Localization Using Fast Monocular Visual Odometry*

*EECE5639 Computer Vision Course Project, Spring 2024

Mohammad Hussein Yoosefian Nooshabadi
Department of Mechanical and Industrial Engineering
Northeastern University
Boston, MA 02115, USA
yoosefiannooshabad.m@northeastern.edu

Milad Alipour Shahraki
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115, USA
alipourshahraki.m@northeastern.edu

***Abstract*—This project studies the detailed design of a Monocular Visual Odometry (MVO) algorithm. MVO is a technique used to localize a robot using only a stream of images acquired from a single camera attached to the robot. Despite its popularity, the conventional MVO suffers from high computational costs that can be problematic for applications involving resource-constrained robots. To reduce this computational burden, a Fast MVO (FMVO) approach is introduced, in which a novel feature management technique is employed. Simulations and implementation results on an open-source dataset show the effectiveness of the proposed FMVO approach in significantly reducing the computation time compared to the conventional MVO.**

***Index Terms*—Monocular Visual Odometry (MVO), Localization, Feature Detection and Tracking, Optimization**

I. Introduction

A. Motivation

In the expanding field of autonomous robotics, navigation, particularly in environments where traditional navigation aids such as GPS are unavailable or unreliable poses a significant challenge. As robotics applications expand into more complex and less structured environments—from disaster recovery areas to extraterrestrial exploration—the ability for robots to independently determine their location and navigate effectively becomes increasingly important. Robots operating in such environments must be capable of real-time localization without the luxury of multiple sensors or external references. Among various sensors available for navigation, cameras are particularly popular due to their low cost and high quality.

This project report explores the design of a Monocular Visual Odometry (MVO) system for a robot equipped with a single camera and further extends the idea of the conventional MVO to Fast MVO (FMVO) that can reduce the inherent high computational cost associated with the conventional MVO.

B. Literature Review

Monocular Visual Odometry (MVO) refers to a technique in which a single camera is used to incrementally estimate the position (up to a scale) and orientation of a robot on which the camera is attached [1]. Visual Odometry (VO) has been extensively studied in the context of various applications, including space exploration [2], unmanned aerial vehicles (UAVs) [3], autonomous underwater vehicles [4], and autonomous driving [5], [6]. In [1], a comprehensive overview of MVO and its applications, emphasizing its cost-effectiveness and simplicity compared to stereo vision systems is provided. Their research highlights the challenges of using a single camera, particularly in accurately estimating depth and scale from motion, which is crucial for precise localization.

Additionally, the combination of VO with other sensory inputs like LIDAR and Inertial Measurement Units (IMUs) has been gaining traction. This multimodal approach significantly boosts the robustness and reliability of autonomous navigation systems [7]. By integrating multiple sensor outputs, these systems can overcome individual sensor limitations, offering a more comprehensive understanding of the environment and ensuring safer navigation decisions in complex scenarios [8].

Recent studies have also explored the integration of machine learning techniques with traditional VO methods to enhance pose estimation accuracy. The integration of Convolutional Neural Networks (CNNs) for feature extraction directly from raw images has shown promising results [9], [10]. These studies have demonstrated how deep learning could be leveraged to improve the robustness of MVO against rapid motion changes and varying lighting conditions, challenges that are often present in traditional approaches.

Although the conventional MVO has been extensively studied and improved throughout the literature by leveraging its combination with other sensors and learning-based methods the conventional MVO on its own relies on extracting and tracking of features (such as corners or blobs) between two consecutive images iteratively. This procedure can be computationally expensive and require significant computing time. This problem can be challenging specifically for resource-constrained robots such as UAVs. Nevertheless, this high computational cost issue has not been well explored in the literature.

In this work, we aim to address the challenge of the high

computational cost of the conventional MVO. To this end, we introduce the Fast MVO (FMVO) method that instead of extracting features in all images, performs feature extraction only for a small number of images, significantly reducing the associated computing time.

The remainder of this report is structured as follows: Section II outlines the problem formulation. Section III presents the technical details of our approach. Section IV presents comparative numerical simulations to evaluate the proposed method. Lastly, Section V concludes the report, summarizes key findings, and suggests potential directions for future research.

II. Problem Formulation

In this section, we present the problem formulation for the MVO. The general pipeline for MVO is given in Fig. 1.

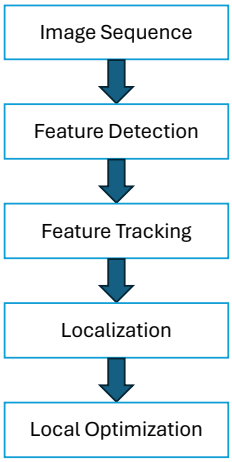


Fig. 1: General flowchart for the MVO algorithm

In the following subsections, we explain the elements present in Fig. 1.

A. Image Sequence

The camera generally captures colored images; however, in VO algorithms, black-and-white images are usually used. Each image is saved as a matrix where the elements of the matrix represent the intensity of pixels of the image. The images saved by the camera are usually distorted. A distorted image sample from the EuRoC dataset [11] is given in Fig. 2.

To solve the distortion problem, each distorted pixel (u_d, v_d) is mapped into a new pixel space to remove the distortion as

$$\begin{bmatrix} u \\ v \end{bmatrix} = (1+k_1 r+k_2 r^2) \begin{bmatrix} u_d \\ v_d \end{bmatrix} + \begin{bmatrix} 2u_d v_d t_1 + (r+2u_d^2)t_1 \\ 2u_d v_d t_2 + (r+2v_d^2)t_2 \end{bmatrix} \quad (1)$$

where $r = \sqrt{u_d^2 + v_d^2}$, and parameters k_1, k_2, t_1, and t_2 that are obtained during camera calibration are given in III-A. The undistorted version of the image sample in Fig. 2 is given in Fig. 3.



Fig. 2: A distorted image sample from the EuRoC dataset [11]



Fig. 3: The undistorted version of the image sample (see Fig. 2)

B. Feature Detection

Each feature in an image can be assigned a descriptor. Feature matching or tracking in multiple images is usually achieved by comparing the descriptors. Features should be repetitive and distinctive. Repetitiveness means that the features should be easily detectable in other images, and distinctiveness means that their descriptors should be different from the descriptors of other features. In general, there are two types of features: corners and blobs. Corners are defined by the intersection of two or more edges and blobs are any other detected pattern beside corners. In general, corners are often preferred over blobs in VO algorithms due to their stability, accuracy, and robustness to motion and rotation, making them more reliable for tracking and localization. In this work, we use the Harris corner detector for its reliability, low numerical complexity, and robustness. In Fig. 4, the Harris corners are extracted in the undistorted image sample shown in Fig. 3.



Fig. 4: Harris corner detection applied to the undistorted image sample (see Fig. 3)

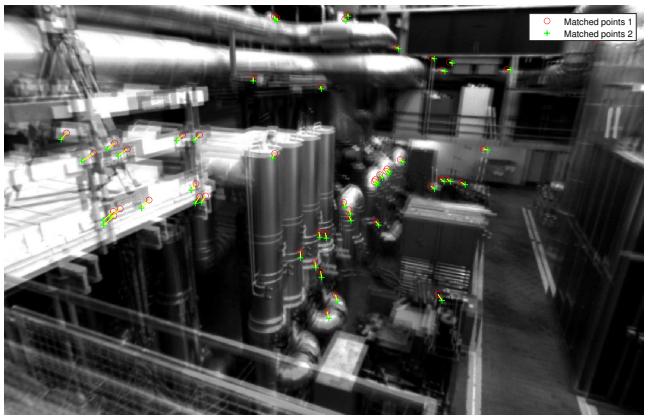


Fig. 5: The KLT tracker applied to two subsequent sample images from the EuRoC dataset [11]

C. Feature Tracking

In this subsection, the feature tracking procedure is explained. After detecting a feature it should be separated from the other features by a descriptor that can be followed in subsequent images. One approach to feature matching is to compare each detected feature in one image to all detected features in subsequent images; however, this incurs a significant computational cost. To reduce computational complexity, algorithms such as the Kanade-Luca-Tomasi (KLT) tracker [12] were introduced.

The KLT feature tracker is a widely used method in computer vision for tracking feature points through sequential images. It operates on the principle of finding the best match for a small window around a feature point (like a corner) between consecutive frames. The core of the KLT tracker is an assumption that the pixel intensities of a feature point remain constant between consecutive images, despite movements in the scene or camera. It employs an iterative refinement process based on the Lucas-Kanade optical flow method, which solves the displacement of the window in the next frame using a gradient descent algorithm. This approach calculates the best displacement (both in x and y directions) that minimizes the difference in appearance between the original and the moved window, effectively tracking the feature through the video. The KLT tracker is especially proficient in handling small and predictable motions, making it highly effective in maintaining tracking fidelity in applications like VO, where continuous and accurate tracking of points is crucial for determining camera movement. In Fig. 5, the KLT tracker is applied to two subsequent images for feature tracking.

D. Localization

In this subsection, the localization method is explained. In conventional MVO, the pose and orientation estimation is achieved by tracking the features between two sequential images by computing the essential or fundamental matrices. A minimum of five points is required for this method which will be proved later. To estimate the motion between two sequential images, the epipolar geometry is used. In this regard, it is assumed that the coordinate system of the first image coincides with the global coordinate system, and the rotation and translation of the second image's coordinate system are computed relative to the first one.

The normalized coordinates for the i-th feature can be defined as

$$ {}^{i}\bar{p} = \begin{bmatrix} {}^{i}\bar{u} \\ {}^{i}\bar{v} \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} {}^{i}u \\ {}^{i}v \\ 1 \end{bmatrix} \tag{2} $$

where the matrix K is the camera calibration matrix. Let ${}^{i}\bar{p}_1$ and ${}^{i}\bar{p}_2$ be the normalized coordinates for the i-th feature detected in the first and second images as shown in Fig. 6.

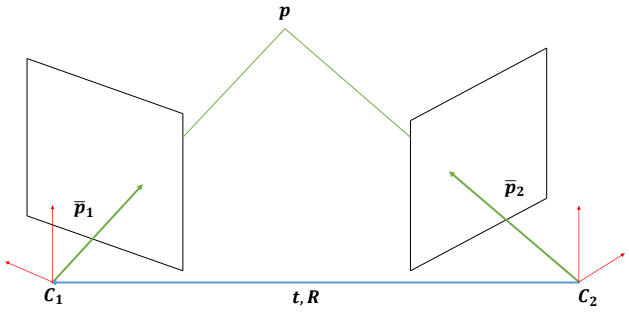


Fig. 6: Epipolar geometry.

Epipolar geometry states that the line (translation vector t in Fig. 6) connecting the center of the two coordinate systems for two sequential images, and the lines connecting the centers of the coordinate systems of each image to the 3D point P (the line segments C_1P and C_2P in Fig. 6) are in one plane.

This means that if the normal vector for the C_1C_2P plane in Fig. 6 is defined as

$$n_1 = t \times \bar{p}_1 \tag{3}$$

then, according to the epipolar geometry, $\bar{p}_2$ is perpendicular to this vector

$$\bar{p}_2^\top \cdot n_1 = 0. \tag{4}$$

In (4), n_1 is expressed in the coordinate system of the first image, while the vector $\bar{p}_2$ is expressed in the coordinate system of the second image. Multiplying n_1 by the rotation matrix, we transfer n_1 to the coordinate system of the second image; therefore, (4) can be rewritten as

$$\begin{aligned} \bar{p}_2^\top \cdot n_1 = \\ \bar{p}_2^\top \cdot t \times \bar{p}_1 = \\ \bar{p}_2^\top t \times (R\bar{p}_1) = \\ \bar{p}_2^\top ([t\times]R)\bar{p}_1 = 0 \end{aligned} \tag{5}$$

where $[t\times]$ represents the skew-symmetric cross-product matrix corresponding to the vector t. By defining

$$E = [t\times]R \tag{6}$$

the epipolar constraint is obtained as

$$\bar{p}_2^\top E \bar{p}_1 = 0 \tag{7}$$

where matrix E is called the *Essential Matrix*.

As mentioned earlier it can be proved that to uniquely determine the Essential Matrix, a minimum of five feature points is required. The quality of the estimation will increase by extracting more feature points. One of the famous algorithms in this regard is the *eight-point* algorithm [13]. In this case, the Essential Matrix is defined as

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \tag{8}$$

It is assumed that the rows and columns of this matrix are linearly independent. This assumption is not correct and leads to undesirable errors. In order to mitigate this problem a local optimization is performed, which is further explained in the next subsection.

We can rewrite equation (7) for the i-th feature point as

$$\begin{aligned} {}^i\bar{u}_2\, {}^i\bar{u}_1 e_{11} + {}^i\bar{u}_2\, {}^i\bar{v}_1 e_{12} + {}^i\bar{u}_2 e_{13} + {}^i\bar{v}_2\, {}^i\bar{u}_1 e_{21} \\ + {}^i\bar{v}_2\, {}^i\bar{v}_1 e_{22} + {}^i\bar{v}_2 e_{23} + {}^i\bar{u}_1 e_{31} + {}^i\bar{v}_1 e_{32} + e_{33} = 0 \end{aligned} \tag{9}$$

For n features, (9) can be written in the matrix form as (10), where $Q_{n\times 9}$ is known and $\bar{E} \coloneqq \text{vec}(E)$ is unknown. For the system of equations (10) to have a unique (up to a scale) non-trivial solution $\bar{E}$, the matrix $Q_{n\times 9}$ should have rank 8. Each feature point correspondence provides 1 independent equation; thus, 8 point correspondences are needed [13]. A solution is to minimize $\|Q\bar{E}\|^2$ subject to the constraint $\|\bar{E}\| = 1$. The solution is the eigenvector corresponding to the smallest eigenvalue of the matrix $Q^\top Q$ (because it is the unit vector x that minimizes $\|Qx\|^2 = x^\top Q^\top Q x$). This can be solved through Singular Value Decomposition (SVD).

In some works, instead of using the Essential Matrix E, the *Fundamental Matrix*, F, is used. Substituting (2) in (7) we obtain

$$p_2^\top \underbrace{K^{-\top} E K^{-1}}_{F} p_1 = 0 \tag{11}$$

where as mentioned earlier, $F = K^{-\top} E K^{-1}$ is the Fundamental Matrix that can be computed in a similar manner to the Essential Matrix E. More specifically, in the equation (10), instead of using ${}^i\bar{p},\ i = 1, \dots n$, we use ${}^i p,\ i = 1, \dots n$ and replace $\bar{E}$ with $\bar{F} \coloneqq \text{vec}(F)$.

E. Local Optimization

As mentioned in the previous subsection, the assumption that the rows and columns of the Essential or Fundamental Matrix are linearly independent can lead to error. To reduce this error we can compute the reprojection error as shown in Fig. 7 and minimize it by formulating a local optimization problem.

The reprojection error is computed as

$$e = \|p_1 - \pi(P, K, I, 0)\|^2 + \|p_2 - \pi(p, K, R, t)\|^2 \tag{12}$$

where π is the reprojection function that projects a 3D point P on an image that has the rotation R and translation t with respect to the first image. In the end, a local optimization is defined that can increase the quality of the estimation and reduce the reprojection error (see subsection III-E). In case the feature is observed in $n_f > 2$ images, the projection error is computed and minimized for all n_f images. For m features in observed in the n_f images, the reprojection error is thus defined as

$$e = \sum_{i=1}^{m} \sum_{k=1}^{n_f} \|{}^i p_k - \pi({}^i P, K, R_k, t_k)\|^2 \tag{13}$$

Now the goal is to minimize (13). Therefore, we can formulate the following optimization problem

$$R_i, t_i = \underset{R_i, t_i}{\arg\min} \sum_{i=1}^{m} \sum_{k=1}^{n_f} \|{}^i p_k - \pi({}^i P, K, R_k, t_k)\|^2 \tag{14}$$

where R_i and t_i are the refined rotation matrices and translation vectors, respectively.

III. Technical Approach

In this section, a detailed explanation of the MVO pipeline shown in 1 is provided. As explained before, in this work, in addition to the ordinary MVO, we have developed and used a Fast MVO, which has comparable performance in terms of accuracy, while significantly outperforming the conventional MVO in terms of update rate. The difference between Fast MVO and conventional MVO is in the local optimization, which is explained in III-F.

$$\underbrace{\begin{bmatrix} {}^1\bar{u}_2\,{}^1\bar{u}_1 & {}^1\bar{u}_2\,{}^1\bar{v}_1 & {}^1\bar{u}_2 & {}^1\bar{v}_2\,{}^1\bar{u}_1 & {}^1\bar{v}_2\,{}^1\bar{v}_1 & {}^1\bar{v}_2 & {}^1\bar{u}_1 & {}^1\bar{v}_1 & 1 \\ {}^2\bar{u}_2\,{}^2\bar{u}_1 & {}^2\bar{u}_2\,{}^2\bar{v}_1 & {}^2\bar{u}_2 & {}^2\bar{v}_2\,{}^2\bar{u}_1 & {}^2\bar{v}_2\,{}^2\bar{v}_1 & {}^2\bar{v}_2 & {}^2\bar{u}_1 & {}^2\bar{v}_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ {}^n\bar{u}_2\,{}^n\bar{u}_1 & {}^n\bar{u}_2\,{}^n\bar{v}_1 & {}^n\bar{u}_2 & {}^n\bar{v}_2\,{}^n\bar{u}_1 & {}^n\bar{v}_2\,{}^n\bar{v}_1 & {}^n\bar{v}_2 & {}^n\bar{u}_1 & {}^n\bar{v}_1 & 1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} e_{11} \\ e_{21} \\ \vdots \\ e_{33} \end{bmatrix}}_{\tilde{E}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{10}$$

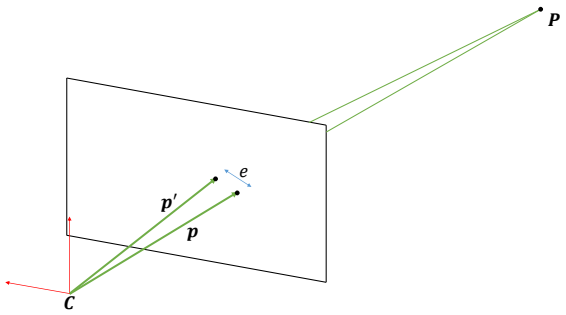


Fig. 7: Reprojection Error.

A. Image Sequence

The distortion model used in this project is radial-tangential. The distortion parameters used in (1), are obtained from the dataset as [11]

$$\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 0.00019359 \\ 1.76187114 \times 10^{-5} \end{bmatrix}$$
$$\begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} -0.28340811 \\ 0.07395907 \end{bmatrix}$$

The MATLAB built-in function `undistortImage` is used to undistort the images.

B. Feature Detection

To extract Harris corners in images, the MATLAB built-in function `detectHarrisFeatures` is used. 1000 corners are extracted originally, among which 800 are uniformly selected to be tracked in the next image.

C. Feature Tracking

To implement the KLT tracking algorithm, the MATLAB built-in function `vision.PointTracker` is used. The maximum bidirectional error, `MaxBidirectionalError`, is set to 0.1, and the patch size, `BlockSize`, is set to [21 21]. Additionally, the MATLAB built-in function `estimateFundamentalMatrix` is used to apply RANdom SAmple Consensus (RANSAC) to remove outliers in tracked features. In this function, the number of trials, `NumTrials`, is set to 3000, and the Sampson distance threshold for finding outliers, `DistanceThreshold`, is set to 10^{-3}.

D. Localization

Camera position and orientation are computed using the well-known eight-point algorithm [13], which will be explained in this subsection. Assume that n features have been extracted in the first image and tracked in the next one. First, we have to normalize these features, since using feature points that are not normalized may cause numerical instabilities [13]. There are several normalization methods in the literature. Similar to [13], we normalize the features such that they are centered at the origin and their mean distance from the origin is $\sqrt{2}$. To do that, we compute the mean and the variance as

$$\mu_j = \sum_{i=1}^{n} \frac{1}{n}\, {}^i p_j$$
$$\sigma_j = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\| {}^i p_j - \mu_j \|^2)} \tag{15}$$

in which ${}^i p_j \in \mathbb{R}^2$ is the i-th feature observed in the j-th image, with $j = 1, 2$, and $\mu_j = \begin{bmatrix} \mu_j^x & \mu_j^y \end{bmatrix}^\top$ and σ_j are the mean and covariance, respectively. The normalization is done using

$${}^i\tilde{p}_j = \underbrace{\begin{bmatrix} \sqrt{2}/\sigma_j & 0 & -\frac{\sqrt{2}}{\sigma_j}\mu_j^x \\ 0 & \sqrt{2}/\sigma_j & -\frac{\sqrt{2}}{\sigma_j}\mu_j^y \\ 0 & 0 & 1 \end{bmatrix}}_{T_j} {}^i p_j \tag{16}$$

The normalization process code is written in the provided MATLAB function `normalize2dpts.m`. The computed Fundamental Matrix using normalized features should be denormalized using

$$F = T_2^\top \tilde{F} T_1 \tag{17}$$

In the next step, the Fundamental Matrix is computed using the normalized points, as explained in II-D. After computing the Fundamental Matrix using (10), we project F onto the set of 3×3 singular matrices by enforcing $\det(F) = 0$ using the following code

```
[u,s,v] = svd(F);
s(3,3) = 0;
F = u*s*v';
```

The code for computing the Fundamental Matrix is provided in `fundamentalEightPoint_normalized.m`. Now, the Essential Matirx can be computed using (11) as

$$E = K^\top F K \tag{18}$$

The corresponding code for computing the Essential Matrix is available in `estimateEssentialMatrix.m` file. Next, the rotation matrix, R, and the translation vector, t, should be recovered from the Essential Matrix. Four possible combinations of R and t are possible (see [14], chapter 21). Among these four combinations, we choose the one for which all 3D points are in front of the camera (see Fig. 8).

To recover the correct R and t, we first compute the SVD of the Essential Matrix. Then, the rotation matrix and the translation vector become

$$R_{1,2} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^\top \qquad t_{1,2} = \pm \frac{u_3}{\|u_3\|} \tag{19}$$

where u_3 is the third column of the matrix U. These formulas are easy to interpret from Fig 8. The code used to decompose the Essential Matrix is `decomposeEssentialMatrix.m`.

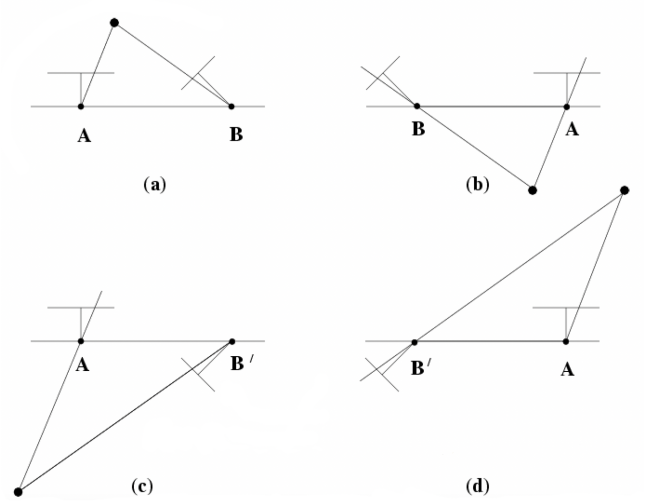


Fig. 8: Four possible configurations for a given Essential Matrix between two images. Configuration (a) is the correct one, as the point is in front of both images.

In order to find the correct configuration, we compute the 3D location of the n features for each combination and choose the one that corresponds to the largest number of points in front of the camera. Note that even with the correct combination of rotation and translation, some points might still be behind one or both of the images. The reason for that is measurement noise and the assumption in the eight-point algorithm that the rows and columns of the Fundamental Matrix are independent. So, we first need to compute the 3D point corresponding to each feature. This process is called *Triangulation*. Ideally, the lines passing camera centers and features should intersect at a single point, which corresponds to the 3D point. This is shown in Fig. 6.

However, in the real world, due to measurement noise, the rays do not intersect at a single point. In this case, as shown in Fig. 9, the closest point to both rays is selected as the 3D point. To compute the triangulated point, we start with the

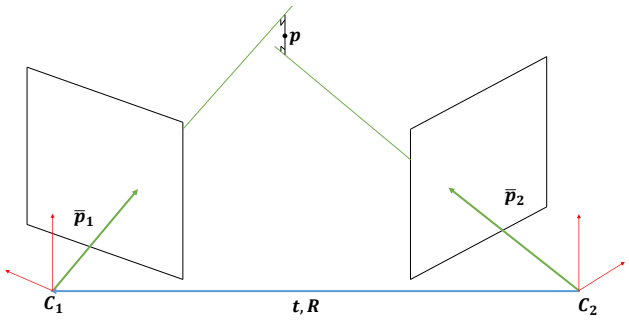


Fig. 9: Triangulation in the real world. In this case, rays from image centers to features do not intersect at a single point.

perspective projection equation. Assuming that the left image is located at the origin of the global frame, we can write the perspective projection equation as

$$\begin{bmatrix} {}^iu_l \\ {}^iv_l \\ 1 \end{bmatrix} = \underbrace{K \begin{bmatrix} I_{3\times 3} & 0_{3\times 1} \end{bmatrix}}_{M_l} \begin{bmatrix} {}^iP_x \\ {}^iP_y \\ {}^iP_z \\ 1 \end{bmatrix} \qquad \begin{bmatrix} {}^iu_r \\ {}^iv_r \\ 1 \end{bmatrix} = \underbrace{K \begin{bmatrix} R & t \end{bmatrix}}_{M_r} \begin{bmatrix} {}^iP_x \\ {}^iP_y \\ {}^iP_z \\ 1 \end{bmatrix} \tag{20}$$

in which ${}^ip_l = \begin{bmatrix} {}^iu_l & {}^iv_l & 1 \end{bmatrix}^\top$ and ${}^ip_r = \begin{bmatrix} {}^iu_r & {}^iv_r & 1 \end{bmatrix}^\top$ are homogeneous coordinates of the i-th feature observed in the left and right images, respectively. Also, ${}^iP = \begin{bmatrix} {}^iP_x & {}^iP_y & {}^iP_z & 1 \end{bmatrix}^\top$ is the coordinates of the 3D point corresponding to the i-th feature. We then cross product both sides of the equations in (20) with ${}^ip_l$ and ${}^ip_r$, respectively, to obtain

$$0 = [{}^ip_l\times]M_l^iP \qquad 0 = [{}^ip_l\times]M_r^iP \tag{21}$$

Now, we can solve a least squares problem to obtain an estimate of ${}^iP$. The code for this process is written in `linearTriangulation.m` file.

The process of triangulation is done for all n features observed in two images and for all four combinations of R and t. The configuration that yields the largest number of points in front of both images is chosen as the correct R and t. The code for the disambiguation procedure is provided in `disambiguateRelativePose.m` file.

E. Local Optimization: MVO

In the previous section, we computed the relative rotation and translation between two consecutive images using feature tracks. As explained in II-E, to enhance the quality of the estimated pose, usually a local optimization is performed. Two famous local optimization methods are Bundle Adjustment (BA)

[15] and Pose Graph Optimization (PGO) [16]. Both algorithms minimize the reprojection error defined in (13). The difference is in the optimization variables; in BA only the camera poses are optimized, while in PGO, in addition to the camera poses, the 3D points are optimized. Usually, the last m frames are used for optimization. In this work, PGO is performed for every two consecutive images. The code for this procedure can be found in `reprojectionErrorOptimization.m` file.

F. Local Optimization: Fast MVO (FMVO)

There is a difference between the local optimization used in this work and the conventional MVO. In the conventional MVO, the number of frames used in PGO or BA, m, is chosen based on the computational resources. However, in this work, a new approach is taken to choose m, which is explained in this subsection.

In the conventional MVO, upon receiving the first image, a number of features, n, is extracted and tracked in the second image. Some features are not observed in the second image, due to several reasons. First, feature trackers like the KLT, which is used in this project, are not able to track features perfectly. Also, the RANSAC algorithm might mistakenly detect some feature matches as outliers and remove them. Furthermore, some features might have fallen outside the field of view of the camera. In either of these cases, new features are extracted from the last recorded image to replace the features that are lost. This feature extraction procedure is computationally expensive.

To reduce the computational effort, we propose a new scheme in which features are only extracted in particular images instead of all of them. The criterion based on which we select these particular images is a new parameter, $n_{f_{min}}$, which corresponds to the minimum number of features that are being tracked. To delineate, in the FMVO, when the first image is received, similar to the conventional MVO, n features are extracted. These features are tracked in the next image. However, contrary to the conventional MVO, no new features are extracted in the second image to replace the lost features. Therefore, upon receiving the next image, only features that are observed in the second image are tracked. As a result, the number of features being tracked is constantly decreasing.

When the number of tracked features reaches $n_{f_{min}}$, the remaining features are used to perform a BA or a PGO, and new features are extracted in the most recent image. As a result, instead of extracting features in all images, feature extraction is performed only in the images in which the number of tracked features is less than $n_{f_{min}}$. Similar to the conventional MVO, we use PGO algorithm to refine camera poses and 3D points in the FMVO as well. The code for this procedure can be found in `reprojectionErrorOptimization2.m` file.

Note that the triangulation process described in subsection III-D, which was explained for two frames, can be easily extended to several images. The code for triangulation of a feature observed in several frames is written in `linear_triangulation.m` file.

As shown in Section IV, this strategic change in feature tracking results in a significant reduction in computation time, while having a comparable accuracy in position and orientation estimation.

IV. Simulations and Results

In this section, we present the simulation results. In this work, we used MH_01 sequence from the EuRoC dataset [11]. The scale factor is computed using the ground truth position provided in the dataset. As explained before, for the conventional MVO, the local optimization process consists of a PGO between every two consecutive frames. We chose $n_{f_{min}} = 50$ for the FMVO algorithm. Both algorithms were executed on a MacBook M2 Pro with 32 GB of RAM using MATLAB R2022b. 2D trajectory and orientation plots are shown in Figs. 10 and 11.

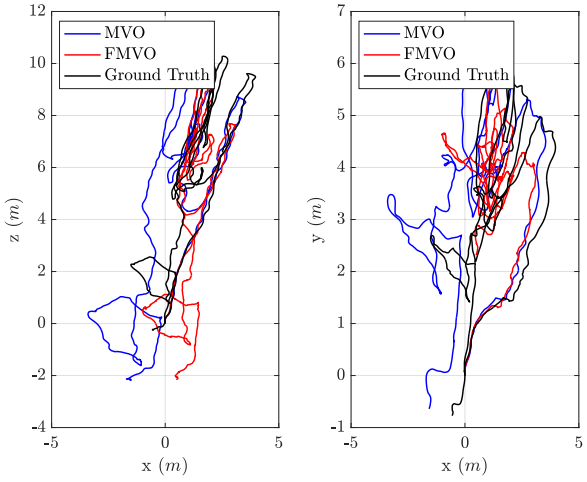


Fig. 10: Estimated trajectory using the conventional MVO and the FMVO along with the ground-truth trajectory.

As it can be seen from Fig. 10, the algorithms perform similarly. However, it is well-known that VO algorithms suffer from accumulative error [17], that can be observed in Fig. 10. One solution to this problem is to fuse camera data with other sensors, such as an IMU [18]. Based on Fig. 11, it can be observed that the performance of the conventional MVO in orientation estimation is slightly better than that of the FMVO. To have a better comparison, the Root Mean Square Error (RMSE) for orientation and position is calculated. The RMSE at iteration t for a generic vector x is computed using

$$\text{RMSE}(t) = \sqrt{\frac{\sum_{i=1}^{t} \|\hat{x} - x\|^2}{N}} \tag{22}$$

in which N is the total number of iterations and $\hat{x}$ is the estimate of the vector x. The RMSE of the orientation and position are plotted in Figs. 12 and 13.

Also, Table I compares the performance of the two algorithms used in this work. The first six rows in this table are

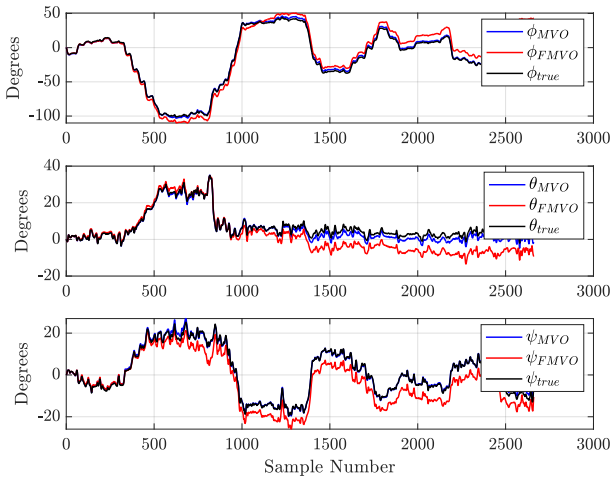


Fig. 11: Estimated orientation using the conventional MVO and the FMVO along with the ground-truth orientation.

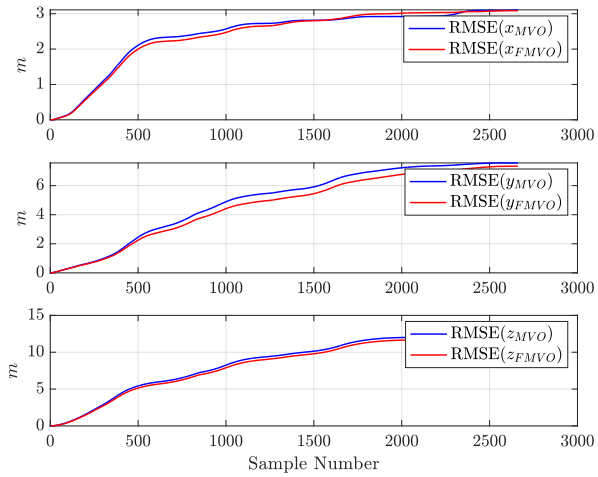


Fig. 12: The RMSE of trajectory estimation in different directions using the conventional MVO and the FMVO.

TABLE I: Comparison of the performance between the conventional MVO and the FMVO.

Algorithm	MVO	FMVO
x RMSE (m)	2.76	**2.72**
y RMSE (m)	5.62	**5.14**
z RMSE (m)	9.66	**9.31**
ϕ RMSE (°)	**0.94**	3.99
θ RMSE (°)	**0.35**	1.42
ψ RMSE (°)	**0.50**	2.77
Run-time (s)	1.03	**0.05**

median of the RMSE results in Figs. 12 and 13. The last row is the *median* run-time of each iteration of the algorithms.

To have a better visual comparison, loop run-time and

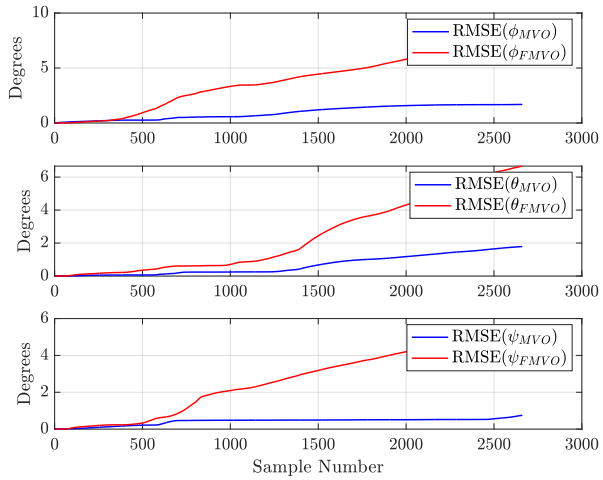


Fig. 13: The RMSE of orientation estimation in different directions using the conventional MVO and the FMVO.

accumulative run-time are illustrated in Figs. 14 and 15. From Figs. 14 and 15, it can be observed that the FMVO has a superior performance in terms of computation run-time compared to the conventional MVO. Superior performance of

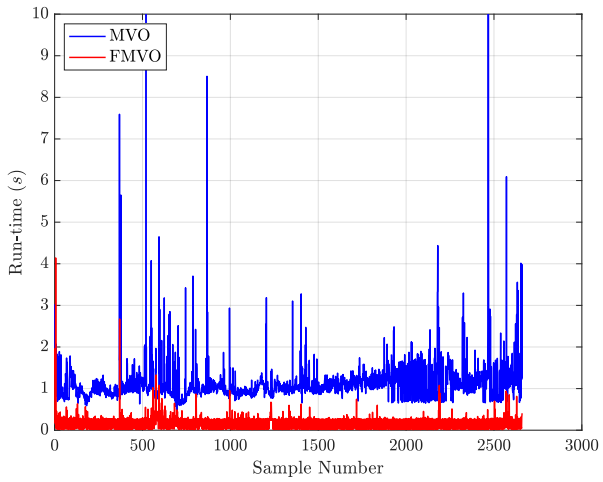


Fig. 14: Loop run-time for MVO and FMVO.

FMVO compared to MVO in terms of update rate is obvious from Figs. 14 and 15.

Remark 1. All codes used in this work are provided in the folder "codes". In order to regenerate the plots shown in Figs. 10 to 15 and the results presented in Table I, the code `plotBothResults.m` should be used. This code uses stored results to make the plots. In addition to this code, there are two folders called "BA2Images" and "BAnImages". These folders contain codes required to run the conventional MVO and the FMVO, respectively. The first step to run these codes is to download the MH_01 sequence of the EuRoC dataset [11]. The downloaded folder "mav0" should be placed in the

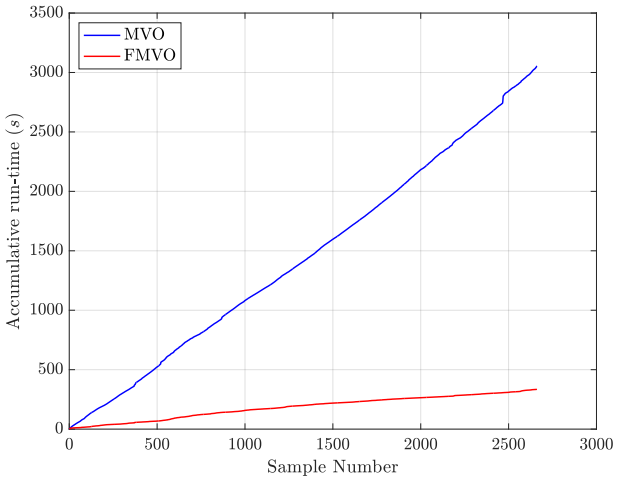


Fig. 15: Accumulative run-time for MVO and FMVO.

"codes" folder. Now, the `runMe.m` code in "BA2Images" and "BAnImages" folders could be used to run the conventional MVO and the FMVO, respectively. The results will be stored and `plotBothResults.m` could be used to generate the plots shown in Figs. 10 to 15.

V. Conclusion and Future Work

In this project, the detailed design of a Monocular Visual Odometry (MVO) pipeline has been studied. In addition, to reduce the inherent computational cost problem associated with the conventional MVO, a novel approach, referred to as Fast MVO (FMVO), has been developed. The superior performance of the proposed FMVO approach compared to the conventional MVO has been verified via implementation on the EuRoC dataset. It was observed that the proposed approach offers a 20.6 times faster update rate, at the expense of a slight increase in orientation estimation error, while offering a comparable performance in position estimation.

Future directions for research include integrating other sensory information with the camera to overcome the scale ambiguity problem and further enhance the performance of the proposed method.

References

[1] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[2] M. Maimone, Y. Cheng, and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *Journal of Field Robotics*, vol. 24, no. 3, pp. 169–186, 2007.

[3] H. Romero, S. Salazar, O. Santos, and R. Lozano, "Visual odometry for autonomous outdoor flight of a quadrotor uav," in *2013 international conference on unmanned aircraft systems (ICUAS)*. IEEE, 2013, pp. 678–684.

[4] M. Dunbabin, J. Roberts, K. Usher, G. Winstanley, and P. Corke, "A hybrid auv design for shallow water reef navigation," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005, pp. 2105–2110.

[5] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry for ground vehicle applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 3–20, 2006.

[6] M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, pp. 1–26, 2016.

[7] M. Aladem and S. A. Rawashdeh, "Lightweight visual odometry for autonomous mobile robots," *Sensors*, vol. 18, no. 9, p. 2837, 2018.

[8] L. R. Agostinho, N. M. Ricardo, M. I. Pereira, A. Hiolle, and A. M. Pinto, "A practical survey on visual odometry for autonomous driving in challenging scenarios and conditions," *IEEE Access*, vol. 10, pp. 72 182–72 205, 2022.

[9] K. Konda and R. Memisevic, "Learning visual odometry with a convolutional network," in *International Conference on Computer Vision Theory and Applications*, vol. 2. SciTePress, 2015, pp. 486–490.

[10] V. Mohanty, S. Agrawal, S. Datta, A. Ghosh, V. D. Sharma, and D. Chakravarty, "Deepvo: A deep learning approach for monocular visual odometry," *arXiv preprint arXiv:1611.06069*, 2016.

[11] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract

[12] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, vol. 2, 1981, pp. 674–679.

[13] R. Hartley, "In defense of the eight-point algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 6, pp. 580–593, 1997.

[14] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[15] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*. Springer, 2000, pp. 298–372.

[16] L. Carlone, R. Aragues, J. A. Castellanos, and B. Bona, "A fast and accurate approximation for planar pose graph optimization," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 965–987, 2014.

[17] J. Qin, M. Li, X. Liao, and J. Zhong, "Accumulative errors optimization for visual odometry of orb-slam2 based on rgb-d cameras," *ISPRS International Journal of Geo-Information*, vol. 8, no. 12, p. 581, 2019.

[18] D. Scaramuzza and Z. Zhang, "Visual-inertial odometry of aerial robots," *arXiv preprint arXiv:1906.03289*, 2019.