Master Thesis

Vision-based motion estimation for mobile robot navigation in harsh environment

carried out for the purpose of obtaining the degree of

Diplom-Ingenieur_in (DI) Mechanical and Mechanical Management

submitted at TU Wien

Faculty of Mechanical and Industrial Engineering

by

Utkarsh Savkare

Mat.No.: 12202174

under the supervision of

Univ.-Prof. Dr.-Ing. Sebastian Schlund

(E330 Institute of Management Science, Head of Department Industrial Engineering
Head of Research Group Human-Machine-Interaction)

PhD Ambra Vandone

(Research fellow at Scuola Universitaria Professionale Della Svizzera Italiana)

Wien, 3rd November 2023

[redacted]

Signature

Ich habe zur Kenntnis genommen, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

Vision-based motion estimation for mobile robot navigation in harsh environment

nur mit Bewilligung der Prüfungskommission berechtigt bin.

Ich erkläre weiters Eides statt, dass ich meine Diplomarbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen selbstständig ausgeführt habe und alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, genannt habe.Weiters erkläre ich, dass ich dieses Diplomarbeitsthema bisher weder im In- noch Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Wien, 3rd November 2023

_____

Signature

Acknowledgements

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of my Master's thesis. This journey has been challenging, yet immensely rewarding, and I owe my accomplishments to the support and encouragement of numerous individuals and organizations.

First and foremost, I am deeply indebted to my thesis advisor **Miss Ambra Vandone**, for their unwavering guidance, mentorship, and invaluable insights throughout this research endeavour. Their expertise and dedication have played a pivotal role in shaping the direction of my study and in nurturing my academic growth. I am truly fortunate to have had such a knowledgeable and supportive mentor.

In line, I am extremely grateful to my supervisor **Univ.-Prof. Dr.-Ing. Sebastian Schlund**, who gave me valuable insight throughout my thesis duration.

I would like to extend my gratitude to the members of my thesis committee, from the Institute of Management Science for their valuable feedback, constructive criticism, and scholarly contributions to this work. Their collective wisdom has been instrumental in refining the quality of my thesis.

I would like to extend my sincere thanks to Matteo Astori from MCH-Tronics Sagl, Manno, for their invaluable scientific support.

I am also thankful to my fellow students and colleagues at Technische Universität Wien (TUW) and Scuola universitaria professionale della Svizzera italiana (SUPSI) for their camaraderie, stimulating discussions, and the shared learning experiences that have enriched my academic journey.

English Abstract

Visual Odometry (VO) is a process of estimating the pose of a moving agent relying solely on analysis of the image streams from one or multiple onboard cameras. It finds diverse applications in Robotics Navigation for both outdoor and Indoor settings, Environmental 3D Reconstruction, Autonomous Vehicles, Augmented Reality (AR), Virtual Reality (VR), and Camera Stabilization. It facilitates safe and efficient navigation, exploration, and interaction with the environment. In outdoor scenarios, stereo visual odometry enables navigating through varied terrains making informed decisions for obstacle avoidance and path planning. This research work focuses on developing a robust stereo visual odometry algorithm for mobile robot navigation in outdoor environments, which can be further complimented by external sensors. The research's primary objective is to create a stereo vision algorithm robustly estimating robot motion based on stereo camera data. The algorithm incorporates state-of-the-art feature detection, tracking, and filtering methods, enhancing the robot's environmental perception and navigation efficiency. By investigating these techniques and integrating them into the stereo visual odometry algorithm, the research aims to improve accuracy and robustness outdoor environments. Extensive real-world dataset evaluations confirm the algorithm's effectiveness in providing incremental online estimation of the robot's position and orientation, showcasing its potential for safer and more reliable navigation in various real-world applications. However, we also identified certain shortcomings during these tests and have diligently sought solutions for them. We rigorously evaluated the algorithm's effectiveness through individual assessments, aiming to uncover its strengths and limitations. These assessments shed light on its performance across a spectrum of real-world scenarios, providing valuable insights for further refinement and optimization. By substantiating its performance through real-world datasets and self-conducted experiments, this research offers comprehensive evidence of its potential. It significantly contributes to the field of stereo visual odometry by introducing a scalable and efficient algorithm. This innovation stands to advance robotics, autonomous vehicles, and augmented reality systems, ultimately improving our ability to seamlessly explore and interact with outdoor environments.

Keywords - Stereo Visual Odometry, Pose Estimation, Mobile Robotics Navigation, Outdoor Environments, Incremental Online Estimation, Environmental Perception.

Deutsche Kurzfassung

Visuelle Odometrie (VO) ist ein Prozess zur Schätzung der Pose eines sich bewegenden Agenten, der ausschließlich auf der Analyse der Bildströme einer oder mehrerer Bordkameras beruht. Es findet vielfältige Anwendungen in der Robotik-Navigation für Außen- und Innenumgebungen, der 3D-Rekonstruktion der Umgebung, autonomen Fahrzeugen, Augmented Reality (AR), Virtual Reality (VR) und Kamerastabilisierung. Es ermöglicht eine sichere und effiziente Navigation, Erkundung und Interaktion mit der Umwelt. In Outdoor-Szenarien ermöglicht die visuelle Stereo-Odometrie die Navigation durch unterschiedliches Gelände und trifft fundierte Entscheidungen zur Vermeidung von Hindernissen und zur Wegplanung. Diese Forschungsarbeit konzentriert sich auf die Entwicklung eines robusten stereovisuellen Odometrie-Algorithmus für die mobile Roboternavigation in Außenumgebungen, der durch externe Sensoren weiter ergänzt werden kann. Das Hauptziel der Forschung besteht darin, einen Stereo-Vision-Algorithmus zu entwickeln, der die Roboterbewegung auf der Grundlage von Stereokameradaten zuverlässig schätzt. Der Algorithmus umfasst modernste Methoden zur Erkennung, Verfolgung und Filterung von Merkmalen und verbessert so die Umweltwahrnehmung und Navigationseffizienz des Roboters. Durch die Untersuchung dieser Techniken und deren Integration in den stereovisuellen Odometrie-Algorithmus zielt die Forschung darauf ab, die Genauigkeit und Robustheit von Außenumgebungen zu verbessern. Umfangreiche Auswertungen realer Datensätze bestätigen die Wirksamkeit des Algorithmus bei der Bereitstellung einer inkrementellen Online-Schätzung der Position und Ausrichtung des Roboters und zeigen sein Potenzial für eine sicherere und zuverlässigere Navigation in verschiedenen realen Anwendungen. Allerdings haben wir bei diesen Tests auch gewisse Mängel festgestellt und sorgfältig nach Lösungen dafür gesucht. Wir haben die Wirksamkeit des Algorithmus durch Einzelbewertungen gründlich bewertet, um seine Stärken und Grenzen aufzudecken. Diese Bewertungen geben Aufschluss über die Leistung in einem Spektrum realer Szenarien und liefern wertvolle Erkenntnisse für die weitere Verfeinerung und Optimierung. Durch die Untermauerung seiner Leistung durch reale Datensätze und selbst durchgeführte Experimente liefert diese Forschung umfassende Beweise für ihr Potenzial. Durch die Einführung eines skalierbaren und effizienten Algorithmus leistet es einen wesentlichen Beitrag zum Bereich der stereovisuellen Odometrie. Diese Innovation soll Robotik, autonome Fahrzeuge und Augmented-Reality-Systeme voranbringen und letztendlich unsere Fähigkeit verbessern, Außenumgebungen nahtlos zu erkunden und mit ihnen zu interagieren.

Table of Contents

Page

Contents

List of Figures

List of Tables

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
ADC	Analog to Digital Conversion
AR	Augmented Reality
ATE	Absolute Trajectory Error
BRIEF	Binary Robust Independent Elementary Features
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DDR4	Double Data Rate Synchronous Dynamic -4
DOF	Degree Of Freedom
DSO	Direct Sparse Odometry
ESVO	Event-Based Stereo Visual Odometry
FAST	Features from Accelerated Segment Test
FLANN	Fast Library for Approximate Nearest Neighbours
FOV	Field Of View
GEM	Global Electric Motorcars
GPS	Global Positioning System
GPU	Graphics Processing Unit
HDL	High-Definition Lidar
HMD	Head Mounted Display
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago
LAGR	Learning Applied to Ground Vehicles
LIDAR	Light Detection and Ranging
LM	Levenberg–Marquardt
NASA	National Aeronautics and Space Administration
NCC	Normalized Cross-Correlation
ORB	Oriented FAST and rotated BRIEF
PLWM	Point Line Weight Mechanism

RAM	Random-Access Memory
RANSAC	Random sample consensus
RGB	Red-Green-Blue
RMSE	Root Mean Square Error
ROV	Remotely Operated Vehicle
RPE	Relative Pose Error
SDK	Software Development Kit
SGBM	Semi-Global Block Matching
SGM	Semi-Global Matching
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization And Mapping
SOFT2	Stereo Odometry based on careful Feature selection and Tracking
SSD	Solid-State Drive
SURF	Speed Up Robust Features
SVO	Stereo Visual Odometry
VGA	Video Graphics ArrayVGA
VIO	Visual-Inertial Odometry
VO	Visual Odometry
VR	Virtual Reality

1 Introduction

The two main types of Visual Odometry (VO) are monocular VO and stereo VO. Monocular VO uses a single camera to track features in a sequence of images and use the motion of these features to estimate the camera pose. Stereo VO uses two cameras to obtain depth information, which can improve the accuracy and robustness of the pose estimation.

This Chapter describes the motivation and problem definition in section 1.1, main research questions and objectives in section 1.2 and the scope and limitations of the thesis in section 1.3 lastly, the section 1.4 provides a brief outline of the thesis document.

1.1 Motivation and Problem Definition

The demand for robust and accurate pose estimation techniques has increased due to the growing popularity of mobile robotics, autonomous vehicles, and augmented reality applications (R. Arun, 2005). Visual Odometry (VO, here after) has emerged as a critical technology to meet this demand, as it allows these systems to navigate and interact with their environment based solely on visual information from cameras (G. Carlone, 2018). VO has found applications in various fields such as Robotics Navigation, 3D Reconstruction, Autonomous Vehicles, Augmented Reality (AR), Virtual Reality (VR), and Camera Stabilization, enabling safe and efficient navigation, exploration, and interaction with the surroundings. However, outdoor environments pose unique challenges that require sophisticated solutions for VO (J. Engel, 2014). Factors such as varying lighting conditions, uneven surfaces, and dynamic changes in the environment can impact the accuracy of motion estimation. One particularly significant challenge is scale ambiguity, which is caused by changes in image scaling factors due to variations in terrain gradient. This ambiguity can lead to inaccurate pose estimation in dynamic outdoor scenarios. To address these challenges, researchers have proposed various techniques and algorithms (C. Forster, 2014). One approach is to use multiple cameras to improve the accuracy of pose estimation (F. Li, 2020). Stereo vision leverages the power of dual cameras to capture depth information through triangulation, which enhances the accuracy and robustness of motion estimation. This distinct advantage becomes even more pronounced when compared to alternative methods. Some methodologies incorporate additional sensors, such as LiDAR providing depth information that can be combined with visual data to enhance the robustness of VO algorithms and bolster accuracy but stereo visual odometry offers a self-contained solution that leverages purely visual cues. Another approach is to incorporate machine learning techniques to adapt to the specific challenges of outdoor environments (R. Mur-Artal, 2017). While Deep learning algorithms, like Convolutional Neural Networks (CNNs), have shown promising results in improving the accuracy of VO in challenging conditions, they are highly dependent on the training dataset and computationally expensive. Furthermore, researchers have also focused on developing methods to handle scale ambiguity in outdoor environments (J. Engel, 2014). One approach is to utilize

additional sensors, such as GPS or IMU, to provide scale information and reduce the ambiguity (R. Mur-Artal, 2017) . However, IMU pose propagation is sensitive to measurement noise (T. Qin, 2018). In the order to address scale ambiguity, the utilization of stereo visual odometry is inherently advantageous. The multi-view nature of stereo setups inherently helps mitigate scale ambiguity by triangulating features from different perspectives, yielding a more precise understanding of the environment's spatial characteristics. The Stereo VO algorithm should effectively combine the input from a stereo camera setup and recently developed feature detection, feature tracking and filtering as well as depth estimation techniques to estimate the robot's motion.

In SVO pipeline as illustrated in Figure 1.1, input images are acquired from 'C1' and 'C2,' representing the centres of a stereo camera system integrated into a graphical render of the UMA robot from SUPSI (D. Gitardi, 2022). Keypoints are detected and descriptors extracted from these images, with feature matching being a critical component of the process. By identifying corresponding features in the images and considering camera intrinsics, the relative pose, represented by a rotation matrix 'R' and a translation vector 't' can be estimated. Notably, the images used are captured outdoor in SUPSI Campus EST, Lugano, offering a practical context for the application of these techniques. Following this estimation, the estimated camera trajectory is plotted, providing a visual representation of the camera movement and positions within the robot's environment. This trajectory visualization is invaluable for mapping, localization, and navigation of robotic systems.

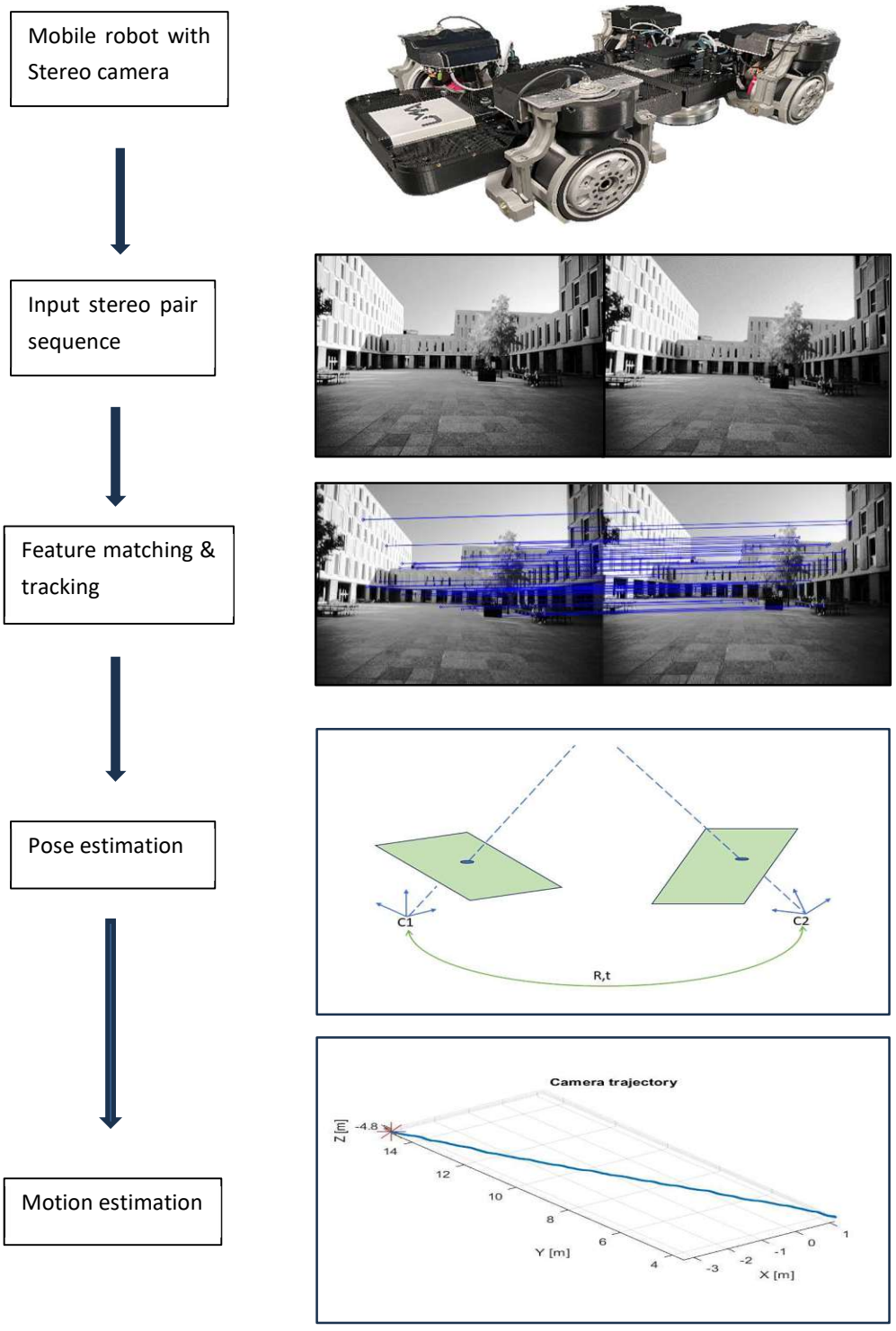


Figure 1.1 SVO Pipeline in Outdoor Environment

1.2 Research Objective and Question

Considering the motivation and the problem definition, we formulate some goals and corresponding research questions to address them.

	Research Objective	Research Question
1	Develop a method to incorporate stereo camera image data for the VO algorithm.	What different image processing and feature based techniques can be utilized to compensate for external sensors?
2	Develop an optimum and feasible method to implement VO.	How can the accuracy, robustness, and computational efficiency of stereo VO be improved using image processing and feature-based techniques?
3	Develop an algorithm which adheres to all the conditions of satisfactory implementation of VO.	How can the parameters of a VO algorithm be tuned to achieve optimal performance in real time, while remaining robust to changes in lighting conditions and occlusions?
4	Conduct and study the performance of the developed algorithm in real-time as well as on publicly available dataset.	What is the accuracy of the estimated motion with respect to the ground truth, and how can the performance of the VO algorithm be further improved?

Table 1.1 Research objectives and questions

1.3 Scope and Limitations

The title of the thesis was carefully refined to highlight particularly ‘Vision-based’ and ‘harsh environment’ part of the mobile robotic motion estimation. These two considerations form the foundation for the main tasks in this thesis research.

The scope includes implementing state-of-the-art feature detection, tracking, and filtering methods, as well as depth estimation techniques, to enhance the robot's environmental perception and navigation efficiency. The algorithm's performance will be extensively evaluated using diverse real-world datasets, capturing various environmental conditions and challenging scenarios.

Despite the comprehensive scope, this thesis project also has certain limitations:

1. Computational Resources: The efficiency of the algorithm may be limited by computational resources, especially when processing large datasets or operating in real-time on resource-constrained apparatus.

2. Real-World Variability: While extensive real-world dataset evaluations will be conducted, the algorithm's performance may still be influenced by unanticipated and highly variable outdoor scenarios.

3. Environmental Constraints: The algorithm's accuracy may be affected by extreme weather conditions or scenarios with limited visual features, leading to challenges in motion estimation.

4. Sensor Noise: The presence of sensor noise, especially in-depth estimation techniques, may introduce errors and affect the overall accuracy of the algorithm.

5. Calibration Requirements: Accurate stereo camera calibration is crucial for successful performance, and any calibration inaccuracies could impact the results.

1.4 Document outline

The Thesis report comprises several key sections illustrated in Figure 1.2. It commences with an "Introduction" that outlines the motivation, research objectives, scope, and document structure. The "Literature Review" delves into fundamental concepts and prior research. The "Methodology" section details system architecture, camera calibration, algorithms, and optimization techniques. "Dataset and Evaluation" covers data sources, evaluation metrics, and experimental setups. "Implementation Details" explains software, hardware, and algorithmic aspects. "Experimental Results" discusses accuracy, comparisons, and computational efficiency. The "Applications and Future Directions" section explores potential uses and future research areas. The "Discussion" addresses the research questions, followed by a "References" section. Additionally, there are "Appendices" with supplementary materials like mathematical formulations, code snippets, and dataset information. This structured outline guides the organization and presentation of the research document.

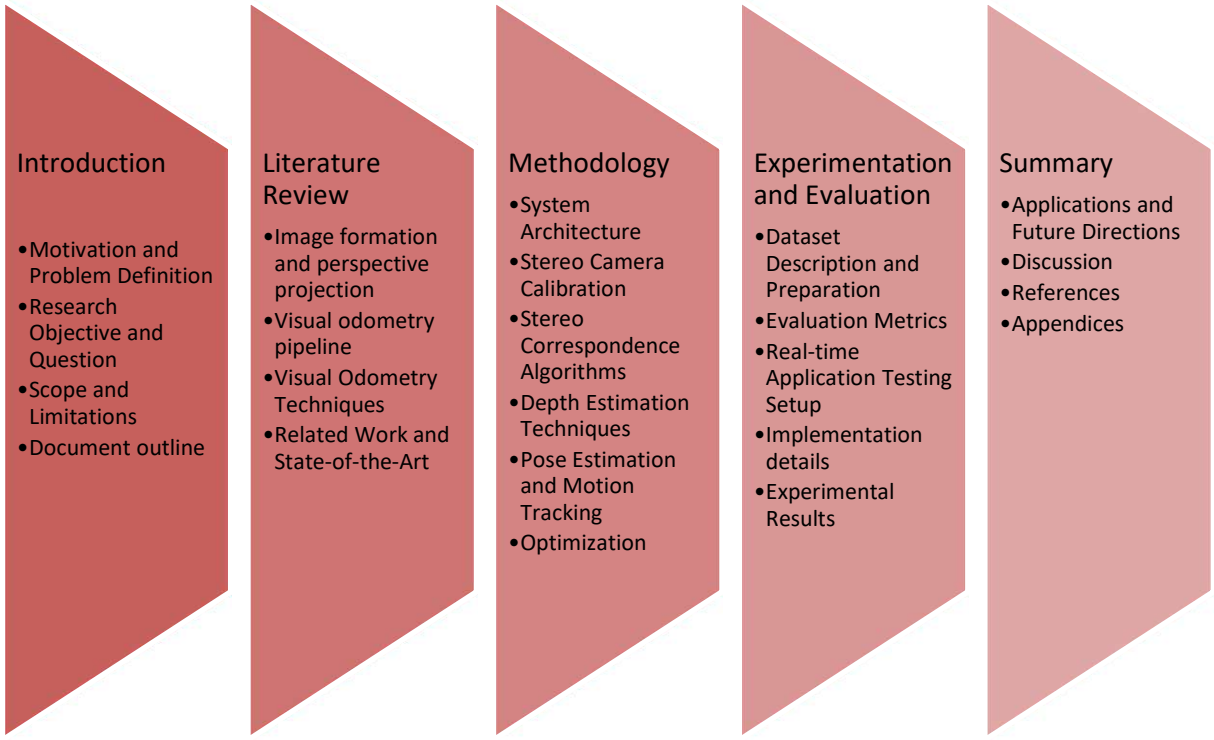


Figure 1.2 Document Outline

2 Literature Review

The literature review in this research work explores the state-of-the-art developments in visual odometry, with a specific emphasis on stereo visual odometry techniques. In Section 2.1, reflects on the basics of visual odometry, a critical pose estimation technique employed by various agents, including vehicles, humans, and robots, relying solely on image streams from attached cameras. Section 2.2 delves into stereo visual odometry techniques, which leverage the information from a stereo camera setup to estimate motion and pose of the agent. Additionally, in Section 2.3, a review of related works, the state-of-the-art algorithms, methodologies in the field are done with a focus on challenges, strengths, and limitations of current approaches. This literature review serves as a foundation for the development of a robust and efficient stereo visual odometry algorithm, addressing challenges in accurate motion estimation in outdoor environments.

2.1 Image formation and perspective projection

For a detailed and elaborate understanding of the Visual Odometry concept, it is essential to put forward the basic of any visual computational technique which is image formation.

The basics of image formation lie at the heart of understanding how images are captured and represented in the digital realm (Bebis, 2017). Image formation is the process by which light rays from a scene interact with a camera's optical system to create a visual representation of the scene on an image sensor or film.

when the light rays pass through a camera's lens, they undergo refraction, causing the rays to converge and form an inverted and reversed image on the camera's image sensor or film plane – thin lens model (Sigmoidal, 2022). The pinhole camera model (Figure 2.1), also known as the geometric camera model, is a simplified theoretical representation (Bebis, 2017). In this model, the camera is represented as a rectangular box with a tiny aperture or "pinhole" on one side.

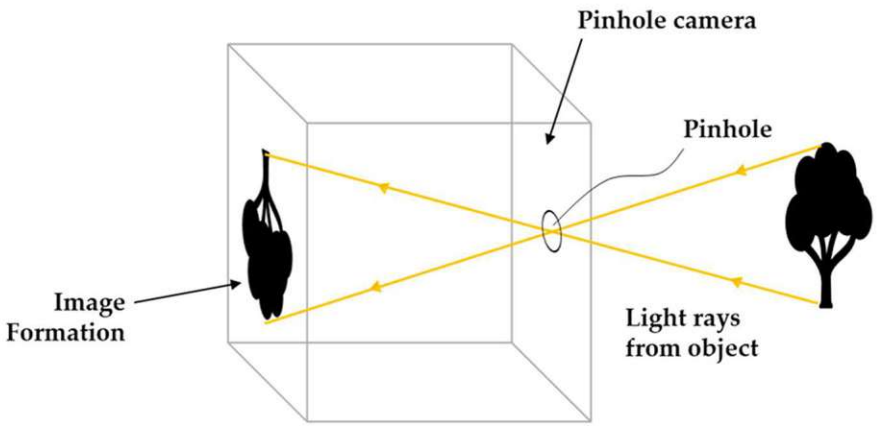


Figure 2.1 Pin-hole camera model (Burger, 2016)

When light rays from a scene pass through this pinhole, they travel in straight lines and strike the opposite side of the camera's interior, forming an inverted and reversed image on a flat image plane located behind the pinhole.

The pinhole camera model comprises several key components that govern image formation. The pinhole is a small, single point on the camera's front side through which light rays from the scene pass. It is the focal point of the camera, where all the incoming rays converge (Pinhole Camera Model , n.d.)The image plane is a flat surface located behind the pinhole and perpendicular to the camera's optical axis. It serves as the recording medium where the image is formed (Forsyth, 2011). The image plane is typically represented by a 2D coordinate system, with each point on the plane corresponding to a specific location in the scene. The optical axis is an imaginary line that passes through the pinhole and extends perpendicularly to the image plane. It represents the principal axis of the camera and plays a significant role in perspective projection (Hartley, 2004). The field of view is the angular extent of the scene that the camera can capture. It is determined by the size of the image plane and the distance between the pinhole and the image plane (Pinhole Camera Model , n.d.). The pinhole camera model exhibits perspective projection, meaning that objects at different distances from the camera are projected onto the image plane with varying sizes. Objects farther away from the camera appear smaller in the image, while closer objects appear larger as seen in the Figure 2.2. This effect creates a sense of depth and three-dimensionality in the 2D image (Sigmoidal, 2022).

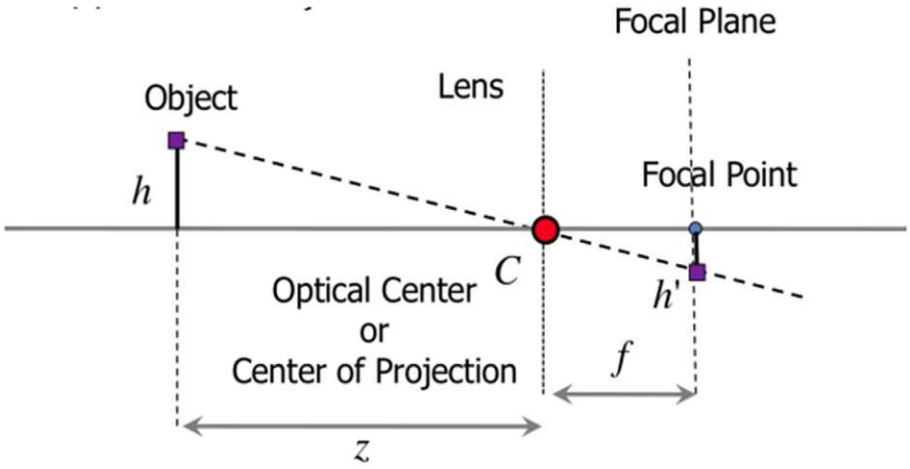


Figure 2.2 Pin-hole Approximation (Scaramuzza D. , 2017)

The camera's image sensor is made up of a grid of photosensitive elements, commonly referred to as pixels. Each pixel records the intensity of light falling on its surface converted into digital data through an analog-to-digital converter (ADC). creating a digital representation of the scene that can be stored and processed by computers and other devices (Sigmoidal, 2022). The resolution of the image is determined by the number of pixels on the sensor.

When a three-dimensional point Pw in the world coordinates is projected onto the camera's image plane using perspective projection, it is transformed into a two-dimensional point (u, v) on the image plane (Sigmoidal, 2022), this flow can be understood in Figure 2.3 .

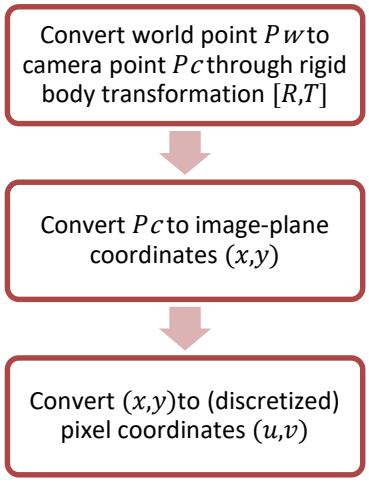


Figure 2.3 Perspective Projection flow

To achieve this goal, the camera's intrinsic parameters, such as focal length and principal point, and the extrinsic parameters, including the camera's position and orientation in the world coordinate system, are considered (Bebis, 2017) as in Figure 2.4.

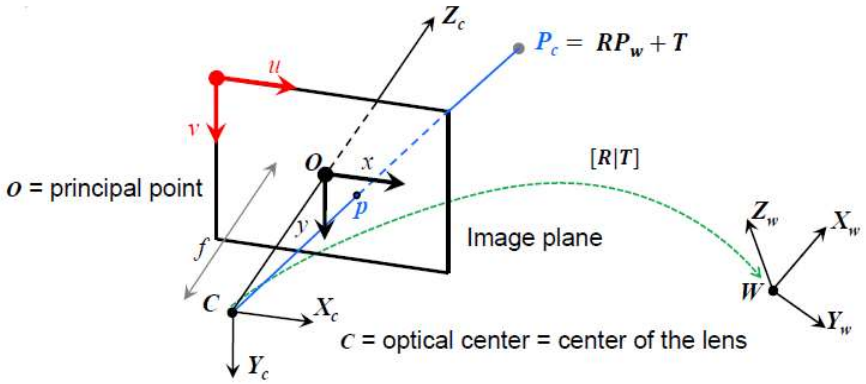
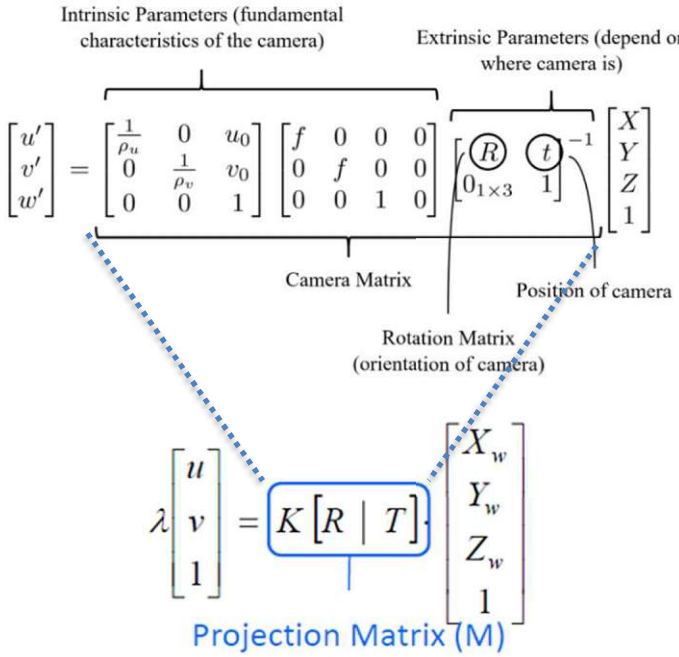


Figure 2.4 Perspective Projection Schematic (Scaramuzza D. , 2017)

By applying the pinhole camera model Equation 2.2 and perspective projection Equation 2.1, the transformation from 3D world coordinates to 2D pixel coordinates is determined The perspective projection process involves computing the projection matrix, which encapsulates the camera's intrinsic and extrinsic matrix [RT].



Equation 2.2 Camera Equation (Corke, 2017)

Equation 2.1 Perspective Projection Equation

2.2 Visual odometry pipeline

By utilizing the principles of perspective projection, VO algorithms can track and match visual features across consecutive frames, enabling incremental estimation of the camera's position and orientation. Following are the steps involved in Visual odometry (Stevenius, 2014) (Geiger A. S., 2012):

1. Feature Detection and Tracking: VO algorithms detect and track distinctive visual features, such as keypoints or corners, in the images captured by the camera. Each feature is represented by its pixel coordinates (u, v) on the image plane. Popular feature detectors, such as FAST, ORB, or SIFT, etc are used to extract keypoints. The Table 2.1 below illustrates a comparison of some major feature detectors.

Feature detector	Description	Pros	Cons
FAST (E. Rosten, 2010)	Computes difference in brightness based off neighbours in Bresenham circle	Efficient	Not robust to significant noise.
Harris (Stephens, 1988)	Computes differential with respect to each direction	Distinguishes corners and edges well	Susceptible to scale variance
ORB (Ethan Rublee, 2011)	Replacement for SIFT that builds off of the FAST detector	Scale and rotation invariant, good for real-time, resilient to noise	Generally, fever features
SIFT (Lowe, 2004)	Computes oriented gradient histograms for patches around a point	Rotation and scale invariant	Computationally expensive. susceptible to blur
Shi-Tomasi (al., 1994)	Very similar to Harris but uses a simpler thresholding method for accepting or rejecting corners	Distinguishes corners and edges well	Susceptible to scale variance
SURF (Herbert Bay, 2006)	A more efficient approximation of SIFT	Faster than SIFT	Susceptible to viewpoint and illumination change

Table 2.1 Feature Detector Comparison

2. Feature Matching: In subsequent frames, the VO algorithm matches the tracked features between the current frame and the previous frame. By associating the pixel coordinates of the same features in different frames, the algorithm establishes correspondences between points in 3D space (the scene) and their 2D projections on the image plane. For stereo camera setup different types of stereo correspondence algorithms are implemented some such popular algorithms are briefly mentioned below,

Block matching algorithms divide the stereo images into small blocks and compare corresponding blocks between the left and right images. Disparity is estimated by finding the block with the most similar intensity or feature matching score. Types of block matching algorithms encompass matching metrics such as Normalized Cross-Correlation (NCC), Sum of Squared Differences (SSD), and Sum of Absolute Differences (SAD).

Semi-Global Matching (SGM) is a global optimization method that takes into account the smoothness constraint of disparities across the entire image. It evaluates the matching cost for each pixel along multiple paths in the image and performs an aggregated cost to find the optimal disparity. SGM is known for its robustness and ability to handle occlusions and textureless regions (Hirschmuller, 2005).

Graph-cut algorithms formulate the stereo correspondence problem as a Markov Random Field (MRF) and apply graph-based optimization techniques, such as graph cuts or belief propagation, to find the best disparity assignment. These algorithms efficiently incorporate global constraints, producing accurate results in textured and occluded regions.

SIFT (Scale-Invariant Feature Transform) is a feature-based approach that identifies distinctive keypoints in both stereo images and matches them based on their feature descriptors. SIFT-based methods are robust to changes in viewpoint, illumination, and occlusions, making them suitable for challenging scenarios.

Recent advancements in deep learning have led to the emergence of stereo correspondence algorithms based on Convolutional Neural Networks (CNNs). These algorithms learn to predict disparities directly from stereo image pairs, leveraging the rich hierarchical features extracted by CNN architectures. Deep learning-based approaches have achieved state-of-the-art performance in stereo matching tasks due to their ability to learn complex feature representations.

3. Perspective Triangulation (Depth estimation): The correspondences obtained through feature matching provide the basis for perspective triangulation. Using the pinhole camera model and perspective projection equations, the 3D positions (X, Y, Z) of the matched features in the camera's coordinate system are estimated. Various depth estimation techniques are used in visual odometry.

   Stereo vision is a popular method for depth estimation in visual odometry. It involves using two cameras (stereo camera setup) to capture the same scene from slightly different viewpoints, mimicking how human eyes perceive depth (Hartley, 2004). By finding point correspondences between the left and right images, the disparity (horizontal shift) can be computed, which is inversely proportional to depth.

In the context of the standard stereo camera setup, where you have a 3D point at coordinates (X, Y, Z), and two cameras with a known focal length "f" and baseline "b," and with their image plane aligned in the XY-plane, the measured disparity can be expressed as,

$$d = \frac{-f * b}{z}$$

Equation 2.3 Stereo Disparity

Stereo vision relies on triangulation to estimate depth and is effective for short to medium-range depth estimation.

Structure from Motion (SfM) is a technique that estimates 3D structure from a sequence of 2D images taken from a moving camera. It involves identifying and matching keypoints between frames, estimating camera motion between frames, and triangulating the keypoints to estimate depth (Szeliski, 2010). SfM can handle single or multiple cameras and is used in both sparse and dense reconstruction approaches.

Visual SLAM (Simultaneous Localization and Mapping) is an extension of visual odometry that not only estimates the camera's motion but also simultaneously builds a map of the environment. It combines depth estimation techniques like stereo vision, SfM, and other sensor data (e.g., IMU, LIDAR) to achieve accurate and robust camera pose estimation and map creation (Scaramuzza D. , 2011).

While stereo vision relies on two cameras, monocular depth estimation uses a single camera to estimate depth. This technique is computationally less expensive but inherently more challenging as it requires leveraging various cues from the scene, such as texture, motion, and perspective. Machine learning approaches, like using deep neural networks, have been employed for monocular depth estimation with promising results (Theobalt, 2016).

Optical flow algorithms can be used for dense depth estimation in visual odometry. Optical flow methods track the motion of every pixel in consecutive frames, which can provide dense depth information. However, optical flow-based methods might be sensitive to illumination changes and occlusions (Cremers D. G., 2011).

In some cases, specialized depth cameras, such as Time-of-Flight cameras or structured light sensors, can be used to directly measure depth information. These cameras provide depth maps with per-pixel depth values, making them valuable for visual odometry tasks (Cremers D. A., 2012).

4. Motion Estimation: With the 3D positions of the matched features in both the current and previous frames, the VO algorithm can calculate the relative motion between the

frames. This motion estimation involves finding the transformation (rotation and translation) between the two sets of 3D feature positions, representing the camera's movement between frames.

5. Incremental Pose Update: By continuously tracking and matching features in consecutive frames, the VO algorithm can incrementally update the camera's pose and position over time. This incremental estimation allows the algorithm to provide real-time feedback on the camera's motion as it moves through the environment.

6. Odometry Integration: The estimated camera motion from visual odometry is integrated over time to obtain a trajectory of the camera's path.

2.3 Visual Odometry Techniques

Visual Odometry techniques can be classified in two main categories: geometry-based and learning-based methods (Duan C, 2023).

2.3.1 Methods Based on Geometry

VO methods based on geometry rely on extracting geometric constraints from images to estimate motion (Duan C, 2023). These methods can be further categorized into: Feature-based method, Direct method, Semi-direct method.

Feature-based methods rely on extracting and tracking distinctive visual features, also known as key points or interest points, in the images captured by the left and right cameras of a stereo setup. These features serve as reference points that are tracked over time to estimate the camera's motion and pose as it moves through the environment. Two important aspects of feature-based methods are feature extraction and feature matching. Feature Extraction is the first step in feature-based stereo VO involving detection and extraction of key point features from both the left and right images. These features are chosen based on their uniqueness, repeatability, and invariance to transformations (e.g., rotation, scale) to ensure reliable tracking. Common feature extraction methods include the Harris corner detector, SIFT (Scale-Invariant Feature Transform), ORB (Oriented FAST and Rotated BRIEF), and FAST (Features from Accelerated Segment Test) (Stewenius, 2011) (Pollefeys, Stereo Visual Odometry, 2008) (Geiger A. L., 2012). Once the features are extracted, the algorithm seeks to find correspondences between matching key points in the left and right images. This process is crucial for establishing the 3D-2D point correspondences required for further triangulation and motion estimation. Various matching techniques are employed, such as brute-force matching, FLANN (Fast Library for Approximate Nearest Neighbours) matching, or more advanced feature matching algorithms based on descriptors and distance metrics (Mikolajczyk, 2005) (Bay, 2008) (Rublee, 2011).

Direct methods in stereo visual odometry (VO) are techniques that estimate the camera motion by directly minimizing the photometric error between corresponding pixels in the left and right stereo images. Unlike feature-based methods that rely on explicit feature extraction and matching, direct methods operate directly on the pixel intensities of the images. These techniques are particularly useful in handling texture-less regions and low-texture environments, where traditional feature-based approaches may struggle (Pollefeys, Stereo Visual Odometry, 2008). The first step in direct stereo VO is image alignment, where the two stereo images are aligned to compare corresponding pixels (Engel, 2012). The core of direct methods lies in the cost function, which represents the sum of squared differences (SSD) or other similarity measures between the pixel intensities in the left and right images. The objective is to find the camera motion that minimizes this cost function (Pollefeys, Stereo Visual Odometry, 2008). Optimization techniques like Gauss-Newton, Levenberg-Marquardt, or gradient descent are used to iteratively update the camera motion parameters and find the minimum of the cost function (Engel, 2012). Direct methods require dense scene representations, such as dense depth maps or semi-dense feature descriptors, to efficiently handle depth variations in the scene (Yang, 2018). To handle outliers and noisy pixel intensities, robust estimation techniques, such as robust cost functions or outlier rejection strategies, are often incorporated to enhance the algorithm's robustness (Glocker, 2014). While direct methods offer advantages like handling texture-less regions and fast motion, they can be computationally intensive, sensitive to photometric changes, and require dense scene representations.

Semi-direct methods in stereo Visual Odometry (VO) combine the advantages of both feature-based and direct methods to estimate the motion and pose of a camera in a stereo camera setup. These methods aim to strike a balance between computational efficiency and robustness to handle texture-less regions, while still leveraging depth information for more accurate motion estimation. Unlike traditional feature-based methods that rely on sparse feature correspondences or direct methods that work directly with pixel intensities, semi-direct methods employ an intermediate scene representation. This representation is typically a dense map of depth or semi-dense features, providing a more structured way of estimating the 3D structure of the scene. Semi-direct methods start with feature detection in both left and right images, such as keypoints, corners, or edges, and then track the detected features across frames to establish correspondences between the stereo image pairs. The detected features are used to create a semi-dense reconstruction of the scene's structure, estimating depths at feature locations using methods like patch-based or probabilistic depth estimation. This semi-dense representation provides additional 3D information for more accurate motion estimation (Newcombe, 2011). Similar to direct methods, semi-direct methods employ photometric error minimization to refine the camera motion estimation. However, instead of using

all pixels in the images, semi-direct methods use the dense or semi-dense representation to calculate photometric errors, measuring the similarity of intensity values between the left and right images. By minimizing the photometric error using the semi-dense representation, semi-direct methods estimate the relative motion between consecutive frames, accounting for camera translation and rotation. These methods are designed to be computationally efficient, leveraging the semi-dense scene representation and feature tracking, while also benefiting from the robustness of feature-based methods to handle texture-less regions and occlusions (Pollefeys, Semi-direct visual odometry, 2011). Semi-direct methods are particularly useful in scenarios where feature-based methods may fail due to a lack of distinct features or in environments with significant texture-less regions.

2.3.2 Methods based on learning

Deep learning-based methods in stereo visual odometry leverage convolutional neural networks (CNNs) to directly estimate camera motion and pose from stereo image pairs. By training on large datasets with ground-truth camera poses, these methods achieve end-to-end learning, bypassing traditional feature extraction and matching steps (Wang W. e., 2021). The CNN takes left and right stereo images as input and outputs the camera's motion or pose. The model is trained with a loss function that measures the discrepancy between the predicted and ground-truth camera motion. Inference on new stereo images allows real-time estimation of camera motion, enabling applications in navigation and localization. While deep learning-based stereo VO offers advantages, obtaining labelled training data can be challenging, and computational resources are required for efficient training (Mur-Artal, 2020).

2.4 Related Work and State-of-the-Art

Stereo visual odometry has witnessed significant advancements to achieve accurate motion estimation and robust navigation in outdoor environments. Various algorithms and methodologies have been proposed, each with its strengths and limitations. This section presents a review of related work and the state-of-the-art in stereo visual odometry, highlighting key approaches and contributions in the field.

Stereo visual odometry techniques leverage the information from a stereo camera setup to estimate the 6-DOF (Degrees of Freedom) pose of an agent, enabling precise localization and navigation. One of the early works in stereo visual odometry introduced a direct approach based on minimizing the photometric error between consecutive stereo images (Scaramuzza D. a., 2009). This direct method demonstrated promising results in terms of accuracy and robustness,

showing its potential for navigation in challenging outdoor environments. Another notable contribution in stereo visual odometry is the semi-direct visual odometry method (J. Engel, 2014). This approach combines the advantages of both direct and feature-based methods, enabling efficient motion estimation while maintaining robustness to varying lighting conditions and dynamic changes in the environment. In recent years, learning-based techniques have also emerged as promising approaches in stereo visual odometry. A learning-based approach that leverages deep neural networks to estimate depth information from stereo image pairs (Schönberger, 2014). This method demonstrated significant improvements in depth estimation accuracy, leading to more accurate motion estimation and navigation. The state-of-the-art in stereo visual odometry has seen the integration of sensor fusion techniques to enhance robustness and accuracy. For instance, proposition for a fusion-based approach that combines stereo visual odometry with IMU data to improve pose estimation in challenging outdoor scenarios (Li, 2019).Despite significant advancements, stereo visual odometry still faces challenges in handling scale ambiguity and dynamic changes in the environment. Researchers continue to explore novel algorithms and improvements to address these limitations and enhance the reliability of motion estimation in dynamic outdoor environments. This review of related work and the state-of-the-art in stereo visual odometry lays the groundwork for the development of a robust and efficient algorithm to address these challenges and achieve accurate pose estimation in diverse outdoor scenarios.

3 Methodology

3.1 System Architecture

a) System Overview

The vision-based motion estimation system is designed to estimate the motion and pose of a mobile robot in harsh environments. The system uses stereo visual odometry techniques to analyse image sequences from a stereo camera setup. The system architecture is modular, with each module responsible for a specific task in the motion estimation process. The stereo camera calibration module ensures precise camera intrinsics and extrinsics, which are crucial for accurate depth estimation and motion tracking. The stereo correspondence algorithms employ feature matching techniques to establish correspondences between the left and right stereo images. Depth estimation techniques are used to determine the 3D structure of the scene and provide depth information, which is vital for accurate pose estimation and environmental perception. The pose estimation and motion tracking module utilize the extracted features and depth information to estimate the robot's movement and position relative to its environment. Optimization techniques are applied to refine the accuracy and stability of the estimated motion. The system is designed with real-time considerations, ensuring that the motion estimation process is efficient and suitable for on-board implementation in mobile robots. Entire sequence of steps implemented to achieve Stereo VO is represented in Figure 3.1. The vision-based approach complements other sensing modalities, such as GPS or IMUs, facilitating sensor fusion for enhanced environmental perception and navigation adaptability. The successful implementation and evaluation of the vision-based motion estimation system will significantly contribute to the field of mobile robotics.

b) Components and Modules

Stereo Camera Calibration

The stereo camera calibration involves determining the intrinsic and extrinsic parameters of the stereo camera setup. The calibration process uses known calibration patterns, such as chessboards or circles, captured from multiple viewpoints. By analysing the correspondences between the calibration pattern points in the images and their corresponding 3D world coordinates, the intrinsic parameters (e.g., focal length, principal point) and extrinsic parameters (e.g., relative pose between the cameras) are computed. These calibrated parameters enable precise mapping of image points to 3D world coordinates, providing the foundation for subsequent motion estimation and environmental perception.

3.2 Data Flow

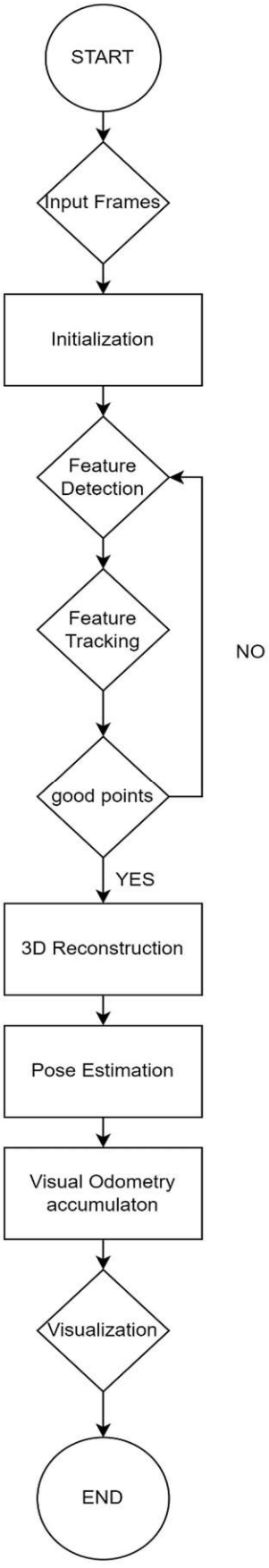


Figure 3.1 Visual Odometry Data flow

Feature Detection

Feature detectors are used for searching specific patterns or specific features which are unique, can be easily tracked and easily compared.

Stereo Correspondence Algorithm

The stereo correspondence algorithms deal with finding corresponding points in the left and right stereo images. Various feature extraction and matching techniques are employed to identify key points in the images and establish correspondences between the stereo image pair. feature detectors are used to extract keypoints, and then matching methods like block matching, semi-global matching, or graph-cut based algorithms are applied to find the correspondences. The matched feature points serve as the basis for triangulating 3D points and estimating the 3D structure of the scene.

Depth Estimation Techniques

The depth estimation techniques focus on estimating the depth information of points in the 3D scene based on the matched feature points from the stereo correspondence module. Triangulation methods, such as triangulating matched feature points using triangulation equations, are commonly employed to determine the depth of scene points. Additionally, the disparity map obtained from the stereo correspondence process can be used to estimate depth, where larger disparities correspond to closer objects and vice versa. Filtering techniques, such as bilateral filtering or depth map fusion, can be applied to improve the accuracy and smoothness of the depth map.

Pose Estimation and Motion Tracking

The pose estimation and motion tracking module utilize the depth information and matched feature points to estimate the robot's motion and pose relative to its environment. This involves tracking the movement of feature points over time and computing the camera's position and orientation changes. Methods like Perspective-n-Point (PnP) algorithms, RANSAC, or iterative algorithms like Bundle Adjustment can be employed for accurate pose estimation. By continuously updating the robot's position and orientation, this module enables real-time motion tracking and navigation.

Optimization

The optimization aims to refine the estimated motion and pose to enhance accuracy and stability. It involves applying optimization techniques like non-linear least squares, Kalman filtering, or bundle adjustment to minimize errors and uncertainties in the motion estimation process. Optimization helps mitigate drift and cumulative errors, resulting in more reliable and precise motion estimates. This step is crucial for long-term navigation tasks and maintaining accuracy during extended robot operations.

3.3 Module Description

3.3.1 Stereo Camera Calibration

Calibration is the process to determine the intrinsic parameters (K, lens distortion) and extrinsic parameters (R, T) of a camera. K, R, T can be determined by applying the perspective projection Equation 3.1 to known 3D-2D point correspondences (Hartley, 2004).

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R \mid T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Equation 3.1 Stereo Camera Calibration

The intrinsic parameters represent the internal characteristics of each camera, such as the focal length, principal point, and lens distortion coefficients. The focal length determines the camera's field of view and influences the scale of the reconstructed 3D scene. The principal point represents the optical centre of the camera's image sensor (Szeliski, 2010). Lens distortion coefficients account for the imperfections introduced by the camera's lens and are used to correct image distortion (Zhang, 2000).

The extrinsic parameters define the relative pose between the two cameras in the stereo setup. They describe the rotation and translation between the coordinate systems of the two cameras. The extrinsic parameters are essential for matching corresponding points between the left and right images (Hartley, 2004).

The calibration process involves capturing images of known calibration patterns from multiple viewpoints and then using these images to compute the camera calibration matrices (Szeliski, 2010). The stereo camera calibration process used involves steps as in Figure 3.2.

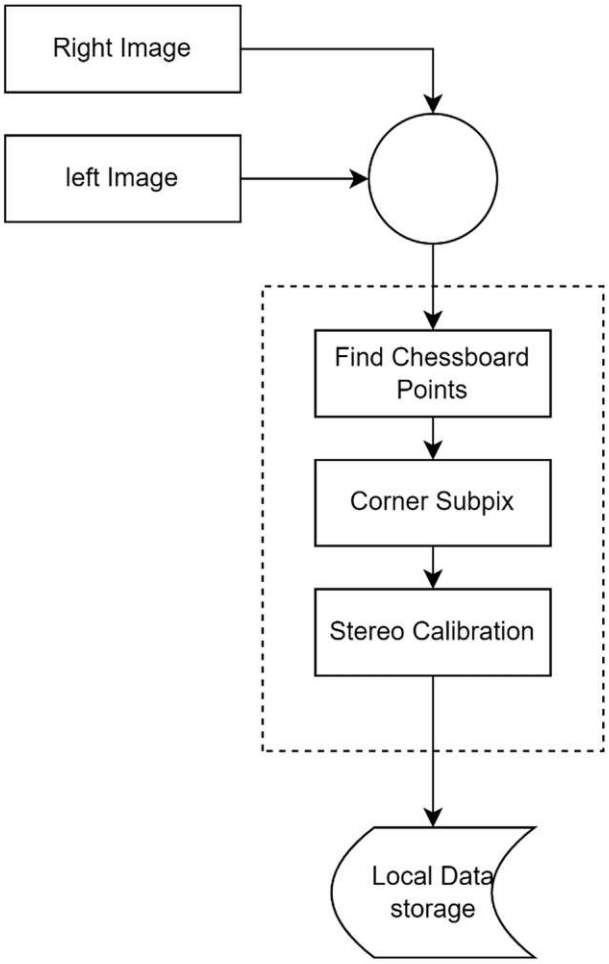


Figure 3.2 Stereo calibration data flow

1. Capturing Calibration Images

   Known calibration patterns, such as chessboards or circles with precisely known dimensions, are placed in the field of view of both cameras. Multiple images of these calibration patterns are captured from various viewpoints and orientations (Hartley, 2004). It is essential to capture a diverse set of calibration images with different patterns covering the entire field of view.

   For this particular project a checkerboard is used. A separate python script 'Acquiring image' is used to acquire infrared images from a stereo camera setup for calibration. It utilizes the Intel RealSense SDK (pyrealsense2) to interact with the RealSense camera and OpenCV (cv2) for image handling and display. The code captures and saves infrared images from both the left and right cameras, as displayed in Figure 3.3.

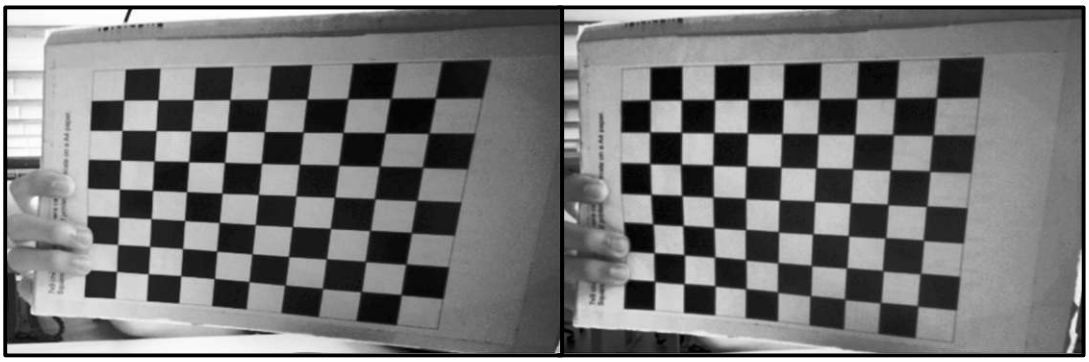


Figure 3.3 Checkerboard stereo pair

Detailed explanation of the code:

i. Importing Libraries:

The necessary libraries are imported, including os, pyrealsense2, numpy (np), and cv2 (OpenCV).

ii. Creating Directories:

The code creates directories named "**left_images**" and "**right_images**" to store the acquired calibration images. These directories are used to organize the saved images.

iii. Initializing the RealSense Pipeline:

The RealSense pipeline and configuration are initialized to enable infrared streams from both cameras (left and right).

iv. Starting the Pipeline:

The pipeline is started with the specified configuration.

v. Accessing Device Information:

Device information is accessed to configure the emitter. The code sets the emitter to be disabled (set_emitter = 0) to use the infrared stream instead of the emitter. This is useful for infrared-based calibration.

vi. Main Loop:

The main loop continuously captures and displays the infrared frames from both the left and right cameras.

vii. Capturing Frames:

The code waits for frames to be available from the pipeline. It acquires infrared frames for both the left and right cameras.

viii. Converting Frames to Numpy Arrays:

The acquired frames are converted to numpy arrays (left_image and right_image) for processing and display.

ix. Displaying Frames:

The infrared frames from both cameras are displayed in separate windows using OpenCV's cv2.imshow() function.

x. Triggering Saving:

When the user presses the '**s**' key, the saving of the current images is triggered.

xi. Saving Images:

If saving is triggered, the left and right infrared images are saved in the respective "**left_images**" and "**right_images**" directories with filenames in the format "**stereoLX.png**" and "**stereoRX.png**," where X is the image counter.

xii. Incrementing Image Counter:

The image counter is incremented to ensure unique filenames for each saved image.

xiii. Exiting the Loop:

The loop can be exited by pressing the 'q' key.

xiv. Stopping the Pipeline and Closing Windows:

When the loop is exited, the RealSense pipeline is stopped, and all OpenCV windows are closed.

2. Detecting Calibration Points

In the calibration images, the corners of the calibration pattern are automatically detected using corner detection algorithms (e.g., Harris corner detector). The detected corners are then matched between the left and right images to form corresponding point pairs as displayed in Figure 3.4.

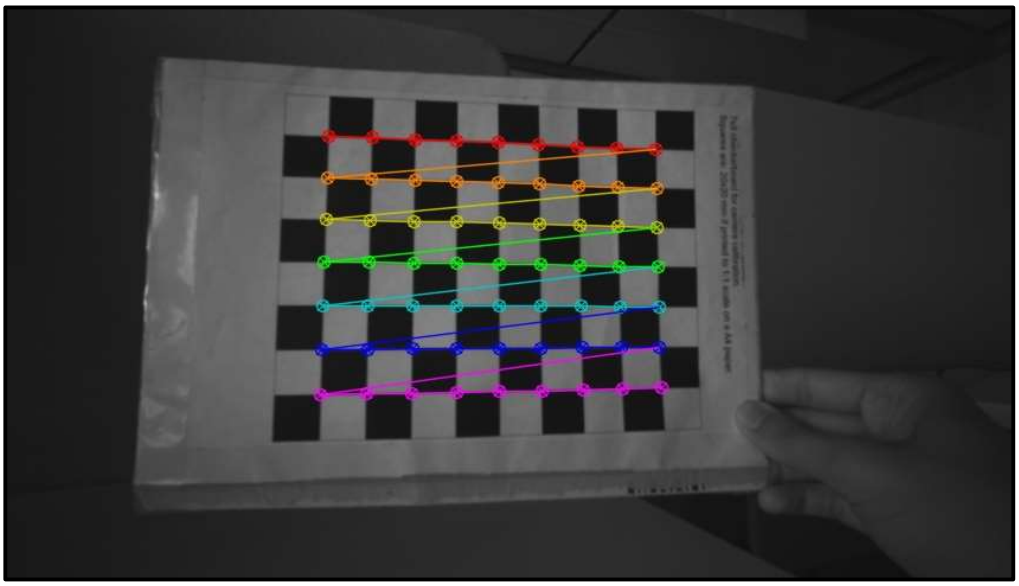


Figure 3.4 Calibration corner detection and correspondence

The following section titled 'FIND CHESSBOARD CORNERS – OBJECT POINTS AND IMAGE POINTS' of the Stereovision_calibartion script detects the chessboard corners in the left and right stereo images and finds the corresponding 2D image points. These image points, along with known 3D object points (chessboard corners in the real world), are used to calibrate the stereo camera.

i. chessboardSize and frameSize:

These variables specify the size of the chessboard pattern (number of internal corners) and the size of the frames acquired from the stereo camera.

ii. criteria:

The criteria variable defines the termination criteria for the corner refinement process. It is used in the cv.cornerSubPix() function to determine when to stop refining the corner positions

iii. objp (Object Points):

This variable is a NumPy array that stores the 3D coordinates of the chessboard corners in the real world. It is initialized with zeros and then the x and y coordinates are assigned based on the chessboard pattern's size. The z-coordinate is set to zero since the chessboard lies in a plane.

iv. size_of_chessboard_squares_mm:

This variable specifies the size of the chessboard squares in millimetres. It is used to scale the object points to the real-world coordinates.

v. Image Point and Object Point Arrays:

Three arrays are defined to store the object points, image points from the left camera (imgpointsL), and image points from the right camera (imgpointsR). These arrays will be used to store the detected corner positions.

vi. Image Acquisition and Loop:

The script reads pairs of left and right stereo images from the specified directories using glob.glob(). It iterates over the pairs of images and performs calibration for each image pair.

vii. Image Preprocessing:

For each left and right image pair, the script reads and converts the images to grayscale using cv.cvtColor().

viii. Detecting Chessboard Corners:

The chessboard corners are detected using the cv.findChessboardCorners() function. This function uses the Harris corner detector to find the corners of the chessboard pattern in the grayscale images. The function takes the grayscale image, chessboardSize, and a None argument for the optional output parameter corners.

ix. Corner Refinement:

If chessboard corners are found in both the left and right images (**'retL'** and **'retR'** are True), the script refines the corner positions using cv.cornerSubPix(). This function performs subpixel corner refinement based on the criteria termination criteria. The refined corner positions are then added to the respective image points arrays (**'imgpointsL'** and '**imgpointsR'**).

x. Visualization:

The script draws the detected corners on the left and right images using cv.drawChessboardCorners() and displays the images using cv.imshow(). This visualization allows you to verify the correctness of the detected corners.

xi. Loop Continuation and User Interaction:

After displaying the images with detected corners, the script waits for a key press. Pressing any key other than '**s**' will proceed to the next image pair. However, if '**s**' is pressed, the detected corners are saved, and the calibration images will be used for stereo camera calibration.

xii. Loop Termination:

The script continues this loop until all image pairs have been processed. To stop the loop, press the 'q' key.

xiii. Cleanup:

After processing all the image pairs, the script stops displaying the images and closes all OpenCV windows using cv.destroyAllWindows().

3. Computing Intrinsic Parameters

The corresponding point pairs from the calibration images are used to compute the intrinsic parameters of each camera. This is typically achieved using camera calibration algorithms, such as Zhang's method.

The sections titled 'CALIBRATION' and 'Stereo Vision Calibration' of the Stereovision_calibartion script performs stereo camera calibration using a set of detected chessboard corners from the left and right stereo images. Stereo calibration involves finding the intrinsic and extrinsic parameters of both cameras and calculating the Essential and Fundamental matrices, which are essential for accurate stereo vision reconstruction.

Here's a detailed explanation of the code:

i. Camera Calibration for Left Camera (imgL):

cv.calibrateCamera: This function calibrates the left camera using the object points (**'objpoints'**) and the image points (**''imgpointsL''**) obtained from the left stereo images. It returns the calibration results, including the camera matrix (**'cameraMatrixL'**), distortion coefficients (**''distL''**), rotation vectors (**'rvecsL'**), and translation vectors (**'tvecsL'**).

cv.getOptimalNewCameraMatrix: This function calculates the optimal camera matrix (**''newCameraMatrixL''**) and the region of interest (**'roi_L'**) for the left camera. The optimal camera matrix is used to correct the distortion and improve the accuracy of the calibration.

ii. Camera Calibration for Right Camera (imgR):

Similarly, the right camera is calibrated using cv.calibrateCamera, and the calibration results are obtained for the right camera, including the camera matrix (**'cameraMatrixR'**), distortion coefficients (**''distR''**), rotation vectors (**'rvecsR'**), and translation vectors (**'tvecsR'**).
cv.getOptimalNewCameraMatrix: The optimal camera matrix (**''newCameraMatrixR''**) and the region of interest (**'roi_R'**) for the right camera are calculated.

4. Stereo Vision Calibration:

flags:
The variable flags are used to set the calibration options. In this code, CALIB_FIX_INTRINSIC flag is set, which means the intrinsic camera matrices (**'newCameraMatrixL'** and **'newCameraMatrixR'**) are fixed during stereo calibration. This is done because the left and right cameras have already been individually calibrated, and only the relative transformation between them needs to be estimated.

cv.stereoCalibrate:
This function performs stereo calibration to calculate the relative transformation between the left and right cameras and estimates the Essential and Fundamental matrices. It takes the object points (objpoints), the image points from the left (**'imgpointsL'**) and right (**'imgpointsR'**) cameras, as well as their respective camera matrices (**'newCameraMatrixL'** and **'newCameraMatrixR'**), distortion coefficients (**'distL'** and **'distR'**), the size of the grayscale images (grayL.shape[::-1]), the termination criteria (criteria_stereo), and the calibration options (flags). The function returns the calibration results, including the rotation matrix (rot), translation vector (trans), Essential matrix (essentialMatrix), and Fundamental matrix (fundamentalMatrix).

5. Stereo Rectification

Stereo rectification transforms the images from both cameras such that their epipolar lines become horizontal and have a constant disparity.

The provided part of the calibration script includes this stereo rectification step.

i. Stereo Rectification Parameters:

The rectifyScale parameter is a scaling factor used in stereo rectification. It determines the scaling of the rectified images. A value of 1 indicates that no scaling is applied.

ii. Stereo Rectification:

cv.stereoRectify:
This function performs stereo rectification using the calibration results from the stereo calibration step. It takes the camera matrices (**'newCameraMatrixL'** and **'newCameraMatrixR'**), distortion coefficients (**'distL'** and **'distR'**), rotation matrix (rot), translation vector (trans), and the scaling factor rectifyScale.The function returns the rectification transformation matrices for both left (**'rectL'**) and right (**'rectR'**) cameras. These matrices can be used to rectify the distorted images, making their epipolar lines horizontal and aligned.

'projMatrixL' and **'projMatrixR'**:
These variables store the projection matrices after rectification for the left and right cameras. They are used to transform the image points into 3D world coordinates (disparity to depth mapping) during depth estimation.

Q:
The disparity-to-depth mapping matrix (**Q**) is used to convert disparities into real-world depths. It allows converting the disparity value at a pixel to the corresponding depth value in the 3D world.

iii. Undistortion and Rectification Mapping:

- cv.initUndistortRectifyMap:
  This function initializes the undistort and rectify maps for each camera using the rectification results. It takes the camera matrices (**'newCameraMatrixL'** and **'newCameraMatrixR'**), distortion coefficients (**'distL'** and **'distR'**), rectification matrices (**'rectL'** and **'rectR'**), projection matrices (**'projMatrixL'** and **'projMatrixR'**), and the size of the images. The function returns the rectification maps (**'stereoMapL'** and **'stereoMapR'**) for both left and right cameras. These maps are used to remap the distorted and unrectified images to their rectified versions efficiently.

The calibration process provides crucial intrinsic and extrinsic parameters for both individual cameras and their relative transformation. These calibration parameters and matrices can be saved and utilized in subsequent visual odometry algorithms.

3.3.2 Feature Detection

In this SVO algorithm we use FAST (Features from Accelerated Segment Test) detector developed by Edward Rosten and Tom Drummond (E. Rosten, 2010). It is a corner detector.

The general procedure for working of the FAST detector is mentioned hereby, choose a pixel (referred to as p) within the image to determine whether it's an interest point or not. This pixel has a certain intensity denoted as I_p. Next, pick a suitable threshold value, t. Now, envision a circular region consisting of 16 pixels centred around the pixel currently being examined. The pixel p is considered a corner when there is a group of n adjacent pixels within the 16-pixel circle that are either all brighter than Ip+t or all darker than Ip−t, n is set to 12. To quickly eliminate many non-corner cases, a high-speed test is proposed. This test focuses on just four pixels at positions 1, 9, 5, and 13. Initially, pixels 1 and 9 are checked to determine if they are significantly brighter or darker. If they pass this test, then pixels 5 and 13 are examined. If p is indeed a corner, then at least three of these four pixels must meet the criteria of being brighter than Ip+t or darker than Ip-t. If none of these conditions are met, p cannot be a corner. The complete segment test criteria are then applied to the remaining potential corner candidates by analysing all the pixels within the circle.

3.3.3 Stereo Correspondence Algorithms

Stereo correspondence algorithms aim to find corresponding points or pixel disparities between the left and right stereo images, which are essential for estimating depth information and reconstructing the 3D scene.

The stereo correspondence algorithm used here is Stereo Semi-Global Block Matching (SGBM) based on H. Hirschmuller algorithm (Hirschmuller, 2005). The goal of the algorithm is to find matching keypoints or feature points in the left and right stereo images, which are essential for estimating the 3D position of those points. the algorithm is single-pass, considering 5 directions (cv::StereoSGBM Class Reference, n.d.). The SGBM architecture is illustrated in Figure 3.5.

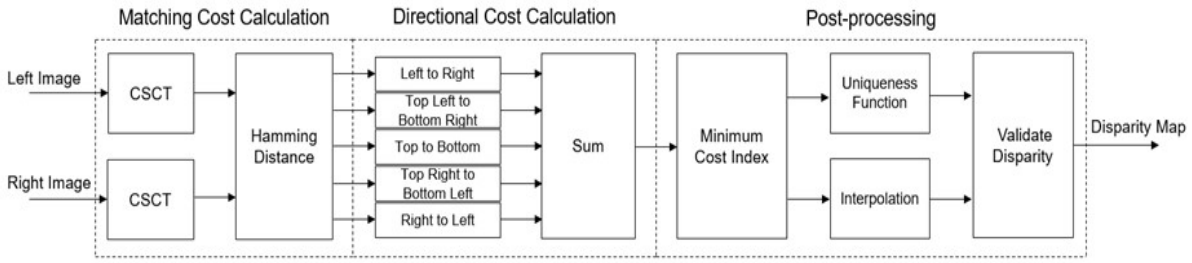


Figure 3.5 SGBM Block diagram (MathWorks Switzerland, n.d.)

The key aspects of the Stereo Semi-Global Block Matching (SGBM) algorithm are:

1. Stereo Correspondence:

Stereo correspondence refers to the process of finding the same point or feature in both the left and right stereo images. It is a crucial step in stereo vision and visual odometry, as it allows the estimation of the 3D position of points in the environment.

2. Block Matching:

   The basic idea behind the Stereo SGBM algorithm is block matching, where a small window (block) in the left image is searched for its corresponding window in the right image. The matching is based on finding similar patterns or intensity similarities between the two windows.

3. Semi-Global Approach:

   The term "Semi-Global" in SGBM means that the algorithm considers the global information along multiple directions (disparities) while finding correspondences. It takes into account not only the local matching cost but also considers consistency along different scanlines and paths in the image.

4. Disparity Map:

   The output of the SGBM algorithm is a disparity map, where each pixel in the left image is associated with a disparity value. The disparity value represents the horizontal shift needed to match the corresponding point in the right image with the point in the left image.

5. Parameters:

   The SGBM algorithm has several parameters that can be tuned to control the matching process, including the block size, minimum and maximum disparity range, uniqueness ratio, and speckle range, among others.

   Tuning these parameters is essential to achieve accurate and reliable stereo correspondences.

6. Efficiency:

   While SGBM provides good results, it can be computationally expensive, especially for high-resolution images or large search ranges.To address this, the algorithm may utilize techniques like multi-resolution or image pyramids to speed up the matching process.

7. OpenCV Implementation:

   In the provided code, the OpenCV library's **cv2.StereoSGBM_create** function is used to implement the Stereo SGBM algorithm.The function takes the input

left and right images and computes the disparity map using the specified parameters.

3.3.4 Depth Estimation Techniques

Depth estimation is a critical aspect of visual odometry as it helps determine the 3D structure of the environment

In this thesis project the depth estimation technique used is Stereo Vision which can be implemented as following:

1. Image Rectification:

   Before depth estimation, the code starts by rectifying the stereo images. Image rectification is a crucial preprocessing step that ensures corresponding epipolar lines in both images are parallel. This simplifies the disparity calculation process. It involves applying a geometric transformation to the images based on the camera calibration parameters.

2. Disparity Calculation:

   Once the images are rectified, the disparity between corresponding points in the left and right images is computed. The disparity represents the horizontal shift of a point in the left image to its corresponding point in the right image. Disparity is inversely proportional to the depth of the scene points. Points that are closer to the camera will have higher disparity values than points that are farther away.

3. Semi-Global Block Matching (SGBM):

   The code utilizes the Semi-Global Block Matching algorithm to compute disparities. SGBM is a popular algorithm for stereo matching that works by comparing small blocks of pixels between the left and right images to find matching blocks with similar patterns. It uses a cost function to find the best disparity value that minimizes the differences between the blocks. SGBM takes advantage of both local and global information to improve the accuracy of disparity calculations.

4. Keypoint Detection and Tracking:

   To efficiently estimate depth for specific points, the code employs the FAST (Features from Accelerated Segment Test) algorithm to detect keypoints in the left image. Keypoints are points of interest in the image that are likely to have distinctive features. The code then tracks these keypoints using optical flow, which estimates the motion of

keypoints between consecutive frames. By tracking keypoints, the code ensures that the same points are used for depth estimation in both images.

5. Right Keypoint Calculation:

   With the disparity values obtained from SGBM, the code calculates the coordinates of corresponding keypoints in the right image. The disparity provides the horizontal shift, and by knowing the position of a keypoint in the left image, the position of its corresponding point in the right image can be determined.

3.3.5 Pose Estimation and Motion Tracking

Pose Estimation is the process of determining the relative position and orientation of a moving camera between consecutive frames. It involves detecting key features in the frames, matching them, and triangulating their 3D positions. By computing the essential matrix, the camera's movement can be estimated.

In the Visual Odometry class, the `**estimate_pose**` method is responsible for estimating the transformation matrix between two frames. This method uses a RANSAC-based approach to robustly estimate the transformation.

Random sample consensus (RANSAC) is an iterative approach for deducing the parameters of a mathematical model based on a collection of observed data points, even in situations where outliers are present. Outliers are disregarded in such a way that they do not exert any influence on the computed parameter values. Consequently, RANSAC can also be viewed as a technique for identifying and isolating outliers within the data (Strutz, 2016).

RANSAC accomplishes its objective through a series of iterative steps:

a) Randomly choose a subset from the original data, labelling it as the hypothetical inliers.
b) Fit a model to this selected set of hypothetical inliers.
c) Evaluate all data points against the model. Any data points in the original dataset that closely conform to the estimated model, as determined by a model-specific loss function, are categorized as the consensus set, i.e., the inliers for the model.
d) The estimated model is considered satisfactory if a sufficient number of data points are classified as part of the consensus set.
e) To enhance the model, it can be refined by re-estimating it using all the members of the consensus set. The quality of the fit, as measured by how well the model aligns with the consensus set as represented in Figure 3.6, is employed to refine

the model fitting as iterations progress. This measure often serves as the fitting quality criterion for the subsequent iteration.

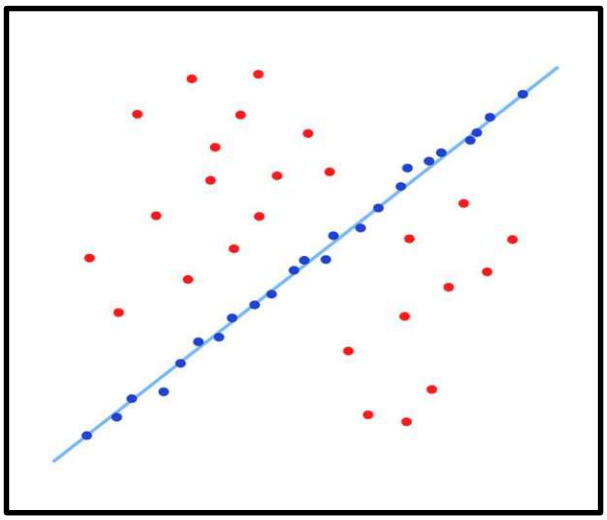


Figure 3.6 fitted line with RANSAC (wikipedia.org, n.d.)

- The RANSAC algorithm randomly selects 6 feature points (keypoints) from the two frames and estimates the transformation matrix using these points.
- The transformation matrix is estimated using a nonlinear least-squares optimization (Levenberg-Marquardt algorithm) that minimizes the reprojection error.
- The reprojection_residuals method calculates the residuals, which are the differences between the observed 2D feature points and the 2D projections of the 3D points using the estimated transformation matrix. The goal is to minimize these residuals to find the optimal transformation.
- The RANSAC process is repeated for a specified number of iterations (`**max_iter**`). The best transformation matrix with the minimum reprojection error is selected as the final estimation.
- The `**estimate_pose**` method returns the estimated transformation matrix.

Motion Tracking is the process of continuously estimating the camera's movement over time by analyzing a sequence of frames. It involves tracking keypoint correspondences between frames using techniques like optical flow. The camera's motion (rotation and translation) is then estimated based on the tracked keypoints and geometric constraints.

- The `**get_pose**` method is responsible for tracking the motion between consecutive frames.

- The process begins by capturing the left and right images of the previous frame (`**old_frame_left**` and `**old_frame_right**`) and the current frame (`**new_frame_left**` and `**new_frame_right**`).
- The algorithm uses FAST feature detection to find keypoints in the left image.
- These keypoints are then tracked in the current frame using optical flow (Lucas-Kanade method). This provides the corresponding keypoints between the previous and current frames.
- Disparity maps are calculated for both the previous and current frames. Disparity represents the difference in the horizontal position of corresponding pixels between the left and right images and is used to estimate depth information.
- Right keypoints are calculated from the left keypoints using the disparity information. This step helps to triangulate the 3D points seen from both images.
- Using the keypoints and 3D points from both frames, the `**estimate_pose**` method is called to estimate the transformation matrix between the previous and current frames.
- The `**get_pose**` method returns the transformation matrix, which represents the motion between the two frames.

3.3.6 Optimization

Optimization in Visual Odometry refers to the process of refining camera poses and 3D point positions to minimize the reprojection error.

The major section of visual odometry where optimization is implemented are:

Reprojection Error:

Once the motion (camera pose) is estimated and 3D points are triangulated using feature matching, each 3D point is projected back into each image frame using the estimated camera poses and intrinsic camera parameters. The difference between the projected 2D point and the observed 2D keypoint is the reprojection error.

Objective Function:

The objective function is formulated to represent the sum of reprojection errors for all matched keypoints across all frames. It quantifies the discrepancy between the observed keypoints in the images and the keypoints projected from the estimated 3D structure and camera poses.

Optimization Algorithm:

The most common optimization algorithm used in Visual Odometry is Bundle Adjustment (BA). BA optimizes the camera poses and 3D point positions

simultaneously to minimize the objective function (i.e., the total reprojection error). It takes into account all the camera poses, 3D points, and their respective uncertainties to refine the motion estimation and 3D reconstruction.

Iterations:

The optimization process is iterative. It starts with an initial estimate of camera poses and 3D points. Then, in each iteration, the algorithm updates the poses and 3D points to minimize the reprojection error further. The optimization continues for several iterations to improve the accuracy of the estimated camera motion and 3D structure.

Convergence:

The optimization process iteratively refines the camera poses and 3D points until a convergence criterion is met. The criterion could be a maximum number of iterations, or when the change in the objective function falls below a certain threshold.

The optimization technique used for estimating the transformation matrix is the Levenberg-Marquardt (LM) algorithm. The optimization is performed to minimize the reprojection error between the 3D points and their corresponding 2D feature points in two consecutive images. The goal is to find the best transformation (rotation and translation) that aligns the 3D points from the previous image to their corresponding 2D points in the current image.

The optimization process is implemented using the scipy.optimize.least_squares function, which provides an interface for least-squares optimization. The LM algorithm is a popular optimization technique for nonlinear least squares problems. It combines the advantages of the Gauss-Newton method and the steepest descent method to find the local minimum of the objective function efficiently.

The core of the optimization process is the reprojection_residuals method, which calculates the residuals between the observed 2D points (keypoints) in the images and the 2D points projected from the 3D points using the transformation matrix. The residuals represent the difference between the observed 2D points and the 2D points obtained by projecting the 3D points using the estimated transformation matrix. The goal of the optimization is to minimize these residuals.

A breakdown of the optimization process is given below:

1. The '**estimate_pose**' method implements the LM optimization loop. It iteratively refines the transformation matrix by randomly selecting 6 feature points from the matched keypoints and corresponding 3D points. It then applies the '**least_squares**' function to find the transformation matrix that minimizes the reprojection residuals.

2. The '**reprojection_residuals'** method calculates the reprojection error. It takes the transformation matrix (composed of rotation and translation), 2D keypoints in the previous and current images, and 3D points corresponding to these keypoints. It uses the projection matrices of the left and right cameras to project 3D points onto the image plane in both images.

3. The LM algorithm iteratively updates the transformation matrix to minimize the residuals. It continues until either the maximum number of iterations ('**max_iter'**) is reached or until no significant improvement is observed for a certain number of iterations (early termination).

4. The '**estimate_pose'** method returns the final transformation matrix estimated by the LM optimization.

4 Dataset and Evaluation

The Master thesis research utilizes a suitable dataset for evaluating the proposed stereo visual odometry algorithm. The dataset includes synchronized stereo image pairs captured from cameras with known calibration parameters. The dataset covers various real-world scenarios, capturing different environments, camera motions, and lighting conditions.

4.1 Dataset Description and Preparation

The dataset's diverse nature and careful preparation allow for a comprehensive evaluation of the stereo visual odometry algorithm's performance, including its accuracy, robustness, and efficiency. This evaluation is essential to validate the algorithm's effectiveness in estimating camera motion and 3D scene structure in real-time applications and various challenging environments.

4.1.1 Offline Dataset

The KITTI Visual Odometry dataset is a well-established benchmark designed to assess the accuracy and robustness of visual odometry algorithms under various real-world driving scenarios. The stereo data collection in the KITTI Visual Odometry dataset is achieved using a sophisticated sensor setup that incorporates two grayscale cameras and a Velodyne LiDAR scanner. The setup is mounted on a Volkswagen Passat B6 (Figure 4.1), which has been equipped with actuators for the pedals and steering wheel, enabling precise control of the vehicle during data collection.

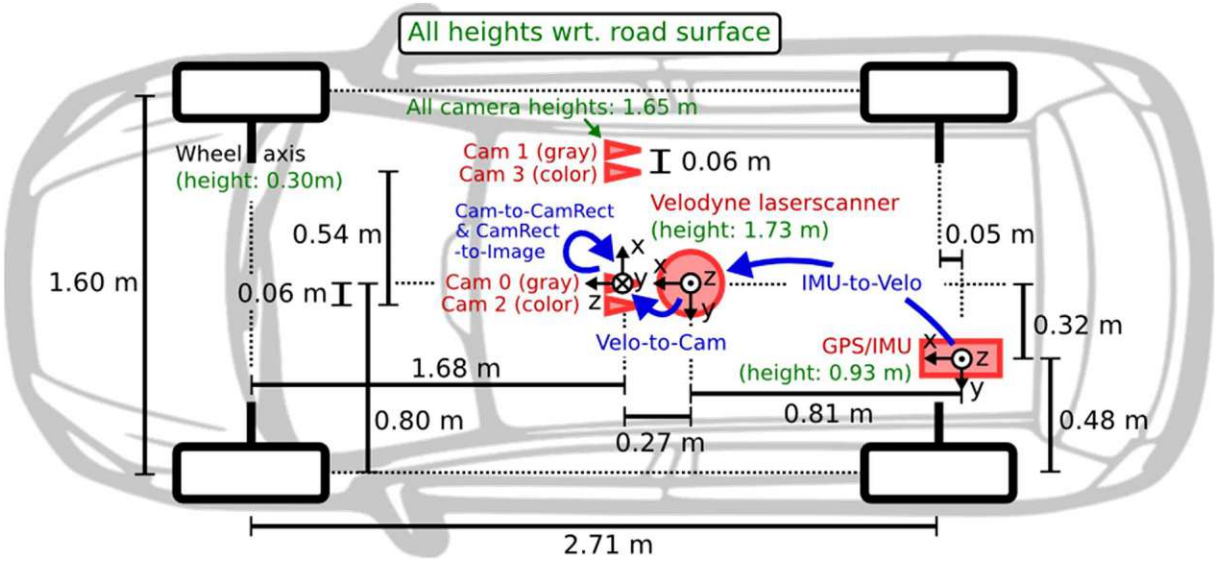


Figure 4.1 Fully equipped sensor setup vehicle (Andreas Geiger, 2012)

The specific details of the stereo data collection are as follows:

1. Grayscale Cameras: Point Grey Flea 2 (FL2-14S3M-C)

   The sensor setup includes two grayscale cameras with a resolution of 1.4 Megapixels each.The cameras are triggered at a rate of 10 frames per second by the Velodyne LiDAR scanner, ensuring temporal synchronization between the camera frames and the 3D LiDAR data.

2. Velodyne LiDAR Scanner: Velodyne HDL-64E

   The Velodyne HDL-64E is a high-quality LiDAR scanner capable of generating a 3D point cloud of the surrounding environment.It spins at 10 frames per second and captures approximately 100,000 points per revolution with the vertical resolution of 64.The LiDAR scanner plays a vital role in providing accurate depth information for the stereo visual odometry pipeline.

3. Data Synchronization

   The cameras are triggered by the Velodyne LiDAR scanner, ensuring that the captured images correspond to the same timestamp as the LiDAR data.This temporal alignment is essential for accurate stereo depth estimation and motion tracking.

4. Image Rectification

   The captured images undergo rectification to transform them into a common reference frame.This process aligns the epipolar lines, simplifying the stereo matching process.

Specifically, we have utilized the grayscale sequence 01 (Figure 4.2) from the KITTI Visual Odometry dataset, encompassing the first 101 frames.

Figure 4.2 One of the stereo pair from KITTI odometry grayscale dataset sequence '01'

4.1.2 Data Collection for Real-time Application

Synchronized infrared image frames from a RealSense camera in real-time are utilized for live stereo visual odometry. A python script is used to perform this function.

a brief explanation of the key components and functionalities of the script:

- Importing Libraries:

  The script begins by importing the necessary libraries, including **os**, **pyrealsense2** (RealSense SDK), **numpy**, and **cv2** (OpenCV).

- Initializing the Camera:

  The **RealsenseCamera** class constructor initializes the RealSense camera's pipeline and configures it to enable infrared streams from both left and right cameras (stream index 1 and 2, respectively). It also sets up alignment to align the infrared frames.

- Frame Capturing:

  The **get_frame_stream** method is the main function for capturing frames from the camera. Inside the method, it waits for new frames (up to a timeout of 30,000 milliseconds) and retrieves synchronized infrared frames from both the left and right cameras.

- Enabling Autoexposure:

The **enable_auto_exposure** function is used from the realsense library to dynamically adjust exposure settings based on the ambient lighting conditions. Whereas the exposure time is set for 100 microseconds.

- Frame Preprocessing:

  Before returning the frames, the script applies histogram equalization using OpenCV (**cv2.equalizeHist**) to enhance the visibility of objects in the infrared images.

- Release Function:

  The **release** method stops the camera pipeline and releases any associated resources.

- Real-Time Streaming:

  The script sets up an infinite loop inside the **get_frame_stream** method to continuously capture and return new frames as they become available. This setup allows real-time streaming of synchronized infrared image frames from the RealSense camera.

Obtaining frames with a resolution of 848 x 480 pixels from the RealSense camera provides a good balance between image quality and processing speed for visual odometry applications.

4.2 Evaluation Metrics

In stereo visual odometry, assessing the performance and accuracy of algorithms is of paramount importance. To gauge the effectiveness and reliability of these algorithms, a suite of evaluation metrics can be used. These metrics provide invaluable insights into the algorithm's capabilities, offering a comprehensive view of its strengths and potential shortcomings.

In the following section, we will delve into a range of critical evaluation metrics tailored for stereo visual odometry. These metrics span various dimensions, including trajectory accuracy, pose estimation, robustness in challenging conditions, computational efficiency, and more. By systematically examining these metrics, we gain a nuanced understanding of how well an algorithm performs under diverse scenarios, ultimately aiding in optimizing the stereo visual odometry systems for specific applications.

4.2.1 Real-time Performance Metrics

1. Absolute trajectory error (ATE)

In the context of visual odometry systems, assessing the global consistency of the estimated trajectory is a vital aspect. This consistency is evaluated by comparing the absolute distances between the estimated trajectory and the ground truth trajectory. However, since these trajectories may initially exist in arbitrary coordinate frames, they need to be aligned for a meaningful comparison. This alignment can be achieved through a closed-form method

developed by Horn, which determines a rigid-body transformation (denoted as S) that optimally maps the estimated trajectory P1:n onto the ground truth trajectory Q1:n.

Once this transformation is obtained, the absolute trajectory error at each time step (i) can be computed using the formula,

$$F_i := Q^{-1}_i \, SP_i$$

Equation 4.1 ATE at time step (i)

The overall performance evaluation can be done by calculating the root mean squared error (RMSE) across all time indices for the translational components, which is expressed as

$$RMSE(F_{1:n}) := \sqrt{(1/n * \sum_{i=1}^{n} ||trans(Fi)||^2)}$$

Equation 4.2 RMSE at all time indices

In simpler terms, this process involves aligning the estimated trajectory with the ground truth, calculating the absolute trajectory error for each time step, and then determining the root mean squared error for the translational components across all time indices (J. Sturm, 2012).

2. Relative Pose Error (RPE)

The Relative Pose Error (RPE) serves as a measure of the local accuracy of a trajectory over a specified time interval (Δ), making it particularly valuable for assessing visual odometry systems. This error metric quantifies the trajectory drift within this interval. At each time step (i), the RPE is calculated as follows:

$$E_i := (Q_i^{-1} \, Q_{i+\Delta})^{-1} \, (P_i^{-1} \, P_{i+\Delta})$$

Equation 4.3 RPE at time step (i)

This procedure generates individual relative pose errors (m = n - Δ) along the trajectory sequence. To provide an overall assessment, we suggest computing the Root Mean Squared Error (RMSE) across all time indices for the translational component as follows:

$$RMSE(E_{1:n}, \Delta) := \sqrt{(1/m * \sum_{i=1}^{m} ||trans(Ei)||^2)}$$

Equation 4.4 RMSE at all time indices

Here, trans (Ei) represents the translational components of the relative pose error Ei. It's worth noting that some researchers may prefer evaluating the mean error instead of RMSE to reduce the influence of outliers. Alternatively, the median can be computed for even greater resilience to outliers. Additionally, while rotational error evaluation is possible, assessing translational errors is often sufficient, as rotational errors manifest as translational errors when the camera is in motion.

Regarding the time parameter Δ, its selection is essential. For visual odometry systems matching consecutive frames, Δ = 1 is a logical choice, where RMSE(E1:n) represents the drift per frame. For SLAM system evaluation, averaging across all possible time intervals Δ is recommended, as expressed by:

$$RMSE(E_{1:n}) := 1/n * \sum_{\Delta=1}^{n} RMSE(E1:n, \Delta)$$

Equation 4.5 RMSE at all time intervals (Δ)

Alternatively, the Relative Pose Error (RPE) can be used to assess the overall trajectory error by averaging it across all potential time intervals (J. Sturm, 2012).

4.2.2 Computational Efficiency

In the context of visual odometry, computational efficiency can be assessed in the following ways:

Processing Speed:

> Evaluate the system's ability to process incoming sensor data, extract features, perform odometry calculations, and update the estimated state.

Resource Utilization:

> Monitor the utilization of CPU and GPU resources during operation. Efficient algorithms should aim to minimize resource wastage.

Memory Usage:

> Assess the amount of memory (RAM) consumed by the system. Efficient algorithms should manage memory efficiently to prevent memory leaks or excessive usage.

Frame Rate:

> Determine whether the system can operate at a specified frame rate (e.g., 30 frames per second) to meet real-time requirements.

4.3 Experimental Setup

For Offline Dataset Testing we outline the chosen dataset, the software and hardware configurations employed, as well as the computational resources utilized. This information will provide a comprehensive understanding of the experimental environment in which the visual odometry algorithm was tested and evaluated.

In the real-time application testing experimental setup, we aim to assess the accuracy and reliability of a visual odometry algorithm by comparing the path it plots with the path tracked by a Leica Hexagon System Laser Tracker. The hardware components include an Intel RealSense D455 camera, securely mounted and calibrated, and the laser tracker set up with an unobstructed line of sight to a reflector attached to the camera. Within a carefully defined courtyard environment, we establish a well-marked path, with varying lighting conditions. Data collection involves synchronizing camera and laser tracker data acquisition, executing the camera's movement along the predefined path, and recording data, including image frames and reflector position and orientation. This setup enables a thorough evaluation of the visual odometry algorithm's accuracy and performance in real-world conditions.

4.3.1 Offline Dataset Testing Setup

- Dataset Selection:

  The experiment utilized the grayscale sequence 01 from the KITTI Visual Odometry dataset.

- Software and Algorithm Setup:

  The experiment was conducted using Ubuntu 20.04 as the operating system and OpenCV version 4.7.0 for implementing and running visual odometry algorithm.

- Computational Resources:

  The processor used for the experiment was a 12th Gen Intel(R) Core (TM) i7-1280P.

  The GPU employed was a VGA compatible controller.

- Offline Testing:

  The visual odometry algorithm was executed offline on the chosen grayscale sequence 01 from the KITTI dataset, encompassing the first 101 frames.

  The system had 32GB of DDR4 RAM available for loading and processing image frames and dataset information.

4.3.2 Real-time Application Testing Setup

a) Hardware Components:

The Intel RealSense D455 camera was securely mounted on the on a fixture as shown in Figure 4.4, which can be easily fixed on a handheld laptop to get a complete camera setup (Figure 4.3). Careful calibration of the camera was ensured to provide accurate depth and image data.

Figure 4.4 Camera fixture

Figure 4.3 Camera setup

The Leica Hexagon System Laser Tracker was positioned in a stable location at height of 4 meters as seen in Figure 4.5 . A clear line of sight from the laser tracker to the camera's reflector was maintained.

Figure 4.5 Tracking System Setup

Reflector and Mounting:

A reflective target or reflector was affixed on a fixture just above the left stereo lens to ensure secure and stable mounting.The reflector was chosen and positioned to facilitate easy detection and tracking by the laser tracker.

b) Environment Setup:

Courtyard and Path:

A defined path or trajectory was established within the courtyard for the camera's movement.The path was marked with, either physical ground markings or other reference points.

The significance of performing different movements and camera orientations in the VO experiment lies in the evaluation and validation of the robustness and accuracy of the pose estimation algorithm under various real-world scenarios. Hereby these movement types are discussed,

1. Straight Line Movement with Forward-Facing Camera (Figure 4.6): This scenario represents typical forward motion, such as moving in a straight line. It helps evaluate the accuracy of your pose estimation algorithm for tracking linear paths.

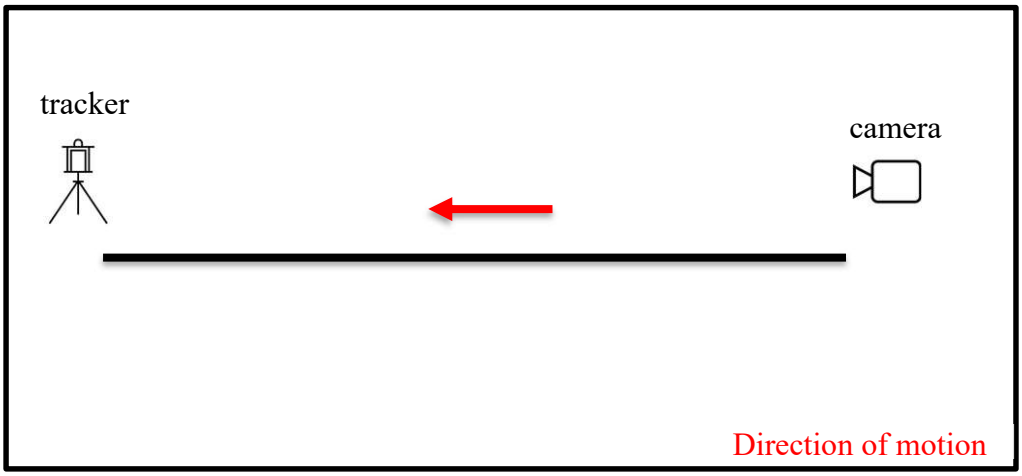


Figure 4.6 Forward straight-line motion

2. Sideways Camera Orientation (Figure 4.7): Turning the camera sideways introduces a challenging scenario where the camera's field of view is perpendicular to the direction of travel. This tests the algorithm's ability to handle scenarios where the camera is not aligned with the primary motion direction, which can occur in real-world situations.

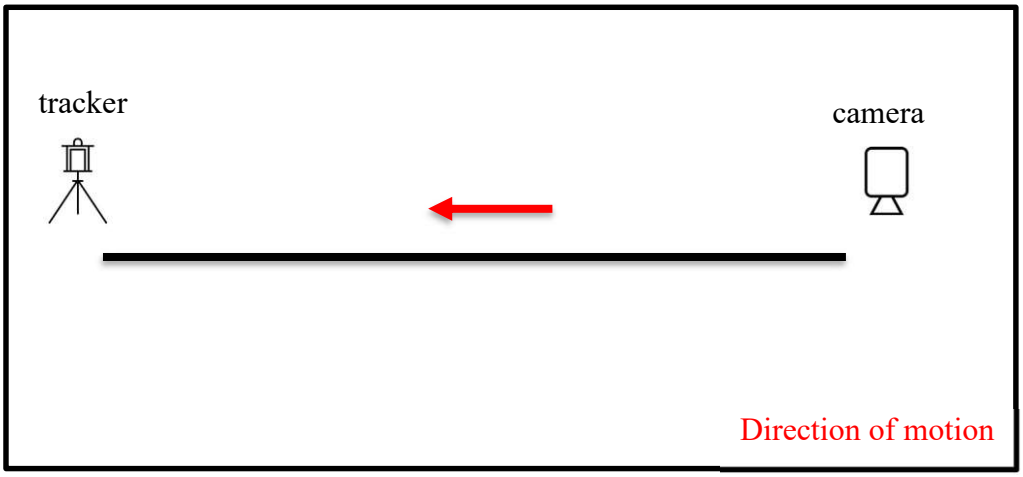


Figure 4.7 Sideways Camera Orientation motion

3. 'L' Shaped Trajectory (Figure 4.8): A 'L' shaped trajectory combines both straight-line and turning movements. It helps assess how well your algorithm handles changes in direction, including sharp turns.

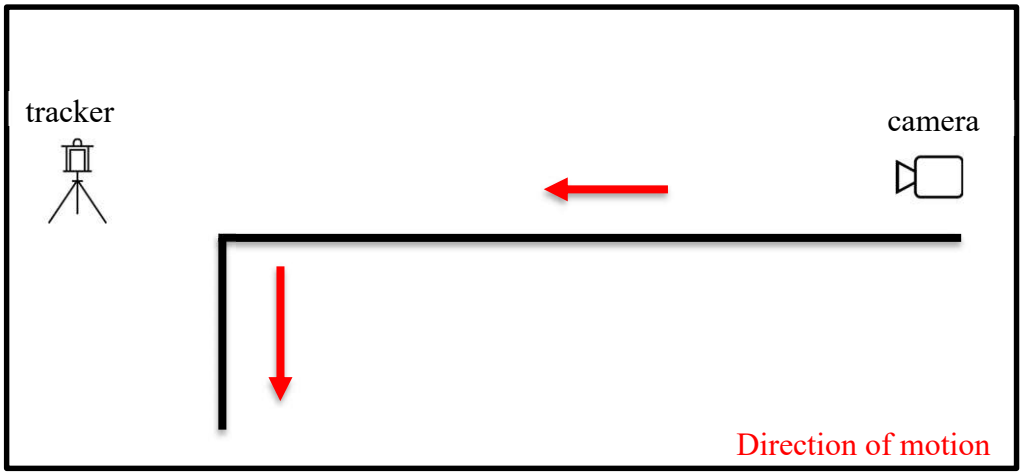


Figure 4.8 'L' Shaped Trajectory motion

4. Free Trajectory with Forward-Facing Camera (Figure 4.9): In this scenario, you allow for more complex and natural movements without specific patterns. It evaluates the algorithm's ability to adapt to unpredictable motion, which is common in real-world environments.

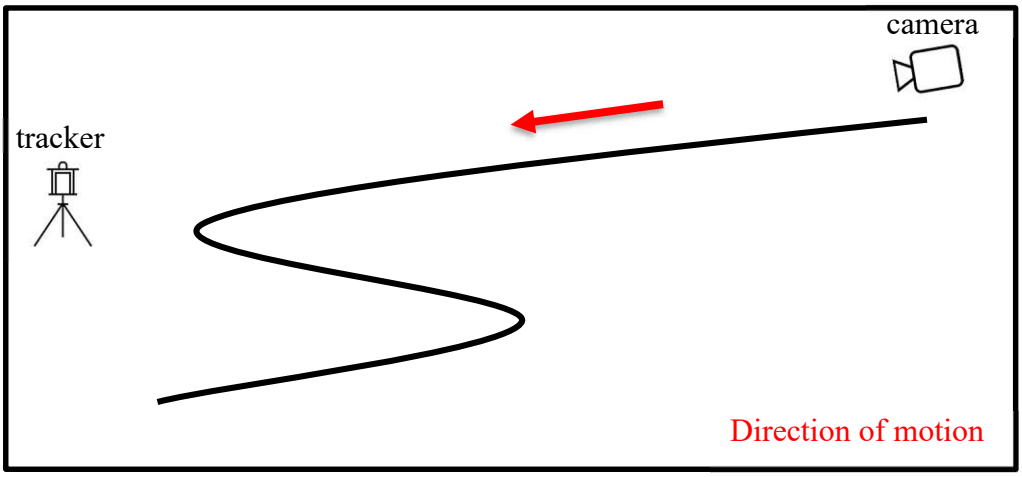


Figure 4.9 Free Trajectory with Forward-Facing Camera

By mimicking various real-world movements, we are testing the algorithm in scenarios that it may encounter in practical applications. This helps ensure the algorithm's reliability in diverse settings. Different movements challenge the algorithm's robustness. It can reveal how well the algorithm copes with changes in motion direction and camera orientation, which are common sources of challenges in pose estimation. It provides an opportunity to fine-tune and calibrate the camera and pose estimation parameters to achieve optimal results in different scenarios. Evaluating the algorithm's performance across different movement types and camera orientations helps assess its generalization capabilities. An algorithm should be able to adapt to a wide range of scenarios. Depending on application (e.g., robotics, augmented reality, autonomous navigation), these different movement types and camera orientations can help you determine if your pose estimation algorithm is suitable for our specific use case.

c) Varied Lighting Conditions:

Lighting conditions within the courtyard were not maintained consistently. The experiment aimed to test the algorithm's robustness by subjecting it to different lighting conditions, including variations in brightness and shadows.

d) Data Collection and Experiment Execution:

Data Recording:

Data recording was carried out for both the Intel RealSense D455 camera, encompassing image frames, and the Leica Hexagon System Laser Tracker, capturing reflector position data as the camera traversed the predetermined path under varying lighting conditions.

Path Execution:

The camera was moved along the predetermined path within the courtyard as planned. Continuous visibility of the camera's reflector to the laser tracker was maintained, ensuring an uninterrupted line of sight.

5 Implementation Details

This section focuses on the practical aspects of building the stereo visual odometry system, including the choice of hardware and software tools, algorithmic implementations, performance optimization techniques, and any specific considerations for real-time processing if applicable.

5.1 Hardware and Software Specifications

Camera Model	Intel RealSense D455 Stereo Camera (Figure 5.1)
Resolution	848x480 pixels
Frame Rate	5,15,30,60,90 fps
Focal Length	1.93 mm

Table 5.1 Intel D455 Specification

Figure 5.1 Intel® RealSense TM Depth Camera D455 (Intel® RealSense™ D455)

Tracking System	Leica Absolute Tracker AT960-MR (Figure 5.2)
3D measurement range	up to 40 m
6DoF measuring range	up to 20 m
measurement rate	up to 1kHz
Accuracy	tens of microns

Table 5.2 Leica AT960-MR Specification

Figure 5.2 Leica Absolute Tracker AT960-MR (hexagon.com, n.d.)

Operating System	Ubuntu 20.04
Programming Language	Python
Integrated Development Environment (IDE)	Visual Studio Code (VSCode)
Libraries and Frameworks	OpenCV 4.7.0, pyrealsense2 2.54.1.5217

Table 5.3 Software Specifications

Processor	12th Gen Intel(R) Core (TM) i7-1280P
GPU	00:02.0 VGA compatible controller: Intel Corporation Device 46a6 (rev 0c)
RAM	32GB DDR4
Storage	512GB SSD

Table 5.4 Computational Resources

Dependencies and Versions**:**

os,numpy(np),cv2(OpenCV),scipy.optimize.least_squares,time,random,mpl_toolkits.mplot3d,tqdm,matplotlib.pyplot(plt),pytransform3d.transformations(pt),pytransform3d.trajectories(ptr),pytransform3d.rotations (pr),pytransform3d.camera (pc), cycler.

5.2 Algorithmic Implementations

5.2.1 Feature Detection and Matching:

The **Detect** method is used for feature detection in an image patch using the **fastFeatures** detector. The **fastFeatures** detector is based on the FAST (Features from Accelerated Segment Test) algorithm, which is a corner detection algorithm. It is computationally efficient and used for detecting keypoints (interest points) in an image for example in Figure 5.3. The keypoints are points that are likely to be stable and repeatable under transformations such as rotations and scale changes.

Figure 5.3 Keypoint detection

- Implementation:

  The code defines a function named **'get_tiled_keypoints'** that takes in three parameters: **'self'**, **'img'**, **'tile_h'**, and **'tile_w'**. This function is used to split an image into tiles and detect the 10 best keypoints in each tile. The **'img'** parameter represents the image to find keypoints in, while **'tile_h'** and **'tile_w'** represent the height and width of each tile, respectively. This function is used for feature matching between stereo frames. Another function defined **'get_kps'** takes in coordinates x and y. It extracts a tile from an image based on

the given coordinates. It then detects keypoints within that tile using FAST feature detection algorithm. The function corrects the coordinates of the keypoints to match the original image. If there are more than 10 keypoints, it selects the top 10 based on their response value. Finally, the function returns a flattened list of all the keypoints detected in the image tiles.

5.2.2 Stereo Correspondence:

The **'track_keypoints'** method tracks keypoints between two frames using optical flow. Optical flow is a computer vision technique used to track the movement of keypoints between consecutive frames in a video. It estimates the displacement of each keypoint from the first frame to the second frame (Figure 5.4).

Figure 5.4 Stereo Correspondence

- Implementation:

  The code defines a function named **'track_keypoints'** that takes in four parameters: **'self'**, **'img1'**, **'img2'**, and **'kp1'**. The function is used to track keypoints between two images. The **'img1'** parameter represents the previous image, **'img2'** represents the current image, and **'kp1'** represents the keypoints in the previous image. There is also an optional parameter **'max_error'** which specifies the maximum acceptable error for tracking. optical flow is used to track keypoints between two images. First, it converts the keypoints from the previous image (kp1) into a vector of points and expands the dimensions. Then, it uses cv2.calcOpticalFlowPyrLK to find the tracked counterparts (trackpoints2) in the current image (img2), based on the previous keypoints (trackpoints1).The status vector (st) is converted to Boolean values to use as a mask. A mask is created to select only the keypoints that are trackable and have an error below the maximum acceptable error. Finally, the mask is applied to filter and update trackpoints1 and trackpoints2. keypoints that are outside the image boundaries are removed. It first retrieves the height and width of the image img1. Then, it

creates a boolean mask in_bounds by checking if the y-coordinates (trackpoints2[:, 1]) are less than the height (h) and if the x-coordinates (trackpoints2[:, 0]) are less than the width (w). Finally, it selects only those keypoints that satisfy this condition and returns them as trackpoints1 and trackpoints2.

The **'calculate_right_qs'** method calculates the right keypoints from the left keypoints and disparity maps. Disparity maps are used in stereo vision to represent the pixel-wise displacement or depth difference between the left and right images. By using the left keypoints and disparity maps, corresponding right keypoints are estimated.

- Implementation:

  This section of the code defines a function named **'calculate_right_qs'** that takes in several parameters: **'q1'**, **'q2'**, **'disp1'**, **'disp2'**, **'min_disp'**, and **'max_disp'**. The function is used to calculate the right keypoints (feature points) based on the given parameters. The default values for **'min_disp'** and **'max_disp'** are set to 0.0 and 100.0 respectively. another function 'get_idxs' that takes in two arrays **'q'** and **'disp'**. It converts the values of **'q'** to integers and uses them as indices to extract corresponding values from the **'disp'**array. It then returns the extracted values along with a Boolean mask indicating whether each value falls within a specified range. The function is called twice with different input arrays (q1, disp1) and (q2, disp2). The returned values are assigned to variables **'disp1'**, **'mask1'**, **'disp2'**, and **'mask2'** respectively. combine the masks **'mask1'** and **'mask2'** using logical AND operation to create a new mask called **'in_bounds'**. Use this mask to filter the feature points (**'q1'**, **'q2'**) and disparities (**'disp1'**, **'disp2'**) that are within the bounds. Then, calculate the right feature points by subtracting the corresponding disparities from the left feature points. Finally, return the filtered left and right feature points (**'q1_l', 'q1_r', 'q2_l', 'q2_r'**).

5.2.3 Depth Estimation:

The **'calc_3d'** method is used to triangulate 3D points from the stereo keypoints. By knowing the stereo calibration parameters and disparity map, this method calculates the depth (3D coordinates) of the keypoints in the scene. This provides an estimation of the depth for the features tracked in the motion.

- Implementation:

  The code defines a method called **'calc_3d'** which takes four parameters: **'q1_l'**, **'q1_r'**, **'q2_l'**, and **'q2_r'**. This method is used to triangulate points from two sets of feature points in stereo images. The purpose of this method is to calculate the 3D coordinates of the feature points seen from two different camera views.
  Convert the feature point arrays **'q1_l'**, **'q1_r'**, **'q2_l'**, and **'q2_r'**to matrices using the np.matrix() function. Also, convert the matrices **'self.P_l'** and **'self.P_r'** to matrices and cast them as float32 using .astype(np.float32). Transpose the matrices **'q1_l_mat', 'q2_l_mat', 'q1_r_mat'**, and **'q2_r_mat'** to get their respective transposes. Then, use the triangulatePoints() function from OpenCV to triangulate points from the i-1'th image (**'P_l_mat', 'P_r_mat', 'q1_l_tran',** and **'q1_r_tran'**) and from the i'th image (**'P_l_mat', 'P_r_mat', 'q2_l_tran'**, and **'q2_r_tran'**). Finally, un-homogenize the resulting homogeneous coordinates by dividing by their fourth element and transpose them using np.transpose().returning the variables **'q1'**and **'q2'**. These variables represent the 3D points seen from two different images.

5.2.4 Camera Pose Estimation:

The **'estimate_pose'** method estimates the transformation matrix (rotation and translation) between two frames using the RANSAC (Random Sample Consensus) algorithm. RANSAC is an iterative method used to estimate parameters of a mathematical model from a set of observed data that may contain outliers. In this case, it helps to estimate the camera's movement between two frames robustly even in the presence of outliers or erroneous feature correspondences.

- Implementation:

  The code defines a function named estimate_pose that takes in five parameters: **'self'**, **'q1'**, **'q2'**, **'q1'**, **'q2'**, and an optional parameter **'max_iter'** with a default value of 100. This function is used to estimate the transformation matrix based on feature points and 3D points seen from two images. The maximum number of iterations can be specified using the **'max_iter'** parameter. here three variables are initialized: early_termination_threshold,min_error,andearly_termination.early_termination_threshold is set to 5, which represents the maximum number of iterations allowed before terminating early.min_error is initialized with a value of infinity (float('inf')). This variable will be used to store the minimum error encountered during the iterations.early_termination is initialized to 0.

It keeps track of the number of consecutive iterations where the error did not improve.

an iterative optimization process is performed. It randomly selects 6 feature points from the given sets of points (**'q1'**, **'q2'**, **'q1'**, **'q2'**). It then uses these sampled points to perform a least squares optimization using the least_squares function. The optimized transformation is evaluated by calculating the error using the reprojection_residuals function. If the error is less than the current minimum error, it updates the minimum error and saves the optimized pose. If no better result is found for a certain number of iterations (early_termination_threshold), the loop breaks. the transformation matrix using the rotation vector and translation vector are obtained from a previous step.First, it extracts the first three elements of out_pose as the rotation vector r. Then, it uses cv2.Rodrigues() function to convert the rotation vector into a rotation matrix R.Next, it extracts the remaining elements of out_pose as the translation vector (t) finally, it calls the _form_transf() method with the rotation matrix R and translation vector t to create the transformation matrix.

The **'get_pose'** method calculates the transformation matrix for each frame in the sequence. It uses the **estimate_pose** method to calculate the camera's relative movement between consecutive frames. By accumulating these transformations, the method provides an estimation of the camera's pose (position and orientation) for each frame in the sequence.

- Implementation:

The method get_pose takes in four parameters: old_imgL, old_imgR, new_imgL, and new_imgR. This method is used to calculate the transformation matrix for a pair of images. It starts by assigning the left images to img1_l and img2_l. Then, it obtains tiled keypoints using get_tiled_keypoints() function. Next, it tracks the keypoints between the two images using track_keypoints() function. After that, it calculates the disparities between the old and new left images using disparity.compute() function. Then, it calculates the right keypoints using calculate_right_qs() function. Finally, it calculates the 3D points using calc_3d() function and estimates the transformation matrix using estimate_pose() function.

After obtaining the camera pose, the 3D points are projected into the image plane using the camera's intrinsic matrix and extrinsic parameters. The difference between the projected 2D keypoints and the actual detected keypoints is computed, giving us the reprojection residuals. These residuals represent the accuracy of the pose estimation and how well the 3D points align with the detected keypoints.

- Implementation:

  The code defines a method called **'reprojection_residuals'**that takes in five parameters: **'dof'**, **'q1'**, **'q2'**, **'q1'**, and **'q2'**. The translation vector is obtained from the '**dof**' array. It then creates a transformation matrix using the rotation matrix and translation vector. Next, it creates projection matrices for two images (i-1 and i) by multiplying the transformation matrix with **'self.P_l'**. Finally, it makes the 3D points homogenize by adding a column of ones to **'q1'** and **'q2'** arrays.3D points from the i-th image are projected onto the i-1 image and vice versa. It first computes the projected points **'q1_pred'** by multiplying **'q2'** (the 3D points in the i-th image) with the forward projection matrix f_projection, and then un-homogenizes them. Similarly, it computes **'q2_pred'** by multiplying **'q1'** (the 3D points in the i-1st image) with the backward projection matrix b_projection, and un-homogenizes them. Finally, it calculates the residuals by subtracting the actual 2D points **'q1'** and **'q2'** from their respective predicted values, and flattens the resulting array before returning it.

5.2.5 Coordinate Transformations:

The **'_form_transf'** method is used to create a transformation matrix from a rotation matrix and a translation vector. In visual odometry, the camera's movement is typically represented as a combination of rotation and translation. The transformation matrix allows for easy conversion between different coordinate systems and enables the accumulation of transformations to estimate the camera's trajectory.

- Implementation:

  The code calculates the transformation matrix **'transf'** using the get_pose function. It then updates the current pose '**cur_pose**' by multiplying it with transf. The updated '**cur_pose**' is appended to the '**camera_pose_list**', and its x and y coordinates are appended to the '**estimated_path**'. Finally, the x and y coordinates of the updated pose are assigned to variables

‘**estimated_camera_pose_x**’ and ‘**estimated_camera_pose_y**’, respectively.

5.3 Performance Optimization Techniques

Visual odometry is a computationally intensive task, and achieving real-time performance while maintaining accuracy is often a delicate balance. Different applications and hardware constraints may require different optimization strategies, and the choice of techniques should be made based on the specific context and performance requirements. Regular benchmarking and profiling are essential to identify performance bottlenecks and ensure that the visual odometry system meets its objectives.

Following performance optimization techniques utilized in this algorithm.

Faster Feature Detection (FAST algorithm):

Feature detection is a fundamental step in visual odometry, where distinctive points in the image, known as keypoints or features, are identified. These keypoints are tracked across consecutive frames to estimate camera motion. It is a popular and efficient feature detection algorithm. It works by comparing the intensity of pixels in a circular neighbourhood around a candidate pixel. It classifies the candidate pixel as a keypoint if a sufficient number of pixels in the circle are significantly brighter or darker than the candidate pixel. The FAST algorithm is designed for speed and reduces computational overhead by minimizing the number of intensity comparisons required. By using the FAST algorithm for feature detection, the code can identify keypoints more quickly, leading to a faster visual odometry pipeline.

Speed and Accuracy of Optical Flow (Lucas-Kanade with pyramids):

Optical flow is the pattern of apparent motion of objects between consecutive frames in a sequence. It is used to track the movement of keypoints from one frame to another. The Lucas-Kanade optical flow method is an iterative algorithm that estimates the motion of keypoints by solving a system of equations. It assumes that the motion of pixels between frames is small and approximates it linearly. Pyramids, in the context of optical flow, refer to the image pyramids, which are multi-scale representations of the image. By creating image pyramids, the optical flow can be estimated at different scales, allowing for large and small motion to be captured effectively. The use of pyramids in the Lucas-Kanade method helps improve the accuracy of optical flow estimation, especially in areas with large motion or significant changes in the scene. More accurate optical flow results in better feature correspondences between frames, which, in turn, enhances the accuracy and robustness of visual odometry.

Efficient Disparity Map Computation (Stereo Block Matching):

In the case of stereo visual odometry, where two cameras are used to perceive depth, computing the disparity map is a crucial step. The disparity map represents the pixel-wise disparity or depth difference between the two stereo images. It helps establish correspondences between points in the two camera views. The stereo block matching algorithm (SGBM) is a common method for disparity map computation. It works by comparing small blocks of pixels in one image with the corresponding blocks in the other image to find the best matching blocks. It is already optimized for speed, making it suitable for real-time applications. By efficiently computing the disparity map, the visual odometry system can obtain reliable depth information, which contributes to better and more accurate motion estimation.

Least Squares Optimization:

Least squares optimization is used in the visual odometry pipeline to refine the initial motion estimates and improve their accuracy. The initial motion between frames is estimated using the motion models (e.g., from optical flow or disparity map). However, these estimates may contain errors due to various factors like noise, occlusions, or dynamic objects in the scene. The least squares optimization formulates an objective function that minimizes the reprojection error, which is the difference between the predicted positions of keypoints (from initial motion) and their actual positions in the image. By iteratively optimizing the motion parameters to minimize the reprojection error, the motion estimates become more accurate, leading to improved camera pose estimation.

6 Experimental Results

6.1 Accuracy and Robustness Analysis

Considering the evaluation metrics discussed in the section 4.2 we evaluate the offline KITTI dataset and the real-time application testing.

6.1.1 Offline Dataset

The Absolute Trajectory Error (ATE) provides a quantitative measure of the accuracy of a pose estimation algorithm by illustrating the absolute positional error between the estimated camera trajectory and the ground truth trajectory. The increasing errors indicate error accumulation over time. This is a common challenge in VO, particularly in scenarios with no loop closures as only first 100 frames of the KITTI sequence '01' is analysed and the aligned trajectories along with error is plotted (Figure 6.1) to check for the basic working and understanding of the Stereo VO algorithm.

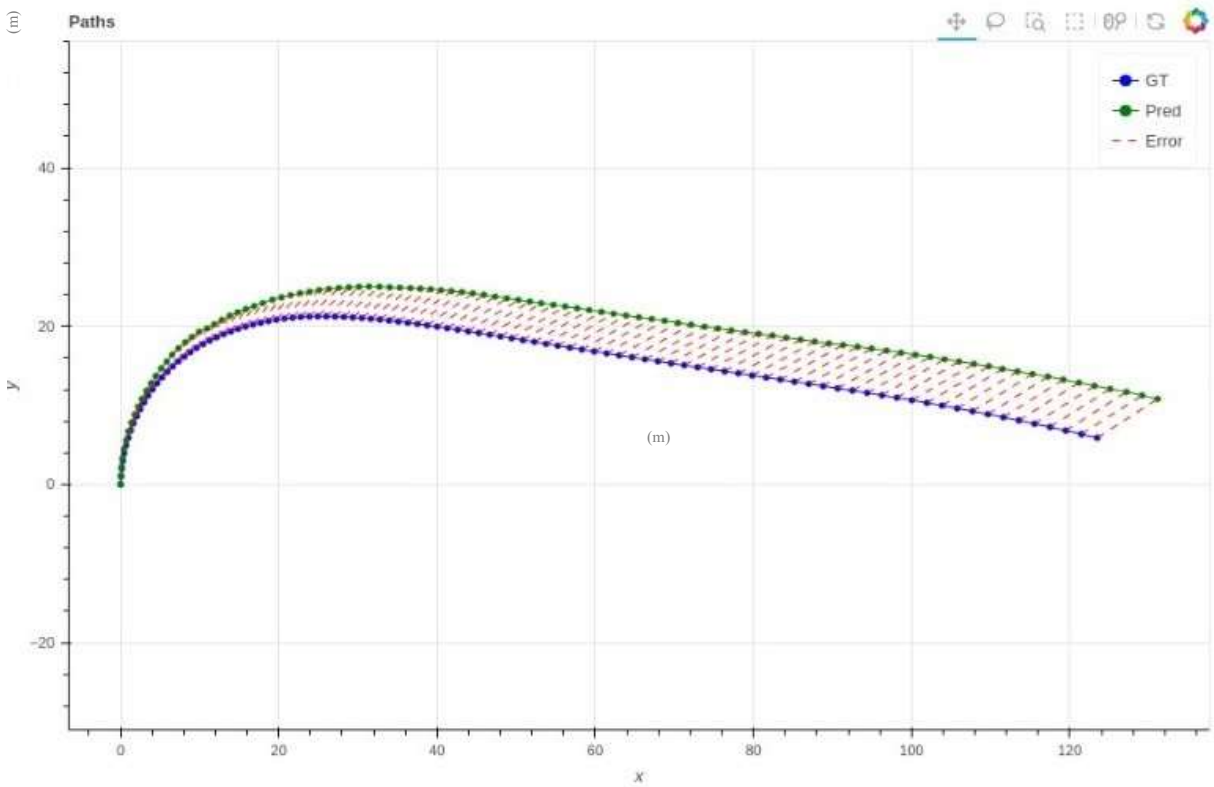


Figure 6.1 Trajectory Plot

This ATE plot (Figure 6.2) reveals a pattern of increasing error values as the image sequence unfolds. Notably, the observed trend corresponds with the sequence's dynamic navigation, characterized by a right turn and changes in direction along the road. The rising ATE values suggest that the VO system encounters challenges in accurately estimating the trajectory during these turns. The dataset presents formidable challenges due to a combination of factors. These challenges include a low frame rate and high driving speeds, resulting in substantial inter-frame motion, with displacements of up to 2.8 meters per frame. Consequently, the potential for establishing feature correspondences between frames is notably constrained. Furthermore, the dataset frequently incorporates moving obstacles, such as passing vehicles, bicycles, and pedestrians, which can exert a significant influence on the performance of visual odometry algorithm. Moreover, in Sequence 01, comprising images captured during highway driving, the difficulty is amplified as it becomes particularly challenging to identify recurring feature points in consecutive frames (Krombach, 2018).

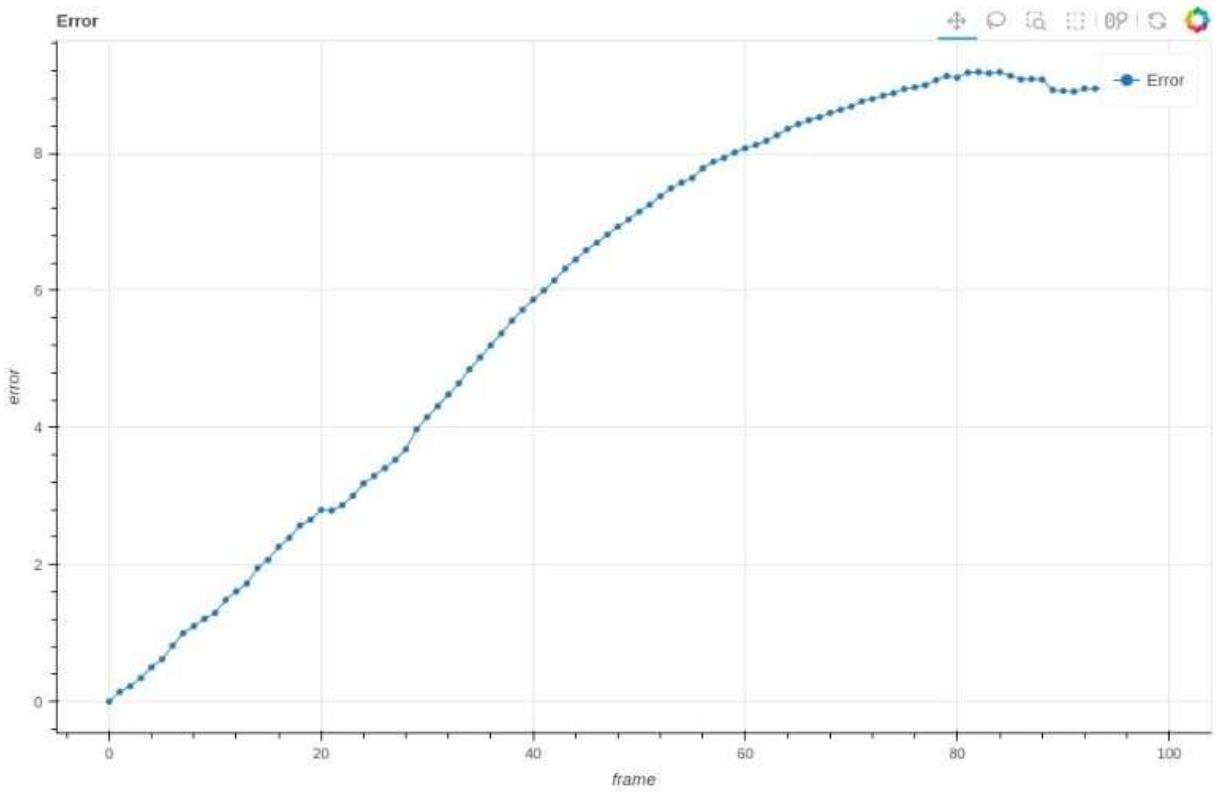


Figure 6.2 ATE Error Plot

6.1.2 Real-time application

Practical application of the developed must be tested in an environment which is semi-controlled so as to imitate the real-world conditions in which an agent would be utilized. The experiment setup is in accordance to the conditions mentioned in the experimental setup section. Here, we delve into a detailed analysis of the results obtained from the experiments designed to evaluate the performance of the stereo visual odometry algorithm under various types of motion.

a) Straight Line Movement with Forward-Facing Camera

The experiment involving straight line movement with a forward-facing camera aimed to assess the algorithm's ability to estimate the motion accurately in a scenario commonly encountered in outdoor navigation and robotics.

Accuracy and Precision

The mean Absolute Trajectory Error (ATE) for the straight-line movement scenario as plotted in Figure 6.3 is 0.733 meters. This value represents the average deviation between the estimated trajectory and the ground truth trajectory.

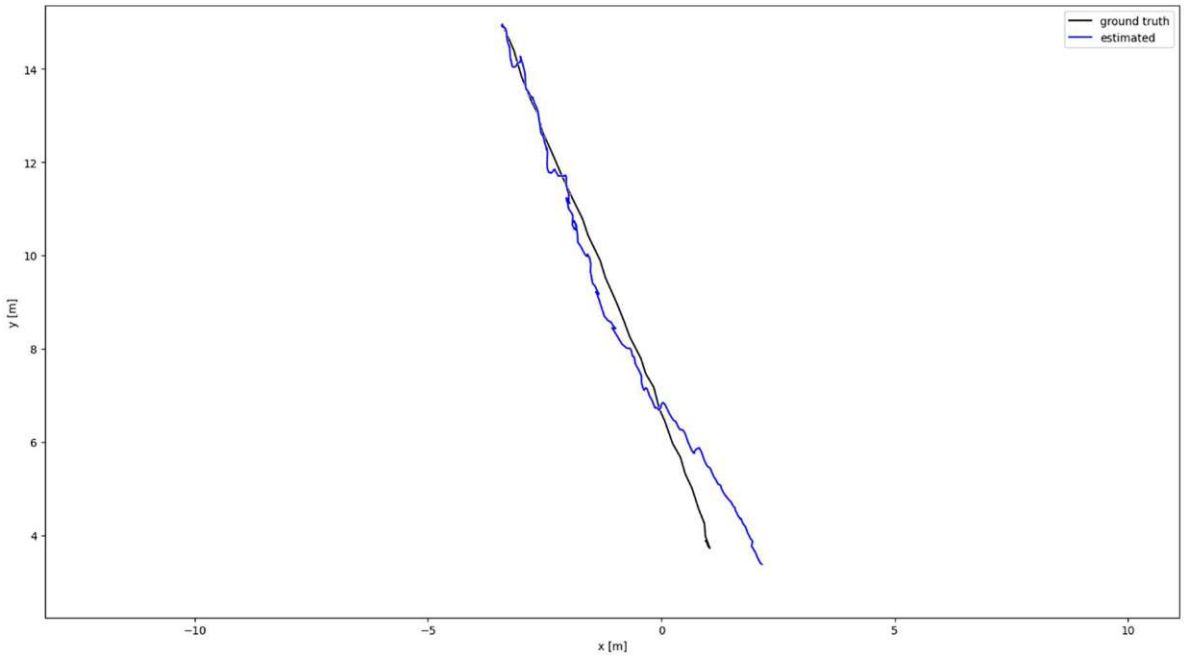


Figure 6.3 Aligned Trajectory: Straight line forward motion

The mean Relative Pose Error (RPE) value of approximately 0.121 meters serves as a measure of the algorithm's overall precision in estimating relative poses. It indicates, on average the algorithm maintains a reasonably consistent level of precision in estimating relative poses between consecutive frames. The majority of RPE plot points cluster closely around the mean, visualized in Figure 6.5. This clustering suggests that the algorithm exhibits a high degree of precision in providing relative pose estimates in most scenarios. When the same type of motion is repeated, the algorithm's output is reliably close to the ground truth. However, the presence of outliers in the RPE plot, with maximum RPE values reaching 0.613 meters and minimum RPE values as low as 0.014 meters, reveals instances where the algorithm's precision deviates significantly from the mean. The observation of the highest outlier at 13.53 seconds, coincides with the first instance of the sun being reflected by the window pane (Figure 6.4), is a

significant reason for this maximum outlier. This increase might be caused by challenges related to changes in lighting conditions or the presence moving reflections impacting the keypoint detection and feature tracking of the algorithm (Figure 6.6).

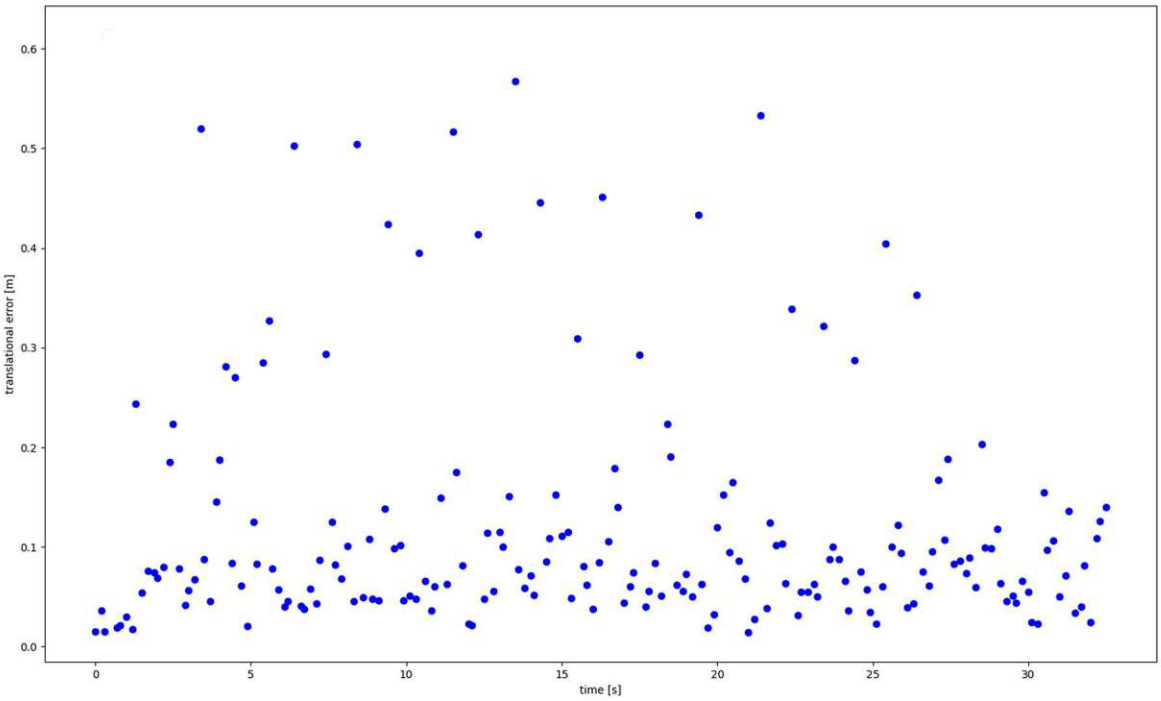


Figure 6.5 Relative pose error: Straight line motion

Figure 6.4 Transition frame: Reflection during the stride (stereo pair)

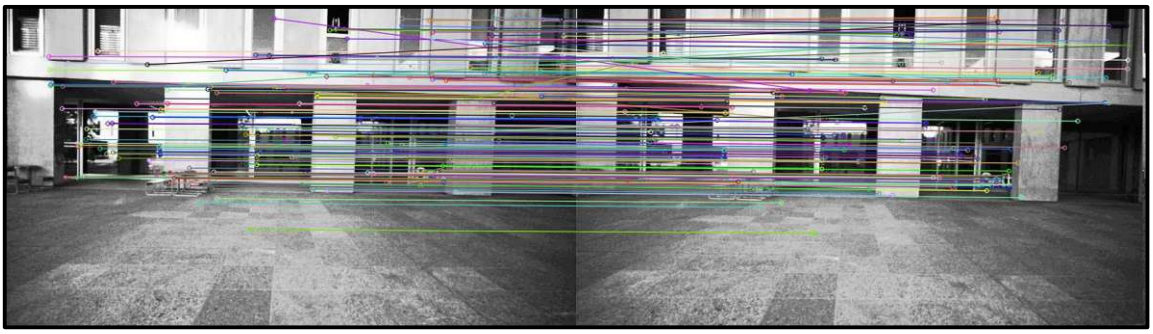


Figure 6.6 feature tracking in transition frame (consecutive left images)

Furthermore, the RPE decreases again when the scene gets closer to the wall and offers more features for the algorithm to track and estimate poses accurately. As the algorithm processes more data and refines its estimation, the decrease over time is achieved through combination of optical flow and Levenberg-Marquardt (LM) optimization. Optical flow is employed to compute the motion of distinct keypoints across successive frames, enabling an initial estimation of the camera's pose. Subsequently, the system computes the reprojection residuals, representing the geometric discrepancy between observed keypoints and their projections in the estimated trajectory. The LM optimization technique iteratively refines the camera's pose by minimizing these residuals, systematically updating the pose parameters to minimize the overall error. This demonstrates the algorithm's adaptability to different scene complexities and its ability to refine accuracy based on available features.

b) Sideways Camera Orientation

The experiment involving sideways camera orientation presented a different challenge, where the camera's orientation was perpendicular to the direction of motion. This scenario is relevant for applications like urban navigation and autonomous driving.

Accuracy and Precision

The mean ATE value of 0.784 meters suggests that, on average, the algorithm's estimated trajectory deviates by approximately 0.7841 units from the ground truth trajectory during sideways camera orientation (Figure 6.7). During this orientation, the camera's field of view included a mix of nearby objects and distant features, which vary the scene's complexity dynamically.

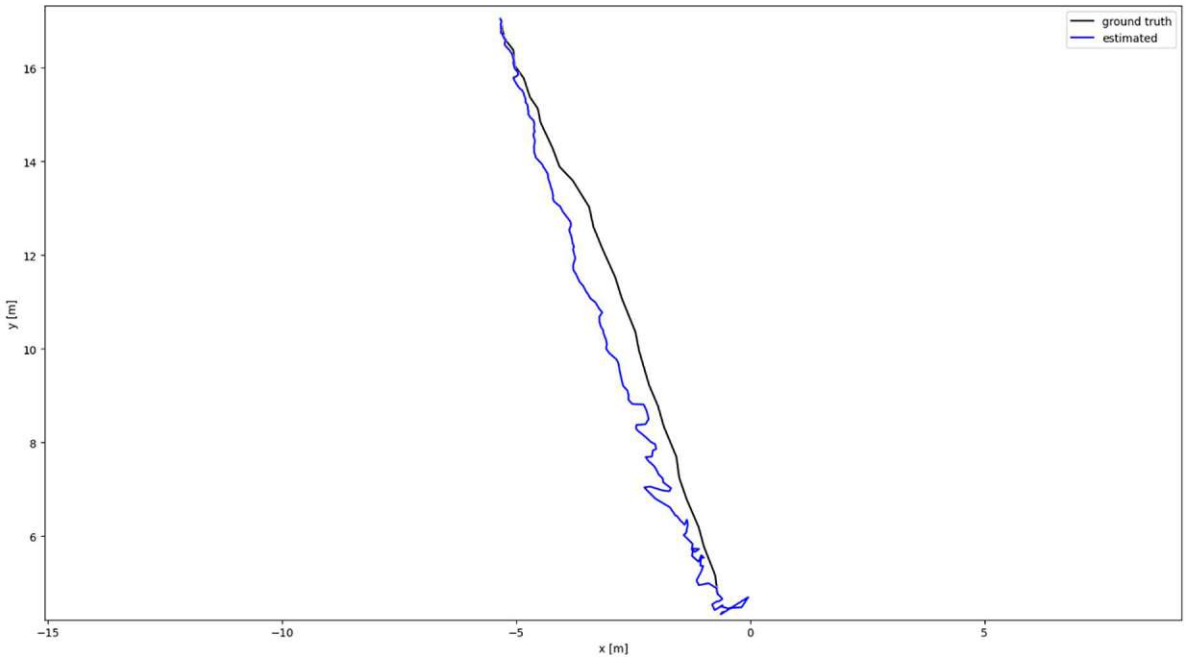


Figure 6.7 Aligned trajectory: sideways camera orientation

The mean RPE value of approximately 0.137 meters indicates the algorithm's average precision in estimating poses relative to ground truth poses during the 'Sideways Camera Orientation' scenario. The presence of outliers in the RPE plot (Figure 6.8), particularly the increasing trend, impacts the algorithm's precision. It's noteworthy that the mean RPE value for forward-facing orientation is 0.121 meters, suggesting that the algorithm performs better when the camera is oriented forward.

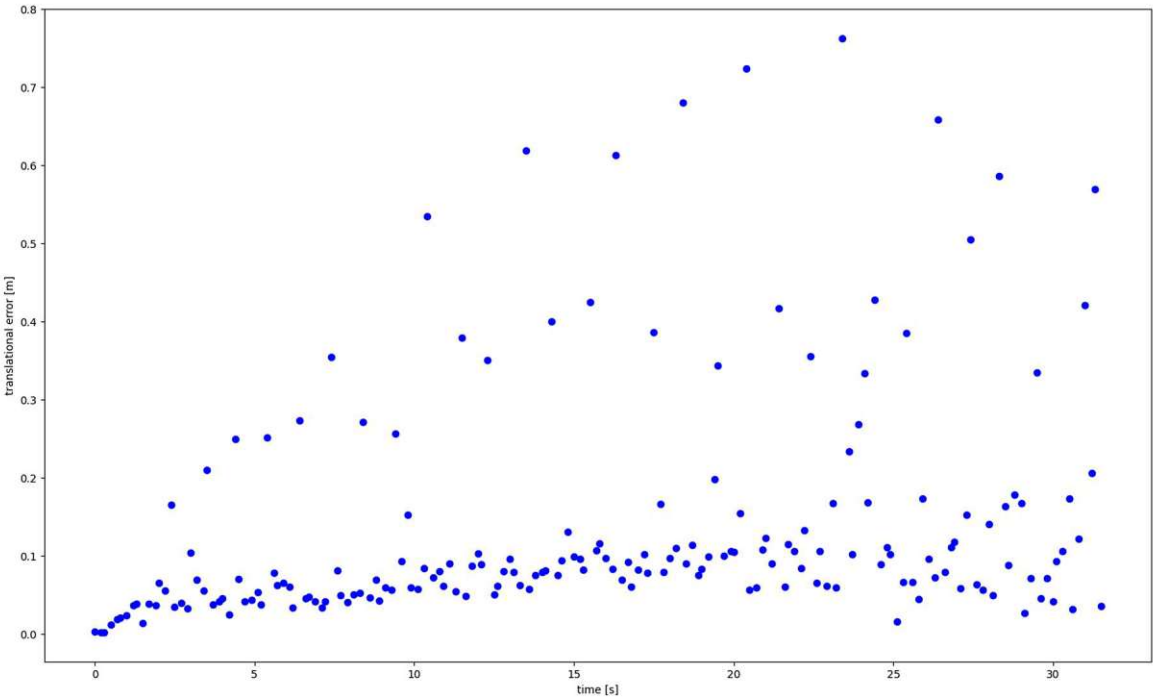


Figure 6.8 Relative pose error: sideways camera orientation

Initially low RPE can be explained by the camera's proximity to various static objects in the scene. When the camera is in close proximity to these objects, the algorithm benefits from more reliable feature tracking and reference points as seen in Figure 6.10 . Consequently, this results in lower RPE values due to the enhanced ability to accurately estimate the camera's pose. The presence of nearby features introduces significant parallax in the camera's field of view, which aids the algorithm in achieving better pose estimation accuracy. The trend of increasing RPE outliers beyond 19 seconds suggests a reduction in the algorithm's precision during this phase. This trend is attributed to the transition of the scene from close proximity to various static objects to a predominantly distant wall, approximately 20 meters away. Distant walls, especially when they lack distinctive features (Figure 6.9), do not provide the same level of parallax as nearby objects. This lack of pronounced parallax in the distant wall scene poses a challenge to the algorithm's ability to estimate the camera's pose accurately. As the camera moves away from the nearby objects, the algorithm faces a challenging situation for pose estimation.

While the number of outliers reduce after the peak at 19 seconds, the fact that their magnitudes continue to increase indicates that, as the scenario progresses, the algorithm's pose estimation errors become more pronounced and less reliable in comparison to ground truth poses. This trend is observed due to the factors such as a lack of distinctive features and reduced scene complexity.

Figure 6.10 Detected close features at 6 sec

Figure 6.9 Detected features at 19 sec

c) 'L' Shaped Trajectory

The 'L' shaped trajectory experiment aims to assess the algorithm's accuracy in handling complex and non-linear motion patterns.

Accuracy and Precision

The mean ATE for the 'L' Shaped Trajectory' scenario is 2.328 meters. The ATE is low until the end of the motion along the x-axis as seen in Figure 6.11. However, the trend changes significantly after a sharp turn is performed. At this point, the ATE increases significantly, indicating that the algorithm struggles to accurately estimate poses after sharp turns.

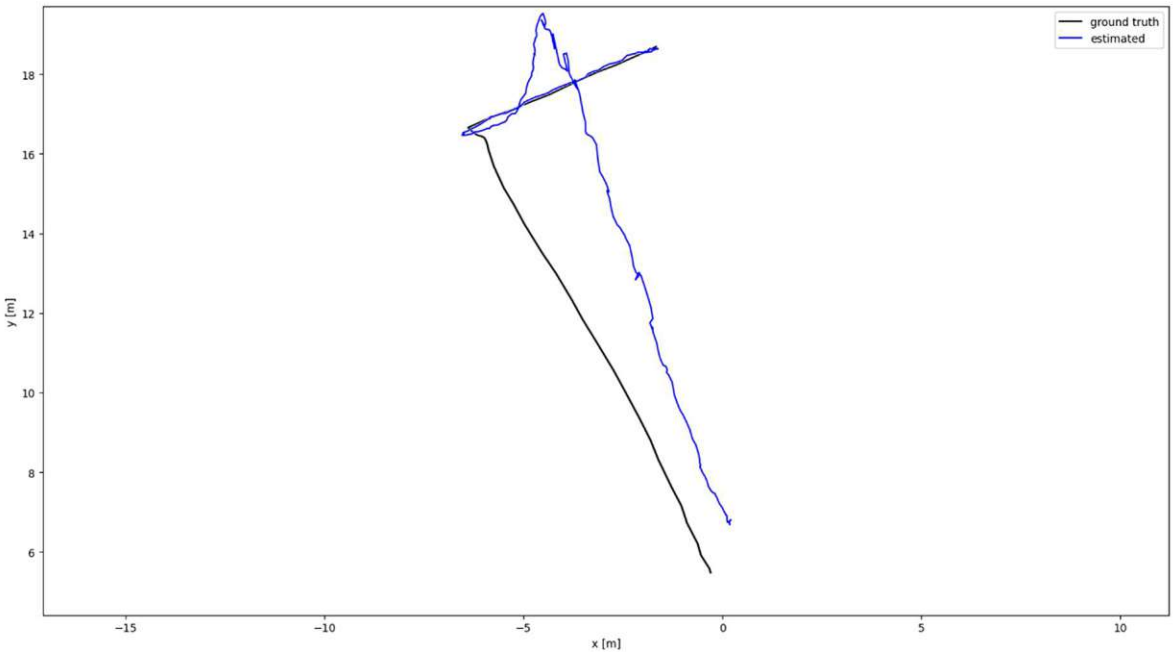


Figure 6.11 Aligned trajectory: 'L' shaped trajectory

During the straight motion along the x-axis, the algorithm exhibits commendable performance, reflected in the RPE mean value of approximately 0.114 meters plotted in Figure 6.12. This relatively low mean RPE indicates that the algorithm maintains a consistent level of precision during this phase, with very few outliers.
Interestingly, the turn phase is characterized by an almost complete absence of RPE outliers, demonstrating the algorithm's reliability in estimating poses during this specific motion pattern. This reliability during the turn can be attributed to the algorithm's ability to handle gradual turn dynamics, even when the camera undergoes simultaneous translation and rotation. During a turn, the algorithm performs well due to abundant motion cues, generating an accurate initial estimate.

However, once the turn is completed and the scene changes abruptly, the algorithm's continued reliance on the outdated initial estimate led to error accumulation. This occurs because subsequent frames are aligned with the inaccurate estimate, causing the camera's estimated pose to drift from the true pose, resulting in increased ATE and RPE values.

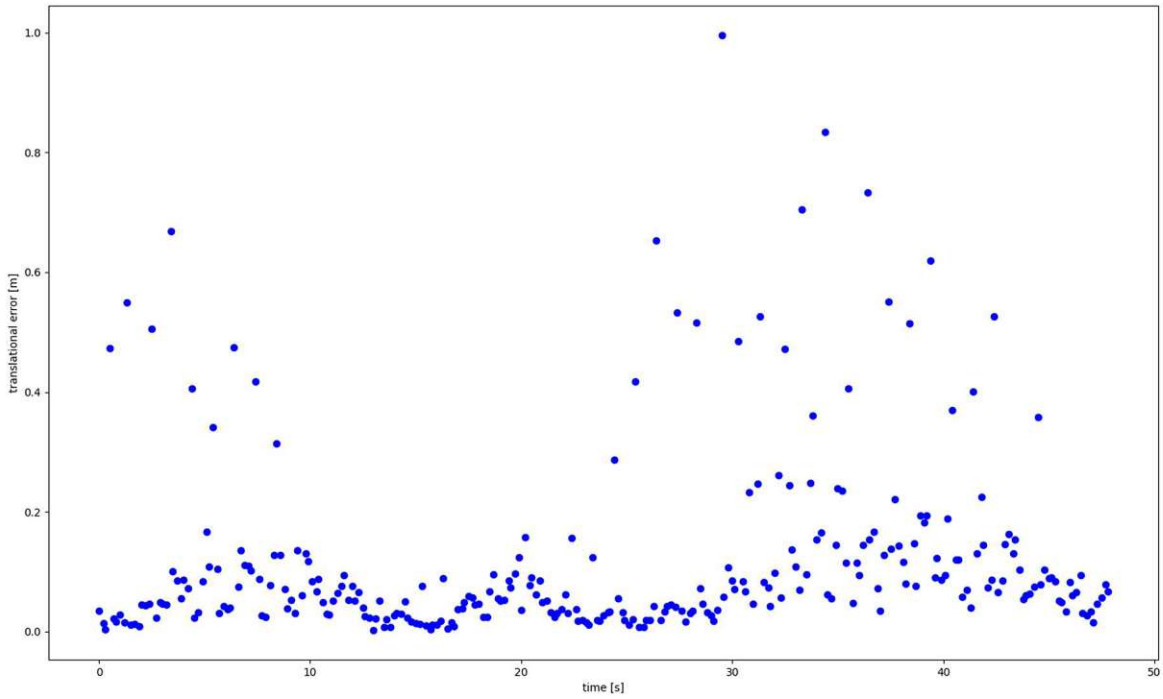


Figure 6.12 Relative pose error: 'L' shaped trajectory

After the turn, the scenario changes as the camera's viewpoint encounters a large reflection in the scene (Figure 6.13). Large reflections introduce confusing depth cues and disrupt feature tracking. In this case, the presence of a large reflection, combined with the algorithm's transition from a well-handled turn to a scene with distorted depth perception, contributes to the sudden increase in RPE outliers. These outliers culminate in the maximum RPE value of approximately 0.932 meters. This observation underscores the impact of scene-specific complexities, such as large reflections on stereo correspondence (Figure 6.14), impacting the algorithm's pose estimation accuracy.

Figure 6.13 Transition frame: large reflection(i.e image saturated areas)

Figure 6.14 Incorrect Stereo Correspondence (transition frame)

d) Free Trajectory with Forward-Facing Camera

The experiment involving a free trajectory with a forward-facing camera simulated real-world scenarios where the robot's motion is unrestricted and can vary significantly.

Accuracy and Precision

The mean ATE for the 'Free Motion Trajectory' scenario is 1.045 meters. The highest ATE value of 1.663 meters observed at the end of the sequence plotted in Figure 6.15, where the motion direction is abruptly flipped. Suggesting that the algorithm performs well in linear straight motions while it struggles more in managing changes in motion direction.

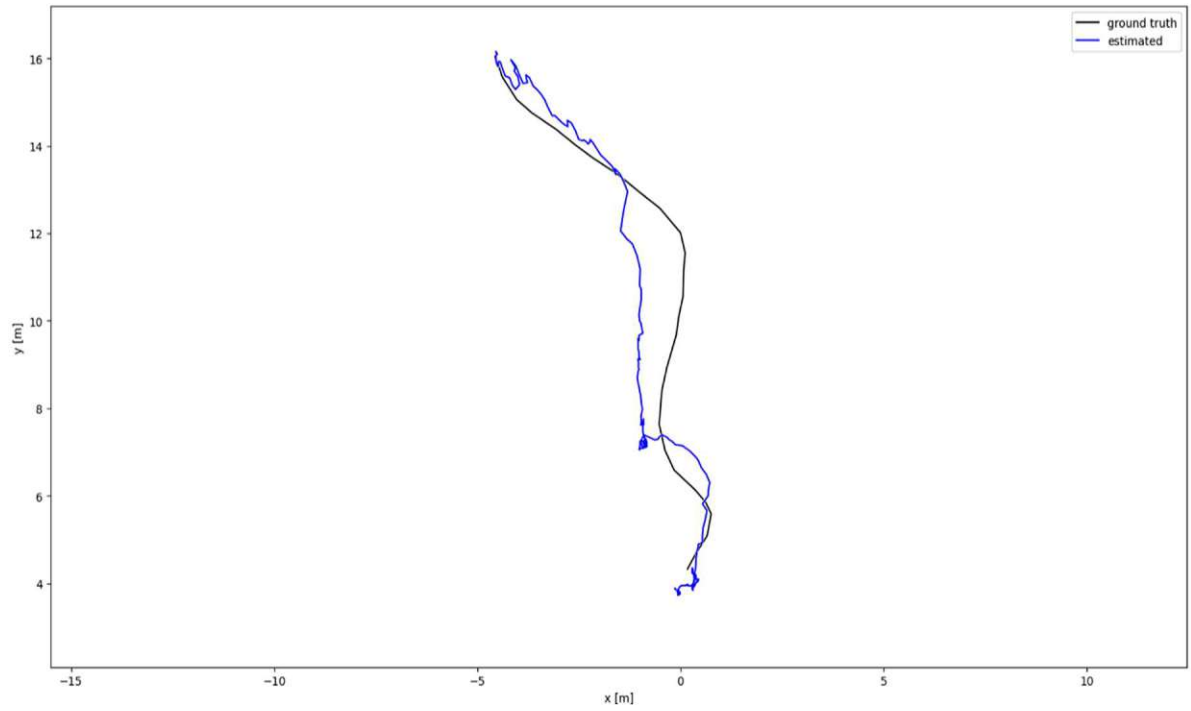


Figure 6.15 Aligned trajectory: Free Trajectory

During straight motion segments, the RPE remains consistent along the mean of 0.175 meters, reflecting the algorithm's ability to maintain precision in estimating poses when the camera motion is relatively predictable and straightforward. The trend of consistent RPE during turns is a positive indicator of precision. It suggests that the algorithm can maintain a reliable level of pose estimation accuracy even during challenging non-linear motion patterns. However, the significant increase in RPE values after the turns is notable particularly around 6 and 15 seconds during the stride as seen in Figure 6.16. This indicates that while the algorithm maintains precision during the turn itself, but faces difficulties in accurately estimating poses immediately following the turn, as observed in the accuracy analysis.

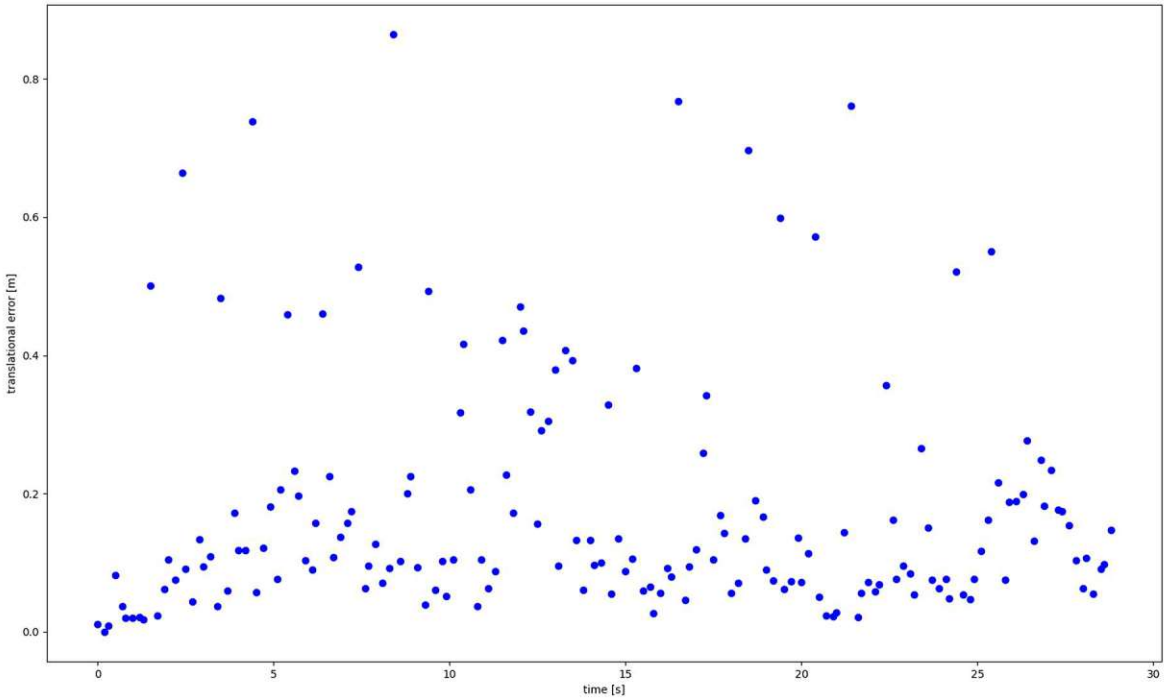
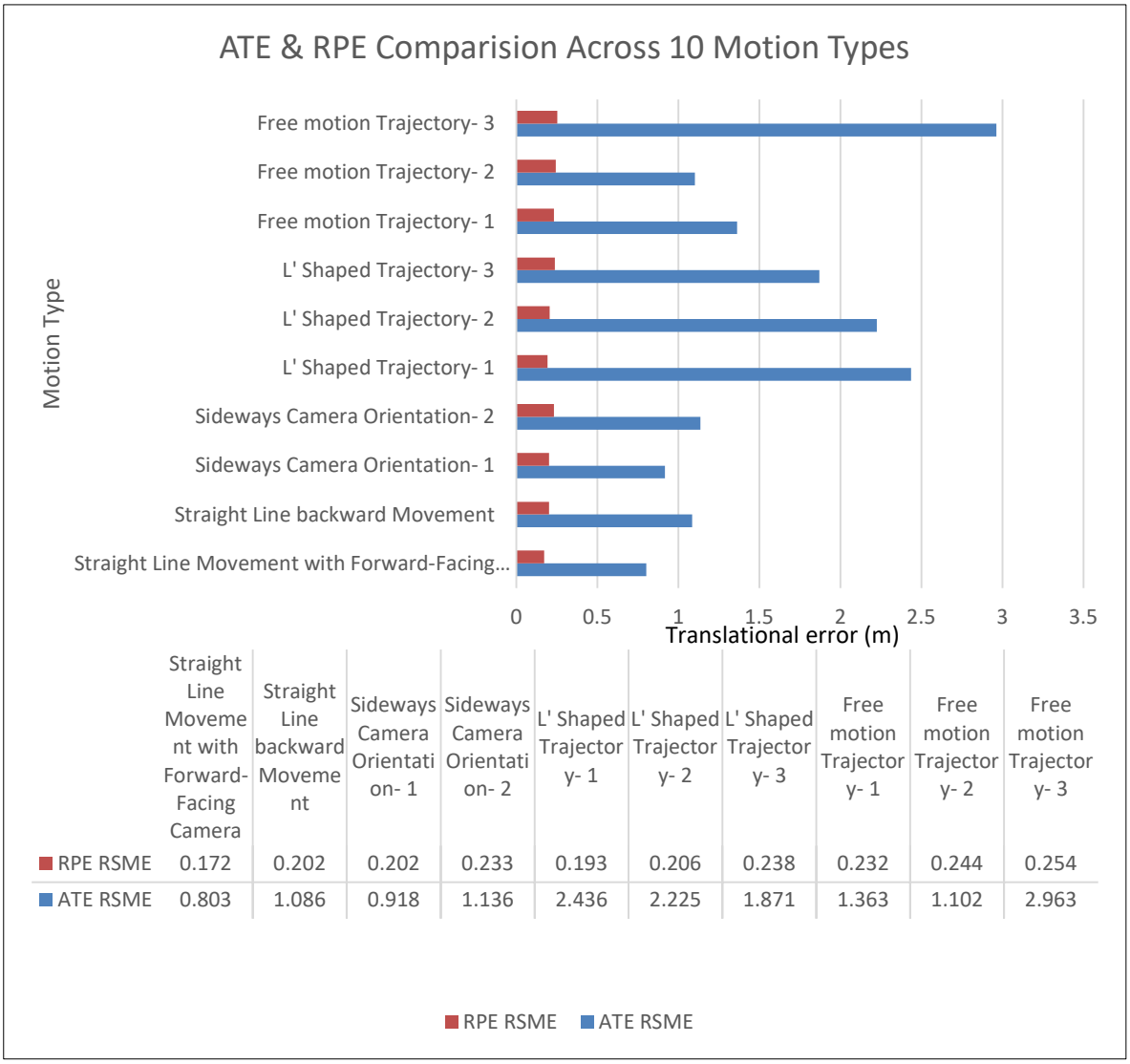


Figure 6.16 Relative pose error: free motion trajectory

The presence of a dynamic moving object (a human being) in the scene (Figure 6.17) at 8.5 second adds complexity to the scenario and this is where the maximum RPE value of 0.989 meters is observed.

Figure 6.17 Transition frame: dynamic object (Human)

Comparative data of all the 10 strides performed during the experiment is enlisted in the Table 6.1.



	Straight Line Movement with Forward-Facing Camera	Straight Line backward Movement	Sideways Camera Orientation- 1	Sideways Camera Orientation- 2	L' Shaped Trajectory- 1	L' Shaped Trajectory- 2	L' Shaped Trajectory- 3	Free motion Trajectory- 1	Free motion Trajectory- 2	Free motion Trajectory- 3
RPE RSME	0.172	0.202	0.202	0.233	0.193	0.206	0.238	0.232	0.244	0.254
ATE RSME	0.803	1.086	0.918	1.136	2.436	2.225	1.871	1.363	1.102	2.963

Table 6.1 ATE & RPE Comparison

Robustness Analysis

The algorithm shows adaptability to changes in scene complexity. When the scene contains more features, such as when it gets closer to a wall, the algorithm refines its accuracy effectively. It can handle variations in scene complexity and progressively improves accuracy as more features become available. This adaptability is a sign of robustness in scenarios with changing environments. Despite its adaptability, the algorithm is sensitive to certain challenging factors. It exhibits an increase in error when there are changes in lighting conditions or the presence of dynamic objects like sun reflections. The presence of large reflections in the scene poses a challenge, leading to a sudden increase in error. This sensitivity suggests that the algorithm might require additional mechanisms to handle abrupt changes in scene characteristics.

In scenes with nearby features having pronounced parallax, the algorithm performs comparatively well. It benefits from parallax when nearby features are present. Higher parallax is beneficial in two critical steps of the Stereo VO algorithm, during feature tracking and triangulation. When tracking keypoints between frames, greater parallax makes it easier to accurately match and track these features, improving feature correspondence accuracy. Moreover, during triangulation, higher parallax enables more accurate and well-conditioned estimation of 3D points from 2D correspondences, contributing to more precise pose estimation, reducing ambiguities in feature tracking and 3D reconstruction. Henceforth, it faces difficulties in scenes with a distant wall and fewer distinctive features.

The algorithm's robustness varies significantly during different phases of a scenario. It struggles during sharp turns, resulting in significant errors. However, during gradual turn dynamics, it maintains good accuracy and precision. The algorithm's performance is consistent during straight motion segments, maintaining precision when the camera motion is relatively predictable and straightforward. However, it faces challenges after abrupt changes in motion direction, leading to increased errors.

Overall, the algorithm exhibits robustness to varying degrees across different scenarios as shown in Table 6.2. It shows adaptability to changing scene complexities, but its sensitivity to certain challenging factors like abrupt changes in lighting or the presence of dynamic objects affects its performance. The algorithm's robustness is influenced by specific scene characteristics.

Motion type	ATE RMSE	ATE mean	ATE max	ATE min	Final ATE	RPE RMSE	RPE mean	RPE max	RPE min
Straight Line Movement with Forward-Facing Camera	0.803	0.733	1.609	0.183	0.506	0.172	0.121	0.613	0.014
Straight Line backward Movement	1.086	0.967	2.267	0.216	0.937	0.202	0.138	0.775	0.004
Sideways Camera Orientation- 1	0.918	0.784	2.020	0.149	0.605	0.202	0.137	0.762	0.002
Sideways Camera Orientation- 2	1.136	1.103	1.807	0.432	0.592	0.233	0.18	0.686	0.005
L' Shaped Trajectory- 1	2.436	2.328	3.618	0.777	1.378	0.193	0.117	0.995	0.002
L' Shaped Trajectory- 2	2.225	2.052	3.148	0.387	2.127	0.206	0.13	0.918	0.006
L' Shaped Trajectory- 3	1.871	1.731	2.823	0.359	1.746	0.238	0.162	0.826	0.009
Free motion Trajectory- 1	1.363	1.217	2.519	0.281	1.323	0.232	0.167	0.766	0.006
Free motion Trajectory- 2	1.102	1.045	1.663	0.352	0.826	0.244	0.175	0.864	0
Free motion Trajectory- 3	2.963	2.671	5.409	1.054	3.752	0.254	0.171	0.989	0

Table 6.2 ATE & RPE Errors

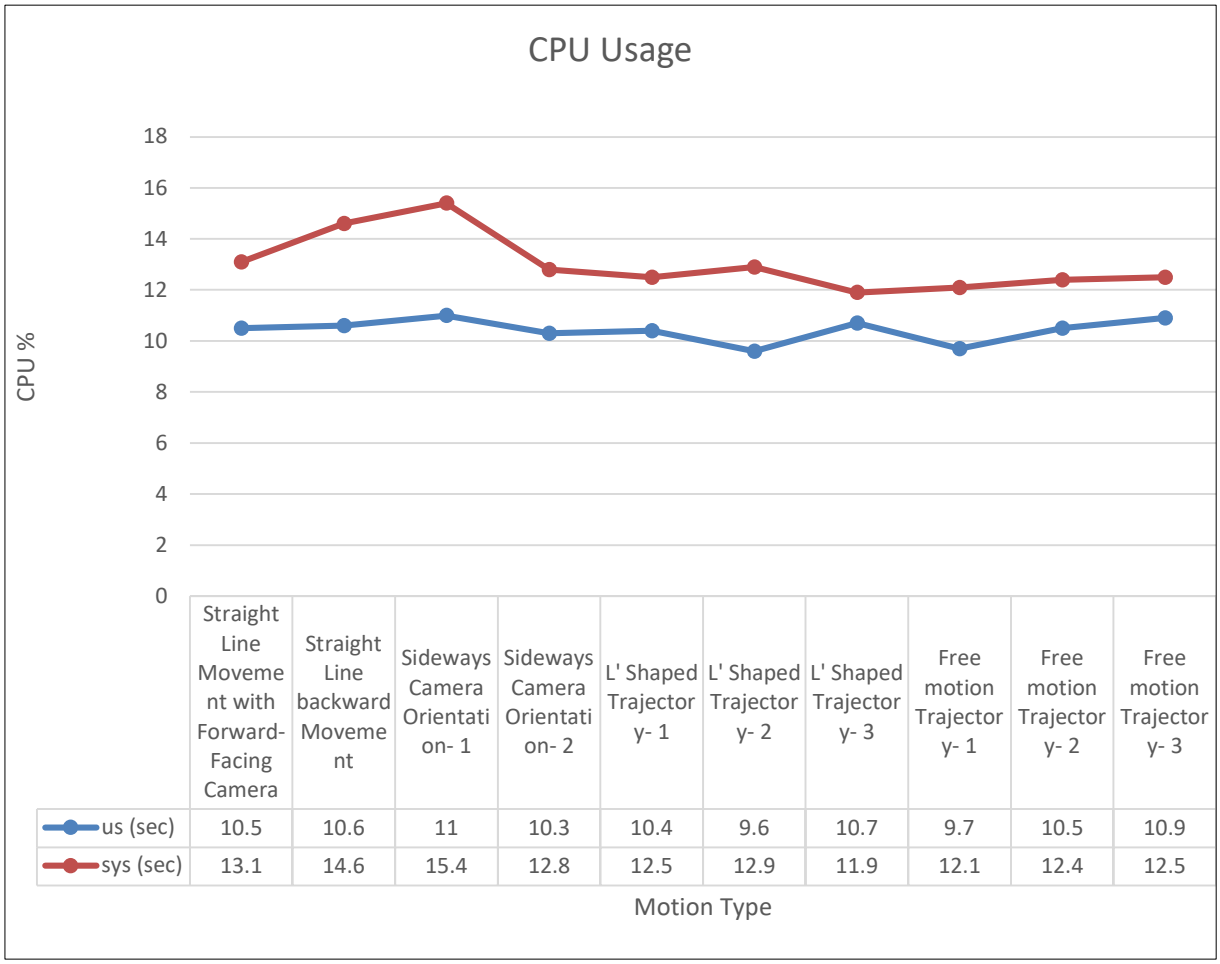
6.2 Computational Efficiency

The performance of VO algorithms not only depends on the ability to accurately estimate pose and trajectory but also on capacity to perform those process swiftly and without burdening the computational resources at disposal. The efficiency with which these estimates are obtained plays a pivotal role in real-time applications. This section delves into two critical aspects of computational efficiency: “CPU Usage” and “Elapsed Time”.

6.2.1 CPU Usage

CPU usage provides valuable insights into the computational demands of the stereo visual odometry algorithm. It is typically categorized into “System” (sys) and “User” (us) components, each representing the proportion of CPU resources allocated to system-level and user-level processes, respectively. System CPU Usage (sys) in context of visual odometry, includes tasks such as data loading, memory management, and other system-related operations necessary for algorithm execution. User CPU Usage (us) encompasses the actual computations, image processing, feature tracking, and pose estimation performed by the algorithm.

Collected CPU Usage data for a diverse set of 10 distinct motion scenarios in Table 6.3 encompass a wide range of motion complexities, from simple linear movements to intricate, high-curvature trajectories. By examining CPU Usage for each of these motions gives insights into algorithm's adaption to different computational demands.

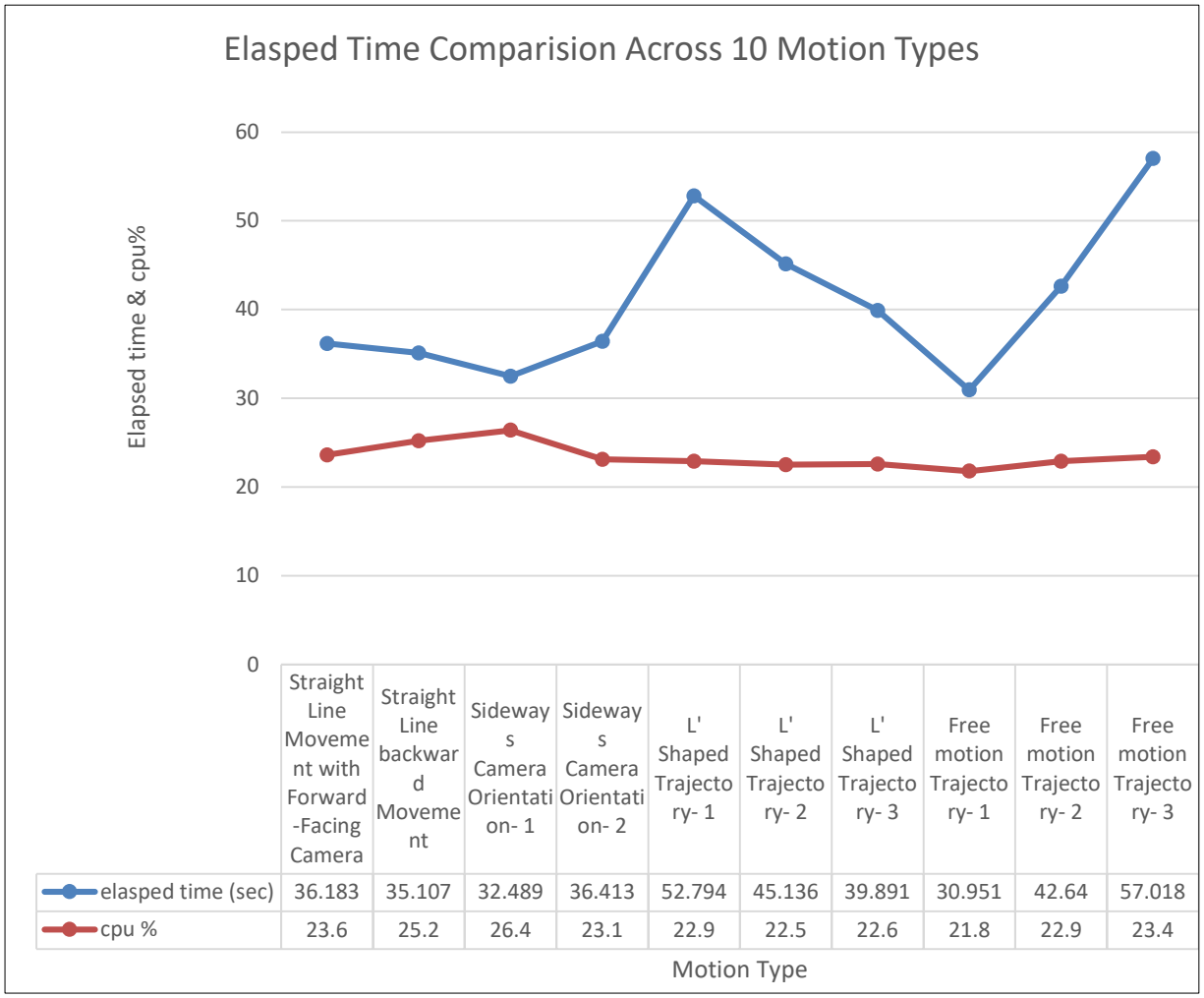


	Straight Line Movement with Forward-Facing Camera	Straight Line backward Movement	Sideways Camera Orientation- 1	Sideways Camera Orientation- 2	L' Shaped Trajectory- 1	L' Shaped Trajectory- 2	L' Shaped Trajectory- 3	Free motion Trajectory- 1	Free motion Trajectory- 2	Free motion Trajectory- 3
us (sec)	10.5	10.6	11	10.3	10.4	9.6	10.7	9.7	10.5	10.9
sys (sec)	13.1	14.6	15.4	12.8	12.5	12.9	11.9	12.1	12.4	12.5

Table 6.3 CPU Usage

6.2.2 Elapsed Time

Elapsed Time, also known as execution time for a visual odometry algorithm measures the actual time it takes for the algorithm to process input data, perform computations, and estimate the robot's pose and trajectory. Elapsed Time encompasses the entire duration of algorithm execution and includes factors like data loading, feature extraction, matching, and pose estimation. This Table 6.4 encapsulates the algorithm's speed and responsiveness when presented with diverse motion complexities and computational workloads.



	Straight Line Movement with Forward-Facing Camera	Straight Line backward Movement	Sideways Camera Orientation- 1	Sideways Camera Orientation- 2	L' Shaped Trajectory- 1	L' Shaped Trajectory- 2	L' Shaped Trajectory- 3	Free motion Trajectory- 1	Free motion Trajectory- 2	Free motion Trajectory- 3
elasped time (sec)	36.183	35.107	32.489	36.413	52.794	45.136	39.891	30.951	42.64	57.018
cpu %	23.6	25.2	26.4	23.1	22.9	22.5	22.6	21.8	22.9	23.4

Table 6.4 Elapsed time comparison

6.3 Limitations and Challenges

This section serves as an exploration into the limitations and challenges that were encountered during the course of this Stereo VO experiment. These challenges encompass various facets of methodology and data collection process, providing essential context for the interpretation of the results.

6.3.1 Impact of Changing Light Conditions

During the outdoor experiment execution, the natural lighting conditions exhibited variations. These fluctuations in natural light conditions are typical in outdoor environments over an extended duration and influence the quality of data and the performance of the sensors and instruments employed in the study. Variations in natural lighting levels resulted in exposure changes, leading to differences in the quality and clarity of captured images between different time intervals. As the outdoor lighting changed, it introduced challenges related to the visibility of features of interest within the experimental scene. Changes in sunlight angles led to the occurrence of shadows or glare, which impact the interpretation of captured data.

To mitigate the impact of changing light conditions, the camera was configured with autoexposure control, which allowed dynamic adjustment of the exposure settings in response to changing light levels. The exposure time was set to a specific value (e.g., 100 microseconds) to ensure that variations were within a manageable range.

6.3.2 Height Placement of Laser Tracker System

The deliberate choice to position the laser tracker system at a height of approximately 5 meters was to satisfy the requirement for the laser beam emitted by the tracking system to be consistently and accurately reflected back from the centre of the reflector. This alignment was critical to obtain larger Field of View (FOV) and ensure the operational reliability of the tracking system. To achieve this alignment, the reflector had to maintain a constant orientation, continually facing the laser tracker. From this vantage point, the tracking system had an unobstructed line of sight to the reflector, and the laser beam could consistently target the reflector's centre. However, variations in height introduce challenges related to calibration and measurement accuracy.

6.3.3 Lack of Start Trigger and System Compatibility

The absence of a synchronized start trigger during the experiment was due to the tracking system and the Stereo VO estimation system operating on different platforms and physically separated during the experiment.

The laser tracking system utilized its proprietary software to collect data, which necessitated a direct cable connection to the tracking system. Conversely, the camera was mounted on a separate laptop, and the VO algorithm was executed on this mobile platform. This physical separation and the differing operating environments posed challenges in achieving a uniform start trigger for both systems. The lack of a synchronized start trigger introduced implications for data synchronization and subsequent analysis. The discrepancies in system start times meant that timestamps for data collection were not perfectly aligned. Consequently, instances occurred where one system collected data while the other did not, leading to timestamps that were occasionally mismatched. These misalignments affected the accuracy of motion estimation and tracking results, especially when precise temporal correlation between ground truth and estimated data was essential.

To mitigate the challenges posed by the absence of a synchronized start trigger and timestamp discrepancies, synchronization of the timestamps was done between the ground truth and estimated data by converting them to the Unix timestamp format. This conversion ensured that the timestamps were expressed in a common time reference, facilitating alignment. To further align the data, interpolation was done to estimate missing data points, ensuring that data collected by one system during periods of misalignment could be estimated and incorporated into the analysis.

7 Applications and Future Directions

7.1 Potential Applications of Stereo Visual Odometry

Stereo Visual Odometry holds immense potential in various domains, revolutionizing industries such as robotics, autonomous driving, virtual reality (VR), augmented reality (AR), and 3D reconstruction. By leveraging stereo camera setups, Stereo VO can accurately estimate the 3D motion of objects, vehicles, or users in their respective environments. Stereo VO enables precise navigation for autonomous vehicles, enhances immersion in virtual environments, aids in robotic tasks, and facilitates the creation of detailed 3D maps.

7.1.1 Robotics

In context of 'Robotics', here only mobile robotic platforms are considered. These platforms can be broadly classified as Land robots, Underwater robots, Space Exploration robots, Agriculture robots. Applications of stereo VO is Land robots are explained in the next section 'Autonomous Driving'.

- Underwater robots

  Fast Visual Odometry for a Low-Cost Underwater Embedded Stereo System (Mohamad Motasem Nawaf, 2018) implemented on BlueROV2 (Figure 7.1), facilitates underwater surveys and run smoothly in real-time. A first post-image acquisition module provides direct visual feedback on the quality of the taken images which helps appropriate actions to be taken regarding movement speed and lighting conditions. It implements a light visual odometry method adapted to the underwater context. The proposed method uses the captured stereo image stream to provide real-time navigation and a site coverage map which is necessary to conduct a complete underwater survey. The visual odometry uses a stochastic pose representation and semi-global optimization approach to handle large sites and provides long-term autonomy, whereas a novel stereo matching approach adapted to underwater imaging and system attached lighting allows fast processing and suitability to low computational resource systems.

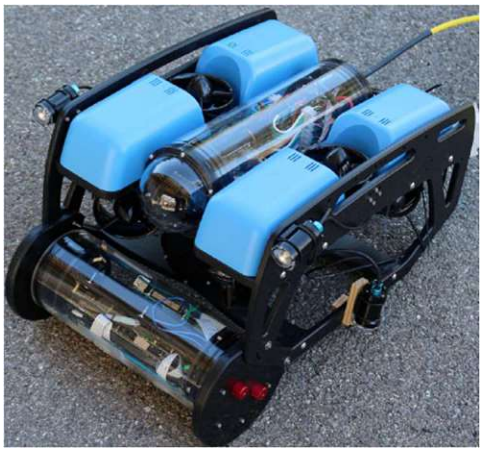


Figure 7.1 BlueROV 2 (Mohamad Motasem Nawaf, 2018)

- Space Exploration robots

Visual odometry for the Mars Exploration Rovers (Figure 7.2) developed by NASA (Y. Cheng, 2006) has presented an approach of position estimation to find features in a stereo image pair and track them from one frame to the next. Visual odometry has been a highly effective tool for maintaining vehicle safety while driving near obstacles on slopes, achieving difficult drive approaches in fewer sols, and ensuring accurate science imaging. Although it requires active pointing by human drivers in feature-poor terrain, the improved position knowledge enables more autonomous capability and better science return during planetary operations.

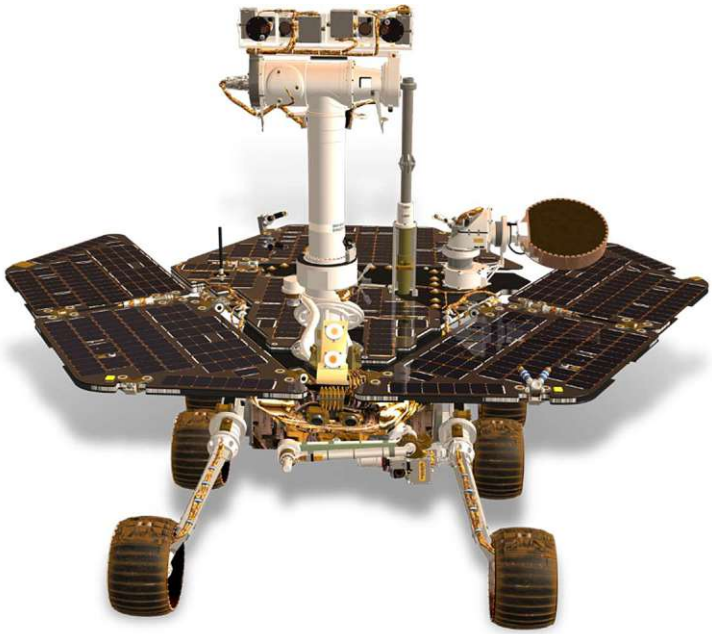


Figure 7.2 NASA's twin MARS rovers (MARS Exploration Rovers, n.d.)

- Agricultural robots

  Vision-based localization and mapping in the agricultural environment is challenging due to the unstructured scene with unstable features, illumination variations, bumpy roads, and dynamic environmental objects. To address these challenges, stereo direct visual odometry system with modifications on Stereo-DSO can be implemented (T. Yu, 2021). Firstly, select some well-matched static stereo points in the latest keyframe to improve the accuracy of inverse depth calculation for tracking. The inverse depth can further distinguish close objects from background, which will avoid large and far-away scene objects in keyframe determination. To boost efficiency and accuracy at the tracking stage, a point selection method to sample map points and remove outliers is used. Furthermore, altitude smoothness verification with a local flat ground assumption and recovery method for tracking failure on bumpy roads are introduced to improve the system's robustness. Finally, a far-away keyframe is reserved in the sliding window to alleviate the orientation drift since the agricultural robots usually move straightly following the crop row.

7.1.2 Autonomous Driving

Autonomous vehicles represent a rapidly advancing and highly challenging application area for visual odometry, especially in unstructured environments. Visual odometry plays a pivotal role in enabling truly autonomous vehicles, as it deals with various critical factors, including dynamic scenes, illumination changes, long-term operation, and large-scale environments. The algorithms designed for this application must address these specific complexities to ensure accurate and reliable motion estimation, allowing autonomous vehicles to navigate safely and effectively in diverse and unpredictable surroundings.

One such approach is vehicle localization in dense urban environments using a stereoscopic system and a GPS sensor (Lijun Wei, 2011). A stereoscopic system is employed to capture stereo video footage, enabling the reconstruction of the environment and estimation of vehicle motion. This process involves feature detection, matching, and triangulation from each pair of images. To ensure accuracy, a relative depth constraint is applied, which eliminates tracking inconsistencies caused by vehicle ego-motion. The optimal rotation and translation between the current and reference frames are computed using an RANSAC-based minimization technique. Additionally, GPS positions obtained from an on-board GPS receiver are periodically used to adjust the estimated vehicle orientations and positions derived from stereovision.

To validate the proposed method, it was tested using real sequences acquired from a GEM vehicle equipped with a stereoscopic system and an RTK-GPS receiver. The results demonstrate that the integration of vision and GPS data yields a trajectory that aligns better with the ground truth compared to using vision alone, particularly concerning vehicle

orientation. Conversely, the stereovision-based motion estimation method helps rectify issues with the GPS signal, such as failures caused by multipath problems or other noises. In essence, this approach combines the strengths of both vision and GPS technologies, providing more robust and accurate motion estimation for autonomous vehicle applications.

Another application is DARPA's LAGR (Learning Applied to Ground Robotics) vehicle (Figure 7.3 DARPA LAGR vehicle (James S. Albus, 2006)) (Howard, 2008). it is equipped with two stereo camera pairs, wheel encoders and a MEMs IMU. A very simple filter is employed that fuses visual odometry, wheel encoder and IMU data. Most of the time, the filter simply integrates motion estimates from visual odometry, but falls back on encoders and IMU data when the visual odometry reports failure.

Figure 7.3 DARPA LAGR vehicle (James S. Albus, 2006)

7.1.3 Virtual Reality and Augmented Reality

VR/AR technology is a highly popular and productive research focus because of its wide range of potential applications and development directions. There are, however, existing technical difficulties relating to three core issues of augmented reality technology: realizing stable and accurate virtual three-dimensional registration, realizing virtual reality integration, and realizing natural human-computer interaction. Visual-inertial odometry (VIO) is the fusion of information measured by the visual sensor and by the inertial measurement unit. The latter is used to calculate the motion relationship of the sensor between adjacent time frames, and so calculate the motion trajectory. In a VR head mounted display (VR HMD), the estimated pose of the user's head is used as the starting point of the virtual reality scene rendering process (Mandal, 2019).

One of the application stereo visual odometry can be easily demonstrated by 'ZED Mini' as shown in Figure 7.4 ZED - mini (zed-mini, n.d.). It is a mixed-reality camera integrating aspects of both virtual and augmented reality. It has a Stereo-IMU camera setup which mimics the way human eyes perceive the world. The camera captures high-resolution stereo video and integrates it with dynamic space mapping and tracking which blends the virtual elements in real world scene. (zed-mini, n.d.)

Figure 7.4 ZED - mini (zed-mini, n.d.)

7.2 Future Research Directions

Stereo odometry based on careful feature selection and tracking SOFT2 is a VO model capitalizing on the limitations established by epipolar geometry and kinematics, tailor-made for setups that lack pure rotation characteristics. The approach minimizes the distances between points and epipolar lines, thus ensuring robustness against uncertainties in object depth. The initial phase involves estimating motion up to scale using a single camera. Subsequently, a simultaneous estimation of absolute scale and extrinsic rotation for the second camera is proposed, mitigating the impact of fluctuating stereo rig extrinsics. To refine motion estimates across a temporal frame window, an epipolar line bundle adjustment procedure is employed. Furthermore, a novel approach employing multiple hypotheses for feature matching is introduced, designed for self-similar planar surfaces that incorporate appearance changes due to perspective shifts.

Human Visual Attention Mechanism-Inspired Point-and-Line Stereo Visual Odometry (PLWM-VO) for Environments with Uneven Distributed Features (Wang C. Z., 2023) is proposed to describe scene features in a global and balanced manner. A weight-adaptive model based on region partition and region growth is generated for the human visual attention mechanism, where sufficient attention is assigned to position-distinctive objects (sparse features in the environment). Furthermore, the sum of absolute differences algorithm is used to improve the accuracy of initialization for line features. Compared with the state-of-the-art method (ORB-VO), PLWM-VO shows a reduction in the absolute trajectory error. Although the time consumption of PLWM-VO is higher than that of ORB-VO, online test results indicate that PLWM-VO satisfies the real-time demand. The proposed algorithm not only significantly

promotes the environmental adaptability of visual odometry, but also quantitatively demonstrates the superiority of the human visual attention mechanism.

T-ESVO: Improved Event-Based Stereo Visual Odometry via Adaptive Time-Surface and Truncated Signed Distance Function (Zhe Liu, 2023), incorporates event cameras which have potential to be an excellent complement for standard cameras within various visual tasks, especially in illumination-changing environments or situations requiring high-temporal resolution. Hereby, an event-based stereo visual odometry (VO) system via adaptive time-surface (TS) and truncated signed distance function (TSDF), namely, T-ESVO, is proposed. The system consists of three carefully designed components, including the event processing unit, the mapping unit, and the tracking unit. Specifically, the event processing unit adopts a novel spatial–temporal adaptive TS that can deal with different camera motions in various environments. The mapping unit introduces the TSDF to describe the 3D representation of environments and achieves depth estimation based on the global historical depth information contained in the environmental TSDF description. The tracking unit achieves the 6-DoF pose estimation through an 3D–2D registration method based on the left/right TS selection mechanism and the depth point selection mechanism. The experimental results show that T-ESVO achieves good performance in both accuracy and robustness when compared with other state-of-the-art event-based stereo VO systems.

Stereo Visual Odometry Approach Based on Optical Flow and Depth Information (Duan C, 2023) is a VO model that capitalizes on the synergy between optical flow and depth information. Unlike certain monocular VO techniques, this approach eliminates the need for extra frames or initialization with external information to establish absolute scale, and it also addresses the consideration of moving objects. By fusing optical flow and depth information, a novel framework for stereo VO is established, leveraging deep neural networks. This framework facilitates the simultaneous generation of optical flow and depth outputs from consecutive pairs of stereo RGB images. This method surpasses established learning-based and monocular geometry-centred techniques achieving real-time performance, thereby establishing the method's dual advantage of effectiveness and efficiency.

Stereo Visual Odometry with Deep Learning-Based Point and Line Feature Matching using an Attention Graph Neural Network (Shenbagaraj Kannapiran, 2023) is an approach to utilize both point and line features and incorporates an innovative feature-matching mechanism driven by an Attention Graph Neural Network. This mechanism is effective in challenging weather conditions like fog, haze, rain, and snow, as well as in situations with varying lighting conditions such as nighttime illumination and glare. The outcomes reveal that this approach outperforms existing line matching algorithms, achieving a higher number of successful line feature matches. When combined with point feature matches, this method consistently demonstrates strong performance in adverse weather and dynamic lighting environments.

8 Discussion

This section provides answers to the research questions that were posed. It addresses these questions in sequence, beginning with "What different image processing and feature-based techniques can be employed to compensate for external sensor limitations?" Following that, it examines "How can image processing and feature-based techniques be utilized to enhance the accuracy, robustness, and computational efficiency of stereo VO?" Subsequently, it delves into the inquiry "How can the parameters of a VO algorithm be adjusted to achieve optimal real-time performance while maintaining robustness in the face of changing lighting conditions and occlusions?".

Various image processing and feature-based approaches to compensate for external sensor limitations

The Feature detector based on the FAST (Features from Accelerated Segment Test) algorithm, Stereo Semi-Global Block Matching (SGBM) correspondence, depth information from stereo images, estimating camera pose using the RANSAC (Random Sample Consensus) algorithm and tracking the camera poses using combination keypoint correspondences optical flow and disparity calculation constitute into an effective visual odometry technique which can compensate for stand-alone navigation using data from IMU (Inertial Measurement Unit), GPS, or wheel encoders.

Properly calibrated stereo cameras provide accurate information about the robot's environment. When fused with IMU data, this enables precise positioning and orientation estimation, even in challenging situations where IMU data alone might suffer from drift or noise. Calibrated cameras can also assist in better feature tracking, improving the quality of data used for navigation. Stereo correspondence algorithms help establish correspondences between stereo images, leading to 3D reconstruction of the environment. By combining this 3D information with data from wheel encoders, it becomes possible to create a more reliable odometry system. Wheel encoders may suffer from slippage or inaccuracies over time, but visual odometry can help correct for these errors. Estimating depth information from stereo images provides valuable context for navigation. Integrating this depth data with GPS can enhance localization, especially in environments with poor GPS signal quality, such as urban canyons or dense forests. Additionally, combining depth data with wheel encoder readings can help correct for wheel slippage or irregularities. Pose Estimation and Motion Tracking modules enable accurate tracking of the robot's motion and pose. When fused with IMU data, they help correct for short-term IMU drift. By continuously refining the robot's estimated position and orientation using visual odometry, IMU data, and GPS information, navigation accuracy improves, and cumulative errors are minimized. Optimization techniques refine motion and pose estimates, making them more accurate and stable. When applied in conjunction with IMU, GPS, or wheel

encoder data, optimization helps maintain consistent and accurate navigation results over time. It can also address sensor noise and bias, further enhancing overall navigation performance.

In summary, integrating these image processing and feature-based techniques with data from IMU, GPS, or wheel encoders offers several advantages for navigation. Enhanced accuracy can be achieved by combining multiple data sources and using visual cues from stereo cameras, navigation systems can achieve higher accuracy than relying on any single sensor alone. These techniques compensate for the limitations of individual sensors. For instance, they can correct for IMU drift, GPS signal loss, wheel encoder errors, and other sensor-related issues. The continuous processing and optimization provided by these techniques allow for real-time adaptation to changing environmental conditions, ensuring robust and reliable navigation. Cumulative Errors are reduced by refining estimates and minimizing errors at each step, assisting the navigation system to maintain accuracy even during extended operations. Incorporating these techniques into navigation systems can significantly improve their performance and reliability when dealing with data from IMU, GPS, or wheel encoders.

Optimization of the precision, resilience, and computational efficiency of stereo visual odometry

While the FAST algorithm primarily focuses on speed, it indirectly contributes to accuracy by rapidly identifying keypoints or distinctive points in stereo images. These keypoints serve as essential reference points for tracking camera motion accurately. Efficiently detecting features, the FAST algorithm helps maintain robustness in challenging scenarios, such as low-texture environments or scenes with motion blur. Robust feature detection ensures a consistent stream of relevant data for motion estimation. The FAST's ability to identify keypoints quickly reduces the computational overhead in the visual odometry pipeline. This efficiency is crucial for achieving real-time performance in stereo VO applications.

The Lucas-Kanade optical flow method, when used in conjunction with image pyramids, improves the accuracy of optical flow estimation. Image pyramids provide a multi-scale representation of the image, enabling the algorithm to handle large and small motion accurately. This enhanced accuracy directly benefits stereo VO by providing precise information about how keypoints move between frames. Accurate optical flow estimation enhances the robustness of stereo VO, especially in scenes with substantial motion or significant changes. It ensures that feature correspondences are reliable, even in complex environments. Despite its iterative nature, the Lucas-Kanade method is computationally efficient, particularly when combined with image pyramids. This computational efficiency supports real-time processing in stereo VO applications.

Efficient Disparity Map Computation using Stereo Block Matching algorithm (SGBM) efficiently computes the disparity map, which represents depth differences between stereo images. This depth information greatly enhances the accuracy of stereo VO. Accurate depth information helps in better understanding the 3D structure of the environment, improving

motion estimation. Reliable disparity map computation contributes to the robustness of stereo VO, especially when dealing with challenging scenes containing occlusions or varying lighting conditions. It ensures that the system can maintain accurate feature correspondences. SGBM is optimized for speed, making it well-suited for real-time applications. Its efficiency in computing the disparity map allows for swift and continuous depth estimation, which is essential for accurate stereo VO.

Least squares optimization plays a crucial role in refining initial motion estimates. By minimizing reprojection errors between predicted and actual keypoints positions, this optimization technique significantly improves the accuracy of camera pose estimation in stereo VO. The iterative nature of least squares optimization helps mitigate errors in initial motion estimates caused by factors like sensor noise, occlusions, or dynamic objects in the scene. This leads to increased robustness in stereo VO. While least squares optimization adds computational complexity, its benefits in accuracy and robustness outweigh the cost. It ensures that stereo VO maintains a high level of accuracy even in challenging situations.

Optimizing Visual Odometry Algorithm Parameters for Real-Time Performance and Robustness in Changing Environments

Fine-tuning parameters used in stereo vision and visual odometry to control the behaviour of the StereoSGBM algorithm, keypoint detection (FAST), and optical flow tracking (Lucas-Kanade) can significantly impact the accuracy and performance of stereo visual odometry**.**

Stereo Vision Parameters namely,

minDisparity is the minimum possible disparity value. It represents the minimum horizontal shift for matching pixels between the left and right images. A smaller value can lead to better depth estimation if your camera setup has a baseline close to zero.

numDisparities is the disparity search range, which defines the maximum horizontal shift for matching pixels between the left and right images. A larger value could potentially capture objects at greater distances but may require more computational resources.

blockSize is the size of the disparity-matching window or block. Larger block sizes can improve robustness but may reduce accuracy in the presence of depth discontinuities. Experimentation can be done with larger or smaller window sizes to optimize the trade-off between accuracy and computational efficiency.

P1 (Penalty1) and P2 (Penalty2) are penalties for disparity change by plus or minus 1 between adjacent pixels within the same block. Suitable value of ‘k’ discourages discontinues or larger disparities.

Formula: $P_n = k_n * blockSize * blockSize$

Equation 8.1 Penalty calculation

Computer Vision Parameters like,

The FAST (Features from Accelerated Segment Test) feature detector is used to detect keypoints in the image. Adjusting the threshold for feature detection (**fastFeatures.setThreshold()**) allows to detect more or fewer features based on your specific requirements. Lower thresholds will detect more features, but it can also lead to false positives.

lk_params (Lucas-Kanade parameters) for the Lucas-Kanade optical flow algorithm, which is used for tracking keypoints between frames. These parameters can affect how well features are tracked between frames. ‘winSize’, size of the window for optical flow tracking. ‘flags’, A flag specifying the algorithm options. ‘maxLevel’, number of levels in the image pyramid. ‘criteria’, termination criteria for the iterative process.

RealSense Camera Parameters:

Emitter Settings segment controls the emitter settings of the RealSense camera, affecting the emission of infrared light.

depth_sensor.set_option(rs.option.emitter_enabled, set_emitter): Enables or disables the emitter, where set_emitter is set to 0 to turn off the emitter.

sensor.set_option(rs.option.enable_auto_exposure, True): Enables automatic exposure control.

depth_sensor.set_option(rs.option.exposure, exposure_time): Sets the exposure time to 100 microseconds.

Histogram Equalization applies histogram equalization to the captured infrared frames to enhance contrast and image quality. cv2.equalizeHist(new_frame_left) and cv2.equalizeHist(new_frame_right) Apply histogram equalization to the left and right infrared frames.

Frame Capture Rate determines the capture rate by setting a timeout for waiting for frames. frames = self.pipeline.wait_for_frames(30000): Waits for frames with a timeout of 30,000 microseconds (30 milliseconds).

9 Bibliography

al., J. S. (1994). Good features to track. In Computer Vision and Pattern Recognition 1994. *1994 IEEE Computer Society Conference*, (pp. 593-600).

Andreas Geiger, P. L. (2012). *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite.* Conference on Computer Vision and Pattern Recognition (CVPR).

Bay, H. T. (2008). SURF: Speeded up robust features. *Computer Vision and Image Understanding, 110*(3), 346-359.

Bebis, D. (2017). Image Formation. *CS791E Notes*. University of Nevada,Reno, Nevada, United States of America. Retrieved from https://www.cse.unr.edu/~bebis/CS791E/Notes/ImageFormation.pdf

Burger, W. (2016). *Zhang's camera calibration algorithm: in-depth tutorial and implementation.* HGB.

C. Forster, M. P. (2014). Visual odometry for small handheld devices. *2014 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 4697-4704). Hong Kong, China: IEEE.

Corke, P. (2017). *Robotics, Vision and Control.* Springer Cham.

Cremers, D. A. (2012). Time-of-Flight Cameras: Principles and Applications. *IEEE Signal Processing Magazine, 29*(6), 93-104.

Cremers, D. G. (2011). Optical flow for camera pose estimation: A tutorial. . *Foundations and Trends® in Computer Graphics and Vision, 6*(1), 1-100.

cv::StereoSGBM Class Reference. (n.d.). Retrieved from https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html.

D. Gitardi, S. S. (2022). UMA Universal Maintenance Automata – an adaptable robotic platform designed to run maintenance operations in harsh environment. *55th CIRP Conference on Manufacturing Systems* (pp. 1473-1478). ELSEVIER B.V.

Duan C, J. S. (2023). StereoVO: Learning Stereo Visual Odometry Approach Based on Optical Flow and Depth Information. *Applied Sciences*, 13(10):5842.

E. Rosten, R. P. (2010). Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 105-119.

Engel, J. e. (2012). Direct visual odometry with scale. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 34*(4), 721-733.

Ethan Rublee, V. R. (2011). Orb: An efficient alternative to sift or surf. *Computer Vision (ICCV) 2011 IEEE international conference*, (pp. 2564-2571).

F. Li, T. Z. (2020). Robust visual-inertial odometry for autonomous driving in urban environments. *IEEE Robotics and Automation Letters, 5*(1), 470-477.

Forsyth, D. A. (2011). *Computer Vision: A Modern Approach- 3rd Edition.* Prentice Hall.

G. Carlone, F. D. (2018). Robust visual-inertial odometry under dynamic conditions. *IEEE Transactions on Robotics, 34*(1), 140-155.

Geiger, A. L. (2012). A tutorial on visual odometry. *Foundations and Trends in Computer Graphics and Vision, 7*(1), 1-86.

Geiger, A. S. (2012). Visual Odometry: The Problem, The Algorithm, and the Application. *Springer*.

Glocker, A. e. (2014). Visual odometry in challenging environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 36*(8), 1529-1545.

Hartley, R. a. (2004). *Multiple View Geometry in Computer Vision - 2nd Edition.* Cambridge University Press.

Herbert Bay, T. T. (2006). Surf: Speeded up robust features. *European conference on computer vision*, (pp. 404-417).

hexagon.com. (n.d.). Retrieved from hexagon.com: https://hexagon.com/products/leica-absolute-tracker-at960

Hirschmuller, H. (2005). Accurate and efficient stereo processing by semi-global matching and mutual information. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (pp. 807-814). San Diego, CA, USA: IEEE .

Howard, A. (2008). Real-time stereo visual odometry for autonomous ground vehicles. *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Intel® RealSense™ Depth Camera D455. (n.d.). Retrieved from https://www.intelrealsense.com/depth-camera-d455/

J. Engel, T. S. (2014). Direct visual odometry with scale estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 36*(8), 1628-1642.

J. Sturm, N. E. (2012). A benchmark for the evaluation of RGB-D SLAM systems. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 573-580). Vilamoura-Algarve, Portugal: IEEE.

James S. Albus, R. V. (2006). THE LAGR PROJECT - Integrating learning into the 4D/RCS Control Hierarchy. *International Conference on Informatics in Control, Automation and Robotics | 3rd | | Set bal Polytechnic Institute.* Setubal, 1, PO.

Krombach, N. &. (2018). Feature-based visual odometry prior for real-time semi-dense stereo SLAM. *Robotics and Autonomous Systems, 109*.

Li, X. e. (2019). Visual-inertial odometry with deep feature integration. *arXiv preprint arXiv:1904.03441 (2019).*

Lijun Wei, C. C. (2011). GPS and Stereovision-Based Visual Odometry. *International Journal of Vehicular Technology*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 91-110.

LS, O. (2022). *Image Formation - Selected Literature References.* Olympus LifeScience. Retrieved from https://www.olympus-lifescience.com/en/microscope-resource/primer/anatomy/imageformationreferences/

Mandal, D. K. (2019). Visual Inertial Odometry At the Edge: A Hardware-Software Co-design Approach for Ultra-low Latency and Power. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (pp. 960-963). Florence.

MARS Exploration Rovers. (n.d.). Retrieved from https://mars.nasa.gov/mer/

MathWorks Switzerland. (n.d.). Retrieved from https://ch.mathworks.com/help/visionhdl/ug/stereoscopic-disparity.html

Mikolajczyk, K. &. (2005). A performance evaluation of local descriptors. *International Journal of Computer Vision, 60*(1), 60-72.

Mohamad Motasem Nawaf, D. M.-P.-M. (2018). Fast Visual Odometry for a Low-Cost Underwater Embedded Stereo System. *Sensors (Basel)*.

Mur-Artal, R. e. (2020). ORB-SLAM3: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Transactions on Robotics, 36*(4), 1080-1096.

Newcombe, R. A. (2011). DTAM: Dense tracking and mapping in real-time. *Proceedings of the IEEE International Conference on Computer Vision (ICCV).*

Perspective Projection - Wikipedia. (2022). Retrieved from Wikipedia, The Free Encyclopedia.

Pinhole Camera Model . (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Pinhole_camera

Pollefeys, M. V. (2008). *Stereo Visual Odometry.* Springer.

Pollefeys, M. V. (2011). Semi-direct visual odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 33*(5), 998-1011.

R. Arun, G. K. (2005). Parallel tracking and mapping for small visual-inertial systems. *IEEE Transactions on Robotics and Automation, 21*(5), 799-812.

R. Mur-Artal, J. M. (2017). Scale-aware visual-inertial odometry. *IEEE Transactions on Robotics, 33*(6), 1487-1505.

Rublee, E. R. (2011). ORB: an efficient alternative to SIFT. *IEEE International Conference on Computer Vision (ICCV)*.

Scaramuzza, D. (2011). Visual Odometry: Theory, Algorithms, and Applications. *Springer*.

Scaramuzza, D. (2017). Lecture: Vision algorithms for mobile.

Scaramuzza, D. a. (2009). Visual odometry for challenging environments. *IEEE Transactions on Robotics, 25*(1), 226-240.

Schönberger, J. L. (2014). Monocular visual odometry for large-scale environments. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014* (pp. 801-808). Washington, D.C: IEEE.

Shenbagaraj Kannapiran, N. B.-Y. (2023). Stereo Visual Odometry with Deep Learning-Based Point and Line Feature Matching using an Attention Graph Neural Network. arXiv:2308.01125.

Sigmoidal. (2022). Fundamentals of Image Formation in Computer Vision. Retrieved from https://sigmoidal.ai/en/fundamentals-of-image-formation/

Stephens, C. H. (1988). A combined corner and edge detector. *Alvey vision conference*, (pp. 23.1-23.6).

Stevenius, H. (2014). Fundamentals of Visual Odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 36*(7), 1432-1446.

Stewenius, H. a. (2011). *Feature-based Visual Odometry.* Springer.

Strutz, T. (2016). Data Fitting and Uncertainty 2nd edition. Springer Vieweg .

Szeliski, R. (2010). *Computer vision: algorithms and applications.* Springer Science & Business Media.

T. Qin, P. L. (2018). VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, 1-17.

T. Yu, J. Z. (2021). Accurate and Robust Stereo Direct Visual Odometry for Agricultural Environment. *IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 2480-2486). Xi'an, China.

Theobalt, C. A. (2016). Monocular depth estimation: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 38*(2), 409-425.

Wang, C. Z. (2023). Human Visual Attention Mechanism-Inspired Point-and-Line Stereo Visual Odometry for Environments with Uneven Distributed Features. *Chin. J. Mech. Eng.*

Wang, W. e. (2021). Deep stereo visual odometry: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 43*(1), 124-144.

wikipedia.org. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Random_sample_consensus

Y. Cheng, M. W. (2006). Visual odometry on the mars exploration rovers - a tool to ensure accurate driving and science. *Ieee Robotics and Automation Magazine*, 54-62.

Yang, S. e. (2018). DeepVO: Deep visual odometry for challenging environments. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

zed-mini. (n.d.). Retrieved from www.stereolabs.com: https://www.stereolabs.com/zed-mini/

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(11), 1330-1334.

Zhe Liu, D. S. (2023). T-ESVO: Improved Event-Based Stereo Visual Odometry via Adaptive Time-Surface and Truncated Signed Distance Function. *Adv. Intell. Syst. 2300027.*, https://doi.org/10.1002/aisy.202300027.

10 Appendices

10.1 Mathematical Formulations

10.2 Code Snippets

Here important code snippets mentioned for implementation of Stereo Visual Odometry are illustrated, subsequent complete codes and files can be accessed through git repository.

https://github.com/UtkarshSavkare/StereoVO.git

10.2.1 Corner detection for Stereo calibration

```
################ FIND CHESSBOARD CORNERS - OBJECT POINTS AND IMAGE POINTS ############################

chessboardSize = (9,7)
frameSize = (848,480)


# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)


# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)qq
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)

size_of_chessboard_squares_mm = 20
objp = objp * size_of_chessboard_squares_mm

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpointsL = [] # 2d points in image plane.
imgpointsR = [] # 2d points in image plane.


imagesLeft = sorted(glob.glob('images/stereoLeft/*.png'))
imagesRight = sorted(glob.glob('images/stereoRight/*.png'))

for imgLeft, imgRight in zip(imagesLeft, imagesRight):

    imgL = cv.imread(imgLeft)
    imgR = cv.imread(imgRight)
    grayL = cv.cvtColor(imgL, cv.COLOR_BGR2GRAY)
    grayR = cv.cvtColor(imgR, cv.COLOR_BGR2GRAY)

    # Find the chess board corners
    retL, cornersL = cv.findChessboardCorners(grayL, chessboardSize, None)
    retR, cornersR = cv.findChessboardCorners(grayR, chessboardSize, None)

    # If found, add object points, image points (after refining them)
    if retL and retR == True:

        objpoints.append(objp)

        cornersL = cv.cornerSubPix(grayL, cornersL, (11,11), (-1,-1), criteria)
        imgpointsL.append(cornersL)

        cornersR = cv.cornerSubPix(grayR, cornersR, (11,11), (-1,-1), criteria)
        imgpointsR.append(cornersR)
```

10.2.2 Stereo calibration and rectification

```
############## CALIBRATION ########################################################

retL, cameraMatrixL, distL, rvecsL, tvecsL = cv.calibrateCamera(objpoints, imgpointsL, frameSize, None,
None)
heightL, widthL, channelsL = imgL.shape
newCameraMatrixL, roi_L = cv.getOptimalNewCameraMatrix(cameraMatrixL, distL, (widthL, heightL), 1,
(widthL, heightL))

retR, cameraMatrixR, distR, rvecsR, tvecsR = cv.calibrateCamera(objpoints, imgpointsR, frameSize, None,
None)
heightR, widthR, channelsR = imgR.shape
newCameraMatrixR, roi_R = cv.getOptimalNewCameraMatrix(cameraMatrixR, distR, (widthR, heightR), 1,
(widthR, heightR))


########## Stereo Vision Calibration #############################################

flags = 0
flags |= cv.CALIB_FIX_INTRINSIC
# Here we fix the intrinsic camara matrixes so that only Rot, Trns, Emat and Fmat are calculated.
# Hence intrinsic parameters are the same

criteria_stereo= (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# This step is performed to transformation between the two cameras and calculate Essential and
Fundamenatl matrix
retStereo, newCameraMatrixL, distL, newCameraMatrixR, distR, rot, trans, essentialMatrix,
fundamentalMatrix = cv.stereoCalibrate(objpoints, imgpointsL, imgpointsR, newCameraMatrixL, distL,
newCameraMatrixR, distR, grayL.shape[::-1], criteria_stereo, flags)


########## Stereo Rectification ################################################

rectifyScale= 1
rectL, rectR, projMatrixL, projMatrixR, Q, roi_L, roi_R= cv.stereoRectify(newCameraMatrixL, distL,
newCameraMatrixR, distR, grayL.shape[::-1], rot, trans, rectifyScale,(0,0))

stereoMapL = cv.initUndistortRectifyMap(newCameraMatrixL, distL, rectL, projMatrixL, grayL.shape[::-1],
cv.CV_16SC2)
stereoMapR = cv.initUndistortRectifyMap(newCameraMatrixR, distR, rectR, projMatrixR, grayR.shape[::-1],
cv.CV_16SC2)
```

10.2.3 Stereo VO on KITTI dataset (Parameters)

```python
    @staticmethod
    def _load_calib(filepath):
        """
        Loads the calibration of the camera
        Parameters
        ----------
        filepath (str): The file path to the camera file

        Returns
        -------
        K_l (ndarray): Intrinsic parameters for left camera. Shape (3,3)
        P_l (ndarray): Projection matrix for left camera. Shape (3,4)
        K_r (ndarray): Intrinsic parameters for right camera. Shape (3,3)
        P_r (ndarray): Projection matrix for right camera. Shape (3,4)
        """
        with open(filepath, 'r') as f:
            params = np.fromstring(f.readline(), dtype=np.float64, sep=' ')
            P_l = np.reshape(params, (3, 4))
            K_l = P_l[0:3, 0:3]
            params = np.fromstring(f.readline(), dtype=np.float64, sep=' ')
            P_r = np.reshape(params, (3, 4))
            K_r = P_r[0:3, 0:3]
        return K_l, P_l, K_r, P_r

    @staticmethod
    def _load_poses(filepath):
        """
        Loads the GT poses

        Parameters
        ----------
        filepath (str): The file path to the poses file

        Returns
        -------
        poses (ndarray): The GT poses. Shape (n, 4, 4)
        """
        poses = []
        with open(filepath, 'r') as f:
            for line in f.readlines():
                T = np.fromstring(line, dtype=np.float64, sep=' ')
                T = T.reshape(3, 4)
                T = np.vstack((T, [0, 0, 0, 1]))
                poses.append(T)
        return poses

    @staticmethod
    def _load_images(filepath):
        """
        Loads the images

        Parameters
        ----------
        filepath (str): The file path to image dir

        Returns
        -------
        images (list): grayscale images. Shape (n, height, width)
        """
        image_paths = [os.path.join(filepath, file) for file in sorted(os.listdir(filepath))]
        images = [cv2.imread(path, cv2.IMREAD_GRAYSCALE) for path in image_paths]
        return images
```

10.2.4 Stereo VO on KITTI dataset (main)

```python
def main():
    data_dir = 'KITTI_sequence_01'  # KITTI_sequence
    vo = VisualOdometry(data_dir)

    play_trip(vo.images_l, vo.images_r)  #  play the image sequence trip

    gt_path = []
    estimated_path = []
    for i, gt_pose in enumerate(tqdm(vo.gt_poses, unit="poses")):
        if i < 1:
            cur_pose = gt_pose
        else:
            transf = vo.get_pose(i)
            cur_pose = np.matmul(cur_pose, transf)
        gt_path.append((gt_pose[0, 3], gt_pose[2, 3]))
        estimated_path.append((cur_pose[0, 3], cur_pose[2, 3]))
    plotting.visualize_paths(gt_path, estimated_path, "Stereo Visual Odometry",
                             file_out=os.path.basename(data_dir) + ".html")


if __name__ == "__main__":
    main()
```

10.2.5 Real sense image acquisition pipeline (sensor settings)

```python
# Start the pipeline
    pipeline_profile = self.pipeline.start(config)
    device = pipeline_profile.get_device()
    depth_sensor = device.query_sensors()[0]
    emitter = depth_sensor.get_option(rs.option.emitter_enabled)
    #print("emitter = ", emitter)
    set_emitter = 0
    depth_sensor.set_option(rs.option.emitter_enabled, set_emitter)
    emitter1 = depth_sensor.get_option(rs.option.emitter_enabled)
    depth_sensor = pipeline_profile.get_device().first_depth_sensor()
    for sensor in pipeline_profile.get_device().sensors:
            if sensor.supports(rs.option.enable_auto_exposure):
                sensor.set_option(rs.option.enable_auto_exposure, True)
    #Set the desired exposure time in microseconds (example: 20000 microseconds)
    exposure_time = 100
    if depth_sensor.supports(rs.option.exposure):
            depth_sensor.set_option(rs.option.exposure, exposure_time)
    #print("new emitter = ", emitter1)
```

10.2.6 Stereo VO Realtime testing (Calibration parameters)

```
    @staticmethod
    def _load_calib(filepath):
        """
        Loads the calibration of the camera
        Parameters
        ----------
        filepath (str): The file path to the camera file

        Returns
        -------
        K_l (ndarray): Intrinsic parameters for left camera. Shape (3,3)
        P_l (ndarray): Projection matrix for left camera. Shape (3,4)
        K_r (ndarray): Intrinsic parameters for right camera. Shape (3,3)
        P_r (ndarray): Projection matrix for right camera. Shape (3,4)
        """
        with open(filepath, 'r') as f:
            line = f.readline().strip()
            params = np.fromstring(line, dtype=np.float32, sep=' ')
            P_l = np.reshape(params, (3, 4))
            K_l = P_l[0:3, 0:3]
            P_l[0, 3], P_l[1, 3] = params[3], params[7]

            line = f.readline().strip()
            params = np.fromstring(line, dtype=np.float32, sep=' ')
            P_r = np.reshape(params, (3, 4))
            K_r = P_r[0:3, 0:3]
            P_r[0, 3], P_r[1, 3] = params[3], params[7]

        return K_l, P_l ,K_r, P_r
```

10.2.7 Stereo VO Realtime testing (main)

```
rs = RealsenseCamera()
while True:

   # Capture frame-by-frame
    ret, new_frame_left,new_frame_right,*_ = rs.get_frame_stream()


    frame_counter += 1

    start = time.perf_counter()

    if process_frames and ret:
        try:

            transf = vo.get_pose(old_frame_left, old_frame_right, new_frame_left, new_frame_right)

            cur_pose = cur_pose @ transf

            # hom_array = np.array([[0,0,0,1]])
            # hom_camera_pose = np.concatenate((cur_pose, hom_array), axis=0)
            camera_pose_list.append(cur_pose)
            estimated_path.append((cur_pose [0, 3], cur_pose[1, 3]))

            estimated_camera_pose_x, estimated_camera_pose_y = cur_pose [0, 3], cur_pose [1, 3]

        except Exception as e:
            print("An error occurred:", str(e))
            break

    elif process_frames and ret is False:
        break

    old_frame_left = new_frame_left
    old_frame_right = new_frame_right

    process_frames = True

    end = time.perf_counter()
    total_time = end - start
    fps = 1 / total_time
```