# manojs_midterm

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 | Manoj Kumar Selvaraj |
| UID | 120511257 |
| Email | manojs@umd.edu |


## 2 Set path to the working directory

[1]:
```
from google.colab import drive
drive.mount('/content/drive/', force_remount=True)
```

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

[2]:
```
path_to_folder = "ENPM673/Midterm"
%cd /content/drive/MyDrive/{path_to_folder}
```

```
/content/drive/MyDrive/ENPM673/Midterm
```

## 3 Import Libraries

[3]:
```
# 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)

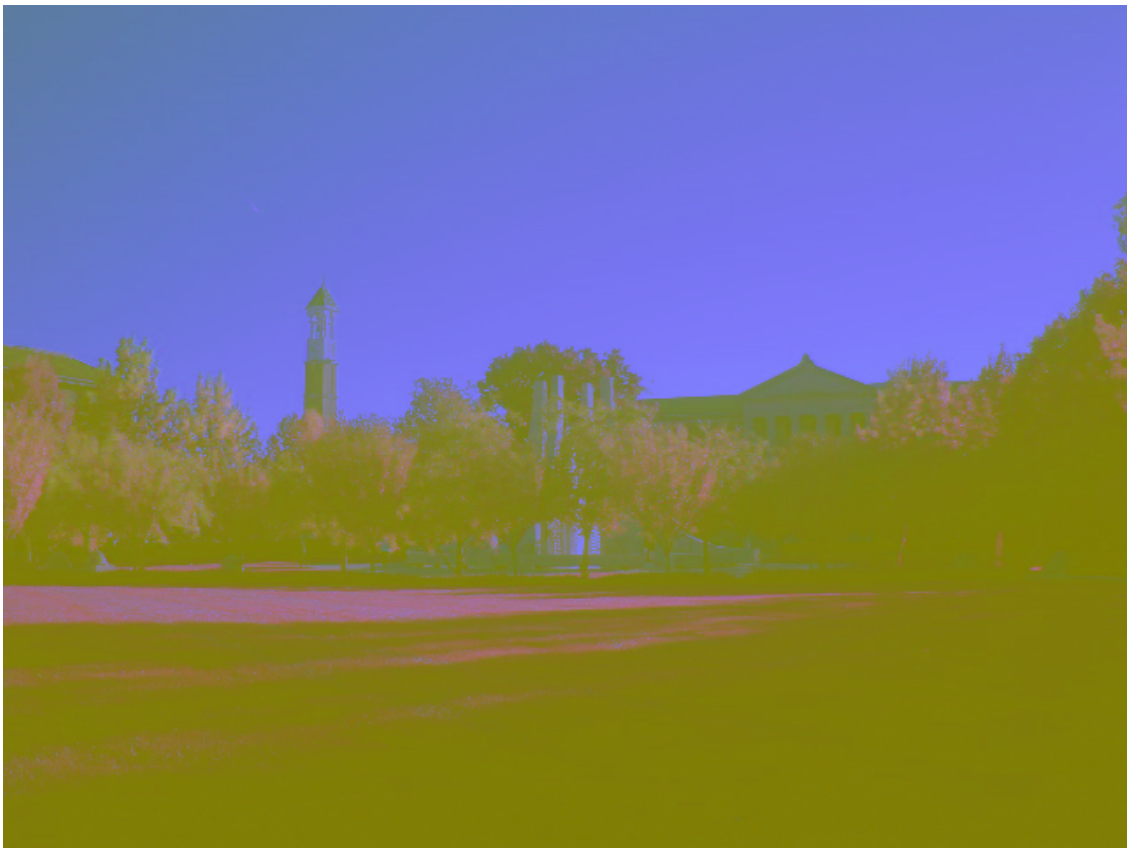
Link to input image: https://drive.google.com/file/d/1X0xir8CrOoKsNXxvZnLVAJWKztpMuUhV/view?usp=dri

**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

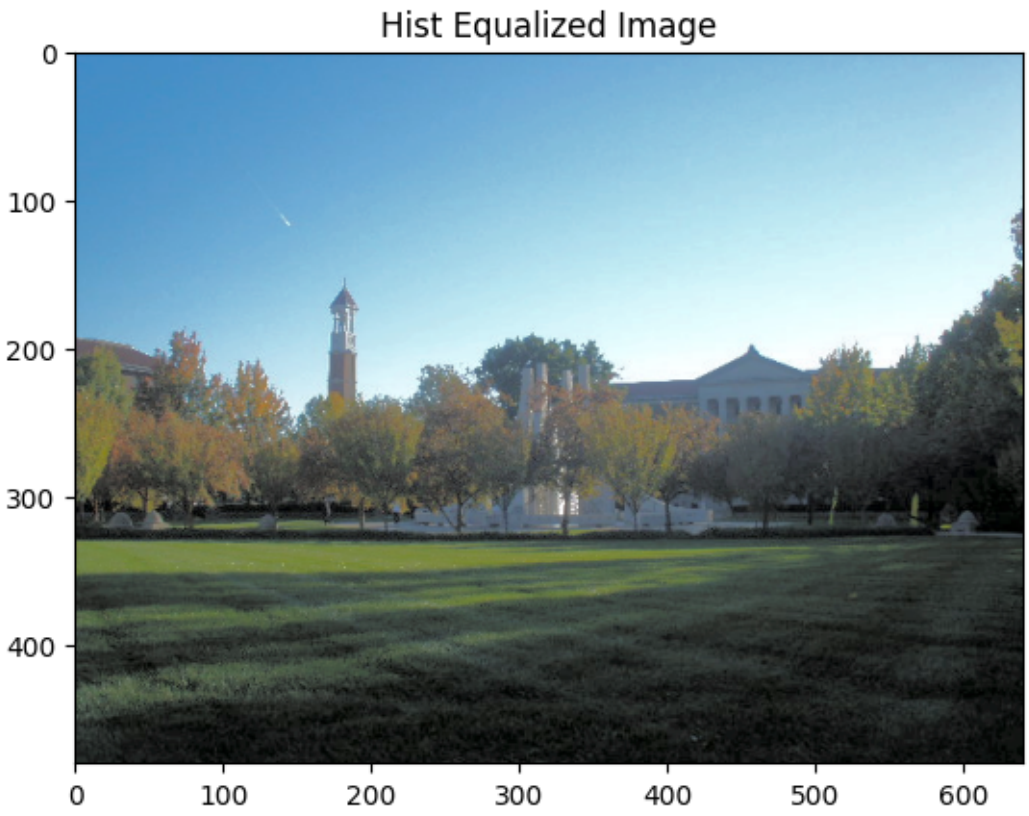
**Steps:** 1. Read the image 2. Convert ot LAB color space 3. Do histogram equalization on L channel, since it contains intensity info. 4. Merge it will A and B channels

[4]:
```
# write code for problem 1 part 1 here .....

#Reading the image and displaying it
```

```
img = cv2.imread("PA120272.JPG")
cv2_imshow(img)

#convert BGR TO LAB color space
img_LAB = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
cv2_imshow(img_LAB)

#Equalize only the intensity--> L(Lightness)
L_equalized = cv2.equalizeHist(img_LAB[:,:,0])
#keep green-red channel same
A_channel = img_LAB[:,:,1]
#keep blue-yellow same
B_channel = img_LAB[:,:,2]

#merge all channels
img_equalized = cv2.merge([L_equalized, A_channel, B_channel])

#Convert to BGR and display
img_final = cv2.cvtColor(img_equalized, cv2.COLOR_LAB2BGR)
cv2_imshow(img_final)

img_plot = cv2.cvtColor(img_final, cv2.COLOR_BGR2RGB)
plt.imshow(img_plot)
plt.title('Hist Equalized Image')
```



[4]: Text(0.5, 1.0, 'Hist Equalized Image')



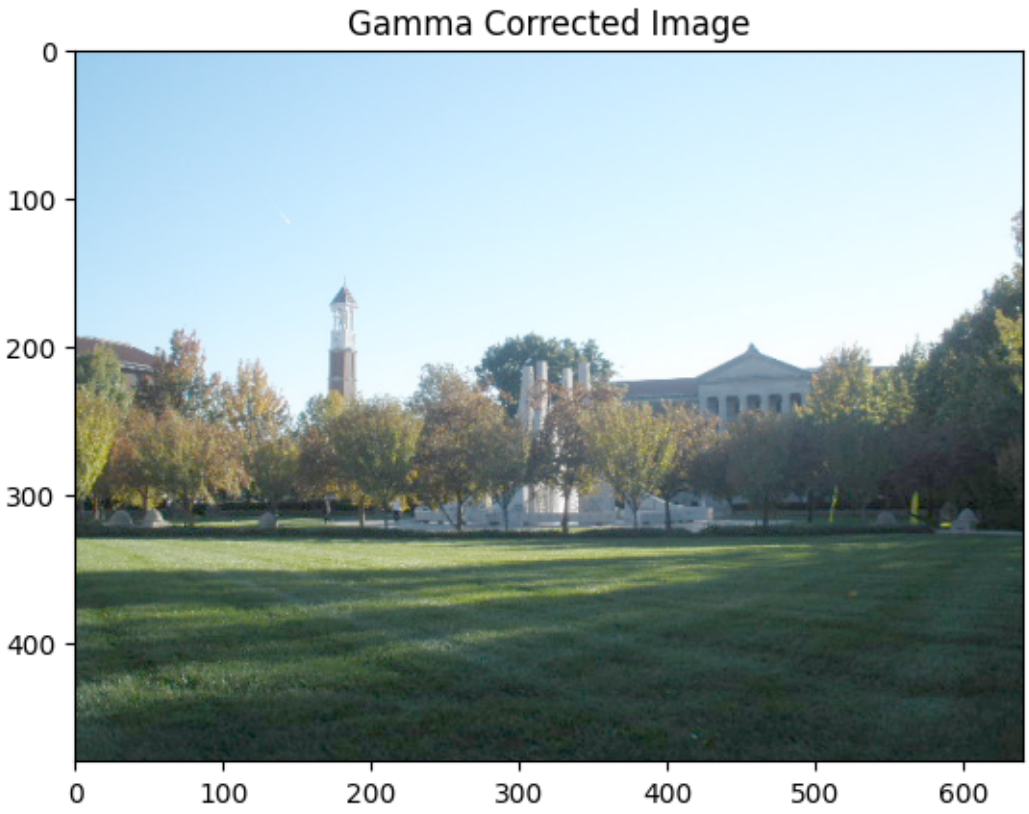

**Part 2: Another way to enhance a dark image is to perform gamma correction. For each intensity pixel, replace it with the new intensity = 255*((intensity/255)^(1/2.2)). Write code to perform gamma correction. Your result should look like below:** Link to the output of part-2: https://drive.google.com/file/d/1rgo7Kl8qK7Byh5QZsIb4CrfdBzY51Aco/view?usp=sharing

[5]:
```
# write code for problem 1 part 2 here .....

#Reading the image and displaying it
Image = cv2.imread("PA120272.JPG")
cv2_imshow(Image)

#Finding the height and width of an Image
height, width, channels = Image.shape
print("Height:",height)
print("Width:",width)

#Iterating in a for loop to do gamma correction
for i in range(0,height):
  for j in range(0,width):
    #gamma correction
```

```
    new_intensity = 255*((Image[i][j]/255) ** (1/2.2))    #Gamma = 2.2
    Image[i][j] = new_intensity

#Display
cv2_imshow(Image)

img = cv2.cvtColor(Image, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.title('Gamma Corrected Image')
```

```
Height: 480
Width: 640
```

[5]: Text(0.5, 1.0, 'Gamma Corrected Image')




## 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

**Problem-2 (1)** Image is n x n

Kernel(H) is m x m

For example take Image as 5x5 and H as 3x3

so,

For each pixel of Image we do 9 times multiplication and 9 times addition.

i.e nxn times multiplication and nxn times addition.

So for all pixels in Image we do 5x5x9 times mutiplication i.e mxmxnxn times multiplication.

Therefore total multiplications is n^2 * m^2

–> O(n^2 x m^2)

**Problem-2 (2)** A kernel is separable if it can be written as the outer product of 2 vectors.

A separable kernel can be broken down into two separate kernels (i.e) typically applied in different directions (vertical and horizontal).

This reduces the time complexity.

Given H value is separable

so For each pixel of Image we do 2*m multiplications. For n number of pixels we do nxnx2m multiplications.

Therefore Time complexity is O(n^2 x m). (2 is constant and removed)

**Problem-2 (3)** By applying principle of separability we get,

3x1 and 1x3 separable matrices as

```
1/16  .  [1
          2        . [1 2 1]
          1]
```

# 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.**

**Problem-3 (1)**

1. Load the x, y, z data from csv.
2. Subtract mean values from each corresponding dimension in the data points. This centers the data around the origin making the mean of each dimension to zero.
3. Calculate covariance matrx.
4. Perform Eigenvalue decomposition on covariance matrix to get Eigenvalues and eigenvectors.
5. The eigenvector correspondng to the largest eigenvalue indicates the direction of the major axis of the ellipse.

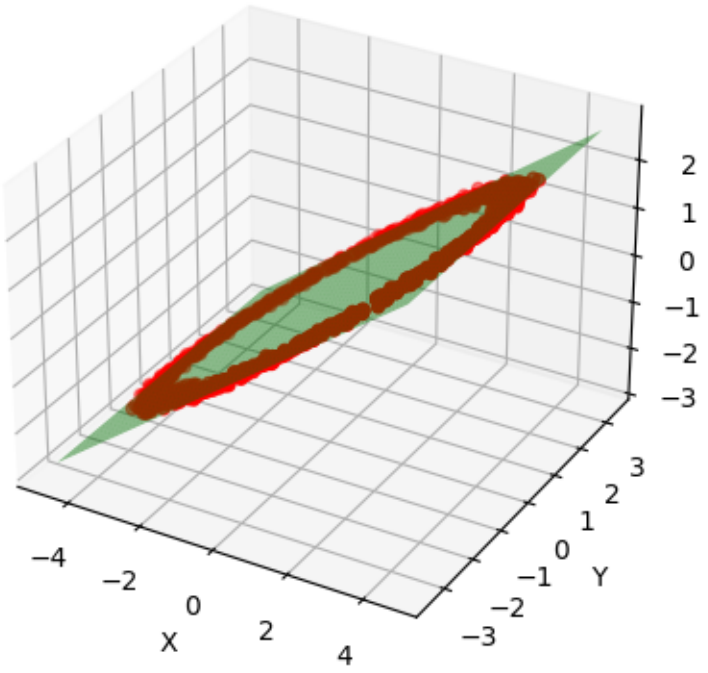
6. Similarly, the eigenvector corresponding to the smallest eigenvalue points in the direction of the minor axis.
7. Compute the lengths of the major and minor axes by taking the square root of the corresponding Eigenvalus.

**Problem-3 (2)**

[6]:
```
import numpy as np

# Load the data from the CSV
pts = np.loadtxt('problem3_points.csv', delimiter=',', skiprows=1) #to skip row␣
 ↪1

# Extract x, y, and z from the pts
x = pts[:, 0]
y = pts[:, 1]
z = pts[:, 2]

# Plot the x,y,z points
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, color='red', label='points')

# Create the A matrix for the plane fitting
A = np.column_stack((x, y, np.ones_like(x)))

# Use least squares to fit a plane
plane_values, _, _, _ = np.linalg.lstsq(A, z, rcond=None)

# Extracting coeffs of the plane equation (a.x + b.y + c.z + d = 0)
a, b, c = plane_values
d = -1

# Print the equation of the fitted surface
print(f"Equation of the surface: {a}*x + {b}*y + {c}*z + {d} = 0")

# Define grid of points to plot
x_grid, y_grid = np.meshgrid(np.linspace(min(x), max(x), 100),
                             np.linspace(min(y), max(y), 100))

z_plane = (a * x_grid + b * y_grid + c)

# Plot the fitted surface
ax.plot_surface(x_grid, y_grid, z_plane, alpha=0.5, color='green',␣
 ↪label='Surface')

# Set labels
ax.set_xlabel('X')
```

```
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

```
Equation of the surface: 0.577350269183371*x + 3.710777815521294e-12*y +
-9.175070607989371e-12*z + -1 = 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)**

1. Read the Image

2. Convert the image to grayscale
3. Give src and dst points for perspectiveTransform.
4. From Transformation matrix , wrap perspective of Input image.
5. Sharpen the edges by addweighted (difference of Image with blurred image)
6. Do Gaussian Blur to remove noise and do thresholding to seperate out similar intensities.
7. Dilate and perform canny to find the edges.
8. Find Hough transformation to find the straight lines.
9. Find vertical straight lines and find the distance between them.

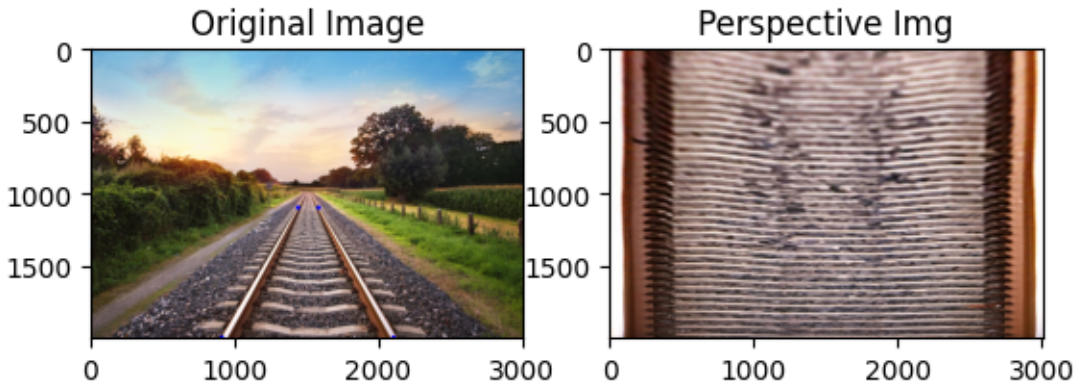
**Problem-4 (2)**

[13]:
```
#Reading the image and displaying it
Img = cv2.imread("train_track.jpg")
Img_copy = Img.copy()

#Reading height and width of image
h,w,c = Img.shape

#converting the image to grayscale
Img_gray = cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)

#Perspective transform src points and dst points
pts1 = np.float32([[1438,1100], [1580,1100], [910,1999], [2100,1999]])

pts2 = np.float32([[0,0], [w-1,0], [0,h-1], [w-1,h-1]])

M = cv2.getPerspectiveTransform(pts1,pts2)
print("Transformation matrix is \n:", M)

#Do wrap perspective of img
dst = cv2.warpPerspective(Img, M, (w-1, h-1))

# Plot circle in chosen src points
for val in pts1:
    cv2.circle(Img_copy, (int(val[0]), int(val[1])), 15, (255,0,0), -1)

#plotting
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(Img_copy, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

#perspective img
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
plt.title('Perspective Img')

plt.show()
```

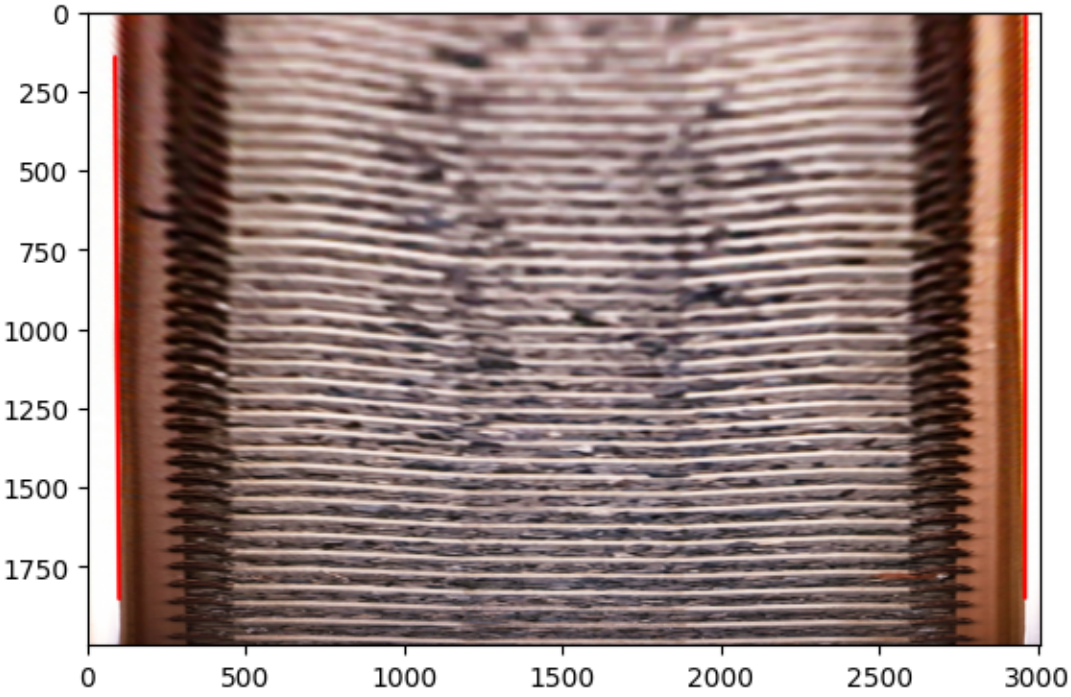
```
# Apply Gaussian blur 5x5
blur_dst = cv2.GaussianBlur(dst, (0, 0), 5)

# Subtract blur image from orig image to sharpen the edges
sharpened_dst = cv2.addWeighted(dst, 1.5, blur_dst, -0.5, 0)

#convert dst to gray
dst_gray =  cv2.cvtColor(sharpened_dst, cv2.COLOR_BGR2GRAY)

#Gausian Blur
blur = cv2.GaussianBlur(dst_gray, (0, 0), 3)

#Do thresholding to get only train tracks with t as 200
_, thresh_dst = cv2.threshold(blur, 200, 255, cv2.THRESH_BINARY)

#Dilate to get lines without breakage with 5x5
dilated_image = cv2.dilate(thresh_dst, (5,5), iterations=5)

#Canny edge detect
edge_dst= cv2.Canny(dilated_image, 255, 255)

# Perform morphologicalEx to make linecontinuous
kernel = np.ones((3,3), np.uint8)
edge_dst = cv2.morphologyEx(edge_dst, cv2.MORPH_CLOSE, kernel)

#Hough transform with intersection points as 20
lines = cv2.HoughLinesP(edge_dst, 1, np.pi / 180, 20, None, 100, 80)

#plot lines
#store 2 lists for 2 parallel lines
ver_lines1 = []
ver_lines2 = []

if lines is not None:
  for i in range(0, len(lines)):
        pts1 = lines[i][0]
        if(pts1[0]<200 or pts1[0]>2870):
          #Condition for parallel line1
          if( (pts1[0] < 300) ):
            ver_lines1.append(pts1)
          #condition for parallel line 2
          else:
            ver_lines2.append(pts1)

# Find the point with the min and max x-coordinates --> line1
min_x_point1 = min(ver_lines1, key=lambda point: point[1])
max_x_point1 = max(ver_lines1, key=lambda point: point[3])
```

```
#plot
cv2.line(dst, (min_x_point1[0], min_x_point1[1]), (max_x_point1[2],␣
 ↪max_x_point1[3]), (0,0,255), 12, cv2.LINE_AA)

# Find the point with the min and max x-coordinates -->line 2
min_x_point2 = min(ver_lines2, key=lambda point: point[1])
max_x_point2 = max(ver_lines2, key=lambda point: point[3])

#plot
cv2.line(dst, (min_x_point2[0], min_x_point2[1]), (max_x_point2[2],␣
 ↪max_x_point2[3]), (0,0,255), 12, cv2.LINE_AA)

#Distance calculation for the line
x1 = int((max_x_point1[0] + min_x_point1[0] ) / 2)
x2 = int((max_x_point2[0] + min_x_point2[0] ) / 2)

#show
plt.imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))

#This distance is for wrap image of size same as original image
print("Distance for warp image is:",x2-x1,"pixels")

#since the src pts [910,1999], [2100,1999] are transformed into [0,h-1],␣
 ↪[w-1,h-1]
#scale is 2.52
#Distance/2.52 is w.r.t original image
print("Distance w.r.t original image is:",(x2-x1)/2.52,"pixels")
```

```
Transformation matrix is
: [[-2.63699370e+00 -1.54875715e+00  5.49562981e+03]
 [-1.73237086e-15 -2.32046877e+00  2.55251565e+03]
 [-9.10054117e-19 -1.02229740e-03  1.00000000e+00]]
```



```
Distance for warp image is: 2868 pixels
Distance w.r.t original image is: 1138.095238095238 pixels
```




## 8 Problem 5 (15 Points)

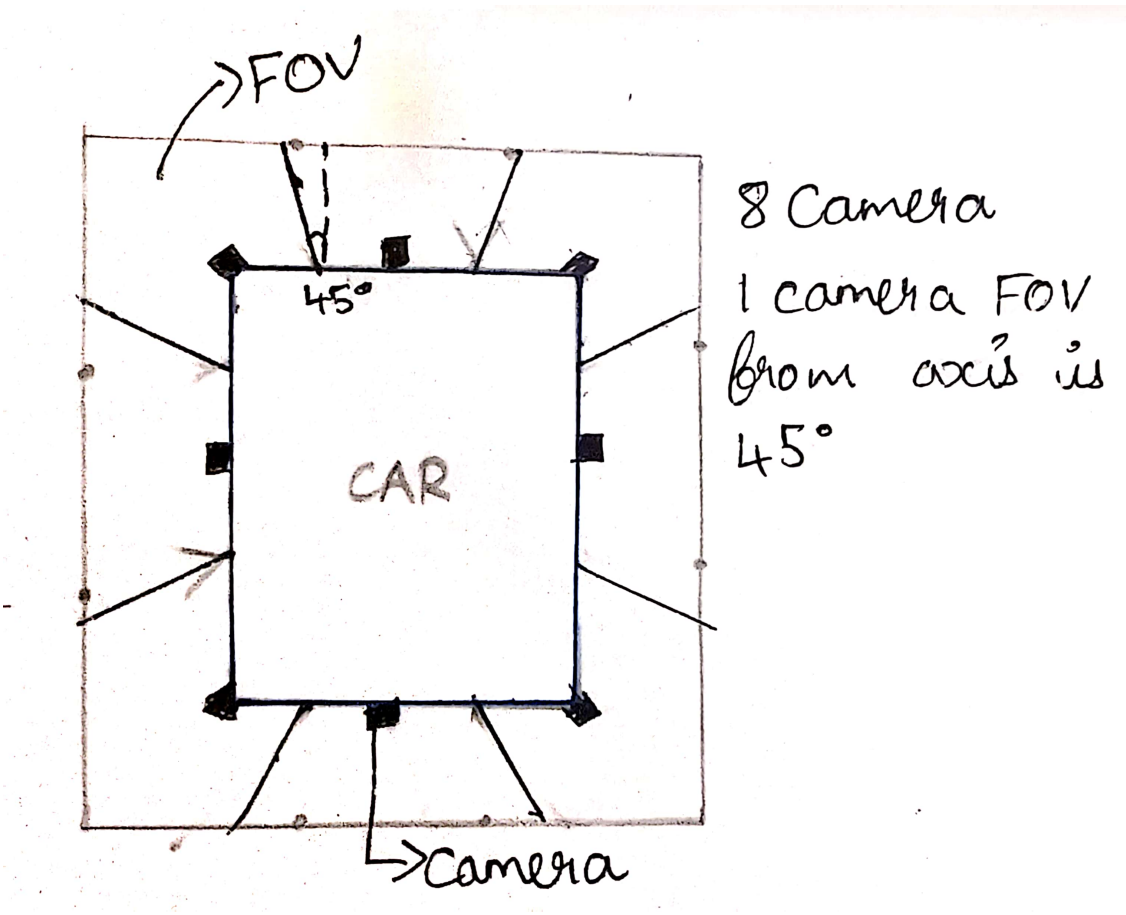
**Let's say you want to design a system that will create a top view of the surrounding around the car, see the image below for referene. Do the following,**

(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.

[8]:
```
Img = cv2.imread("car.jpg")
cv2_imshow(Img)
```




1) From the above image,

Totally 8 cameras(monocular) are required to cover the birdview of the car.

FOV is 45 degree for each camera along axis and 8 cameras are placed on all 4 sides and 4 corners.

Also focal length plays an important role

As f is samller FOV is wide

As f is large FOV is narrow

Some cars come with 360 degree camera- with two ultra wide fisheye lenses, one on each sides of the camera.

2) **pipeline**

1. Read the image from all 8 cameras
2. Convert all the images to grayscale.
3. Use Feature Detectors and Matching (SIFT,Knn) for all images.
4. Find homography matrix for all consecutive pairs.
5. Apply homograph matrix to warp perspective to stitch the images.i.e stitch consecutive pairs.(Remove black areas)

6. Repeat the warping and stitching process for all consecutive pairs of images to create a panoramic view.
7. Adjust the stitching process to ensure continuity and smooth blending across all pairs of images.
8. By stitching all the images with consecutive image , a 360 degree view is obtained.