École Polytechnique de Montréal

Navigation System

Monocular Visual Odometry for Drone Navigation in Indoor Environment

HALLOUM YANIS

Table of Contents

1 Introduction

Autonomous navigation in indoor environments, hence GPS-denied environment, presents unique challenges for unmanned aerial vehicles (UAVs). In that regard, visual odometry enables a vehicle to estimate its own motion by analyzing consecutive images captured by an onboard camera. Compared to systems relying on multiple sensors, monocular visual odometry — which uses only a single camera — reduces the payload and allow lower energy consumption as well as better simplicity, making it particularly attractive for lightweight platforms such as drones.

This project focuses on the development of a monocular visual odometry system for real-time drone navigation in indoor environments. The objective is to estimate the three-dimensional position and trajectory of a drone, using only the images provided by a single camera mounted on the platform. By tracking visual features across frames and reconstructing the drone's motion, the system aims to provide accurate, real-time localization without external references. This work contributes to the broader field of indoor drone autonomy, where reliable positioning is crucial for tasks such as exploration, inspection, and search-and-rescue operations.

2 EuRoC Dataset

To validate the proposed monocular odometry system, this project uses the EuRoc MAV Dataset. The EuRoc dataset provides images captured by stereo cameras mounted on a drone, along with synchronized ground truth trajectories obtained from a motion capture system and High-rate IMU data. For this project, only the images from a single camera will be used to simulate a monocular setup. The ground truth data will serve to quantitatively evaluate the accuracy of the estimated trajectories.

3 Methodology

In this project, we aim to estimate the motion of a drone equipped with a monocular camera as it navigates through an indoor environment like presented in [4]: *Implementation of a Monocular ORB SLAM for an Indoor Agricultural Drone*, Kanjanapan Sukvichai, Noppanut Thongton, Kan Yaja. To achieve this, we adopt a classical visual odometry pipeline based on feature detection, feature matching, motion estimation, and 3D point triangulation. Throughout this report, the terms *robot*, *drone*, and *camera* will refer to the same entity.

3.1 Reference Frames

We define the following coordinate frames:

- **World frame** (w): A fixed inertial reference frame in which the drone's trajectory is expressed.
- **Camera frame** (c): A moving reference frame attached to the camera at each time instant.

- **Image frame** (i): The 2D frame associated with the camera sensor where pixel coordinates are defined.

The goal is to estimate the **motion matrix** between consecutive camera frames A and B. The motion matrix:

$$\mathbf{T}_B^A = \begin{bmatrix} \mathbf{R}_B^A & \mathbf{t}_B^A \\ \mathbf{0}^\top & 1 \end{bmatrix}$$

where $\mathbf{R}_B^A$ and $\mathbf{t}_B^A$ are respectively the rotation matrix and the translation vector from frame A to frame B.

3.2 Notations

The following notations will be used throughout the project:

- $\mathbf{P}^w$: 3D point expressed in the world frame.
- $\mathbf{P}^c$: 3D point expressed in the camera frame.
- $\mathbf{p}^i$: 2D pixel coordinates in the image frame.
- $\mathbf{R}$: Rotation matrix.
- $\mathbf{t}$: Translation vector.

3.3 Pipeline Overview

The visual odometry pipeline is structured into the following main stages:

1. **Feature Extraction**: Identify **keypoints** i.e salient points in each image that will allow the robot pose tracking throughout consecutive frames.
2. **Feature Matching**: Establish **descriptors** i.e correspondences between keypoints detected in consecutive frames.
3. **Pose Estimation**: Estimate the relative pose (rotation and translation) between frames based on the matched keypoints.
4. **Triangulation**: Reconstruct 3D landmarks from multiple observations to refine motion estimation and improve robustness.

Each of these stages will be detailed in the subsequent sections.

4 Feature Extraction

Given a grayscale image, the feature detection procedure is performed as follows.

First, for every pixel p in the image (excluding a 3-pixel-wide border), we extract its intensity value I_p. A circle of radius 3 pixels is then defined around p, and 16 discrete pixel locations are sampled uniformly along this circle. The intensity values of these 16 surrounding pixels are retrieved. [3]

A threshold T is introduced to classify the local contrast around p. We check if there exists a sequence of N *consecutive* pixels on the circle that are either all **brighter** than $I_p + T$, or all **darker** than $I_p - T$. If such a sequence exists, the pixel p is classified as a **feature point**.

The full implementation can be found on my github repository

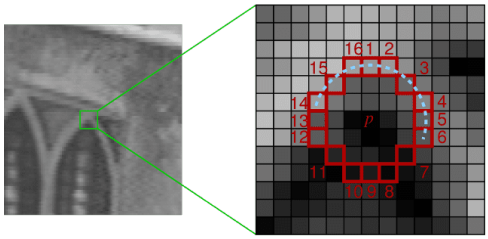

Figure 1: Circle of radius 3 pixels around a pixel https://docs.opencv.org/4.x/df/d0c/tutorial_py_fast.html

The test is typically performed with values $N \in \{9, 11, 12\}$ and threshold T can be tuned for sensitivity.

We also perform a comparative evaluation of feature detection across different parameter settings. We vary $N \in \{9, 11, 12\}$ and threshold $T \in \{10, 20, 30\}$ to observe how feature sensitivity changes. Results are visualized for each configuration, showing the number and distribution of detected keypoints.

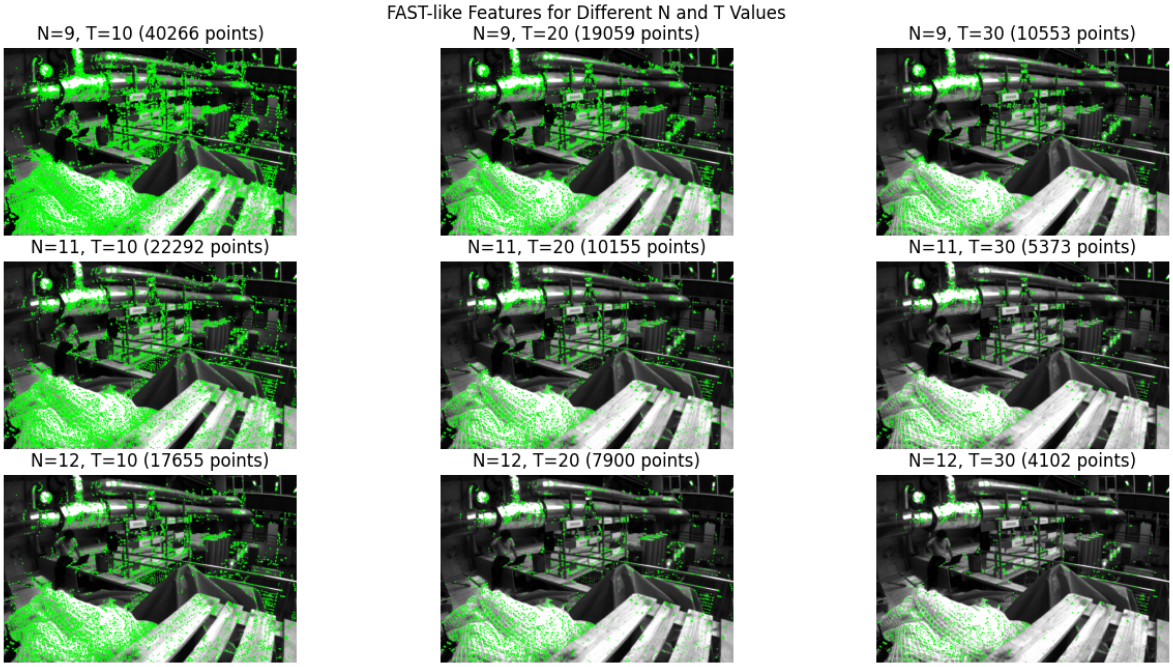

Figure 2: Comparative evaluation of feature detection across different parameter settings

For N and T small, the number of features extracted are the highest. However, now let's analyze the quality of these features for the features matching part.

5 Feature Matching

Once features are detected on two consecutive frames, we perform feature matching based on the ORB (Oriented FAST and Rotated BRIEF) descriptors. The algorithm estimates the orientation of each keypoint to achieve rotation invariance. For a patch centered around a keypoint, image moments are computed as [3]

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

where $I(x, y)$ denotes the intensity at pixel (x, y). The dominant orientation θ of the patch is then derived by

$$\theta = \arctan 2 \left(\frac{m_{01}}{m_{00}}, \frac{m_{10}}{m_{00}} \right)$$

which provides an angle relative to the patch center that aligns the descriptor.

Each binary descriptor is constructed from a fixed number of random point pairs within a square patch centered at the keypoint. Let $p_i^1, p_i^2 \in \mathbb{R}^2$ denote the 2D offsets of the i-th point pair from the patch center (e.g., $p_i^1 = (\Delta x_i^1, \Delta y_i^1)$). These offsets are sampled randomly once and reused across all keypoints.

Given an orientation angle θ assigned to the keypoint, the rotated positions are computed as $R_\theta p_i^1$ and $R_\theta p_i^2$, where

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Each descriptor bit is then defined by intensity comparison as follows:

$$d_i = \begin{cases} 1 & \text{if } I(R_\theta p_i^1) < I(R_\theta p_i^2) \\ 0 & \text{otherwise} \end{cases}$$

The complete descriptor is a binary vector $\mathbf{d} = (d_1, d_2, \ldots, d_n) \in \{0, 1\}^n$, where n is typically 256. To compare two descriptors $\mathbf{d}^{(a)}$ and $\mathbf{d}^{(b)}$, the Hamming distance is used, defined as

$$\text{Hamming}(\mathbf{d}^{(a)}, \mathbf{d}^{(b)}) = \sum_{i=1}^{n} \mathbf{1} \left[d_i^{(a)} \neq d_i^{(b)} \right]$$

where $\mathbf{1}[\cdot]$ is the indicator function. This counts the number of differing bits between the two descriptors (The comparison between the descriptors using the Hamming distance falls under the cv2.BFMatcher() function).

The full implementation can be found on my github repository

In order to determine the optimal parameters for the feature detector, we compared different combinations of the threshold T and the number of consecutive brighter/darker pixels N on two consecutive frames of the EuRoC dataset.

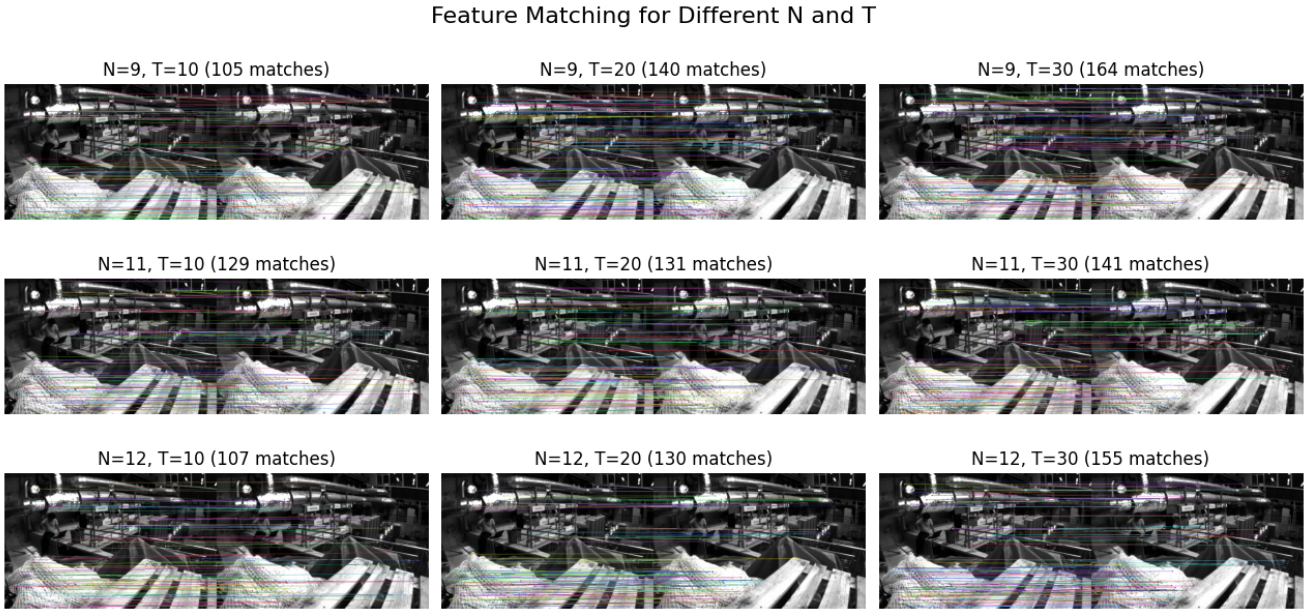

Figure 3: Comparative evaluation of feature matching between two consecutive images across different parameter settings

The metrics considered were **the total number of keypoints detected** (features extracted) and the number of successful matches between the two frames.

The table below summarizes the results:

N	T	Features Extracted	Matches Found
9	10	40266	105
9	20	19059	140
9	30	10553	164
11	10	22292	129
11	20	10155	131
11	30	5371	141
12	10	17655	107
12	20	7900	130
12	30	4802	135

The results show that increasing the threshold T reduces the number of features significantly, but tends to improve the quality of matches (more correct correspondences). Lower values of N (e.g., $N = 9$) extract many more features, which may include noisy or unstable points, resulting in fewer reliable matches. Higher values of N (e.g., $N = 12$) are more selective, leading to fewer features but better match ratios.

The best match count (164) was obtained for $(N, T) = (9, 30)$, but at the cost of detecting over 10,000 features. On the other hand, $(N, T) = (11, 30)$ provides a good trade-off with 5371 features and 141 matches, implying better efficiency in terms of matches per feature.

Hence, for the algorithm we will now consider $(N, T) =$ **(11, 30)**.

6 Camera Pose Estimation

In the previous sections, we discussed feature extraction and matching. These features are used for estimating the camera pose, as they help to identify the position and orientation of the camera with respect to the environment.

Now, let us focus on the process of camera pose estimation, which involves determining the transformation between different coordinate systems: from world coordinates to camera coordinates, and subsequently to image coordinates. In the following sections, we will use "Camera", "Robot", and "Drone" equivalently.

6.1 Notation

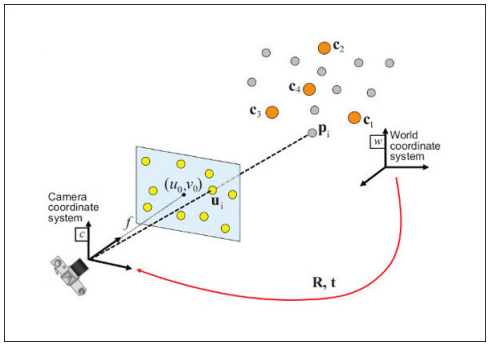

Figure 4: The different frames used https://blog.roboflow.com/pose-estimation-algorithms-history/

Let $P^w = (X_w, Y_w, Z_w)$ represent the coordinates of a point in the world frame, and $P^c = (x_c, y_c, z_c)$ the coordinates of a point in the camera frame. The image coordinates of the point are denoted by $p^i = (x_i, y_i)$, where p^i are the 2D coordinates on the image plane.

6.2 Transformation from World Coordinates to Camera Coordinates

To convert a point from the world coordinate frame to the camera coordinate frame, we use the following transformation:

$$P^c = [R \mid t] \cdot P^w$$

where:

- R is the rotation matrix R_c^w, representing the orientation of the camera with respect to the world frame.
- t is the translation vector t_c^w, representing the position of the camera relative to the world frame.
- $[R \mid t]$ is the extrinsic matrix (a 4x4 matrix) that combines rotation and translation.

This transformation allows us to obtain the 3D coordinates of a point in the camera frame based on its position in the world frame.

6.3 Transformation from Camera Coordinates to Image Coordinates

After capturing an image, the 3D points in the camera coordinate frame need to be projected onto the 2D image plane. This is done using the intrinsic camera matrix K, which encodes the internal parameters of the camera, such as focal length and principal point. The relationship between the 3D camera coordinates and the 2D image coordinates is given by:

$$p^i = K \cdot P^c$$

In this manner, the 3D information captured by the camera is projected onto the 2D image plane, where it can be processed for pose estimation.

6.4 3D reconstruction

In monocular odometry, we estimate motion using two sets of 2D points, with no 3D reconstruction. Epipolar geometry captures the relationship between two different camera views of the same 3D scene.

The goal is to identify rotation R and translation t of B w.r.t A. To do that, keypoints p^A in A frame and p^B in B frame (A and B being consecutive images frames) must project the same 3D point P. The fundamental matrix F defines this relationship [1]:

$$p^{A\top} F p^B = 0$$

The equation becomes

$$W \cdot f = 0$$

- $W \in \mathbb{R}^{N\times 9}$
- f is the vectorized form of the fundamental matrix:

$$f = \begin{bmatrix} F_{11} & F_{12} & F_{13} & F_{21} & F_{22} & F_{23} & F_{31} & F_{32} & F_{33} \end{bmatrix} \top$$

To estimate the fundamental matrix F, we solve the homogeneous linear system $Wf = 0$, where $W \in \mathbb{R}^{N\times 9}$ is constructed from $N \geq 8$ point correspondences, and f is the 9-element vector formed by flattening F.

Once we have F, we can determine E with :

$$E = (K^{-1})^\top F K$$

with K the camera's intrinsic matrix

$$K = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

- f_x: focale dans la direction horizontale
- f_y: focale en pixels dans la direction verticale
- u_0: coordonnée x du point principal dans l'image

- v_0: coordonnée $y$du point principal dans l'image

The values are defined in the sensor.yaml file in the EuRoc dataset :

$$K = \begin{bmatrix} 458.654 & 0 & 367.215 \\ 0 & 457.296 & 248.375 \\ 0 & 0 & 1 \end{bmatrix}$$

Finally, we can determine R and t with :

$$R \cdot t = E$$

Given the essential matrix E, the relative rotation R and translation t between frames are extracted as follows. We perform an SVD on E:

$$E = U\Sigma V^\top$$

If $\det(U) > 0$ and $\det(V) > 0$, the rotation matrix R is computed as:

$$R = UWV^\top$$

where

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If the trace of R is negative, $W^\top$ is used instead. The translation vector t is taken as the third column of U. Finally, the full 4x4 homogeneous transformation matrix T is built by combining R and t.

The full implementation can be found on my github repository (For feature extraction and matching, OpenCV is used as it is compilated using C/C++, hence highly more efficient than the algorithms I designed in the first two sections.

We start by initializing the camera position at the origin. For each subsequent frame, the camera's position is updated using the rotation matrix R and the translation vector t:

$$\text{new\_position} = R \times \text{current\_position} + t$$

We then get this 2D estimation of the camera pose:

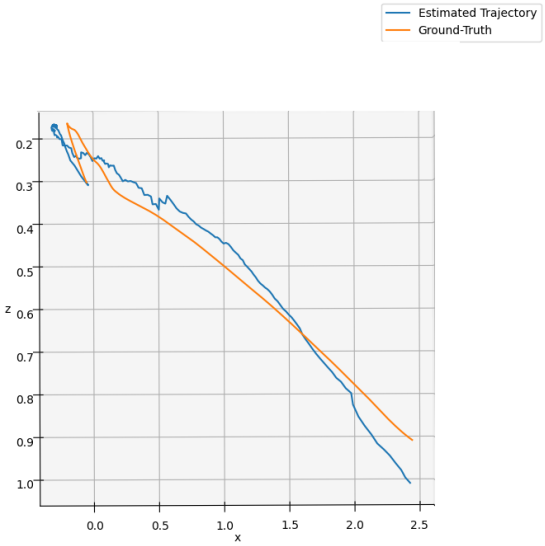

Figure 5: 2D camera pose estimation compared to ground truth

7 Triangulation

After the *Camera Pose Estimation* step, we obtain the relative motion of the camera between consecutive frames. In order to recover the 3D structure of the scene, we rely on a process called **triangulation**.

Let us denote by $p^A = (x_A, y_A)$ and $p^B = (x_B, y_B)$ the pixel coordinates of a feature point as observed in two consecutive image frames A and B. These observations correspond to the projection of a common 3D point P^w in the world coordinate system.

The relation between the 3D point P^w and its 2D projection p^A in the first image frame A is given by:

$$p^A = K \cdot \Pi_0 \cdot P^w, \tag{1}$$

where K is the camera intrinsic matrix, and $\Pi_0 = [I_3 \mid \mathbf{0}_{3\times1}]$ is the standard projection matrix.

In the second frame B, which is rotated and translated relative to frame A, the motion is represented by the homogeneous transformation matrix $\bar{T}_B^A$:

$$\bar{T}_B^A = \begin{bmatrix} R_B^A & \mathbf{t}_B^A \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{2}$$

where R_B^A and $\mathbf{t}_B^A$ represent the rotation and translation from frame A to frame B.

The projection of the same 3D point in frame B is given by:

$$p^B = K \cdot \Pi_0 \cdot \bar{T}_B^A \cdot P^w. \tag{3}$$

Since a 2D point and its corresponding projection must lie on the same line of sight, their cross product must be zero:

$$p^A \times (K \cdot \Pi_0 \cdot P^w) = 0, \tag{4}$$
$$p^B \times (K \cdot \Pi_0 \cdot \bar{T}_B^A \cdot P^w) = 0. \tag{5}$$

This gives us a homogeneous linear system of equations of the form:

$$A \cdot \mathbf{x} = 0, \tag{6}$$

where $\mathbf{x}$ represents the unknown homogeneous coordinates of the 3D point P^w.

We will solved this system via Singular Value Decomposition (SVD).

7.1 Triangulation via SVD

As described previously, the projection equations are:

$$p^A = K \cdot \Pi_0 \cdot P^w, \tag{7}$$
$$p^B = K \cdot \Pi_0 \cdot \bar{T}_B^A \cdot P^w, \tag{8}$$

with $\Pi_0 = [I_3 \mid \mathbf{0}_{3\times1}]$ and $\bar{T}_B^A$ being the homogeneous transformation from frame A to B.

First we construct projection matrices:

$$P_A = K \cdot \Pi_0, \tag{9}$$
$$P_B = K \cdot \Pi_0 \cdot \bar{T}_B^A. \tag{10}$$

Then we build the linear system: Using the cross product constraint $p \times (P \cdot P^w) = 0$, we generate two constraints per frame. Let $p^A = (x_A, y_A, 1)^T$ and $p^B = (x_B, y_B, 1)^T$. We derive the following homogeneous linear system:

$$A \cdot \mathbf{x} = 0, \tag{11}$$

where $\mathbf{x} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T$ is the homogeneous 3D point, and the matrix A is given by [1]:

$$A = \begin{bmatrix} x_A P_A^{(3)} - P_A^{(1)} \\ y_A P_A^{(3)} - P_A^{(2)} \\ x_B P_B^{(3)} - P_B^{(1)} \\ y_B P_B^{(3)} - P_B^{(2)} \end{bmatrix}, \tag{12}$$

where $P^{(i)}$ denotes the i-th row of matrix P.

We apply SVD to solve $A \cdot \mathbf{x} = 0$.

$$A = U \Sigma V^T,$$

and the solution $\mathbf{x}$ is the last column of V, which corresponds to the smallest singular value.

Finally we retrieve the 3D point We divide by the fourth component:

$$P^w = \left[\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}\right]^T.$$

The full implementation can be found on my github repository

After applying this algorithm to the 2D points obtained from the camera pose estimation, we get this 3D estimation :

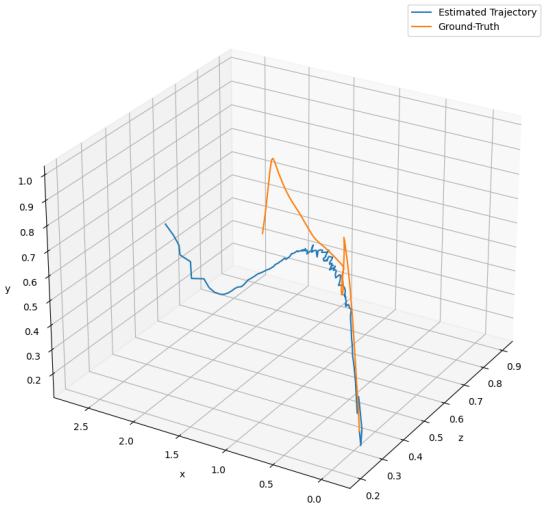

Figure 6: 3D camera pose estimation compared to ground truth

We define the following error metrics to analyze our results :
Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where: n is the number of frames processed, y_i is the ground truth, and $\hat{y}_i$ is the estimated value.
Orientation Error (Angle-Axis):

$$\text{Err}_\theta = \cos^{-1}\left(\frac{\text{Tr}(\mathbf{R}_{\text{true}}^\top \hat{\mathbf{R}}) - 1}{2}\right)$$

where $\mathbf{R}$ are rotation matrices derived from the quaternion estimates.

Thus, we obtained these errors plots :

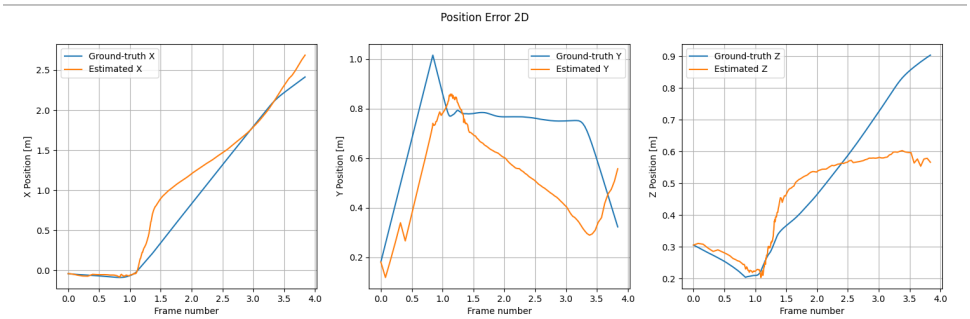

Figure 7: Position Error in every direction

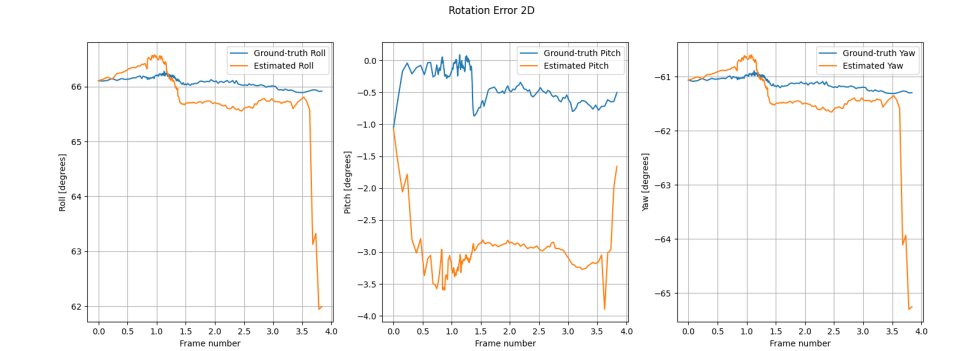

Figure 8: Rotation Error around each axis

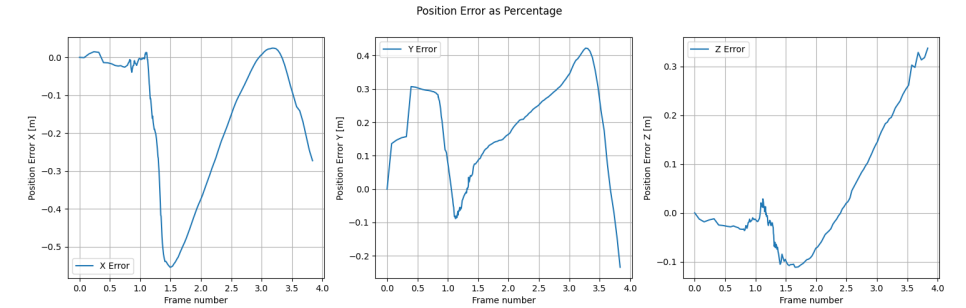

Figure 9: Position error as percentage

Figure 10: Comparison of 3D camera pose estimation and ground-truth

The esimation is close to the ground-truth at first but the errors accumulated at each frame are never compensated. Hence, the estimation diverges from the ground-truth. In this regard, we will now implement an **EKF algorithm**.

8 Extended Kalman Filter (EKF)

The **Extended Kalman Filter (EKF)** is used to estimate the state of a nonlinear system by fusing visual and inertial data in **Visual-Inertial Odometry**. It estimates the **position**, **velocity**, and **orientation**. The implementation of this algorithm is inspired by [2].

The state vector $\mathbf{x}$ is given by:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{q} \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix}$$

where in world frame:

- $\mathbf{p}$ is the **position** vector (3D),
- $\mathbf{v}$ is the **velocity** vector (3D),
- $\mathbf{q}$ is the **quaternion** representing **orientation** (4D),
- $\mathbf{b}_g$ is the **gyroscope bias** (3D),
- $\mathbf{b}_a$ is the **accelerometer bias** (3D).

The **prediction step** uses the motion model to propagate the state based on **IMU measurements**. The state evolves as:

$$\mathbf{v}_k = \mathbf{v}_{k-1} + \mathbf{a}_{\text{world}} \Delta t, \quad \mathbf{p}_k = \mathbf{p}_{k-1} + \mathbf{v}_{k-1} \Delta t + \frac{1}{2} \mathbf{a}_{\text{world}} \Delta t^2$$

The **quaternion update** is:

$$\delta \mathbf{q}_k = \begin{bmatrix} 1 \\ \frac{1}{2} \left(\boldsymbol{\omega}_{\text{meas},k} - \mathbf{b}_{g,k-1} \right) \Delta t \end{bmatrix}, \quad \mathbf{q}_k = \mathbf{q}_{k-1} \otimes \delta \mathbf{q}_k$$

The **biases** evolve as random walks (as defined by the EuRoc dataset):

$$\mathbf{b}_{g,k} = \mathbf{b}_{g,k-1} + \mathbf{w}_{b_g}, \quad \mathbf{b}_{a,k} = \mathbf{b}_{a,k-1} + \mathbf{w}_{b_a}$$

The **Jacobian matrices** are used to propagate uncertainty in the system:

- **State Transition Jacobian** F:

$$F = \begin{bmatrix} I_3 & \Delta t \cdot I_3 & 0 & 0 & 0 \\ 0 & I_3 & \Delta t \cdot R_{\text{b}}^{w} \cdot (-\text{skew}(\mathbf{a})) & 0 & 0 \\ 0 & 0 & I_3 & 0 & 0 \\ 0 & 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & 0 & I_3 \end{bmatrix}$$

- **Process Noise Jacobian** G:

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -R_{\text{b}}^{w} & 0 & 0 \\ 0 & 0 & 0 & -I_3 & 0 \\ 0 & 0 & 0 & 0 & I_3 \end{bmatrix}$$

The **measurement model** relates state variables to the sensor measurements:

- **IMU Measurement**: $\mathbf{z}_{\text{IMU}} = h(\mathbf{x}_k) + \mathbf{v}_k$

- **Camera Measurement**: $\mathbf{z}_{\text{camera}} = \mathbf{K}\left(\mathbf{R}_k^\top(\mathbf{p}_{\text{landmark}} - \mathbf{p}_k)\right) + \mathbf{v}_k$

The **noise models** are defined as: - **Process Noise Covariance** Q_k:

$$Q_k = \text{diag}(\sigma^2_{\text{gyro}}\mathbf{I}_3, \sigma^2_{\text{acc}}\mathbf{I}_3, \sigma^2_{\text{gyro bias}}\mathbf{I}_3, \sigma^2_{\text{acc bias}}\mathbf{I}_3)$$

where the values defined in the dataset are:

- $\sigma_{\text{gyro}} = 0.015\,\text{rad/s}/\sqrt{\text{Hz}}$
- $\sigma_{\text{acc}} = 0.019\,\text{m/s}^2/\sqrt{\text{Hz}}$
- $\sigma_{\text{gyro bias}} = 0.0001\,\text{rad/s}^2$
- $\sigma_{\text{acc bias}} = 0.0002\,\text{m/s}^3$

- **Measurement Noise Covariance** R_k:

- For IMU:

$$R_{\text{IMU}} = \text{diag}(\sigma^2_{\text{gyro noise}}\mathbf{I}_3, \sigma^2_{\text{acc noise}}\mathbf{I}_3)$$

- For Camera:

$$R_{\text{camera}} = \sigma^2_{\text{camera noise}}\mathbf{I}_2, \quad \sigma_{\text{camera noise}} = 1.0\,\text{pixels}$$

In the **update step**, the measurement $\mathbf{z}_k$ is compared with the predicted measurement $h(\mathbf{x}_k)$ to compute the **innovation** $\mathbf{y}_k$:

$$\mathbf{y}_k = \mathbf{z}_k - h(\mathbf{x}_k)$$

The **Kalman gain** K_k is computed:

$$K_k = \Sigma_{k|k-1}H_k^\top(H_k P_{k|k-1}H_k^\top + R_k)^{-1}$$

The state is corrected by the **Kalman gain**:

$$\mathbf{x}_k = \mathbf{x}_{k|k-1} + K_k\mathbf{y}_k, \quad \Sigma_k = (I - K_k H_k)\Sigma_{k|k-1}$$

The full implementation can be found on my github repository

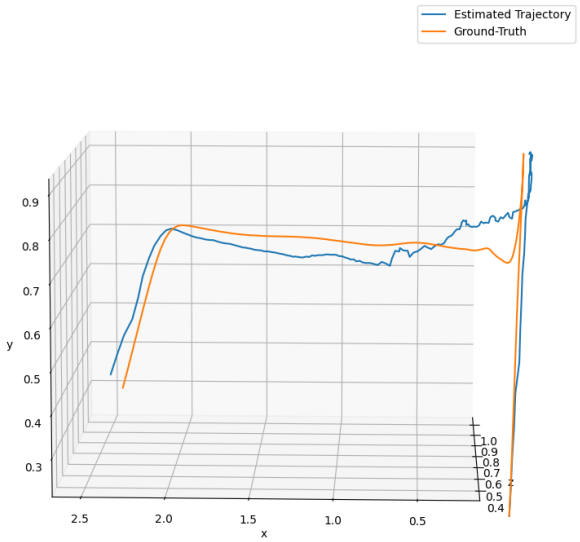

Figure 11: 3D camera pose estimation compared ton ground-truth after EKF

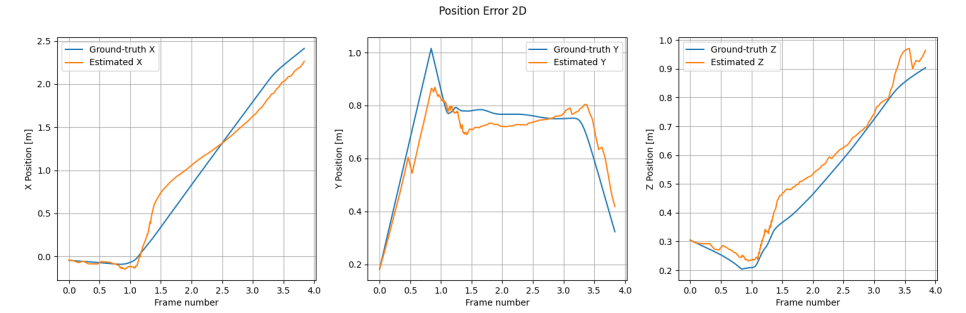

Figure 12: Position Error in every direction after EKF

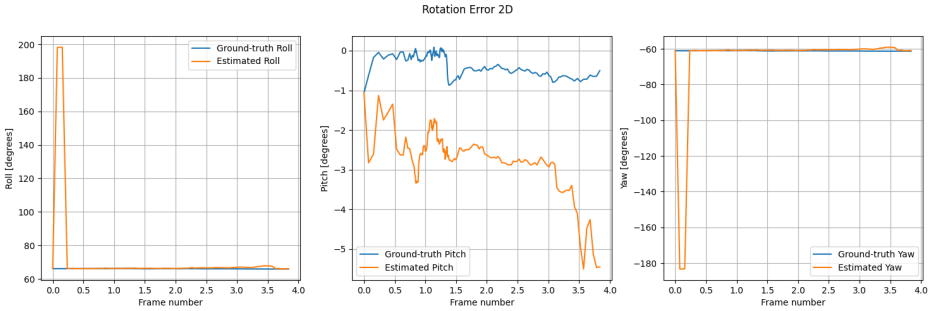

Figure 13: Rotation Error around each axis after EKF

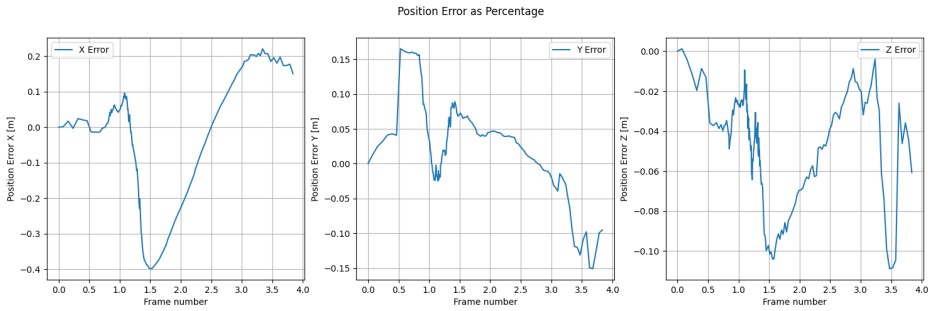

Figure 14: Position error as percentage after EKF

Figure 15: Comparison of 3D camera pose estimation and ground-truth after EKF

We observe that after the use of EKF, the estimated pose is closer to the ground truth in Y and Z directions. However, the rotation errors are similar before and after the EKF implemetation. Moreover the initialization of the EKF algorithm must have have created this peak at the start of the Roll and Yaw estimations.

9 Simulated trajectories

With the ROS quadrator simulator library, we can simulate basic trajectory and add white gaussian noise to the accelerometer and gyroscope. We simulate a basic curved 3D trajectory.

We get this estimation before EKF:

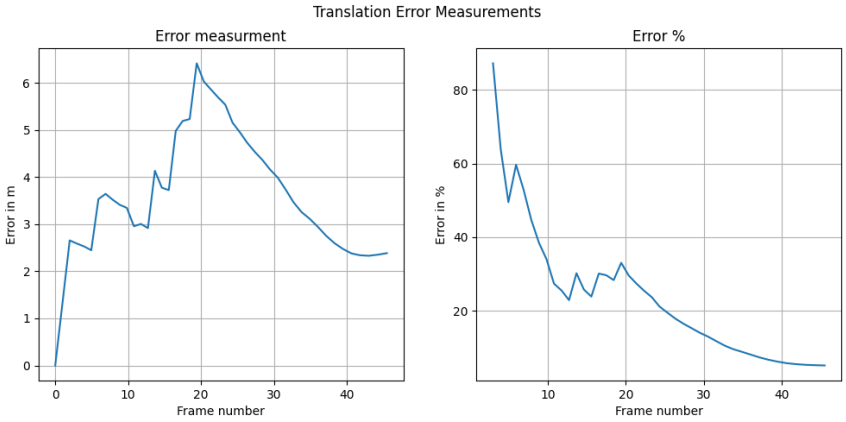

Figure 16: Position Error in every direction before EKF

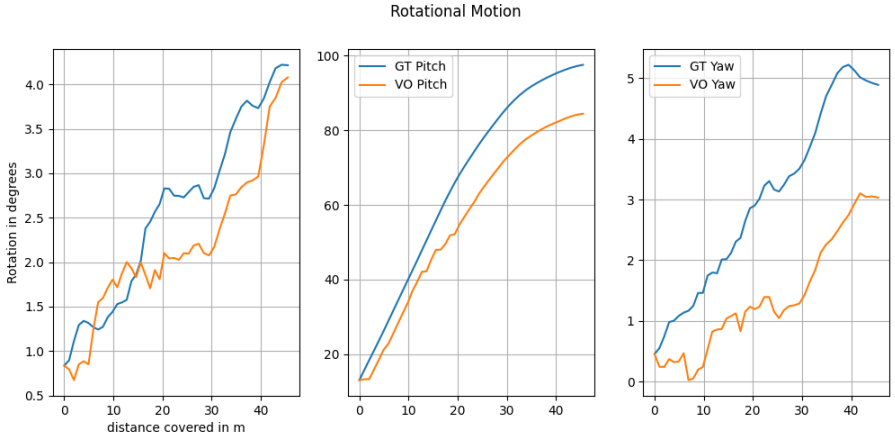

Figure 17: Rotation Error around each axis before EKF

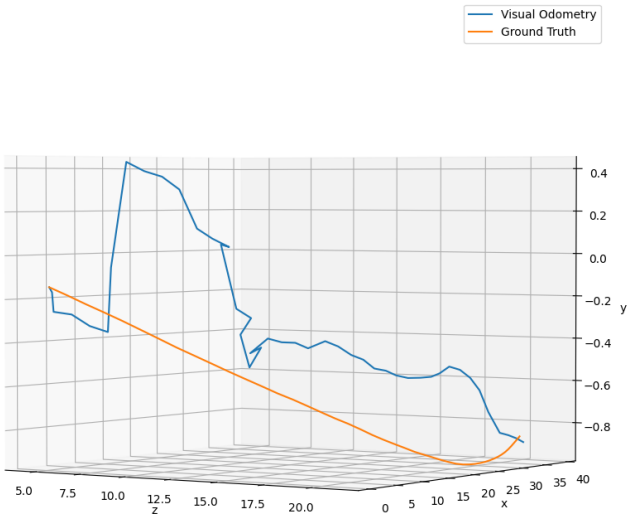

Figure 18: 3D pose estimation comparison before EKF

Figure 19: Comparison of 3D camera pose estimation and ground-truth before EKF

The discrepencies between both estimated and ground-truth trajectories are significant. Hence, this implementation alone is not sufficiant for a precision drone pose esimation purpose.

After the use of EKF, we obtain

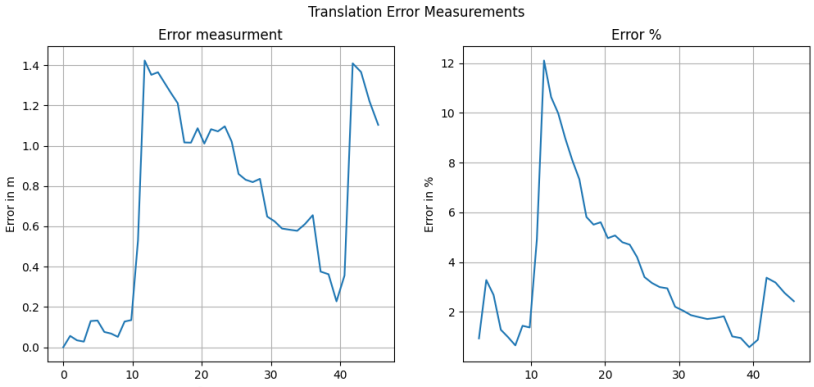

Figure 20: Translation error after EKF

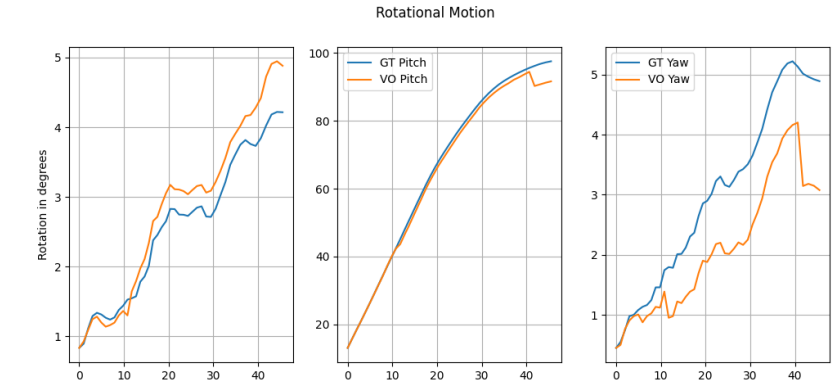

Figure 21: Rotation comparison after EKF

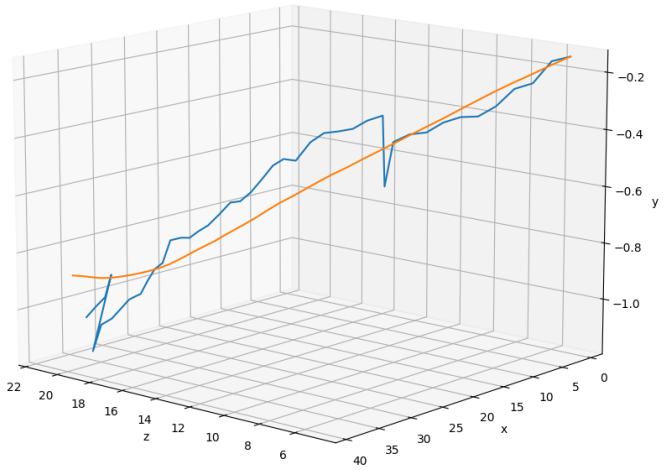

Figure 22: 3D pose estimation comparison after EKF

Figure 23: Comparison of 3D camera pose estimation and ground-truth after EKF

After the EKF algorithm, we can see that the estimation is smoothed and brought

closer to the ground truth.

10 Conclusion

In this project, we implemented a monocular visual odometry system for drone pose estimation in indoor environments, using an Extended Kalman Filter (EKF) to fuse visual information. The developed algorithm shows good performance for simple, basic trajectories; however, for more complex trajectories involving abrupt motions or significant rotations, robustness quickly deteriorates.

Several limitations were identified: the use of monocular vision leads to scale ambiguity (uncorrect depth estimation), the EKF linearization introduces estimation errors, and the system remains sensitive to feature tracking failures under poor visual conditions. Despite these challenges, the project demonstrates the feasibility of monocular EKF-based visual odometry for indoor navigation. Improvements can be made such as better feature management or the implementation of loop closure detection (it would then become a SLAM algorithm).

11 Mention

I authorize this work to be reused.

Références

1. Kenji Hata and Silvio Savarese. *CS231A Course Notes 3: Epipolar Geometrys*,
2. Christopher Doer. *An EKF Based Approach to Radar Inertial Odometry*
3. Ethan Rublee, Willow Garage, Menlo Park, Vincent Rabaud. *ORB: An efficient alternative to SIFT or SURF*
4. Kanjanapan Sukvichai, Noppanut Thongton, Kan Yaja. *Implementation of a Monocular ORB SLAM for an Indoor Agricultural Drone*