Stereo Visual Odometry

Ruprecht Karl University of Heidelberg
Computer Vision: 3D WiSe 2022/23

Author Project group H:
Liu, Yaxi (Data and Computer Science, MtkNr. 3769313)
Xu, Shengzhe (Data and Computer Science, MtkNr. 3769534)
Schrepfer, Jule Katharina (Physics Master, MtkNr. 3736320)
Patil, Sanmati (Scientific Computing, MtkNr. 3444150)

Abstract

This report provides a brief overview of an algorithm for stereo odometry, which has been adapted from the paper [1]. The algorithm, implemented on PYTHON, relies on careful selection and matching of features over a pair of stereo images, followed by egomotion estimation using the features chosen. Evaluating the code over KITTI dataset sequences yield reasonably accurate odometry estimates compared to the original algorithm in [1].

1. Introduction

Accurate localization of a vehicle is a fundamental challenge and one of the most important tasks of mobile robots. For autonomous navigation, motion tracking, and obstacle detection and avoidance, a robot must maintain knowledge of its position over time. Vision based odometry is a robust technique utilized for this purpose. It allows a vehicle to localize itself robustly by using only a stream of images captured by a camera attached to the vehicle. In robotics and computervision, Visual Odometry is the process of determining the position and orientation of a robot by analyzing the associated camera images. The idea was first introduced for planetary rovers operating on Mars – Moravec in the early 1980s. Visual Odometry is a technique used to localize a robot by using only a stream of images acquired from a single or multiple cameras attached to the robot. The images contain a sufficient amount of meaningful information (color, texture, shape, etc.) to estimate the movement of a camera in a static environment. Stereo visual odometry estimates the camera's egomotion using a pair of calibrated cameras. Stereo camera systems are inherently more stable than monocular ones because the stereo pair provides good triangulation of image features and resolves the scale ambiguity.

Related Work

In recent years many algorithms for visual odometry have been developed, which can roughly be devised into two categories, namely methods using monoscopic cameras or methods using stereo rigs. These approaches can be further separated into methods which either use feature matching between consecutive images or feature tracking over a sequence of images. If a calibrated multi-ocular camera setup is available, the 3-dimensional scene can be reconstructed via triangulation. Based on the point clouds of the static scene in two consecutive images, the iterated closest point (ICP) algorithm is often used for egomotion estimation as described in [7]. Monocular cameras mainly require tracking image features (e.g. corners) over a certain number of images. Using these feature tracks, also the scene structure can be computed using structure from motion [8]. In most cases, the multiocular algorithms yield better performances than monocular approaches [5]. Additionaly, if multi-camera approaches are used, the scale ambiguity present in the monocular case is eliminated [4]. Further approaches combine visual odometry with other sensors to increase the accuracy of the results and reduce drift, a problem inherent to all incremental positioning methods.

2. Overview

2.1 Dataset used

KITTI Vision Benchmark Suite : The KITTI Odometry dataset was used in our project. The dataset contains 21 sequences of stereo video sequences in 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. Poses files: The folder 'poses.txt' contains the ground truth poses (trajectory) for the first 11 sequences.

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

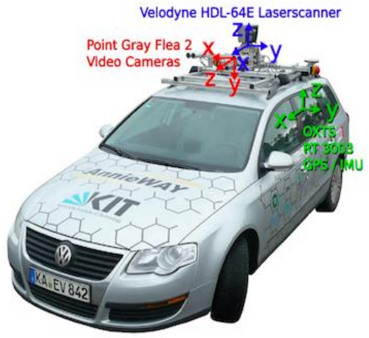

Fig. 1: KITTI dataset recording platform: VW Passat station wagon is equipped with four video cameras (two color and two grayscale cameras), a rotating 3D laser scanner and a combined GPS/IMU inertial navigation system. (Image taken from the KITTI dataset paper)

2.2. Our Approach

The basic algorithm is as follows: for a given pair of frames, (1) detect features in the left image of the first frame (SIFT Algorithm), (2) track the features to the second frame, (3) calculate disparity and depthmap for the image, especially for the feature points using the right image (4) project the points to 3D using the depths (5) find the largest set of self of consistent matches (inliers), and (6) find the frame-to-frame motion that minimizes the re-projection error for features in the inlier set. The inlier detection step (5) is the key distinguishing feature of the algorithm. The feature matching stage inevitably produces some incorrect correspondences, which, if left intact, will unfavorably bias the frame-to-frame motion estimate. A common solution to this problem is to use a robust estimator that can tolerate some number of false matches (e.g., RANSAC) or select a consistent subset of data with inlier detection.

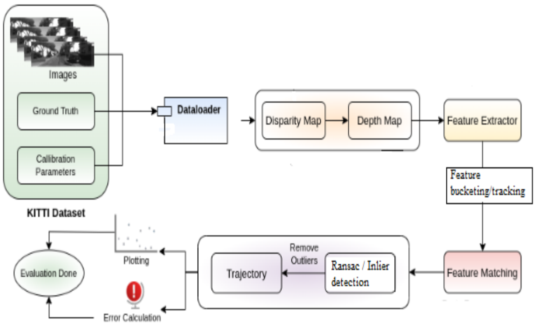

Fig. 2 Pipeline of Stereo Visual Odometry

3. Detailed Methodology

Problem Formulation

Input Our input consists of a stream of gray scale images obtained from a pair of cameras. In KITTI dataset the input images are already corrected for lens distortion and stereo rectified. Let the pair of images captured at time t and t+1 be ($I_{l,t}$, $I_{r,t}$) and ($I_{l,t+1}$, $I_{r,t+1}$) 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 Disparity and Depth maps

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. The dataset provides left and right images of each instance.

3.1.1 Disparity maps

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.

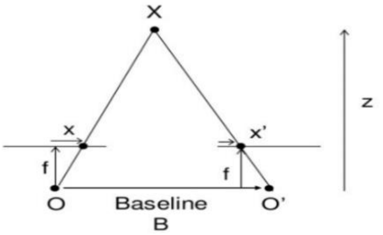

Fig. 3: Geometry for calculating the depth from disparity

To construct disparity map, we have used the StereoSGBM function from OpenCV which is an implementation of Hirschumuller Algorithm. A disparity map is depicted below:

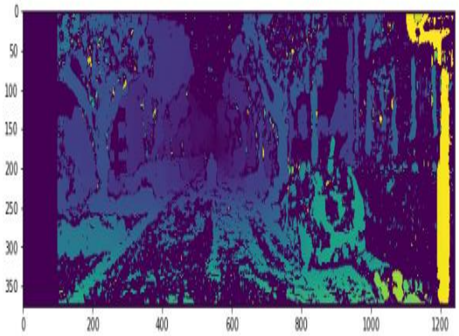

Fig. 4: Stereo disparity map of first sequence image

The rectangle at the left side is caused by the right camera. This is the space where the right camera does not see anything due to the baseline. Think that to calculate depth, we need an overlapping between the left camera and the right camera. However, there is some space in the right image that is not overlapping with the left image. This rectangle is useless.

Using the disparity map, we get the depth of the scene using the formula:

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

x and x′ are the distance between points in image plane corresponding to the scene point 3D and their camera center. B is the distance between two cameras and f is the focal length of camera.

3.1.2. Estimate camera intrinsic and baseline

the intrinsic parameters of a camera depend on how it captures the images. Parameters such as focal length, aperture, field-of-view, resolution, etc govern the intrinsic matrix of a camera model. The intrinsic matrix transforms 3D camera coordinates to 2D homogeneous image coordinates. The focal length *fx ,fy* is the distance between the pinhole and the film. In a true pinhole camera, both *fx* and *fy* have the same value. The "principle point offset" *Ox,Oy* is the location of the principle point relative to the film's origin. The exact definition depends on which convention is used for the location of the origin. Axis skew s causes shear distortion in the projected image**.**

$$\underset{\text{Projection Matrix}}{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}} = \underset{\text{Intrinsic Matrix}}{\begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}} \underset{\text{Extrinsic Matrix}}{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

Here we load the projection matrices from calib.txt, and use the calibration to calculate the intrinsic matrix and baselines. Based on first two frames from sequence 00, we get insight of the below results.

The Facal Length is : (718.856, 718.856)

The Image Center is : (607.1928,185.2157)
The Baseline is :0.5371657188644179

3.1.3 Depth Maps Depth is inversely proportional to disparity. The more the disparity is, the closer the object is to the baseline of the camera. The less the disparity is, the farther the object is to the baseline.

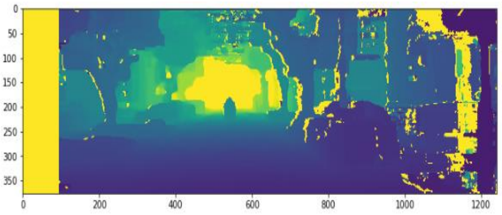

Fig. 5: Depth map of first sequence image

Yellow colour points in figure 5 represents objects that are really far away from the camera.

3.2 SIFT algorithm for keypoint detection

The keypoint detection function is implemented using SIFT algorithm provided by OpenCV. The SIFT(Scala-Invariant Feature Transformation) algotrithm is a technique introduced by David Lowe in 1999 and is widely used for extracting features of images in the field of computer vision. The SIFT algorithm consists four major steps: Scale-space extrema detedtion; Keypoint localization; Orientation assignment; and Keypoint descripter. The code for the Keypoint detection runs in the beginning and always when the number of tracked points drops below 100.

3.2.1 Scale-space extrema detection

The fisrt step in SIFT algorithm is convoling image with a Guassian kernel to construct an image pyramid, which ensures that each SIFT feature has corresponding relationship at different scale[1]. The method is Difference od Guassians(DoG).

Given the original image *I(x,y)*, DoG is defined as follows:

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma)$$

$$L(x,y,k\sigma) = G(x,y,k\sigma) \ast I(x,y)$$

where *L(x,y,kσ)* is obtained by convolution the original image *I(x,y)* and the Guassian kernel *G(x,y,kσ)*.

3.2.2 Keypoint localization

Due to the discreteness of scale sapce and pixels, the Duassian difference pyramid is also discrete. Some potential keypoints are therefore unstable. Talyor series expansion is used to determine the position of keypoints. The Taylor expansion is given by:

$$f(X) = f(x_0) + \frac{\partial f^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 f}{\partial X^2} X$$

where *X=(x,y,σ)* and *f* is Difference of Guassians. The derivative is:

$$\frac{\partial f(X)}{\partial X} = \frac{\partial f^T}{\partial X} + \frac{\partial^2 f}{\partial X^2} X$$

Let it be 0 and the position of extreme points can be obtained.

3.2.3 Orientation assigment

The algorithm adopts gradient histogram method. It calculates the gradient of pixels firstly within a cell. After completing the gradient calculation of the keypoints, it uses the histogram to count the gradient and direction of the pixels in the neighborhood. The gradient histogram divides the direction range from 0 to 360 degrees into 36 bins, each of which represents a range of 10 degrees.

3.2.4 Keypoint descripter

In this step, the patch is divided into 4×4 blocks, each of which is 16×16 pixels. Each block has an 8-bin histogram to store the gradient orientations. Histogram increases by weighted magnitude for each gradient.

The keypoint detection function converts the image to gray and then uses it as input of SIFT function. It returns the coordinate of kepoints. One input and output images are shown as follows:

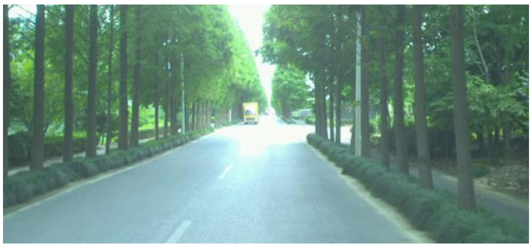

Fig. 6: Original image

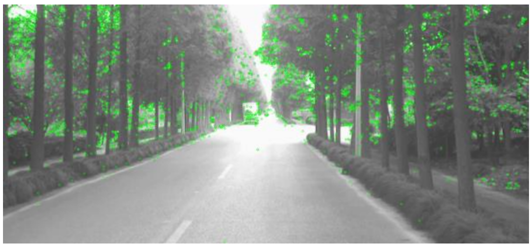

Fig. 7: Output of SIFT

3.3 Feature bucketing

Feature bucketing is a method that ensures feature points are well distributed in image and reduces computation complexity of algorithm. It divides the image into several cells, which are called buckets. For each bucket, it selects 10 points with strongest response and then return the coordinte of keypoint after bucketing. Response indiactes how good the keypoints are. The strongest response corresponds the best keypoint.

The followwing is the ouput of feature bucketing method:

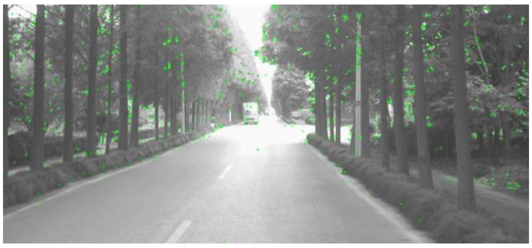

Fig. 8: Ouput of bucketing

Compared with Figure 7, the number of keypoinys in Figure 8 is reduced.

3.4 Feature tracking

Feature tracking is a problem to track keypoints in an image sequence, which can be used for motion analysis, object tracking, etc. One of the important method is called Lucas-Kanade method, which is based on Optical Flow.

3.4.1 Optical Flow

Suppose the pixel *(x,y)* at T moves to *(x+dx,y+dy)* after time dt. According to britness consistancy:

$$I(x,y,t) = I(x+dx, y+dy, t+dt)$$

where *I(x,y,t)* is britness of pixel *(x,y)* at time t. Applying Taylor expansion:

$$I(x+dx, y+dy, t+dt)$$

$$= I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

then we have:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$$

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0$$

and the Optical flow equation is:

$$I_x V_x + I_y V_y + I_t = 0$$

3.4.2. Lucas-Kanade method

The main principle of Lucas-Kanade optical flow estimation is to assume constant brightness to find the velocity vector between two consecutive frames (t and t+1)[2] by the following equation:

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_y(p_n) & I_y(p_n) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -I_t(p_1) \\ \vdots \\ -I_t(p_n) \end{bmatrix}$$

Python library OpenCV provide a function *calcOpticalFlowPyrLK()*, which implements Lucas-Kanade method. The function uses the coordinate of keypoints at time t that returned by SIFT function in the last step to calculate the position of the same keypoints at time t+1. It returns the value of status for each input point. If the corresponding keypoint is found, the value of status will be 1.

After that, the feature tracking function will return the coordianate of corresponding points with status equal to 1. The data obtained will be used to generate 3D point cloud.

3.5 Projecting feature points to camera coordinate system

Using the depthmap that was calculated from the disparity map, the depth of the feature points in the left image is extracted. Using the projection matrix and the depth, the points are projected from the screen of left camera to the 3-dimensional coordinate system of the left camera.

3.6 Inlier detection

For the estimation of the RT matrix, only the points should be considered where no errors in tracking or triangulation occur and that are not moving relative to other points. The distance of the 3D points relative to each other point should not change when moving the camera. This fact is used for a inlier detection with the clique algorithm. This algorithm was also used in the paper. First, the distance of the *n* 3D points in frame 1 to each other is calculated and stored in a *n*x*n* dimensional array, same for the points in frame 2. This two arrays are subtracted. A critical difference (e.g. 0.2m) is chosen. The whose distance to the most other points does not change (difference less than critical difference) is added to the clique first. After that, step by step new points are added to the clique. Potential inliers are all points with a distance difference of less than d_crit with all points in the clique. In each step, the potential inlier that is consistent with most other potential inliers is added to the clique.

3.7 Estimating the transformation matrix

The transformation of the camera position between two frames is given by 6 parameters: 3 rotational and 3 translational. The rotation part is described by a rotation matrix that is a product of 3 rotation matrices around x, y and z axis. The translation is added. The complete transformation is given by a 4x4 dimensional Matrix:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{21} & r_{22} & r_{23} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

The 6 parameters of the matrix are estimated using least squares. The 3-dimensional points in camera position 1 are transformed to the cooredinate system of camera position 2 using the RT Matrix and projected to the screen in position 2 using the projection matrix. The error function is the sum of the squared distances of the points projected to screen in position 2 to the tracked points in the image. This error function is used for least squares. Using the optimized parameters, the RT matrix of the step is calculated. The inverse RT matrix is multiplied from the right side to the matrix of all the steps before to get the total position. The most interesting thing about the matrix is the last column which shows the tracked position of the camera.

3.8 RANSAC outlier detection

Instead of the inlier detection, RANSAC can be used to filter out the outliers in the data. Here, in each iteration 6 random pairs of points (6 3D points in frame 1, 6 2D points in frame 2) are selected. The RT-Matrix is calculated using these 6 points. Using this matrix, the reprojection error of all other points is calculated and the number of point pairs that give an error smaller than a critical value is counted. In the end, the RT-Matrix from the iteration with the most matching point pairs is used.

4. Experiments and Results

The algorithm is applied to the KITTI odometry dataset (sequence 00) and the camera trajectory computed from the images is plotted together with the true trajectory. Especially for the first 500 frames, the camera trajectory computed by visual odometry is near to the true trajectory. Later, the errors in rotation add up and the trajectory is rotated (in the end about 45 degrees after 4000 frames) compared to the true one. This problem ocours becaurse the camera movement is calculated from frame to frame and a rotation error in a single frame can influence all the trajectory afterwards very strong. A video with the results for 4000 including tracked feature points, depthmap and camera trajectory has been produced.

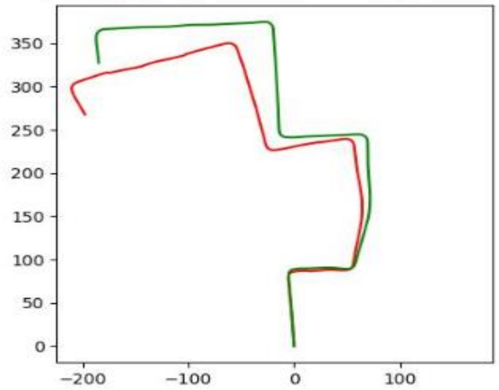

Fig. 9 : Ground Truth(green) vs Eastimated Trajectory(red) for first 4000 frames of sequence 00

Also, experiments with RANSAC instead of inlier detection have been done. This algorithm takes much time, even with only 200 RANSAC-iterations. The resulting trajectory after 300 images is a little bit better than for inlier detection.

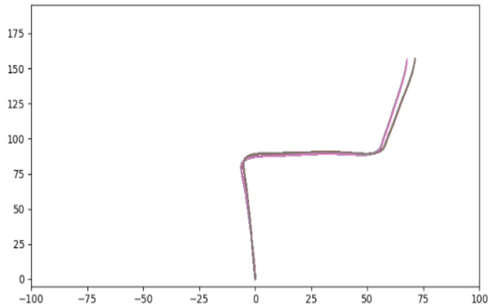

Fig. 10 : Ground Truth(purple) vs Eastimated Trajectory(pink) for first 300 frames of sequence 00 using RANSAC

In general, the running time can be improved much by stopping least squares after a certain amount of iterations. However, this also changes the quality of the results a bit.

How important a inlier detection or RANSAC is, shows figure 11 a experiment without both techniques:

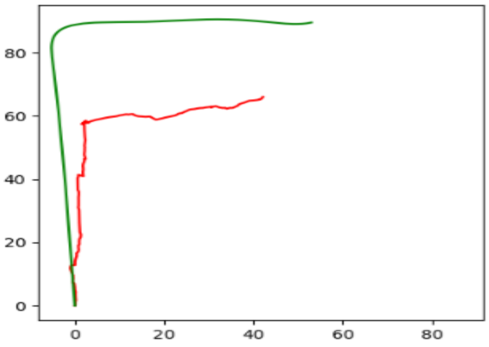

Fig. 11 : Ground Truth(green) vs Eastimated Trajectory(red)

The graphics show a huge difference between estimated position and ground truth even after the first few frames. Explenation for that can be found when looking at the video: Some points are not tracked correctly. In addition, there are moving objects in parts of the video. These points which are moving relative to the others are filtered out by inlier detection and RANSAC.

5 Conclusion and Future Work

We successfully implemented a stereo visual odometry algorithm. The algorithm has been tested on the KITTI odometry dataset. The resulting camera trajectory matches the ground truth well, especially in the beginning. Later, the trajectory is rotated relative to the ground truth. The error resulting from wrong rotation in later stages could be improved be using a loop closure with SLAM. The inlier detection is a key to good results, alternatively and with more computing time, a RANSAC approach can be used.

We used the SIFT feature detector to detect featurepoints in our images. We could do this using ORB or FAST also. ORB stands for Oriented Fast and Rotated Brief. The ORB is fast, efficient in detecting features and solves many issues with other detectors like SIFT,SURF, etc.

References :

[1] Stereo Visual Odometry: Chirayu Garg, Utkarsh Jain University of Wisconsin-Madiso

[2]Abdullah-Al N, Kong Y . Histopathological Breast-Image Classification Using Local and Frequency Domains by Convolutional Neural Network[J]. Information, 2018, 9(1):19-19.

[3] O. Haggui, C. Tadonki, F. Sayadi and B.Ouni, "Evaluation of an OPENMP Parallelization of Lucas-Kanade on a NUMA-Manycore," *2018 30th International Symposium on Computer Architecture and* High Performance Computing (SBAC-PAD), Lyon, France, 2018, pp. 436-441, doi: 10.1109/CAHPC.2018.8645936.

[4] M. Agrawal and K. Konolige, "Real-time localization in outdoor environments using stereo vision and inexpensive gps," in Proceedings of the 18th International Conference on Pattern Recognition, 2006, pp. 1063 – 10 068.

[5] H. Badino, "A robust approach for ego-motion estimation using a mobile stereo platform," in First International Workshop on Complex Motion, 2004, pp. 198 – 208.

[6] NafBZ/Stereo-Visual-Odometry: Visual Odometry Pipeline using KITTI dataset (github.com)

[7] A. Milella and R. Siegwart, "Stereo-based ego-motion estimation using pixel-tracking and iterative closest point," in Proceedings of the Fourth IEEE International Conference on Computer Vision Systems, 2006.

[8] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in ´ IEEE Computer Society Conference Computer Vision and Pattern Recognition, vol. 1, 2004, pp. 652 – 659.

[9] Implementation of Stereo Odometry Using Careful Feature Selection and Tracking Mayank Mittal (14376), Ritwik Bera (14561)

[10] I. Cvišić and I. Petrović. Stereo odometry based on careful feature selection and tracking. In Mobile Robots (ECMR), 2015 European Conference on, pages 1–6. IEEE, 2015.

[11] Improved Pose Estimation by Inlier Refinement for Visual Odometry 2017 IEEE 3rd International Conference on Sensing, Signal Processing and Security (ICSSS)

[12] The KITTI Vision Benchmark Suite (cvlibs.net) for dataset

Team Contribution :

First we all would like to say thanks to Prof. Carsten Rotherand his team for giving us approtunity to work on this project. As a team we enjoyed working together on this Project and able to finish successfully. Below gives in brief all team members contribution.

Very beginning, the data image processing and intrinsic camera parameters are done by Shengzhe Xu. We couldn't include all results as we learned later images are stereo rectified. The part Disparity Map is done by Sanmati Patil. Depth Map is done together by Shengzhe Xu and Sanmati Patil. The complete Feature detection, buckting and tracking is done by Yaxi Liu. Eastimating RT matrix, point projection and Inlier detection is done by Jule Schrepfer. Ransac outlier detection is done together by Yaxi Liu and Jule Schrepfer. All programming code putting together and run by Jule. Everyone prepared written report based on the parts of algorithm they worked. Written report abstract introduction and putting all reports into one is done by Sanmati Patil.Written Report last part regarding conclusion and results is done by Jule Schrepfer. Read me file is maintained by Jule Schrefer.

Thank you!!