Perception for Autonomous Robots

Project #2

Wednesday, March 8, 2023

Student(s):
Arshad Shaik
UID: 118438832

Instructor:
Dr. Samer Charifa

TA(s):
Mr. Madhu Narra Chittibabu
Mr. Arunava Basu

Grader:
Mr. Kumara Ritvik Oruganti

Semester:
Spring 2023

Course Code:
ENPM673

Table of Contents

1 Problem 1:

1.1 Pipeline

The solution implementation for this problem is given as below:

a) Own function to find homography, given the 4 co-ordinates of the world-axes and 4 co-ordinates of camera projective matrix.
b) Own function to find lines in an image.
c) Read a given video.
d) For each image frame in a video, do the following:
    a. Convert a BGR image into HSV image.
    b. Create a masked image, with object of interest (white paper)
    c. Apply erosion and dilation to remove noise.
    d. Process this gray image.
    e. Detect the edge using Canny edge detection.
    f. Detect lines using own function.
    g. For each line, do the following:
        i. compute the slope and y-intercept.
        ii. remove extraneous lines by using similar slopes and intercepts.
        iii. if the number of lines is lesser than 4, then take the previous frame data.
        iv. (Atleast 4 points are required to compute homography)
        v. ensure that atleast 4 lines are detected in each frame.
    h. Compute corner points of the lines and designate in an image.
e) Pseudo code for computing intersection points:

```
def intersection(m1: float, b1: float, m2: float, b2: float):
    # Consider y to be equal and solve for x
    # Solve:
    #   m1 * x + b1 = m2 * x + b2
    x = (b2 - b1) / (m1 - m2)
    # Use the value of x to calculate y
    y = m1 * x + b1

    return int(round(x)), int(round(y))
```

f) display the processed image.
g) Compute the homography using the computed corner points when there are atleast 4 corner points.
h) Decompose the H-matrix into corresponding rotation and translation matrices.
i) Compute the camera pose estimation, using r1, r2, r3 as below:

Assume all points lie in one plane with Z=0:

$$\boldsymbol{X} = (X, Y, 0, 1)$$

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{P}\boldsymbol{X} \\ &= \boldsymbol{K}[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{r}_3\boldsymbol{t}]\begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \\ &= \boldsymbol{K}[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}]\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \\ &= \boldsymbol{H}\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \end{aligned}$$

$$\boldsymbol{H} = \lambda \boldsymbol{K}[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}]$$

$$\boldsymbol{K}^{-1}\boldsymbol{H} = \lambda[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}]$$

–r_1 and r_2 are unit vectors ⇨ find lambda

–Use this to compute t

–Rotation matrices are orthogonal ⇨ find r3

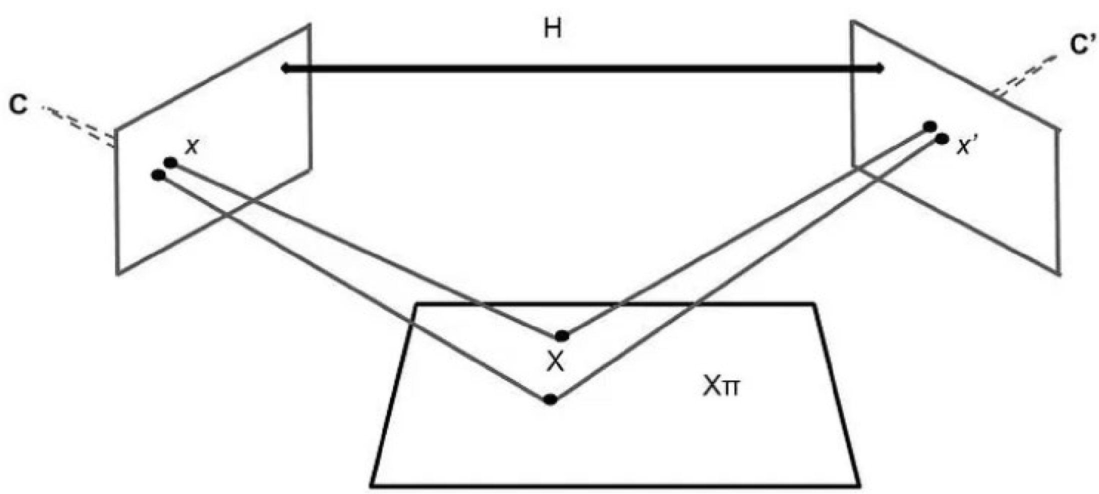
$$\boldsymbol{P} = \boldsymbol{K}\left[\begin{array}{cccc} \boldsymbol{r}_1 & \boldsymbol{r}_2 & (\boldsymbol{r}_1 \times \boldsymbol{r}_2) & \boldsymbol{t} \end{array}\right]$$

a.

j) Repeat for all images in video to accumulate the camera pose estimations and plot them.

Algorithm for Homography Computation:

Homography describes the projective geometry of two cameras and a world plane. In simple terms, homography maps images of points which lie on a world plane from one camera view to another. It is a projective relationship since it depends only on the intersection of planes with lines. As shown in the figure below:



Planar Homography

Lets assume $x = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$ and $x' = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ in homogeneous coordinates

and $H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$

$$c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

By eliminating c we can formulate the above equation in the form

$$Ah = 0$$

where $A = \begin{pmatrix} -x & -y & -1 & 0 & 0 & 0 & ux & uy & u \\ 0 & 0 & 0 & -x & -y & -1 & vx & vy & v \end{pmatrix}$

and $h = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{pmatrix}^T$

$$\begin{pmatrix} -x1 & -y1 & -1 & 0 & 0 & 0 & x1*xp1 & y1*xp1 & xp1 \\ 0 & 0 & 0 & -x1 & -y1 & -1 & x1*yp1 & y1*yp1 & yp1 \\ -x2 & -y2 & -1 & 0 & 0 & 0 & x2*xp2 & y2*xp2 & xp2 \\ 0 & 0 & 0 & -x2 & -y2 & -1 & x2*yp2 & y2*yp2 & yp2 \\ -x3 & -y3 & -1 & 0 & 0 & 0 & x3*xp3 & y3*xp3 & xp3 \\ 0 & 0 & 0 & -x3 & -y3 & -1 & x3*yp3 & y3*yp3 & yp3 \\ -x4 & -y4 & -1 & 0 & 0 & 0 & x4*xp4 & y4*xp4 & xp4 \\ 0 & 0 & 0 & -x4 & -y4 & -1 & x4*yp4 & y4*yp4 & yp4 \end{pmatrix} * H = 0$$

$H^*\underset{H}{\text{argmin}} = \|AH\|^2$ subject to $\|H\| = 1$

We can't use least square since it's a homogeneous linear equations (the other side of equation is 0 therfore we can't just multyly it by the psudo inverse). To solve this problem we use **Singular-value Decomposition** (SVD).

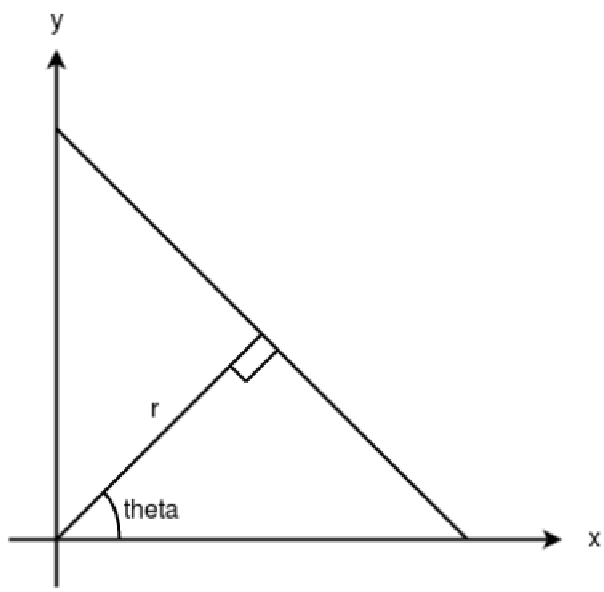
For any given matrix $A_{M\times N}$

$$\underbrace{\mathbf{A}}_{M\times N} = \underbrace{\mathbf{U}}_{M\times M} \times \underbrace{\mathbf{\Sigma}}_{M\times N} \times \underbrace{\mathbf{V}^{\mathrm{T}}}_{N\times N}$$

U is an $M \times M$ matrix with orthogonal matrix (columns are eigen vectors of A).
Σ is an $M \times N$ matrix with non-negative entries, termed the singular values (diagonal entries are eigen values of A).
V is an $N \times N$ orthogonal matrix.

<u>Algorithm to compute the HoughLine Method:</u>

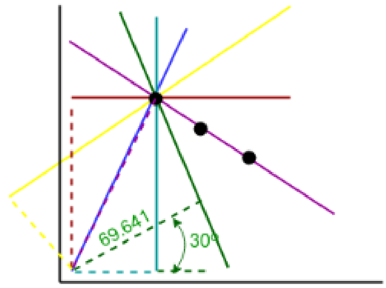
A line can be represented as y = mx + c or in parametric form, as r = xcosθ + ysinθ where r is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise (That direction varies on how you represent the coordinate system. This representation is used in

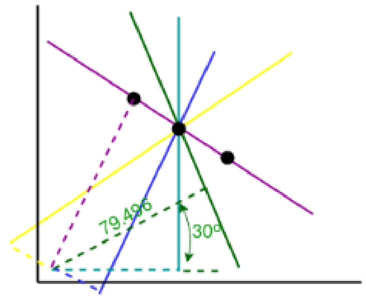


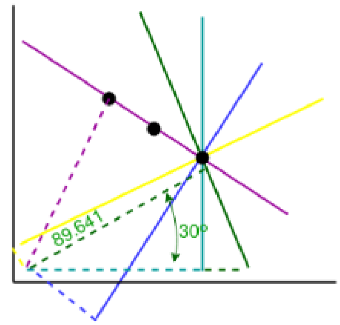
OpenCV).

Working of Houghline method:

a) First it creates a 2D array or accumulator (to hold values of two parameters) and it is set to zero initially.
b) Let rows denote the r and columns denote the (θ)theta.
c) Size of array depends on the accuracy you need. Suppose you want the accuracy of angles to be 1 degree, you need 180 columns(Maximum degree for a straight line is 180).
d) For r, the maximum distance possible is the diagonal length of the image. So taking one pixel accuracy, number of rows can be diagonal length of the image.







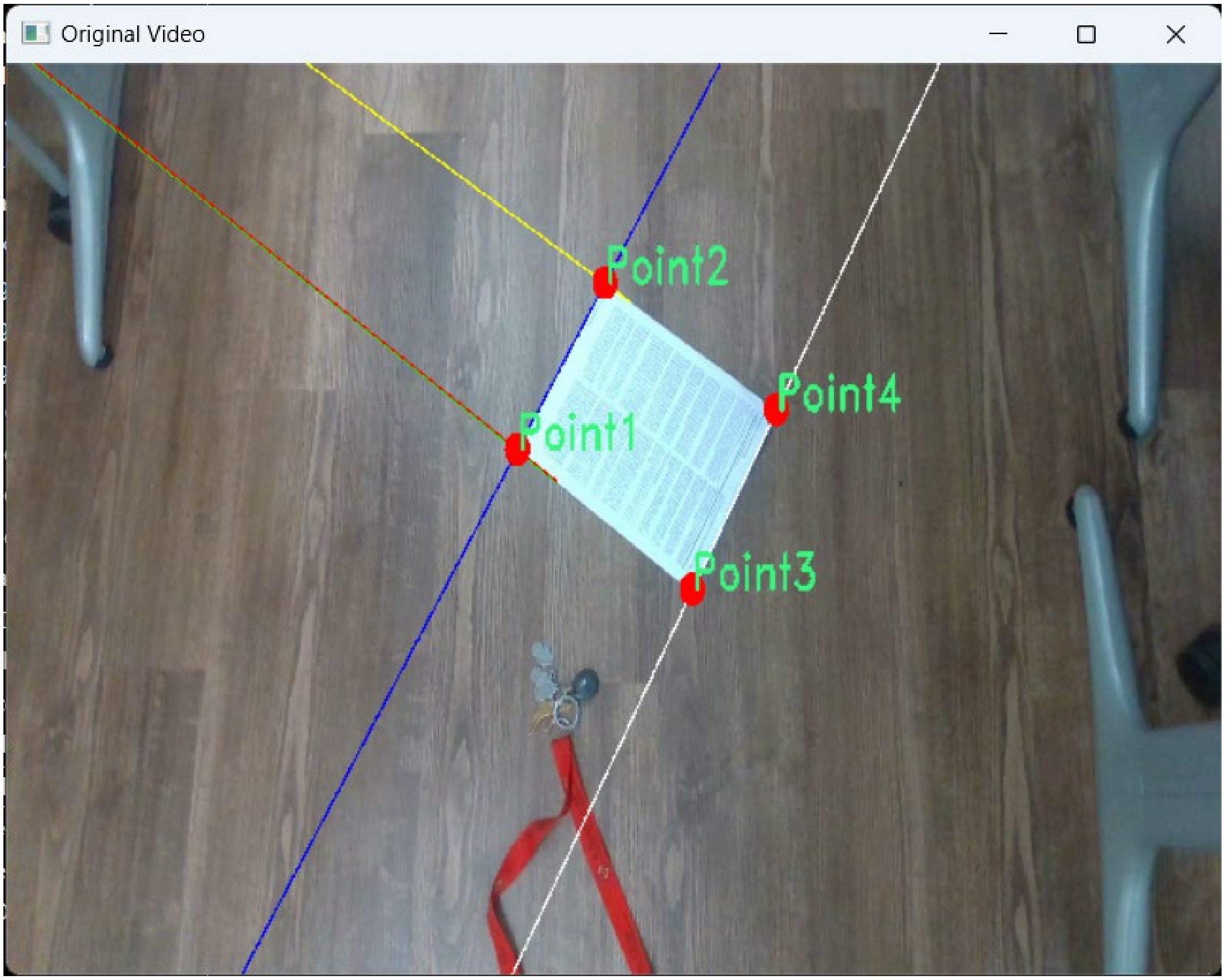
Angle	Dist.
0	40
30	69.6
60	81.2
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
120	23.4
150	-19.5

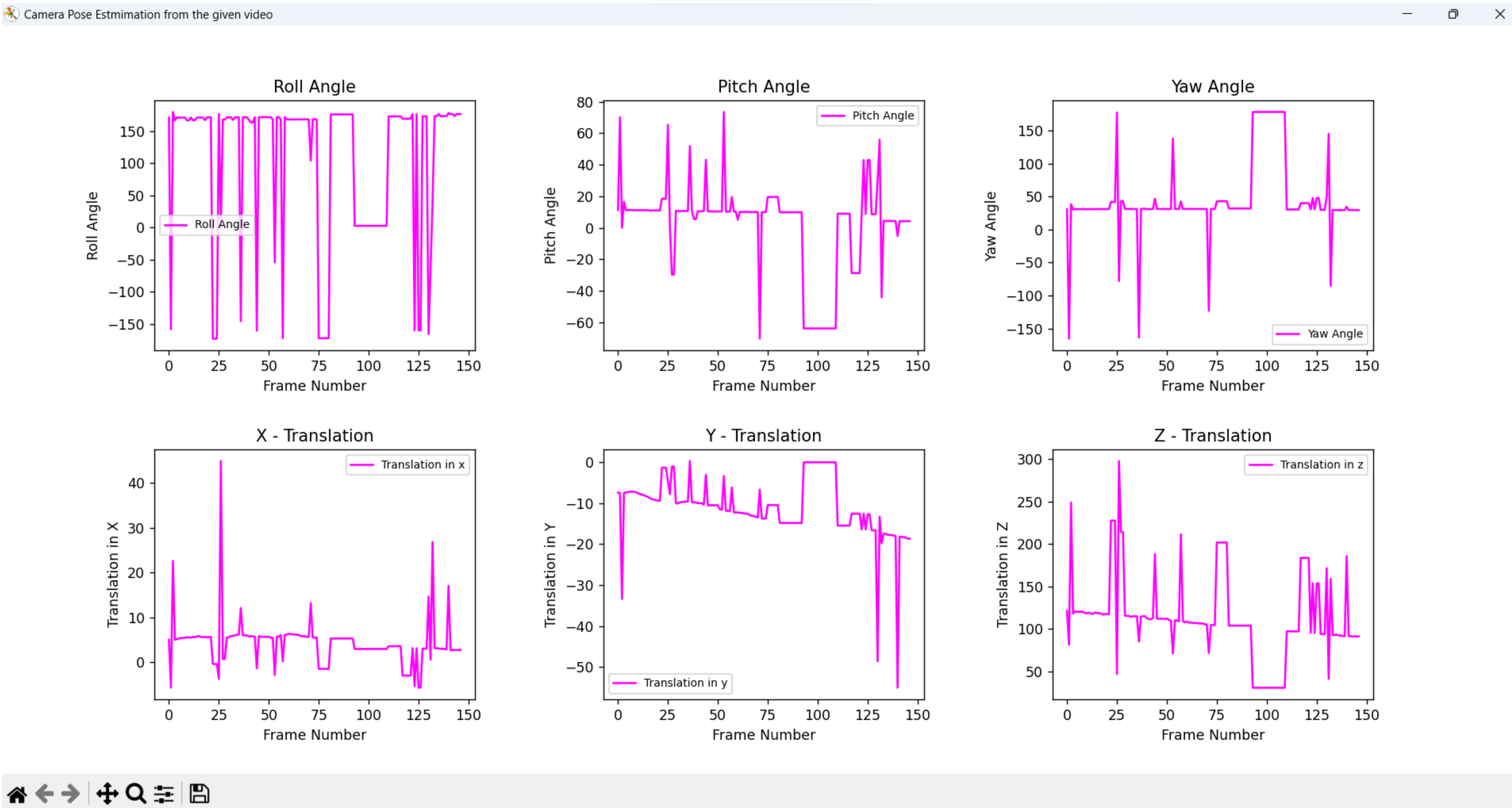
Angle	Dist.
0	74.6
30	89.6
60	80.6
120	6.0
150	-39.6

1.2 Results

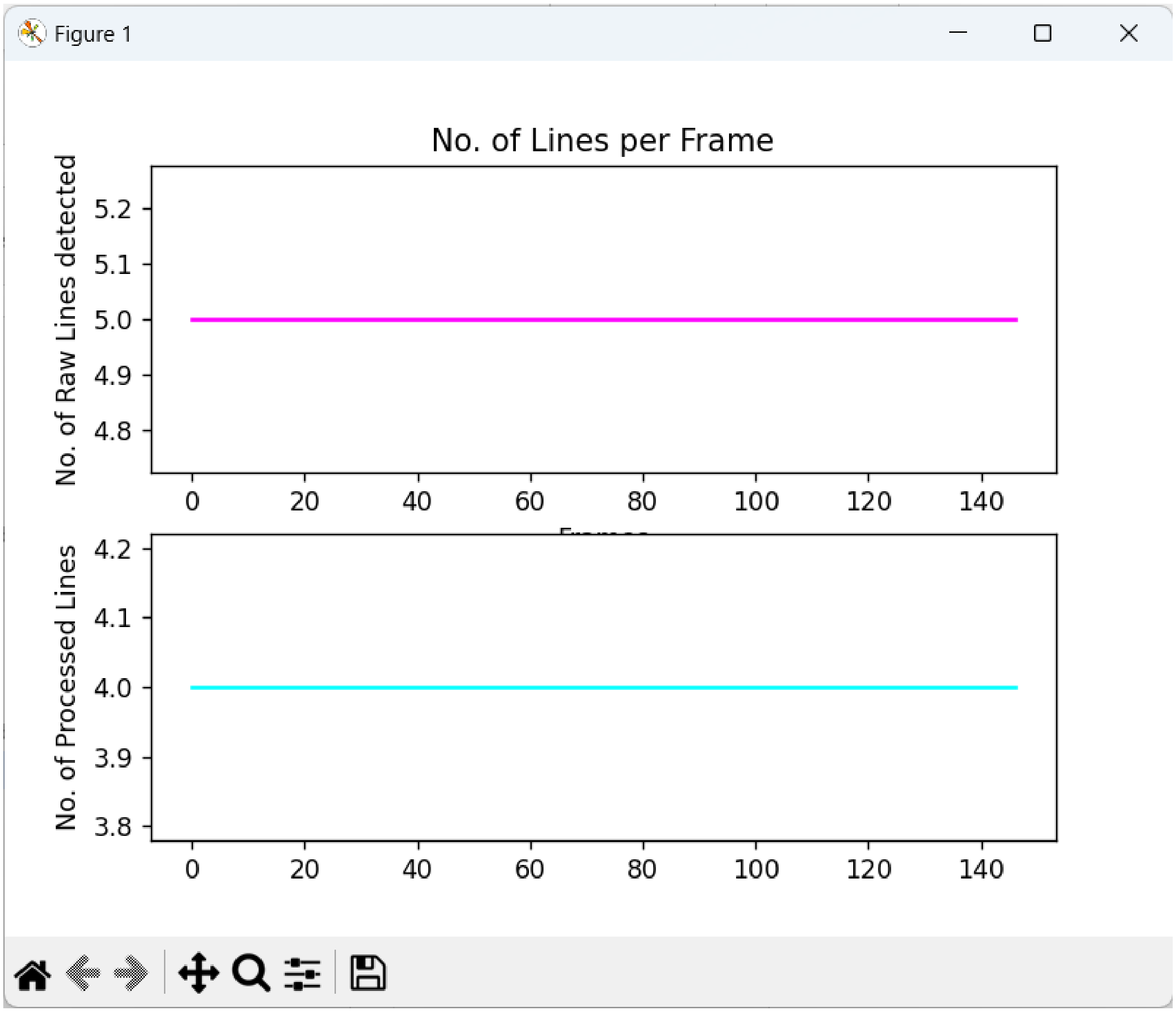
a) Sample Image from the Edge and corner detection Video Output:



b) <u>Camera Pose Estimation Graphs:</u>



c) Number of Lines detected and processed Lines:



1.3 Code:

The code file – "Problem1.py" - is included as part of the project submission folder.

1.4 README and Problems Encountered:

The README file – "README.md" is included as part of the project submission folder.

The following problems are encountered while solving for the above problem.

a. Consistency of lines for the entire duration of the video
b. Own Hough lines function is relatively very slower than the HoughLines library
c. Fine-tuning of filtering of image in the aspects of color filtering, white and black noise filtering
d. Converting a cartesian co-ordinates to homogenous co-ordinates

e. Fluctuations in the camera pose angles due to the noise in the continuous processing of the image frames in the video.

2 Problem 2:

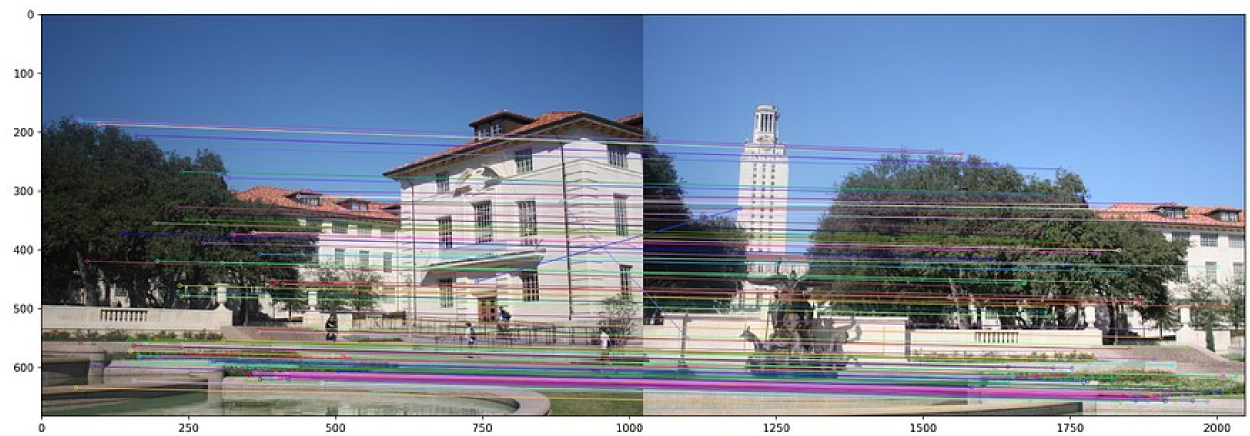
2.1 Pipeline

The following solution is implemented for panoramic image creation.

- Key point detection
- Local invariant descriptors (SIFT, SURF, etc.)
- Feature matching
- Homography estimation using RANSAC
- Perspective warping

Steps implemented:

1. Read all the images and store it in array.
2. Detect all the features using SIFT.
3. Match all the features between the pair of images using knnMatch
4. Use RANSAC method to remove any outliers.



5. Compute the homography matrix for the inliers.
6. Stitch the pair image using the 'warpPerspective' and repeat until all the four images are stitched.

Sample Panoramic Images:

Input image pair

Panoramic Image

2.2 Results

Stiched Image 1+2+3+4

2.3 Code:

The code file – "Problem2.py" - is included as part of the project submission folder.

2.4 README and Problems Encountered:

The README file – "README.md" is included as part of the project submission folder.

The following problems are encountered while solving for the above problem.

a. AttributeError: 'module' object has no attribute 'xfeatures2d' [Python/OpenCV 2.4]

This error was solved by replacing the above function as below:

```
sift = cv2.xfeatures2d.SIFT_create()
```

to

```
sift = cv2.SIFT_create()
```

b. Removing outliers through RANSAC. Even if the built-in library is not so efficient.

3 References:

1. Youtube:

Warping and Blending Images | Image Stitching

https://www.youtube.com/watch?v=D9rAOAL12SY&list=PL2zRqk16wsdp8KbDfHKvPYNGF2L-zQASc&index=6

Stackoverflow /

2. How to stitch two images using homography matrix in OpenCv?

https://stackoverflow.com/questions/61146241/how-to-stitch-two-images-using-homography-matrix-in-opencv

3. Understanding Hough Transform With Python

https://alyssaq.github.io/2014/understanding-hough-transform/

4. Process the output of cv2.HoughLines

https://arccoder.medium.com/process-the-output-of-cv2-houghlines-f43c7546deae

5. How to detect different types of arrows in image?

https://stackoverflow.com/questions/66718462/how-to-detect-different-types-of-arrows-in-image

6. Line detection in python with OpenCV | Houghline method
   https://www.geeksforgeeks.org/line-detection-python-opencv-houghline-method/

7. "Multiview Geometry in Computer Vision" by Richard Hartley.
   https://medium.com/all-things-about-robotics-and-computer-vision/homography-and-how-to-calculate-it-8abf3a13ddc5

8. https://www.askpython.com/python-modules/opencv-puttext

9. https://stackoverflow.com/questions/6618515/sorting-list-according-to-corresponding-values-from-a-parallel-list

10. https://www.geeksforgeeks.org/python-ways-to-sort-a-zipped-list-by-values/