Assignment_2___Q2___Solution

April 13, 2023

1 Khushdev Pandit

2 Roll no: 2020211

3 *Assignment Question-2*

4

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
import open3d as o3d
import glob
```

[ ]:
```
# Helper functions to load the camera parameters
def load_camera_parameters(camera_matrix_path, distortion_coefficients_path):
    with open(distortion_coefficients_path, 'r') as f:
        contents = f.read()
        distCoeff = contents.split('\n')[:1][0].split(' ')
        distCoeff = np.array(distCoeff, dtype=np.float32)
    return load_matrix(camera_matrix_path), distCoeff

# Load the matrix from the file
def load_matrix(file_path):
    with open(file_path, 'r') as f:
        contents = f.read()
        matrix = contents.split('\n')[:3]
        matrix = [i.split(' ') for i in matrix]
        matrix = np.array(matrix, dtype=np.float32)
    return matrix

# Load the vector from the file
def load_vector(file_path):
    with open(file_path, 'r') as f:
        contents = f.read()
        vector = contents.split('\n')[:3]
```

```
        vector = np.array(vector, dtype=np.float32)
    return vector

# Helper functions to load the camera parameters
def load_camera_coord_param_for_image(folder_path):
    camera_normal = load_vector(os.path.join(folder_path, 'camera_normals.txt'))
    rotation_matrix = load_matrix(os.path.join(folder_path, 'rotation_matrix.
 ↪txt'))
    rotation_vector = load_vector(os.path.join(folder_path, 'rotation_vectors.
 ↪txt'))
    translation_vector = load_vector(os.path.join(folder_path,␣
 ↪'translation_vectors.txt'))
    return camera_normal, rotation_matrix, rotation_vector, translation_vector
```

[ ]:
```
# load all the lidar scans, camera images and camera parameters
lidar_scans = glob.glob('lidar_scans/*.pcd')
camera_images = glob.glob('camera_images/*.jpeg')
camera_coord_param = glob.glob('camera_parameters/*.jpeg')
camera_param = glob.glob('camera_parameters/*.txt')
indices = np.arange(0, len(lidar_scans))
indices = np.random.choice(indices, 25, replace=False)

lidar_scans.sort()
camera_images.sort()
camera_coord_param.sort()

camera_images = np.array(camera_images)[indices]
lidar_scans = np.array(lidar_scans)[indices]
camera_coord_param = np.array(camera_coord_param)[indices]
```

[ ]:
```
from tqdm import tqdm

# define the size of the checkerboard
checkerboard_size=(4,6)

# define the world coordinates of the checkerboard
objects_points = []
image_points = []

# termination criteria for the sub-pixel refinement
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# loop over all the images to find the chessboard corners
for image_name in tqdm(camera_images):
    img = cv2.imread(image_name)
    inverted_img = np.array(255 - img, dtype=np.uint8)
    gray_img = cv2.cvtColor(inverted_img, cv2.COLOR_BGR2GRAY)
```

```
    # 3D points in real world space
    objp = np.zeros((1, checkerboard_size[0]*checkerboard_size[1], 3), np.
↪float32)
    objp[0,:,:2] = np.mgrid[0:checkerboard_size[0], 0:checkerboard_size[1]].T.
↪reshape(-1, 2)

    # Find the chessboard corners
    cornersFound, corners = cv2.findChessboardCorners(image=gray_img,␣
↪patternSize=checkerboard_size, flags=cv2.CALIB_CB_NORMALIZE_IMAGE + cv2.
↪CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK)

    if cornersFound == True:
        # Refine the corners
        corners_refined = cv2.cornerSubPix(gray_img, corners, (11,11), (-1,-1),␣
↪criteria)
        objects_points.append(objp)
        image_points.append(corners_refined)

        # Draw the corners on the image
        corners_img = cv2.drawChessboardCorners(img, checkerboard_size,␣
↪corners_refined, cornersFound)
        plt.imsave("chess_corners/" + str(image_name.split('\\')[-1].split('.
↪')[0] + '.png'), corners_img)
        # plt.imshow(corners_img)
        # plt.show()
    else:
        print("Corners not found for image: ", image_name)
```

```
100%|      | 25/25 [00:20<00:00,  1.23it/s]
```

5 Q2 Part-1

(10 points) Compute the chessboard plane normals nLi, i ∈ {1, . . . , 25} and the corresponding offsets di , i ∈ {1, . . . , 25} using the planar LIDAR points in each of the pcd files. You can estimate these by making use of singular value decomposition.
Load the Lidar plane normal and offsets

[ ]:
```
# load the lidar normals and offsets
lidar_plane_normals = []
lidar_plane_offsets = []

# read all the lidar scans and segment the plane
for lidar in lidar_scans:
    pcd = o3d.io.read_point_cloud(lidar)
    plane_model, inliers = pcd.segment_plane(distance_threshold=0.01,
                                             ransac_n=3,
```

```
                                            num_iterations=1000)
    # extract the plane normal and offset
    normal = plane_model[:3]
    offset = plane_model[3]
    lidar_plane_normals.append(normal)
    lidar_plane_offsets.append(offset)
lidar_plane_normals, lidar_plane_offsets = np.array(lidar_plane_normals), np.
 ↪array(lidar_plane_offsets)
```

[ ]:
```
# load the lidar plane normals and offsets
lidar_plane_normals = []
lidar_plane_offsets = []

for lidar in lidar_scans:
    pcd = o3d.io.read_point_cloud(lidar)
    plane_model, _ = pcd.segment_plane(distance_threshold=0.01,
                                            ransac_n=3,
                                            num_iterations=1000)
    # Load the lidar points from the point cloud
    lidar_pts = np.asarray(pcd.points)
    # Estimate the center of the all the lidar points
    center = np.mean(lidar_pts, axis=0)
    centered_points = lidar_pts - center

    # Perform SVD on the shifted lidar points to estimate the plane normal
    U, S, V = np.linalg.svd(centered_points)

    # Estimate plane normal
    normal = V[-1, :]
    # Estimate offset of the plane
    offset = np.dot(normal, center)
    lidar_plane_normals.append(normal)
    lidar_plane_offsets.append(offset)
lidar_plane_normals, lidar_plane_offsets = np.array(lidar_plane_normals), np.
 ↪array(lidar_plane_offsets)
```

[ ]:
```
for i in range(len(lidar_plane_normals)):
    print('Image: ', camera_images[i].split('\\')[-1])
    print('Lidar Plane Normal:', lidar_plane_normals[i], ', Offset: ',␣
 ↪lidar_plane_offsets[i], '\n')
```

```
Image:  frame_1430.jpeg
Lidar Plane Normal: [ 0.93304343 -0.24514621  0.26331215] , Offset:
-5.677121251221289

Image:  frame_1153.jpeg
Lidar Plane Normal: [ 0.83314866 -0.55225269  0.02966954] , Offset:
-5.218196142268216
```

```
Image:  frame_256.jpeg
Lidar Plane Normal: [ 0.90010289  0.39945287 -0.17393159] , Offset:
-8.136621127295427

Image:  frame_1093.jpeg
Lidar Plane Normal: [ 0.93943283 -0.22709611  0.25669694] , Offset:
-5.2091459751568125

Image:  frame_795.jpeg
Lidar Plane Normal: [-0.66359359  0.74415079 -0.07670168] , Offset:
7.751094872928159

Image:  frame_339.jpeg
Lidar Plane Normal: [ 0.89254813  0.27159693 -0.35999019] , Offset:
-7.657530287368455

Image:  frame_1850.jpeg
Lidar Plane Normal: [ 0.90902262 -0.30337832 -0.28572622] , Offset:
-8.028105909848696

Image:  frame_1163.jpeg
Lidar Plane Normal: [ 0.91681411 -0.365512    0.16078824] , Offset:
-5.680449173100984

Image:  frame_1580.jpeg
Lidar Plane Normal: [ 0.75784863 -0.56451352  0.32709316] , Offset:
-4.95330327312476

Image:  frame_3329.jpeg
Lidar Plane Normal: [ 0.73647692 -0.57159433 -0.36177572] , Offset:
-3.9660331329322425

Image:  frame_2800.jpeg
Lidar Plane Normal: [ 0.76358121 -0.64566627 -0.00766891] , Offset:
-6.982664452537301

Image:  frame_690.jpeg
Lidar Plane Normal: [ 0.99387179 -0.10977532  0.01297095] , Offset:
-8.858215503763514

Image:  frame_2030.jpeg
Lidar Plane Normal: [0.94771907 0.13432312 0.28945787] , Offset:
-9.487076665625487

Image:  frame_1638.jpeg
Lidar Plane Normal: [-0.4756374   0.76281511  0.43804357] , Offset:
5.271755320218852
```

```
Image:  frame_607.jpeg
Lidar Plane Normal: [ 0.9911679  -0.12728147 -0.03722399] , Offset:
-7.3158333255124965

Image:  frame_1558.jpeg
Lidar Plane Normal: [-0.60458291  0.77393509 -0.18842502] , Offset:
5.064521965089079

Image:  frame_2771.jpeg
Lidar Plane Normal: [-0.43596404  0.89563057 -0.08821134] , Offset:
6.606344143256976

Image:  frame_2717.jpeg
Lidar Plane Normal: [ 0.96185972 -0.01844703 -0.27292049] , Offset:
-8.760640974469617

Image:  frame_1816.jpeg
Lidar Plane Normal: [ 0.87306275 -0.42511073  0.23883532] , Offset:
-7.416036031729739

Image:  frame_1075.jpeg
Lidar Plane Normal: [-0.70066152  0.70140708 -0.13077287] , Offset:
4.714348163592222

Image:  frame_595.jpeg
Lidar Plane Normal: [ 0.93339537 -0.18945843 -0.30475989] , Offset:
-6.26644696326236

Image:  frame_2822.jpeg
Lidar Plane Normal: [-0.62090946  0.78130396 -0.06352603] , Offset:
6.595072218920993

Image:  frame_1139.jpeg
Lidar Plane Normal: [ 0.72145807 -0.69044482 -0.05276548] , Offset:
-4.782886433487035

Image:  frame_812.jpeg
Lidar Plane Normal: [ 0.73832964 -0.67439621  0.00768762] , Offset:
-6.982981083928964

Image:  frame_2991.jpeg
Lidar Plane Normal: [-0.6018589   0.79375047 -0.08789801] , Offset:
3.696370753500789
```

6 Q2 Part-2

(10 points) Now that you have the plane normals nCi and nLi in camera and LIDAR frame of reference respectively for all the selected images, derive the set of equations that you would use to solve for estimating the transformation CTL = [CRL|CtL], i.e. the transformation from the LIDAR frame to the camera frame. Explain how the rotation and translation matrices are calculated. [Hint: You may refer to this thesis (Sec. 5) for deriving the necessary equations.]

7 Q2 Part-3

(5 points) Using the above equations, implement the function that estimates the transformation CTL. Recall that the rotation matrix has determinant +1.

[ ]:
```
# Compute the camera plane normals and offsets from the camera parameters for␣
 ↪each image
camera_plane_normals = []
camera_plane_offsets = []

for img in camera_coord_param:
    # camera_normal, rotation_matrix, rotation_vector, translation_vector
    nCi, RCi, rCi, tCi = load_camera_coord_param_for_image(img)
    orientation_ci = nCi            # Orientation ( c,i)
    alpha_ci = np.dot(rCi, tCi)     # Distance ( c,i)
    camera_plane_normals.append(orientation_ci)
    camera_plane_offsets.append(alpha_ci)

camera_plane_normals = np.array(camera_plane_normals)
camera_plane_offsets = np.array(camera_plane_offsets)
```

[ ]:
```
def get_lidar_to_camera_transform(lidar_plane_normals, lidar_plane_offsets,
                                  camera_plane_normals, camera_plane_offsets):
    mat1 = np.linalg.inv(np.dot(camera_plane_normals.T, camera_plane_normals))
    mat2 = np.dot(camera_plane_normals.T, (camera_plane_offsets -␣
 ↪lidar_plane_offsets))

    # SVD of the lidar plane normals and camera plane normals to estimate the␣
 ↪rotation matrix
    U, S, V_T = np.linalg.svd(np.dot(lidar_plane_normals.T,␣
 ↪camera_plane_normals))

    CtL = np.dot(mat1, mat2)        # Translation vector (t)
    CRL = np.dot(V_T, U.T)          # Rotation matrix (R)

    # Transform matrix from lidar to camera
```

```
    CTL = np.hstack((CRL, CtL.reshape(3, 1)))
    return CTL, CRL, CtL

CTL, CRL, CtL = get_lidar_to_camera_transform(lidar_plane_normals,␣
 ↪lidar_plane_offsets, camera_plane_normals, camera_plane_offsets)
```

[ ]:
```
print('Transform matrix from lidar to camera: \n', CTL)
```

```
Transform matrix from lidar to camera:
 [[ 1.48526094e-01 -1.07468205e-01  9.83051669e-01  1.06036248e+01]
 [-9.88764565e-01  8.20766101e-04  1.49478964e-01 -3.59631412e+00]
 [-1.68710913e-02 -9.94208183e-01 -1.06138847e-01 -1.20750982e+01]]
```

[ ]:
```
print('Rotation matrix from lidar to camera: \n', CRL)
```

```
Rotation matrix from lidar to camera:
 [[ 1.48526094e-01 -1.07468205e-01  9.83051669e-01]
 [-9.88764565e-01  8.20766101e-04  1.49478964e-01]
 [-1.68710913e-02 -9.94208183e-01 -1.06138847e-01]]
```

[ ]:
```
print('Translation vector from lidar to camera: \n', CtL)
```

```
Translation vector from lidar to camera:
 [ 10.60362481  -3.59631412 -12.07509816]
```

[ ]:
```
np.linalg.det(CRL)
```

[ ]:
```
0.9999999999999992
```

8 Q2 Part-4

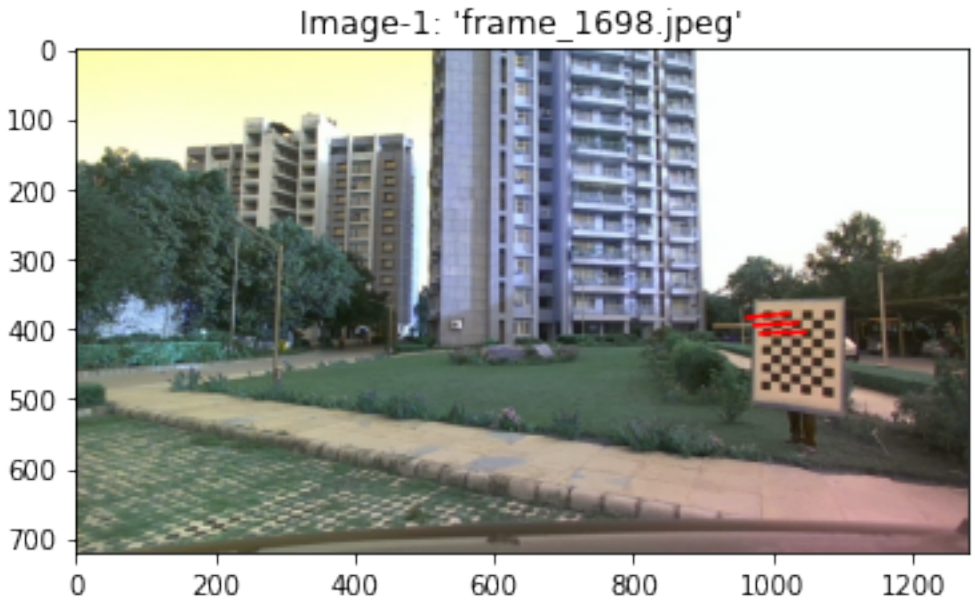
(5 points) Use the estimated transformation CTe L to map LIDAR points to the camera frame of reference, then project them to the image plane using the intrinsic camera parameters. Are all points within the checkerboard pattern's boundary in each image?

[ ]:
```
# Load the Intrinsic and Distortion parameters of the camera
intrinsic_matrix, distCoeff = load_camera_parameters(*camera_param)
camera_intrinsic = np.hstack((intrinsic_matrix, np.zeros((3, 1))))

# Extrinsic parameters of the camera
camera_extrinsic = np.vstack((CTL, np.array([0, 0, 0, 1])))

# Camera projection matrix
camera_projection_matrix = np.dot(camera_intrinsic, camera_extrinsic)
```

[ ]:
```
# compute the image points from the lidar points
camera_image_points = []
lidar_image_points = []
```

```
for lidar in lidar_scans:
    pcd = o3d.io.read_point_cloud(lidar)
    plane_model, inliers = pcd.segment_plane(distance_threshold=0.01,
                                             ransac_n=3,
                                             num_iterations=1000)
    # Load the lidar points from the point cloud
    lidar_pts = np.asarray(pcd.points)
    lidar_pts = np.vstack((lidar_pts.T, np.ones((1, len(lidar_pts))) ))

    # Compute the image points from the lidar points
    camera_pts = np.dot(camera_projection_matrix, lidar_pts)
    camera_pts = camera_pts[:2, :] / camera_pts[2, :]
    camera_image_points.append(camera_pts)
    lidar_image_points.append(lidar_pts)
```

[ ]:
```
for j, image_name in enumerate(camera_images):
    img = cv2.imread(image_name)

    image_points_x = camera_image_points[j][0, :]
    image_points_y = camera_image_points[j][1, :]

    for i in range(image_points_x.shape[0]):
        cv2.circle(img=img, center=(int(image_points_x[i]),␣
 ↪int(image_points_y[i])),
                   radius=1, color=(255, 0, 0), thickness=3)
    plt.imshow(img)
    plt.title(f"Image-{j+1}: '" + image_name.split('\\')[-1] + "'")
    plt.show()
```



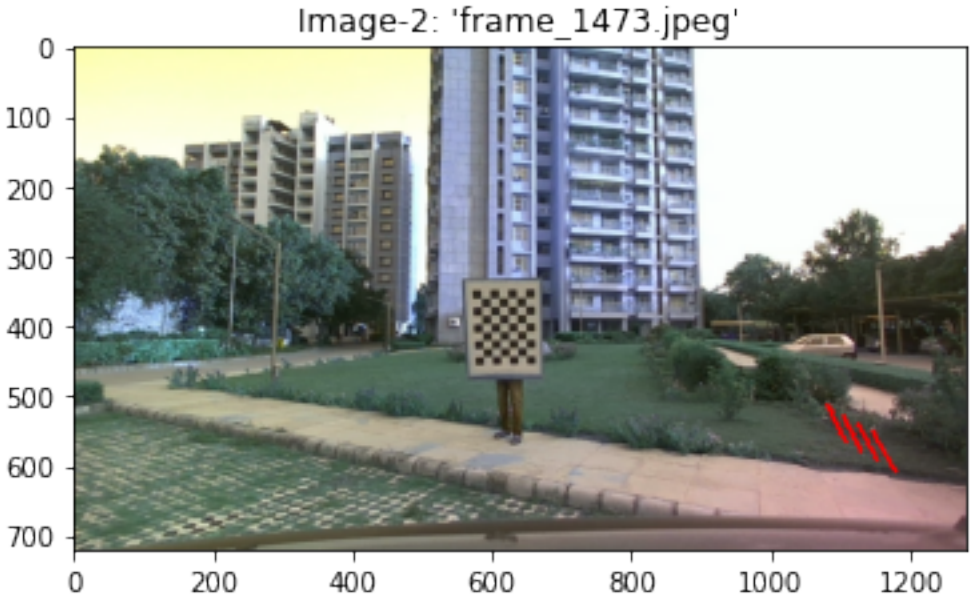


Image-2: 'frame_1473.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

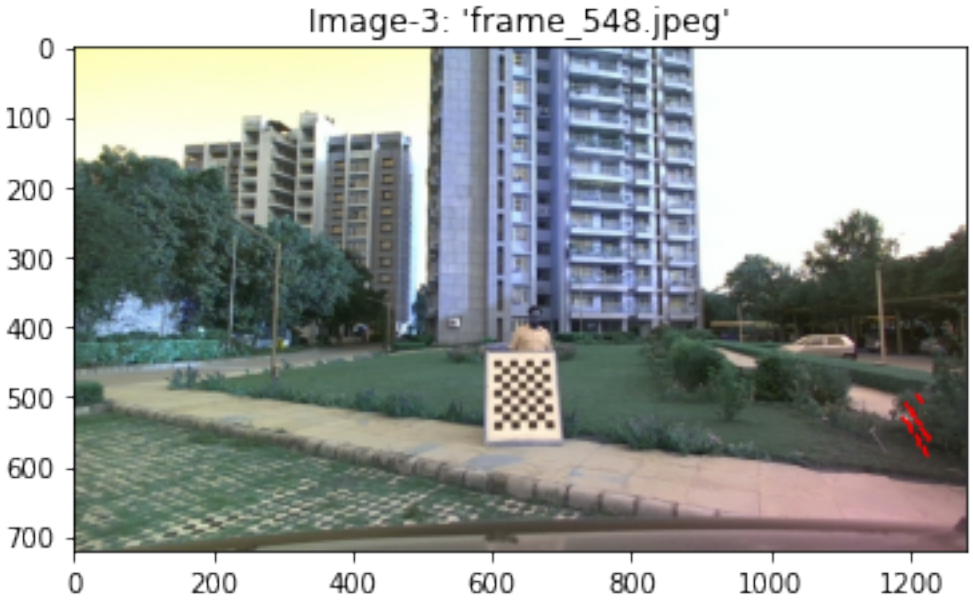


Image-3: 'frame_548.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

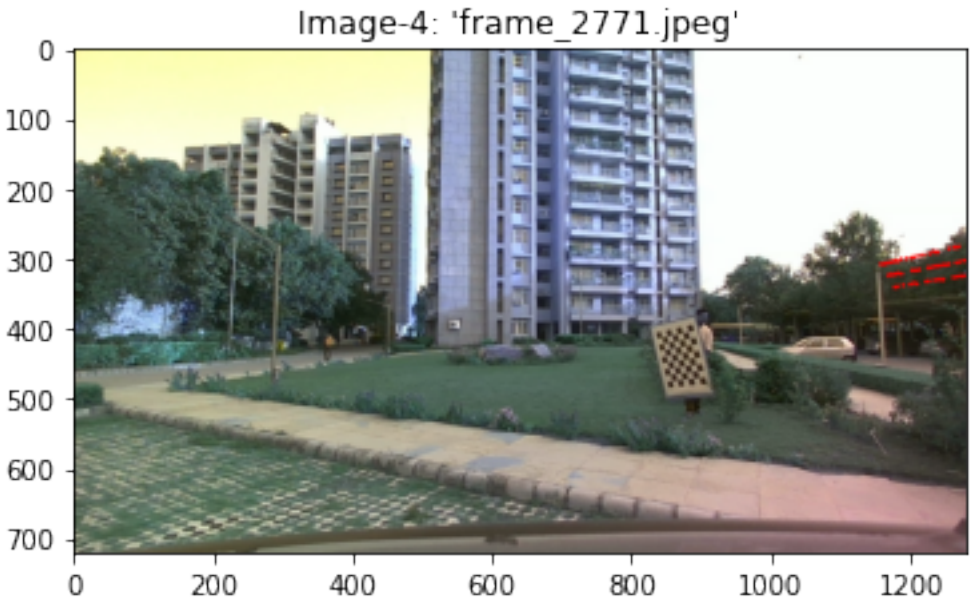


Image-4: 'frame_2771.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

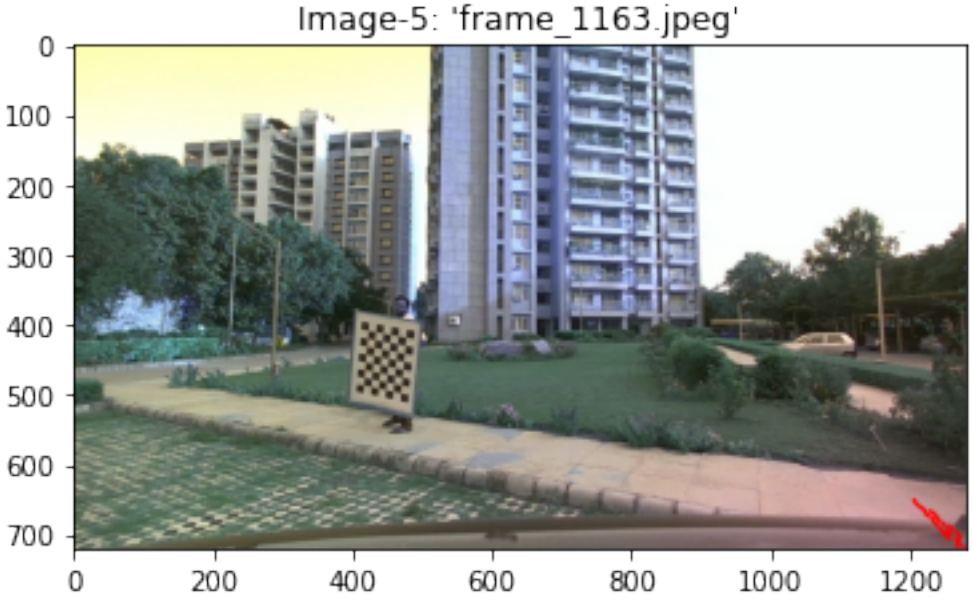


Image-5: 'frame_1163.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

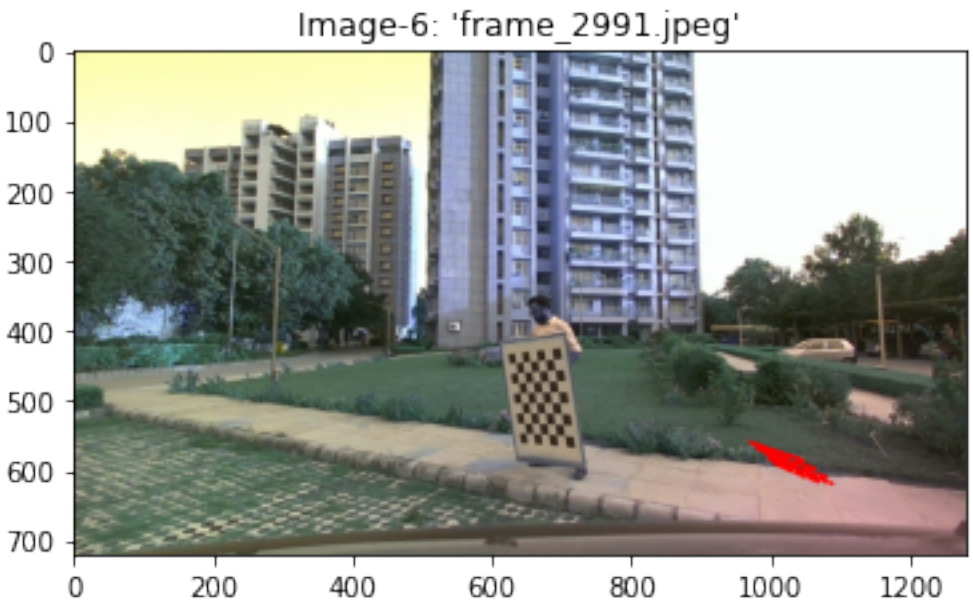


Image-6: 'frame_2991.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

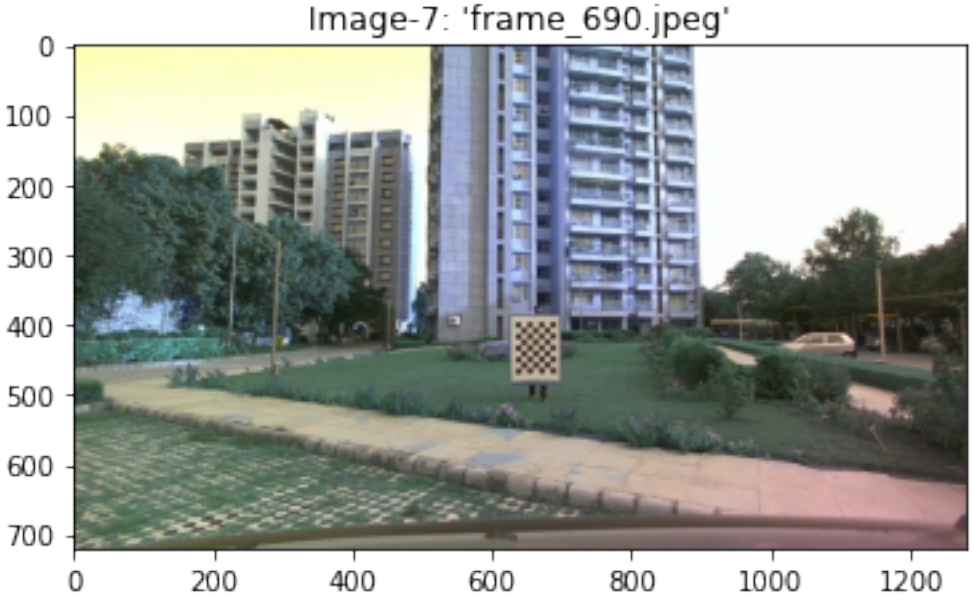


Image-7: 'frame_690.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

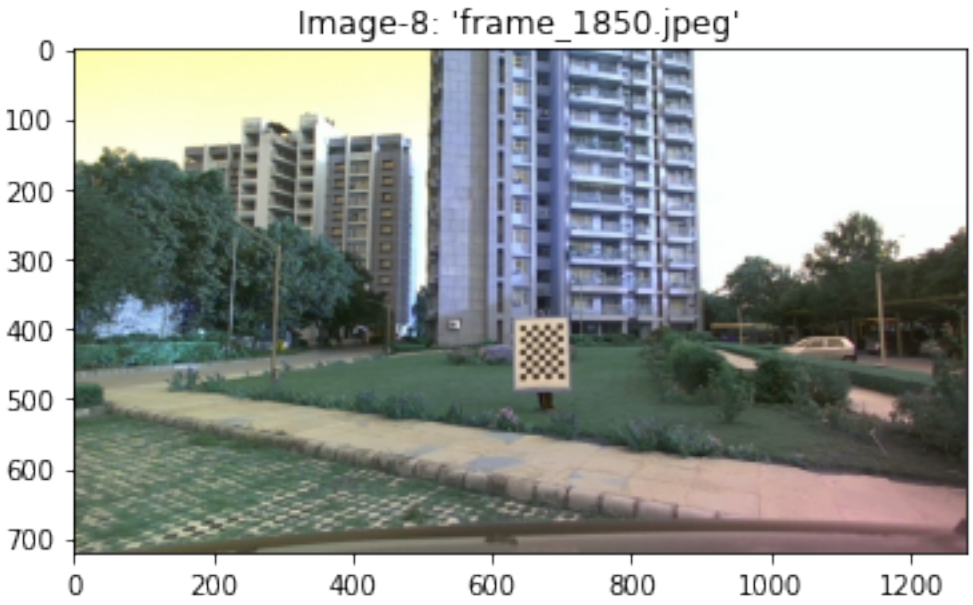


Image-8: 'frame_1850.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

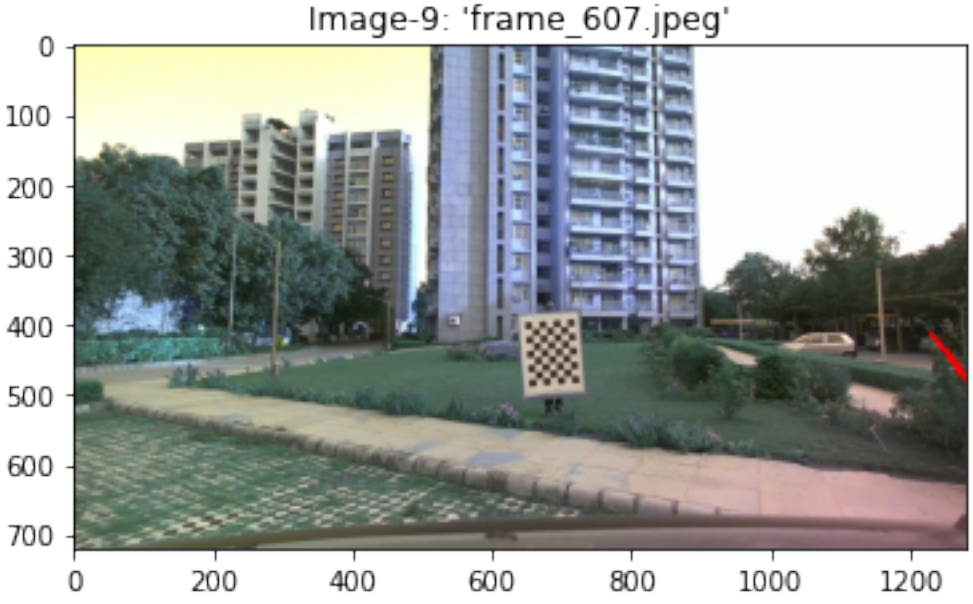


Image-9: 'frame_607.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

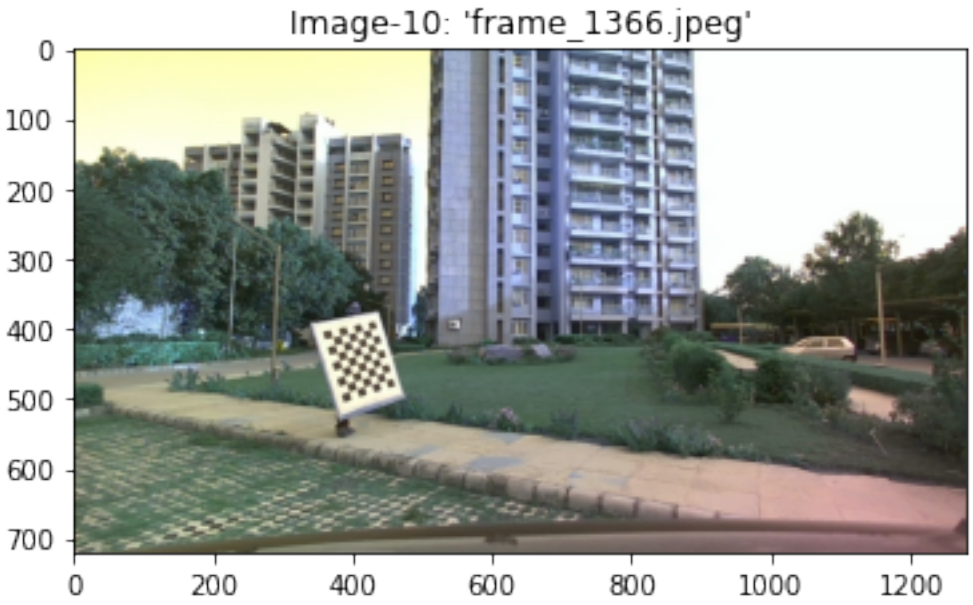


Image-10: 'frame_1366.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

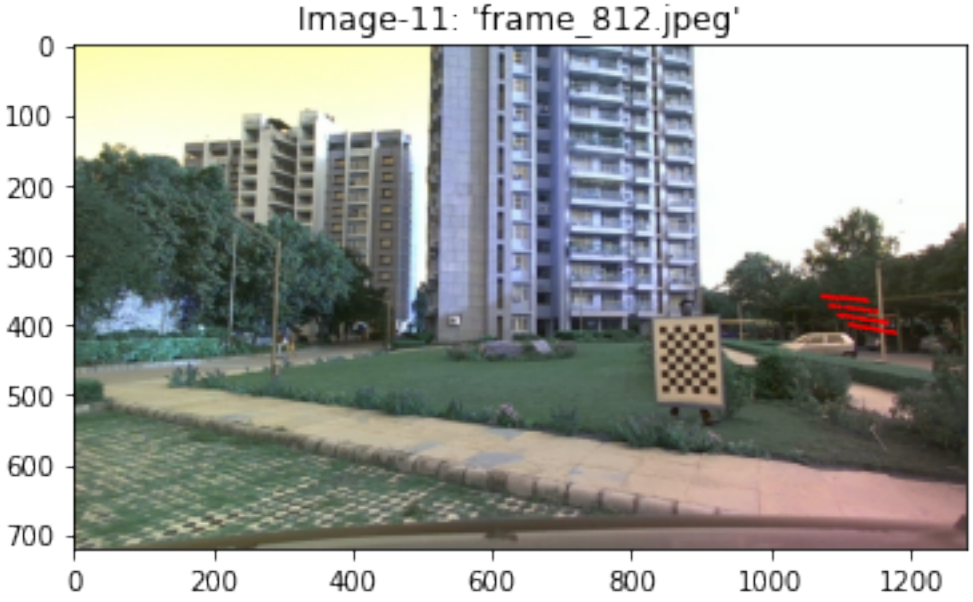


Image-11: 'frame_812.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

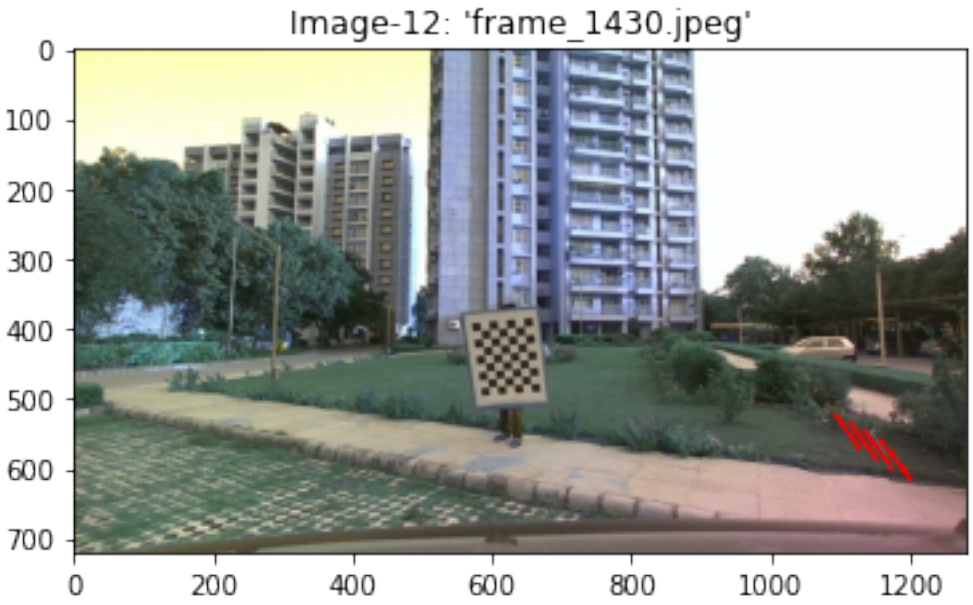


Image-12: 'frame_1430.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

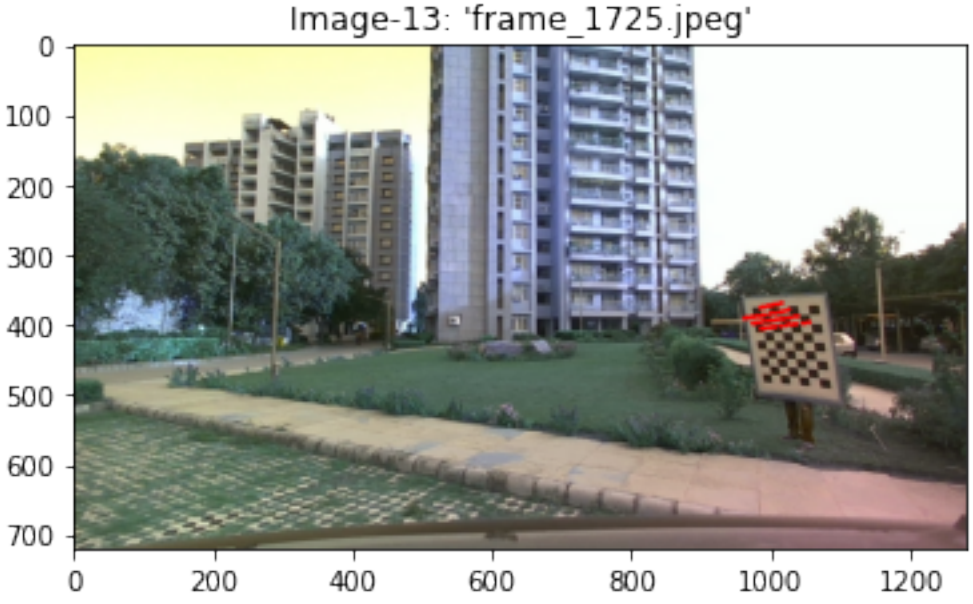


Image-13: 'frame_1725.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Image-14: 'frame_1580.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

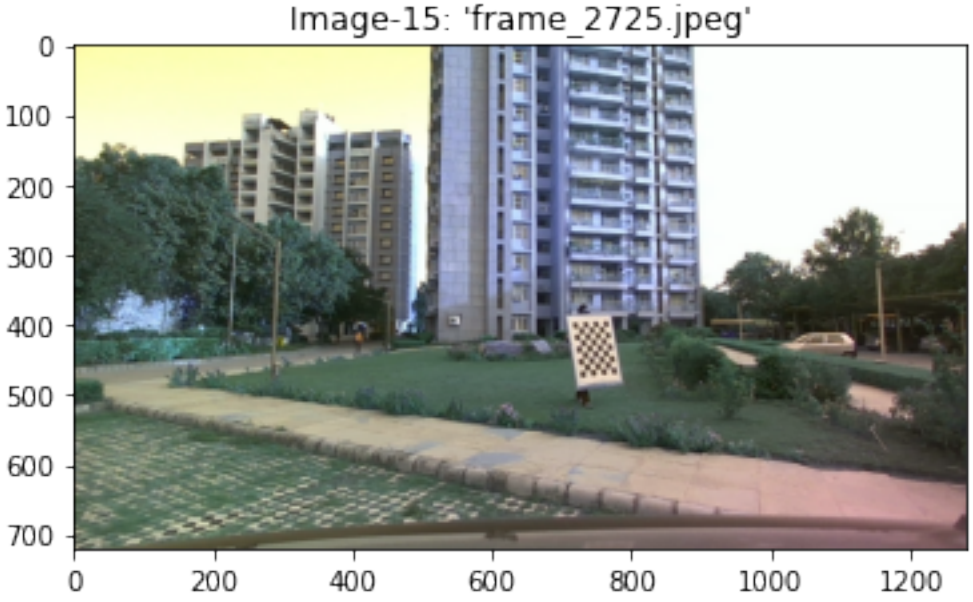


Image-15: 'frame_2725.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

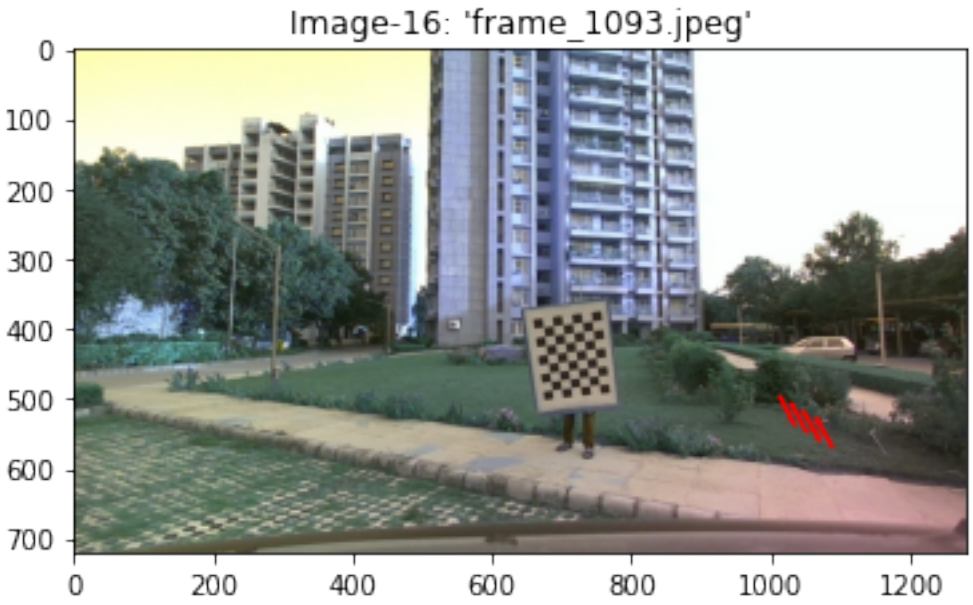


Image-16: 'frame_1093.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

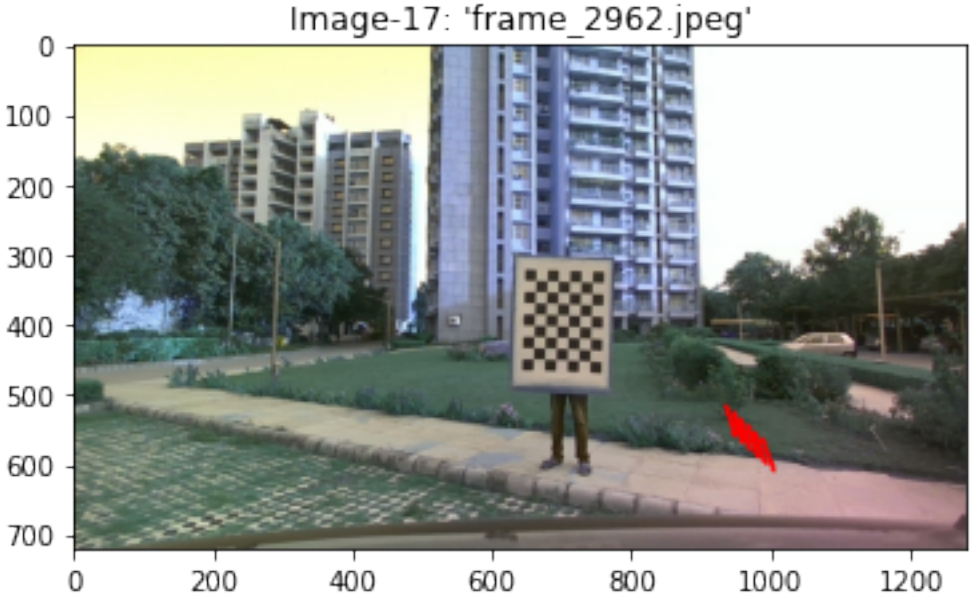


Image-17: 'frame_2962.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

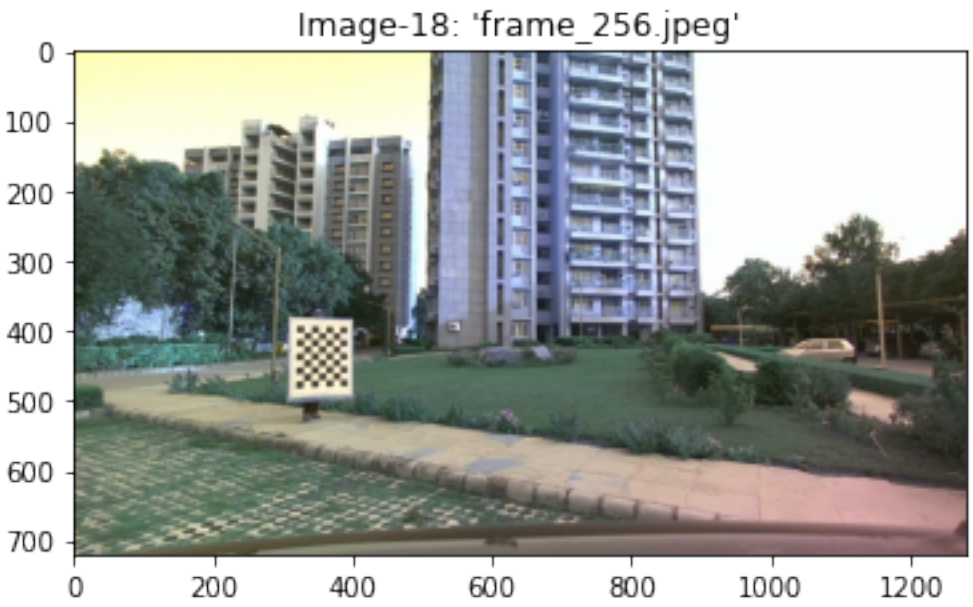


Image-18: 'frame_256.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

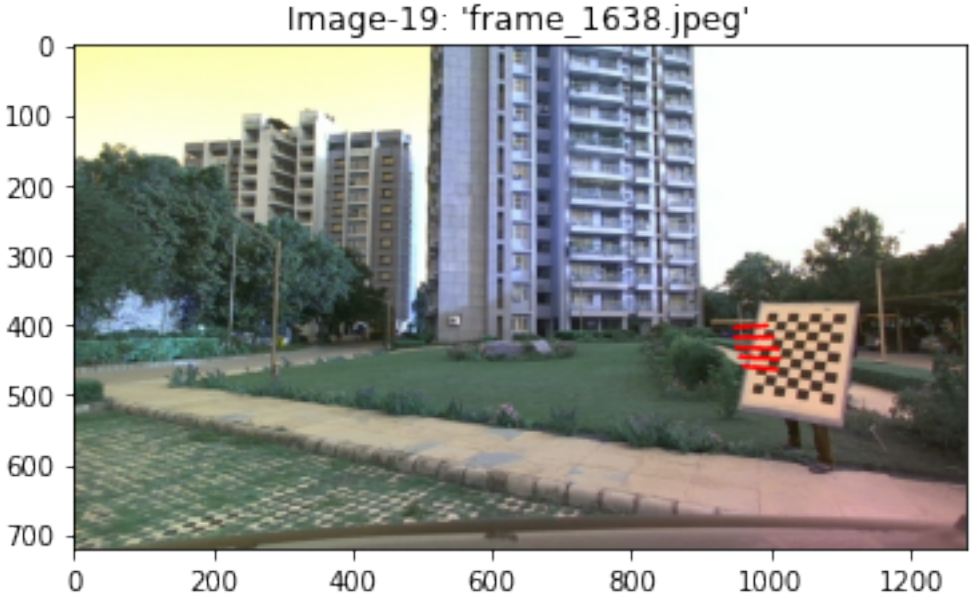


Image-19: 'frame_1638.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

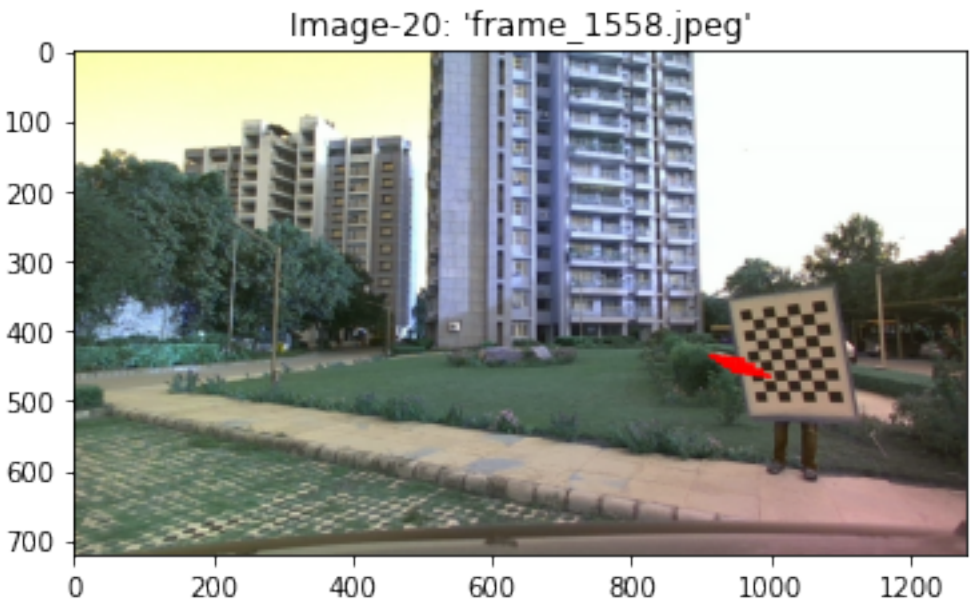


Image-20: 'frame_1558.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

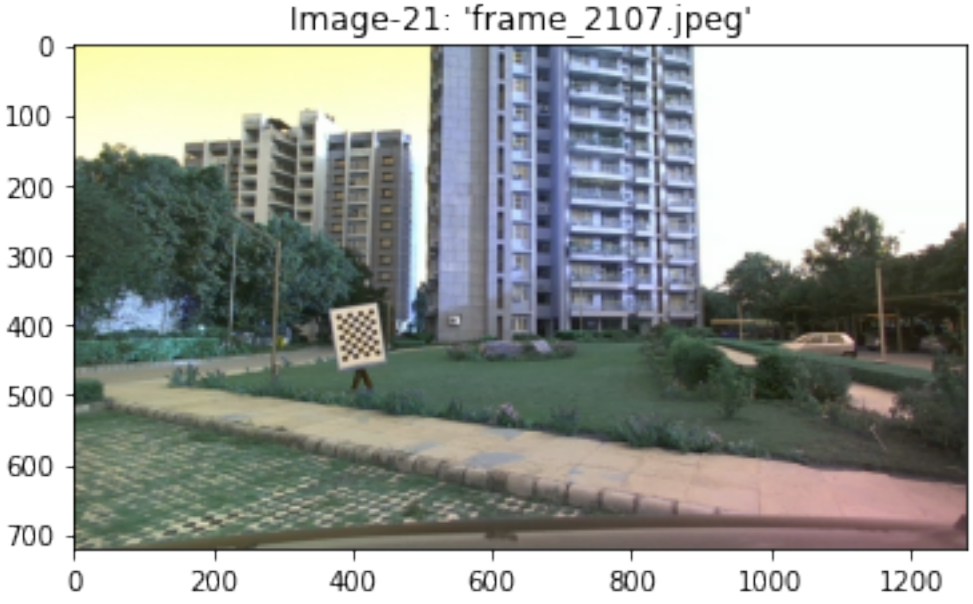


Image-21: 'frame_2107.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

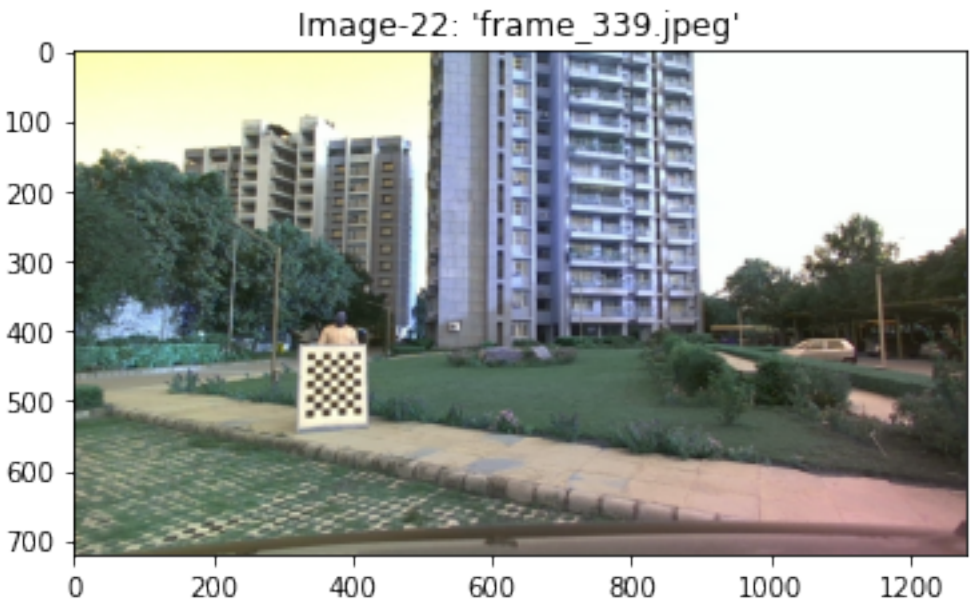


Image-22: 'frame_339.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

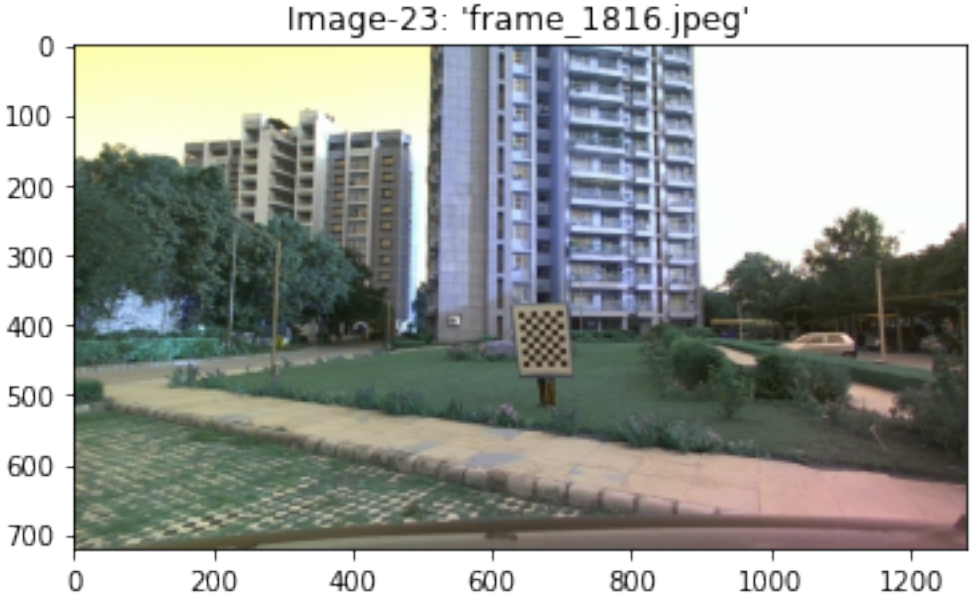
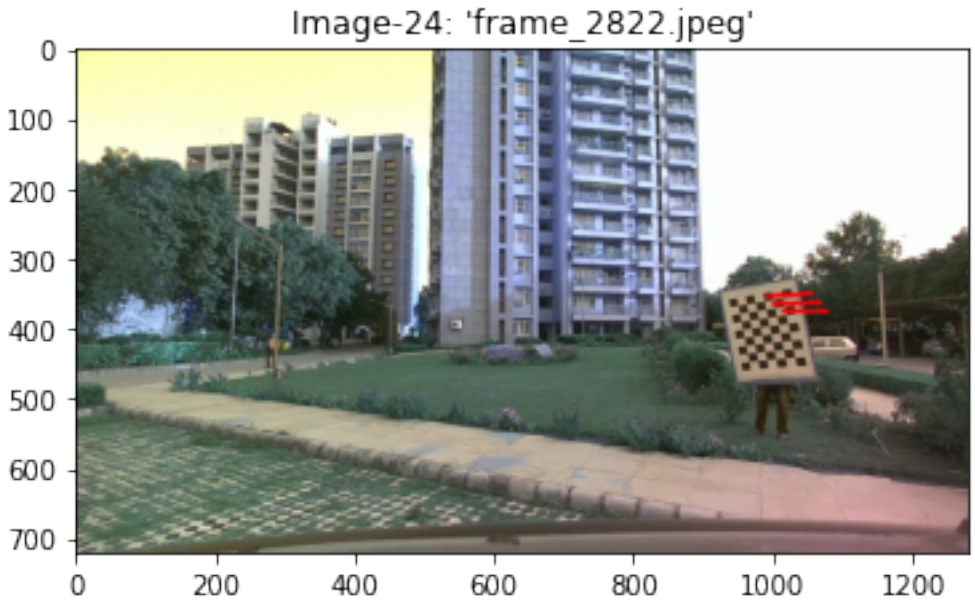
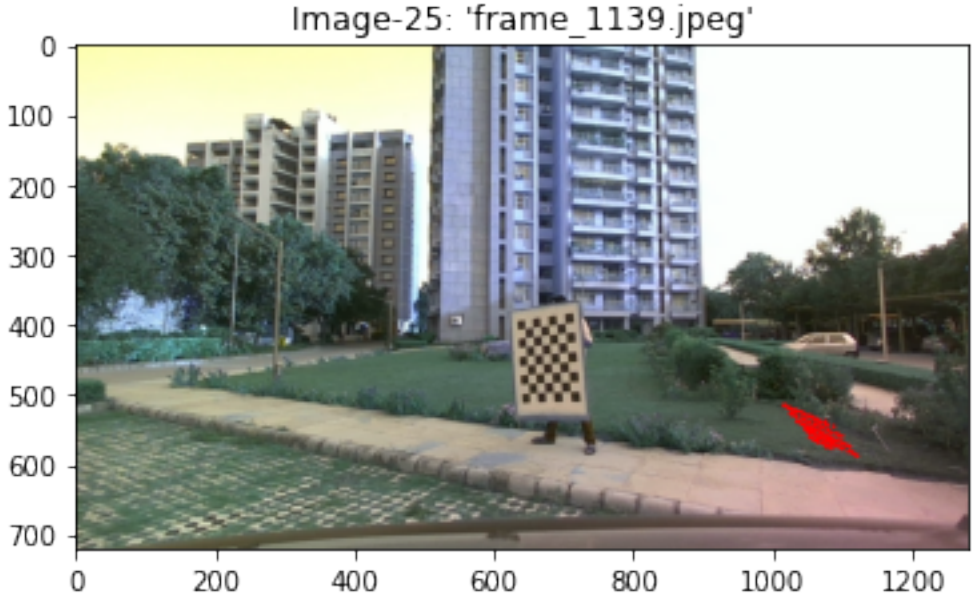


Image-23: 'frame_1816.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

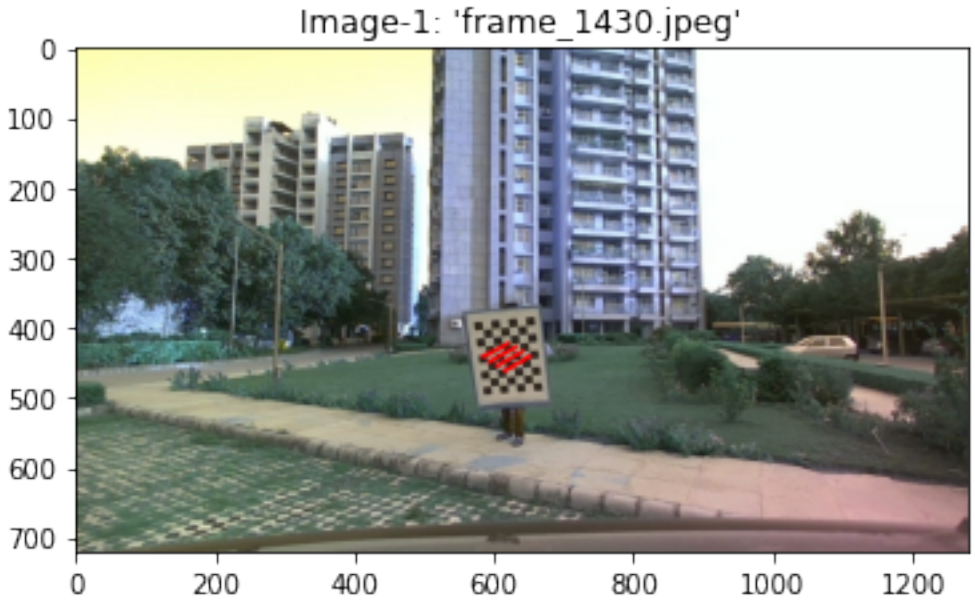




Observation: No all points are not within the checkerboard pattern's boundary in each image

[ ]:
```
for j, image_name in enumerate(camera_images):
    img = cv2.imread(image_name)
    mean_u = np.mean(image_points[j][:,:, 0])
    mean_v = np.mean(image_points[j][:,:, 1])
```

```
    image_points_x = camera_image_points[j][0, :]
    image_points_y = camera_image_points[j][1, :]
    cam_mean_u = np.mean(image_points_x)
    cam_mean_v = np.mean(image_points_y)
    image_points_x = image_points_x + mean_u - cam_mean_u
    image_points_y = image_points_y + mean_v - cam_mean_v

    for i in range(image_points_x.shape[0]):
        cv2.circle(img=img, center=(int(image_points_x[i]),␣
↪int(image_points_y[i])),
                   radius=1, color=(255, 0, 0), thickness=3)
    plt.imshow(img)
    plt.title(f"Image-{j+1}: '" + image_name.split('\\')[-1] + "'")
    plt.show()
```



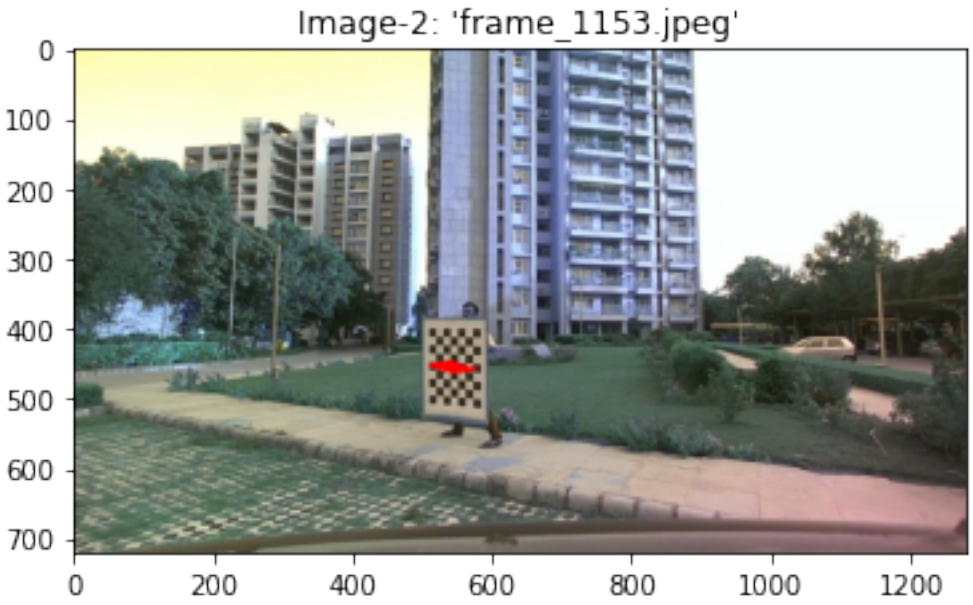


Image-2: 'frame_1153.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

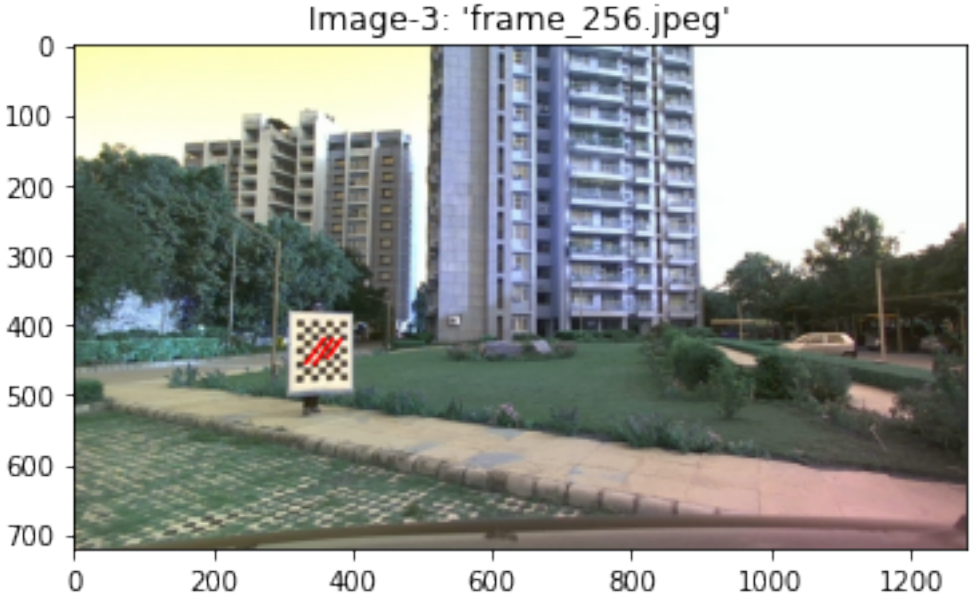


Image-3: 'frame_256.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

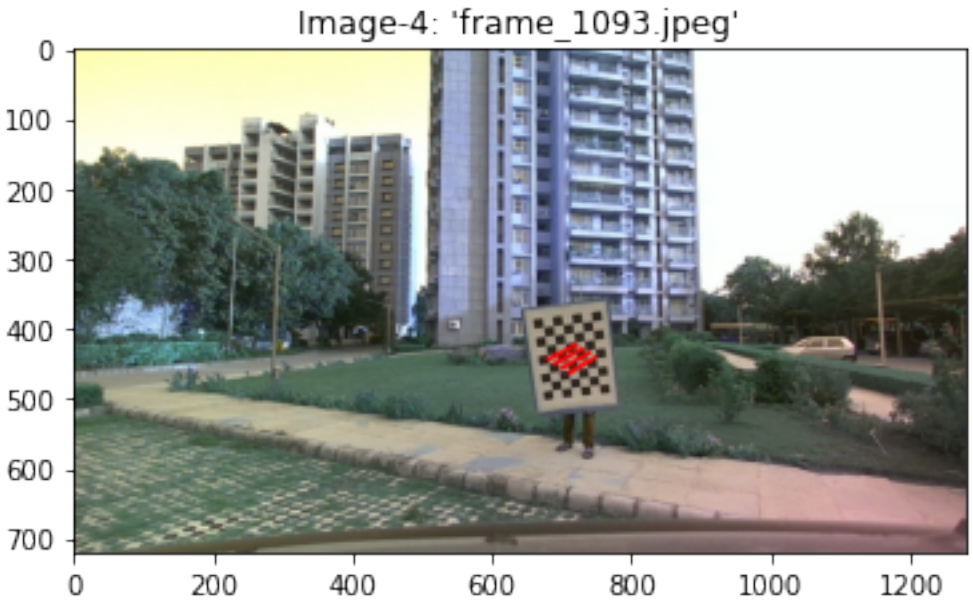


Image-4: 'frame_1093.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

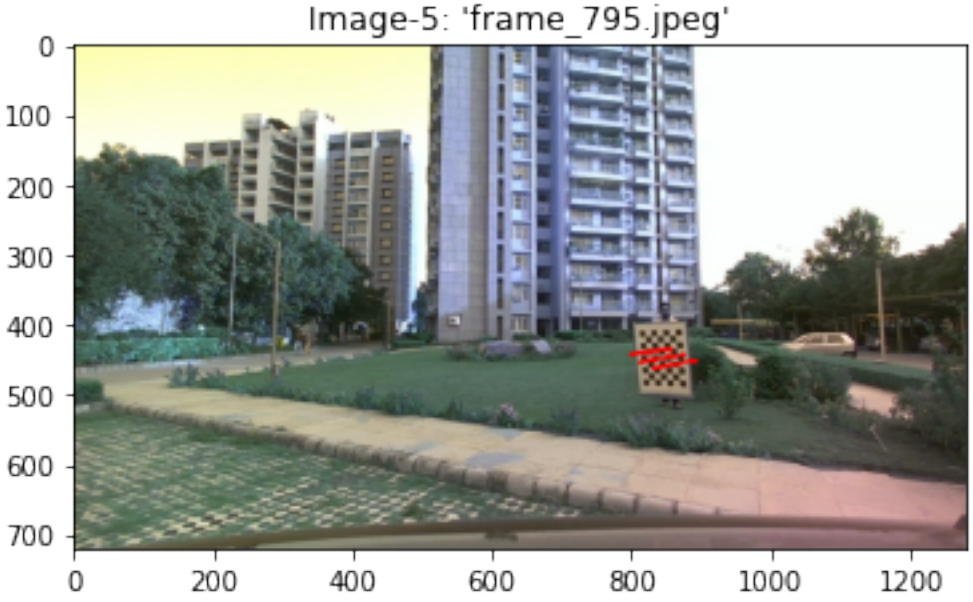


Image-5: 'frame_795.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

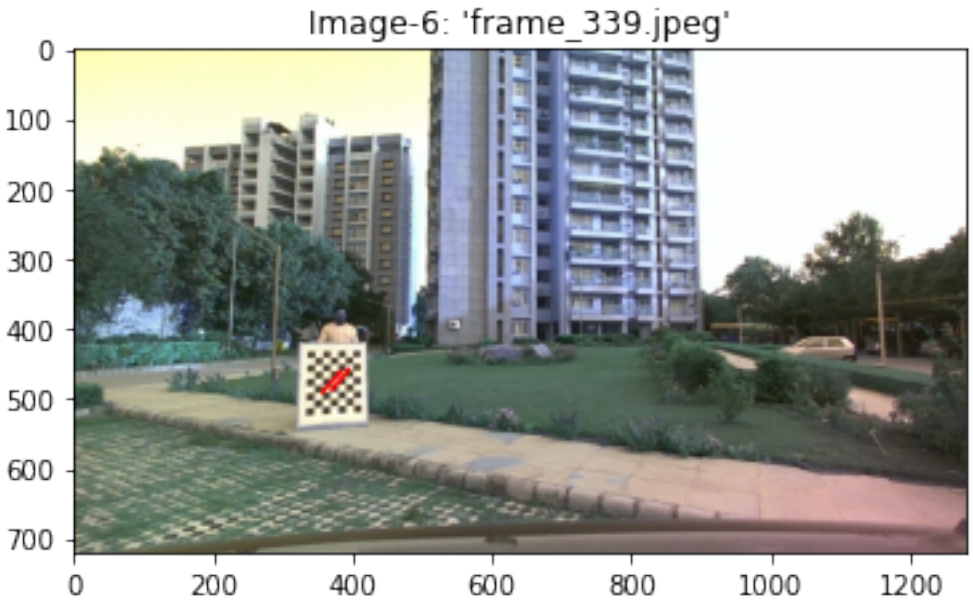


Image-6: 'frame_339.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

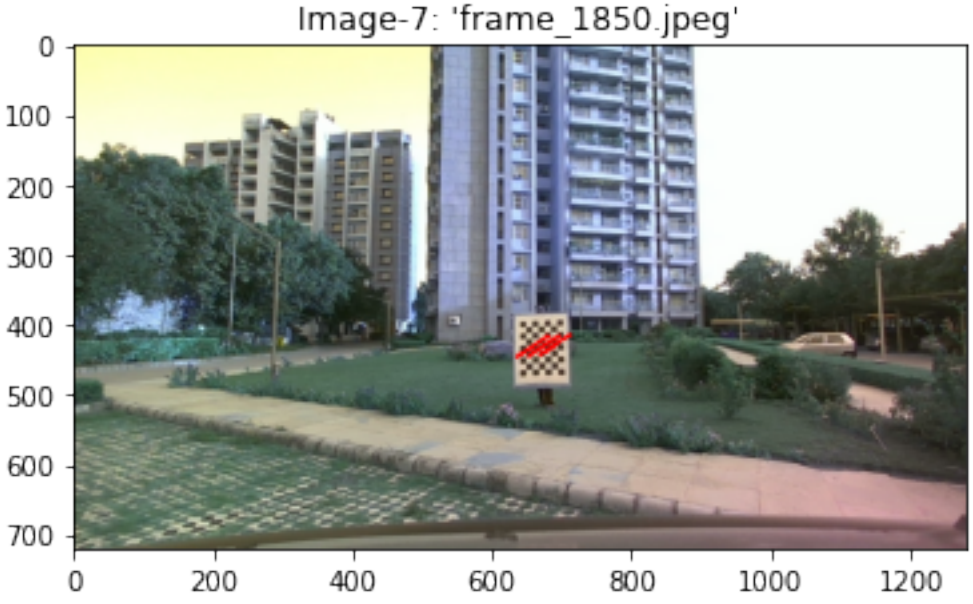


Image-7: 'frame_1850.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

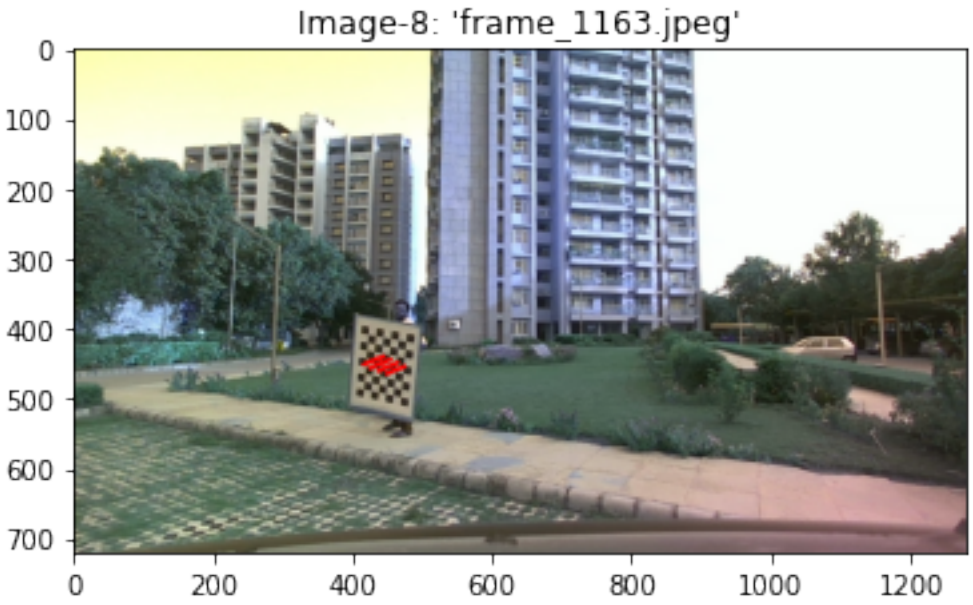


Image-8: 'frame_1163.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

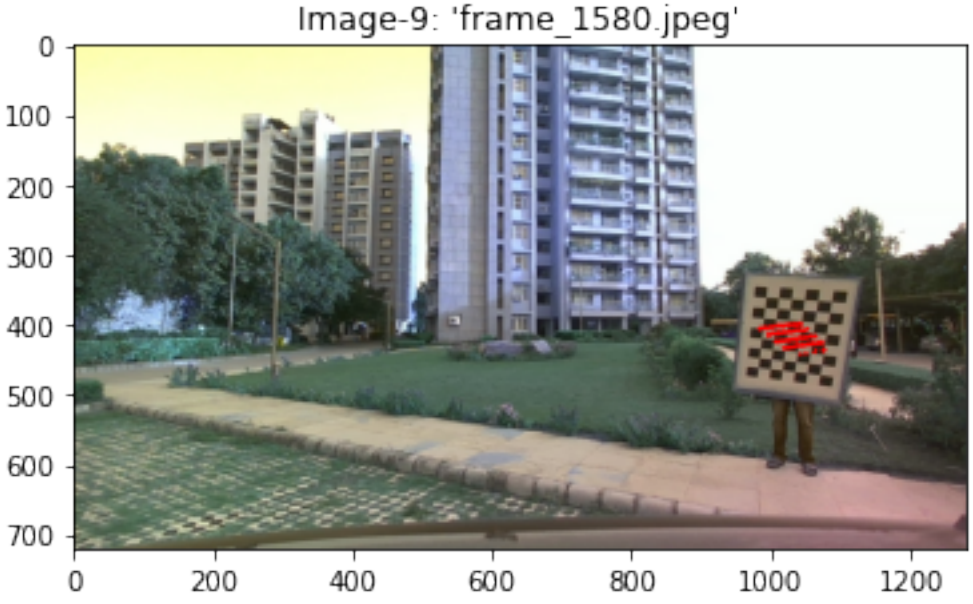


Image-9: 'frame_1580.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

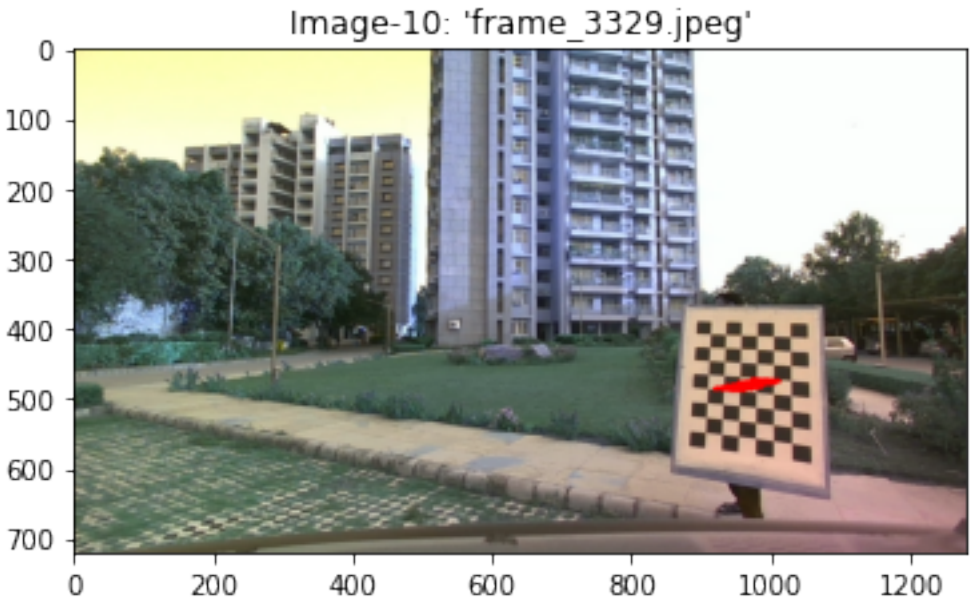


Image-10: 'frame_3329.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

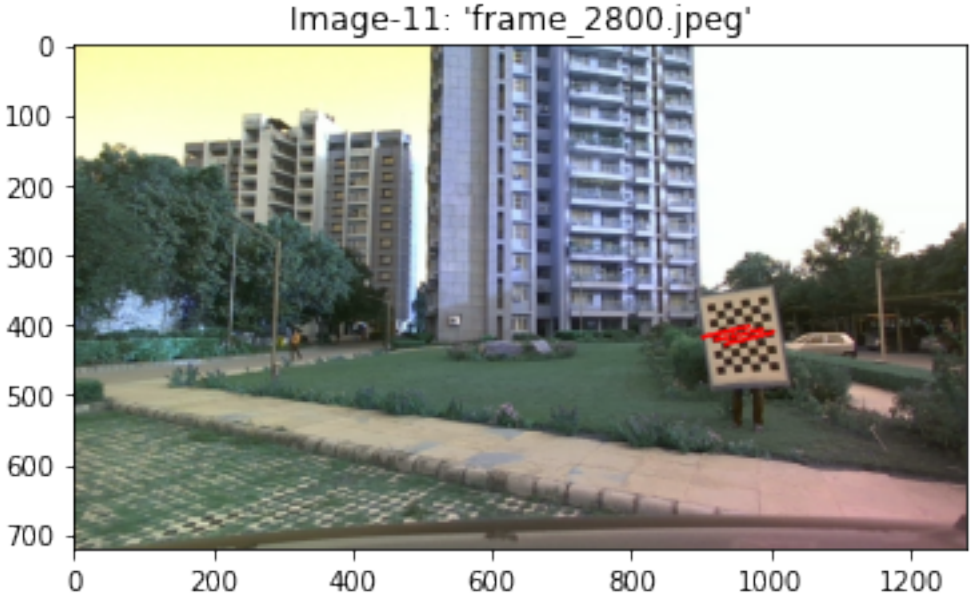


Image-11: 'frame_2800.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

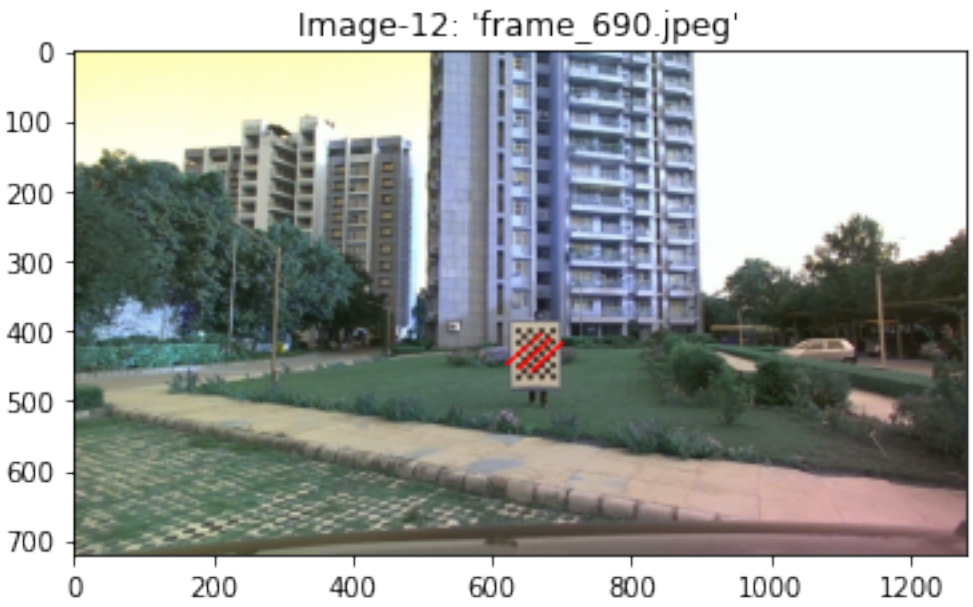


Image-12: 'frame_690.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

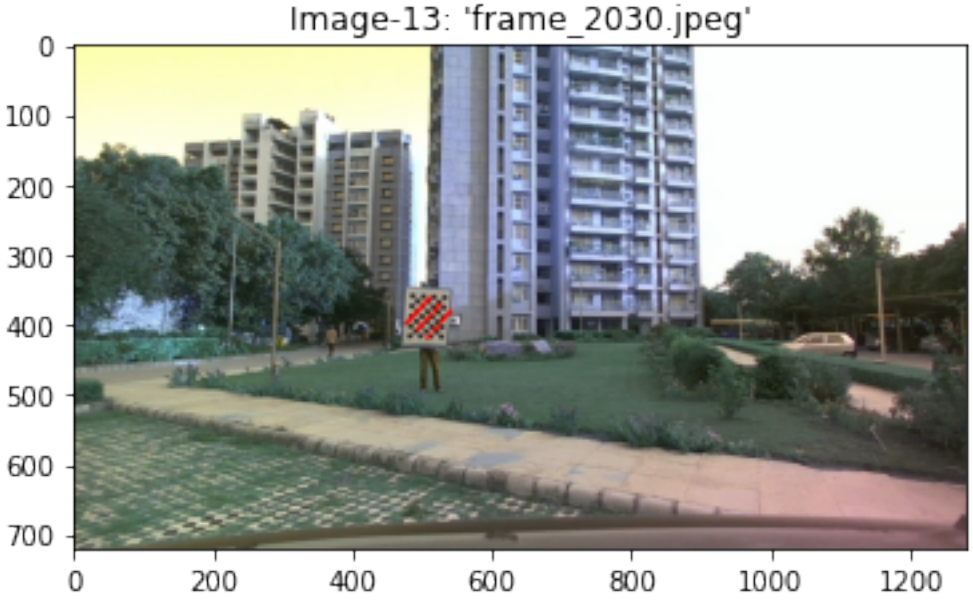


Image-13: 'frame_2030.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

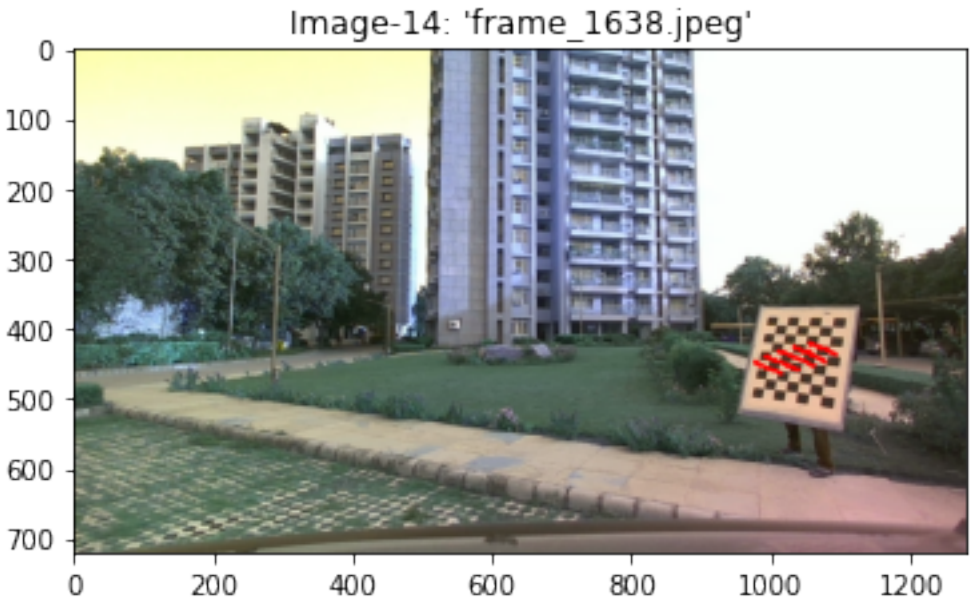


Image-14: 'frame_1638.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

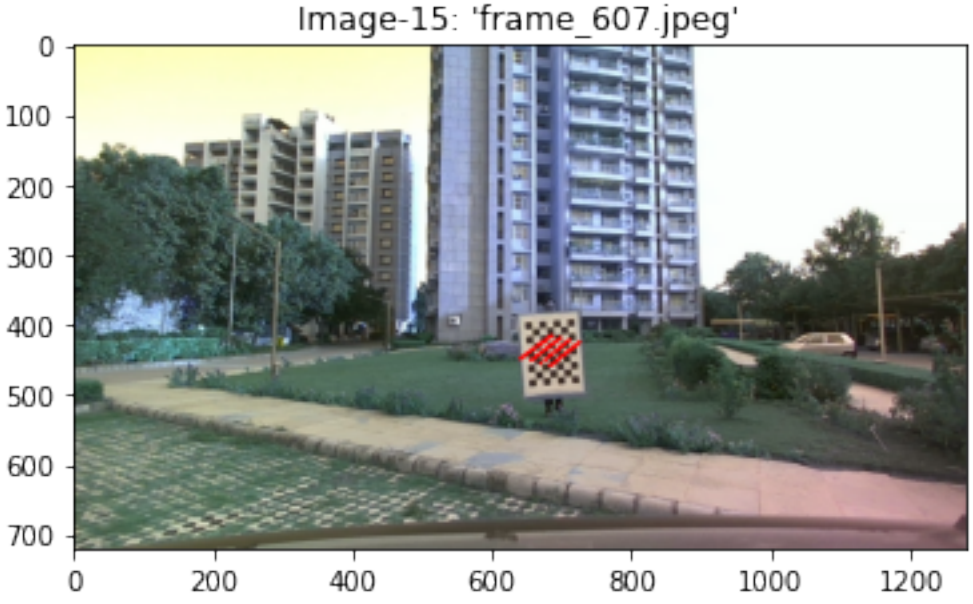


Image-15: 'frame_607.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

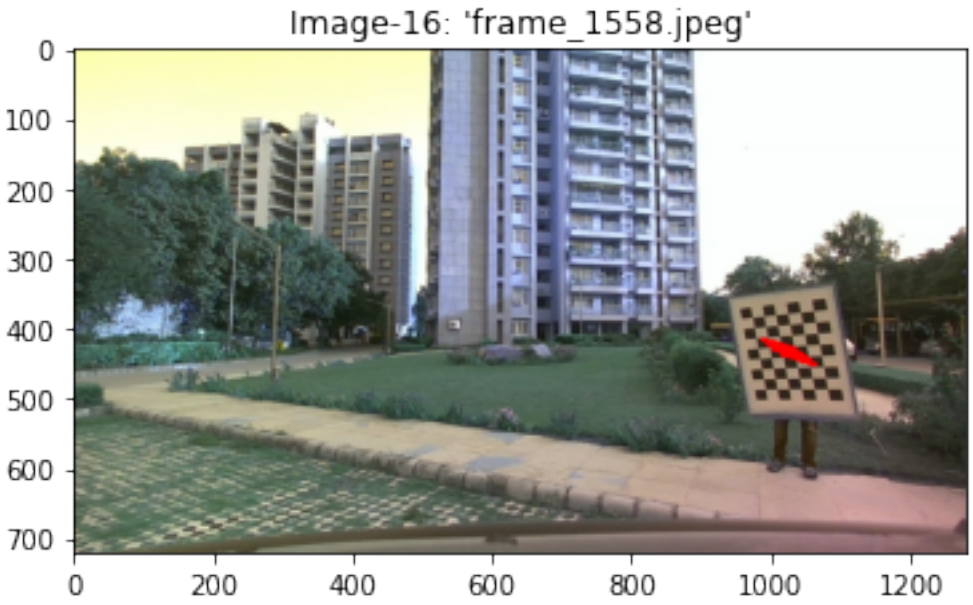


Image-16: 'frame_1558.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Image-17: 'frame_2771.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

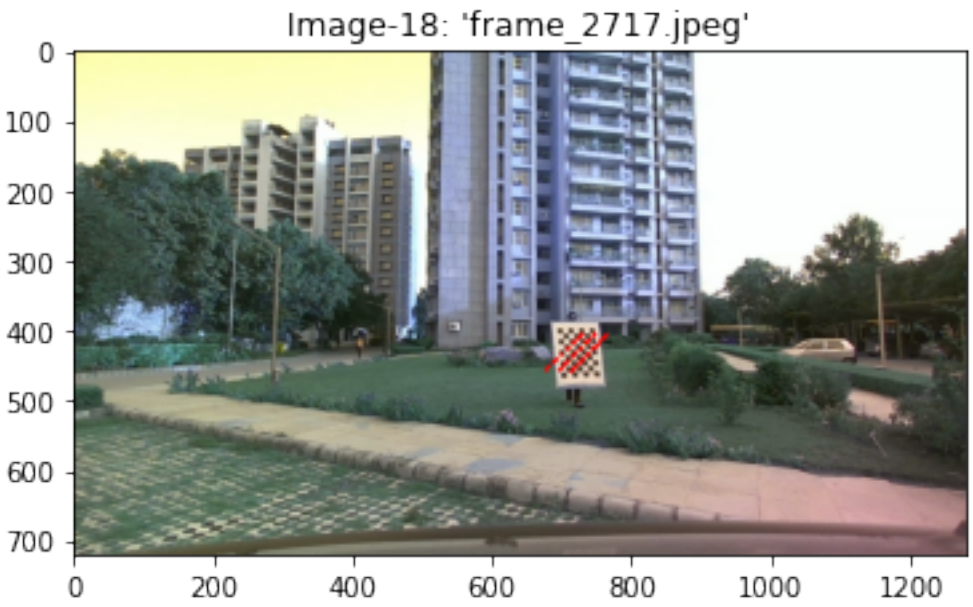


Image-18: 'frame_2717.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

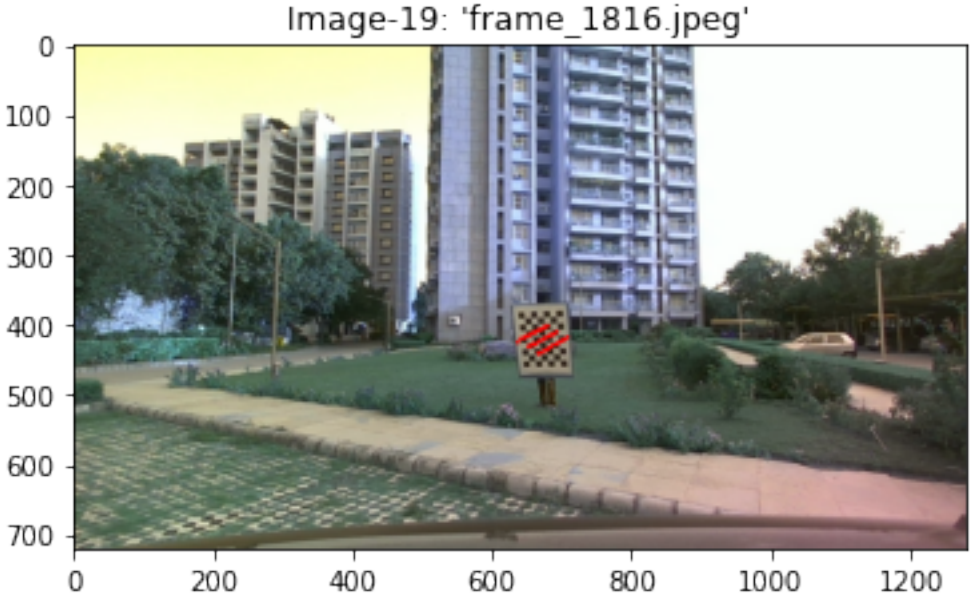


Image-19: 'frame_1816.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

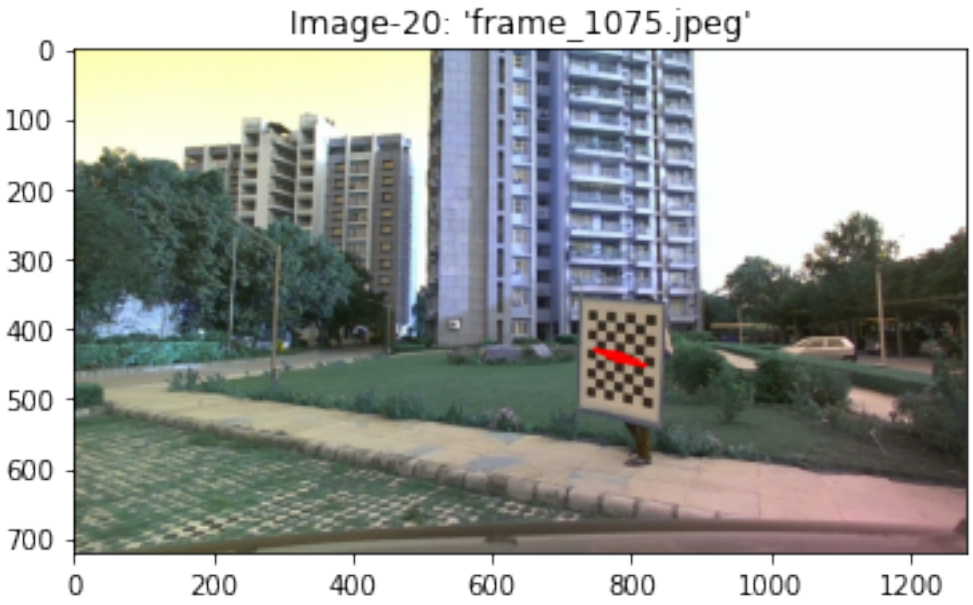


Image-20: 'frame_1075.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

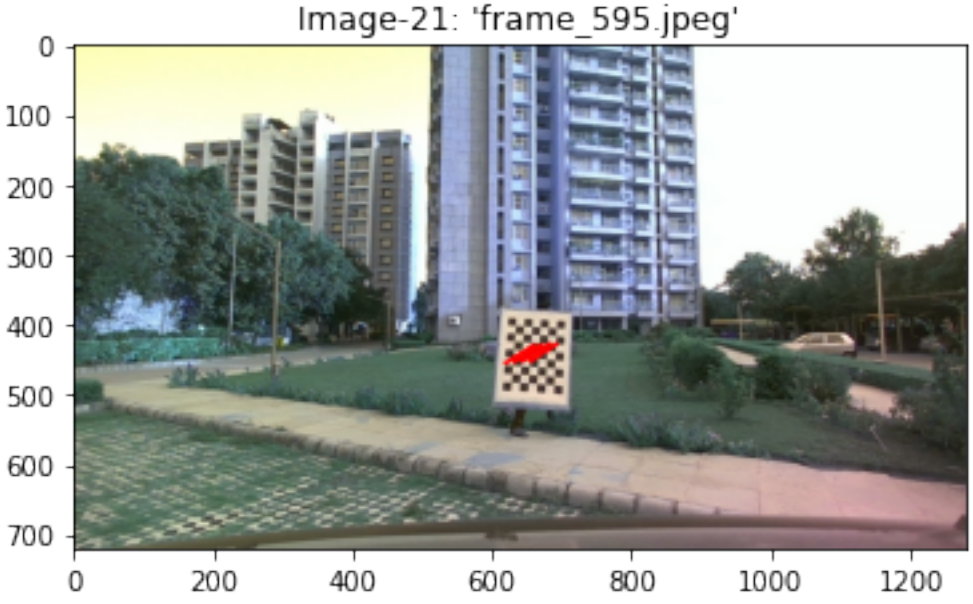


Image-21: 'frame_595.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

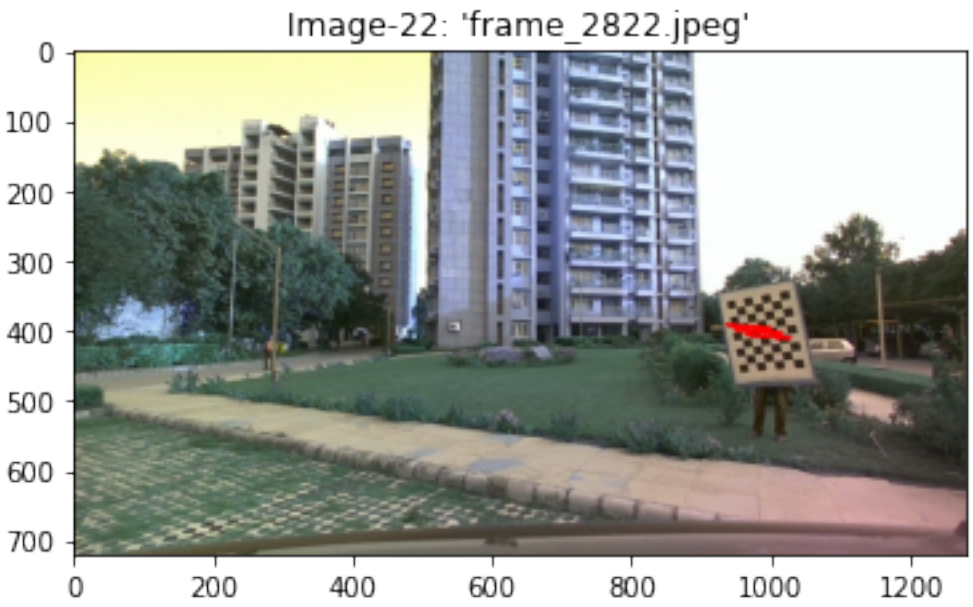


Image-22: 'frame_2822.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

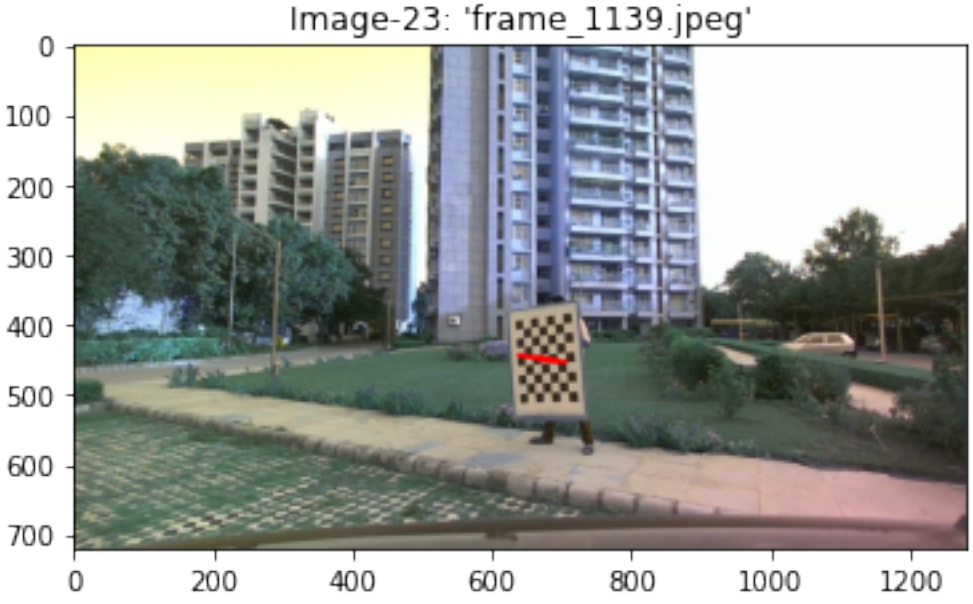
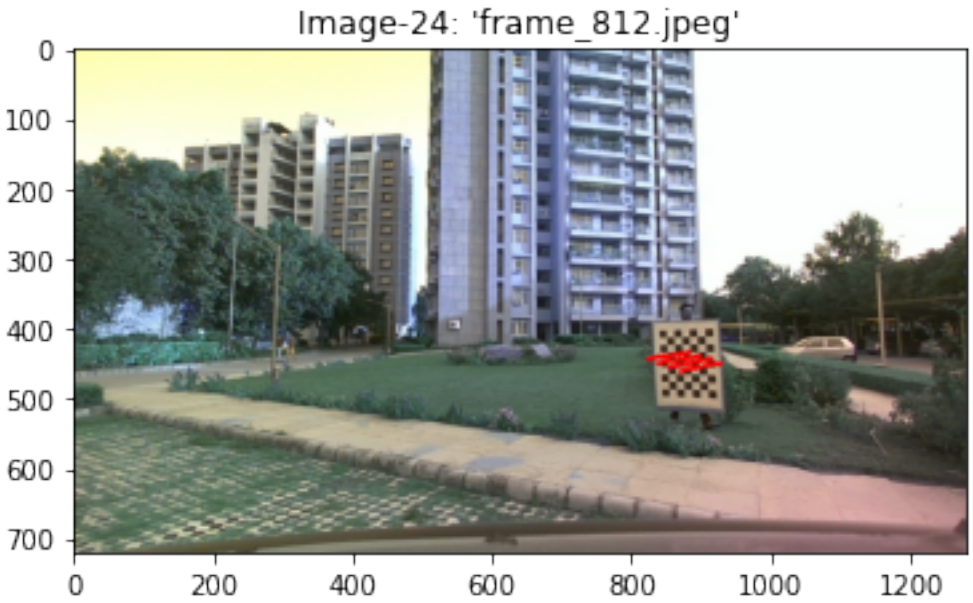
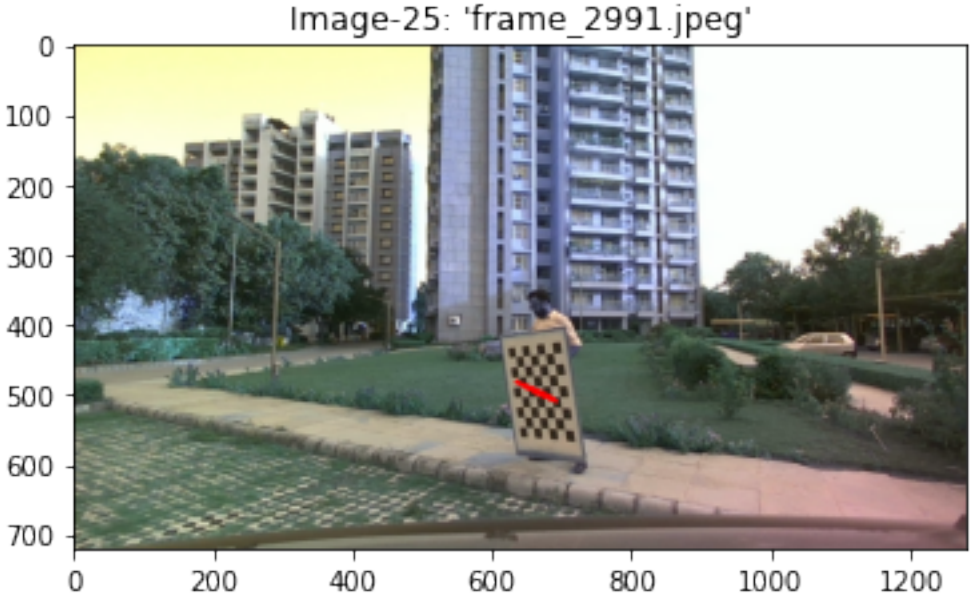


Image-23: 'frame_1139.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200





Observation:

Yes after mean transformation of Lidar points all points are within the checkerboard pattern's boundary in each image

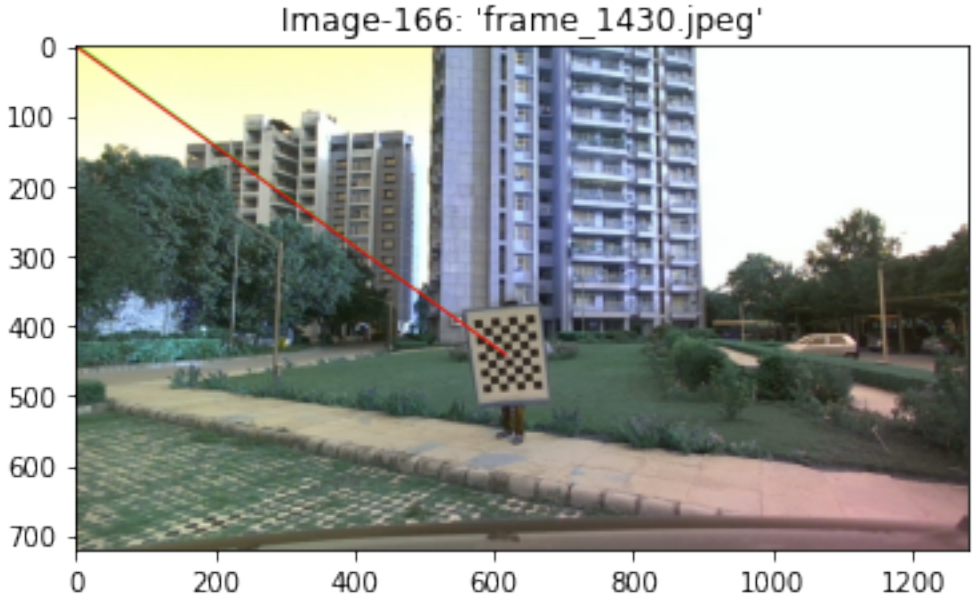
9 Q2 Part-5

(10 points) Plot the normal vectors nL_i, nC_i, CRL_nL_i for any 5 image and LIDAR scan pairs. Compute the cosine distance between the camera normal nCi and the transformed LIDAR normal, CRL_nL_i for all 38 image and LIDAR scan pairs, and plot the histogram of these errors. Report the average error with the standard deviation.

[ ]:
```
def draw_image_normals(image, normal, mean_u, mean_v, color=(0, 0, 255)):
    # Compute the z coordinate of the line joining the mean point and the␣
 ↪normal vector
    z = -(normal[0] * mean_u + normal[1] * mean_v)  / normal[2]

    # Compute the end point of line joining the mean point and the normal vector
    line_end_point = np.array([mean_u + normal[0], mean_v + normal[1],␣
 ↪normal[2] + z, 1])
    line_end_point = np.dot(camera_extrinsic, line_end_point)
    line_end_point = line_end_point[:2] / line_end_point[2]

    # Drawing the line joining the mean point and the normal vector on the image
    image_with_normal = cv2.line(img=image, pt1=(int(mean_u), int(mean_v)),␣
 ↪pt2=(int(line_end_point[0]), int(line_end_point[1])),
                                 color=color, thickness=2, lineType=100, shift=0)
    return image_with_normal
```

[ ]:
```
# Array to store the transformed lidar normals
transformed_lidar_normals = []

# Plot Camera normal, Lidar normal and Transformed lidar normal on the image
for ind in range(len(camera_images)):
    image_name = camera_images[ind]
    img = cv2.imread(camera_images[ind])

    # Load the camera normal, lidar normal and transformed lidar normal for the␣
 ↪image
    lidar_normal = lidar_plane_normals[ind]
    camera_nomral = camera_plane_normals[ind]
    transformed_lidar_normal = np.dot(CRL, lidar_normal)

    # Load the camera points, lidar points and transformed lidar points for the␣
 ↪image
    camera_pts = camera_image_points[ind]
    lidar_pts = lidar_image_points[ind]
    lidar_to_camera_pts = np.dot(CTL, lidar_pts)
    transformed_lidar_normals.append(transformed_lidar_normal)

    # Compute the mean of the image points
    mean_x = np.mean(image_points[ind][:,:, 0])
```

```
    mean_y = np.mean(image_points[ind][:,:, 1])

    # Camera normal is plotted in blue color
    normal_plotted_image = draw_image_normals(img, camera_nomral, mean_x,␣
↪mean_y, color=(0, 0, 255))
    # Lidar normal is plotted in green color
    normal_plotted_image = draw_image_normals(normal_plotted_image,␣
↪lidar_normal, mean_x, mean_y, color=(0, 255, 0))
    # Transformed lidar normal is plotted in red color
    normal_plotted_image = draw_image_normals(normal_plotted_image,␣
↪transformed_lidar_normal, mean_x, mean_y, color=(255, 0, 0))

    plt.title(f"Image-{i+1}: '" + image_name.split('\\')[-1] + "'")
    plt.imshow(normal_plotted_image)
    plt.show()
```



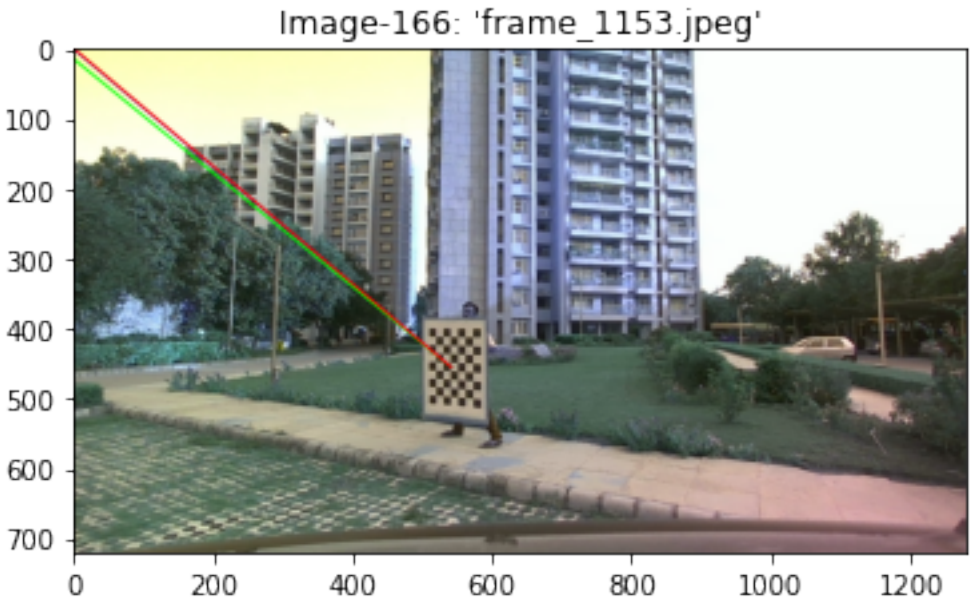


Image-166: 'frame_1153.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

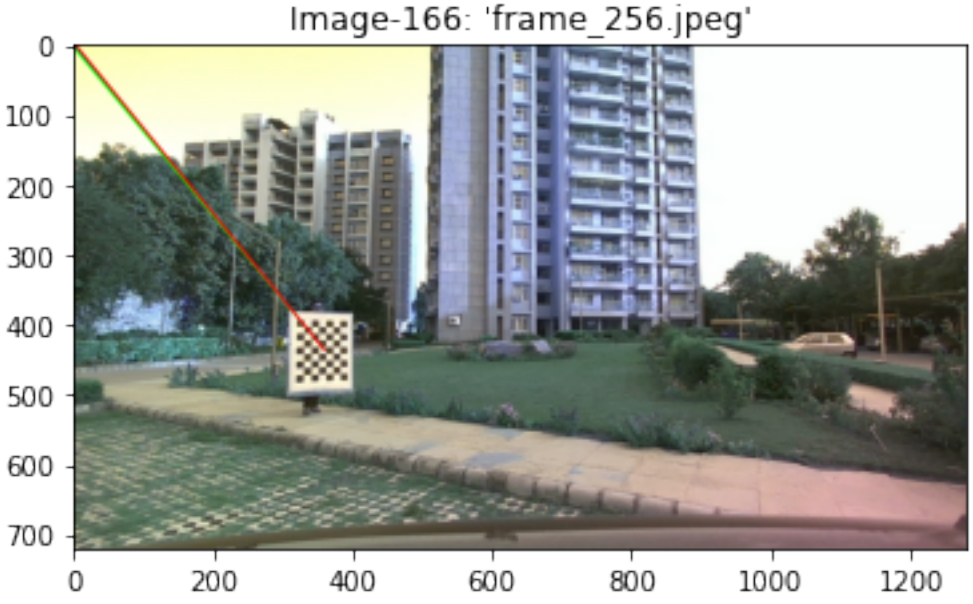


Image-166: 'frame_256.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

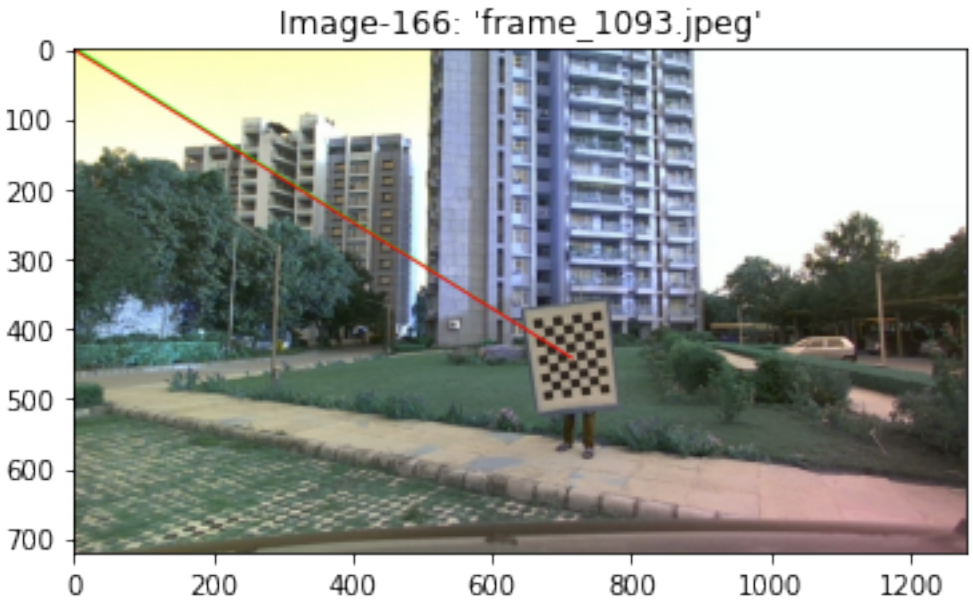


Image-166: 'frame_1093.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

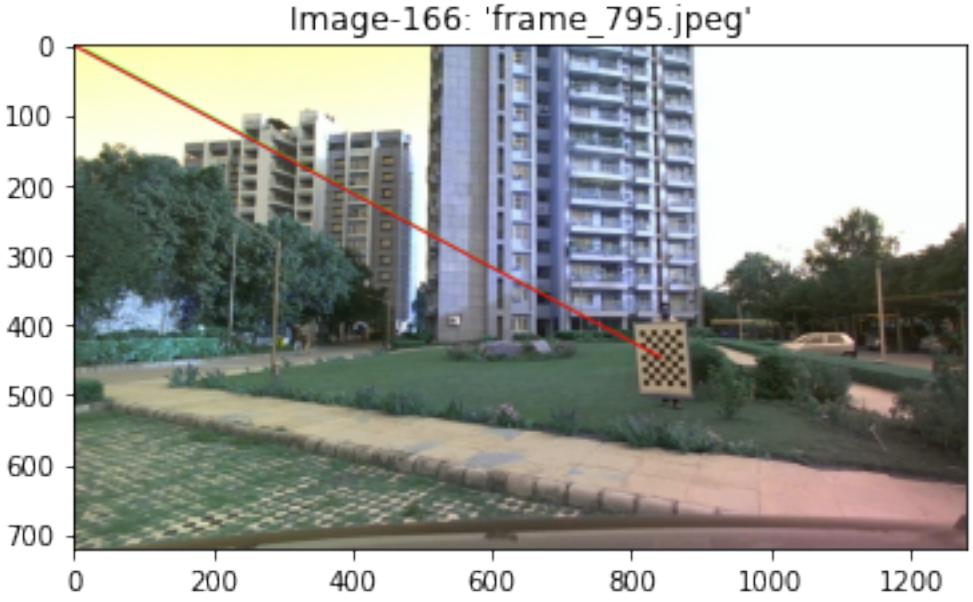


Image-166: 'frame_795.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

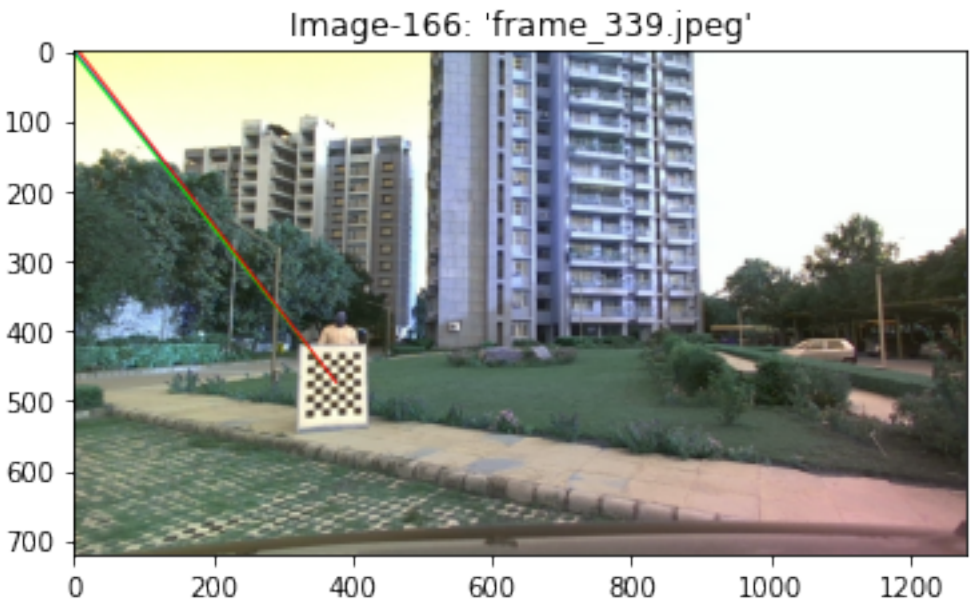


Image-166: 'frame_339.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

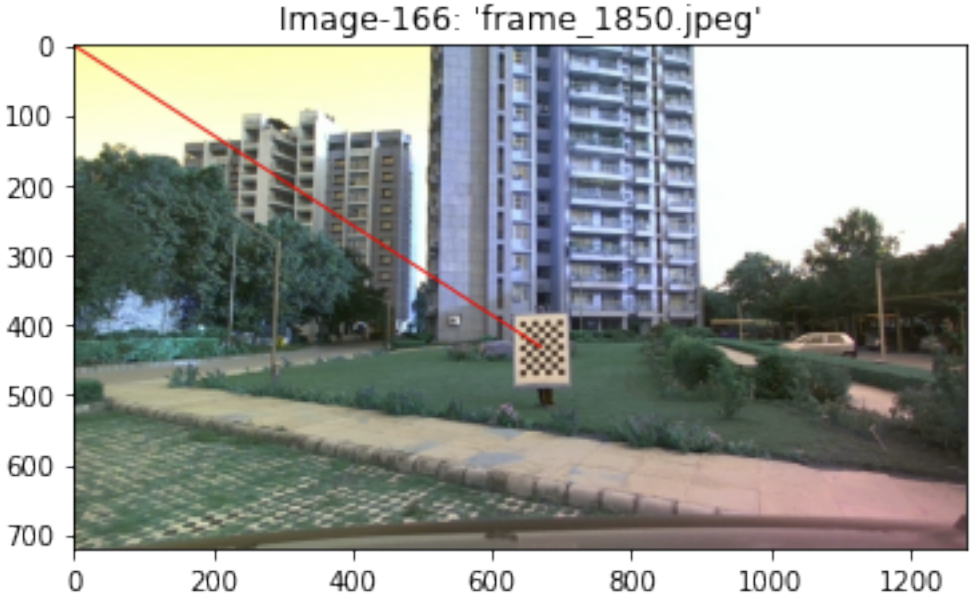


Image-166: 'frame_1850.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

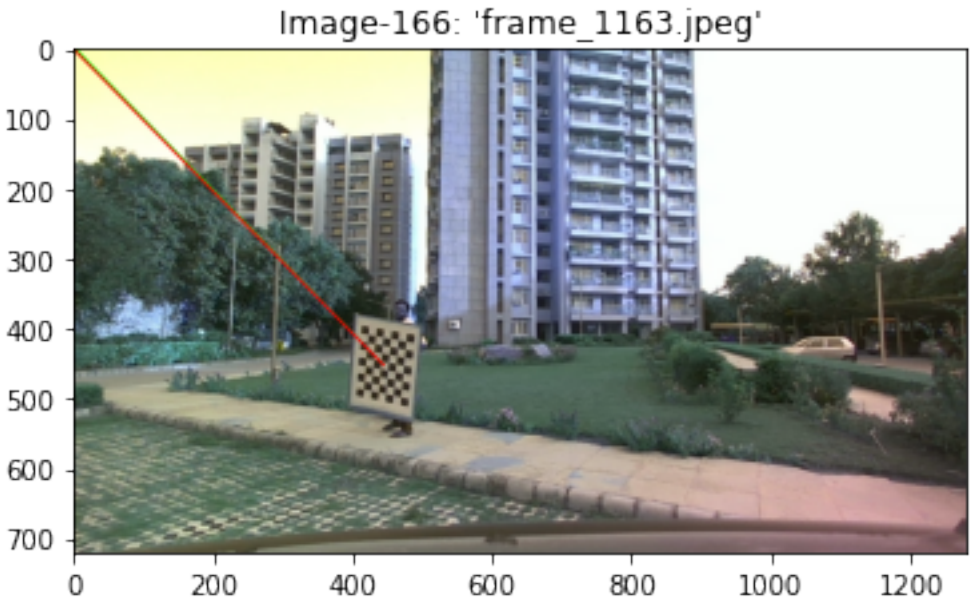


Image-166: 'frame_1163.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

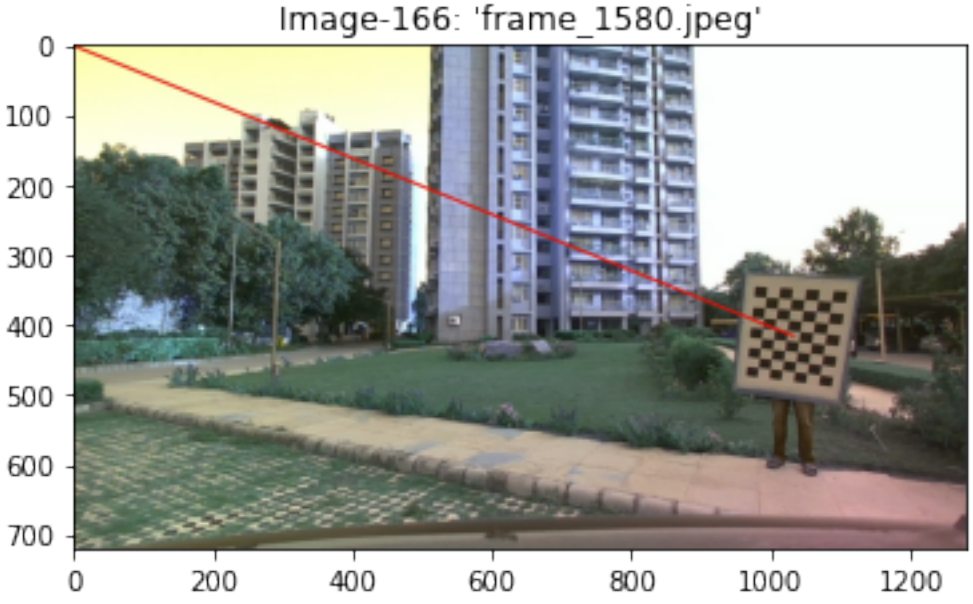


Image-166: 'frame_1580.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

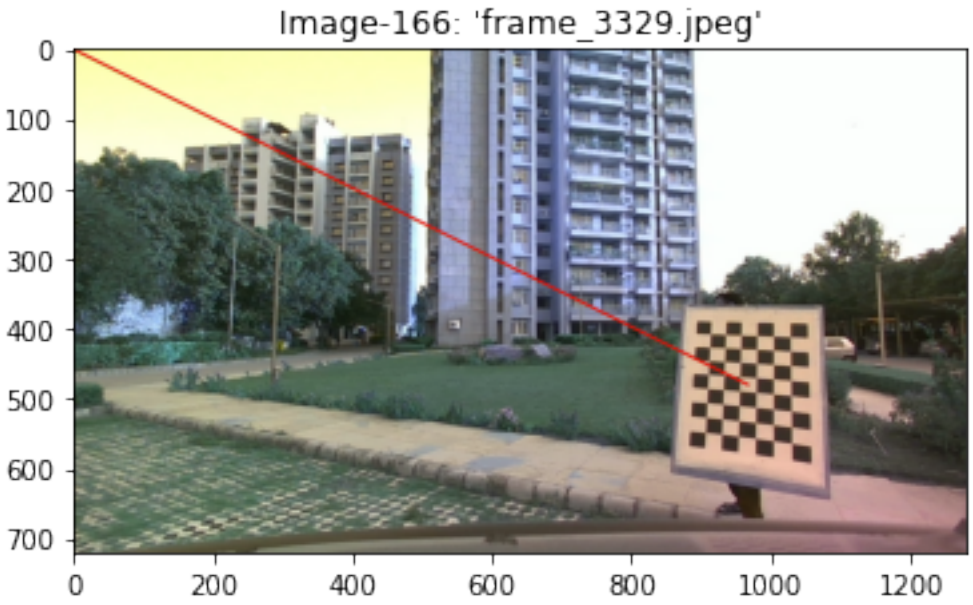


Image-166: 'frame_3329.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

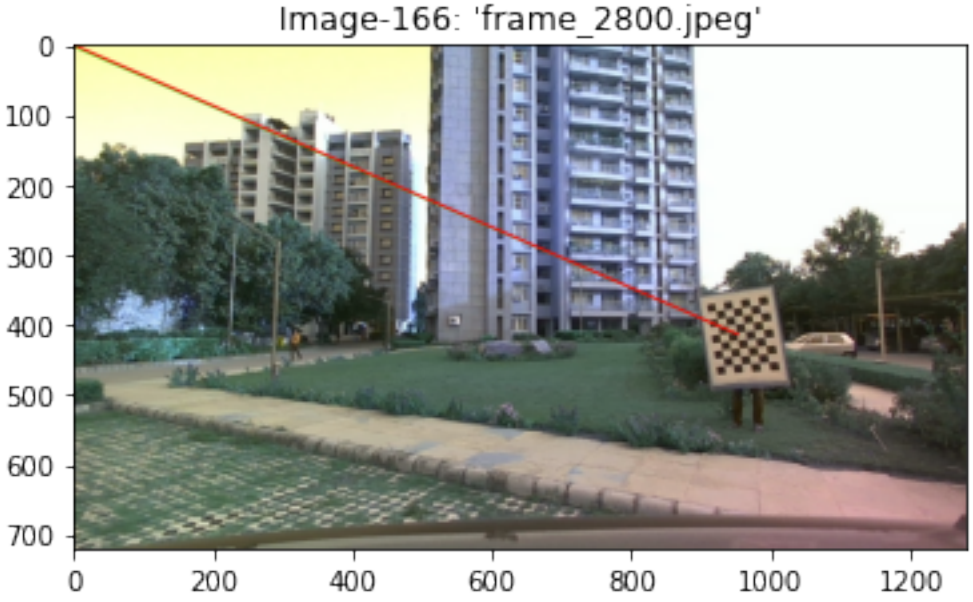


Image-166: 'frame_2800.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

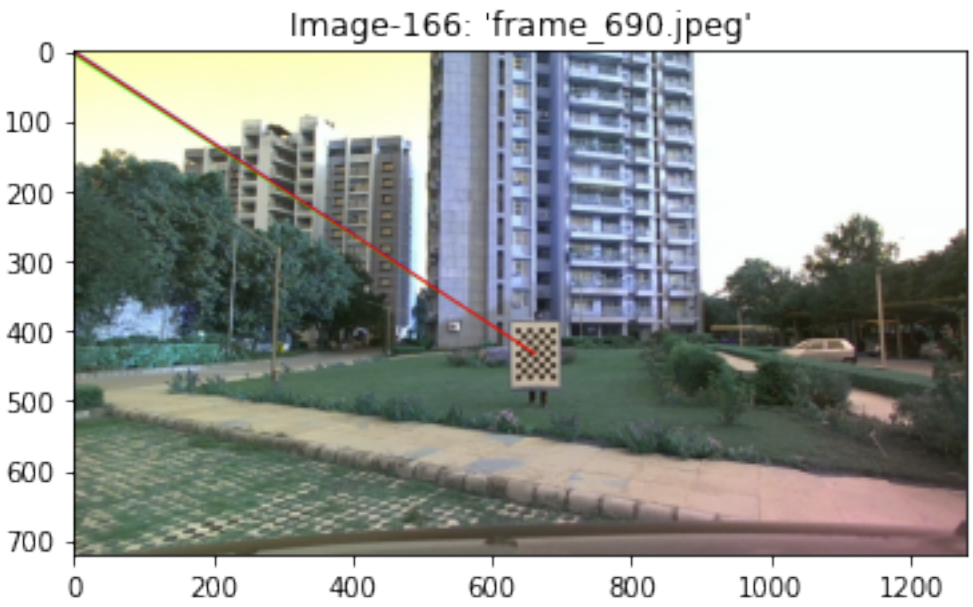


Image-166: 'frame_690.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

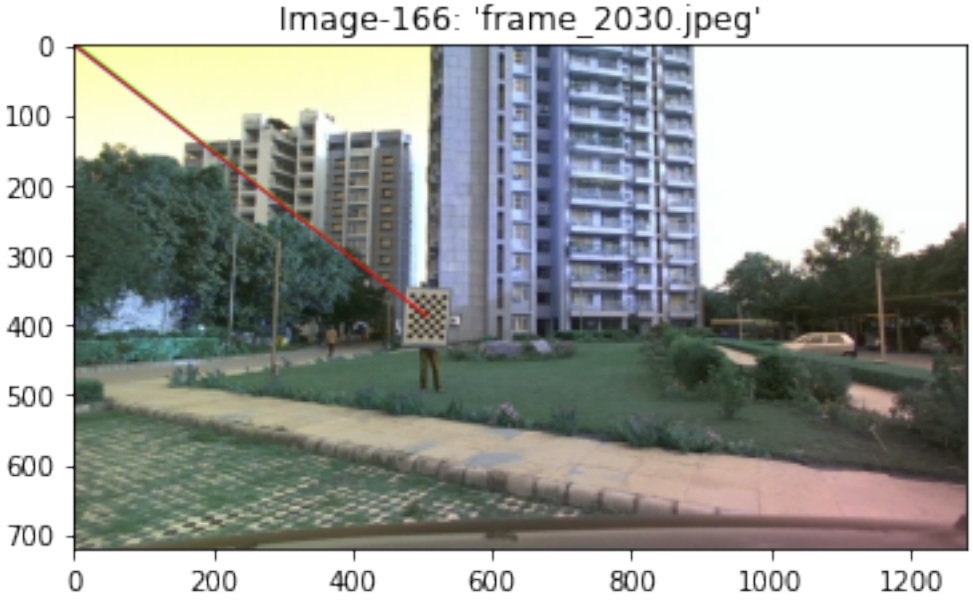


Image-166: 'frame_2030.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

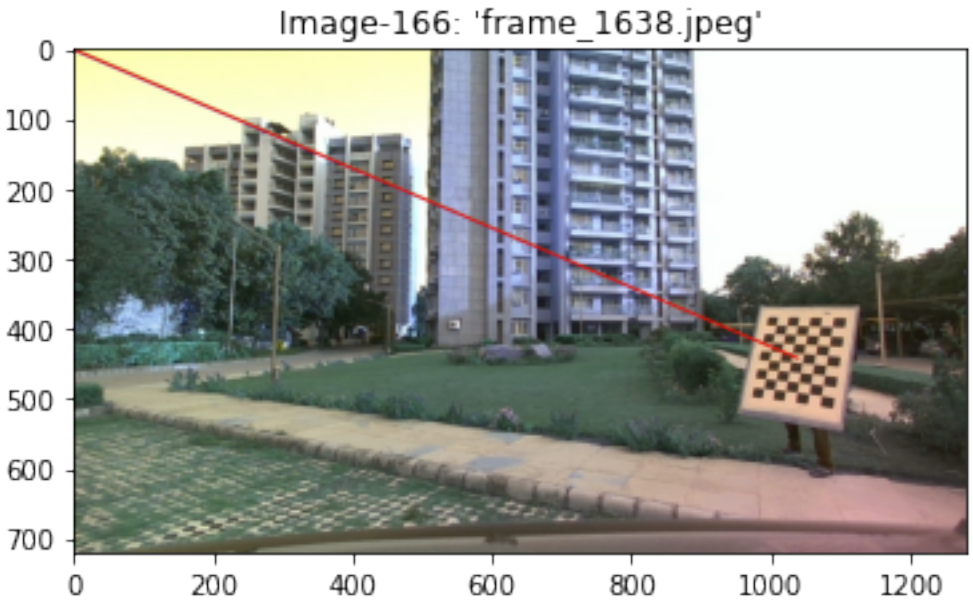


Image-166: 'frame_1638.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

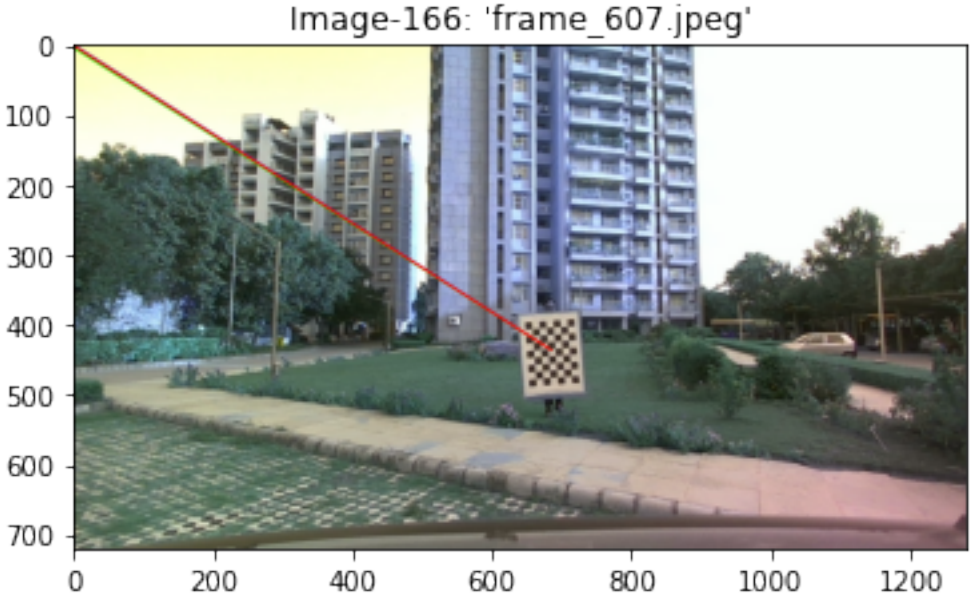


Image-166: 'frame_607.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

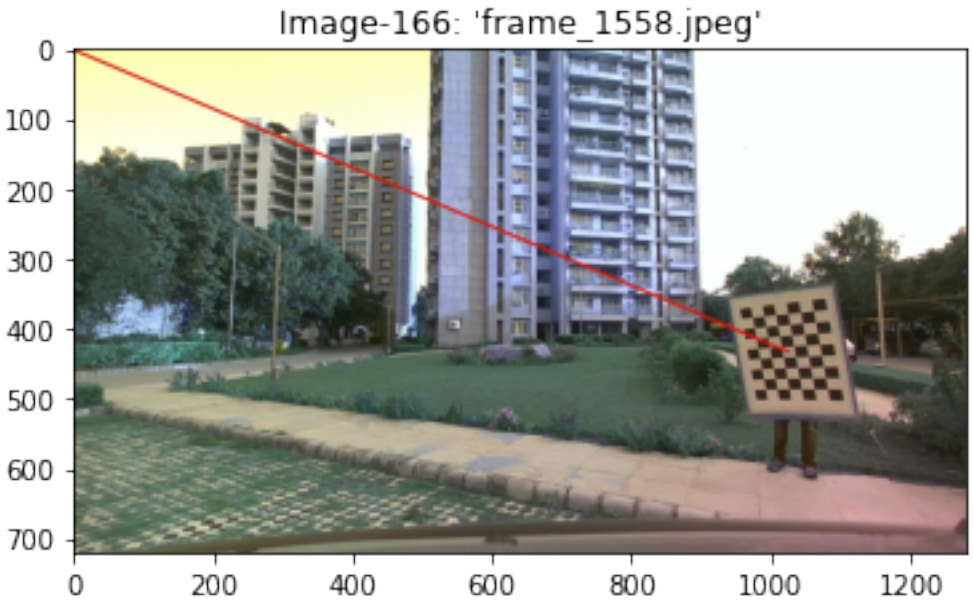


Image-166: 'frame_1558.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

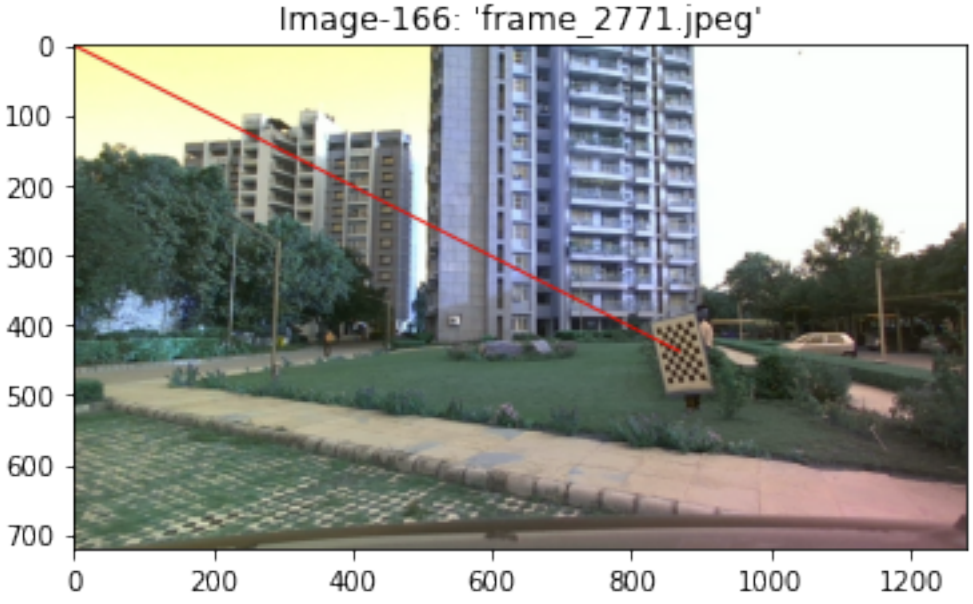


Image-166: 'frame_2771.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

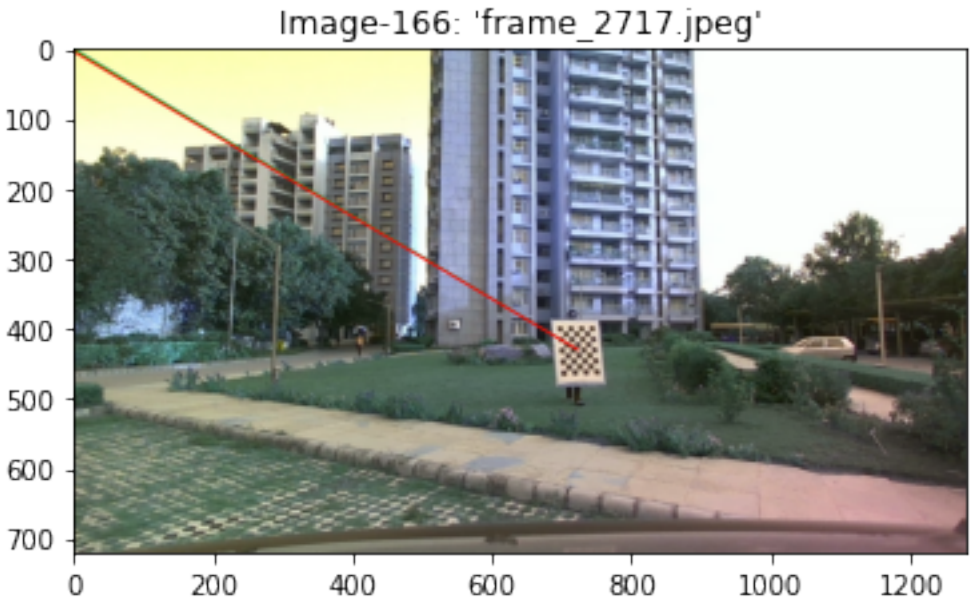


Image-166: 'frame_2717.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

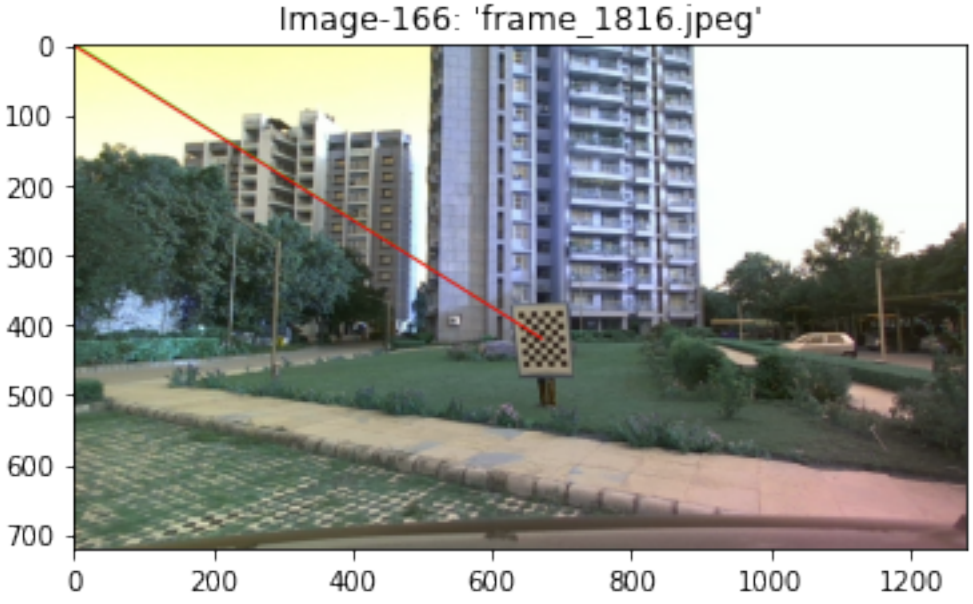


Image-166: 'frame_1816.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

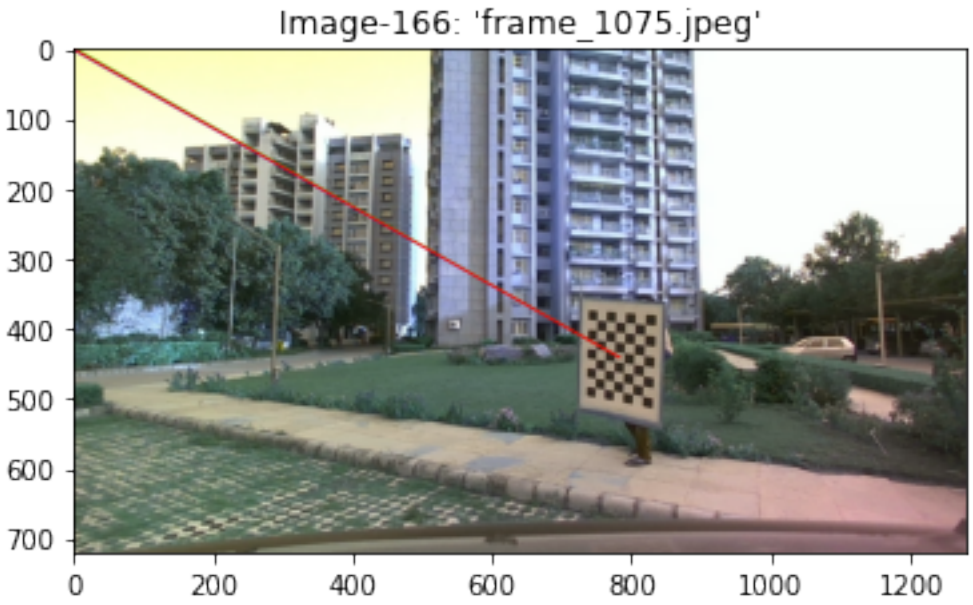


Image-166: 'frame_1075.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

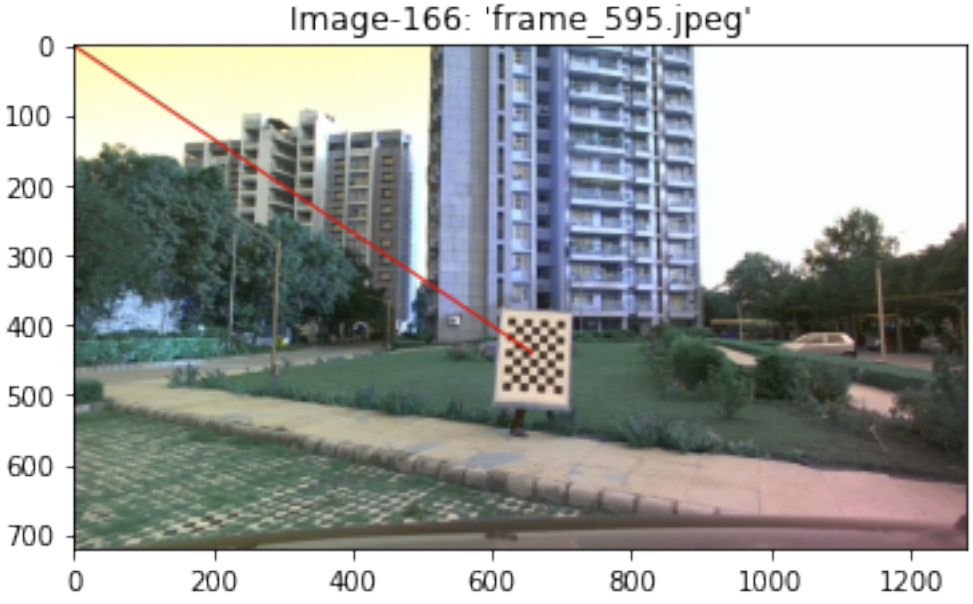


Image-166: 'frame_595.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

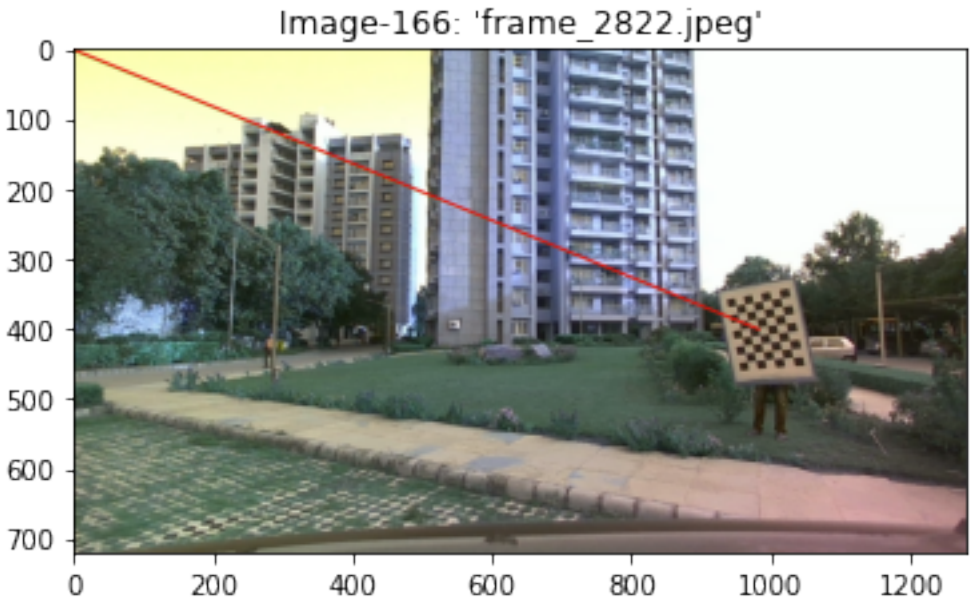


Image-166: 'frame_2822.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

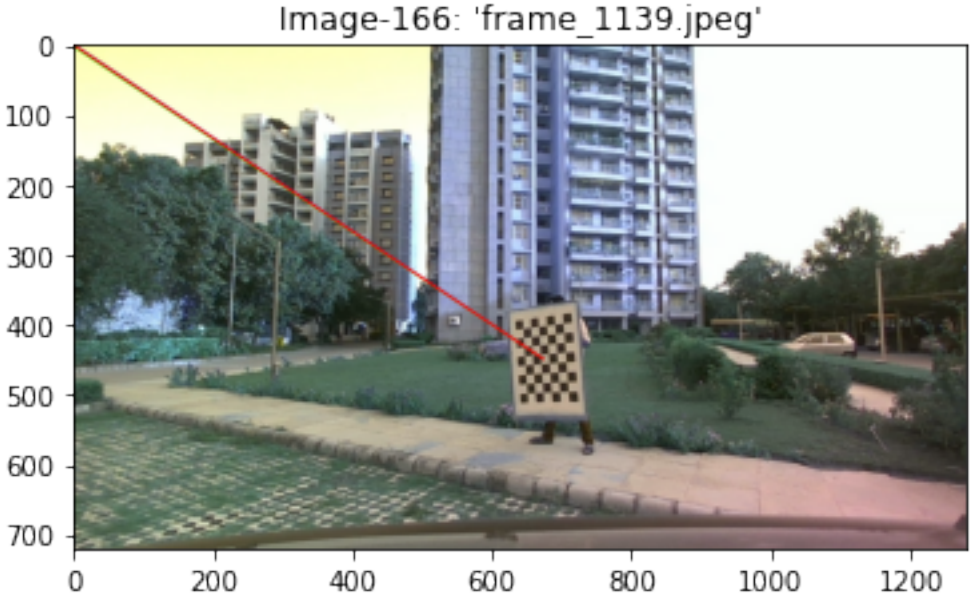
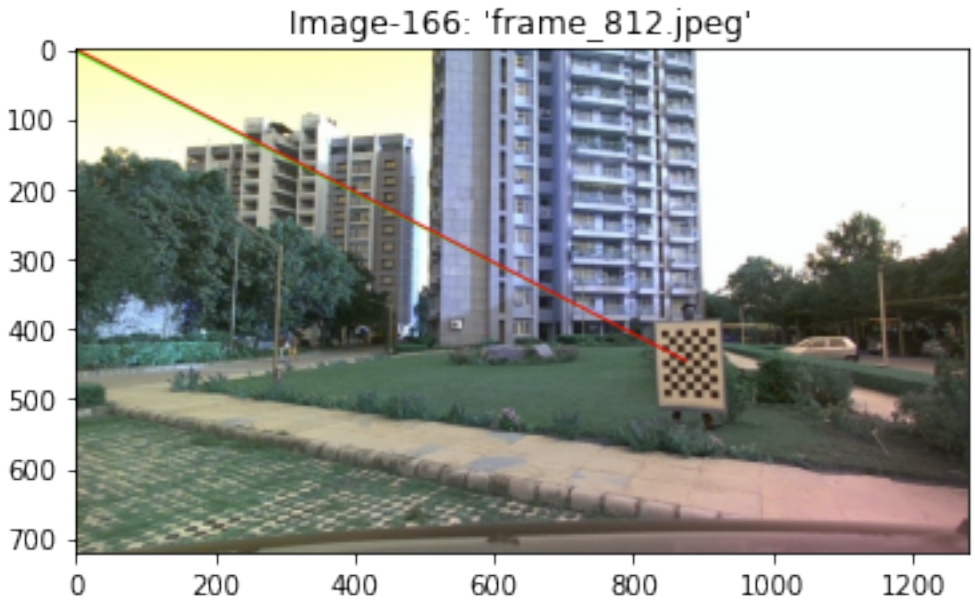
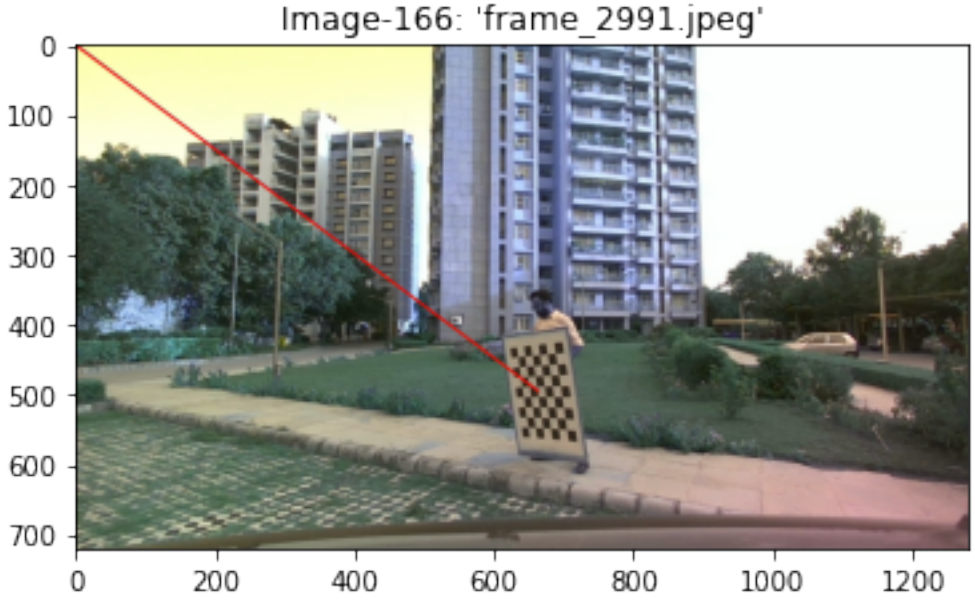


Image-166: 'frame_1139.jpeg'
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200





[ ]:
```
transformed_lidar_normals = np.array(transformed_lidar_normals)
error = np.dot(camera_plane_normals, transformed_lidar_normals.T)
error = np.clip(error, -1, 1)
cosine_distance = np.arccos(error)
```

[ ]:
```
print('Mean cosine distance between camera and transformed lidar normals: ', np.
  ↪mean(cosine_distance))
print('Standard deviation of cosine distance between camera and transformed␣
  ↪lidar normals: ', np.std(cosine_distance))
```

```
Mean cosine distance between camera and transformed lidar normals:
1.5368298290641207
Standard deviation of cosine distance between camera and transformed lidar
normals:  0.6017813450071197
```

[ ]:
```
error_for_each_img = np.mean(cosine_distance, axis=1)
plt.hist(cosine_distance.flatten(), bins=25)
plt.ylabel('Frequency of Cosine Distance')
plt.xlabel('Cosine Distance (in radians)')
plt.title('Histogram plot of Cosine Distances')
plt.show();
```