Programming Assignment - 1, Computer Vision

Q1. Harris Corner Detection

Functions for generating Sobel and Gaussian Filters from scratch

```python
def generate_sobel_filters(n):
    if n % 2 == 0:
        raise ValueError("The size of the filter 'n' must be odd")

    sobel_x = np.zeros((n,n),dtype="float64")
    sobel_y = np.zeros((n,n),dtype="float64")

    mid = n//2

    for i in range(n):
        for j in range(n):
            sobel_x[i,j] = (mid - i)*(2 - (mid!=j))
            sobel_y[i,j] = (mid - j)*(2 - (mid!=i))

    return sobel_x,sobel_y

# Gaussian Filter Generator Function
def generate_gaussian_filter(n,sigma):
    if n % 2 == 0:
        raise ValueError("The size of the kernel 'n' must be odd")

    kernel = np.zeros((n, n))

    mid = n // 2

    for i in range(n):
        for j in range(n):
            x = i - mid
            y = j - mid
            kernel[i, j] = np.exp(-(x**2 + y**2) / (2 * sigma**2))

    kernel = kernel / np.sum(kernel)
    return kernel
```

Function for convolution

```python
# Convolution Function
def convolve(img, kernel):
    if kernel.shape[0] % 2 != 1 or kernel.shape[1] % 2 != 1:
        raise ValueError("The dimensions of kernel must be both odd")

    img_height,img_width = img.shape[0],img.shape[1]

    pad_height,pad_width = kernel.shape[0]//2 , kernel.shape[0]//2

    pad = ((pad_height, pad_height), (pad_height, pad_width))

    g = np.empty(img.shape, dtype=np.float64)
    img = np.pad(img, pad, mode='constant', constant_values=0)

    # Convolution
    for i in np.arange(pad_height, img_height+pad_height):
        for j in np.arange(pad_width, img_width+pad_width):
            roi = img[i - pad_height:i + pad_height + 1, j - pad_width:j + pad_width + 1]
            g[i - pad_height, j - pad_width] = (roi*kernel).sum()

    if (g.dtype == np.float64):
        kernel = kernel / 255
        kernel = (kernel*255).astype(np.uint8)
    else:
        g = g + abs(np.amin(g))
        g = g / np.amax(g)
        g = (g*255)
    return g
```

Function for detecting corners following Harris Corner Detecting Algorithm, from scratch

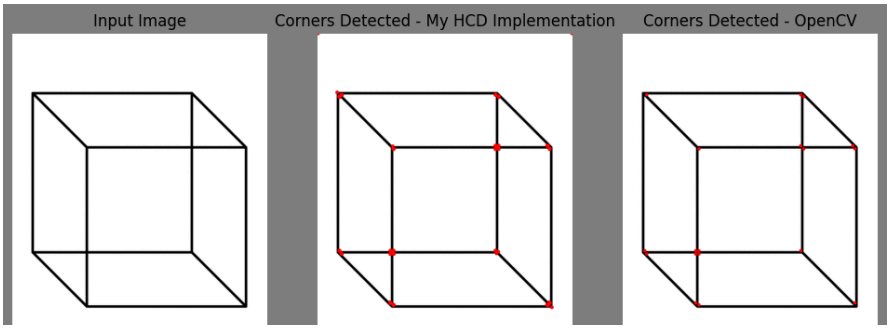
```python
def harris(img, threshold=0.6, window_size=3,sobel_filter_size=3,k=0.12):
    img_gray = img.copy()
    img_gray = np.array(img_gray.convert('L'))

    sobel_filter_x , sobel_filter_y = generate_sobel_filters(sobel_filter_size)

    gaussian_filter = generate_gaussian_filter(window_size,1)

    Ix = convolve(img_gray, sobel_filter_x) # convolving with sobel filter on X-axi
    Iy = convolve(img_gray, sobel_filter_y) # convolving with sobel filter on Y-axi

    # square of derivatives
    Ix2 = np.square(Ix)
    Iy2 = np.square(Iy)

    IxIy = Ix*Iy #cross filtering

    g_Ix2 = convolve(Ix2, gaussian_filter)
    g_Iy2 = convolve(Iy2, gaussian_filter)
    g_IxIy = convolve(IxIy, gaussian_filter)

    # Harris Function
    # R(harris) = det - k*(trace**2)
    harris = g_Ix2*g_Iy2 - np.square(g_IxIy) - k*np.square(g_Ix2 + g_Iy2)

    # Normalizing output image values
    cv2.normalize(harris, harris, 0, 1, cv2.NORM_MINMAX)

    # find all points above threshold (nonmax supression line)
    loc = np.where(harris >= threshold)

    output_img = img.copy()
    output_img = np.array(output_img.convert('RGB'))

    # drawing filtered points on the output image
    for pt in zip(*loc[::-1]): cv2.circle(output_img, pt, 5, (255, 0, 0), -1)

    return output_img
```

Code for reaching the image's directory

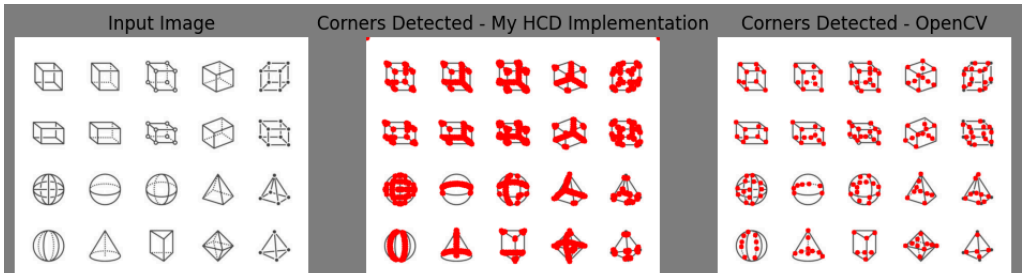
```
[7]
    path = './Question 1'
    list_images = os.listdir(path)
    print(list_images)

... ['1.gif', '10.jpg', '11.jpg', '2.jpeg', '3.png', '5.jpg', '6.jpg', '7.jpg', '8.jpg', '9.jpeg']
```

Comparing the Results from both our implementation and OpenCV's Library functions after optimizing the hyperparameters:

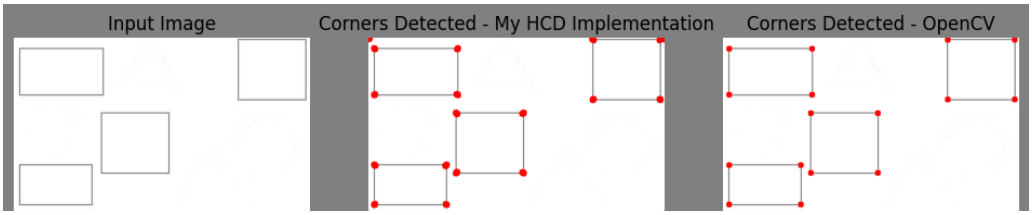
1. On Cube, both algorithms perform nicely



2. On Different Shapes, both confuse the round edges as corners for spheres



3. On Rectangles of different sizes

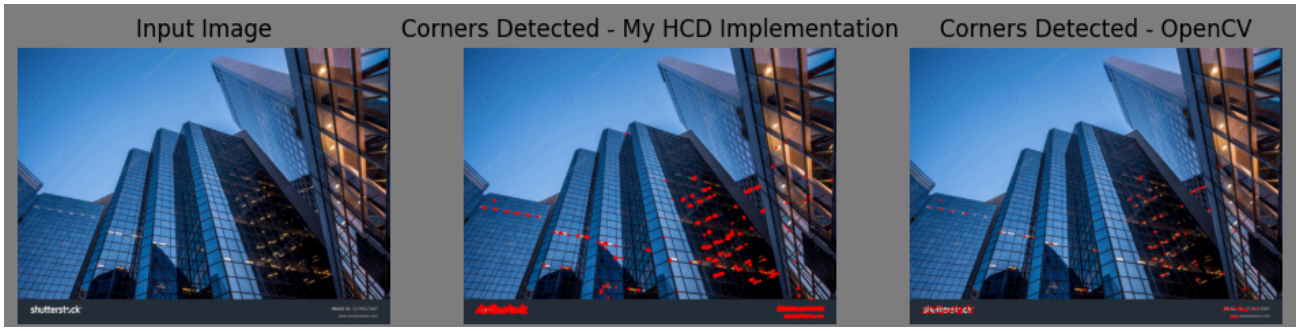


4. On Bird, our implementation looks into more detail and confuses more round edges with corners, this can be reduced by adding a min_distance property which specifies the minimum distance between two detected corners.

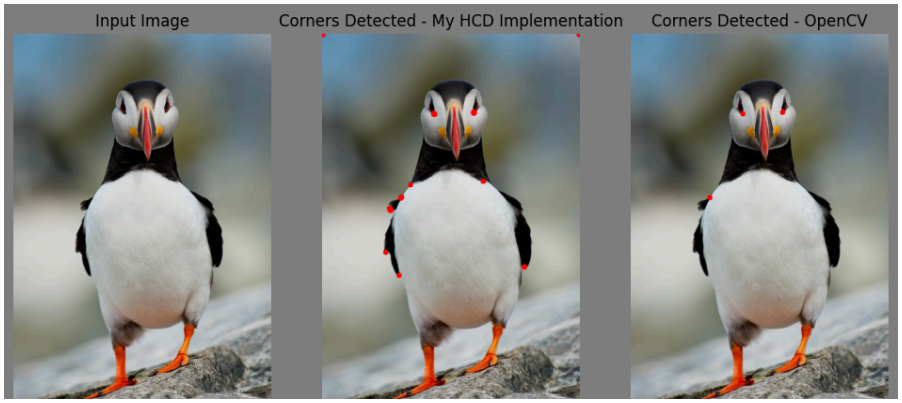
5. On houses, due to less intensity difference the middle house remains undetected by both algorithms

6. On flower, both algorithms perform nicely on the image

7. On buildings, both algorithms are unable to detect small changes in intensities by the borders of windows of buildings but our algorithm going in more detail of intensity changes performs better



8. On Penguin, both algorithms perform nicely



Q2. 3D Reconstruction, Depth Map, Disparity Map

Input Images:

Finding the disparity map using block size = 23 and num_diparities = 144 using template matching, and then using the disparity values pixel-wise to obtain a depth map like disparity map. Since **Disparity = b*f/Depth => Lower depth values correspond to higher disparity**

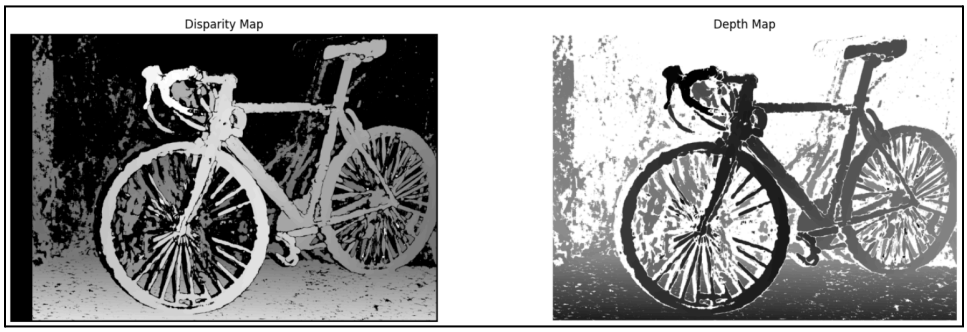
Note: values of num_disparity and block size are chosen after hyperparameter tuning

Code for finding the Depth and Disparity Maps:

```
def calcDisparityAndDepthMaps(imageL,imageR):
    img1 = imageL.copy()
    img1 = np.array(img1.convert('L'))

    img2 = imageR.copy()
    img2 = np.array(img2.convert('L'))

    stereo = cv2.StereoBM_create(numDisparities=144, blockSize=23)
    b = baseline
    f = cam0[0][0]

    disparity_map = stereo.compute(img1,img2)

    depth_map = np.zeros((disparity_map.shape))

    for i in range(len(depth_map)):
        for j in range(len(depth_map[i])):
            if disparity_map[i][j]==0: # A point which doesn't have disparity
                depth_map[i][j] = depth_map[i][j-1]
            else:
                disparity_map[i][j] = abs(disparity_map[i][j])
                depth_ij = ((b*f)/disparity_map[i][j])
                depth_map[i,j] = depth_ij

    return disparity_map,depth_map

disparity_map,depth_map = calcDisparityAndDepthMaps(image_L,image_R)
```

Disparity and Depth Maps Obtained:

#Note: The cmaps are used to show the depth and disparity values nicely



Code for 3D point cloud formation

```
def transformInto3DCloud(points):
    # Making the point cloud in accordance to Left Camera

    X,Y,Z = [],[],[]
    ox,oy = cam0[0][2],cam0[1][2]
    fx,fy = cam0[0][0],cam0[1][1]

    for i in points:
        x = i[0]
        y = i[1]
        z = i[2]

        X.append( ((x-ox)*z)/fx )
        Y.append( ((y-oy)*z)/fy )
        Z.append( z )

    return X,Y,Z

points = []
(r,c) = np.array(depth_map).shape

for i in range(r):
    for j in range(c):
        points.append((i,j,round(depth_map[i][j],2)))

X,Y,Z = transformInto3DCloud(points)
```

Using depth values we already have Z, now for each pixel we can use the below equations to find the X,Y coordinates in the camera system

Xc = (u - Ox)*Z / fx | Yc = (v - Oy)*Z / fy , here u,v represents the pixel coordinates

P. T. O.

Sampling down points because of lesser plotting resources:

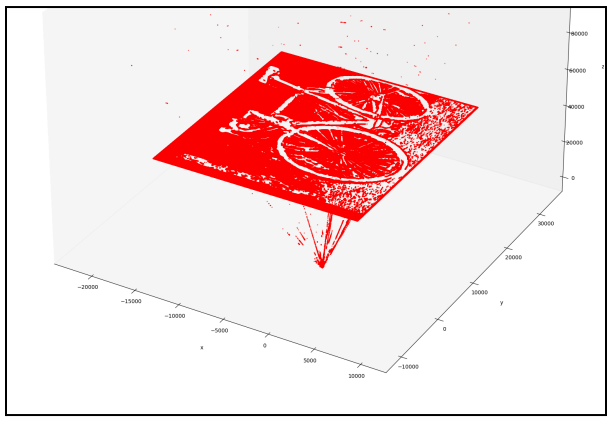
```
points = {
    'X':X[::50],
    'Y':Y[::50],
    'Z':Z[::50]
}
```

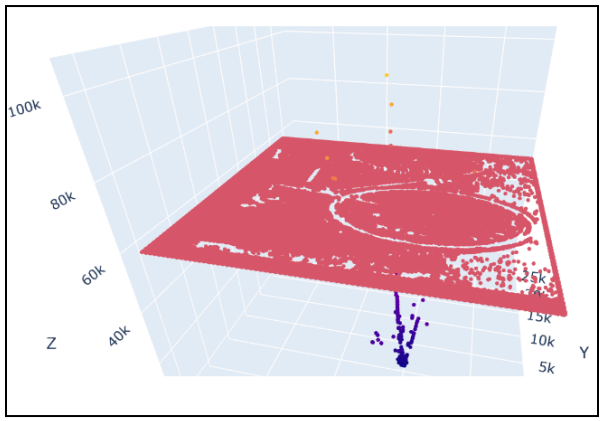
Code for Plotting Point Clouds:

```
fig = plt.figure(figsize = (20,20))
ax = plt.axes(projection='3d')
ax.grid(visible=False)

ax.scatter(X, Y, Z, c = 'r', s = 1)
ax.set_title('3D Scatter Plot')

# Set axes label
ax.set_xlabel('x', labelpad=20)
ax.set_ylabel('y', labelpad=20)
ax.set_zlabel('z', labelpad=20)

plt.show()
```

```
df = pd.DataFrame(points)
fig = px.scatter_3d(df,x='X',y='Y',z='Z',color='Z')
fig.update_traces(marker=dict(size=2))
fig.show()
```

3D Scatter Plots for Point Clouds Obtained:





P. T. O.

Q3. Epipolar Geometry

Part 1: Finding Epipolar Lines

Code for selecting points and finding the Epipolar Lines
And Plotting them

```python
# Function for picking up points and finding their epipolar lines, on the images
def selectPointsAndfindEpipolarLines(imageL,imageR,F):
    points_L = []
    points_R = []

    mid_point_1 = (imageL.shape[0]//2,imageL.shape[1]//2)
    mid_point_2 = (imageR.shape[0]//2,imageR.shape[1]//2)

    points_L = [[mid_point_1[0],i,1] for i in range(1,imageL.shape[1],10)]
    points_R = [[mid_point_2[0],i,1] for i in range(1,imageR.shape[1],10)]

    lines_L = np.array([ np.dot(F,i) for i in points_R])
    lines_R = np.array([ np.dot(F.transpose(),i) for i in points_L])

    return points_L,points_R,lines_L,lines_R
```

```python
# Function for drawing Epipolar Lines on images
def drawPointsAndEpipolarLines(pts_1,pts_2,imageL,lines_L,imageR,lines_R):

    output_imageL_points = imageL.copy()
    output_imageR_points = imageR.copy()

    output_imageL_lines = imageL.copy()
    output_imageR_lines = imageR.copy()

    for p in pts_1:
        color = (255,0,0)
        x,y = p[0],p[1]
        output_imageL_points = cv2.circle(output_imageL_points,(x,y)[::-1],5,color,-1)

    for p in pts_2:
        color = (255,0,0)
        x,y = p[0],p[1]
        output_imageR_points = cv2.circle(output_imageR_points,(x,y)[::-1],5,color,-1)

    for r in lines_L:
        color = (0,255,0)
        x0, y0 = map(int, [0, -r[2]/r[1]])
        x1, y1 = map(int, [imageL.shape[1], -(r[2]+r[0]*imageL.shape[1])/r[1]])
        output_imageL_lines = cv2.line(output_imageL_lines, (x0,y0), (x1,y1), color, 2)

    for r in lines_R:
        color = (0,255,0)
        x0, y0 = map(int, [0, -r[2]/r[1]])
        x1, y1 = map(int, [imageR.shape[1], -(r[2]+r[0]*imageR.shape[1])/r[1]])
        output_imageR_lines = cv2.line(output_imageR_lines, (x0,y0), (x1,y1), color, 2)
```

Using 000023.png as Left and 000000.png as right Image for the above code I got the following results:

Part 2: Finding Correspondences

Using Sift feature descriptors for points in the image

```python
def find_descriptor(image, pixel_coordinates):
    # Convert pixel coordinates to keypoint format
    keypoint = cv2.KeyPoint(x=pixel_coordinates[1], y=pixel_coordinates[0], size=1)

    # Initialize SIFT detector
    sift = cv2.SIFT_create()

    # Compute SIFT descriptors for the keypoint
    keypoints, descriptors = sift.compute(image, [keypoint])

    # Return the descriptor
    return descriptors[0]
```

P. T. O.

1. Picking up uniformly spaces points on a random epipolar line,
2. Finding Their Descriptors and Storing them
3. Plotting them on the Left Image

```
temp1 = imageL.copy()
temp2 = imageR.copy()

colors = [(0,0,255),(0,255,0),(255,0,0),(0,255,255),(0,0,255),(0,255,0),(255,0,0),(0,255,255),(0,0,255),(0,255,0)]

# First Epipolar Line
left_epipolar_line = lines_L[22]

# 10 points on First Epipolar Line
uniformly_spaced_points_L = [(
    -(left_epipolar_line[2]+left_epipolar_line[0]*(imageL.shape[1]//2 + 20*i))/left_epipolar_line[1],
    imageL.shape[1]//2 + 20*i
) for i in range(-5,5)]

# Finding Descriptors of the 10 points
descriptors_L = np.array([ find_descriptor(imageL,(round(i[0]),round(i[1]))) for i in uniformly_spaced_points_L ])


# Plotting First Epipolar Line
x0, y0 = map(int, [0, -left_epipolar_line[2]/left_epipolar_line[1]])
x1, y1 = map(int, [imageL.shape[1], -(left_epipolar_line[2]+left_epipolar_line[0]*imageL.shape[1])/left_epipolar_line[1]])
temp1 = cv2.line(temp1, (x0,y0), (x1,y1), (0,0,0), 2)


# Plotting 10 uniformly spaced points on First Epipolar Line
c = 0
for i in uniformly_spaced_points_L:
    x,y = round(i[0]),round(i[1])
    temp1 = cv2.circle(temp1,(y,x),6,colors[c],-1)
    c+=1
```

4. Computing and Plotting the second epipolar line

```
# Computing the Second Epipolar Line
epipolar_line_on_right = [0,0,0]

for i in uniformly_spaced_points_L:
    temp_line = np.dot( F.transpose() , (i[0],i[1],1) )

    epipolar_line_on_right[0] += temp_line[0]
    epipolar_line_on_right[1] += temp_line[1]
    epipolar_line_on_right[2] += temp_line[2]

epipolar_line_on_right[0]/=10
epipolar_line_on_right[1]/=10
epipolar_line_on_right[2]/=10

# Plotting the Second Epipolar Line
x2, y2 = map(int, [0, -epipolar_line_on_right[2]/epipolar_line_on_right[1]])
x3, y3 = map(int, [imageR.shape[1], -(epipolar_line_on_right[2]+epipolar_line_on_right[0]*imageR.shape[1])/epipolar_line_on_right[1]])
temp2 = cv2.line(temp2,(x2,y2),(x3,y3),(0,0,0),2)
```

P. T. O.

5. Finding the corresponding points for the 10 points we chose using our min Euclidian square distance between feature descriptors of points on the second epipolar line and our 10 points.

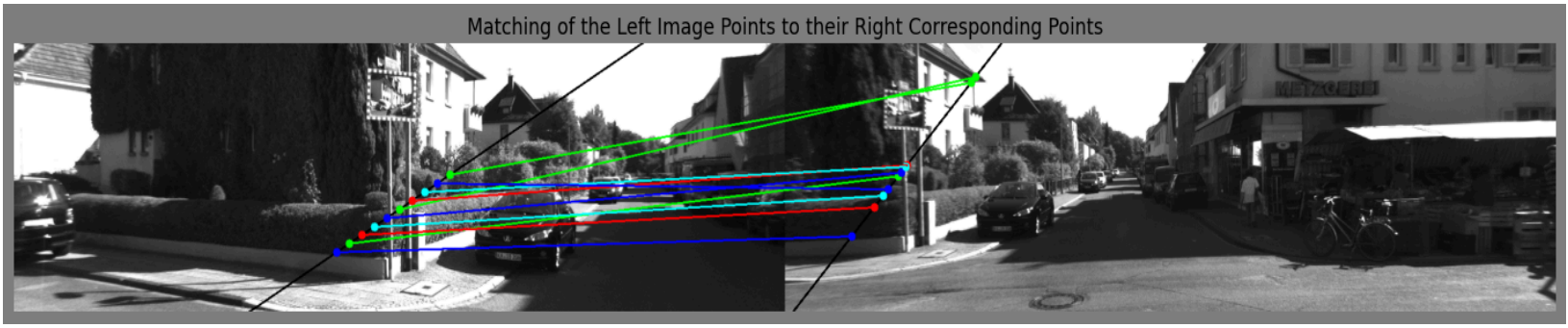
```
# Finding the Corresponding Points
x_min = 0
y_min = 0

x_max = imageR.shape[0]
y_max = imageR.shape[1]

corr_points_R = [None for _ in range(10)]
error_in_corr_R = [float('inf') for _ in range(10)]

for xi in range(x_min,x_max):
    yi = round(-(epipolar_line_on_right[1]*xi + epipolar_line_on_right[2])/epipolar_line_on_right[0])
    if y_min <= yi < y_max:
        fdi = np.array(find_descriptor(imageR,(xi,yi)))

        for p in range(10):
            diff = np.sum((fdi-descriptors_L[p])**2)

            if error_in_corr_R[p] >  diff:
                error_in_corr_R[p] = diff
                corr_points_R[p] = (xi,yi)

# Plotting the corresponding points
c=0
for i in corr_points_R:
    temp2 = cv2.circle(temp2,i[::-1],6,colors[c],-1)
    c+=1
```

6. Matching the 10 points in left image on first epipolar line with their corresponding right points on the second epipolar line

```
c=0
result = np.hstack((temp1, temp2))
for (x1, y1), (x2, y2) in zip(uniformly_spaced_points_L, corr_points_R):
    result = cv2.line(result, (round(y1), round(x1)), (temp1.shape[1] + round(y2), round(x2)), colors[c], 2)
    c+=1

# Show the result
fig = plt.figure(figsize=(18,10),facecolor='gray')
plt.axis('off')
plt.title("Matching of the Left Image Points to their Right Corresponding Points")
plt.imshow(result)
```

P. T. O.

Using the above code blocks I obtained the below matching of points from the left image to the right image



Following the same code for 10 points on the right image and finding their corresponding points in the left image I got the following results: