Principles of Robot Autonomy: Homework 3

Wei-Lin Pai
10/28/24

Other students worked with:
Time spent on homework: 6 hrs

Problem 1: Chessboard Camera Extrinsics

1. Feature Extraction

   (a) Plot cross-junction pixels location

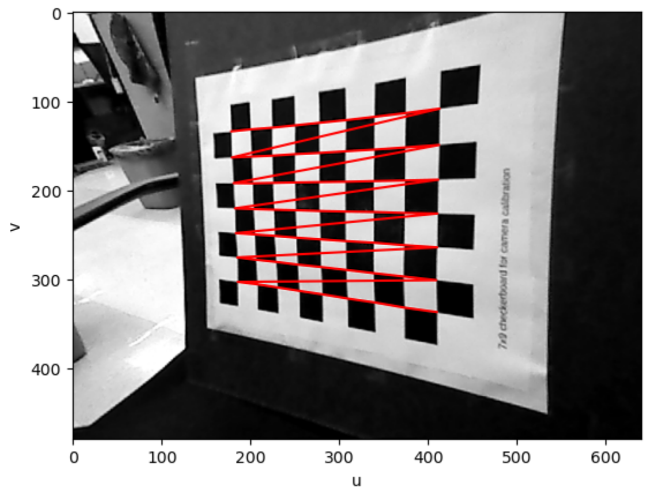

Figure 1: cross-junction pixel locations

   (b) If we have noisy and low-resolution image, it will be harder for the detection algorithm to locate precise positions of the corners, potentially leading to incorrect or missed corner detection. We can increase the corner position accuracy by using cv.cornerSubPix(). Or we can do image pre-processing, which apply image sharpening or noise reduction filters to improve contrast.

   (c)

$$R = \begin{bmatrix} 0.84 & 0.025 & 0.55 \\ 0.046 & 1.0 & -0.12 \\ -0.54 & 0.13 & 0.86 \end{bmatrix} \tag{1}$$

$$t = \begin{bmatrix} -0.060 \\ -0.055 \\ -0.25 \end{bmatrix} \tag{2}$$

(d) The errors in intrinsic parameters could be propagate into the extrinsic by solving $K^{-1}H = \lambda[r_0 r_1 t]$ to get extrinsic matrix.
We can validate our resulting matrices/parameters through calculating re-projection error. We can calculate the norm distance between what we got thorough transformation and the position we got from corner finding algorithm.

(e) World to Camera Projection

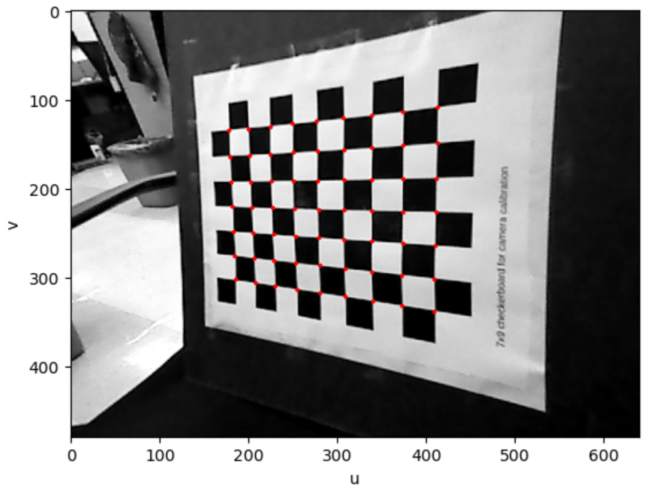

Figure 2: actual corner projection

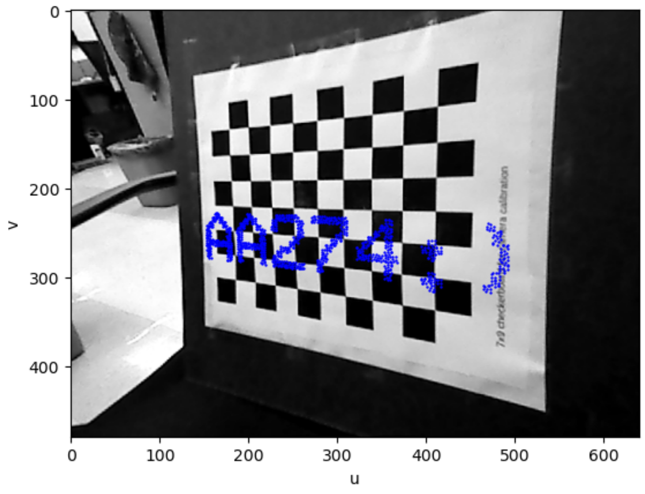

Figure 3: secret projection

Problem 2: Object Detection using Pytorch

1. Detection results with threshold 0.6

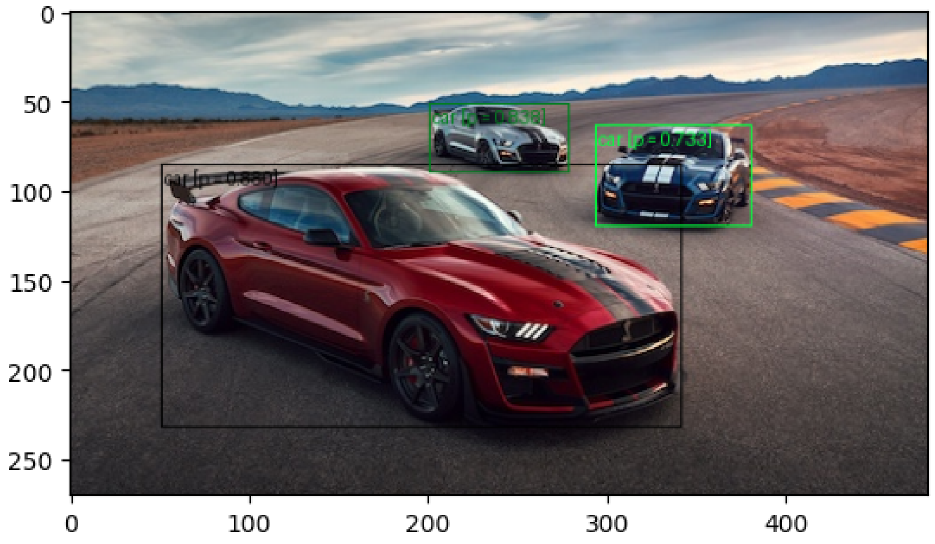

Figure 4: object detection with threshold = 0.6

2. If we change score threshold to 1.0, this mean only the detection result of 1.0 confidence score will be shown. However, there is no detection result of 100% in this image, and normally the confidence of detection is hard to reach 100%. Therefore, there is no bounding box on the image.

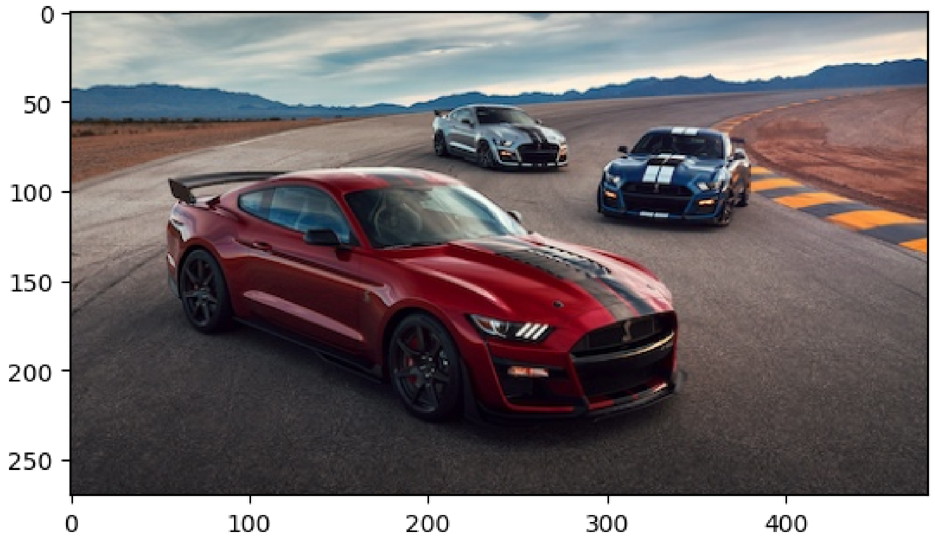

Figure 5: object detection with threshold = 1

3. If we change score threshold to 0.01, this mean all the detection result over 0.01 confidence score will be shown. For object detection, 0.01 is a very low value and hence almost all the detected bounding boxes will be shown, even with low confidence.

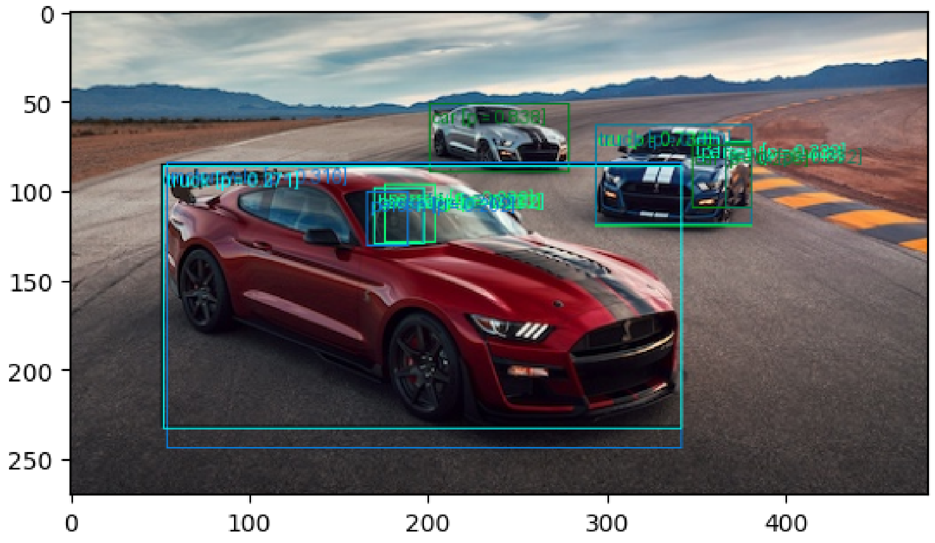

Figure 6: object detection with threshold = 0.01

Problem 3: Iterative Closest Point (ICP) algorithm for Point Cloud Registration

1. Basic ICP

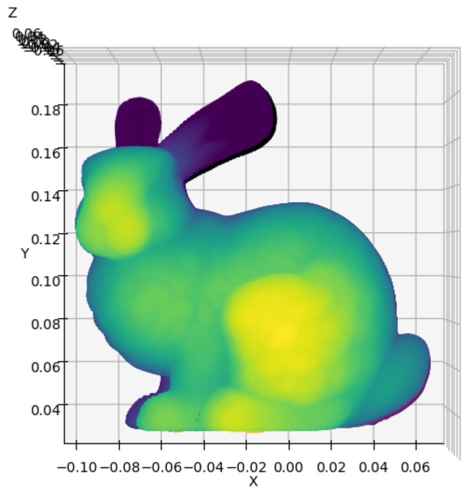

Figure 7: Basic ICP with full source point cloud

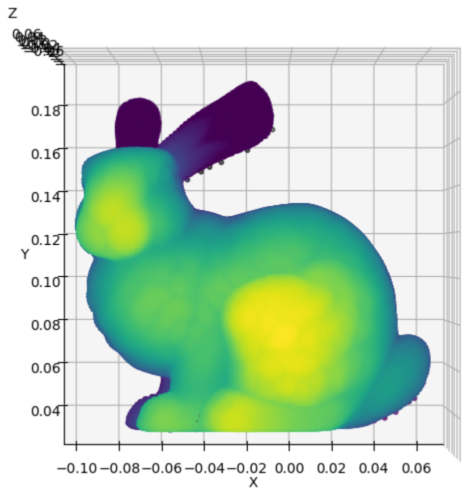

Figure 8: Basic ICP with downsampled point cloud

2. Robust ICP using open3D

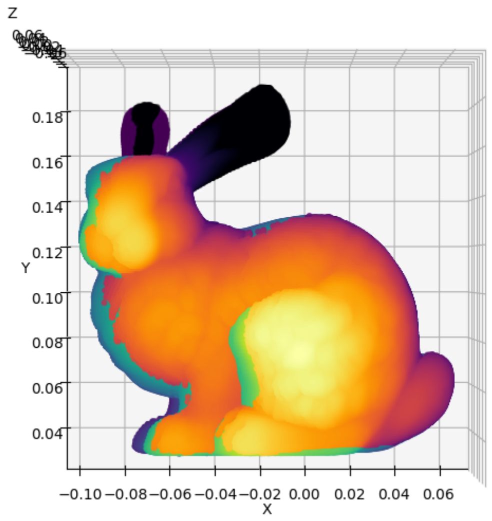

Figure 9: Robust ICP

Appendix A: Code Submission

P1.1

```
 # --------- YOUR CODE STARTS HERE ---------------
ret, corner = cv2.findChessboardCorners(image, (ncorners_x, ncorners_y))
if ret:
    ax.plot(corner[:,:,0],corner[:,:,1],'r')
# --------- YOUR CODE ENDS HERE -----------------
```

P1.2

```
# --------- YOUR CODE STARTS HERE ---------------
x = np.arange(ncorners_x)*SQUARE_SIZE
y = np.arange(ncorners_y)*SQUARE_SIZE
X_grid,Y_grid = np.meshgrid(x,y)
X_pts = X_grid.reshape(-1)
Y_pts = Y_grid.reshape(-1)
ones = np.ones(ncorners_x*ncorners_y)
P = np.hstack((X_pts[:, np.newaxis],
              Y_pts[:, np.newaxis],
              ones[:,np.newaxis]))
# print(P.shape)
M = np.zeros((2*ncorners_x*ncorners_y,9))
for i in range(ncorners_x*ncorners_y):
    M[2*i,:3] = -P[i,:]
    M[2*i+1,3:6] = -P[i,:]
    M[2*i,6:] = P[i,:]*corner[i,:,0]
    M[2*i+1,6:] = P[i,:]*corner[i,:,1]
# print(M)
U,S,Vh = np.linalg.svd(M)
H = Vh[-1].reshape((3,3))

KinvH = np.linalg.inv(K) @ H
KinvH = KinvH / np.linalg.norm(KinvH[:,0])
r0 = KinvH[:,0]
r1 = KinvH[:,1]
r2 = np.cross(r0,r1)
t = KinvH[:,2]
R = np.column_stack((r0,r1,r2))
print(R)
print(t)
# --------- YOUR CODE ENDS HERE -----------------
```

P1.3

```
def transform_world_to_camera(K, R, t, world_coords):
    # --------- YOUR CODE STARTS HERE ---------------
    t = t.reshape((3,1))
    N = world_coords.shape[0]
    homogeneous_coords = np.hstack((world_coords, np.ones((N, 1))))
    Extrinsic = np.hstack((R,t))
    pixel = K @ Extrinsic @ homogeneous_coords.T
    pixel = pixel/pixel[2]
    uv = pixel[:2,:].T
    # --------- YOUR CODE ENDS HERE -----------------
    return uv

ax = plt.subplot()
ax.imshow(image, cmap="gray")
ax.set_xlabel("u")
ax.set_ylabel("v")

# --------- YOUR CODE STARTS HERE ---------------
zeros = np.zeros(ncorners_x*ncorners_y)
world_coords = np.hstack((X_pts[:, np.newaxis],
               Y_pts[:, np.newaxis],
               zeros[:,np.newaxis]))
uv = transform_world_to_camera(K, R, t, world_coords)
ax.scatter(uv[:, 0], uv[:, 1], s=3.0, c="r")
# --------- YOUR CODE ENDS HERE -----------------
```

P2

```
from torchvision.models.detection import fcos_resnet50_fpn, FCOS_ResNet50_FPN_Weights

weights = FCOS_ResNet50_FPN_Weights.DEFAULT
########## Code starts here ##########>>>
model = fcos_resnet50_fpn(weights)
image = image.float() / 255.0
image = image.unsqueeze(0)
model.eval()
with torch.no_grad():
  predictions = model(image)
pred = predictions[0]
outputs = {
    'boxes': pred['boxes'],   # Bounding boxes [N, 4] tensor
    'scores': pred['scores'], # Confidence scores [N] tensor
    'labels': pred['labels']  # Class labels [N] tensor
}
########## Code ends here ############


from torchvision.utils import draw_bounding_boxes
def draw_result(img, boxes, labels, scores):
    """
    draw bounding box visualization on top of the raw image
    """
    ########## Code starts here ##########

    img = img.squeeze()
    categories = [weights.meta['categories'][i] for i in labels]
    labels = [categories[j] + f" [p = {scores[j]:.3f}]" for j in range(len(categories))]
    return draw_bounding_boxes(image = img,boxes = boxes,labels = labels)
    ########## Code ends here ##########

########## Code starts here ##########
def draw_result(img, boxes, labels, scores):
    """
    draw bounding box visualization on top of the raw image
    """
    idx = scores >= score_threshold
    print(idx)
    score = scores[idx]
    print(score)
    label = labels[idx]
    box = boxes[idx]
    img = img.squeeze()
    categories = [weights.meta['categories'][i] for i in label]
    labels = [categories[j] + f" [p = {score[j]:.3f}]" for j in range(len(categories))]
    return draw_bounding_boxes(image = img,boxes = box,labels = labels)
img_viz = draw_result(image, outputs["boxes"], outputs["labels"], outputs["scores"])
########## Code ends here ############
```

P3

```
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
    neigh = NearestNeighbors(n_neighbors=1, radius=radius, algorithm='kd_tree')

    # Fit the model to the destination points
    neigh.fit(dst)

    # Find nearest neighbors
    distances, indices = neigh.kneighbors(src, return_distance=True)

    ########################################################################
    return distances.ravel(), indices.ravel()


def icp(A, B, init_pose=None, max_iterations=20, tolerance=0.001, knn_radius=0.01):
    '''
    The Iterative Closest Point method: finds best-fit transform that maps points A on to
        points B
    Input:
        A: Nxm numpy array of source mD points
        B: Nxm numpy array of destination mD point
        init_pose: (m+1)x(m+1) homogeneous transformation
        max_iterations: exit algorithm after max_iterations
        tolerance: convergence criteria
    Output:
        T: final homogeneous transformation that maps A on to B
        distances: Euclidean distances (errors) of the nearest neighbor
        i: number of iterations to converge
    '''
    # get number of dimensions
    m = A.shape[1]

    # make points homogeneous, copy them to maintain the originals
    src = np.ones((m+1,A.shape[0]))
    dst = np.ones((m+1,B.shape[0]))
    src[:m,:] = np.copy(A.T)
    dst[:m,:] = np.copy(B.T)

    # apply the initial pose estimation
    if init_pose is not None:
        src = np.dot(init_pose, src)

    prev_error = 0
    ######################################################################
    ######################### 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)
    for i in trange(max_iterations):
      distances, indices = nearest_neighbor(src[:-1,:].T,dst[:-1,:].T,radius = knn_radius)
      mean_error = np.mean(distances)
      if abs(prev_error - mean_error) < tolerance:
            break
      prev_error = mean_error
      dst_ = dst[:,indices]
      T,R,t = best_fit_transform(src[:-1,:].T,dst_[:-1,:].T)
      # print(T.shape)
      src = T@src
    ######################################################################
    # calculate final transformation
    T,_,_ = best_fit_transform(A, src[:m,:].T)

    return T

print("Apply point-to-point ICP")
######################################################################
######################### YOUR CODE HERE #############################
source, target, source_down, target_down, source_fpfh, target_fpfh = prepare_dataset(pcd,
    full_bunny, 0.01)
trans_init = execute_global_registration(source_down, target_down, source_fpfh, target_fpfh,
    voxel_size=0.01)
reg_p2p = o3d.pipelines.registration.registration_icp(
    source, target, threshold, trans_init,
    o3d.pipelines.registration.TransformationEstimationPointToPoint())
######################################################################
print(reg_p2p)
print("Transformation is:")
print(reg_p2p.transformation)
```