A Stereo-Visual Odometry Benchmark for Autonomous Navigation

Jerrin Bright[1]

[1]Vellore Institute of Technology, School of Mechanical Engineering,
Chennai, Tamil Nadu, India
Jerrin.bright2018@vitstudent.ac.in

Abstract: ***This work deals with determining the best algorithm with respect to the environment (indoor or outdoor) for determining the visual odometry which results in efficient pose estimation for visual based autonomous navigation. Thus, various approaches and combinations of visual odometry has been studied with various combinations and examined closely in terms of accuracy and computational cost. We thus in this research examine all the commonly used feature extraction methods including ORB, SIFT, SURF, AKAZE, BRISK, Shi-Tomasi and Matching methods including Brute-Force Matcher and Fast Library for Approximate Nearest Neighbor. Then outlier rejection techniques including Random Sample Consensus and K-Nearest Neighbors are compared with and resulting stats are logged. Followingly, estimation techniques including various versions of perspective-n-points (PnP) including EPnP and MLPnP are studied for efficient retrieval of the rotational and translational vectors. Additionally, various pre-processing methods including Bilateral Filtering, Gaussian Blur, Rotation, Warping are experimented. The trajectory of the pose estimated via various combinations as mentioned above are studied and evaluated using KITTI and EUROC benchmarked datasets and conclusions stating the best blend and environment conditions for efficient visual odometry are conferred. The source code is available at https://github.com/jerriebright/visual-odometry***

Keywords: visual odometry, stereo vision, benchmark blend, pose estimation

1. INTRODUCTION

Visual Odometry (VO) is a very vital part of autonomous navigation used with autonomous ground vehicles, aerial vehicles and underwater or surface vehicles. Visual odometry is in layman terms, determining the position information for accurate navigation of the robot using camera image sequences. Usage of encoders has slippage issues which in turn will reduce the accuracy of estimation. Thus, VO provides a computationally cheaper and accurate odometry results when compared to GPS, wheel odometry, sonar localization or INS.

There are many sensor-based SLAM techniques including camera, lidar, sonar, radar, etc. We are using camera-based SLAM technique (reasons of which are detailed in Table 1) considering the environmental conditions and the mission requirements for our system.

Table 1. Comparison of various sensors

	Camera	Lidar	Radar
Computation	3	1	2
Weight	1	3	2
Cost	1	3	2
Weather Dep	2	3	1
Heat	1	3	2
Range	3	2	1

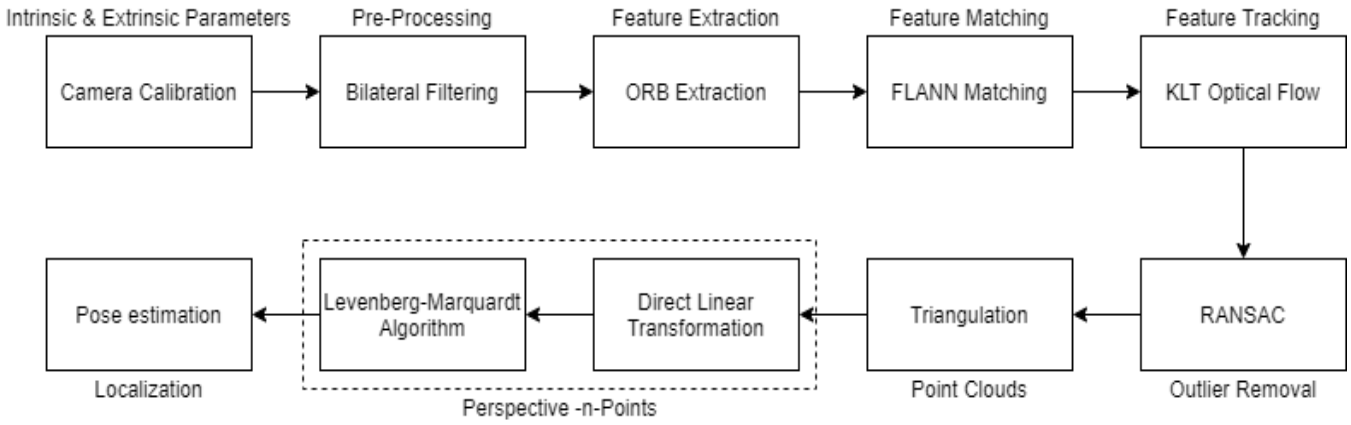


Figure 1. Architecture of the Visual Odometry

In table 1, 1 stands for most and 2 for moderate and 3 for least. From the above-mentioned table, we can conclude that:

- **RADARS** are very strong at accurately determining distance and speed — even in challenging weather conditions — but cannot read street signs or "see" the color of a stoplight.
- **CAMERAS** do very well at reading signs or classifying objects, such as pedestrians, bicyclists, or other vehicles. However, they can easily be blinded by dirt, sun, rain, snow, or darkness.
- **LIDARS** can accurately detect objects, but they do not have the range or affordability of cameras or radar. The lidar uses infrared sensors to determine the distance to an object. They fail in case of translucent or transparent objects.

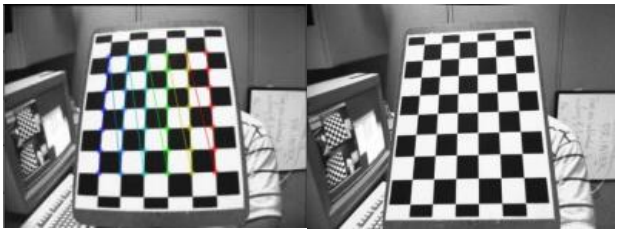
Thus, considering our requirements and the specifications mentioned, we have decided to use visual-based SLAM techniques.

2. METHODOLOGY

The basic workflow of stereo visual odometry includes Feature Extractor, Feature Matching, Feature Tracking, Outlier removal, Triangulation and PnP for efficient estimation of the pose using Direct Linear Transformation and Levenberg Marquardt Algorithm in its backend. The architecture can be visualized in fig. 1. In this section we will explain all the steps involved in the visual odometry algorithm in detail.

1. Camera Calibration

Calibration reduces the radial and tangential distortion. Calibration helps to determine the intrinsic and extrinsic parameters along with the distortion coefficients. Intrinsic parameters include focal length and optical centers and extrinsic correspond to the rotational and translational vectors. Implemented using OpenCV library and OpenCV's sample dataset of chess board. First the image points are determined in the chess board as in the first image fig. 1 (a). Then, calibration can be done using the **calibrateCamera()** library. The second image that's fig. 1 (b) shows the undistorted image after calibration is done.



(a) (b)

Figure 2. Camera Calibration

2.Pre-Processing

2.1 Smoothening/ blurring

We have used ORB extractors upon the smoothened frames and analyzed with respect to frames prior pre-processing. Some of the smoothening methods experimented are image filtering, gaussian blurring, bilateral filtering and median blurring.

2.2 Rotation and Warping

Experimented in the pre-processing steps to observe the results with such types of pre-processing steps. It for sure will increase computational cost of the system. Its effect on the accuracy of the feature extraction was thus tested precisely.

3. Feature Extraction

There are a lot of feature extraction methods including FAST, SIFT, ORB, SURF, Shi-Tomasi, etc. Basic working of these extractors and comparison with outdoor and indoor environment with different contrasts are shown in this section.

3.1 FAST

FAST stands for Features from Accelerated Segment Test. It is one of the fastest feature extraction technique which could be used for extracting features and then to track and map. It is best for live real-time application point of view with efficient computation. It takes a pixel (p) from the image and circle it with 16 pixels called as the Bresenham circle as the first step to detect corners. Now we will label each pixel from 1 to 16. Then check random N labels in the circle if the intensity of the labeled pixel is brighter than the pixel around which 16 pixels has been selected. Thus, the conditions for the pixel p to be a corner is that if the intensity of x should be greater than the intensity of p + threshold or if the intensity of x should be less than the intensity of p – threshold. The main significant step in this method is the point where we decide the N and the threshold. N is the pixels out of the 16 pixels to be considered. More the N, more accurate but more computation. N is usually taken as 12. So, this trade-off has to be considered smartly to get desirable results. The difference between the pixel p and the surrounding N pixel values is calculated and stored in a variable V. Now 2 adjacent keypoints will be considered and its corresponding V value will be calculated. The lowest V value will be castoff. This way multiple interest points will be avoided.

3.2 SIFT

SIFT stands for Scale-Invariance Feature Transform. The working of SIFT algorithm comprises of 5 steps. At first, **Scale-Space Extrema Detection** is done as follows Blurring of the images using Gaussian filters. Then the Octave is taken. That's a pixel will be selected and compared with the 8 pixels around it. These

9 pixels are considered to be an octave. After which 2 more octaves above and below the pixel we considered before is chosen. Then, local extremum is searched for and if found, then it will be considered a potential keypoint. After the extrema detection, **Keypoint Localization** is done. Its process majorly does 2 important things. They are removing edges using eigen values and ratios and removing the extrema of intensities lesser than the threshold values set. Thus, removal of low contrast keypoints and edge keypoints happens in this step. After all this removal processes, **Orientation Assignment** is done. In this step orientation will be assigned to the keypoints. With these orientations for the keypoints being created, an orientation histogram will be derived from the orientations. And when the orientation is more than 80% for a keypoint that keypoint will be taken into consideration for calculating the orientation. After the orientation setup, **Keypoint Descriptor** is derived. The size of the vector is the number of keypoints * 128. The algorithm will create a 16*16 neighbor around the keypoint in which each subblock will have 4*4 pixels. Now, each of this pixel will have 8 pixels around it. Thus, 8*4*4 gives 128. After derivation of the description from the orientation **Keypoint Matching** will be done. It is the process where keypoints are matched between images irrespective of the rotation, contrast or the scaling. This is done by identifying the nearest neighbors and doing ratio analysis between closest and second closest neighbors. Also, if this ratio is greater than 0.8, then the keypoints are rejected. This is done to avoid closest match of neighbors which happens due to noises.

3.3 SURF

(Speeded Up Robust Features) is a robust and fast technique preferred for its fast computation of operators using box filters, thus enabling real-time applications such as tracking and object recognition. Consists of 2 steps, Feature Extraction and Feature Description. Feature extraction. Feature extraction uses integral image, which is a way of calculating the average intensity of pixels in a selected box. Surf uses the Hessian matrix because of its good performance in computation time and accuracy. Descriptors are assigned by first assigning a fixed orientation to each interest point, and then extract feature descriptor.

3.4 BRIEF

BRIEF stands for Binary Robust Independent Elementary Features. It uses binary strings as feature point descriptor. In other words, instead of the 128-bin vector in the SIFT, we will use binary strings in BRIEF. It easily outperforms SIFT in terms of speed and recognition rate. Here, the neighbors around the keypoint are called patches. Thus, the major function of BRIEF is to identify the patches around the keypoint and convert it into a binary vector, so that they can represent an object. Thus, each keypoint will be described with the help of the binary 1's and 0's. One major consideration about the BRIEF is that it is very noise sensitive as it deals closely with pixel-level images. Thus, smoothening is very important to reduce the noise from the image. Smoothening is done using Gaussian Kernel in BRIEF. Once smoothening is done, the next step is the feature descriptor.

Now the problem will come when we will have to calculate the n (x, y), that's the x, y pairs. This (x, y) pairs are called random pair inside the patch. N is the length of the binary vector and τ is the binary test response of the vector. Finding the random pair can be easily done if the length of the binary vector is decided as we can pick from the patch easily then. There are 5 different methods used to calculate the length. They are uniform (GI), gaussian (GII), gaussian (GIII), coarse polar grid (GIV), coarse polar grid (GV). The major advantage of BRIEF is that it is faster than SIFT in terms of speed and recognition rate. Also, BRIEF relies on less intensity difference between the image patches.

3.5 ORB

ORB stands for Oriented FAST and Rotated Brief. FAST algorithm is used to calculate and find the keypoints from the image. Thus, it was most often used to detect and differentiate edges, corner and flat surfaces. It doesn't calculate the orientation just like SIFT method. Thus, thus algorithm ORB, solves this

issue. Also FAST isn't invariant of the scaling of the image. ORB also solves this issue. The computation of ORB is done using BRIEF algorithm., Thus it is a mixture of both FAST and BRIEF making it better than SIFT in terms of speed and computation.

3.6 SHI-TOMASI

Harris' corner detector takes the differential of the corner score into account with reference to direction directly. A corner is an important feature in an image, as they are interest points which are invariant to translation, rotation, and illumination. The idea is to consider a small window around each pixel p in an image. Then we measure gradient shift of the central pixel and the pixels around in all 8 directions (using SSD values of pixel value). If shift is above a threshold value, a new feature descriptor is mapped. Small modifications in the Harris Corner detection resulted in the Shi-Tomasi corner detection using the good features to track algorithm. The major difference was observed in the scoring pattern:

4. Feature Matching

Figure 3. Feature Matching

4.1 BF Matcher

It takes the descriptor of each and every feature of first set and matches with all the other features from the second set using distance calculation. It is time consuming and matches all features. There are a lot of distance measurement methods including NORM_L2/ NORM_L1, NORM_HAMMING, etc. Also, Lowes's test is done after matching in general to remove outliers.

4.2 FLANN

Stands for Fast Library for Approximate Nearest Neighbor. It has a collection of optimized algorithms to for searching the nearest neighbors in big datasets. It is faster and more accurate than BF Matcher. It has two dictionaries index and search parameters.

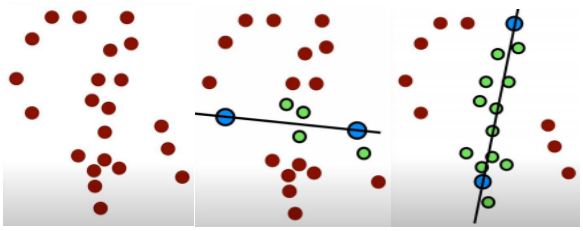
5. Feature Tracking

Figure 4. KLT Optical Flow

6. Outlier Removal

RANSAC stands for Random Sample Consensus. Deals with removing outliers from inliers contained in a data. No real-world sensor readings are perfect. Thus, we use RANSAC which is a simple trial and error method with groups the inliers and outliers separately. Thus, it helps us to throw away the outliers and work with inliers alone which will save our computation and time. It involves a 4-step processing, and they are sampling, scoring, computing and repeating.

Let's assume there are 2D points (Fig. 14 (a)) in which a line has to be fitted. Now we will sample and randomly take one line marked in fig. 14 (b) as our inlier. Now we will compute the number of supporting inliers satisfying the line we drew. This will be done by parallelly extending imaginary lines on both the side of the line assumed to be the inlier with a uniform distance of **δ.** Now each and every point lying inside these imaginary lines constitute to the scoring of the inliers. Now for the fig. 14 (b) there are 4 inliers.



(a) (b) (c)

Figure 5. Representation of RANSAC algorithm

Now we will repeat the steps mentioned above repeatedly and the score will be overwritten with the best score after every iteration. In the Fig. 14 (c), that's after some iterations, we got a line which has 12 points satisfying that line model which is the best score to be obtained. Thus, it will be best fit for the line. This way we can eliminate all the outliers easily.

7. Rotation and Translation

$$S \times p_c = K \times [R \mid T] \times p_w \tag{1}$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2}$$

Some of the assumptions taken are: Camera parameters are known. Image co-ordinate is known. For Mono, find scaling factor. For Stereo, find landmarks via triangulation. Then, find Rotational and translational matrixes.

For mono, we will first estimate the estimation matrix derived from the Fundamental matrix. Then we will use recover pose to find the rotation and translational vectors. In stereo, we will use a variant of PnP (that's Perceptive-n-Points). Some of PnP variants include EPnP, MLPnP and the frame-PnP. Basic framework of PnP is as follows:

- Direct Linear Transform (DLT) finds approximate value of Rv and Tv using the projection matrix.
- Levenberg-Marquardt Algorithm is a trial-and-error optimization step used to reduce the reprojection error thereby optimizing the estimated Rv and Tv values.

3. DISCUSSION/ ANALYSIS

Pre-processing

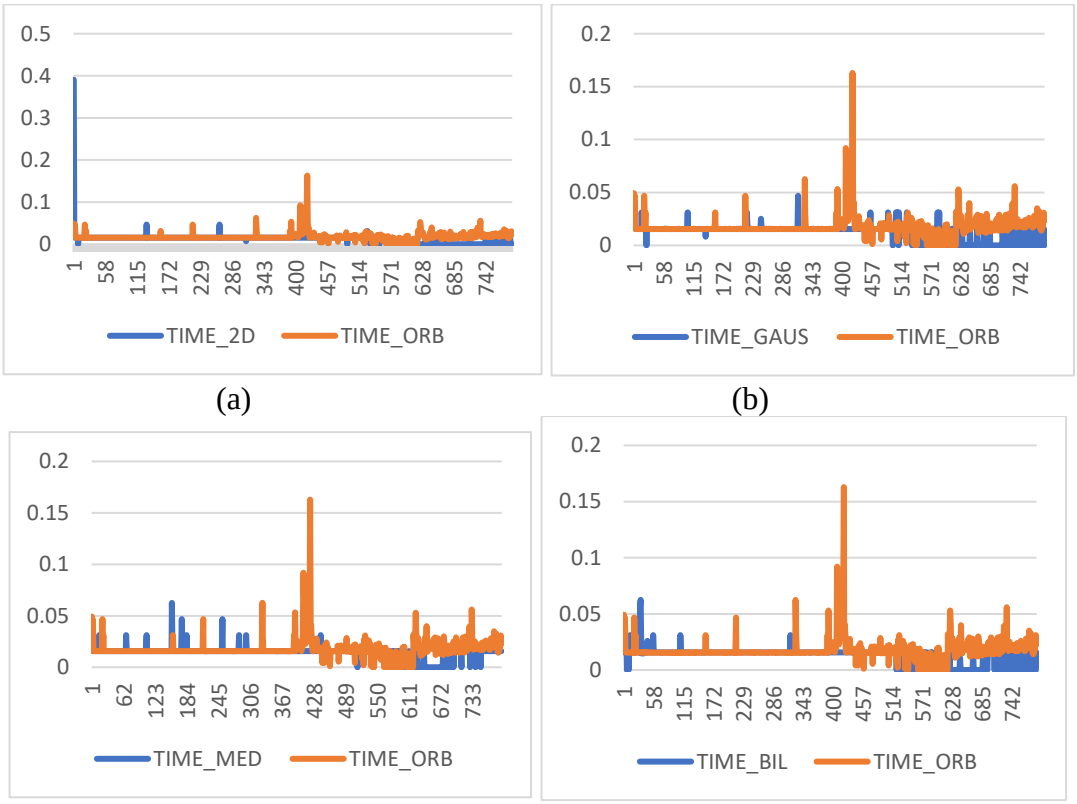


Figure 6. Time taken after smoothening

In fig. 6 represents time taken for ORB extractors with and without smoothening (a) using Image Filtering, (b) using Gaussian Filtering, (c) using median blurring and (d) using bilateral filtering technique.

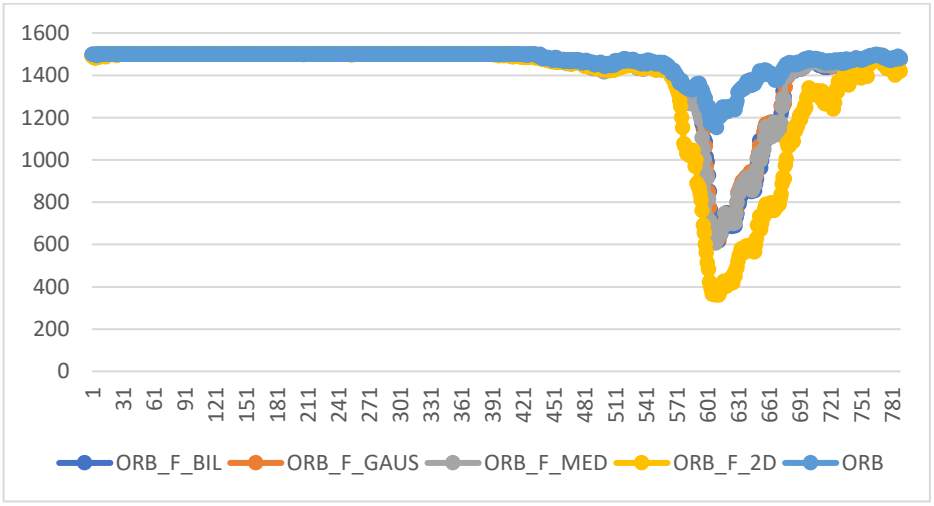


Figure 7. Number of features extracted after smoothening

In fig. 7, we can see the number of features extracted in each and every frame after applying different smoothening and edge sharpening filters. It is observed that there is a significant decrease in the number of features extracted when filters are used over the frames.

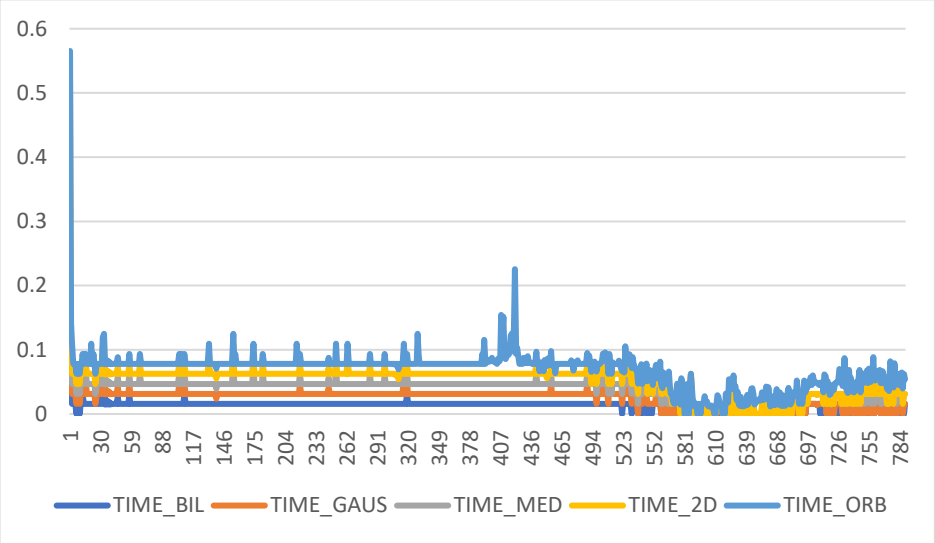


Figure 8. Time taken for extraction with post-processing

Fig. 8 depicts that smoothening reduces the total time for processing and finding the number of features. This is an effect of reducing the number of features while smoothening occurs as visualized in fig. 7.

2.2 Rotation

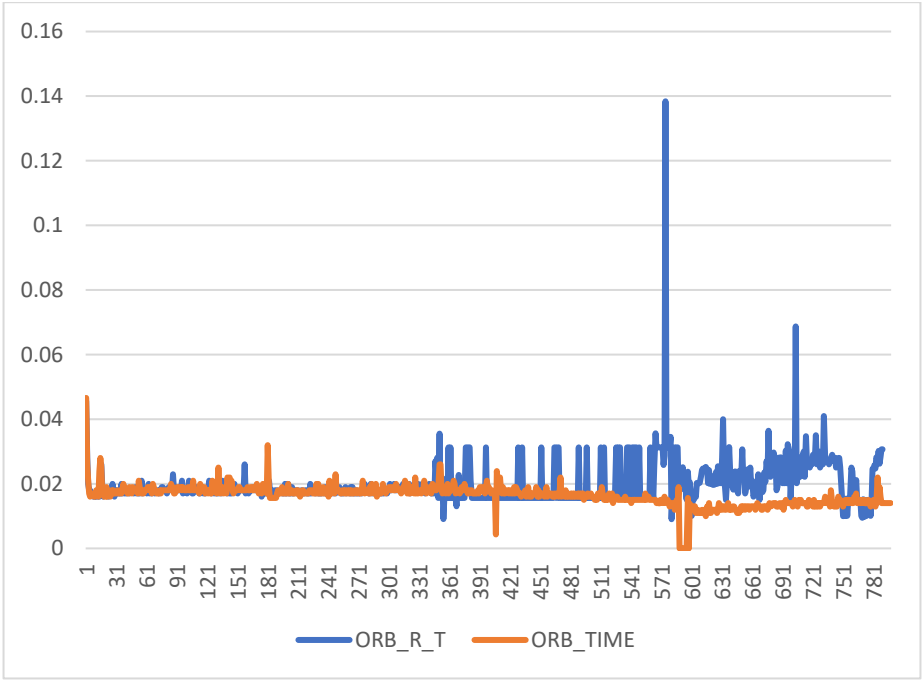


Figure 9. Comparison with and without rotation

Fig. 9 clearly shows an increase in the processing time when rotation is applied to one frame. Especially a significant increase is observed while transitioning to a low light condition

Feature Extraction

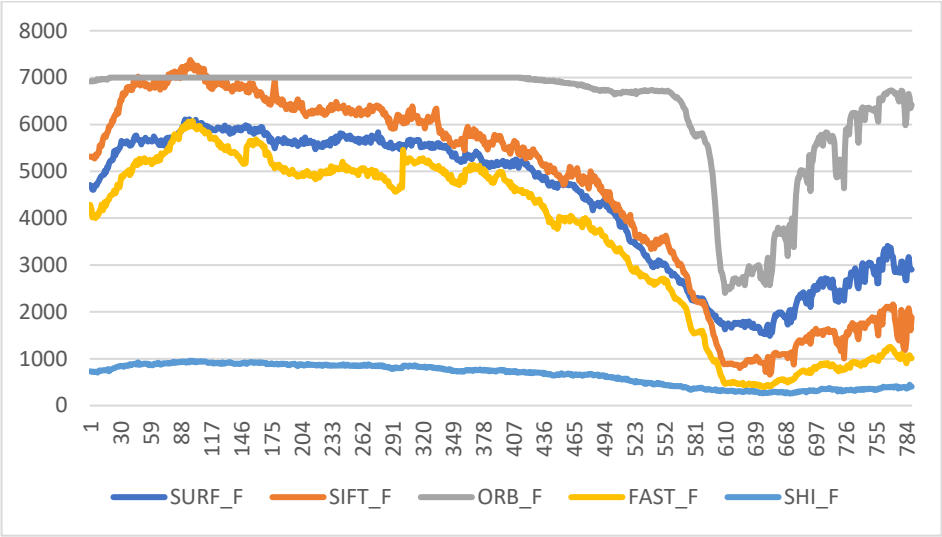


Figure 10. Feature Extractors

Fig. 10 visualizes all the number of features with respect to the frames of various feature extractors used throughout this section. Some of the extractors included in this figure are ORB, SURF, SIFT and FAST.

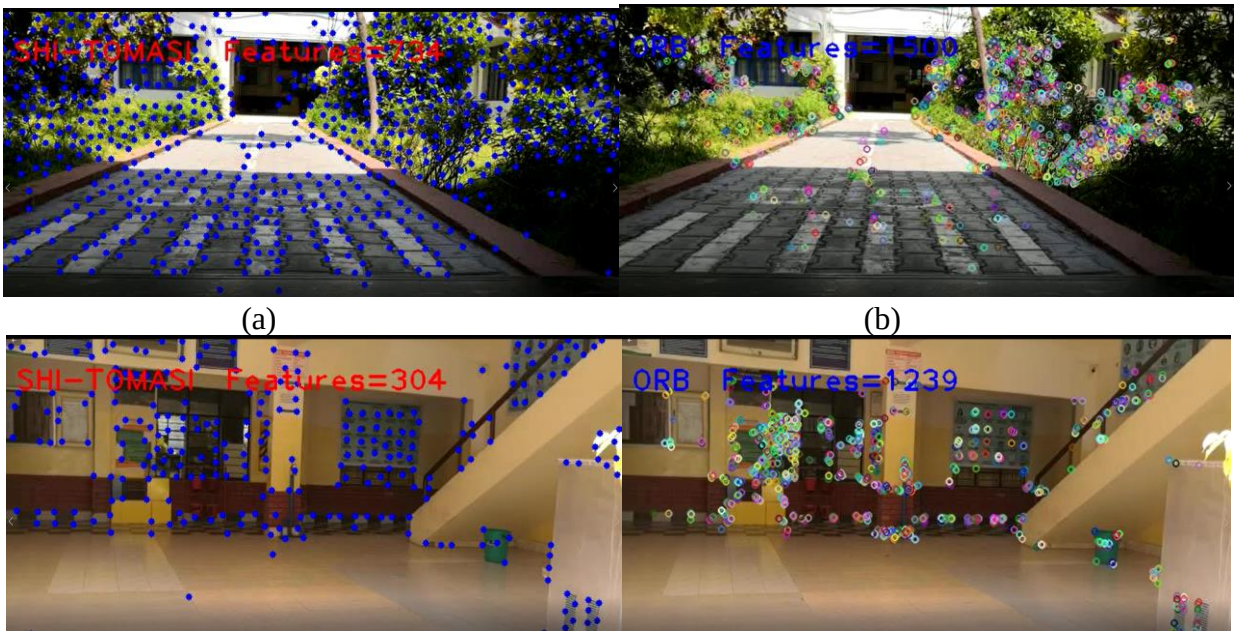


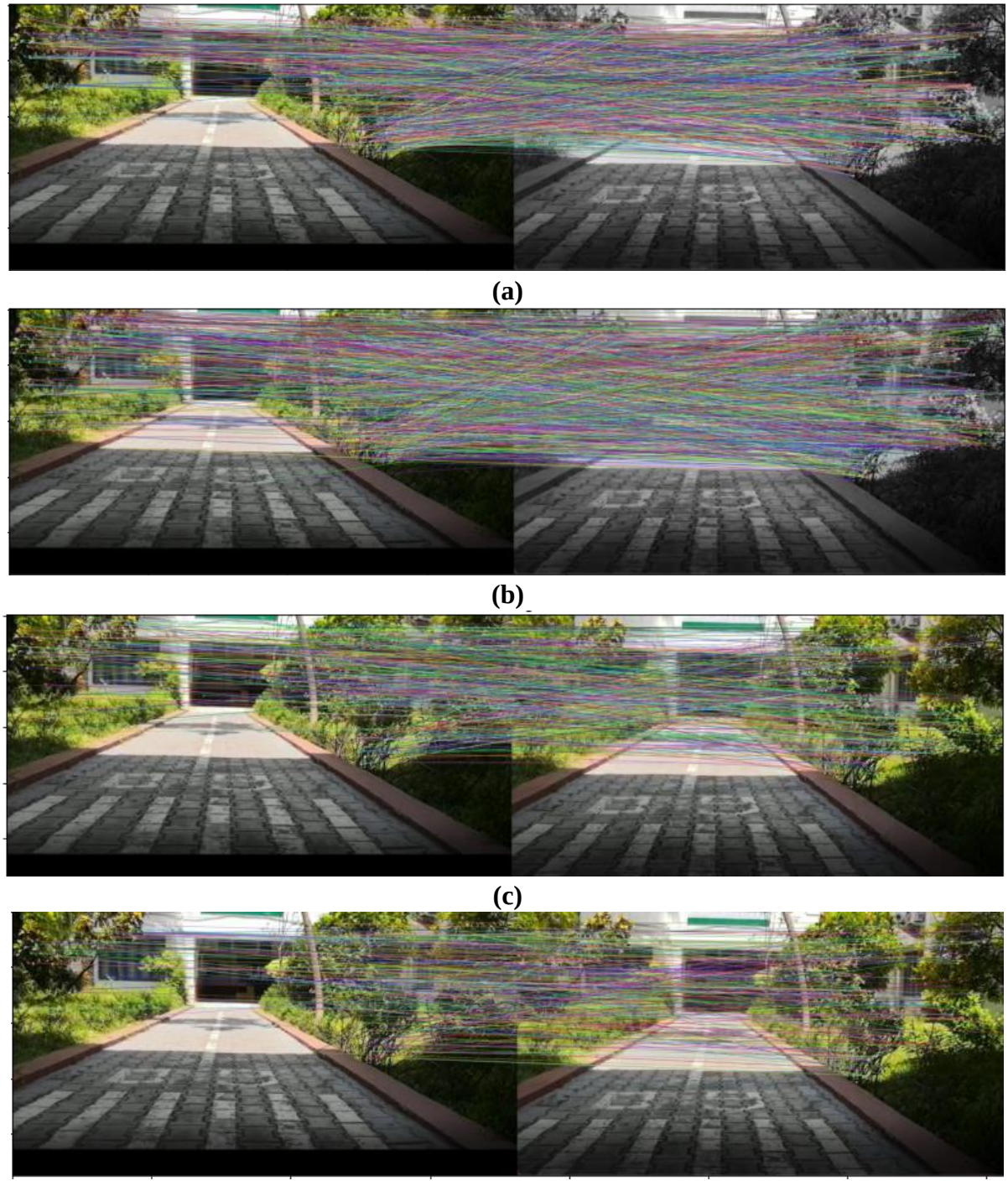
Figure 11. ORB vs Shi-Tomasi in different environments

In fig. 11, (a) and (b) shows Shi-Tomasi and ORB in Outdoor environment. (c) and (d) represents Shi-Tomasi and ORB in indoor environment. Maximum of 1500 features were given to both the extractors. All the frames were processed in grayscale for more meaningful comparison.

Thus, ORB focuses on close range and does dense extraction thereby leaving out far off features unnoticed. SIFT and SURF performs similarly in outdoor environment and is computationally expensive focusing on sparse extraction. Controlling the number of features to be extracted of SIFT and SURF is not possible thereby making it difficult to manage tradeoff in extraction process. Shi-Tomasi has the least number of features comparatively in outdoor environment. But has the sparsest features extracted. Clearly

defines the boundary of the environment compared to other techniques. Rotation based preprocessing isn't required. Bilateral filtering is most preferred when compared with the computation cost and the number of features extracted.

Feature Matching



(a)

(b)

(c)

(d)

Figure 12. Feature matching using various extractors

In fig. 12, (a) represents feature matching using FLANN with ORB extractors, (b) represents feature matching with SIFT extractors and (c) represents feature matching with SURF extractors.

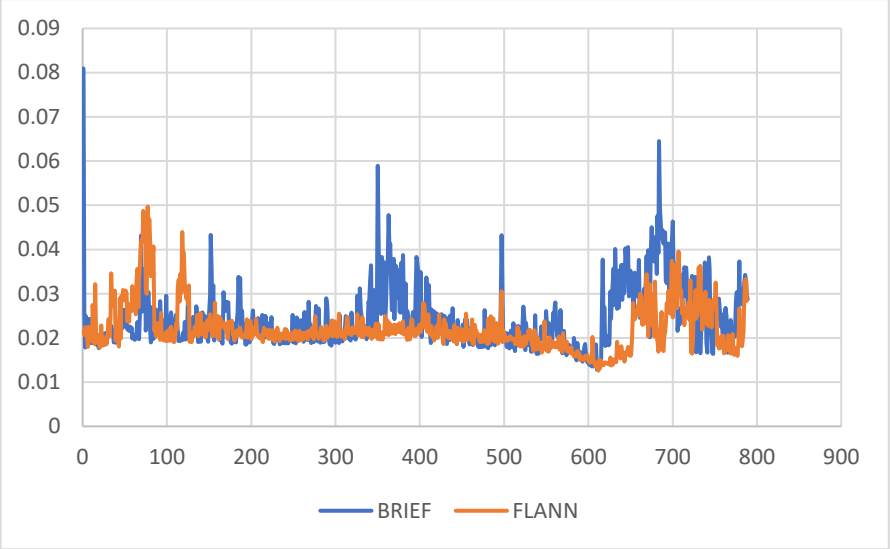


Figure 13. FLANN vs BF Matcher

Fig 13 clearly visualizes the FLANN and BF Matcher using ORB feature extractors. It is analyzed that FLANN is constantly faster than BF Matcher with respect to individual frames.

Pose estimation

Figure 14. Trajectory of the estimated pose

We are using KITTI dataset for comparison and estimating the error/ overlap percentage of our pose estimation algorithm. Some of the specs of KITTI dataset are Grayscale frames, consists of 22 stereo sequences (We have used 1st sequence which has 4500 stereo frames), 11 sequences with ground truth (for finding error/ overlap ratios). We are experimenting with 0th sequence of grayscale frames (right and left frames). In fig. 14, (a) represents the error vs frame number for FAST, (b) represents for SIFT, (c) for SURF and (d) for ORB.

4. CONCLUSION

In this work we have analyzed a lot of possible phases of visual odometry as can be studied from the previous section and we have wrapped with some interesting conclusions: ORB Feature Extractors are used considering its efficiency over other extractors and FLANN Matchers are used as it is faster than BFMatcher and accurate as per BFMatcher. For outlier RANSAC was used as it considerably outperformed PROSAC and MSAC. For obtaining the rotational and translational vectors, Effective PnP was used. Thus, we have concluded the following phases via this research study.

5. REFERNCES

[1]. https://github.com/felixchenfy/Monocular-Visual-Odometry

[2]. https://github.com/anubhavparas/visual-odometry

[3]. https://github.com/polygon-software/python-visual-odometry