gezeigt, dass zwei kreisförmige Bewegung und eine Ebenenerkennung ausreichen, um den Roboter in unterschiedlichen Umgebungen die ein Mindestmass an Textur aufweisen, autonom zu kalibrieren. Die vorgestellte Methode wird sowohl in einer Computersimulation als auch in realen Szenarien evaluiert.

Contents

List of Figures

List of Tables

1 Introduction

The use of vision as a sensor to provide information for controlling autonomous systems, such as AMR[1] and redundant manipulators, has grown significantly in recent years [1]. After Microsoft released the Kinect RGB-D sensor (figure 1.1) as a new natural user interface in November 2010 for its XBOX 360 gaming platform, RGB-D sensors, also known as depth or 3D[2] cameras, got more popular in branches of robotics and computer vision as they became cheaper, more accurate, and smaller compared to the previous existing sensors.

Figure 1.1: Microsoft Kinect RGB-D sensor.

An RGB-D sensor senses the depth information of environment using an infrared camera and projection of structured infrared light to the scene. It then registers depth information with the visual information from a RGB camera. The resulted information opens up new opportunities to solve fundamental problems such as object and activity recognition, people tracking, 3D mapping, SLAM[3], segmentation and 3D reconstruction, etc [2]–[6].

In order to interpret the collected data by a 3D camera from the scene into useful information for the tasks mentioned before, a rigid transformation between the camera and a reference point is always necessary. For example, If a manipulator robot detects an object in the scene using its camera and wants to manipulate it with its gripper, this transformation must be used to calculate

[1] Autonomous Mobile Robots
[2] Three Dimensional
[3] Simultaneous localization and mapping

the object location in real world coordinate system in order to control its arm for reaching this exact point in space. Another application example for this rigid transformation is data fusion between multiple sensors on a robot. For example, when there is two 3D cameras or a camera and a laser range scanner this transformation is necessary in order to align point clouds together. The parameters which are used to describe this transformation are called extrinsic parameters of a camera and extrinsic calibration or camera pose calibration are the terms used in literature for describing the methods that determine these parameters.

The state-of-the-art methods of extrinsic calibration include:

- Measuring distances directly or having the CAD model of the robot available (for extrinsic calibration);
- Using reference objects on the scene with precalibrated position and orientation [7];
- Knowing the geometry of the scene (photogrammetry resectioning method [8]);
- Using calibration pattern like checkerboard (used in OpenCV and Matlab [9], [10]).

1.1 Problem Description

In practice, extrinsic parameters of a camera are not always constant and may change in multiple cases such as:

- Wear and tear in robot parts through time;
- Collision accidents;
- Changing mounting place of the camera on body of the robot by user to adopt different environments;
- The camera is mounted on a pan-tilt unit.

All of these displacements, violate the prior assumption of known extrinsic transformations. Thus, recalibration of the camera is unavoidable.

Recalibration is the starting point of the problems, because all of the state-of-the-art methods of the extrinsic calibration aforementioned are challenging, need long procedures which makes them time consuming, need external precalibrated objects or calibration patterns and not easily repeatable without an expert in the loop. For example, for filling the images into the the system with showing a calibration template to the cameras.

The current study aims to develop a fast and autonomous method to estimate this rigid transformation between an RGB-D camera, on-board of a mobile robot, and a reference point on the robot. It is assumed that the working area of the robot is a flat floor on which:

- The robot can freely move around on the floor;
- The floor can be observed by the cameras;
- The floor has some minimal texture (there is always some on the floor);
- The cameras are mounted in front of the robot.

It is shown that, driving the robot in two circular paths by benefiting from its locomotion combined with a visual motion estimation method to determine camera trajectory while movement and one plane detection are sufficient to autonomously calibrate the robot in different environments with some minimal texture without need to any calibration object or pattern.

The remainder of this study is structured as follows. Chapter 2 summarizes the previous similar research pertaining to camera calibration problem. Chapter 3 introduces basic principles of rotation matrix in 3.1 and camera calibration in section 3.2. Extrinsic and intrinsic parameters of a camera will be discussed in more detail there. Section 3.3 of this chapter explains the principles behind visual motion estimation also known as visual odometry. In chapter 4 the approach taken to solve the problem is discussed. Chapter 5 describes system implementation in ROS[4] with information about structure of nodes, launch files, services and etc. The method is evaluated through the experiments in chapter 6 and the outcome results are analyzed. Chapter 7 concludes the study and gives some directions for further research.

[4]Robot Operating System

2 Related Work

This chapter presents related previous researches and state-of-the-art in the field of motion base camera calibration methods. Most of related researches focus on the calibration of intrinsic camera parameters.

Classic methods of calibration using patterns are well known and accurate for both intrinsic and extrinsic parameters. The principles behind these methodes are explaind in section 3.2. However, there exist several methods that use the motion of a camera to obtain sequence of images from a static scene with invariant features to match in order to calibrate the intrinsic parameters of a camera without need to any calibration pattern such as following works.

Pollefeys et al. [11] proposed a calibration method for varying intrinsic camera parameters in zooming/focusing cameras using image sequences. It shows that the absence of skew alone is enough to allow selfcalibration.

Maybank et al. [12] proposed a method for intrinsic calibration of RGB camera when it has displacements with respect to a rigid scene. It uses the epipolar transformations relate with this displacements of the camera. Correspondences points between pair of images of the scene are used in order to estimate the epipolar transformation. However, Hartley [13] reported that this method requires extreme accuracy of computation and it is unworkable. Instead it suggests another parameter estimation algorithm applicable to multiple number of images.

Luong et al. [14] shows that correspondences points between three images from a static scene is sufficient to calculate the epipolar transformations of each pair of images, intrinsic parameters and the motion parameters.

Most of the researches on extrinsic parameters calibration without using any specialized reference objects has focused on the case of multiple cameras such as following work.

Carrera et al. [15] estimates rigid transformation between multiple cameras (RGB) mounted on a mobile robot with no image overlap for data fusion by a

full horizontal rotation movement and capturing a synchronized image sequence from each camera. It detects correspondences points using invariant SURF feature between different images from each camera and perform 3D alignment to fuse.

Miller et al. [16] estimates rigid transformation between multiple 3D cameras fixed in the environment. It uses the unstructured motion of objects in the scene for calibrating the relative pose and time offsets of a pair of depth sensors autonomously. Initial extrinsic parameters are estimated from candidate point correspondences at each time frame which are extracted from positions of moving objects in the scene. Finally, this initial parameters are refined with an occlusion-aware energy minimization.

Pathirana et al. [17] proposes an autonomous method to calibrate multiple 3D cameras fixed in a rehabilitation environment to achieve data fusion by detecting skeleton data (human joints' positions).

There are relatively very little researches on extrinsic calibration of a single 3D camera that we are considering in this study.

3 Background

This chapter presents the principles behind defining pose of an object in 3D space in section 3.1, camera calibration in section 3.2 and visual motion estimation methods in section 3.3.

3.1 Pose in 3D Space

In computer vision and robotics, a pose is defined as the combination of position and orientation of an object in 3D space relative to a reference coordinate system. This combination defines a coordinate system (frame) attached to the object in the reference frame such as world coordinate system. Translation and rotation transformations are used to describe relationship between these two coordinate systems which defines pose of the object in the reference coordinate system.

3.1.1 Translation Transformation

The parallel move or translation is an affine geometric mapping that moves every point of the space in the same direction by the same distance. It is identified by a displacement vector $\mathbf{t} = [X,Y,Z]^T$ (cf. figure 3.1).

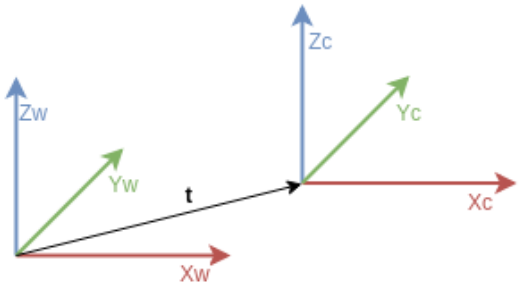

Figure 3.1: Translation vector $\mathbf{t}$.

A point $\mathbf{p}$ has the position vectors ${}_W\mathbf{p}$ in the coordinate system $\sum_W$ and ${}_C\mathbf{p}$ in the coordinate system $\sum_C$. The relationship between them is calculated by applying translation vector $\mathbf{t}$ from $\sum_C$ to $\sum_W$:

$$ {}_C\mathbf{p} = {}_W\mathbf{p} + {}^W_C\mathbf{t}. \tag{3.1}$$

Remark.

$$ {}^W_C\boldsymbol{t} = -({}^C_W\boldsymbol{t}) $$

3.1.2 Rotation Transformation

Rotation transformation $\mathbf{R}$ is a 3×3 matrix that defines relationship between two coordinate systems which are rotated respect to each other. An elemental rotation is a rotation about one of the axes of a coordinate system. Basic rotation matrices (equations 3.2 – 3.4) are known that rotate a vector by an angle about X,Y or Z axis in mathematical positive direction. However, the question arises: how can be a general rotation transformation in 3D space easily represented ?

$$ \mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.2}$$

$$ \mathbf{R}_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{3.3}$$

$$ \mathbf{R}_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Remark.

$$ ({}^C_W\boldsymbol{R})^{-1} = {}^W_C\boldsymbol{R} = ({}^C_W\boldsymbol{R})^T $$

$$ \boldsymbol{R}_x\boldsymbol{R}_y \neq \boldsymbol{R}_y\boldsymbol{R}_x $$

Any orientation in 3D space is parameterized by three DoF[1]. Therefore, three elemental rotations are enough to define a general rotation. There exist different methods for parameterization of a general rotation using elemental rotations. Two most common methods, Euler and RPY[2] angels, will be explained in the following and later on a completely different method called quaternion will be discussed.

Euler Angels

A general rotation matrix $\mathbf{R}$ in 3D space is defined by the Euler triple (α, β, γ) as a combination of elemental rotations:

$$\overset{C}{W}\mathbf{R} = \mathbf{R}_{z,\alpha}\mathbf{R}_{x,\beta}\mathbf{R}_{z,\gamma}. \tag{3.5}$$

It is rotated first about the Z axis of the original system by α, then about the current X axis by β and finally by γ about the now current Z axis. By using the elemental rotation matrices, parameterized rotation matrix becomes ($sin(\alpha)$ and $cos(\alpha)$ are abbreviated to s_α and c_α):

$$\overset{C}{W}\mathbf{R} = \begin{bmatrix} c_\alpha c_\beta c_\gamma - s_\alpha s_\gamma & -c_\alpha c_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta \\ s_\alpha c_\beta c_\gamma + c_\alpha s_\gamma & -s_\alpha c_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta \\ -s_\beta c_\gamma & s_\beta s_\gamma & c_\beta \end{bmatrix} \tag{3.6}$$

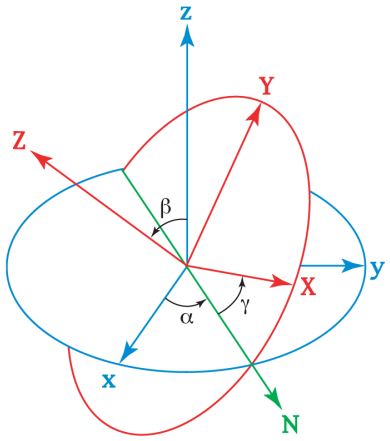

Figure 3.2: Euler angels rotation.[3]

[1]Degrees of Freedom
[2]Roll – Pitch – Yaw
[3]https://commons.wikimedia.org/wiki/File:Euler.png

RPY Angels

An alternative to the Euler angles is so-called roll pitch yaw angles. Unlike the Euler angles, these refer to a fixed reference frame. A general rotation matrix $\mathbf{R}$ in 3D space is defined by the RPY triple (ϕ, θ, ψ) as a combination of elemental rotations:

$$\overset{C}{W}\mathbf{R} = \mathbf{R}_{z,\psi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi} \tag{3.7}$$

whose angles refer to the reference coordinate system as in figure 3.3. ϕ becomes as roll, θ as pitch and ψ as yaw angle. An explicit representation of parameterized rotation matrix is ($sin(\alpha)$ and $cos(\alpha)$ are abbreviated to s_α and c_α):

$$\overset{C}{W}\mathbf{R} = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{3.8}$$

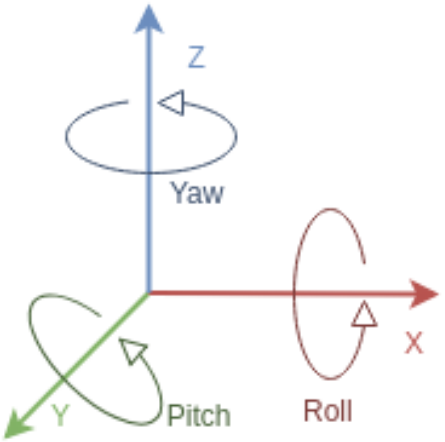

Figure 3.3: RPY angels reference.

The parameterization of a general rotation by Euler or RPY angles is very descriptive. However, it is proved at critical points both of them have problem. This phenomenon is called Gimbal lock. At this points, both methods lose one DoF and their ability to represent all rotations in 3D space. This problem appears when two axes are driven into a parallel configuration. One way of overcoming this problem is using quaternions.

Quaternion

Quaternions are extension of complex numbers to three dimensions. A quaternion $q \in \mathbb{H}$ is represented using imaginary units i, j and k as:

$$q = q_1 + iq_2 + jq_3 + kq_4. \tag{3.9}$$

As with the complex numbers, there are also multiplication rules for quaternions which are agreed on as following:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3.10}$$

$$ij = +k \quad jk = +i \quad ki = +j \tag{3.11}$$

$$ji = -k \quad kj = -i \quad ik = -j. \tag{3.12}$$

Remark.

$$Re(q) = q_1$$

$$Im(q) = iq_2 + jq_3 + kq_4$$

$$\bar{q} = q_1 - iq_2 - jq_3 - kq_4$$

$$|q| = q\bar{q} = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}$$

Two vectors $\mathbf{c} = [c_x,c_y,c_z]^T$ and $\mathbf{w} = [w_x,w_y,w_z]^T$ can be represented as purely imaginary quaternion:

$$q_c = ic_x + jc_y + kc_z,$$

$$q_w = iw_x + jw_y + kw_z.$$

There exists a quaternion $\rho \in \mathbb{H}$ with $|\rho| = 1$, that causes a rotation from $\mathbf{c}$ to $\mathbf{w}$ by evaluating:

$$q_W = \rho \quad q_C \quad \bar{\rho}. \tag{3.13}$$

ρ defines a mathematically positive quaternion rotation by the angle α about the rotation axis in the direction of the unit vector $\mathbf{r}$ as:

$$\rho = cos\left(\frac{\alpha}{2}\right) + sin\left(\frac{\alpha}{2}\right)(ir_x + jr_y + kr_z). \tag{3.14}$$

A rotation matrix $\mathbf{R}$ in 3D space results using the given unit quaternion in equation 3.9 as following.

$$\mathbf{R} = \begin{bmatrix} 1 - 2(q_3^2 + q_4^2) & 2(q_2q_3 - q_1q_4) & 2(q_2q_4 + q_1q_3) \\ 2(q_2q_3 + q_1q_4) & 1 - 2(q_4^2 + q_2^2) & 2(q_3q_4 - q_1q_2) \\ 2(q_2q_4 - q_1q_3) & 2(q_3q_4 + q_1q_2) & 1 - 2(q_2^2 + q_3^2) \end{bmatrix} \tag{3.15}$$

This assignment of rotations to unit quaternions ρ is not unique. Any general rotation in 3D space is determined by two unit quaternions.

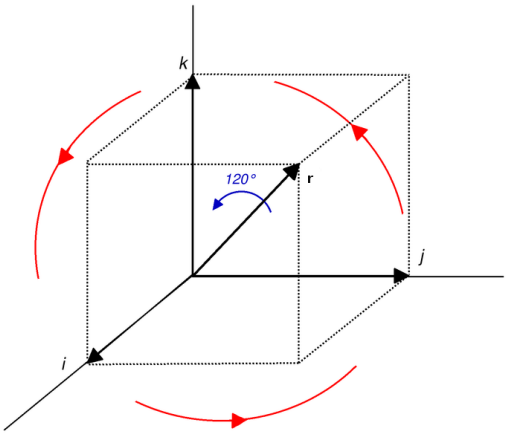

Figure 3.4: The rotation about the axis $r = i + j + k$ by rotation angle of 120°.[4]

[4] https://en.wikipedia.org/wiki/File:Diagonal_rotation.png

3.2 Camera Calibration

The world and camera coordinate system are related by the lens and image sensor parameters, namely focal length of the lens, pixels size of sensor, center of image position (principle point) and the position and orientation of the camera. Estimating these parameters is referred to as camera calibration or camera resectioning. These parameters are used to correct for lens distortion, measure size of an object or finding the location of the camera in scene. The cameras are used to get information about the environment so it is necessary to know the relationship between image and the world coordinate system. Therefore, these parameters are categorize in three main parts:

- Extrinsic parameters, which define transformation from 3D world coordinate to 3D camera coordinate system;
- Intrinsic parameters, which define transformation from 3D coordinate system of camera to 2D image plane;
- Lens distortion (radial and tangential) coefficients.

Figure 3.5 illustrates these transformations between different coordinate systems. When the camera is calibrated and all the parameters are estimated, it will be possible to conduct quantitative 3D measurements in real world through the image of camera [18]. At first, for modeling the camera pinhole model is being used, which has the general assumption that the lens of camera obeys the thin lens approximation:

$$\frac{1}{z} + \frac{1}{z'} = \frac{1}{f} \tag{3.16}$$

wherein z is the distance to the object and z' is the image distance to the lens and f is the focal length (figure 3.6).

Using homogeneous coordinates, ${}^{W}\mathbf{P} = [X,Y,Z,1]^T$ for a point in the world coordinate system and ${}^{C}\mathbf{P} = [x,y,z,1]^T$ for the same point in camera coordinates and $\mathbf{p} = [u,v,1]^T$ for a point in image frame (pixel coordinates), helps to easily represent perspective projection transformation in the form of a 3×4 matrix M as following [18]:

$$\mathbf{p} = \frac{1}{z}\mathbf{M}({}^{W}\mathbf{P}) \tag{3.17}$$

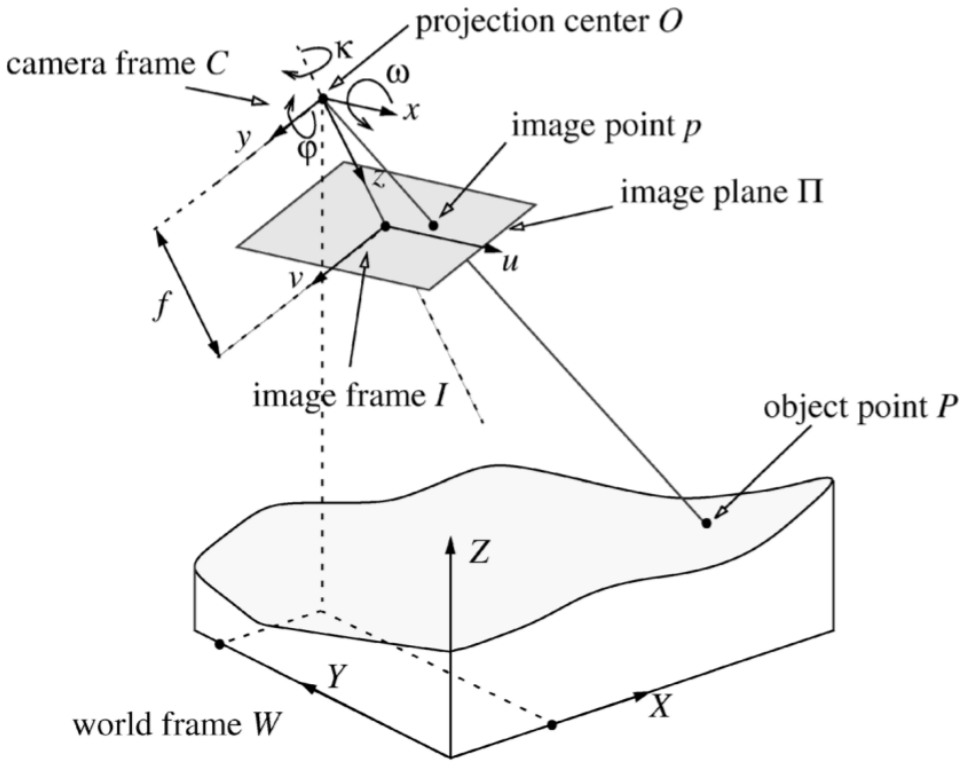

Figure 3.5: Pinhole camera model and relation between the image plane Π, camera center frame (C) and fix coordinate system W. [19]

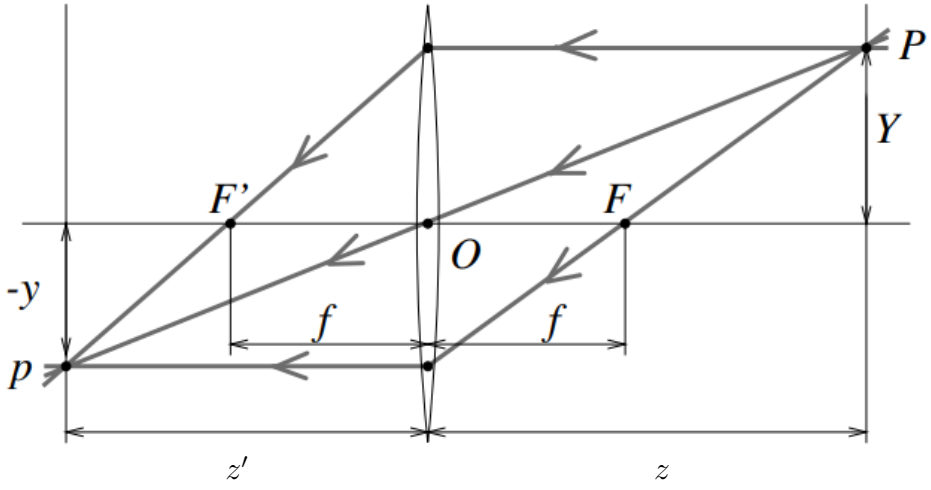

Figure 3.6: A thin lens. Rays passing through center are not refracted. Rays parallel to the optical axis are focused on the focal point F'. [18]

The perspective projection matrix $\mathbf{M}$ in equation 3.17 is decomposed into two parts. As discussed previously these are the intrinsic parameters matrix $\mathbf{K}$ and extrinsic parameters consisted of rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$.

$$\mathbf{M} := \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tag{3.18}$$

3.2.1 Extrinsic Parameters

Extrinsic parameters, $\mathbf{R}$ and $\mathbf{t}$, relate the 3D camera center coordinate C to the (arbitrary) 3D world coordinate system W which is base of the mobile robot in this case. $\mathbf{R}$ is a 3×3 rotation matrix defined by three independent parameters resulting from the product of three elementary rotations such as Euler angels (ω,φ,κ in figure 3.5) and $\mathbf{t}$ is a 3×1 translation vector. Adding these two together results in six extrinsic parameters which specifies the position and orientation (pose) of the camera in space with respect to the robot base. The transformation of coordinates between C and W is rigid, therefore it can be written as:

$${}^{C}\mathbf{P} = \begin{bmatrix} {}^{C}_{W}\mathbf{R} & {}^{C}_{W}\mathbf{t} \\ \mathbf{0}^{T} & 1 \end{bmatrix} ({}^{W}\mathbf{P}). \tag{3.19}$$

A rigid body in three dimensional space has six DoF. The extrinsic parameters represent these six. Our main focus in this study is to autonomously define these six extrinsic parameters.

3.2.2 Intrinsic Parameters

Intrinsic parameters, also referred to as intrinsic matrix $\mathbf{K}$ (3×3), relate the 3D camera center (optical center) to the 2D image coordinate system by a projective transformation. By considering only the intrinsic part of equation 3.17 it is obvious that:

$$\mathbf{p} = \frac{1}{z}\begin{bmatrix}\mathbf{K} & \mathbf{0}\end{bmatrix}({}^{C}\mathbf{P}). \tag{3.20}$$

The equations 3.21 are known from the mathematics of the pinhole camera model (figure 3.5) for perspective projection form 3D to ideal 2D real image coordinates.

$$\begin{cases} \tilde{u} = f\dfrac{x}{z} \\ \tilde{v} = f\dfrac{y}{z} \end{cases} \tag{3.21}$$

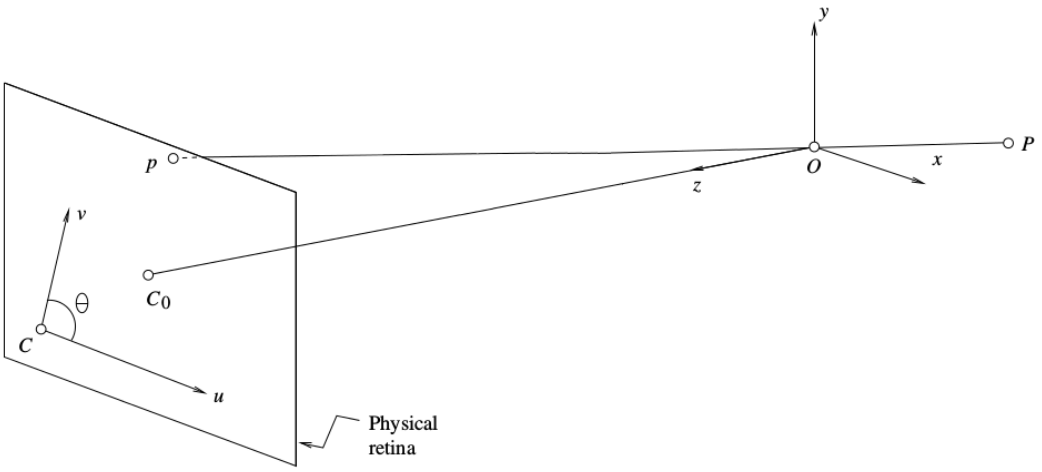

Figure 3.7: Image coordinate system. [18]

Point $\boldsymbol{p} = [u,v]^T$ (before the homogeneous coordinate vector of $\mathbf{p}$) on the computer image plane is defined in pixel units instead of metric unit, contrary to f which is in milliliters, and usually pixels are not completely square (figure 3.7). Therefore some scaling factors should be introduced to compensate for these problems instead of f (α for x axis and β for y axis). The equations 3.21 can be written as:

$$\begin{cases} u' = \alpha\dfrac{x}{z} \\ v' = \beta\dfrac{y}{z} \end{cases} \tag{3.22}$$

Additionally, due to some manufacturing error camera pixel coordinate system may be skewed, therefore the angle θ between u and v axes is not exactly 90 degrees (figure 3.8). Trigonometry proves the relation between actual u',v' and measured u'',v'' as following:

$$\begin{cases} u'' = u' - cos(\theta)v'' = u' - cot(\theta)v' \\ v'' = \dfrac{v'}{sin(\theta)} \end{cases} \tag{3.23}$$

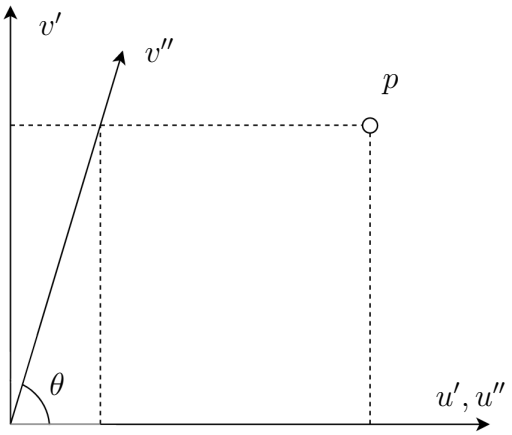

Figure 3.8: Skew between camera pixel axes.

Finally, the actual origin of the camera coordinate system is usually not at the center. For example as shown in the figure 3.7, the origin C is at the lower left corner. To define this position two more parameters u_0 and v_0 are added to equations. Thus, equations 3.23 becomes:

$$\begin{cases} u = u'' + u_0 \\ v = v'' + v_0 \end{cases} \tag{3.24}$$

By substituting equations 3.22 and 3.23 into 3.24 it will be obtained:

$$\begin{cases} u = \alpha\dfrac{x}{z} - \beta cot(\theta)\dfrac{y}{z} + u_0 \\ \\ v = \dfrac{\beta}{sin(\theta)}\dfrac{y}{z} + v_0 \end{cases} \tag{3.25}$$

These equations are written as following by converting to homogeneous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z}\begin{bmatrix} \alpha & -\beta cot(\theta) & u_0 & 0 \\ 0 & \dfrac{\beta}{sin(\theta)} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{3.26}$$

Comparing these equations and 3.20 gives us the intrinsic matrix:

$$\mathbf{K} = \begin{bmatrix} \alpha & -\beta cot(\theta) & u_0 \\ 0 & \dfrac{\beta}{sin(\theta)} & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.27}$$

Substituting α and $\dfrac{\beta}{sin(\theta)}$ with focal length f and aspect ratio between two with a, $[c_x,c_y]$ for coordinates of the principal point in pixels and s for skew coefficient of the camera, leads to a simpler notation for intrinsic matrix with five DoF.

$$\mathbf{K} = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.28}$$

If the camera does have square pixels on its sensor and no skew exists between the axes the intrinsic matrix becomes even more simple:

$$\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.29}$$

3.2.3 Distortion Coefficients

The pinhole camera model is a useful model which gives us mathematical means to describe the relationship between world and image points. However, it does not account for lens aberrations because of the general assumption of thin lens. A more realistic camera model also includes these aberrations and models them using distortion coefficients. Two major of these are radial and tangential distortions.

Radial Distortion

Most commonly confronted distortion is radially symmetric ones due to cheap lens manufacturing. As a result of light rays bending more near edges than optical center of the lens. Straight lines in scene turn out curved in the image with radial distortion. Symmetric radial distortion are categorized as following types:

- Barrel radial distortion (figure 3.9-a)
- Pincushion radial distortion (figure 3.9-b)
- Mustache radial distortion (figure 3.9-c)

The radial distortion is approximated in most applications using a low degree polynomial [20]:

$$\lambda = 1 + \sum_{i=1}^{n} k_i r^{2i}, n \leq 3 \tag{3.30}$$

wherein

$$r^2 := \tilde{u}^2 + \tilde{v}^2. \tag{3.31}$$

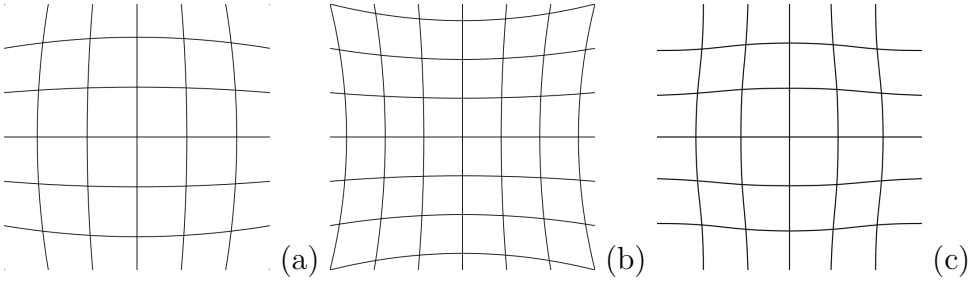

Figure 3.9: Radial distortion.

$\tilde{u}$ and $\tilde{v}$ are the coordinates of an undistorted point in real image coordinate system, calculated using perspective projection from equation 3.21. k_1, k_2, k_3 are coefficients for radial distortion. Applying λ to this point gives the distorted point in real image coordinates $[u_d, v_d]^T$:

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix}. \tag{3.32}$$

Tangential Distortion

Tangential distortion occurs when the lens of the camera is not perfectly parallel to image sensor (cf. figure 3.10). The approximation for this distortion is often

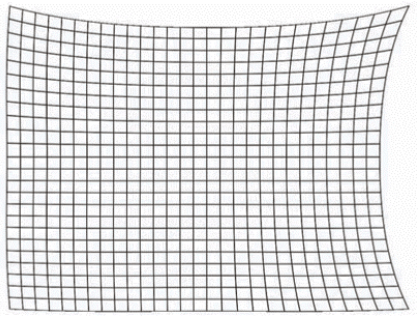

Figure 3.10: Tangential distortion. [21]

written as following [20]:

$$\begin{bmatrix} u_d \\ u_d \end{bmatrix} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} + \underbrace{\begin{bmatrix} 2p_1\tilde{u}\tilde{v} + p_2(r^2 + 2\tilde{u}^2) \\ p_1(r^2 + 2\tilde{v}^2) + 2p_2\tilde{u}\tilde{v} \end{bmatrix}}_{\mathbf{d}} \tag{3.33}$$

wherein p_1, p_2 are coefficients for tangential distortion.

Considering both distortions and combining together 3.32 and 3.33 leads to distorted points u_d, v_d:

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = \lambda \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} + \mathbf{d}. \tag{3.34}$$

Finally, an accurate camera model is the mixture of the pinhole model with corrections for radial and tangential distortions:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix}. \tag{3.35}$$

3.2.4 Camera Parameters Estimation

The common way of camera parameters estimation in machine vision approaches, which also referred to as camera calibration, is using fiducial points (such as checkerboards) whose positions are known in world coordinate system with homogeneous vectors $\mathbf{P}_i$ and finding correspondence for them in the image positions $\mathbf{p}_i$.

$$\mathbf{P}_i = [X_i, Y_i, Z_i, 1]^T, i = 1, ..., n$$
$$\mathbf{p}_i = \begin{bmatrix} u_i, v_i \end{bmatrix}^T$$

The equation 3.17 and enough points gives a set of equations which are solved for camera matrix $\mathbf{M}$.

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{3.36}$$

Each known point i and its correspondence define a pair of equations (3.37). There are 11 unknowns in camera matrix to be calculated therefore at least six points are needed ($n \geq 6$).

$$\begin{cases} u_i = \dfrac{m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} \\ \\ v_i = \dfrac{m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} \end{cases} \tag{3.37}$$

Different methods are proposed to solve this problem such as:

- DLT[5] method [22];
- Zhang's method [23];
- Tsai's method [24].

[5]Direct Linear Transformation

Only the DLT method will be discussed in this study in following chapter. After calculating M, it should be decomposed to get both intrinsic and extrinsic camera parameters.

DLT Method

This linear set of equations are written in a homogeneous set of equations (3.38) as following:

$$u_i(m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}) = m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14}$$
$$v_i(m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}) = m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24}$$

$$\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -u_iX_i & -u_iY_i & -u_iZ_i & -u_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -v_iX_i & -v_iY_i & -v_iZ_i & -v_i \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.38}$$

and for n points it becomes:

$$\underbrace{\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix}}_{\mathbf{A}_{2n\times 12}} \underbrace{\begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix}}_{\mathbf{m}_{12\times 1}} = \underbrace{\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathbf{0}_{2n\times 1}}. \tag{3.39}$$

Obviously, $\mathbf{m}$ should not be equal to zero in order to solve $\mathbf{Am} = \mathbf{0}$ homogeneous set of equations, Therefore this is a total least squares minimization problem.

$$min\|\mathbf{Am}\| \tag{3.40}$$

$\mathbf{m}$ is valid up to scale, so it is assumed to be a unit vector which means its magnitude is one. The solution is to minimize the magnitude of $\mathbf{Am}$ under the constraint $\|\mathbf{m}\| = 1$.

Using SVD[6] it is possible to decompose matrix $\mathbf{A}_{2n\times12}$ to a diagonal matrix $\mathbf{D}_{2n\times12}$ written in decreasing order of absolute values and two orthogonal matrices $\mathbf{U}_{2n\times2n}$ and $\mathbf{V}_{12\times12}$ as below. The columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal bases.

$$\mathbf{A} = \mathbf{UDV}^T \tag{3.41}$$

Plugging this to minimization problem 3.40 gives us:

$$min\left\|\mathbf{UDV}^T\mathbf{m}\right\|. \tag{3.42}$$

Since $\mathbf{U}$ and $\mathbf{V}$ are made up of orthogonal unit vectors multiplying them does not change the magnitude, so:

$$\left\|\mathbf{UDV}^T\mathbf{m}\right\| = \left\|\mathbf{DV}^T\mathbf{m}\right\| \tag{3.43}$$

$$\|\mathbf{m}\| = \left\|\mathbf{V}^T\mathbf{m}\right\| = 1. \tag{3.44}$$

Considering 3.43 minimization problem 3.42 subject to 3.44 becomes:

$$min\left\|\mathbf{DV}^T\mathbf{m}\right\|. \tag{3.45}$$

By substituting of unite vector $\mathbf{y} := \mathbf{V}^T\mathbf{m}$ it becomes:

$$min\|\mathbf{Dy}\| \tag{3.46}$$

subject to $\|\mathbf{y}\| = 1$.

[6]Singular Value Decomposition

Minimization problem 3.46 is minimum when $\mathbf{y} = [0,...,0,1]^T$ because, as mentioned before, $\mathbf{D}$ is a diagonal matrix with decreasing values and this $\mathbf{y}$ puts weight only on the smallest element of $\mathbf{D}$. Since $\mathbf{V}$ is orthogonal and its transpose is its inverse and $\mathbf{y} = \mathbf{V}^T\mathbf{m}$ then:

$$\mathbf{m} = \mathbf{V}\mathbf{y} \tag{3.47}$$

So the solution for $\mathbf{m}$ is the multiplication of $\mathbf{V}$ with $\mathbf{y}$ which pulls out only the last column in $\mathbf{V}$. It is known that the columns of $\mathbf{V}$ are eigenvectors of $\mathbf{A}^T\mathbf{A}$.

Remark.

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$$
$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{V}\boldsymbol{D}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$$
$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{V}\boldsymbol{D}^T\boldsymbol{I}\boldsymbol{D}\boldsymbol{V}^T$$
$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^T$$

Thus, the m vector which satisfies the minimization 3.40 is the eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalues.

3.3 Visual Motion Estimation

Successful path planning and obstacle avoidance in mobile robots which are capable to move around depends on a good pose estimation of the robot in the environment. Using sensors data to estimate change in pose over time is called odometry. Rotary encoders, GPS[7], cameras, laser scanners, inertial sensors and etc are all used before for this purpose. Each of these sensors has its special use cases and both its pros and cons.

For example, rotary encoders are cheap and used often on wheeled robots but they suffer from lack of precision sometimes, because wheels can slip and slide on the floor creating an unreliable traveled distance as compared to actual traveled one.

Rotary encoders can not be used in drone odometry. Instead GPS-based odometry techniques are used in these cases but GPS suffers also from problems like losing significant signal power indoors and low precision. Cameras are also used successfully for visual odometry methods.
According to Chhaniyara [25], "Visual odometry is an image processing technique for incremental, on-line estimation of robot position and velocity from image sequences".

NASA's two MER[8], Spirit and Opportunity, are two successful application examples of visual odometry. They achieved accuracy as small as 2 mm with high rates of successful convergence of 97% on Spirit and 95% on Opportunity in slip ratios as high as 125% while driving on slopes as high as 31 degrees [26]. In visual odometry methods compared to others, neither prior knowledge of the scene nor the motion is necessary.

The fundamental of visual odometry is motion estimation or camera motion tracking. Camera motion tracking also referred as Ego-Motion in literature is the estimation of the transformation between camera and the world coordinate system. There is a large range of possible applications for camera tracking such as SLAM, SfM[9] and AR[10] [27], object tracking, camera stabilization, etc.

The basic idea behind motion estimation methods is to compare successive images iteratively to find the translation and rotation between them. Depend-

[7]Global Positioning System
[8]Mars Exploration Rovers
[9]Structure from Motion
[10]Augmented Reality

ing on the used method, according to [28] motion estimation is categorized as following:

- Direct methods;
- Feature based methods.

State-of-the-art methods combine these two approaches together, wherein feature based approaches are used for long range matches and direct approaches are used for shorter range ones [29]. Also alternative approaches were tried which combine visual and other types of sensors, namely visual-inertial [30] or GPS and stereo-vision odometry [31].

3.3.1 Direct Methods

Direct methods, also referred to as dense methods, recover motion at each pixel using spatio-temporal (location x,y and time t) image brightness variation. This image brightness variation during time is known as optical flow. Optical or optic flow is the apparent motion of an object in the image. Calculating optical flow in whole image characterizes optic flow fields which are used for forming or segmenting a scene.

Figure 3.11 shows a simple scene on the left and its optical flow fields on right. Suppose a camera is viewing this scene parallel to the white rectangle and moving left. The flow fields will be seen as the image on the right. The optical flow on the white rectangle is constant and small. It is constant since the plane is parallel to the image plane. It is small because it is far away from the camera. The optical flow field for the light gray rectangle is also constant but larger because it is near to the camera. On the inclined gray plane, field is small for far points and it is larger for nearby points. As it becomes obvious, because different structures correspond with different flow fields, such flow fields are used for scene segmentation.

These methods are sensitive to appearance variations and they are useful when image motion is small and objects do not move very quickly (higher frame rate). Slow moving assumption guaranties the constancy of brightness for nearby pixels from one frame to another, then it is written:

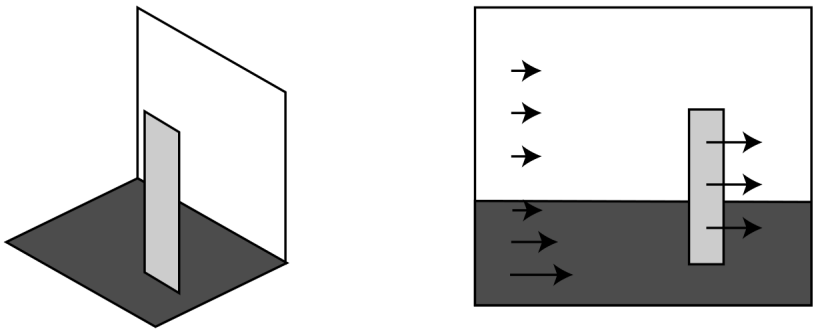

Figure 3.11: A simple scine (left) and optical flow fields for with a camera parallel to white rectangle moving left (right). [18]

$$\mathbf{I}(x,y,t) = \mathbf{I}(x + dx, \quad y + dy, \quad t + dt) \tag{3.48}$$

using Taylor series approximation without higher order terms:

$$\mathbf{I}(x,y,t) \approx \mathbf{I}(x,y,t) + \frac{\partial \mathbf{I}}{\partial x}dx + \frac{\partial \mathbf{I}}{\partial y}dy + \frac{\partial \mathbf{I}}{\partial t}dt \tag{3.49}$$

and finally, replacing derivatives of the image with $\mathbf{I}_x, \mathbf{I}_y, \mathbf{I}_t$ notation and $u = \frac{dx}{dt}, v = \frac{dy}{dt}$ it becomes:

$$\mathbf{I}_x u + \mathbf{I}_y v + \mathbf{I}_t = \mathbf{0}. \tag{3.50}$$

3.50 is known as the brightness constancy constraint equation, in which u and v are changes in location of pixel during the time which determine direction of motion (velocity). The equation 3.50 is written also in gradient form as:

$$(\nabla \mathbf{I})^T . \begin{bmatrix} u \\ v \end{bmatrix} = -\mathbf{I}_t \tag{3.51}$$

therefore direct methods are sometimes referred to as gradient methods.

In order to get the direction of motion, the equation 3.50 should be solved for each pixel to find two unknowns u and v. But one encounter a problem here because there is only one equation with two unknowns. The problem is known as the aperture problem of optical flow algorithms. To overcome this some additional constraints are needed.

Horn and Schunk in [32] have suggested smoothness assumption method by introducing a global energy functional to be minimized as additionally constraint formulated below :

$$e = \int\int [(\mathbf{I}_x u + \mathbf{I}_y v + \mathbf{I}_t)^2 + \gamma(u_x^2 + u_y^2 + v_x^2 + v_y^2)]dxdy \tag{3.52}$$

wherein $u_x = \dfrac{du}{dx}$, $u_y = \dfrac{du}{dy}$, $v_x = \dfrac{dv}{dx}$, $v_y = \dfrac{dv}{dy}$ and γ is a weighting factor. The global motion methods use all pixels in the image to estimate motion, therefore they are used for removing camera motion, object-base segmentation and generating mosaics.

More commonly, local constraints are used to solve this problem. Lucas and Kanade in [33] have suggested least squares method. General assumption in this approach is the optical flow invariance in a local neighborhood of the pixel under consideration. By considering a 3 by 3 window around the pixel, each pixel p_i $(i = 1, ..., 9)$ in the window gives an equation according to 3.50 so there will be an overdetermined system with two unknowns and nine equations to solve. These are written in matrix form as:

$$\underbrace{\begin{bmatrix} \mathbf{I}_x(p_1) & \mathbf{I}_y(p_1) \\ \mathbf{I}_x(p_2) & \mathbf{I}_y(p_1) \\ \vdots & \vdots \\ \mathbf{I}_x(p_9) & \mathbf{I}_y(p_1) \end{bmatrix}}_{\mathbf{A}_{9\times 2}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{\mathbf{d}_{2\times 1}} = \underbrace{\begin{bmatrix} -\mathbf{I}_t(p_1) \\ -\mathbf{I}_t(p_2) \\ \vdots \\ -\mathbf{I}_t(p_9) \end{bmatrix}}_{\mathbf{b}_{9\times 1}} \tag{3.53}$$

In order to solve system of equations 3.53, standard least square method is applied:

$$min \left\|\mathbf{Ad} - \mathbf{b}\right\|^2 \tag{3.54}$$

The matrix **A** is not square, so it is not invertible, therefore pseudo inverse method is used to invert it.

Remark.

$$\underbrace{(\boldsymbol{A}_{2\times 9}^{T}\boldsymbol{A}_{9\times 2})}_{\boldsymbol{G}}\,\boldsymbol{d}_{2\times 1} = \boldsymbol{A}_{2\times 9}^{T}\boldsymbol{b}_{9\times 1}$$

$$\boldsymbol{d}_{2\times 1} = (\boldsymbol{A}^{T}\boldsymbol{A})_{2\times 2}^{-1}(\boldsymbol{A}^{T}\boldsymbol{b})_{2\times 1}$$

So, direction of motion computes:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \underbrace{\begin{bmatrix} \sum_i I_x(p_i)I_x(p_i) & \sum_i I_x(p_i)I_y(p_i) \\ \sum_i I_x(p_i)I_y(p_i) & \sum_i I_y(p_i)I_y(p_i) \end{bmatrix}^{-1}}_{(\mathbf{A}^T\mathbf{A})^{-1}} \underbrace{\begin{bmatrix} -\sum_i I_x(p_i)I_t(p_i) \\ -\sum_i I_y(p_i)I_t(p_i) \end{bmatrix}}_{\mathbf{A}^T\mathbf{b}} \tag{3.55}$$

So far, it is assumed that the motion is small, but it is not always the case. If it is larger than a pixel then first order Taylor series approximation will not hold. Solution to this comes from hierarchical L-K[11] (Coarse-to-fine) method. It uses Gaussian pyramid for downsampling each image. This makes the motion small enough to be tracked. Figure 3.12 shows two successive frames **H** and **I**, downsampled in 3 levels. Then it continues with running L-K between two smallest images in level 3 which produces a flow field of moving pixels. In the next step it upsamples this flow field and applies it to **H** in level 2 then warps the result which gives the image I in level 2 with a little small differences. Finally it runs L-K again between warped and image **I** in level 2. This is done iteratively until to reach the original image in level 0.

Brightness constancy is assumed in direct methods, but this assumption does not hold always due to change in illumination of the scene or object reflection. In addition, the motion could be large and also a point may not move like its neighbors. These conditions cause problems for these methods. However, it

[11] Lukas and Kanade

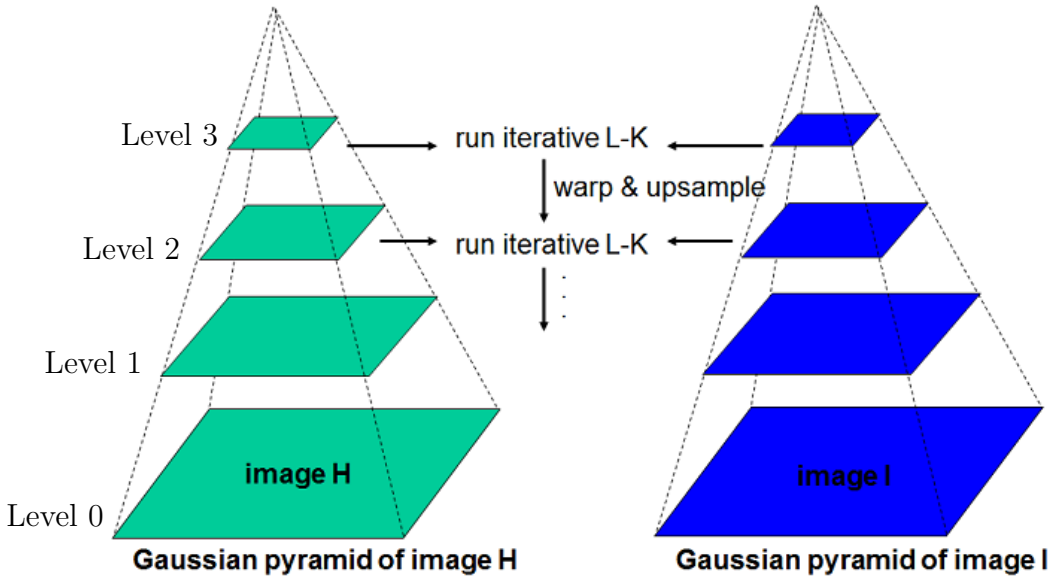

Figure 3.12: Hierarchical Lukas and Kanade Gaussian pyramid. [34]

is possible to use features in the image to solve this problems which will be discussed in next section.

3.3.2 Feature Based Methods

Feature based methods, also known as indirect methods, extract a set of specific features from each image and determine the motion by finding correspondences, in contrast to the direct methods which determine motion in each pixel of the image [28]. All parts of an image does not contain useful information about motion, for example it is only possible to recover motion orthogonal to the straight edges. Accordingly, it makes sense to use features in the regions with a rich enough texture. These methods are robust tracking methods and suitable for object tracking tasks when its motion is large (more than 10 to 20 pixels). One of the most known feature based methods is KLT[12] tracker algorithm [35]. KLT tracker algorithm (similarly to direct method) calculates matrix

$$\mathbf{G} = \mathbf{A}^T\mathbf{A}$$

from equation 3.55 for each pixel with a window around it, but it keeps only some of them (contrary to direct method) which are useful for solving the equation system. System 3.55 is solvable when 2×2 matrix $\mathbf{G}$ is invertible. This means that it should have a low condition number κ.

$$\kappa(\mathbf{G}) := \frac{\lambda_{max}}{\lambda_{min}}$$

wherein $\lambda_{\max}$ and $\lambda_{\min}$ are maximal and minimal eigenvalues of $\mathbf{G}$ respectively. Low condition number is said to be well-conditioned, if it is not the inverse is unstable. It implies that it is not possible for eigenvalues to have a very large magnitude difference. Also, to decrease noise effect they must be large enough. Two large eigenvalues demonstrate existence of different gradients within a window, which represent features that are trackable reliably like corners, salt and pepper textures, or any other. Two small eigenvalues mean a non constant brightness within a window. A large and a small eigenvalue correspond to a unidirectional pattern which is not suitable for tracking.

In practice, when the smaller eigenvalue is larger than a threshold the noise condition is satisfied and the matrix $\mathbf{G}$ is also usually well-conditioned, because the brightness variation in a window is limited by the maximum allowable pixel value, so the greater eigenvalue can not be arbitrarily large. To determine threshold for the camera to be used during tracking, the eigenvalues for images of a region with approximately uniform brightness should be measured.

[12] Kanade-Lucas-Tomasi

Algorithm 1 gives an example for choosing the best feature for KLT tracker.

Algorithm 1: KLT feature selection

1. Compute the **G** matrix and its minimum eigenvalue λ_{min} at every pixel in the image.
2. Call λ_{max} the maximum value of λ_{min} over the whole image.
3. Keep the image pixels that have a λ_{min} value larger than a threshold.
4. Keep the local maximum pixels (a pixel is kept if its λ_{min} value is larger than that of any other pixel in its 3×3 neighborhood).
5. Keep the subset of those pixels so that the minimum distance between any pair of pixels is larger than a given threshold distance.

4 Approach

The main goal of the approach presented in this thesis is to develop a method to determine the pose (extrinsic parameters) of a 3D camera with respect to the base coordinate system of a mobile robot. We assume in this chapter that the base is at the mass center of the robot. The pose of a camera in 3D space is described by a translation vector $\mathbf{t} = [X, Y, Z]^T$ and a rotation matrix $\mathbf{R} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi)$ with respect to a reference point (cf. figure 4.1). Therefore, there are six parameters (6 DoF) to be determined:

- There elements of the translation vector: X, Y, Z;
- Three angles of rotation matrix: Roll(ϕ), Pitch(θ), Yaw(ψ).

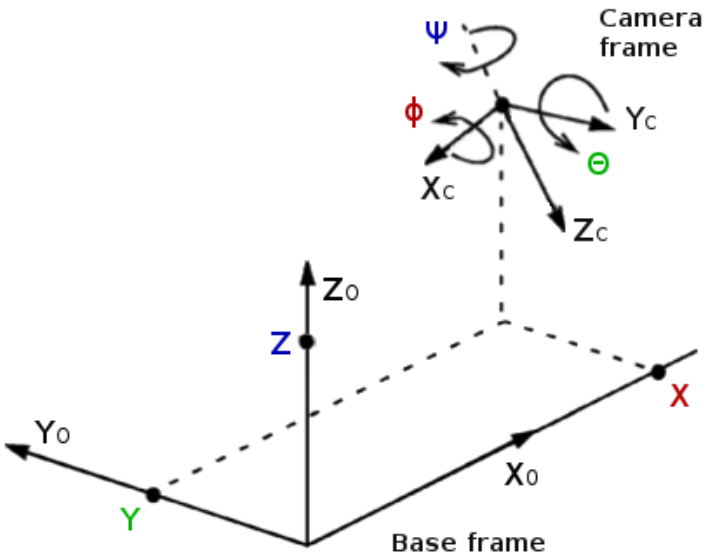

Figure 4.1: Pose of a camera in 3D space with respect to base frame of the robot.

These six parameters will be calculated in three steps as following:

1. Ground plane detection (section 4.1);
2. Two-rotation drive (section 4.2);
3. Straightforward drive (section 4.3).

Figure 4.2 shows an overview of these three steps and relation between sub steps in the algorithm. After obtaining the image sequence from the camera the algorithm start with detecting ground plane using RANSAC[1] [36] in first step for determination of Z, ϕ and θ parameters. Then it uses these calculated roll and pitch angels in two-rotation drive step for calculation of X and Y parameters using a camera tracker for determination of camera trajectory during robots movements in two circular paths. Finally, in last step straightforward drive it uses the same camera tracker during a straight line movement of robot for determination of ψ parameter.

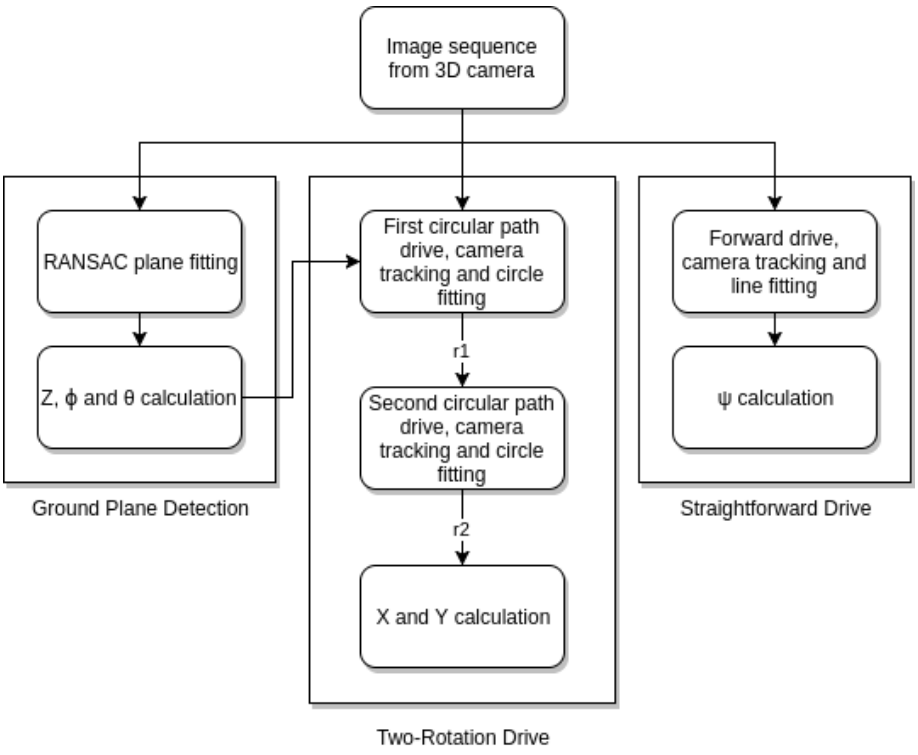

Figure 4.2: Overview of the different steps in the approach.

[1]Random Sample Consensus

4.1 Ground Plane Detection

It is assumed that the robot is working on a flat floor. Detecting the ground plane in FoV[2] of the camera, is enough to calculate the parameters of the first step. The segmentation algorithm finds all the points within a point cloud that support a plane model using RANSAC as a robust estimator of choice. A threshold for distance determines how close a point must be to the model in order to be considered as an inlier. Finally the contents of the inlier set, are used to estimate coefficients of the plane equation in 3D space:

$$n_x x + n_y y + n_z z + d = 0 \tag{4.1}$$

wherein d represents the distance between plane and the camera, which is equivalent to the distance of the camera from the ground which is the height of the camera defined by Z parameter, therefore:

$$Z = d. \tag{4.2}$$

The vector $\mathbf{n}^T = [n_x,n_y,n_z]$ represents the normalized normal vector of the plane which is perpendicular to the surface. The formed angles between this normal vector and the coordinate system of camera provides roll and pitch angles in this step, which is calculated simply using trigonometry as illustrated in figure 4.3.

The pitch angle of camera θ is equal to the angle between the normal vector of the ground plane $\mathbf{n}$ and the $x_c - y_c$ plane of camera coordinate system, so it is calculated with:

$$\theta = \arctan\left(\frac{n_z}{n_y}\right). \tag{4.3}$$

The roll angle of camera ϕ is equal to the angle between the normal vector of the ground plane $\mathbf{n}$ and the $y_c - z_c$ plane in the camera coordinate system, so it is calculated with:

$$\phi = \arctan\left(\frac{n_x}{n_y}\right). \tag{4.4}$$

Hence all three parameters in this step have been determined.

[2]Field of View

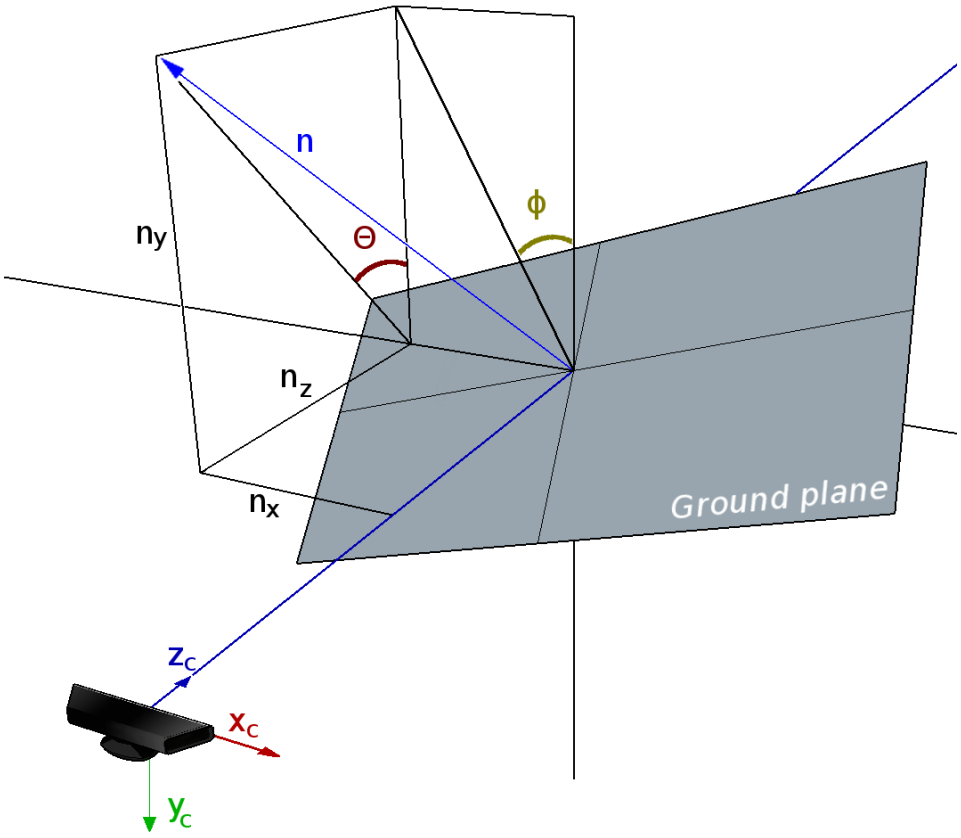

Figure 4.3: Roll(ϕ), Pitch(θ) with respect to detected ground plane in camera coordinate system.

4.2 Two-Rotation Drive

This method is used to calculate the second step parameters, including X and Y distances of the camera in the robot base coordinate system. In this step, the robot rotates along two circular paths with different radiuses. If the trajectory of the camera is determined during these rotations, then X and Y distances are calculated using simple geometry. To obtain the camera trajectory, the V4R[3] camera tracker implemented. This tracker will be discussed with more detail in section 5.1.

The camera tracker provides these transformations from camera perspective in the base frame of the robot, which is selected as reference frame. Before any calculation is started the camera trajectory should be transformed to compensate for roll and pitch angels that have been found in previous section, since the camera coordinate and robot base coordinate systems are rotated respectively.

When the robot drives two times with circular path, the camera also has circular movement trajectories with respect to the center of rotation. For the first drive, if the rotation radius is selected to be zero then the robot rotates exactly around itself on a spot and the center of rotation will be equal to the base origin. For the second drive, if the rotation radius is selected to be half of the distance between two wheels then it will rotate exactly around one of the wheels. This keeps one of the wheels fix in place and reduces movement error and noise production from the motor during the drive. Figure 4.4 shows a robot from a top view with two differential-drive wheels in the base coordinate system and two camera trajectories that would be taken during the first and second circular path drive by the camera mounted on it. The yaw angle is irrelevant here because it does not affect the camera trajectory during a circular movement of the robot (cf. figure 4.4).

The camera trajectory is a set of points $\mathbf{p}$ in 3D space. Since the camera height is fixed and it has been calculated in the previous section, trajectory data in the z direction is irrelevant here. r_1 and r_2 radiuses are calculated by applying 2D circle fit algorithm on this set of points. This algorithm will be explained in following section 4.2.1.

After calculation of r_1 and r_2, the two camera trajectories equations in the $x - y$ plane are written as below:

[3]Vision for Robotics group, ACIN, TU Vienna

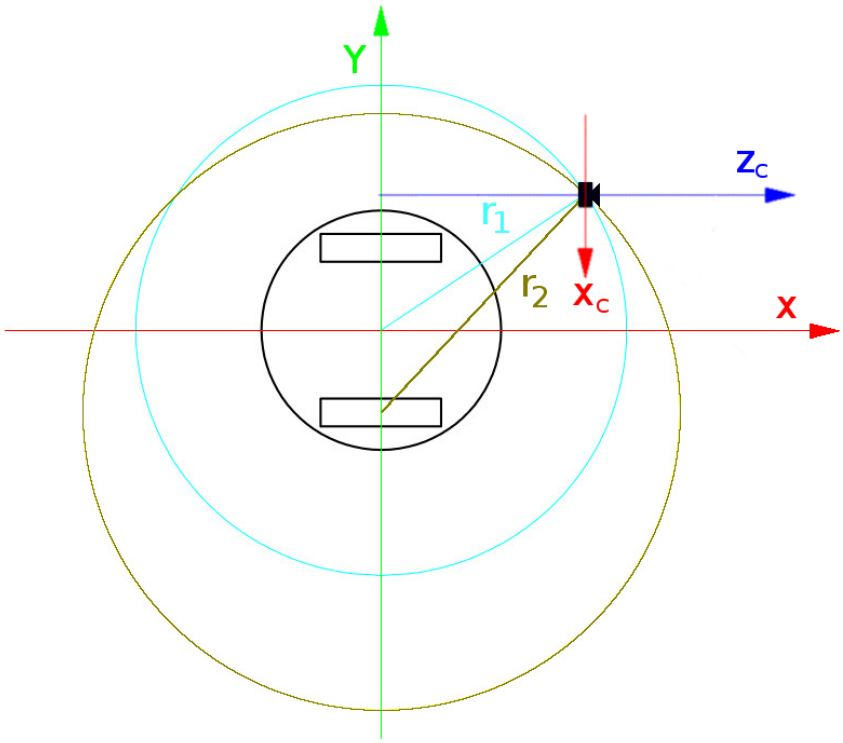

Figure 4.4: Top view of a robot with two differential-drive wheels in the base coordinate system and a 3D camera on board for Two-Rotation drive.

$$x^2 + (y - r_F)^2 = r_1^2 \tag{4.5}$$

$$x^2 + (y - r_S)^2 = r_2^2 \tag{4.6}$$

with r_F and r_S are the first and second drive radiuses. X and Y distances are calculated by solving two circles equations for an intersection point as the flowing:

$$Y = \frac{r_1^2 - r_2^2 + r_S^2 - r_F^2}{2(r_S - r_F)} \tag{4.7}$$

$$X = \sqrt{r_1^2 - (Y - r_F)^2}. \tag{4.8}$$

It is obvious from equation 4.7 that r_F and r_S radiuses could be chosen arbitrary but they should not be equal, because division by zero is undefined.

It is assumed that the camera is looking forward on the robot, therefore the calculated negative value for X will be discarded. Hence the two unknown parameters of this step have been determined.

4.2.1 Circle Fit Algorithm

The algorithm is an implementation of direct least squares fitting a circle to 2D points in [37]. The goal is to fit a set of points with a circle equation:

$$(x - a)^2 + (y - b)^2 = r^2 \tag{4.9}$$

where $[a,b]^T$ is the circle center and r is the circle radius, which is important in calculation of X and Y. The error function to be minimized for n points in set is:

$$E = \sum_{i=1}^{n} (L_i - r)^2 \tag{4.10}$$

where $L_i = \sqrt{(x_i - a)^2 + (y_i - b)^2}$. Setting to zero of partial derivatives of equation 4.10 with respect to a, b and r leads to:

$$r = \frac{1}{m} \sum_{i=1}^{n} L_i \tag{4.11}$$

$$a = \frac{1}{m} \sum_{i=1}^{n} x_i + r \frac{1}{m} \sum_{i=1}^{n} \frac{\partial L_i}{\partial a} \tag{4.12}$$

$$b = \frac{1}{m} \sum_{i=1}^{n} y_i + r \frac{1}{m} \sum_{i=1}^{n} \frac{\partial L_i}{\partial b}. \tag{4.13}$$

These equations are solved using fixed-point iteration to obtain radius r and center of the circle.

4.3 Straightforward Drive

This method is used to calculate the yaw angle of the camera in the last step. For this calculation the robot starts driving straightforward for a short distance ($\approx$ 10 cm) while the camera tracker is providing the camera trajectory. As shown in figure 4.5, considering camera trajectory with respect to the camera coordinate system forms a line in the $z_c - x_c$ plane. Calculation of the angle between z_c axis of the camera and this line yields to the yaw angle. In order to get the slope of the trajectory, a line fit algorithm is applied to the set of camera trajectory points.

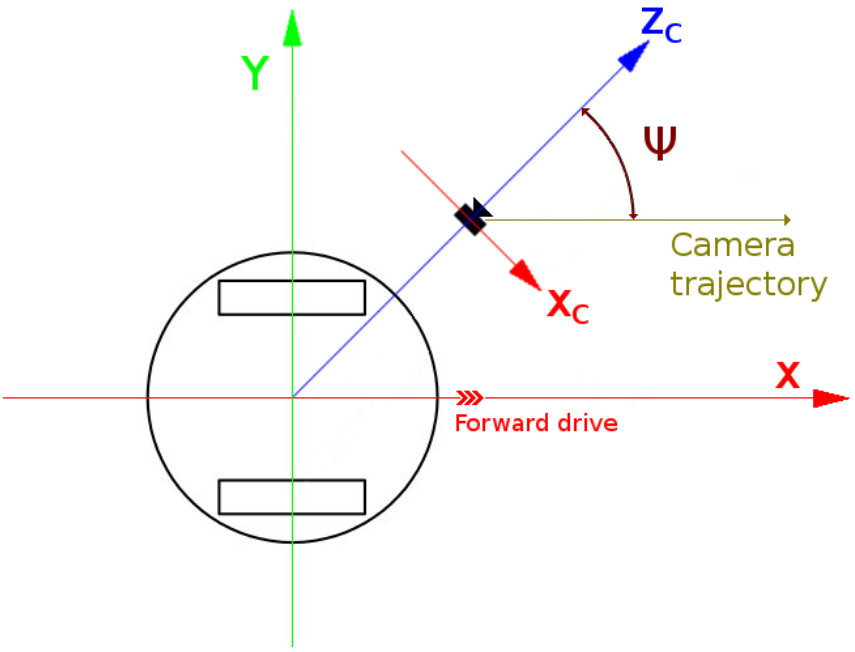

Figure 4.5: Top view of a robot with two differential-drive wheels in the base coordinate system and a 3D camera on board for straightforward drive.

The line fit algorithm will be explained in the following section 4.3.1. Finally, the yaw angle is calculated with following equation, wherein A is the line slope from fitting algorithm:

$$\psi = \arctan(A). \tag{4.14}$$

Hence yaw angle parameter in this step have been determined.

4.3.1 Line Fit Algorithm

The algorithm is an implementation of linear fitting of 2D points in [37]. The goal is to fit a set of points with a line equation:

$$y = Ax + B. \tag{4.15}$$

The error function to be minimized is sum of the squared errors between the y values and the line values (only in y-direction).

$$E = \sum_{i=1}^{n} [(Ax_i + B) - y_i]^2 \tag{4.16}$$

Setting gradient of equation 4.16 to zero leads to a system of two linear equations:

$$\begin{bmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & n \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} \begin{bmatrix} \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} y_i \end{bmatrix} \tag{4.17}$$

which is solved to obtain A and B.

$$A = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i} \tag{4.18}$$

$$B = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i} \tag{4.19}$$

5 System Description

The method is implemented using C++ and Python in ROS. This chapter gives information about implemented nodes and services and describes available settings in launch files in order to control the calibration process.

5.1 V4R Camera Tracker Node

A camera tracking algorithm provides us the trajectory of the camera in 3D space. This trajectory is determined using the camera tracker of V4R-library [38]. The method combines two approaches:

- Feature-based method using a pyramidal implementation of the KLT-tracker as discussed in section 3.3.2;
- A keyframe-based refinement step.

The algorithm detects FAST[1]-keypoints [39] first to initialize a keyframe and assign them to the corresponding 3D locations. Then it tracks, frame by frame, the keypoints using pyramidal KLT-tracker, which allows tracking large camera motions. Finally, it uses RANSAC to robustly estimate the rigid transformation (the camera pose) from the corresponding depth information of the organized RGB-D frames. Additionally, it applies a keyframe-based refinement step by projecting patches to the current frame to account for the accumulated drift for individual point correspondences and optimizing their locations.

In more detail, when a keyframe is created, normals are estimated and in combination with estimation of the camera pose a locally correct patch warping from the keyframe to the current frame is generated. Then a KLT-style refinement step including the normalized cross correlation of the patches gives sub-pixel-accurate image locations for bundle adjustment and ability to reduce the drift while tracking. This tracker is able to model large environments because keyframes are generated depending on the tracked camera pose which

[1] Features from accelerated segment test

makes it suitable for our case in this study.

This algorithm produces as output:

- A set of keyframes $K = \{K^1, ..., K^n\}$;
- A set of transformations $T = \{T^1, ..., T^n\}$.

for camera pose adjusting the corresponding keyframes to the reference frame which is defined by the first camera frame or by user. [40]

5.2 Camera Tracker Simulator Node

Since there is no guarantee that camera tracker performs accurately and it does not perform well in the simulation environment due to lack of traceable features, it is not possible to use camera tracker in order to execute proof-of-concept experiments on the method. Therefore, an additional node is added to simulation in order to simulate the behavior of the camera tracker. It calculates camera trajectory based on the tf tree of the robot and publishes the trajectory of it while the robot moves.

The node looks up for transformation between *world_frame* and *camera_link* frame and takes the first pose of the camera as initial pose, then in each iteration of the node it calculates the difference between the current pose of the camera and previous one. This defines the trajectory of the camera in world coordinate system. This trajectory should be rotated with respect to roll, pitch and yaw angles of the *camera_link*, which are obtained by looking up the transformation between *camera_link* and *base_link* frames. And finally, the trajectory should be rotated again to compensate for rotation of axes in camera coordinate system compare to reference coordinate system (cf. figure 4.1).

5.3 Planar Segmentation Node

In order to detect the ground plane, planar segmentation algorithm, which is an open source software from PCL[2], has been used. The PCL is a large scale, open source project for 2D/3D image and point cloud processing. The PCL framework contains numerous state of the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation [41].

[2]Point Cloud Library

5.4 Calibration Node

The calibration node has two lunch files. One for running on the real robot, and the other for simulation environment. The reason for this is, as mentioned before (section 5.2), the camera tracker which does not perform properly in simulation. These files also contain necessary settings for the calibration process. Table 5.1 describes these settings. The rotation radius in settings only defines the second circle radius in two-rotation drive (0.158 m is the half of distance between two wheels of the V4core robot) because the radius for the first circle is defined as zero so the robot rotates on a spot. When the calibration process is finished the node publishes calculated parameters (X, Y, Z, roll, pitch, yaw) to ROS parameter server using name space */v4r_core_sensor_calibration/...* and if the *params_to_file* option has been set to true in the launch files, it saves them in a file.

The launch files also bring up other necessary nodes during the calibration such as the planar segmentation and the marker publisher. Finally, they call the drive node which has the duty of executing three main steps of the algorithm for calibration process.

5.5 Drive node

The drive node controls the whole actions of the robot during the calibration process. It commands the robot to drive forward by publishing linear twist messages in X direction and calls the camera tracker *start_tracking* service by sending the request message and awaiting for reply. After short amount of time ($\approx$ 3 seconds) it stops the robot and calls for *stop_tracking* service. Afterwards it calls for *yaw_calculation* service provided by calibration node for calculation of the yaw parameter. Similar routines are done for two-rotation drive parts. The node publishes angular twist messages in Z direction and in order to detect circle path completion, it subscribes to odometry message of the robot and calculates the difference between initial yaw angle of the robot at starting moment of the drive with current angle of it during the drive. It continues the rotation until the difference between them is bigger than 0.1 radian. Then it stops the rotation. This method is not very precise but it enough for this case because the algorithm fits a circle to the trajectory and it does not mater if the circle is perfectly close or not completed. *first_circle_calculation* and *second_circle_calculation* services are called respectively. Table 5.2 describes all the available services in the method.

Argument	Default value	Description
camera	camera_top	The camera topic for the one which will be calibrated.
straight_speed	0.1	Linear speed of the robot in straightforward drive section.
rotation_speed	0.1	Angular speed of the robot in two-rotation section.
rotation_radius	0.158	The radius of the circle for the second rotation in two rotation drive.
params_to_file	false	If true, saves the calculated calibration parameters to */tmp/v4r_core_calibration_params.yaml* file.

Table 5.1: Launch files arguments for calibration node.

Node	Service	Description
calibration	first_circle_calculation	Fits a circle to the current camera trajectory with method described in section 4.2.1.
	second_circle_calculation	Fits a circle to the current camera trajectory and calculates the X and Y parameters with method described in section 4.2.
	yaw_calculation	Fits a line to the current camera trajectory and calculates the yaw angle with method described in section 4.3.1.
camera_tracker	start_tracker	Starts the camera tracking.
	stop_tracker	Stops the camera tracking.

Table 5.2: ROS services of the nodes.

6 Performance Evaluation

This chapter evaluates the proposed method on both computer simulation and real-life scenarios with an experimental setup and presents gathered data from the experiments and their analyze.

6.1 Experimental Setup

This section explains the used experimental setup for testing the proposed method and rviz environment used to debug the method.

6.1.1 V4core Platform

The V4core mobile robot platform is used for testing the presented method and obtaining data in real-life scenarios. V4core is a mobile robot system for research and development based on a Pioneer P3-DX [42] platform. Figure 6.1 shows V4core robot that is equipped with three 3D cameras looking to the floor in front of it. These three are called top, middle and bottom cameras mounted respectively at 1.33 m, 0.75 m and 0.44 m from the floor. They are mounted at different poses to get versatile results during the experiments.

6.1.2 Ground Truth Measurement

Ground truth for X, Y and Z lengths of these cameras are measured manually using tape measure and laser measuring tool and the ground truth angels are measured by cameras looking at a fiducial marker (figure 6.2) that is fixed in the environment of the robot.

fiducials [43] ROS package detects this kind of markers by the *aruco_detect* node using corner detection. For each marker visible in FoV of camera it dedicates a coordinate frame. The pose of this fiducial frame relative to the camera frame can be estimated by knowing the intrinsic parameters of the camera and the size of the fiducial. The image coordinates of each detected

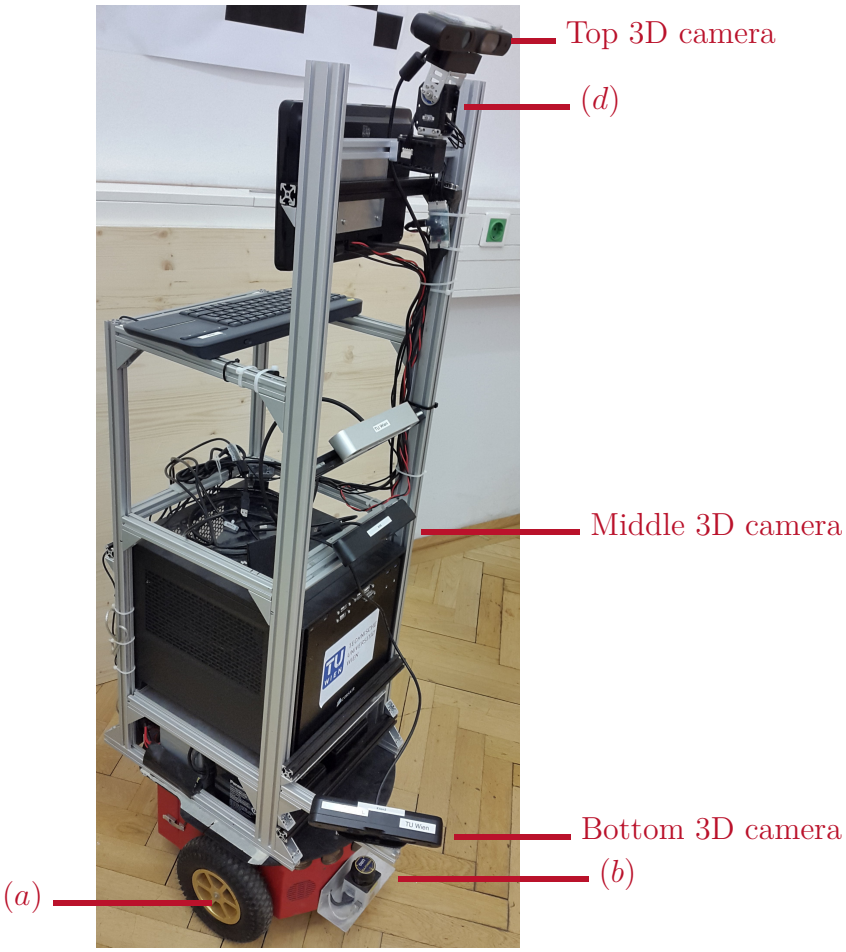

Figure 6.1: V4core mobile robot system for research and development based on a Pioneer P3-DX. It is equipped with (*a*) Two differential-drive wheels, (*b*) Hokuyo URG-04LX scanning laser in front, (*3D cameras*) ASUS Xtion PRO LIVE, (*d*) a pan-tilt unit for the top camera.

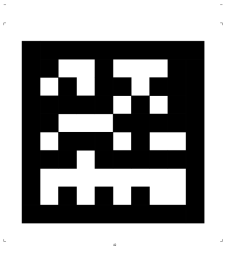

Figure 6.2: Fiducial marker.

corner corresponds to a ray in space definable by a linear equation. The pose estimation code solves a set of these equations to determine the transform from coordinate frame of the fiducial to the camera's coordinate system.

Once the marker is placed on the floor and aligned exactly with base coordinate system of the robot, the calculated transform gives the roll, pitch and yaw angles of the camera in base coordinate system (cf. figure 6.3).

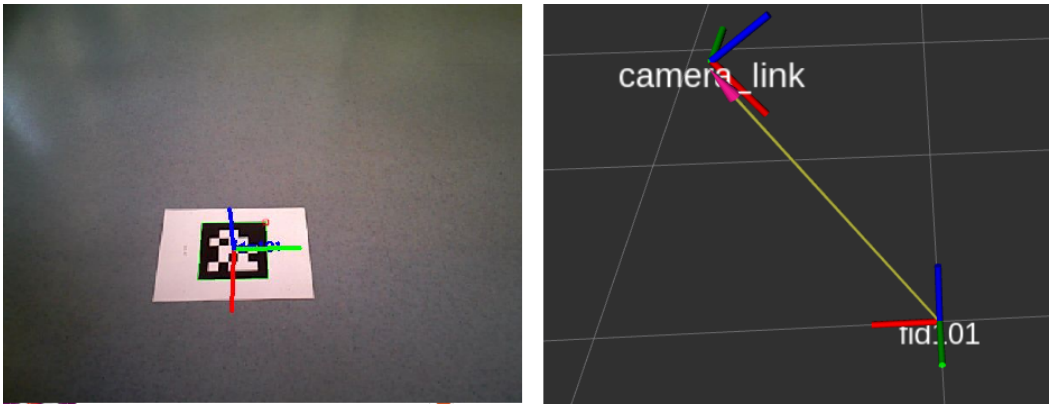

Figure 6.3: Fiducial marker detected in RGB image (left) and the calculated transformation displayed in rviz (right) from marker frame *fid_01* to camera frame *camera_link*.

The ground truth data is measured for the three cameras mounted on the V4core and the results are shown in table 6.1.

Camera	**X(m)**	**Y(m)**	**Z(m)**	**Roll(°)**	**Pitch(°)**	**Yaw(°)**
Top camera	0.173	0.020	1.333	0.0	51.6	0.0
Middle camera	0.115	0.020	0.755	0.2	34.4	0.2
Bottom camera	0.280	-0.150	0.440	1.1	26.9	-29.8

Table 6.1: Ground truth data for three cameras in experiments.

6.1.3 Gazebo Simulation

Gazebo simulator makes it possible to rapidly test algorithms without depending on real machine using an URDF[1] model of the robot. In order to build a model of the V4core robot an already existing model of the Pioneer P3-DX [44] has been modified and gazebo plugins are used to give more functionality to the model. These plugins connect ROS with Gazebo to get sensor readouts from simulation environment and send motors control signals to it. For V4core model, two depth camera plugins are added to the Pioneer model at the same position as their real world positions (Figure 6.4). The top camera is mounted on a pan-tilt unit on the robot, therefore two revolute joints, each with one degree of freedom (around Z axis for panning and around Y axis for tilting), are also added to the joint of top camera and transmission elements are used to describe the relationship between motors and these joints in the model, which allow ROS controlling the motors during simulation.

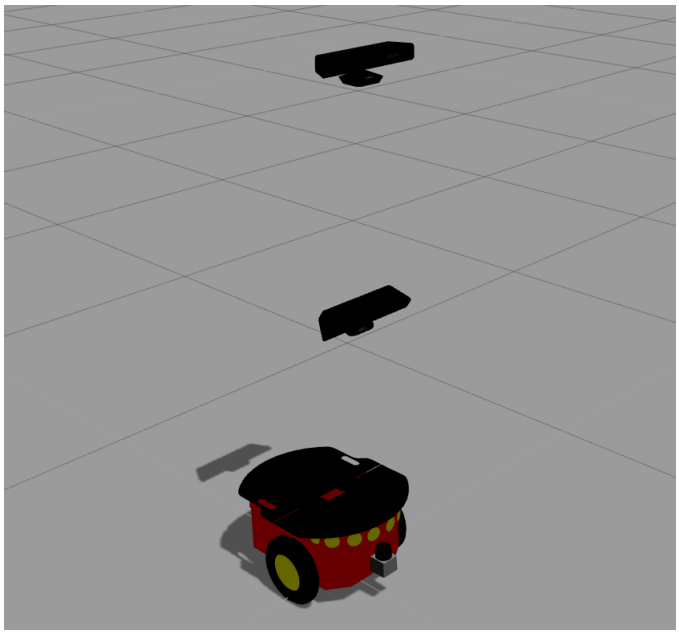

Figure 6.4: GAZEBO simulation of the V4core robot.

[1] Universal Robotic Description Format

6.1.4 Rviz

Rviz is a 3D visualization environment to display state information and sensor data such as camera images, point cloud, laser measurements and path planning from view point of the robot in ROS. There are also marker messages to send rviz data in order to visualize and compare and have better insight of what is going on during an operation of the robot. These markers are added to implantation of the method for visualizing different steps in marker publisher node in order to make debugging process easier and less time consuming.

Marker Publisher Node

The marker publisher node uses *visualization_msgs/Marker* massages to visualize different information from the method in rviz. Figure 6.5 illustrates these markers in rviz after a calibration method is completed and table 6.2 gives more detail about all available markers.

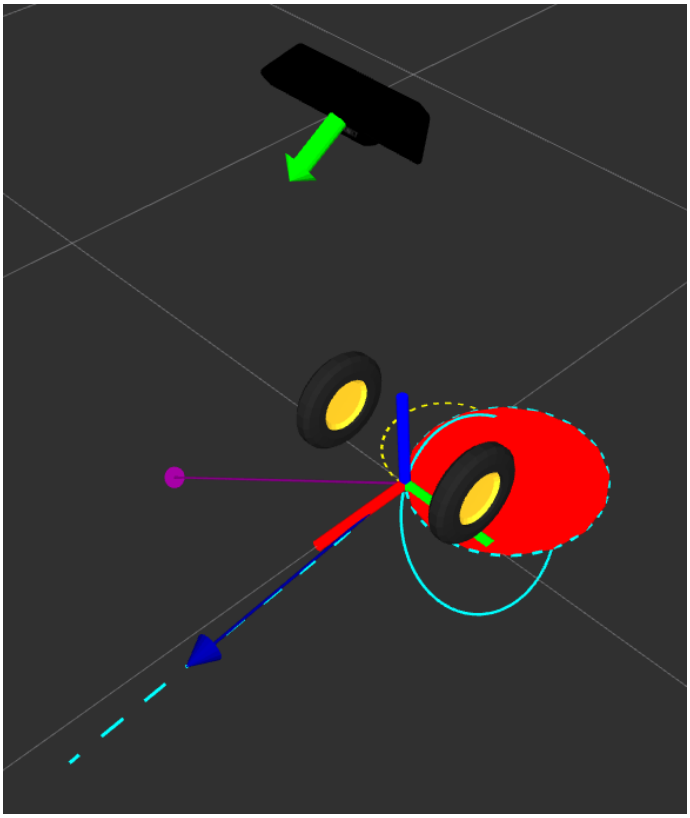

Figure 6.5: Markers in rviz.

Topic	Namespaces	Color	Description
/camera_tracker	trajectory	Green	Trajectory of the camera in reference coordinate system.
	trajectory_simulation	Blue	Trajectory of the camera from camera tracker simulator.
/camera_pose	circle	Red	Fitted circle to the projected camera trajectory during rotation drives.
	line	Dark blue	An arrow showing fitted line to projected camera trajectory during straightforward drive.
	plane_normal	Purple	An arrow showing normal vector of detected ground plane.
	camera_pose	Green	An arrow showing estimated position of the camera.
/camera_tracker_projection	line	Blue	Dashed line showing camera trajectory during straightforward drive after roll and pitch angles compensation.
	circle_1	Yellow	Dashed line showing camera trajectory during first rotation drive after roll and pitch angles compensation.
	circle_2	Blue	Dashed line showing camera trajectory during second rotation drive after roll and pitch angles compensation.

Table 6.2: Rviz markers description.

6.2 Experiments and Results

This section explains the outcome of the experiments and introduces a factor to compare between different results.

6.2.1 Pose Estimation Quality Factor (PEQF)

As a means of comparison between the calibration results of the same cameras in different scenarios, PEQF is defined as the average of six absolute values of pose estimation parameters' error each divided by the absolute value of correspondence ground truth parameter.

$$PEQF = \frac{1}{6}\left(\left|\frac{X_{error}}{X_{GT}}\right| + \left|\frac{Y_{error}}{Y_{GT}}\right| + ... + \left|\frac{\theta_{error}}{\theta_{GT}}\right| + \left|\frac{\psi_{error}}{\psi_{GT}}\right|\right) \tag{6.1}$$

In order to avoid division by zero in cases that the ground truth is zero a very small number (0.001) is used. The smaller PEQF gets the more accurate pose estimation result is (less error in the pose estimation).

6.2.2 Simulation

Ground truth data is used to build a simulation model of the robot in Gazebo simulation environment as described in section 6.1.3 in which the proof-of-concept tests are conducted. Since the camera tracker is not able to work properly in simulation environment, camera tracker simulator which has been described in section 5.2 is used during these experiments. Simulation is ran for each of three cameras and their poses are estimated. The difference between ground truth data and estimated pose are calculated to get error. The results showed that the pose estimation error for translation parameters (X, Y, Z) was from 0 mm to 3.3 mm and for rotation parameters (ϕ, θ, ψ) was from 0° to 0.5° (cf. figures 6.7 – 6.12).

The outcome proved functionality of the method under ideal circumstances of the simulation considering almost zero pose estimation error for all three cameras. The source of small errors in the simulation results is approximation in the line and circle fitting algorithms. PEQF for pose estimations in simulation are calculated as 0.01, 0.02 and 0.02 respectively for bottom, middle and top cameras. The resulted data from the experiments are available in tables 6.3 – 6.5.

6.2.3 Real-Life Scenarios

For real-life scenarios, two kinds of experiments are conducted several times with V4core robot for each of the three cameras. First, the method is tested in an area with mosaic floor structure. Secondly, the same experiments are done in an area with wooden floor structure. Figure 6.6 shows two samples of these areas. The whole process of calibration for a camera took under three minutes each time. The pose estimation error is calculated as difference between

Figure 6.6: Mosaic and wooden floor structure.

ground truth data and the mean value of results in both area similar to the previous section (cf. tables 6.3 – 6.5). Following graphs give more clear insight of experiments' results by illustrating data in the tables.

Results

Figures 6.7 and 6.8 illustrate the pose estimation error and its SE[2] of the nearest camera to the floor which is the bottom camera mounted at 44 cm from the floor. The estimation error amount for the X parameter in the wooden floor and mosaic floor is around 6 mm. The error of the Y parameter is approximately 5 mm in the wooden floor and the mosaic floor. For the estimation of the Z it can be seen that the error is close to 3 mm for wooden floor and around 7 mm for mosaic floor. The rotation parameters graph shows 0.3° and 0.4° error for the roll parameter estimation respectively for wooden floor and mosaic floor areas. For the pitch parameter, examined in the wooden floor, error is 0.3° and for the mosaic floor 0.1°. For the yaw parameter this amount is 0.3° for wooden floor and 0.2° for mosaic floor in both areas.

[2]Standard Error

Overall, the estimations for the bottom camera is considered as accurate in both of the wooden floor area with PEQF of 0.1 and the mosaic floor area with the same PEQF. All errors are under 1 cm for the translation parameters and under 0.5° for the rotation parameters.

Bottom camera	**X(m)**	**Y(m)**	**Z(m)**	**Roll(°)**	**Pitch(°)**	**Yaw(°)**
Ground truth	0.280	-0.150	0.440	1.1	26.9	-29.8
Simulation	0.280	-0.147	0.439	1.1	26.8	-29.8
Simulation error	0.000	0.003	0.001	0.0	0.1	0.0
PEQF = 0.01						
Mosaic floor						
Data mean	0.286	-0.155	0.447	0.7	26.9	-30.0
Error mean	0.006	0.005	0.007	0.4	0.1	0.2
STDV	0.003	0.003	0.004	0.3	0.1	0.0
SE	0.001	0.001	0.001	0.1	0.0	0.0
PEQF = 0.1						
Wooden floor						
Data mean	0.274	-0.153	0.443	1.2	26.7	-29.5
Error mean	0.006	0.005	0.003	0.3	0.3	0.3
STDV	0.004	0.004	0.002	0.2	0.3	0.0
SE	0.001	0.001	0.001	0.1	0.1	0.0
PEQF = 0.1						

Table 6.3: Bottom camera pose estimation data.

Figures 6.9 and 6.10 show the pose estimation error and its SE for the middle camera. It is observable that estimation error of the X in the wooden floor has the highest error around 45 mm while this amount in the mosaic floor is close to 18 mm. Estimation error of the Y in general has the lowest difference with ground truth among the other translation parameters with 8 mm and 5 mm. Estimation error of the Z is around 13 mm for the wooden and mosaic floor. The roll parameter estimation error in the wooden floor is around 0.7° and in the mosaic floor is about 0.3°. The highest error is for the pitch parameter in the wooden floor, close to 2.7° and 2.4° in the mosaic floor. Estimation error for the yaw parameter through both areas resulted in almost the same number

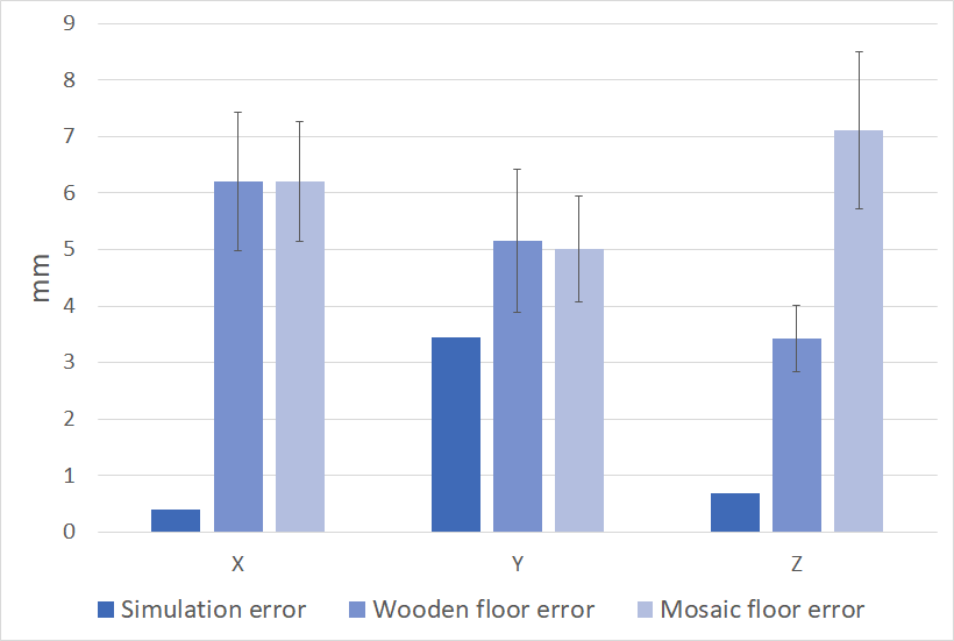

Figure 6.7: Bottom camera translation parameters estimation error.

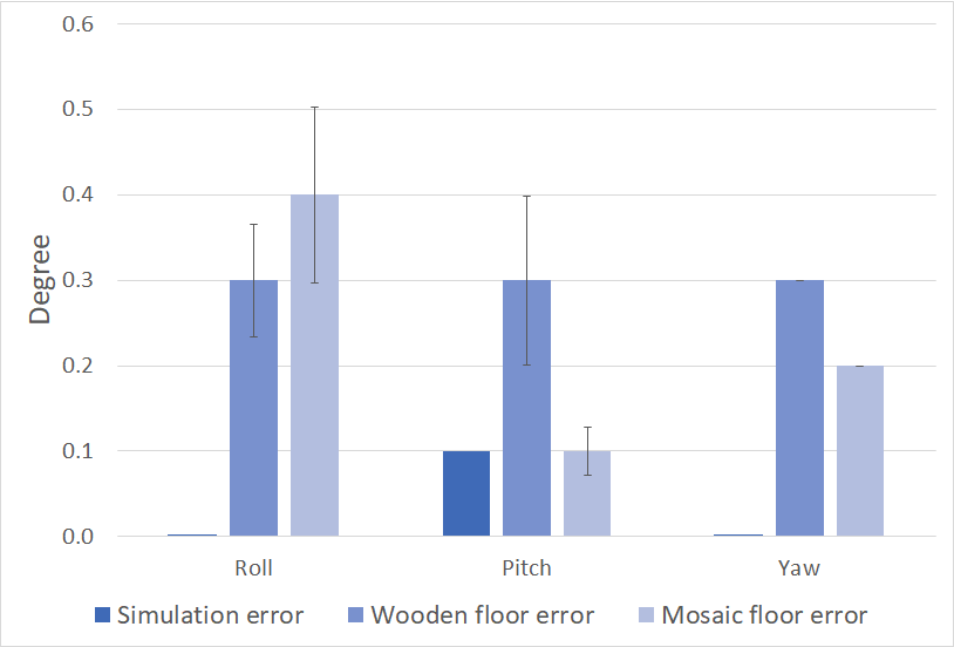

Figure 6.8: Bottom camera rotation parameters estimation error.

around 0.2°.

Overall, the estimations for the middle camera is considered as acceptable. Because all of the errors for the mosaic floor area are under 2 cm for the translation parameters and under 2.5° for the rotation parameters with PEQF of 0.4. PEQF of 1.1 quantifies the noticeable weaker performance of the method in the wooden floor area because of 5 cm error for the translation parameters and 3° for the rotation parameters.

Middle camera	**X(m)**	**Y(m)**	**Z(m)**	**Roll(°)**	**Pitch(°)**	**Yaw(°)**
Ground truth	0.115	0.020	0.755	0.2	34.4	0.2
Simulation	0.113	0.022	0.755	0.0	34.9	0.0
Simulation error	0.002	0.002	0.000	0.2	0.5	0.2
PEQF = 0.02						
Mosaic floor						
Data mean	0.099	0.025	0.741	0.0	36.8	0.1
Error mean	0.018	0.005	0.014	0.3	2.4	0.2
STDV	0.006	0.008	0.002	0.2	0.4	0.3
SE	0.002	0.002	0.001	0.0	0.1	0.0
PEQF = 0.4						
Wooden floor						
Data mean	0.070	0.028	0.742	-0.5	37.1	0.1
Error mean	0.045	0.008	0.013	0.7	2.7	0.2
STDV	0.003	0.003	0.002	0.3	0.3	0.4
SE	0.001	0.001	0.001	0.1	0.1	0.1
PEQF = 1.1						

Table 6.4: Middle camera pose estimation data.

In figures 6.11 and 6.12, the top camera pose estimation error and its SE are illustrated. This is the most distanced camera from the floor mounted at 1.33 m height. Estimation error for the X parameter in the wooden floor is about 70 mm and in the mosaic floor around 45 mm. For the Y parameter, error is about 17 mm for wooden and around 11 mm for mosaic floor. In estimation of the Z parameter in wooden and mosaic floor this amount is close to 38 mm. In

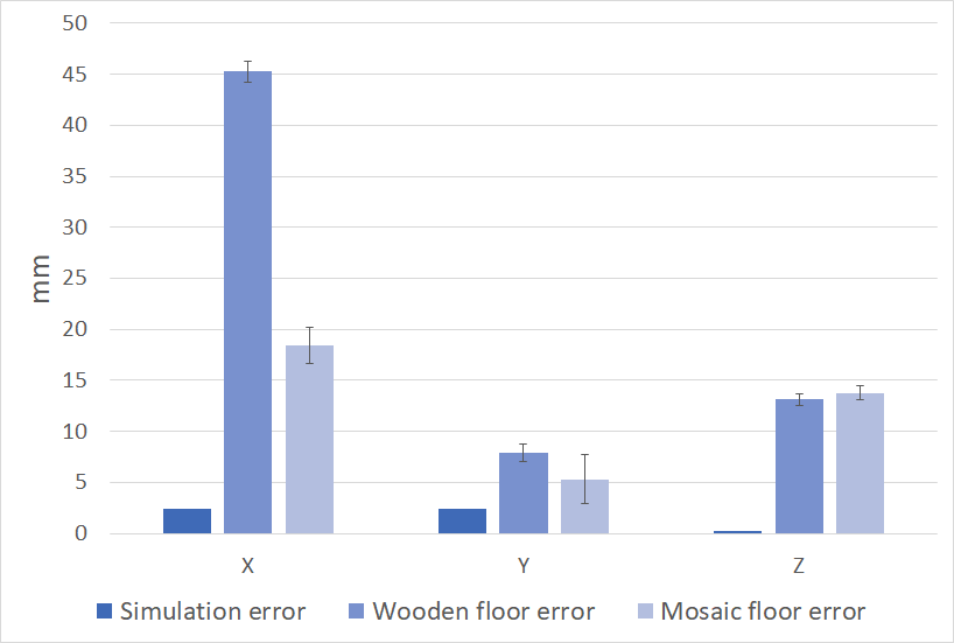

Figure 6.9: Middle camera translation parameters estimation error.

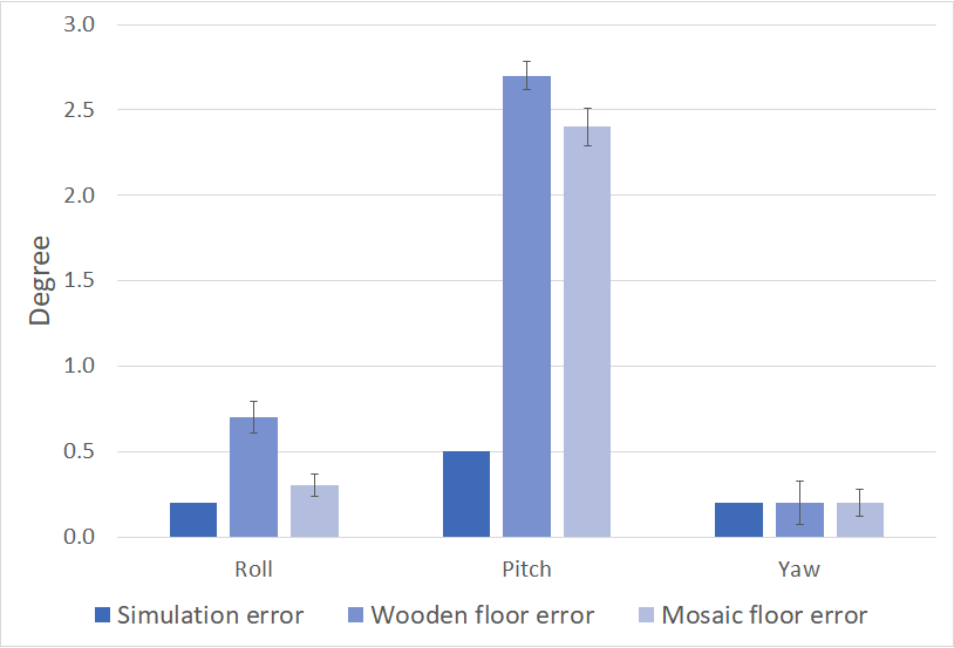

Figure 6.10: Middle camera rotation parameters estimation error.

the roll parameter estimation error is around 1.9° and 1.3° respectively for the wooden and mosaic floor. In estimation of the pitch parameter in the wooden floor error is 2.3° and in mosaic floor it is around 2.6°. The yaw estimation peaks in the wooden floor and gets more than 4° and in the mosaic floor it has dramatically better performance with 0.4° error.

Overall, the estimations for the top camera is considered as not accurate in both of the wooden floor area with PEQF of 5.7 and the mosaic floor area with the PEQF of 1.6. The weaker performance in the wooden floor area is also noticeable in this case. It is also obvious from the graphs that the estimation errors are way more than the previous cases.

Top camera	**X(m)**	**Y(m)**	**Z(m)**	**Roll(°)**	**Pitch(°)**	**Yaw(°)**
Ground truth	0.173	0.020	1.333	0.0	51.6	0.0
Simulation	0.171	0.022	1.333	0.0	51.4	-0.1
Simulation error	0.002	0.002	0.000	0.0	0.2	0.1
PEQF = 0.02						
Mosaic floor						
Data mean	0.128	0.022	1.294	0.7	54.2	0.4
Error mean	0.045	0.011	0.039	1.3	2.6	0.4
STDV	0.009	0.009	0.005	1.0	0.4	0.1
SE	0.003	0.002	0.002	0.3	0.1	0.0
PEQF = 1.6						
Wooden floor						
Data mean	0.103	0.029	1.296	1.9	53.9	4.1
Error mean	0.070	0.017	0.037	1.9	2.3	4.1
STDV	0.009	0.009	0.006	1.3	0.4	0.0
SE	0.003	0.003	0.002	0.4	0.1	0.0
PEQF = 5.7						

Table 6.5: Top camera pose estimation data.

The simulation results have already proved that the method is able to estimate pose of the camera accurately. The calculated SE values for real-life scenarios, which are illustrated in the graphs, show the maximum variation from

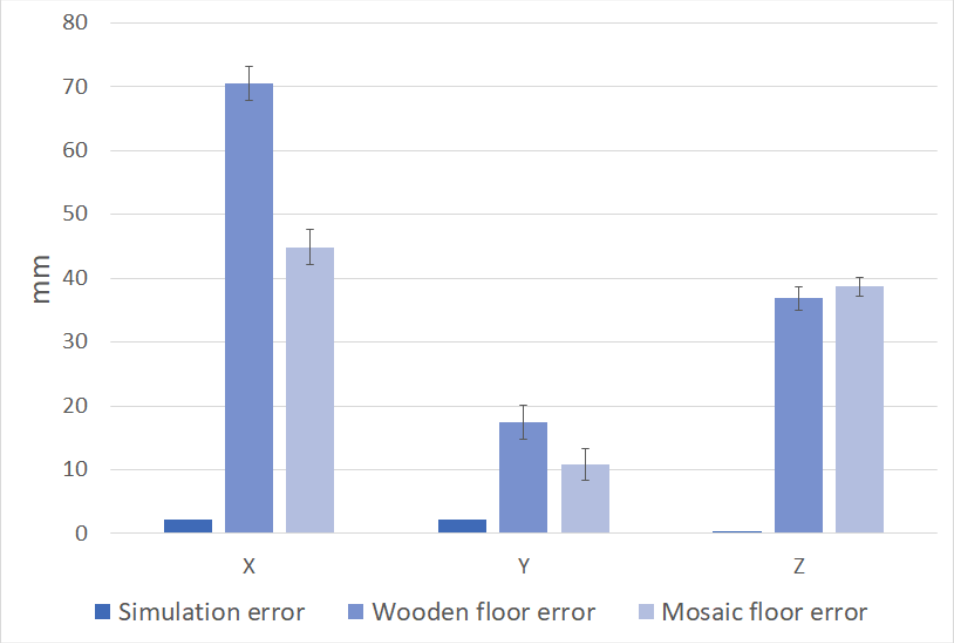

Figure 6.11: Top camera translation parameters estimation error.

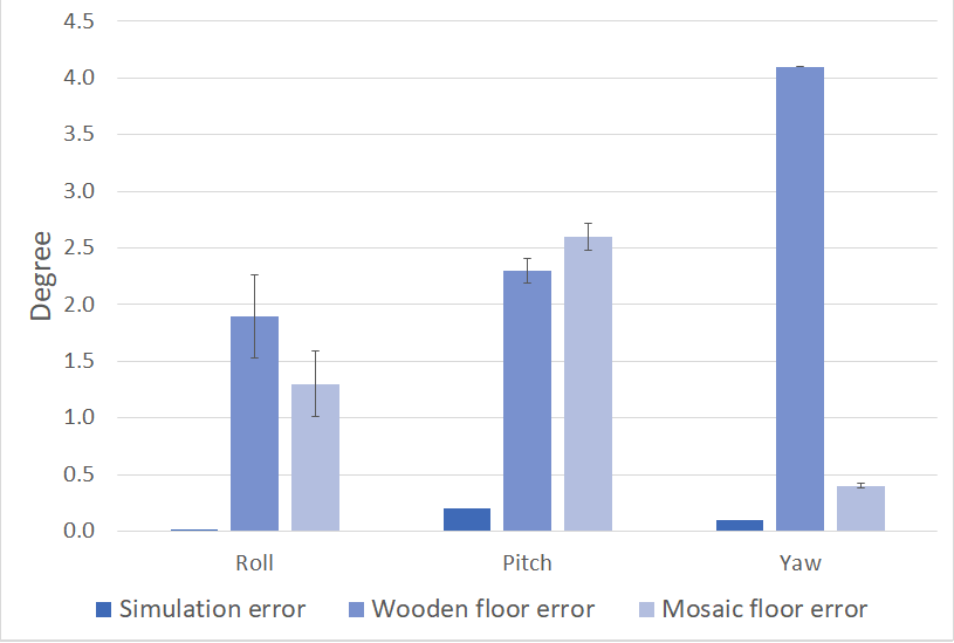

Figure 6.12: Top camera rotation parameters estimation error.

error mean of 3 mm for the translation parameters and 0.4° for the rotation parameters. This proves the internal consistency of the method which means estimated parameters during the multiple experiments are almost the same. Therefore, the source of the pose estimation errors is in the other participating parts of the calibration process such as camera tracker or the camera itself. Comparing calculated PEQFs for each camera in different areas clarifies the effect of the floor structure in the pose estimation processes, which only effects the camera tracker algorithm. This incident is investigated more in the next chapter. Another noticeable point when the camera is getting far from the floor in both scenarios is the increment of error in estimation of the Z parameter and the rotation parameters which caused by inaccurate depth information from the camera.

6.2.4 Camera Tracking Error

In order to evaluate the performance of the camera tracker another kind of experiment is staged. In which the camera trajectory data from camera tracker and fitted circle to it are recorded during a circular path drive of the robot in the both wooden and mosaic floor areas using all three cameras. The ground truth trajectory of the cameras are also calculated by knowing the ground truth pose of them from section 6.1.2.

Figures 6.13 and 6.14 illustrate comparison between these trajectories for all three cameras. Bottom camera provides very good tracking results in both areas. Because the camera tracker trajectories are almost in a perfect circle shape and fitted circles to them are very close to the ground truth trajectories. This type of good matching results in accurate estimation of the parameters with PEQF of 0.1 in both areas.

The PEQF increases for the middle camera in wooden floor area to 1.1 compare to its value in the mosaic floor which is 0.4. This means a weaker performance of pose estimation in wooden floor area. The reason behind this is obvious from comparing the camera trajectories for middle camera in both areas. The camera trajectory for this camera in the wooden floor area has much more tracking error than then equivalent on in the mosaic floor area.

The PEQF for the top camera in mosaic floor is 1.6 but it increases dramatically to 5.7 in the wooden floor area. The results show that the trajectory tracking for this camera has more tracking error than the others in the mosaic floor area and it has the the most error for tracking among the other in wooden floor area.

Tracking error increment is also noticeable by looking to the results of this experiment from bottom camera to the top one in each area. As the camera gets higher from the floor the tracking trajectory accuracy starts to decline.

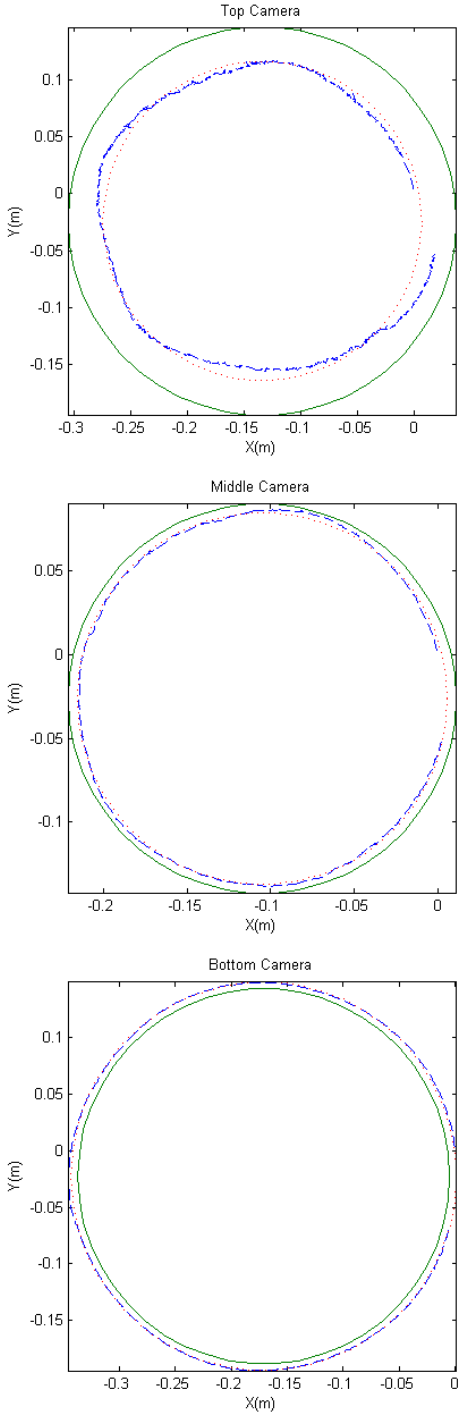

Figure 6.13: Ground truth (—), tracking trajectory (- - -) and circle fitting (· · ·) result for three cameras mounted at different heights (0.44m, 0.75m, 1.33m) from mosaic floor during a circular path drive of the robot.

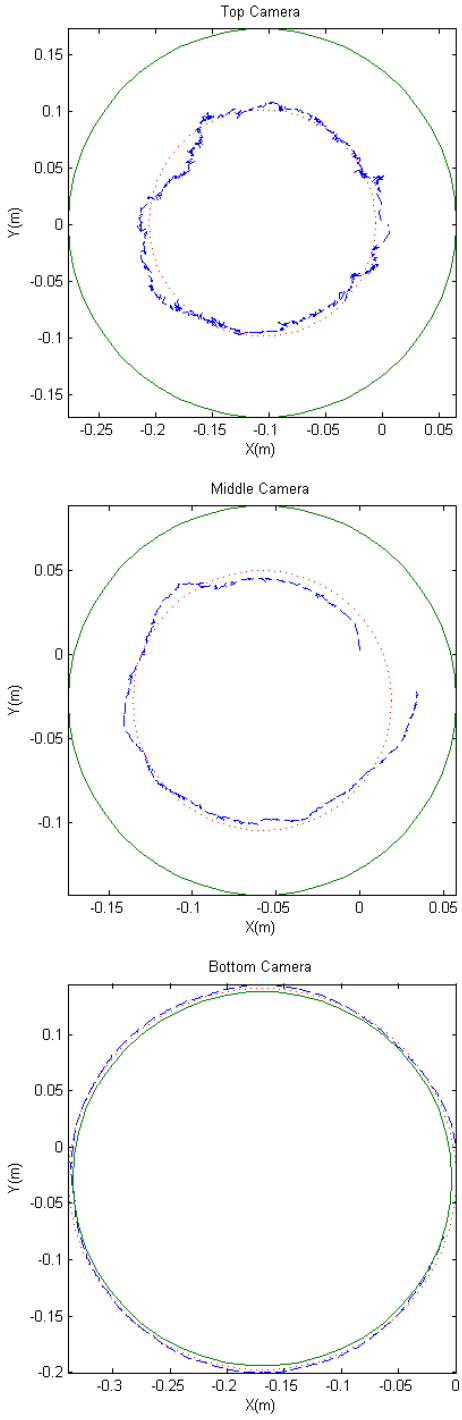

Figure 6.14: Ground truth (—), tracking trajectory (- - -) and circle fitting (· · ·) result for three cameras mounted at different heights (0.44m, 0.75m, 1.33m) from wooden floor during a circular path drive of the robot.

7 Conclusion

The simulation experiment results proved functionality of the method under ideal circumstances, considering almost zero pose estimation error for all three cameras.

The real-life scenarios provided pose estimations data for the three cameras mounted on the V4core robot at different poses in two areas with the wooden and mosaic floor structure. These experiments revealed the internal consistency of the method (cf. figures 6.7 – 6.12). They also showed that there are other sources causing the pose estimation errors in participating parts of the calibration process such as the camera tracker.

In order to evaluate the performance of the camera tracker, the camera trajectory data and fitted circle to it are recorded during a circular path drive of the robot in the both wooden and mosaic floor areas using all three cameras. The ground truth trajectory of the cameras are also calculated by knowing the ground truth pose of them (cf. 6.13 – 6.14). The outcome of this experiment proved that the camera tracker has better performance on the mosaic floor area compared to the wooden one which leads to less estimation error in this area. The reason behind this incident is the lack of enough traceable features (texture) in the wooden floor area compare to the mosaic floor area.

Two other noticeable points in both real-life scenarios and camera tracker error experiments, when the camera is getting far from the floor, are:

- the increment of error in estimation of the Z parameter and the rotation parameters, and
- accuracy decline in the tracking trajectory quality.

This result was expected because it is already shown in previous studies that the depth accuracy of a 3D camera decreases when the distance between the camera and the planar surface increases [45].

This study focused on presenting a novel autonomous and fast method for extrinsic calibration of a 3D camera on board of a mobile robot without any

need for artificial targets, using only the data provided by camera and and mobility of the robot.

The simulation results proved the concept and the real-life scenarios also demonstrated that the method provided good results in term of accuracy for practical cases, with consideration of the accuracy range of the 3D camera and sufficient texture of working environment of the robot. It is known from stereo systems that the floor always contains some texture or stains which are sufficient to be tracked contrary to walls, that might be really textureless.

The method has a significant advantage over present systems to use the existing static scene of the robot's working environment as a calibration pattern. Another advantage of the method is its speed. The full calibration is done autonomously under three minutes, unlike manual methods which are much more slower and human intervention is always needed in the calibration process. Furthermore, the robot is able to check its calibration when needed.

Future studies should examine the effects of the intrinsic calibration of the 3D camera on the method to obtain more accuracy and refinement of the method to live camera calibration during SLAM without need for any predefined paths with optimization of the camera tracker algorithm.

Bibliography

[1] S. M. Grigorescu, G. Macesanu, T. T. Cocias, D. Puiu, and F. Moldoveanu, „Robust camera pose and scene structure analysis for service robotics," *Robotics and Autonomous Systems*, vol. 59, no. 11, pp. 899–909, 2011.

[2] L. Shao, J. Han, D. Xu, and J. Shotton, „Computer vision for rgb-d sensors: kinect and its applications [special issue intro.]," *IEEE transactions on cybernetics*, vol. 43, no. 5, pp. 1314–1317, 2013.

[3] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, „3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[4] T. Fäulhammer, R. Ambruş, C. Burbridge, M. Zillich, J. Folkesson, N. Hawes, P. Jensfelt, and M. Vincze, „Autonomous learning of object models on a mobile robot," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 26–33, 2017.

[5] K. Tateno, F. Tombari, and N. Navab, „Real-time and scalable incremental segmentation on dense slam," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, IEEE, 2015, pp. 4465–4472.

[6] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, „Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 169, 2013.

[7] C.-C. Wang, „Extrinsic calibration of a vision sensor mounted on a robot," *ieee Transactions on Robotics and Automation*, vol. 8, no. 2, pp. 161–175, 1992.

[8] C McGlone, E Mikhail, and J Bethel, „Manual of photogrammetry, american society for photogrammetry and remote sensing," *Bethesda, MD*, 2004.

[9] (2015). Camera calibration, [Online]. Available: `https://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html`.

[10] (2018). Camera calibration, [Online]. Available: `https://www.mathworks.com/help/vision/camera-calibration.html`.

[11] M. P. R. K. L. Van, „Gool. self calibration and metric reconstruction in spite of varying and unknown internal camera parameters,“ in *ICCV'98*, 1998.

[12] S. J. Maybank and O. D. Faugeras, „A theory of self-calibration of a moving camera,“ *International Journal of Computer Vision*, vol. 8, no. 2, pp. 123–151, 1992.

[13] R. I. Hartley, „An algorithm for self calibration from several views,“ in *Cvpr*, Citeseer, vol. 94, 1994, pp. 908–912.

[14] Q.-T. Luong and O. D. Faugeras, „Self-calibration of a moving camera from point correspondences and fundamental matrices,“ *International Journal of computer vision*, vol. 22, no. 3, pp. 261–289, 1997.

[15] G. Carrera, A. Angeli, and A. J. Davison, „Slam-based automatic extrinsic calibration of a multi-camera rig,“ in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 2652–2659.

[16] S. Miller, A. Teichman, and S. Thrun, „Unsupervised extrinsic calibration of depth sensors in dynamic scenes,“ in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, IEEE, 2013, pp. 2695–2702.

[17] S. Li, P. N. Pathirana, and T. Caelli, „Multi-kinect skeleton fusion for physical rehabilitation monitoring,“ in *Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE*, IEEE, 2014, pp. 5060–5063.

[18] D. A. Forsyth and J. Ponce, *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[19] J. Heikkila, „Geometric camera calibration using circular control points,“ *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 10, pp. 1066–1077, 2000.

[20] J. Heikkila and O. Silven, „A four-step camera calibration procedure with implicit image correction,“ in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, IEEE, 1997, pp. 1106–1112.

[21] B. Ribbens, V. A. Jacobs, C. Vuye, J. Buytaert, J. Dirckx, and S. Vanlanduit, „High-resolution temporal profilometry using fourier estimation,“ *Recent advances in topography research*, pp. 61–108, 2013.

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

[23] Z. Zhang, „A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[24] R. Tsai, „A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses," *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[25] S. Chhaniyara, K. ALTHOEFER, and L. D. SENEVIRATNE, „Visual odometry technique using circular marker identification for motion parameter estimation," in *Advances In Mobile Robotics*, World Scientific, 2008, pp. 1069–1076.

[26] M. Maimone, Y. Cheng, and L. Matthies, „Two years of visual odometry on the mars exploration rovers," *Journal of Field Robotics*, vol. 24, no. 3, pp. 169–186, 2007.

[27] D. Marimon, Y. Maret, Y. Abdeljaoued, and T. Ebrahimi, „Particle filter-based camera tracker fusing marker and feature point cues," in *Proceedings of IS&T/SPIE Conference on Visual Communication and Image Processing*, vol. 6508, 2007.

[28] M. Irani and P Anandan, „About direct methods," in *International Workshop on Vision Algorithms*, Springer, 1999, pp. 267–277.

[29] T. Brox, C. Bregler, and J. Malik, „Large displacement optical flow," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, 2009, pp. 41–48.

[30] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, „Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[31] L. Wei, C. Cappelle, Y. Ruichek, and F. Zann, „Gps and stereovision-based visual odometry: application to urban scene mapping and intelligent vehicle localization," *International Journal of Vehicular Technology*, vol. 2011, 2011.

[32] B. K. Horn and B. G. Schunck, „Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[33] B. D. Lucas, T. Kanade, *et al.*, „An iterative image registration technique with an application to stereo vision," 1981.

[34] (2005). Computer vision, [Online]. Available: `http://courses.cs.washington.edu/courses/cse455/05wi/notes/LucasKanade.ppt`.

[35] C. Tomasi and T. Kanade, „Detection and tracking of point features,“ 1991.

[36] M. A. Fischler and R. C. Bolles, „Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,“ *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[37] D. Eberly, „Least squares fitting of data,“ *Chapel Hill, NC: Magic Software*, 2000.

[38] (2015). The vision4robotics library, [Online]. Available: `https://www.acin.tuwien.ac.at/en/vision-for-robotics/software-tools/v4r-library/`.

[39] E. Rosten, R. Porter, and T. Drummond, „Faster and better: a machine learning approach to corner detection,“ *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 1, pp. 105–119, 2010.

[40] J. Prankl, A. Aldoma, A. Svejda, and M. Vincze, „Rgb-d object modelling for object recognition and tracking,“ in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, IEEE, 2015, pp. 96–103.

[41] (2017). Point cloud library, [Online]. Available: `http://pointclouds.org/about`.

[42] (2016). Pioneer 3-dx, [Online]. Available: `http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx`.

[43] (2018). Fiducials, [Online]. Available: `https://github.com/UbiquityRobotics/fiducials`.

[44] (2018). Pioneermodel, [Online]. Available: `https://github.com/SD-Robot-Vision/PioneerModel`.

[45] H. Haggag, M. Hossny, D. Filippidis, D. Creighton, S. Nahavandi, and V. Puri, „Measuring depth accuracy in rgbd cameras,“ in *Signal Processing and Communication Systems (ICSPCS), 2013 7th International Conference on*, IEEE, 2013, pp. 1–7.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 15. Juni 2018

Farhoud Malekghasemi