Midterm_Questions

March 16, 2024

1 ENPM673 Perception for Autonomous Robots (Spring 2024) - Midterm Exam

1.0.1 Deadline: 16th March 7AM

Rohan Maan, Jay Prajapati, Samer Charifa, Tommy Chang

1.1 Submission guidelines

- The midterm exam is to be done and submitted individually.
- Write the solution to **all the problems in one Google Colab File. (Handwritten answers will not be allowed)**
- Your submission on ELMS/Canvas should be the following:
  a. Google Colab (.ipynb) file
  b. Google Colab (.pdf) file (Convert the same .ipynb file to a .pdf file) following the naming convention YourDirectoryID_midterm.
- Points will be deducted if you don't follow the submission guidelines.
- Submit all files in one attempt.
- Provide detailed explanations for each step using text cells in Google Colab.
- Ensure the code is well-formatted for readability.
- Comment on each section (preferably every 2 lines) of the code to explain its purpose and functionality.
- Include relevant output images within the text cells where required.
- Note that the use of ChatGPT (or any other generative AI websites) is not allowed.
- You are free to use any in-built OpenCv functions **except for PCA (In Question-3)**.

1.2 How to use this file

- Links to all the input data have been given in the text cells.
- Add this .ipynb file to your working directory on google drive.
- Add the given input files to your working directory on google drive.
- Make sure you are using Google Colab and not any other IDE.
- Placeholders to write the answers have been given in this file.
- Do not change the structure of the questions.
- You may add extra code cells or text cells for any particular question.
- Make sure that the extra cell is added under the right question.

1.3 Enter Student Details Here

Name	Tathya Bhatt
UID	120340246
Email	tathyab@umd.edu

2 Set path to the working directory

[2]:
```
from google.colab import drive
drive.mount('/content/drive/', force_remount=True)
```

```
Mounted at /content/drive/
```

[3]:
```
path_to_folder = "ENPM673/Midterm/"
%cd /content/drive/My\ Drive/{path_to_folder}
```

```
/content/drive/My Drive/ENPM673/Midterm
```

3 Import Libraries

[4]:
```
# Import all your libraries here ....

import cv2
import numpy as np
import random
import matplotlib.pyplot as plt
import csv
import plotly.graph_objects as go
import pandas as pd
from google.colab.patches import cv2_imshow
```

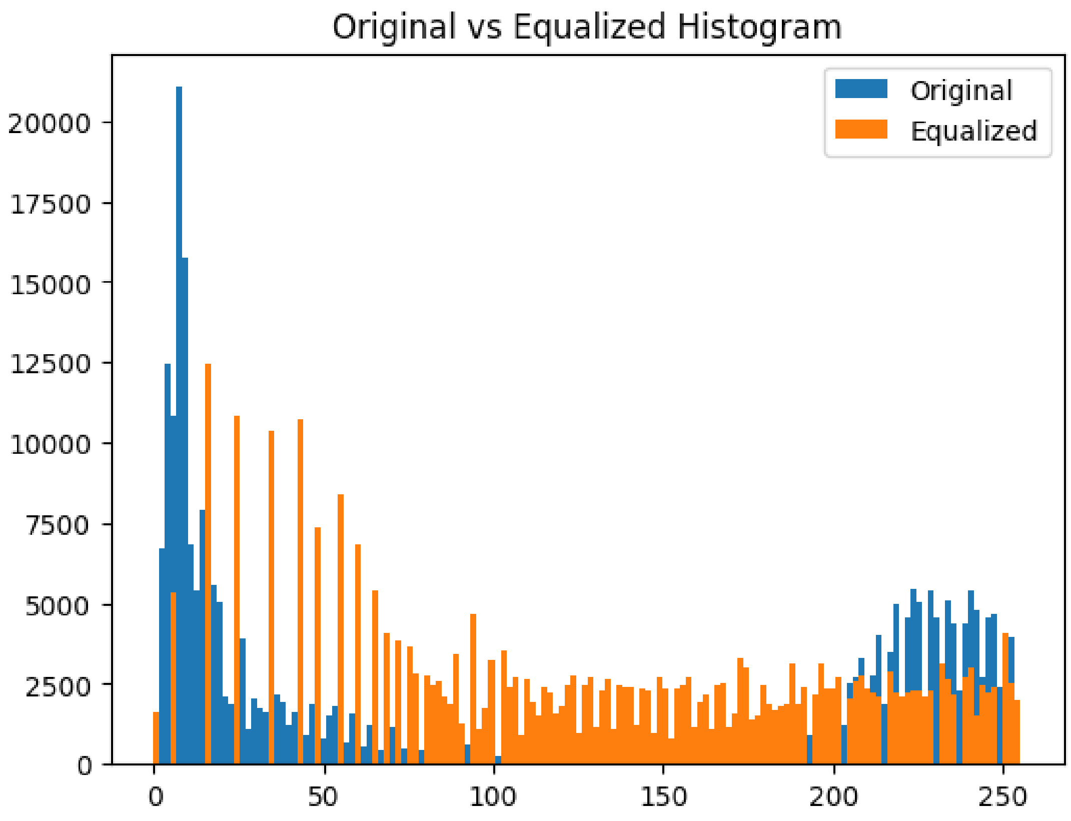
4 Problem 1 (20 Points)

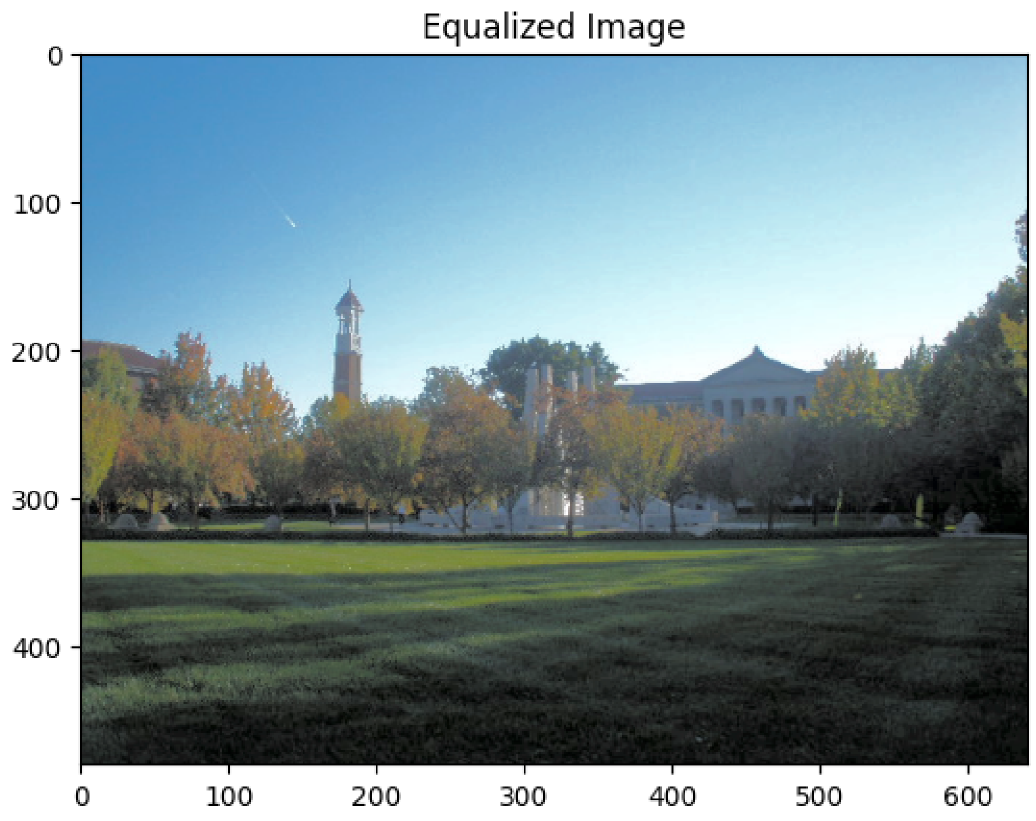
Link to input image: https://drive.google.com/file/d/1X0xir8CrOoKsNXxvZnLVAJWKztpMuUhV/view?usp=dri

Part 1: Write code and describe the steps to perform histogram equalization of the given color image. Hint: First convert to a color space that separates the intensity from the color channels (eg. LAB or LUV color spaces work the best). Your result should look like the below: Link to the output of part-1: https://drive.google.com/file/d/1nIhoE0PyCdFE3LfAwGcUqUt_GaQ2NoG-/view?usp=sharing

[5]:
```
######## PART1 ########



# Loading the input image
input_img = cv2.imread("PA120272.JPG")

```

```
# Converting a BGR Image to LAB Color Space
lab_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2LAB)

# Separating into Luminosity, Red/Green Value, Blue/Yellow Value
l, a, b = cv2.split(lab_img)

# Applying histogram equalization
hist_eq = cv2.equalizeHist(l)

# Updating the LAB Channel with the updated L Channel Values
new_lab1 = cv2.merge((hist_eq, a, b))

# Output Image with regular histogram equalization
output_img1 = cv2.cvtColor(new_lab1, cv2.COLOR_LAB2BGR)

# # To get the similar output, we Contrast Adaptive Histogram Equalization can␣
 ↪be used
# clahe = cv2.createCLAHE(clipLimit=3,  tileGridSize=(4,4))
# # Applying CLAHE to the previous equalized L channel
# cla_img = clahe.apply(hist_eq)

# # Updating with the improved L channel values
# new_lab2 = cv2.merge((cla_img, a,b))


# PLotting the histogram of intensity values
plt.hist(l.flat, bins=150, range=(0,255), label='Original')
plt.hist(hist_eq.flat, bins=150, range=(0,255), label='Equalized')
plt.title("Original vs Equalized Histogram")

plt.legend()
plt.show()

plt_img = cv2.cvtColor(output_img1, cv2.COLOR_BGR2RGB)
plt.imshow(plt_img)
plt.title('Equalized Image')
plt.show()

# cv2_imshow(output_img1)
```

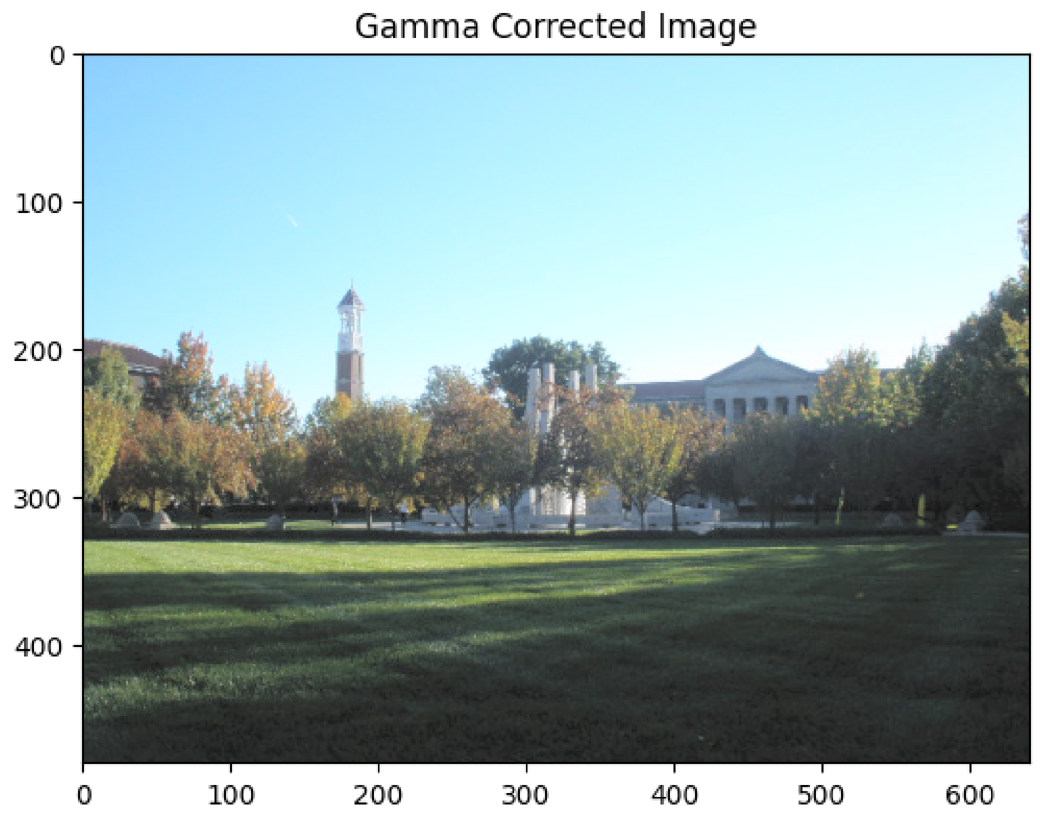

Original vs Equalized Histogram
Original
Equalized
20000
17500
15000
12500
10000
7500
5000
2500
0
0
50
100
150
200
250



Part 2: Another way to enhance a dark image is to perform gamma correction. For each intensity pixel, replace it with the new intensity = 255*((intensity/255)^(1/2.2)). Write code to perform gamma correction. Your result should look like below: Link to the output of part-2: https://drive.google.com/file/d/1rgo7Kl8qK7Byh5QZsIb4CrfdBzY51Aco/view?usp=sharing

[6]:
```
####### PART2 #########
input_img_ = cv2.imread("PA120272.JPG")

# Converting BGR to LAB
lab_img_ = cv2.cvtColor(input_img_, cv2.COLOR_BGR2LAB)

# Extracting the Luminosity Channel
l_ = lab_img_[:,:,0]

# Performing Gamma Correction
l_new = (255*(np.power(l_/255, (1/2.2)))).astype(np.uint8)

# Updating the Luminosity Value
new_lab_gam = cv2.merge((l_new, a, b))
```

```python
# Converting to BGR and then RGB
output_gam = cv2.cvtColor(new_lab_gam, cv2.COLOR_LAB2BGR)
plt_img_gam = cv2.cvtColor(output_gam, cv2.COLOR_BGR2RGB)
plt.imshow(plt_img_gam)
plt.title('Gamma Corrected Image')
plt.show()
```



5 Problem 2 (15 Points)

1. For an image that is n by n and assuming that we have a random kernel call it H, where H is m by m matrix, what is the big O notation when performing convolution (how many multiplications are needed, in terms of n)?

2. Describe the meaning of "separable" kernel? Redo the previous question knowing that the H kernel is separable?

3. Apply the principle of separability to the following kernel to separate it into two kernels. Kernel

5.0.1 Problem-2 (1)

ANS:

With a standard mxm kernel size convolved over an nxn image, each pixel will need $ M^2 $ operations and for the final image it would become $ N^2 M^2 $ Operations.

Thus

$$O = M^2 N^2$$

5.0.2 Problem-2 (2)

ANS:

A 2D Kernel is called separable kernel if it can be expressed as a product of a column vector and row vector. This is basically similar to performing two 1D convolutions as column kernel and row kernels.

- If the image having (nxn) pixels is performed with a separable kernel of (mxm), the big-O notation describing the cost would be

$$O = 2MN^2$$

5.0.3 Problem-2 (3)

ANS:

After applying the principle of separabilit, we get these two 1D kernels.

$$A = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix} = \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}$$

6 Problem 3 (20 Points)

Link to the csv file: https://drive.google.com/file/d/1LGG32bpU0sTIp-lOxOXCZJELGoZqVaZk/view?usp=sharing

Given x, y, z coordinates of n number of data points(problem3_points.csv). Consider matrix A where matrix A represents the coordinates of the n points shown in the figure below. Let's say we want to fit these data points onto an ellipse.

1. Describe the steps to find the major and minor axis of this ellipse along with their prospective length.

2. Similarly, given the data that represents a flat ground in front of the robot in 3D, try to extract the equation of the surface that describe the ground.

6.0.1 Problem-3 (1)

To extract the major and minor axis,

1. Loading the CSV file containing the coordinates and forming a matrix out of it.
2. Next, we can calculate mean of x and y coordinates to further normalize the points by subtracting the mean with every point in x and similarly in y coordinate as well.
3. Computing the covariance matrix around that subtracted points
4. Finding eigenvalues and eigenvectors of the covariance matrix
5. By analysing those values, we can find the strongest eigen values to be a major axis of the ellipse and eigenvector denotes the direction of the major axis

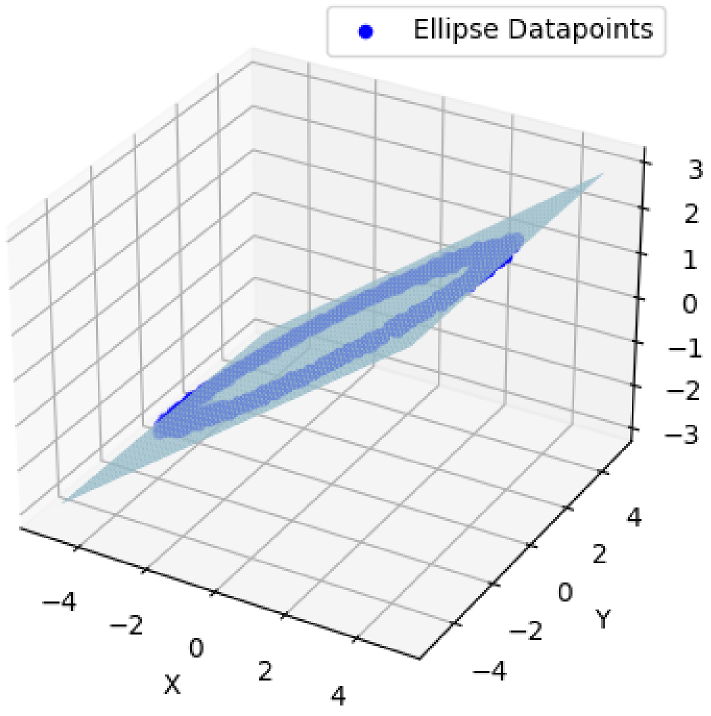
The second strongest value corresponds to the length of minor axis

The weakest eigen value and eigen vector denotes the axis perpendicular to the plane of ellipse

Problem-3 (2)

[7]:
```
# Importing pandas to help read the files
import pandas as pd

# Reading it as a pandas dataframe
df = pd.read_csv("problem3_points.csv")
# df[0:5]

# Extracting x,y,z values
x_points = df['X'].values
y_points = df['Y'].values
z_points = df['Z'].values

# The a_matrix to plot the ellipse datapoints and just to visualize and not␣
 ↪used in any calculation
a_matrix = np.column_stack((x_points, y_points, z_points))

# The equation of the plane follows ax + by + c = z and now we construct Ax=Z␣
 ↪equation to solve the coefficients
a_mat = np.column_stack((x_points, y_points, np.ones(len(x_points))))
z = z_points

# We solve for 'x' vector which is the coefficient vector using Least Squares
x_mat = np.linalg.inv(a_mat.T @ a_mat ) @ a_mat.T @ z

# Here we find the unknown co-efficients of the equation
a = x_mat[0]
b = x_mat[1]
c = x_mat[2]

#  Generating points for fitting the plane
```

```
x = np.linspace(-5, 5, 80)
y = np.linspace(-5, 5, 80)

# Generating a 2D grid based on the coordinates
xx, yy = np.meshgrid(x,y)

z_ = a*xx + b*yy + c

# Scateer PLot of the Ellipsoid Points
fig = plt.figure()
axis = fig.add_subplot(111, projection='3d')
axis.scatter(a_matrix[:, 0], a_matrix[:, 1], a_matrix[:, 2], c='blue',␣
 ↪label='Ellipse Datapoints')

# Generated PLane Plot
axis.plot_surface(xx, yy, z_, color='lightblue')
axis.set_xlabel('X')
axis.set_ylabel('Y')
axis.set_zlabel('Z')
axis.legend()

plt.show()

# Finding the equation of the plane
print("Equation of the Ellipse is: {}x + {}y + {}z = 0.0".format(np.
 ↪round(a,3),np.round(b,3),np.round(c,3)))
```



```
Equation of the Ellipse is: 0.577x + 0.0y + -0.0z = 0.0
```

7 Problem 4 (30 Points)

Link to the input image: https://drive.google.com/file/d/1VeZyrPIwyg7sqi_I6N5zBUJP9UHkyDEb/view?usp=sh

Given the photo of a train track, describe the process to find the mean distance in pixels between the train tracks for every row of the image where train tracks appear. This question contains two parts:

1. Design Pipeline and explain the requirement of each step

2. Write code to implement the pipeline

Problem-4 (1) The following pipeline has been implemented successfully

1. Performing Perspective Transformation

- Basically used for getting an overall bird's eye view to better visualize the complex road networks like rail tracks

2. Gaussian Filter

- To make sure we work with noise reduced image and making an image smooth and get the better results while detecting edge.

3. Edge Detection

- Performing CANNY Edge Detection to detect the binary edges of the rail tracks

4. Hough Transform

- Finding hough lines using the edges detected previosuly to get the mose suitable line required

5. Finding the Distance between two lines

- Iterating through the image pixels to find the particular line pixel and then calculating the mean of the distances between adjacent blue pixels in same row

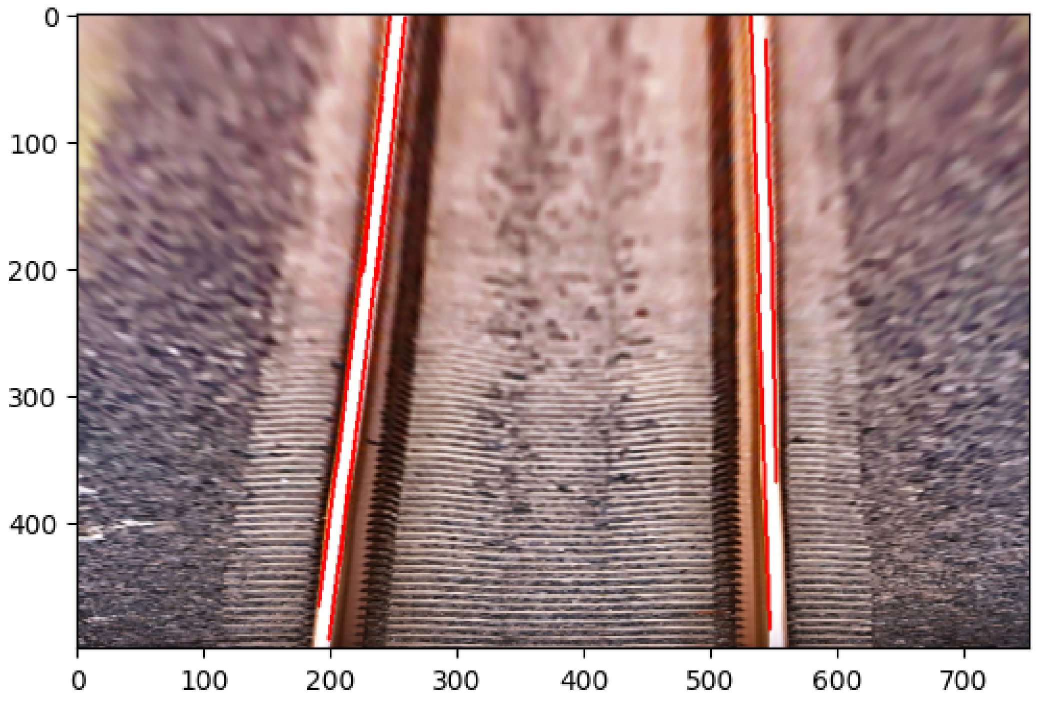
Problem-4 (2)

[8]:
```python
img = cv2.imread("train_track.jpg")
w = img.shape[1]
h = img.shape[0]

#### Defining a function which converts gray pixel values to binary values
def convert2bin(frame):
  binary_mask = (frame > 220) * 255
  return binary_mask.astype(np.uint8)

'''
source : Matrix containing the coordinates of four points which is to be␣
 ↪transformed to be top view.
The below points follows, top-left, top-right, bottom-right, bottom-left this␣
 ↪order
'''
source = np.float32([[1450, 1060],[1560, 1060],[2090, 2000], [950, 2000]])

'''
dest : Matrix containing the coordinates of four points of destination image␣
 ↪dimention.
The below points follows, top-left, top-right, bottom-right, bottom-left this␣
 ↪order
'''
dest = np.float32([[800, 660],[2200, 660],[2200, 2000], [800, 2000]])

# This generates the transformation matrix between source and destination images
trans_matrix = cv2.getPerspectiveTransform(source,dest)
trans_img = cv2.warpPerspective(img, trans_matrix,(img.shape[1], img.shape[0]))

# Resizing it to be more accessible
img_ = cv2.resize(trans_img, (int(w/4), int(h/4)) )

# Grayscaling the image to convert it to a single channel
gray = cv2.cvtColor(img_, cv2.COLOR_RGB2GRAY)
```

```python
# Applying gaussian blur to the grayscale image
gauss = cv2.GaussianBlur(gray, (5,5), 2)

# Converting it to a binary image which will aid in finding edges
bw_img = convert2bin(gauss)

# Using Canny Edge Detector with the following threshold
edges = cv2.Canny(bw_img, 190, 255)

# Calculating hough lines from the image containing edges
lines = cv2.HoughLinesP(edges, 2, np.pi/180, 90, minLineLength=100,␣
 ↪maxLineGap=60)

# Drawing those lines
for line in lines:
    x_1h, y_1h, x_2h, y_2h = line[0]

    # if np.sqrt((x_1h - x_2h)**2 + (y_1h - y_2h)**2 ) > 200:
    cv2.line(img_, (x_1h, y_1h), (x_2h, y_2h),(0, 0, 255), 2)


# Making the copy of the image
mask_img = img_

# Line mask which converts the needed pixels to white and rest to black
line_mask = cv2.inRange(mask_img, (0,0,200), (0,0,255))
# Retains the pixel value of the ones which are masked in white
retained = cv2.bitwise_and(mask_img,mask_img, mask=line_mask)

mean_distances = []
for row in range(retained.shape[0]):
   if np.any(retained[row,:] == 255):

      # If a blue pixel is found, corresponding other blue pixels in same row␣
 ↪aare found using np.where
      columns = np.where(retained[row, :] == 255)

      # Calculates the distance between adjacent column value of blue pixel in␣
 ↪same row
      distances = np.diff(columns)

      # Applying advanced indexing where all distance greater than 100 are kept␣
 ↪valid and rest are discarded
      filtered_distances = distances[distances > 100]

      if len(filtered_distances) > 0:
        # To filter out empty list and then adding the distances to the␣
↪mean_distance
        mean_distances.append(filtered_distances)

# Calculating the mean of all distances from the list
mean_d = np.mean(mean_distances)
print("Average Distance between two tracks is:", mean_d)


img_plt = cv2.cvtColor(img_, cv2.COLOR_BGR2RGB)
# # cv2_imshow(img_)
plt.imshow(img_plt)
```

```
Average Distance between two tracks is: 306.4579055441478
```

[8]:
```
<matplotlib.image.AxesImage at 0x7837b75c3e20>
```



8 Problem 5 (15 Points)

Let's say you want to design a system that will create a top view of the surrounding around the car, see the image below for referene. Do the following,

(1) Describe and point the location of the sensors on the car in the given picture. Also, show the field of view of the sensors.

(2) Write the pipeline that will take input from these sensors and output a top-view image similar to the one shown below.

Note: Explain the answer in detail and provide necessary justifications for the steps you use in the pipeline.

Problem 5 (Part-1) There can be 6 camera for right side view, left side view, rear view, and front angle views to detect the blind spots and a LiDAR can be useful as well if the vehicle is autonomously navigating. Here is the illustration of the same:

8.0.1 Problem 5(Part-2)

The following is the pipeline which will take input from the six cameras attached to the vehicle. Basically, all the camera feeds are used in stitching images to create a panorama of 360 degree view from the top

1. Obtaining the live video feed of the cameras
2. Convert each frame to grayscale

- Mainly useful to apply the laplacian filter to it and which will be used to decide if the frame is blurry or not

3. Detection of keypoints based on SIFT

- Mainly because SIFT has this amazing scale invariant property which is versatile and reduces distortion while transforming

4. Feature Matching

- For feature making based on the needs, we can use BruteForce for accurate feature detection and can reduce the computation using FLANN which will efficiently choose features based on keypoints necesssary.

5. Homography Estimation

- After matching the feature, homography matrix will help transform one of the image to other's coordinate frame and will be able to warp the image accordingly

6. Blending Images

- After creating the panorama using different images from six feeds, images are blended to create a more smooth and continuous image of the surroundings

Overall, this pipeline is followed by all camera feeds to create an entire stiched image

```
[ ]:
```