Visual Odometry - Stereo Matching and Rectification : ENPM673

Rahul Karanam
Robotics Graduate Student
University of Maryland, College Park
College Park, MD
rkaranam@umd.edu

I. Introduction

This report explains the process of implementing a stereo vision system by computing a 3D depth image of the scene taken from two vantage points.

There are three datasets provided to us for implementing this pipeline.

First section will explain about the calibration process of computing fundamental matrix and essential matrix.Later decoupling essential matrix into transnational and rotational matrices for computing the pose of the cameras.

Later sections discussed about stereo rectification using the camera pose using chirality condition and then computing homography matrices to warp two images into a common plane.From then we compute the disparity map and depth map using Simple sum of squared differences and semi-global block matching algorithm.

Please refer to the figure -1 for the pipeline used to achieve the above process

II. Stereo Calibration

Stereo Calibration:

In this section , we will look how to calibrate our two images and find out the fundamental matrix and essential matrix.

Steps followed to achieve the process :

1) First we need to find best feature descriptors among both the images which correspond to similar points in the images. After matching these points we take the best n points based upon the distance.
2) Second, we use create a ORB Detector using the opencv inbuilt function cv2.ORBcreate(). We use this detector to extract the keypoints from the image.
3) I have initialised my keypoint features as 10000 because, I need more features in the start and later on i can disregard them based upon the best matches
4) We use orb.detectandcompute to find the key points and descriptors from both the images. (Think keypoints as corner points which are invariant to image transformations).
5) Now we have computed the features , we need to match these features using a cv2.DescriptorMatcher

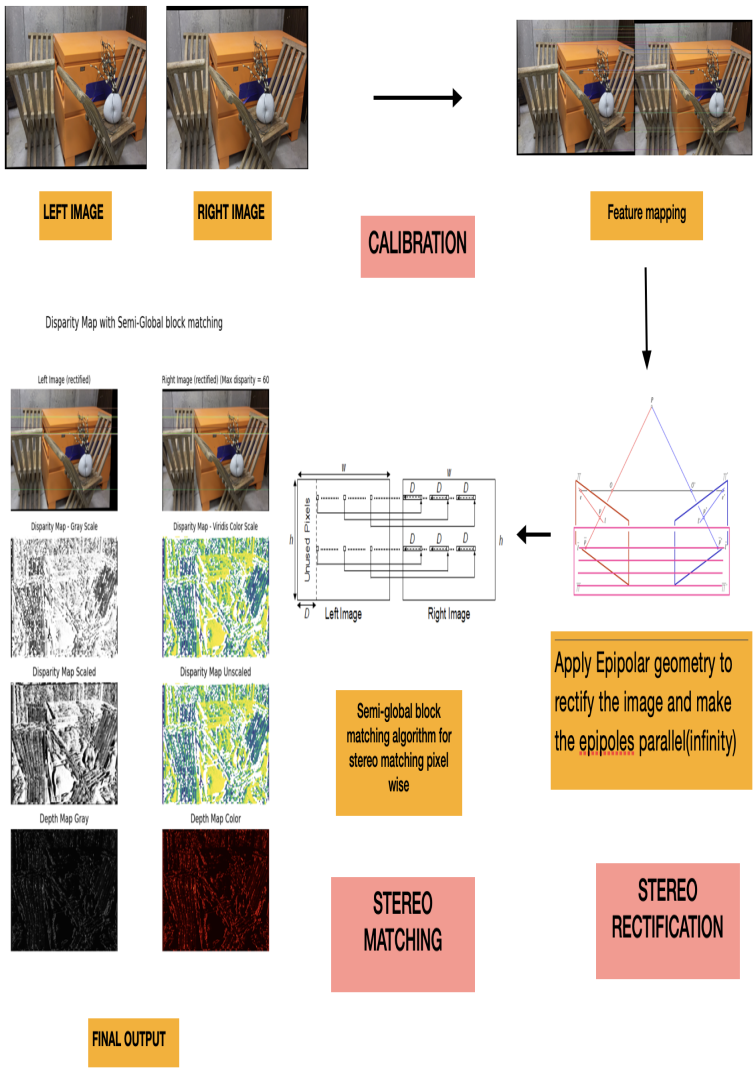


Fig. 1: Pipeline for Visual Odometry

function(using Brute Force hamming) i.e it find and matches features which are closely related to the euclidean distance between them.

6) Now we need to sort these matched in the increasing order of distance between them,and take the best matches from it (top 10 percent) i.e best match is the once which has less distance.
7) We normalize these points before applying RANSAC.
8) Now we will find the fundamental matrix and use RANSAC to find the best fundamental matrix in order to remove outliers.
9) We use the Intrinsic matrix provided in the dataset to compute the Essential Matrix.
10) We now decouple essential Matrix to rotational and translational matrices using SVD.

Fundamental Matrix & Essential Matrix

It is also known as eight point algorithm , because we take eight features from each image.It gives us the mapping between points in one image to lines in the other image.Please refer figure - 2 to explain about the process to find out fundamental and essential matrix using epipolar geometry.

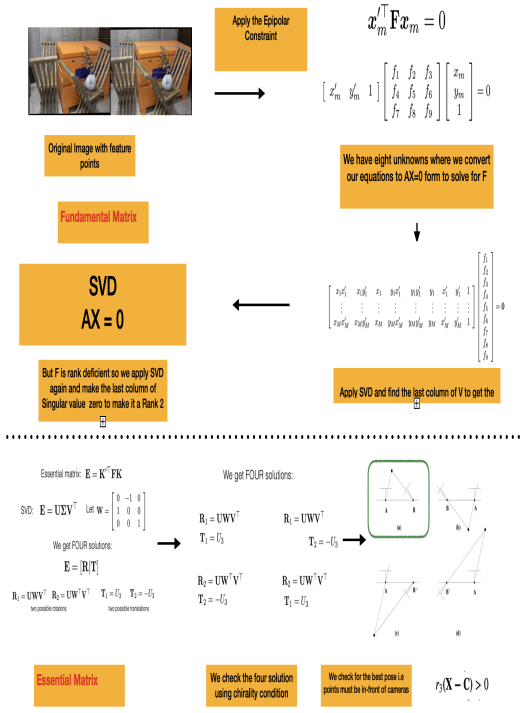


Fig. 2: Fundamental Essential Matrix

Essential Matrix is the mapping between the points and images of a uncalibrated stereo images. We use the internal camera parameters to find out the essential matrix. After computing the essential matrix, we now find the camera pose using 4 different orientations. We check all the four orientation and look for the one where we get the positive determinant

Please refer figure - 3 to show the feature matches for each dataset.

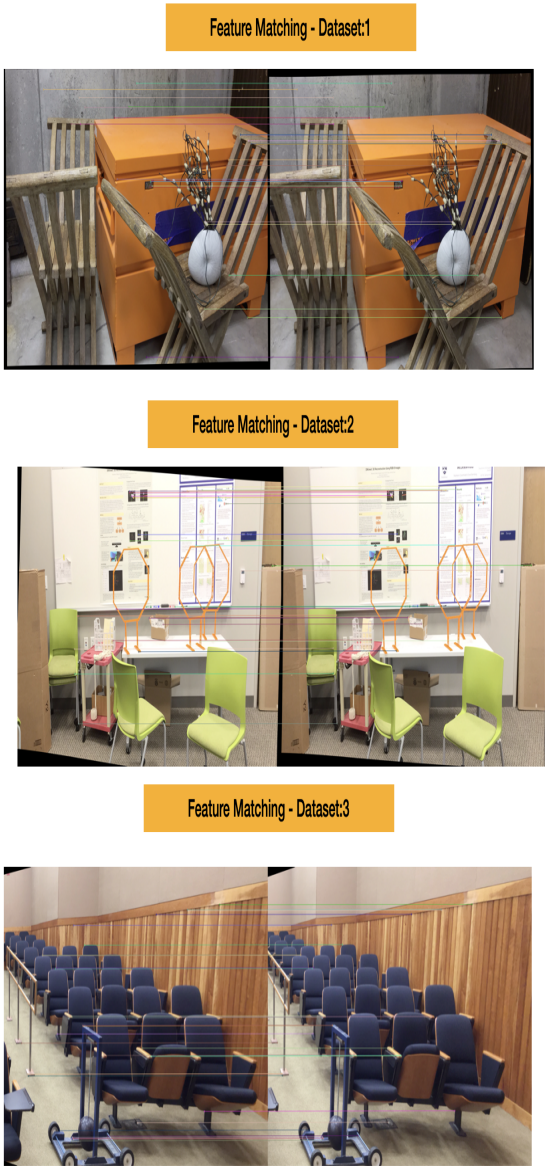


Fig. 3: Feature Matching

III. Stereo Rectification

Rectification of Images to make epipolar lines parallel: The goal of this section is to rectify the images in order to make the epipolar lines parallel to each other so as to compute disparity and depth in the later sections. By making these axis parallel will reduce the computation to 1-D which is done by making the camera setups parallel using homography transformations.

Please find below the steps in order to rectify the image and make the epipolar lines parallel

1) In order to rectify the image I have used opencv inbuilt function stereoRectifyUncalibrated because we have not been provided the distortion parameters , we cannot directly use the stereoRectify function.

2) Using the above function returns two homography matrices between the two images(left and right).
3) Now we transform our feature points to homogenous coordinates using the H1 and H2.
4) Then we warp the images to make them parallel or make the epipoles point at infinity.
5) In order to draw epilines , I have used computeCorrespondEpilines inbuilt function to return the epilines.
6) It takes the input as both images and feature points and return the lines.
7) I have used opencv circle and line to show the epilines and feature points on the warped images.

IV. Stereo Matching

After making the images and epipoles parallel , now we need to compute the disparity between images.Disparity measures the displacement of a point between two images.Higher the disparity, closer the object in the scene.

I have used two techniques to match the stereo using a simple slide block match technique using Sum of Squared Distance(SSD) as the matching cost and the other one is to use Semi-global block matching algorithm(SGBM) to find the best match.

I have used python package **Numba** which is Just in Time tool which runs your machine code in runtime which have reduced the overall computaion time by 60% percent.

A. Sum of squared differences

In case of the sum of squared differences the matching process is penalised quadratically instead of linearly making use of the squared difference instead.

SSD mainly calculates the sum of difference between these points in one image corresponding to a sliding block in other image, it is penalised quadratically.

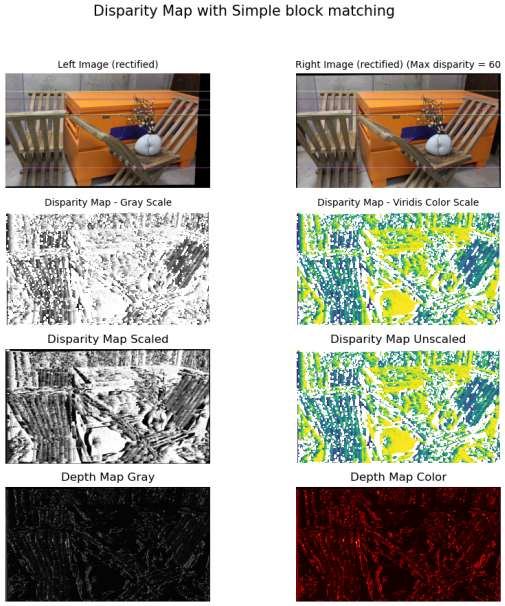
$$SSD(p,q) = \sum_{x=1}^{W}\sum_{y=1}^{H}(p(x,y) - q(x,y))^2$$

B. Semi-global matching

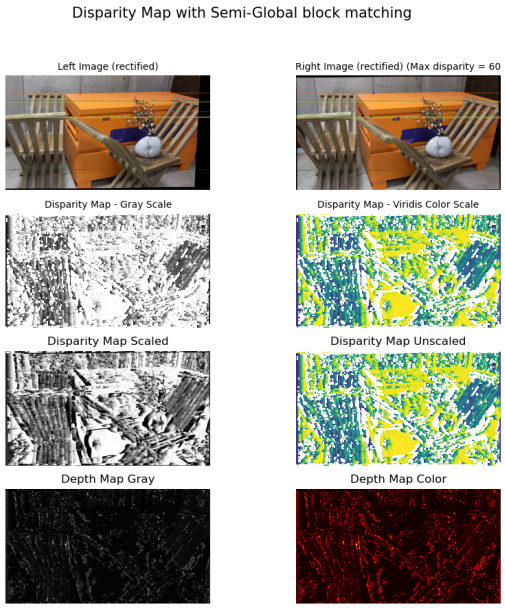
In this approach, a semi-global approach is taken rather than looking for the best fit in the scanlines, we use a global optimization technique is used. We calculate cost between each pixel and its surrounding pixel in 4-connected directions which have a similar depth. The energy equation can be written as below:

$$\min_{z}\left[\sum_{i\in\mathcal{V}} g_i\left(z_i\right) + \sum_{(i,j)\in\mathcal{E}} f_{i,j}\left(z_i, z_j\right)\right]$$

where $\mathcal{V}$ are the image pixels, $\mathcal{E}$ the edges, the connections between two pixels. The g_i are given by the cost volume and the pairwise cost $f_{i,j}$ defines a penalty for jumps between neighbouring pixels.

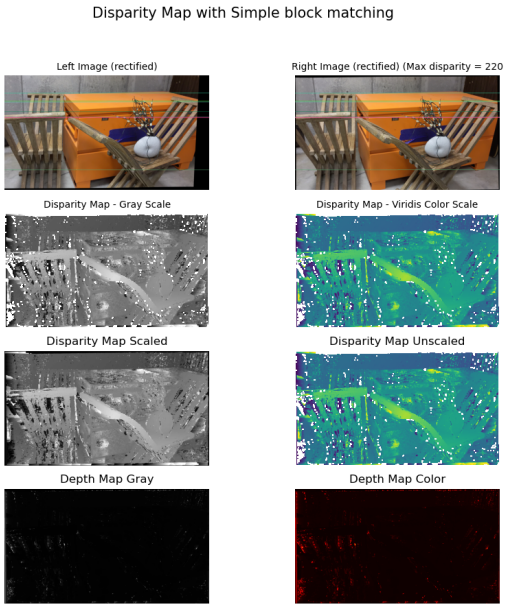


(a) Dataset -1 output with simple block matching with ndisp = 60

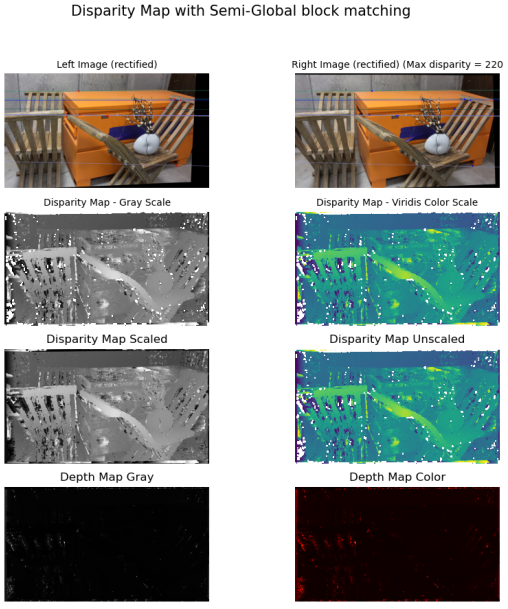


(b) Dataset - 1 output with Semi-global block matching with ndisp = 60

Fig. 4: Stereo Matching Rectification output for Dataset - 1



(a) Dataset -1 output with simple block matching with ndisp = 220



(b) Dataset - 1 output with Semi-global block matching with ndisp = 220

Fig. 5: Stereo Matching Rectification output for Dataset - 1

$$f_{i,j}\left(z_i, z_j\right) = \begin{cases} 0, & \text{if } z_i = z_j \\ L_1, & \text{if } |z_i - z_j| = 1 \\ L_2 & \text{else} \end{cases}$$

This is done as following: First, we calculate disparity in all the directions , then we assign it to $\overrightarrow{0}$.

$$m_{i+1}^{a}(t) = \min_{s \in \mathcal{D}} \left[m_i^a(s) + f_{i,i+1}(s,t) + g_i(s)\right]$$

We calculate the cost in every direction and add a penalty to correct the noise.We get a 1-D cost which we need to find the minimum cost that implies the best match.

$$b_i(s) = g_i(s) \sum_{a \in \{L,R,U,D\}} m_i^a(s)$$

Disparity is believed to be best by following the below condition.

$$\hat{d}(x,y) \in \arg\min_d b(x,y,d)$$

After finding the final cost between the images , we then calculate the disparity between the images and save it as grayscale and a heatmap image.

I have scaled the disparity map to be in the range between 0 -255 usin cv2.normalize.

Problems faced and Solutions

- First the time complexity of the algorithm was too large and it initially took longer time for implementing the SBGM algorithm.
- Using **Numba** as a JIT tool have reduced my computation time. Changing the Number of disparity level have also resulted in faster computation and I have generated different outcomes for various disparity level, please refer figure - 5,6,7,8,9,10 for your reference.
- I have been comparing with ndisp of 60 and 220 for the first dataset.
- As you can see if I increase the ndisp the disparity map looks smoother while the depth map is lost due to noise.

V. Compute Depth Image

Now we compute the depth image using the disparity map found in the previous section. We need the baseline and focal length for calculating the disparity. Depth is inversly proportional to disparity and directly proportional to the product of baseline and focal length of the camera.

Please refer figure - 5,7,8 for the images of depth image in both grayscale and heatmap color.

A. Github

Please refer to this github repository for the above code base.

Github: https://github.com/karanamrahul/Visual-Odometry.git

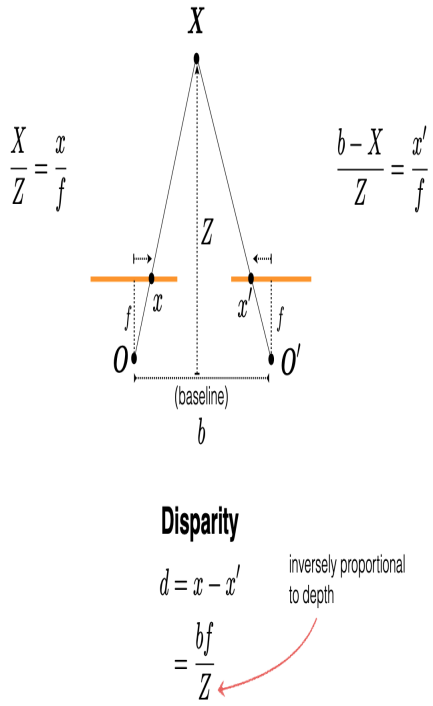
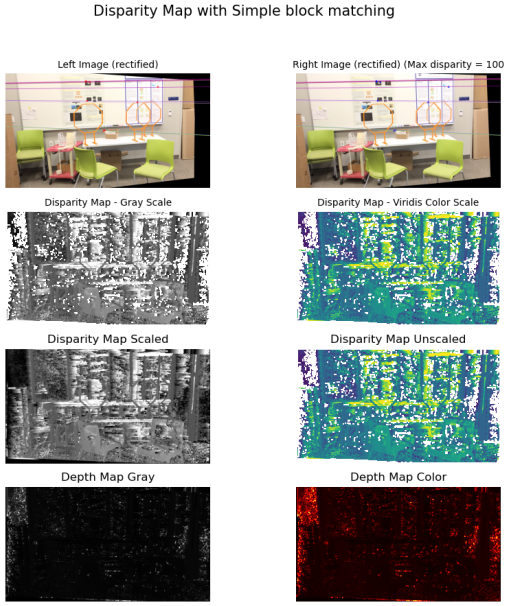


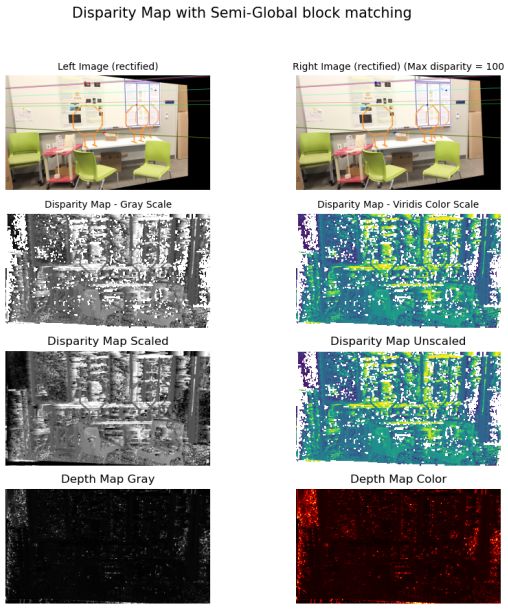
Fig. 6: Computing Depth using Disparity

REFERENCES

[1] ENPM 673, Robotics Perception Theory behind Homography Estimation Document
[2] https://cmsc733.github.io/2022/proj/p3/
[3] https://www.mathworks.com/help//visionhdl/ug/stereoscopic-disparity.html
[4] http://www.cs.toronto.edu/ fidler/slides/2015/CSC420/lecture12hres.pdf
[5] https://numba.pydata.org
[6] https://www.cs.cmu.edu/ 16385/s17/Slides/
[7] https://docs.opencv.org/3.4/da/de9/tutorialpyepipolargeometry.html

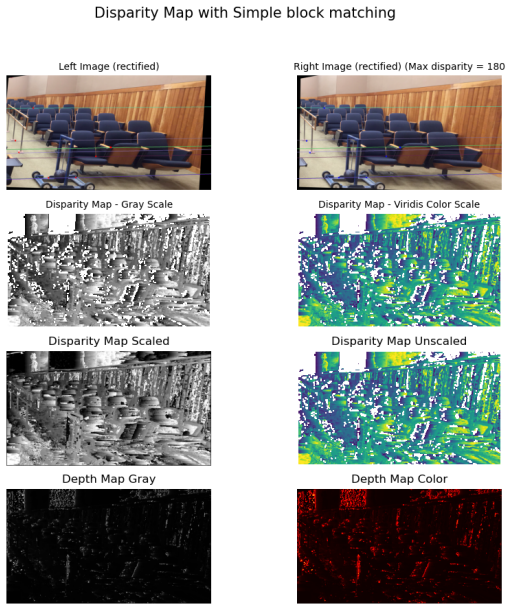


(a) Dataset - 2 output with simple block matching with ndisp = 80

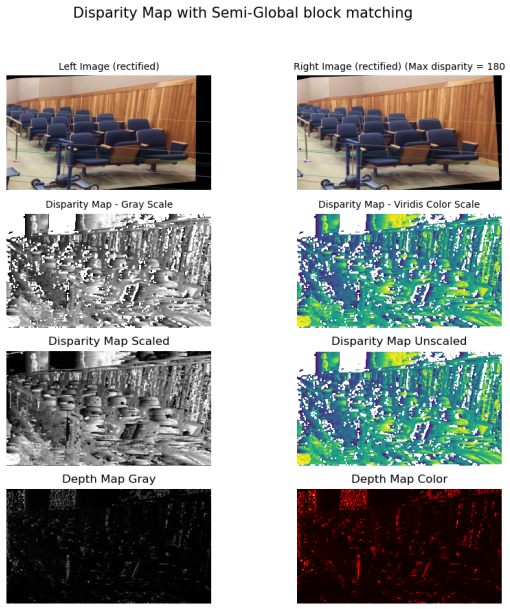


(b) Dataset - 2 output with Semi-global block matching with ndisp = 80

Fig. 7: Stereo Matching Rectification output for Dataset - 2



(a) Dataset - 3 output with simple block matching with ndisp = 180



(b) Dataset - 3 output with Semi-global block matching with ndisp = 180

Fig. 8: Stereo Matching Rectification output for Dataset - 3