ENPM 673 Project 3

Sahruday Patti

University of Maryland, College park

Summary The concept of stereo vision will be implemented in this project. We'll be given three data sets, each of which comprises two photographs of the same scenario shot from two distinct camera perspectives. We can gather 3D information by comparing information about a scene from two vantage points and assessing the relative locations of objects. We also estimate the camera pose wrt world.

Keywords · Calibration aka Camera Pose · Rectification · Correspondence · Depth Image ·

✉ sahruday patti: sahruday@umd.edu

Calibration

In this section we estimate the cameras position wrt world coordinates. we have been given the intrinsics of the camera already. By using this information, the following pipeline is followed to estimate the camera pose.

Feature Matching

- The given image size is big. So, resize the images to 60 percent of scale.
- Convert the resized images to gray scale
- Use SIFT to detect corner points
- Use brute force matcher to detect matching corner points in both images
- The brute force algorithm uses the eulidean norm as a distance measure to match the feature points.
- sort the matching points in order of their distance and choose the top 100 feature points i.e points with less distance between them.
- The correspondence between the top points are also not accurate. we can choose the best corresponding points for our computation of fundamental matrix using RANSAC.

Fundamental matrix

- We know that, a point in the image can be anywhere along the ray drawn from that point to the world. We also know that, lines correspond to lines in the perspective projection. so a line containing a center of projection and point P in the left image will project a line onto the right image. Now, any point in the world along that ray will be on the line(epipolar line) of right image. The epipolar line is made up by the plane containing image centers and point p in the world and image plane intersection. The converse is also true. so any point on the epipolar line of right image will be on the epipolar line of left image. This is the epipolar constraint. this constraint helps in reducing the correspondence problem to a one dimensionel search meaning, we are looking for matching points, and above correspondence points some are not accurate. so, we can search for that points on the epipolar line. if our cameres are parallel our lines will be horizontal and we can search along that line. if our cameras are converged, we rectify it, meaning make our cameras parallel, to get parallel epipolar lines and then search for the match.
- The Fundamental matrix is an algebraic representation of epipolar constraint geometry.($P^T F P^{'} = 0$) Thus, The fundamental matrix is a relationship between any two images of the same scene that limits the location of point projections from the scene in both images and is computed by The eight-point algorithm, which is used in the code. we can use more than eight points to compute the F. I tried this approach by taking 15 points and using ransac to estimate F(9 unknowns) by SVD. I have also played around with the normalization of the points, which gives quicker results but haven't found any difference between normalizing and not normalizing the points, where if i was not normalizing i was getting singular matrices more often.
- we know that, epipolar line is given by $F * P$. the line needs only two parameters, so the rank of matrix F is 2. so i corrected the rank by setting the last column of F to 0. I have played with the output of first dataset image, where i can see the epipole by not setting the rank to 2.

Essential matrix

- After computing the Fundamental matrix, the essential matrix can be found if we know the camera intrinsics. The essential matrix gives us the information about the camera pose that is how much the camera is rotated and translated from one image to other.$E = K_{right}^T * F * K - left$

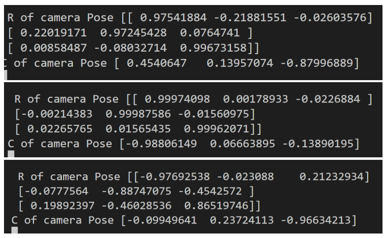
Camera Pose

- The essential matrix E can be decomposed to

4 mathematically possible rotation and translation matrices which has 5 DOF.

- i have followed the steps given in the reference link of project PDF. "Buildings Built in minutes" where the Cheirality condition has been explained. I have done triangulation by using direct linear transformation.

- the need for triangulation is we dont know the coordinates of point in the world. But, if we know the projection matrix and the image coordinates we can obtain a non trivial solution by total least square minimization. essentially Homography. Then the obtained points are passed through the cherality condition and the pose that satisfies the conditions of depth for the maximum number of points is chosen as the pose.



```
R of camera Pose [[ 0.97541884 -0.21881551 -0.02603576]
[ 0.22019171  0.97245428  0.0764741 ]
[ 0.00858487 -0.08032714  0.99673158]]
C of camera Pose [ 0.4540647   0.13957074 -0.87996889]

 R of camera Pose [[ 0.99974098  0.00178933 -0.0226884 ]
 [-0.00214383  0.99987586 -0.01560975]
 [ 0.02265765  0.01565435  0.99962071]]
C of camera Pose [-0.98806149  0.06663895 -0.13890195]

 R of camera Pose [[-0.97692538 -0.023088    0.21232934]
 [-0.0777564  -0.88747075 -0.4542572 ]
 [ 0.19892397 -0.46028536  0.86519746]]
C of camera Pose [-0.09949641  0.23724113 -0.96634213]
```

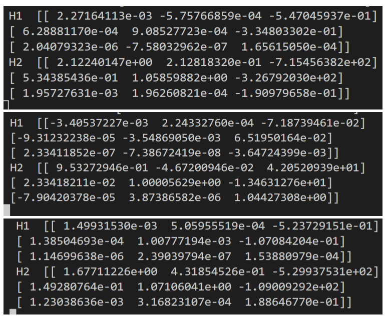
Figure 1: Camera pose for Datasets 1,2,3 respectively

Rectification

- Rectification means to make the image planes parallel to each other. This makes the epipolar lines parallel to each other. This makes our correspondence search one dimensional, that is horizontal only.

- Used Opencv inbuilt functions for rectification. I have tried to compute the epipoles and do the rectification process manually, but haven't had time to go all the way through.

- Using open cv, i computed the homography matrices H1 and H2 for the left and right camera images using the function cv2.stereoRectifyUncalib() which takes the estimated fundamental matrix and point correspondences as input. The computed H1 and H2 are used to project the images to the desired position with the help of warp perspective.



```
H1  [[ 2.27164113e-03 -5.75766859e-04 -5.47045937e-01]
[ 6.28881170e-04  9.08527723e-04 -3.34803302e-01]
[ 2.04079323e-06 -7.58032962e-07  1.65615050e-04]]
H2  [[ 2.12240147e+00  2.12818320e-01 -7.15456382e+02]
[ 5.34385436e-01  1.05859882e+00 -3.26792030e+02]
[ 1.95727631e-03  1.96260821e-04 -1.90979658e-01]]

H1  [[-3.40537227e-03  2.24332760e-04 -7.18739461e-02]
[-9.31232238e-05 -3.54869050e-03  6.51950164e-02]
[ 2.33411852e-07 -7.38672419e-08 -3.64724399e-03]]
H2  [[ 9.53272946e-01 -4.67200946e-02  4.20520939e+01]
[ 2.33418211e-02  1.00005629e+00 -1.34631276e+01]
[-7.90420378e-05  3.87386582e-06  1.04427308e+00]]

 H1  [[ 1.49931530e-03  5.05955519e-04 -5.23729151e-01]
 [ 1.38504693e-04  1.00777194e-03 -1.07084204e-01]
 [ 1.14699638e-06  2.39039794e-07  1.53880979e-04]]
 H2  [[ 1.67711226e+00  4.31854526e-01 -5.29937531e+02]
 [ 1.49280764e-01  1.07106041e+00 -1.09009292e+02]
 [ 1.23038636e-03  3.16823107e-04  1.88646770e-01]]
```

Figure 2: H1, H2 for 3 datasets respectively

Correspondence

- We match a window along the epipolar line from the left image to the right image. i couldn't figure out a way to do the same along the epipolar line but instead matched it along the whole image, since after rectification, lines are parallel, so left pixel window will be along the same line. But, only difference is i have also calculated disparity for the whole length of image i.e where i couldn't find the epipolar lines also. I have used sum of squared differences for my window matching. the lesser the squared difference the better.

- later,I have used dynamic program formulation to achieve the correspondence mapping. with this approach we can solve for occlussions and ordering constraints.

- I have created a matrix with the shape of width - blocksize of our image and maximum depth(vmax given in the calib text). Now, i need to find a path from the left pixel block to the right pixel with minimum cost. There are three possible paths, one - one correspondence, not seen in left and not seen in right. So, for a window in left image, i match the corresponding window in right image by calculating the SSD and store this value in the matrix created. if the left image window is not seen in the right image i increment the right image window and vice versa for other two paths and increment the index. This formulation is explained [1] and i tried to implement this. But the results have not been great. With more time i would have tried to refine this approach.

- Now, i take the minimum SSD value in the matrix along the width of the matrix and use it as my disparity. Later, i have normalized my disparity to be 0 - 255 and plotted the image.

- Disparity is the difference between the optical center of left image minus the location of point in the world that is xl and the optical center of right image minus the location of point in the world that is xr. If we found an exact match the dispairty is zero.

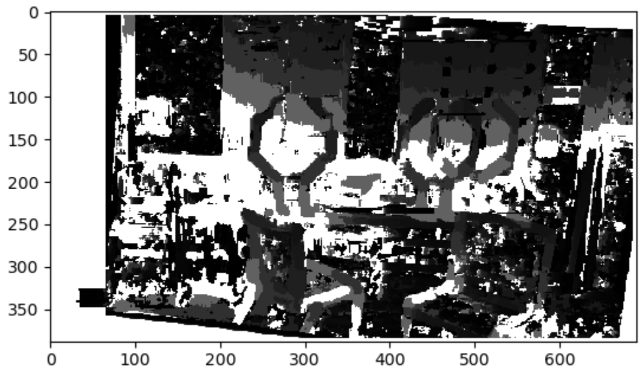
Depth

- the depth of image can be found by baseline that is the length of line joining the camera centers which contains the epipoles, the focal length and the disparity.

- i have used window based matching and computed the images. But, the results were poor. results are included in the folder.

- The computation time of the dynamic programming formulation is very huge. it takes 25 minutes on the state of the art computer system to run. so i havent been able to tune values to all the images. I have managed to show some results with dataset 2. which can be found below. Please look in the folder for the images of all the datasets.

Some of the better results i got of all the datasets are below. All the results can be found in this link: `https://drive.google.com/drive/folders/1qoMNxwzzBhpbq9r01XSNyLaOfEKa5ZM1?usp=sharing`



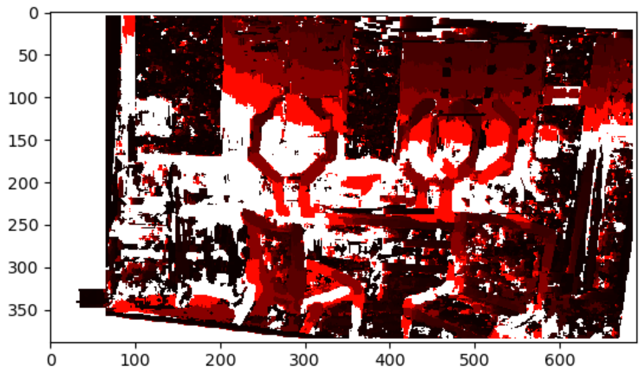


Figure 3: Depthmaps of Dataset3

References [1] https://classroom.udacity.com/courses/ud810/ 5343-4451-8682-ee0a855a6cee/concepts/52d85d50-a92a-493d-ab03-7781e5f1a3d4