ENPM673: Project 2

UID: 119197257

Problem 1:

Approach Followed:

1. Firstly, all the libraries needed are imported such as OpenCV, numpy and scipy.
2. The given video is read using Video_Capture function of OpenCV, the specific path of the video is given in this.
3. Empty lists are created for storing the roll, pitch, yaw, and translations received from all the frames.
4. As the process is highly computational, the frames are resized by half the size.
5. The frames are converted to gray scale.
6. For getting the white paper, applied threshold using inRange function.
7. To remove the noise applied morphological closing and opening operations.
8. Applied Canny edge detection algorithm to get the edges of the masked white paper.
9. Calculated the max height and width of the binary image received from the canny edge detection, this is obtained by. shape function.
10. In the hough space, d is the perpendicular distance of the line from the origin, the max d will be the hypotenuse of the rectangle of the detected binary image.
    The equation of line in hough space is given by,

    ```
    d = x cos(theta) + y sin(theta)
    ```

    where d is the perpendicular distance from the line to the origin and theta is the angle this perpendicular makes with the x axis.

    ```
    Diagonal_length = sqrt(width**2 + height**2)
    ```

11. The accumulator for hough transform is created with all the values being zeros initially, the dimensions of this accumulator are kept at `(2* Diagonal_length, 180)` which means that d will vary from, 0 to 2* Diagonal_length and theta will vary from 0 to 180, with step size 1.
12. The hough transform algorithm is iterated through all the edge pixels, when edge pixel is not equal zero in the binary image.
13. For every theta in the range 0 to 180, corresponding d value is calculated by using formula,
    `d = int(x*cos(theta) + y*sin(theta))`
14. The hough accumulator is updated by 1 at the index of this calculated d and theta value. This process if followed for all the edge pixels and for all the values of theta.
15. So, in the end of the process , we get the accumulator matrix where the strong edges have max value, the d , theta belonging to maximum index value will give us strong lines.
16. For getting 4 lines following approach is followed:
    a. Initialize empty array to store the detected lines.
    b. Loop 4 times to extract top 4 lines, initial a variable max frequency, which will be use to store maximum for votes for that particular line.
    c. Loop over the accumulator and find the indices that correspond to maximum values.
    d. For each max value check if its frequency is greater than the current max frequency, if it is then update it by the new value.
    e. In the end append the detected line, with (d, theta) in the lines list.

17. While updating the hough transform accumulator, it is updated by d + diagonal_length because calculated values of d can be negative, but my range for d is positive. To compensate this change, diagonal_length is again subtracted from the calculated d in the loop.
18. Next function is written to not to detect the same strong line again. The code sets the maximum value in the Hough transform to 0 to remove it from consideration for the next iteration. This prevents the same line from being detected multiple times.
19. Next loop is to make the neighbors of the maximum value zero. This is done because it is possible that adjacent lines will have similar voting. There is a more chance of happening this because my step size for d and theta is very fine. So similar voting can be given to adjacent or neighboring lines.
20. The if statement checks if the current d and theta pair falls within the bounds of hough transform accumulator. If it does, the value at theta d and theta is set to zero.
21. The loop follows the if statement runs for values of j and k between -10 and 10. This ensures that the pixels near the local maxima are set to zero.
22. The next step is to draw the detected 4 lines. First x0 and y0 are calculates as follows:
    ```
    X0 = d*cos(theta)
    Y0 = d*sin(theta)
    ```
23. The end points of this line are calculated by offsetting the line by 800 length. The lines are then plotted using cv.line function.
24. To find the intersections of these 4 lines this formula is used,
    ```
    X = (b2 - b1)/(m1-m2)
    Y = m1*X + b1
    ```
    Where m1, m2 and b1, b2 are the slopes and y intercepts of the lines.
25. The intersections are found for all the possible combinations of lines. If lines are parallel, there will be no intersections.
26. After getting the intersections of corners, the corners should be in sequence, for doing that the intersections are along y coordinate.
27. The world coordinates are define in an array as per given height and width of the paper.
28. To calculate the homography, following approach is followed,
    a. X1 and Y1 belong to the real world coordinates and x1_dash and y1_dash belong to image coordinates.
    b. Matrix A is created as follows,

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A} \quad \mathbf{h} \quad \mathbf{0}$$
$$\mathbf{2n \times 9} \quad \mathbf{9} \quad \mathbf{2n}$$

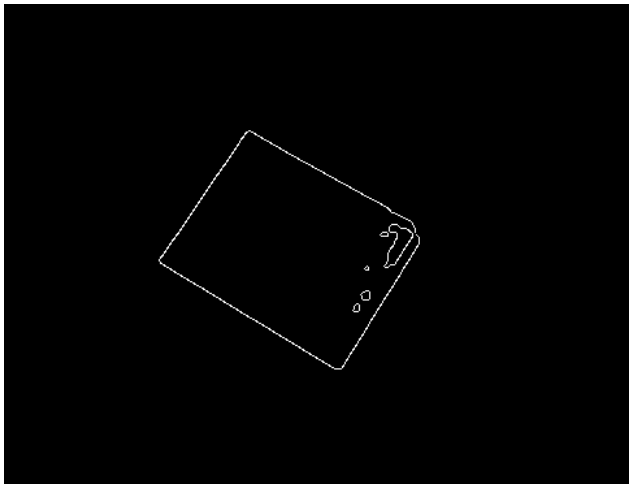
    c. The equation Ah = 0 is solved using SVD, the homography matrix is calculated by taking the last column of matrix vt, U and Vt are orthogonal matrices, and S is a diagonal matrix of singular values.
    d. The column needs to be reshaped by (3, 3). The homography matrix is normalized by dividing all the elements by the of third row and column.
29. The intrinsic matrix K is multiplied by the scaling factor because we have resized the frames.
30. Extrinsic matrix is calculated using following formula,

```
extrinsic_matrix = K_inv @ H_norm
```

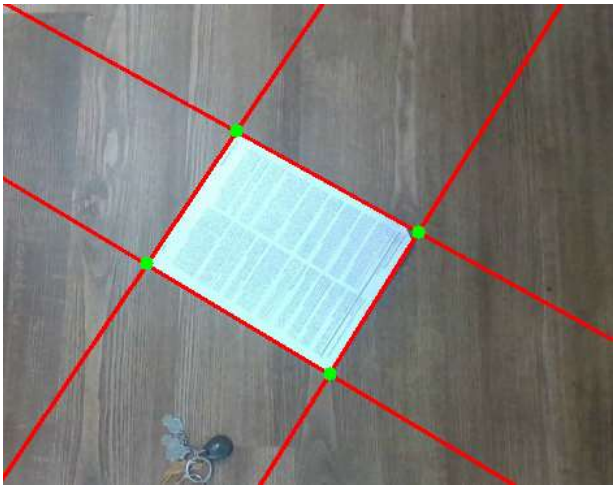
31. The rotation and translation are calculated using formula,
    Rotation = [r1 r2 (r1xr3)]
    Translation = lamba * h3
    Where r1 and r2 are the unit vectors of first two columns of extrinsic matrix and lamda is given by 1/norm(h1).
32. To get the roll, pitch and yaw following source is used - http://lavalle.pl/planning/node103.html
33. Finally, translation and roll, yaw and pitch are plotted against no of frames.

Results:

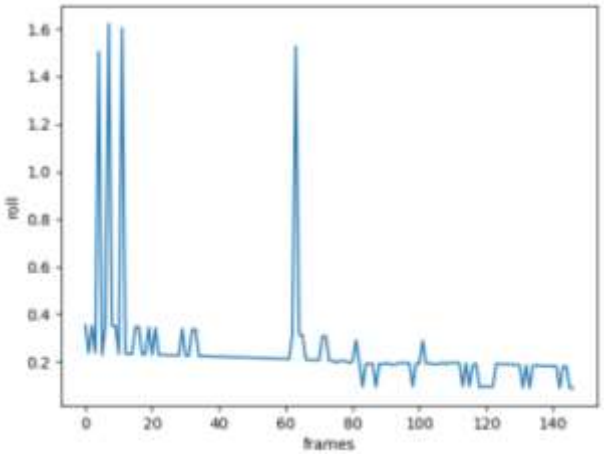
Edge detection using canny



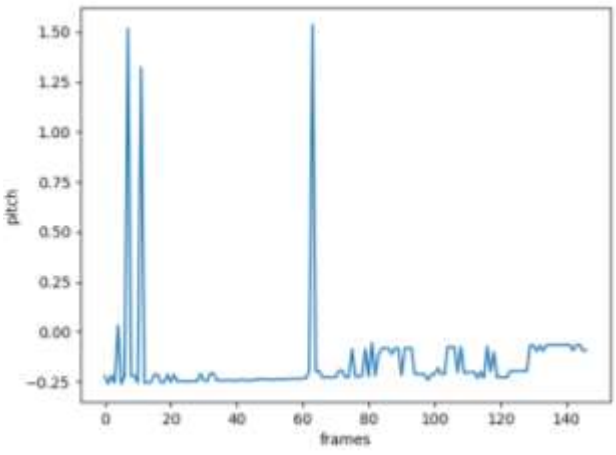
Corner detection using hough transform



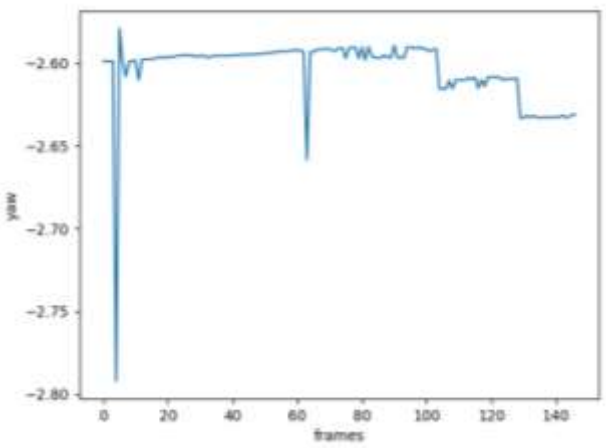
Roll values



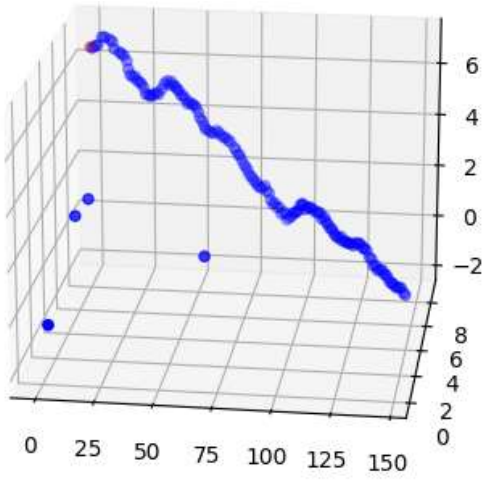
Pitch values



Yaw values



Translation



Problems faced:

1. The process was computationally expensive, the frames were shown very slowly. Resizing the frames helped to solve this problem.
2. The edges were detected correctly by choosing correct values for masking and canny edge detection by trial and error.
3. The dimension of array in hough transform had to be selected carefully, because sometimes the calculated value of d was out of index.
4. If used threshold, more than 4 lines were detected. To solve this error top 4 maxima were selected by iterating for 4 times throughout the accumulator matrix.
5. Closer lines were getting detected, setting the neighbors zero solved this issue.

Problem 2:

Approach Followed:

1. homography_initial function is first created to calculate the homography. Two arguments are given to this function i.e., keypoints1 and keypoints2. X1, y1, x1_dash, y1_dash are calculated from these key points. Matrix A is created using following formula.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A}_{2n \times 9} \qquad \mathbf{h}_{9} \qquad \mathbf{0}_{2n}$$

2. Eigenvalues and eigen vectors are calculated for this matrix. Homography is given by last column of the eigen vector belonging to the min eigen value. The vector is reshaped to 3 by 3 matrix to get the homography. This function returns the homography matrix.
3. As we cannot use perspective transform directly, a function is created which gives the same output. It converts transformed_points array to homogeneous coordinates by adding the third column of ones.
4. Next, create new homogeneous coordinates by multiplying src_hom with transpose of homog. Next function converts new homogeneous coordinates to Euclidean coordinates. This function returns the Euclidean coordinates.
5. The ransac function takes two arguments as input images, convert the images to gray scale using cvt.color.
6. A SIFT(Scale-invariant Feature Transfrom) feature detector in created using cv.SIFT_create().
7. SIFT keypoint and descriptor extraction is performed on gray images using sift.detectAndCompute(). The resulting keypoints and descriptors are stored in keypoints1 and descriptors1. Same for image 2.
8. A FLANN matcher is created using cv.FlannBasedMatcher_create(). This is used to find nearest neighbor matches between the two sets of descriptors using flann_matcher.knnMatch().
9. The ratio test is applied to the matches to find the best matches, For each match, only the nearest neighbor match whose distance is less than 0.25 times the distance to the second nearest neighbor is kept.
10. The pixels coordinate of the keypoints of the first image are extracted from the best matches and converted into numpy array and stored in src_pts, same is done for 2^{nd} image and stored in dst_pts.
11. Threshold and num of iterations for ransac are defined. For each iteration 4 points are randomly generated from src_pts and dst_pts, and new_homography is created using the function written earlier (homography_initial). It then applies perspective transform to get new set of transformed points.
12. It then computes the distance between transformed points and original points and counts the number of inliers. This is done using the threshold distance.
13. If the number of inliers is greater than the current maximum inliers, max inliers is updated as current inliers.
14. In this way the homography matrix is after iterating for 1000 iterations we will get best homography between two images.
15. The cv.warpPerspective function is used to warp the second image to the reference frame of first image using the best homography.
16. Then the region in final image corresponding to image 1 is replace by image 1.
17. The stitching can be done in two ways.
    a. From image 1 to image 4
    - First image 1 and image 2 are stitched together, but this gives some black region in the right side of the image.
    - This black region is removed by cropping the image, for that black region is found using white pixels.
    - Similarly, image 3 and image 4 are stitched and cropped.
    - Ransac is then applied to these two cropped images to get the final image.

    b. From image 4 to image 1
    c. First image 3 and image 4 are stitched together, then the generated image and image 2 are stitched together.
    d. After that, the generated image of these two is stitched with the image 1 to get the final image.
18. Only difference in ransac1 and ransac2 is the output size of the image in the warpperspective. Rest everything is the same in those two functions.
19. Finally, the final panorama is displayed.

Results:

Feature detection for image 1

Feature matching between image 1 and image 2

Feature matching between image12 and image 3

Feature matching between image 3 and image 4

Final panorama image by stitching from image 4 to image 1

Final panorama image by stitching from image 1 to image 4

Problems faced:

1. Issues to install the sift detector.
2. Issues to detect and match the features.
3. The homography calculated using SVD was not giving the correct results because of presence of out liers.
4. This issue was solved by implementing ransac, which gave the best homography.
5. The sequence of stitching matters a lot. Stitching forward was giving black regions which were removed by cropping the image.
6. The perspective of image is still from the right-hand side, it should be from the front.