RBE 549: Computer Vision, Fall 2021

Project Report

VISUAL ODOMETRY USING CLASSICAL COMPUTER VISION

Team Megatron

Member	Signature	Contribution %
Aniket Patil		25
Chinmay Madhukar Todankar		25
Nihal Suneel Navale		25
Prathamesh Bhamare		25

Grading:		
	Approach	/15
	Justification	/5
	Analysis	/15
	Testing and Examples	/15
	Documentation	/10
	Difficulty	/10
	Professionalism	/10
	Presentation	/20
	Total	/100

Contents

List of Figures

Abstract

This project report explains the Visual Odometry pipeline for a Stereo Camera setup, which has been implemented in C++ environment using OpenCV libraries. We have used various KITTI dataset image sequences of stereo camera and documented the results. Our results show that the Visual odometry works well given that features are tracked over the straight path or paths which doesn't have any sharp turns. In case of sharp turns, due to the error accumulation, the drift increases between the actual and estimated trajectory. This can be solved using loop closure which is considered as the back end of SLAM process. The scope of this project is limited to only till Visual odometry implementation i.e., front end of SLAM process.

1. Introduction

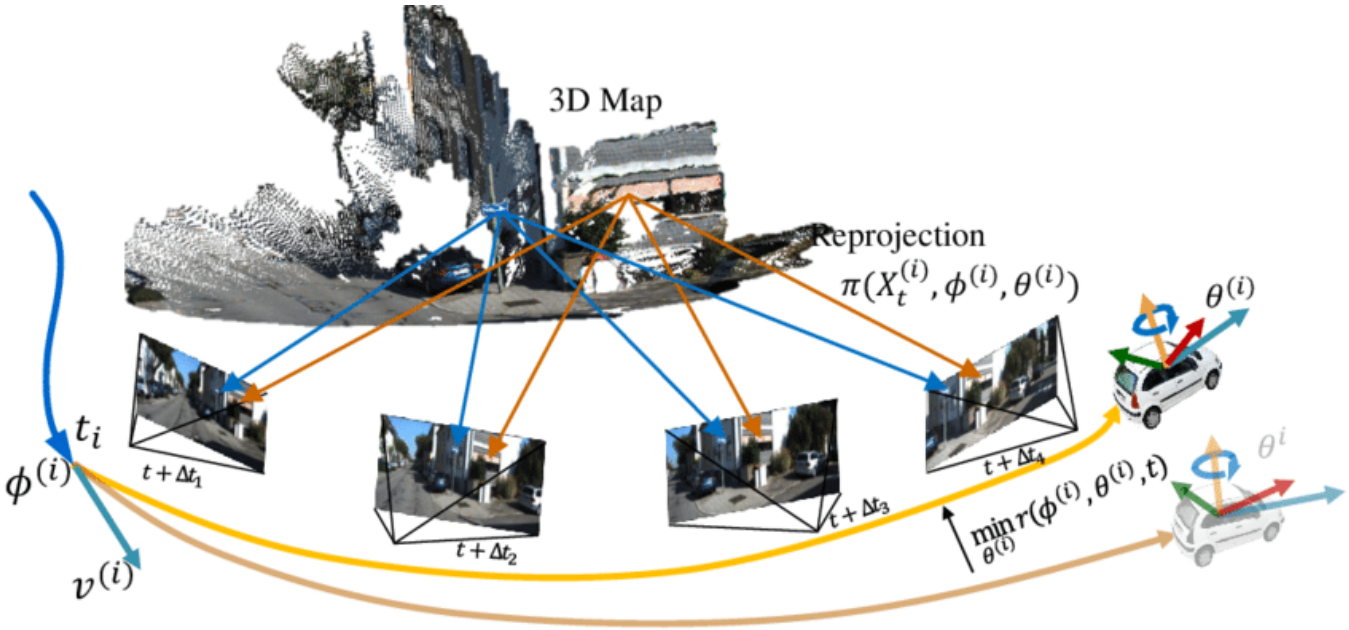

Fig. 1. Visual Odometry using Stereo Camera

Image Courtesy: https://www.researchgate.net/figure/The-illustration-of-our-visual-odometry-framework-The-initial-state-vti-of-current_fig2_332103736

Visual odometry (VO) is the process of determining the position and orientation (ego-motion) of a robot/agent by analyzing images taken from a monocular or stereo camera system attached to the robot/agent. There is a myriad of domains where Visual Odometry can be implemented, a few are Robotics, Augmented Reality, Automotive Industry and Wearable Computers. The term Visual Odometry is like wheel odometry, both solve the same problem i.e., estimate the ego-motion of a robot/agent. In wheel odometry the motion is estimated by integrating the number of turns the

wheel has made over time. Likewise, Visual Odometry operates by estimating the pose of the robot/agent by analyzing the changes that motion induces on the images of its onboard cameras. Visual Odometry requires a sufficiently illuminated environment to perform effectively, with rich textured scene where features can be extracted, and the apparent motion can be estimated. Furthermore, the consecutive frames captured must ensure that they have sufficient scene overlap. Visual Odometry is not affected by wheel slip in uneven terrain, whereas this problem is present in Wheel Odometry. It has been shown that Visual Odometry provides more accurate trajectory estimates, with relative position error starting from 0.1 up to 2% when compared to Wheel Odometry.

2. Methodology

GitHub link of project: https://github.com/aniketmpatil/Visual-Odometry

This project report explains the working and our implementation of various computer vision concepts to achieve Visual Odometry goals. We have used a standard odometry dataset from KITTI, the dataset that we chose contains stereo gray images which is fed as an image sequence to the feature detector. We are using ORB feature detector because it is scale and rotation invariant, and it is also two orders of magnitude faster than SIFT hence making it suitable for real-time applications.

The detected features are then used for feature matching/mapping. In this project we are using FLANN feature matcher. FLANN contains a collection of algorithms optimized for fast nearest neighbor search in large datasets, for high dimensional features and works fast for large datasets.

After keypoint matching, we triangulate the image feature correspondences to get spatial location of 2D points in the camera coordinate system. Then we track the features from left image at time (t) to left image obtained at time (t+1) using LK optical flow.

Using these 3D-2D correspondences, we use PnP to obtain rotation vector and translation vector which brings the points from the world coordinate system to camera coordinate system. Multiplying the translation vector obtained with the inverse of rotation matrix would give the camera position in world reference.

Below given flowchart summarizes our understanding of the implementation of visual odometry pipeline. In the subsequent sections we have explained each process/methods/function in brief details.

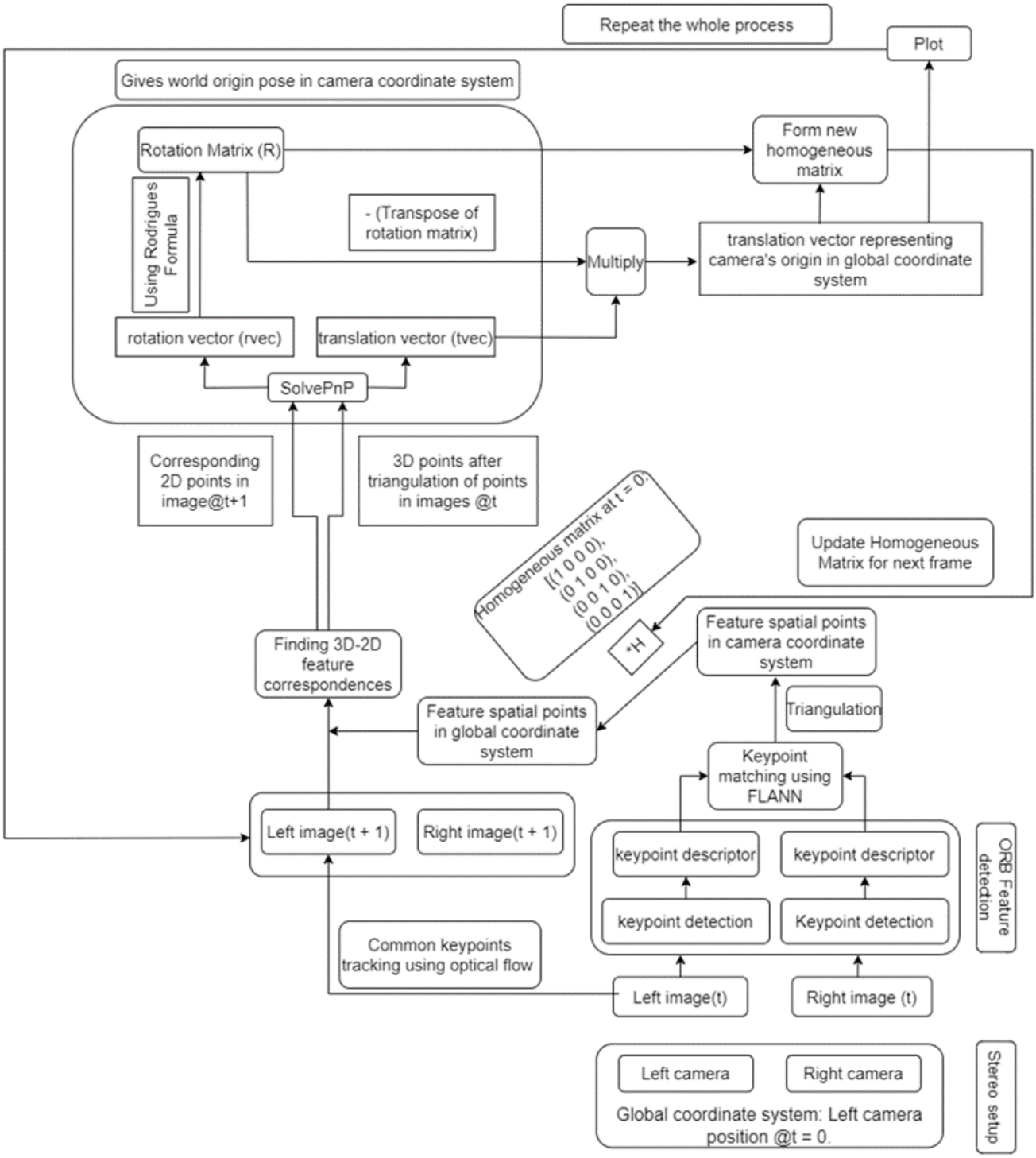

Fig. 2. Flowchart for Stereo Visual Odometry Pipeline

2.1. The KITTI Dataset:

The KITTI Vision Benchmark Suite [1] contains a collection of vision tasks built using an autonomous driving platform. The full benchmark contains many tasks such as stereo, optical flow, visual odometry and many more. This benchmark suite contains the object detection dataset, including the monocular images and bounding boxes.

For the implementation of Visual Odometry we are using grayscale image sequence available on the website. (http://www.cvlibs.net/datasets/kitti/eval_odometry.php)

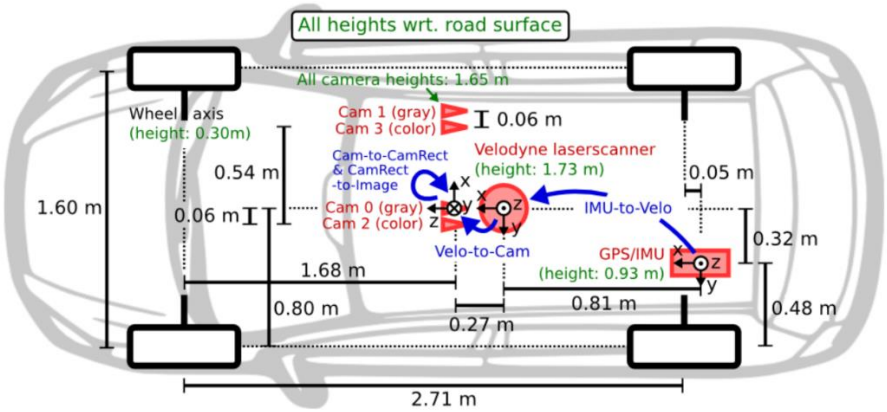

Fig. 3. Setup of the Stereo Camera on top of the test car
Image Courtesy: http://www.cvlibs.net/datasets/kitti/images/setup_top_view.png

In Fig. 3. We can see the detailed setup of the stereo camera on top of the test car. Cam 0 and Cam 1 give the grayscale image sequence of the left and right cameras respectively. Cam 2 and Cam 3 give the RGB image sequence of the left and right cameras respectively. The baseline distance of the stereo camera setup is 0.54m.

Fig. 4. This figure shows the fully equipped vehicle
Image Courtesy: http://www.cvlibs.net/datasets/kitti/images/passat_sensors_920.png

In Fig. 4. We can see the fully equipped vehicle with video cameras, laser scanner and GPS/IMU sensors. The frames of the respective sensors are also shown in the above figure.

Henceforth, left camera position on the car at time t = 0 sec will be referred as world coordinate system.

2.2. Feature Detection: ORB

Feature matching is one of the foundational concepts in computer vision and is frequently used to solve many computer vision problems such as object detection and matching. In our project we are implementing ORB feature detector [4]. ORB (Oriented Fast and Rotated Brief) is rotation invariant and is resistant to noise and is also two orders of magnitude faster than SIFT [6]. ORB is built on the two well-known FAST keypoint detector and BRIEF descriptor.

A. FAST keypoint detector:

FAST is an abbreviation for **Features from accelerated Segment Test**. It is a corner detector method which is used to track and map objects. The most prominent advantage of the FAST keypoint detector is its computational efficiency hence making it suitable for real-time image/video processing applications.

B. BRIEF descriptor:

BRIEF stands for **Binary Robust Independent Elementary Features**. In this method they use binary strings as an efficient feature point descriptor. BRIEF is fast to build, to match and has higher recognition rates (if invariance to large in-plane rotations is not a requirement). These pros mean that real-time matching can be achieved even on devices with low computational power with good performance.

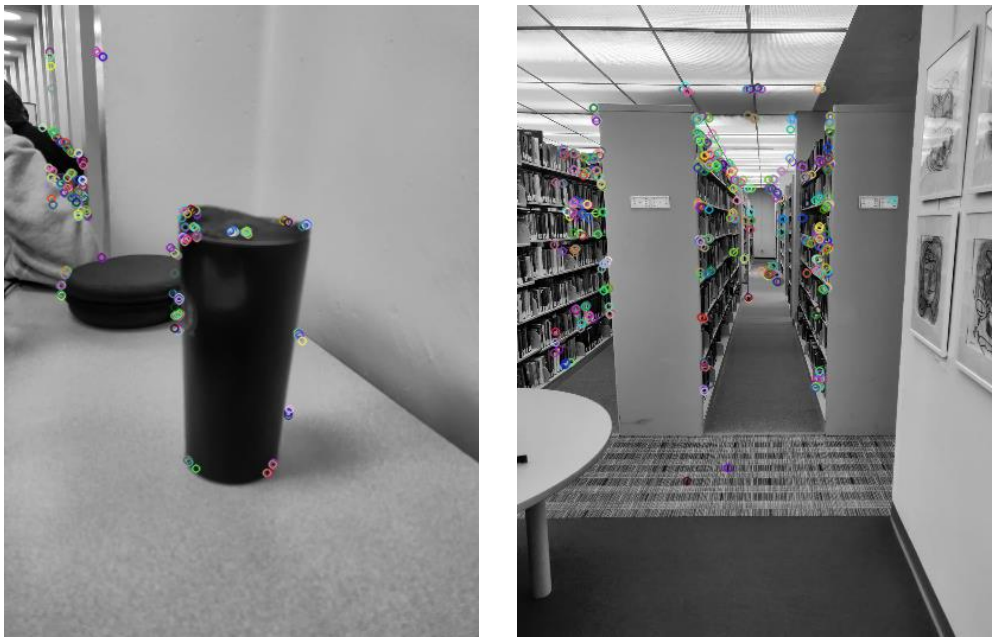

Fig. 5. Detected Feature Points using ORB Feature Detector

2.3. Feature Matching: FLANN

After detecting the features, the next step is to match the features from the left image to the features of right image. To achieve this, there are two matchers that are widely used.

A. Brute Force Matcher:

Brute Force Matcher, as the name suggests, matches all the key point descriptors of left images with the descriptors of the right images. For each matched descriptor pair, the algorithm calculates the hamming distance between the descriptors. The matches are then filtered out to get the best matches based on these hamming distances.

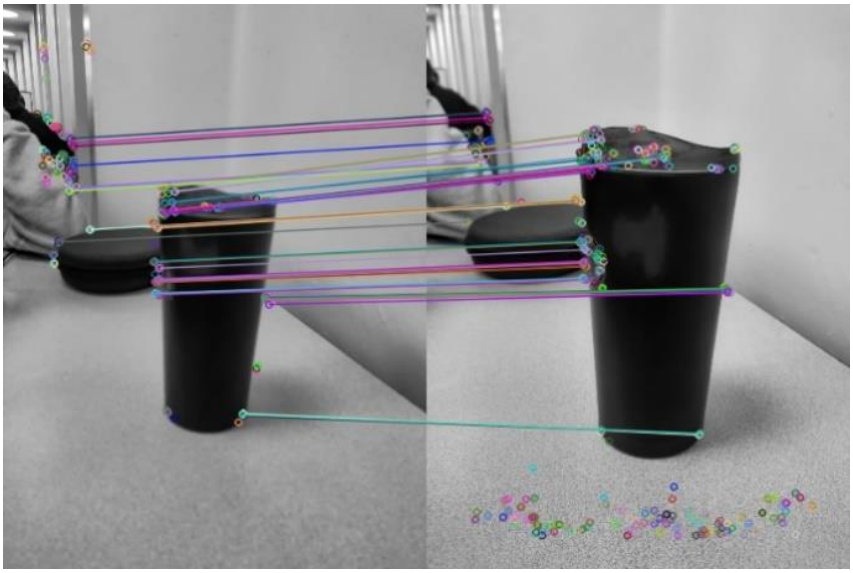

Fig. 6. Results of features matched using Brute Force Matcher

B. Fast Library for Approximate Nearest Neighbors (FLANN):

FLANN is a library that performs fast approximations of nearest neighbor searches in higher dimensional spaces. FLANN consists of a collection of few reliable approximate nearest neighbor searches algorithms and a system that decides which algorithm will provide the best results for the given dataset in terms of speed and accuracy. It also allows us to set the accuracy required by simply passing a parameter, the system calculates rest of the optimization parameters automatically. [3]

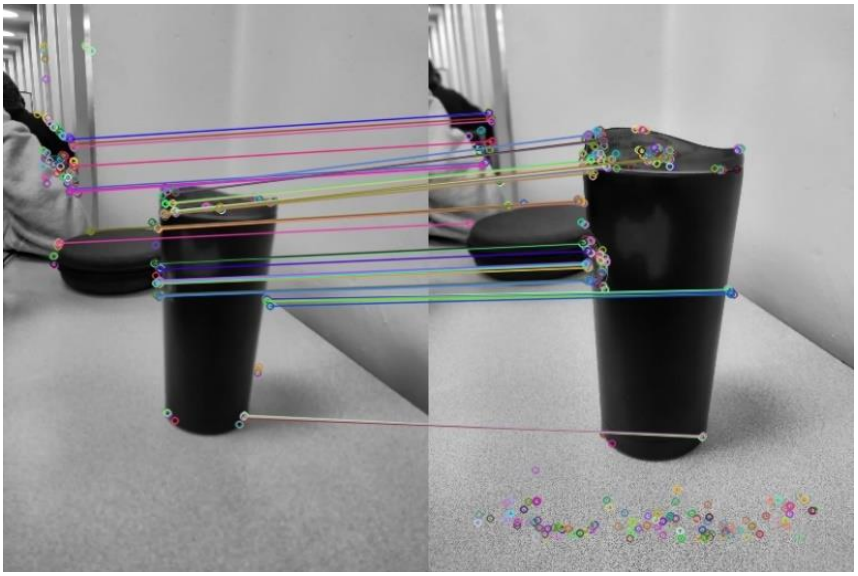

Fig. 7. Results of features matched using FLANN feature matcher

If the input dataset is large, the brute force matches slow down considerably, which does not meet our real time requirements. But since the FLANN is performing nearest neighbor approximations, the accuracy of the results of the Brute Force Matcher is more than the accuracy of the FLANN matcher. Hence, there is a tradeoff between speed and accuracy that we must consider when it comes to choosing one of these two matchers.

In our implementation, the number of descriptors is small so brute force can be used, but in case we decide to extend our project later to implement SLAM, then the features are matched to a 3D map which has increasing dataset which will slow down the brute force matcher. Thus, we decided to use the FLANN Matcher to match the descriptors of the left and right images.

2.4. Finding 3D-2D Correspondence for Motion Estimation

To estimate motion, we need to find matches in the left and right frames at time = t. For this process, there are correspondences between 2D-2D, 3D-3D and 3D-2D which can be used.

We used 3D-2D correspondence, as it does not use epipolar constraints for motion estimation and can give a very good result using the least number of matched features. In 3D-2D correspondence, the 2D feature matches among the two stereo images (left and right) will give us a set of features. Using these 2D features, we need to find points in the 3D world which we would observe in the world. But knowing only the location of these 3D points is not enough to estimate motion. There should be some correspondence between the points at time = t and their respective points at time t = t+1, that is the next image.

There are two stages in this 3D-2D correspondence process:

2.4.1 Triangulation (locate 3D correspondence):

One of the most fundamental tasks in multiple view geometry is the process of triangulation. The principle behind this process is to determine the location of a 3D point given its perspective projections into two or more images. This is useful in moving out of the 2D image plane out into the world frame. Once we have good feature matches, we can use these matched features to find corresponding 3D points which we can use in order to track the motion of our camera system.

Coming to our implementation, after detecting the key points in left and right stereo images, we used triangulation to estimate the spatial positions of these key points. We used the OpenCV function "**triangulatePoints" [11]** which takes input as projection matrix of both left and right camera, input array of feature points in left image and its corresponding feature points in right image. The output is a list of 3D locations corresponding to these matched feature points in both left and right images in the camera coordinate system. We need to multiply these 3D points with the homogeneous transformation matrix to bring them into world coordinate system. At time t = 0, homogeneous matrix is given as [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]].

Working (Geometric Intuition behind Triangulation):

Ideal case: The image given below shows a geometric representation of how Triangulation can be perceived. In this ideal case, the rays from the object point on the two image planes are intersecting each other out in the world. Therefore, it is possible to proceed with this pair of rays.

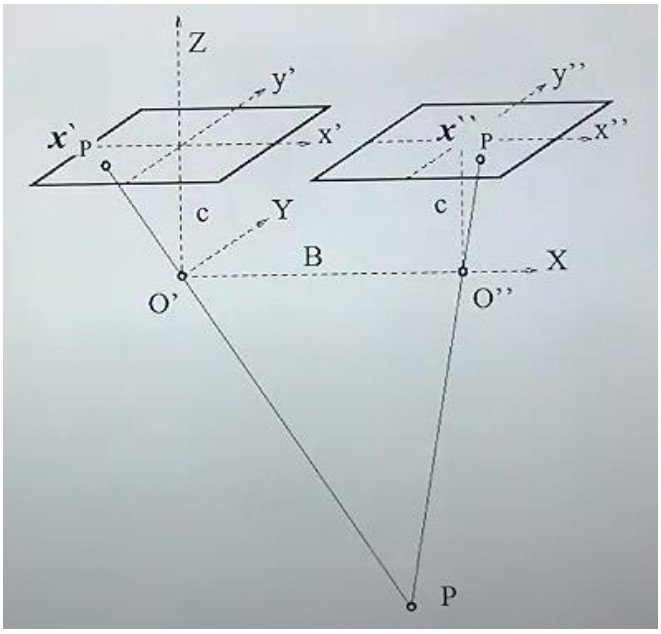

Fig. 8. Geometric interpretation of Triangulation
P: Any spatial point
x'p: Projection of point P in left image.
x" p: Corresponding projection of point P in right image.
B: Baseline (distance between 2 cameras)

Problem: This ideal case may not work every time. That is, the rays may not intersect and may pass over each other with some offset in between. This can happen because the relative orientation information that we have is not perfect, or the location of the matched features are not correctly estimated.

Solution: We find the location where the two rays seem to be the closest to each other and find the mid-point of these two locations.

2.4.2 Locate 2D correspondence:

After getting the 3D location of matched features in stereo images at time = t, we need to find the corresponding 2D location of the respective 3D point in the next left image frame i.e., at time = t + 1. We used OpenCV's "**calcOpticalFlowPyrLK**" **[7]** function which calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.

2.5. Motion Estimation

2.5.1 PnP:

PnP i.e., Perspective-n-Point estimates the position of world origin in the camera coordinate system. We have used OpenCV function named **"solvePnPRansac" [8].** As an input to this function, we pass an array of 3D points in the world coordinate system which we got after triangulation method at time t and their corresponding 2D features in left image at time t = t + 1. This function also needs the camera intrinsic parameters as an input. Since we already know camera projection matrix from the calibration files of KITTI stereo vision dataset, we can calculate camera intrinsic matrix by decomposing projection matrix using RQ decomposition. For our purpose, we used OpenCV function **"decomposeProjectionMatrix" [9]**. There are various methods to solve PnP like Direct Linear Transformation (DLT), Efficient PnP (EPnP), P3P etc. We used default method I.e., iterative method which uses Levenberg-Marquardt optimization to find a pose that minimizes reprojection error, that is the sum of squared distance between the observed projection image points and projected 3D points to the image plane.

We read that **EPnP** method is supposed to be good compromise in terms of precision and speed, therefore we also tried solving PnP problem with EPnP but observed that the estimated trajectory which we got as an output after our code implementation had a higher error drift from the ground truth trajectory at the initial stages itself and trajectory was observed to be discontinuous at various stages of trajectory plotting.

Output of this function gave us rotation vector and translation vector. For forming the homogeneous transformation matrix between camera coordinate system and world coordinate system we would need rotation matrix instead of rotation vector. So, we used OpenCV's "**Rodrigues" [10]** function to obtain rotation matrix. Using the rotation matrix and translation vector we formed Homogeneous which would give us coordinates of world origin in the camera

coordinate system. But humans can easily perceive the intuition of finding the camera pose in world coordinate system, hence we took the inverse of the received homogeneous matrix.

Rodrigues's formula is given as:

$$\mathbf{R = \cos\theta I + (1 - \cos\theta)\, nn^T + \sin\theta n^\wedge}$$

Where R is rotation matrix; n is unit length vector representing rotation axis and symbol ∧ is a vector to skew-symmetric conversion.

We used the formed Homogeneous matrix to multiply it with the 3D points which we got after the triangulation process to bring those points from camera coordinate system to world coordinate system. We receive an array of 4x4 after this process. To get the actual x, y, z coordinates of the 3D point in world coordinate system, the first 3 rows of the matrix must be divided by its last row. And we reiterate from step _ to step _ for the whole sequence.

2.5.2 RANSAC:

After using ORB feature detection to find 2D feature correspondences between the left and right stereo images at time 't', we got many matches which also included outliers which when used for the 3D-2D motion estimation using PnP method would not give us good results. To resolve this, we used the RANSAC i.e., RANdom SAmple Consensus algorithm.

RANSAC in the case of Stereo Visual Odometry computes a mathematical model hypothesis that gives relative motion i.e., Rotation and Translation between 2 camera positions from the data points comprising of 3D-2D feature correspondences. Inlier points are found by estimating the first-order approximation also known as Sampson distance between hypothetical inlier points to epipolar line.

2.6. Evaluation

After computing the estimated trajectory, an important task is to evaluate the translational points obtained from the estimated trajectory with those of the ground truth. There are many ways to compute this error. The most common way is ATE (Absolute Trajectory Error), which is basically subtracting the estimated co-ordinates from the respective ground truth co-ordinates. The plots shown in the Results section show the absolute error as a function of image sequence number.

Using this Absolute error, it is possible to evaluate RMSE, MSE and Mean Errors as well. We can also compute error as a function of distance travelled by the car, as the error will accumulate over the image sequence. Since the position is with respect to the initial starting point, the error could also include a factor of distance from the initial location.

3. Results

We implemented our algorithm on various image sequences in the KITTI dataset and obtained good results for visual odometry. The trajectories seem to drift after some time, especially when the path involved multiple turns and loops. This can be corrected in future, by including key features from the backend of SLAM such as Bundle Adjustment and Loop Closure.

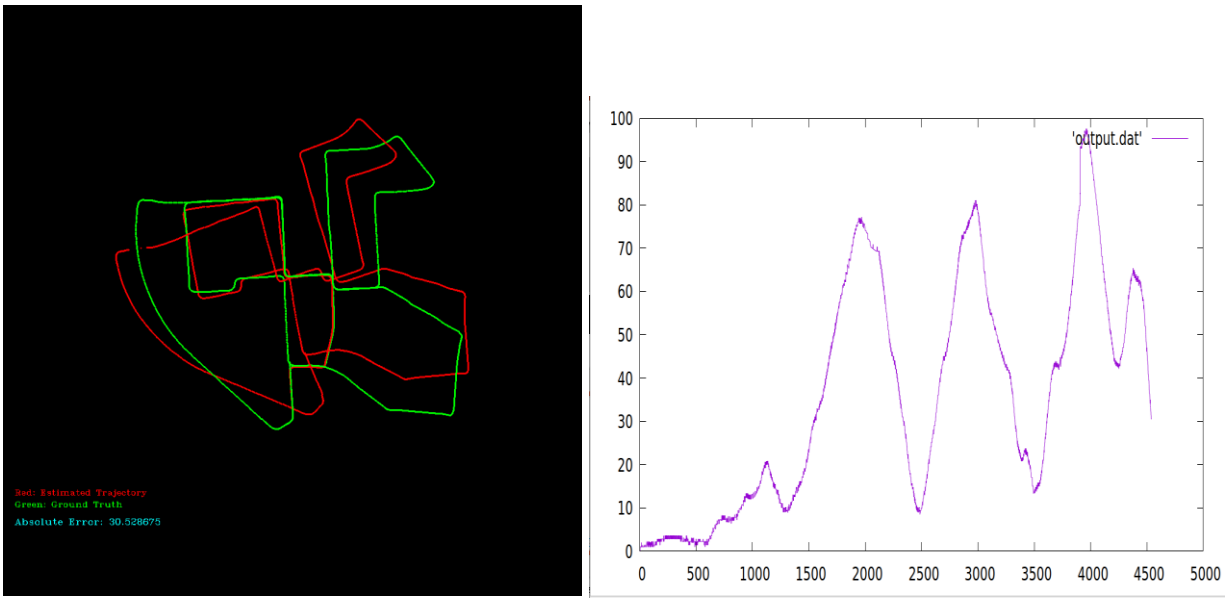

Fig. 9. Trajectory Image (left) and Absolute Error plot (right) for KITTI Image Sequence 00

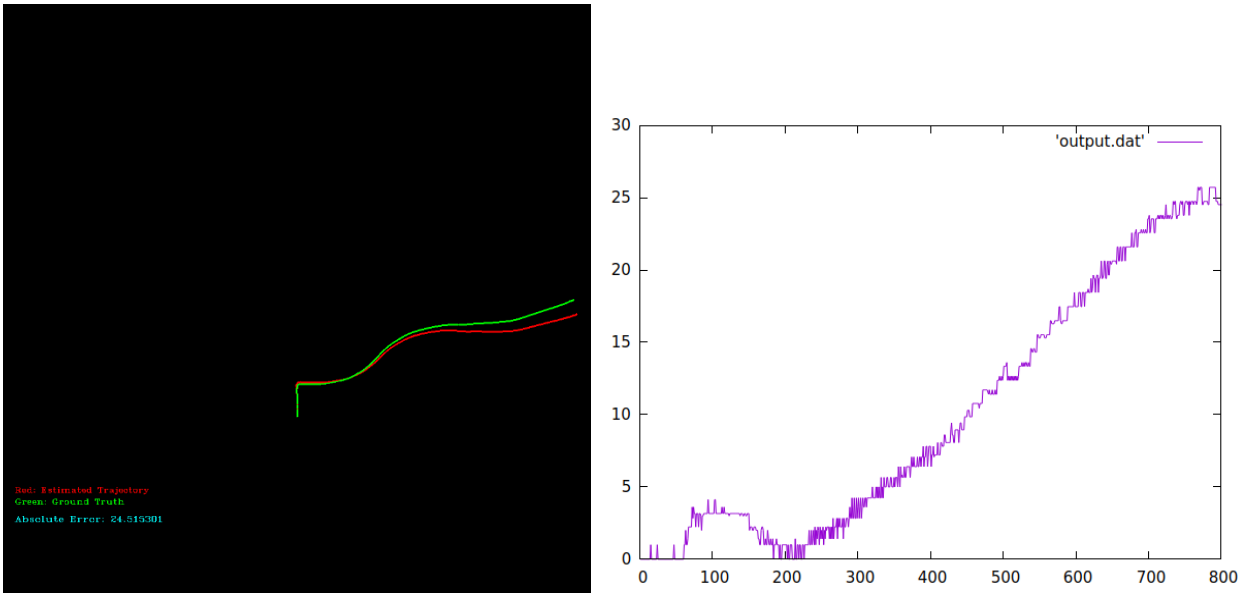

Fig. 10. Trajectory Image (left) and Absolute Error plot (right) for KITTI Image Sequence 03

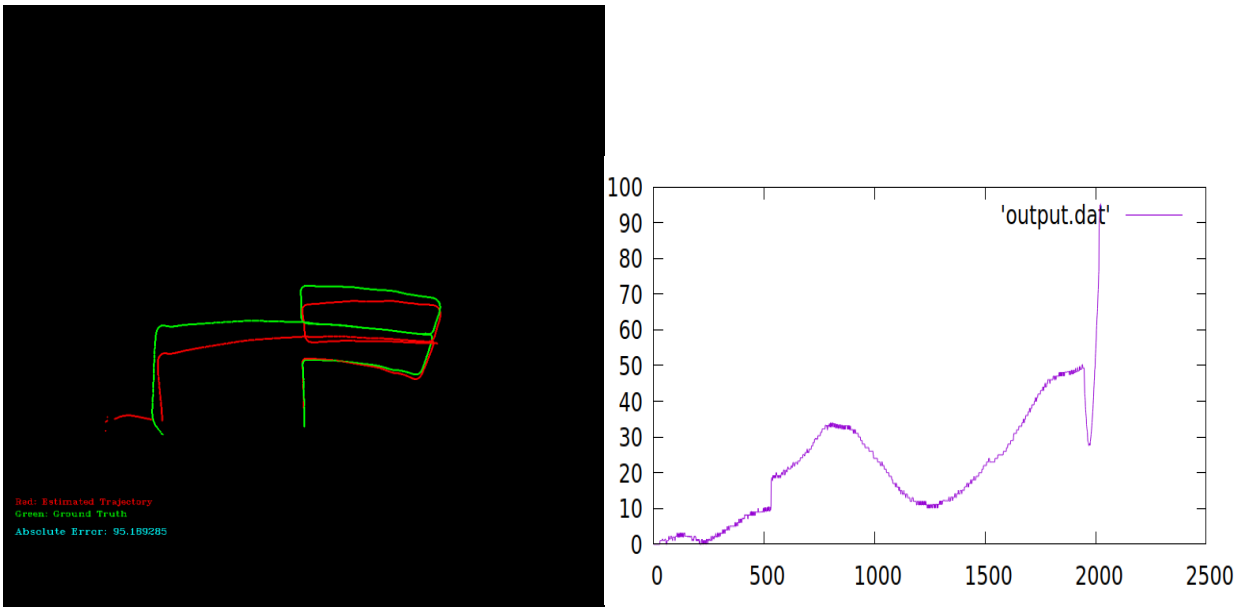

Fig. 11. Trajectory Image (left) and Absolute Error plot (right) for KITTI Image Sequence 05

4. Conclusion

From the results obtained we can see that our trajectory matches the ground truth trajectory initially, but the small errors or drifts accumulate over time resulting in egregious poses. To reduce the drift/error we can implement SLAM characteristics like loop closure to correct measurement.

All brightness dependent motion trackers perform poorly when subjected to sudden changes in luminance, therefore a robust brightness invariant motion tracking algorithm is required to accurately estimate the motion. We can use a fast inlier detection with added heuristics such as an estimate to how accurate each 2D-3D point pair can help with early termination can be implemented.

5. References

[1] The KITTI Vision Benchmark Suite (http://www.cvlibs.net/datasets/kitti/eval_odometry.php)

[2] https://www.researchgate.net/figure/The-illustration-of-our-visual-odometry-framework-The-initial-state-vti-of-current_fig2_332103736

[3] Marius Muja and David G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration", in International Conference on Computer Vision Theory and Applications (VISAPP'09), 2009

[4] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," 2011 International Conference on Computer Vision, 2011, pp. 2564-2571, doi: 10.1109/ICCV.2011.6126544.

[5] http://rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf

[6] Karami, Ebrahim, Siva Prasad, and Mohamed Shehata. "Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images." arXiv preprint arXiv:1710.02726 (2017).

[7] https://docs.opencv.org/3.4/dc/d6b/group__video__track.html#ga473e4b886d0bcc6b65831eb88ed93323

[8] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnpransac

[9] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#decomposeprojectionmatrix

[10] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#rodrigues

[11] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#triangulatepoints

[12] https://www.ipb.uni-bonn.de/msr2-2020/-

[13] https://github.com/gaoxiang12/slambook