E-VORTAR: EFFICIENT VISUAL ODOMETRY FOR REAL TIME APPLICATION IN ROS2

AAKASH AGARWAL [AAKASH24@SEAS.UPENN.EDU], KARTHIKEYA JAYARAMA [JKARTHIK@SEAS.UPENN.EDU],

ABSTRACT. For many years, the Visual Odometry and Mapping method has been a leader in odometric accuracy benchmarks. The objective of this project is to solve the problem of efficient visual odometry. We implement a complete end to end system for visual odometry in ROS2. For the same we first had to implement a data streamer to visualize the data effeciently. We are implementing the streamer and the algorithm in C++ because it is computationally efficient and helps us process in realtime. We evaluated evaluation metrics between the estimated motion and the ground truth and saw that our method works pretty good. Visually we can see the tracking is good. Our main contribution of this project is fully scale deployment of visual odometry system in ROS2.

Keywords: Visual Odometry, Motion Estimation, ROS2 Foxy, KITTI Dataset

1. INTRODUCTION

With the rise in the autonomous automobile industry and with the rise of powerful hardware able to process high definition data, the use of cameras, to help in motion planning, have gained popularity in the recent past. Visual odometry is estimation of motion by intelligent systems making use of inputs from the visual sensor (camera). Visual odometry is crucial for various robotic applications because knowing the pose of the robot is essential for almost all tasks. It plays a critical role in robot control, simultaneous localization and mapping (SLAM), and robot navigation, particularly in scenarios where external reference information about the environment, such as GPS data, is not available.

1.1. **Contributions.**

- We implemented complete and efficient visual odometry system using C++14 and ROS2 Foxy.
- We implemented a Dataset Streamer which takes in raw data points as inputs from KITTI Odometry Dataset and convert into ROS2 topics. It also maintains both static and dynamic transforms between different links.
- We compared different detectors and descriptors for feature estimation and matching in a given image frame.
- We have solved nonlinear least squares problem using Ceres solver in order to predict the transformation from current pose to next pose and stream it into output file for further analysis.

2. BACKGROUND

The goal of Visual Odometry algorithm which we implement is to precisely determine the 6-degree-of-freedom (6-DOF) motion and create a spatial, metric representation of the environment in real-time, onboard a robot navigating in an unfamiliar environment.

3. KITTI DATASET

The KITTI datase [1] comprises 22 sequences of high-resolution stereo RGB and grayscale images, along with Velodyne laser data and ground truth poses. The laser scanner gathers 360-degree point data at a rate of 10 frames per second, and the cameras are synchronized to the laser scanner, capturing image data at the same 10 frames per second. The data was captured using a mobile vehicle sensor platform in Karlsruhe, Germany. This dataset offers a variety of real-world environments with diverse structures, lighting conditions, lengths, and visual complexities, presenting challenging scenarios for analysis. Detailes can be found on their official website.

4. RELATED WORK

A lot of work has been done in the domain of vision based odometry in the past recent years. A major drawback of using monocular camera based visual odometry as done in [2], [3] you cannot solve for the scale of the motion without the help of assisting sensors or without making several assumptions about the motion. In the recent past, with the development of RGB cameras, it has become easier to associate a depth value to the visuals. This helps to conduct the estimation with scale as in [4], [5]. Very promising results are obtained in the work done by the authors of [6], [7] using RGB cameras. These work, as a drawback, only make use of the areas of the image with known depth values.

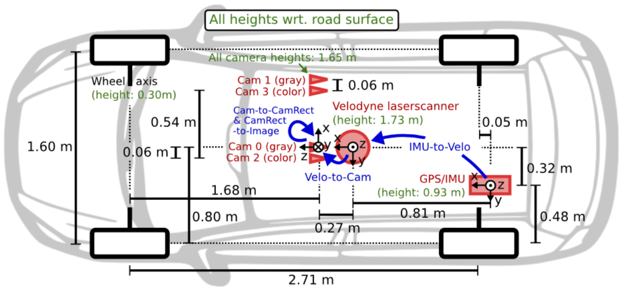

FIGURE 1. The KITTI car which has a combination of 3D Lidar, an IMU/GPS and 4 cameras. Image taken from their website

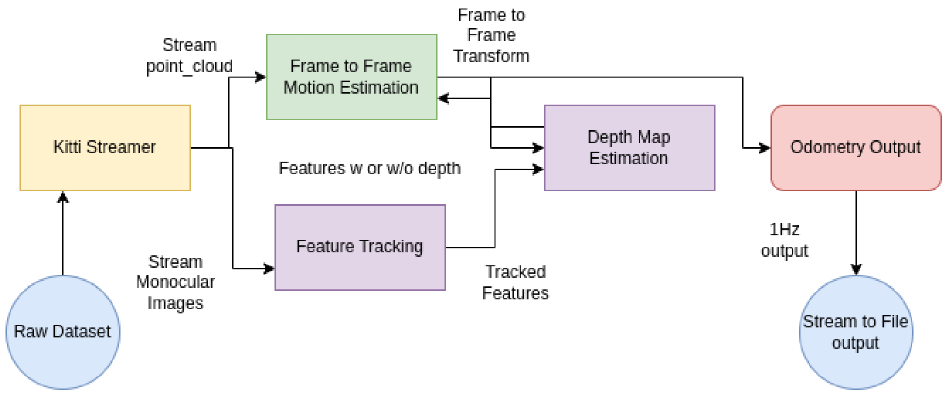

FIGURE 2. Overview of our Visual Odometry system

This method thus lose a lot of information as a lot of area in the image might be un-depth marked. The implementation of Visual Odometry that we have done uses features from areas of the images with and without depth.

5. APPROACH

5.1. **System Overview.** : Fig. 2 presents the high level overview of our system. The goal of visual odometry is to estimate motion between two consecutive frames obtained by the visual sensor (camera in our case). The methodology started by detecting keypoints and useful features in an image, we call this the feature tracking block. The next block is the depth map estimation block. The images from the camera are fed into this block to get a dept value for each point. This is done by the help of the onboard Lidar sensor. If a point in space has unknown depth, it's depth is marked as "unknown". The last module is the frame to frame motion estimation block which makes use of the depth values and the features obtained from the previous block. More details about the calculations are presented in the next section.

Block 1: Feature Tracking. The keypoints and features are detected using different state-of-the-art keypoint detection algorithm. We compare different algorithm for the detection. These algorithm are AKAZE [8], BRISK [9], FAST [10], ORB [11] and Shi-Tomasi [12]. Upon our observation we see that Shi-Tomasi performs the best out of all the algorithm, in terms of practicality. We talk about it in the result section. This module takes around 30 ms to process.

Block 2: Depth Map Estimation. The depth map is estimated using the Lidar sensor. The assumption that Lidar to camera transformation matrix is given, is made. We use this calibration data directly provided in the KITTI dataset. The process in this module is described as below.

- First the Lidar Point clouds are transformed to the camera frame using the transformation matrix. This gives us the depth map.
- Then we query a keypoint in this depth map.
- If we have a depth value available for this particular queried point, return the depth value otherwise set the depth value of this point to "unknown.

Block 3: Frame to Frame Motion Estimation: We do the frame to frame motion estimation in this module. It takes into account the calculated keypoints and the depth value of the consecutive frames. Now we see the mathematics behind this module. To do so, let us introduce some notation. Let a frame k be such that the feature i in this frame with known distance is represented as a vector $\mathbf{X}_i^k = \begin{bmatrix} x_i^k & y_i^k & z_i^k \end{bmatrix}^T$. Similarly a feature with unknown distance be represented as $\hat{\mathbf{X}}_i^k = \begin{bmatrix} \hat{x}_i^k & \hat{y}_i^k & \hat{z}_i^k \end{bmatrix}^T$. Let us also assume the norm is 1, since the depth is unknown. Estimating motion implies, that we have an estimate at frame k and we want to find a rotation and translation matrix that when applied to the current frame, given the next frame, that is,

$$\mathbf{X}_i^{k+1} = \mathbf{R}\mathbf{X}_i^k + \mathbf{Tx} \tag{1}$$

Now since we might not know the depth, we can write,

$$\mathbf{X}_i^{k=1} = d_i^{k+1}\hat{\mathbf{X}}_i^{k+1} \tag{2}$$

where d_i^{k+1} is the distance of that point.

Our goal is to find $\mathbf{R}$ and $\mathbf{T}$ that satisfy the above equations.

Depending on the knowledge of the depth values we can have two cases.

- **Case 1: Depth value is known**: In this case we can directly substitute the value of $\mathbf{X}_i^{k=1}$ from eq 2 in eq 1. Doing the math we obtain, [13]

$$\begin{aligned} \left(\hat{z}_i^{k+1}\mathbf{R}_1 - \hat{x}_i^{k+1}\mathbf{R}_3\right)\mathbf{X}_i^k + \hat{z}_i^{k+1}T_1 - \hat{x}_i^{k+1}T_3 = 0 \\ \left(\hat{z}_i^{k+1}\mathbf{R}_2 - \hat{y}_i^{k+1}\mathbf{R}_3\right)\mathbf{X}_i^k + \hat{z}_i^{k+1}T_2 - \hat{y}_i^{k+1}T_3 = 0 \end{aligned} \tag{3}$$

  the numbering in $\mathbf{R}$ and T means the row number.
- **Case 2: Depth value is not known**: In this case we first substitute use the normalized form of $\mathbf{X}_i^k$ and $\mathbf{X}_i^{k+1}$ and solving by eliminating d_i^k and d_i^{k+1}, we obtain,

$$\begin{bmatrix} -\hat{y}_i^{k+1}T_3 + \hat{z}_i^{k+1}T_2 \\ -\hat{x}_i^{k+1}T_3 - \hat{z}_i^{k+1}T_1 \\ -\hat{x}_i^{k+1}T_2 + \hat{y}_i^{k+1}T_1 \end{bmatrix} \mathbf{R}\hat{\mathbf{X}}_i^k = 0 \tag{4}$$

The above two systems of equation can be stacked up into one system with 6 variables (3 for each $\mathbf{R}$ and $\mathbf{T}$). To solve this system of equation we use the Levenberg-Marquardt algorithm, also known as damped least squares method [14]. We use the huber loss function which is known for the robust learning [15]

6. Experimental Results

The KITTI dataset is used to evaluate the algorithm. We use sequence number *00* and sequence number *02* of the dataset to test the algorithm. Although the complete dataset has 22 sequences we only evaluate on two, due to the large size of the dataset. Sequence *01* (1100 as compared to 4540 (Sequence *00*) and 4660 (Sequence *02*)) is very small and hence not chosen.

First, we implement a data streamer for ROS2 Foxy [16] to visualize the KITTI dataset. This converts the dataset directly to ROS2 topics. This step is an important aspect of our project. Almost all the implementation, in the literature, have been done in ROS1. With the industry moving to ROS2 it becomes essential to move developed algorithms to newer platforms. Fig. 3 shows the dataset streamed using ROS2. This helps us in visualizing our algorithm.

Algorithm	**Time Taken (ms)**	**Keypoints matches**
AKAZE	65.4523	1164
BRISK	238.679	2713
FAST	0.305689	196
ORB	261.322	929
Shi-Tomasi	13.5232	498

TABLE 1. Comparison of different algorithm for keypoint-detections.

We then implemented and compared 5 different algorithm for keypoint detection. The comparison is done based on the time it takes to compute and the number of keypoints matched. This comparison is presented in Table. 1. Based on the trade off between time taken and the number of matches we decided to use Shi-Tomasi detector. The

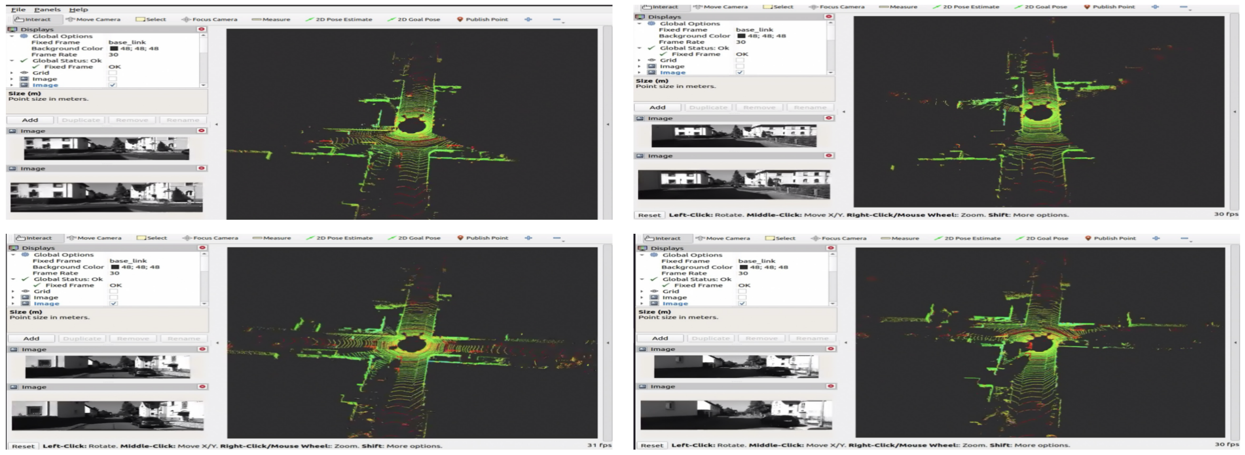
Global Options
Fixed Frame base_link
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK

FIGURE 3. KITTI dataset Streamed using ROS2 Foxy

FAST algorithm was the best in terms of latency but the keypoint match was very bad. This would in turn affect the performance of the odometry. On the other hand BRISK algorithm was the best in terms of keypoint match but it takes almost 240ms just to find keypoints in one image. Along with keypoint matching it takes almost 260ms which is infeasible for real time application.

We then implement the keypoint-matching algorithm between two frames based on the K-nearest neighbours algorithm. Using Shi-Tomasi, the feature extraction and matching took about 30 ms. A result of matching between two consecutive frames, is shown in figure 4. Keypoints in fig. 4 (a) are matched with the keypoints present in fig. 4 (b). This keypoint extracting and matching algorithm is very efficient and hence makes real-time visual odometry possible.

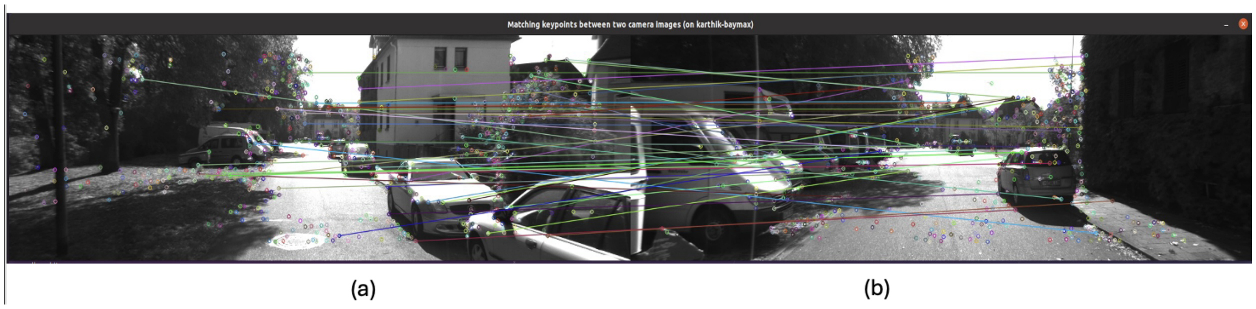
Matching keypoints between two camera images (on karthik-baymax)
(a)
(b)

FIGURE 4. Keypoint matching between two consecutive frames.

The depth map estimation block is implemented. We take the lidar point cloud and project is on the camera image, using the transformation matrix provided in the dataset. The projection is shown in figure 5 (a). 5 (b) shows the depth value assignment to each point. We take the average of three nearest neighbour to assign the depth. Different values of neighbours can be taken. We did not do a comparative analyze of that, due to time constraints.

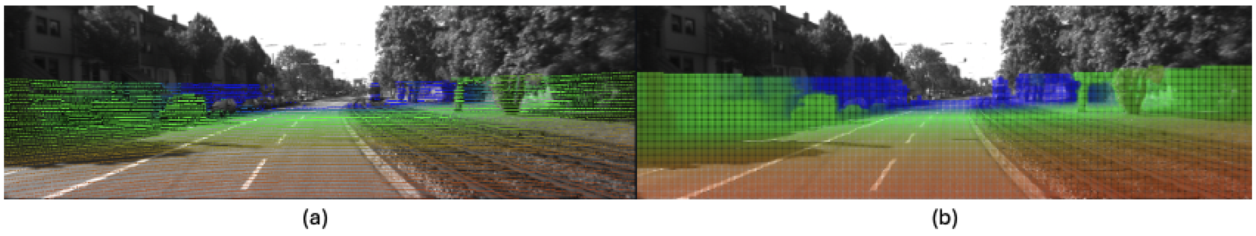

FIGURE 5. Lidar point cloud on a) Image b) 3 NN averaged

Finally the frame to frame motion estimation is done. We use two sequences (0 and 2) for our evaluation. Fig. 6 shows the ground truth trajectory and the trajectory estimated using the Visual Odometry algorithm. Fig. 6 (a) corresponds to the first sequence while Fig. 6 (b) corresponds to the second sequence. We can see that the estimated trajectory follows the ground truth pretty well. Although for the first sequence we can see some drifting error. This is a usual problem with visual based odometry. To overcome this we can do some sort of filtering (like Kalman filtering) by making use of the GPS data.

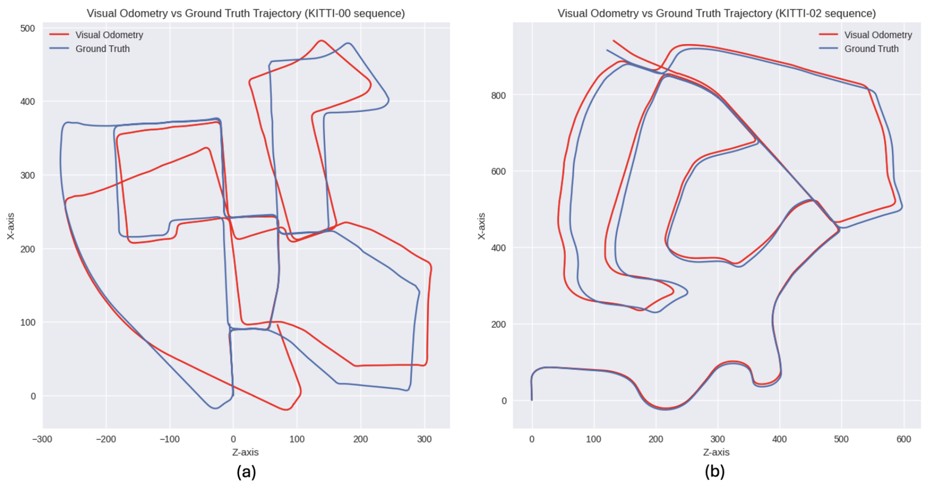

FIGURE 6. Visual Odometry Estimation vs Ground truth a) Sequence 00 b) Sequence 02

We present some mathematical evaluation metrics in Table 2 The rotational and translational errors between the ground truth and the estimates are presented. The average rotational RMSE we get is, 2.5129 and the average rotional

Sequence	**Translational RMSE** (%)	**Rotatonal Error**(deg/m)	Sequence Length
00	2.9227	0.0148	4540
02	2.1031	0.0095	4660

TABLE 2. Different evaluation metric for different sequences

error is 0.01215.

7. DISCUSSION

We see that visual odometry algorithm that we implemented produce good tracking results and is efficient for real-time application. We should evaluate the algorithm on several other sequence to be sure. We saw that the results for sequence *00* are not that great due to drifting. To overcome this the obvious next step would be to include lidar sensor's reading into motion estimation. We could potentially use other sensors as well to get some smoothing estimates and then do the motion estimation. Another drawback of just visual odometry is that the algorithm will give poor estimates in low light environment. To overcome this, again, inclusion of Lidar sensor becomes a crucial aspect.

8. CONCLUSION

In summary, we implemented an efficient Visual Odometry algorithm in ROS2. Although visual odometry task in not new, but our goal was to write an efficient algorithm in ROS2. Most of the contributions we saw in literature are done on ROS1 which, if not now, will be outdated in a few years. We implemented a dataset streamer that helps us to visualize the KITTI dataset sequences. We compared the different keypoint extraction and matching algorithms. We saw that we had to trade off between latency of calculation and accuracy of the solution. To create more robust motion estimators we should fuse different sensor data.

9. ACKNOWLEDGEMENT

We would like to thank Professor. Pratik Chaudhari and the teaching staff for their support throughout this course project.

REFERENCES

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," pp. 3354–3361, 2012.

[2] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE, 2007, pp. 225–234.

[3] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *2011 international conference on computer vision*. IEEE, 2011, pp. 2320–2327.

[4] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, "Real-time 3d visual slam with a hand-held rgb-d camera," in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, vol. 180, 2011, pp. 1–15.

[5] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense rgb-d mapping," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5724–5731.

[6] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *Robotics Research: The 15th International Symposium ISRR*. Springer, 2017, pp. 235–252.

[7] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The international journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[8] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "Accelerated-kaze features," in *European Conference on Computer Vision*. Springer, 2012, pp. 214–227.

[9] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2548–2555.

[10] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*. Springer, 2006, pp. 430–443.

[11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[12] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[13] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: low-drift, robust, and fast," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2174–2181, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:6054487

[14] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[15] P. J. Huber, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, pp. 73–101, 1964.

[16] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074