Stereo Vision

UNIVERSITY OF MARYLAND 1856

A. JAMES CLARK
SCHOOL OF ENGINEERING

Pooja Kabra

April 2021

Table of Contents

Drive Link:

https://drive.google.com/drive/folders/1trD3qvgjHGw-T7HxmjufsPF9VeHjNt42?usp=sharing

Project Description

In this project, we implement the concept of Stereo Vision. We have 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.

A brief explanation of the terms used in the ground truth files:

```
cam0,1:        camera matrices for the rectified views, in the form [f 0 cx; 0 f cy; 0 0 1], where
  f:           focal length in pixels
  cx, cy:      principal point  (note that cx differs between view 0 and 1)

doffs:         x-difference of principal points, doffs = cx1 - cx0

baseline:      camera baseline in mm

width, height: image size

ndisp:         a conservative bound on the number of disparity levels;
               the stereo algorithm MAY utilize this bound and search from d = 0 .. ndisp-1

isint:         whether the GT disparites only have integer precision (true for the older datasets;
               in this case submitted floating-point disparities are rounded to ints before evaluating)

vmin, vmax:    a tight bound on minimum and maximum disparities, used for color visualization;
               the stereo algorithm MAY NOT utilize this information

dyavg, dymax:  average and maximum absolute y-disparities, providing an indication of
               the calibration error present in the imperfect datasets.
```

Calibration

First, we need to compare the two images in each dataset and select a set of matching features. You **can** use any inbuilt function for feature matching. SIFT and corner detection methods are recommended feature matching techniques.

• Estimate the Fundamental matrix using the features obtained in the previous step. Refer to section 3.2.2 in this link to get an overall understanding of Fundamental matrix estimation. You can use inbuilt SVD function to solve for the fundamental matrix. Note that you have the choice of using RANSAC method or the straight least square method to estimate the fundamental matrix.

• Estimate Essential matrix E from the Fundamental matrix F by accounting for the calibration parameters. You should implement the functions to estimate the Essential matrix and also to recover the rotation/translational matrices.

• Decompose E into a translation T and rotation R.

Sol: First, we must find as many as possible features in the left and right images. We use ORB(Oriented FAST and Rotated BRIEF) to do so; it is an efficient alternative to SIFT and SURF in computation cost, performance and patents. Unlike SIFT and SURF, we don't have to pay to use it.

Once we have the matches, we use cv.BFMatcher() to find the best matches. For distance measurement between matches, we specify Hamming distance. crossCheck is set to true to ensure consistent results. It is a good alternative to the Lowe Test in SIFT paper. Now we sort the matches by distance and pick the top 50. match.trainIdx and match.queryIdx give us indexes for corresponding keypoints. They cv.keyPoints are then converted into usable integer x,y pairs.

Flann based matcher is faster and can also be used, however as our dataset is not large, it will not make a big difference.

The 8-point algorithm requires us to have 8 correspondences between the two images. Unlike Homography matrix, we need 8 point-pairs instead of 4 because, in system of equations for H, each pair gives 2 rows of A whereas for F, it only gives 1.

The best 8 pairs are chosen by running RANSAC over the 50 shortlisted before. For every iteration, 8 random pairs are chosen, the fundamental matrix F_iter is calculated by solving the system, which is a collection of 8 constraints as such:

$$[x'_i \quad y'_i \quad 1]\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

$$[x'x \quad x'y \quad x' \quad y'x \quad y'y \quad y' \quad x \quad y \quad 1]\begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

By stacking 8 such constraints, we obtain the equation Ax = 0. This system can be solved using Singular Value Decomposition (SVD) on A. The last column of V in the decomposition USV^T will be divided by its last value and rearranged to obtain F_iter. Every time, $x'^T F\_iter\ x$ is calculated and saved in a variable min_zero. The best F is the one which gives lowest value for min_zero.

As RANSAC was taking too long for considerable number of iterations and also was giving inaccurate result when rank became 3 for F due to noise, the best F matrix was frozen for further processing.

The Essential matrix operates on image points expressed in
normalized coordinates i.e. points have been aligned (normalized) to camera coordinates.
Essential matrix is given as E=${K_1}^T$FK_0 where K_0 and K_1 are camera calibration matrices for left and right cameras.

Now we move on to decompose E matrix into Rotation and translation matrices. To do this, we compute SVD of E.

$$\mathbf{E} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^T$$

$$\mathbf{\Sigma} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The diagonal entries of $\sum$ are the singular values of E which, according to the internal constraints of the essential matrix, must consist of two identical and one zero value. Define

$$\mathbf{W} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ with } \mathbf{W}^{-1} = \mathbf{W}^T = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

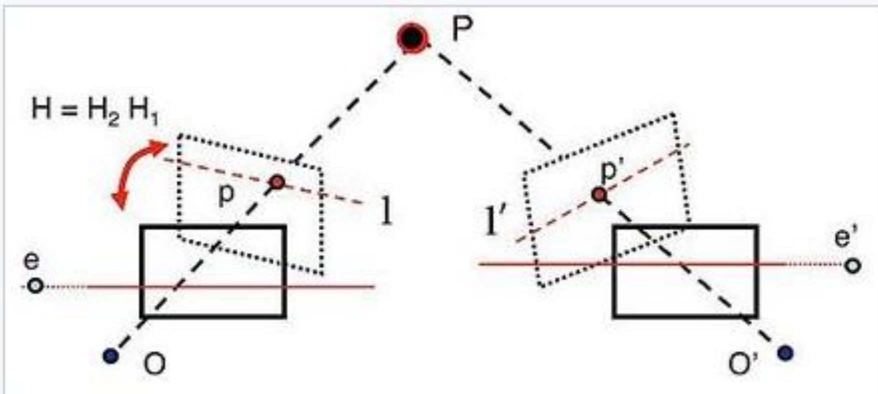
to make the following solution:

$$[\mathbf{t}]_\times = \mathbf{U}\,\mathbf{W}\,\mathbf{\Sigma}\,\mathbf{U}^T$$

$$\mathbf{R} = \mathbf{U}\,\mathbf{W}^{-1}\,\mathbf{V}^T$$

Rectification

• Apply perspective transformation to make sure that the epipolar lines are horizontal for both the images.
• You **can** use inbuilt functions for this purpose.
• Print H1 and the homography matrices for both left and right images that will rectify the images.
• Plot epipolar lines on both images along with feature points.



Model used for image rectification example.

The cv.stereoRectifyUncalibrated() function accepts the F matrix, left point set pts0, right point set pts1 and input image dimensions and outputs homography matrices H_0 and H_1.

We find the left and right rectified images using cv.warpPerspective() with H_0 and H_1.

We can now map feature points pts0 to new feature points in the rectified image pts0_new using H0.

$$H_0 p_0 = p_0\_new$$

Similarly, for the right image,

$$H_1 p_1 = p_1\_new$$

$$({H_1}^{-1} p_1\_new)^T F \, ({H_0}^{-1} p_0\_new) = 0$$

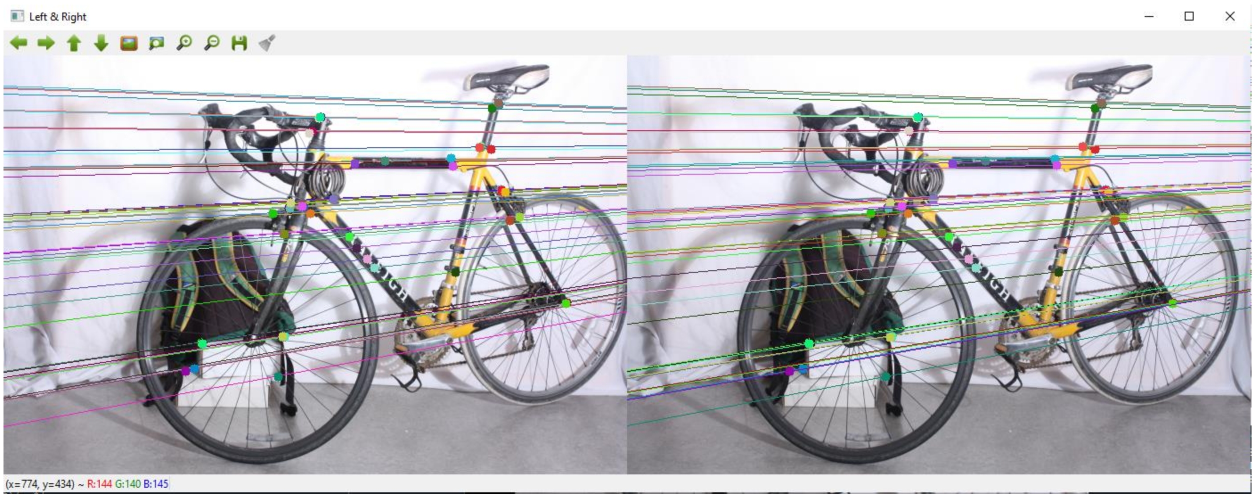
$${p_1}_{new}^{\,T} ({H_1}^{-1^T} F \, {H_0}^{-1}) p_0\_new = 0$$

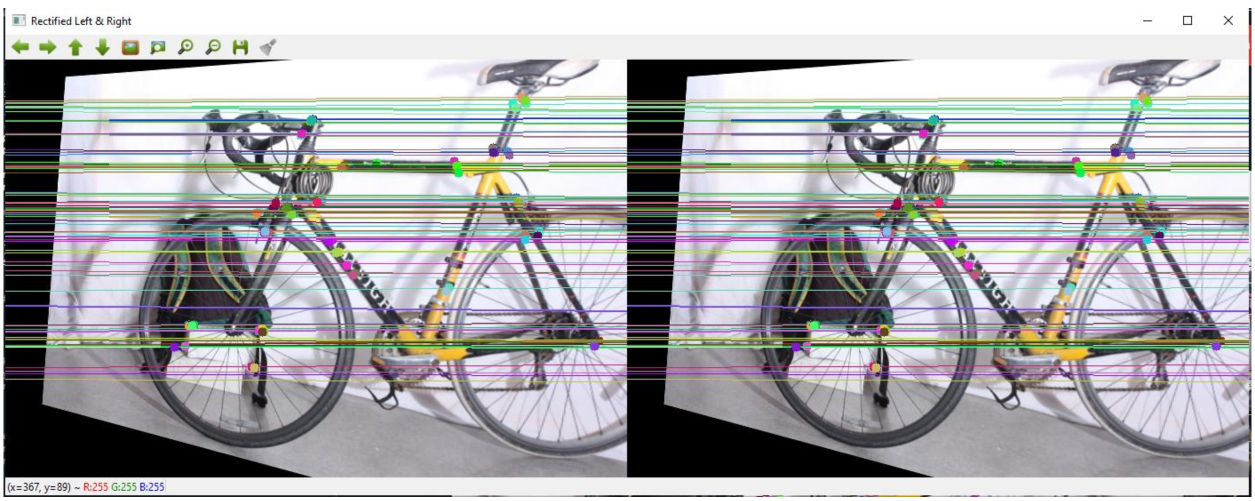
Comparing with

$${p_1}_{new}^{\,T} F_{new} \, {p_0}_{new} = 0$$

$$F_{new} = {H_1}^{-1^T} F \, {H_0}^{-1}$$

cv.computeCorrespondEpilines() finds epipolar lines using pts1_new on img0_rectified and drawlines() returns a copy of the rectified image with epipolar lines0. We repeat the same for img1_rectified. We concatenate using np.concatenate() and display using cv. imshow()

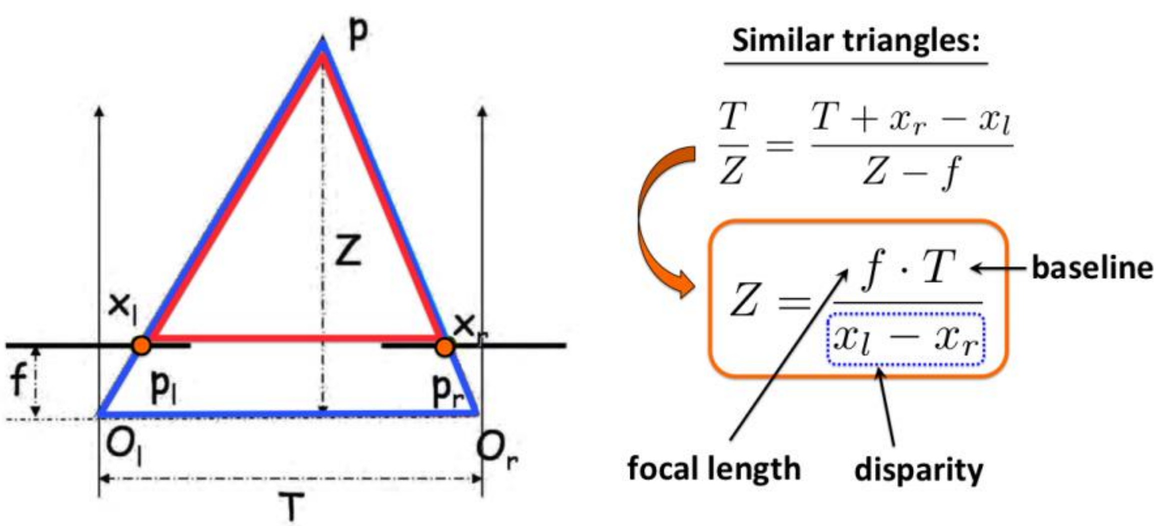




Correspondence

- For each epipolar line, apply the matching windows concept (such as SSD or Cross correlation).
- Calculate Disparity
- Rescale the disparity to be from 0-255 and save the resulting image.
- You need to save the disparity as a gray scale **and** color image using heat map conversion.

The difference in location of an object as viewed by the left and right eye is created due to parallax or horizontal separation between our eyes. This difference is called disparity and our brain uses it to compute depth information.

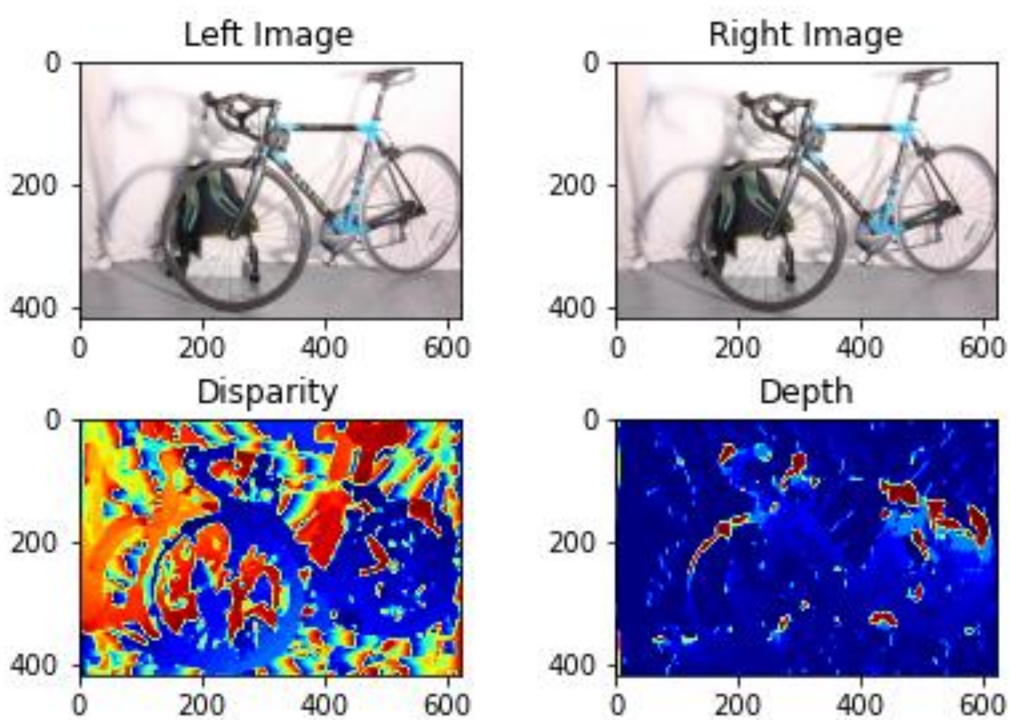


As we can see, disparity is inversely proportional to the depth. This is because, farther away the object is in the scene, the lesser it seems to move.

To calculate the disparity map, we convert the rectified images to grayscale. It is better to use resized images for the operation as the process is computationally expensive. The gist of the the process is that we want to find the twin in the right image for every pixel in the left image. We know the twin lies somewhere along the epipolar line of that point. Now that we are working on rectified images, the epilines for both images are more or less the same horizontal line(camera calibration error could introduce y disparities). We could match intensity, however, as there could be many such pixels of the same intensity, we use a small patch with certain features to find an identical region. This is also called 'Template matching'.

Depending on the template/block size we wish to use, we pad our images with zero pixels using cv.copyMakeBorder() function. For the left image, we consider pixels within the border. We fix the template on one pixel (y, xl) and slide the template along the same y in the right image. However, we don't go all the way along the horizontal line either. We consider 10% pixels to the left and right of xl to reduce what might be unnecessary computation. SAD – Sum of absolute differences is calculated for each comparison. SSD – Sum of squared differences can also be used in place. The best twin pixel (y, xr_best) is the one that gives least value for this metric. The corresponding disparity value disparity for pixel (y, xl) will be *xl - xr_best*. This however gave lighter color for farther images, so we used *xr_best – xl*, it seemed to fix the problem. Additionally, if we are using a resized image, say one that is almost $1/10^{th}$ the size of the original, the disparity values will also be scaled to 10 times.

Once this is done for all pixels, the disparity matrix disp obtained is rescaled from 0-255. This is in grayscale. cv.applyColorMap() converts it to a heatmap. We use the commonly used JET colormap.



Compute Depth Image

- Using the disparity information obtained above, compute the depth information for each pixel image. The resulting depth image has the same dimensions of the disparity image, but it has depth information instead.
- You need to save the depth image as a gray scale **and color image** using heat map conversion.

We build on the disparity matrix disp1 which contains *xl - xr_best* values. To calculate depth matrix, all values x in the disparity matrix will become
Bf/x, where B and f are Baseline and focal length respectively. To avoid divide by zero errors, all zero pixels are replaced by a close-to-zero value. The result is again normalized between 0-255 and displayed as a grayscale image and a colormap. Also, as the baseline is in mm, the depth matrix information will be in mm. The nearer points will show in shades of Red and farther points will be in Blue.

References

1. Buildings built in minutes - An SfM Approach
   https://cmsc733.github.io/2019/proj/p3/

2. Epipolar Geometry
   https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_epipolar_geometry/py_epipolar_geometry.html

3. Image rectification
   https://en.wikipedia.org/wiki/Image_rectification

4. Essential Matrix
   https://en.wikipedia.org/wiki/Essential_matrix

5. Stereo: Parallel Calibrated Cameras
   http://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12_hres.pdf

6. Disparity Map
   https://jayrambhia.com/blog/disparity-mpas

7. Depth Map from Stereo Images
   https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html

8. Disparity map calculator
   https://github.com/khmariem/disparity_map/blob/main/disparity.py