ENPM 673: Perception for Autonomous Robots
PROJECT 3 - REPORT

Rishabh Mukund (UID: 117556124)

1. Introduction

In this project, we are going to implement the concept of Stereo Vision. We will be given 3 different datasets, each of them contains 2 images of the same scenario but taken from two different camera angles. By comparing the information about a scene from 2 vantage points, we can obtain the 3D information by examining the relative positions of objects.

Fig: Left and Right input images respectively.

2. Calibration

To find the relative rotation and translation between the two images, we have to first extract the matching features from both the images, This can be done using SIFT or Corner detection. In this project, we used ORB features to get the matching features and the descriptors between the two images. After finding the matching features, we sort them by distance and extract the first 30 prominent features.

To get the fundamental matrix, we find the best fitting line across the prominent features using RANSAC and taking the fundamental matrix that fits best and has the least error. Error is calculated as the dot product of transpose of key feature points of any image with the dot product of F matrix and key feature points of the same image.

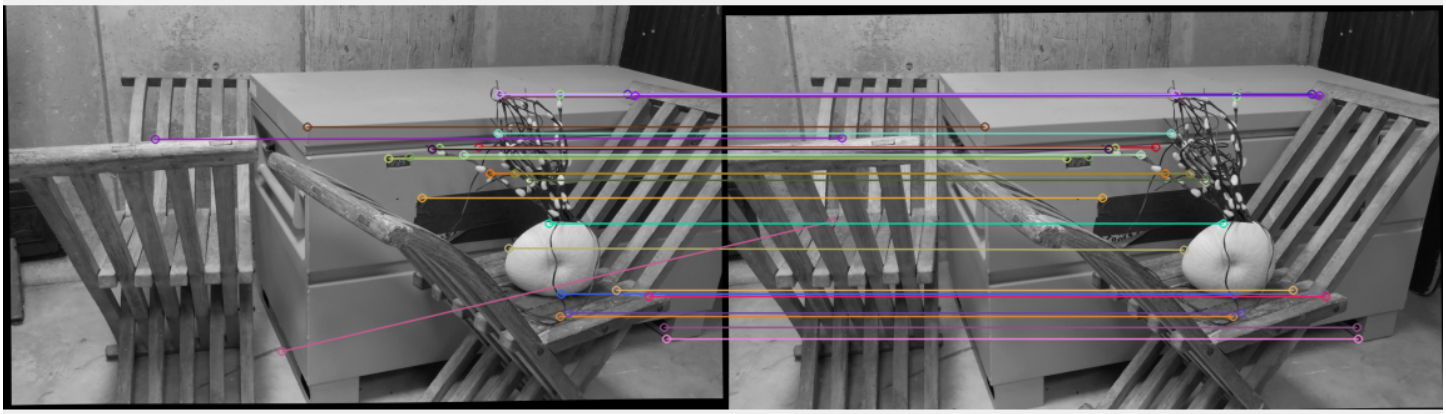


Fig: 30 prominent key features of left and right images.

We can generate the Estimation matrix with the best fit Fundamental matrix and the camera intrinsic parameters of both the cameras.

To estimate the pose of the camera, we perform SVD on the E matrix and W matrix which is [0, -1, 0; 1, 0 , 0; 0, 0, 1]. We compute the translation matrices with the first 3 left singular vectors after performing SVD and we can compute the rotation matrices with the U, V and W matrix defined above.

We will get 4 pairs of rotation and translation matrices, and we should choose the best camera pose from these by using the key features of either image and calculating the distance between key feature and translation matrix. The maximum length will define the best camera pose for the two input images.

3. Rectification

In this step, our goal is to project both the images in the same plane. We use the python inbuilt function "*stereoRectifyUncalibrated " on* the key features obtained from the first image, second image, the F matrix and image size. This function will generate the two homography matrices which will be used to warp the first and second image respectively to get the images projected onto the same plane.

Using H1 and H2 obtained we will also rectify the key feature points. Using the inbuilt python function "*computeCorrespondEpilines*" to find the epipolar lines between the two rectified images and plot them to make sure that the bipolar lines are parallel

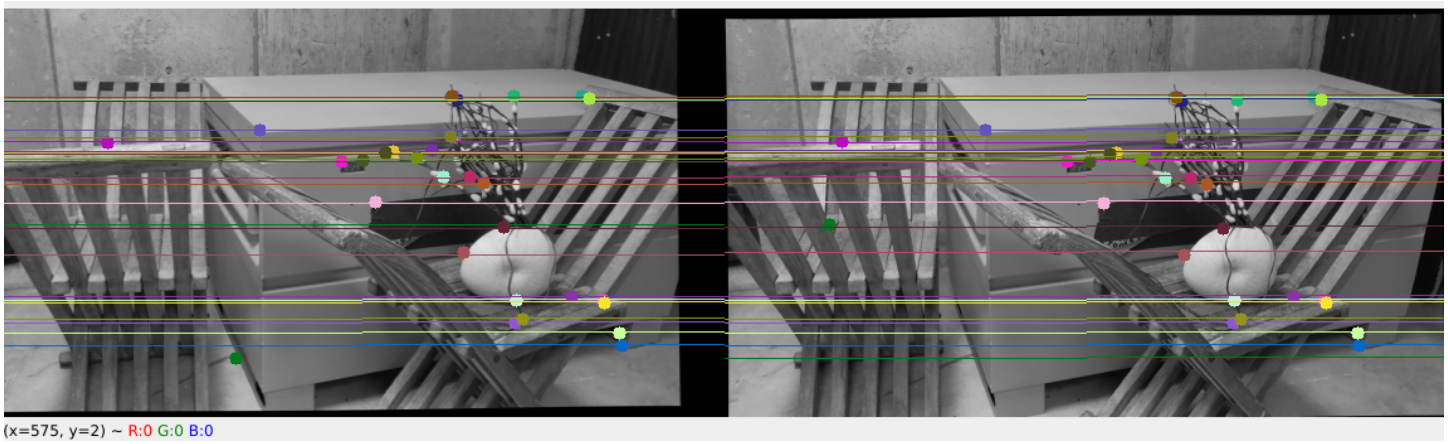
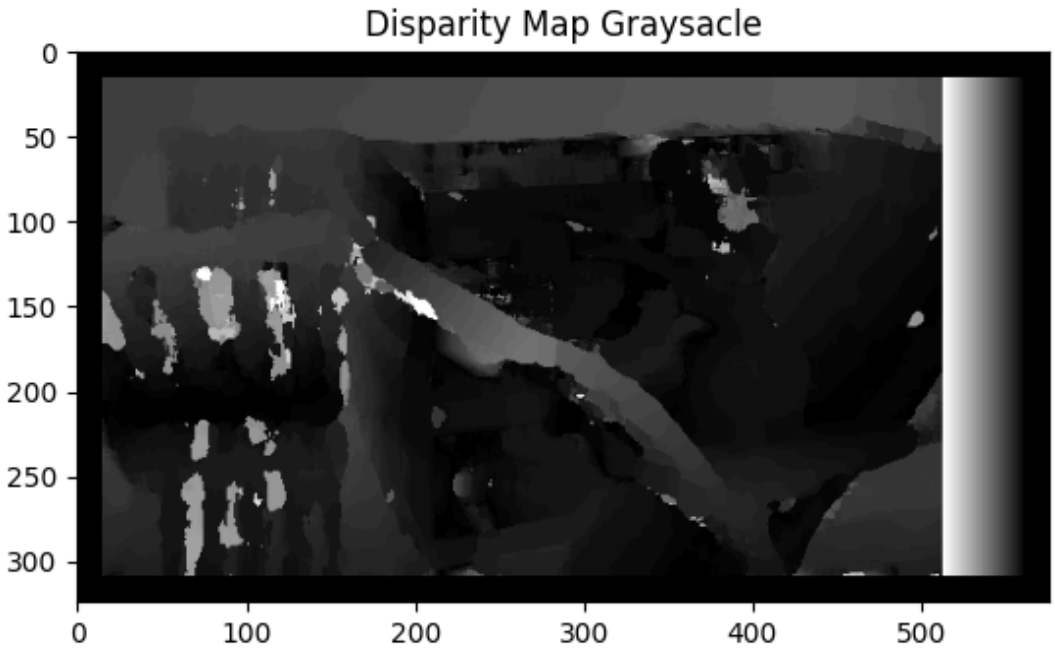


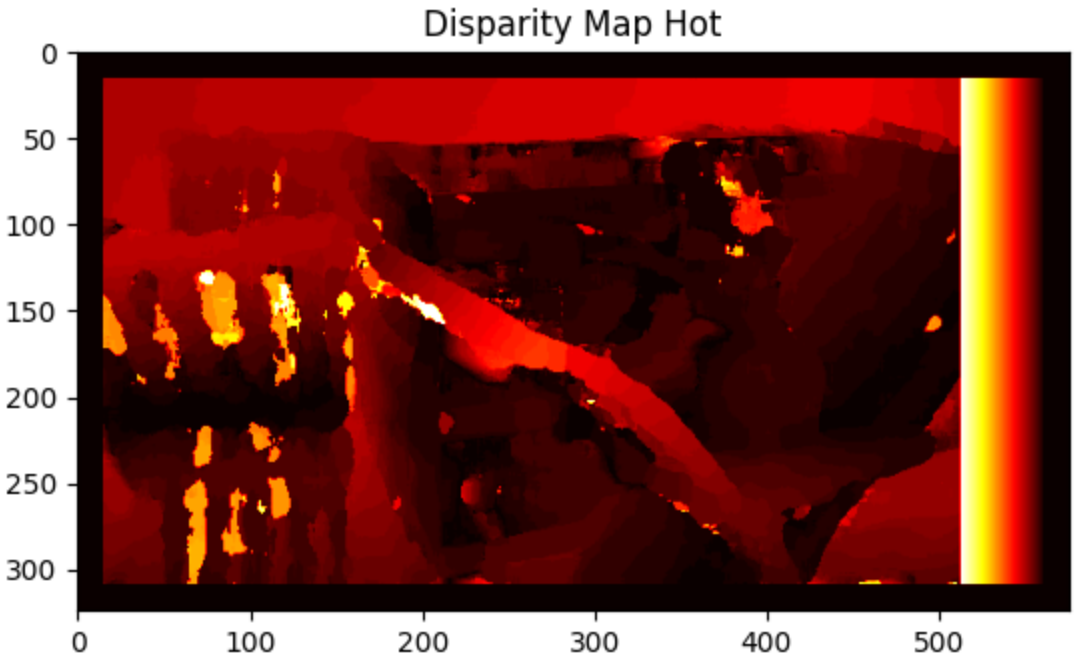
Fig: Ratified left and right images with epipolar lines and rectified key features

4. Correspondence

For each epipolar line, we apply a matching window to compute the correlation between the two rectified images. To calculate the correlation we use Sum of Squared distances. SSD uses matching window technique on the rectified images and rectified key feature points.

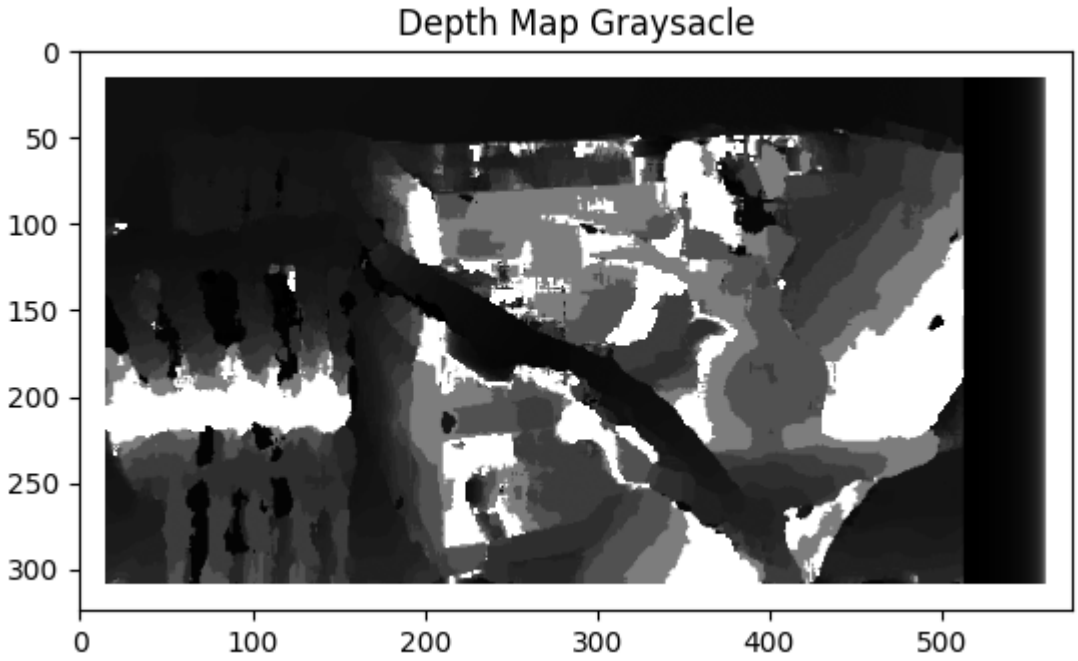
The results obtained from SSD are the disparity map of the two rectified images, we rescale the disparity obtained to be from 0-255. Using matplaotlib's *"imshow"* function, we can plot the heat map by setting the type to *"Hot"* and the frauscale by setting the type to *"gray"* for the disparity map obtained.

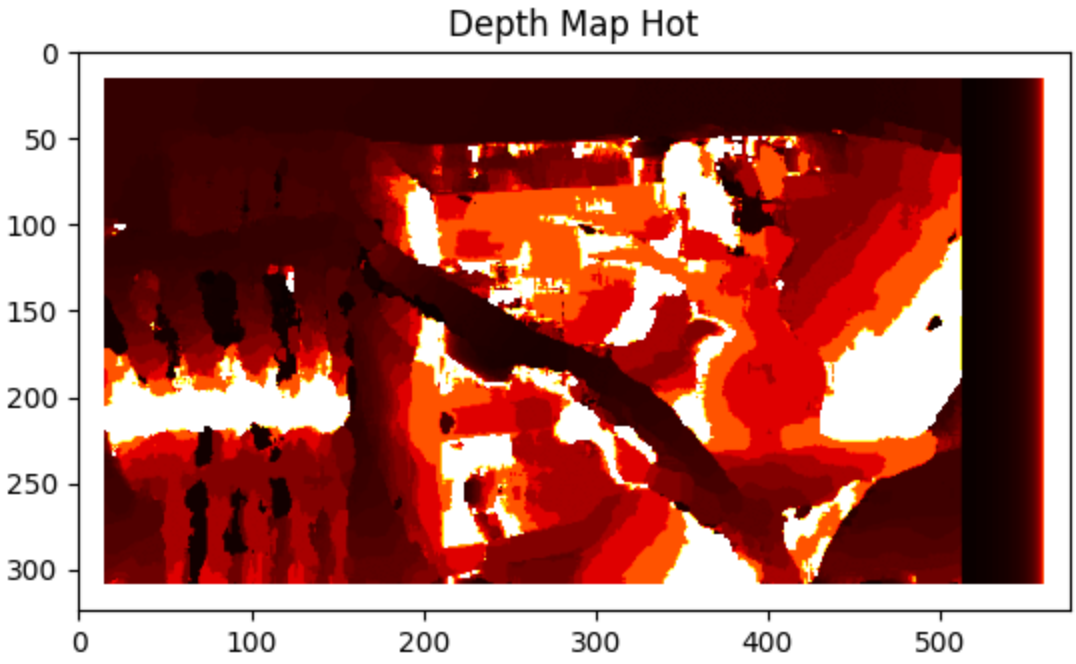




5. Compute Depth

The Disparity map is the inverted (element wise) depth map of the objects in the image, so we can compute the depth using the disparity map.





As you can see in the heat map, the objects that are closer to the camera are dark red and the objects that are far away from the camera are light red.

6. Problems Faced

The main problems faced were implementing the RANSAC to compute the best fit F matrix, I believe this can be made better by normalizing the optioned key feature points and then calculating the F matrix. I used a while loop to get the best for the F matrix, which might take longer to get the results, as it depends on the random function.

Problems were also faced while implementing the SSD and it is very computationally expensive as it has 3 for loops. When we try to eliminate one for loop by vectorising, we get noise in the results.

Got different results while using cv2.cvtColor to convert to grayscale and using imread(img, 0) as both methods give different results and the latter works better. Not sure why