Principles of Robot Autonomy: Homework 3

[Chetan Reddy] [Narayanaswamy]
10/28/2024

Other students worked with: None
Time spent on homework: 7 hours

Problem 1:

Part (i)

Question: Use the cv2.findChessboardCorners(...) along with the corner parameters specified in the code to extract the cross-junction pixel locations, and plot them on top of the image using the ax.plot(...) function. Include this plot in your write-up.

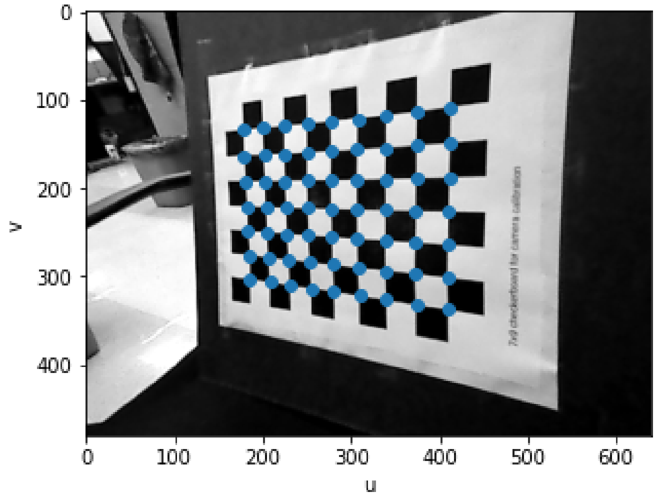


Figure 1: Cross Junction Pixel Locations using cv2.findChessboardCorners(...)

Part (ii)

Question: If we were to use noisy and lower-resolution images, how would it impact corner detection? How can we get to ultimately enhance the results of camera calibration, and what techniques can we use?

Answer:
Impact of Noise and Low Resolution:

- Edges and corner detection is usually done by using gradient information which is easily affected by noise. Therefore, noise can lead to inaccurate detections with false positives.
- Very low resolution can affect corner detection as well. If the resolution is of the order of the size of the square in the image, the corner detections will be inaccurate.

Techniques to Enhance Calibration Results:

- Use **denoising filters** to reduce the impact of noise (Like Gaussian blur)
- Take **multiple images** from different angles to increase the number of datapoints and thus reduce the variance of the estimator.
- Apply **image sharpening** techniques to increase edge contrast, making corners easier to detect.

Part (iii)

Question: Use the method outlined above, and the corner locations from (i) to compute the rotation, R, and translation, t, of the camera with respect to the chessboard in the image, and include the values that you compute in your write up with elements up to two significant figures.

Answer

$$R = \begin{bmatrix} 0.84 & 0.025 & 0.55 \\ 0.05 & 1.02 & -0.12 \\ -0.54 & 0.13 & 0.86 \end{bmatrix}$$

$$t = \begin{bmatrix} -0.060 \\ -0.055 \\ -0.25 \end{bmatrix}$$

Part (iv)

Question How could errors in the camera's intrinsic parameters propagate into the extrinsic calibration when using this checkerboard method, and how can we validate our resulting matrices/parameters?

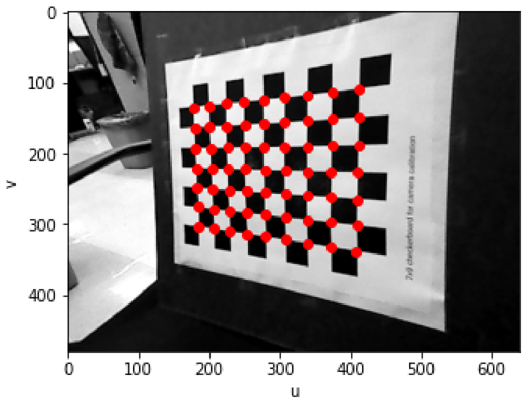
Answer

- Errors in intrinsic parameters (focal length, distortion) affect the perceived position of the checkerboard and the accuracy of extrinsic calibration.
- Reprojection error measures the difference between projected 3D points and actual 2D image points. This can help validate our calibration matrices.
- One method is to use easily identifiable objects (like aruco markers) whose location is known in 3D and find its 2D location on the image by tracking the aruco marker (inbuilt opencv functions do this). We can see if the camera matrix with the extrinsic parameters can reproject the 3D point accurately.

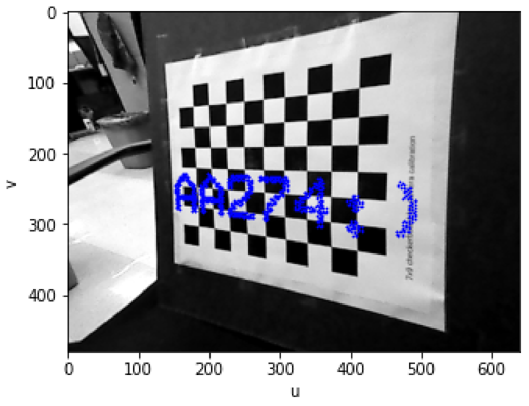
Part (v)

Question: Complete the function transform_world_to_camera, and use it to project the world-frame chessboard corner locations onto the image along with the actual corner locations. Include the resulting plot in your write-up.

Answer



(a) Reprojection



(b) Augmented Reality Application

Figure 2: Reprojections

Problem 2:

Part (i)

Question Include a screenshot of the cars image with bounding boxes with score_threshold of 0.6.

Answer

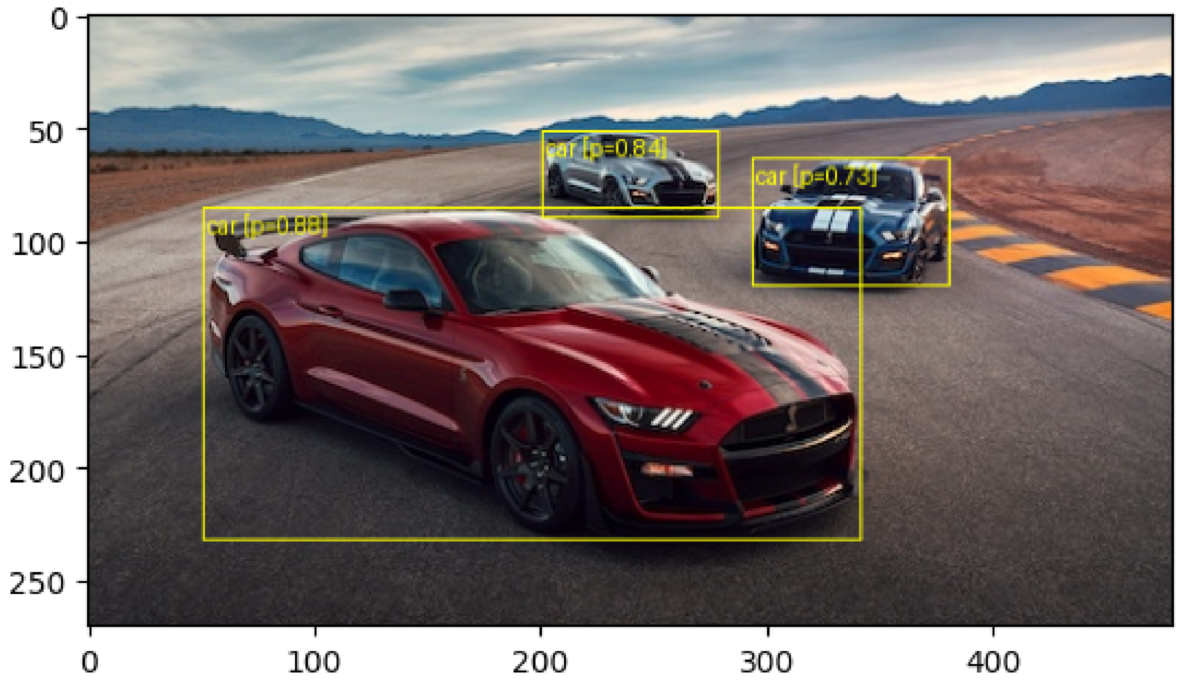


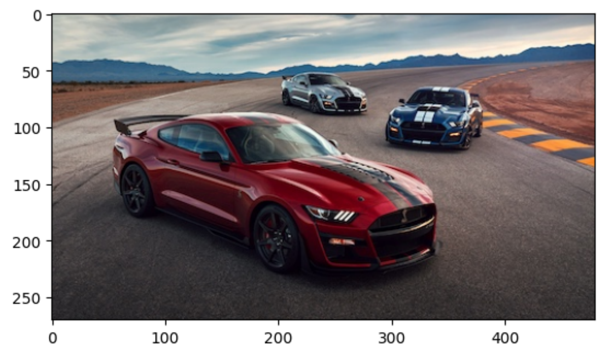
Figure 3: Object detection (with threshold of 0.6)

Part (ii)

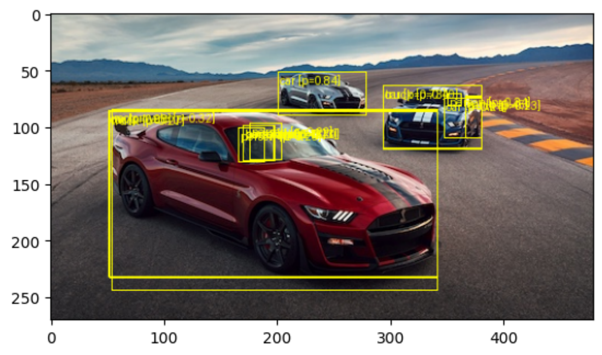
If the threshold is set to 1, no bounding boxes would be drawn because the score represents the confidence probability that has a range of 0 to 1, therefore, there are zero boxes whose score is greater than one.

Part (iii)

If the threshold is set to 0.01, almost all the bounding boxes (even with a low confidence score) will be displayed. This is not ideal because there would be many false positives and the image would be very cluttered.



(a) Threshold = 1 (no detections)

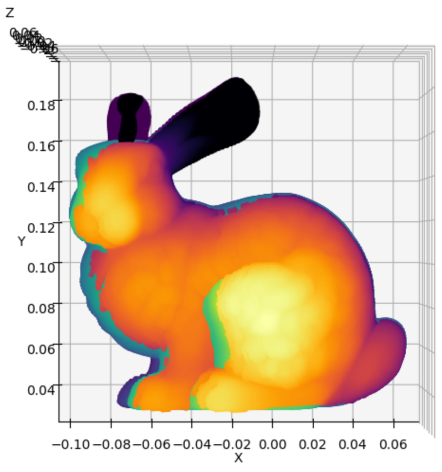


(b) Threshold = 0.01 (too many detections)

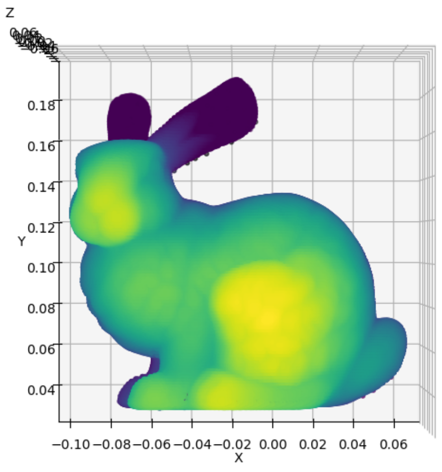
Figure 4: Object Detection in Cars

Problem 3:

Part (i)



(a) Full Source Point Cloud



(b) Downsampled Point Cloud

Figure 5: Full and Downsampled Point Clouds

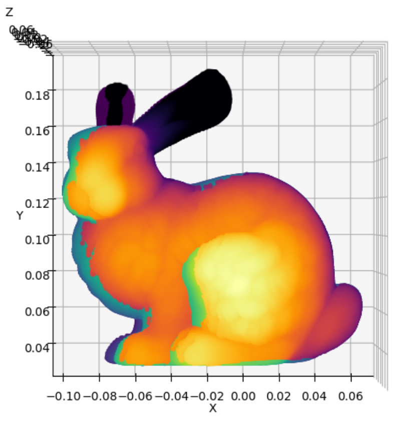


Figure 6: Using Robust ICP

Appendix A: Code Submission

Problem 1

```python
ncorners_y = 7
ncorners_x = 9
# --------- YOUR CODE STARTS HERE ---------------
ret, corners = cv2.findChessboardCorners(image, (ncorners_x, ncorners_y))

if ret:
    ax.plot(corners[:, 0, 0], corners[:, 0, 1], 'o')  # Plotting in red ('ro')
else:
    print("No Corners under the given parameters")

# --------- YOUR CODE ENDS HERE -----------------
```

```python
    # --------- YOUR CODE STARTS HERE ---------------
# 1.a
# Finding World Coordinates
x = np.arange(ncorners_x) * SQUARE_SIZE
y = np.arange(ncorners_y) * SQUARE_SIZE
X, Y = np.meshgrid(x, y)

# Stack and reshape into the format P = [[X_0, Y_0, 1], ..., [X_n, Y_n, 1]]
P = np.hstack((X.reshape(-1, 1), Y.reshape(-1, 1), np.ones((X.size, 1))))
print("Shape of P: {} | Length of corner: {}".format(P.shape,len(corners)))


# 1.b
N = P.shape[0]
M = np.zeros((2*N,9))

for i in range(N):
    ui,vi = corners[i][0]
    Xi,Yi,_ = P[i]
    M[2*i,:] = np.array([-Xi,-Yi,-1,0,0,0,ui*Xi,ui*Yi,ui])
    M[2*i+1,:] = np.array([0,0,0,-Xi,-Yi,-1,vi*Xi,vi*Yi,vi])

print("Shape of M:",M.shape)

### 1.c
U, S, Vt = np.linalg.svd(M) # Singular values are returned in descending order
H = Vt[-1].reshape(3, 3)

### 2.a
KinvH = np.linalg.inv(K)@H
print("Shape of KinvH:",KinvH.shape)

### 2.b
lambd = np.linalg.norm(KinvH[:, 0])

### 2.c
r0 = KinvH[:,0]/lambd
r1 = KinvH[:,1]/lambd
r2 = np.cross(r0,r1)
t = KinvH[:,2]/lambd
```

```

### 2.d
R = np.column_stack((r0,r1,r2))

print("R Matrix:")
print(R)

print("\nt:")
print(t)
# --------- YOUR CODE ENDS HERE -----------------
```

```
def transform_world_to_camera(K, R, t, world_coords):
    """
    Args:
        K: np.array with shape (3, 3), camera intrinsics matrix.
        R: np.array with shape (3, 3), camera rotation.
        t: np.array with shape (3, ) or (3, 1), camera translation.
        world_coords: np.array with shape (N, 3), cartesian coordinates (X, Y, Z)
            in world frame to transform into camera pixel space.
    Return:
        uv: np.array with shape (N, 2), with (u, v) coordinates of that are
            the projections of the the world_coords on the image plane.
    """
    # --------- YOUR CODE STARTS HERE ---------------
    N = world_coords.shape[0]
    t = t.reshape(-1,1)

    cTw = K@np.hstack([R,t]) # Shape = 3,4
    world_coords_homogenous = np.hstack([world_coords,np.ones((N,1))])
    print

    assert world_coords_homogenous.shape==(N,4), "World Coordinates Homogenous
    Coordinates shape is not N,4"
    assert cTw.shape==(3,4), "cTw.shape is not (3,4)"

    uv_homogenous = (cTw@world_coords_homogenous.T).T

    assert uv_homogenous.shape==(N,3), "uv_homogenous.shape is not (N,3)"

    uv = uv_homogenous[:,:2]/uv_homogenous[:,2:3]
    assert uv.shape == (N,2),"uv.shape is not (N,2)"
    # --------- YOUR CODE ENDS HERE -----------------
    return uv
```

```
# --------- YOUR CODE STARTS HERE ---------------
N = P.shape[0]
world_coords = np.hstack([P[:,:2],np.zeros((N,1))])
uv_reprojected = transform_world_to_camera(K,R,t,world_coords)
ax.plot(uv_reprojected[:, 0], uv_reprojected[:, 1], 'ro')  # Plotting in red ('ro')

# --------- YOUR CODE ENDS HERE -----------------
```

Problem 2

```
weights = FCOS_ResNet50_FPN_Weights.DEFAULT
########## Code starts here ##########
```

```python

model = fcos_resnet50_fpn(weights=weights)
model.eval()

outputs = model(image.unsqueeze(0)/255)[0] # The extra [0] is to slice out from the
    batch
########## Code ends here ############
```

```python
def draw_result(img, boxes, labels, scores):
    """
    draw bounding box visualization on top of the raw image
    """
    ########## Code starts here ##########
    class_descriptions = weights.meta['categories']
    label_descriptions = ["{} [p={:.2f}]".format(class_descriptions[label],score) for
    label, score in zip(labels, scores)]

    result_image = draw_bounding_boxes(
    img,
    boxes=boxes,
    labels=label_descriptions,
    colors="yellow")

    return result_image

    ########## Code ends here ##########
```

```python
########## Code starts here ##########
filtered_indices = outputs["scores"]>score_threshold
filtered_boxes = outputs["boxes"][filtered_indices]
filtered_labels = outputs["labels"][filtered_indices]
filtered_scores = outputs["scores"][filtered_indices]

img_viz = draw_result(image, filtered_boxes, filtered_labels, filtered_scores)
########## Code ends here ############
```

Problem 3

```python
def nearest_neighbor(src, dst, radius=0.01):
    '''
    Find the nearest (Euclidean) neighbor in dst for each point in src
    Input:
        src: Nxm array of points
        dst: Nxm array of points
    Output:
        distances: Euclidean distances of the nearest neighbor
        indices: dst indices of the nearest neighbor
    '''
    #######################################################################
    ######################### YOUR CODE HERE ##############################
    # Use the NearestNeighbors class sklearn.neighbors to compute the nearest
    # neighbor of each point in the source dataset to the target dataset
    # Note that by using this class and its methods correctly, you should be
    # able to get the distance between a point and its nearest neighbor,
    # as well as the indices of the nearest neighbor in the target point cloud

    # The full scan is the target, we are trying to map points in the source to the
    points in the target
    nearest_neighbor_obj = NearestNeighbors(n_neighbors=1,radius = radius)
    nearest_neighbor_obj.fit(dst) # Fitting it on the destination samples to query with
    source next

    distances, indices = nearest_neighbor_obj.kneighbors(src, return_distance=True)
    ######################################################################
    return distances.ravel(), indices.ravel()
```

```
    ######################### YOUR CODE HERE #############################
    # Write the loop for the ICP algorithm here
    # Follow the steps outlined in the prompt above.
    # Hints:
    # - Use the functions `nearest_neighbor()` and `best_fit_transform()`
    # - src and dst matrices are defined above with shape (m+1,N), while the above
    #   two function expect matrices of shape (N,m)
    print("Shape of dst = ",dst.shape) # It must be (4,larger_number)
    print("Shape of src = ",src.shape) # It must be (4,smaller_number)

    for i in trange(max_iterations):

      # Nearest Neighbours expects shape of N,3
      distances,indices = nearest_neighbor(src.T[:,:m],dst.T[:,:m],knn_radius)
      error = np.linalg.norm(distances)
      dst_points = dst[:,indices] # 4,smaller number

      T_i,_,_ = best_fit_transform(src[:m,:].T,dst_points[:m,:].T)
      src = T_i@src  # shape = 4,smaller_number
      print("Error:{:.6f}".format(error))
      if abs(error-prev_error)<tolerance:
        break

    ######################################################################
    # calculate final transformation
    T,_,_ = best_fit_transform(A, src[:m,:].T)
```

```
######################### YOUR CODE HERE #############################
reg_p2p = o3d.pipelines.registration.registration_icp(
    source, target, 0.02, T,
    o3d.pipelines.registration.TransformationEstimationPointToPoint())
######################################################################
```