Visual Odometry for Self-Driving Car Project Report

Rishabh Biyani
Saimouli Katragadda

May 16, 2017

List of Figures

1 Pipeline

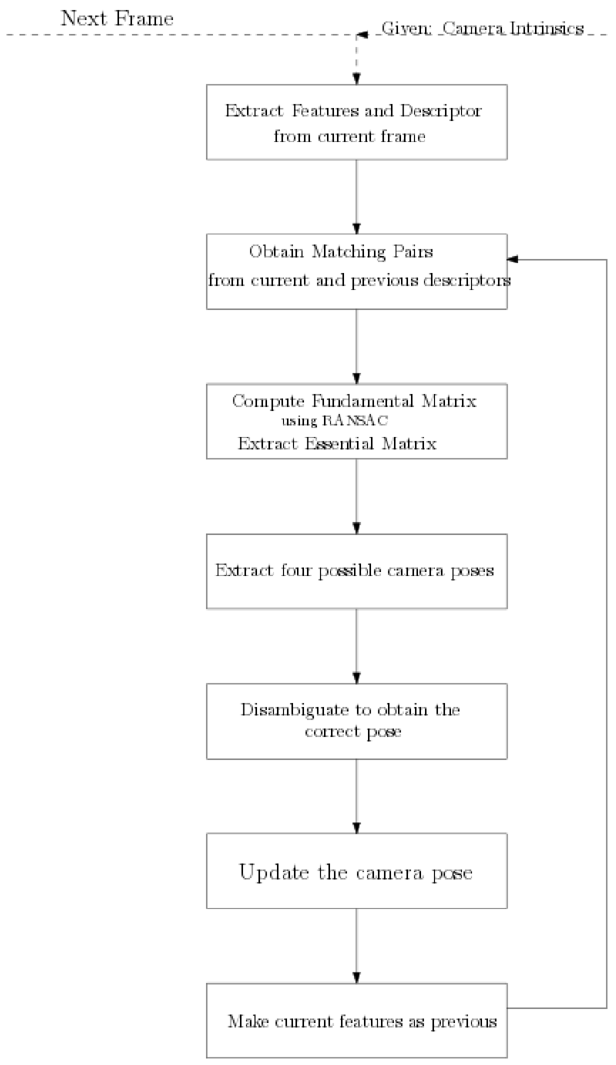

Next Frame
Given: Camera Intrinsics
Extract Features and Descriptor from current frame
Obtain Matching Pairs from current and previous descriptors
Compute Fundamental Matrix using RANSAC
Extract Essential Matrix
Extract four possible camera poses
Disambiguate to obtain the correct pose
Update the camera pose
Make current features as previous

Figure 1: Pipeline used for Monocular Visual Odometry

This project an attempt was made to estimate the position of car from visual cues. Pipeline can be summarized shortly as shown in the figure 1. The first step is to get the camera intrinsics. This information is provided for this project. For each new frame we extract the **SURF** features and form a descriptor for each of these frames. Then we perform a matching between the descriptors of the current frame with that of previous frame. The matching is performed on the basis of thresholding the SSD(sum of square distances) between the descriptors. Hence, this step provides us with all the matching points. The next step is to use these matching points to compute the Fundamental Matrix. **8-point Normalized Algorithm** was used to calculate this matrix. Because of the noisy matching data we use the RANSAC algorithm to sample 8 matching points at a time and get the best matrix following the RANSAC sampling scheme and end conditions. For getting fitness score we check whether the epipolar constraints are obeyed in a threshold value of 3 pixels. Usually, if the matrix is correct, the error value is less than 1 pixel. Essential matrix is extracted using the camera intrinsic. It is well known that we have four possible combinations of R and t that can be obtained from the fundamental matrix. These four combinations are calculated in the next step. For this project, linear triangulation is not performed, thus **cheirality conditions** were **not** used to remove the disambiguity. Rather, we employ some filtering operations based on some knowledge about our camera setting viz. that it can only move forward in X-Z plane and rotation can only happen about the Y-axis. Such series of operations are performed in the block 'Disambiguate to obtain the correct pose'. Finally, we update the global pose of the camera to get the world-position of the camera. Following sections explain some of these blocks in brief.

2 Matching Features

To obtain point correspondences SURF features were used to reduce the computation time. **detectSURFFeatures** and **extractFeatures** are the MATLAB toolbox functions used for finding the features and getting the descriptors respectively. Another toolbox function **MatchFeatures** was used to obtain the matching points between the two images. The matching as mentioned before is performed using SSD metric.

3 Fundamental Matrix computation

Using the matching points, a RANSAC function was implemented to randomly sample 8 set of matching points and checking the score of the resulting matrix with reference to other matching points. The **8-point normalized** algorithm was implemented to obtain the fundamental matrix. . Fitness score is calculated by verifying the epipolar constraints-

$$e = \frac{\mid x_2^T F x_1 \mid}{\sqrt{(Fx_1)_1^2 + (Fx_1)_2^2}}$$

where, e is the perpendicular distance of matching point x_2 from the epipolar line Fx_1. This infact can be thought of as an error. Ideally, the matched point should lie on the epipolar line and the perpendicular distance should be zero. This error is limited to 3 pixels. If matching point error is less than this values we increment the score by 1. Thus, the fundamental matrix gets updated if its score is greater than the earlier score.

Essential Matrix(E) is thus computed after this by using the camera intrinsic and making sure that the rank of the matrix is two by only keeping two singular values in the diagonal matrix of the **svd**.

4 Disambiguate to obtain correct pose

Figure 2 explains the series of filtering steps taken to arrive at the correct pose from the set of possible poses.

- First of all, we select only that set which gives local-Z translation as positive. This assumption makes sense in the current setting, where car is not moving backwards.

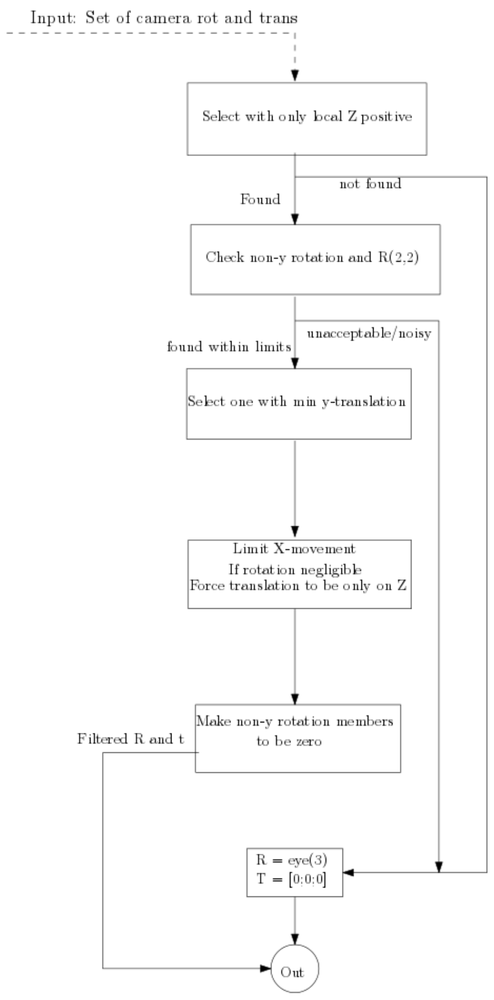
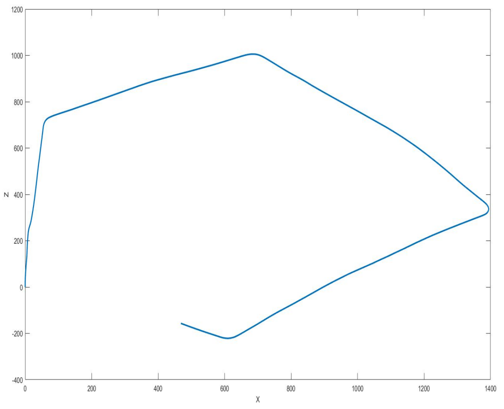

Input: Set of camera rot and trans
Select with only local Z positive
not found
Found
Check non-y rotation and R(2,2)
unacceptable/noisy
found within limits
Select one with min y-translation
Limit X-movement
If rotation negligible
Force translation to be only on Z
Make non-y rotation members
to be zero
Filtered R and t
R = eye(3)
T = [0;0;0]
Out

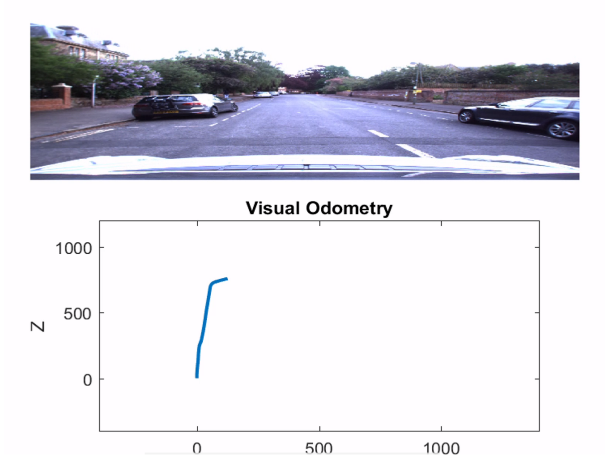
Figure 2: Filtering steps to disambiguate and obtain a valid relative pose

- Then from the remaining candidates, we check for specific values in the rotation matrix. For the current setting we know that only rotation possible is about the y-axis. Thus, we take only those Rotation Matrices which has its R(1,2), R(2,1), R(2,3) and R(3,2) values close to zero with some tolerance. Also, we check that R(2,2) value should be close to one.
- Then if we have more than on candidates after the previous step we take the set with minimum y-translation. Since, only translation in this case is in the X-Z plane.
- Then, if the rotation is negligible(can be checked from R(1,1)), we force local translation to be only about the z-axis.
- Lastly, to obtain further smoother results we make the non-y rotation members as zero.

If any of the checks fail we return an identity rotation matrix and zero translation vector.



(a) Visual Odometry Output



(b) Snap showing plotting process

Figure 3: Final Result

5 Update the Camera Pose

For the current problem the world co-ordinate system is taken to be the first camera pose. That is first camera pose is our zero. This implies that we take our world-Z direction as forward and world-X direction towards the right. The camera poses obtained in the previous step are relative poses and hence have to be mapped to the global frame. Rotation and translation can be mapped using the equations -

$$R_{k+1} = R_k \times^k R_{k+1}$$

$$t_{k+1} = t_k + R_k \times^k t_{k+1}$$

where, k and $k+1$ represent the current and next frame respectively. So, the relative rotation is computed between k and $k+1$ frames. R_{k+1} and t_{k+1} at any instant represent the global position. The problem with using the equation for estimating global translation is that the scale is unknown in the case of monocular visual odometry. The scale has to be recovered using triangulation. For the current project, we estimate translation using this anyway and assume that a global scale is unknown.

6 Conclusion

The final result is as shown in figure 3. The odometry is able to capture the topology of the square loop. The rotation are not accurate to 90 degrees but the geometry resembles a square. Clearly, this basic pipeline is inadequate for the current problem. It is not possible to attain a loop closure and there is a lot of drift as can be seen in figure 3a. **Bundle Adjustment** is a non-linear triangulation technique to reduce the reprojection errors and thereby obtain a more accurate pose estimation. In future we intend to augment our implementation with our own implementation of bundle adjustment.