Application of Epipolar Geometry and Singular Value Decomposition in Monocular Visual Odometry

Karol Yangqian Poetracahya, 13523093[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*13523093@std.stei.itb.ac.id* [1]*karolyangqian14@gmail.com*

***Abstract*—Odometry is an essential functionality in autonomous agents, such as cars, robots, and unmanned aerial vehicles, allowing them to accurately estimate their poses. This paper introduces an odometry technique based on epipolar geometry (EP) within a monocular camera system. The focus is on the implementation of Monocular Visual Odometry (MVO), which includes camera calibration, feature detection and matching, and the application of Singular Value Decomposition (SVD) for calculating translational and rotational transformations, as well as pose estimation. The MVO algorithm is evaluated using samples from the KITTI dataset and validated against ground truth poses.**

***Keywords*—Computer Vision, Epipolar Geometry, Monocular Visual Odometry, Singular Value Decomposition.**

I. Introduction

Visual odometry (VO) is a process of estimating the ego motion of an agent (e.g., vehicle, human, and robot) using the visual data gained from a single or multiple camera systems attached to the agent [5]. The term VO was first used in a paper by Nistér in 2004 [4]. The VO system operates by incrementally estimating the pose of the agent over time through observation of the changes that the motion induces to the image frame captured by the camera.

VO has been practically used in many computer vision and robotics applications and has several advantages compared to other navigation systems such as Global Positioning System (GPS), Light Detection and Ranging (Lidar), Inertial Measurement Unit (IMU), and wheel odometry. While GPS excels in outdoor and long-range navigation, it lacks accuracy and reliability in urban areas and GPS-denied environments. Lidar offers high resolution data but is cost inefficient and has a limited range. IMU provides information about angular velocity and acceleration of the agent, but it struggles with the accumulation of errors over time. Wheel odometry, while it is also commonly used, suffers from wheels slip error in uneven terrain or other unfavorable conditions [5]. VO is an interesting supplementary for these navigation systems as it is unaffected by wheel slip and can offer better accuracy in urban or GPS-denied areas, such as underwater, aerial, or indoors, as long as provided with sufficient illumination. A demonstration has shown that VO results in more accurate trajectory estimates compared to wheel odometry, with errors ranging from 0.1 to 2 percent [5]. A camera sensor is also relatively cost effective and can provide color, semantic, and geometric information for a more comprehensive understanding of the scene.

In this work, an MVO technique based on epipolar geometry is presented. The approach involves camera calibration, feature detection and matching, and the use of SVD to compute rotational and translational transformations. The method is evaluated using samples of a publicly available dataset fetched from The KITTI Vision Benchmark Suite. The remainder of this paper is structured as follows: Section II elaborates theoretical foundations used in this work. Section III outlines the methodology, detailing each step of the MVO algorithm and acquirement of data. Section IV presents the results and discusses the performance of the proposed approach. Finally, Section V concludes the paper and highlights potential directions for future research, followed by acknowledgements in Section VI.

II. Theoretical Foundations

A. Singular Value Decomposition

SVD is a technique to factorize a matrix into 3 distinct components. The SVD of a given matrix A is shown as follows:

$$A = U\Sigma V^T \tag{1}$$

where U and V are orthonormal matrices and Σ is a diagonal matrix with the non-zero diagonal entries as the singular values of A denoted as σ. If A has a size $m \times n$, then U, Σ, and V have sizes $m \times m$, $m \times n$, and $n \times n$ respectively. Suppose u_i and v_i is a column of U and V respectively in the form vectors. Given k the rank of A, then $u_1, u_2, \dots, u_k$ are the left singular vectors of A, $v_1, v_2, \dots, v_k$ are the right singular vectors of A, and $\sigma_1, \sigma_2, \dots, \sigma_k$ are the singular values of A.

$$A^T A = V\Lambda V^T \tag{2}$$

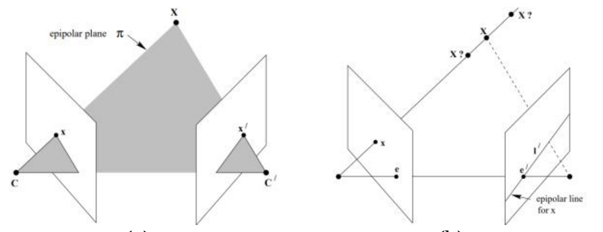
Equation (2) is a result of multiplying (1) on the left by A^T and from it can be inferred that V diagonalizes $A^T A$. The non-zero diagonal entries of Λ, denoted as λ_i, are the eigenvalues of A^TA which are positive and each of the corresponding σ_i is equivalent to $\sqrt{\lambda_i}$.

$$A = U\Sigma V^T = u_1\sigma_1 v_1^T + \cdots + u_k\sigma_k v_k^T \tag{3}$$

The SVD of A can also be rewritten as a sum of the product of u_i, σ_i, and v_i^T up to k as shown in (3).

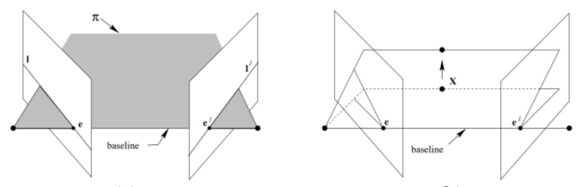
B. Epipolar Geometry

Epipolar geometry describes the geometric relationship between two views, focusing on the intersection of their image planes with a pencil of planes sharing the baseline (the line connecting the camera centers) as an axis. This geometry is central to stereo matching, which involves finding corresponding points between two images. If a 3D point X is projected onto the images as x in the first view and x' in the second, these points, along with X and the camera centers, lie on a common plane π. The back-projected rays from x and x' meet at X and are coplanar within π, a key property for correspondence search.



(a) (b)

Fig. 1. Point correspondence geometry
Source: www.robots.ox.ac.uk



(a) (b)

Fig. 2. Epipolar geometry
Source: www.robots.ox.ac.uk

Several terminologies used in epipolar geometry are the epipole, epipolar plane, and epipolar line. The epipole is the point where the baseline (the line connecting the camera centers) intersects the image plane. An epipolar plane is a plane containing the baseline and there exists a one-parameter family (a pencil) of epipolar planes. An epipolar line is the intersection of the image plane with the epipolar plane [2].

C. Linear Camera Model and Calibration

In a perspective or linear camera model, an upper right triangular matrix K is defined as the intrinsic camera matrix containing intrinsic parameters α_u and α_v as focal lengths and u_0, v_0 the image coordinates of the projection center. Let $X = [z \quad y \quad z]^T$ represent a 3D point in the camera's reference frame, and $p = [u \quad v]^T$ denote its 2D projection on the image plane in pixel coordinates. The mapping from 3D-world to 2D image of the point is expressed with the perspective projection equation (4) where λ is the depth factor.

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = KX = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4}$$

The intrinsic camera matrix must be known for an MVO system to operate. It can be obtained using camera calibration modules in existing computer vision libraries such as OpenCV. One commonly used method for camera calibration involves a planar checkerboard-like pattern. Several pictures of the board are taken at different positions and angles. The positions of the squares on the board are known or can be either obtained manually or using available detector tools. The intrinsic and extrinsic parameters of the camera are then calculated using a least-square minimization method.

C. Feature Detection and Matching

Feature detection is a process of identifying and marking points of interest in an image which are well-localized and characterized with distinct texture [1]. In this work, the ORB (Oriented FAST and Rotated BRIEF) feature detector is used to detect and describe local features of an image.

The ORB detector algorithm is a combination of FAST (Features from Accelerated Segment Test) corner detector and the BRIEF (Binary Robust Independent Elementary Features) descriptor. The FAST algorithm identifies corner-like structures rapidly and the BRIEF algorithm generates compact binary descriptors for the detected points. Additionally, ORB provides orientation information of the detected features, ensuring rotational invariance and enhancing robustness of feature matching process.

Feature matching is a process of pairing feature descriptors of one image with their corresponding descriptors in another image. The FLANN (Fast Library for Approximate Nearest Neighbors) based matcher from OpenCV is used to efficiently approximate matches of given descriptors from two sequential image frames. This method computes the hamming distance to match the descriptors. The k Nearest Neighbors (KNN) algorithm is then performed to find the k closest matches for a given descriptor.

D. Pose Estimation and Concatenation

Two images I_k and I_{k-1} at frame k taken from a calibrated camera have geometric relations described by the essential matrix E which contains the camera motion parameters up to an unknown scale factor for the translation component shown in (5).

$$E_k \simeq \hat{t}_k R_k \tag{5}$$

The symbol $\simeq$ indicates that the equivalence is valid up to a multiplicative scalar with $R_k \in \mathbb{R}^{3\times3}$ as the rotation matrix with, $t_k = [t_x \quad t_y \quad t_z]^T$ as the translation vector, and $\hat{t}_k$ a skew-symmetric matrix containing components of t_k described in (6).

$$\hat{t}_k = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \quad (6)$$

The essential matrix is computed from 2-D-to-2-D feature correspondence obtained from the feature matching process. The epipolar constraint is applied in the computation of E, which is formulated by $\tilde{p}^T E \tilde{p} = 0$, where $\tilde{p} = [\tilde{u} \quad \tilde{v} \quad 1]^T$ is a feature location in an image of the previous frame (I_{k-1}) and $\tilde{p}'$ is the location of its corresponding feature in the image of the following frame (I_k), both are normalized image coordinates. An efficient solution to E is presented in [3].

The rotation and translation vector matrix can be extracted from E using SVD. Let

$$D = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

and the SVD of E:

$$E = U\Sigma V^T, \quad (8)$$

where U and V are chosen such that $\det(U) > 0$ and $\det(V) > 0$. Then, the translation vector $t \sim t_u \equiv [u_{13} \quad u_{23} \quad u_{33}]^T$ and the rotation matrix R is equal to $R_a \equiv UDV^T$ or $R_b \equiv UD^TV^T$, all satisfying the epipolar constraint. Assuming the initial pose of the camera defined as the camera matrix $P = [I|0]$ of size 3×4 with t in unit length, there are four possible solutions for the second camera matrix: $P_A \equiv [R_a|t_u]$, $P_B \equiv [R_a|-t_u]$, $P_C \equiv [R_b|t_u]$, and $P_D \equiv [R_b|-t_u]$, one which corresponds to the true configuration. The efficient solution to the true configuration R and t is presented in [3]. The extracted R and t are then used to determine the transformation matrix of the camera from frame $k-1$ to k as shown in (9).

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \quad (9)$$

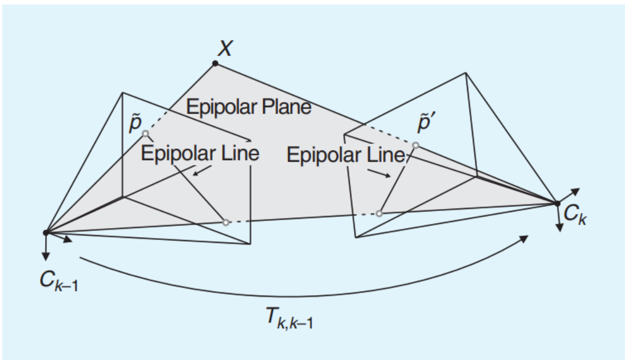


Fig. 3. An illustration of the epipolar constraint and the camera transformation
Source: ieeexplore.ieee.org/document/6096039

The obtained transformation matrix can now be used to determine the global pose of the camera if given the initial camera matrix that corresponds with the world coordinates of the camera. All subsequent motions can be defined as the set $T_{1:n} = \{T_{1,0}, \dots, T_{n,n-1}\}$ and the transformations of the camera with respect to the arbitrary initial pose at frame $k = 0$ can be defined as $C_{0:n} = \{C_0, \dots, C_n\}$. The current pose of the camera C_n can be calculated by concatenating all the previous transformations using the formula shown in (10).

$$C_n = C_{n-1}T_{n,n-1} \quad (10)$$

In a world frame of reference with an inverted axis, the transformation needs to be inverted in advance for the poses to be calculated with the correct alignment with respect to the world coordinate.

III. Methodology

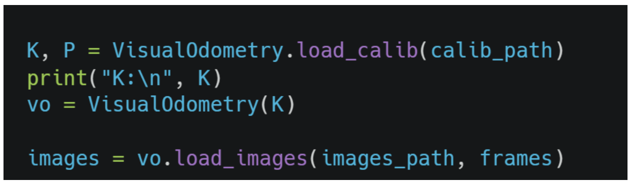
A. Dataset

The data used in this work are sampled from the KITTI Vision Benchmark Suite. The KITTI dataset provides real-world computer vision benchmarks using the Annieway autonomous driving platform. These benchmarks cover tasks such as stereo, optical flow, visual odometry, 3D object detection, and 3D tracking. Data is collected with a station wagon equipped with high-resolution cameras, a Velodyne laser scanner, and a GPS system, capturing urban, rural, and highway scenarios in Karlsruhe. The dataset includes up to 15 cars and 30 pedestrians per image and offers raw data along with task-specific benchmarks, evaluation metrics, and an evaluation website [9].

Fig. 4. A sample from the KITTI odometry dataset
Source: www.cvlibs.net/datasets/kitti

Specifically, this work uses the odometry benchmark provided by KITTI. This benchmark consists of 22 stereo sequences saved in loss less PNG format. The first 11 sequences are provided with ground truth trajectories and the remainder without the ground truth trajectories. Sequence 00 and 01 of the datasets, sampled up to 500 frames, are used in this work.

B. Initialization

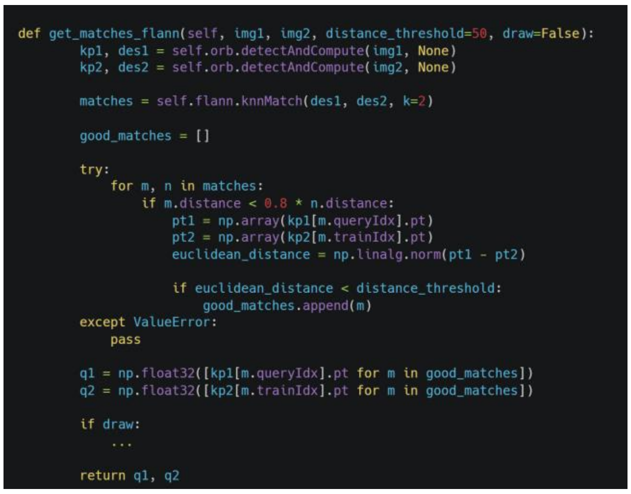


```
K, P = VisualOdometry.load_calib(calib_path)
print("K:\n", K)
vo = VisualOdometry(K)

images = vo.load_images(images_path, frames)
```

Fig. 5. Calibration parameters initialization

The MVO module is built as a class with the intrinsic camera matrix as its attribute that can be set upon instantiation. The calibration parameters, provided as a text file, are parsed to extract the intrinsic camera parameters and the initial pose. The intrinsic camera parameters throughout the experiment are constant.

C. Motion Estimation



```
def get_matches_flann(self, img1, img2, distance_threshold=50, draw=False):
        kp1, des1 = self.orb.detectAndCompute(img1, None)
        kp2, des2 = self.orb.detectAndCompute(img2, None)

        matches = self.flann.knnMatch(des1, des2, k=2)

        good_matches = []

        try:
            for m, n in matches:
                if m.distance < 0.8 * n.distance:
                    pt1 = np.array(kp1[m.queryIdx].pt)
                    pt2 = np.array(kp2[m.trainIdx].pt)
                    euclidean_distance = np.linalg.norm(pt1 - pt2)

                    if euclidean_distance < distance_threshold:
                        good_matches.append(m)
        except ValueError:
            pass

        q1 = np.float32([kp1[m.queryIdx].pt for m in good_matches])
        q2 = np.float32([kp2[m.trainIdx].pt for m in good_matches])

        if draw:
            ...

        return q1, q2
```

Fig. 6. Feature detection and matching function

The feature detection and matching processes are defined as a single function that accepts the previous and current image frames as input. Utilizing the OpenCV library, the features from both images are detected using the ORB detector and matched using the FLANN based KNN matcher with k neighbors set as 2. To deal with ambiguities, the matched features are then filtered by comparing the first score and the second score. A match is selected if the first score is less than the second score up to a threshold, which is chosen arbitrarily to be 0.8. The function then returns the keypoints of the corresponding matched descriptors. A distance threshold in pixel units is applied to filter out feature matches that are too far away from each other in terms of image coordinates. This is done to reduce false matches because of feature descriptor biases with the assumption that matching features should ideally have a relatively close distance in the image plane.

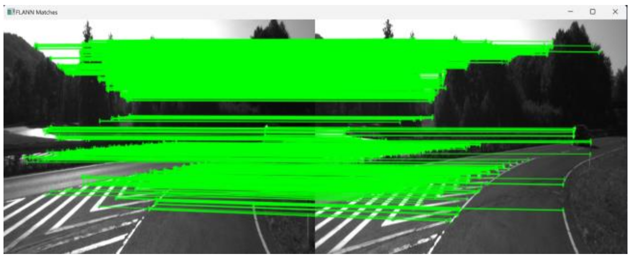
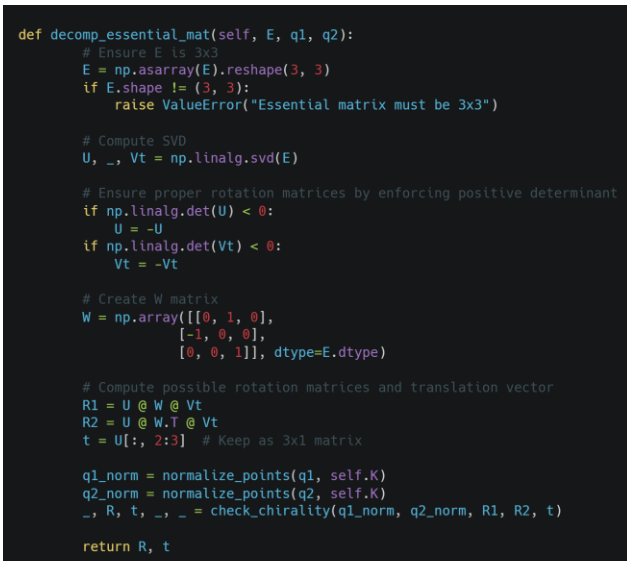


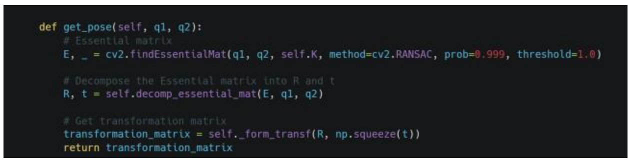
Fig. 7. Feature detection and matching visualization

Having the intrinsic camera matrix and the locations of features from two sequential images known, the essential matrix can be calculated. The computation of the essential matrix is refined using the Random Sample Consensus (RANSAC) algorithm. It is then decomposed by applying SVD and the resulting components are used to extract the rotation and translation matrices. The rotation and translation matrices are then formed into the transformation matrix as a 4×4 NumPy array.



```
def decomp_essential_mat(self, E, q1, q2):
        # Ensure E is 3x3
        E = np.asarray(E).reshape(3, 3)
        if E.shape != (3, 3):
            raise ValueError("Essential matrix must be 3x3")

        # Compute SVD
        U, _, Vt = np.linalg.svd(E)

        # Ensure proper rotation matrices by enforcing positive determinant
        if np.linalg.det(U) < 0:
            U = -U
        if np.linalg.det(Vt) < 0:
            Vt = -Vt

        # Create W matrix
        W = np.array([[0, 1, 0],
                      [-1, 0, 0],
                      [0, 0, 1]], dtype=E.dtype)

        # Compute possible rotation matrices and translation vector
        R1 = U @ W @ Vt
        R2 = U @ W.T @ Vt
        t = U[:, 2:3]  # Keep as 3x1 matrix

        q1_norm = normalize_points(q1, self.K)
        q2_norm = normalize_points(q2, self.K)
        _, R, t, _, _ = check_chirality(q1_norm, q2_norm, R1, R2, t)

        return R, t
```

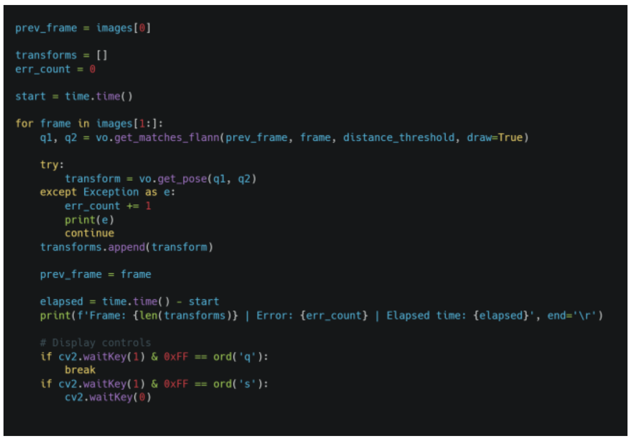
Fig. 8. Essential matrix decomposition function



```
    def get_pose(self, q1, q2):
        # Essential matrix
        E, _ = cv2.findEssentialMat(q1, q2, self.K, method=cv2.RANSAC, prob=0.999, threshold=1.0)

        # Decompose the Essential matrix into R and t
        R, t = self.decomp_essential_mat(E, q1, q2)

        # Get transformation matrix
        transformation_matrix = self._form_transf(R, np.squeeze(t))
        return transformation_matrix
```

Fig. 9. Transformation matrix calculation function

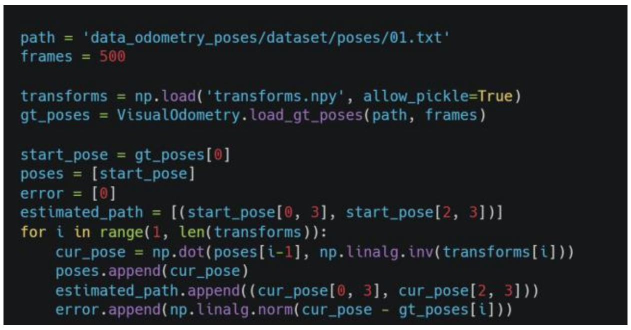
Errors might occur in the process of calculating the transformation matrix. Some are the result of the feature detector not detecting enough features for the essential matrix computation to work, resulting in the essential matrix not having the appropriate size and values. To handle this issue, image frames causing the error are skipped with the previous frame containing sufficient successfully detected features being held. The features from the previous image frame are then matched with the next image frame containing sufficient successfully detected features. The list of computed transformation matrices is then saved as a .npy file.



```
prev_frame = images[0]

transforms = []
err_count = 0

start = time.time()

for frame in images[1:]:
    q1, q2 = vo.get_matches_flann(prev_frame, frame, distance_threshold, draw=True)

    try:
        transform = vo.get_pose(q1, q2)
    except Exception as e:
        err_count += 1
        print(e)
        continue
    transforms.append(transform)

    prev_frame = frame

    elapsed = time.time() - start
    print(f'Frame: {len(transforms)} | Error: {err_count} | Elapsed time: {elapsed}', end='\r')

    # Display controls
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    if cv2.waitKey(1) & 0xFF == ord('s'):
        cv2.waitKey(0)
```

Fig. 10. Motion sequence estimation code

D. Performance Evaluation



```
path = 'data_odometry_poses/dataset/poses/01.txt'
frames = 500

transforms = np.load('transforms.npy', allow_pickle=True)
gt_poses = VisualOdometry.load_gt_poses(path, frames)

start_pose = gt_poses[0]
poses = [start_pose]
error = [0]
estimated_path = [(start_pose[0, 3], start_pose[2, 3])]
for i in range(1, len(transforms)):
    cur_pose = np.dot(poses[i-1], np.linalg.inv(transforms[i]))
    poses.append(cur_pose)
    estimated_path.append((cur_pose[0, 3], cur_pose[2, 3]))
    error.append(np.linalg.norm(cur_pose - gt_poses[i]))
```

Fig. 11. Pose concatenation and evaluation code

The evaluation of the MVO algorithm involves calculating the sequence of poses with respect to the initial calibrated pose by applying the transformation concatenation formula (10). It needs to be noted that the subsequent pose is a product of the pose from the previous frame with the inverse of the corresponding transformation to ensure that the resulting poses are properly aligned with the world frame of reference axes. The pose components that are parallel to the image baseline and perpendicular to the image plane are extracted to be visualized. Each pose is evaluated by calculating the euclidean distance with respect to the corresponding ground truth pose as the error.

$$error = \|current\ pose - ground\ truth\ pose\| \quad (11)$$

The whole performance of the algorithm is also evaluated using the mean squared error (MSE) with n as the number of poses.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(error_i)^2 \quad (12)$$

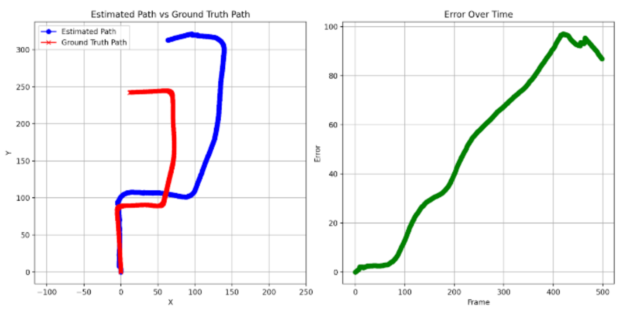
IV. RESULTS

Table 1. Calibrated intrinsic camera parameters

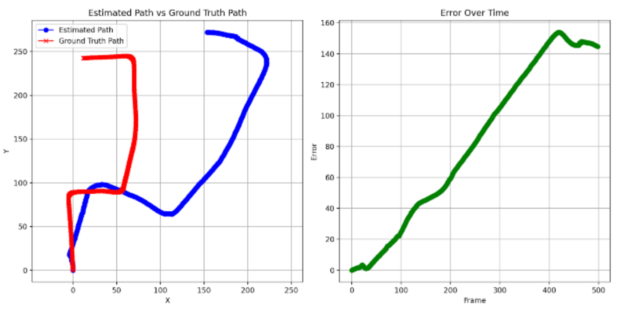
Intrinsic Camera Parameter	Value
$\boldsymbol{\alpha_u}$	718.856
$\boldsymbol{\alpha_v}$	718.856
$\boldsymbol{u_0}$	607.1928
$\boldsymbol{v_0}$	185.2157

Table 2. Monocular visual odometry evaluation benchmark

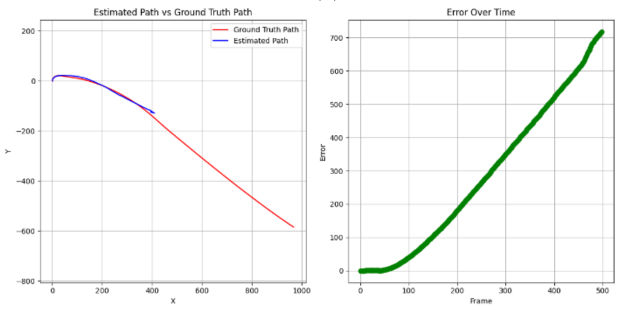
Estimated trajectory	KITTI sequence	Frames	Trajectory estimation time (s)	Feature distance threshold (pixel)	Error frames count	MSE
a	00	500	63.6727	80	0	3761.1470
b	00	500	89.1119	100	0	9435.3809
c	01	500	51.0520	80	0	130770.3815
d	01	500	82.4384	100	0	145502.4608



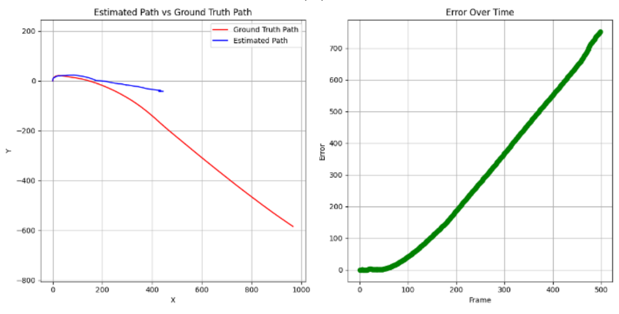
(a)



(b)



(c)



(d)

Fig. 12. Trajectory estimation results of monocular visual odometry on KITTI image sequence

The performance of the monocular visual odometry system was evaluated using the KITTI dataset on multiple sequences and summarized in Table 2. Key metrics include trajectory estimation time, feature distance threshold, error count, and (MSE). The corresponding trajectory estimation results and error analysis are visualized in Fig. 12 for four trajectories (a, b, c, d).

The trajectory estimation performance analysis is as follows:

- Trajectory (a) achieved an MSE of 3761.1470 with a feature distance threshold of 80 pixels. The system required 63.6727 seconds to estimate 500 frames, with no error frames counts reported.
- Trajectory (b), using a higher feature distance threshold of 100 pixels, resulted in a higher MSE of 9435.3809 and a longer trajectory estimation time of 89.1119 seconds, with no error frames recorded.
- Trajectory (c) demonstrated the worst performance in terms of MSE (130770.3815), despite utilizing the same feature distance threshold as trajectory (a). The estimation time for this sequence was shorter (51.0520 seconds) for the same number of frames.
- Trajectory (d), with a feature distance threshold of 100 pixels, achieved a higher MSE (145502.4608) compared to trajectory (c), with an estimation time of 82.4384 seconds.

Estimated trajectories (a) and (b), despite sharing the same ground truth poses, produced different results. With a lower feature distance threshold compared to (b), trajectory (a) demonstrated greater consistency and better overall performance. In contrast, trajectory (b) exhibited significant drifts after approximately 50 frames. This behavior is attributed to the feature matcher in (b), which covered a broader search area for feature pairs, increasing the likelihood of false matches and leading to inaccuracies in the calculated transformations. This effect is also showcased in the comparison of trajectories (c) and (d)

Despite its better performance, trajectory (a) also suffered from drifts after the first turn, causing cumulative errors over time. Additionally, low texture richness in the images could further degrade performance by limiting the feature detector's ability to identify distinct features. This challenge is particularly evident in urban areas, where roads and building facades often lack sufficient distinctive points for detection. Nevertheless, despite these limitations, the MVO algorithm demonstrated capability in identifying turns within the trajectory.

Sequence 01 presents a distinct scenario compared to the estimations in Sequence 00. Rather than featuring sharp turns, it primarily showcases a straight trajectory. At first glance, the MVO algorithm appears to perform reasonably well in estimating the agent's direction of motion, with minimal offsets especially observed in trajectory (c). However, the algorithm significantly underperforms in accurately capturing the agent's translational movement, leading to notable discrepancies in positional estimates.

Sequence 01 portrays the agent as a car traveling at high speed along a highway. The substantial motion between consecutive frames presents a significant challenge for the algorithm in detecting matching features that accurately follow the agent's true trajectory. This challenge, referred to as motion ambiguity [1], can greatly affect the accuracy of trajectory estimation, particularly for agents operating across a wide range of speeds.

Other unexamined external factors might contribute to drift errors. The KITTI dataset includes scenarios with moving objects such as cars that might disrupt the feature matching process by introducing inconsistent motions relative to the overall scene.

V. Conclusion

The evaluation of the MVO system using the KITTI dataset revealed strengths and limitations across different sequences and trajectories. The system demonstrated reasonable performance in estimating motion direction, particularly on straight trajectories like sequence 01. However, significant challenges were noted in accurately capturing translational movements, with issues like motion ambiguity and drift errors impacting overall trajectory estimation with cumulative errors.

Key findings include the influence of feature distance thresholds on performance, as trajectories with lower thresholds (e.g., trajectory (a)) consistently outperformed those with higher thresholds (e.g., trajectory (b)), highlighting the trade-off between computational efficiency and matching accuracy. Despite these improvements, cumulative errors were observed in more complex scenarios, especially after turns or over extended sequences.

External factors such as low texture richness and dynamic elements (e.g., moving cars, pedestrians) further degraded performance by complicating the feature matching process. Urban settings, in particular, posed unique challenges due to limited distinctive features in the environment.

Despite these limitations, the MVO algorithm exhibited competency in identifying turns and maintaining reasonable performance on specific trajectories. Future work should focus on mitigating the impact of motion ambiguity, improving feature matching robustness in dynamic scenes, and addressing texture limitations to enhance the system's reliability in diverse and challenging scenarios. Alternative methods in matrix processing such as SVD for sparse matrix can also be explored to open other possibilities of improvement in performance.

VI. Acknowledgment

The author would like to express sincere gratitude to the providers of the KITTI dataset for offering a comprehensive benchmark for evaluating visual odometry algorithms. Sincere thanks also go to the lecturers of IF2123 Linear Algebra and Geometry, whose guidance and knowledge were invaluable, and to the assistants of the IRK Laboratory for their assistance and expertise. The author is deeply appreciative of friends, who provided support and motivation along the way. Lastly, the author is profoundly grateful to God for granting the strength and perseverance to complete this work. Finally, the author acknowledges the computational resources and facilities that made this research possible.

REFERENCES

[1] S. Sellak, Y. Alj and Y. Salih-Alj, "Monocular Visual Odometry in Mobile Robot Navigation," 2024 IEEE 12th International Symposium on Signal, Image, Video and Communications (ISIVC), Marrakech, Morocco, 2024, pp. 1-6, doi: 10.1109/ISIVC61350.2024.10577766.

[2] R. Hartley, A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, 2004.

[3] D. Nister, "An efficient solution to the five-point relative pose problem," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756-770, June 2004, doi: 10.1109/TPAMI.2004.17.

[4] D. Nister, O. Naroditsky and J. Bergen, "Visual odometry," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Washington, DC, USA, 2004, pp. I-I, doi: 10.1109/CVPR.2004.1315094.

[5] D. Scaramuzza and F. Fraundorfer, "Visual Odometry [Tutorial]," in *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80-92, Dec. 2011, doi: 10.1109/MRA.2011.943233.

[6] V. Guruswami and R. Kannan, “Notes 2: Singular Value Decomposition,” 15-496/15-859X: Computer Science Theory for the Information Age, Spring 2012. [Online]. Available: https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/. [Accessed: Jan. 1, 2025].

[7] G. Strang, “Lecture 2: The Singular Value Decomposition (SVD) of a Matrix,” 18.095 - Mathematics Lecture Series, IAP 2016, held in 4-270, Massachusetts Institute of Technology, Cambridge, MA, Jan. 6, 2016. [Online]. Available: https://math.mit.edu/classes/18.095/2016IAP/. [Accessed: Jan. 1, 2025].

[8] Y. Xie, Q. Wang, Y. Chang, and X. Zhang, “Fast target recognition based on improved ORB feature,” *Applied Sciences*, vol. 12, no. 2, p. 786, 2022. [Online]. Available: https://doi.org/10.3390/app12020786.

[9] A. Geiger, P. Lenz, and R. Urtasun, ‘Are we ready for autonomous driving? The KITTI vision benchmark suite’, in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, 2012.

STATEMENT

Hereby, I declare that the paper I have written is my own work, not an adaptation or translation of someone else's paper, and is not plagiarism.

Bandung, January 2nd, 2025

[illegible]

Karol Yangqian Poetracahya
13523093