Visual Odometry

Charith Reddy Gannapu Reddy
University of Maryland
College Park, Maryland
charith@umd.edu

Koushik Alapati
University of Maryland
College Park, Marland
akoushik@umd.edu

Abstract

This report presents the implementation of a visual odometry algorithm designed to estimate the position of objects by analyzing frame-to-frame motion captured by successive pairs of stereo images. A key feature of this implementation is that it operates without any prior assumptions about camera motion and utilizes dense disparity images from stereo cameras. The algorithm is initially tested using the KITTI dataset and subsequently applied to data from a stereo camera in a laboratory setting. This approach demonstrates the algorithm's effectiveness in real-world scenarios.

1. Introduction

Accurately determining the position of a robot with respect to the world frame is crucial for any autonomous vehicle. Traditionally, this task has been accomplished using wheel odometry and inertial sensors. Wheel odometry measures the wheel rotations to estimate the vehicle's movement, while inertial sensors, such as accelerometers and gyroscopes, measure the acceleration and angular velocity. However, these methods have several significant limitations. Inertial sensors are prone to drift over time, leading to cumulative errors in position estimates. On the other hand, wheel odometry can be unreliable on rough terrains where wheels may slip or encounter obstacles, causing inaccuracies in the measurements.

To address these challenges, visual odometry has emerged as a promising solution. Visual odometry estimates the position and motion of the vehicle by analyzing visual information captured by cameras. This method provides all six degrees of freedom (6DOF) of motion, which include translations along the x, y, and z axes, as well as rotations around these axes. Additionally, visual odometry has lower drift compared to inertial measurement units (IMUs), making it a more reliable method for long-term navigation.

This report describes a technique for real-time stereo visual odometry, which is especially well-suited for ground vehicle applications. We opted for a stereo method instead of a monocular one because many ground vehicles already have stereo cameras installed for terrain sensing and obstacle recognition. Stereo cameras provide two slightly different views of the same scene, allowing for depth perception through triangulation. By leveraging the stereo range data these systems create, we can achieve visual odometry that is typically faster and more accurate compared to monocular methods. The combination of stereo ranging and stereo visual odometry enhances the robustness and precision of the position estimates.

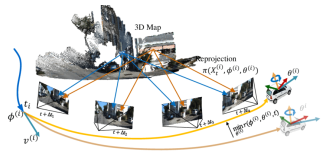

Figure 1. Visual Odometry Example

The KITTI dataset, created through a collaboration between the Toyota Technological Institute in Chicago and the Karlsruhe Institute of Technology, was used to test our algorithm. The KITTI dataset is a comprehensive collection of images and LIDAR data designed for autonomous driving research. It includes data for various computer vision tasks such as stereo vision, optical flow, visual odometry, 3D object detection, and 3D tracking. The dataset's high-quality, annotated data makes it an excellent benchmark for evaluating visual odometry algorithms.

In this report, we implement a visual odometry algorithm and test it using the KITTI dataset. Our approach involves processing successive pairs of stereo images to estimate the vehicle's motion frame-by-frame. We also apply the algorithm to data from a stereo camera in a laboratory setting to demonstrate its practical effectiveness in real-world scenarios. The results showcase the algorithm's capability to accurately estimate the vehicle's trajectory, highlighting its potential for improving autonomous navigation systems.

2. Methodology

2.1. Using KITTI Dataset

In our process, we use already rectified images from the KITTI dataset to determine the rotation and translation matrices between consecutive images captured by the camera. We begin by loading the rectified stereo images and sorting them into left and right pairs. Next, we read the calibration file to obtain the intrinsic and extrinsic parameters of the stereo cameras. For each left image, we detect keypoints and compute descriptors using the SIFT algorithm. We then compute the disparity map between the left and right images using the StereoSGBM algorithm.

From this disparity map, we calculate the depth map, which provides the distance of each point in the image from the camera. We match features between consecutive left images using the BFMatcher with a ratio test, identifying corresponding points between the images. Using the matched features and depth information, we estimate the relative pose (rotation and translation) between consecutive frames with the solvePnPRansac algorithm. This gives us the rotation and translation matrices that describe the camera's movement. We update the homogeneous transformation matrix with these new matrices, keeping track of the camera's overall position and orientation in the world frame.

Finally, we visualize the results by plotting the disparity map, the current left image, and the depth map, as well as the camera's estimated trajectory compared to the ground truth trajectory. This methodology allows us to accurately estimate the position of the robot with respect to the world, overcoming the limitations of traditional methods like wheel odometry and inertial sensing.

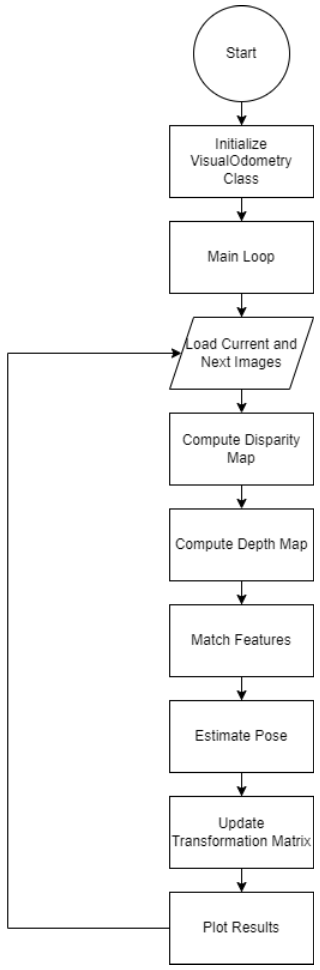

Figure 2. Algorithm Used for Visual Odometry

2.2. Hardware Implementation

To create our own dataset using the knowledge gained from the KITTI dataset, we used a Zed camera to capture the frames. The algorithm employed is the same as the one used for the KITTI dataset. A major task in this process is rectifying the images, as the KITTI dataset images are already rectified. To rectify the images, we first calibrated the camera using a checkerboard pattern. This calibration process involves capturing multiple images of the checkerboard at different angles and positions. The checkerboard provides known reference points which allow us to determine the intrinsic and extrinsic parameters of the camera.

After calibration, we use these parameters to rectify the images. Rectification involves transforming the images so that corresponding points in the stereo pair lie on the same horizontal line. This is essential for accurate depth estimation and subsequent visual odometry calculations.

In detail, we started with camera calibration. We captured multiple images of a checkerboard pattern from different angles and distances. For each image, we detected the corners of the checkerboard, which provided the necessary reference points. Using these detected corners, we computed the camera's intrinsic and extrinsic parameters, which describe the camera's optical characteristics and its position in space relative to the checkerboard.

Next, we moved on to image rectification. We used the features detected in both images to compute the fundamental matrix, which relates the corresponding points in the stereo images. From the fundamental matrix and the intrinsic camera parameters, we computed the essential matrix, encoding the relative rotation and translation between the cameras. Using the fundamental matrix, we computed the homography for each image, which allowed us to warp the images so they are rectified. We applied the computed homography to the images and checked the quality of rectification by drawing epilines on the images. The epilines should be horizontal and align with corresponding points in the stereo pair if rectification is done correctly.

Following rectification, we detected features in both rectified images using the SIFT algorithm. We matched the detected features between the two images using the BFMatcher with a ratio test to find good pairs.

From this point, the implementation is the same as the one used for the KITTI dataset. This involves computing disparity maps to get depth information, matching features between consecutive frames, estimating the relative pose using the solvePnPRansac algorithm, updating the homogeneous transformation matrix, and plotting the results. This approach ensures that the images are accurately rectified and suitable for visual odometry, similar to the KITTI dataset.

2.3. Data Preparation

The dataset is generated using a Zed camera, capturing 985 frames in sequence, following the path of motion and including turns. Before processing this data with the algorithm, several pre-processing steps are undertaken. The steps within the algorithm are outlined below.

2.3.1 Feature Extraction

To find the fundamental matrix, we need to extract keypoints from adjacent frames. These keypoints are then used to compute the fundamental matrix. For feature extraction, we use the SIFT (Scale-Invariant Feature Transform) detector from the OpenCV library. SIFT detects distinctive keypoints in each image that are invariant to scale and rotation, making them robust for matching.

After detecting the keypoints, we use BFMatcher (Brute Force Matcher) to match the features between the images. BFMatcher compares descriptors of the keypoints and identifies the best matches. We apply a ratio test to filter out poor matches and retain only the good pairs. These good pairs are crucial for accurately computing the fundamental matrix, which is used in subsequent steps for image rectification and pose estimation.

2.3.2 Computing Fundamental Matrix

The fundamental matrix describes the relationship between corresponding points in stereo images. It is a 3x3 matrix with 9 components and can be defined by the following equation, where x_1 and x_2 are the coordinates of a feature point in two images:

$$x_1^T F x_2 = 0 \tag{1}$$

This equation is derived from the epipolar geometry constraint. The fundamental matrix, denoted by F, has 9 components and represents 9 unknowns in a single equation. The expanded form of the equation is:

$$\begin{aligned} x_1 x_2 f_{11} + y_1 x_2 f_{21} + x_2 f_{31} + x_1 y_2 f_{12} \\ + y_1 y_2 f_{22} + y_2 f_{32} + x_1 f_{13} + y_1 f_{23} + f_{33} = 0 \end{aligned} \tag{2}$$

To solve for F, we use the OpenCV functions, which implement robust methods for estimating the fundamental matrix from matched keypoints. These methods typically involve using RANSAC (Random Sample Consensus) to handle outliers and ensure a reliable estimation of the fundamental matrix. By applying these functions, we accurately compute the fundamental matrix, which is essential for tasks such as image rectification and pose estimation.

2.3.3 Essential Matrix Calculation

The essential matrix is calculated using the parameters obtained from camera calibration and the fundamental matrix. The essential matrix is given by the equation:

$$E = K^T F K \tag{3}$$

2.3.4 Decomposition Essential Matrix

Using the OpenCV function recoverPose, we decompose the essential matrix to obtain the rotation and translation between the two frames. By extracting the z-component and combining it with the data obtained from the disparity images, we can accurately determine the pose in 3D space. This step is crucial for estimating the vehicle's trajectory and ensuring precise localization.

2.3.5 Image Rectification

Image rectification is a crucial step in stereo vision, transforming stereo images so that corresponding points align horizontally. This facilitates accurate depth estimation and visual odometry.

To begin, the stereo images are converted to grayscale. Using the keypoints extracted from both images and the fundamental matrix, we compute the homography matrices H_1 and H_2 using the cv2.stereoRectifyUncalibrated function. These homography matrices transform the images to align corresponding points.

The transformation is applied to the images with cv2.warpPerspective, resulting in rectified images. The keypoints are also transformed using cv2.perspectiveTransform to align with the rectified images.

To ensure the rectification is accurate, epilines are computed and drawn on the rectified images. The cv2.computeCorrespondEpilines function calculates the epilines for corresponding points, which should be horizontal and aligned across both images if rectification is correct.

Finally, the rectified images are visualized to verify the alignment, ensuring that they are suitable for subsequent steps in the visual odometry process. This step is essential for accurate depth estimation and reliable pose estimation.

2.3.6 Disparity and Depth Calculation

Disparity calculation is a crucial step in visual odometry, enabling depth determination by comparing left and right images captured by the stereo camera. For each frame pair, the images are loaded in grayscale format using OpenCV. The disparity map is computed using the StereoSGBM algorithm, which compares blocks of pixels between the images to find corresponding points and calculate disparity, the pixel coordinate difference.

Once the disparity map is obtained, the depth map is derived using the intrinsic parameters of the left camera and the baseline distance between the stereo cameras. The baseline is the distance between the two cameras, and the focal length is obtained from the intrinsic matrix. Disparity values are used to compute the depth for each pixel, with adjustments for zero or negative disparity to avoid errors.

These steps provide essential depth information about the scene, crucial for subsequent visual odometry stages, including pose and trajectory estimation. This depth information enhances the accuracy and reliability of the visual odometry system, enabling precise 3D reconstruction and navigation.

2.3.7 Pose Estimation

Pose estimation is the final step in the visual odometry process, where the relative rotation and translation between consecutive frames are determined to estimate the camera's motion. It begins with keypoint matching using the SIFT algorithm on the current and next left images, followed by matching features with BFMatcher and a ratio test.

Using the disparity map and camera calibration parameters, 3D points are reconstructed from 2D keypoints by calculating the depth from disparity values and converting 2D image coordinates into 3D world coordinates. The solvePnPRansac algorithm then estimates the pose by solving the Perspective-n-Point (PnP) problem, providing robust and accurate rotation and translation vectors.

The rotation vector is converted to a rotation matrix using the Rodrigues function, and the transformation matrix is updated to reflect the new pose. This updated matrix is used to compute the camera's new position, added to the trajectory. This process is repeated for each frame pair, building a comprehensive trajectory of the camera's motion.

In summary, pose estimation involves matching keypoints, reconstructing 3D points, and using robust algorithms to estimate the relative pose between frames. This step updates the camera's transformation matrix and builds an accurate trajectory of its movement, ensuring reliable tracking of the camera's position and orientation over time.

3. Experiments

3.1. KITTI Dataset

As discussed in the methodology, we tested the algorithm on the KITTI dataset, specifically the KITTI Visual Odometry Evaluation 2012. The KITTI dataset consists of a total of 21 sequences. To evaluate our results, we need sequences with ground truth data. Therefore, we selected two sequences for our experiments: one with 4540 images and another with 800 images. The images are already in grayscale, eliminating the need for any image conversions to run them through the pipeline. This setup allows us to directly apply our algorithm and assess its performance using the provided ground truth data.

The camera calibration details are not given individually we are given a projection matrix that contains both the camera calibration matrix and translation.

Projection Matrix

$$\begin{bmatrix} 718.8560 & 0.0000 & 607.1928 & 0.0000 \\ 0.0000 & 718.8560 & 185.2157 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 \end{bmatrix}$$

P0

$$\begin{bmatrix} 718.8560 & 0.0000 & 607.1928 & 0.0000 \\ 0.0000 & 718.8560 & 185.2157 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 \end{bmatrix}$$

P1

The next step of the solution is extracting features and matching them with the adjacent frames the results of feature extraction are shown in Figure 3 and Figure 4

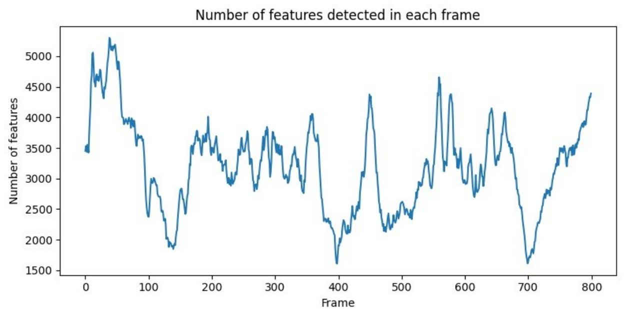

Figure 3. Number of Features for sequence of 800 images

The feature extraction is one step the next step is crucial as the count can decide how well the fundamental matrix can be calculated based on the number of matches and the number of good matches among them. The number of matches is shown in Figure 5 and Figure 6

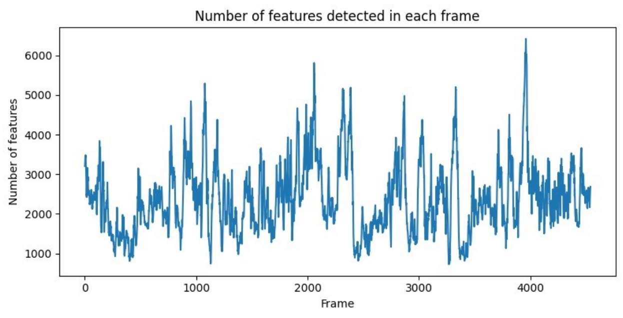

Figure 4. Number of Features for a sequence of 4540 images

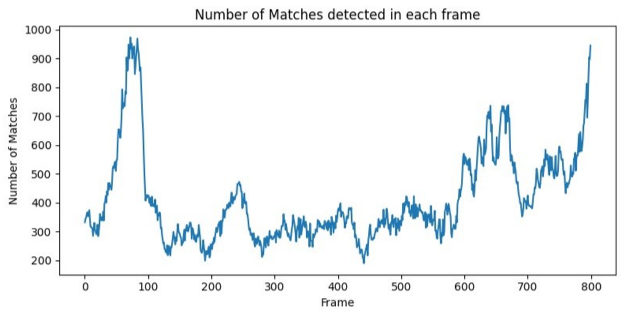

Figure 5. Feature Matched in the sequence of 800 images

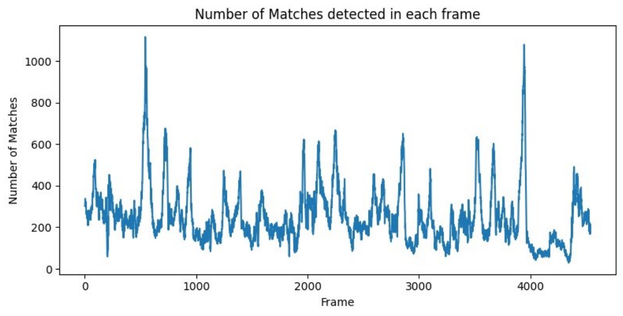

Figure 6. Feature Matched in the sequence of 4540 images

The depth of the images obtained from the disparity map is calculated at each iteration, and the disparity map is calculated according to the procedure mentioned in the methodology.

3.2. Hardware Implementation

To evaluate the algorithm with a dataset created by us with our camera in a region with low light conditions.

The experimentation is done in a region of $0.3 * 0.2m^2$ using a Zed Camera, the camera is first calibrated using a checkerboard, and the results of calibration are as follows:

Camera left
width = 960
height = 540

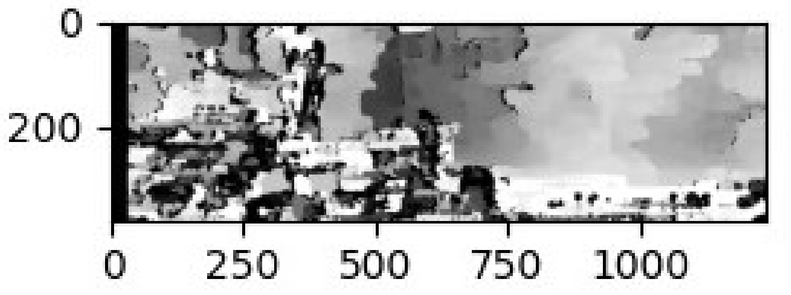

Figure 7. Disparity Map while running the 800 image sequence

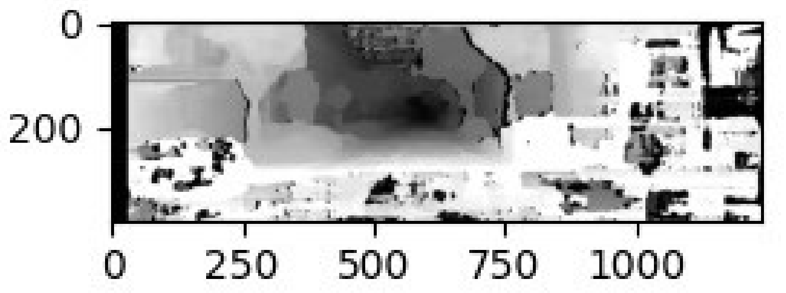

Figure 8. Disparity Map while running the 4540 image sequence

Figure 9. Zed Camera

$$\begin{bmatrix} 698.356373 & 0.000000 & 509.231709 \\ 0.000000 & 700.237830 & 280.344823 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Camera Right
width = 960
height = 540

$$\begin{bmatrix} 700.210649 & 0.000000 & 510.011996 \\ 0.000000 & 700.680856 & 286.941621 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Initially, we assumed that the data obtained from the Zed camera was already rectified. However, this was not the case, so we had to rectify the dataset ourselves. This involved the classical approach of computing features and the fundamental matrix to obtain a homography matrix. The homography matrix was then used to warp the perspective of the images, ensuring that the epipolar lines were horizontal.

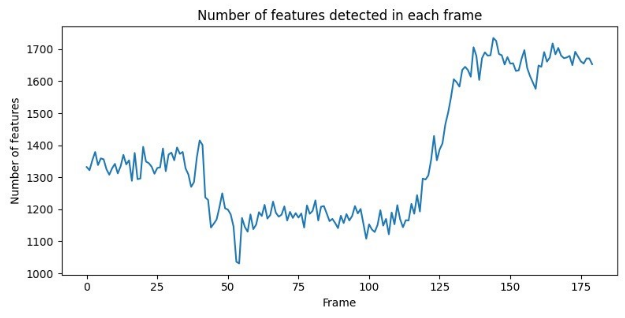

Figure 10. Features Extracted for our dataset

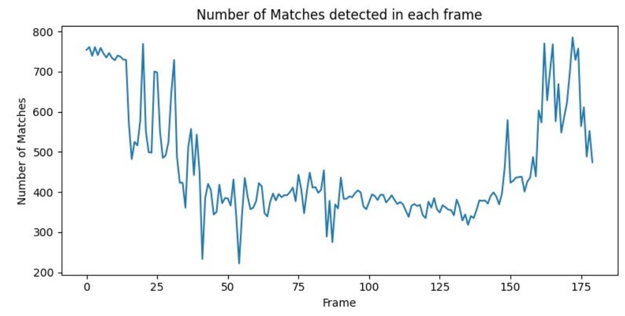

Figure 11. Feature Matching of our datatset

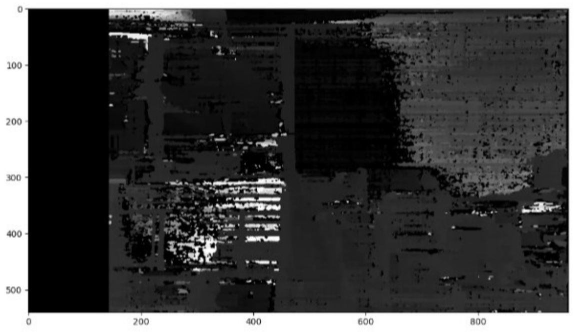

Figure 12. Disparity Map of the dataset

4. Results

After testing the algorithm for two sequences from KITTI dataset and implementing it on the Hardware, the results are quite interesting as we see that the error is visible as we are not dealing with the accumulated error as

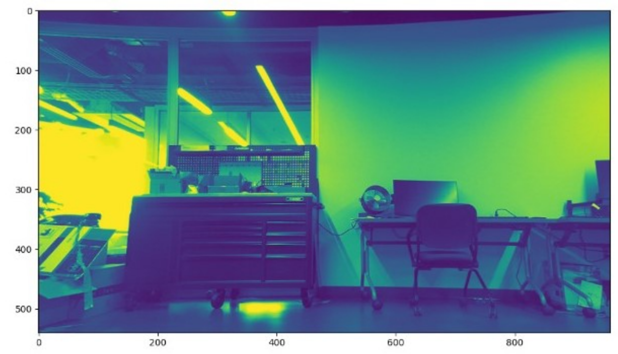

Figure 13. Depth Map of the dataset

time moves which resulted in error in pose increasing as we move forward. The error accumulation graph are shown in the Figure 10 the pose error is calculated as the distance between the ground truth and the result of Visual odometry position.

Since more matches offer greater restrictions for pose estimation, it has been discovered that the pose error tends to decrease as the number of feature matches grows. The accumulation of errors over time causes a divergence from the ground truth path, even with this initial reduction in error. The projected trajectory starts to diverge from the real path as the sequence goes on, and this drift gets more noticeable.

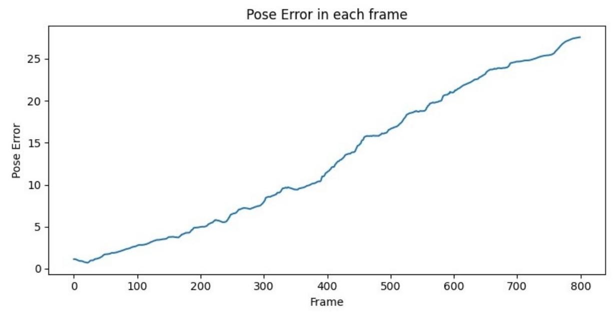

Figure 14. Pose Error with 800 sequence

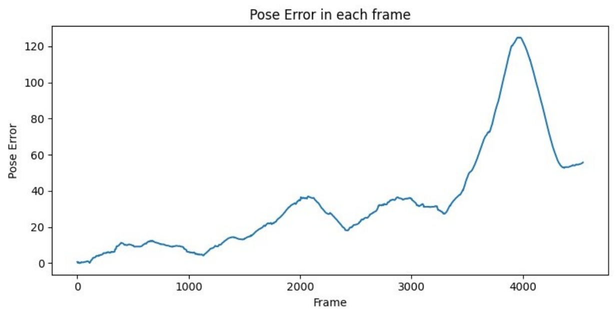

Figure 15. Pose Error with 4540 sequence

The ground truth is the actual position of the data, this data can be used to give a comparison of how much deviation a robot moves from the original path.

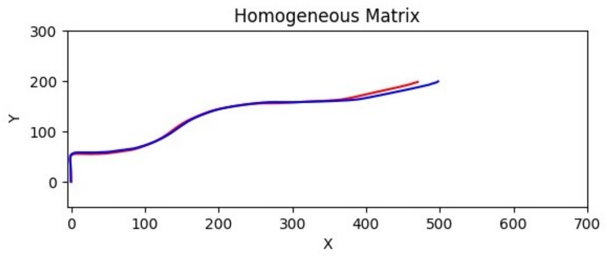

Figure 16. x vs y Graph with Ground truth 800 sequence

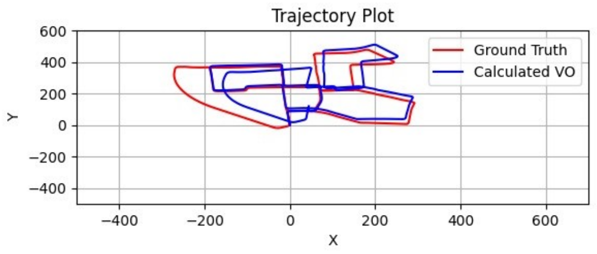

Figure 17. x vs y Graph with Ground truth 4540 sequence

While the hardware implementation was successful, we encountered issues during camera rotation. The number of matches between adjacent images decreased significantly when the camera was rotated, making the data around these rotations less useful. Initially, we suspected improper camera calibration as the cause. However, we later discovered that the problem stemmed from our data collection process. Specifically, the frames were not properly captured in certain areas due to rapid movement, leading to gaps and inconsistencies in the data.

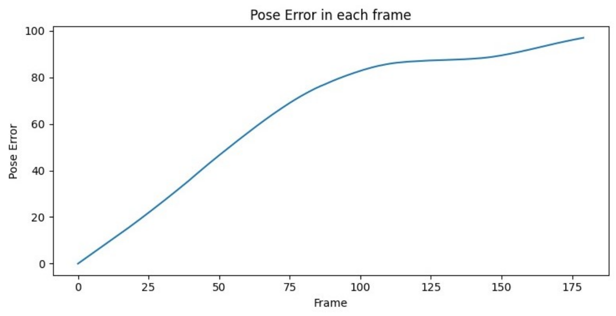

Figure 18. Pose Error from Hardware Implementation

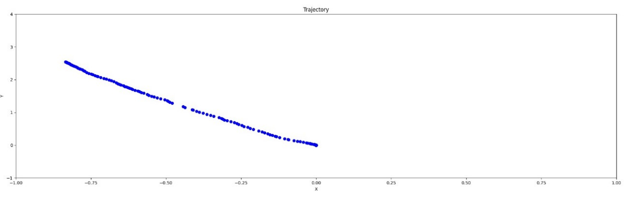

Figure 19. Trajectory of Hardware Implementation

5. Contributions

Both of us have worked equally on this project, sharing the workload and collaborating closely throughout the process. Instead of dividing tasks, we tackled each challenge ether, ensuring that we could support each other whenever one of us encountered difficulties. This approach allowed us to leverage our combined skills and knowledge effectively. By working simultaneously on different aspects of the project, we ensured that every component received thorough attention and benefited from our collective expertise. This collaborative effort was crucial in overcoming obstacles and achieving the project's goals.

References

[1] MathWorks, "Stereoscopic Rectification," https://www.mathworks.com/help/visionhdl/ug/stereoscopic-rectification.html.

[2] S. Seitz, "Stereo Rectification," Carnegie Mellon University, https://www.cs.cmu.edu/~16385/s17/Slides/13.1_Stereo_Rectification.pdf.

[3] A. Howard, "Real-Time Stereo Visual Odometry for Autonomous Ground Vehicles," NASA Jet Propulsion Laboratory, California Institute of Technology, https://www-robotics.jpl.nasa.gov/media/documents/howard_iros08_visodom.pdf.

[4] N. Merrill and G. Huang, "Lightweight Unsupervised Deep Loop Closure"

[5] OpenCV Documentation

[6] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. Bt. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," SpringerPlus, vol. 5, no. 1, pp. 1897, 2016, http://dx.doi.org/10.1186/s40064-016-3573-7.