Stereo and Monocular Visual Odometry

Utkarsh Sinha
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
usinha@andrew.cmu.edu

Jai Prakash
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
jprakash@andrew.cmu.edu

Abstract

In this project, we localize the camera using visual odometry. The major component of the project is to generate keyframes according to pre-defined heuristics and triangulate the points to create 3D reconstruction of the scene. The poses of the intermediate frames can be found using Perspective-n-point algorithm. In addition, we also perform local bundle adjustment over last few frames so that the localization is locally consistent. The stereo mononocular performs the best and is close to the ground-truth when compared to the monocular visual odometry. We also plan to exploit the onboard inertial sensors to get prior for the localization.

1 Introduction

Augmented reality has been around for years, yet not all problems are solved in the domain. One of the challenges is precise localization of the device in global coordinates. Augmented reality applications on smartphones are often based on markers. One good example of marker-based AR is Vuforia. On the other hand, there are standalone devices like Hololens, which has number of sensors to understand the scene and localize the head mounted display in the scene. Hololens like devices are globally consistent and performs full-fledge localization and mapping. However, we don't need to understand the scene in most of the cases.

There are applications where understanding the scene is not critical. Localizing the camera in the world is enough to solve certain problems. In this project we focus on localizing the camera using a sequential set of images.

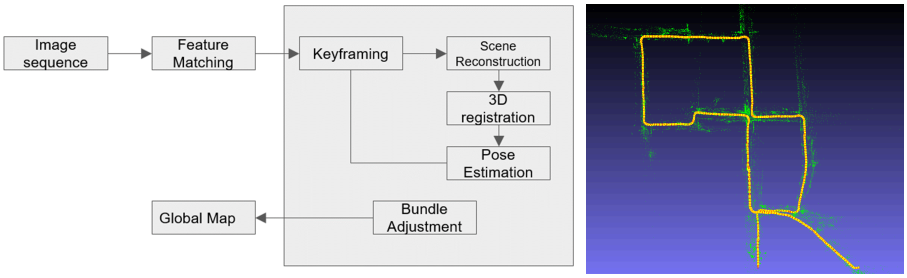
2 Background

The localization and mapping has been of great interest to reserachers over past three decades. The computer vision community has called them as structure from motion and the robotics community has be tackling the same problem using simultaneous localization and mapping (SLAM).

SFM: In structure from motion algorithms, the goal is to generate the scene structure and localize camera positions for each image. The images are in no particular order and can be taken after several timesteps. This is usually an offline process and takes hours to get output and this is not intended to operate real-time.

Visual SLAM: The focus in the visual SLAM techniques is both in reconstructing the scene and also localizing the camera in the scene. One distinction between SFM and SLAM is that the input images are sequential. This provides us with both challenges like short baselines and additional constraints on the system for faster optimization. Such system also aim to produce dense reconstructions and operate real-time.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.



(a) (b)

Figure 1: (a) Overall system architecture. Boxes on the right are what we implemented in this project (b) Results from visual odometry on one of the datasets.

Visual Odometry: The goal of visual odometry is to only localize the camera in sequential images. Any 3D reconstruction is only a side effect of the main problem of localization. Usually, only sparse feature matches are enough to solve this task adequately.

For this project, our main focus is on localizing the camera in 3D space. We focus only to be locally consistent. So our system's camera position might drift over time - however, we are only interested in accurate localization in a short timespan on the order of a few seconds. We do not explore ideas like loop closure in this project. Our work is inspired by the work of Nister et al [**?**].

3 System

Our overall system consists of multiple algorithms as described in Figure 1a. We developed most of the algorithms from scratch and experimented and found the best parameters for each block in the system diagram. Our implementation uses Opencv[**?**] for computer vision tasks and Ceres solver[**?**] for the non-linear optimization.

- We take a sequence of images and find good feature matches between two frames.
- Each frame is tested against few heuristics to decide if it is a keyframe or not.
- Once keyframes are obtained, then we reconstruct the scene using the key-frames.
- The poses of the intermediate frames (not the key-frames) are found using PnP algorithm.
- Bundle adjustment is then performed using data from last few frames.

We discuss each of the methods in details in the following sections.

3.1 Feature Extraction and Matching

We have experimented with KLT features, AKAZE features and ORB features. The KLT features can also be used for tracking the features in the subsequent frames using optical flow. For AKAZE[**?**] and ORB features, the correspondences are found using feature matching. Experimentally, we found that AKAZE was able to generate good matches and in greater number than the other two approaches for the KITTI datasets[**?**].

To remove outliers from the matches, we used three tests. The first is the **ratio test** which ensures that the feature is discriminative enough. This is accomplished by calculating the ratio of the distances between the best match and the second best match for a feature. A discriminative feature would have a high ratio and thus be a good feature to keep.

The second test is the **symmetry test** where we ensure that matches from image $\mathbf{I}_i$ to $\mathbf{I}_j$ also match in the reverse direction - that is from $\mathbf{I}_j$ to $\mathbf{I}_i$. The third test is the **epipolar test** where we discard matches that do not satisfy the epipolar geometry of the two frames used for 3D reconstruction. Using these three criteria we get rid of the most of the outliers in feature matching. An example of feature matches is shown in Figure 2.

Figure 2: An example of feature matches between two consecutive fraames of the KITTI dataset. These are results after all outliers have be removed.

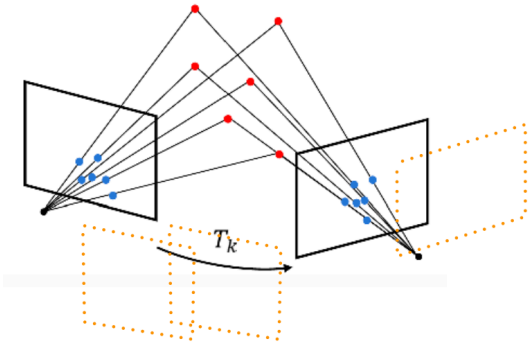


Figure 3: Vizualization of the keyframes and the intermediate frames. The keyframes are used for 3D reconstruction and the pose of the intermediate frames are found using Perpective-n-point algorithm

3.2 Keyframing

3D reconstruction is expensive and thus we want to limit it to only a few frames. This is done using keyframing where only specific frames are picked for 3D reconstruction and other frames use the point cloud generated from these frames as shown in the figure 3.

Our first approach was to have every frame be a keyframe - this is expensive to compute since you need to evaluate the feature matches, triangulate points and also recover pose. Our second approach was to have every 5th frame be a keyframe. This approach worked very well for motion along a straight line but drifted significantly when the camera turned. Our third approach and current approach is based on the number of feature matches between the current and previous keyframe. If the count is below a certain threshold (45% in our case), then we trigger creating a new keyframe and the corresponding 3D reconstruction.

Another approach we considered was using a homography based test[**?**]. However, we found that the feature match count approach worked better for our dataset.

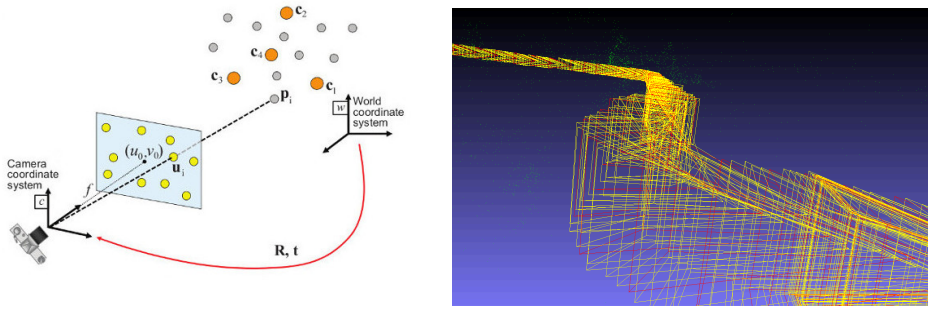
3.3 Three dimensional reconstruction

Using the feature matching, we can triagulate the points. The fundamental matrix gives the relationship between the feature points. We use a RANSAC based 8-point algorithm to find the fundamental matrix.

Using this fundamental matrix, we can estimate the camera pose between the two images. However, this gives rise to four possible camera configurations (with W/W^T and $\pm t$) as described in [**?**]. The correct location can be found using the camera configuration in which all the points are in front of both the cameras.

We use 3D reconstruction in two modes: one with stereo image pairs and one with a monocular images. Reconstruction in both modes was successful. However, as hypothesized, we noticed qualitatively that the monocular 3D reconstruction had more drift compared to the stereo reconstruction.

We run 3D reconstruction only on keyframes. A new keyframe is created everytime the number of matches to the previous keyframe goes below a certain threshold. Currently, we use a threshold



(a) (b)

Figure 4: (a) Using PNP to estimate the camera pose using the global 3D point cloud and the corresponding observations in the image. (b) Three dimensional reconstruction (points in green) and the camera poses corresponding to them

of 45%. This results in a stable reconstruction and allows for registering the different pointclouds together as well.

3.4 Point cloud registration

As an image sequence proceeds, we use the current keyframe and the previous keyframe (or the current keyframe and the corresponding right-eye image in case of stereo) to reconstruct a point cloud based on feature matches. However, the point clouds might be of a different scale and would need to be registered to a global coordinate system.

One approach to calculate the scale of the point cloud is to fit gaussian on the pairwise distances between 3D points. Using the mean of the distances in the local pointcloud and the global pointcloud, we can calculate the scale. Once we have the scale, we can compute the rotation and translation of the point cloud using the common feature points.

To compute the rotation and translation, something here.

$$equation = something$$

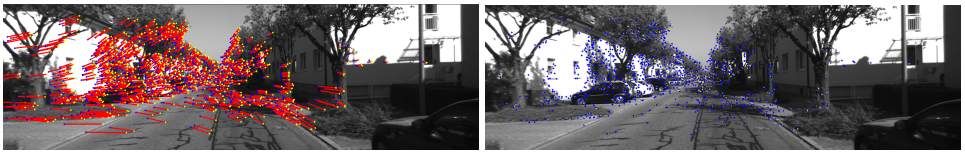
Another approach is to ensure that all projection matrices and transformations occur in one global frame of reference. We assume the first keyframe to be at the origin and chain transformations together to get the current pose.

3.5 Camera pose recovery

Once the scene is reconstructed using the keyframes, the camera pose for normal frames can be recovered using Perspective-n-point (PnP) algorithms. By knowing the 3D points from the reconstruction, and its corresponding feature location in any image the camera pose can be recovered using PnP, as shown in Figure 4a.

We chain these camera poses together to produce an initial guess of the camera trajectory. The trajectory from such chaining drifts as the sequence proceeds and errors accumulate. To account for these errors in the camera projection matrix (and also the point cloud reconstruction), we use bundle adjustment over the past m frames.

$$something = everything$$



(a) (b)

Figure 5: (a) Before bundle adjustment, the red lines show the reprojection error between observed keypoints (yellow) and reprojected points (blue). (b) After bundle adjustment, all reprojected points align very closely to the observed feature points

3.6 Bundle Adjustment

In visual odometry, the current camera pose is obtained by adding the last observed motion to the current detection change. This leads to a superlinear increase in pose error over time. In this section, we look at the techniques we intend to use to correct this pose drift.

One solution is to use bundle adjustment to impose geometrical constraints over multiple frames. The computational cost increases with the cube of the number of frames used for computation. Thus, we limit the number of frames to a small window from the previously captured frames. This approach is called local bundle adjustment.

We use a local bundle adjustment that affects the global point cloud and the recent m camera poses. We track visibility of 3D points in each frame and use that to construct a cost function.

$$\min_{\mathbf{M}_i, \mathbf{X}_j} \sum_i \sum_j ||\mathbf{x}_j^i - \mathbf{M}^i \mathbf{X}_j||_2^2$$

After accounting for visibility of keypoints, our optimization has about fifteen thousand variables. This number depends on the type of image features used (ORB, AKAZE, etc). We use the dense schur method for optimization and use Ceres solver[**?**]. The optimization is able to reduce reprojection error across multiple frames as shown in Figure 5.

4 Results

So far, we have been working with the datasets available online. The results are illustrated on the Middlebury Temple dataset [**?**]. We first find the feature correspondences and then remove the outliers using Epipolar constraints. The outliers can be found by using a threshold on distance from epipolar line and along the epipolar line.

We are able to recosntruct the temple structure using the two keyframes (handpicked for now). The results are shown in figure **??**. Once the reconstruction is done, we are able to recover the camera poses using PnP algorithm as illustrated in the figure **??**. We are using OpenCV for feature matching and visualization.

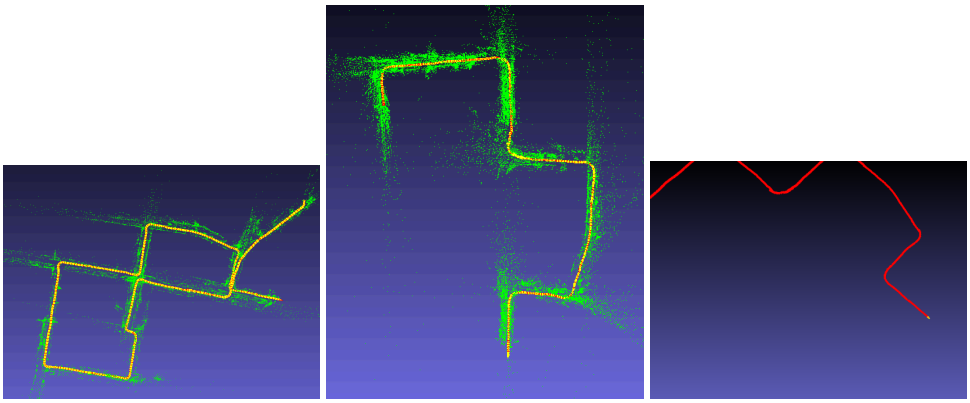
4.1 Stereo visual odometry

In steroscopic visual odometry, we used a stereo pair at every keyframe. The baseline between the stereo pair was about **WHAT** cm and this helped produce very accurate visual odometry.

We used this experiment as a baseline to test how good the monocular visual odometry is.

4.2 Monocular visual odometry

With the monocular case, we used the previous keyframe's image instead of the right eye of the stereo pair. Doing this required writing special code to get the system initialized (the first keyframe would have no 3D reconstruction associated with it). However, once initialized, the system works well and produces the familar trajectory.



(a) (b) (c)

Figure 6: (a) Visual odometry using stereo based reconstruction (b) Using only monocular image pairs (c) Using monocular image pairs and bundle adjustment.

4.3 Monocular visual odometry with bundle adjustment

To improve results for the monocular visual odometry, we enabled bundle adjustment and executed it on the same datasets as in previous experiments. We found that the results were better qualitatively and produce a good trajectory of the camera.

However, the code execution speed slowed down quite a bit. We found that an optimization over the past 10 frames requires about fifteen thousand variables and takes several milliseconds to execute. We currently optimize over the extrinsic matrix and assume that the intrinsics are known and correct. Our hypothesis is that optimizing over the SE(3) manifold would help converge the optimization in fewer iterations and potentially to a better solution as well. This is because it would enforce the rotation matrix to be orthonormal.

5 Future Work

Currently, we calculate the fundamental matrix using the Ransac version of the 8-point algorithm. However, it is possible to improve upon this by using the three-point algorithm. This requires integration with the IMU as an additional clue for estimating the fundamental matrix.

Another area to improve is selecting when to generate a new keyframe. We use the number of keyframe feature matches as the selection criteria. A more mathematically grounded technique would be to use the covariance of the various points to do this.

Our data structures used for storing point correspondences, the 3D point cloud, etc can be made much more memory and complexity efficient. While this wouldn't contribute to computer vision, it would allow us to run experiments faster and better tune the algorithm to be more generic.

Finally, we hope to model the point cloud as probability densities or planes. This would reduce the memory footprint and potentially improve the quality of pose estimation.