NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF MECHANICAL ENGINEERING
DEPARTMENT OF M.D. & C.S.
Control Systems Laboratory

Postgraduate Thesis

Multicamera Visual SLAM For Mobile Robots

Christos Vasileios Kokas

Supervisor: E.G. Papadopoulos

ATHENS, 2023

Περίληψη

Η παρούσα μεταπτυχιακή εργασία παρουσιάζει την ανάπτυξη και εφαρμογή μιας visual Simultaneous Localization and Mapping (SLAM) μεθόδου για τη χαρτογράφηση του περιβάλλοντος και του εντοπισμού της θέσης του ρομπότ μέσα σε αυτό. Αυτή η υλοποίηση έχει σχεδιαστεί για να λειτουργεί τόσο στο rover όσο και στο ARGOS, που δημιουργήθηκαν στην ομάδα Legged Robots του Εργαστηρίου Αυτομάτου Ελέγχου στο ΕΜΠ. Αυτός ο αλγόριθμος είναι το πρώτο βήμα για τον διάσχιση δύσκολων περιβαλλόντων. Στόχος του αλγορίθμου είναι να παρέχει σε κάθε ρομπότ πληροφορίες για την θέση του ρομπότ, για το περιβάλλον και, σε συνδυασμό με έναν σχεδιαστή διαδρομής (path planner), να διασχίζει το χαρτογραφημένο περιβάλλον.

Στο πρώτο κεφάλαιο εξετάζονται εκτενώς σε διάφορα κινούμενα ρομπότ που θα μπορούσαν ενδεχομένως να χρησιμοποιήσουν τον αναπτυγμένο αλγόριθμο, και παρουσιάζονται οι τρόποι που χρησιμοποιούνται για τον εντοπισμό της θέσης τους. Επιπλέον, παρουσιάζονται αλγόριθμοι SLAM που έχουν ήδη αναπτυχθεί μαζί με τα πλεονεκτήματα και τα μειονεκτήματά τους. Το κεφάλαιο ολοκληρώνεται με τη δομή αυτής της εργασίας.

Στο δεύτερο κεφάλαιο, παρουσιάζονται τα κύρια στοιχεία ενός αλγόριθμο vSLAM. Το κάθε στοιχείο αναλύεται και παρουσιάζονται οι περιορισμοί τους.

Το τρίτο κεφάλαιο αυτής της εργασίας παρουσιάζει τις λεπτομέρειες υλοποίησης του αναπτυγμένου αλγόριθμου SLAM. Οι μέθοδοι που χρησιμοποιούνται παρουσιάζονται και αναλύονται, με έμφαση στην εφαρμογή τους. Ενώ υπάρχουν ήδη πολλοί αλγόριθμοι SLAM, αυτή η εργασία εισάγει μια νέα προσέγγιση με στόχο τη βελτίωση της ακρίβειας και της ευρωστίας της χαρτογράφησης του περιβάλλοντος και του εντοπισμού της θέσης του ρομπότ.

Στο τέταρτο κεφάλαιο παρουσιάζονται τα αποτελέσματα του αλγορίθμου που αναπτύχθηκε. Ο αλγόριθμος χωρίζεται σε δύο διαφορετικές λειτουργίες. Η πρώτη λειτουργία σχεδιάστηκε για να δοκιμαστεί σε γνωστά datasets εικόνων και η δεύτερη λειτουργία περιλαμβάνει τον πλήρη αλγόριθμο, ο οποίος δοκιμάστηκε σε datasets που δημιουργήθηκαν στο CSL, τόσο σε περιβάλλοντα προσομοίωσης όσο και στο φυσικό περιβάλλον. Επιπλέον, παρουσιάζεται ο χρόνος επεξεργασίας κάθε στοιχείου του αλγορίθμου, για να διασφαλιστεί η καταλληλόλητά του για εφαρμογές σε πραγματικό χρόνο.

Το πέμπτο και τελευταίο κεφάλαιο αυτής της εργασίας παρουσιάζει αρκετούς μελλοντικούς ερευνητικούς τομείς που θα μπορούσαν να βελτιώσουν περαιτέρω τον αλγόριθμο SLAM που παρουσιάζεται σε αυτή τη μελέτη. Αυτοί οι τομείς έρευνας παρουσιάζονται λεπτομερώς, μαζί με τα πιθανά οφέλη και τους λόγους για τους οποίους μπορεί να βελτιώσουν την απόδοση του αλγορίθμου που εφαρμόζεται.

Abstract

The present thesis presents the development and implementation of a visual Simultaneous Localization and Mapping (SLAM) application for the mapping of the environment and localization of mobile robots. This implementation has been designed to operate in both the rover, and ARGOS, created at the Legged Robots Team of the Control Systems Lab at NTUA. This algorithm is the first step in localization and traversing challenging environments. Its aim is to provide each robot with information about its position, the environment and, in combination with a path planner and to allow it to traverse the mapped environment.

In the first chapter, an extensive study is conducted on various mobile robots that can potentially employ the developed algorithm, and their localization methods are presented. Additionally, different visual SLAM algorithms already developed are presented along with their advantages and disadvantages. The chapter concludes by presenting the structure of this thesis.

In the second chapter, the main components in a visual SLAM algorithm are presented. Each component is analyzed and their limitations are presented.

The third chapter of this thesis presents the implementation details of the developed visual SLAM algorithm. The methods used are discussed and analyzed, with a focus on their implementation. While there are many visual SLAM algorithms already developed, this thesis introduces a novel approach to improve the accuracy and robustness of environment mapping and robot localization.

In the fourth chapter the results of the developed algorithm are presented. The algorithm is separated into two different operations. The first operation was designed to be tested on well-known image datasets, and the second operation was the complete algorithm, which was tested on custom datasets, both in simulation and real-world environments. Additionally, the processing time of each component of the algorithm is presented, to ensure its suitability for real-time applications.

The fifth and final chapter of this thesis presents several future research areas that could further enhance the visual SLAM algorithm presented in this study. These areas of research are discussed in detail, along with their potential benefits and reasons why they may improve the performance of the implemented algorithm.

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor Professor E. Papadopoulos for his guidance and support throughout my research. His valuable feedback and insights helped me in successfully completing this thesis. I feel truly fortunate to have had the opportunity to work under his guidance.

I would also like to express my sincere gratitude to the three Ph.D. candidates and friends, Athanasios Mastrogeorgiou, Konstantinos Machairas and Konstantinos Koutsoukis, for their assistance and support. In particular, I would like to extend a special thanks to Athanasios Mastrogeorgiou for his exceptional support and guidance throughout the project. Without his help, this thesis would not have been possible.

Moreover, I would like to thank Alexandros Kokas and Christianna Geni for their profound support and help. Last but not least, I would like to thank my family and friends for their love and encouragement.

Dedicated to my Family and Friends

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations	Definitions
AR	Augmented Reality
BA	Bundle Adjustment
BF	Brute-Force
BOVW	Bag Of Visual Words
BRIEF	Binary Robust Independent Elementary Feature
CSL	Control Systems Lab
DBoW	Dorian Bag of Words
DoG	Difference of Gaussian
DJI	Da-Jiang Innovations
FAST	Features From Accelerated Segment Test
FoV	Field of View
GPS	Global Positioning System
IMU	Inertial-Measurement Unit
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
LiDAR	Light Detection and Ranging
LoG	Laplacian of Gaussian
MAV	Micro Aerial Vehicle
MIT	Massachusetts Institute of Technology
ORB	Oriented FAST and Rotated BRIEF
PTAM	Parallel Tracking and Mapping
RGB-D	Red Green Blue-Depth
RMSE	Root Mean Square Error
RTK-GPS	Real Time Kinematics Global Positioning System
SATA	State of The Art
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SSC	Suppression via Square Covering
SURF	Speeded Up Robust Features
SVO	Semi-direct Visual Odometry
VINS	Visual-Inertial System
VIO	Visual-Inertial Odometry
VO	Visual Odometry

Notation

Notation	Definitions
b	Back Stereo Camera
f	Front Stereo Camera
cl	Left Camera
cr	Right Camera
clb	Back Left Camera
crb	Back Right Camera
C	Either Front or Back Camera
$\boldsymbol{p}$	4-by-1 Homogeneous Coordinates
$\boldsymbol{pc}$	4-by-1 Homogeneous Coordinates in the Camera Coordinate Frame
$\boldsymbol{pw}$	4-by-1 Homogeneous Coordinates in the World Coordinate Frame
$\boldsymbol{T_{AB}}$	4-by-4 Transformation from A Coordinate Frame to B Coordinate Frame
V	Either Right or Left Camera

1 Introduction

1.1 Motivation

With the growing demand for autonomous robots, a reliable localization system, which allows robots to determine their position and orientation accurately within their environment, is becoming vital. Although there are many GPS-based solutions, such as RTK GPS, these solutions fail to provide accurate results in GPS-denied environments where the view of the GPS satellites is obstructed or severely limited. This limitation poses a significant challenge in various scenarios, including underground tunnels, dense urban environments with tall buildings, dense forests, and indoor settings. Therefore, there is an increasing demand for a localization system that can operate reliably in such environments.

The objective of this thesis is the creation of a vision-based localization system for mobile robots, that does not rely on GPS for localization, but rather, on images captured from cameras. By using images, the robot can simultaneously localize itself, and map the environment (SLAM). Visual SLAM systems are becoming increasingly popular due to their advantages over GPS-solutions and their lower cost. A thorough literature review of different robots with their localization systems, and of visual SLAM systems, is of utmost importance to in order to identify the most suitable localization solution. This will assist in the development of a robust visual SLAM system that builds upon the strengths of pre-existing solutions and addresses their limitations.

Mobile robots that are designed for tasks in GPS-denied environments and the State of the Art (SOTA) SLAM systems are presented in the next sections.

1.2 Literature Review: Mobile Robots

1.2.1 MIT Cheetah 3

The MIT Cheetah 3 [1] (Figure 1-1) is a four-legged robot that uses a variety of sensors to localize itself.

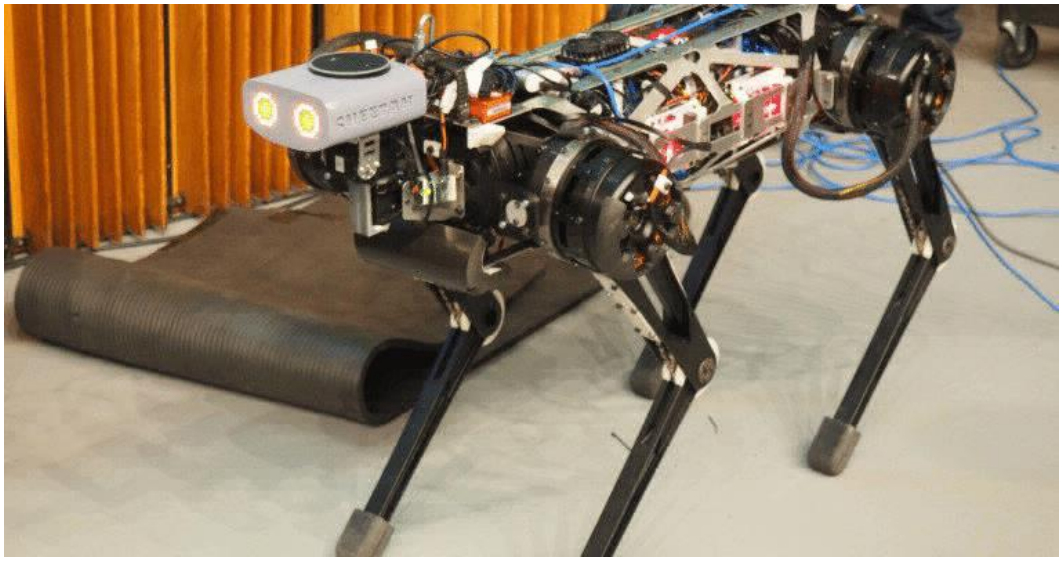


Figure 1-1. The MIT Cheetah 3.

These sensors are [2] :

- LIDAR (Light Detection and Ranging): uses light to create a 3D map of the environment.
- Inertial Measurement Unit (IMU): measures the robot's acceleration and angular velocity.
- Cameras: provide visual information about the environment.
- Encoders: measure the rotation of each or the robot's joints.

These sensors provide an accurate and reliable estimation of the robot's position and orientation in the environment enabling it to navigate complex environments.

1.2.2 Boston Dynamics Spot

Spot [3] (Figure 1-2) is a highly advanced and versatile robotic platform developed by Boston Dynamics. As a quadruped robot, Spot is designed to navigate and operate in various terrains and environments, making it well-suited for a wide range of applications, including inspection, mapping, surveillance, and research.

Spot's perception capabilities are a key aspect of its functionality. Equipped with a suite of sensors, including cameras, depth sensors, and inertial measurement units (IMUs), Spot can perceive and understand its surroundings. The cameras enable visual perception, allowing the robot to detect objects, identify landmarks, and navigate through complex environments. The depth sensors enhance spatial awareness, aiding in obstacle avoidance and mapping. The IMUs contribute to estimating the robot's pose and motion.

Figure 1-2. The Boston Dynamics Spot.

1.2.3 Anymal X

Anymal X [4] (Figure 1-3) is the world's first Ex-Proof legged robot developed by ANYbotics. It is specifically designed and certified for safe usage in hazardous and potentially explosive environments. It is equipped with a 3D LIDAR for obstacle avoidance and localization to navigate even on GPS-denied environments.

Figure 1-3. The ANYmal X Legged Robot.

1.2.4 Husky Rover

Husky [5] (Figure 1-4) is an four-wheeled ground vehicle that was designed for research and development purposes. It was developed by ClearPath Robotics, and its specifications enable it to traverse a wide range of environments. Its accessories for localization include an IMU, a 3D LIDAR and a GPS.

Figure 1-4. The Husky Rover.

1.2.5 DJI Mavic 3

DJI Mavic 3 [6] (Figure 1-5) is the latest flagship camera drone developed by DJI technology. For self-localization purposes, it is equipped with a GPS and multiple wide-angle vision sensors that work seamlessly with a high-performance vision computing engine to sense obstacles in all directions precisely and plan a safe flight route that avoids them.

Figure 1-5. The DJI Mavic 3 Drone.

1.3 Literature Review: vSLAM Algorithms

1.3.1 ORB-SLAM3

ORB-SLAM3 [7] (Figure 1-6) is a real-time SLAM library able to perform Visual, Visual-Inertial and Multi-Map SLAM with monocular, stereo and RGB-D cameras, using pin-hole and fisheye lens models. ORB-SLAM3 is based on ORB-SLAM [8], a monocular visual SLAM algorithm, and ORB-SLAM2 [9], a complete SLAM system for monocular, stereo and RGB-D cameras, including map reuse, loop closing and relocalization capabilities.

Loop Closure is a crucial process in SLAM systems, that assists in error correction and the creation of a more accurate map. ORB-SLAM3 detects loop closures using image similarity, with the use of DBoW2 [10], a library for indexing and converting images into a bag-of-word representation. Loop closure is detected when a new image's bag-of-word representation is similar enough to a previous image's representation.

ORB-SLAM3 has been evaluated in many well-known image datasets, e.g., KITTI, EuRoC, etc., and has demonstrated high accuracy and robustness. As a result, ORB-SLAM3 has become a popular choice in various applications such as robotics and autonomous vehicles, among others.

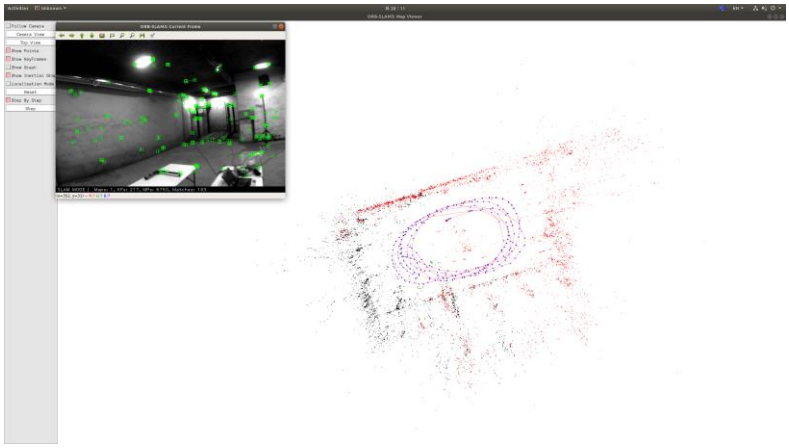


Figure 1-6. ORB-SLAM3.

1.3.2 VINS-Fusion

VINS-Fusion [11] (Figure 1-7) is a visual inertial SLAM system that combines information from visual and inertial sensors to estimate accurately the camera pose in real time. VINS-Fusion is an extension of VINS-Mono [12], a monocular visual inertial SLAM system. VINS-Fusion leverages a tightly coupled optimization-based approach, which allows for efficient and accurate state estimation. Additionally, it includes loop closure detection using DBoW2, similarly with ORB-SLAM3 (Described in Chapter 1.2.6). VINS-Fusion has been tested in a range of challenging environments, in well-known image datasets, displaying accuracy and robustness.

Figure 1-7. Vins-Fusion.

1.3.3 PTAM

PTAM (Parallel Tracking and Mapping) [13] is a monocular visual SLAM system introduced by Kein and Murray in 2011. PTAM was the first algorithm to introduce the use of keyframes for mapping. Keyframes are frames with significant importance, such as the addition of new 3D points, or a significant change in camera motion, that are used in the optimization process to reduce computational complexity. This allows for real-time operation, and as a result many visual SLAM systems have adopted a keyframe-based approach. Additionally, PTAM can

handle fast camera motions due to the lightweight feature-based tracking process. However, many feature-based approaches lose accuracy in low textured environments due to the reduced number of features available for tracking. This can lead to wrong camera pose estimation, and as a result, an inaccurate map. Figure 1-8 presents the tracked features of the PTAM algorithm, where it can be observed that less-textured objects, such as the table, have no available features for tracking.

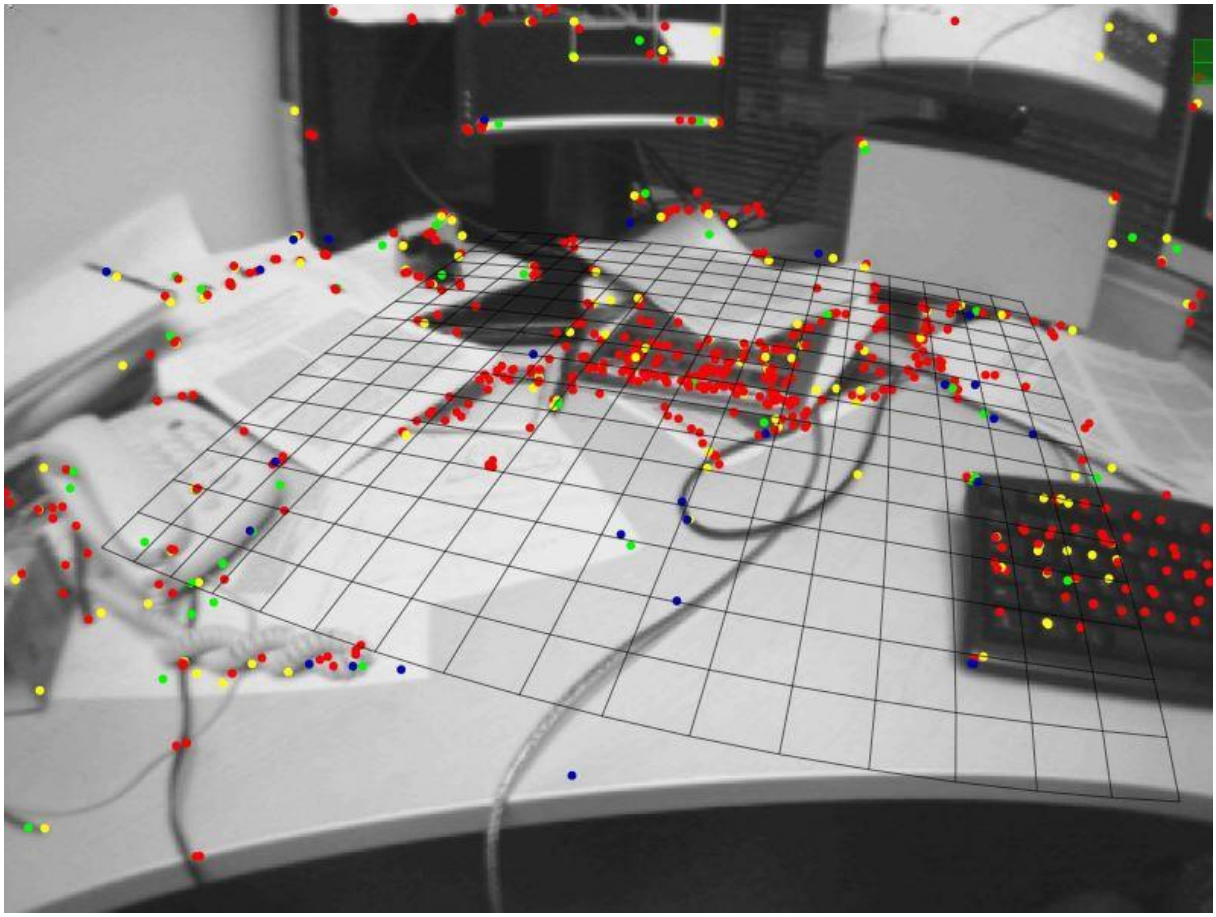


Figure 1-8. PTAM Tracked Features.

1.3.4 Kimera VIO

Kimera-VIO [14] is a Visual Inertial Odometry pipeline for accurate State Estimation from Stereo & IMU data. It can optionally use Mono & IMU data instead of stereo cameras. Kimera extends on pre-existing visual and visual-inertial SLAM systems (e.g., ORB-SLAM, VINS-Mono, OKVIS, ROVIO), by enabling mesh reconstruction (technique for creating a 3D representation of an object from point cloud data) and semantic labelling in 3D (the process of assigning semantic meaning to elements of a 3D model). Kimera is designed with modularity in mind and has four key components: a visual-inertial odometry (VIO) module for fast and accurate state estimation, a robust pose graph optimizer for global trajectory estimation, a lightweight 3D mesher module for fast mesh reconstruction, and a dense 3D metric-semantic reconstruction module. The meshes created provide the ability for fast obstacle avoidance. Additionally, Kimera VIO has loop closure detection capabilities, using DBoW2, similarly with VINS-Fusion and ORB-SLAM3. Figure 1-9 presents Kimera VIO, with the constructed meshes of the environment, in comparison to the ground truth model.

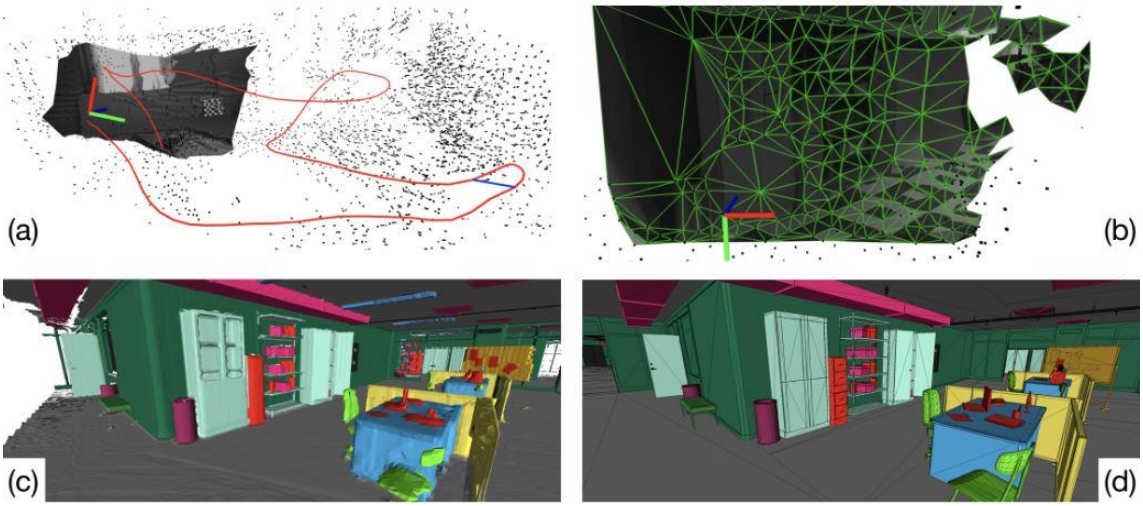


Figure 1-9. Kimera VIO. (a) denotes the visual intertial state estimator, (b) denotes the fast mesh reconstruction of the scene and (c), (d) is the globally semantically annotated 3D mesh (c) in comparison with the ground truth model(d).

1.3.5 SVO

SVO [15] (Figure 1-10) is semi-direct monocular visual odometry system developed by Forster, Pizzoli and Scaramuzza in 2014. It can operate on both pinhole and fisheye cameras. The semi-direct method uses feature-correspondence rather than feature extraction and matching, thus eliminating the need of costly feature extraction and robust matching techniques for motion estimation. Feature extraction is only required when a new keyframe is inserted, to initialize new 3D points. SVO operates directly on pixel intensities, which results in subpixel precision at high framerates and fast camera pose estimation. SVO lacks a loop closure optimization process, resulting in no error reduction, and consequently, it is essential to maintain high frame rates to minimize error accumulation, compared to other state of the art vSLAM systems.

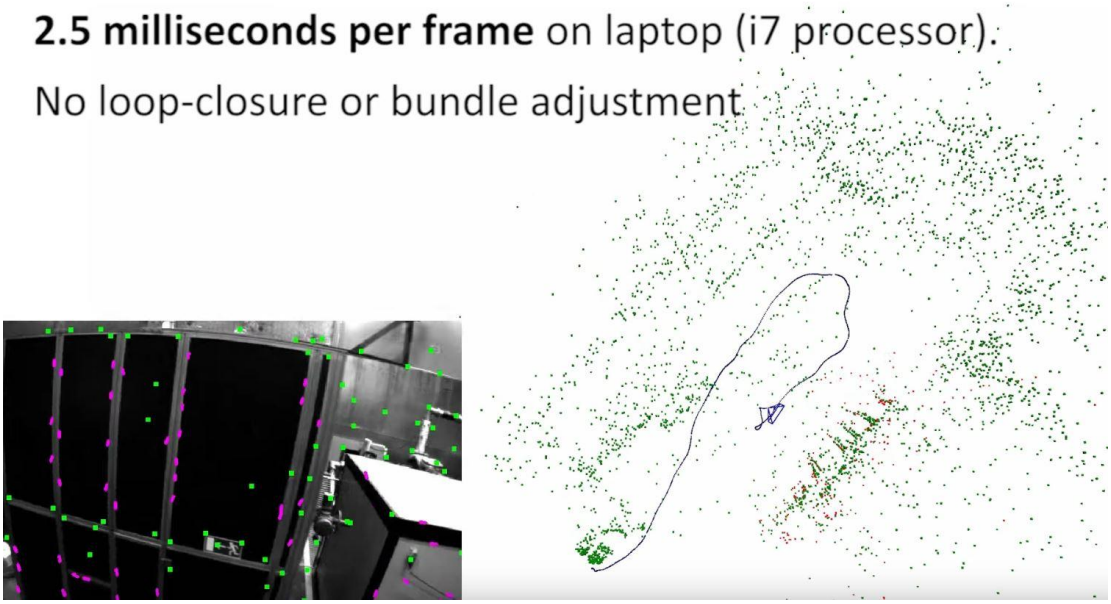


Figure 1-10. SVO Tracking and Map.

1.3.6 SVO 2.0

SVO 2.0 [16] is the extension of SVO to multicamera systems. The system operates with multiple monocular cameras (Figure 1-11), both pinhole and fisheye, providing more accurate camera pose estimation results, and a more detailed map of the environment. SVO 2.0, although an extension of SVO, still lacks a loop closure optimization process, resulting in less accurate camera pose estimation and mapping on longer paths due to error accumulation.

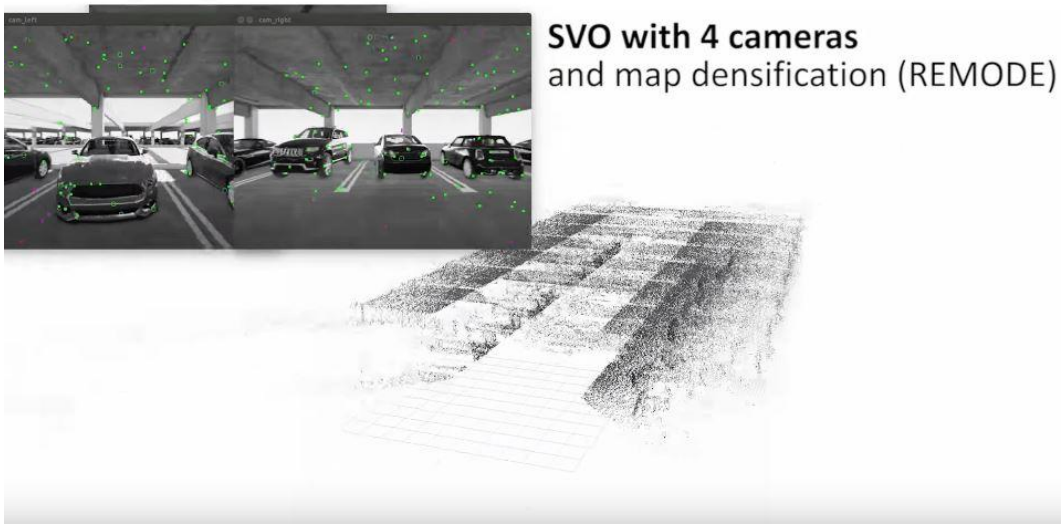


Figure 1-11. SVO 2.0 with Multiple Cameras.

1.3.7 SLAM for Arbitrary Multi-Camera Systems

A SLAM system for multiple cameras was proposed in 2020 [17]. The system can operate with up to six cameras without the need for sensor-specific modifications or tuning. In a setup with multiple cameras, when a camera's field of view is blocked, the additional cameras can help estimate the camera pose. However, the proposed system is missing a loop closure optimization process, resulting in error accumulation over time. Figure 1-12 presents the system operating with three fisheye cameras in a real-world environment.

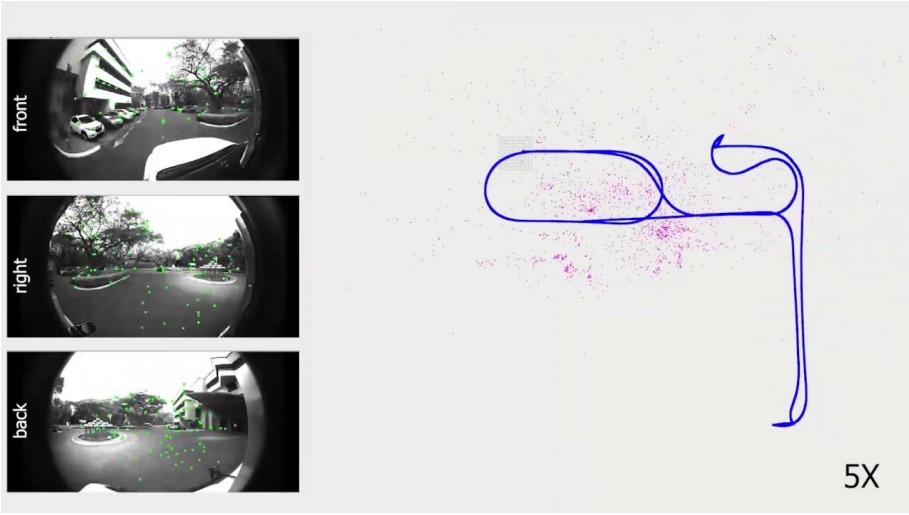


Figure 1-12. SLAM for Multi-Camera Systems.

1.4 Thesis Structure

The present thesis is structured as follows: Chapter 2 provides an overview of the main components of a SLAM system, including key components that are utilized in the developed SLAM system. In Chapter 3, the developed SLAM system is described in detail, analyzing the different parts of the system. Chapter 4 presents the results of the developed SLAM system, on well-known image datasets, in a Gazebo simulation and at the CSL lab environment. Finally, Chapter 5 summarizes the work and discusses potential future research directions and improvements for the system.

2 Fundamentals of a Visual SLAM Algorithm

This chapter provides a detailed overview of the fundamental components of a visual SLAM algorithm. Firstly, a thorough description of the stereo camera and its properties is provided. Following that, various feature extraction methods and matching techniques are explained. Additionally, different Bundle Adjustment algorithms are described that aim to reduce any error accumulation and improve the accuracy of visual SLAM algorithm. Lastly, a detailed explanation of Loop Closure and its significance in visual SLAM algorithms is provided.

2.1 Stereo Camera

A stereo camera [18] (Figure 2-1) is a camera type with two lenses, each with a separate image sensor. By capturing two slightly different perspectives of a scene, a stereo camera can mimic human binocular vision, enabling the creation of three-dimensional images. In robotics, stereo cameras are widely used because they offer the through triangulation. Stereo cameras come in different models:

- Pinhole: A pinhole camera is a simple camera without a lens but with a tiny aperture (the so-called pinhole). Light from a scene passes through the aperture and projects an inverted image on the opposite side of the box [19].
- Fisheye: A fisheye lens is an ultra wide-angle one that produces strong visual distortion intended to create a wide panoramic or hemispherical image. Fisheye lenses achieve extremely wide angles of view, typically exceeding 180 degrees [20].

This section examines the pinhole camera model.

Figure 2-1. A ZED Stereo Camera [18].

2.1.1 Image Distortion

Image distortion, presented in Figure 2-2 [21], is when the straight lines of an image appear to be deformed or curved unnaturally. More specifically, the pinhole camera model is subject to distortion, which is primarily observed at the edges of the image. To accurately calculate depth, it is necessary to minimize distortion as much as possible. This ensures that the calculations closely approximate the true values. Techniques such as lens calibration and image correction algorithms can be employed to reduce distortion and improve the accuracy of depth measurements.

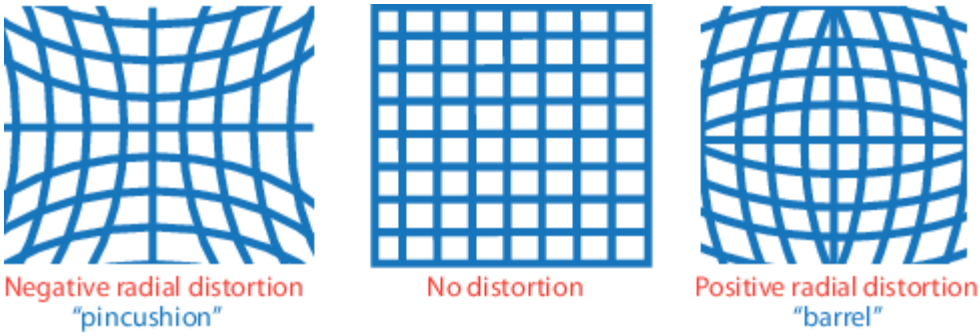


Figure 2-2. Image Distortion [21].

2.1.2 Camera Calibration

Camera calibration is the process of estimating the parameters of a camera lens and image sensor. The calibration algorithm estimates the extrinsic and intrinsic parameters of a camera's lens. The extrinsic parameters represent the location and the orientation of the camera in the 3D scene, while the intrinsic parameters include the focal length, the optical center, and the skew coefficient.

The calibration process includes a set of images, usually 15-20 images, (Figure 2-3 [22]) that include a calibration object with known dimensions, typically a chessboard. The images are then analyzed to estimate the camera's intrinsic and extrinsic parameters, using a calibration algorithm, e.g., Zhang Algorithm [23], Jean-Yves Bouguet [24], etc. The intrinsic parameters in combination with the extrinsic parameters form the projection matrix, that can map 3D points in the world to their corresponding 2D locations in the image plane.

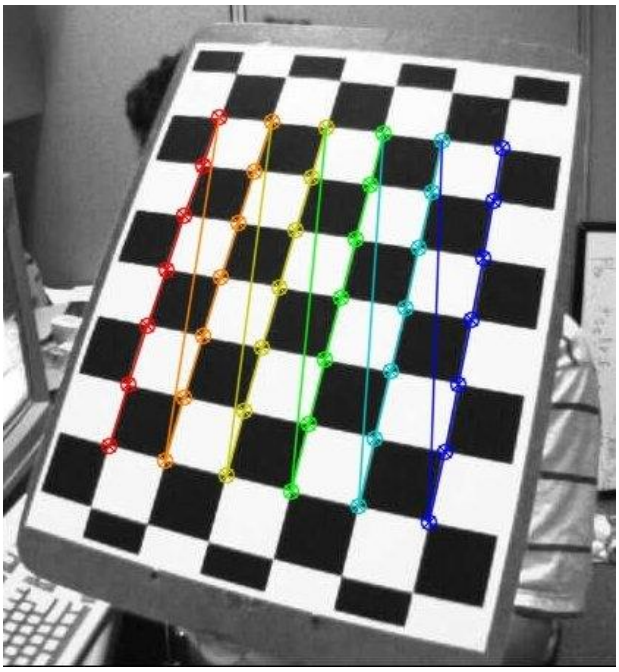


Figure 2-3. Image used for Camera Calibration [22].

The intrinsic parameters are represented in an intrinsics matrix K defined as:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

Where f_x, f_y is the focal length in pixels and c_x, c_y is the optical center in pixels. The extrinsic parameters are the location and orientation of the camera with respect to the world coordinate system and consist of a rotation matrix R and a translation displacement t. R is a 3-by-3 matrix, while t is a 3-by-1 matrix. The camera projection matrix P is a 3-by-4 matrix that maps the 3-D world scene into the image plane [25] and it is calculated using Equation (2.2).

$$P = K[R \quad t] \tag{2.2}$$

2.1.3 Image Rectification

Image rectification [26] is a transformation process used to project images onto a common image plane. In the context of stereo cameras, image rectification is utilized to enable the use of epipolar geometry (Described in Chapter 2.1.4). Specifically, image rectification remaps the pixels of each image such that the epipolar lines are horizontal, which means that each pixel on the left image has the same y-value (assuming a horizontal stereo camera) as the corresponding pixel on the right image, as presented in Figure 2-4 [27]. After the remapping process, the images are cropped to the same size as the original ones. That results in loss of information, but the benefits far outweigh the downsides.

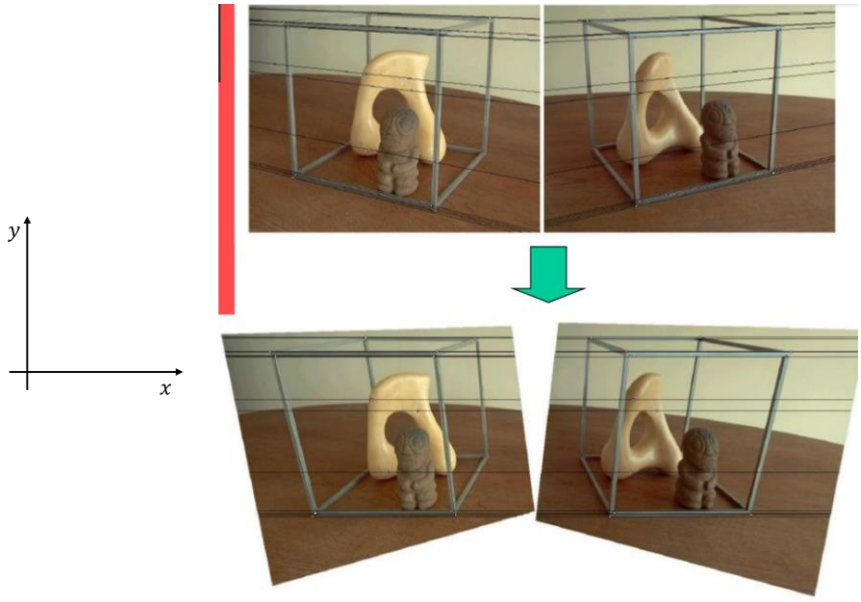


Figure 2-4. Image Rectification [27].

2.1.4 Epipolar Geometry

Epipolar geometry [28] is the geometry of stereo vision. When two cameras view a 3D scene from two distinct positions, there are geometric relations between the 3D points and their projections onto the 2D images that lead to constraints between the image points. These relations are derived based on the assumption that the cameras can be approximated by the pinhole camera model. Since the optical centers of the cameras lenses are distinct, each center projects onto a distinct point into the other camera's image plane. These two image points, denoted by e_L and e_R in Figure 2-5, are called epipoles or epipolar points. Both epipoles e_L and e_R in their respective image planes and both optical centers O_L and O_R lie on a single 3D line. The line $O_L - X$ is seen by the left camera as a point because it is directly in line with that camera's lens optical center. However, the right camera sees this line as a line in its image plane. That line ($e_R - x_R$) in the right camera is called an epipolar line. [28]. By

using epipolar geometry (presented in Figure 2-5 [28]) on a rectified image each pixel on the left image has the same y value as each corresponding pixel on the right image (if the stereo camera is horizontal). As a result, if the transformation matrix between the two cameras is known, through triangulation the depth of each pixel, seen on both images, can be calculated.

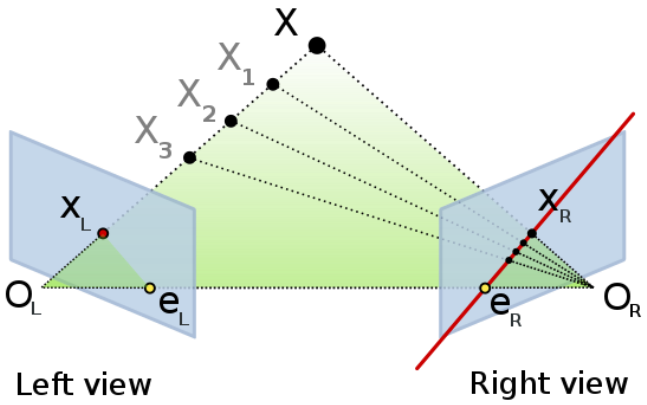


Figure 2-5. Epipolar Geometry [28].

2.1.5 Disparity

With the use of rectified images on a horizontal stereo camera setup, each pixel on the left camera, that is visible in the right camera, corresponds to a pixel on the right camera with the same y value, but with a different x value. The difference in the x values of the corresponding pixels is called disparity.

The disparity of each pixel can be calculated using an algorithm, such as Semi-Global Block Matching [29], that scans both the left and right images for matching pixels. A common approach to this problem is to form a small image patch around every pixel in the left image. These image patches are compared to all possible disparities in the right image by comparing their corresponding image patches. For example, for a disparity of 1, the patch in the left image would be compared to a similar-sized patch in the right, shifted to the left by one pixel.

By finding the disparity between the two images, the depth (the distance from the camera in meters) can be calculated from Equation (2.3) [30], where Baseline is the distance between the two camera lenses.

$$Depth = \frac{Baseline * FocalLength}{Disparity} \tag{2.3}$$

An image with its disparity map is presented in Figure 2-6 [31].

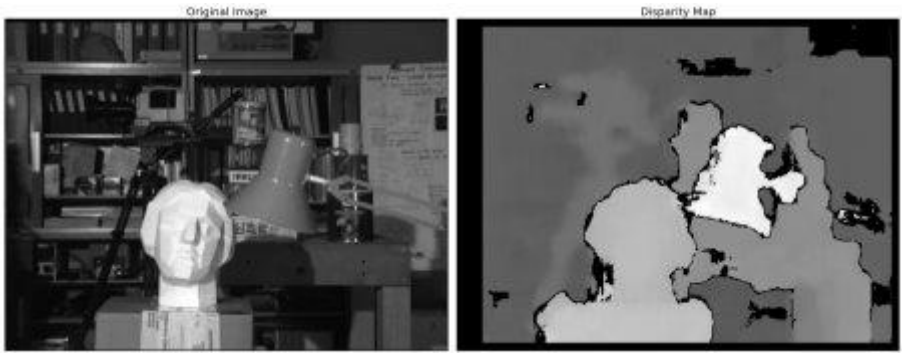


Figure 2-6. Disparity Map [31].

2.2 Features

A feature is typically defined as a measurable and distinctive attribute or characteristic part of an image, shape etc. For example, a square has 4 corners and 4 edges, they can be called features of the square, and they help identify it's a square. Features are used as a starting point for many computer vision algorithms, e.g., object recognition etc. Some common types of features in an image include:

- Edges: Edges are abrupt changes in the intensity or color of an image, which can indicate the boundaries of objects or regions.
- Blobs: Blobs are regions of an image that have a similar intensity or color. They can be used to identify objects or regions of interest.
- Corners: Corners are points in an image where the intensity changes in multiple directions. Corners can be used to identify key points in an image.

The different types of features are presented in Figure 2-7 [32], [33], [34].

For visual SLAM applications, grayscale images are commonly used, and the features that are typically selected for these images are corners.

To differentiate features, a method of describing them is required, which is where descriptors become useful. Descriptors provide a unique representation of a feature, allowing them to be distinguished from one another.

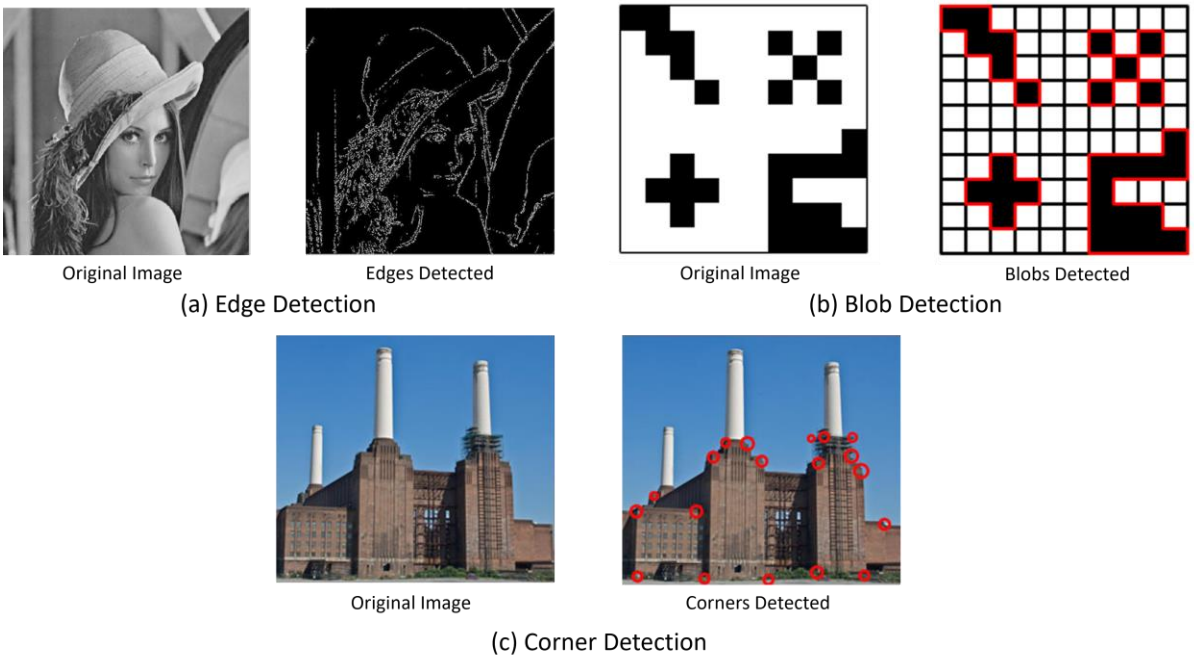


Figure 2-7. Different types of features [32], [33], [34].

2.2.1 Feature Extraction

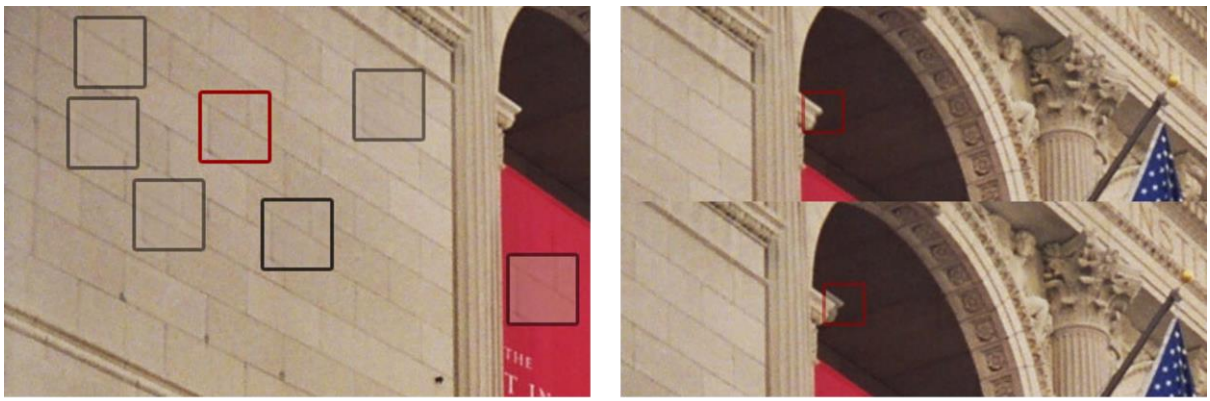
There are several feature extraction methods. In this paragraph, the most used ones, are presented.

Harris Corner Detector

Harris Corner Detector is a method for selecting features in an image by choosing one patch of the image and comparing it with equal sized patches around it. The comparison is performed by adding up the differences in intensity of each pixel on the first patch with the

pixel on the same position on the other patches. If a patch generates a significant difference, more than a threshold, then it is considered a feature.

In Figure 2-8 [35] two examples are presented. In Figure 2-8(a), where a selected patch (red) is being compared with different patches (black), the comparison does not show a significant difference. In Figure 2-8(b), where the selected patch (red) yields a significant difference when compared with a patch that has been shifted to the right, the presence of a distinctive feature in that region is suggested.



(a) (b)

Figure 2-8. Harris Corner Detector Selection Method [35].

Scale-Invariant Feature Transform (SIFT) [36]

The SIFT algorithm has several steps for selecting features in an image.

- The first step is the selection of features that have scale and rotation invariance. Features with scale invariance are selected using a difference-of-Gaussian (DoG) filter to detect local extrema in the scale space of the image.
- Once the features have been detected, their locations are refined by fitting a model to the DoG response function. This assists in eliminating features that are poorly localized.
- To achieve rotation invariance, each feature is assigned an orientation, which is computed using a histogram of gradient directions in the local region around the feature.
- Lastly, a descriptor needs to be calculated for each feature, so that the features are unique and easily distinguishable. The descriptor is calculated by computing the gradient magnitude and orientation of the image in a region around the feature and then constructing a histogram of oriented gradients.

Although the SIFT algorithm extracts features with rotation and scale invariance, the process is time consuming, and it is generally not fit for real time applications.

Speeded-Up Robust Features (SURF) [37]

SURF approximates LoG with Box Filter, in contrast with SIFT, that approximates LoG with Difference of Gaussian. To detect features, SURF uses an integer approximation of the determinant of Hessian blob detector, which can be computed with 3 integer operations using a precomputed integral image.

Its feature descriptor is based on the sum of the Haar wavelet response around the point of interest. It is partly inspired by the SIFT descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image

transformations than SIFT. Although SURF is faster than SIFT, its processing time is still inadequate for real time applications. The SURF algorithm is described in depth in OpenCV docs [38].

Features from Accelerated Segment Test (FAST)

Features from accelerated segment test (FAST) is a corner detection method, which is used to extract features and later used to track and map objects. The most promising advantage of the FAST corner detector is its computational efficiency. Referring to its name, it is indeed faster than many other well-known feature extraction methods, such as DoG used by the SIFT and SURF. The FAST corner detector is very suitable for real-time video processing applications because of its computational efficiency.

FAST corner detector uses a circle of 16 pixels (a circle of radius 3 pixels created using the midpoint circle algorithm [39] to classify whether a candidate point p is a feature. Each pixel in the circle is labeled clockwise with an integer number from 1 to 16. If a set of N contiguous pixels in the circle are all brighter than the intensity (denoted by I_p) of candidate pixel p (named fast threshold) or all darker than the intensity of candidate pixel p minus threshold value t, then p is classified as feature. The conditions can be written as:

- Condition 1: A set of N contiguous pixels S, $\forall x \in S$, the intensity of $x > I_p + t$.
- Condition 2: A set of N contiguous pixels S, $\forall x \epsilon S, I_x < I_p - t$.

When either of the two conditions is met, candidate p is classified as a feature. There is a tradeoff of choosing N (usually chosen as 12), the number of contiguous pixels and the threshold value t. On one hand the number of detected corner points should not be too many, on the other hand, the high performance should not be achieved by sacrificing computational efficiency. This process is presented in Figure 2-9 [40].

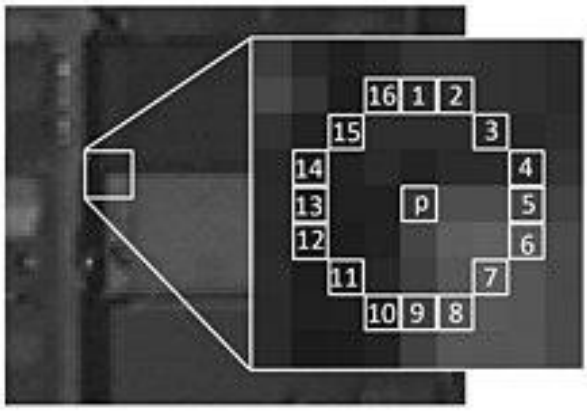


Figure 2-9. FAST Corner Detector [40].

The FAST corner detector extracts features rapidly [41] making them suitable for real time applications. However, the detected features lack rotation and scale invariance, which are crucial properties for achieving accuracy and robustness in a visual SLAM system.

Binary Robust Independent Elementary Feature (BRIEF)

BRIEF takes all features and converts them into a binary feature vector so that together they can represent an object. Binary feature vector also known as binary feature descriptor is a feature vector that only contains 1 and 0. In brief, each keypoint is described by a feature vector which is 128–512 bits string. Because BRIEF uses binary strings, it provides a fast method [42] for feature description and matching.

Oriented FAST and Rotated BRIEF (ORB) [43]

The ORB (Oriented FAST and Rotated BRIEF) algorithm is an improvement over the FAST corner detector and the BRIEF (Binary robust independent elementary feature) descriptor. The ORB algorithm builds on the strengths of both FAST and BRIEF to extract features that are both robust and computationally efficient.

ORB's main contributions are as follows:

- The addition of scale invariance to FAST features.
- The addition of an accurate orientation component to the features.
- Analysis of variance and correlation of oriented BRIEF features.
- A learning method for decorrelating BRIEF features under rotational invariance, leading to better performance in nearest-neighbor applications.

ORB uses the FAST algorithm for feature extraction. However, to extract FAST features that are scale invariant it uses a multiscale image pyramid (presented in Figure 2-10). An image pyramid is a multiscale representation of a single image, that consists of sequences of images all of which are versions of the image at different resolutions. Each level in the pyramid contains a down sampled version, depending on the scale selected, of the image than the previous level. For example, in Figure 2-10 [44] the scale selected is 2 so the images in each pyramid level are 2 times smaller (on each axis) than the image on the previous level. Once ORB has created a pyramid, it uses the FAST algorithm to detect features in the images. By detecting features at each level ORB is effectively locating features at a different scale. In this way, ORB produces features that are scale invariant.

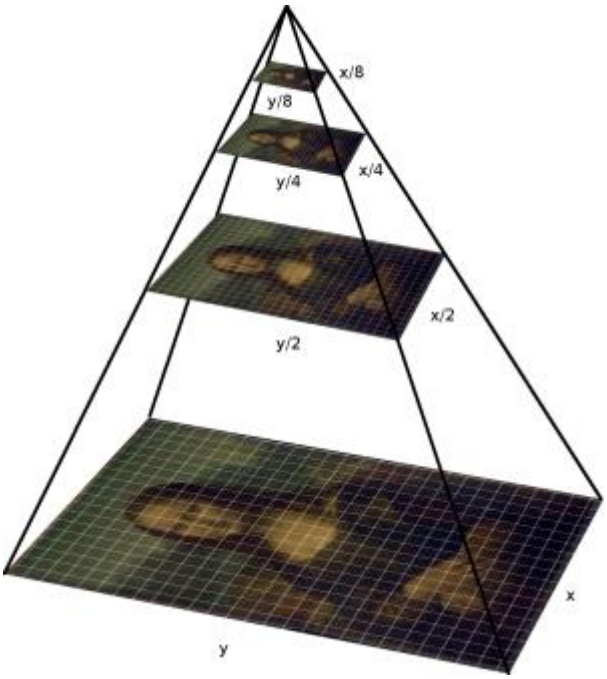


Figure 2-10. Multiscale Image Pyramid [44].

To create rotation invariant features, ORB calculates an orientation for each of the detected features depending on how the levels of intensity change around each feature. In more detail, to calculate the orientation of each feature, the ORB algorithm uses the intensity centroid. The intensity centroid assumes that the patch around the feature has higher intensity in a direction offset from its center, and this direction is used as the orientation.

To compute the orientation of each feature, the moments of the patch around each feature are calculated using Equation (2.4). The moment of a patch is defined as the average values from the single pixels` intensities of a patch.

$$m_{pq} = \sum_{x,y} x^p \, y^q I(x,y) \tag{2.4}$$

Where m_{pq} is the moment of the patch around each feature, p, q denote the order $(p+q)$ of the moment and $I(x,y)$ is the intensity of the pixel with coordinates x, y. To calculate the centroid of the patch, also known as "center of mass", p, q take values of either $0\ or\ 1$ as presented in Equation (2.5) where C denotes the coordinates of the centroid.

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \tag{2.5}$$

Finally, the orientation of each feature can be calculated by constructing a vector from the center of the feature to the coordinates of the centroid C presented in Equation (2.6), where θ is the orientation of the feature.

$$\theta = atan2(m_{01}, m_{10}) \tag{2.6}$$

The vector used to calculate the orientation of the patch is presented in Figure 2-11.

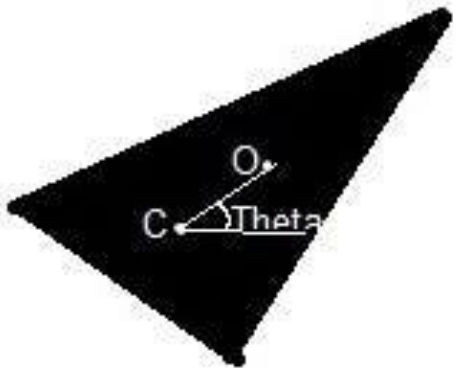


Figure 2-11. Vector used to calculate the orientation of the patch [45].

The ORB feature detection algorithm has several parameters that can be adjusted to control its behavior:

- nfeatures: The maximum number of features to detect.
- scaleFactor: The scale factor between pyramid levels.
- nlevels: The number of pyramid levels. Maximum number of pyramid levels depending on the resolution of the image and the scaleFactor selected.
- edgeThreshold: The size of the border where features are not detected.
- firstLevel: The level of pyramid to start detection.
- WTA_K: The number of points that produce each element of the descriptor.
- scoreType: What type of scoring to use for each feature. Score indicates the likelihood each feature is actually a feature.
- patchSize: The size of the patch used to compute the descriptor.
- fastThreshold: The threshold used by the FAST algorithm.

ORB features extract FAST features, that are computationally efficient [41] and improve them with scale and rotation invariance properties, making them robust and suitable for real

time applications. These properties make ORB features a popular choice for various computer vision applications such as object recognition and visual SLAM.

Each feature stores information for:

- pt: coordinates of the feature in the image plane (x, y).
- Angle: computed orientation of the keypoint (-1 if not applicable); it's in [0,360) degrees and measured relative to image coordinate system, i.e., clockwise.
- octave: octave (pyramid layer) from which the keypoint has been extracted.
- Size of the feature. This is used for the calculation of the feature descriptor. The size of the feature depends on the pyramid level from which it was extracted, the smaller the image, the bigger the size.
- Response: the response by which the strongest feature has been selected. Can be used for further sorting or subsampling.

Comparison of the Different Feature Extraction Methods

Figure 2-12 [46] presents a comparison in the processing time among various feature extraction methods. As shown, the FAST algorithm is the fastest, while SIFT and SURF are the slowest. Despite being slower than FAST, the ORB algorithm is still suitable for real time applications; and its ability to extract scale and rotation invariant features make it suitable for visual SLAM applications.

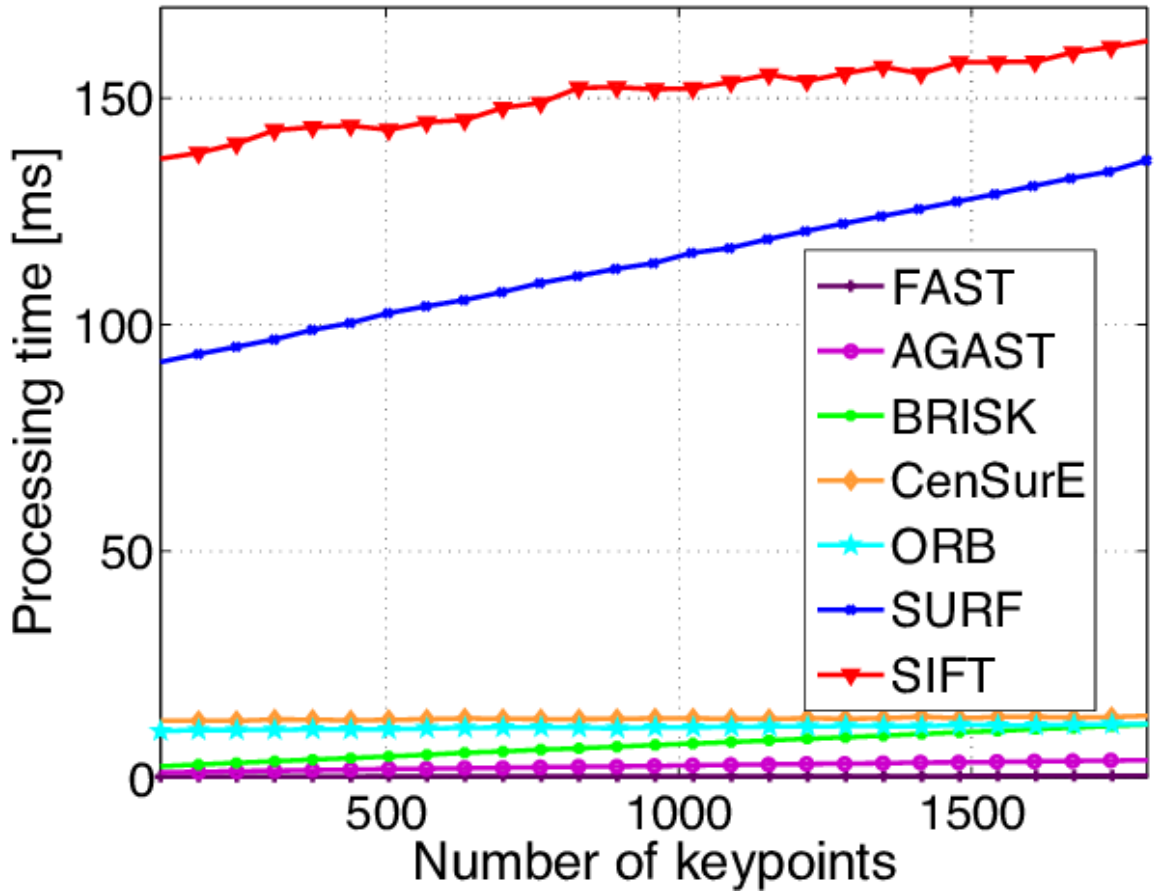


Figure 2-12. Processing Time Comparison of Feature Extraction Methods [46].

Figure 2-13 [47] presents the features of the different feature extraction methods on the same image.

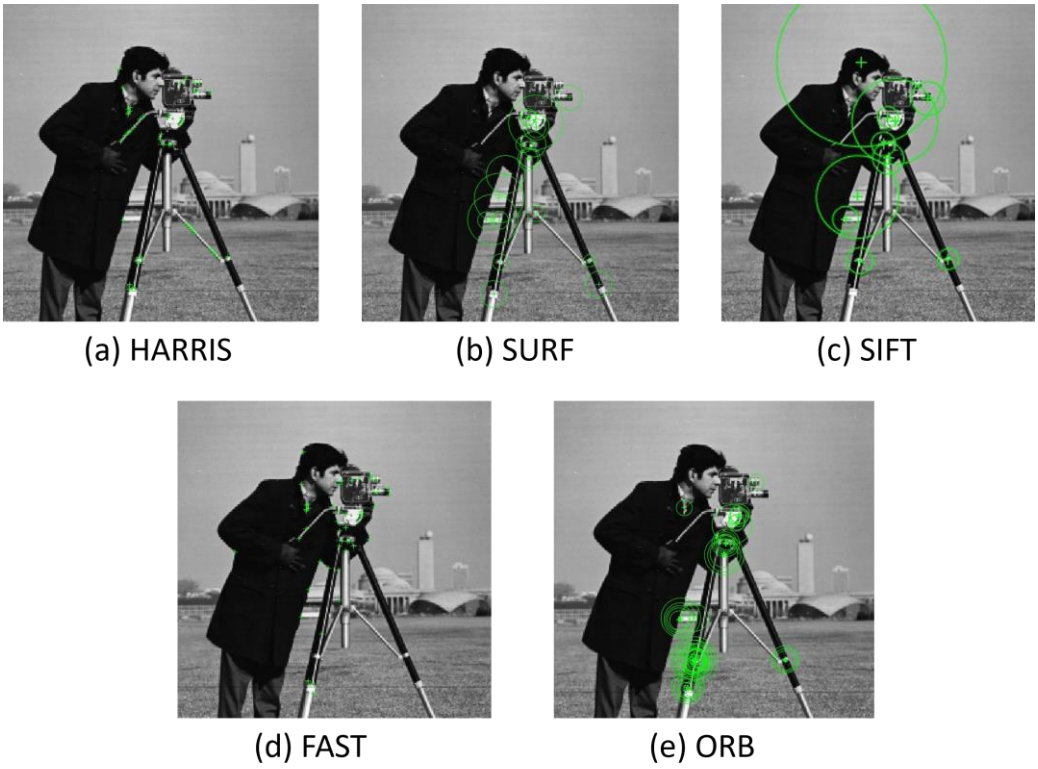


Figure 2-13. The Different Feature Extraction Methods [47].

For visual SLAM applications, feature extraction is a critical step that has an impact on the accuracy and robustness of the algorithm. The features have to be extracted efficiently, while maintaining rotation and scale invariance.

2.2.2 Matching

Brute-Force (BF) Matcher [48]

The Brute-Force (BF) matcher is a simple algorithm used in computer vision and image processing for feature matching. It takes the descriptor of each feature on one set and compares it with the descriptor of each feature on the second set, using a distance calculation, e.g., L1 norm, L2 norm, Hamming Distance, etc. The closest one is selected as a match.

Although the BF matcher is a simple and effective (as the closest match for each feature is selected) algorithm, it can be slow and computationally expensive due to the need to compare each feature with all others resulting in a $O(n * m)$ time complexity. Figure 2-14 [49] presents the matched features using BF Matcher.

Figure 2-14. BF Matcher [49].

Fast Library for Approximate Nearest Neighbors (FLANN) Based Matcher [50].

The FLANN (Fast Library for Approximate Nearest Neighbors) algorithm is a method for efficiently searching high-dimensional spaces for nearest neighbors. The main steps of the FLANN algorithm are:

- Build an index: The first step is to construct an index structure using the FLANN library. The type of index depends on the application and the data being searched. FLANN supports various types of index structures, such as k-d trees, randomized trees, and hierarchical clustering. The index is constructed based on the feature descriptors of the image features.
- Query the index: Given a query feature descriptor from one image, the algorithm uses the FLANN index to efficiently search for the nearest neighbors in the other image. The algorithm performs a hierarchical search, starting with a coarse search and progressively refining the search until the desired level of accuracy is achieved.
- Evaluate the search results: The algorithm returns a set of nearest neighbors, along with their distances to the query point or feature. Depending on the application, additional filtering or post-processing may be applied to the results to remove false matches or outliers.

The FLANN algorithm is designed to work with high-dimensional data, which can be challenging for traditional nearest neighbor search algorithms such as the BF matcher. By using an index structure, FLANN can efficiently search large datasets and return approximate nearest neighbors with a high degree of accuracy. Figure 2-15 [51] presents the matches found using the FLANN based matcher.

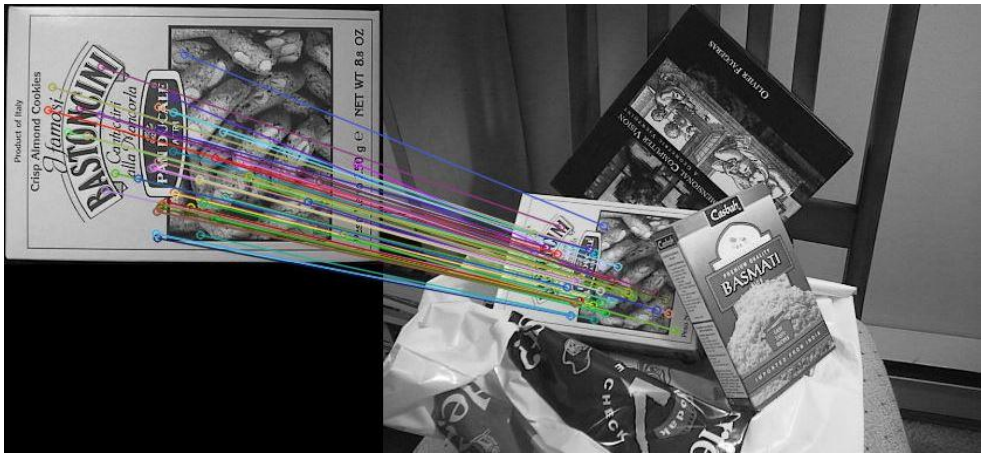


Figure 2-15. FLANN Based Matcher [51].

Match Filtering

BF and FLANN matchers may produce incorrect matches due to image noise, but there are various techniques available to filter out false matches.

One of the most common techniques is called the ratio test. In the ratio test, for each feature, two of the closest matches are calculated. Then the ratio of the distance between the two closest matches is calculated. If the ratio is below a certain threshold, the match is considered valid. However, if the ratio is above the threshold, the match is considered ambiguous or false, and it is discarded. A lower threshold will result in fewer false matches

being accepted but could discard some valid matches. The threshold value can vary depending on the application, but generally, it is set around 0.75.

Another common technique is a threshold on the distance between the matches. If the distance of a match is higher than the threshold, even if it is the closest one, it is discarded.

Stereo Matching

A special case of matching is stereo matching which is the process of matching features of the left camera with the right camera. With stereo matching, the depth can be calculated for every match as described in Section 2.1.5. Figure 2-16 [52] presents the parameters used in stereo matching.

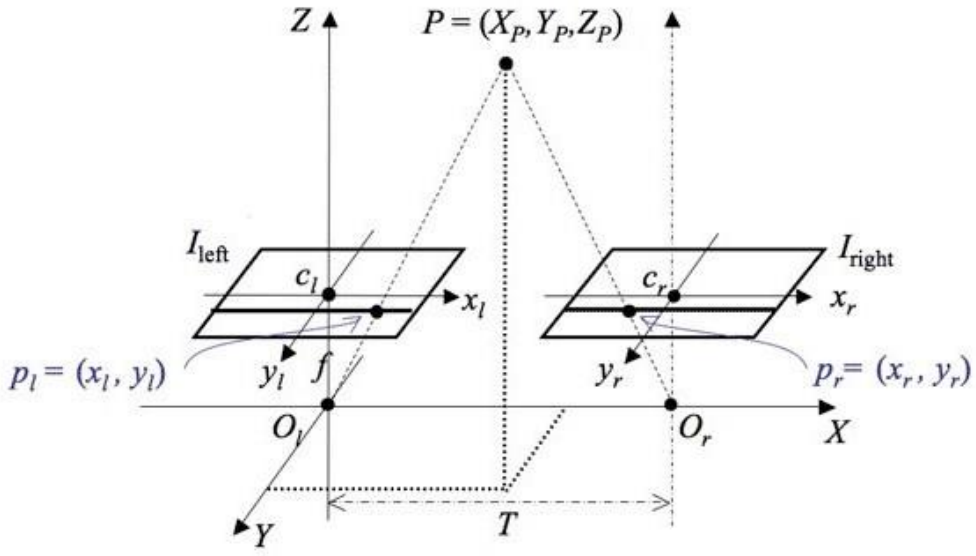


Figure 2-16. Stereo Matching Parameters [52].

Using Equations (2.7)–(2.9) a 3D point $\boldsymbol{p}(X_p, Y_p, Z_p) = \begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix}$ can be created as a function of its position in the image.

$$Z_p = Depth \tag{2.7}$$

$$X_p = (p_{lx} - c_{lx}) * \frac{Z_p}{f_x} \tag{2.8}$$

$$Y_p = (p_{ly} - c_{ly}) * \frac{Z_p}{f_y} \tag{2.9}$$

The position of a feature in the left camera is denoted by $\boldsymbol{p_l}(x, y) = \begin{bmatrix} p_{lx} \\ p_{ly} \end{bmatrix}$ and the optical center of the left camera is represented by $c_l(x, y)$, while f_x and f_y are the intrinsic parameters of the left camera. Using these parameters, the 3D point is computed with the world origin 0,0 being the optical center of the left camera. The 3D point can also be calculated using the parameters from the right camera, resulting in a 3D point with the world origin 0,0 at the optical center of the right camera. In visual SLAM systems, the 3D points computed using stereo matching are used for the mapping process.

2.3 Camera Pose Estimation

To estimate the camera pose using a stereo camera configuration, 3D points with their matching 2D image features are used. The 3D points are represented in world coordinates, and the camera pose is estimated with respect to the world coordinate system. The camera pose is estimated by minimizing the reprojection error [53].

The reprojection error is the difference between the observed 2D image points and the 2D image points that are reprojected from the estimated 3D points using the estimated camera pose. The reprojection error is calculated with Equation (2.10).

$$reproj = \sqrt{\left(x_{pro} - x_{point}\right)^2 + \left(y_{pro} - y_{point}\right)^2} \tag{2.10}$$

where x_{pro}, y_{pro} are the x, y coordinates of the projection of the 3D point, the x_{point}, y_{point} are the coordinates of the observed 2D image point and the $reproj$ is the calculated reprojection error.

The reprojection error is presented in Figure 2-17 [54]. The process involves the following steps:

- Find stereo matches between left and right images.
- Triangulate the 3D points from the stereo matches.
- On the next frame, find features and match with the previously computed 3D points.
- Estimate the camera pose by minimizing the reprojection error.

Figure 2-18 [55] presents visually the process.

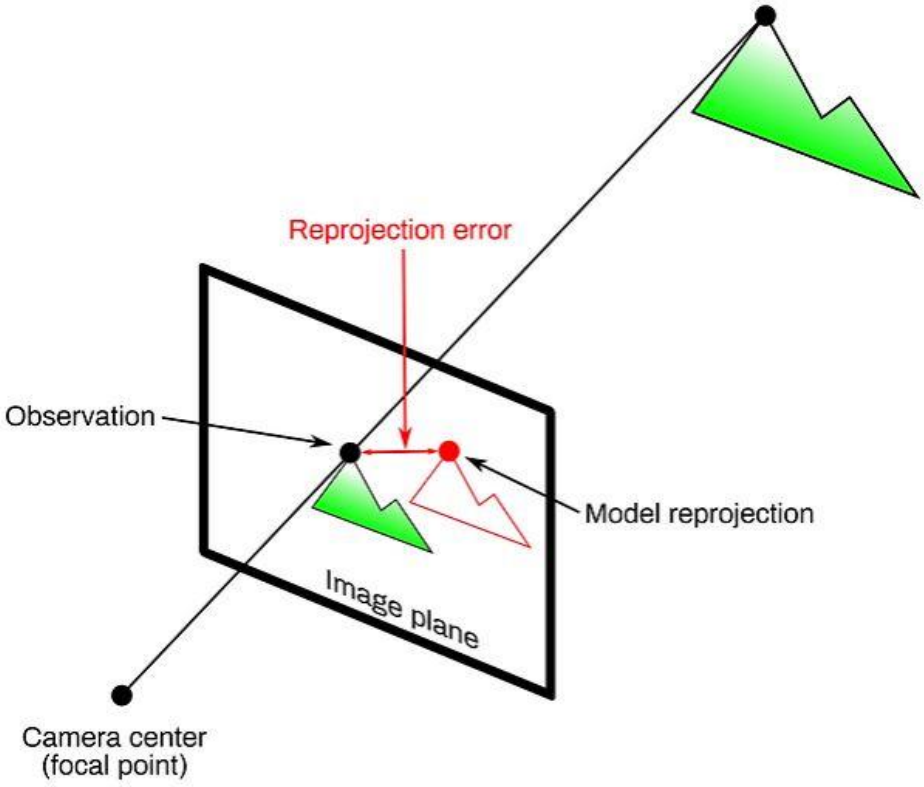


Figure 2-17. Reprojection Error [54].

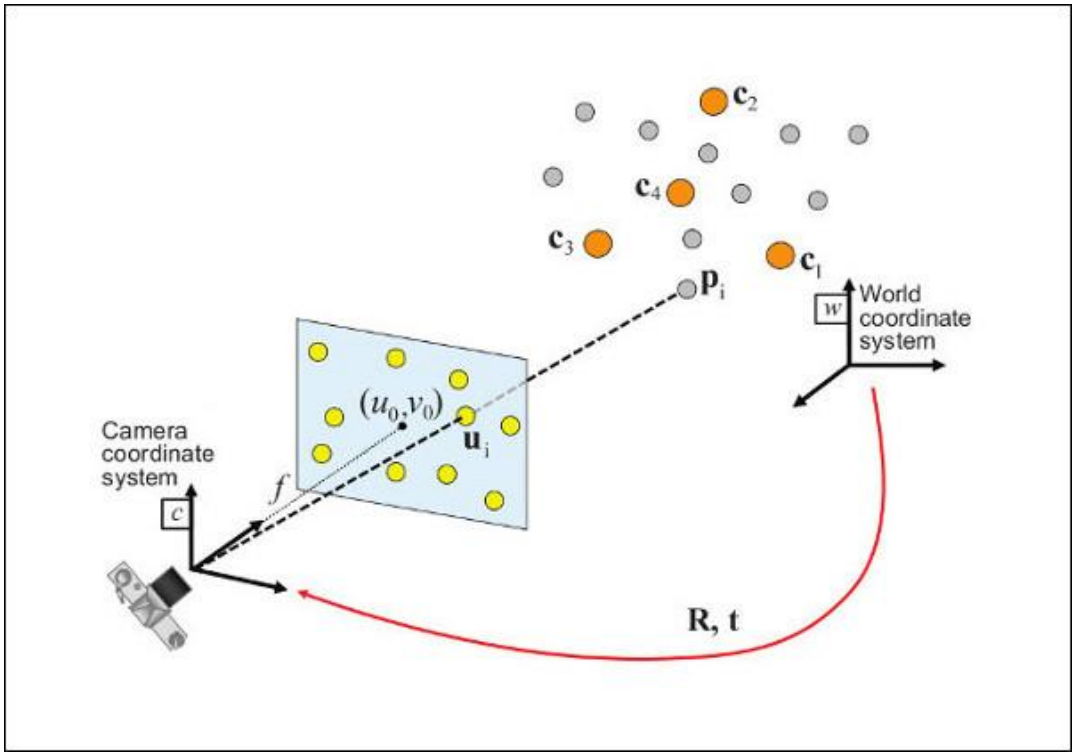


Figure 2-18. Camera Pose Estimation [55].

One of the most used methods to estimate the camera pose with the 3D point and 2D image correspondence is by using the SolvePnP function provided by OpenCV. The process is described in detail in the OpenCV docs [55].

2.4 Bundle Adjustment (BA)

In visual SLAM applications, estimating the camera pose for each new frame for an extended period can lead to error accumulation due to various factors such as image noise or bad matches that were considered valid. This can result in a significant drift in the estimated trajectory over time. To reduce this error and correct the estimated camera poses, a technique called bundle adjustment was introduced [56].

Bundle adjustment is a method that simultaneously optimizes the camera poses and the 3D points of the observed features to minimize the overall reprojection error of all the observed features over the entire sequence of images. In other words, it refines the initial estimates of the camera poses and locations of the 3D points by finding the optimal solution that best fits all 3D points observed from each camera pose.

Bundle adjustment is an iterative process. It is initialized using the estimation of the camera poses and locations of the 3D points. Then, the reprojection error is computed for all the observed features, and the camera poses and locations of the 3D points are refined using a non-linear optimization algorithm, e.g., the Levenberg-Marquardt algorithm [57] or Schur complement method [58]. This process is repeated until the reprojection error converges to a minimum.

The Bundle adjustment is computationally expensive, as it involves optimizing many parameters. However, it can significantly improve the accuracy of the estimated camera poses and the locations of the 3D points and reduce the accumulated error over time. Figure 2-19

[59] presents an example where bundle adjustment could be used. The 3D model is assumed to be stationary.

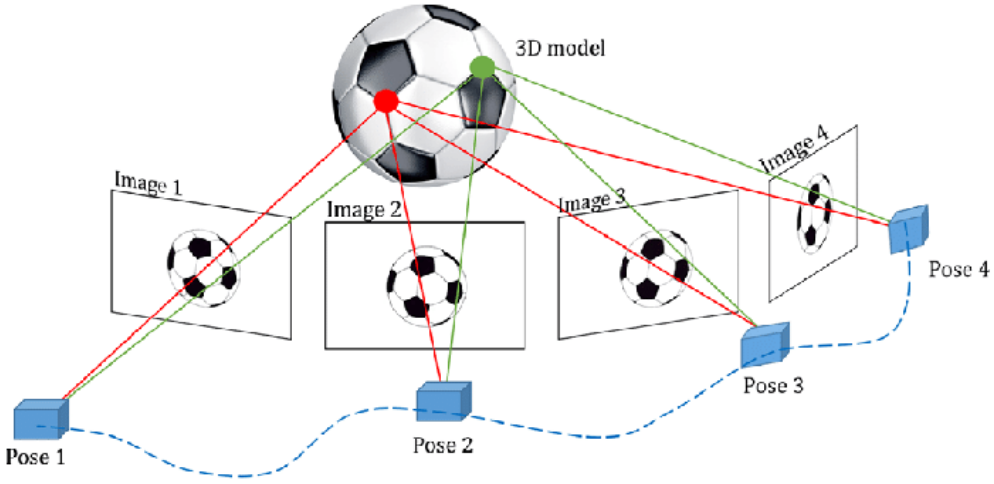


Figure 2-19. Bundle Adjustment Example [59].

The objective of Bundle Adjustment is to improve the accuracy of initial camera pose and locations of 3D points estimates, by refining them in a simultaneous optimization process, in order to find the optimal set of parameters that best predict the locations of 3D points in a set of images. Let there be n 3D points visible in m views, with $\boldsymbol{x_{ij}}$ representing the projection (coordinates in the image plane) of the 3D point i on image j. The v_{ij} denotes a binary variable that is equal to 1 if point i is visible in image j, and 0 otherwise. Each camera pose j is represented by a vector $\boldsymbol{a_j}$, and each 3D point i by a vector $\boldsymbol{b_i}$. Bundle Adjustment aims to minimize the total reprojection error by optimizing all locations of the 3D points and camera poses as presented in Equation (2.11).

$$min_{\boldsymbol{a_j},\boldsymbol{b_i}} \sum_{i=1}^{n} \sum_{j=1}^{m} u_{ij} d\left(Q\left(\boldsymbol{a_j}, \boldsymbol{b_i}\right), \boldsymbol{x_{ij}}\right)^2 \tag{2.11}$$

where $Q\left(\boldsymbol{a_j}, \boldsymbol{b_i}\right)$ is the predicted projection of point i on image j and $d(Q(a_j, b_i), x_{ij})$ denotes the Euclidean distance between the image points represented by vectors $Q(a_j, b_i)$ and x_{ij} [60].

2.4.1 Motion Only BA

To estimate the current pose of the camera, a specific case of bundle adjustment can be used known as Motion Only bundle adjustment. This involves optimizing the camera poses while the locations of the 3D points are assumed to be known and fixed in the 3D scene. By doing so, Motion Only BA has reduced computational cost compared to bundle adjustment, since the number of parameters to be optimized are significantly reduced. This makes it possible to perform the optimization in real-time, which is especially useful in applications where fast and accurate camera pose estimation is required, such as in visual SLAM applications.

2.4.2 Keyframes

Due to the high computational cost of BA, performing it for every new camera pose (with each new image) would be impractical. As a result, it is performed only on specific frames (images), i.e., keyframes. Keyframes (Figure 2-20 [61]) were first introduced by Klein and Murray in 2011

in their Parallel Tracking and Mapping for Small Augmented Reality (AR) Workspaces [13] and they are significant frames in a camera trajectory, that are selected based on certain criteria,

- Fast camera movement: To optimize the camera poses at that particular point.
- Number of features currently tracked: To add new features to track.
- Elapsed time from the last keyframe insertion: To maintain a balance between computational efficiency and capturing important changes in the environment.

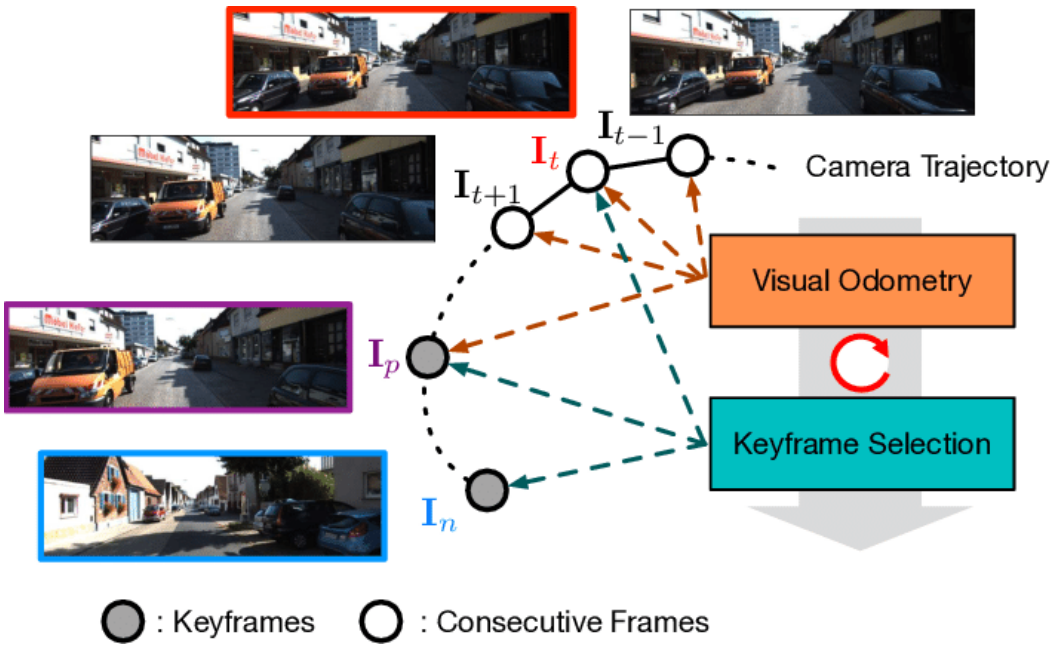


Figure 2-20. Keyframe Selection [61].

Keyframes are an essential component for visual SLAM systems because they can be used to perform BA and reduce the error accumulation over time, while reducing computational cost as BA is only performed when a new keyframe is inserted. To estimate camera poses that are not refined by BA, a reference pose is required, and keyframes are used as that reference. The unoptimized camera poses are then calculated relative to the reference pose of the corresponding keyframe.

Overall, keyframes play a crucial role in visual SLAM systems by enabling the reduction of the computational cost of BA and by providing a compact representation of the camera trajectory.

2.4.3 Local BA

To maintain local consistency in the map of visual SLAM systems, The Local Bundle Adjustment (BA) is often employed. The Local BA is a variant of the classic BA algorithm that focuses on optimizing a subset of camera poses and 3D point locations, as opposed to optimizing all of them. This approach allows Local BA to be used more frequently and with less computational power (as only a subset of camera poses is optimized) compared to the original BA, while still preserving a reasonable level of accuracy.

The process typically starts by identifying the keyframes that have visibility of, or share, 3D points that the newly added keyframe can observe. Keyframes that share 3D points with the new keyframe are considered local, or active, and their poses can be optimized. Keyframes that share 3D points with the local keyframes but not with a new one, are not

considered local, and therefore their poses are fixed. A similar process is repeated for the 3D points. If the 3D points are visible from the new or the local keyframes, their locations are optimized, while if the 3D points are visible only from the fixed keyframes, their locations are fixed and are not optimized.

Local BA is often used in real-time applications where quick and accurate estimation of camera pose is crucial. By limiting the number of parameters used in the optimization process, Local BA can still operate in real-time, while still achieving reasonable accuracy. Figure 2-21 [62] presents visually the identified keyframes and 3D points in a Local BA problem.

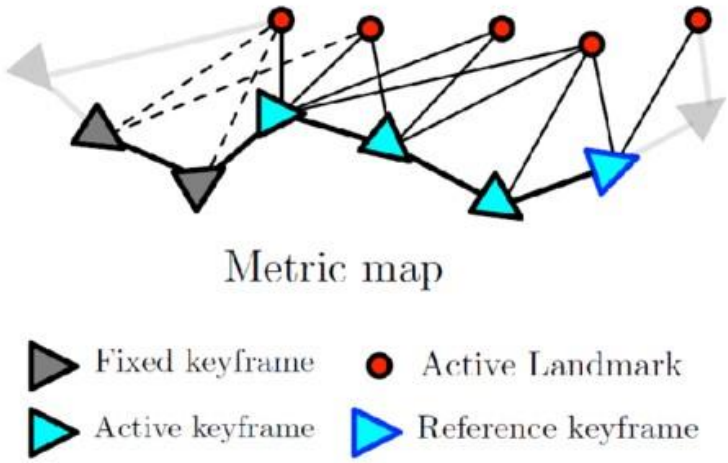


Figure 2-21. Identified Keyframes and 3D Points for Local BA [62].

2.4.4 Global BA

Global Bundle Adjustment involves optimizing all the keyframe poses and 3D point locations in the map. This process is typically performed to reduce error accumulation over a long period of time in a camera trajectory. However, due to the large number of parameters that require optimization, global BA is not performed frequently, but rather only when new information is discovered, such as when the camera revisits a previously seen location or when new sensor data becomes available. In other words, global BA is initiated by an external event, rather than from the feature tracking process itself. To ensure a fixed map during global BA optimization, the first keyframe - usually the first frame of a camera trajectory - is fixed and cannot be optimized.

2.5 Loop Closure

In the context of visual SLAM, loop closure is the process of recognizing that the camera has returned to a previously visited location and then, using this information, correct accumulated errors in the camera trajectory and the estimated map of the environment.

When the camera revisits a location, it captures new images of the scene, which can be compared with the images captured during the previous visit. If the camera has returned to the same location, these images will have similar features and descriptors, and the camera

poses estimated using these images will be close to each other. The process of recognizing that the camera has revisited a previous location is called loop closure detection.

The most common loop closure detection method is the bag-of-visual-words approach (BOVW) which represents features, such as ORB or SURF, as a histogram of visual words. Each visual word corresponds to a cluster of similar features (Figure 2-22 [63]). This representation is then used to compare with all the previous images (already represented in a bag-of-visual-words) and determine if they are from the same location. The most commonly used package for a bag-of-words representation is DBoW2 [10] used by many well-known visual SLAM algorithms, e.g., ORB-SLAM3 [7], VINS-Fusion [11] and Kimera-VIO [14] to name a few.

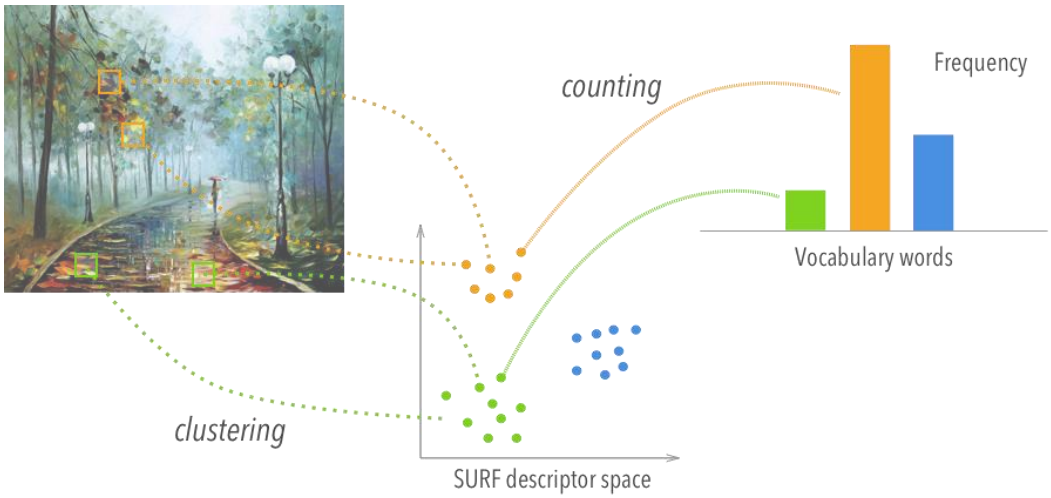


Figure 2-22. Bag-of-Words Representation of Image Features [63].

Once a loop closure is detected, global BA can be performed on the estimated camera poses and 3D point locations to correct any errors that accumulated due to noise or drift during the camera trajectory estimation. Figure 2-23(a) presents the trajectory of ORB-SLAM (blue) compared to the ground truth (red) trajectory with no loop closure optimization and Figure 2-23(b) presents the trajectory of ORB-SLAM (blue) with loop closure optimization compared to the same ground truth trajectory. From Table 2-1 it is evident that the trajectory has improved in accuracy due to the optimization process.

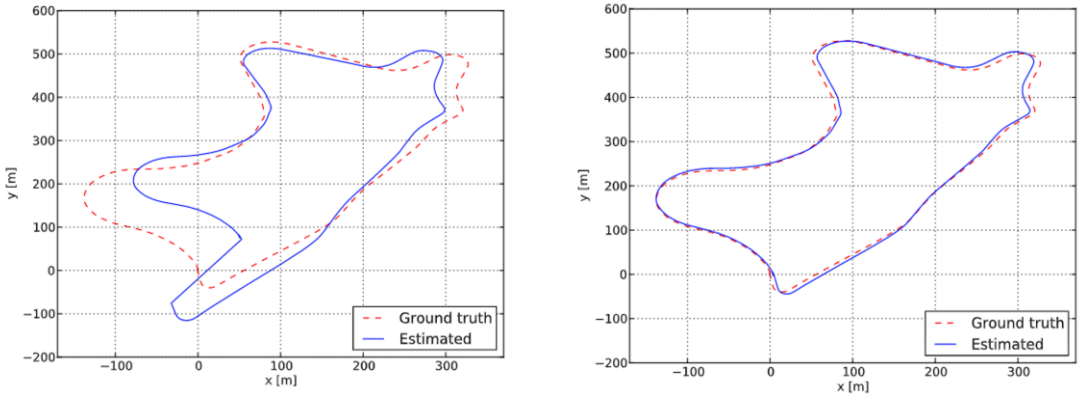


Figure 2-23. Loop Closure Optimization, Ground Truth (Red), (a)ORB-SLAM (Blue) With No Loop Closure Optimization, (b) ORB-SLAM (Blue) With Loop Closure Optimization.[8]

Table 2-1 RMSE Comparison of Estimated Trajectory With and Without Loop Closure Optimization [8].

	RMSE
ORBSLAM No Loop Closure	7.62
ORBSLAM With Loop Closure	6.62

Loop closure is a crucial part of visual SLAM systems, as it helps mitigate the effects of error accumulation and aids in the creation of an accurate.

2.6 AprilTags

Apriltags were firstly developed by Wang and Olson at the University of Michigan in 2011 [64] [65] [66]. They are a type of visual marker designed to be easily detected and recognized by computer vision systems and are commonly used in robotics, augmented reality, and other computer vision applications where precise localization and tracking of objects or devices is required. They are similar in concept to QR codes, as they are a type of two-dimensional bar code. However, they are specifically engineered to encode smaller data payloads (ranging from 4 to 12 bits) to enable more reliable and longer-range detection. The unique pattern of each Apriltag allows for its accurate and robust detection, even in challenging lighting conditions. They are typically detected and tracked using specialized software libraries, such as the AprilTag library [67] and when detected, provide the following information:

- The identity of the tag: A unique ID number encoded within its pattern, which can be used to identify the specific tag being detected.
- The position of the tag: Its precise 3D position relative to the camera that detected it.
- The orientation of the tag: Its orientation relative to the camera that detected it.

In Figure 2-24 [68], a set of various sized Apriltags is presented.

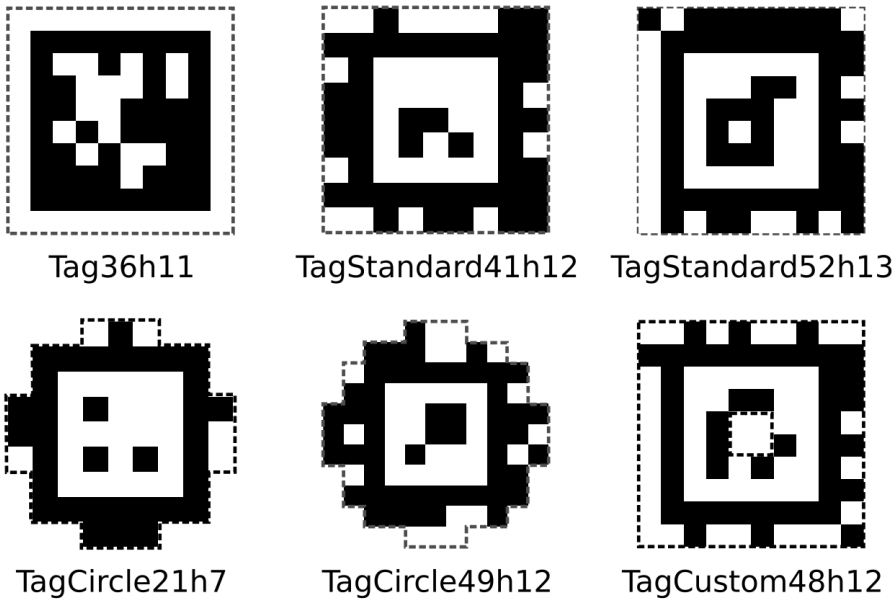


Figure 2-24. Different Apriltags [68].

To accurately compute the 3D position and orientation of the tag, the intrinsic parameters of the camera and the size of the tag need to be available. Tags are separated in classes and the tag size is represented differently in different classes. Figure 2-25 [69] presents 2 different tags with different size representations.

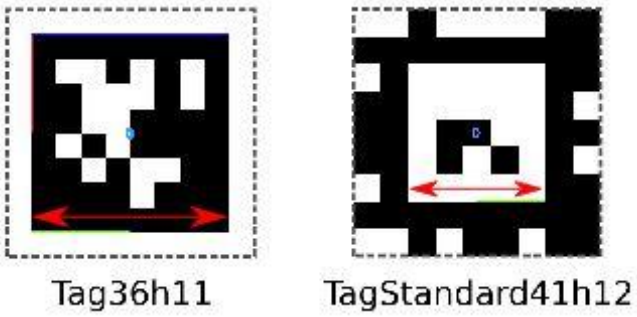


Figure 2-25. Tag Size Representation [69].

Furthermore, the camera pose can be estimated relative to the Apriltag, and if the tag's world pose is known, then the world pose of the camera can be estimated with high accuracy. Moreover, Apriltags can be used in combination, allowing for the tracking of multiple tags simultaneously. This enables even greater accuracy in camera pose estimation, as presented in Figure 2-26 [70], where the position and orientation of multiple tags are detected. This makes Apriltags a valuable tool for many applications in robotics, augmented reality, and computer vision, where precise camera localization is essential.

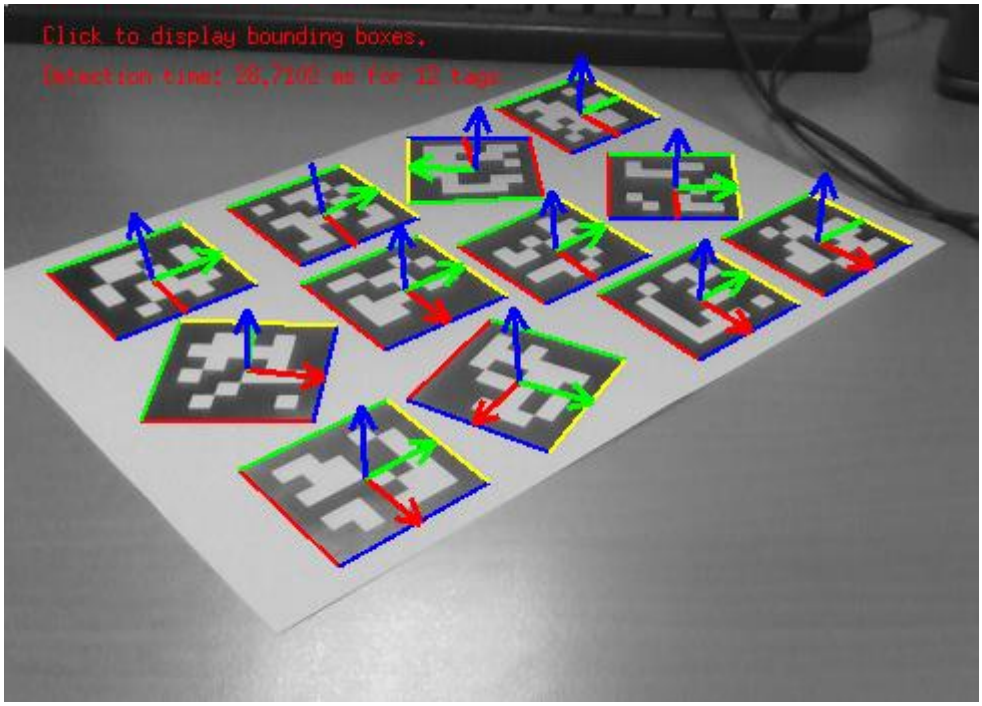


Figure 2-26. Multiple Apriltags Detected [70].

3 Implementation of the Visual SLAM Algorithm

This chapter provides a detailed overview of the visual SLAM algorithm developed for this thesis. To overcome challenges posed by environmental factors such as obstructions in the camera's field of view (FoV), a dual stereo camera setup was used. Stereos cameras were placed at the back front and the front of rover as presented in Figure 3-1, increasing the overall FoV while at the same time addressing issues such as featureless objects covering most of the FoV of one of the stereo cameras.

Additionally, Apriltag detection was chosen for loop detection, to address situations where the environment is homogeneous, such as in vineyards, where algorithms using bag-of-words approaches, e.g., ORB-SLAM3, Kimera-VIO, etc. could potentially result in incorrect loop detection.

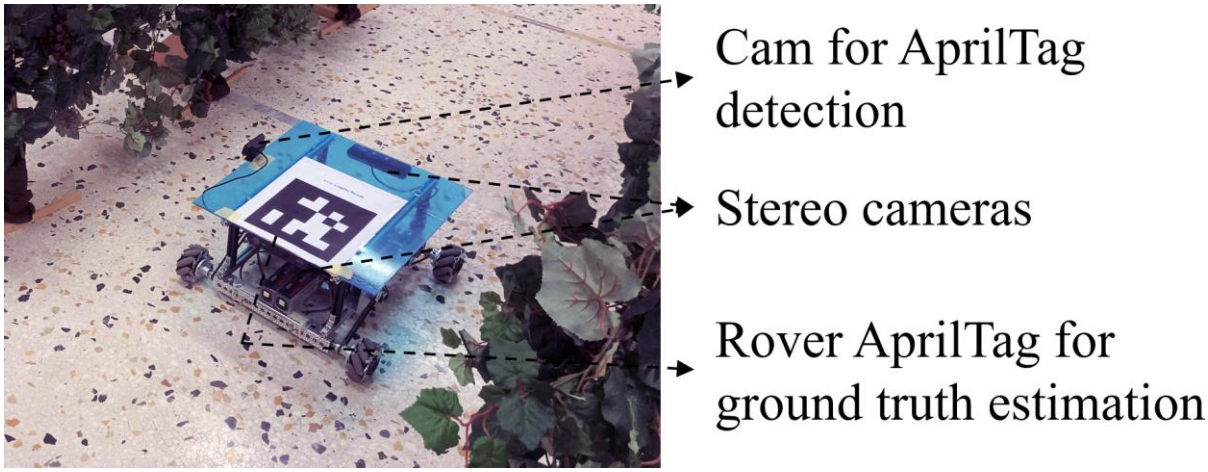


Figure 3-1. CSL Rover.

3.1 Visual SLAM Pipeline

The system is separated into five different threads,

1. Tracking: Tracks features in images and estimates the current camera pose.
2. Local mapping: Inserts new 3D points and performs local BA optimization.
3. Loop closing: Performs Global BA optimization when a loop is detected.
4. AprilTag detection: Detects Apriltags and initiates the loop closure thread.
5. Visualization: Visualizes the 3D points and keyframes.

Initialization is performed once during the system startup in the tracking thread. The map is initialized by extracting features from both stereo cameras (front and back) and by creating 3D points using stereo matching. Next, the tracking thread extracts features from each new camera frame and matches them with the 3D points. In turn, the camera pose is estimated by minimizing the reprojection error of the matches with Motion-Only BA using the Ceres Solver [71]. When a keyframe is inserted, the local mapping thread searches for new 3D points between the newly added keyframe and its connected keyframes and launches the process of the local BA. Local BA is applied to achieve local consistency of the camera poses and refine the estimation of the 3D point positions. Lastly, AprilTag detection aids in identifying any loops, and if one is detected, the loop closing thread is initialized. The loop closing thread performs a global BA when an AprilTag is detected for the second time (meaning that the

rover passed from this AprilTag in the past). The visualization thread visualizes the trajectory of the camera, with its keyframes, along with all the 3D points mapped. The flowchart of the developed visual SLAM pipeline is presented in Figure 3-2. The diagram was developed using [72].

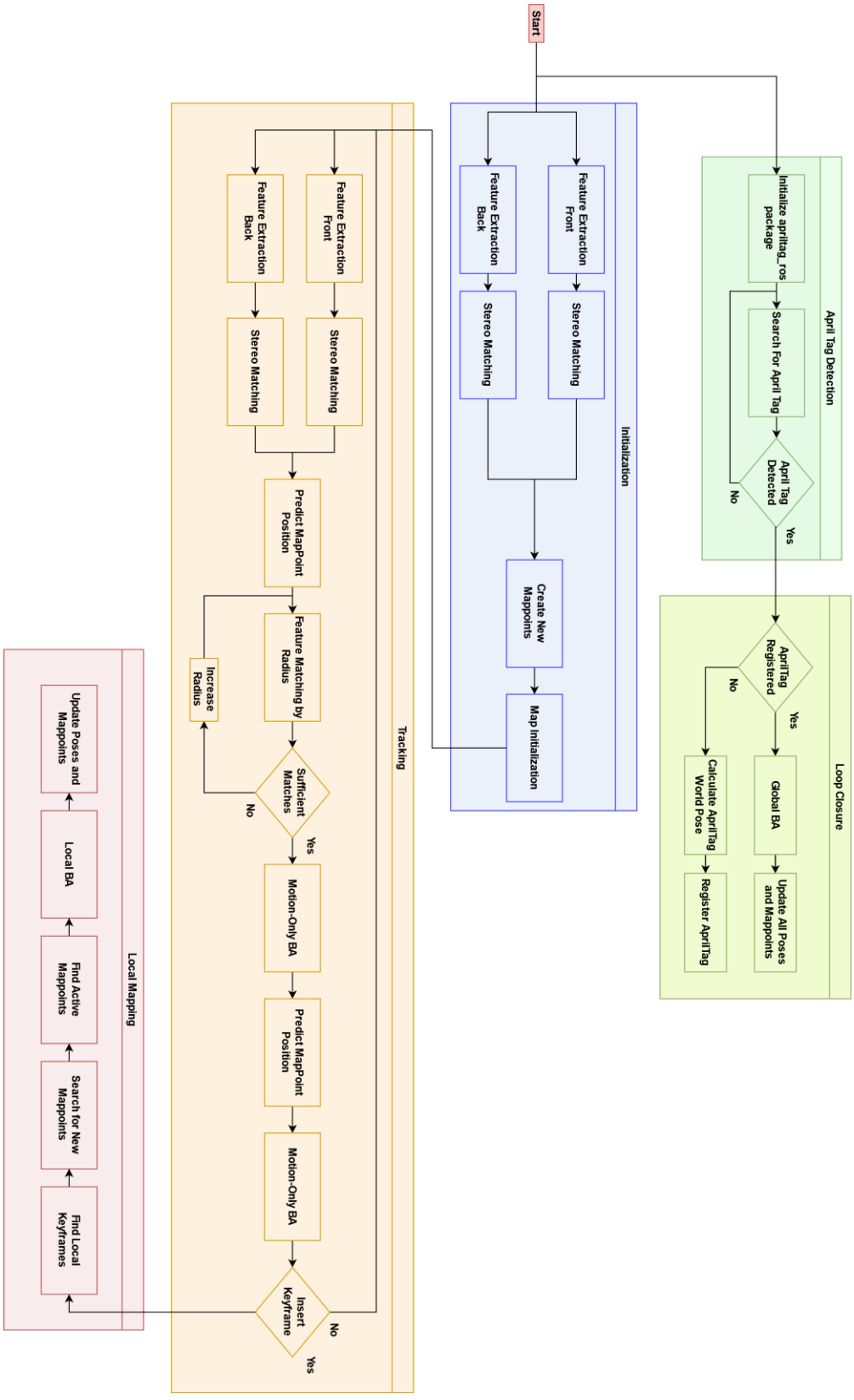


Figure 3-2. The Developed visual SLAM Pipeline.

3.2 Application on mobile robots using ROS

The developed visual SLAM software package can operate in two modes. In the first mode, images from a folder are loaded one by one for testing in well-known datasets, such as KITTI or EuRoC. The second mode is a real time operation, that while a camera is moving in real time, its trajectory is estimated. This real-time operation is performed using the Publish/Subscribe Communication Model that ROS [73] offers.

The Robot Operating System (ROS) is an open-source framework that helps researchers and developers build and reuse code between robotics applications. A Publisher publishes messages of some standard Message Type to a particular topic. The Subscriber on the other hand subscribes to the topic so that it receives the messages whenever any message is published to the topic [74, p. 3]. This communication model is presented in Figure 3-3 [75].

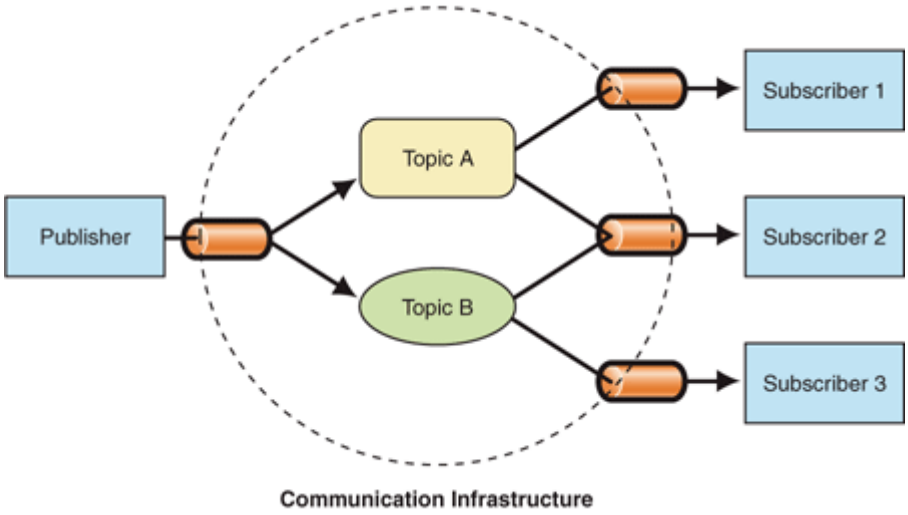


Figure 3-3. Publish/Subscribe Communication Model [75].

By using ROS, this communication model can be utilized, that allows image messages to be shared between different processes. This way, any process that needs the image information can subscribe to a certain topic and receive the image published in that topic.

ROS also provides the rosbag package [76], which is a set of tools for recording and playing back ROS topics. It is a high-performance experiment replay tool that avoids deserialization and reserialization of the messages. A certain camera trajectory can be recorded in a rosbag file. That rosbag can be replayed so that the same recorder experiment can be used for testing and evaluation against other visual SLAM algorithms.

3.3 Initialization

The visual SLAM system is initialized by starting all the required threads. A configuration file is loaded providing the following information:

- Rectified: If the images are rectified.
- Distortion Parameters: If the images are not rectified, the distortion parameters are needed to rectify the images.
- Stereo Camera Parameters: The camera parameters for each stereo camera, e.g., baseline.
- Intrinsic Parameters: The intrinsic parameters for each camera.
- ORB Parameters: The ORB parameters chosen for each test dataset.

The normal values for each ORB parameter are presented in Table 3-1.

Table 3-1 Normal Values for each ORB parameter.

Parameter	Value
$nFeatures$	500
$nLevels$	8
$scaleFactor$	1.2
$edgeThreshold$	31
$FastThreshold$	20
$patchSize$	31

The first step is the initialization of the map, which involves adding the first 3D points. The world origin of the map was chosen as the first pose of the camera. One advantage of using stereo cameras is that the system can be initialized with just one frame, as stereo cameras provide depth information. The tracking thread rectifies all images, if needed, extracts features from both stereo cameras, and performs stereo matching.

3.3.1 Stereo Matching

Despite OpenCV providing functions for feature matching, as described in Section 2.2.2, a custom stereo matching function was developed from scratch to allow each feature to be compared with fewer features[1], reducing the computational power needed.

Epipolar Constraint

The stereo matching process begins after features from all frames have been extracted and assigned to a grid. As described in Section 2.1.3, each pixel on the left image has the same y-value as the corresponding pixel on the right image, called Epipolar Constraint. For this reason, the features from the right camera lenses are grouped based on their y-coordinate for quicker matching with their corresponding feature from the left camera. This means that each left feature is only compared to a subset of features in the right frame. To further optimize the matching process, the left features are compared only with the features from the right frame that have been extracted from similar (either the same pyramid level or one pyramid level difference) image pyramid levels, and that have a lower x-coordinate. The descriptor of the left feature is then compared to the descriptors of the right feature, and the two lowest distances are recorded. These two distances are used for the ratio test, discussed in Section 2.2.2, where the threshold chosen was 0.75, which is a commonly used threshold value [49].

Sliding Window Search with Parabola Fitting

Once the correct feature has been identified, a sliding window search is performed to achieve sub-pixel accuracy for each match. A 5x5 patch is centered around the left feature and compared with a 5x5 patch around the right feature. The patch around the right feature is moved from -5 to +5 of the x value of the feature, recording the distances between the patches.

[1] Reduce the number of potential correct matches by eliminating features that are definitely incorrect matches.

These distances are then used for parabola fitting to calculate the subpixel accuracy of each stereo match.

Specifically, parabola fitting begins with Equation (3.1).

$$y(x) = ax^2 + bx + c \tag{3.1}$$

Where y is the distance calculated from the sliding window search, and x denotes the position of the window. Parabola fitting aims to find to lowest y value, the lowest distance between the two patches, for an x value (x_{min}), which is the subpixel accurate match of the left feature. As the x value of the window is shifting by one pixel each time, the lowest distance (d_0) is selected representing the best pixel accurate x value (x_0) for the right feature. Parabola fitting is performed using the distances around the best value of x (x_0), namely -1, 0, and +1, where 0 represents the best value of x. Figure 3-4 presents the patch around the left feature that is compared with patches on the right image. Figure 3-5 shows the different patches around the right feature. x_0 is used as the starting point ($x_0 = 0$) as it is the closest one to the sub pixel accurate match. x_{-1} is the patch 1 pixel to the left and x_1 is the patch 1 pixel to the right.

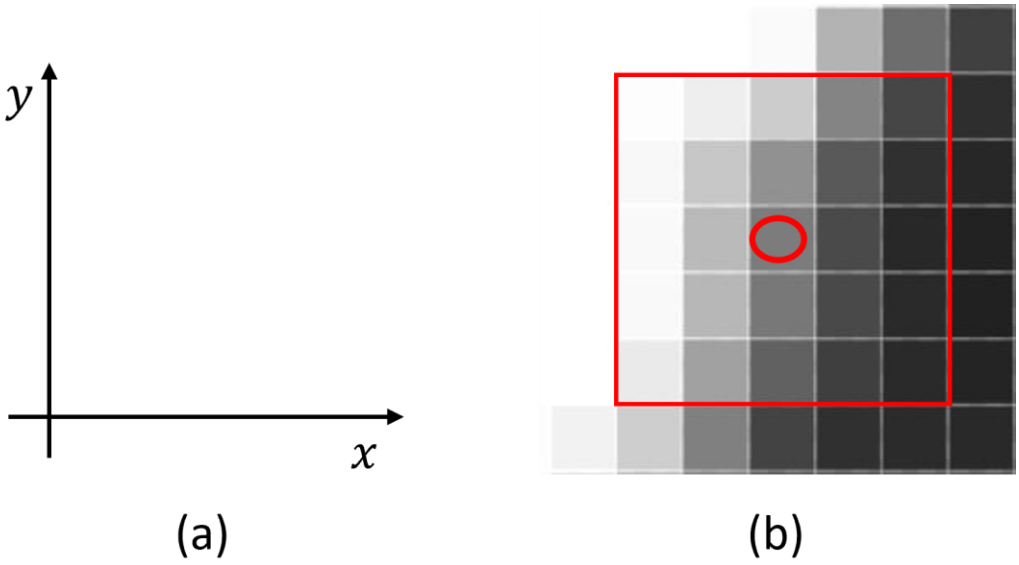


Figure 3-4 (a) Axes of image patches (b) Patch Around Left Feature. The red circle represents the feature, while the red square represents the 5x5 patch around the feature.

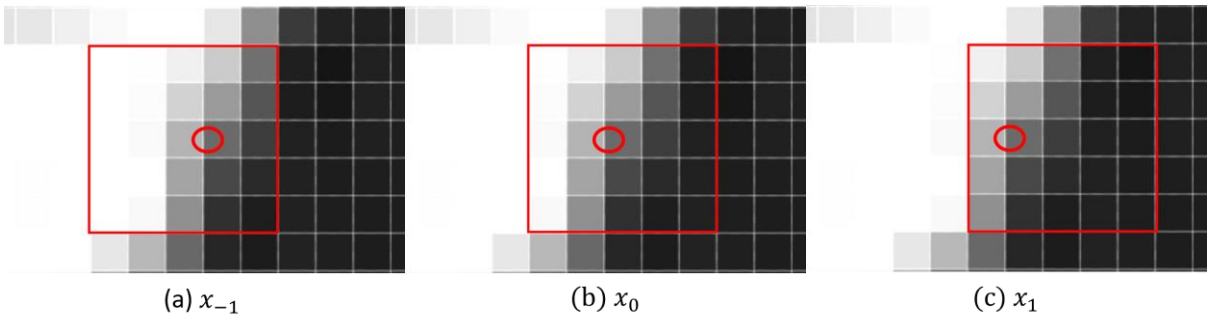


Figure 3-5. Right image patches.

Figure 3-6 presents the parabola fitting. Distance d_0 is the lowest distance found, and d_{-1}, d_1 are the distances found 1 pixel to the left and one pixel to the right respectively. The distances

are calculated comparing Figure 3-4 with Figure 3-5(a) for d_1, Figure 3-5(b) for d_0 and Figure 3-5(c) for d_1. With parabola fitting the x_{min} (subpixel accurate x value) can be calculated.

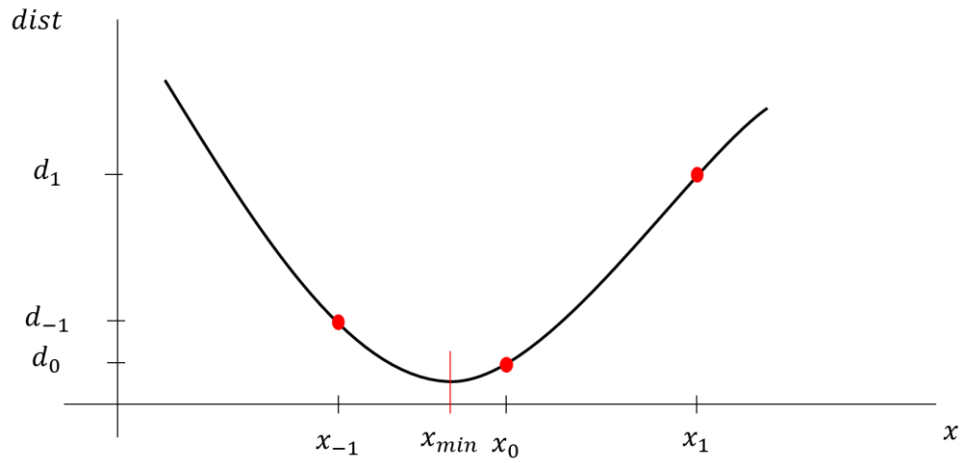


Figure 3-6. Parabola Fitting.

Equations (3.2)-(3.4) present the calculation of the subpixel accurate x value of the right feature.

$$y(-1) = d_{-1} = a - b + c \tag{3.2}$$

$$y(0) = d_0 = c \tag{3.3}$$

$$y(1) = d_1 = a + b + c \tag{3.4}$$

The values d_{-1}, d_0, d_1 are the distances that were previously calculated from the sliding window search. With the distances known the variables a, b, c can be calculated using Equations (3.5)-(3.7).

$$a = \frac{d_{-1} + d_1 - 2 * d_2}{2} \tag{3.5}$$

$$b = \frac{d_1 - d_{-1}}{2} \tag{3.6}$$

$$c = d_0 \tag{3.7}$$

With the variables a, b, c known, the minimum can be found for the Equation 3.1 using the derivative, which is presented in Equation (3.8).

$$x = best = -\frac{b}{2 * a} \tag{3.8}$$

Equation (3.8) with the use of Equations (3.5)-(3.7) can calculate the subpixel accurate x coordinate of the stereo match, shown in Equation (3.9).

$$y(best) = \frac{d_{-1} - d_1}{2 * (d_{-1} + d_1 - 2 * d_0)} \tag{3.9}$$

With Equations (3.1)-(3.9), subpixel accuracy on the x coordinate can be calculated for each left feature as presented in Equation (3.10).

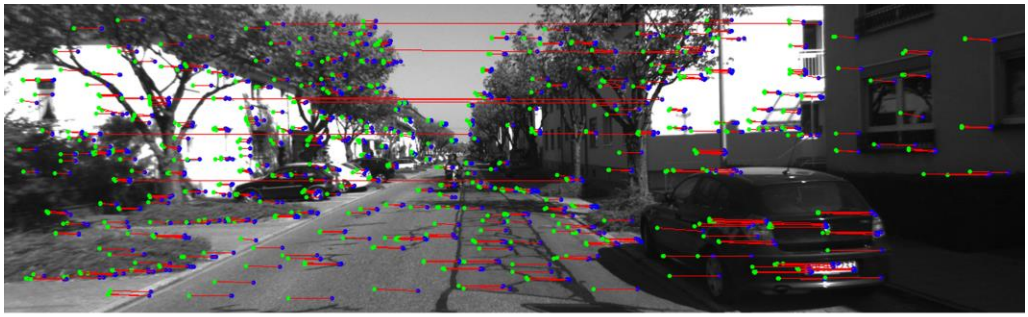
$$x_{subpixel} = x_{best} + y(best) \tag{3.10}$$

where x_{best} is the best x coordinate (so $x_0 = x_{best}$) found from the sliding window search and $x_{subpixel}$ denotes the subpixel accurate x coordinate of the matched right feature. After each stereo match is identified, the depth for each match is calculated and stored, as previously described in Section 2.1.5 using Equation (2.3).

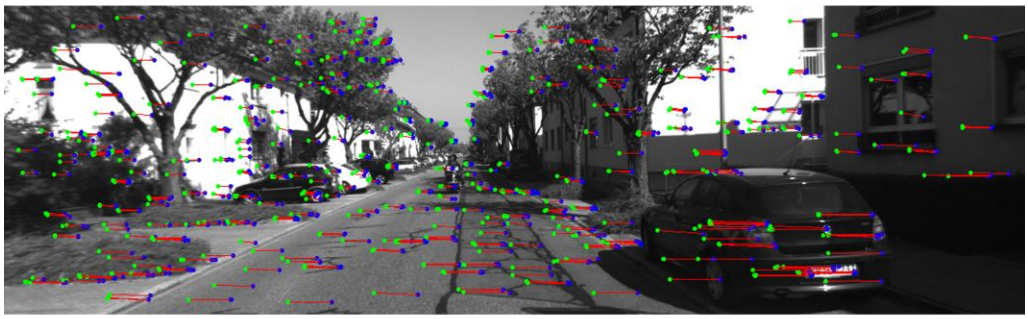
Match Filtering

To ensure the selection of correct matches, match filtering is carried out by storing all best distances obtained from the sliding window search and calculating their median value. Stereo matches with distances exceeding twice the median of the distances are considered invalid and are removed from the final set of matches. This filtering method eliminates matches with unusually high distances compared to the rest of the matches, thus improving the overall accuracy of the matches.

After many iterations of the stereo matching process, many stereo matches with the lowest depth value were found to be incorrect, so a final filtering step was applied to the stereo matches by removing the closest 1% of matches. Figure 3-7(a)-(b) presents a comparison of the stereo matches with and without applying these filtering methods. In Figure 3-7, the green circles represent the features from the right image, the blue circles represent the features from the left image, and the line connecting each one represents the match between them. The longer the red line the closer the point is to the camera lenses. For example, the car in Figure 3-7 is closer than the trees, and consequently, the red lines connecting the points on the car are longer. In Figure 3-7(a), it can be observed that a number of close matches are incorrect, as indicated by the long red lines connecting improperly matched features. On the other hand, Figure 3-7(b) has eliminated these incorrect matches while retaining most of the correct ones. In SLAM systems with large numbers of stereo matches, it is preferable to remove incorrect matches, even if it means some correct matches are also removed.



(a) Matches with No Match Filtering



(b) Matches with Match Filtering

Figure 3-7. Difference of Correct Matches with the Match Filtering techniques. Green circles represent the features extracted from the right image while the blue circles represent the features extracted from the left image.

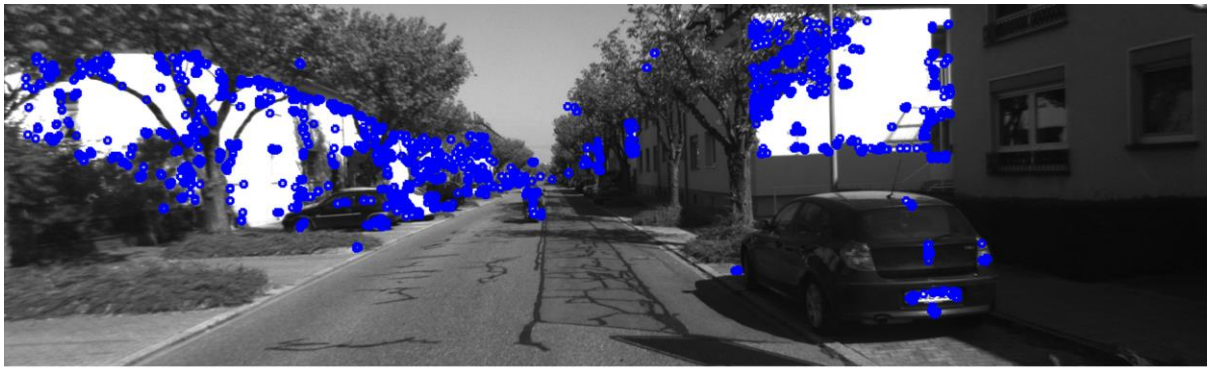
3.3.2 Feature Extraction Process

Feature extraction for visual SLAM applications is an important step towards providing accurate and robust localization and mapping. In this thesis, ORB Features [43] were selected, due to (a) their scale and rotation invariance properties, and (b) the fact that their extraction requires low processing power and as a result they are suitable for real-time SLAM applications. The optimal ORB parameters were chosen for each tested dataset. Although OpenCV offers a function to extract ORB features [77], the extraction algorithm was implemented from scratch to ensure a homogenous distribution of the features by separating each image into grids and then proceed to the features process on each grid separately.

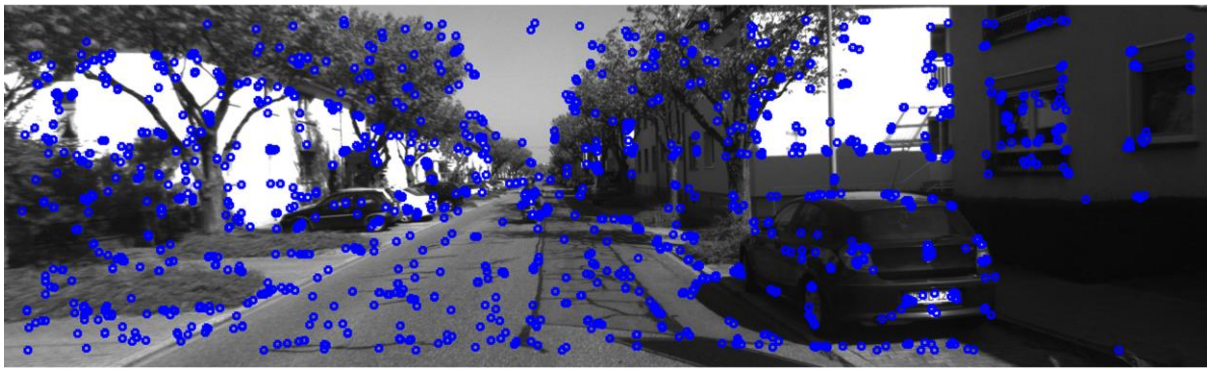
As described in Section 2.2.1, to extract ORB features, FAST features are extracted on each image in an image pyramid, to provide scale invariance, and then an orientation component is calculated for each feature, to provide rotation invariance. Specifically, the image pyramid was created by down sampling each image a certain number of times based on the number of pyramid levels chosen (ORB parameter: nlevel), using the image scale specified by the scale factor (ORB parameter: scaleFactor). For a homogeneous distribution of features across the entire camera frame, each image in the image pyramid was separated into grids, and FAST features were extracted on each grid separately, using the FAST algorithm provided by OpenCV [78]. Each image in the image pyramid was separated into a different number of grids. The number of grids per image is adjusted according to the frame resolution. Additionally, to extract features even on less-textured image grids, adaptive FAST thresholding was used. Specifically, the value of the fastThreshold parameter was decreased one time, if no features were detected in a grid of the image, making the feature detection more robust. Lastly, suppression via Square Covering (SSC) [79] ensured the selection of the strongest (The strongest features are selected based on the response parameter, which demonstrates the level of certainty that the selected feature is indeed a feature) features, while maintaining homogeneity. SSC requires input features to be sorted in decreasing order of strength. Then the features are processed in that order and any feature located within a predefined range, referred to as the suppression range, of a stronger feature is removed. The process is repeated for all the features, and if the resulting number of features significantly deviates from the desired number, the suppression range is adjusted, and the process is repeated.

Figure 3-8 (a) illustrates the feature distribution when grids and SSC is not used, while Figure 3-8 (b) and Figure 3-8 (c) illustrate the variation when grids are applied and when grids & SSC are applied, respectively. Both Figure 3-8 (b) and Figure 3-8 (c) demonstrate an increase in feature distribution across the entire image, indicating the importance of these techniques.

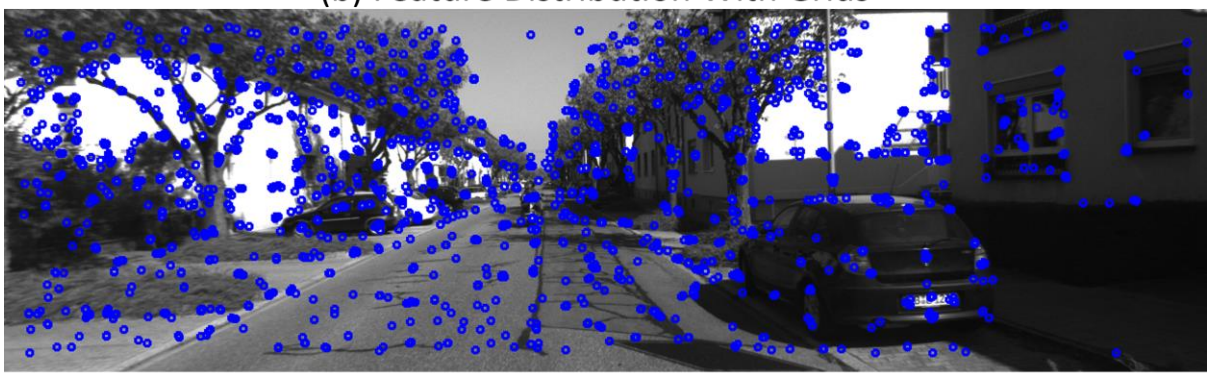
After the best features are selected, the orientation is calculated. The moments of a patch around each feature need to be calculated. A circular patch is created around the x, y position of each feature. The first and second moments of the image intensity are calculated in the circular patch. The first moment provides information about the distribution of pixel intensities along the vertical axis of the patch, while the second moment provides the same information along the horizontal axis of the patch. This information is calculated by the intensity of each pixel along the rows and columns of the patch. After the moments have been calculated, the orientation is computed using the arctangent function using Equation (2.6).



(a) No Feature Distribution



(b) Feature Distribution With Grids



(c) Feature Distribution With Grids + SSC

Figure 3-8. Feature Distribution with Different Methods.

After the calculation of the orientation of each extracted feature, it is assigned to a grid for accelerated matching. Lastly, the rotated BRIEF descriptor is calculated using the OpenCV function for ORB feature extraction.

With features extracted from both stereo cameras and their corresponding descriptors calculated, stereo matching is initialized.

3.3.3 Mappoints

In the initialization step, after the stereo matches have been identified, all the 3D calculated points are added to the map, creating mappoints. The 3D points created from stereo matches using Equations (2.7)-(2.9), have the left camera optical center as their world origin (they are created with the left camera optical center as their 0,0,0, and as a result the 3D points are not defined in their world coordinates). To calculate their world coordinates a transformation is

calculated from the pose of the camera that observes them, front or back, to the map's origin (the map's center 0,0,0) as presented in Equations (3.11)-(3.12).

$$\boldsymbol{pw} = \boldsymbol{T_{wc}} * \boldsymbol{pc} \tag{3.11}$$

$$\boldsymbol{pw} = \boldsymbol{T_{wcb}} * \boldsymbol{pc} \tag{3.12}$$

$\boldsymbol{P_w}$ is a 4-by-1 matrix that denotes the world position in homogeneous coordinates, $\boldsymbol{P_c}$ is the position of the mappoint with the left camera optical center as their world origin. $\boldsymbol{T_{wc}}\boldsymbol{T_{wcb}}$ are 4-by-4 matrices that denote the transformations from the map's origin to the front camera or the back camera, respectively.

In the developed algorithm, Mappoints were chosen to store the following information for faster access:

- World Position: the world position of the mappoint.
- Pyramid level: the latest image pyramid level on which each mappoint was detected.
- Matched Keyframes and Features: A list of all the keyframes and matching features with which each mappoint is matched. This is used for bundle adjustment.
- In frame: Indicates whether the mappoint is present in the current frame or not. If the mappoint is present in the current frame, it is considered active.
- Predicted position: an estimation of where the mappoint is likely to be visible in the next frame.
- Predicted position in right camera lens: an estimation of where the mappoint is likely to be visible in the next frame on the right camera lens.
- Predicted level: at which pyramid level each mappoint is predicted to be found in the next frame.
- Descriptor: the descriptor of each mappoint for matching.
- Reference Camera: Which camera can view the mappoint, front or back.
- Outlier: if each mappoint is an outlier and should not be taken into consideration in the camera pose estimation.

3.3.4 Calculation of the Mappoint Descriptor

To accurately match newly extracted features with the mappoints, a more robust descriptor is calculated for each mappoint. As explained in Section 3.2.2, mappoints store information regarding all the features with which they have been matched with. Using this information, the descriptors of all the matched features are compared to each other. A distance between each descriptor is calculated and the one with the least distance to all others is selected as the descriptor of the mappoint. This way, the mappoint is described by the closest descriptor to all the features. When the mappoint is updated, either by adding a new keyframe to the list of matches with the mappoint, or removing a keyframe from the list, the descriptor is recalculated. This creates a more robust description of the mappoint resulting in a more accurate matching process.

3.4 Tracking Thread

The tracking thread starts the camera pose estimation process after the initialization of the map. For each new frame, features are extracted and stereo matched. Subsequently, a new matching process is created, named matching by projection, which estimates the next position of the active mappoints (the mappoints that can be observed in the image) and projects them

onto the current image plane. The projected mappoints are then matched with the current features.

3.4.1 Mappoints Position Prediction

The prediction of the position of the mappoint in the current pose is calculated using the constant velocity model (assumption that the camera has a constant velocity, and its pose is predicted using that velocity). The acceleration or rotation is taken in account the second time the camera pose is estimated described in Section 3.4.4. All poses are represented in a $4x4$ transformation matrix in the $SE(3)$ Euclidean group [80], and all mappoint positions are in homogeneous coordinates. The next camera world pose $Tpred_{wc_l}$, which denotes the transformation from world origin w to the left camera framec_l, can be estimated as presented in Equation (3.18).

$$\boldsymbol{Tpred_{wcl} = T_{wcl} * Tprev_{clw} * T_{wcl}} \tag{3.18}$$

where $\boldsymbol{T_{wcl}}$ is the transformation from world w to left camera frame cl, $\boldsymbol{Tprev_{clw}}$ denotes the inverse of the previous camera world pose, and $\boldsymbol{Tpred_{wcl}}$ is the predicted camera world pose assuming the camera maintains a constant velocity in the next frame. With the information of the world position of the mappoint, it can be moved to its predicted position in front of the camera using the inverse of $\boldsymbol{Tpred_{wcl}}$, denoted as $\boldsymbol{Tpred_{clw}}$, to transform the mappoint from world to camera coordinates. For example, suppose a mappoint is located at $(0,0,100)$ and the camera is predicted to be at $(0,0,40)$. In the camera coordinate system, the mappoint is at $(0,0,60)$. However, the only known parameter is the mappoint's world position $(0,0,100)$. Consequently, the mappoint needs to be transformed by the inverse of the camera's position, so the mappoint needs to be moved by $(0,0,-40)$. This transformation ensures that the camera observes the mappoint at the correct position. This transformation is given by:

$$\boldsymbol{pc_l = Tpred_{clw} * pw} \tag{3.19}$$

where $\boldsymbol{pc}$ is the position of the mappoint in the camera coordinates and $\boldsymbol{pw}$ is the position of the mappoint in the world coordinates.

To predict the position of the mappoint in the right camera lens, a similar approach is followed. The predicted right camera world pose is calculated using the already predicted left camera world pose as:

$$\boldsymbol{Tpred_{wcr} = Tpred_{wcl} * T_{clcr}} \tag{3.20}$$

where $\boldsymbol{T_{clcr}}$ is the transformation from the left camera coordinate frame to the right camera coordinate frame and $\boldsymbol{Tpred_{wcl}}$ denotes the predicted world pose of the right camera. To transform the mappoint from world to the right camera coordinate frame Equation (3.21) is used.

$$\boldsymbol{pcr = Tpred_{crw} * pw} \tag{3.21}$$

Concerning the mappoints created from the back stereo camera data, the predicted camera poses for that camera are similarly calculated as :

$$\boldsymbol{TpredB_{wcl} = Tpred_{wcl} * T_{clclb}} \tag{3.22}$$

$$\boldsymbol{TpredB_{wcr} = Tpred_{wcl} * T_{clcrb}} \tag{3.23}$$

where B denotes the back camera, and $\boldsymbol{T_{clclb}}, \boldsymbol{T_{clcrb}}$, denote the transformations from the front left camera frame to the back left and back right camera frames respectively. The mappoint's predicted position is calculated using the respective transformation via Equation (3.19).

3.4.2 Match by Projection

The matching by projection process starts after predicting all mappoint positions. The predicted coordinates of each mappoint are projected onto the image frame using:

$$u = \frac{f_x * pc_x}{pc_z} + c_x \tag{3.24}$$

$$v = \frac{f_y * pc_y}{pc_z} + c_y \tag{3.25}$$

where u and v represent the new pixel coordinates of the projected mappoint in the image frame, and pc_x, pc_y, pc_z, denote the X, Y and Z coordinates, respectively, of the predicted position of the mappoint in camera coordinates. f_x, f_y is the focal length in pixels and c_x, c_y is the optical center in pixels as described in Section 2.1.2.

For each projection, a list of candidate features is created based on a radius around the projected positions of the left and right camera frame. For example, if the projection of the mappoint is at $(250{,}270)$, then all current features that are within a radius around the point $(250{,}270)$ are considered candidates.

All features from the left and the right camera frames, located inside that radius, and having a similar image pyramid level, are added to the list of candidates. The radius is determined based on a predefined constant variable and the predicted image pyramid level of each mappoint, i.e.:

$$r_{map} = r_{const} * pred_{impyr} \tag{3.26}$$

where r_{map} is the calculated radius, r_{const} is the predefined constant variable, and $pred_{impyr}$ is the predicted image pyramid level of each mappoint. In turn, similarly to the stereo matching process, each mappoint's descriptor is compared with all the candidate features in the list and the two lowest distances are stored. With the two lowest distances, a ratio test is performed comparing the ratio of these distances to 0.75, which is the most common value for the ratio test. The ratio test is described in detail in Section 2.2.2. The matches that are valid are considered correct and stored for the camera pose estimation process.

During the initialization step, a fixed value of 120 pixels (empirically chosen) is used as the predefined radius (denoted as r_{const}) since the camera has not moved yet and a constant velocity model cannot be used. After the first camera pose estimation, the radius r_{const} is set to 10 pixels. In case there are less than 50 correct matches (50 were found to produce the best results), resulting from a rapid camera movement, the radius is increased, and the matching process is repeated.

3.4.3 Camera Pose Estimation Using Motion-Only BA

After more than 50 matches have been identified, the camera pose estimation process starts. As described in Section 2.4.1, for the world pose estimation of the camera Motion Only BA is performed. Matches from all camera lenses, front and back, left and right, are used to estimate the front left camera pose, and for that reason, mappoints that were observed from the front-

right or back (left/right) lenses, require a transformation to the front-left lens to be factored in the calculations.

Mappoints that have been matched with stereo matches from either front or back camera features, are used to provide scale information, which assists in the translation estimation of the camera. As described in [81], the depth value of a stereo match is considered valid, if its estimation is less than 40 times the stereo baseline[2] (namely close stereo matches), otherwise it is considered not accurate and is only used for rotation estimation (namely far stereo matches).

For example, in Figure 3-9 let's assume that the stereo baseline ($O - O'$) is 0.10m. That means that the depth of a stereo match is considered valid when depth<4.0m. If the depth is considered valid, Motion Only BA will use both the left and the right feature in the calculations. This way, the same 3D point is seen from two different angles, the left camera lens and the right camera lens, at the same time, providing scale information. If the depth is not valid, the 3D point is calculated as seen only from one angle (the left camera lens), providing no information on the translation of the camera pose.

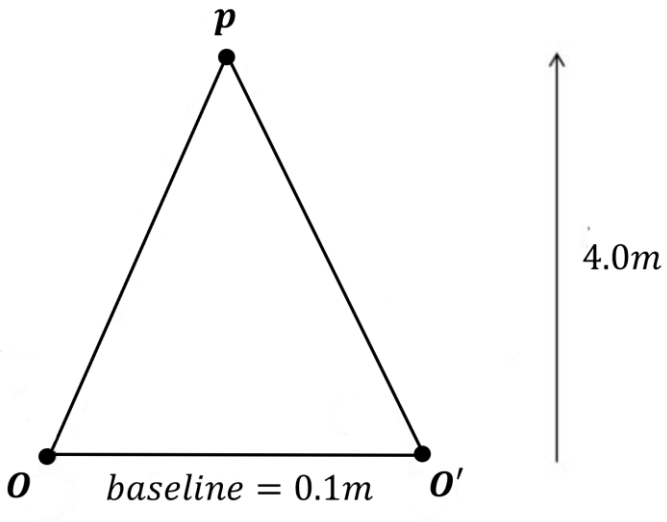


Figure 3-9. Valid Depth of 3D point.

For stereo matches with valid depth (less than 40 times the stereo baseline), to provide scale information (displacement information of the camera pose), both the left and the right feature are used in the estimation of the camera world pose, while for stereo matches with inaccurate depth only one of the features is used.

To formulate the optimization problem, the world coordinates of all mappoints are stored along with their matched features. Depending on the matched feature, a transformation is used to transform the mappoint to the front-left camera coordinate frame. If the feature is from the front-left camera frame, then this transformation is the inverse of the current world pose $\boldsymbol{T_{wcl}}$. For all the other features, depending on which camera lens observes this feature, a transformation is used as presented in Equation (3.27)-(3.29).

$$\boldsymbol{T_{crw}} = (\boldsymbol{T_{wcl}} * \boldsymbol{T_{clcr}})^{-1} \tag{3.27}$$

[2] The distance between the two lenses (left and right).

$$\boldsymbol{T_{clbw}} = (\boldsymbol{T_{wcl}} * \boldsymbol{T_{clclb}})^{-1} \quad (3.28)$$

$$\boldsymbol{T_{crbw}} = (\boldsymbol{T_{wcl}} * \boldsymbol{T_{clcrb}})^{-1} \quad (3.28)$$

Where b denotes the back camera, and l, r denote the left and right camera lenses, respectively. After the transformation to the camera coordinate frame, using Equation (3.12), the mappoints are projected in the image frame using Equations (3.17)-(3.18), where the corresponding f_x, f_y, c_x, c_y are used depending on the front or back camera parameters. The residuals used by Ceres Solver are calculated using Equations (3.30)-(3.31).

$$residual_x = weight * (observ_x - proj_x) \quad (3.30)$$

$$residual_y = weight * (observ_y - proj_y) \quad (3.31)$$

Where $observ$ denotes the matched feature position, $proj$ denotes the projection of the mappoint on the current frame, and $weight$ is a weight depending on the image pyramid level of the matched feature. Ceres Solver minimizes all the residuals (the reprojection error) in Equations (3.30)-(3.31), by estimating a rotation matrix and a translation displacement for the front-left camera frame.

To further understand the minimization of the reprojection error, Figure 2-17 is used. In Figure 2-17 suppose that the red model (named Model reprojection) was observed on the previous image. In the next image, the same model is observed at a different position (observation). The difference between the model reprojection and the observation is the reprojection error.

To minimize this error, Ceres Solver translates and rotates the camera iteratively, in different directions. In Figure 2-17 the camera would need to, for example, translate to the right so that the observed model is aligned with the red model (Model reprojection). This particular example may have multiple solutions. However, when a substantial number of 3D points are observed in different positions, the camera pose can be accurately estimated as only one correct solution exists. To estimate as accurately as possible the camera pose, it is essential to establish correct matches between the observations and the 3D points.

The Motion Only BA algorithm minimizes Equation (3.32) by finding the optimal rotation matrix ($\boldsymbol{R}$) and translation displacement ($\boldsymbol{t}$) of the front-left lens:

$$\boldsymbol{R, t} : \min_{\boldsymbol{R,t}} \sum_{i=1}^{M} \rho w_i \boldsymbol{d}(\boldsymbol{x_i}, \boldsymbol{u}(\boldsymbol{F_i}))^2 \quad (3.32)$$

$$\boldsymbol{F_i} = \boldsymbol{K}_c(\boldsymbol{R}_{CV}(\boldsymbol{Rpw}_i + \boldsymbol{t}) + \boldsymbol{t}_{CV}) \quad (3.33)$$

$$\boldsymbol{u}(\boldsymbol{F_i}) = \begin{bmatrix} \frac{\boldsymbol{F_i}(\boldsymbol{0})}{\boldsymbol{F_i}(\boldsymbol{2})} \\ \frac{\boldsymbol{F_i}(\boldsymbol{1})}{\boldsymbol{F_i}(\boldsymbol{2})} \end{bmatrix} \quad (3.34)$$

where C denotes the front or rear camera observing the mappoint, and V denotes the left or right lens. $\boldsymbol{R}_{CV}, \boldsymbol{t}_{CV}$ is the rotation matrix and translation displacement representing the transformation from the camera observing the mappoint, to the front-left lens and $\boldsymbol{K}_C$ is the intrinsics matrix for either the front or the rear camera. $\boldsymbol{F}_i$ is a 3-by-1 matrix, ρ denotes the robust Huber [82] loss function, and w_i is a weight based on the image pyramid level of each feature. M represents the total number of mappoints to be optimized, $\boldsymbol{pw}_i$ is the world coordinates of the current mappoint, and x_i is the matched feature on the image plane. $\boldsymbol{d}(\boldsymbol{x}, \boldsymbol{y})$

is the Euclidean distance between vectors x and y and they represent the residuals in Equations (3.30)-(3.31).

Once the front-left camera pose has been estimated, an outlier rejection step is performed. Each mappoint is projected onto the image frame, and the reprojection error with its matched feature is computed. If the reprojection error exceeds a certain threshold, the match is considered an outlier, and it is removed from consideration. Once all matches have been checked, the camera pose is re-estimated. After a more accurate estimation of the camera world pose has been obtained, the matches are checked once more to see if they are outliers, and if they are, they are removed.

3.4.4 Re-Estimation of Camera Pose

To estimate as accurately as possible each camera pose, the unmatched mappoints from the match by projection process are reconsidered for matching. Their position is predicted again, using the newly estimated camera pose, which is more accurate compared to the constant velocity model. This time, the predefined radius r_{const} is reduced further to a value of 4 pixels, as the camera pose is more accurately estimated (previously, the camera pose was estimated with the constant velocity model, this time the camera pose has already been estimated one time, providing a more accurate estimation). This matching process is only used once to increase the number of matches and is not repeated compared to the pose estimation, as 50 correct matches have been found from the previous pose estimation. Once the matching process finishes, the camera pose is estimated once again and all the matches are checked for rejection with the reprojection error, similarly to Section 3.3.3. This time, in contrast to the initial pose estimation, if the reprojection error exceeds the threshold, the mappoint is considered an outlier and is removed from the map.

3.4.5 Deciding on the keyframe selection

Once the camera pose estimation is complete, the next step is to determine whether to insert a new keyframe into the map. The conditions that decide whether a new keyframe is required are:

1. Number of mappoints tracked: If the number of currently tracked mappoints falls below a certain threshold which is determined based on the number of mappoints that were tracked by the last keyframe. After trial and error, this threshold was chosen as 90% of the number of mappoints tracked.
2. Frames passed: if more than 5 frames (empirically selected) have passed from the lastly added keyframe.
3. Low number of stereo matches: If the number of stereo matches providing scale information[3] to the camera estimation drops below 80 (empirically chosen).

To insert a keyframe, Condition 1 and either Condition 2 or Condition 3 have to be satisfied. Condition 1 checks whether there has been a significant decrease in the number of tracked mappoints, and if so, aims to increase them by inserting a new keyframe. Condition 2 or 3 ensures that keyframes are not inserted too frequently, which could occur in cases where the environment lacks texture, where tracking a large number of features is challenging.

After a new keyframe is inserted, the algorithm stores all features from the current frame and calculates their connections with the keyframe. Mappoints that were inserted from

[3] Stereo matches with valid depth (less than 40 times the stereo baseline) to provide scale information (displacement information of the camera pose),

previous keyframes and are now tracked in the new keyframe, are considered shared and added to the connections. Two keyframes are considered connected if they share more than 15 mappoints. In addition, the tracked mappoints are updated, the new keyframe is added to the list of matched keyframes, and the mappoint descriptors are recalculated. If Condition 3 is met, which indicates a low number of stereo matches providing scale information, the 100 closest stereo matches are added to the map.

Keyframes hold the following information:

- World Pose: The world pose of the current keyframe.
- Features: All features of the frame, including both matched and unmatched features.
- Previous Keyframe: the previous keyframe.
- Reference Pose: A reference pose that together with the pose of the previous keyframe the world pose of the current keyframe can be computed. The reference pose is used when an optimization process changes poses of previous keyframes.

Reference Poses are calculated using:

$$\boldsymbol{T_{ref}} = \boldsymbol{T_{prevKF}^{-1}} * \boldsymbol{T_{KF}} \tag{3.35}$$

where $\boldsymbol{T_{ref}}$ is the reference pose, $\boldsymbol{T_{prevKF}^{-1}}$ is the inverse pose of the previous keyframe and $\boldsymbol{T_{KF}}$ is the pose of the current keyframe.

If a keyframe is not needed, the tracking process continues with the next frame. On the other hand, if a new keyframe is required, after the keyframe insertion, the local mapping process initializes in a different thread, while the tracking process continues.

3.5 Local Mapping Thread

When a new keyframe is added to the map, the local mapping thread is initiated. All keyframes that are connected with the newly added keyframe with more than 15 mappoints are considered as local keyframes and stored. The first task of the local mapping thread is to add new mappoints to the map that are connected with at least 3 keyframes.

3.5.1 Matching Between Keyframes

To add new mappoints to the map, a matching process is carried out between the local keyframes. This allows stereo matches with inaccurate depth values, which can be observed from previous keyframes, to be included in the map as their position can be triangulated accurately from multiple views.

Firstly, all stereo matches from the newly added keyframe, close and far, that are not matched with a mappoint, are stored. These stereo matches are converted to 3D points with Equations (2.7)-(2.9), and their world position is calculated using Equations (3.11)-(3.12). The stereo matches' position and pyramid level from each camera, front or back, are the, predicted on each local keyframe for both the left and the right camera lenses of the corresponding camera. The predicted position is projected on the image plane and matched with unmatched features of each local keyframe as described in Section 3.3.2. The matches for each stereo match with each keyframe are stored in a list, along with the corresponding features. With the matches for each stereo match, Ceres Solver is used to optimally triangulate the 3D position of the stereo match. This provides an accurate depth estimation for the stereo match, even if it is further than 40 times the stereo baseline.

Ceres Solver is used to minimize the reprojection error of the 3D points by finding the optimal position of the 3D point while keeping the camera poses constant. The estimated world position is transformed depending on the pose of each keyframe and compared to the matched feature according to:

$$\boldsymbol{p_{KF}} = \boldsymbol{T_{KFW}} * \boldsymbol{pw} \tag{3.36}$$

$$\boldsymbol{proj} = \boldsymbol{K_{KF}} * \boldsymbol{p_{KF}} \tag{3.37}$$

Where $\boldsymbol{p}_{KF}$ is the 3D point transformed depending on the pose of each keyframe, $\boldsymbol{K}_{KF}$ is the intrinsics matrix of the keyframe and $\boldsymbol{proj}$ is the projection of the 3D point on the image plane. The residuals used are the same as Equations (3.30)-(3.31).

After the estimation of each 3D point's position, the reprojection error is calculated for each match and compared to a threshold to ensure that the estimation is valid, and the matches are correct. If the 3D points that are matched correctly with at least 3 separate keyframes are considered valid they are added to the map as mappoints. The threshold of 3 keyframes ensures that the mappoint is robust and visible from more than a single keyframe.

To ensure that the map is as clear as possible, for each mappoint that is observed by any local keyframe and is not active (not observable in the current frame), if it is not matched with at least 3 other keyframes, it is removed from the map. This helps eliminate mappoints that may be inaccurate and ensures that the map remains robust and clear of incorrect mappoints.

3.5.2 Local BA

Local BA is performed with every new keyframe, after new mappoints have been added to the map. Local BA aims at the improvement of the map, while also maintaining local consistency of the camera trajectory.

In more detail, Local BA optimizes the poses of all local keyframes and all mappoints that are observed by these keyframes. The first step is to gather all the mappoints to be optimized. Using the matched keyframes list of each mappoint, all keyframes are stored and separated to local keyframes and fixed keyframes.

The optimization process involves minimizing the reprojection error of all mappoints observed by the local keyframes and refining the local keyframe poses themselves, while keeping the poses of the fixed keyframes fixed. This is accomplished by formulating a nonlinear least squares problem, which is then solved using Ceres Solver. The local BA minimizes Equation (3.38) by finding the optimal rotation matrices ($\boldsymbol{R_k}$) and translation displacements ($\boldsymbol{t_k}$) for each keyframe, where k denotes the keyframe and K the total number of keyframes:

$$\boldsymbol{R_k}, \boldsymbol{t_k} : \underset{\boldsymbol{R_k}, \boldsymbol{t_k}}{min} \sum_{i=1}^{M} \sum_{k=1}^{K} \rho w_i \boldsymbol{d}(\boldsymbol{x_i}, \boldsymbol{u}(\boldsymbol{F_{i,k}}))^2 \tag{3.38}$$

$$\boldsymbol{F_{i,k}} = \boldsymbol{K}_C(\boldsymbol{R}_{CV}(\boldsymbol{R}_k \boldsymbol{pw}_i + \boldsymbol{t}_k) + \boldsymbol{t}_{CV}) \tag{3.39}$$

Together with Equation (3.34) form the local BA optimization problem.

This optimization process is performed twice. The first time, the Huber loss function is used, to reduce the impact of outliers on the optimization. After the first optimization is completed, the reprojection errors of all the mappoints with their matches are compared with a threshold to find any outliers. After the reprojection error check, the optimization process

begins again with no loss function and without factoring into the calculations any matches that are deemed as outliers, to acquire the best possible result.

After the optimization process is completed for the second time, the reprojection errors are recalculated, and any matches that were previously deemed invalid are rechecked. If this time the matches are valid, they are not removed from the map. Otherwise, they are considered as outliers and removed from the map. In turn, the local keyframe poses are updated, along with the world positions of the mappoints and the recalculation of their descriptors. When the process finishes, the local mapping thread communicates to the tracking thread that the optimization has finished in order to update the current camera poses and mappoints.

3.5.3 Pose Update

After an optimization process, the tracking thread needs to update any new keyframe poses that have been added to the map and their mappoints according to the optimized keyframes. The local mapping thread updates the poses of the previous keyframes, but if a new keyframe has been added while the local mapping thread is still running, the new keyframe is not updated, as it is not included yet in the current optimization process. To update the poses Equation (3.35) is used along with :

$$\boldsymbol{T_{new}} = \boldsymbol{T_{prevKF}} * \boldsymbol{T_{ref}} \tag{3.38}$$

All the new keyframe poses not yet optimized are updated to the new pose $\boldsymbol{T_{new}}$ according to their previous keyframe. Each keyframe that had created new mappoints (adding the 100 closest stereo matches described in Section 3.3.2) updates the position of the mappoints with:

$$\boldsymbol{p_{new}} = \boldsymbol{T_{new}} * \boldsymbol{T_{prev}^{-1}} * \boldsymbol{p_{old}} \tag{3.39}$$

where $\boldsymbol{T_{prev}^{-1}}$ is the previous keyframe pose, $\boldsymbol{p_{old}}$ is the previous world position of the mappoint, while $\boldsymbol{p_{new}}$ is the new one.

The current camera pose is also updated along with its predicted next pose (using the constant velocity model). With each new frame, along with the predicted pose, a reference pose for the predicted pose is calculated.

$$\boldsymbol{Tpred_{ref}} = \boldsymbol{T_{prevCam}} * \boldsymbol{T_{newCam}} \tag{3.40}$$

where $\boldsymbol{T_{prevCam}}$ is the previous camera pose, $\boldsymbol{T_{newCam}}$ is the newly estimated camera pose and $\boldsymbol{Tpred_{ref}}$ is the reference pose for the predicted pose.

With Equation (3.40), the new camera pose with its predicted next pose can be calculated using Equations (3.41)-(3.42).

$$\boldsymbol{T_{upCam}} = \boldsymbol{T_{latestKF}} * \boldsymbol{T_{refCam}} \tag{3.41}$$

$$\boldsymbol{T_{predCam}} = \boldsymbol{T_{upCam}} * \boldsymbol{Tpred_{ref}} \tag{3.42}$$

where $\boldsymbol{T_{latestKF}}$ denotes the pose of the latest keyframe (already updated), $\boldsymbol{T_{refCam}}$ is the reference pose of the current camera pose, $\boldsymbol{T_{upCam}}$ denotes the updated current camera pose and $\boldsymbol{T_{predCam}}$ is the updated predicted pose for the camera using the constant velocity model. Figure 3-10 illustrates the difference between a straight-line trajectory with the use of Local Bundle Adjustment (BA) and without. The figure clearly demonstrates that local BA optimization maintains the local consistency and reduces estimation errors of camera poses in each frame.

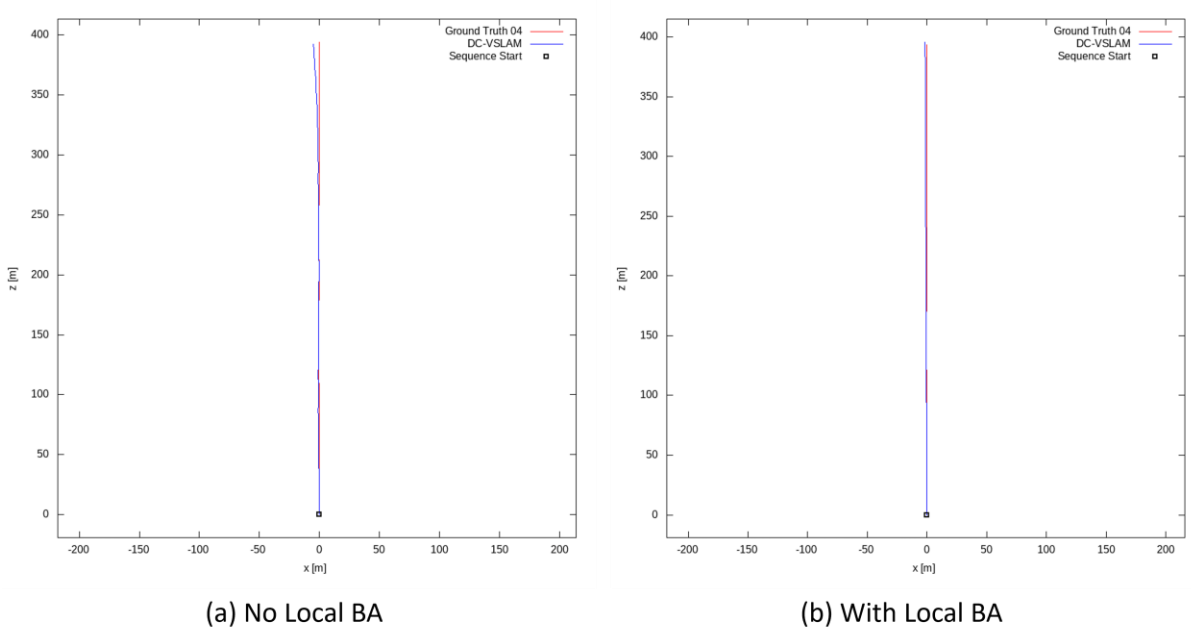


Figure 3-10. Comparison With and Without Local BA.

3.6 Loop Closure Thread

A separate thread is dedicated to the loop closure process to avoid overloading the tracking or local mapping threads. This thread is used to optimize the camera trajectory and the map when a loop is detected.

As discussed in Section 2.5, the bag-of-words representation is a widely used method for detecting loops in visual SLAM systems. It involves comparing the visual features of each frame with those of new frames, and when a similar enough image is detected, the SLAM system begins optimizing the camera poses and the map. While this approach is effective in most environments, it may incorrectly detect loop closures in homogeneous or repetitive environments, leading to incorrect optimizations that make the map and camera trajectory worse or unusable.

In this thesis, the main purpose of the VO software developed was to enable a mobile robot to traverse and map a vineyard, i.e.: a highly repetitive and homogeneous environment, rendering the use of a bag-of-words approach for loop closure inadequate. Therefore, a novel approach was introduced that relied on Apriltags for loop closure detection. The robustness of the Apriltag detection algorithms allows for accurate loop closure detection in all types of environments. Additionally, Apriltags can be easily placed in the environment and provide a reliable reference point for the system.

3.6.1 First AprilTag Detection

A third camera was added on the right side of the rover for the Apriltag detection process. The Apriltag detection was performed on a separate thread using the apriltag_ros package [69]. The apriltag_ros package takes as input the image and its parameters, as well as a settings file that contains the size and the identity of the Apriltag. When an Apriltag is detected, the pose of the Apriltag relative to the camera pose that detected it is returned.

The tracking and local mapping thread estimate and optimize camera poses, while the Apriltag thread searches for Apriltags using the right camera. Upon detection of the first

Apriltag, its world pose is calculated relative to the front left camera, which is the camera estimated from the tracking thread. In order for the detection of the Apriltag to be considered valid, the Apriltag has to be detected 10 times consecutively. If the detection is valid, the world pose is calculated using :

$$\boldsymbol{T_{tag} = T_{wc} * T_{ccr} * T_{crAT}} \tag{3.43}$$

where $\boldsymbol{T_{wc}}$ is the estimated world pose of the front left camera, obtained from the tracking thread, $\boldsymbol{T_{ccr}}$ denotes the transformation from the front left camera to the right camera used for detecting the Apriltag, and $\boldsymbol{T_{crAT}}$ represents the transformation from the right camera to the Apriltag, which is the output of the apriltag_ros package. Once the world pose of the tag is calculated, the Apriltag detection process requires that the tag be absent for a certain number of frames before it can be detected again for loop closure.

3.6.2 Second Apriltag Detection

With the world pose of the tag, the next time the same Apriltag is detected, means that a loop has been performed. With the world pose of the tag already known, the front left camera pose can be calculated using :

$$\boldsymbol{T_{wc} = T_{tag} * T_{crAT}^{-1} * T_{ccr}^{-1}} \tag{3.44}$$

This calculated front left camera pose is considered more accurate than the estimated camera pose obtained from the tracking process, as there is no error accumulation from Apriltag detection. After a front left camera pose has been calculated from the Apriltag Detection, the loop closure optimization is initiated.

3.6.3 Loop Closure Optimization

The loop closure optimization is performed when an Apriltag has been detected for the second time. With a more accurate world pose for the front left camera, a Global BA problem is formed, where the keyframe poses that detected the Apriltag are fixed, while all other poses (except for the starting pose) can be optimized. To maintain consistency across all optimization processes, the local BA optimization is temporarily paused while the loop closure optimization begins. The global BA optimization is then performed, and once it has been completed, the local BA optimization can continue. This ensures that the mappoints used by the global BA remain unchanged and that the reprojection errors for all mappoints remain consistent throughout the optimization process.

The equations utilized by the Global BA optimization are identical to those used by the Local BA optimization, which include Equations (3.34), (3.37)-(3.38). However, in this case, the optimization is performed on all the mappoints and keyframes. As this process is time-consuming, it is performed on a separate thread. After the optimization is completed, the reprojection errors for all mappoints with their connected keyframes are compared with a threshold, and any error larger than the threshold indicates a wrong match, and it is removed from the connections. Any mappoint that ends up with less than 3 connected keyframes is removed from the map.

Once the global optimization is complete, the tracking thread is notified that the camera poses and map have been updated, and updates the poses of the keyframes that were not optimized according to Section 3.4.3. The global BA optimization removes any accumulated error during the camera movement, resulting in a more accurate camera trajectory and map

of the environment. Figure 3-11 presents the two different AprilTag Detection along a camera trajectory. After the second detection global BA is performed to remove any accumulated error during the camera trajectory.

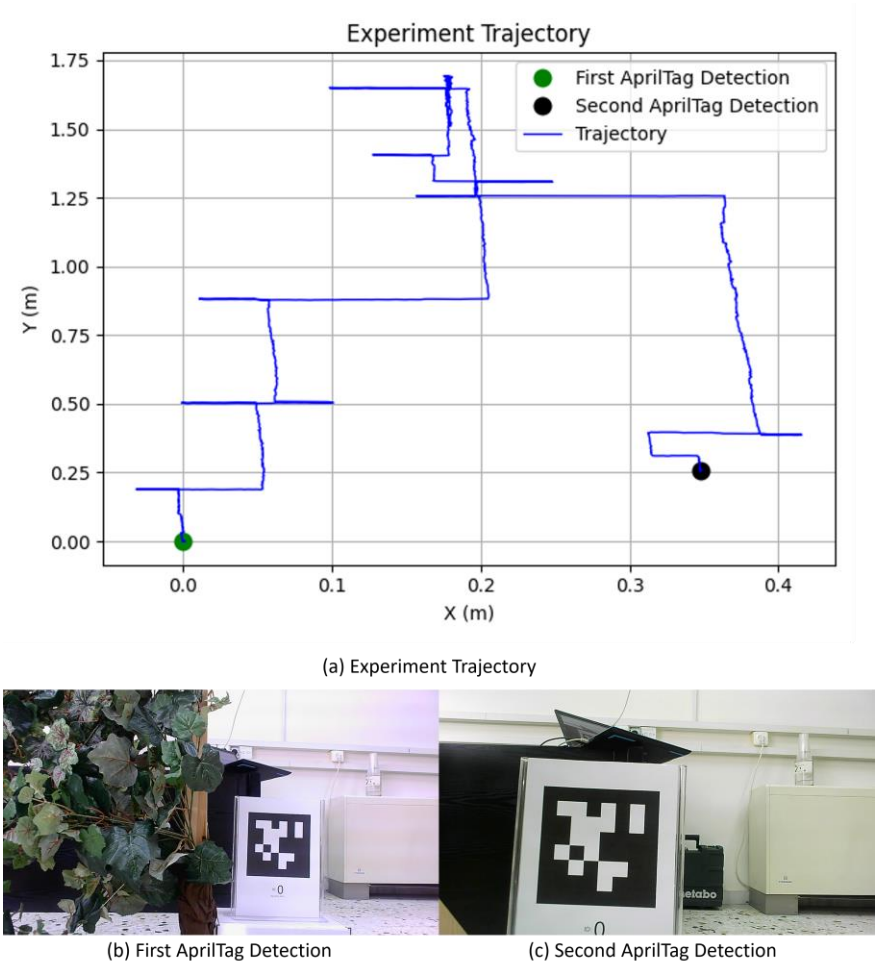


Figure 3-11. First and Second AprilTag Detection on a Trajectory. After the Second Detection Global BA is performed.

3.7 Visual Thread

The visual thread is only used for the visualization process, and it is not required for the system to operate. This thread visualizes all mappoints, keyframes and current camera pose estimation, providing a visual representation of the environment. This representation can also be used to detect any errors or inaccuracies in the system.

For visualization purposes, the Pangolin libraries [83] are used. Pangolin is a set of lightweight and portable utility libraries for prototyping 3D, numeric or video-based programs and algorithms. The system visualizes mappoints from both front and back stereo cameras along with all keyframes, but only mappoints that are connected to at least 3 keyframes are shown. Active mappoints are represented in green, while inactive ones are in white. Keyframes are represented in blue, and the current camera pose estimation is represented in yellow. Figure 3-12 shows the visual environment created by the dual stereo camera setup using the Pangolin.

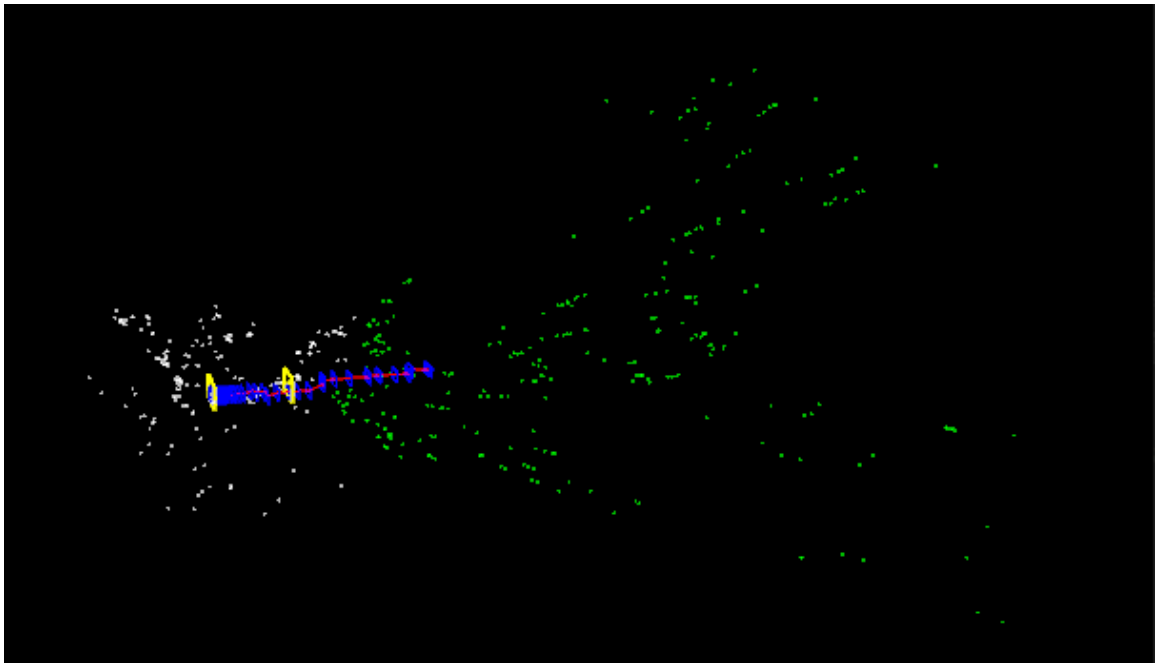


Figure 3-12. Visual Environment created using Pangolin. White points are inactive mappoints, green points are active mappoints. Blue squares are the keyframes created (connected with a red line), while the yellow squares are the current front and back camera.

4 Results

In this Chapter, the developed visual SLAM algorithm is evaluated employing commonly used image datasets by researchers worldwide as well as camera trajectories in a simulated environment and in a realistic vineyard setup at CSL. Although this visual SLAM algorithm was designed for two stereo cameras, single camera operation is also available and is tested with datasets that provide single stereo camera images. In this Chapter, the setups developed for experiments in simulation and at the CSL lab are presented. In turn, results of the single stereo camera operation are presented, tested with the KITTI and EuRoC datasets. Finally, the results of the simulation and the experiment at the CSL are presented, where the dual stereo camera variant of the developed software is used.

4.1 Experimental Setup in Simulation

To test the performance of the visual SLAM algorithm under controlled conditions, a simulation setup was implemented using Gazebo [84], which includes an accurate model of the RP as well as sensor plugins to simulate multiple stereo cameras and STL CAD models of grapevines acquired from [85], see Figure 4-1, Figure 4-2.

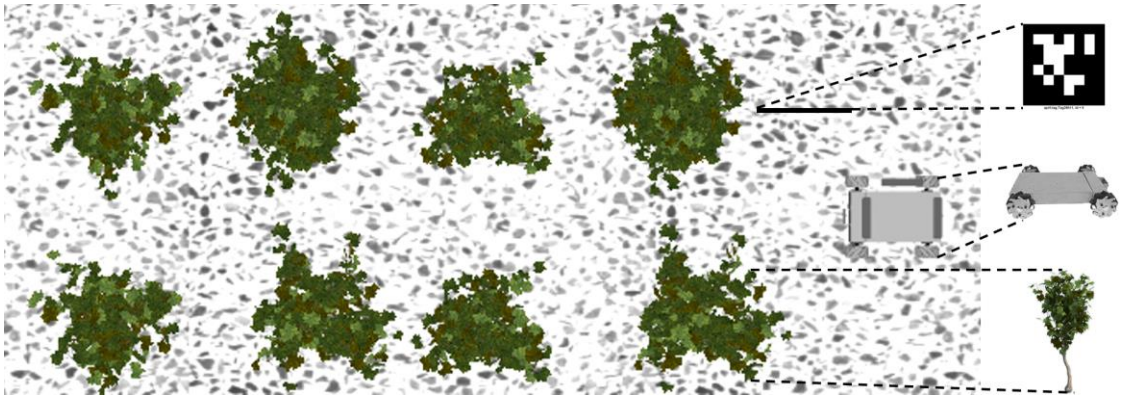


Figure 4-1. Gazebo environment that resembles the CSL synthetic vineyard setup.

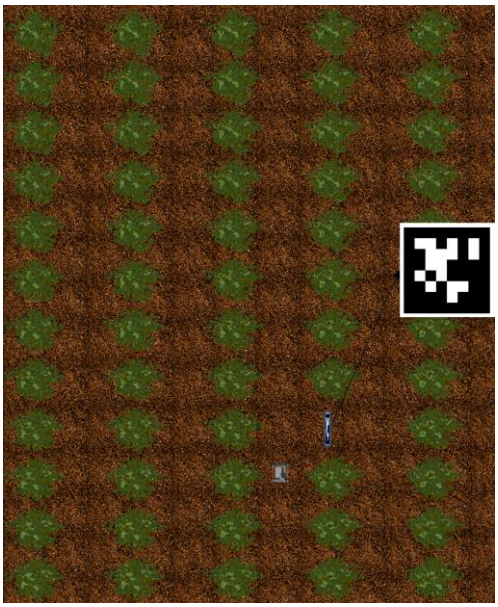


Figure 4-2. Simulated Vineyard.

Gazebo combined with rosbags provided the ability to assess the performance of the developed visual SLAM algorithm against other state-of-the-art visual SLAM systems. Two simulated environments were created; the first accurately resembles the synthetic vineyard created at CSL, NTUA and the second resembles an actual much bigger vineyard.

4.2 Experimental Setup at CSL

4.2.1 Realistic Grapevine Canopy

To perform experiments easily with varying and controlled light conditions, a vineyard with artificial grapes and leaves was built at CSL (Figure 4-3). Each row consists of multiple plants on a trellis system so that the canopy form resembles a natural canopy. The basic vineyard row parameters, such as the distance between plants ($\sim 1m$) and grapes' minimum height ($0.60m$) is based on common viticulture practices in Greece. The artificial grapes' grid features varying density, grape size, creating different visibility conditions since some grapes are partly covered with leaves, whereas others lie on the front plane.

Figure 4-3. Vineyard experimental setup at CSL. The grapevine canopy consists of two leaf types with different color, i.e., green & green-yellow leaves.

To acquire the ground truth position of the rover, a camera was fixed to the ceiling of the room and an Apriltag was placed on the top cover of the rover. With this setup the camera located at the ceiling tracked the rover Apriltag and published its pose in a ROS topic. The vineyard consists of three 4-meter-long rows on even terrain as presented in Figure 4-3 & Figure 4-4.

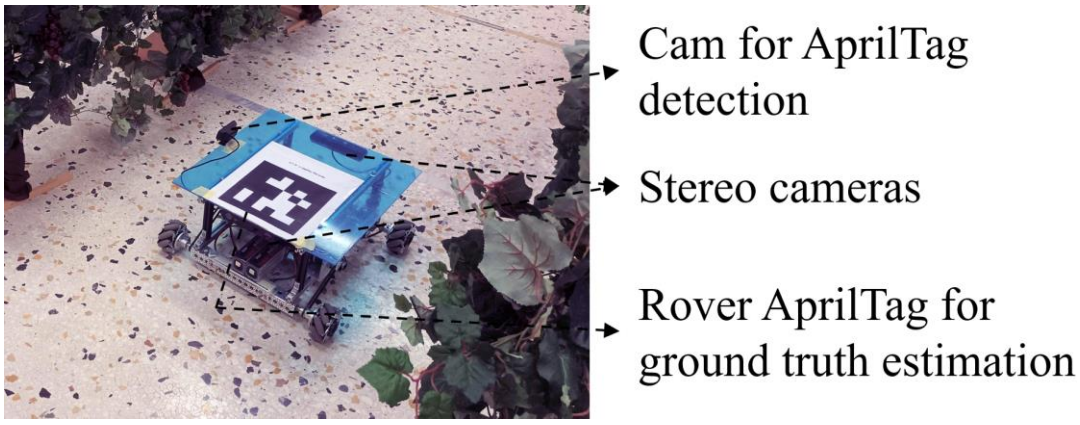


Figure 4-4. The rover in the synthetic vineyard developed at CSL, NTUA.

4.2.2 CSL's Rover

A wheeled robotic platform (RP) was used to validate the concept (Figure 4-5). The RP is designed and constructed for research purposes, comprising custom-built in-house parts as well as off-the-shelf parts (e.g., aluminum profiles, bearing units etc.). Its motion system features four mecanum wheels [86] to provide the robot with omnidirectional motion capabilities. The wheels are powered by four Maxon DC motors (RE 35) combined with planetary gearboxes (GP 42) and incremental encoders (HEDL 5540), providing 5 Nm of continuous torque per wheel. GT2 timing belts and pulleys are used to protect actuator shafts from increased robot payloads and to transmit power to the wheels. Two RoboClaw [87] 2x30A motor controllers are used to drive the actuators, since each controller can drive two DC brushed motors. The two motor controllers are connected via USB to the rover's master computer, which is a Raspberry Pi 2 model B (RPi) running the Raspbian OS. The operator can connect to the RPi using WiFi and Secure Shell (SSH) Network Protocol to run a Python script that establishes two serial connections with the motor controllers and sends the desired commands. The system is powered by two LiPo batteries for the RoboClaw controllers and a powerbank for the Raspberry Pi.

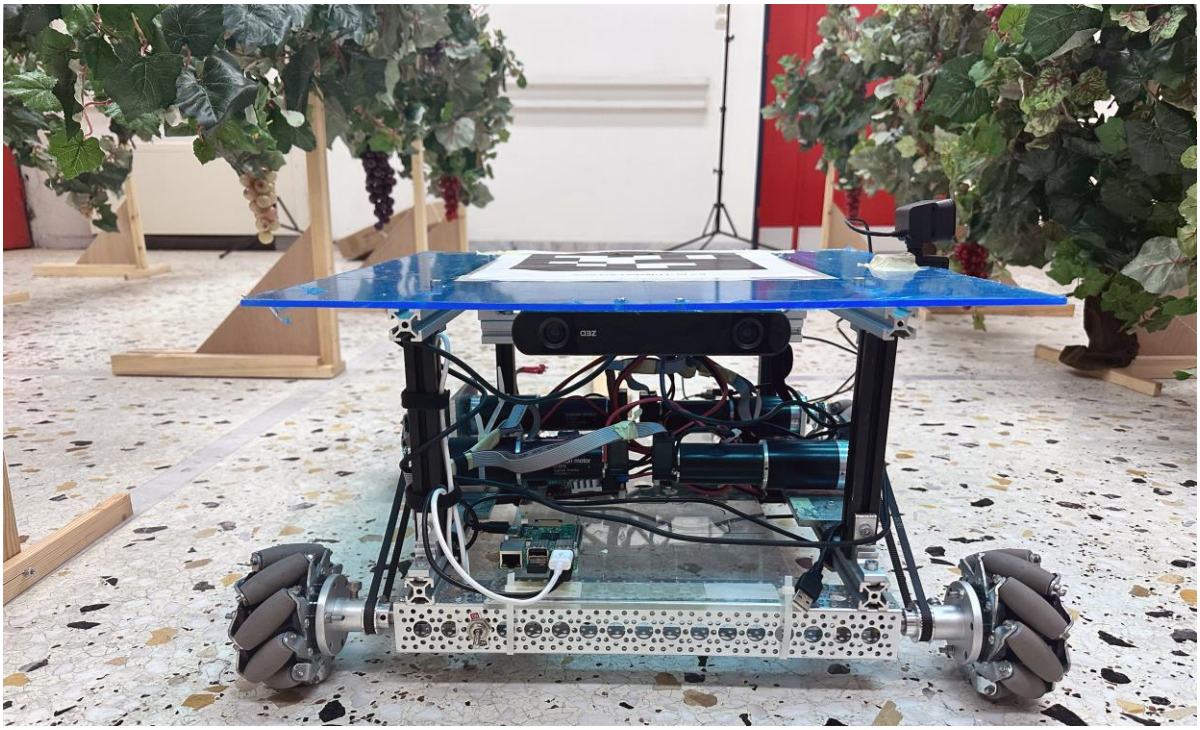


Figure 4-5. The Robotic Platform (RP).

4.3 KITTI Dataset

The KITTI dataset [88] is one of the standard computer vision datasets. The dataset was recorded using a station wagon car equipped with two high-resolution color and grayscale video cameras. Accurate ground truth was recorded using a Velodyne laser scanner and a GPS localization system. The sensors at the KITTI dataset are shown in Figure 4-6.

Figure 4-6. KITTI Dataset Sensors [88].

The dataset consists of various paths in Karlsruhe, in rural areas and on highways. All data are provided in raw and rectified images. For each one of KITTI's benchmarks, an evaluation metric is provided where visual SLAM algorithms test their performance.

To test the developed visual SLAM algorithm for the single stereo camera operation, the grayscale rectified stereo datasets were used. The stereo dataset consists of a stereo camera producing images $1241x376$ pixels at $10fps$. The stereo sensor has a $\sim 0.54m$ baseline. The stereo cameras parameters are summarized in Table 4-1.

Table 4-1. The KITTI Dataset Stereo Parameters.

Parameter	Value
f_x	718.8560
f_y	718.8560
c_x	607.1928
c_y	185.2157

The KITTI dataset consists of 22 different stereo camera sequences, but ground truth camera poses are provided only for the first 11 of them. Figure 4-7 presents the setup for the recording of the KITTI dataset where distances from the sensors utilized are given.

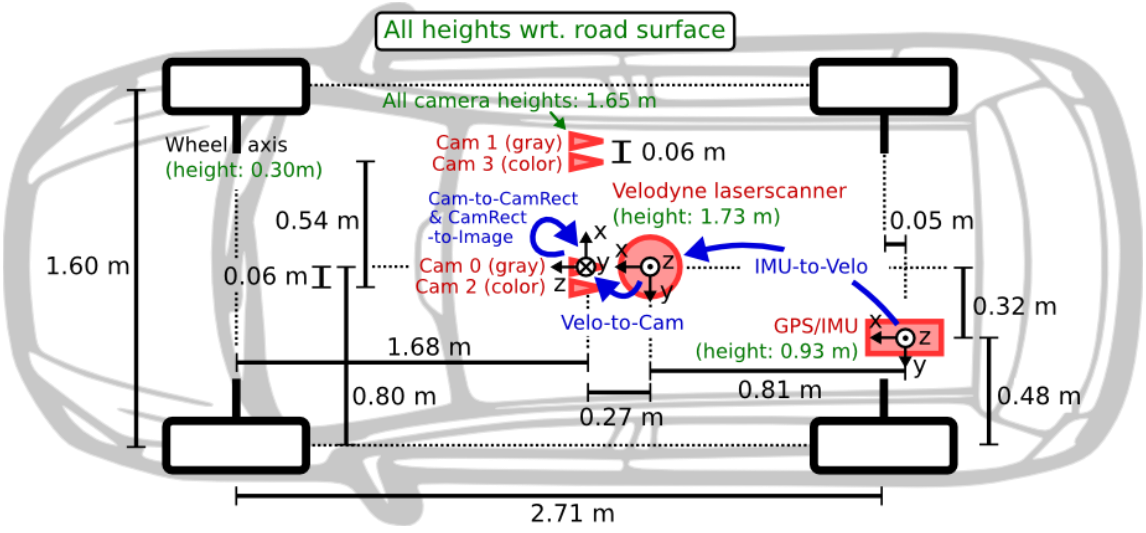


Figure 4-7. KITTI Dataset Setup [88].

The developed visual SLAM algorithm was tested on the first 11 sequences where ground truth was provided. The ORB parameters used for this dataset are presented in Table 4-2. 2000 features were chosen considering the high resolution of each image in the dataset.

Table 4-2. ORB Parameters used in KITTI Dataset.

Parameter	Value
$nFeatures$	2000
$nLevels$	8
$scaleFactor$	1.2
$edgeThreshold$	19
$maxFastThresh$	20
$minFastThresh$	7
$patchSize$	31

The sequences of the KITTI dataset consist of camera trajectories that include loop closures, so visual SLAM algorithms that use the bag-of-words approach for loop detection can detect loops and reduce the error accumulation. The developed single stereo camera operation could have better results in this dataset if the bag-of-words approach for loop closure was implemented instead of the AprilTag loop closure detection. However, incorrect loop closures due to the similarity of the environment could be detected as mentioned in previous chapters. To evaluate the performance of the developed algorithm in the dataset the average relative translation error t_{rel} and rotation error r_{rel} are used, proposed in [89]. Table 4-3 presents the results of the developed visual SLAM algorithm DC-VSLAM in comparison to ORB-SLAM3.

Table 4-3. Results of the Developed Visual SLAM Algorithm on the KITTI Dataset in comparison to ORB-SLAM3.

	DC-VSLAM		ORB-SLAM3		
Sequence	$\mathbf{t_{rel}}(\%)$	$\boldsymbol{r_{rel}(deg/100m)}$	$\mathbf{t_{rel}}(\%)$	$\boldsymbol{r_{rel}(deg/100m)}$	**Loop**
00	0.7775	0.52	0.6858	0.45	Yes
01	1.4734	0.20	1.7811	0.58	No
02	0.8069	0.46	0.7571	0.42	Yes
03	0.9025	0.37	0.9334	0.27	No
04	0.6281	0.40	0.4693	0.13	No
05	0.7010	0.41	0.6117	0.59	Yes
06	0.9835	0.49	0.5814	0.32	Yes
07	0.8677	0.87	0.4310	0.48	Yes
08	1.0599	0.55	1.0351	0.53	No
09	0.9378	0.46	1.0356	0.55	Yes
10	0.7190	0.68	0.6686	0.57	No

The results presented in Table 4-3 demonstrate that the performance of the algorithm using a single stereo camera is impressive, with the majority of the results showing a translation error less than 1% and a rotation error less than $1\ deg/100m$ across all the tested sequences. Sequences 00, 02, 05, 06, 07 and 09 contain loops as indicated in the last row of the table. It is apparent that the developed SLAM algorithm produces comparable results to

ORB-SLAM3 while achieving better results in 4 out of 10 sequences, mainly in the ones that don't contain loops since in our approach loop closure recognition was based on AprilTag detection. In Sequence 09, despite the presence of a loop that ORB-SLAM3 successfully detects, DC-VSLAM still achieves superior results. We have to note that in all sequences, the single camera variant of our software was tested since there is no dataset available for dual camera VSLAM algorithms.

Figure 4-8 presents different sequences off the KITTI dataset with the resulting trajectory of the visual SLAM algorithm. Six sequences in the dataset have loops that can significantly reduce the impact of error accumulation. If these loops are detected correctly, the results could be further improved.

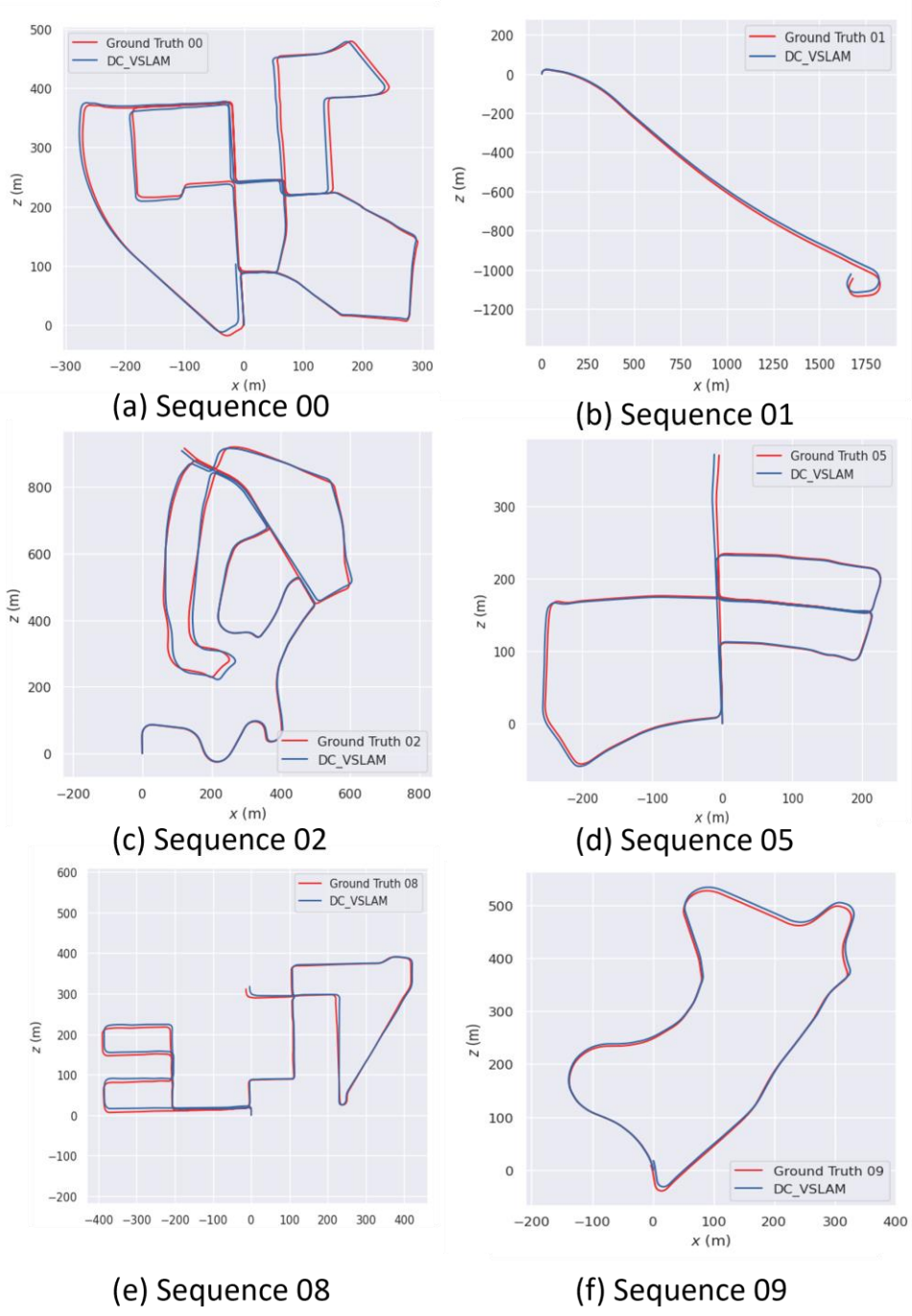


Figure 4-8. DC-VSLAM (blue) with Ground Truth (red) trajectories in the KITTI Dataset.

4.4 EuRoC Dataset

For further testing of the single stereo camera operation of the developed algorithm, the EuRoC Dataset [90] was utilized. The EuRoC Dataset consists of 11 stereo sequences recorded from a micro aerial vehicle (MAV) flying around a large industrial environment and two different rooms. The stereo sensor provides images of size $752x480$ pixels at $20fps$. It has a baseline of $\sim 0.11m$ and the images it provides are unrectified so image rectification is needed. The sequences are separated into *easy, medium,* and *difficult,* depending on the motion of the MAV, and the lighting conditions. Figure 4-9 presents the MAV with its sensors.

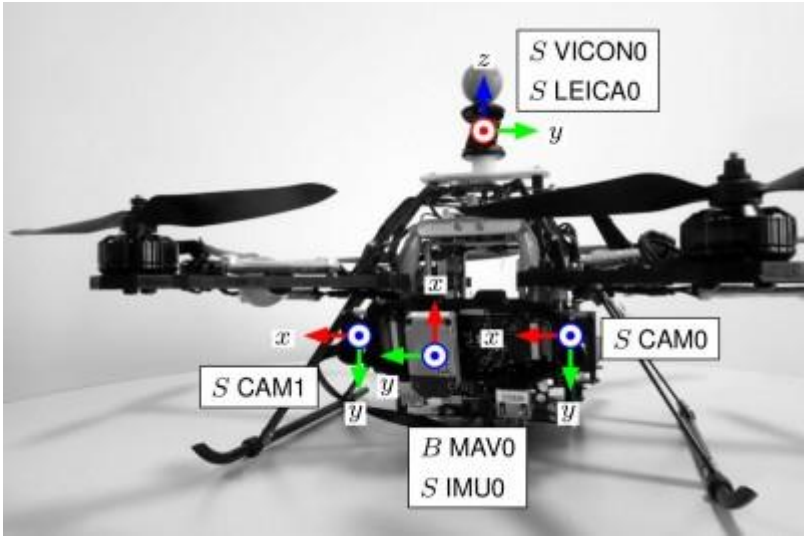


Figure 4-9. EuRoC Dataset MAV [90].

For the ground truth pose estimation of the MAV 2 drone, the following sensors were used:

- Leica MS50 laser tracker and scanner.
- Vicon 6D motion capture system.

The MAV stereo camera parameters are presented in Table 4-4.

Table 4-4. The EuRoC Dataset Stereo Parameters.

Parameter	Value
f_x	435.2046
f_y	435.2046
c_x	367.4517
c_y	252.2008

The ORB parameters used for this dataset are presented in Table 4-5.

Table 4-5. ORB Parameters used in EuRoC Dataset.

Parameter	Value
$nFeatures$	1000
$nLevels$	8
$scaleFactor$	1.2
$edgeThreshold$	19
$maxFastThresh$	20
$minFastThresh$	7
$patchSize$	31

The EuRoC dataset revisits several times a previous position, that could potentially reduce any error accumulation if the loop is correctly detected. To evaluate the results of the algorithm, the absolute translation RMSE was calculated for each resulting trajectory. The results are summarized and compared to ORB-SLAM3 in Table 4-6.

Table 4-6. Results of the Developed Visual SLAM Algorithm on the EuRoC Dataset in comparison to ORB-SLAM3.

	DC-VSLAM	ORB-SLAM3
Sequence	**RMSE**	**RMSE**
MH_01_easy	0.035167	0.029
MH_02_easy	0.077663	0.019
MH_03_medium	0.158545	0.024
MH_04_difficult	0.234725	0.085
MH_05_difficult	0.186501	0.052
V1_01_easy	0.056128	0.035
V1_02_medium	0.073557	0.025
V1_03_difficult	0.262493	0.061
V2_01_easy	0.063262	0.041
V2_02_medium	0.125586	0.028
V2_03_difficult	1.933980	0.521

In both datasets (KITTI and EuRoC) our single stereo camera VSLAM software achieves impressive results. The RMSE in most cases are below 0.3, even in sequences, such as MH_04, MH_05, where the camera has very few features to track as presented in Figure 4-10. The presence of loops in each sequence improves the performance of ORB-SLAM3 compared to the developed algorithm which implemented a different method for loop closure detection, i.e.: AprilTags.

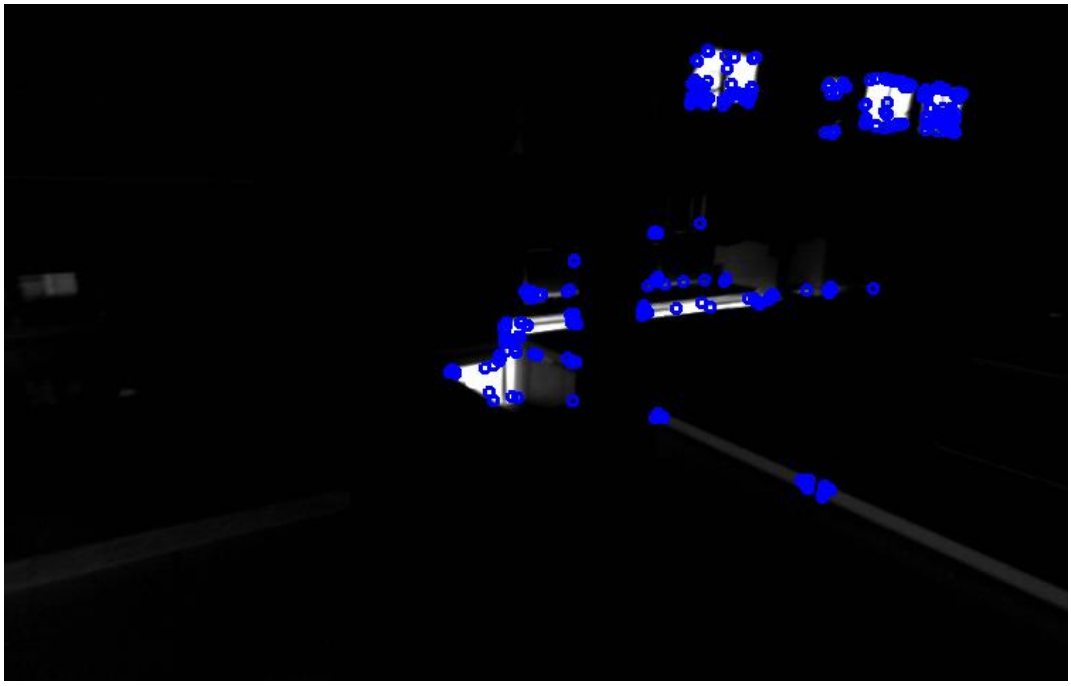


Figure 4-10. Challenging Light Conditions in Sequence MH_04 [90].

In the $V2\_03$ sequence, the MAV performs abrupt maneuvers resulting in motion blur, making feature tracking significantly more difficult compared to other sequences. Incorrect matches lead to an increase in error during camera pose estimation resulting in a high RMSE. Trajectories from some of the sequences in the EuRoC dataset, along with their values on each axis are presented in Figure 4-11 and Figure 4-12.

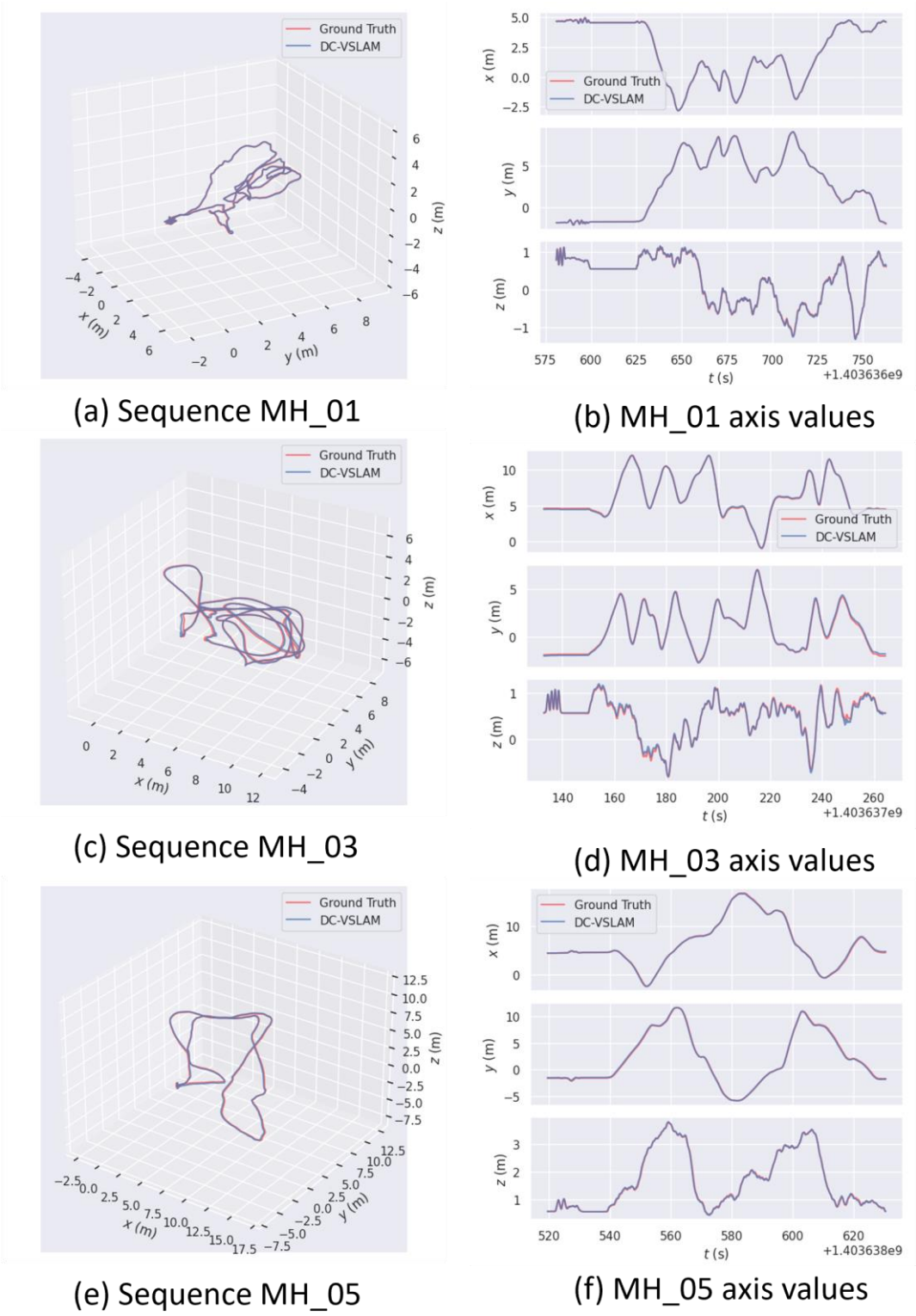


Figure 4-11. DC-VSLAM (blue) with Ground Truth (red) trajectories (MH_01, MH_03, MH_05) in the EuRoC Dataset with their values on each axis.

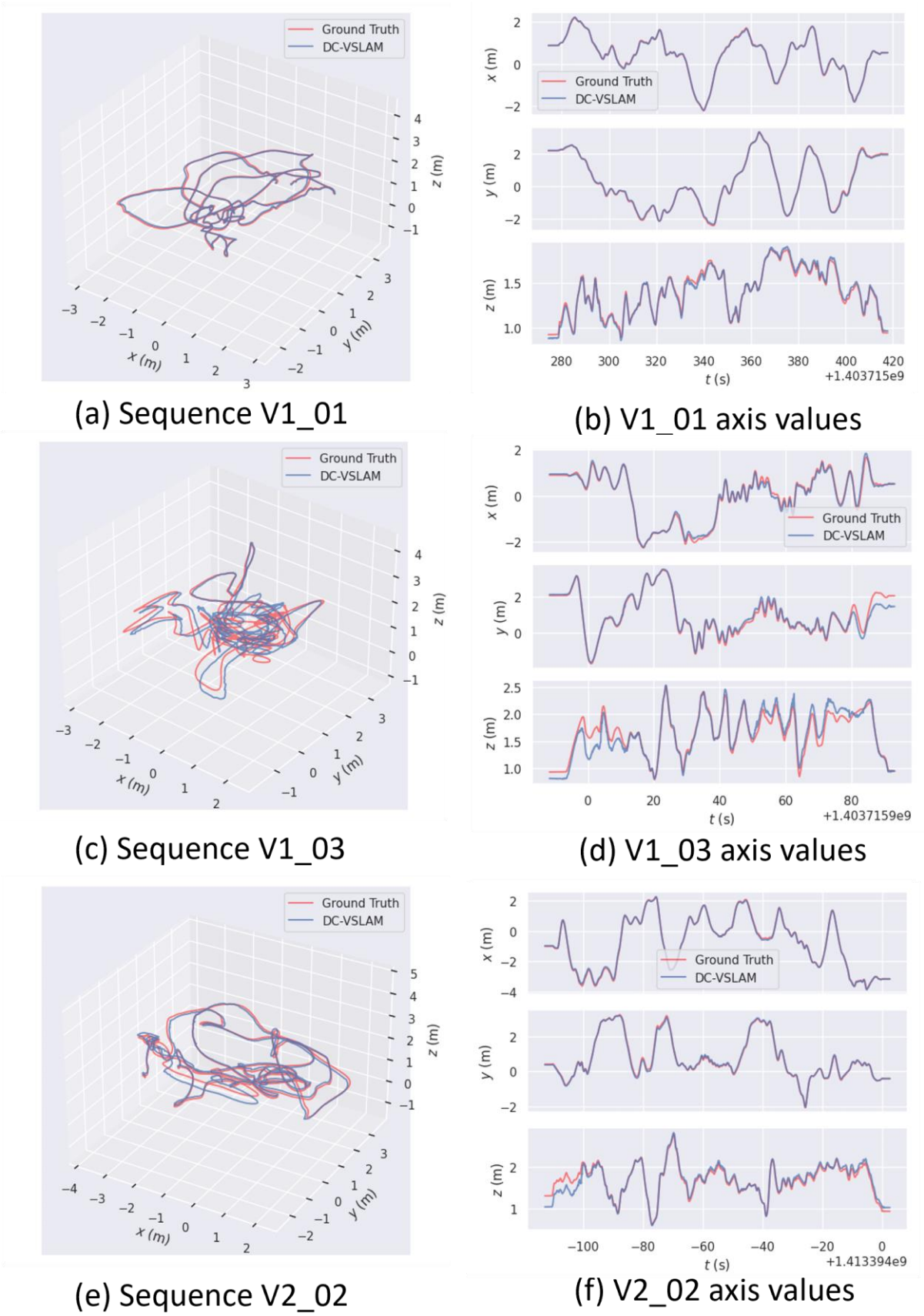


Figure 4-12. DC-VSLAM (blue) with Ground Truth (red) trajectories (V1_01, V1_03, V2_02) in the EuRoC Dataset with their values on each axis.

4.5 Gazebo Realistic Vineyard Canopy

The RP used in Gazebo is described in Section 4.2.2. It is equipped with two stereo cameras, one in the front and one in the back, that are used for camera pose estimation in the visual SLAM algorithm, operating at $752x480px$ resolution, and one single RGB camera located at the right side of the rover used for AprilTag detection, operating at $1920x1080px$ resolution. The HD resolution for the RGB camera was selected to increase the accuracy of the AprilTag detection. The two stereo cameras have $12cm$ baseline (identical to the ZED2 [91] stereo camera) and operate at $15fps$. Two environments were created; one simulates the realistic environment built at CSL, and the second simulates a typical vineyard with many rows of grapevines. The simulated camera parameters are presented in Table 4-7.

Table 4-7. Simulated Stereo Camera Parameters.

Parameter	Value
f_x	263.2786
f_y	263.2786
c_x	376.5000
c_y	240.5000

The ORB parameters used for all the simulation environments are presented in Table 4-8.

Table 4-8. ORB Parameters used in the Simulation Environments.

Parameter	Value
$nFeatures$	1000
$nLevels$	8
$scaleFactor$	1.2
$edgeThreshold$	19
$maxFastThresh$	20
$minFastThresh$	7
$patchSize$	31

Two experiments were performed with the first simulation environment. The first consisted of the rover inspecting the vineyard where a brown carton box was added close to the path that the RP should follow. The box has very few features available to track since its surface is homogeneous and monochromatic. The goal was to evaluate the algorithms when a featureless object covers a large part of the FoV of the front camera.

The ORB-SLAM3 failed to estimate the trajectory of the camera in this environment and ultimately the tracking process was reset as presented in Figure 4-13. In contrast, the developed algorithm outperformed ORB-SLAM3 and displayed robustness. It continued the camera pose estimation thanks to the features available through the rear camera. Eventually, it performed loop closure optimization when the AprilTag was detected at the end of the inspection process. Loop closure detection further improved the estimation of all previous poses since at this point the algorithm had the maximum confidence that the same point in the vineyard was visited in the past.

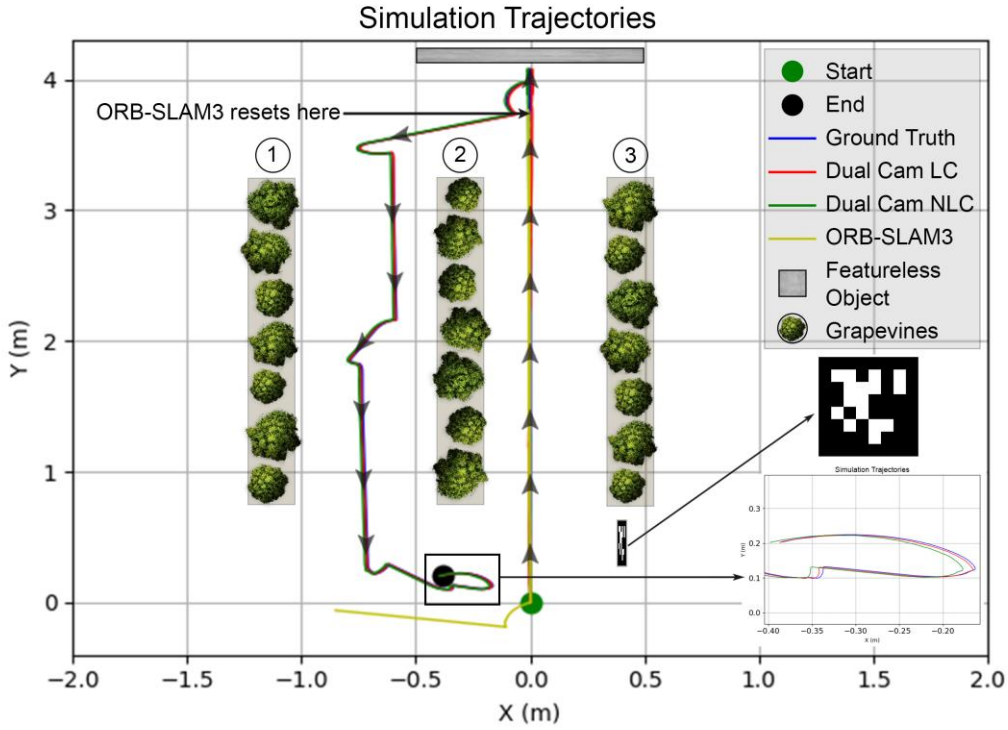


Figure 4-13. The first simulation experiment in Gazebo. Dual Cam LC denotes our approach with loop closure, while Dual Cam NLC denotes our approach without loop closure. ORB-SLAM3 resets since there are very few features available to track. Our approach continues thanks to the features available from the rear camera.

During the second experiment, the RP was instructed to follow a longer path, map the entire vineyard, and arrive at its starting point. The purpose was to test the error accumulation over time and its effect on the overall accuracy of the developed algorithm. The developed algorithm achieved cm-level accuracy in this scenario by detecting the AprilTag at the end of the path and performing loop closure optimization, as presented in Figure 4-14.

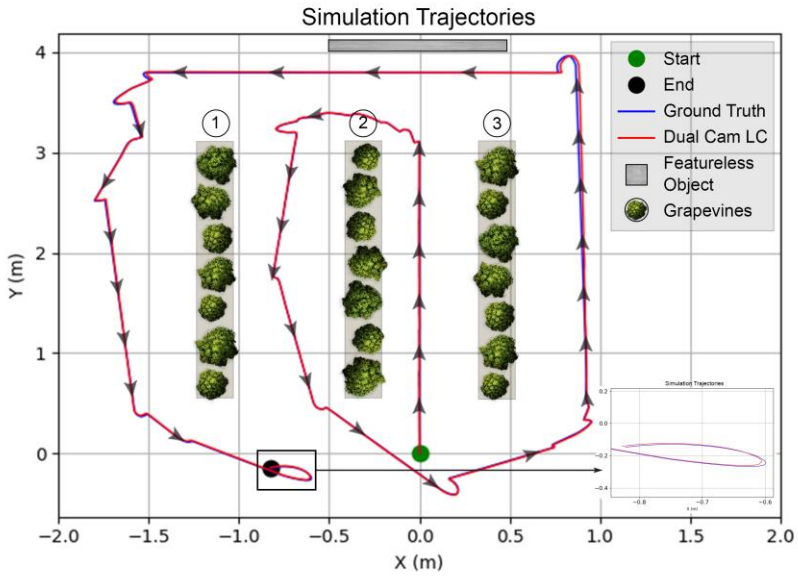


Figure 4-14. The second simulation experiment in Gazebo. The developed visual SLAM algorithm achieves cm-level accuracy.

4.6 Gazebo Realistic Vineyard

The second simulation environment was created to test the loop detection accuracy of the bag-of-words approach, on images with high similarity. The robot had to follow a long path in the vineyard as presented in Figure 4-15 and return to its home position.

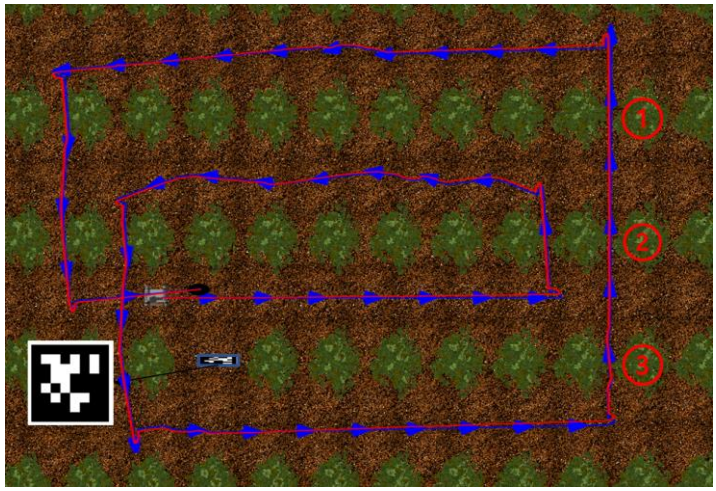


Figure 4-15. Followed Path through the simulated Vineyard.

Intentionally, loops were avoided except while returning to the home position where one loop is completed. In this experiment ORB-SLAM3 detected 3 loop closures, two of them being incorrect, due to the similarities in the environment, resulting in a highly inaccurate pose tracking. In contrast, the developed algorithm tracked the path with an impressive cm-level accuracy and correctly detected the loop closure at the end of the path, where the AprilTag is located, ultimately resulting in excellent tracking performance. See Figure 4-16.

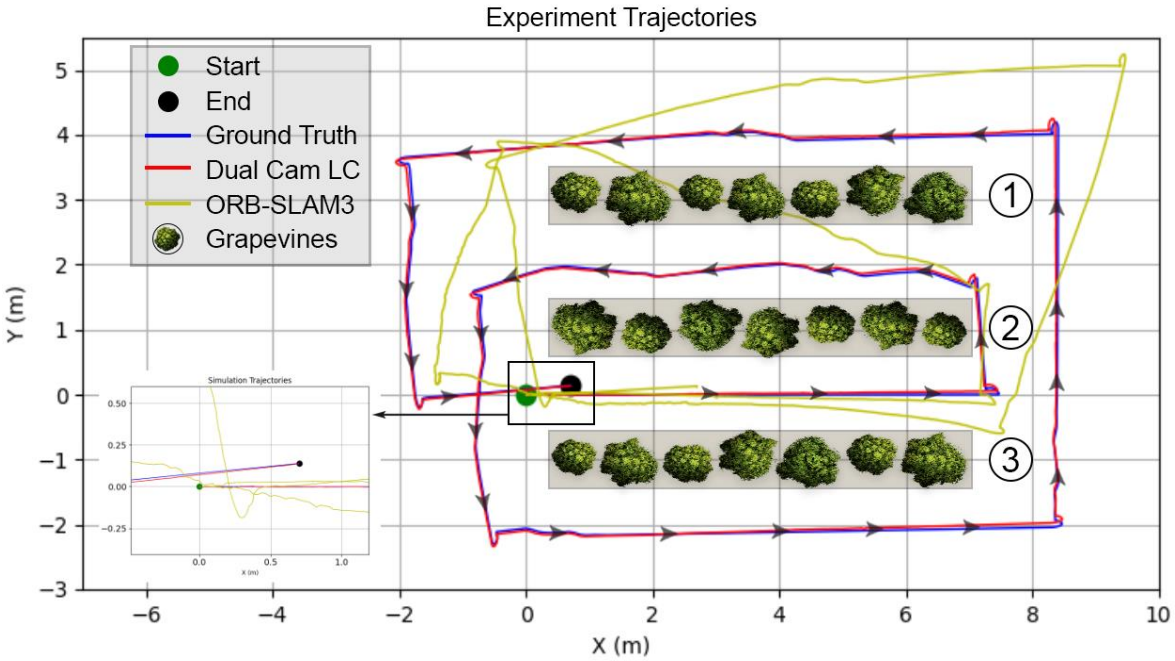


Figure 4-16. ORB-SLAM3 incorrectly detects two loop closures resulting in inaccurate pose tracking, while our approach tracks the path accurately and performs loop closure optimization only when the registered AprilTag is detected at the end of the trajectory.

Figure 4-17 and Figure 4-18 present the maps generated by ORB-SLAM3 and DC-VSLAM. The map generated by ORB-SLAM3 is inaccurate due to incorrect loop detection, due to the homogeneity of the environment. On the other hand, the map generated by DC-VSLAM is accurate.

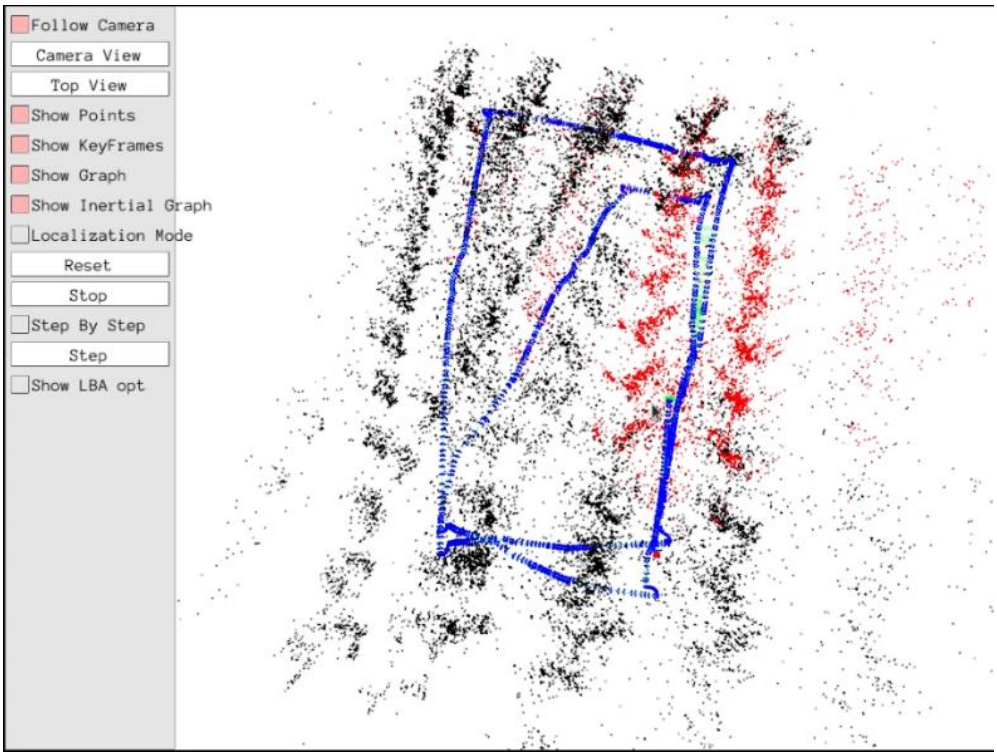


Figure 4-17. ORB-SLAM3 Created Map.

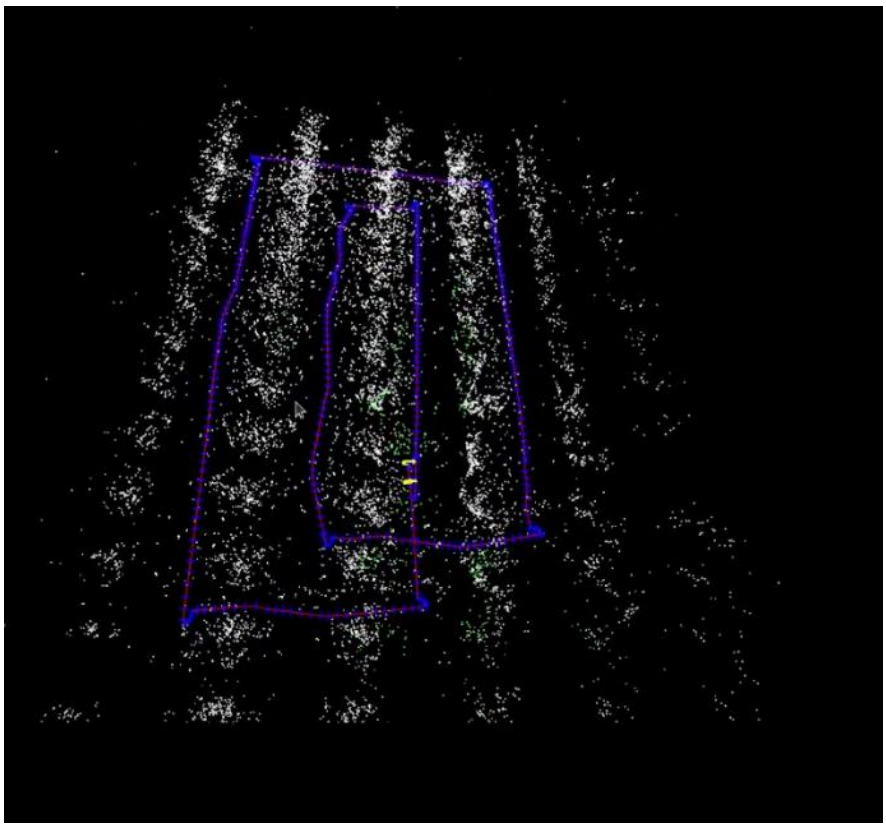


Figure 4-18. DC-VSLAM Created Map.

Table 4-9 presents the absolute translation RMSE of the developed Visual SLAM Algorithm on the Simulation Experiments. The results demonstrate the developed algorithm's exceptional accuracy.

Table 4-9. Results of the Developed Visual SLAM Algorithm on the Simulation Experiments.

DC-VSLAM	
Sequence	**RMSE**
First Simulation Experiment	0.006930
Second Simulation Experiment	0.015403
Vineyard Experiment	0.053063

4.7 CSL Experiment

Additional experiments were conducted on the developed synthetic vineyard at CSL. The RP presented in Section 4.4.2 was utilized for this set of experiments. It was equipped with (a) two ZED2 cameras (front and rear) with $\sim 12cm$ baseline operating at $640x360 @ 15fps$ resolution and (b) a Creative Webcam [92] operating at $1920x1080$ resolution attached at the right side of the rover used for AprilTag detection. To acquire the ground truth position of the rover a camera was fixed to the ceiling of the room and an AprilTag was placed on the top cover of the rover. With this setup the camera located at the ceiling tracked the top AprilTag and published its pose in a ROS topic. Again, a brown carton box was placed in the middle of the row to evaluate the algorithms when a featureless object covers a large part of the FoV of the front camera. The stereo camera parameters for the front and rear cameras are presented in Table 4-10 & Table 4-11, respectively.

Table 4-10. Front ZED2 Stereo Camera Parameters.

Parameter	Value
f_x	261.7559
f_y	261.7559
c_x	323.5115
c_y	180.4974

Table 4-11. Back ZED2 Stereo Camera Parameters.

Parameter	Value
f_x	262.9667
f_y	262.9667
c_x	322.1598
c_y	186.5257

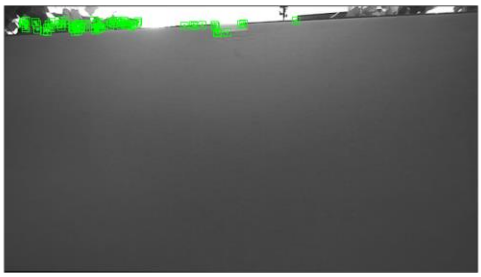
The ORB parameters used for the CSL experiment are presented in Table 4-12.

Table 4-12. ORB Parameters used in the CSL Experiment.

Parameter	Value
$nFeatures$	1000
$nLevels$	8
$scaleFactor$	1.2
$edgeThreshold$	19
$maxFastThresh$	20
$minFastThresh$	7
$patchSize$	31

As presented in Figure 4-19, ORB-SLAM3 fails to find and track features and resets, whereas the developed algorithm demonstrates robustness and estimates correctly the camera pose thanks to the features available through the rear camera.

(a) Left lenses (front and rear), our algorithm



(b) Left lens, ORB-SLAM3

Figure 4-19. Feature tracking, (a) When the front camera is covered by a featureless object the algorithm continues tracking the available features from the second camera, (b) ORB-SLAM3 resets when a featureless object covers most of the camera's FoV.

In Figure 4-20, the trajectory estimation of the developed algorithm and the resetting point of ORB-SLAM3 are presented. The grapevines are not shown because they are located at $x = -1, x = 1$ and the rover moves in the middle. The developed visual SLAM algorithm continues the camera pose estimation without losing accuracy and performs global BA when the Apriltag is detected for the second time for further optimization of the camera trajectory.

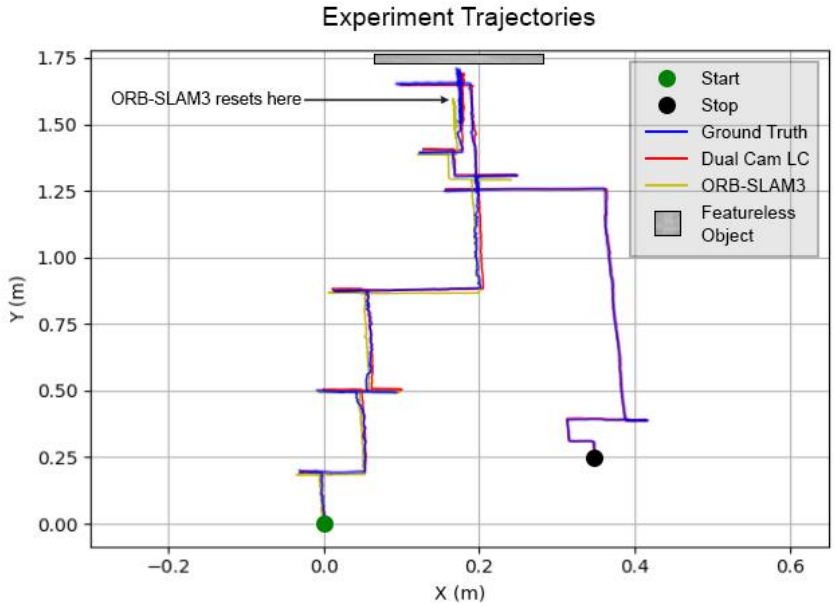


Figure 4-20. Experiment at the synthetic vineyard at CSL. ORB-SLAM3 resets since there are very few features available to track. The developed algorithm continues thanks to the features available from the rear camera.

4.8 Timings

The execution time of each process is an important part to ensure that the developed algorithm is suitable for a real time application. The processing times for each operation were calculated as an average of five runs and are presented in Table 4-13. While the Single stereo camera operation extracts features only from the left and right images, the processing time required for feature extraction is comparable to that of the dual stereo camera operation. This is because the feature extraction process for each stereo camera is separated into different threads. The processing times for the dual camera setup are improved by separating every process for each camera into different threads (the processes that can be completed separately).

Table 4-13. Processing Times of the Developed Algorithm in Different Datasets.

	Single Camera Operation	**Single Camera Operation**	**Dual Camera Operation**
Dataset	**EuRoC**	**KITTI**	**CSL Experiment**
Resolution	$752x480$	$1226x370$	$640x360$
Camera FPS	$20Hz$	$10Hz$	$15Hz$
Features	1000	2000	1000
Tracking Thread			
Feature Extraction	$16.6344ms$	$22.4063ms$	$27.5005ms$
Stereo Matching	$4.43852ms$	$8.67086ms$	$1.18557ms$
Predict Position	$0.1087474ms$	$0.1577892ms$	$1.0875ms$
Match Features	$0.238904ms$	$0.49884ms$	$0.776784ms$
Estimate Pose	$3.30832ms$	$5.3741ms$	$6.12086ms$
Insert Keyframe	$0.29586ms$	$0.655361ms$	$0.0675693ms$
Remove Outliers	$0.0179947ms$	$0.0318811ms$	$0.0117192ms$
Total	$\mathbf{25.1363226ms}$	$\mathbf{37,7951313ms}$	$\mathbf{36.7505025ms}$
Local Mapping Thread			
New Mappoints	$8.24918ms$	$10.6435ms$	$9.58088ms$
Local BA	$103.721ms$	$165.579ms$	$321.473ms$
Total	$\mathbf{111,97018ms}$	$\mathbf{176,2225ms}$	$\mathbf{331,05388ms}$
Loop Closing Thread			
Global BA	-	-	$1138.63ms$

In the ORB-SLAM3 paper [7] the only processing times provided are for the EuRoC dataset and they are presented in Table 4-14.

Table 4-14. Processing Times of ORB-SLAM3 in the EuRoC Dataset [7].

Dataset	EuRoC
Resolution	$752x480$
Camera FPS	$20Hz$
Features	1200
Tracking Thread	
Feature Extraction	$15.68ms$
Stereo Matching	$3.35ms$
Predict Position	$2.69ms$
Match Features + Estimate Pose	$6.31ms$
Insert Keyframe	$0.12ms$
Total	$\mathbf{25.1363226ms}$
Local Mapping Thread	
Insert KF	$8.03ms$
Remove Mappoints	$0.32ms$
Add new Mappoints	$18.23ms$
Local BA	$134.60ms$
Remove KF	$5.49ms$
Total	$\mathbf{158.84ms}$
Global BA	-

As is evident, the developed algorithm demonstrates comparable performance to the ORB-SLAM3 in terms of processing times in the EuRoC Dataset. The average processing time per frame is lower than the camera's FPS, ensuring real time operation capabilities. Even during dual camera operation, while there is a slight increase in processing time, it remains below the camera's FPS, maintaining real-time functionality.

5 Conclusion And Future Work

5.1 Conclusion

Visual Simultaneous Localization and Mapping (SLAM) is a well-researched field in robotics and computer vision that involves the mapping of an unknown environment while simultaneously localizing the robot within it using visual sensors. Over the years, various approaches have been developed to address this challenge and provide an accurate way of navigating and mapping an unknown environment. However, many of these approaches suffer from limitations such as featureless objects covering the field of view of the cameras, or incorrect loop detections, due to the homogeneity of the environment. This thesis aims to address these limitations by utilizing a dual stereo camera setup and a new and robust way of detecting loop closures. The setup and tuning of the algorithm are focused on vineyard rows that allow loop detection with apriltags, other environments may have different requirements. The developed algorithm achieves impressive results on well-known datasets as well as on simulated and real-world environments.

With a dual camera setup, the camera pose can be estimated by utilizing the features from both stereo cameras, continuing the correct pose estimation where the single stereo camera setup may fail. Additionally, the second stereo camera assists in the mapping of the environment, providing a more detailed and accurate map, as it provides an additional perspective.

Another key aspect of the developed algorithm is the loop closure detection. Loop closures are crucial for reducing the error accumulation in the visual SLAM algorithm over time, and for producing consistent and accurate maps. In this thesis, a new and robust way of detecting loop closures using Apriltags was developed. By utilizing Apriltags for loop closure detection, the developed algorithm can detect loop closures accurately and efficiently, even in homogeneous environments where other state-of-the-art approaches may fail.

To evaluate the performance of the developed single stereo camera algorithm, the KITTI and EuRoC datasets were used, which are widely used in the visual SLAM community. The results show that the developed single stereo camera algorithm achieves impressive accuracy, even when tested on challenging datasets that contain images with rapid camera movement and difficult lighting conditions.

To test the algorithm's performance, on the dual stereo camera setup with the Apriltag loop closure detection, custom datasets were created, in simulation and in real-world scenarios. The results show that the developed algorithm shows impressive accuracy in all experiments and outperforms ORB-SLAM3 in some scenarios, i.e.: when the cameras field of view is partially obscured or when the bag-of-words approaches incorrectly detect loops due to the similarities in the environment. The developed algorithm is also tested and can be used on-line at 15 FPS with resolution $640x360$ on the dual camera setup.

In conclusion, the developed algorithm addresses core limitations of visual SLAM and shows robustness in a variety of challenging environments. However, there is always room for further research and development. Future work can focus on several directions to improve the accuracy, robustness, and efficiency of the developed algorithm.

5.2 Future Work

The first possible extension of this thesis is the integration of additional sensors such as an inertial measurement unit (IMU). IMUs operate at a higher frequency than visual sensors and can provide pose information faster, and therefore, improve the estimation. IMUs can additionally be employed for a better prediction of the next camera pose, making feature tracking more accurate.

To improve camera pose estimation and mapping of the environment, another potential area of future research could be scaling the dual camera setup to include more stereo cameras. With an increased number of stereo cameras, the FoV is increased, providing more features to track and as a result the pose estimation accuracy and the mapping of the environment could be greatly improved. However, this will come at the cost of increased computational complexity, requiring more processing power and the need for better computer hardware.

To optimize this visual SLAM algorithm for its primary use case, which is the inspection of vineyards, the loop closure detection method can be improved further. One possible enhancement is to increase the number of Apriltags being detected. For instance, an Apriltag can be placed at the start of each row of grapevines. As the robot traverses each row, it can perform a loop closure at the end, leading to a more accurate map and camera pose estimation. Additionally, a grape inspection method can be developed and integrated to the visual SLAM algorithm. While the robot traverses the vineyard, the grape inspection algorithm can help monitor the condition of the grapes with precision and prevent diseases from spreading, if one has been detected. This improvement would make the algorithm more suitable for vineyard inspection and similar applications.

Another potential enhancement is the integration of a path planner to operate at the same time with the visual SLAM algorithm. This path planner can receive information about the environment, such as the presence of obstacles or the terrain of the vineyard, from the visual SLAM algorithm and plan the path of the robot according to the current task being performed.

Furthermore, integrating a GPS sensor can provide additional location. By combining the GPS data with the visual SLAM algorithm's estimates, the robot's pose can be estimated more accurately, improving the robot's ability to navigate through the environment.

Last but not least, the processing times should be profiled and optimized to ensure that the visual SLAM algorithm can operate optimally. This would enable the algorithm to perform efficiently on hardware with reduced capabilities compared to the ones tested and expand the potential applications of the algorithm to various platforms.

As I conclude my thesis, I am excited about the endless possibilities for the future of visual SLAM and autonomous robotics. While there are many unknowns, I hope that the work I have done will contribute to the advancement of the field, and I will look with great interest to see what the future holds.

6 References

[1] G. Bledt, M. Powell, B. Katz, J. Carlo, P. Wensing, and S. Kim, “MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot,” Oct. 2018. doi: 10.1109/IROS.2018.8593885.

[2] H.-W. Park, P. M. Wensing, and S. Kim, “Jumping over obstacles with MIT Cheetah 2,” *Robot. Auton. Syst.*, vol. 136, p. 103703, Feb. 2021, doi: 10.1016/j.robot.2020.103703.

[3] “Spot® - The Agile Mobile Robot,” *Boston Dynamics*. https://www.bostondynamics.com/products/spot (accessed Apr. 05, 2023).

[4] “ANYmal X: Ex-Proof Inspection Robot,” *ANYbotics*. https://www.anybotics.com/anymal-ex-proof-inspection-robot/ (accessed Apr. 07, 2023).

[5] “Husky UGV - Outdoor Field Research Robot by Clearpath,” *Clearpath Robotics*. https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/ (accessed Apr. 07, 2023).

[6] “DJI Mavic 3 - Imaging Above Everything,” *DJI*. https://www.dji.com/gr/photo (accessed Apr. 07, 2023).

[7] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM,” *IEEE Trans. Robot.*, vol. 37, no. 6, pp. 1874–1890, Dec. 2021, doi: 10.1109/TRO.2021.3075644.

[8] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM: A Versatile and Accurate Monocular SLAM System,” *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015, doi: 10.1109/TRO.2015.2463671.

[9] R. Mur-Artal and J. D. Tardos, “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras,” *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103.

[10] D. Galvez-López and J. D. Tardos, “Bags of Binary Words for Fast Place Recognition in Image Sequences,” *IEEE Trans. Robot.*, vol. 28, no. 5, pp. 1188–1197, Oct. 2012, doi: 10.1109/TRO.2012.2197158.

[11] T. Qin and S. Shen, “Online Temporal Calibration for Monocular Visual-Inertial Systems,” *arXiv.org*, Aug. 02, 2018. https://arxiv.org/abs/1808.00692v1 (accessed Apr. 08, 2023).

[12] T. Qin, P. Li, and S. Shen, “VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator,” *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018, doi: 10.1109/TRO.2018.2853729.

[13] G. Klein and D. Murray, “Parallel Tracking and Mapping for Small AR Workspaces,” in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, Nov. 2007, pp. 225–234. doi: 10.1109/ISMAR.2007.4538852.

[14] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, “Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping.” arXiv, Mar. 03, 2020. doi: 10.48550/arXiv.1910.02490.

[15] C. Forster, M. Pizzoli, and D. Scaramuzza, “SVO: Fast semi-direct monocular visual odometry,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 15–22. doi: 10.1109/ICRA.2014.6906584.

[16] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, “SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems,” *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 249–265, Apr. 2017, doi: 10.1109/TRO.2016.2623335.

[17] J. Kuo, M. Muglikar, Z. Zhang, and D. Scaramuzza, “Redesigning SLAM for Arbitrary Multi-Camera Systems,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 2116–2122. doi: 10.1109/ICRA40945.2020.9197553.

[18] “Stereo camera,” *Wikipedia*. Mar. 23, 2023. Accessed: Apr. 05, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stereo_camera&oldid=1146154963

[19] “Pinhole camera,” *Wikipedia*. May 16, 2023. Accessed: May 17, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pinhole_camera&oldid=1155128872

[20] “Fisheye lens,” *Wikipedia*. Apr. 08, 2023. Accessed: May 17, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Fisheye_lens&oldid=1148832126

[21] “Image Undistortion - MATLAB & Simulink.” https://www.mathworks.com/help/visionhdl/ug/image-undistort.html (accessed Jun. 06, 2023).

[22] “OpenCV: Camera Calibration.” https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html (accessed Jun. 06, 2023).

[23] W. Burger, *Zhang’s Camera Calibration Algorithm: In-Depth Tutorial and Implementation*. 2016. doi: 10.13140/RG.2.1.1166.1688/1.

[24] “Camera Calibration Toolbox for Matlab.” http://robots.stanford.edu/cs223b04/JeanYvesCalib/ (accessed May 17, 2023).

[25] “What Is Camera Calibration? - MATLAB & Simulink.” https://www.mathworks.com/help/vision/ug/camera-calibration.html (accessed Jul. 23, 2023).
[26] “Image rectification,” *Wikipedia*. Mar. 10, 2023. Accessed: Apr. 11, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Image_rectification&oldid=1143835516
[27] “Lab: Stereo.” https://cs.brown.edu/courses/cs129/labs/lab_stereolab/ (accessed Jun. 06, 2023).
[28] “Epipolar geometry,” *Wikipedia*. Dec. 04, 2022. Accessed: Apr. 11, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Epipolar_geometry&oldid=1125486185
[29] H. Hirschmuller, “Accurate and efficient stereo processing by semi-global matching and mutual information,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, Jun. 2005, pp. 807–814 vol. 2. doi: 10.1109/CVPR.2005.56.
[30] A. Garg, “Stereo Vision: Depth Estimation between object and camera,” *Analytics Vidhya*, Feb. 25, 2022. https://medium.com/analytics-vidhya/distance-estimation-cf2f2fd709d8 (accessed Jul. 23, 2023).
[31] “OpenCV: Depth Map from Stereo Images.” https://docs.opencv.org/4.x/dd/d53/tutorial_py_depthmap.html (accessed Jun. 06, 2023).
[32] Ashish, “Understanding Edge Detection (Sobel Operator),” *Medium*, Sep. 26, 2018. https://medium.datadriveninvestor.com/understanding-edge-detection-sobel-operator-2aada303b900 (accessed Jun. 06, 2023).
[33] T. R. Society, “Blob Detection,” *COMPUTER VISION & ROBOTICS*, Oct. 15, 2019. https://medium.com/image-processing-in-robotics/blob-detection-309226a3ea5b (accessed Jun. 06, 2023).
[34] “How to accomplish corner detection in C#,” *Ozeki Camera SDK*. https://camera-sdk.com/p_6756-how-to-accomplish-corner-detection-in-c-sharp.html (accessed Jun. 06, 2023).
[35] D. Tyagi, “Introduction to Harris Corner Detector,” *Data Breach*, Apr. 07, 2020. https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6 (accessed Jun. 06, 2023).
[36] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Sep. 1999, pp. 1150–1157 vol.2. doi: 10.1109/ICCV.1999.790410.
[37] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded Up Robust Features,” in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 404–417. doi: 10.1007/11744023_32.
[38] “OpenCV: Introduction to SURF (Speeded-Up Robust Features).” https://docs.opencv.org/3.4/df/dd2/tutorial_py_surf_intro.html (accessed Apr. 13, 2023).
[39] “Midpoint circle algorithm,” *Wikipedia*. Apr. 23, 2023. Accessed: Jun. 17, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Midpoint_circle_algorithm&oldid=1151318349
[40] “Features from accelerated segment test,” *Wikipedia*. Nov. 12, 2022. Accessed: Jun. 06, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Features_from_accelerated_segment_test&oldid=1121443700
[41] E. Rosten and T. Drummond, “Machine Learning for High-Speed Corner Detection,” in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 430–443. doi: 10.1007/11744023_34.
[42] “OpenCV: BRIEF (Binary Robust Independent Elementary Features).” https://docs.opencv.org/3.4/dc/d7d/tutorial_py_brief.html (accessed May 29, 2023).
[43] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” in *2011 International Conference on Computer Vision*, Nov. 2011, pp. 2564–2571. doi: 10.1109/ICCV.2011.6126544.
[44] R. Pillay <ruven@users.sourceforge.io>, “IIPImage » Images,” *IIPImage*. https://iipimage.sourceforge.io/documentation/images (accessed Jun. 06, 2023).
[45] D. Tyagi, “Introduction to ORB (Oriented FAST and Rotated BRIEF),” *Data Breach*, Apr. 07, 2020. https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf (accessed Jul. 23, 2023).
[46] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla, “Evaluation of low-complexity visual feature detectors and descriptors,” presented at the 2013 18th International Conference on Digital Signal Processing, DSP 2013, Jul. 2013, pp. 1–7. doi: 10.1109/ICDSP.2013.6622757.
[47] “Detect SURF features - MATLAB detectSURFFeatures.” https://www.mathworks.com/help/vision/ref/detectsurffeatures.html (accessed Jun. 06, 2023).

[48] “OpenCV: cv::BFMatcher Class Reference.” https://docs.opencv.org/4.x/d3/da1/classcv_1_1BFMatcher.html (accessed Apr. 13, 2023).
[49] “OpenCV: Feature Matching.” https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html (accessed Jun. 06, 2023).
[50] “OpenCV: cv::FlannBasedMatcher Class Reference.” https://docs.opencv.org/3.4/dc/de2/classcv_1_1FlannBasedMatcher.html (accessed Apr. 13, 2023).
[51] “OpenCV: Feature Matching with FLANN.” https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html (accessed Jun. 06, 2023).
[52] “COMPSCI773S1T: Vision Guided Control.” https://www.cs.auckland.ac.nz/courses/compsci773s1t/lectures/773-GG/topCS773.htm (accessed Jun. 06, 2023).
[53] “Reprojection error,” *Wikipedia*. Mar. 28, 2023. Accessed: Jun. 06, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reprojection_error&oldid=1146983350
[54] “The Reprojection Error?,” *camcalib*, Apr. 11, 2022. https://www.camcalib.io/post/what-is-the-reprojection-error (accessed Jun. 06, 2023).
[55] “OpenCV: Perspective-n-Point (PnP) pose computation.” https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html (accessed Apr. 14, 2023).
[56] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, “Bundle Adjustment — A Modern Synthesis,” in *Vision Algorithms: Theory and Practice*, B. Triggs, A. Zisserman, and R. Szeliski, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 298–372. doi: 10.1007/3-540-44480-7_21.
[57] J. J. Moré, “The Levenberg-Marquardt algorithm: Implementation and theory,” in *Numerical Analysis*, G. A. Watson, Ed., in Lecture Notes in Mathematics. Berlin, Heidelberg: Springer, 1978, pp. 105–116. doi: 10.1007/BFb0067700.
[58] F. Zhang, Ed., *The Schur Complement and Its Applications*, vol. 4. in Numerical Methods and Algorithms, vol. 4. New York: Springer-Verlag, 2005. doi: 10.1007/b105056.
[59] R. Azzam, T. Taha, S. Huang, and Y. Zweiri, “Feature-based visual simultaneous localization and mapping: a survey,” *SN Appl. Sci.*, vol. 2, Feb. 2020, doi: 10.1007/s42452-020-2001-3.
[60] “Bundle adjustment,” *Wikipedia*. Apr. 17, 2023. Accessed: Jun. 17, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Bundle_adjustment&oldid=1150309618
[61] L. Sheng, D. Xu, W. Ouyang, and X. Wang, “Unsupervised Collaborative Learning of Keyframe Detection and Visual Odometry Towards Monocular Deep SLAM,” Oct. 2019, pp. 4301–4310. doi: 10.1109/ICCV.2019.00440.
[62] V. Behret and F. Palme, “Examination of Real World Bias on the Localization Accuracy of Indirect SLAM,” 2019. doi: 10.13140/RG.2.2.13828.27528.
[63] “Simple bag-of-words loop closure for visual SLAM.” https://nicolovaligi.com/articles/bag-of-words-loop-closure-visual-slam/ (accessed Jun. 06, 2023).
[64] E. Olson, “AprilTag: A robust and flexible visual fiducial system,” in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 3400–3407. doi: 10.1109/ICRA.2011.5979561.
[65] J. Wang and E. Olson, “AprilTag 2: Efficient and robust fiducial detection,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 4193–4198. doi: 10.1109/IROS.2016.7759617.
[66] M. Krogius, A. Haggenmiller, and E. Olson, “Flexible Layouts for Fiducial Tags,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 1898–1903. doi: 10.1109/IROS40897.2019.8967787.
[67] “AprilTag 3.” AprilRobotics, May 29, 2023. Accessed: May 29, 2023. [Online]. Available: https://github.com/AprilRobotics/apriltag
[68] “AprilTag.” https://april.eecs.umich.edu/software/apriltag (accessed Jun. 06, 2023).
[69] “apriltag_ros.” AprilRobotics, Apr. 18, 2023. Accessed: Apr. 20, 2023. [Online]. Available: https://github.com/AprilRobotics/apriltag_ros
[70] fspindle, “AprilTag integration – ViSP.” https://visp.inria.fr/apriltag-integration/ (accessed Jun. 06, 2023).
[71] S. Agarwal, K. Mierle, and The Ceres Solver Team, “Ceres Solver.” Mar. 2022. Accessed: Apr. 16, 2023. [Online]. Available: https://github.com/ceres-solver/ceres-solver
[72] “vSLAM.drawio - draw.io.” https://app.diagrams.net/ (accessed Jun. 19, 2023).
[73] “ROS: Home.” https://www.ros.org/ (accessed Apr. 20, 2023).

[74] A. Anwar, “Part 3: Create Your First ROS Publisher and Subscriber Nodes,” *The Startup*, Feb. 15, 2021. https://medium.com/swlh/part-3-create-your-first-ros-publisher-and-subscriber-nodes-2e833dea7598 (accessed Jun. 17, 2023).
[75] cetus, “Event Publishing and Subscribing with ActiveMQ and M2MQTT,” *Ketek*, Jan. 13, 2015. https://www.ketek.ro/2015/01/13/event-publishing-and-subscribing-with-activemq-m2mqtt/ (accessed Jun. 06, 2023).
[76] “rosbag - ROS Wiki.” http://wiki.ros.org/rosbag (accessed Apr. 20, 2023).
[77] “OpenCV: cv::ORB Class Reference.” https://docs.opencv.org/3.4/db/d95/classcv_1_1ORB.html (accessed Apr. 16, 2023).
[78] “OpenCV: cv::FastFeatureDetector Class Reference.” https://docs.opencv.org/3.4/df/d74/classcv_1_1FastFeatureDetector.html (accessed Apr. 16, 2023).
[79] O. Bailo, F. Rameau, K. Joo, J. Park, O. Bogdan, and I. S. Kweon, “Efficient adaptive non-maximal suppression algorithms for homogeneous spatial keypoint distribution,” *Pattern Recognit. Lett.*, vol. 106, pp. 53–60, Apr. 2018, doi: 10.1016/j.patrec.2018.02.020.
[80] “Euclidean group,” *Wikipedia*. Sep. 21, 2022. Accessed: Apr. 19, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Euclidean_group&oldid=1111504010
[81] L. M. Paz, P. PiniÉs, J. D. TardÓs, and J. Neira, “Large-Scale 6-DOF SLAM With Stereo-in-Hand,” *IEEE Trans. Robot.*, vol. 24, no. 5, pp. 946–957, Oct. 2008, doi: 10.1109/TRO.2008.2004637.
[82] P. J. Huber, “Robust Estimation of a Location Parameter,” *Ann. Math. Stat.*, vol. 35, no. 1, pp. 73–101, Mar. 1964, doi: 10.1214/aoms/1177703732.
[83] S. Lovegrove, “What is Pangolin.” Apr. 20, 2023. Accessed: Apr. 20, 2023. [Online]. Available: https://github.com/stevenlovegrove/Pangolin
[84] N. Koenig and A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Sep. 2004, pp. 2149–2154 vol.3. doi: 10.1109/IROS.2004.1389727.
[85] “Vineyard 3D Models for Download | TurboSquid.” https://www.turbosquid.com/3d-model/vineyard (accessed Apr. 22, 2023).
[86] “4 in. HD Mecanum Wheels,” Feb. 18, 2022. https://www.andymark.com/products/4-in-hd-mecanum-wheel-set-options (accessed Apr. 21, 2023).
[87] “Pololu - RoboClaw 2x30A Motor Controller (V5E).” https://www.pololu.com/product/3286 (accessed Apr. 21, 2023).
[88] M. Menze and A. Geiger, “Object scene flow for autonomous vehicles,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 3061–3070. doi: 10.1109/CVPR.2015.7298925.
[89] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: the KITTI dataset,” *Int. J. Robot. Res.*, vol. 32, pp. 1231–1237, Sep. 2013, doi: 10.1177/0278364913491297.
[90] M. Burri *et al.*, “The EuRoC micro aerial vehicle datasets,” *Int. J. Robot. Res.*, vol. 35, Jan. 2016, doi: 10.1177/0278364915620033.
[91] “ZED 2 - AI Stereo Camera.” https://www.stereolabs.com/zed-2/ (accessed Apr. 22, 2023).
[92] “Creative Live! Cam Sync 1080p,” *Creative Store - Greece*. https://gr.creative.com/p/peripherals/creative-live-cam-sync-1080p (accessed Apr. 22, 2023).

Appendix A.

DC-VSLAM is a multicamera visual SLAM designed for vineyard inspection. To address the challenge of homogeneous environments, loop closures are detected using AprilTags.
DC-VSLAM has been tested with OpenCV 4.2.0, Eigen 3.3.7 on Ubuntu 20.04 with ROS Noetic.

Installation

DC-VSLAM has many dependencies, and all can be downloaded using a script provided in this repo. ROS Noetic is needed for the installation.

We recommend the users to create an empty workspace. Clone the package on the catkin workspace and run the build script. Python 3 has to be set as default for Pangolin installation.

```
cd ${WORKSPACE_PATH}/src
git clone https://ChristosKokas@bitbucket.org/csl_legged/dc-vslam-
case2023.git
cd dc-vslam-case2023
chmod +x build.sh
./build.sh
```

Quick Start

Several launch files are provided. The RT denotes real-time and the AT denotes the use of AprilTag Loop Closure. Change the launch files to match the config file name and the topic of the image msgs for AprilTag detection.

DC-VSLAM can run both with images and with rosbags. Images need to be provided as presented below (the bullets are folders):

- Dc-vslam-case2023
- Images
    - Dataset_name
    - Left
        - 000000.jpg(.png)
        - 000001.jpg(.png)
        - …
    - Right
        - 000000.jpg(.png)
        - 000001.jpg(.png)
        - …
    - leftBack
        - 000000.jpg(.png)
        - 000001.jpg(.png)
        - …
    - rightBack
        - 000000.jpg(.png)
        - 000001.jpg(.png)
        - …

And the full path to the dataset folder has to be provided in the config file.

Replaying Recorded Experiments

Rosbags for each experiment can be downloaded from https://centralntuagr-my.sharepoint.com/:f:/g/personal/amast_central_ntua_gr/Ehrrt-IUFB5DsIH4nJnh6tUBfJvtzW-FmX1IaixmhvRuSg?e=G41m63

Version 1 of this developed algorithm can be found at the CSL team bitbucket repository at https://bitbucket.org/csl_legged/dc-vslam-case2023/commits/ with version v1.0.