DESIGN AND IMPLEMENTATION OF VSLAM NAVIGATION SYSTEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ÇANKAYA UNIVERSITY

BY

ARDA BEKCAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHATRONICS ENGINEERING

FEBRUARY 2021

ABSTRACT

DESIGN AND IMPLEMENTATION OF VSLAM NAVIGATION SYSTEM

BEKCAN, Arda

M.Sc., Department of Mechatronics Engineering

Supervisor: Assistant Prof. Dr. Halit ERGEZER

February 2021, 90 pages

It is very important to guess the location of the redetected objects and loop closures with the visual simultaneous localization and mapping system (VSLAM), one of the biggest problems of a mobile robot. Noise in the sensor measurements and non-linear motion of the mobile robot make it difficult to determine the robot pose and causes low quality map. VSLAM makes it possible to eliminate and/or reduce these applications' errors and improve the robot's direction and position by using previously detected environment.

The aim of this thesis is to achieve an autonomous navigation of a ground vehicle using VSLAM algorithm in an unknown environment. In this context, a differential drive mobile robot with a monocular camera was designed and an experimental platform was built for robot to explore. A VSLAM algorithm was implemented according to theoretical information and the performance of visual odometry and the loop closing process were compared. As a result of the experiments, it was determined that loop closing had a great role in reducing the accumulating drift errors.

Keywords: Localization, Mapping, Computer Vision.

ÖZ

VSLAM SEYİR SİSTEMİ TASARIMI VE UYGULAMASI

BEKCAN, Arda
Yüksek Lisans, Mekatronik Mühendisliği Anabilim Dalı
Tez Danışmanı: Dr. Halit ERGEZER

Şubat 2021, 90 Sayfa

Bir mobil robotun en büyük sorunlarından biri olan görsel eşzamanlı konumlama ve haritalama sistemi (VSLAM) ile yeniden tespit edilen nesnelerin ve döngü kapanışlarının konumunu tahmin etmek çok önemlidir. Sensör ölçümlerindeki gürültü ve mobil robotun doğrusal olmayan hareketi, robot duruşunun belirlenmesini zorlaştırmakta ve düşük kaliteli haritaya neden olmaktadır. VSLAM, bu uygulamaların hatalarını ortadan kaldırmayı ve / veya azaltmayı ve önceden tespit edilen ortamı kullanarak robotun yönünü ve konumunu iyileştirmeyi mümkün kılar.

Bu tezin amacı, bilinmeyen bir ortamda VSLAM algoritması kullanarak bir kara taşıtının otonom navigasyonunu sağlamaktır. Bu bağlamda, monoküler kameralı diferansiyel sürüşlü bir mobil robot tasarlanmış ve robotun keşfetmesi için deneysel bir platform oluşturulmuştur. Teorik bilgilere göre bir VSLAM algoritması uygulanmış ve görsel odometri ile döngü kapama işlemi performansı kıyaslanmıştır. Deneyler sonucunda biriken kayma hatalarının azaltılmasında döngü kapatmanın büyük rolü olduğu tespit edilmiştir.

Anahtar Kelimeler: Konumlama, Haritalama, Bilgisayarla Görme.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Halit ERGEZER for his supervision, special guidance, suggestions, and encouragement through the development of this thesis.

It is a pleasure to express my special thanks to my family for their valuable support.

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM .....**Hata! Yer işareti tanımlanmamış.**
ABSTRACT iii
ACKNOWLEDGEMENTS v
LIST OF FIGURES viii
LIST OF ABBREVIATIONS x
CHAPTERS
1. INTRODUCTION 1
1.1 Background 1
1.2. Objective and Overview 4
2. VEHICLE MODEL 6
2.1 Dead Reckoning Localization 7
2.2 Ackermann Vehicle Model 8
2.3 Differential Drive Model 10
3. TECHNIQUES USED TO REDUCE AND ELIMINATE ERRORS IN SLAM.. 14
3.1 Kalman Filter 15
3.2 Extended Kalman Filter 18
3.3 Least Squares 19
3.4 RANSAC (RANdom SAmple Consensus) 22
3.5 Loop Closure 23
3.5.1 Bag of Words 23
3.5.2 K-Medoid Clustering 24
4. OBJECT TRACKING WITH IMAGE PROCESSING 26
4.1 Object Tracking in VSLAM Algorithms 28
4.1.1 Harris Corner Detection 29
4.1.2 FAST Corner Detection 31
4.1.3 BRIEF Descriptor Extraction 33
4.1.4 Steered BRIEF Descriptor 34
4.2 Camera Calibration 35
4.3 Distance Measurement 41
4.3.1 Distance Measurement with Stereo Triangulation Method 42

4.3.2 Distance Measurement with Epipolar Geometry 43
4.3.3 Stereo Rectification and Disparity Calculation 49
5. EXPERIMENTS AND RESULTS 53
5.1 Hardware Setup 53
5.2 Software Setup 54
5.3 Camera Calibration 54
5.4 Bag of Words Construction 55
5.5 Experiment Overview 56
5.6 Experiment Platform 58
CONCLUSION 67
REFERENCES 69
APPENDIX A 74
Materials Used in the Experiment 74
APPENDIX B 80
DC Motor Control 80
K-Medoid 81
Bag of Word Histogram Computation 84
APPENDIX C 86
Tracking and Mapping 86

LIST OF FIGURES

Figure 1 Uncertainty in-vehicle position as the vehicle progresses. 2
Figure 2 Mimicked stereo camera system 3
Figure 3 External forces acting during navigation 7
Figure 4 Ackermann steering geometry 9
Figure 5 Differential drive kinematics 12
Figure 6 The illustration of how Kalman filter optimizes states. 15
Figure 7 Gaussian distribution after sin function. 19
Figure 8 Line fitting using RANSAC. 22
Figure 9 (a) Randomly selected medoids, (b) medoids after one iteration 25
Figure 10 Feature detection and matching algorithms. 29
Figure 11 Fast corner detection [39] 32
Figure 12 Randomly chosen pixel pairs in a patch [40] 34
Figure 13 Checkerboard pattern 36
Figure 14 Lens Distortion Types 40
Figure 15 Distorted and undistorted images, respectively. 41
Figure 16 Stereo geometry [44]. 43
Figure 17 Epipolar geometry [45] 44
Figure 18 Diamond search [50] 52
Figure 19 Design autonomous mobile robot 53
Figure 20 Checkerboard pattern setup 54
Figure 21 Detected corner points in camera calibration 55
Figure 22 Calibration error 55
Figure 23 Layers created using k-medoid clustering 56
Figure 24 Images used to construct bag of words. 56
Figure 25 Detected FAST corners. 57
Figure 26 Matched inlier points between frames 57
Figure 27 Constructed robot environment 59
Figure 28 Path drawn by the mobile robot 59

Figure 29 Corners used for the ground truth estimation 60
Figure 30 Measured ground truth in cm. 60
Figure 31 Computed robot path by monocular visual odometry 61
Figure 32 Mobile robot path computed by SURF-SLAM 61
Figure 33 Computed robot path by the encoder. 62
Figure 34 Encoder noise fit in Gaussian model 63
Figure 35 The size of covariance ellipses as mobile robot moves. 64
Figure 36 Mobile robot path and detected points in the corrected scale. 65
Figure 37 Robot path before and after loop closing 66
Figure 38 Brushed DC motor 74
Figure 39 L298 motor driver. 74
Figure 40 Wheels 75
Figure 41 Caster wheel 75
Figure 42 IPS LCD capacitive touch screen 76
Figure 43 Raspberry Pi 4 8GB - model B 76
Figure 44 Xiaomi 20000 mAh Powerbank 3 Pro. 77
Figure 45 4 mm 24 cm x 35 cm transparent plexi mica plexi. 77
Figure 46 Magnetic Encoders 78
Figure 47 Logitech C270 720p webcam. 78
Figure 48 Soldering iron, solder wire, flux pasta solder, and jumper wire. 79

LIST OF ABBREVIATIONS

BRIEF	Binary Robust Independent Elementary Features
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
GPS	Global Positioning System
IMU	Inertial Measurement Unit
KF	Kalman Filter
RANSAC	Random Sample Consensus
RMSE	Root Mean Squared Error
RPE	Relative Pose Error
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SURF	Speeded-Up Robust Features
UAV	Unmanned Aerial Vehicle
1D	One-dimensional
2D	Two-dimensional
ND	N-dimensional
RGB-D	Red, Green, Blue, and Depth
PAM	Partitioning Around Medoids
GNA	Gauss-Newton Algorithm
ORB	Oriented Fast Rotated Brief
LIDAR	Light Detection and Ranging
AR	Augmented Reality

CHAPTER I

INTRODUCTION

1.1 Background

VSLAM refers to visual simultaneous localization and mapping. Its working principle is based on determining an autonomous vehicle's location by using visual information taken from surrounding objects while mapping the environment simultaneously where location and environment are unknown. Today, VSLAM is used in many different areas like; 3D modeling, AR (Augmented Reality), and unmanned aerial vehicles (UAVs) [1].

There are four main processes in VSLAM:

- Tracking: Processing input images to calculate the depth and vehicle position by extracted features.
- Mapping: Drawing a map using depth information acquired from an input image.
- Global Optimization: Correcting tracking and mapping information depending on passing from the same path and detecting loop closures during this stage.
- Relocalization: Estimating the pose by comparing current input and previous inputs when a track is lost [2].

There is a chicken-egg problem in SLAM algorithms. Which one should be done first, mapping or localization? Without a map, localization is not possible because a reference is needed to detect which direction the vehicle is faced and its position. Also, without a known position and orientation, surroundings cannot be imperceptible or mapped. Some applications choose an estimated starting location, which causes errors during localization and mapping, whereas some applications define landmarks as

having well-known locations to prevent pose drift [1]. The errors in the sensor measurement, environmental factors, and complex non-linear motion of the vehicle make it difficult to correctly estimate the vehicle's pose. These errors accumulate as the vehicle progresses, gradually causing the vehicle's pose to drift from the ground truth. Since the distances of the objects are measured with respect to the vehicle's pose, the vehicle's pose's errors affect the determination of the objects' coordinates, which affects the map quality.

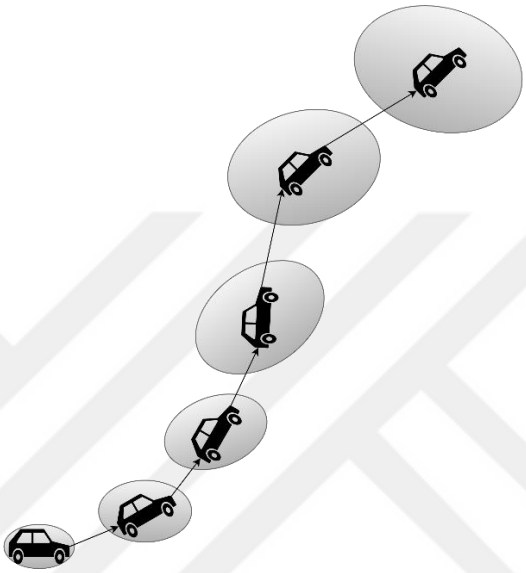


Figure 1 Uncertainty in-vehicle position as the vehicle progresses.

Furthermore, some applications use the loop closure technique to reduce the pose drift. Because of the drift, the same spots visited at different times do not match. The current environment and the previously visited environment are compared. If the environments are similar, the loop is assumed to be detected. Various algorithms such as SIFT (Scale Invariant Feature Transform), SURF (Speeded-Up Robust Features), and ORB (Oriented Fast Rotated Brief) descriptors make it possible to compare environments with each other [3] and deep learning algorithms help speed up the process [4], [5]. Different types of cameras can be used to get information about the environment, such as monocular, stereo, RGB-D (Red Green Blue Depth) cameras. Estimating depth using a monocular camera is very hard and includes complex mathematical calculations that are time-consuming. With a monocular camera, rotations can be computed, but translation can be computed up to a scale.

There are different ways to estimate the scale, such as:

- line angles
- perspective
- object size
- image position
- atmospheric effects
- extra sensor usage (range, IMU (Inertial Measurement Unit))

Also, deep learning algorithms can increase the accuracy of depth measuring [6]. Each of these methods has its drawbacks. Stereo cameras are superior to monocular cameras as it is easy to detect depth by calculating the pixel position difference between camera frames. This difference is called a disparity. Unfortunately, the stereo camera also has a limitation as to the distance between the camera and the object increases, the accuracy of the depth measurement decreases. After some distance, which is a function of the length between stereo lenses and camera resolution, stereo cameras start to act like monocular cameras [7]. The stereo camera model consists of two lenses aligned perfectly side by side. Also, it can be mimicked by using two different cameras (Figure 2). Nevertheless, it is impossible to align lenses perfectly in real-world conditions, so remaining errors can be reduced by rectification technique.

Figure 2 Mimicked stereo camera system

VSLAM is one of the multiple techniques to solve the problem mentioned above. There are different approaches, for example, inertial sensors, LIDAR (Light Detection and Ranging), and control strategies to overcome this problem. In this thesis, VSLAM

is chosen because of its advantages over other methods. With today's technology, it is possible to extract useful information from visual data thanks to computer vision advancement. The biggest superiority compared to IMU is using a map of the environment built by taken visual data. A map helps to carry out autonomous tasks such as preventing collision, obstacle analysis, path planning, and a better understanding of the environment to the system's primary user. Also, local features previously detected on the map can decrease the drift caused by an error in sensor measurements. This error makes VSLAM less prone to deviations than other methods [8], [9]. Furthermore, cameras used in VSLAM are smaller, lighter, and cheaper than laser counterparts, making them suitable for mobile systems where weight and design complexity are critical.

1.2. Objective and Overview

As mentioned above, a major challenge in VSLAM is errors in sensor measurements while localizing mobile robots and sensing the environment. The errors increase as the robot moves, and correlated errors lead the robot's current pose to diverge from the ground truth. Various techniques have been developed to minimize errors. According to the theoretical information, a VSLAM algorithm was implemented in this thesis. The thesis focuses on testing the theoretical information in real-world conditions and showing the performance of loop closure compared to visual odometry.

In this framework, a differential wheeled mobile robot was designed. A monocular camera was used to get information from the environment. The camera was calibrated using a checkerboard pattern. The main idea behind this is, converting the camera to a bearing sensor using image-processing techniques. FAST (Features from Accelerated Segment Test) corner detection and BRIEF (Binary Robust Independent Elementary Features) descriptor extraction algorithms were used as an image processing technique to find the robot orientation. Since the translation is computed up to a scale in a monocular camera, encoder outputs were used to determine the scale factor. A BRIEF descriptor set was trained using bag of words model to detect the revisited places. Finally, to perform loop closing and global optimization, a 2D pose graph model was chosen. Optimization was solved using the nonlinear least square method.

The overview of the thesis after the introduction is as follows:

- Chapter 2 explains vehicle models and compares their advantages and disadvantages. The reason for chosen vehicle model was discussed considering the usefulness and the ease of implementation.
- Chapter 3 includes optimization techniques used in SLAM to deal with errors in sensor measurements. The reason behind the chosen technique is reviewed. A method to eliminate wrong matches and a way to further optimize the map are given.
- Chapter 4 explains object-tracking methods and geometry between image frames. How to convert the camera to a bearing sensor. What are the requirements needed to convert a camera to a range sensor are described.
- Chapter 5 expresses the experiments conducted in the framework of the theoretical basis formed in the previous chapters. The results are compared with the state-of-the-art method, and critical parts are discussed.
- Chapter 6 presents a conclusion with the key components of SLAM algorithms.

CHAPTER II

VEHICLE MODEL

The fast advancement in technology increased the demand for robots where humans are insufficient, such as precision works and hazardous areas. The main purpose of robots is automating an operation to increase productivity. Different operations require robots with different features such as; autonomous ground vehicles, underwater vehicles, and UAVs. Some were designed to work under conditions unsafe for living organisms to carry out important high radioactivity [10], [11]. The most common robots are ground robots. They are widespread robots because of their advantages in designing simplicity, powering, strength, size-weight ratio, and cost-effectiveness over others. Ground robots consist of two groups, wheeled and legged robots.

In most cases, wheeled robots are used on smooth surfaces, while legged robots are used on rough surfaces. The advantages of wheeled robots are being cheaper, less complex, and besides they do not need to balance themselves. Wheeled robots have vast usage of an area in the industry such as hospitals, military, educational purposes, and space. For these reasons, a wheeled robot model is chosen in this thesis.

Many different types are designed for wheeled robots. Three-wheeled [12], [13], four and more wheeled robots have been studied. Two different kinematic and dynamic modelling techniques were used to model these robot designs, vector [14], [12], [15] and transformation [16], [17], [18], based solution. There is no standard in computing kinematic and dynamic models of moving robots, including complex computation of transformation. A kinematic model will be sufficient for understanding the motion of the robot. For this reason, commonly used models such as Ackermann and differential drive models where kinematics can be easily computed from trigonometry are focused on this thesis.

Furthermore, as mentioned before, translation can be computed up to a scale using a monocular camera, and another information source is required to find the scale. In the absence of this extra source, dead-reckoning localization can be used by assuming vehicles moving with a constant velocity. In this thesis, a localization simulation was conducted in Matlab based on the Kalman filter and the dead-reckoning model (Chapter 3). Dead reckoning, Ackermann, and differential drive models are explained, and the reasons for the suitable model chosen for the thesis are discussed in the following pages.

2.1 Dead Reckoning Localization

Dead reckoning is an old method used to navigate ships. The new position is estimated using the previous position, heading, and the average speed in an elapsed time.

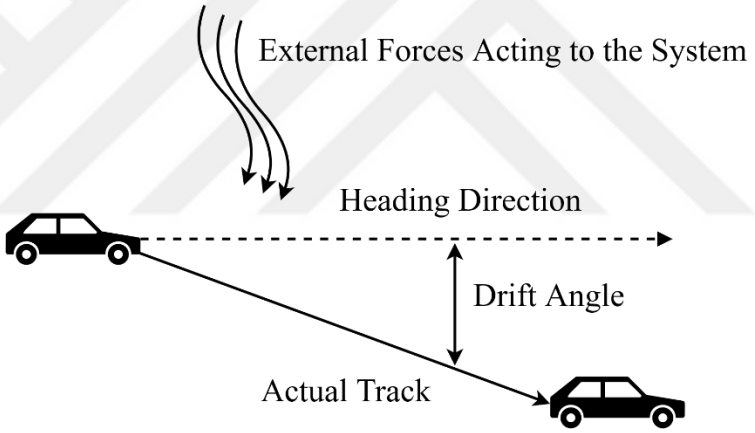


Figure 3 External forces acting during navigation

When a new position is estimated according to forces acting on the system and the previous position, frame transformation can be used. We need to express input vector $u_r(k)$ in map coordinate frame. If we assume our map coordinates consist of the x and y-axis, the mobile robot rotates in the z-axis. The counter-clockwise rotation matrix is:

$$\text{Rot(z)}=\begin{bmatrix}\cos(\theta_r) & -\sin(\theta_r) & 0\\ \sin(\theta_r) & \cos(\theta_r) & 0\\ 0 & 0 & 1\end{bmatrix}, u_r(k)=\begin{bmatrix}v_{r_x}(k)\\ v_{r_y}(k)\\ \theta_{r_b}(k)\end{bmatrix} \quad (2.1)$$

The translation between steps can be found by the multiplication of the rotation matrix and control input.

$$\begin{bmatrix} x_r(k) \\ y_r(k) \\ \theta_r(k) \end{bmatrix} = \begin{matrix} x_r(k-1) + v_{r_x}\cos\big(\theta_r(k)\big) - v_{r_y}\sin\big(\theta_r(k)\big) \\ y_r(k-1) + v_{r_x}\sin\big(\theta_r(k)\big) + v_{r_y}\cos\big(\theta_r(k)\big) \\ \theta_r(k-1) + \theta_{r_b} \end{matrix} \tag{2.2}$$

2.2 Ackermann Vehicle Model

In Ackermann geometric model, the main goal is to prevent wheels from sliding laterally when moving on a curved path. To solve this problem, wheels are arranged in a way where they turn around a common center. With enough inputs, the change in the vehicle's position can be solved for this model as follows [19]:

$$X_{state} = \begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}, \tag{2.3}$$

$$v_r = \frac{dx}{dt} \tag{2.4}$$

$$\dot{x_r} = v_r \cos(\theta_r) \tag{2.5}$$

$$\dot{y_r} = v_r \sin(\theta_r) \tag{2.6}$$

Where v_r and θ_r are robot's instantaneous forward velocity and rotation angle, respectively.

$$S_r = r_r\theta_r, \omega_r = \frac{d\theta_r}{dt} \tag{2.7}$$

Where S_r and ω_r are arc length of a circle and change of angular position, respectively while r_r is the radius of the circle. If we take the derivative of S_r with respect to time, linear velocity can be written utilizing radius and change of angular position.

$$S_r = r_r\theta_r \tag{2.8}$$

$$v_r = \frac{dS_r}{dt} = r_r\frac{d\theta_r}{dt} = r_r\omega_r \tag{2.9}$$

$$\frac{L_r}{r_r} = \tan(\phi_r) \tag{2.10}$$

$$\omega_r = \frac{v_r}{L_r}\tan(\phi_r) \tag{2.11}$$

Where ϕ_r and L_r are steer angle and distance between the back and front wheels, respectively. As the model is nonlinear, it is in the following form:

$$x_r(k+1) = f\big(x_r(k), u_r(k)\big); u_r(k) = \begin{bmatrix} v_r(k) \\ \phi_r(k) \end{bmatrix} \tag{2.12}$$

$$\begin{bmatrix} x_r(k+1) \\ y_r(k+1) \\ \theta_r(k+1) \end{bmatrix} = \begin{bmatrix} x_r(k) + dtv_r(k)\cos\big(\theta_r(k)\big) \\ y_r(k) + dtv_r(k)\sin\big(\theta_r(k)\big) \\ \theta_r(k) + \dfrac{dtv_r(k)\tan\big(\phi_r(k)\big)}{L_r} \end{bmatrix} \tag{2.13}$$

Where $u_r(k)$ is an input vector, and it consists of velocity and steering angle. The outer wheel steer angle should be smaller than the inner steer angle as the outer wheel draws a circle with a bigger radius. For this reason, in a perfect Ackerman steering model, the outer wheel's angular velocity must be higher than the inner wheel to travel a longer distance at the same time.

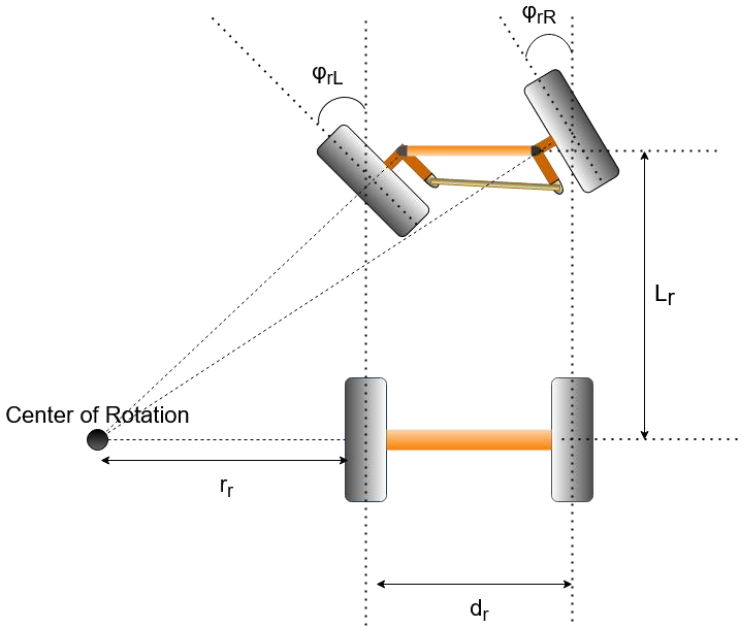


Figure 4 Ackermann steering geometry

From taking the inverse of Eq. (2.10), a more compact solution can be found for wheel steer angles [20].

$$\frac{r_r}{L_r} = tan(\frac{\pi}{2} - \phi_r) \tag{2.14}$$

$$\tan\left(\frac{\pi}{2} - \boldsymbol{\phi_{r_L}}\right) = \frac{r_r}{L_r} \tag{2.15}$$

$$\tan\left(\frac{\pi}{2} - \boldsymbol{\phi_{r_R}}\right) = \frac{r_r + d_r}{L_r} \tag{2.16}$$

The width of the robot is noted as d_r**.** Taking the inverse of the tangent function, the angles ϕ_{r_L}, ϕ_{r_R} left and right steel angles respectively can be calculated.

$$\phi_{r_L} = \frac{\pi}{2} - arctan(\frac{r_r}{L_r}) \tag{2.17}$$

$$\phi_{r_R} = \frac{\pi}{2} - arctan(\frac{r_r + d_r}{L_r}) \tag{2.18}$$

The rate of change of angular position ω_r is the same for both wheels as they are both attached to the robot body. Linear velocities of each wheel can be found by using Eq. (2.11).

$$\omega_r = \frac{v_{r_L}}{L}\tan(\phi_{r_L}) \tag{2.19}$$

$$\omega = \frac{v_{r_R}}{L_r}\tan(\phi_{r_R}) \tag{2.20}$$

If we substitute Eq. (2.10) in Eq. (2.20) and Eq. (2.21), linear wheel velocities can be derived [21].

$$v_{r_R} = \omega_r(r_r + d_r) \tag{2.21}$$

$$v_{r_L} = \omega_r r_r \tag{2.22}$$

2.3 Differential Drive Model

The differential drive model is similar to the Ackermann vehicle model. The difference is instead of having a steering angle, the model includes two wheels placed on the same axis, and both wheels can rotate clock and counter-clockwise direction independently from each other. As in the Ackermann model, wheels circle around a common center called the instantaneous center of curvature. Center location and radius are determined

by changing the wheel's angular velocities. The vehicle's position for this model can be solved as follows [22]:

$$\omega_r \left(r_r + \frac{l_r}{2}\right) = v_{r_R} \tag{2.23}$$

$$\omega_r \left(r_r - \frac{l_r}{2}\right) = v_{r_L} \tag{2.24}$$

$$r_r = \frac{l_r(v_{r_L} + v_{r_R})}{2(v_{r_R} - v_{r_L})} \tag{2.25}$$

$$\omega_r = \frac{v_{r_R} - v_{r_L}}{l_r} \tag{2.26}$$

Radius and change of angular position is related to wheel velocities and the distance between the left and right wheel, which is noted as l_r.

There are three special case scenarios:

- If $v_{r_R} = v_{r_L}$, there is no rotation.
- If $v_{r_R} = -v_{r_L}$, the vehicle does a pure rotation around itself without translation.
- If $v_{r_R} = 0 \; or \; v_{r_L} = 0$, vehicle rotates around the stationary wheel.

The following equation can find the instantaneous center of curvature:

$$ICC = [x_r - r_r sin(\theta_r), y_r + r_r cos(\theta_r)] \tag{2.27}$$

To apply rotation, ICC must be moved to the center of the world origin.

$$\begin{bmatrix} x_r - ICC_x \\ y_r - ICC_y \\ \theta_r \end{bmatrix} \tag{2.28}$$

It is about rotating a vector in 2D. To rotate the state vector, a rotation matrix can be used.

$$\begin{bmatrix} x_r' \\ y_r' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\omega_r) & -sin(\omega_r) & 0 \\ sin(\omega_r) & cos(\omega_r) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - ICC_x \\ y_r - ICC_y \\ 1 \end{bmatrix} \tag{2.29}$$

To find the robot position, ICC is moved back to its previous position from the origin.

$$\begin{bmatrix} x_r' \\ y_r' \\ \theta \end{bmatrix} = \begin{bmatrix} cos(\omega_r) & -sin(\omega_r) & 0 \\ sin(\omega_r) & cos(\omega_r) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - ICC_x \\ y_r - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega_r \end{bmatrix} \tag{2.30}$$

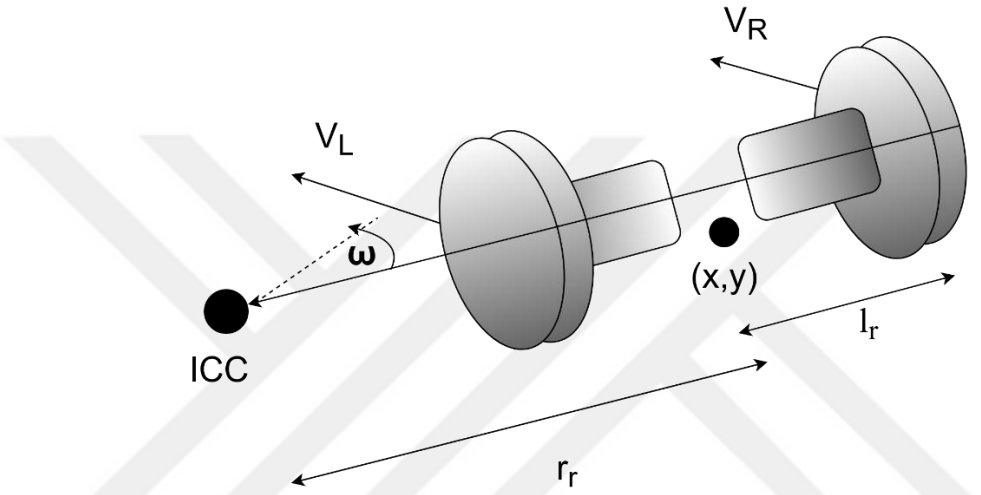


Figure 5 Differential drive kinematics

The differential model was evaluated as the most useful model for the thesis, considering the explanation above. The following reasons influence the chosen vehicle model.

In the Ackermann model, the minimum turning angle of the vehicle is non-zero. In other words, the vehicle cannot perform a pure rotation, whereas the vehicle in the differential model can move more freely and avoid obstacles in the environment easily. Therefore, the differential model is more convenient in indoor conditions.

Besides, the front wheels must have different steering angles from each other in the Ackermann model to prevent slippage. For example, the left wheel must turn more than the right one when the vehicle turns left. Namely, the wheel in the vehicle rotation direction must have a greater steer angle than the other wheel. Such a mechanism is too complex to build and increases the cost.

The differential vehicle has a disadvantage when it comes to moving in a straight line. Even a small difference between right and left wheel speeds causes the vehicle to move in an arc instead of a line. This difference can be minimized using encoder readings and a speed controller.

CHAPTER III

TECHNIQUES USED TO REDUCE AND ELIMINATE ERRORS IN SLAM

As mentioned before, the main problem of VSLAM is accumulating random errors. Unmodeled system inputs and noisy sensor data cause the final pose to drift and mapping the environment wrong. For this reason, some recursive algorithms are used to estimate the state. For prediction algorithms, the Kalman Filter, Extended Kalman Filter, and Particle Filter can be given examples, whereas Gauss-Newton, Levenberg-Marquardt for least squares approach.

Another important method is the loop closing method used to further reduce the drift caused by sensor errors. Loop closing is the detection of the previously visited places having low drift error when the robot moves. Thus, drift error is reduced in the robot's current pose. This method is used in all SLAM algorithms and not just specific for used recursive algorithms.

Besides sensor noise, errors occur during the feature matching process (see chapter 4 for detailed information). Although matching accuracy is not expected to be 100%, a general consistency is desired among matched features as wrong matches make the measurement model diverge from the truth. To eliminate the wrong matches, the RANSAC method can be used.

The methods mentioned above are explained, and the reasons for the practical recursive method chosen for the thesis are discussed in the following pages.

3.1 Kalman Filter

Kalman filter is a state estimator that can reduce the system noise and fuse sensors. By combining the predicted state, which is calculated according to the system model and sensor output, the Kalman filter comes up with a better estimate [4].

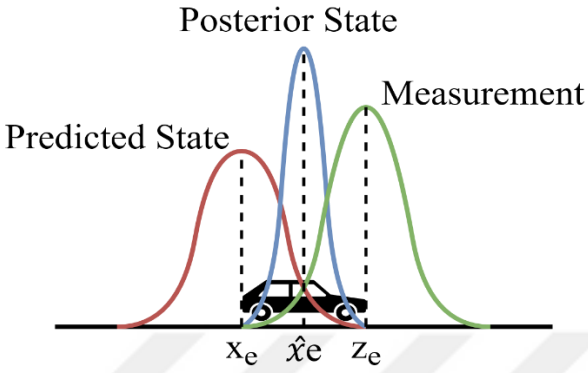


Figure 6 The illustration of how Kalman filter optimizes states.

Kalman filter consists of two parts called prediction and update. The update part is derived from The Bayes rule. We have an estimated value x_e of our state. We have a given measurement z. This measurement somehow related to our estimate, and this is expressed in likelihood function $p(z_e|x_e)$. Probability of event z, given that event x, occurs.

From Bayes rule, a posterior estimate is [19]:

$$p(x_e|z_e) = \frac{p(z_e|x_e)p(x_e)}{p(z_e)} \tag{3.1}$$

$$\mathrm{P}(z_e) = \sum (z_e|x_e)p(x_e) \; or \; \mathrm{P}(z_e) = \int p(z_e|x_e)p(x_e) \tag{3.2}$$

$P(z_e)$ is constant relative to x_e. It is called normalizing or scaling constant therefore its neglected in practice.

$$p(x_e|z_e) \approx p(z_e|x_e)p(x_e) \tag{3.3}$$

When $p(z_e|x_e)$ and $p(x_e)$ on x_e assumed Gaussian distribution, observation model becomes:

$$z_e = H_e x_e + w_e, \tag{3.4}$$

Where w_e is a gaussian noise with zero mean and covariance R_e. The prior estimate can be described with mean $x_{e-}$and covariance P_{e-}.

$$p(z_e|x_e) = \frac{1}{(2\pi)^{\frac{nz_e}{2}} |R_e|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(z_e - H_e x_e)^T R_e^{-1}(z_e - H_e x_e)\right\} \tag{3.5}$$

$$p(x_e) = \frac{1}{(2\pi)^{\frac{nx_e}{2}} |P_e|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(x_e - x_{e-})^T {P_{e-}}^{-1}(x_e - x_{e-})\right\} \tag{3.6}$$

We know that the multiplication of two Gaussian functions is also Gaussian. So above multiplication can be expressed with the below notation:

$$(x_e - x_{e+})^T {P_{e+}}^{-1}(x_e - x_{e+}) \tag{3.7}$$

x_{e+} is a new mean and P_{e+} is a new covariance.

The prediction part is derived from the MMSE (minimum mean square error) estimate [19].

The new state and its covariance are needed to be predicted in the next step.

The new state is predicted from the state model:

$$x_e(k) = F_e x_e(k-1) + B_e u_e(k) \tag{3.8}$$

Where F_e and B_e are system coefficients while u_e is the input of the system.

However, the system has a noise. Real state is expressed as:

$$x_e(k) = F_e x_e(k-1) + B_e u_e(k) + v_e(k) \tag{3.9}$$

Where v_e is Gaussian noise with zero mean and covariance Q_e.
MMSE estimation of x_e for a given Z_e^k:

$$\hat{x}_{e_{mmse}} = tr\{E\{(\hat{x}_e - x_e)(\hat{x}_e - x_e)^T\}\} \quad (3.10)$$

Where Z_e^k is k set of measurements. We can define covariance with the above notation.

$$P_e(k|k-1) = E\left\{\left(x_e(k) - \hat{x}_e(k|k-1)\right)\left(x_e(k) - \hat{x}_e(k|k-1)\right)^T \middle| Z^{k-1}\right\} \quad (3.11)$$

It is found by substituting corresponding equalities in below into Eq. (3.11):

$$x_e(k) = F_e x_e(k-1) + B_e u_e(k) + v_e(k) \quad (3.12)$$
$$\hat{x}_e(k|k-1) = F_e x_e(k-1) + B_e u_e(k) \quad (3.13)$$

(k | k) denotes updated variable at step k, and (k | k-1) denotes the predicted variable at step k. The final form of the prediction step is as follows:

$$\hat{x}_e(k|k-1) = F_e \hat{x}_e(k-1|k-1) + B_e u_e(k) \quad (3.14)$$
$$P_e(k|k-1) = F_e P_e(k-1|k-1) F_e^T + Q_e \quad (3.15)$$

The final form of the update step is as follows:

$$x_e(k|k) = \hat{x}_e(k|k-1) + W_e(k) v_e(k) \quad (3.16)$$
$$P_e(k|k) = P_e(k|k-1) - W_e(k) S_e W_e(k)^T \quad (3.17)$$

Where v_e and W_e are innovation and Kalman gain respectively while H_e is the observation model.

$$v_e(k) = z_e(k) - H_e \hat{x}_e(k|k-1) \quad (3.18)$$
$$S_e = H_e P_e(k|k-1) H_e^T + R_e \quad (3.19)$$
$$W_e(k) = P_e(k|k-1) H_e^T S_e^{-1} \quad (3.20)$$

3.2 Extended Kalman Filter

Most systems in the world are nonlinear. Therefore, the Kalman Filter cannot be directly used. For this reason, the system is linearized using a Taylor series expansion before filtering [19]. The prediction step of the Kalman Filter is changed as follows:

$$\hat{x}_e(k|k-1) = f\left(\hat{x}_e(k-1|k-1), u_e(k)\right) \tag{3.21}$$

$$P_e(k|k-1) = \nabla F_e P_e(k-1|k-1) \nabla F_e^T + Q_e \tag{3.22}$$

Symbol ∇ represents the Jacobian, which consists of partial derivatives with respect to each variable in the system. Update step of the Kalman filter is changed as follows:

$$x_e(k|k) = \hat{x}_e(k|k-1) + W_e(k) v(k) \tag{3.23}$$

$$P_e(k|k) = P_e(k|k-1) - W_e(k) S_e W_e(k)^T \tag{3.24}$$

Where:

$$v_e(k) = z_e(k) - \nabla H_e \hat{x}_e(k|k-1) \tag{3.25}$$

$$S_e = \nabla H_e P_e(k|k-1) \nabla H_e^T + R_e \tag{3.26}$$

$$W_e(k) = P_e(k|k-1) \nabla H_e^T S_e^{-1} \tag{3.27}$$

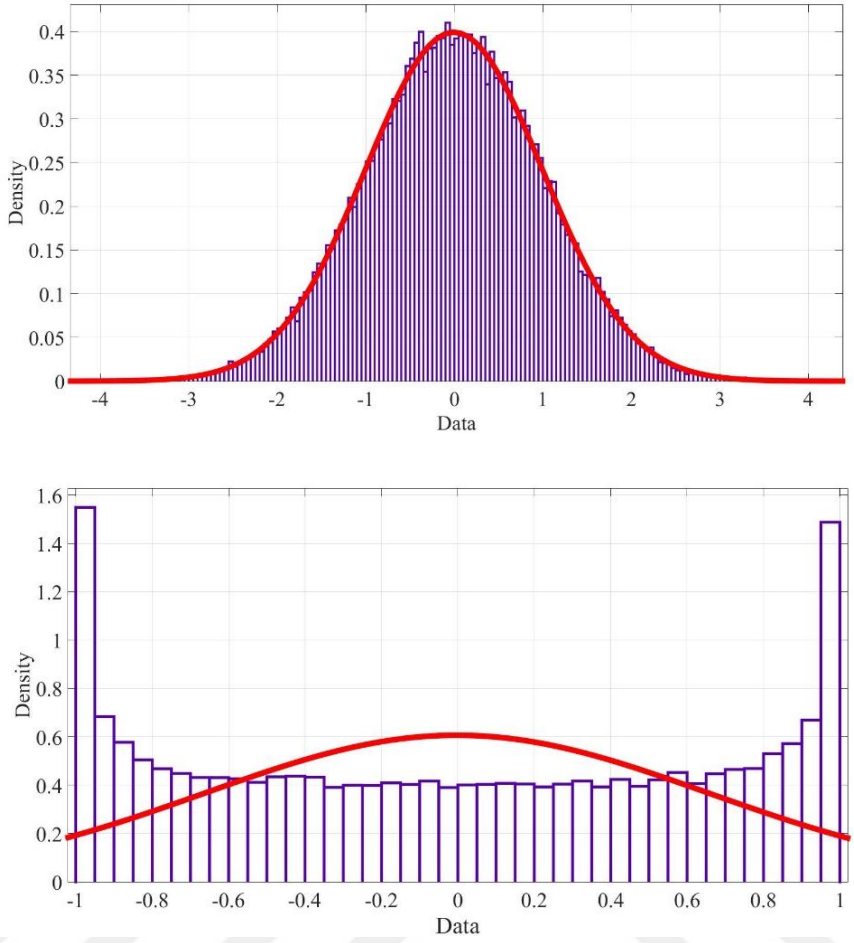


Figure 7 Gaussian distribution after sin function.

3.3 Least Squares

In determined systems, there is only one solution. However, in overdetermined systems, this may not be true due to how the real world works. As no system is flawless, errors in the acquired data lead to each equation's solution not matching with each other. In such a situation, the best solution is which makes the error in each equation minimum. “Least squares” is a method to find the best solution for an overdetermined system of equations by approximating. In linear systems, least squares can be solved in closed form. The goal is to minimize the squared errors described as follows [23]:

$$Ax = b \tag{3.28}$$

$$e^2 = \left\| Ax - \hat{b} \right\|^2 \tag{3.29}$$

$$e^2 \underset{min}{\Longrightarrow} \frac{\partial e^2}{\partial x} = 0 \tag{3.30}$$

Where A and b are independent and dependent variables in a matrix-vector form, respectively, a minimum error can be found by setting the derivative to zero.

$$\frac{\partial e^2}{\partial x} = \frac{\partial[(Ax - \hat{b})^T (Ax - \hat{b})]}{\partial x} = 0 \tag{3.31}$$

$$\frac{\partial e^2}{\partial x} = \frac{\partial(\hat{b}^T\hat{b} - \hat{b}^T Ax - x^T A^T \hat{b} + x^T A^T Ax)}{\partial x} = 0 \tag{3.32}$$

$$x = (A^T A)^{-1} A^T \hat{b} \tag{3.33}$$

Unfortunately, in the real world, most systems are nonlinear and in the form of a function, $f(x) = \hat{z}$ that maps inputs to outputs. The commonly used non-linear method is Gauss-Newton. According to the Gauss-Newton method, the solution is found by iterating as follows [23].

$$e(x) = z - f(x) \tag{3.34}$$

$$e(x + \Delta x) \approx e + J_x \Delta x \tag{3.35}$$

The error function is linearized by Taylor series expansion and J_x is the jacobian matrix while Δx is the increment.

$$e^2(x + \Delta x) \approx (e + J_x \Delta x)^T (e + J_x \Delta x) \tag{3.36}$$

$$e^2(x + \Delta x) \approx e^T e + e^T J_x \Delta x + \Delta x^T J_x^T e + \Delta x^T J_x^T J_x \Delta x \tag{3.37}$$

Eq. (3.37) can be simplified as:

$$e^2(x + \Delta x) \approx c + 2b^T \Delta x + \Delta x^T H_h \Delta x \tag{3.38}$$

Where $c = e^T e$, $b^T = e^T J_x$ and $H_h = J_x^T J$. The equation can be solved by taking derivative with respect to Δx and setting derivative to zero.

$$\frac{\partial c + 2b^T \Delta x + \Delta x^T H_h \Delta x}{\partial \Delta x} = 0 \tag{3.39}$$

$$2b + 2H_h \Delta x = 0 \tag{3.40}$$

$$\Delta x = -H_h^{-1} b \tag{3.41}$$

$$\Delta x = -(J_x^T J_x)^{-1} J_x^T e \tag{3.42}$$

The state is updated as $x = x + \Delta x$ and all steps iterated until it converges. In other words, Δx becomes zero or very close to zero.

In EKF SLAM methods, the newly detected objects' positions are initialized by adding them to the state matrix. This addition causes the state matrix to grow gradually with the increase of the detected objects. So, the computation in a reasonable time becomes impossible after a while. To optimize the robot's state, the same objects must be determined in the next images again. Therefore, EKF methods assume that the data association problem has been solved. If objects cannot be detected again, they are defined as new objects, this type of repetition will cause the state matrix to grow faster. For this reason, a graph-based SLAM model was chosen in the thesis. In graph-based SLAM, robot poses are described as nodes, and observations are represented as links between the nodes. In other words, objects are not included in the state matrix. Thus, a more compact state model is constructed. To solve the graph-based SLAM problem, non-linear least squares methods can be used. In this thesis, the Gauss Newton algorithm is chosen for non-linear least-squares methods. The following factors also contribute to the choice of the graph-based SLAM method. EKF and GNA use Taylor series expansion to linearize non-linear functions and require the computation of Jacobian matrices. Both of them assume data distribution to be Gaussian, and one minimizes the variance error, and the other minimizes the squared errors. In other words, their working principles are similar. GNA can be thought of as the update part of the EKF, and it does not contain a prediction process. Therefore, the computational complexity of GNA for one iteration is lower than EKF. GNA is easier and faster to implement in programming languages.

Furthermore, as GNA is an iterative method, it deals with high nonlinearity better than EKF. Last but not least, besides the prediction model, process noise also needs to be known accurately. Otherwise, the EKF result diverges from the truth.

3.4 RANSAC (RANdom SAmple Consensus)

Random sample consensus method estimates the inliers by repeatedly iterating in a small portion of the data set. Due to errors, some of the data do not fit among the model and thus called outliers. A small random set is chosen to eliminate these outliers, and the model is estimated according to this set. Model fitness is compared with the rest of the data. Multiple random small sets are chosen, and computed models are compared with the data. Among these models, a model that best expresses the data is selected. According to a defined threshold, data parameters far from the final chosen model are selected as outliers and discarded from the data. A new model can be computed from the remaining inliers [24].

For example, when a sensor measures a tree trunk's height and diameter, some of the measurements will be wrong due to mismatches. If the model is known, for example, diameter and height are linearly dependent, and then incorrect measurements can be eliminated using RANSAC.

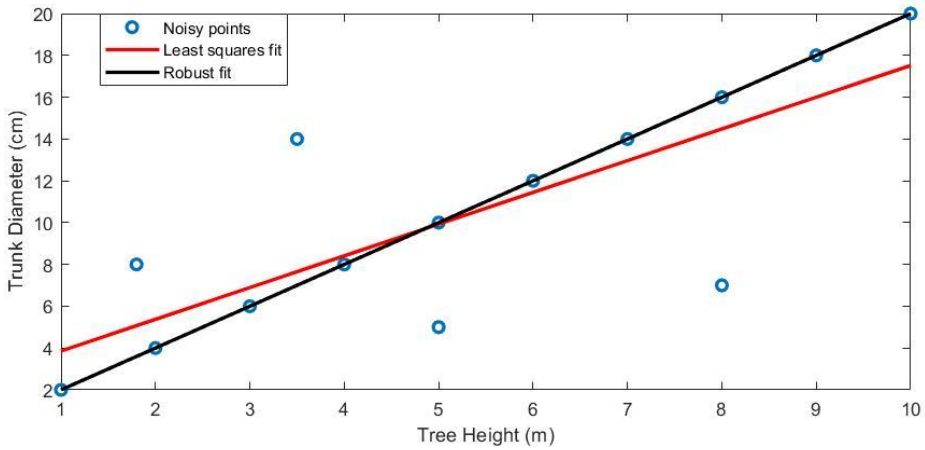


Figure 8 Line fitting using RANSAC.

If the outlier ratio is known, the number of required iterations for a specific accuracy can be found as follows [24]:

$$p_{in} = (1 - e_{outlier})^{s_{set}} \tag{3.43}$$

Where p_{in} is the probability of selecting one inlier, while $e_{outlier}$ and s_{set} are outlier ratio and minimum required random set size, respectively. Then the probability of selecting outlier T_{tm} times are:

$$1 - p_{in} = (1 - (1 - e_{outlier})^{s_{set}})^{T_{tm}} \tag{3.44}$$

Iteration number T_{tm} can be found by taking the logarithm of both sides.

$$\log(1 - p_{in}) = T_{tm} \log(1 - (1 - e_{outlier})^{s_{set}}) \tag{3.45}$$

$$T_{tm} = \frac{\log(1 - p_{in})}{\log(1 - (1 - e_{outlier})^{s_{set}})} \tag{3.46}$$

Probability of selecting inlier decreases as the minimum required random set size increases. For this reason, s_{set} is tried to be kept as small as possible.

3.5 Loop Closure

Loop closure is the detection of where the mobile robot has passed before. As errors accumulate while moving, the position of the robot shifts from the actual value. When a previously seen area is detected, this observation adds a dependency between the current pose and pose at the previously seen area [25]. According to the measurement's reliability, a weight can be assigned, and robot poses can be converged together. The problem is detecting the previously seen area. If the poses are mismatched, the robot pose will diverge. There are different methods to detect similar places for different slam techniques, such as iterative closest point for radar-based SLAMs and bag of words for visual SLAMs. Various methods use statistics and leverage deep learning algorithms to further increase accuracy [26]. Since the feature-based method is chosen in this thesis, the bag of words method is explained below.

3.5.1 Bag of Words

In a bag of words model, vocabulary vectors represent the images. Matching every taken image with each other is very time-consuming, so corresponding vocabulary vectors are compared to calculate the similarity between images. Vocabulary vectors

are commonly constructed based on the frequency of detected features in an image. In other words, a bag of words model is representing images with a histogram of detected features. Due to the presence of many features with different properties, using whole features is inefficient.

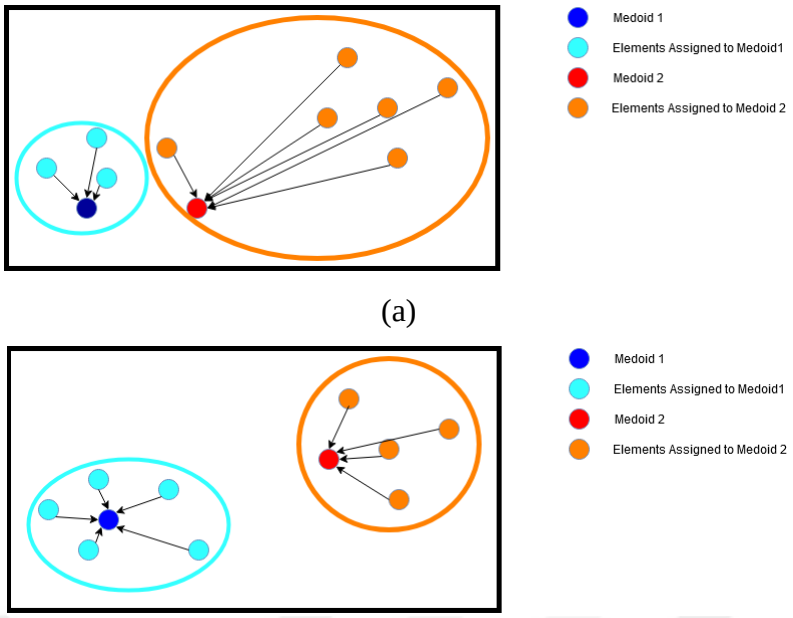
For this reason, similar features are grouped, and centers of these many groups are selected as words for vocabulary vectors. There are different ways to group features, such as Euclidian distance, Mahalanobis distance, cosine similarity, and hamming distance. The similarity between images can be computed using the following equation [27], [28].

$$Sim = 1 - \frac{1}{2}\left|\frac{H_{his_1}}{|H_{his_1}|} - \frac{H_{his_2}}{|H_{his_2}|}\right| \tag{3.47}$$

Where H_{his_1} and H_{his_2} are histograms of the first and second image, respectively.

3.5.2 K-Medoid Clustering

K-medoid clustering is a method of classifying similar data under the same group. K represents the number of groups that data is partitioned. Medoid is part of the data in the center of a group, while the mean of a data set is just an algebraic distance. Being part of the data makes medoid clustering less vulnerable to noise and extreme outliers as they can shift the mean from the actual center. Also, medoid clustering can be used on data types with no algebraic distance, such as binary data. Binary data consists of only zeros and ones, and hamming distance can be used as a similarity criterion. The most used medoid clustering technique is PAM, partitioning around medoids and converges faster than brute force searching but can be stuck at local minima as a small area is focused on. First, k medoids are chosen randomly or according to their similarity between the rest of the data. Then each point in the data is assigned to the closest medoid. In each medoid group, every point is selected and tested. Similarities between the selected point and other points in the group are calculated and summed together. The point with the smallest sum of similarity is chosen as the new medoid. After new medoids are found, each point in the data is assigned to the closest medoid. This step is iterated until there is no medoid change in all groups [29].



(a)

(b)

Figure 9 (a) Randomly selected medoids, (b) medoids after one iteration

CHAPTER IV

OBJECT TRACKING WITH IMAGE PROCESSING

Analyzing objects by taking their images has become important in computer systems in recent years. Availability of high computing performance, superior cameras for a much reasonable price, and the need for automated systems attracted attention to object tracking. Image analysis grouped under three main heading:

- Detection of objects according to their pattern, form, and shape.
- Tracking detected objects in other frames.
- Examination of tracks to understand the object's motion.

Object tracking applications focus on identifying object's movements (sensing unexpected movement, creature spotting, gesture recognition, managing traffic flow, planning vehicle navigation, and obstacle detection) and their interactions with each other. In brief, object tracking is computing the object's path moving around the camera scene while taking information about the object such as its shape, size, and orientation. Although tracking algorithms of objects and creatures are becoming important, their problems have not been fully solved yet because of some reasons;

- losing information while transforming from a 3D world to a 2D plane; some motions cannot be determined because of their complexity and randomness;
- deformable objects cannot be tracked in the next frame due to change of shape; light state of the scene causes a change in reflection and shadowing which affect the object color;
- Last but not least, processing time constrains where algorithm must run in a reasonable time like real-time applications [30].

There are different tracking algorithms, and some have been developed to solve these problems. A unique property of an object that defines it is critical in object tracking.

Various object properties are used to differentiate them from each other. They can be investigated under four main categories. Color, this property is affected by physical factors. The wavelength and frequency of the light and the object's response to the light determine the object's color. The most used color space in image processing is RGB (Red, Green, and Blue). Unfortunately, it is not a uniform space for human eye recognition. The human eye is sensitive to green color more than others [31]. There are other uniform color spaces, such as Lab and HSV, but they are susceptible to noise [32].

The second property is the edges of the object. Pixel intensity difference is high around the boundaries of an object. These are areas that are not much affected by the properties of light, and algorithms often use the edge of an object to make detection more reliable. Some of the most popular edge detectors are Canny, Laplace, and Sobel edge detectors. Another visual property is optical flow. When an object moves, its intensity pattern also moves with it. Optical flow can be defined as the distribution of apparent motion velocities of these intensity patterns in an image. In other words, optical flow consists of displacement vectors that describe the motion of pixels in a small region of the image (5x5). The last property is texture. The object's smoothness and roughness are the main factors affecting the detection of the object. Texture properties are less affected by light than colors but need an extra step to extract descriptors [30].

There are several methods developed to detect similarities between frames by using these object properties to their advantage. These methods can be grouped under four titles [30].

- Point Detectors

  Point Detectors are based on texture comparison. To find distinctive textures first finds interest points like corner and edge points where neighbor points have a unique sequence.

- Segmentation

  The segmentation method uses color and gradient information to group pixels in an image to multiple regions according to their similarities.

- Background Subtraction

Background subtraction uses both color and texture properties to detect changes between frames by subtracting a previous frame from the current frame.

- Machine/Deep Learning

  Learning Algorithms are functions that try to find the local minima or, if lucky, the global minima of the given problem set. They consist of nodes carrying weights, which are calculated by training lots of data. By using these weights, inputs are mapped to outputs. In other words, learning algorithms give an estimated output for a certain input.

How the aforementioned tracking methods using object properties are adapted to VSLAM algorithms and how object tracking can be used to compute camera orientation and translation are explained in the following pages.

4.1 Object Tracking in VSLAM Algorithms

VSLAM Algorithms can be classified under two main groups as a direct and feature-based method [2], [3]. The direct method achieves tracking by using whole image pixels. This causes high-quality mapping but runs slow as there is a high amount of pixel information. There are three general approaches for the direct method such as DTAM, LSD-SLAM, and DSO. The feature-based method achieves tracking by using descriptive image features. This causes low-quality maps but runs fast as less information is used to describe objects. There are three main approaches for a feature-based method: MonoSLAM, PTAM, and ORB-SLAM [33].

Some methods come to the fore considering the advantages of algorithms using the properties of the objects mentioned above. The biggest factor in VSLAM algorithms is the computation time. Due to the real-time requirements, the system has to make quick decisions. Therefore, the most used method in VSLAM algorithms is texture-based identification and classification like feature point finding and descriptor extraction. The other advantage of this method is the accuracy can be optimized by an increasing amount of feature points detected, using outlier detection algorithms like RANSAC or modified descriptor algorithms that are invariant to affine transform at

the expense of the calculation time. For these reasons, the feature-based method was chosen in this thesis.

Feature-based image matching consists of two main parts, descriptive feature detection, and feature descriptor extraction. Descriptive feature detection is the detection of pixels or pixel patches having rich information. For example, corners and areas having different brightness or color than surrounding areas (neighbor pixels). Feature descriptor extraction is to get information about neighbor pixels around descriptive feature points for matching. Namely, it can be thought of as a DNA or fingerprint to differentiate descriptive feature points from each other. Various feature detection and descriptor extraction algorithms are shown in Figure 10, and those used in this thesis are explained in detail below.

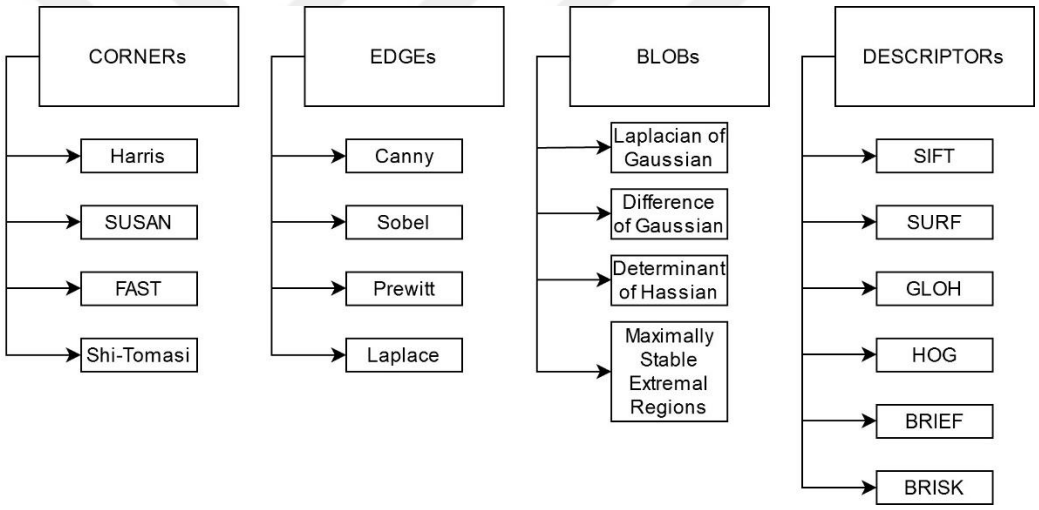


Figure 10 Feature detection and matching algorithms.

4.1.1 Harris Corner Detection

In Harris detector, a small patch in size $S_p \times S_p$ is defined to scan the whole image by shifting 1-by-1. In each shift operation, it calculates an intensity difference between the current patch and the previous patch. The mathematical calculation is given below [34]:

$$E(u_p.v_p) = \sum_{x_p,y_p} w_{win}(x_p, y_p)\left[I(x_p + u_p, y_p + v_p) - I(x_p, y_p)\right]^2 \qquad (4.1)$$

Where u_p, v_p is the displacement of the pixel in horizontal and vertical directions, respectively while w_{win} is the window function determines the weight of the pixels. w_{win} can be a square window ($w_{win} = 1$) or a Gaussian one. x_p and y_p are the positions of pixels inside the patch area. The above equation can be approximated by Taylor series expansion. According to [35], multivariable functions can be expanded as follows:

$$T(x_1, \dots, x_d) = \sum_{n_1=0}^{\infty} \sum_{n_d=0}^{\infty} \frac{(x_1 - a_1)^{n_1} \dots (x_d - a_d)^{n_d}}{n_1! \dots n_d!} \left(\frac{\partial^{n_1+\cdots+n_d} f}{\partial x_1^{n_1} \dots \partial x_d^{n_d}} \right) (a_1, \dots, a_d) \quad (4.2)$$

The final equation is derived as shown below by using the first three expansion elements:

$$T(x, y) = f(a, b) + (x - a) f_x(a, b) + (y - b) f_y(a, b) \quad (4.3)$$

$$T(x_p + u_p, y_p + v_p) = f(x_p, y_p) + (x_p + u_p - x_p) f_x(x_p, y_p) + (y_p + v_p - y_p) f_y(x_p, y_p) \quad (4.4)$$

$$I(x_p + u_p, y_p + v_p) = f(x_p, y_p) + u_p I_x + y_p I_y \quad (4.5)$$

When Eq. (4.5) substituted in Eq. (4.1)

$$E(u_p, v_p) \approx w(x_p, y_p) \sum_{x,y} [(I(x_p, y_p) + I_x u_p + I_y v_p - I(x_p, y_p)]^2 \quad (4.6)$$

$$E(u_p, v_p) \approx w_{win}(x_p, y_p) \sum_{x,y} u_p^2 I_x^2 + 2 u_p v_p I_x I_y + v_p^2 I_y^2 \quad (4.7)$$

$$E(u_p, v_p) \approx [u_p \quad v_p] \left(w_{win}(x_p, y_p) \sum_{x_p, y_p} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u_p \\ v_p \end{bmatrix} \quad (4.8)$$

$$M_{tensor} = \left(w_{win}(x_p, y_p) \sum_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \quad (4.9)$$

M_{tensor} is called structure tensor while I_x and I_y are derivatives of image I in horizontal and vertical directions. To find the image derivatives, a differentiation operator that approximately computes the gradient of the image intensity is used. These operators use $N_k x N_k$ kernels to convolute images together where N_k is the size of the kernel. According to Prewitt, a $3x3$ kernel is used as follows [36]:

$$K_{p_x} = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix}, K_{p_y} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad (4.10)$$

In addition, for Sobel, a 3x3 kernel is used as below [37]:

$$K_{s_x} = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, K_{s_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (4.11)$$

Where $K_{p_x}, K_{s_x}$and $K_{p_y} K_{s_y}$ are kernels for x and y directions, respectively. Image derivative can be found by convolving the image with respective kernels.

$$I_x = K_{s_x} * I, I_y = K_{s_y} * I \quad (4.12)$$

Harris response is then calculated to decide if the patch area is a corner. It is calculated by using eigenvalues of the matrix M_{tensor}.

$$R_{harris} = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \quad (4.13)$$

$$\lambda_1 \lambda_2 = \det(M_{tensor}) \quad (4.14)$$

$$\lambda_1 + \lambda_2 = tr(M_{tensor}) \quad (4.15)$$

$$R_{harris} = \det(M_{tensor}) - k_{harris}\left(tr(M_{tensor})\right)^2 \quad (4.16)$$

k_{harris} is a constant and suggested to be between 0,04 and 0,06. λ_1 and λ_2 are eigenvalues of the matrix M_{tensor} and R_{harris} is Harris response, and $tr(\,)$ is trace operation. If R_{harris} is bigger than a predefined threshold, the patch area is considered as the corner. In [38], it is proposed to use a minimum of eigenvalues for the response criteria.

$$R_{harris} = min(\lambda_1, \lambda_2) \approx \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{det(M_{tensor})}{tr(M_{tensor})} \quad (4.17)$$

4.1.2 FAST Corner Detection

Features from the accelerated segment test, FAST is one of the fastest corner detection algorithms. It is mostly preferred in projects needed to run in real-time. Instead of a

square patch, FAST uses a circle around the candidate pixel. The pixel is considered a corner according to the properties of this circle that consists of neighbor pixels. The working principle of the algorithm is as follows [39]:

- In an 8-bit greyscale image, a pixel's intensity is defined with integers between 0 and 255.
- Pixel having intensity I_{ints_p} is selected.
- An appropriate threshold value t_{fast} is selected.
- A circle consists of 16 pixels is defined around the selected pixel.
- Selected pixel is corner if 12 contiguous pixels' intensities in the circle are greater or smaller than $I_{ints_p} + t_{fast}$ and $I_{ints_p} - t_{fast}$, respectively.

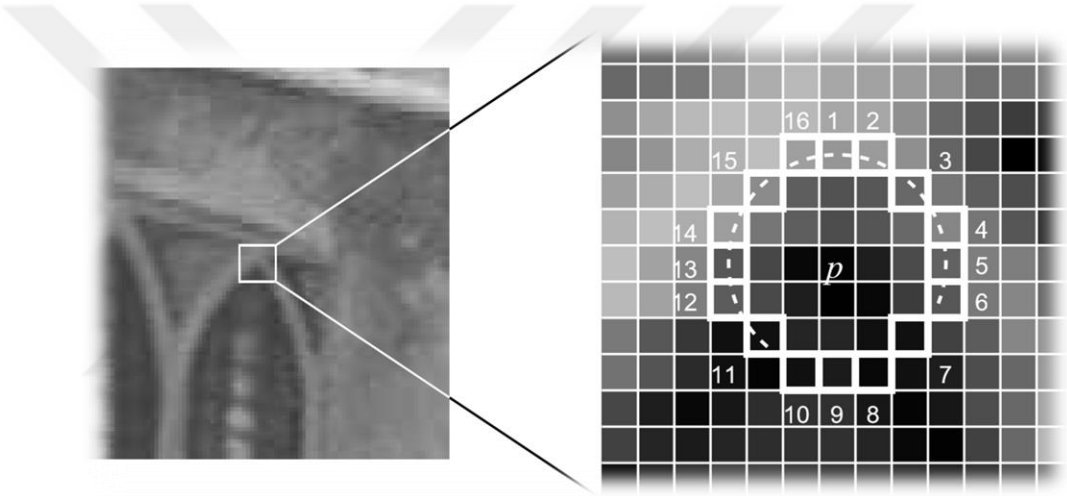


Figure 11 Fast corner detection [39]

To increase the computation speed [39], two horizontal pixels (Pixel 13 and Pixel 5 in Figure 11) in the circle are compared with the selected pixel. If both pixels are not in the required intensity, then the selected pixel cannot be a corner. Else, two vertical pixels (Pixel 1 and Pixel 9 in Figure 11) in the circle are compared with the selected pixel. Three of these four pixels must satisfy the rule, or the selected pixel cannot be a corner. If at least three of the four pixels satisfy the rule, the rest of the eight pixels in the circle are compared. In this way, comparing all the sixteen pixels is prevented if the selected pixel is not a corner.

4.1.3 BRIEF Descriptor Extraction

Binary robust independent elementary features, BRIEF is one of the fastest descriptor algorithms. The BRIEF vector consists of binary values zeros and ones. For this reason, a comparison of two binary vectors is made by computing Hamming distance between two vectors. As Hamming distance computation is just an XOR operation, it is at least two orders of magnitude faster than the other state-of-the-art matching algorithms such as SIFT and SURF. The working principle of the algorithm is as follows [40]:

- A patch in size of $S_p \times S_p$ is defined around detected features.
- Random two pixels are selected in Gaussian distribution where the standard deviation is:

$$\sigma = 0.04 S_p^2 \tag{4.18}$$

- n numbers of pixel pairs are selected and only initialized once.
- n pixel pairs are converted to a bit-string by following criteria:

$$v_{i,\dots,n} = \begin{cases} 1 & if\ I_{ints_{p1}} < I_{ints_{p2}} \\ 0 & if\ I_{ints_{p1}} > I_{ints_{p2}} \end{cases} \tag{4.19}$$

- A bit-string is assigned for every feature detected.
- Hamming distance is calculated between every feature's corresponding bit-string for matching.
- A threshold value t_{brief} is selected. Features with a Hamming distance less than the threshold t_{brief} are matched.
- In multiple matching, features having the smallest distance are chosen.

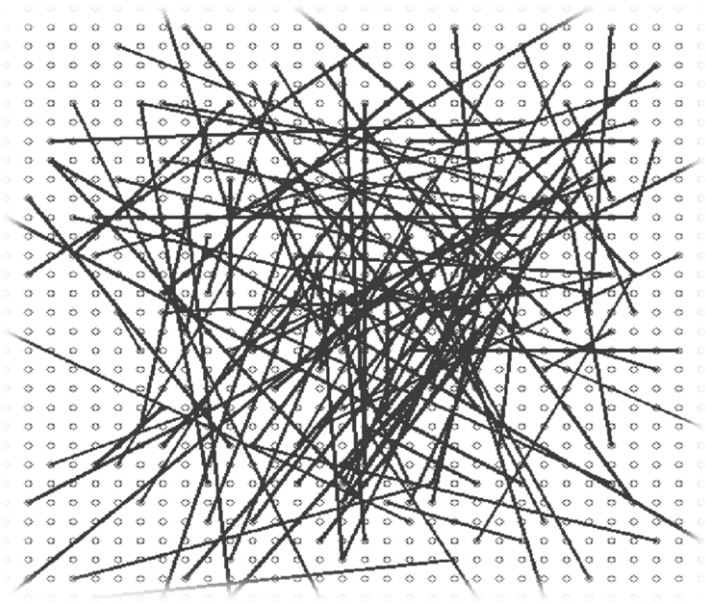


Figure 12 Randomly chosen pixel pairs in a patch [40]

4.1.4 Steered BRIEF Descriptor

Steered BRIEF is the modified version of the BRIEF descriptor, and it is invariant to rotation changes. The rotation invariance of the descriptor is achieved by computing patch orientation beforehand. To find the orientation, the intensity centroid is used. According to Rosin, patch centroid can be calculated as follows [41]:

$$m_{ab} = \sum_{x_p, y_p} x^a y^b I_{ints}(x_p, y_p), \quad x_p, y_p \tag{4.20}$$

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \tag{4.21}$$

Where m and C are the moments and intensity centroid, respectively. $I_{ints}(x_p, y_p)$ is the intensity of the pixel at location x_p, y_p in image patch where $x_p, y_p = 0$ at the center of the patch, O. Then a vector is defined from center to centroid $\overrightarrow{OC}$. The rotation angle θ_p, then can be derived as:

$$\theta_p = atan2(m_{01}, m_{10}) \tag{4.22}$$

After rotation of the patch is calculated, it is applied to the positions of n pixel pair selected in the original BRIEF algorithm by multiplying the corresponding rotation matrix.

$$S_p = \begin{pmatrix} x_1, \dots, x_n \\ y_1, \dots, y_n \end{pmatrix} \tag{4.23}$$

$$S_\theta = R_{\theta_p} S_p \tag{4.24}$$

Where S_p and S_{p_θ} are the positions of pixel pairs and steered pixel pairs, respectively. R_{θ_p} is the 2D rotation matrix corresponds to θ_p angle. Furthermore, instead of a randomly selected pixel set, a specific pixel set can be chosen according to trained data to increase the accuracy [42].

4.2 Camera Calibration

To measure depth from cameras, a geometry problem must be solved. Some assumptions are made to simplify the problem. For example, a camera aperture is a point and has no lens to focus light rays. That is why this mathematical method of describing the relationship between a real scene and its projection onto an image plane is called a pinhole camera model. The pinhole camera model is used to calibrate the camera by defining camera constants related to the sensor size and its orientation with respect to the aperture inside a camera. The pinhole model is in the following form [43]:

$$p = \lambda A_c [R|t] P \tag{4.25}$$

Where P and p are world coordinates of a point and corresponding pixel coordinates in an image, respectively, in a column vector. $[R|t]$ is a rotation-translation matrix that shows the camera's orientation in the world frame. A_c is called a camera matrix and consists of constant camera parameters. As transforming from 3D to 2D, there is an information loss. Consequently, this geometry problem can be computed up to a scale noted as λ [43].

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (4.26)$$

The camera matrix consists of focal lengths and principal points in pixel length in horizontal and vertical directions. An object or a pattern having a known size can be used to find the camera matrix. One of the most used ways is a checkerboard pattern. It is easy to detect corner points with minimal error using corner detection algorithms. The world coordinate origin is selected on the checkerboard pattern to simplify the pinhole camera equation. If checkerboard is assumed a perfect plane, then detected corner points would have zero displacements in the z-axis [43].

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \quad (4.27)$$

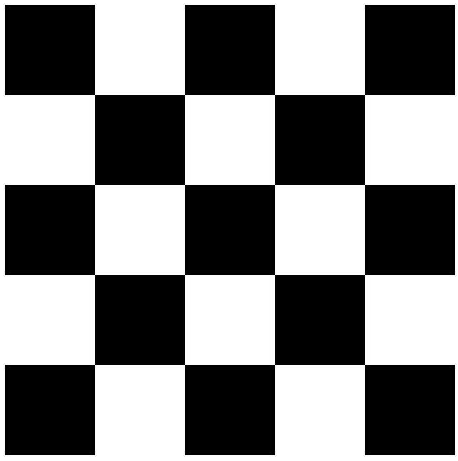


Figure 13 Checkerboard pattern

In matrix-vector multiplication, the elements of a rotation-translation matrix that corresponds to vector element Z will be zero. This column will not affect the multiplication, so it can be discarded. A homography then can be defined as [43]:

$$H = \lambda A_c [r_1 r_2 t] \quad (4.28)$$

Where H is a $3x3$ homography matrix while $\lambda = \frac{1}{s}$ is a scale factor and r_1, r_2, t are non-discarded columns in the rotation-translation matrix, respectively. Homography can be found by singular value decomposition after simplified to a vector h.

$$\begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} us \\ vs \\ s \end{bmatrix} \tag{4.29}$$

$$h_1 X + h_2 Y + h_3 - us = 0 \tag{4.30}$$

$$h_4 X + h_5 Y + h_6 - vs = 0 \tag{4.31}$$

$$h_7 X + h_8 Y + h_9 = s \tag{4.32}$$

These equations can be combined in a matrix-vector multiplication

$$Lh = 0. \tag{4.33}$$

Where:

$$L = \begin{bmatrix} X & Y & 1 & 0 & 0 & 0 & -uX & -uY & -u \\ 0 & 0 & 0 & X & Y & 1 & -vX & -vY & -v \end{bmatrix}, \qquad h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} \tag{4.34}$$

h vector is a right singular vector of L matrix that corresponds to the smallest singular value. Vector h has nine elements but has only eight unknowns; one of them is a scale. When all elements divided to, for example, the last element h_9 vector h left with eight unknowns. For this reason, there must be eight observations to solve the equation. Each point detected in the calibration pattern image contributes to two observations. At least four points are necessary to be detected to solve for homography.

If the homography matrix is inspected under columns, then columns r_1 and r_2 can be derived as:

$$[h_1 h_2 h_3] = \lambda A_c [r_1 r_2 t] \tag{4.35}$$

$$r_1 = A_c^{-1} h_1 \tag{4.36}$$

$$r_2 = A_c^{-1} h_2 \tag{4.37}$$

It is known that rotation axes are in the unit distance, and they are perpendicular to each other. So, their magnitudes will be one, and dot products will be zero [43].

$$r_1^T r_2 = 0, \tag{4.38}$$

$$\|r_1\| = \|r_2\| = 1 \tag{4.39}$$

When equations (4.36) and (4.37) substitute in (4.38) and (4.39), respectively

$$h_1^T A_c^{-T} A_c^{-1} h_2 = 0 \tag{4.40}$$

$$h_1^T A_c^{-T} A_c^{-1} h_1 - h_2^T A_c^{-T} A_c^{-1} h_2 = 0 \tag{4.41}$$

To simplify the equations, camera matrix components will be described as B_c.

$$B_c = A_c^{-T} A_c^{-1} \tag{4.42}$$

Then equations (4.40) and (4.41) becomes:

$$h_1^T B_c h_2 = 0 \tag{4.43}$$

$$h_1^T B_c h_1 - h_2^T B_c h_2 = 0 \tag{4.44}$$

Furthermore, the matrix B_c can be simplified to a vector using Cholesky decomposition. Cholesky decomposition says that a symmetric matrix M_{sym} can be decomposed into the following form:

$$M_{sym} = NN^T \tag{4.45}$$

The matrix B_c is also in the same form, so it is a symmetric matrix.

$$B_c = A_c^{-T} (A_c^{-T})^T = A_c^{-T} A_c^{-1} \tag{4.46}$$

A vector b_c with six elements is constructed by using non-similar elements of the matrix B_c.

$$B_c = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} \tag{4.47}$$

$$b_c = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}] \tag{4.48}$$

Corresponding terms in multiplication are also described as a vector v_{ij} where [43]:

$$v_{ij} = \left[h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}\right]^T \tag{4.49}$$

Eq. (4.43) and Eq. (4.44) further simplified to:

$$h_1^T B_c h_2 = v_{12}^T b_c = 0 \tag{4.50}$$

$$h_1^T B_c h_1 - h_2^T B_c h_2 = (v_{11} - v_{12})^T b_c = 0 \tag{4.51}$$

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b_c = V b_c = 0 \tag{4.52}$$

As vector b_c has six unknowns, there need to be six observations to solve the equation. Each taken pattern image contributes two observations to the equation. For this reason, there must have at least three images of the calibration pattern taken from different orientations. Because of errors, the multiplication result cannot be zero for four or more taken images; instead, equations are solved to find the minimum error for vector b_c. The vector b_c is found by singular value decomposition. Its equal to the right singular vector of V corresponds to the smallest singular value. Once vector b_c is found, camera parameters can be calculated as follows [43]:

$$cy = \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B_{22} - B_{12}^2} \tag{4.53}$$

$$\lambda = B_{33} - \frac{[B_{13}^2 + cy(B_{12}B_{13} - B_{11}B_{23})]}{B_{11}} \tag{4.54}$$

$$fx = \sqrt{\frac{\lambda}{B_{11}}} \tag{4.55}$$

$$fy = \sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}^2}} \tag{4.56}$$

$$cx = \frac{cy}{fy} - \frac{B_{13} fx^2}{\lambda} \tag{4.57}$$

Errors stem from different reasons. The camera sensor is affected by Gaussian random noise; shifts occur due to alignment of the lens, and impurities on the lens surface cause distortion. Distortion has the most visible effect on the image. For this reason, many applications undistort the image before extracting information.

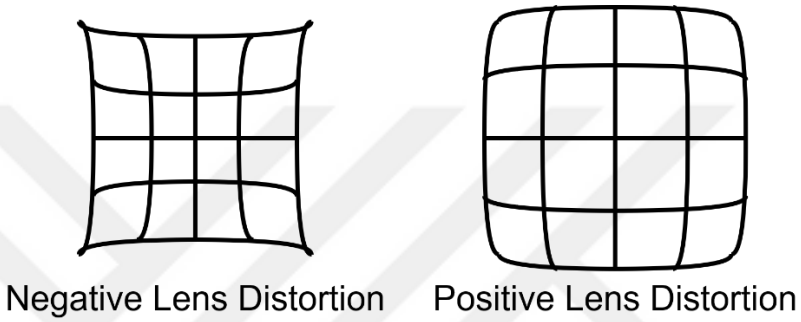


Figure 14 Lens Distortion Types

The distortion model is in the following form [43]:

$$\bar{x} = \breve{x} + \breve{x}[k_1 r + k_2 r^2] \tag{4.58}$$

$$\bar{y} = \breve{y} + \breve{y}[k_1 r + k_2 r^2] \tag{4.59}$$

$$r_p = \breve{x}^2 + \breve{y}^2 \tag{4.60}$$

Where $\breve{x}$, $\breve{y}$ and r_p are the normalized camera coordinates of the point and norm of pixel coordinates, respectively while $\bar{x}$ and $\bar{y}$ is the ideal distortion-free normalized image coordinates. k_1 and k_2 are the distortion coefficients. Previously estimated camera parameters are used to estimate the ideal pixel positions [43].

$$\begin{bmatrix} (u - cx)r & (u - cx)r^2 \\ (v - cy)r & (v - cy)r^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} \hat{u} - u \\ \hat{v} - v \end{bmatrix} \tag{4.61}$$

$$Dk = d \tag{4.62}$$

Where u and $\hat{u}$ are ideal pixel position (not observable) and the observed pixel position of the point.

Distortion coefficients are found from linear least square solution by the following form:

$$k = (D^T D)^{-1} D^T d \tag{4.63}$$

Camera parameters and distortion coefficients can be further refined using non-linear minimization algorithms [43].

Figure 15 Distorted and undistorted images, respectively.

4.3 Distance Measurement

Detecting objects' positions is necessary to understand the environment around the robot. According to object positions, the robot should effectively take actions like path planning and obstacle avoidance to achieve the given task. For a useable map of the environment, object positions must be measured accurately, or at least the drawn map should have some consistency between regions. There are several ways to measure the distance of tracked objects and further refine them.

4.3.1 Distance Measurement with Stereo Triangulation Method

A depth equation can be found from the pinhole camera model for stereo systems. When the world frame is parallel and has the same origin as the camera, the rotation matrix becomes an identity matrix, and translations corresponding to x, y, and z-axis become zero.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.64}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.65}$$

$D = Z$ is the distance between the feature point and the camera. As stereo cameras are conditioned next to each other on the x-axis, depth can be computed using the distance between two cameras and the row of the camera matrix corresponding to the x-axis, which is the first row.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fxX + cxD \\ fyY + cyD \\ D \end{bmatrix}, \qquad u = \frac{f_x X}{D} + cx \tag{4.66}$$

$$u_1 = \frac{f_1 X_1}{D} + c_1, \qquad u_2 = \frac{f_2 X_2}{D} + c_2 \tag{4.67}$$

Where X_i is the distance between the point and the corresponding camera on the x-axis while f_i and u_i are the focal length and pixel coordinate in the horizontal axis, respectively. The distance between the point and the cameras can be found as follows [44]:

$$\frac{(u_1 - c_1)}{f_1} = \frac{X_1}{D}, \qquad \frac{(u_2 - c_2)}{f_2} = \frac{X_2}{D} \tag{4.68}$$

$$\frac{f_2(u_1 - c_1) - f_1(u_2 - c_2)}{f_1 f_2} = \frac{X_1 - X_2}{D} \tag{4.69}$$

$$b = X_1 - X_2 \tag{4.70}$$

$$D = \left(\frac{f_2(u_1 - c_1) - f_1(u_2 - c_2)}{f_1 f_2 b} \right)^{-1} \tag{4.71}$$

Where b is the distance between two cameras.

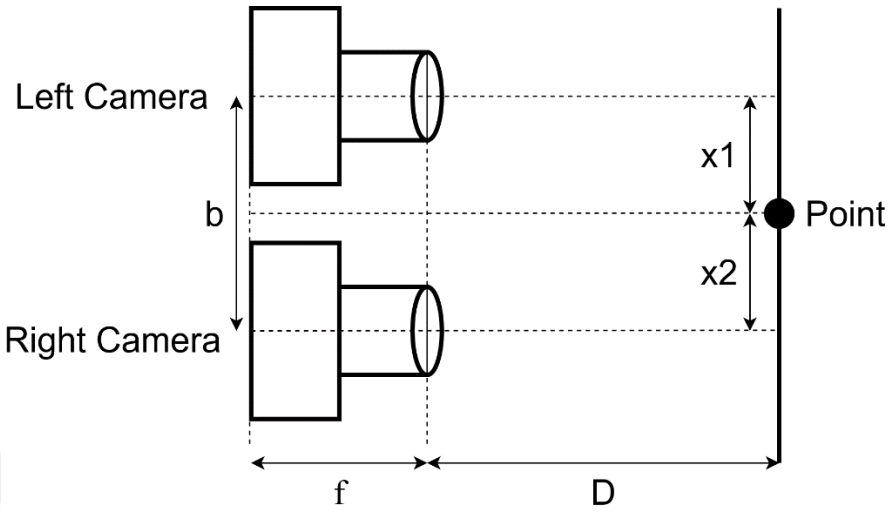


Figure 16 Stereo geometry [44].

4.3.2 Distance Measurement with Epipolar Geometry

In the triangulation method, a physical distance constraint between stereo cameras is used. Simultaneously, epipolar geometry is based on view geometry where an epipolar plane intersects both image planes in line. For this reason, epipolar geometry can work on single-camera systems but up to a scale factor. Any point X in the world is seen by two cameras and noted as x and x' in the first and second camera, respectively. If an imaginary line is drawn from the first camera's center to point X, this imaginary line is also seen as a second camera line. The line seen by the second camera is called an epipolar line. This situation is also the same for vice versa. If another imaginary line is drawn from the first camera's center to the second camera's center, this line intersects the first image plane at point e and the second image plane at point e'. The line and intersection points are called baseline and epipole points, respectively [45].

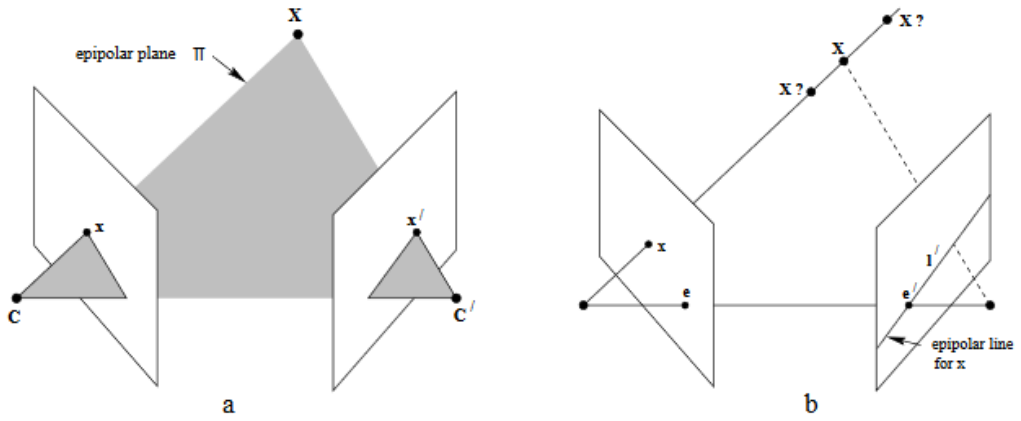


Figure 17 Epipolar geometry [45]

Both x and x' are the projection of point X and the transformation from x to x' can be described with a 2D homography H where:

$$x' = Hx \tag{4.72}$$

To find the epipolar line, the geometric definition of a standard line can be used:

$$ax + by + c = 0 \tag{4.73}$$

$$l = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{4.74}$$

The dot product of line l and any point $x = [x \quad y \quad 1]'$ on line l is zero, so they are orthogonal to each other.

$$x^T . l = [x \quad y \quad 1] \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 0 \tag{4.75}$$

This orthogonality is valid for both epipole e' and point x' on epipolar line l'. In other words, both epipole e' and point x' are orthogonal to epipolar line l'.

$$e'^T . l' = 0 \tag{4.76}$$

$$x'^T . l' = 0 \tag{4.77}$$

So, to find a line orthogonal to both point vectors, the cross product rule can be used.

$$e' \times x' = l' \tag{4.78}$$

Substituting Eq. (4.72), one can derive the epipolar line as:

$$e' \times Hx = l' \tag{4.79}$$

The term $e' \times H$ is called as F, fundamental matrix.

$$Fx = l' \tag{4.80}$$

The fundamental matrix describes a transformation from a 2D plane to a 1D line. When l' is substituted in Eq. (4.77), the fundamental matrix's basic features obtained [45].

$$x'^{T} Fx = 0 \tag{4.81}$$

In Eq. (4.81), if normalized image coordinates are used, the transformation is described by an E, essential matrix instead of a fundamental matrix.

$$E = \begin{bmatrix} es_1 & es_2 & es_3 \\ es_4 & es_5 & es_6 \\ es_7 & es_8 & es_9 \end{bmatrix} \tag{4.82}$$

In normalized camera coordinates, the transformation between cameras can be directly described with a rotation and a translation [45].

$$\check{x}' = R(\check{x} - t) \tag{4.83}$$

$$E = [t]_{\times} R \tag{4.84}$$

Where R is a rotation matrix and $[t]_x$ is the matrix representation of a translation vector cross product. This representation is a skew-symmetric matrix. Where:

$$[t]_{\times} = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}, \qquad t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{4.85}$$

$$\breve{x}'^{T} E\, \breve{x} = 0 \tag{4.86}$$

Where $\breve{x}$ and $\breve{x}'$denote the normalized image coordinates of the pixels x and x', respectively.

$$\breve{x} = \begin{bmatrix} \breve{u} \\ \breve{v} \\ 1 \end{bmatrix} \tag{4.87}$$

$$\breve{x}' = \begin{bmatrix} \breve{u}' \\ \breve{v}' \\ 1 \end{bmatrix} \tag{4.88}$$

An essential matrix can be derived by simplification of Eq. (4.86) into vector multiplication.

$$es.\tilde{x} = 0 \tag{4.89}$$

Where es and $\tilde{x}$ are:

$$es = \begin{bmatrix} es_1 \\ es_2 \\ es_3 \\ es_4 \\ es_5 \\ es_6 \\ es_7 \\ es_8 \\ es_9 \end{bmatrix}, \qquad \tilde{x} = \begin{bmatrix} \breve{u}'\breve{u} \\ \breve{u}'\breve{v} \\ \breve{u}' \\ \breve{v}'\breve{u} \\ \breve{v}'\breve{v} \\ \breve{v}' \\ \breve{u} \\ \breve{v} \\ 1 \end{bmatrix} \tag{4.90}$$

The vector es has nine unknowns, but one is a scale, so there needs to be a minimum of eight points to solve for the essential vector using singular value decomposition [45].

$$es^{T}\tilde{x}_k = 0, \qquad k = 1 \ldots 8, \ldots, n \tag{4.91}$$

The vector es is the left singular vector of $\tilde{x}_k$ matrix corresponding to the smallest singular value. Due to errors in the pixel coordinates, the resulting essential matrix does not satisfy a true essential matrix's constraints. As it is a multiplication of a rotation and a skew-symmetric matrix, the essential matrix must have skew-symmetric

properties. Any 3×3 skew-symmetric matrix has two equal singular values, and the last singular value is equal to zero. The third and smallest singular value is set to zero to estimate a better essential matrix [45].

$$E \underset{svd}{\Leftrightarrow} U\Sigma V^T \tag{4.92}$$

Symbol Σ indicates the diagonal matrix of singular values while U and V are left and right singular vectors of essential matrix E.

$$\Sigma = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix}, \qquad s_1 > s_2 > s_3 \tag{4.93}$$

$$\hat{\Sigma} = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.94}$$

$$\hat{E} = U\hat{\Sigma}V^T \tag{4.95}$$

After the new essential matrix is computed, rotation and translation values are found by applying singular value decomposition on this new essential matrix.

$$\hat{E} \underset{svd}{\Leftrightarrow} \hat{U}\hat{\Sigma}'\hat{V}^T \tag{4.96}$$

Where $\hat{U}, \hat{\Sigma}', \hat{V}$ are singular value decomposition components of the newly estimated essential matrix, $\hat{E}$.

To find rotation and translation, an orthogonal and skew-symmetric matrix, W is constructed.

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.97}$$

First, an educated guess is made to derive rotation and translation using W matrix, and then it is proven to be true.

$$[t]_{\times} = \hat{U}\, W \hat{\Sigma}' \hat{U}^T \tag{4.98}$$

$$R = \hat{U} W^{-1} \hat{V}^T \tag{4.99}$$

When equations (4.98) and (4.99) are substituted in Eq. (4.84), the result is equal to an essential matrix. So, these equations are proven to be correct.

$$[t]_{\times} R = \hat{U}\, W \hat{\Sigma}' \hat{U}^T\, \hat{U} W^{-1} \hat{V}^T = \hat{U} \hat{\Sigma}' \hat{V}^T = E \tag{4.100}$$

Taking the inverse of W and W^{-1} terms in Eq. (4.98) and Eq. (4.99) also satisfy Eq. (4.100).

$$[t]_{\times_2} = \hat{U}\, W^{-1} \hat{\Sigma}' \hat{U}^T \tag{4.101}$$

$$R_2 = \hat{U} W \hat{V}^T \tag{4.102}$$

For this reason, there are four possible solutions to find orientation between camera frames. Although there are four solutions, three of them are imaginary, and only one is real. To find the real solution, points are projected according to four orientation solutions; in three solutions, projected points are behind the first or second camera. One solution will give the correct position of the points where they are in front of both cameras. In other words, only one solution gives points that are seen by both cameras. Due to errors, it is advised to derive the translation vector using singular value decomposition [45].

$$[t]_{\times} R t = E t = 0 \tag{4.103}$$

Translation vector t is equal to the right singular vector of in $\hat{V}^T$ which corresponds to the smallest singular value. Inversing W in Eq. (4.98) only changes the sign of the translation vector. For this reason, $t_2 = -t$ equation can be used to find four possible solutions [41]. Detected points can be projected as follows [46]:

$$\breve{u}' = \frac{X'}{Z'} = \frac{r_1 (P - t)}{r_3 (P - t)} \tag{4.104}$$

Where $P = [X, Y, Z]^T$ and $P' = [X', Y', Z']^T$ are world coordinates of a point P respect to the first and second camera, respectively. P vector is normalized by dividing to Z, to simplify the equation.

$$\breve{u}' = \frac{X'}{Z'} = \frac{r_1\left(\breve{x} - \frac{t}{Z}\right)}{r_3(\breve{x} - \frac{t}{Z})} \tag{4.105}$$

Then equation is solved for Z

$$Z = \frac{(r_1 - \breve{u}' r_3)t}{(r_1 - \breve{u}' r_3)\breve{x}} \tag{4.106}$$

After Z is found, X and Y coordinates of the point P can be computed as

$$\begin{bmatrix} X \\ Y \end{bmatrix} = Z\breve{x} \tag{4.107}$$

The vector P can also be found by using $\breve{v}'$ and r_2 instead of $\breve{u}'$ and r_1 respectively [46].

4.3.3 Stereo Rectification and Disparity Calculation

Rectification is transforming image planes so that they become parallel to each other. In other words, epipolar lines in each image become parallel. Rectification helps to simplify the data association problem from 2D to 1D. After rectification, detected points in the first camera will be observed in the same row in the second camera. For image planes to become parallel, they must share the same x-axis. The baseline that intersects both image planes can be used as a common x-axis. The vector in the baseline direction is the translation vector between cameras that are calculated before in Eq. (4.98) or Eq. (4.101) [47].

$$r_{rect_1} = \frac{t}{\|t\|} \tag{4.108}$$

The only thing known about the new y-axis is it is orthogonal to the new x-axis r_{rect_1}. An orthogonal vector can be found by the cross product of the new x-axis and old z-axis then normalized. Due to anticommutativity, the cross product $i \times k = -j$

$$-r_{rect_2} = r_{rect_1} \times [0\ 0\ 1]^T = [t_2\ -t_1 0]^T \tag{4.109}$$

$$r_{rect_2} \underset{norm}{\Longrightarrow} \frac{1}{\sqrt{t_1^2 + t_2^2}} [-t_2\ t_1 0]^T \tag{4.110}$$

The new z-axis is the cross product of two new axes r_{rect_1} and r_{rect_2}.

$$r_{rect_3} = r_{rect_1} \times r_{rect_2} \tag{4.111}$$

Rectification transformation can be represented as a matrix R_{rect}.

$$R_{rect} = \begin{bmatrix} r_{rect_1} \\ r_{rect_2} \\ r_{rect_3} \end{bmatrix} \tag{4.112}$$

$$R_{l_{rect}} = AR_{rect}A^{-1} \tag{4.113}$$

$$R_{r_{rect}} = AR_{rect}R^{-1}A^{-1} \tag{4.114}$$

$$\breve{x} = \begin{bmatrix} \breve{u} \\ \breve{v} \\ 1 \end{bmatrix} = R_{l_{rect}} x \tag{4.115}$$

$$\breve{x}' = \begin{bmatrix} \breve{u}' \\ \breve{v}' \\ 1 \end{bmatrix} = R_{r_{rect}} x' \tag{4.116}$$

Where $\breve{x}$ and $\breve{x}'$ are new rectified pixel coordinates and $\breve{v}$, $\breve{v}'$ are equal to each other. For this reason, it is enough to search pixel correspondence only in rows of the images [47].

After images are rectified, a disparity map can easily be computed by using brute force matching. This computation can be performed via block-matching algorithm using the sum of absolute differences (SAD), mean absolute difference (MAD), or mean squared error (MSE) as a cost function. Block matching uses two small kernels sized $S \times S$. One is for the reference image, and the second one is for searching for the best fit in

the other image. The kernels are centered at pixels to be compared. The sum of intensity differences of corresponding pixels in both kernels is computed and stored. The previous operation repeats until the kernel in the second image scans the entire search region. Stored sums of intensity differences are compared, and the one that satisfies the cost function best is chosen. The commonly used cost functions are as follows [48]:

$$SAD = \sum_{i=0}^{S-1}\sum_{j=0}^{S-1}\left|p_{l_{ij}} - p_{r_{ij}}\right| \tag{4.117}$$

$$MAD = \frac{1}{S^2}\sum_{i=0}^{S-1}\sum_{j=0}^{S-1}\left|p_{l_{ij}} - p_{r_{ij}}\right| \tag{4.118}$$

$$MSE = \frac{1}{S^2}\sum_{i=0}^{S-1}\sum_{j=0}^{S-1}\left(p_{l_{ij}} - p_{r_{ij}}\right)^2 \tag{4.119}$$

The kernel size is important, according to [48]. If kernel size is chosen too small, it causes a poor-quality disparity map. Since there is not enough information, it is susceptible to noise. On the other hand, if kernel size is too big, it causes blurred regions around the edges. The bigger the kernel area is, the more edges it can contain. Edges can be defined as rapid changes in pixel intensities. These rapid intensity changes alter the result and lead to wrong pixel matching, as the disparity is affected by all the pixels in the kernel. This phenomenon is called the fattening effect. There are different techniques to decrease the fattening effect, such as semi-global matching and multi-block matching, which use global cost function and multiple kernels, respectively. Comparing every pixel in one image to every pixel in another image is computationally intensive. There are different ways to minimize the search area and decrease the computation time. Rectangular, diamond, and hexagonal shape searching patterns have been developed [49]. The one having a better accuracy of them is diamond search. It consists of two diamond shapes; one is large, and the other one is small. Block cost function is computed for each point of diamond shape. The center of the diamond shape is shifted towards the point with minimum cost. This step is repeated until the center point has the minimum cost function. Then at the last step, block cost functions are computed in the form of a small diamond shape around the

found center point. The point with the minimum cost function is chosen as the final best match [50].

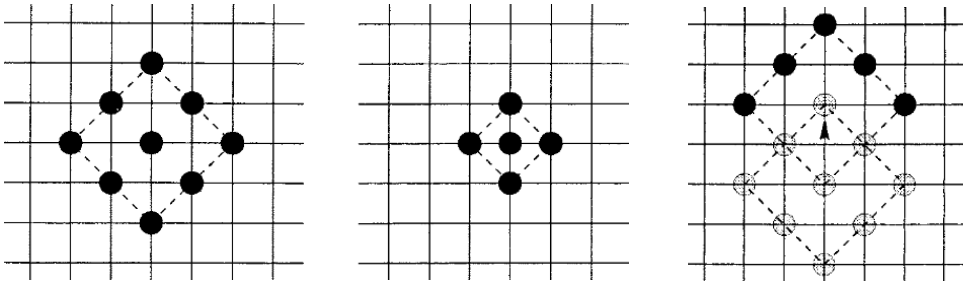


Figure 18 Diamond search [50]

CHAPTER V

EXPERIMENTS AND RESULTS

5.1 Hardware Setup

Our study investigates how theoretical information performs in the real world with the infrastructure we have created. Below, an unmanned ground vehicle used in this study was described. As a differential drive model was adopted, the vehicle has two wheels with a diameter of 80 mm, and a caster wheel of 12.7 mm diameter. Wheels were mounted on two brushed DC motors, and encoders were attached to each motor shaft to read the motor speed. L298 motor driver was used to control the speed of both motors via PWM. Raspberry Pi 4B, an ARM processor, was used to program the vehicle. Two 3A Li-Po power banks were used as 5V voltage sources, one for the processor and the other for motors. USB camera is mounted on the robot frame to observe the environment. Lastly, an LCD screen was connected for monitoring the output and debugging problems quickly.

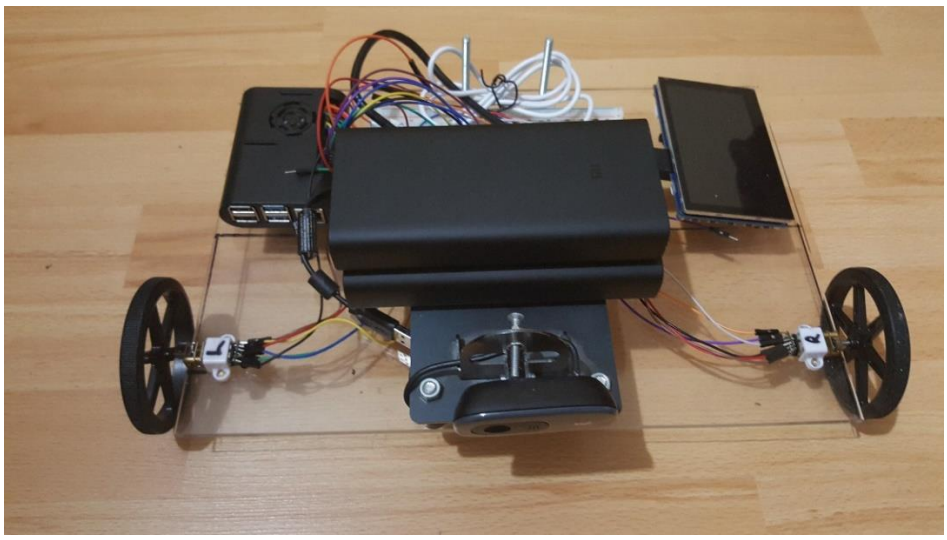


Figure 19 Design autonomous mobile robot

5.2 Software Setup

Raspberry Pi OS 64-bit version was installed as a main operating system to use all 8GB system memory. Pi OS was chosen because it is designed and optimized better for ARM processors and Pi hardware. Python 3 programming language and an open-source library called OpenCV (version 4.5.0) were used for image processing tasks.

5.3 Camera Calibration

In this section, a camera calibration procedure has been performed to find intrinsic and extrinsic parameters. When corner points are detected, it is assumed that the points are on a plane. However, in real life, no surface is smooth enough to be considered a plane and causes error in calibration. Besides, adhesives and tapes create air bubbles under the checkerboard paper, which increases error. For this reason, the checkerboard pattern was placed between the dark chipboard and window glass. The camera resolution was set to 640x480, and images were taken with the camera from nine different positions.

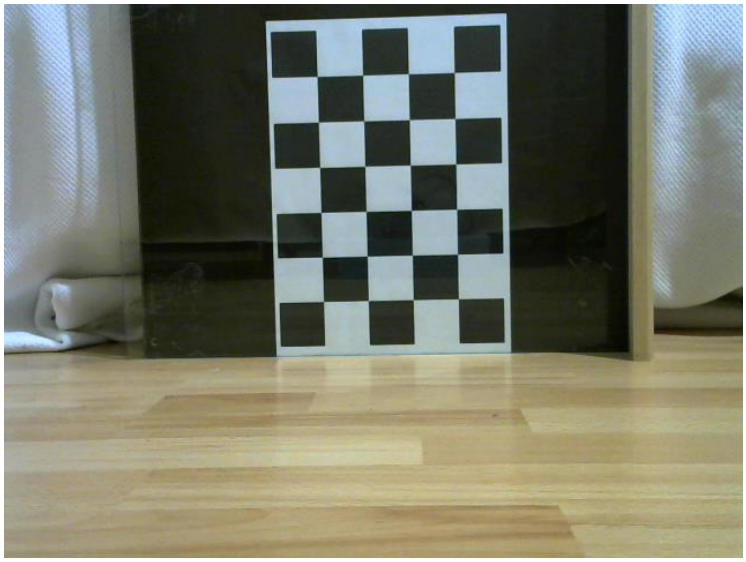


Figure 20 Checkerboard pattern setup

Harris corner detector explained in chapter 4 was used to find corner points. Harris detector gives better results than the FAST detector as it directly finds the intersection of edges. The corner points found from the images were used to estimate the camera

parameters with homography (Figure 21). As a result of this technique, the camera was calibrated with sub-pixel accuracy where the mean pixel re-projection error was less than 0.05.

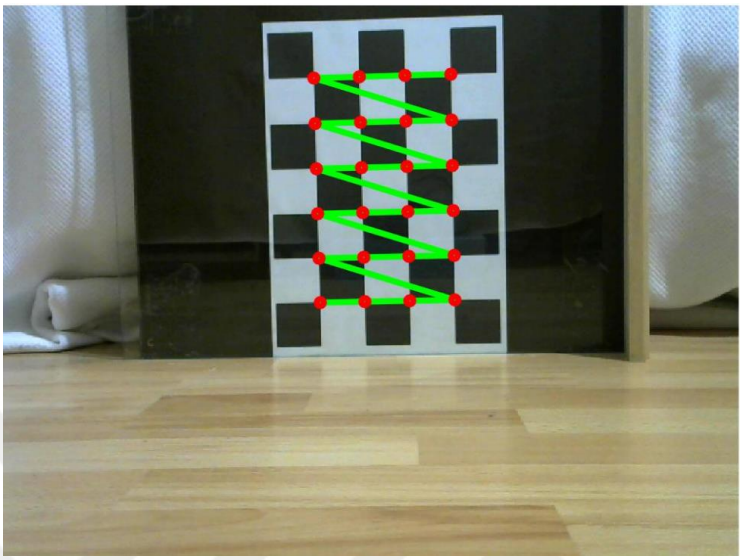


Figure 21 Detected corner points in camera calibration

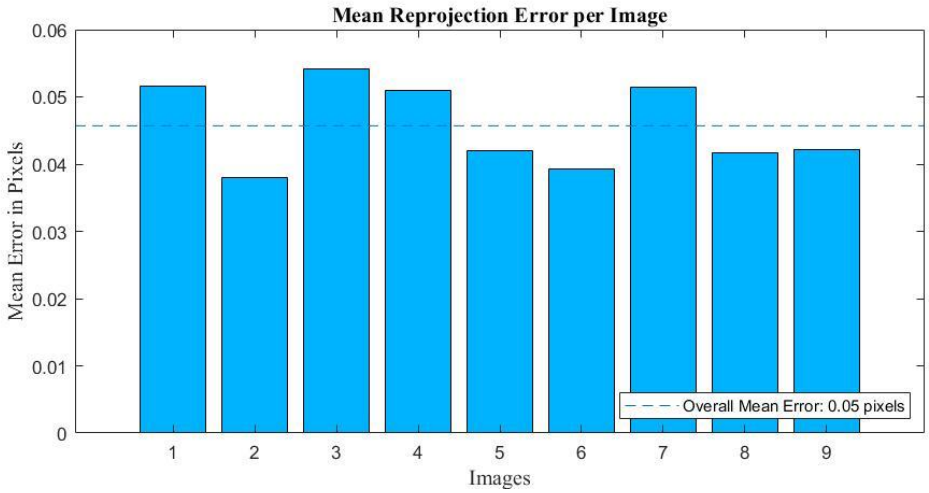


Figure 22 Calibration error

5.4 Bag of Words Construction

A small bag of words was generated using randomly taken images of the home instance to detect previously passed areas. A total of 2600 detected FAST corners were used over five images, and BRIEF descriptors were extracted around each detected corner. Using the K-medoids algorithm, BRIEF descriptors were classified under ten groups.

Then descriptors in each group were also classified under ten groups. A total of 100 groups were created. The reason for creating layers of groups is decreasing the comparison amount. Instead of comparing all the descriptors of detected corners with 100 words, only 20 comparisons will be made. Ten comparisons in the first layer and ten comparisons in the second layer. If there were a bag with a million words having six layers, instead of a million comparisons, only 60 comparisons would be made. Related code is given in Appendix B.

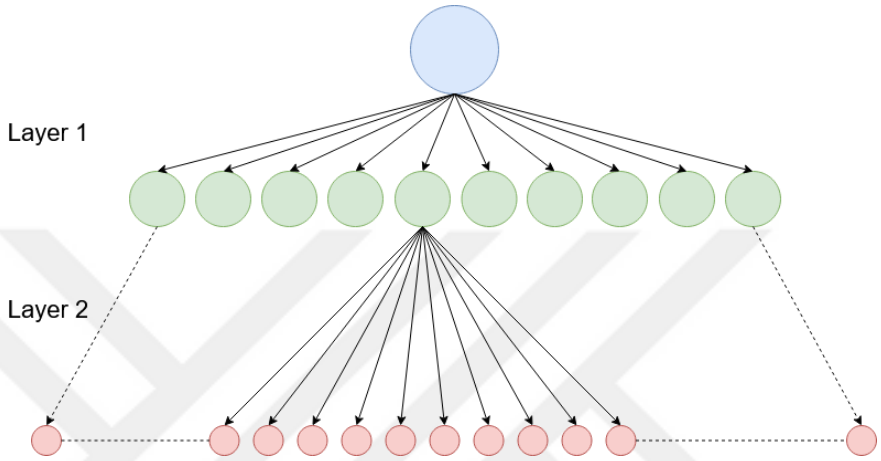


Figure 23 Layers created using k-medoid clustering

Figure 24 Images used to construct bag of words.

5.5 Experiment Overview

Due to Covid-19, experiments were conducted in a home instance with different furniture. FAST corner detection and BRIEF descriptor extraction are used for motion tracking. Orientation and translation of the robot were computed using epipolar geometry. 5-point essential matrix algorithm gave poor results; thus, the 8-point fundamental matrix algorithm was used in Eq. (4.91).

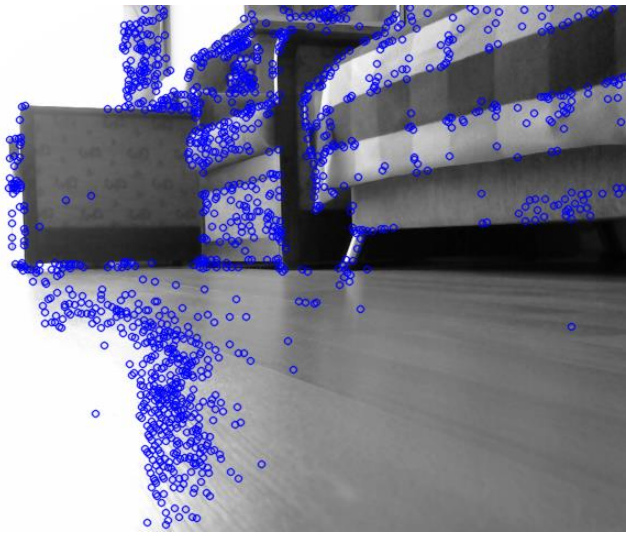


Figure 25 Detected FAST corners.

Epipolar constraint was set to 1, and matched points were cleaned from outliers using RANSAC. In other words, points having more than 1-pixel error were discarded. The essential matrix was computed from the resulting fundamental matrix using camera parameters determined in the camera calibration part.

$$E = K^T F K \tag{5.1}$$

Where E and F are essential matrix and fundamental matrix, respectively, while K is the camera matrix, the essential matrix was used to find the rotation and translation of the robot. Yet, translation is computed only up to a scale. For this reason, robot motion was assumed as a constant velocity model.

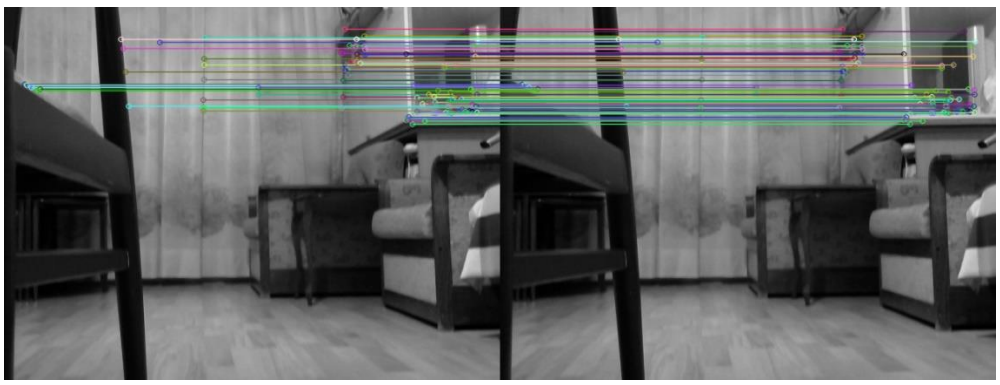


Figure 26 Matched inlier points between frames

2D pose-graph optimization model was used to close detected loops. It is based on the nonlinear least square explained in Chapter 3. Frames are represented as nodes, while observations are represented as edges in the graph. Detected loops are also new observations and create a dependency between nodes. Weight is associated with each edge showing how reliable the observation is. The system was solved by minimizing the following error function [23]:

$$e = R_{ij}(R_i(t_j - t_i) - t_{ij}) \tag{5.2}$$

t_i and t_j are robot poses at steps i and j. R_i is the heading angle of the robot at step i while R_{ij} and t_{ij} are measurements of rotation and translation between steps i and j, respectively. For weights, the inverse of the covariance matrix was used. If a measurement is predicted to be wrong, associated weight can be decreased further. Loops were detected between frames having a similarity of more than 85%. As explained in Chapter 3, the similarity score was computed using the constructed bag of words model.

5.6 Experiment Platform

The robot was programmed to move in a square to make it easier to measure the ground truth. First, both motor speeds were set to 20 RPM manually by using PWM clock cycles. The required time was computed to perform a 90-degree turn using Eq. (2.24) in Chapter 2. Ground velocities of the wheels were computed using the wheels' perimeter, which is 25.1327 cm.

Figure 27 Constructed robot environment

As shown in Figure 27, an environment was constructed by multiple boards attached with printed images to test the robot's performance. A board marker was placed on the robot to mark its route (Figure 28). It is impossible to detect all the measurements corresponding to points on the robot path. Therefore, the corners of the rectangular path were chosen as ground truth points, and their coordinates were measured manually.

Figure 28 Path drawn by the mobile robot

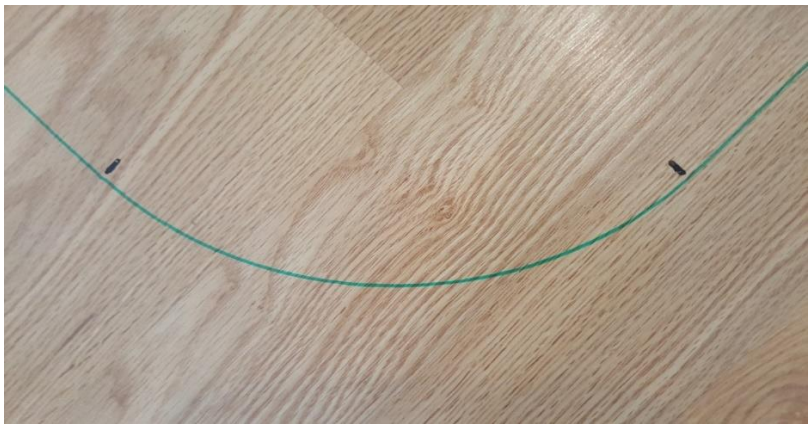


Figure 29 Corners used for the ground truth estimation

Figure 30 Measured ground truth in cm.

The camera data was taken every 0.1 seconds, but translation between camera frames was not enough to measure mobile robot movement, as robot speed was slow. For this reason, frames were processed every 1 second. As mentioned before, the translation in each step is computed up to a scale; therefore, robot movement was assumed as a constant velocity, and scale factors were set as one unit. Unfortunately, raspberry pi was not powerful enough to perform image processing in real-time, so images are processed offline. The computed robot path is given in Figure 31 and compared with another SLAM algorithm called SURF-SLAM in Figure 32 [51]

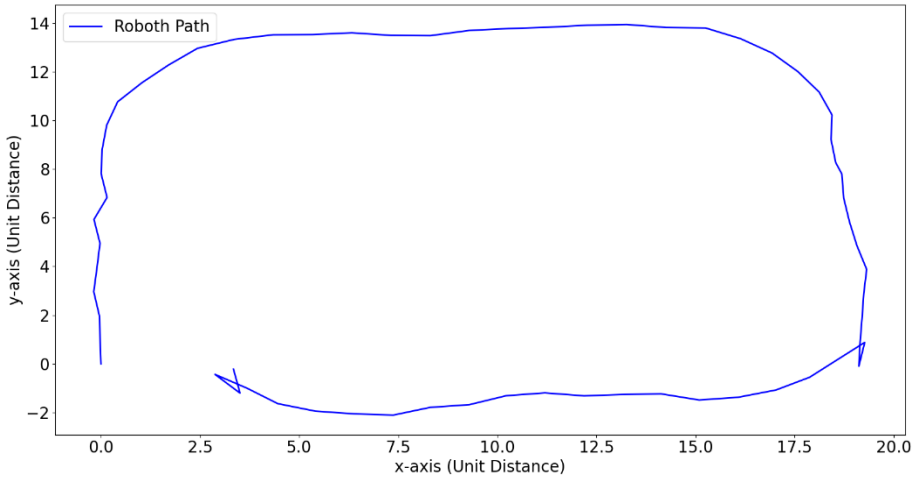


Figure 31 Computed robot path by monocular visual odometry

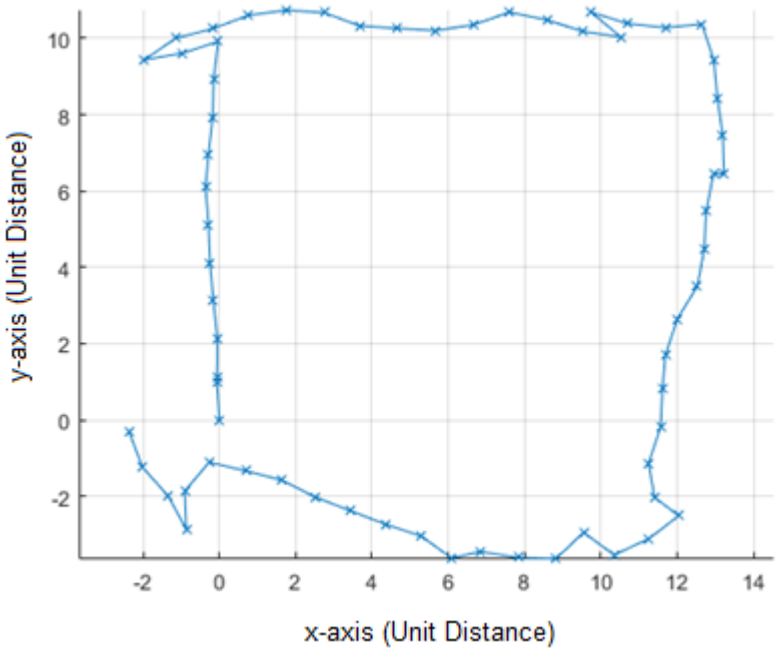


Figure 32 Mobile robot path computed by SURF-SLAM

As seen from Figures 31 and 32, mobile robot paths are similar and rectangular due to scale ambiguity. Nevertheless, computed translation vectors are not accurate due to wrong matches during feature matching. In addition, the constructed environment consists of planar surfaces, which causes degeneration. Degeneracy refers to the existence of multiple solutions that satisfy a system of equations. In epipolar geometry, this means that multiple fundamental matrices are satisfying the epipolar constraints [52]

The robot path was also computed by using encoder readings to solve the scale ambiguity. Firstly, the error model of the mobile robot was decided. As mentioned in Chapter 2, the differential vehicle model has special cases. Different equations are required to compute the mobile robot's movement while it is rotating and moving straight. To bypass this problem, the dead reckoning model was used as in Eq. (2.30). The mobile robot's rotation and velocity were computed using Eq. (2.24) and averaging left and right wheel velocities, respectively. Data from the encoder was also taken every 0.1 seconds to synchronize the encoder with the camera. Since raspberry pi was not powerful enough to read data from the camera and encoder simultaneously, related code was written by using multiprocessing, and camera outputs were taken in compressed MJPEG format instead of raw YUYV format. The robot path computed by encoder readings is given in Figure 33. The measured ground truth corners and the encoder readings corresponding to the corners were compared, and RMSE was calculated as 3.1433 cm by following equation where x and $\hat{x}$ represent ground truth and encoder corner values, respectively:

$$RMSE(\hat{x}) = \sqrt{\frac{\sum_{t=1}^{t=n}(x_t - \hat{x}_t)^2}{n}} \tag{5.3}$$

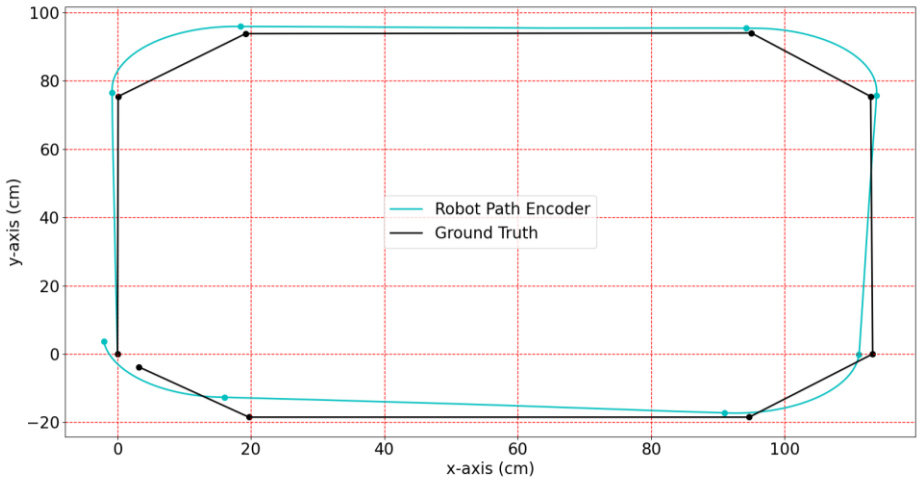


Figure 33 Computed robot path by the encoder.

Sensor noises are commonly assumed as Gaussian noise. Data taken from both encoders were tested to determine the noise model, and encoder noise was found in Gaussian distribution, as shown in Figure 34. The mean and standard deviation of both

encoders were almost equal and computed as 19.9752 RPM and 1.1440 RPM, respectively.

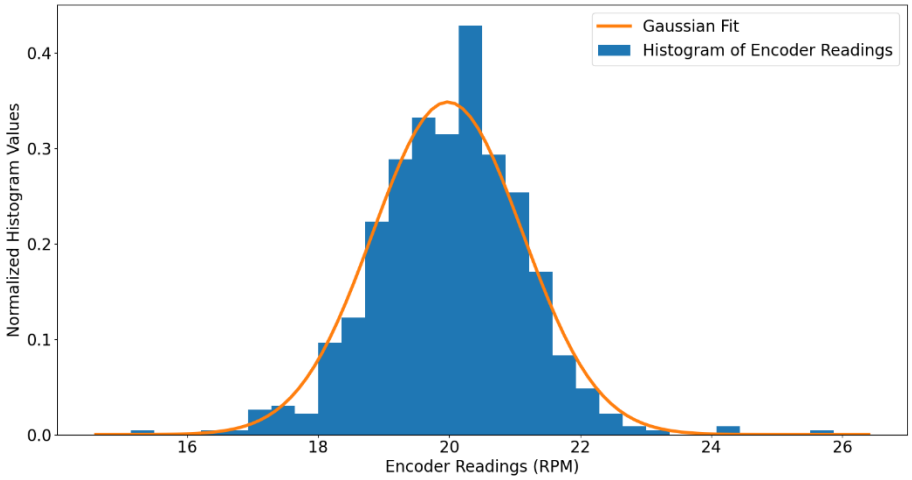


Figure 34 Encoder noise fit in Gaussian model

The Gaussian noise in both encoders causes the uncertainty in the mobile robot's pose to increase gradually as the mobile robot moves. This uncertainty can be found with the MMSE estimator, as explained in Chapter 3, Eq. (3.15). Since the dead reckoning model is nonlinear, the system must be linearized using Taylor series expansion. This expansion can be done by taking partial derivatives with respect to each state variable. In addition, inputs to the system are velocities in x, y-axis, and change in rotation, a 3D vector. However, the state vector is a 3D vector consisting of a mobile robot's pose in x, y-axis, and heading angle. In such a case, noise in the input is a multiplicative noise and can be propagated by taking partial derivatives with respect to each input variable. Gradually increasing uncertainty in robot pose can be computed as follows:

$$J_s = \begin{bmatrix} 1 & 0 & -V_x \sin(\theta) - V_y \cos(\theta) \\ 0 & 1 & V_x \cos(\theta) - V_y \sin(\theta) \\ 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

$$J_i = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

$$P_t = J_s P_{t-1} J_s^T + J_i Q J_i^T \tag{5.6}$$

Where J_s and J_i are the Jacobian matrices of state and input, respectively while P_t and Q are the uncertainties or covariance matrices of the mobile robot pose at step t and encoder, respectively. The uncertainty in the robot pose can be visualized by covariance ellipses computed from the covariance matrix P_t. The size and orientation of the ellipses are related to Eigenvalues and Eigenvectors of covariance matrix P respectively. The covariance ellipses can be drawn by the following equation [19]:

$$x = V\sqrt{D}y \tag{5.7}$$

Where V and D are Eigenvector and Eigenvalues of covariance matrix P respectively while x and y are coordinates of points on ellipse and unit circle respectively, drawing the covariance ellipses is important to understand how error propagates during localization. Ellipses represent the errors in the system proportionally. The covariance ellipses of the mobile robot's pose are shown in Figure 35.

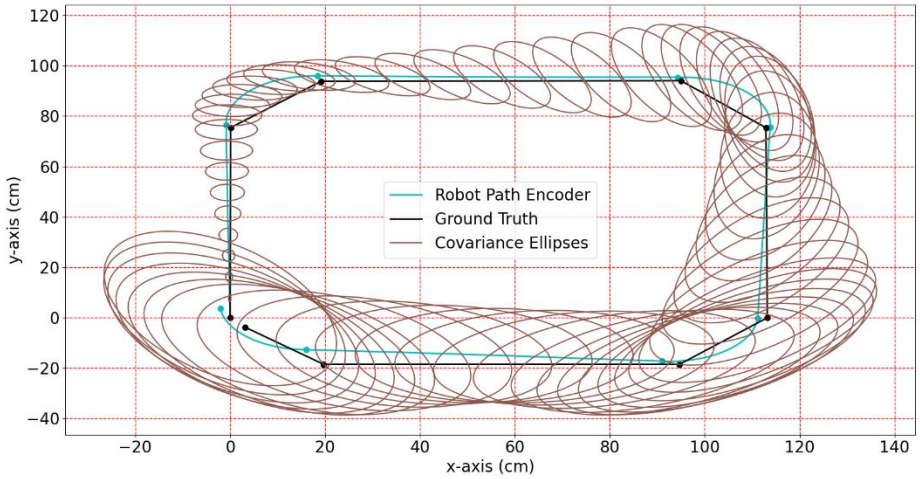


Figure 35 The size of covariance ellipses as mobile robot moves.

The translation vectors computed from corresponding fundamental matrices are normalized so that their Euclidean norm is one. To find the scale factors, translation vectors were multiplied by the corresponding velocity values computed from encoder readings. The resulting mobile robot path is given in Figure 36. As seen in Figure 36, due to noise and degeneracy, projected point locations are also not accurate. In addition, the camera field of view is only 60 degrees and can only detect points in front of it.

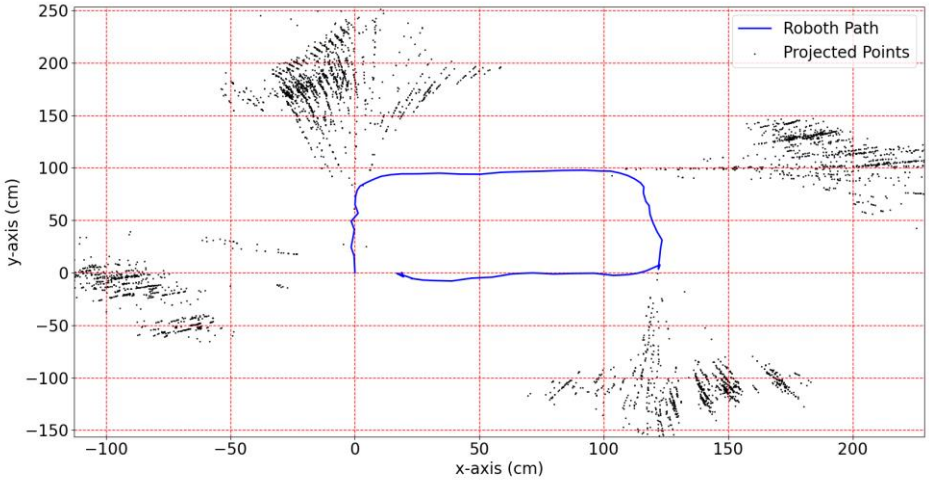


Figure 36 Mobile robot path and detected points in the corrected scale.

The mobile robot's position drifts as it moves due to noise in measurements. To decrease the drift, the loop closure technique was used in the form of a 2D graph-SLAM. Loop was closed by using the error function in Eq. (5.2). To minimize the error function, GNA was performed. To linearize the non-linear error function, partial derivatives were taken with respect to the mobile robot's states. Jacobian matrices were computed as follows [23]:

$$A_{ij} = \begin{bmatrix} -R_{ij}^T R_i^T & R_{ij}^T \dfrac{\partial R_i^T}{\partial \theta_i}(t_j - t_i) \\ 0^T & -1 \end{bmatrix} \tag{5.8}$$

$$B_{ij} = \begin{bmatrix} R_{ij}^T R_i^T & 0 \\ 0^T & 1 \end{bmatrix} \tag{5.9}$$

$$J_{ij} = \begin{pmatrix} 0 \dots 0 & \underbrace{A_{ij}}_{step\ i} & 0 \dots 0 & \underbrace{B_{ij}}_{step\ j} & 0 \dots 0 \end{pmatrix} \tag{5.10}$$

Errors in the mobile robot's positions were minimized using Eq. (3.41) and a weighting factor. The inverse of covariance matrix in Eq. (5.6) was used as the weighting factor. For camera covariance matrix, an identity matrix was used since camera noise model is unknown. The final form of the equation is as follows:

$$\Delta x = \sum_{i,j} \left(J_{ij}^T P_i^{-1} J_{ij}\right)^{-1} J_{ij}^T P_i^{-1} e_{ij} \tag{5.11}$$

Mobile robot path before and after loop closing is given in Figure 37. Before and after loop closing, the RMSEs of mobile robot path were computed 10.3323 cm and 4.8011 cm, respectively.

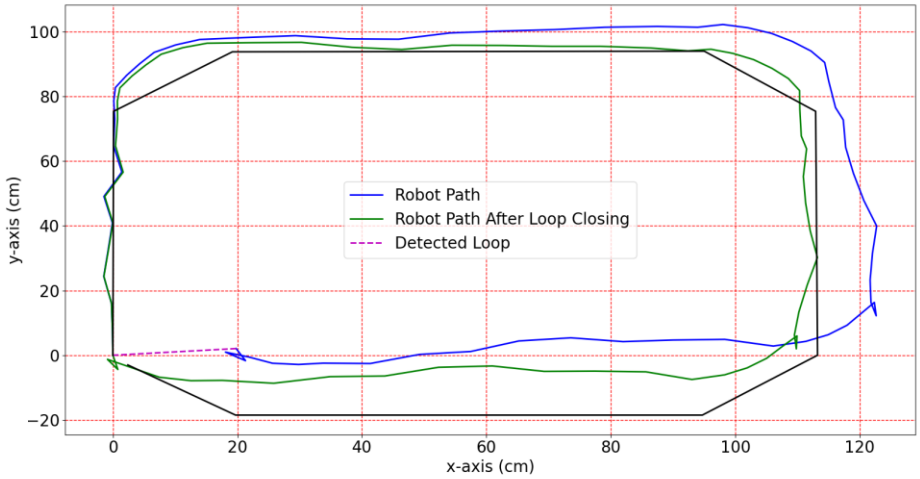


Figure 37 Robot path before and after loop closing

CHAPTER VI

CONCLUSION

This thesis aims to design and implement a VSLAM algorithm for autonomous navigation of a mobile robot. According to the theoretical information given in thesis, a VSLAM algorithm was designed and implemented. The algorithm was tested in real-world conditions. To perform the test, an autonomous vehicle was constructed. The robot was made to follow a specific path with a control input, and the environment was mapped.

Differential drive model was chosen for the mobile robot model. Information from the environment was taken from a monocular camera. Camera was calibrated using checkerboard pattern with nine images taken from different locations and orientations. Bag of words model was chosen to detect similar places since it is fast and easy to train. Information from the images was extracted using FAST corner and BRIEF descriptors algorithms. Since BRIEF descriptors are binary data, they were classified under 100 groups using K-medoid (PAM) algorithm. By the nature of the algorithm, due to local searches algorithm could not converge to global minimum. For this reason, algorithm was run several times and best solution was chosen. Images having similarity score higher than 85% were chosen as loop closure candidates.

Rotation and translation between consecutive images were computed using epipolar geometry. 8-point algorithm was chosen to solve epipolar constraints. Wrong matches (outliers) were removed using RANSAC method. By nature of epipolar geometry, there are degeneracy cases. RANSAC method cannot eliminate outliers in degeneracy cases. The poor translation computed from visual odometry in the experiments were caused by the planar surfaces used in the construction of the platform. The implemented algorithm was compared with another VSLAM algorithm called SURF-SLAM. Translation result was also similar to the result of thesis experiment due to

degeneracy. Detected loops were closed by 2D pose graph optimization. Gauss-Newton algorithm was chosen since it converged faster. Mobile robot path before and after loop closing were compared with the ground truth. The accuracy of the loop closure was detected to be higher than visual odometry.

As understood from the experiments, it is important to detect similar places. Because of sensor noise, the uncertainty of robot pose gradually increases without bound. As the map gets bigger, the accumulated error also gets bigger and resulting map becomes very different from the actual map. To reduce the drift and construct a consistent map, loop closure techniques are very important. The loop closure performance is directly dependent to loop detection algorithm. If loops cannot be detected or mismatched, the map quality will be poor. In other words, without a proper loop detection algorithm, a VSLAM algorithm cannot work efficiently. On the other hand, the accuracy of consecutive measurements are also very important. The detection of degeneracy cases in epipolar geometry and using different techniques in such cases can greatly increase the accuracy of the resulting map.

REFERENCES

[1] Thrun, S., Burgard, W., Fox, D. (2005), Probabilistic Robotics, MIT Press.

[2] Milz, S., Arbeiter, G., Witt, C., Abdallah, B., Yogamani, S., (2018), Visual SLAM for Automated Driving: Exploring the Application of Deep Learning, Conference on Computer Vision and Pattern Recognition Workshops, pp. 360-370.

[3] Hidalgo, F., Bräunl, T., (2020), Evaluation of Several Feature Detectors/Extractors on Underwater Images towards vSLAM. Sensors 2020, 20, 4343; doi 10.3390/s20154343.

[4] Xia,. Y., Li, J., Qi, L., Yu, H., Dong, J., (2017), An Evaluation of Deep Learning in Loop Closure Detection for Visual SLAM, IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyb.

[5] Chen, L., Jin, S., Xia, Z., (2021), Towards a Robust Visual Place Recognition in Large-Scale vSLAM Scenarios Based on a Deep dİSTANCE lEARNİNG. sENSORS 2021, 21, 310. dOİ 10.1109/ıcca.2019.8899937.

[6] Tao, C., Gao, Z., Yan, J., Li, C., Cui, G., (2020), Indoor 3D Semantic Robot VSLAM Based on Mask Regional Convolutional Neural Network. IEEE Access, Volume 8, Page(s): 52906- 52916 Doi 10.1109/ACCESS.2020.2981648.

[7] Eigen, D., Puhrsch, C., Fergus, R., (2014), "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network", Conference on Neural Information Processing Systems.

[8] Woodman, O. J., (2007), An introduction to inertial navigation. University of Cambridge, Computer Laboratory, Tech. Rep. UCAMCL-TR-696, 14, 15.

[9] Hu, J. S., Chen, M. Y., (2014), A sliding-window visual-IMU odometer based on tri-focal tensor geometry, September, Proceedings - IEEE International Conference on Robotics and Automation DOI: 10.1109/ICRA.2014.6907434.

[10] Griffiths, A., Dikarev, A., Green, P. R., Lennox, B., Poteau, X., Watson, S., (2016), "AVEXIS—Aqua Vehicle Explorer for In-Situ Sensing," IEEE Robot. Autom. Lett., vol. 1, no. 1.

[11] Bogue, R., (2011), "Robots in the nuclear industry: a review of technologies and applications," Ind. Robot An Int. J., vol. 38, no. 2, pp. 113–118, Mar. 2011.

[12] Saha, S. K., Angeles, J., (1989), "Kinematics and Dynamics of a Three-wheeled 2-DOF AGV," Proceedings of IEEE International Conference on Robotics and Automation, Scottsdale, AZ, pp. 1572-1577.

[13] D'Andrea-Novel, B., Bastin, B., Campion, G., (1991), "Modeling and Control of Nonholonomic Wheeled Mobile Robots," Proceedings of IEEE Confrence on Robotics and Automation, Sacramento, CA, pp.1130-1135.

[14] Wada, M., Mori, S., (1996), "Holonomic and Omnidirectional Vehicle with Conventional Tires," Proceedings of IEEE International Conference on Robotics and Automation, Vol. 1, pp. 265-270.

[15] Yi, B. J., Kin, W. K., (2002), "The Kinematics for Redundantly Actuated Omnidirectional Mobile Robots," Journal of Robotic Systems, Vol. 19, No. 6, pp. 255-267.

[16] Muir, P. F., Numan, C. P., (1987), "Kinematc Modeling of Wheeled Mobile Robots," Journal of Robotic System. Vol. 4, No 2, pp. 281-329.

[17] Holmberg, R., (2000), August, "Design and Development of Powered-caster Holonomic Mobile Robots," PhD. Dissertation, Stanford University.

[18] Cheng, R. M. H., Rajagopalan, R., (1992), "Kinematics of Automated Guided Vehicles with an Inclined Steering Column and an Offset Distance-Criterion for Existence of Inverse kinematics colution," Journal of Robotics System., 9(8), 1059-1081.

[19] Newman, P., (2006), "EKF Based Navigation and SLAM", Oxford Press, Summer School.

[20] Klančar, G., Zdešar, A., Blažič, S., Škrjanc, I., (2017), Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems, Butterworth-Heinemann.

[21] Hartani, K., Miloud, Y., Miloudi, A., (2010), "Electric Vehicle stability with rear Electronic differential Traction," Int. Symp. Enviroment Friendly Energies Electrcal Appllcations, no. November, pp. 3, 2010.

[22] Dudek, G., Jenkin, M., (2010), Computational Principles of Mobile Robotics, 2nd ed., Cambridge University Press.

[23] Giorgio G., Rainer, K., Cyrill, S., Wolfram, B., (2010), A tutorial on graph-based SLAM. IEEE Transactions on Intelligent Transportation Systems Magazine. 2. 31-43. 10.1109/MITS.2010.939925.

[24] Fischler, M. A., Bolles, R. C., (1981), Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24, 381–395.

[25] Wang, H., Li, J., Ran, M., Xie, L., (2019), Fast Loop Closure Detection via Binary Content. 2019 IEEE 15th International Conference on Control and Automation (ICCA), Edinburgh, United Kingdom, 2019, pp. 1563-1568, Doi: 10.1109/ICCA.2019.8899937.

[26] Zhang, W., Yan, Z., Wang, Q., Wu, X., Zuo, W., (2020), Learning second-order statistics for place recognition based on robust covariance estimation of CNN features. Neurocomputing 398 (2020) 197–208, doi.org/10.1016/j.neucom.2020.02.001.

[27] Sivic, j., Zisserman, A., (2009), "Efficient Visual Search of Videos Cast as Text Retrieval," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 4, pp. 591-606, April 2009, doi: 10.1109/TPAMI.2008.111.

[28] Zhang, G., Yan, X., Ye, Y., (2019), Loop Closure Detection via Maximization of Mutual Information. IEEE Access, Volume 7, Page(s):124217-124232 Doi 10.1109 / ACCESS.2019. 2937967.

[29] Schubert E., Rousseeuw P.J., (2019), Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. In: Amato G., Gennaro C., Oria V., Radovanović M. (eds) Similarity Search and Applications. SISAP 2019. Lecture Notes in Computer Science.

[30] Yilmaz, A., Javed, O., Shah, M., (2006), “Object tracking: A survey,” ACM Computing Surveys (CSUR), vol. 38, no. 4, 2006.

[31] Paschos, G., (2001), Perceptually uniform color spaces for color texture analysis: an empirical evaluation. IEEE Trans. Image Process. 10, 932–937.

[32] Song, K. Y., Kittler, J., Petrou, M., (1996), Defect detection in random color textures. Israel Verj. Cap. J. 14, 9, 667–683.

[33] Zhao, Y., Vela, P. V., (2020), Good Feature Matching: Toward Accurate, Robust VO/VSLAM With Low Latency. IEEE Transactions on Robotics, Vol. 36, No. 3, Jun, Page(s): 657- 675 Doi 10.1109/TRO.2020.2964138.

[34] Harris, C., Stephens, M., (1988), "A Combined Corner and Edge Detector". Alvey Vision Conference.

[35] Hörmander, L., (1990), The analysis of partial differential operators, volume 1, Springer, Eqq. 1.1.7 and 1.1.7′.

[36] Prewitt, J. M. S., (1970). "Object Enhancement and Extraction". Picture processing and Psychopictorics. Academic Press.

[37] Sobel, I., Feldman, G., (1973), “A 3x3 Isotropic Gradient Operator for Image Processing,” Pattern Classification and Scene Analysis, 1973, pp. 271-272.

[38] Jianbo, S., Tomasi, C., (1994), Good features to track. In IEEE Conference on Computer Vision and Pattern Recognition, pages 593-600, 1994.

[39] Rosten, E., Porter, R., Drummond, T., (2010), Faster and better: a machine learning approach to corner detection. January 2010, IEEE Transactions on Software Engineering 32(1):105-19 DOI: 10.1109/TPAMI.2008.275.

[40] Calonder, M., Lepetit, V., Strecha, C., Fua, P., (2010), BRIEF: Binary Robust Independent Elementary Features In: Daniilidis K, Maragos P, Paragios N, editors. Proc. of the 11th European Conference on Computer Vision – Lecture Notes in Computer Science.

[41] Rosin, P.L., (1999), Measuring corner properties. Computer Vision and Image Understanding, 73(2):291 – 307, 2.

[42] Rublee, E., Rabaud, V., Konolige, K., Bradski, G., (2011), Orb: An efficient alternative to sift or surf. 2011 International conference on computer vision. IEEE; 2011. p. 2564–2571.

[43] Zhang, Z., (2000), A flexible new technique for camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. 22, 1330–1334.

[44] Mahammed, M.A., Melhum, A.I., F. A. Kochery, F. A., (2013). "Object Distance Measurement by Stereo VISION", International Journal of Science and Applied Information Technology, March, pp. 5-8.

[45] Hartley, R., Zisserman, A., (2004), Multiple View Geometry in Computer Vision, 2nd ed. Cambridge: Cambridge University Press.

[46] Longuet-Higgins, H. C., (1981), A computer algorithm for reconstructing a scene from two projections. Nature 293, 133–135.

[47] Trucco, E., Verri, A., (1998), Introductory Techniques for 3-D Computer Vision, Prentice Hall PTR, Upper Saddle River, NJ, USA, pp. 157-161.

[48] Einecke, N., Eggert, J., (2015), "A multi-block-matching approach for stereo," 2015 IEEE Intelligent Vehicles Symposium (IV), Seoul, pp. 585-592, doi: 10.1109/IVS.2015.7225748.

[49] Khawase, S. T., Khamble, S. D., Thakur, N. V., Patharkar A. S., (2017), "An Overview of Block Matching Algorithms for Motion Vector Estimation" RICE 2017, Proceedings of the Seconds International Conference on Research in Intelligent and Computing in Eng.

[50] Zhu, S., Ma, K. K., (2000), "A new diamond search algorithm for fast block-matching motion estimation," in IEEE Transactions on Image Processing, vol. 9, no. 2, pp. 287-290, Feb. 2000, doi: 10.1109/83.821744.

[51] Chavan S., Gafoor, N., Nash, A., Yu, M-Y., Zhang, X., (2018), Team 4: NAVARCH 568 Final Project Report: SURF-SLAM. https://github.com/audrow/matlab-orb-slam/blob/master/docs/report.pdf.

[52] Torr, P. H. S., Zisserman, A., Maybank, S. J., (1995), Robust detection of degenerate configurations for the fundamental matrix. In Proceedings of the IEEE International Conference on Computer Vision, Cambridge, MA, 20–23 June 1995; pp. 1037–1042.

APPENDIX A

Materials Used in the Experiment

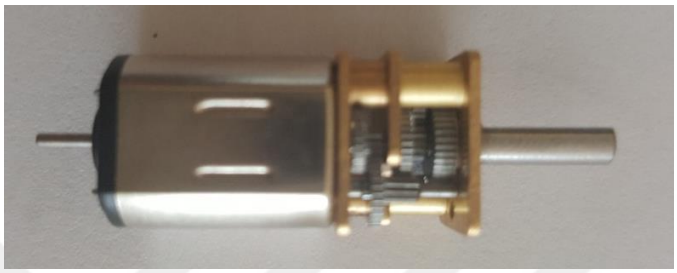


Figure 38 Brushed DC motor

Operating Voltage: 6v; Min. Current: 70mA; Max. Load Current: 1.6A; Stall Torque: 4kg/cm; Max. Speed: 100rpm; Weight: 9.5g; Gear Ratio: 298/1; In our project, two previously shown motors are used to drive and rotate the vehicle.

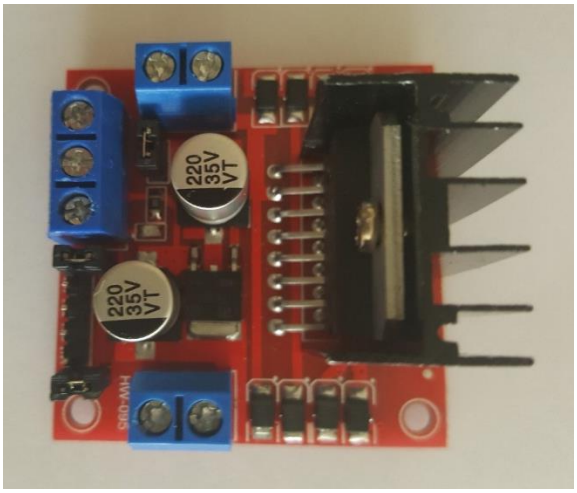


Figure 39 L298 motor driver.

Operating Voltage: 4.8-24V; Output Voltage: 5V; Motor Channels: 2; Current per Channel: 2A; Motor driver is used to controlling the speed of the motors and their rotation direction.

Figure 40 Wheels

Diameter: 80mm; Width: 10mm; Weight: 20g; Shaft Diameter: 3mm. wheels are surrounded by plastic material

Figure 41 Caster wheel

Wheel Diameter: 12.7mm; Total Width: 20.3mm; Weight: 9g

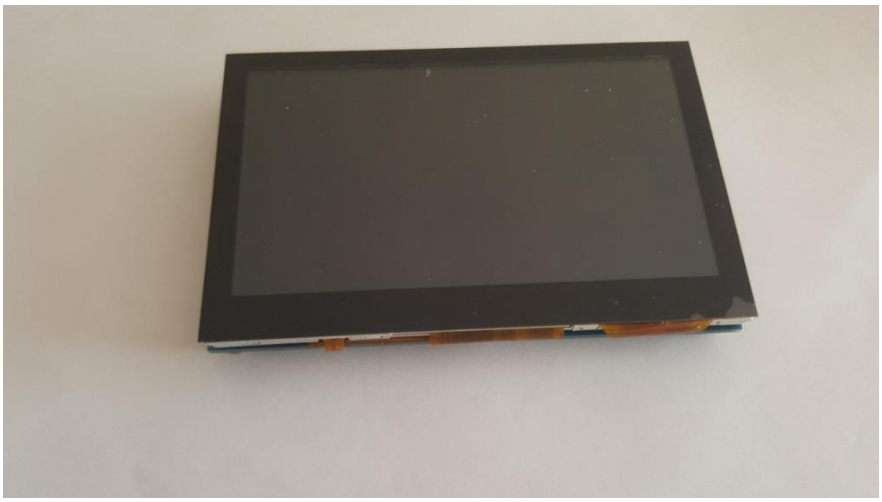


Figure 42 IPS LCD capacitive touch screen

Resolution: 800x480; Interface: HDMI

Figure 43 Raspberry Pi 4 8GB - model B

Raspberry Pi 4 is a small DIY computer with 28nm based 1.5G Quad-Core CPU and 8Gb DRR4 RAM. Raspberry Pi 4 features 4K Micro HDMI, USB 3.0, BLE Bluetooth 5.0, dual-band 2.4 / 5.0 GHz Wireless LAN to achieve PC-like capabilities USB-C power input, and True Gigabit Ethernet compatible with PoE. The main aspect of Raspberry Pi 4 is better performance; It has a 1.5 GHz Quad-Core ARM Cortex-A72 CPU with better VideoCoreVI Graphics. The Pi 4 can display a 4K 60fps HEVC video via a Micro-HDMI port and simultaneously connect to two 4K displays with only a 30 frame per second refresh rate. There are 1000Mbps True Gigabit Ethernet and four USB ports, but two are USB 3.0 ports with ten times the USB 2.0 transfer speed.

Additionally, the Raspberry Pi 4 uses a new 5V / 3A power supply and connects via a USB Type C port, which has better endurance and can work on both sides.

Figure 44 Xiaomi 20000 mAh Powerbank 3 Pro.

Voltage: 5.1 Volts; Max. Watt: 45 watts; Battery Capacity: 20000 Milliamp Hours; Number of Ports: 3; Product Dimensions: 15 x 7.35 x 3 cm; 400 g

Figure 45 4 mm 24 cm x 35 cm transparent plexi mica plexi.

It is a kind of plastic that is crystal clear, compatible with all environments, easily shaped with heat, and available in colored and colorless varieties. Plexi, which we can also call plastic glass, is a remarkable and useful material in aesthetics. Plexi, which is lighter than glass, is even more durable. Plexi, which is preferred because it adapts

easily with many materials, gives a sense of spaciousness and spaciousness due to its transparency.

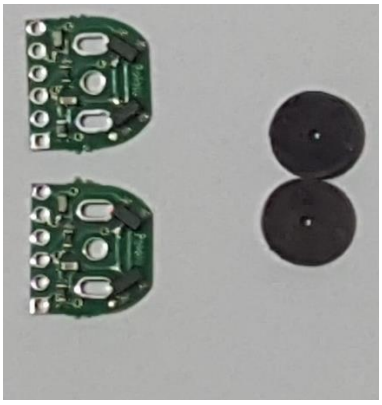


Figure 46 Magnetic Encoders

6-pole magnetic disks and TLE4946-2K hall effect sensors. Operating Voltage: 2.7-24V; Rise and Fall Time: 1µs; Switching Frequency: 15 kHz

Figure 47 Logitech C270 720p webcam.

Technical specifications: Max Resolution: 720p/30fps; Focus type: fixed focus; Lens technology: standard; Built-in mic: mono; FoV: 60°; Cable length: 1.5 m.

Figure 48 Soldering iron, solder wire, flux pasta solder, and jumper wire.

APPENDIX B

DC Motor Control

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
c=17
pin27=27
en=22
inr=23
inrr=24
inl=13
inll=19
en2=26
GPIO.setup(c, GPIO.IN)
GPIO.setup(pin27, GPIO.IN)

GPIO.setup(inr, GPIO.OUT)
GPIO.output(inr,GPIO.LOW)
GPIO.setup(inrr, GPIO.OUT)
GPIO.output(inrr,GPIO.LOW)
####
GPIO.setup(inl, GPIO.OUT)
GPIO.output(inl,GPIO.LOW)
GPIO.setup(inll, GPIO.OUT)
GPIO.output(inll,GPIO.LOW)

GPIO.setup(en, GPIO.OUT)
l=GPIO.PWM(en,1000)
l.start(100)
####
GPIO.setup(en2, GPIO.OUT)
l2=GPIO.PWM(en2,1000)
l2.start(100)

b=GPIO.input(c)
p=GPIO.input(pin27)
Time1=time.process_time()
count=0
count2=0
kp=0.85
kp2=kp
ki=0.01
ki2=ki
kd=0.11
kd2=kd
wrpm=30
wrpm2=30
wlc=100
wlc2=wlc
ep=0
ep2=0
```

```
inte=0
inte2=0
while 2>1:

 bb=GPIO.input(c)
 pp=GPIO.input(pin27)

 if(bb!=b):
  count=count+1
  ####
 if(pp!=p):
  count2=count2+1

 Time2=time.process_time()
 if((Time2-Time1)>0.1):
  e=wrpm-count/1788*60*10
  inte=inte+(e+ep)/2*0.1
  dtv=kp*e+kd*(e-ep)/0.1+ki*inte
  ep=e
  ####
  e2=wrpm2-count2/1788*60*10
  inte2=inte2+(e2+ep2)/2*0.1
  dtv2=kp2*e2+kd2*(e2-ep2)/0.1+ki2*inte2
  ep2=e2
  ####
  if(wlc+dtv)>100:
   dtv=dtv-(wlc+dtv-100)

  if(wlc+dtv)<10:
   dtv=dtv-(wlc+dtv-10)
   ####
  if(wlc2+dtv2)>100:
   dtv2=dtv2-(wlc2+dtv2-100)

  if(wlc2+dtv2)<10:
   dtv2=dtv2-(wlc2+dtv2-10)

  wlc=wlc+dtv
  l.start(wlc)
  ####
  wlc2=wlc2+dtv2
  l2.start(wlc2)


  print(count/1788*60*10)
  print(" ")
  print(count2/1788*60*10)
  count=0
  count2=0
  Time1=Time2
 b=bb
 p=pp
```

K-Medoid

```
import cv2
import numpy as np
from random import seed
from random import choice
#import cPickle
index=[]
```

```
lst=[]
lst2=[]
lst3=[]
m=[]
m2=[]
t=[]
t2=[]
mm=[]
mm2=[]
lyr2=[]
tgroup=[0,0,0,0,0,0,0,0,0,0,0]
group=[0,0,0,0,0,0,0,0,0,0]
group2=[0,0,0,0,0,0,0,0,0,0]
group3=[0,0,0,0,0]
im1=cv2.imread("opencv_frame_0.png")
im2=cv2.imread("opencv_frame_1.png")
im3=cv2.imread("opencv_frame_2.png")
im4=cv2.imread("opencv_frame_3.png")
im5=cv2.imread("opencv_frame_4.png")
gr1=cv2.cvtColor(im1,cv2.COLOR_BGR2GRAY)
gr2=cv2.cvtColor(im2,cv2.COLOR_BGR2GRAY)
gr3=cv2.cvtColor(im3,cv2.COLOR_BGR2GRAY)
gr4=cv2.cvtColor(im4,cv2.COLOR_BGR2GRAY)
gr5=cv2.cvtColor(im5,cv2.COLOR_BGR2GRAY)

fast = cv2.FastFeatureDetector_create()
brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
kp1=fast.detect(gr1)
kp2=fast.detect(gr2)
kp3=fast.detect(gr3)
kp4=fast.detect(gr4)
kp5=fast.detect(gr5)
cr1, des1=brief.compute(gr1,kp1)
cr2, des2=brief.compute(gr2,kp2)
cr3, des3=brief.compute(gr3,kp3)
cr4, des4=brief.compute(gr4,kp4)
cr5, des5=brief.compute(gr5,kp5)
for point in des1:
    #temp = (point.pt[0], point.pt[1], point.size, point.angle, poin
t.response, point.octave,
        #point.class_id)
    #index.append(temp)
    lst.append(point)
for point in des2:
    lst.append(point)
for point in des3:
    lst.append(point)
for point in des4:
    lst.append(point)
for point in des5:
    lst.append(point)
arr = np.array(lst)
tt=0
tt2=0
for x in range(10):
    #print(x)
    med=choice(arr)
    m.append(med)
n=np.array(m)
nn=n
while(True):

 for z in range(10):
  print(group)
  group=[0,0,0,0,0,0,0,0,0,0]
```

```
  for x in range(len(arr)):
   matches = bf.match(arr[x:x+1],n)
   group[matches[0].trainIdx]=group[matches[0].trainIdx]+1
   if matches[0].trainIdx==z:
    lst2.append(x)
  for x in range(len(lst2)):
   for y in range(len(lst2)):
     matc = bf.match(arr[lst2[x]:lst2[x]+1],arr[lst2[y]:lst2[y]+1])

     tt=tt+matc[0].distance
   t.append(tt)
   tt=0
  mm.append(arr[lst2[t.index(min(t))]])
  lst2.clear()
  t.clear()
  print(len(mm))
 nm=np.array(mm)
 if (n==nm).all():
  break
 else:
  n=nm
  mm.clear()

####
group=[0,0,0,0,0,0,0,0,0,0]
for y in range (10):
 for x in range(len(arr)):
    matches = bf.match(arr[x:x+1],n)
    if y==0:
     group[matches[0].trainIdx]=group[matches[0].trainIdx]+1
    if matches[0].trainIdx==y:
     lst2.append(x)
     tt=tt+matches[0].distance
for x in range(11):
 if x>0:
  tgroup[x]=tgroup[x-1]+group[x-1]
 else:
  tgroup[x]=0

#### 2.layer
for c in range(10):
 print(c)
 m2.clear()
 for x3 in range(10):
     #print(x)
     med=choice(arr[lst2[tgroup[c]:tgroup[c+1]]])
     m2.append(med)
 n2=np.array(m2)
 nn2=n2

 while(True):

  for z in range(10):
   print(group3)
   group3=[0,0,0,0,0,0,0,0,0,0]
   for x2 in range(tgroup[c],tgroup[c+1]):

    matches = bf.match(arr[lst2[x2]:lst2[x2]+1],n2)
    group3[matches[0].trainIdx]=group3[matches[0].trainIdx]+1
    if matches[0].trainIdx==z:
     lst3.append(x2)
   for x in range(len(lst3)):
    for y in range(len(lst3)):
      matc = bf.match(arr[lst2[lst3[x]]:lst2[lst3[x]]+1],arr[lst2[ls
t3[y]]:lst2[lst3[y]]+1])
     tt2=tt2+matc[0].distance
```

```
    t2.append(tt2)
    tt2=0
   #if len(t2)==0:
   if group3[z]==0:
    mm2.append(n2[z])
   else:
    mm2.append(arr[lst2[lst3[t2.index(min(t2))]]])


   lst3.clear()
   t2.clear()
   print(len(mm2))
  nm2=np.array(mm2)

  if (n2==nm2).all():
    mm2.clear()
    break
  else:
   print('iter')
   n2=nm2
   mm2.clear()

 lyr2.append(n2)
#### 2.layer
t.append(tt)
tt=0
print(t)
print(group)
summ=sum(t)
summ=np.array(summ)
summ=np.array_str(summ)
t=np.array(t)
t=np.array_str(t)
####
a_file = open("des5.txt", "w")
for row in n:
    np.savetxt(a_file, row)

a_file.close()
####
a_file = open("des55.txt", "w")
for x in range(10):
 for row in lyr2[x]:
    np.savetxt(a_file, row)
a_file.close()


####
f = open("distance5.txt", "w")
for point in t:
 f.write(point)
for point in summ:
 f.write(point)
f.close()
```

Bag of Word Histogram Computation

```
import cv2
import numpy as np
# Load Trained Medoids
medoid1= np.loadtxt("des5.txt").reshape(10, 32)
medoid1=medoid1.astype('uint8')
medoid2= np.loadtxt("des55.txt").reshape(100, 32)
```

```
medoid2=medoid2.astype('uint8')
# Initialize Algorithms
fast = cv2.FastFeatureDetector_create()
brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
hlist=[]
#### Image Proccessing Loop
i=1
while(i<50):
# Read Image and Convert to Gray

 im1=cv2.imread("opencv_frame_left%s.png" % i)
 gr1=cv2.cvtColor(im1,cv2.COLOR_BGR2GRAY)
 i=i+1
# Detect Corners and Extract Descriptors

 kp1=fast.detect(gr1)
 cr1, des1=brief.compute(gr1,kp1)
 ucr1=cv2.undistortPoints(cr1[0].pt, K, distcf,None,K)
 ucr1=np.append(ucr1,1)
# Create Bag of Words Histogram of Each Image

 hist=np.zeros(100)
 for x in range(len(des1)):
  matches = bf.match(des1[x:x+1],medoid1)
  tindx1=matches[0].trainIdx
  matches = bf.match(des1[x:x+1],medoid2[tindx1*10:tindx1*10+10])
  tindx2=matches[0].trainIdx
  hist[tindx1*10+tindx2]=hist[tindx1*10+tindx2]+1
 hist=hist/len(des1)
 hlist.append(hist)
# Computing Similarity Example
dsim=sum(np.abs(hlist[0]-hlist[1]))/2
sim=1-dsim
```

APPENDIX C

Tracking and Mapping

```
from copy import deepcopy
import cv2
import numpy as np
import transforms3d.euler as eul
import math
from copy import deepcopy
import matplotlib.pyplot as plt
##########
K1=np.array([816.2478,0,301.3167,0,809.7487,242.6526,0,0,1]).reshape
(3,3)
Ki1=np.linalg.inv(K1)
distcf1=np.array([0.016,0.4792,0,0])
fast = cv2.FastFeatureDetector_create(threshold=10)
#fast.setNonmaxSuppression(0)
brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
p1=[]
p2=[]
d1=[]
d2=[]
pnt1=[]
pnt2=[]
rot=[]
trn=[]
##########
ic=0
while(ic<48):
    p1=[]
    p2=[]
    pnt1=[]
    pnt2=[]
    d1=[]
    d2=[]
    ic=ic+1
    imgn = "opencv_frame_left{}.png".format(ic)
    imgn2 = "opencv_frame_left{}.png".format(ic+1)
    frame1=cv2.imread(imgn)
    frame2=cv2.imread(imgn2)
    gr1=cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
    gr2=cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)
    kp1=fast.detect(gr1)
    kp2=fast.detect(gr2)
    cr1, des1=brief.compute(gr1,kp1)
    cr2, des2=brief.compute(gr2,kp2)
    matches = bf.match(des1,des2)
    matches2=[]
    for x in range(len(matches)):
     if matches[x].distance<40:
      matches2.append(matches[x])
    lm=len(matches2)
    for x in range(lm):
```

```
51.      p1=np.append(p1,cr1[matches2[x].queryIdx].pt)
52.      p2=np.append(p2,cr2[matches2[x].trainIdx].pt)
53.     ####
54.     p1=p1.reshape(lm,2)
55.     p2=p2.reshape(lm,2)
56.     p1=cv2.undistortPoints(p1, K1, distcf1,None,K1)
57.     p2=cv2.undistortPoints(p2, K1, distcf1,None,K1)
58.     F, mask=cv2.findFundamentalMat(p1,p2,cv2.FM_RANSAC,0.5,0.999)
59.     count=0
60.     for x in range(lm) :
61.      if mask[x]!=0:
62.      #if p2[x]*F*p1[x].transpose()<50:
63.       #if p2[x]*F*p1[x].transpose()>0:
64.         pnt1=np.append(pnt1,p1[x])
65.         pnt2=np.append(pnt2,p2[x])
66.         d1=np.append(d1,des1[matches2[x].queryIdx])
67.         d2=np.append(d2,des2[matches2[x].trainIdx])
68.         count=count+1
69.     #######
70.     pnt1=pnt1.reshape(count,2)
71.     pnt2=pnt2.reshape(count,2)
72.     d1=d1.reshape(count,32).astype('uint8')
73.     d2=d2.reshape(count,32).astype('uint8')
74.     pp1=[]
75.     pp2=[]
76.     mlist=[]
77.
78.     for x in range(len(pnt1)):
79.      pp1.append(cv2.KeyPoint(pnt1[x,0],pnt1[x,1],float(7.0)))
80.      mlist.append(cv2.DMatch(x,x,0))
81.     for x in range(len(pnt2)):
82.      pp2.append(cv2.KeyPoint(pnt2[x,0],pnt2[x,1],float(7.0)))
83.
84.     img3 = cv2.drawMatches(frame1,pp1,frame2,pp2,mlist,None,flags=cv
    2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
85.     cv2.imshow('frame',img3)
86.     cv2.waitKey(1)
87.     E=np.matmul(K1.transpose(),F)
88.     E=np.matmul(E,K1)
89.     pointst,R,t,mask2t=cv2.recoverPose(E,pnt1,pnt2,K1)
90.     angt = eul.mat2euler(R, axes='sxyz')
91.     angtt=np.dot(angt,57.2957795)
92.     angle=angtt[1]
93.     rot.append(angle)
94.     trn.append(t)
95. ############################
96. Rot2=[]
97. Rot2.append(0)
98. for x in range(len(rot)):
99.  o=sum(rot[0:x+1])/57.2957795
100.  Rot2.append(o)
101.
102. Pos2=[]
103. v=np.matrix([0,0.15]).transpose()
104. p=np.matrix([0,0]).transpose()
105. Pos2.append(p)
106. for x in range(len(Rot2)-1):
107.  t=Rot2[x]
108.  rotz=np.matrix([[math.cos(t),-
    math.sin(t)],[math.sin(t),math.cos(t)]])
109.  tr=-np.matrix([trn[x][0,0],trn[x][2,0]])
110.  p=p+rotz*tr.transpose()
111.  Pos2.append(p)
112. Posx=[]
113. Posy=[]
114. for x in range(len(Pos2)):
```

```
 Posx.append(float(Pos2[x][0]))
 Posy.append(float(Pos2[x][1]))
plt.plot(Posx,Posy)
############################
s=(3,3)
ww=np.eye(3)
ww[2,2]=1
ss=np.matrix(np.zeros(s))
hi=[]
hi.append([1,45])
ic=0
Pos=deepcopy(Pos2)
Rot=deepcopy(Rot2)
while ic<15:
    el=[]
    j=[]
    jj=[]
    hh=[]
    bb=[]
    countt=len(Pos)
    hcount=0
    for i in range(countt-1+len(hi)):
     if i<countt-1:
        t1=Rot[i]
        t2=rot[i]/57.2957795
        e1=Rot[i+1]-Rot[i]-t2

        #########
        tr=-np.matrix([trn[i][0,0],trn[i][2,0]]).transpose()
        Ri=np.matrix([[math.cos(t1),-
math.sin(t1)],[math.sin(t1),math.cos(t1)]])
        Rij=np.matrix([[math.cos(t2),-
math.sin(t2)],[math.sin(t2),math.cos(t2)]])
        e0=Rij.transpose()*(Ri.transpose()*(Pos[i+1]-Pos[i])-tr)
        e=np.vstack([e0,e1])
        A0=-Rij.transpose()*Ri.transpose()
        A1=np.vstack([A0,np.matrix([0,0])])
        Rid=np.matrix([[-math.sin(t1),-math.cos(t1)],[math.cos(t1),-
math.sin(t1)]])
        A2=Rij.transpose()*Rid.transpose()*(Pos[i+1]-Pos[i])
        AA=np.vstack([A2,-1])
        Aij=np.hstack([A1,AA])
        B0=Rij.transpose()*Ri.transpose()
        B1=np.vstack([B0,np.matrix([0,0])])
        Bij=np.hstack([B1,np.matrix([0,0,1]).transpose()])
        el=np.append(el,e)
        for y in range(countt):

          if y==i:
           j.append(Aij)
          elif y==i+1:
           j.append(Bij)
          else:
           j.append(ss)
        j1=np.concatenate(j,axis=1)
        jj=np.append(jj,j1)
        j=[]
     else:
        c1=hi[hcount][0]
        c2=hi[hcount][1]
        t1=Rot[c1]
        t2=0
        e1=Rot[c2]-Rot[c1]-t2

        #########
        tr=-np.matrix([0,0]).transpose()
```

```
        Ri=np.matrix([[math.cos(t1),-
    math.sin(t1)],[math.sin(t1),math.cos(t1)]])
        Rij=np.matrix([[math.cos(t2),-
    math.sin(t2)],[math.sin(t2),math.cos(t2)]])
        e0=Rij.transpose()*(Ri.transpose()*(Pos[c2]-Pos[c1])-tr)
        e=np.vstack([e0,e1])
        A0=-Rij.transpose()*Ri.transpose()
        A1=np.vstack([A0,np.matrix([0,0])])
        Rid=np.matrix([[-math.sin(t1),-math.cos(t1)],[math.cos(t1),-
    math.sin(t1)]])
        A2=Rij.transpose()*Rid.transpose()*(Pos[c2]-Pos[c1])
        AA=np.vstack([A2,-1])
        Aij=np.hstack([A1,AA])
        B0=Rij.transpose()*Ri.transpose()
        B1=np.vstack([B0,np.matrix([0,0])])
        Bij=np.hstack([B1,np.matrix([0,0,1]).transpose()])
        el=np.append(el,e)
        for y in range(countt):

          if y==c1:
           j.append(Aij)
          elif y==c2:
           j.append(Bij)
          else:
           j.append(ss)

        j1=np.concatenate(j,axis=1)
        jj=np.append(jj,j1)
        j=[]
    jj=jj.reshape(countt-1+len(hi),3,(countt)*3)
    el=el.reshape(countt-1+len(hi),3)
    weighti=50
    for x in range(countt-1+len(hi)):
         if x<48:
          we=ww*(weighti)
         else:
          we=ww*0.5
         h1=np.matmul(jj[x].transpose(),we)
         h2=np.matmul(h1,jj[x])
         b1=np.matmul(jj[x].transpose(),we)
         b2=np.matmul(b1,el[x])
         hh=np.append(hh,h2)
         bb=np.append(bb,b2)
         weighti=weighti-1

    bb=bb.reshape(countt-1+len(hi),countt*3,1)
    hh=hh.reshape(countt-1+len(hi),countt*3,countt*3)
    H=sum(hh)
    B=sum(bb)
    hz=np.zeros((len(H),len(H)))
    hz[0][0]=1
    hz[1][1]=1
    hz[2][2]=1
    H2=H+hz
    HH=np.linalg.inv(H2)
    dltx=-np.matmul(HH,B)
    yy=0
    for x in range(int(len(dltx)/3)):
     Pos[x]=deepcopy(Pos[x])+deepcopy(dltx[yy:yy+2])
     Rot[x]=deepcopy(Rot[x])+deepcopy(dltx[yy+2])
     yy=yy+3
    ic=ic+1
######################
Posx=[]
Posy=[]
for x in range(len(Pos)):
```

```
241.  Posx.append(float(Pos[x][0]))
242.  Posy.append(float(Pos[x][1]))
243. plt.plot(Posx,Posy)
244. plt.show()
```