CS/RBE 549 Computer Vision, Fall 2018

Project Report

Team 14: VISUAL ODOMETRY

Abstract

Visual Odometry is the process of incrementally estimating the pose of a vehicle using the images obtained from the onboard cameras. Its applications include, but are not limited to, robotics, augmented reality, wearable computing, etc. In this work, we implement stereo visual odometry as well as monocular visual odometry using images obtained from the KITTI Vision Benchmark Suite[1] and present results of both approaches. We implement stereo visual odometry using 3D-2D feature correspondences while we use 2D-2D feature correspondences for the monocular case. We find that between frames, using a combination of feature matching and feature tracking is better than implementing only feature matching or only feature tracking. Also, we find that monocular odometry gives lesser error than stereo only if we have a reliable source to obtain absolute scale, while stereo odometry gives a reliable trajectory without the need of an absolute scale.

Table of Contents

List of Figures

1. Introduction

Odometry is the use of sensors and actuators to estimate the change in position of a robot over time[2]. Traditionally for wheeled robots motion estimation was performed with the use of rotary wheel encoders. However, for mobile robots with non-standard locomotion like legged robots wheel odometry cannot be used.[2] Moreover, odometry tends to suffer from precision problems because wheels tend to slip or slide which is inferred as a non-uniform by the encoders. This problem worsens when the surface is not smooth or uneven. Therefore, odometry readings become unreliable and these errors accumulate and compound over time.

In robotics and computer vision, visual odometry is the process of computing the pose of a robot by analyzing the associated camera images.[3] Visual odometry also helps in localizing robots with considerable amount of precision which operate in GPS denied environments (eg. indoors, places with no GPS coverage like mars). It has been used in a wide variety of robotic applications, such as on the Mars Exploration Rover[3]. The idea of Visual Odometry was first introduced for planetary rovers operating on Mars – Moravec 1980.

2. Overview

2.1. Dataset used

- **KITTI Vision Benchmark Suite[1]:** The KITTI Odometry dataset was used in our project. The dataset contains 20 sequences of stereo video sequences in color as well as greyscale.

- **Calibration Files:** Camera and projection matrices are provided to the user with every video sequence in the form of a calibration file. Time stamps of every frame is also provided.

- The camera orientation is as follows and shown in Fig. 1 below:
  - The X-axis is parallel to the ground and towards the right of the driver.
  - The Y-axis is perpendicular to the ground and facing downwards.
  - The Z-axis is is parallel to the ground and facing forward

Fig. 1: KITTI dataset recording platform: VW Passat station wagon is equipped with four video cameras (two color and two grayscale cameras), a rotating 3D laser scanner and a combined GPS/IMU inertial navigation system. (Image taken from the KITTI dataset paper)

2.2. High level approach for Visual Odometry

The typical pipeline of Visual Odometry is as follows[4]:

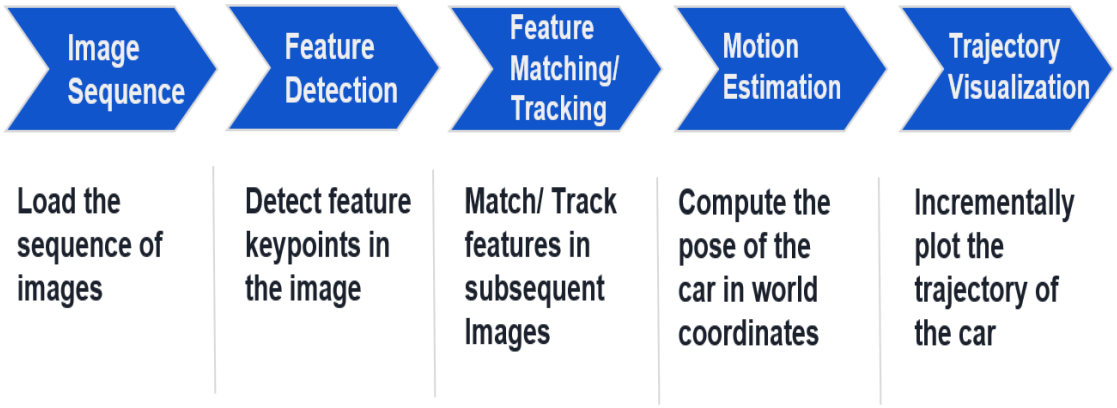


Fig. 2: Pipeline of Visual Odometry

2.3. Our Approach:

We have implemented both Monocular and Stereo Visual Odometry and our approach is as follows:

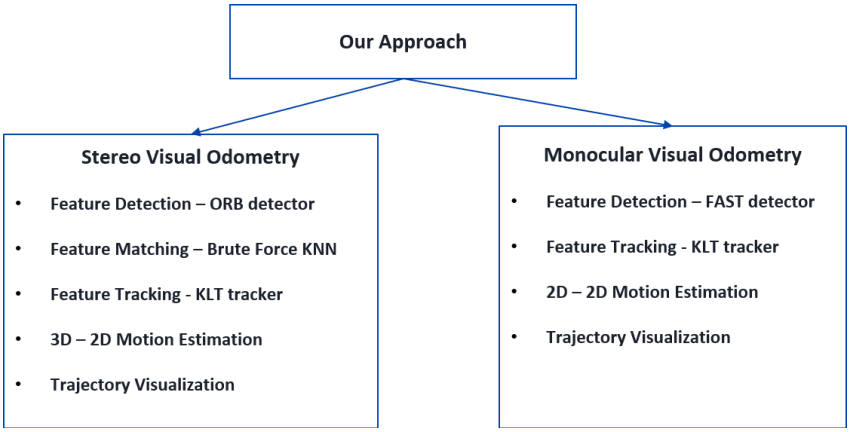


Fig. 3: Our approach of implementing Visual Odometry

3. Detailed Methodology

3.1 Stereo Visual Odometry

3.1.1. Problem Formulation

Input

Our input consists of a stream of gray scale or color images obtained from a pair of cameras. This data is obtained from the KITTI Vision Benchmark Suite. Let the pair of images captured at time k and $k+1$ be $(I_{l,k}$

$I_{r,k}$) *and* ($I_{l,k+1}$, $I_{r,k+1}$) respectively. The intrinsic and extrinsic parameters of the cameras are obtained via any of the available stereo camera calibration algorithms or the dataset

Output
For every stereo image pair we receive after every time step we need to find the rotation matrix *R* and translation vector *t,* which together describes the motion of the vehicle between two consecutive frames.

3.1.2. Algorithm Outline

1. Read left ($I_{l,0}$) and right ($I_{r,0}$) images of the initial car position
2. Match features between the pair of images
3. Triangulate matched feature keypoints from both images
4. Iterate:
    a. Track keypoints of $I_{l,k}$ in $I_{l,k+1}$
    b. Select only those 3D points formed from $I_{l,k}$ and $I_{r,k}$ which correspond to keypoints tracked in $I_{l,k+1}$
    c. Calculate rotation and translation vectors using PNP from the selected 3D points and tracked feature keypoints in $I_{l,k+1}$
    d. Calculate inverse transformation matrix, inverse rotation and inverse translation vectors to obtain coordinates of camera with respect to world
    e. The inverse rotation and translation vectors give the current pose of the vehicle in the initial world coordinates. Plot the elements of the inverse translation vector as the current position of the vehicle
    f. Create new features for the next frame:
        i. Read left ($I_{l,k+1}$) and right ($I_{r,k+1}$) images
        ii. Match features between the pair of images
        iii. Triangulate matched feature keypoints from both images
        iv. Multiply the triangulated points with the inverse transform calculated in step (d) and form new triangulated points
    g. Repeat from step 4.a

3.1.3. Reading images

We used the imread function[5] from OpenCV to read the images from the KITTI dataset and converted the images to grayscale since many of the methods used ahead in the approach work only with grayscale images.

3.1.4. Undistortion [15]

Pinhole cameras can have radial and tangential distortions. These are compensated by finding the distortion coefficients during camera calibration. The corrected x, y coordinates are given by the following equations:

For radial distortion,

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

For tangential distortion,

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

$$y_{corrected} = y + [p_1(r^2 + 2y) + 2p_2 xy]$$

Therefore we need to find 5 parameters in all to compensate for distortion.

Distortion coefficients = [k_1 k_2 p_1 p_2 k_3]

In our dataset the images are already undistorted and hence we use the images as provided. In addition to this, we need to find a few more information, like intrinsic and extrinsic parameters of a camera. Intrinsic parameters are specific to a camera. It includes information like focal length (f_x, f_y) optical centers (c_x, c_y) etc. It is also called camera matrix. Once calculated, it can be stored for future purposes. It is expressed as a 3x3 matrix:

$$Camera\ Matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The KITTI dataset providers have provided the Camera Matrix for the various image sequences as well as other parameters as required.

3.1.5. Feature detection[6]

We used the ORB feature detector to detect keypoints in our images. The reason is that it is fast, efficient in detecting features and solves many issues with other detectors like SIFT,SURF, etc. ORB stands for Oriented Fast and Rotated Brief. This algorithm was developed at OpenCV by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in 2011. Unlike SIFT and SURF, ORB is open source and thus free to use.

ORB combines FAST keypoint detector with BRIEF descriptor in additions to making several improvements. It employs FAST to find keypoints and then uses Harris corner measure to find the best N keypoints among them. It also employs a pyramid to produce multiscale-features.

To solve the rotation invariance problem of FAST detectors, it calculates the centroid of the detected patch with our keypoint at the center. The centroid is calculated by an intensity weighting scheme. Now the orientation is determined by the direction of the vector from the keypoint to the centroid.

Next, ORB use BRIEF descriptors. To improve the performance of BRIEF descriptors, ORB rotates them as per the orientation of keypoints. For any given feature set of *n* features, it creates a *2*n* matrix *S* which contains the coordinates of the pixels of the keypoints. It then calculates the rotation matrix of the patch using the orientation Θ and rotates matrix *S* to get the rotated version S_θ

ORB converts the angle to discrete components of 12 degrees and builds a lookup table of precomputed BRIEF patterns. The correctness of the set of points S_θ is ensured by checking the consistency of the keypoint orientation Θ across views.

Fig. 4: Keypoints detected in left and right image pair

3.1.6. Feature Matching

We matched features between left and right images of each pair and used Brute Force Matcher for the same. Brute Force matcher takes the descriptor of one feature in first set of points and matches it with all other features in the second set[7]. We used the KNN matcher with the default k=2 which would return the 2 best matches. Additionally, to extract the good matches out of all the matches returned, we apply Lowe's ratio test[8] with a threshold of 0.7

Fig. 5: Best Feature matches found out by Brute Force matcher

3.1.7. Triangulation of feature points[9]

Whenever we detect key points in images, we can only find the position of that keypoint in the image. It is being assumed that the coordinate frame of the car is same as the coordinate frame of the camera and we are interested in finding the position of the car in real-world coordinate system. For this, we need to find the coordinates of the keypoints in the real world coordinate system. It is quite impossible or hard to get the accurate 3D world coordinates of the keypoints from a monocular camera or a single frame of the scene. Like the human eye, we need a pair of stereo images to accurately find the depth of the keypoints in the world.

For this problem of 3D-2D motion estimation, we have used the stereo KITTI dataset. It provides left and right images of each instance. The keypoints detected in the left and right images are matched as described in the previous section. The best matches among all the matches are chosen so that we can we can find the corresponding 3D points. IN order to find the 3D world coordinates, we need to have complete information about the camera coordinate system. Fig. is the orientation of the camera coordinate system. The z- coordinates comes out of the image plane. The complete method behind this computation is that we first find the disparity of the whole image.

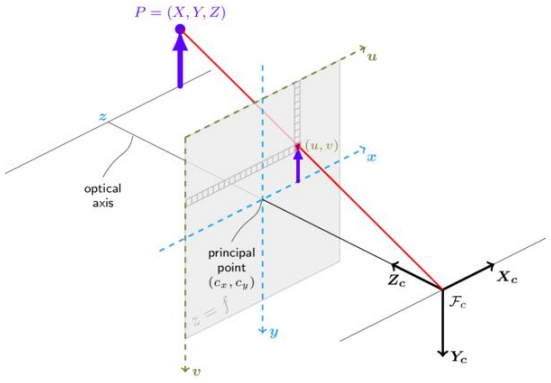


Fig. 6: Camera coordinate system
Courtesy:OpenCV(https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html)

In case of a stereo camera the coordinate system of the camera is assumed to be situated midway between the two cameras. We will also be considering the left image as the reference image throughout the project. If we are considering a horizontal stereo camera, each matched keypoint will have the same y coordinate. Only the x coordinate of the pixels with the keypoints in left and right images will be different. The difference between these two x coordinates is known as the disparity.

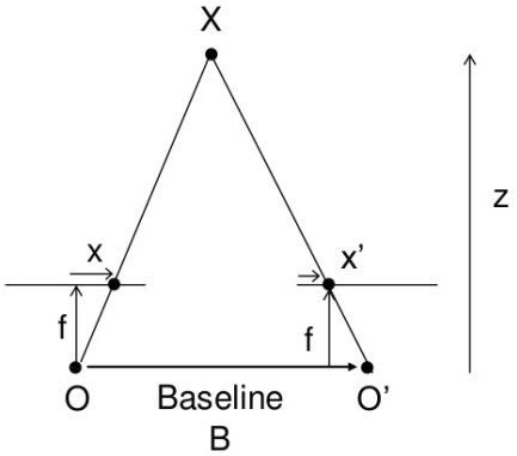


Fig. 7: Geometry for calculating the depth from disparity

$$disparity = x - x' = \frac{Bf}{Z}$$

The disparity map of the whole image looks as given below-

Fig. 8: Disparity map of an image used in the project

Z is the depth of the point X that we are interested in. We can easily calculate the depth from the above equation

$$Z = \frac{Bf}{x - x'}$$

3.1.8. Feature tracking

Optical flow tracking techniques are widely used in computer vision to estimate the motion of desired feature points in the image. It is extensively used in the "Structure from motion" problems such as ours. In this method, we are tracking the motion of detected keypoints/features by virtue of their brightness. The following are assumptions when we solve this problem[10]-

1. The feature point undergoes very small motion between two instances
2. The brightness of the feature point remains the same after the move.

The second constraint can be derived into a mathematical equation. We equate the brightness of the point before and after movement, apply Taylor series expansion and arrive at the following "Brightness constancy constraint equation(BCCE)"-

$$I_x u + I_y v + I_t = 0$$
$$u = velocity\ of\ motion\ in\ x-direction,\ v = velocity\ in\ y\ direction$$
$$I_m = rate\ of\ change\ of\ intensity\ of\ the\ pixel\ with\ respect\ to\ m$$

Here u and v are the two unknowns but we have only one equation. Therefore, the system of equations is underdetermined. Hence, we use the *Lucas-Kanade* tracking scheme which follows the first constraint and in fact, assumes that the motion of neighboring pixels to the pixels under consideration is the same in small time instances. A window is considered centred at the pixel under consideration and the BCCE is written for all the pixels with the same u and v. If suppose a 3x3 window is taken, we have nine equations in two unknowns. The system of equations becomes over determined. So, the LK method uses the least square principle to solve the system of equations as explained below.

The n equations for the n pixels in neighborhood can be written the form as,

$$AV = b$$
$$A = \left[[I_{xn}]\ [I_{yn}]\right],\ V = [u\ v]^T,\ b = [[-I_{tn}]]$$

$[I_{cn}]$ is a nx1 vector for the n points in the window for c=x, y, t. The least square methods works as follows:

$$A^T AV = A^T b$$

$A^T A$ is now a 2x2 matrix and V can easily be solved as,

$$V = (A^T A)^{-1} A^T b$$

Contrary to the assumption, if the feature point undergoes a considerably large motion, the method fails to track the point. The Lucas Kanade method also offers another robust technique which forms a pyramid of the images.[11] This pyramid has the same image with different scales with the base of the pyramid holding the unscaled image and the top of the pyramid holding the small image. If suppose the feature point undergoes motion of 5 pixels in the unscaled image, the same feature will undergo a movement of only one pixel in the topmost image of the pyramid. The topmost layers of successive image pyramids are compared and the feature motion is estimated. This makes the whole tracking process very robust and efficient as we can even track points which have moved long distances.

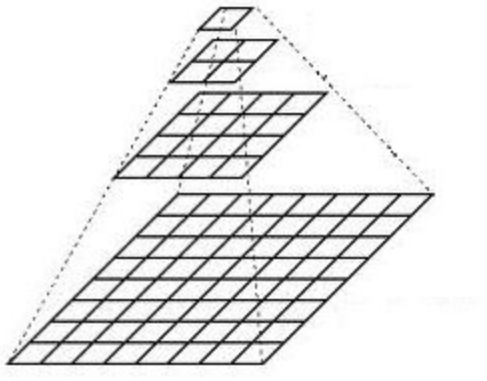


Fig. 9: Pyramid of images obtained from a single image in the KLT tracker
Courtesy:OpenCV(https://docs.opencv.org/2.4/doc/tutorials/imgproc/pyramids/pyramids.html)

In the visual odometry project, we first detect features in each image and then the Lucas -Kanade optical flow tracking algorithm is used to track these features in the next frame so that we can estimate the motion of the features.

3.1.9. Estimating the transformation matrix

For pose estimation, we use the SolvePNPRansac[12] function from OpenCV where PnP stands for Perspective from N Points. It is the problem of defining the 6 degree of freedom pose of the camera given its intrinsic camera parameters, 3D world points and their corresponding projections on the 2D image planes.[13] The problem is also known as the camera resection problem. The solution (transformation matrix) is found by determining the transformation that minimizes the reprojection error :

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg\min_{Tk\Sigma} \|p_{ik} - \hat{p}^{i}_{k-1}\|^2$$

Where,
T_k - resultant transformation matrix
$R_{k,k-1}$ - Rotation matrix between 3D points of frame k and 2D points at frame k-1
$T_{k,k-1}$ - Translation matrix between 3D points of frame k and 2D points at frame k-1
p^{i}_{k} - Set of 2D points of frame k
$\hat{p}^{i}_{k-1}$ - Set of 3D points of frame k-1

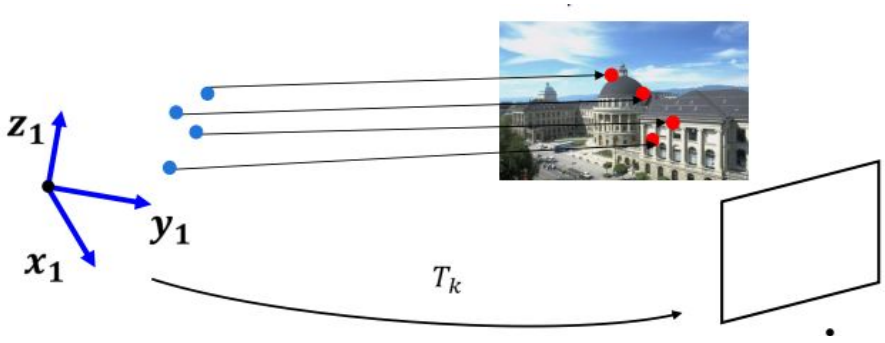


Fig. 10: 3D points on the building being transformed by matrix T_k to the 2D points on the image

It follows from the perspective project model for cameras,

$$sp_c = K\,[R|T]\,p_w$$

s - scale factor for the image point
$p_w = [x\ y\ z\ 1]^T$ homogeneous world point
$p_c = [u\ v\ 1]^T$ corresponding homogeneous image point
K - intrinsic camera matrix (fx and fy are the scaled focal lengths, γ is the skew parameter, (u_0, v_0) - principal point)
R - 3D rotation of the camera
T - 3D translation of the camera

Since this is an underdetermined system of equations, we require at least 4 set of points to obtain a solution
There are some common methods to solve the PNP problem:[13]

1. P3P :
   Here the PNP problem is in its minimal form and we consider just 3 point correspondences. However, this method yields four possible solutions for the R and T matrices. To choose one solution among the four, a

fourth world point and its corresponding image point is chosen and the best among the four solutions are found

2. EPnP
   EPnP stands for Efficient PNP and solves the PnP problem for the general case of n>=3. The solution is based on the assumption that every one of the n reference points can be written as a linear combination of four virtual control points. Thus solving the problem boils down to these four unknowns and the pose is calculated from these four points
3. Using RANSAC
   The PnP method is sensitive to noisy data i.e. outliers in the the 3D and 2D points passed onto it. Thus if incorrect point correspondences are passed, then the resultant rotation and translation vectors would not be accurate. To combat this issue of outliers, we use the RANSAC outlier removal scheme in conjunction with PnP. It works if the number of points passed is greater than 4
   The way it works is as follows:
   1. Iterate over a set number of iterations :
      a. Randomly select 5 points and construct the 'Minimal Sample Sets' (MSS) and estimate the camera pose (rotation and translation) using the EPnP method
      b. Identify inliers among all the 3D points by checking if the point gives a reprojection error below a threshold under the rotation and translation calculated in step 1a
      c. Form the Consensus Set (CS) from all the inlier points found
      d. Repeat from step 1a
   2. The process stops when the number of iterations are complete or when the probability of finding a larger consensus set falls below a set threshold

However this matrix obtained gives us the pose of the world feature points in the camera (car) frame, hence to obtain the pose of the car in the world coordinates, we take the inverse of the transformation matrix. It is done as follows:
Let R_{3*3} represent the 3*3 rotation matrix and t_{3*1} represent the 3*1 translation matrix obtained by SolvePnPRansac method of OpenCV,

$$\begin{bmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1_{1x1} \end{bmatrix}$$

the inverse transformation matrix is calculated as :

$$\begin{bmatrix} R^{-1} & -R^{-1}t \\ 0 & 1 \end{bmatrix}$$

This is done in every iteration, this ensures that the translation and rotation we obtain at every frame is always with respect to the initial frame which is the frame corresponding to the images $I_{l,0}$ and $I_{r,0}$

3.1.10. Creating new features

We find that using only feature matching or only feature tracking does not yield satisfactory results. Hence we employed a combination of feature matching and tracking. For this, for every new frame, we match features in the left and right images using the feature detection and matching techniques mentioned above. We then triangulate the matches and obtain the corresponding 3D points for the matched keypoints. However, the 3D points are not in the initial frame of reference. To express them in the initial frame, we multiply the 3D points with the inverse of the transformation matrix calculated in the previous step.

3.1.11. Stereo Visual Odometry Results

The following section presents the trajectory predicted by our Stereo Visual Odometry code. Due to the nature of the problem, we don't require the ground truth for estimating the scale, rather the stereo pair is used for the same. The

images below show the final result of the ground truth trajectory (green) and our estimated trajectory (red). It also shows the estimated x,y,z coordinates the car finally reached in the initial world frame.

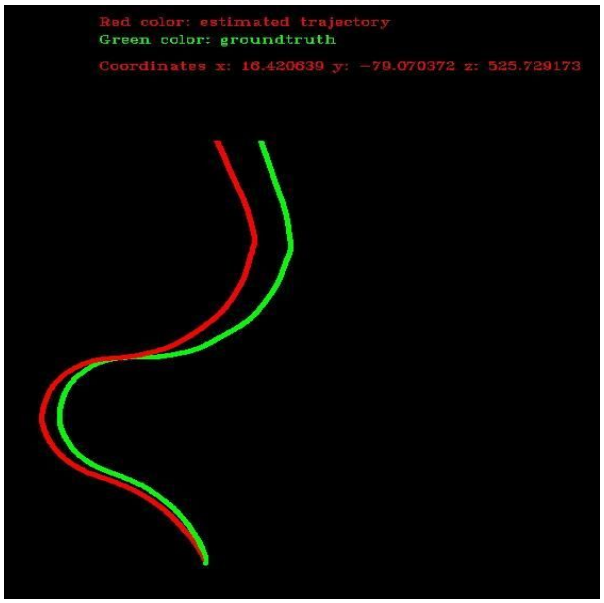


Fig. 11: Estimated and ground truth trajectories on a curved path(sequence 09)

As we can see, the error between the estimated and ground truth trajectory is initially low and increases considerably at the second turn. This was because of a car moving at a high speed in the opposite direction.

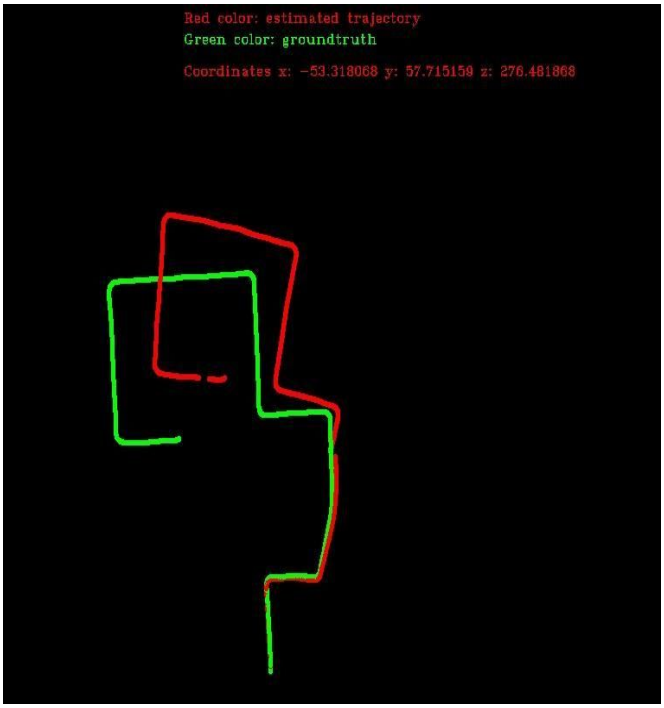


Fig. 12: Estimated and ground truth trajectories for paths with sharp turns(sequence 00)

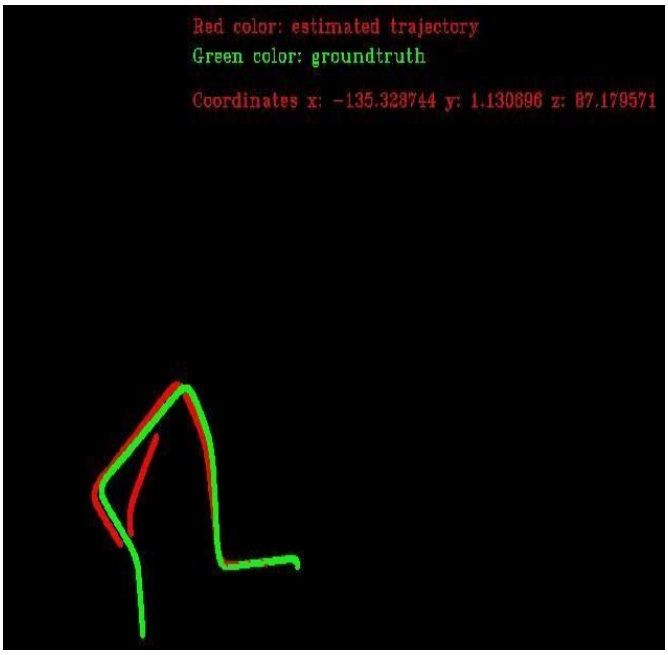
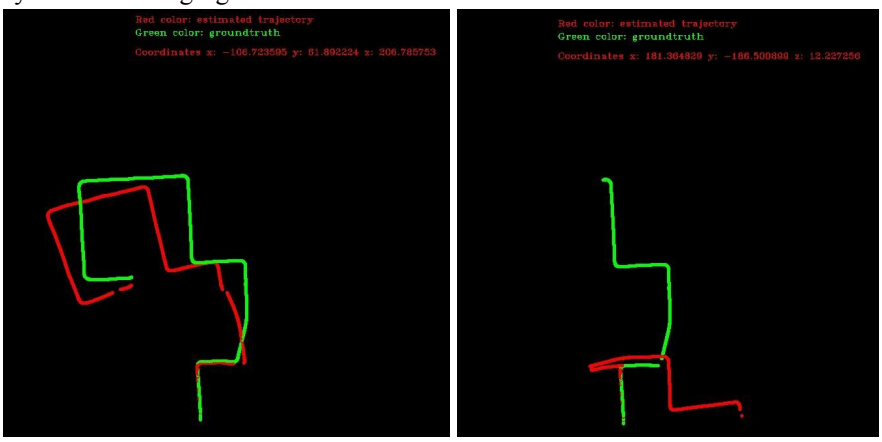


Fig. 13: Estimated and ground truth trajectories for a path with a large truck crossing the road (sequence 07)

In the above figure, the point where the trajectory drifts was when there was a large truck occupying the entire frame of the cameras thereby misleading the system with the features and their directions.

The error metric followed for estimating the accuracy of these trajectories is the position disparities between the estimated and ground truth trajectories measured in each frame. The mean error is then found in the end. The mean error in the sequence 00 is 14.8217metres over 700 consecutive frames. The mean position error in sequences 07 is 68.2997 over 1230 frames and mean position error in sequences 09 is 42.77 metres over 1200 consecutive frames.

We also observed that changing the window size of the KLT Optical Flow Tracker significantly affects the estimated trajectory. The following figure shows the same.



Sequence 00 window size (7,7) Sequence 00 window size (21,21)

Fig. 14: Effect of KLT window size on estimated trajectory

The video link for implemented system can be found at: https://www.youtube.com/watch?v=B-6oqZwLLEs

3.2. Monocular Visual Odometry

3.2.1. Problem Formulation

Input

Our input consists of a stream of gray scale or color images obtained from a pair of cameras. For the monocular odometry problem we only use one camera out of the stereo pair. We use the left camera for our problem. This data is obtained from the KITTI Vision Benchmark Suite Odometry dataset. The dataset originally contains stereo image sequences and we have only used the left camera image sequence for our purpose. Let the frames captured at time t and $t+1$ be I^t, I^{t+1} respectively. The intrinsic and extrinsic parameters of the cameras are obtained via any of the available stereo camera calibration algorithms.

Output

For every stereo image pair (for the stereo case) we receive after every time step we need to find the rotation matrix R and translation vector t, which together describes the motion of the vehicle between two consecutive frames. The vector t can only be calculated upto a scale factor in our monocular scheme.

3.2.2. Algorithm Outline[14]

1. Capture camera frame I^t.
2. Undistort the captured image.
3. Detect features using FAST algorithm in frame I^t and track the same features in frame I^{t+1}. If the number of detected features drops below a certain threshold a new detection is triggered.
4. Feature tracking is done Lucas-Kanade Optical Flow Method with Pyramids to account large motions as well. When we go up in a pyramid, small motions are removed and large motions become small motions. Therefore, applying Lucas Kanade there we get optical flow along with scale.
5. Use Nister's 5-point algorithm with RANSAC to compute the Essential matrix.
6. Decomposition of Essential matrix to get *Rotation* and *translation.*
7. Use R and t to estimate motion between between the two frames.
8. Obtain scale information from some external source and concatenate the translation vectors and rotation matrices.

3.2.3. Reading images

We used the imread function from OpenCV to read the images from the KITTI dataset and converted the images to grayscale since many of the methods used ahead in the approach work only with grayscale images

3.2.4. Feature Detection using FAST detector

In our approach we use the FAST (Features from Accelerated Segment Test)[16] algorithm to detect features from the image scene. We chose the FAST corner detector because other detectors are not fast enough for real-time applications. One good example is a SLAM (Simultaneous Localization and Mapping) mobile robot which has limited computational resources.

As shown in Fig. 15 consider that there is a test pixel **P** which we want to test if it a corner or not. Consider the intensity of the pixel to be $\mathbf{I_p}$ We draw circle of 16 pixels circumference around point **P** as shown in figure (number). For every pixel which lies on the circumference of this circle, we check if there exists a continuous set of η pixels which are brighter than **P** by a certain threshold **T** i.e. ($\mathbf{I_p}$+**T**) or which are darker than pixel **P** i.e. ($\mathbf{I_p}$-**T**). If such a set exists we mark this pixel as a corner pixel. A high-speed test is used to exclude a large number of non-corners. This test examines only the four pixels at 1, 9, 5 and 13 (First 1 and 9 are tested if they are too brighter or darker. If so, then checks 5 and 13). If **P** is to be considered a corner, then at least three of these must all be brighter than

($\mathbf{I_p}$+**T**) or darker than ($\mathbf{I_p}$-**T**)**.** If these two conditions do not satisfy then **P** cannot be a corner. After this a full segment test is applied to the passed candidates by examining all pixels in the circle.

Performing this test on every pixel would be computationally expensive and time consuming. Therefore, to speeden up the process a high-speed test is performed. Pixel 1,9,6,13 are tested against the above conditions. If any of the three pixels satisfy the condition pixel P will pass the corner candidacy test. Now a full segment test will be performed on the passed corner pixels.

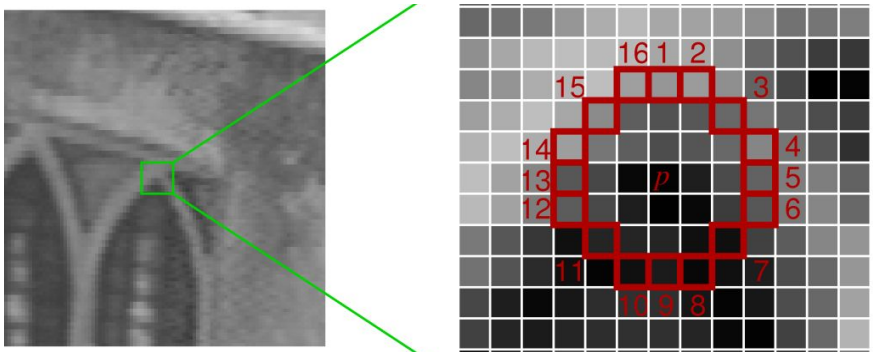


Fig. 15: FAST corner detection heuristic. (Image taken from original FAST paper[16])

Another issue is the the detection of many interest points in nearby locations around a pixel. It is solved by using Non-maximum Suppression. A score function is defined. Adjacent interest points with lower score are discarded and we end up lesser number of feature points which avoids cluttering of features around one location. Fig. 16 shows keypoints without NMS and Fig. 17 shows keypoints with NMS applied. There is a considerable amount of reduction in keypoints from 11263 to 3012.

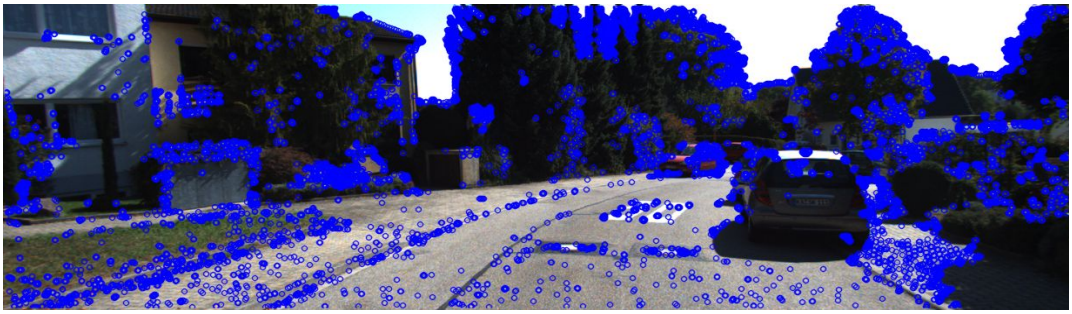


Fig. 16: Keypoints Detected without Non-Maxima Suppression (11263 points)

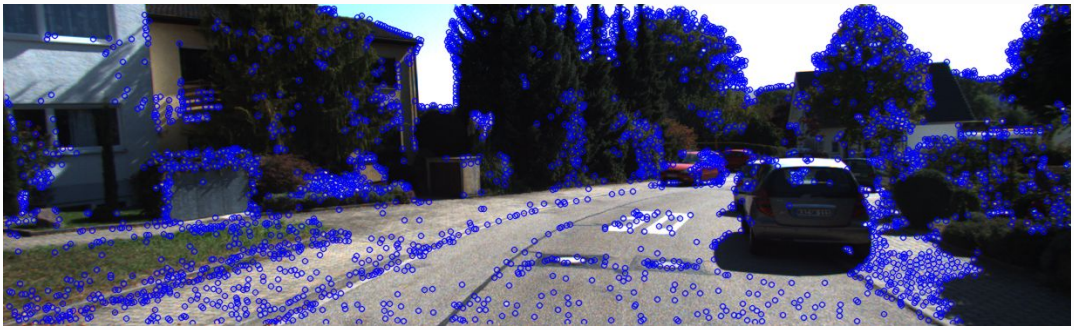


Fig. 17: Keypoints detected using Non Maxima Suppression (3012 points)

3.2.5. Feature Tracking

The fast corners found in the last step are used as input for feature tracking. The Kanade-Lucas-Tomasi feature tracker[17] uses these corners as input and looks for pixel wise correspondences i.e uses to previously tracked corners to look for corners in the next image. A point is considered in the nearby space and image gradients are calculated and used to find the most precise motion for that point. The tracking procedure comes to a halt if the available feature points drop to less than 2000 and at that point a new search is started.

Fig. 18: Illustration of Sparse Optical Flow (Image taken from OpenCV docs)

3.2.6. Estimating the Essential Matrix

After getting the point correspondences, there are many techniques that can be applied to estimate the essential matrix. It can be defined as follows:

$$(y')^T Ey = 0$$

In this equation, y' and y represent the homogenous normalised image coordinates and E represents the essential matrix. There are other simpler algorithms like [Higgins] which need eight point correspondences however, recent research has shown that there is improvement in the results using the Nister five point algorithm. This algorithm essentially develops solutions for a number of equations that are non-linear by nature while needing the least number of points possible.

We used the OpenCV function findEssentialMat[18] for computing the essential matrix. It uses the aforementioned five point algorithm.

$$[p_2; 1]^T K^T E K [p_1; 1] = 0$$

$$K = \begin{bmatrix} f & 0 & x_{pp} \\ 0 & f & y_{pp} \\ 0 & 0 & 1 \end{bmatrix}$$

Here, K is the camera matrix, p1 and p2 are the corresponding points in the first and the second image respectively.

One of the issues with feature tracking algorithms is that they give several erroneous feature correspondences. It is important to remove these erroneous correspondences to ensure accurate motion estimation. RANSAC is used for finding such correspondences. It is a simple but effective iterative algorithm which randomly select five points from a set of feature correspondences and estimates the essential matrix. Then, a check is performed to see if the

remaining points are inliers when this essential matrix is used. This process continues for a number of iterations which can be defined and finally, the Essential matrix having the maximum number of inliers is chosen.

3.2.7. Calculating Pose from the Essential Matrix

For calculating the rotation and the translation matrix from the essential matrix, we use the following relation :

$$E = R[t]_X$$

Here, R= Rotation Matrix, t_x = cross product of t represented as a matrix

Simply calculating the Singular Value Decomposition (SVD) of the essential matrix and performing the following computations, we get the Rotation and the translation matrix:

$$E = U\sum V^T$$

$$R = UW^{-1}V^T$$

$$[t]_x = UW\sum U^T$$

3.2.8. Computing the Trajectory

The camera pose can be defined as R_{cam} t_{cam}. Now for estimating the R_{cam} and t_{cam} we use the previously calculated R and t matrices and iteratively keep updating the new rotation and translation values using the following equations :

$$R_{cam} = RR_{cam}$$

$$t_{cam} = t_{cam} + [t]_x t_{cam}$$

For calculating the translation vector, information about the scale has to be obtained from some external sources. In our case , we can use the ground truth data available to calculate the scale throughout the trajectory. The scale also keeps on changing and is calculated using the distance formula as the distance between the predicted and the ground truth points.

3.2.9. Monocular Visual Odometry Results

In monocular visual odometry we don't have depth information directly available to us. We calculate the absolute scale using external resources (e.g. speedometer). Without the absolute scale we can just say that we have moved 'a' units in the z direction and 'b' units in the x direction. We do not know the exact units i.e millimeters, centimeters, meters, etc. Hence we have to infer how much we moved based on the velocity readings we get from the speedometer and the time between frames. In our case since we did not have information available for the KITTI dataset, we used the the distance between the estimated trajectory point and the ground truth points as a measure of how much we have moved. This is one of the reasons why the monocular visual odometry results are a little better than the stereo. In reality we won't have the ground truth data and our estimated trajectory could be off quite a bit because our speedometer readings would not be exact.

Fig. 19: Estimated v/s ground truth trajectories of video sequence 00 and 08

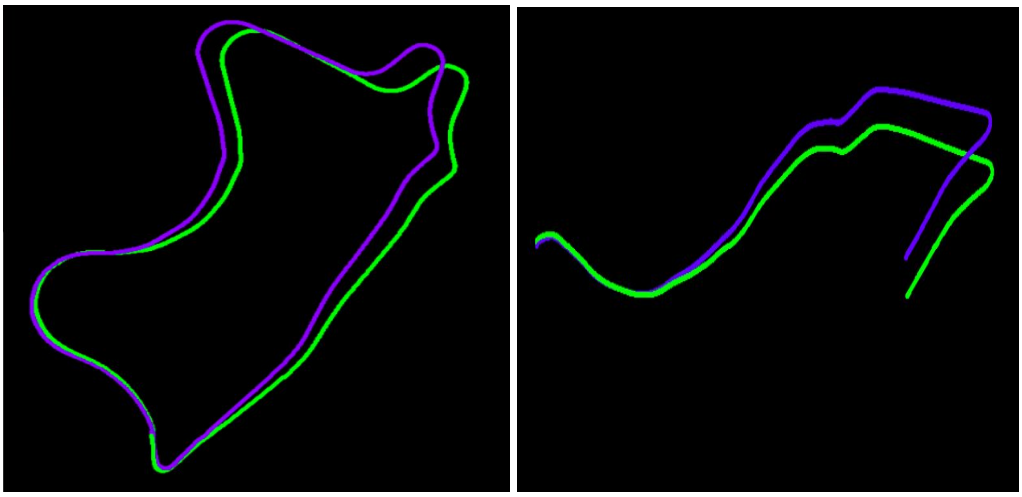
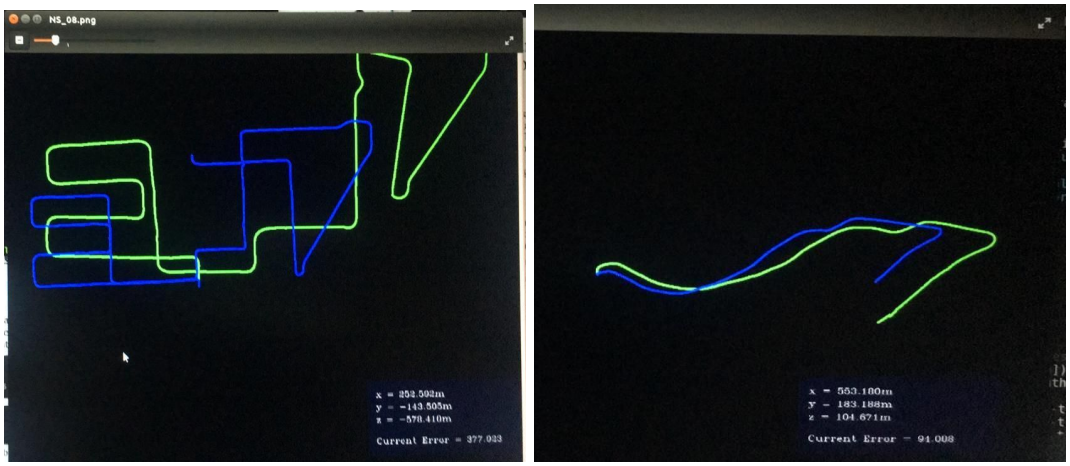


Fig. 20: Estimated v/s ground truth trajectories of video sequence 09 and 10

The error metric used here is the distance between the estimated position and the ground truth position in a particular frame i.e. how off our estimates are from the ground truth. We provide two types of errors, i.e. the maximum error and the mean error for every sequence. The results above show visual odometry estimates the trajectory correctly in the initial 500-1500 frames with reasonably good error bounds, after which it drifts away from the ground truth. The maximum error for sequences 00, 08, 09, 10 are 28.51 meters, 76.12 meters, 37.56 meters and 42.59 meters respectively. While the mean error for sequence 00 is 9.08 meters over 4500 frames, for sequence 08 its 19.86 meters over 4000 frames, for sequence 09 its 18.85 meters over 1600 frames while for sequence 10 it is 22 meters over 1200 frames. Since, visual odometry is a dead reckoning method there is bound to be such errors in our estimation. It was observed that the error usually increases after a sharp turn, taking a right and left turn, when

another vehicle passes across the scene when the vehicle is stationary and when there is rapid change in speed.

Also, in case we don't have a reliable source for the absolute scale, we assumed a value of 1 and plotted the trajectories and obtained errors of 223 meters for sequence 08 of 4000 frames and error of 76 meters for sequence 10 of 1200 frames



Sequence 08 Sequence 10

Fig. 21: Estimated v/s ground truth trajectories of video sequence 08 and 10 without absolute scale provided

The video link for implemented system can be found at:
https://www.youtube.com/watch?v=g9-e1hjRvks&feature=youtu.be
Our visual odometry problem also assumes that our environment is static and the dominant motion is in the forward direction. Hence, when we are at a red signal and we have a vehicle moving across the scene, would lead our algorithm to believe that our vehicle is moving sideways which is not possible.

4. Conclusion and Future Work

We implemented visual odometry using the stereo and monocular cases. We were able to reduce the overall estimation error to within 30 meters over an image sequence of 1000 images for the stereo case when errors were averaged over all sequences considered. Due to the nature of the stereo problem, we did not require the ground truth and were able to obtain the scale from the images itself. While for the monocular case, if there is a reliable source for absolute scale, the error reduces to within 10 meters over an image sequence of 1000 images when averaged over all sequences considered. If there is no absolute scale information provided, the error is about 100 meters over 1000 images due to unsurety in scale calculations. We also found that implementing only feature matching fails when there is fast motion of the car while only feature tracking fails since the number of features tracked keep gradually decreasing and at sharp turns, we completely lose most of the tracked features. Hence a combination of feature matching and tracking is required for a robust estimation. In the future, we plan to implement windowed bundle adjustment as an optimization technique for loop closure. Also we plan to dynamically optimize the window size of the KLT tracker if we are aware of the map of the terrain beforehand, since it results in much better performance as seen in stereo results. We would also implement grid based feature detection to ensure even coverage of feature points.

5. References

1. http://www.cvlibs.net/datasets/kitti/
2. https://en.wikipedia.org/wiki/Odometry
3. https://en.wikipedia.org/wiki/Visual_odometry
4. http://rpg.ifi.uzh.ch/visual_odometry_tutorial.html
5. https://docs.opencv.org/3.0-beta/modules/imgcodecs/doc/reading_and_writing_images.html
6. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html
7. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html
8. https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html
9. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
10. https://en.wikipedia.org/wiki/Kanade%E2%80%93Lucas%E2%80%93Tomasi_feature_tracker
11. https://docs.opencv.org/2.4/doc/tutorials/imgproc/pyramids/pyramids.html
12. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
13. https://en.wikipedia.org/wiki/Perspective-n-Point
14. https://www.ifi.uzh.ch/dam/jcr:5759a719-55db-4930-8051-4cc534f812b1/VO_Part_I_Scaramuzza.pdf
15. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html
16. Tom Drummond Edward Rosten. Machine learning for high-speed corner detection. In European Conference on Computer Vision, 2006.
17. Takeo Kanade Carlo Tomasi. Detection and tracking of point features. In CMU Tech Report, 1991.
18. https://docs.opencv.org/3.0-beta/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html