CSCI 599 Stereo Visual Odometry

Bhuvan Tej Kanigiri
MS in Mechanical Engineering
bhuvantejkanigiri@mines.edu

Abstract

*Visual Odometry system is widely used in robotics for localizing the robot in order to make a determined decision for desired trajectories. This project addresses the goal to develop a Stereo Visual Odometry (**SVO**) system, which estimates the pose of the camera rig at each instance of the live video feed. The system will be applied to a standard stereo data sets and custom made dataset taken from campus building. The advantage of this project is that the scale was achieved using the Stereo cameras and hence, the need of ground truth is not necessary.*

1. Introduction

Recent development in computer vision, brought a lot of surge in solving the localization problems in different ways and Visual Odometry (VO) is one of the prime ways to achieve it. Particularly automotive industry is pushing the borders in this field due to the necessity of achieving a high accuracy with cheaper sensors such as cameras. The best way to implement VO is by utilizing stereo cameras as it does not have any scale factor issue. VO means using visuals or images as sensor data to predict the pose over time. It was first implemented real-time on a robot by Hans Moraveck at NASA Jet Propulsion Laboratory for Mars rover project in 1980[1]. The pose of the robot was estimated using visual input alone, from there on NASA completely dominated the VO research for the next two decades. Later the word ”Visual Odometry” was coined by Nister in 2004[2].

Currently VO is heavily used in robotics, wearable computing, augmented reality and autonomous vehicles. However there are still some challenges such as robustness to lighting conditions, lack of features, and non-overlapping images.

The biggest challenges faced in VO is whenever there is a textureless environment, it would be highly impossible for the algorithm to detect the pose and orientation. But recent development in computer vision is trying to overcome these problems by detecting lines and points [3], as lines have proven to be an interesting alternative to points in man-made environments.

VO is the subsection of visual SLAM (vSLAM) which stands for visual Simultaneous Localization And Mapping. It can be expressed as **VO + Loop closure**. While VO just takes care of the pose estimation, vSLAM also does loop closure followed by pose optimization. This uses bag of words (BoG) technique to search for the loop closure and executes bundle adjustment for pose optimization. State-of-the-art vSLAM algorithms are ORB-SLAM [4] and OPEN-VSLAM [5].

In this study, a stereo test rig was built and was used to captured a custom dataset in the indoor environment. The goal of this project was to implement a SVO algorithm, test it on KITTI dataset [6] and custom indoor dataset.

2. Related work

The general approaches to VO can be broadly categorized into two types based on the number of cameras i.e., monocular and stereo. Another classification is based on the techniques used for collecting feature points i.e., feature matching or feature tracking.

RTSVO algorithm[7]: RTSVO stands for Real-Time Stereo Visual Odometry. In 2008, Howard [7] implemented **RTSVO** for autonomous ground vehicles which uses feature matching rather than feature tracking with stereo inlier detection method and motion estimation is done using the standard Levenberg- Marquardt least-squares algorithm. The main application of this algorithm is to be used in real-time and low-latency methods.

Hirschmuller H. algorithm[8]: This method calculates the motion without any prior knowledge or prediction about the feature location, which helps in attaining lower frame rates. Used euclidean constraints are used to incrementally select inliers in place is using statistically methods that can handle inliers.

Visual Odometry tutorial - I & II[9]: The tutorial provides a comprehensive understanding of the visual odometry implementation. Part-I describes about camera modeling and calibration, main motion estimation pipelines for both monocular and binocular scheme. Part-II elaborates on feature matching, robustness and few applications of SVO.

Iterative closest point (ICP)[10]: This paper implements VO by extracting the corners of objects in the images using Harris corner detector and SIFT for obtaining descriptors using K-nearest neighbor match. Followed by triangulation method to obtain 3D points from a pair a 2D points and used ICP for motion estimation. The exact point correspondences of the 3D points are not known apriori for the ICP to obtain good results.

Probablistic combination approach[11]: Recently in 2016, Gomez-Ojeda proposed a novel probabilistic approach to stereo images based on the combination of both point and line segment which robustly estimated the pose in a wide variety of scenarios using less number features on the image in a textureless environment. It uses the non-linear minimization of the projection errors of both point and line segment features.

All of the above mentioned algorithm follow more or less a similar pattern in getting pose of the camera, but using different techniques. The goal in this project get the commonality from them, and try to implement a basic stereo model.

3. Approach

Goal: In this study, a minimal prototype of the stereo pose estimation algorithm is to be implemented on an indoor image sequence captured by a custom stereo camera rig. The implementation of the stereo pose estimation algorithm is based on the RTSVO[7] algorithm.

3.1. Problem statement

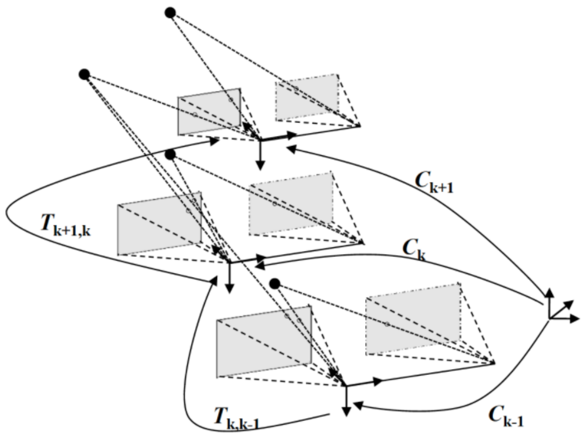


Figure 1: Sample image sequences [9]

Input to the pipeline is a sequence of pair of images generated from the left and right cameras which outputs the 3D stereo rig pose information. The stereo rig is made up of two monocular Microsoft HD-3000 model cameras attached five-teen centimeters apart which are placed parallel to each other on a 3D printed base. The baseline is ensured to not change during the data acquisition. The data from the two cameras are stored simultaneously through USB. The algorithm computes the pose by reading the captured frames and updating the rotation and translation in the global frame.

From the Fig.1 the basic understanding of the image sequence format is as follows: For the first iteration, the left and right images are stored as the "previous" sequence. For the following sequences the another set of left and right images are stored as the "current" sequence. The pose is now estimated using these two sequence. At the end of every iteration the current sequence becomes the previous sequence and a new set of images are stored to be the current sequence.

C_{k-1} = Previous instance camera center

C_k = Current instance camera center

C_{k+1} = Next instance camera center

$T_{k-1,k}$ = Transformation of camera from previous step to current instance

$T_{k,k+1}$ = Transformation of camera from current step to next instance

Equipments used:

Model name: Microsoft LifeHD-3000

Video dimensions: 640 x 480 pixels

Frames per second: 30

Baseline: 14.224 cms

Connection: USB

Laptop: i7 9th Gen, 2.60 GHz Hexa-core, GeForce® GTX 1650

3.2. Hardware rig setup

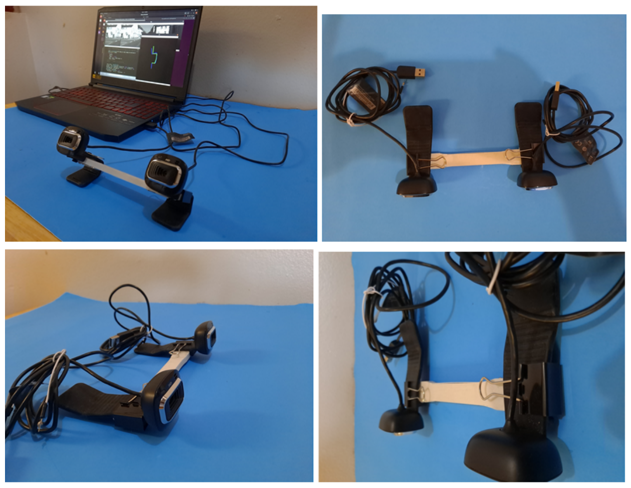


Figure 2: Algorithm flowchart

Fig.2 shows the stereo rig - on the either side with Microsoft life HD - 3000 webcam connected to the laptop. In the centre, a 3D printed base holds the cameras on either ends firmly attached by clips. This setup ensures that the baseline or orientation is not changed during data acquisition.

3.3. Camera calibration

The goal of camera calibration is to accurately measure the intrinsic and extrinsic parameters, where extrinsic parameters explains the mutual position and orientation between left and right images and intrinsic parameters provides the intrinsic values that are confined to the particular camera. The intrinsic parameters would differ from one camera to another camera. In order to find these parameters, 35 to 40 images of checkerboard with known block dimensions are captured and processed by using OpenCV function `findChessboardCorners` and `calibrateCamera`. The camera matrix and distortion coefficients are obtained as output [12]. The extrinsic parameters for KITTI dataset are already provided [6]. The Baseline of the setup was calculated using Aruco markers and hence the extrinsic parameters for the custom dataset was found.

3.4. Algorithm

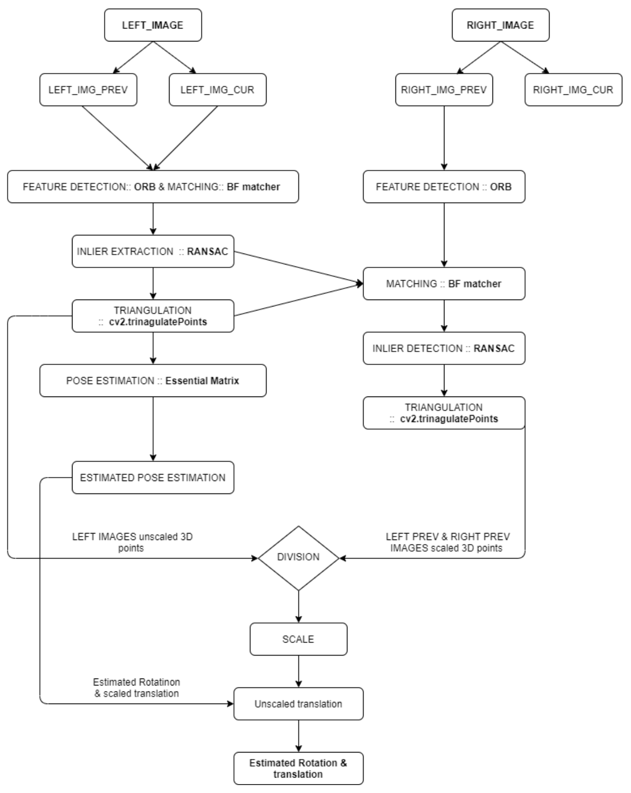

LEFT_IMAGE
RIGHT_IMAGE
LEFT_IMG_PREV
LEFT_IMG_CUR
RIGHT_IMG_PREV
RIGHT_IMG_CUR
FEATURE DETECTION:: ORB & MATCHING:: BF matcher
FEATURE DETECTION :: ORB
INLIER EXTRACTION :: RANSAC
MATCHING :: BF matcher
TRIANGULATION :: cv2.trinagulatePoints
INLIER DETECTION :: RANSAC
POSE ESTIMATION :: Essential Matrix
TRIANGULATION :: cv2.trinagulatePoints
ESTIMATED POSE ESTIMATION
LEFT IMAGES unscaled 3D points
LEFT PREV & RIGHT PREV IMAGES scaled 3D points
DIVISION
SCALE
Estimated Rotatinon & scaled translation
Unscaled translation
Estimated Rotation & translation

Figure 3: Algorithm flowchart

Fig.3 talks about the high level idea of the algorithm using a flowchart. Initially in the feature detection section, all the important details from the images are extracted. Further, 3D points in the triangulation section are obtained from previously detected 2D points. Finally, rotation and translation in the pose estimation section is obtained using those keypoints. The following section describes the algorithm in more detail:

3.4.1 Images sequences

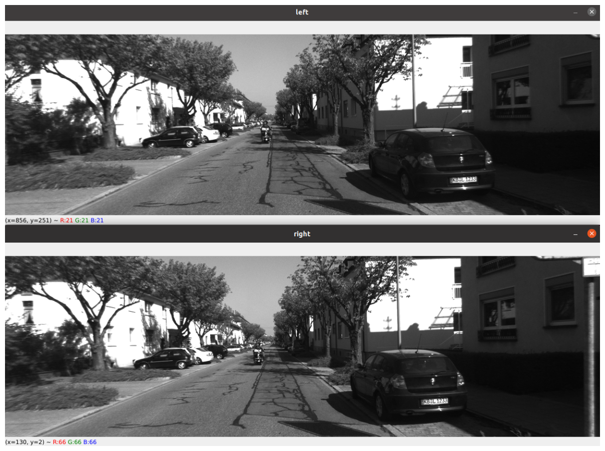


Figure 4: KITTI :: Raw Image sequence

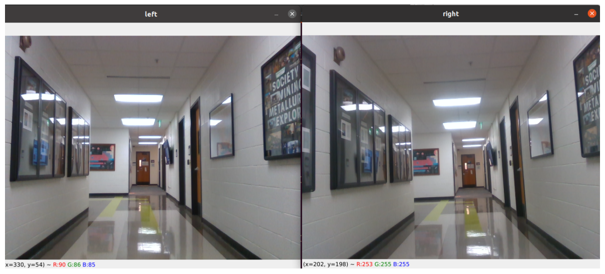


Figure 5: CUSTOM :: Raw Image sequence

Initially, the left and right image paths are stacked and loaded for every iteration. Two sets of images i.e., one is left current and right current images and the other one is left previous and right previous, put together four images are loaded and used to estimate the pose.

3.4.2 Feature detection and matching

The raw images are used for detecting the features and descriptors. Used FAST detector to detect the keypoints in the scene and used Oriented Fast and Rotated Brief (ORB) to obtain the descriptors for the respective keypoints. In order

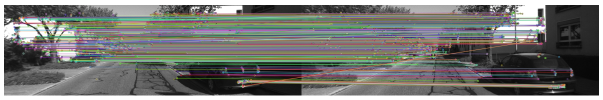


Figure 6: Inlier matches between two left consecutive images

to match the descriptors, Brute Force matcher is used with the hamming distance as a criteria. After obtaining the desired matches, the good keypoints are filtered using Lowe's [13] ratio test. Based on the finalized matches that are obtained, the corresponding keypoints, descriptors and point coordinates are extracted.

3.4.3 Motion estimation from 2D to 3D

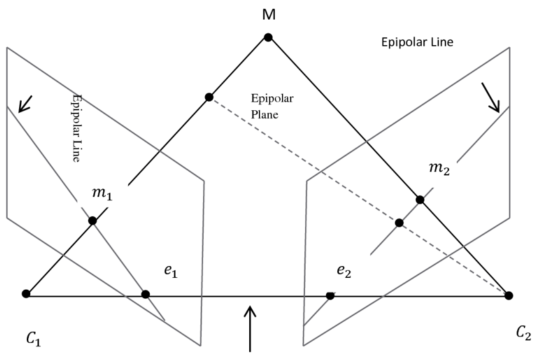


Figure 7: Epipolar lines: [14]

The essential matrix and mask are obtained using OpenCV's `findEssentialMat` function by passing reduced matched point coordinates of left consecutive frame and camera intrinsic parameters. Essential matrix basically encodes the epipolar geometry of two different views. The rotation and translation between left consecutive frames are determined using OpenCV's `recoverpose` function using calculated essential matrix and reduced point coordinates. The inlier keypoints, descriptors and coordinate points are estimated by performing RANSAC using the obtained mask while finding the Essential Matrix.

Assuming, if the vectors C_1M_1, C_2M_2, and C_1 C_2 are co-planar, then,

$$
\begin{aligned}
(C_1 * M_1).(C_1 * C_2 x C_2 * M_2) &= 0 \\
M_1.(tXRM_2) &= 0 \\
M_1^T[t]_x RM_2 &= 0 \\
M_0^T EM_2 &= 0
\end{aligned}
$$

$$Where, E = [t]_x R$$

Extracting R and t from E

Out of the four different combinations for R and t of one essential matrix, one of the combination estimates the actual R and t pair. This actual pair will be returned from OpenCV's `findEssentialMat` function. The singular vector decomposition obtained from the essential matrix are stored in U, V, and W matrix.

$$
\begin{aligned}
E_k &= t_k * R_k \\
R &= U(\pm W^T)V^T \\
\hat{t} &= U(\pm W)SU^T
\end{aligned}
$$

Where,

$$W^{\mathrm{T}} = \begin{bmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.4.4 Triangulation

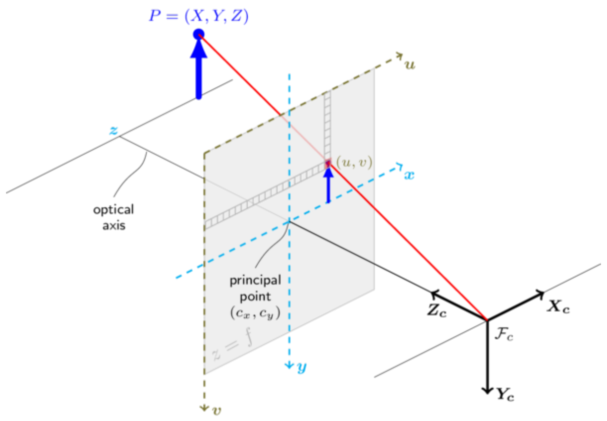


Figure 8: Triangulation method :: source: csdn blog

$$
\begin{aligned}
Z &= (f * b)/d. \\
X &= (x_1 - p_{\mathrm{cu}}) * b/d. \\
Y &= (y_1 - p_{\mathrm{cv}}) * b/d.
\end{aligned}
$$

$$
\begin{aligned}
\text{Where, } X, Y, Z &= \text{coordinates of 3D point} \\
x_1, y_1 &= \text{left coordinates of feature keypoint} \\
p_{\mathrm{cu}}, p_{\mathrm{cv}} &= \text{principal axis coordinates} \\
b &= \text{baseline} \\
d &= \text{disparity} \\
f &= \text{focal length}
\end{aligned}
$$

Triangulation is a process of finding 3D points from the 2D feature point correspondences. Initially, the 3D points are obtained from left previous and left current frame where left previous is considered as the reference frame. The 3D points are calculated using OpenCV's `triangulatepoints` function by passing inlier 2D feature point correspondence and projection matrix between the images. The obtained 3D points have unknown scale as that of the estimated pose.

3.4.5 Estimating the Known Scale

The left previous image (reference image), and the right previous frame are considered for feature detection and triangulation. The feature parameters for the left previous frame was already obtained in the previous section. The feature parameters are now obtained for the right previous frame and the desctriptors of these two images are matched. With the help of Lowe's ratio test, the good matches are obtained and the respective parameters for both the previous frames are stored. Using the good matches, 3D points of left previous and right previous images are obtained using triangulation. The Essential matrix and mask for the left and right previous images are found and the inlier 3D points are obtained. The obtained 3D points has known scale for pose estimation. The above mask is also used in the 3D points obtained in the previous section to obtain the point corresponding between the two sets of 3D points.

3.4.6 Estimating the Unknown Scale

The generated 3D points from monocular combination and stereo combination can be used for calculating scale. The length of the the 3D point sets are ensured to be the same. Initially, the magnitude of 3D points from monocular, for example $Pmono$, and magnitude of 3D point from the stereo, for example $Pstereo$, was calculated. In order to obtain the scale, for every 3D point correspondence, the $Pmono's$ magnitude was divided by the $Pstereo's$ magnitude. All the individual scales are stacked up in an array and final scale was obtained by calculating the mean of the array. This obtained scale will be the scale of the 3D points and translation which was previously obtained.

3.4.7 Pose wrt the world coordinates

In order to obtain the current transformation of the stereo rig with respect to the world coordinates, the obtained Rotation and Translation are multiplied with the previous Rotations and translation in the form of,

$$R_{current} = R_{previous} * R$$
$$t_{current} = t_{previous} + scale * (R_{previous} * t)$$

Where t is the translation, R is rotation and the $scale$ is the scale which was obtained from section.3.4.6.

3.5. PSEUDO CODE

```
MAIN BLOCK
Input: The sequence of image pairs.
Output: Set of poses.
  1.: Load the camera parameters in the program i.e., focal length, principal axis, baseline, distortion parameters.
  2.: From the stereo rig camera, read the left & the right image and store them as leftIMG and rightIMG respectively for the left & right camera.
  3.: Perform stereo rectification, i.e., warp the left/right images such that the epipolar lines are aligned with the image rows.
For each left & right frame read:
    a.: Store the first instance of left & right images as previous frames and the next frame consecutive frames as current frames.
    b.: CALL "FEATURES" function.
        Input: Left previous and left current frames.
        Output: Keypoints, descriptors.
    c.: CALL "REDUCED MATCHES" function.
        Input: Left previous and left current frames key-points and descriptors.
        Output: Reduced keypoints, descriptors & co-ordinate key-points.
    d.: CALL "POSE ESTIMATION" function.
        Input: Reduced keypoints, reduced descriptors.
        Output: Rotation & translation (only direction is known) of consecutive left frames and inlier points.
    e.: CALL "REDUCED FEATURES" function.
        Input: Reduced keypoints, reduced descriptors and inlier points.
        Output: Inlier key-points, inlier descriptors and inlier points.
    f. CALL "TRIANGULATION" function.
        Input: Inlier co-ordinate points, camera projection matrices (left previous and left current).
        Output: 3D points.
    g. CALL "REDUCED MATCHES2" function.
        Input: Inlier co-ordinate points, keypoints and descriptor points of left image, right previous image keypoints and keypoints, camera projection matrices (left previous and left current) and monocular unscaled 3D points .
        Output: Inlier 3D points of left consecutive images and left-right previous.
    h. CALL "GET SCALE" function.
        I Input: Inlier 3D points of left consecutive images and left-right previous.
        II Output: Scale.
  4. Calculate the cumulative rotation and translation: say
```

cumulative translation & cumulative rotation. The formula is as follows:
Cumulative translation = cum_t
Cumulative rotation = cum_R
Rotation = R
Translation = t
Scale = S

a. cum_t = cum_t + S * (cum_R @ t)

b. cum_R = R @ cum_R

5. Also, to check whether the car has moved significant distance over time, calculate the net translation for consecutive translations, for example $deltatranslation$.

if $deltatranslation$ greater than some threshold:

a. Update the translation & rotation as the true estimated rotation & translation.

b. Draw the above-obtained R & t to draw on the GUI black image(only the Z, X).

FEATURE BLOCK
Description: This function takes left and right images and detects the key distinct features using FAST (Features from Accelerated Segment Test) & ORB (Oriented FAST and Rotated BRIEF) features detectors, these features contain key points & descriptors of left and right images.
Input: Left previous and left current frames.
Output: Keypoints, descriptors & matches of both left consecutive frames.

1. Initialize the FAST and ORB feature detectors.

2. Using the FAST detector get the key points of the image frame. Do it for both images.

3. Now, using ORB taking key points and image as input, it computes the descriptors. Do it for both images.

5. RETURN Keypoints, descriptors.

REDUCED MATCHES:
Description: Based the matches from the left consecutive image descriptors, filter out our feature parameters, i.e., key-points, descriptors and points.
Input: Keypoints, descriptors of both left consecutive frames.
Output: Reduced keypoints, descriptors & co-ordinate key-points.
1. After obtaining the good matches using BFmatcher and imposing lowe's ratio test contraint [15], good matches are calculated - using this extract the the only feature parameters (keypoints, descriptors and point coordinates) which are in the matches.
2. Finally RETURN all the reduced feature parameters in numpy array format.

POSE ESTIMATION BLOCK:
Description:
Input: Reduced keypoints, reduced descriptors.
Output: Rotation & translation (only direction is known) of consecutive left frames and inlier points.
For loop of length(matches):

a. Essential matrix(E) and inlier mask is generated from RANSAC using cv2.findEssentialMat.

b. Using the E value and reduced feature points, rotation and translation are calculated.

c. RETURN Rotation and translation.

Note: While calculating the left consecutive images, exact rotation value of camera pose and scaled up translation in pose.
REDUCED FEATURES
Description: This function gives us the inlier feature parameters, using mask.
Input: Reduced keypoints, reduced descriptors and inlier points.
Output: Inlier keypoints, inlier descriptors and inlier points.
1. Obtain mask using findessentialMat, using this mask which consist zeros and ones, basically the mask variable is filled with ones wherever there is a inlier points and zeros for outliers.
2. Extract all the required inliers.
4. RETURN inlier feature parameters.

TRIANGULATION BLOCK
Input: Inlier co-ordinate points, camera projection matrices (left previous and left current).
Output: 3D points.
For each match in matches:

a. Calculate both the coordinates of the train image and query image, that is left & right images.

b. By subtracting corresponding x values in the left image and right image, the disparity is calculated

c. If the disparity is not None:

I calculate the Z coordinate, using formula Z = (focal length* baseline)/disparity.

II Calculate the X coordinate, using formula X=(left x coordinate - x principal axis)*baseline/disparity.

III Calculate the Y coordinate, using formula Y=(left y coordinate - y principal axis)*baseline/disparity.

2. Append all these X, Y, Z in a list.
3. RETURN X, Y, Z.

REDUCED MATCHES2:
Description: This is the function where stereo, comes into picture - Consider right previous image and left previous image as stereo combination, similar things are done to get the monocular, except the pose estimation, triangulation section is executed to save all the unscale 3D points generated and already obtained obtained 3d scaled points from left.
Input: Inlier co-ordinate points, keypoints and descriptor

points of left image, right previous image keypoints and keypoints, camera projection matrices (left previous and left current) and monocular unscaled 3D points.
Output: Inlier 3D points of left consecutive images and left-right previous.
1. Find the matches between left inlier descriptors and right previous image.
2. Reduce the left 3D points to inlier 3D points and also call the triangulation function for finding the 3D points of left and right
3. Also make sure that, all the six coordinate point should have the same length.
4. RETURN the six 3D points in numpy array format.

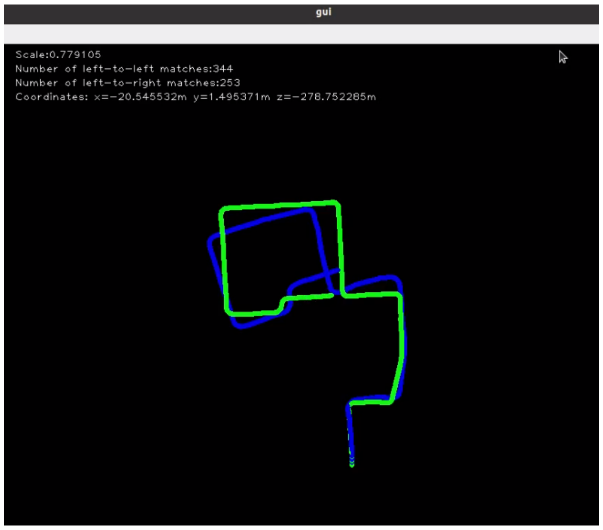
GET SCALE:
Description: The 2 set of coordinate points are provided to find the scale between them.
Input: Inlier 3D points of left consecutive images and left-right previous.
Output: Scale.
For length of the 3D points:
  1. Capture the individual six values of the two sets of 3d points.
  2. Take the magnitude of unknown scaled 3D points and save them as magnitude of generated from monocular.
  3. Similarly, take the magnitude of unscaled 3D points as magnitude of generated from stereo.
  4. Now, to get the scale, divide the magnitude of mono by magnitude of stereo.
  5. RETURN scale.

4. Experiments and Results

Dataset :: KITTI [6]

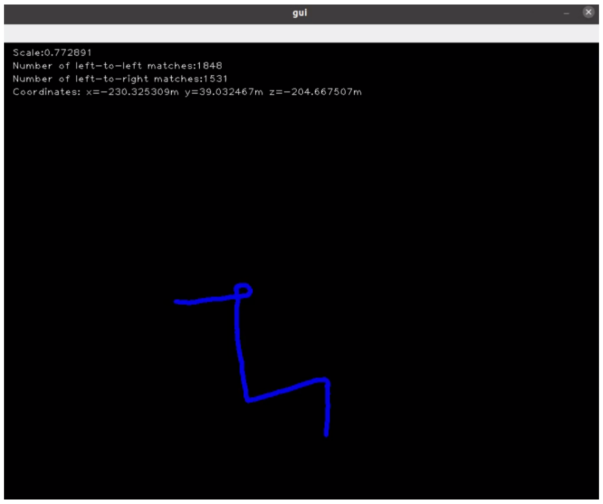
Fig.9 are the results generated from the Kitti stereo dataset, the green line is the ground truth and the blue line is the algorithm generated path from images.

Dataset :: Custom Fig.10 are the results generated from the brown dataset, the algorithm works well at different secnaio's including loop, turnings etc.



```
Scale:0.779105
Number of left-to-left matches:344
Number of left-to-right matches:253
Coordinates: x=-20.545532m y=1.495371m z=-278.752285m
```

Figure 9: SVO results on KITTI dataset with sequence zero



```
Scale:0.772891
Number of left-to-left matches:1848
Number of left-to-right matches:1531
Coordinates: x=-230.325309m y=39.032467m z=-204.667507m
```

Figure 10: SVO results on Custom dataset

5. Discussion

In the generic dataset, the stereo algorithm performs well in most of the secnaio's except few cases (mentioned in limitations section.6), the challenging thing in this project was to get a correct scale from the left and the right camera at every instance, speeding the car would be impacting factor in getting the correct scale.

In the custom dataset, the results look promising, even though the ground truth was not calculated for custom dataset, the results after a view approximate looks good.

The amount of features, length of the matches matters a lot while computing the pose, it is good to have more matches than required, minimal matches required is 7, as it uses seven-point algorithm to compute the pose. Outliers removal plays a crucial role in the getting good results, In some cases few outliers can be inevitable. Better outlier removal method leads to the correct third correspondence.

6. Applications and future work

Visual odometry has been used in many applications, mainly in mobile robotics particularly in space robotics - where there is a lot of drift from the wheels visual odometry comes in handy. Latest development in the computation power made the visual odometry to work at real-time effortlessly.

Another place of application is in air and underwater navigation where the robot can't rely on GPS for pose estimation. Used in coral-reef inspection [16] also.
In the domains like automotive - additional things like driver lane assistance systems, vision based braking system, VO would play a prominent role. It is considered as a cost-effective solution compared with LIDAR [17] and also considered as the most inexpensive solution for underwater localization systems, as robot don't get GPS in underwater vehicles. Furthur, kalman filter can improve the accuracy and would refine the visual odometry pose.

Limitations: Our algorithm for the standard KITTI dataset seems to work well in most of the instances, except while taking waited turns, low lighting areas, sudden altering in the speed and power turns. In the Fig.9, due to the waited turn to the right the pose got disturbed, as a result, the error increase over time - even though the pose was correctly computed later. For the custom dataset the algorithm is working well at different secnaio's - their might be error in scale, but ultimately this algorithm achieved best results for both custom and generic dataset. There are there few drawbacks as mentioned above would be more concentrated in future work.

In Future work, the pose could be improved by loop closure bundle adjustment, after loop closure , the algorithm tries to re-optimize the pose and bundle adjustment keeps a track of last n images thereby improve the estimated pose. Better camera quality and stabilization methods should be used. High computation and use of deep learning would improve the results.

7. Conclusion

The paper tried to cover an extensive algorithmic review on Visual Odometry. Section.1, outlines the Introduction and when and where can VO applied based on our application. Section.2 discussed the related work and how Visual odometry evolved to VSLAM with the addition of loop closure.Finally it talks about the how the algorithm is build, as our main goal of this project is having an algo-centric approach. Finally on closing note, this took a lot of effort in accomplishing the results at every stage.

8. Acknowledgement:

I would like to thanks my minor advisor Prof. Dr. William Hoff for helping me every week and guiding in every which way possible and thanks to all my friends who helped in this process. It wouldn't be possible without them. Thanks for reading this paper.

References

[1] M. Maimone, Y. Cheng, and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *J. Field Robotics*, vol. 24, pp. 169–186, 03 2007.

[2] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[3] T. Koletschka, L. Puig, and K. Daniilidis, "Mevo: Multi-environment stereo visual odometry," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 4981–4988.

[4] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[5] S. Sumikura, M. Shibuya, and K. Sakurada, "OpenVSLAM: A Versatile Visual SLAM Framework," in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM '19. New York, NY, USA: ACM, 2019, pp. 2292–2295. [Online]. Available: http://doi.acm.org/10.1145/3343031.3350539

[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[7] A. Howard, "Real-time stereo visual odometry for autonomous ground vehicles," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 3946–3952.

[8] H. Hirschmuller, P. R. Innocent, and J. M. Garibaldi, "Fast, unconstrained camera motion estimation from stereo without tracking and robust statistics," in *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.*, vol. 2, 2002, pp. 1099–1104 vol.2.

[9] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]." *IEEE Robotics Autom. Mag.*, vol. 18, no. 4, pp. 80–92, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/ram/ram18.html#ScaramuzzaF11

[10] K. U. Pavan, M. P. V. Sahul, and B. T. V. Murthy, "Implementation of stereo visual odometry estimation for ground vehicles," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, 2017, pp. 1173–1177.

[11] R. Gomez-Ojeda and J. Gonzalez-Jimenez, "Robust stereo visual odometry through a probabilistic combination of points and line segments," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2521–2526.

[12] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[13] A. Kaplan, T. Avraham, and M. Lindenbaum, “Interpreting the ratio criterion for matching sift descriptors,” vol. 9909, 10 2016, pp. 697–712.

[14] D. Bappy and H. Rahman, “A study in 3d structure detection implementing forward camera motion,” Ph.D. dissertation, 05 2012.

[15] F. Fraundorfer and D. Scaramuzza, “Visual odometry: Part ii - matching, robustness, and applications,” *IEEE Robotics Automation Magazine - IEEE ROBOT AUTOMAT*, vol. 19, pp. 78–90, 06 2012.

[16] M. Dunbabin, J. Roberts, K. Usher, G. Winstanley, and P. Corke, “A hybrid auv design for shallow water reef navigation,” *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2105–2110, 2005.

[17] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. Ismail, “Review of visual odometry: types, approaches, challenges, and applications,” *SpringerPlus*, vol. 5, 2016.