Northeastern University

Localization using Visual Inertial Odometry

EECE 5554 Final Project Report

Kaushal Sorte, Anway Shirgaonkar, Neeraj Sahasrabudhe, Girish Raut
Instructor: Prof. Kris Dorsey

Articulation of Application

Visual Inertial Odometry is a technique used to localize the robot with the aid of a stream of images and simultaneous inertial readings from an IMU. Visual Odometry is most known for its application in the 2004 Mars Mission. Methods such as the GPS are prone to effects such as the urban canyon and other problematic environments [1]. Wheel odometry is susceptible to significant drift and inaccuracy because of wheel slippage. Thus, visual odometry is an excellent alternative to wheel odometry, GPS, or pure inertial localization. The accuracy of Visual Odometry (VO) is improved by fusing its localization output with that of an Inertial Measurement Unit (IMU) using state estimators such as Error State Kalman Filters [2].

In this work, we have successfully achieved real-time visual-inertial localization in a straight line and the following sections detail the approaches used for the same.

Approaches

Approach 1

The MATLAB example "Visual-Inertial Odometry Using Synthetic Data" [6] was adopted as a template to achieve visual-inertial odometry. As the name suggests, the example generates its own synthetic camera and IMU data and fuses it using Error State Kalman Filters.

Changes implemented to the above example:

- The models of visual odometry and IMU which gave out synthetic data were deactivated.
- IMU data (acceleration and gyroscope rates) from a VN-100 unit and visual odometry output from the camera were given to the algorithm. Initially, a separate MATLAB script was implemented to estimate odometry based on a monocular camera. The details of the script are discussed below.

Implementation of Visual Odometry in Approach 1

The MATLAB example used an artificial model of a camera with induced noise parameters which provided an estimate of the visual odometry. Our task was to integrate an actual camera and estimate odometry by implementing either of the following algorithms

- **Feature-based approach:** Extracting image features (such as corners, lines, and curves) between sequential image frames, matching or tracking the distinctive ones among the extracted features, and finally estimating the motion [1].
- **Appearance-based approach:** Monitors the changes in the appearance of acquired images and the intensity of pixel information therein instead of extracting and tracking features. The camera speed and vehicle flow can be estimated using optical flow [1].

We initially intended to mount the camera such that it faces the floor. Our hypothesis was that we'll get a uniform and texture-free image which will be suitable for an appearance-based approach. Subsequently, we implemented the Lucas-Kanade method of computing optical flow, but it did not work due to the following reasons:

1. A floor has extremely low texture which results in little to no detectable change in the intensities of pixels and hence the optical flow cannot be calculated
2. A robust method could not be found to accurately convert the obtained optical flow from the units of pixels/frame to m/s

Due to these challenges in the appearance-based approach, we then decided to shift on to feature-based approach. To implement this technique, the following steps were implemented:

1. Capture images I_t, I_{t+1}
2. Undistort the above images using the camera calibration parameters.
3. Use SURF algorithm to detect features in I_t, and track those features to I_{t+1}
4. Compute the essential matrix.
5. Estimate R (Rotation) and t (translation) from the essential matrix that was computed in the previous step.
6. Concatenate the translation vectors, and rotation matrices and estimate position [9]

We tried to implement the above algorithm in MATLAB on real time data from a camera with reference of [10] in which they have implemented monocular visual odometry on synthetic data. However, we were not able to obtain satisfactory results in MATLAB with this approach. The code crashed multiple times while estimating the essential matrix, or was not able to find the optimum solution for the same. Debugging the bundle adjustment process to optimize the solution was beyond the scope of this project. Hence we decided to shift to Python which has abundant open source libraries and literature available to implement monocular visual odometry. A similar algorithm as described earlier was implemented in Python with reference of [11]. The main change is that we shifted to ORB feature detector instead of SURF due to the following advantages:

ORB performs as well as SIFT on the task of feature detection (and is better than SURF) while being almost two orders of magnitude faster. ORB builds on the well-known FAST keypoint detector and the BRIEF descriptor. Both of these techniques are attractive because of their good performance and low cost [12]

The caveat in this approach was the estimation of scaling for the translation vectors which is estimated using triangulation of points. The scale estimation obtained was not consistent with the ground truth. The exact reason for this to happen was not clear, however we still decided to proceed with this approach and used the Error State Kalman Filter to fuse this visual odometry data with IMU data. The results obtained from this sensor fusion can be further improved by tackling the following challenges.

Challenges with Approach 1 (Fusion using state estimator such as ESKF) :

- As discussed, the scaling issue in the feature tracking algorithm thwarted our attempts at a good fusion. Modifying the visual odometry pipeline ourselves is beyond the scope of the project.
- The source code of “Predict” and “Estimate” functions used by MATLAB was not available and therefore could not be tuned to our application.
- The integration of the camera and IMU in real-time was problematic because of platform incompatibility - VO in Python and IMU parsing in MATLAB.

Approach 2 :
ORB Slam 3 library was adopted to implement the real-time fusion for visual-inertial odometry [5] [7]. The fusion of the two sensors is achieved using a keyframe based approach, in which only a few frames are selected, and information in the in-between frames is deleted [7]. The ORB Slam 3 Library allowed us to achieve real-time localization in a straight line. The work [7] is primarily relied upon for this project from here on.

This library was chosen due to its ease of implementation in real-time, which was the primary issue we faced in Approach 1. The complete setup here is in ROS Noetic.

Steps followed for implementing ORB Slam 3 :
1. Tuning/updating the parameters in the YAML file
    - Intrinsic matrix parameters of the camera
    - Radial and tangential distortion values of the camera
    - ORB feature detection parameters
    - IMU noise density and random walk (accelerometer and gyroscope)
2. The IMU and camera drivers were used which published to /imu and /camera/image_raw topics respectively. The Monocular Inertial node subscribes to these topics and sends the information to the algorithm which performs localization.

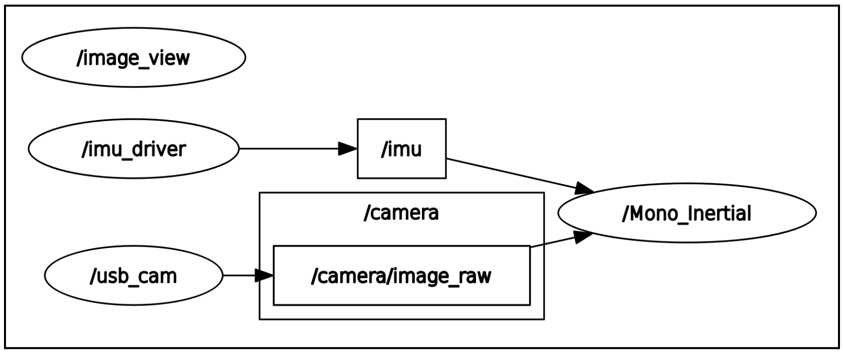

Fig. 1 : RQT graph of system

Sensing Principle + Expected Sensor Performance

The IMU is used to measure the acceleration of the mobile platform at a frequency of 40 HZ. IMU-only dead-reckoning falls prey to errors such as scale factors, axis misalignment, and random walk noise in gyroscope and accelerometer readings, resulting in rapid and significant drifts in just a short amount of time [8].

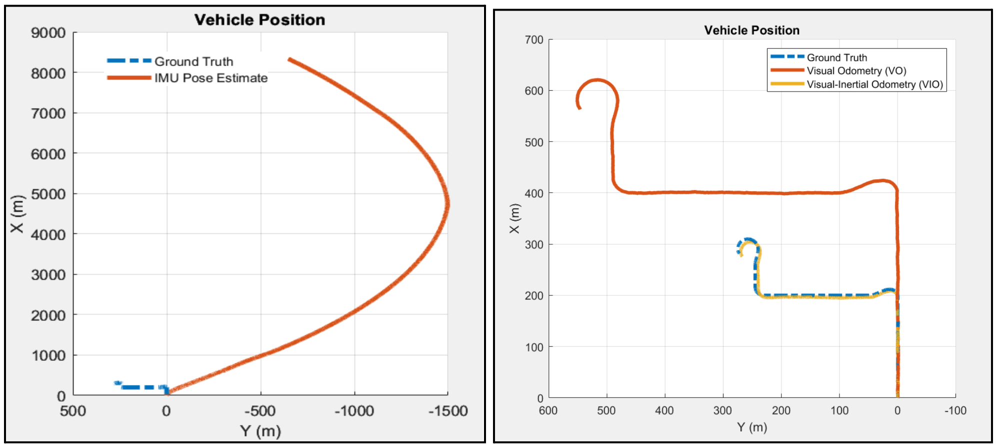

Fig. 2 IMU dead reckoning (left) and of visual inertial odometry (right) - Matlab Example [6] [approach 1]

Thus, it was observed that IMU only dead-reckoning diverged significantly from the true path. The code [5] was modified for IMU-only dead-reckoning.

A monocular camera with a frame rate of 15 fps was used for visual odometry. Vision which operates at a lower frequency serves as a correction for IMU drift - either through filtering or key-frame based approaches. It was observed that the camera works well in straight lines with good features. However, feature detection faces problems during turns.

Dataset Collection

The algorithm was tested on both online and offline datasets. The mobile platform (trolley) was used for emulating a grounded vehicle with a monocular camera and IMU mounted on its top. The offline dataset was collected by recording a bag file consisting of /imu and /camera ROS topics. The dataset was collected in an apartment basement that had multiple high-contrast hanging photo frames on the walls. The area was chosen considering the requirement of distinct unique features of the ORB-SLAM 3 algorithm for localization.

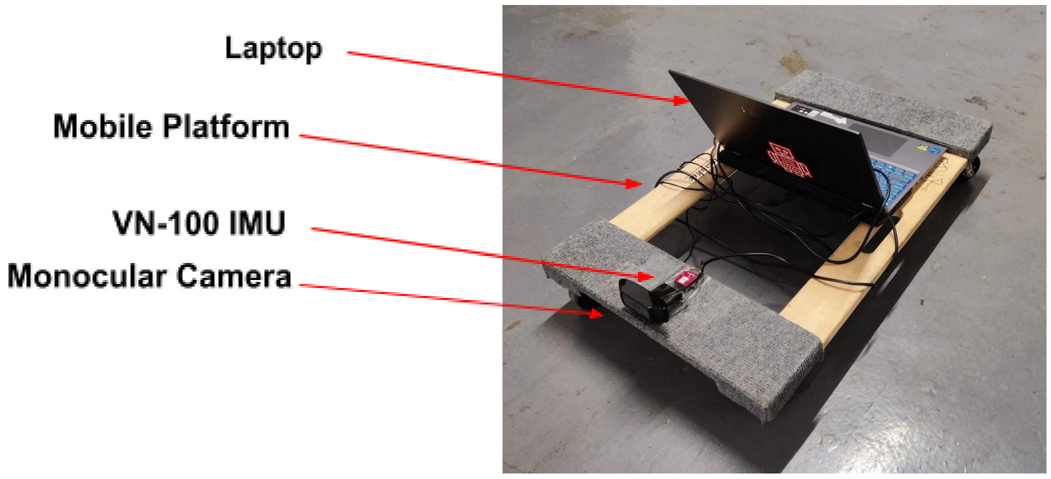

Fig 3: Equipment Setup for Data Collection

Principle behind Algorithm

In Approach 1, we relied on Error State Kalman Filter to fuse the two sensor data. Errors of IMU-integration are collected in an error state in this filter which is predicted parallelly to integration of IMU. Then, using the less-frequent vision data, a posterior estimate of the error is achieved which is subtracted from the nominal state obtained from integration of IMU [4].

Approach 2 saw a complete overhaul of the algorithms used as we shifted from filter-based technique to a keyframe based approach for achieving localization. The process begins with initialization of IMU variables : body velocities, gravity direction and IMU biases. Then the algorithm is nothing but a keyframe based minimization problem which seeks to reduce the errors (residues) from camera and IMU. This optimization needs a good initialization for convergence [7]. As mentioned in [7], for tracking the optimization is achieved by considering the states from only the last two frames. This approach is very different from the state estimation approach using Kalman Filter used in the previous approach.

Final Results

Figure 4 shows the results obtained while performing Visual-Inertial odometry using ORB-SLAM3 algorithm in an outdoor environment.

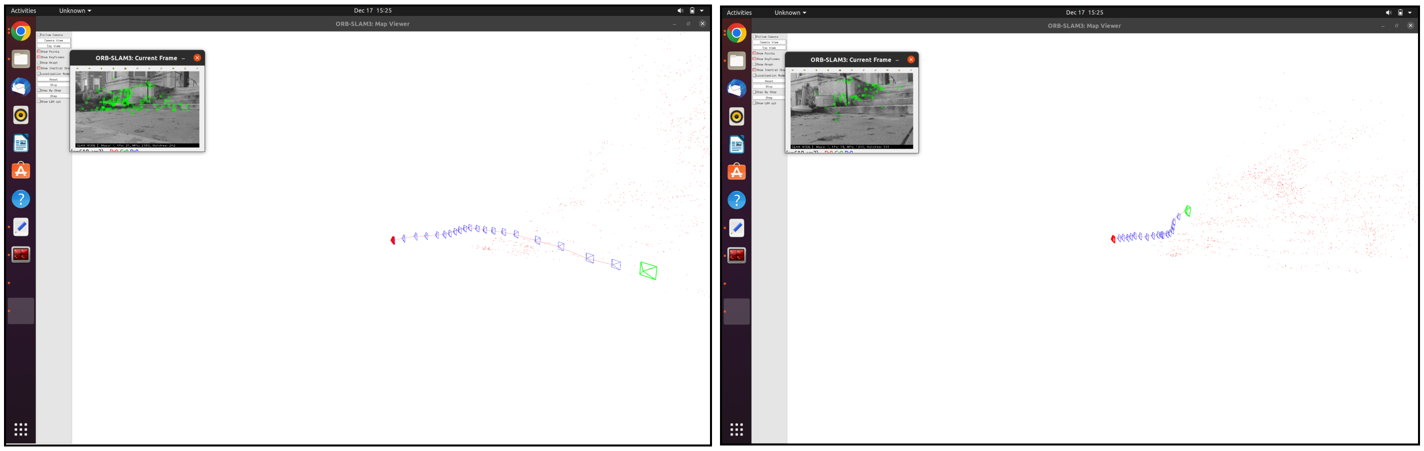

Fig. 4: Output of Visual-Inertial odometry [approach 2]

Concluding Remarks

1. Various methods for trajectory tracking were explored and real time localization of a vehicle traveling in a straight line was achieved.
2. A better result can be obtained by carrying out a comprehensive camera-IMU calibration and tuning the hyperparameters in the ORB Slam3 library
3. The challenges that we have encountered in approach 1 can be tackled by creating an entire pipeline of visual-inertial odometry implementing an Error State Kalman Filter from scratch.

Citations/References

1. Aqel, M.O.A., Marhaban, M.H., Saripan, M.I. et al. Review of visual odometry: types, approaches, challenges, and applications. SpringerPlus 5, 1897 (2016). https://doi.org/10.1186/s40064-016-3573-7
2. D. M. Helmick, Yang Cheng, D. S. Clouse, L. H. Matthies and S. I. Roumeliotis, "Path following using visual odometry for a Mars rover in high-slip environments," 2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720), 2004, pp. 772-789 Vol.2, doi: 10.1109/AERO.2004.1367679.
3. Yousif, K., Bab-Hadiashar, A. & Hoseinnezhad, R. An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics. Intell Ind Syst 1, 289–311 (2015). https://doi.org/10.1007/s40903-015-0032-7
4. Solà, Joan. (2015). Quaternion kinematics for the error-state KF
5. C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM," in IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1874-1890, Dec. 2021, doi: 10.1109/TRO.2021.3075644.
6. https://www.mathworks.com/help/driving/ug/visual-inertial-odometry-using-synthetic-data.html
7. https://github.com/UZ-SLAMLab/ORB_SLAM3
8. Brossard, M., Barrau, A., & Bonnabel, S. (2019). AI-IMU Dead-Reckoning. arXiv. https://doi.org/10.48550/arXiv.1904.06064
9. https://avisingh599.github.io/vision/monocular-vo/
10. https://www.mathworks.com/help/vision/ug/monocular-visual-odometry.html
11. https://github.com/niconielsen32