Real-Time Monocular Visual Odometry on iOS Device

Abhishek Bhatia
Robotics Institute
Carnegie Mellon University
abhatia1@andrew.cmu.edu

Shiyu Dong
Robotics Institute
Carnegie Mellon University
shiyud@andrew.cmu.edu

December 10, 2016

Abstract

Visual Odometry is the process of determining the position and orientation by analyzing the associated camera images. This is an interesting topic to study in robotics and mobile applications. Localization and odometry with mobile devices are widely used in drones, virtual reality and other platforms.

In navigation, odometry is the use of data from the movement of actuators to estimate change in position over time. While useful for many wheeled or tracked vehicles, traditional odometry universally suffers from precision problems, since wheels tend to slip and slide on the floor. Odometry readings become increasingly unreliable over time as these errors accumulate and compound over time. Visual odometry is the process of determining equivalent odometry information using sequential camera images to estimate the distance traveled. [10]

We first implemented the various individual components like feature extraction, feature tracking, essential matrix estimation and, essential matrix decomposition on a linux machine in cpp. We utilized opencv 3.1 packages to implement these components and tested the individual accuracies on both, the KITTI grayscale dataset as well as the dataset we generated. Later on we integrated everything together and developed an opencv based visualizer to analyze our generated trajectories. Finally, once we got everything working, we moved our focus on the second, more important aspect of this project, porting our code as a real-time iOS application and validation.

There are a lot of components involved in building this pipeline and the presence of many such diverse components is what makes this project extremely challenging.

1 Introduction

The idea behind this project was to develop an application for efficient real-time trajectory generation using the monocular visual odometry method. We wanted to utilize the concepts learnt in the class and develop the complete pipeline that includes feature extraction and matching, essential matrix estimation, calculating the rotation and translation to generate the trajectory, and implement everything as a real-time iOS application.

2 Background

As described previously, visual odometry is a very famous technique and hence there are a lot of ways to achieve this, either by using feature based methods, or by using direct methods. For the purpose of this project, we decided to go ahead with the feature based methods and used Avi Singhs blog post as a reference [6]. Besides, we reviewed a lot of opencv packages and papers which have been described in detail in the approach section.

3 Approach

3.1 Image Preprocessing

As mentioned above, we first divided the whole pipeline into individual components and then integrated everything together as part of an iOS application to generate the trajectory in real time. The first and very important component was the image pre-processing.

There were two important procedures we followed as part of the image preprocessing:

3.1.1 Undistortion

To compensate for the lens distortion, we utilized the opencv implementation undistort. This helped us remove the lens distortion and improve the overall performance.

3.1.2 Removing the image blur

Blurring causes error in feature detection and matching. We calculated the variation of Laplacian for every frame and set a threshold to not select a frame when that frame is blurry.

3.2 Feature Detection

The next step in the pipeline was to detect keypoints. To do so, we identified and tested lots of opencv implementations for feature detection (extracting keypoints and determining matches). The two that worked really well for us were based on [2]

3.2.1 FAST algorithm

Features from accelerated segment test (FAST) is a corner detection method, which is used to extract feature points. The most promising advantage of the FAST corner detector is its computational efficiency. FAST is faster than many other well-known feature extraction methods, such as difference of Gaussians (DoG) used by the SIFT, SUSAN and Harris detectors.[8]

3.2.2 ORB (Oriented BRIEF)

Oriented FAST and rotated BRIEF (ORB) is a fast robust local feature detector that is used in computer vision tasks like object recognition or 3D reconstruction. It is based on the FAST keypoint detector and the visual descriptor BRIEF (Binary Robust Independent Elementary Features). Its aim is to provide a fast and efficient alternative to SIFT. Results for the feature extraction can be found in the results section. [9]

3.3 Feature Tracking

Once we have identified the features, we used two different methods to track the features:

3.3.1 Brute-Force Descriptor matcher

Bruteforce matcher is simple. It takes the descriptor of one feature in first set and matches it with all other features in second set using distance calculation. And the closest one is returned. We utilized the opencv implementation for the BF Matcher. [3]

3.3.2 Lucas-Kande based motion analysis and object tracking

This method calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids. We utilized the opencv implementation for this method as well. Results for the motion analysis can also be found in the results section.[4]

3.4 Pose Estimation

Finally, after developing the feature detection and feature tracking components, we worked on estimating the essential matrix and recovering pose from the extracted essential matrix. We reviewed [5] that discusses an efficient approach to estimate the essential matrix from 5-point correspondences and also talks about decomposing the essential matrix to recover the relative rotation and translation between the two frames. [7]

We also reviewed a few other texts solving the similar problem and reviewed the opencv Camera Calibration and 3D reconstruction package which has the implementations for various pin-hole camera methods. We utilized the two below mentioned methods to extract the essential matrix and eventually the relative pose:[1]

- findEssentialMat: This method calculates essential matrix from corresponding points in the two frames.
- recoverPose: This method recovers the relative camera rotation and translation from the computed essential matrix.

3.5 Trajectory Generation

Two camera positions at adjacent time instants $k-1$ and k are related by the rigid body transformation $T_{k-1,k} \in R^{4\times 4}$ with the following form:

$$T_{k-1,k} = \begin{bmatrix} R_{k-1,k} & t_{k-1,k} \\ 0 & 1 \end{bmatrix} \tag{1}$$

where $R_{k-1,k}$ and $t_{k-1,k}$ are the relative pose we recovered from essential matrix in the previous section.

Then, The current camera pose C_k can be computed by concatenating all the transformations $T_{k-1,k}$. Therefore,

$$C_k = T_{k-1,k}C_k \tag{2}$$

We didn't consider any scale factor in our implementation and we are arbitrarily setting C_0, the camera pose at the instant 0 to best display on the iPad screen.

3.6 iOS development

After testing the complete pipeline on KITTI dataset, we moved our focus on the iOS application development. The application we developed is a simple Single-View application where the camera setup is done within the viewDidLoad member function. Frame by frame processing is performed within the processImage member function which is part of the opencvs iOS CvVideoCameraDelegate. The video camera is set to capture frames at a resolution of 640×480 at a frame rate of 30 frames per second. However, to ensure that we see considerable pixel drift between corresponding images, we are processing these at a slower frame rate by ignoring frames in-between. Our final output window displays the pixel movement between the two frames in the upper half, and the generated trajectory in the lower half.

4 Results

4.1 Feature Detection and Tracking

Figure 1 and 2 shows the feature tracking on KITTI dataset using Brute-Force matcher and LK tracker. Figure 3 and 4 shows the feature tracking on images taken from iPad using Brute-Force matcher and LK tracker.

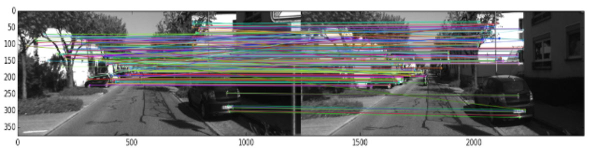


Figure 1: Brute-Force matcher for KITTI dataset

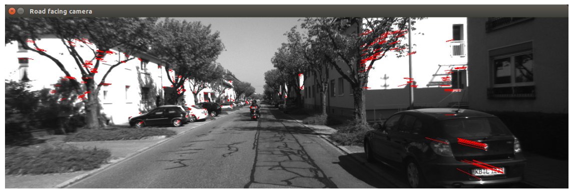


Figure 2: LK tracker for KITTI dataset

4.2 Final trajectory

The videos of final trajectory generation can be found at `https://www.youtube.com/watch?v=gA_KqEO9z-I` and `https://www.youtube.com/watch?v=A7QLlS33MqY`

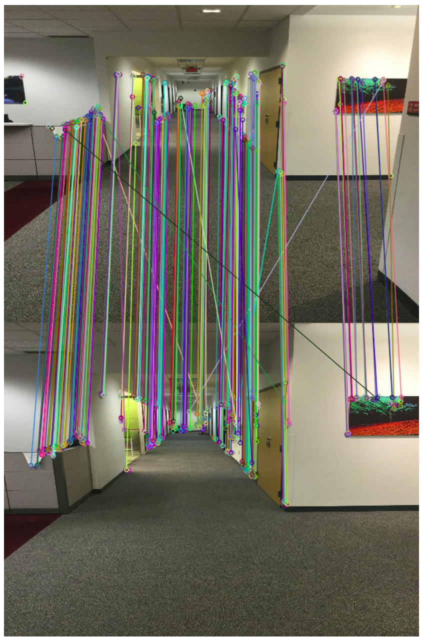


Figure 3: Brute-Force matcher for images taken from iPad

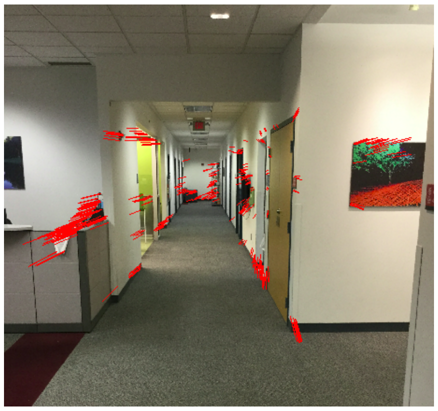


Figure 4: LK tracker for images taken from iPad

Figure 5 shows the final trajectory generated for KITTI dataset. Figure 6 shows the final trajectory generated for real-time camera view from an iPad.

We couldn't generate the ground truth of the trajectory because we were unable to get the scale factor. But from visual interpretation, we were able to generate reasonable and robust trajectory at a FPS of approximately 25.

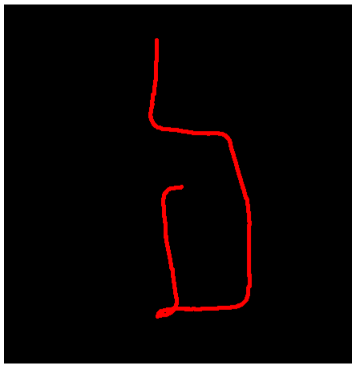
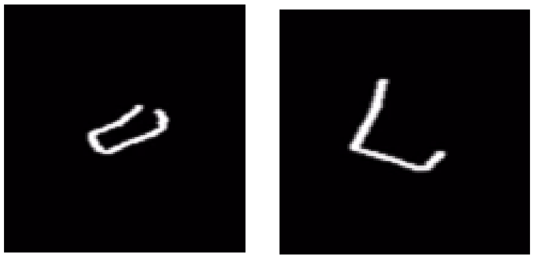


Figure 5: Trajectory for KITTI dataset



(a) NSH A floor (b) NSH 1st floor

Figure 6: Trajectory for real-time camera view from an iPad

5 Challenges and Future Work

There were a lot of challenges in making this pipeline work that includes: proper tuning of the various opencv parameters, proper tuning of the processing frame rate, fluctuations associated with the human movement, featureless environments, e.g, white walls, and error in feature detection because of illumination changes.

There are a lot of things that can be added to this pipeline to make this process more accurate. We could not tackle these as part of this project due to time constraints and hence, we are listing some of these as future work: optimization - use some optimization techniques such as bundle adjustment for trajectory smoothing and loop-closure, scale - use some external metric sensor to induce scale information and estimate absolute scale factor for the generated trajectory, ground truth data generation - for efficient accuracy evaluation, improvement - use the IMU data from the iOS device to improve essential matrix calculation and use direct method instead of features for robust tracking.

6 List of Work

Abhisheks focus: Literature Review, Algorithm Development (Feature Detection and Matching), iOS Application Development and Verification.

Shiyus focus: Literature Review, Algorithm Development (Essential Matrix Estimation based on RANSAC), Image Preprocessing (Undistortion and Removing image blur), iOS Application Verification and Validation.

7 GitHub Page

The GitHub Page of our final project can be found at: `https://github.com/abhishekbhatia1/16623AdvancedComputerVisionApps-FinalProject.git`

Modifications: After the project checkpoint, we made modifications in our core algorithm (feature_matching, pose_estimation), modified the readme to include the results (links to our YouTube videos), and added the 2 Xcode projects we had developed as part of our iOS application.

References

[1] OpenCV 3.0 Documentation. *Camera Calibration and 3D Reconstruction.* URL: `http://docs.opencv.org/3.0-beta/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`.

[2] OpenCV 3.0 Documentation. *Feature Detection and Description.* URL: `http://docs.opencv.org/3.0-beta/modules/features2d/doc/feature_detection_and_description.html`.

[3] OpenCV 3.0 Documentation. *Feature Matching.* URL: `http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html`.

[4] OpenCV 3.0 Documentation. *Motion Analysis and Object Tracking.* URL: `http://docs.opencv.org/3.0-beta/modules/video/doc/motion_analysis_and_object_tracking.html`.

[5] David Nistér. “An efficient solution to the five-point relative pose problem”. In: *IEEE transactions on pattern analysis and machine intelligence* 26.6 (2004), pp. 756–770.

[6] Avi Singn. *Monocular Visual Odometry using OpenCV.* URL: `https://avisingh599.github.io/vision/monocular-vo/`.

[7] Lunds University. *Camera Computation and the Essential Matrix.* URL: `http://www.maths.lth.se/matematiklth/personal/calle/datorseende13/notes/forelas6.pdf`.

[8] Wikipedia. *Features from accelerated segment test.* URL: `http://docs.opencv.org/3.0-beta/modules/features2d/doc/feature_detection_and_description.html`.

[9] Wikipedia. *Oriented FAST and rotated BRIEF.* URL: `https://en.wikipedia.org/wiki/Features_from_accelerated_segment_test`.

[10] Wikipedia. *Visual odometry.* URL: `https://en.wikipedia.org/wiki/Visual_odometry`.