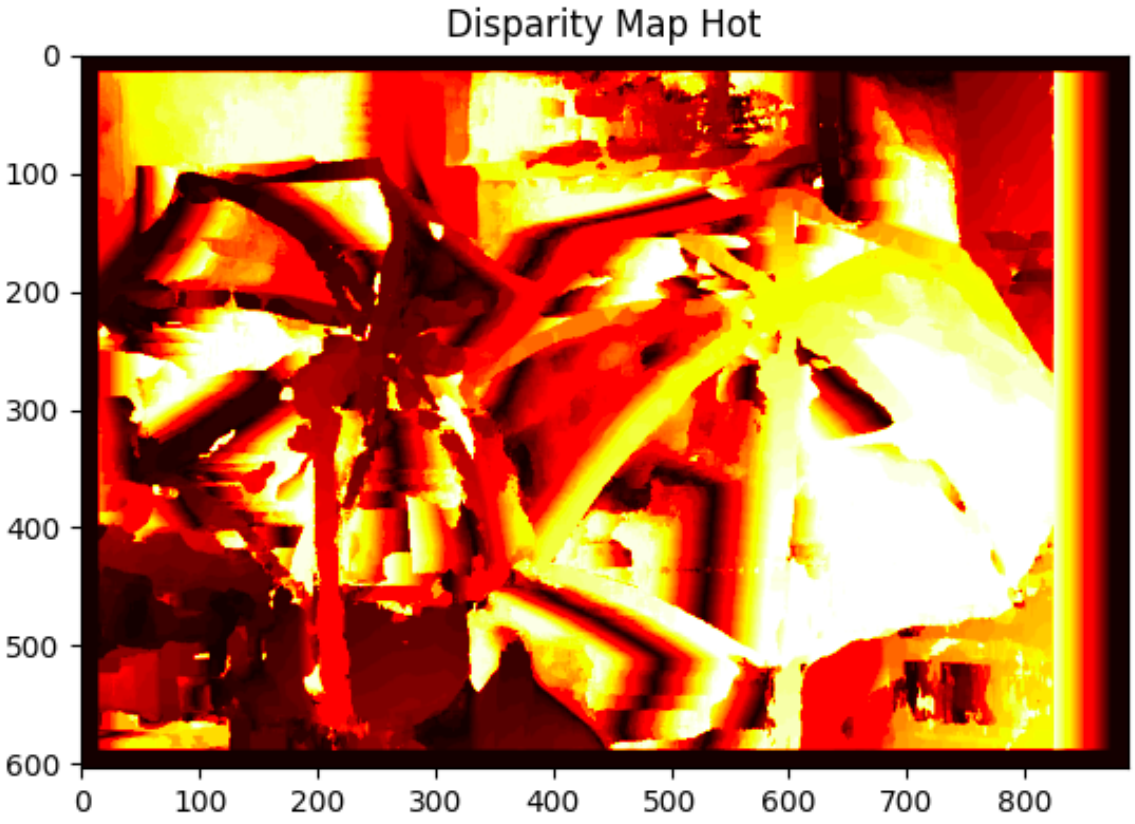
PROJ3 REPORT

ENPM673 - Perception For Autonomous Robots - Spring 21



Paras Savnani

04.26.2021

University Of Maryland College Park

For stereo-vision applications, we have to apply a pipeline of operations to find the depth map of a region in space. For this we require two cameras to capture an image at the same time from different angles and positions, and we triangulate the location of each pixel from the cameras to find the z coordinate.

main.py

It has the main function which combines the pipeline for the stereovision.

1. Camera parameters for all the datasets are defined here, which are used to calculate the essential matrices for them.
2. For the calibration step it outputs the F, E, Camera pose to use for rectifying the image.
3. The rectification step both the images, points and F matrix as input and outputs the rectified images with the corresponding points.
4. **cv2.computeCorrespondEpilines()** is used to find the epilines for the images.
5. In the Correspondence step, disparity map (scaled and unscaled) is returned.
6. This disparity is fed into the disparity_to_depth function, with baseline and f corresponding to the dataset.
7. Finally the disparity and depth images are plotted in the matplotlib window.

CALIBRATION

Objective: Compute the feature points and estimate the fundamental matrix and the essential matrix from them. Furthermore find the correct Camera Pose (Rotation and Translation) from the E matrix.

CODE EXPLANATION AND PROCEDURE:

calibration.py

For this step we have to find the fundamental matrix, the essential matrix and the camera pose (Rotation and Translation).

def draw_keypoints_and_match(img1, img2):

1. Here we create an **ORB feature detector**, to generate max keypoints(10000) find keypoints and descriptors in the given image. The SIFT feature detector is not used because it can be patented in future and they may break the code.
2. Then, we use a **Brute Force**/ FLANN based matcher to find the best matches in the keypoints between 2 input images based on hamming distance.
3. Finally, the matches are sorted based on distance and 30 best matches are selected from the total matches.
4. To fill the final list of feature points, points are extracted from the matches and the matches are plotted simultaneously on both images to visualize them.

def calculate_F_matrix(list_kp1, list_kp2):

1. Here we calculate the Fundamental matrix using the least squares method. We have 8 equations and we input exactly 8 points to form the A matrix from them.

2. We the take the Singular Value Decomposition of the F matrix to solve the homogeneous equation:

$$Ax = 0$$

3. After getting the A matrix,we find the lowest singular value using the last column of the V_t matrix.
4. Due to noise in the correspondences, the estimated F matrix can be of rank 3 so we downgrade the rank of the F matrix to 2, by making the last column of the V_t matrix of svd decomposition of F.

def RANSAC_F_matrix(list_of_cood_list):

Here, the **RANSAC algorithm** is applied to choose the best F matrix. 8 random points are sampled and fed into the calculate_F_matrix function iteratively and we calculate the error in estimating the points. Comparing this with a predefined threshold, we can classify the inlier points and choose the Best F matrix based on the max_inliers criteria.

def calculate_E_matrix(F, K1, K2):

Here, the Essential matrix is calculated using the Fundamental matrix and the intrinsic camera parameters.

$$E = K_2^T F K_1$$

def extract_camerapose(E):

This function is used to extract the 4 camera pose solutions form the E matrix by calculating the svd of the E matrix as UΣVT:

C1=U(:,3) and R1=UWVT

C2=−U(:,3) and R2=UWVT

C3=U(:,3) and R3=UWTVT

C4=−U(:,3) and R4=UWTVT

Where **W** =

[0,-1, 0],

[1, 0, 0],

[0, 0, 1]]

This function returns a list of Rotation and Translation vector solutions for each camera pose possible.

def disambiguate_camerapose(camera_poses, list_kp1):

This function is used to shortlist the best camera pose based on the **triangulation check for the chirelity condition**. It finds the depth signs for all the points present in the space for all camera poses by checking the following condition:

$$R3(X - C) > 0$$

where, R3 is the 3rd column of the rotation matrix, X is point in homogeneous coordinates and C is the Camera centre.

def drawlines(img1src, img2src, lines, pts1src, pts2src):

This function is used to visualize the epilines of the feature points found in the image. It takes lines input from the **cv2.computeCorrespondEpilines()** function and takes in the left and right images with the feature points to plot the lines.

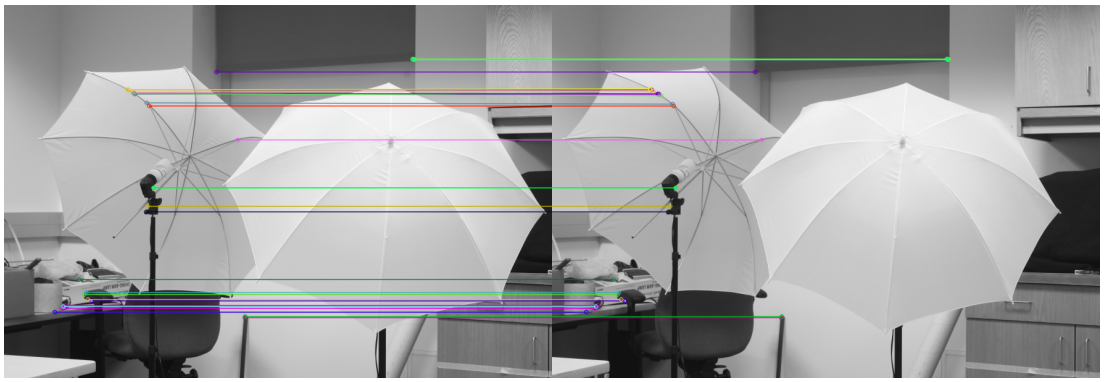
PROBLEMS FACED AND SOLUTIONS:

- **ORB** feature detector is not scale invariant, so when I computed the feature points on different, it gave different points, so it affected the robustness of the algorithm. To counter this, a constant scale value is used.
- As **Ransac** is a sampling based algorithm, it is not guaranteed to always give best results, so sometimes the F matrix varied which gave bad warping in the rectification step. To counter this, a while(1) loop is used to re-generate feature points and apply the calibration pipeline again if the error occurs in the rectification step and break if a good solution is found.
- F matrix depends on the number of matches chosen from the BF matcher. The value of the matches had to be fine-tuned and RANSAC iterations had to be fine tuned to get the best F matrix for all 3 scenarios.

RESULTS:

Original left image

Original right image



Feature point matches

RECTIFICATION

Objective: Warp the images to make the epipolar lines horizontal (epipoles at infinity), so that disparity calculation is restricted to one dimension only. Essence of this step is to make the camera setups parallel after homography transformation.

CODE EXPLANATION AND PROCEDURE:

rectification.py

def rectification(img1, img2, pts1, pts2, F):

1. To rectify the images we use **cv2.stereoRectifyUncalibrated()** function, because camera distortion parameters are not given so we cannot use **cv2.stereoRectify()** and we input the F matrix into this function with feature points and image size.
2. It returns the homography matrices for both the cameras (H1 and H2).
3. We transform the feature points (in homogeneous cood.) by multiplying them with the H matrices.
4. Finally, we warp both the images using H matrices to make the epilines horizontal.

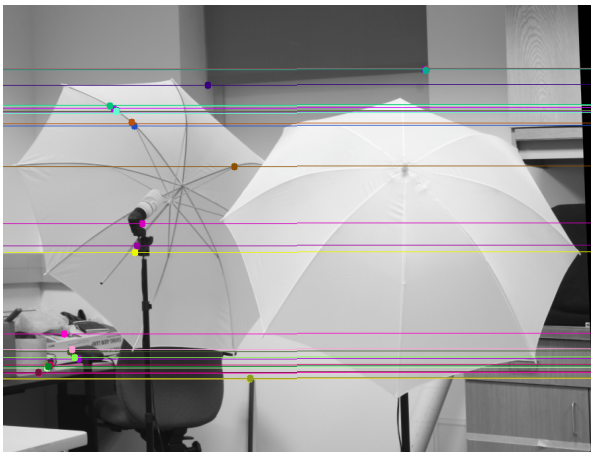
PROBLEMS FACED AND SOLUTIONS:

- Even after rectification the feature points and corresponding epilines were not horizontal, so to correct this each feature point has to be multiplied by the homography matrix to transform it according to the rectified images.

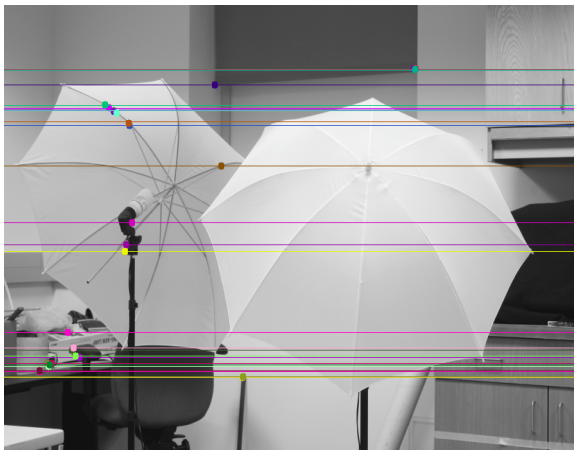
RESULTS:

Rectified image1

Rectified image 2



Rectified image1 with epilines



Rectified image 2 with epilines

CORRESPONDENCE

Objective: For each pixel in the image calculate disparity using a sliding window and sum of squared differences approach and save the grayscale and heatmap images.

CODE EXPLANATION AND PROCEDURE:

correspondence.py

def sum_of_squared_diff(pixel_vals_1, pixel_vals_2):

This function calculates the sum of squared differences between two input pixels blocks and returns the following:

$$\Sigma(x_1 - x_2)^2$$

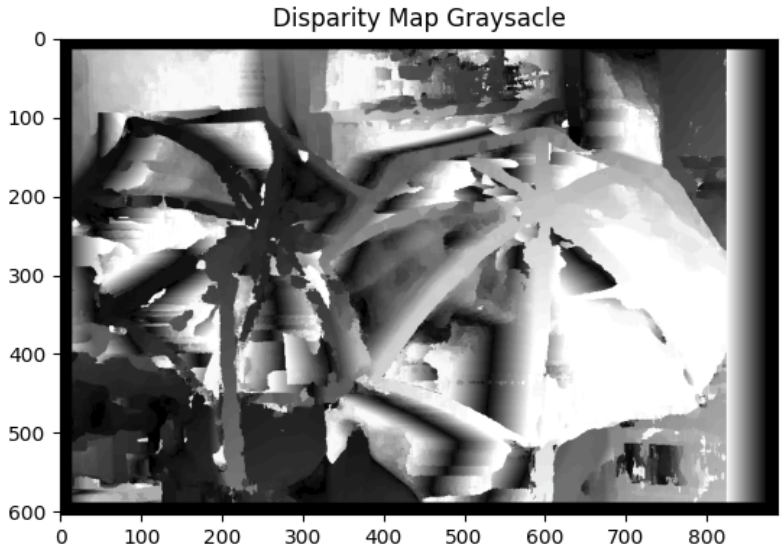
def block_comparison(y, x, block_left, right_array, block_size, x_search_block_size, y_search_block_size):

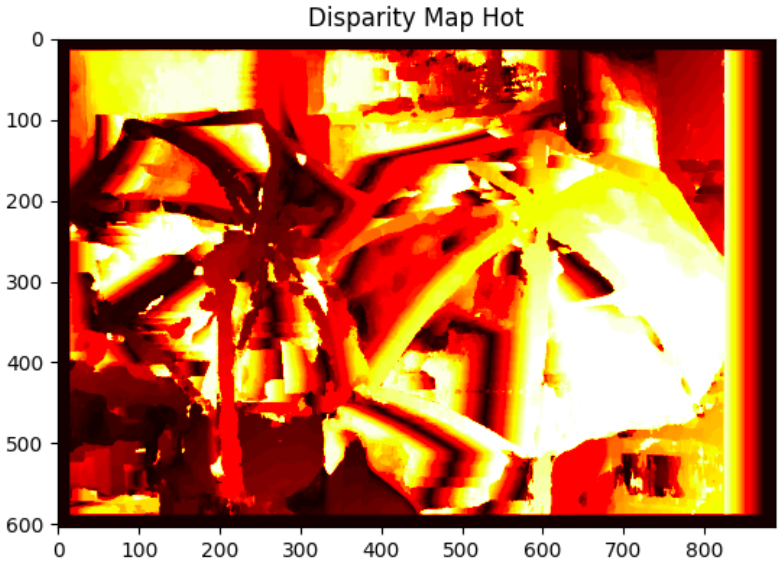
1. This function is used to search for the best matching pixel in the right image for a particular pixel in the left image. It takes the x,y position of the pixel, right image, left block, block size and search ranges for x and y dimension.
2. For given search ranges it calculates the ssd of the blocks and finds the index with the minimum difference in the block intensity values.

def ssd_correspondence(img1, img2):

1. This function is the SSD correspondence function, which applies the block comparison on the whole left image and generates the disparity map by subtracting the values of the pixel intensities from the left and right images.
2. Finally, we scale the disparity map to increase its dynamic range(0-255) and improve visualization.
3. This function returns both scaled and unscaled disparity maps.

RESULTS:


Disparity Map Graysacle
0
100
200
300
400
500
600
0
100
200
300
400
500
600
700
800


Disparity Map Hot
0
100
200
300
400
500
600
0
100
200
300
400
500
600
700
800

PROBLEMS FACED AND SOLUTIONS:

- This step has 4 nested loops - 2 for image iteration and 2 for x/y search range iteration. So it has time complexity of $\mathbf{O}(n^4)$, therefore increasing the image resolution or search block ranges makes the code to run for long times. Also, there is a trade-off between disparity noise and compute times. So, this problem can be solved by using efficient algorithms to compute disparity rather than using naive for loops.
- Just an observation: Scaling the disparity map has the same result as applying the histogram equalization to increase the dynamic range of the image.

COMPUTE DEPTH IMAGE

Objective: Using the disparity map from previous step compute the depth information for each pixel using the below formula:

$$depth = \frac{baseline * f}{disparity}$$

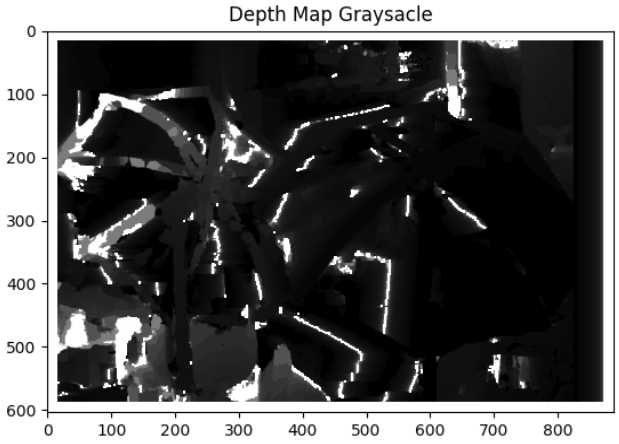
CODE EXPLANATION AND PROCEDURE:

depth.py

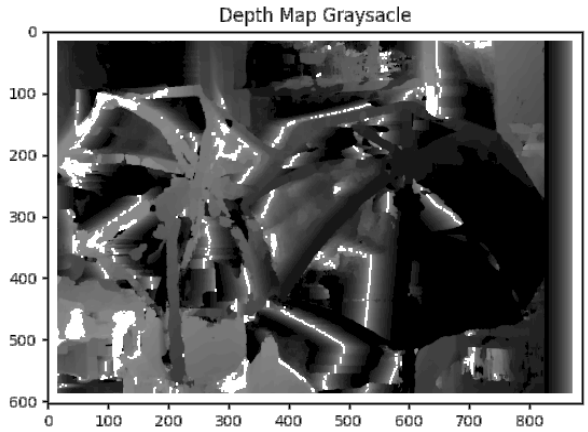
def disparity_to_depth(baseline, f, img):

This function calculates the depth map of the from the disparity map and the baseline and focal length information. To calculate the depth values it uses above formula and scales the images afterwards to improve the dynamic range. It also returns the **depth array** having actual depth values at the corresponding locations.

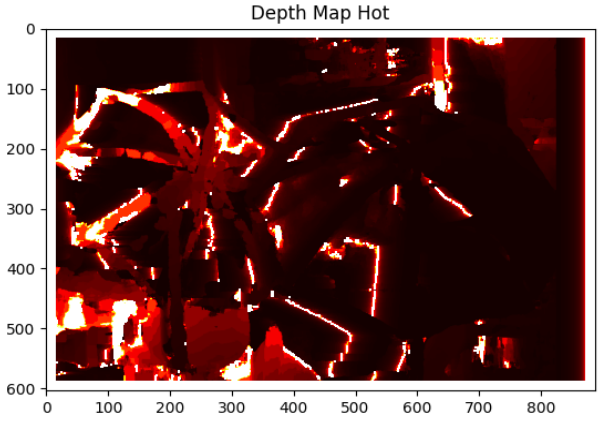
RESULTS:



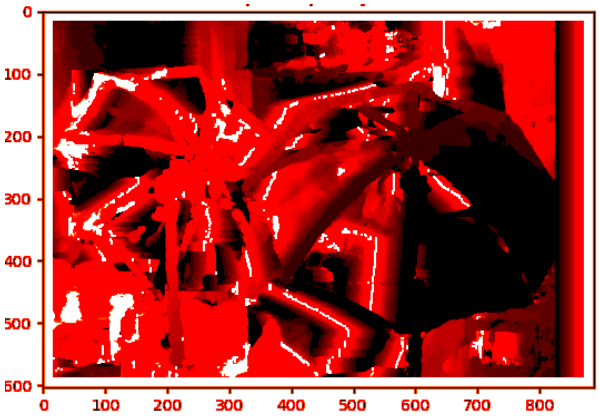
Depth map unscaled



Depth map scaled



Depth heatmap unscaled



Depth heatmap scaled

REFERENCES

- https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html
- https://cmsc733.github.io/2019/proj/p3/#estE
- https://www.cc.gatech.edu/classes/AY2016/cs4476_fall/results/proj3/html/sdai30/index.html
- https://stackoverflow.com/questions/30716610/how-to-get-pixel-coordinates-from-feature-matching-in-opencv-python
- https://stackoverflow.com/questions/27856965/stereo-disparity-map-generation
- https://stackoverflow.com/questions/36172913/opencv-depth-map-from-uncalibrated-stereo-system
- https://www.cc.gatech.edu/classes/AY2016/cs4476_fall/results/proj3/html/sdai30/index.html
- https://pramod-atre.medium.com/disparity-map-computation-in-python-and-c-c8113c63d701
- https://stackoverflow.com/questions/46689428/convert-np-array-of-type-float64-to-type-uint8-scaling-values/46689933
- https://stackoverflow.com/questions/59478962/how-to-convert-a-grayscale-image-to-heatmap-image-with-python-opencv
- https://www.youtube.com/watch?v=KOSS24P3_fY