report

- I use the dictionary to save the target point in the opencv coordinate image, and click the left mouse to choose the start point. (these point are in opecv coordinate)

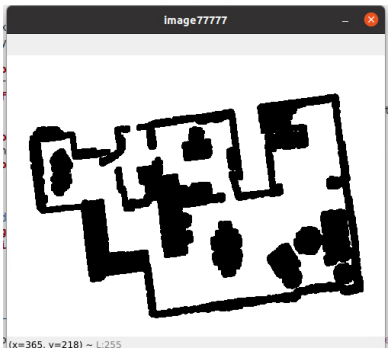
```
target_location = {'refrigerator':[177, 205], 'rack':[283, 139], 'cushion':[400, 211], 'lamp':[357, 297], 'cooktop':[145, 253]}
```

- Use the coordinate in the o3d world and use o3d.visualization.draw_geometries() to print it to test the lower and upper limit that I need to cut, and save the .png in the below code.

```
axis_pcd = o3d.geometry.TriangleMesh.create_coordinate_frame(size=0.5, origin=[0, 0, 0])  # 用此印到點雲上可知0, 0, 0和sim場景的0, 0, 0相同
```

```
for i in range(data.shape[0]):
    if (data[i][1] < -0.01 and data[i][1] > -0.035):  # 用axis_pcd測試出來的值
        current_row = np.array([[data[i][0]*10000/255, data[i][1]*10000/255, data[i][2]*10000/255, data[i][3], data[i][4], data[i][5]]])
        new_data = np.append(new_data, values = current_row, axis = 0)
        new_data = np.array(new_data)
        print(new_data.shape)
plt.scatter(new_data[:, 2], new_data[:, 0], s = 5, color = new_data[:, 3:], alpha=1) # slice
plt.xlim(-6, 11) #設定x軸顯示範圍
plt.ylim(-4, 8) #設定y軸顯示範圍
plt.axis("off")

plt.savefig("map.png", bbox_inches='tight', pad_inches = 0)
plt.show()
```



- First I use the img to change in gray level and erode it (the upper left image), and then change it to the binary image (0 and 1) to check the collision of the RRT. The left image is the result.

```
img = cv2.imread(args.imagePath, cv2.IMREAD_GRAYSCALE) # load grayscale maze image
kernel = np.ones((3,3), np.uint8)
img = cv2.erode(img, kernel, iterations = 1)  #影像侵蝕
t1, img = cv2.threshold(img, 253, 255, cv2.THRESH_BINARY)
```

- In the **def RRT fuction**, I use the check_collision(nx, ny, nearest_x, nearest_y) to check is collosion or not. nx and ny is a random point that we spread, and then use the **def nearest_node(x,y) function** to find the nearest point beside the nx and ny. Having these point and then check the collision between random point and nearest point. If the orientation of it has no collision, it will create the point in this orientation in the stepsize that I set.

```
def check_collision(x1, y1, x2, y2):
    _,theta = dist_and_angle(x2, y2, x1, y1)
    x=x2 + stepSize* np.cos(theta)
```

```
    y=y2 + stepSize* np.sin(theta)
    print(x2, y2, x1, y1)
    print("theta", theta)
    print("check_collision", x, y)

    # TODO: trim the branch if its going out of image area
    # print("Image shape",img.shape)
    hy,hx=img.shape
    if y<0 or y>hy or x<0 or x>hx:
        print("Point out of image bound")
        directCon = False
        nodeCon = False
    else:
        # check direct connection
        if collision(x,y,end[0], end[1]):
            directCon = False
        else:
            directCon=True

        # check connection between two nodes
        if collision(x, y, x2, y2):
            nodeCon = False
        else:
            nodeCon = True

    return(x, y, directCon, nodeCon)
```

- Use the parent list to append the point and

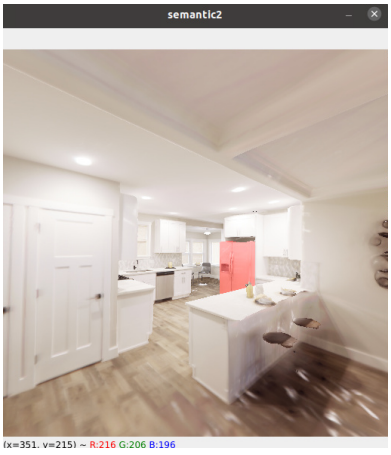
```
node_list[i].parent_x.append(tx)
node_list[i].parent_y.append(ty)
```

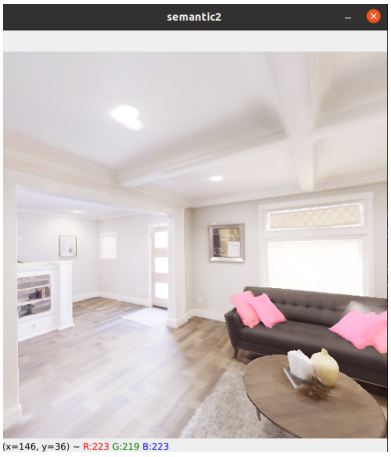
- Use the for loop to draw the blue line on the final path (parent list)

```
for j in range(len(node_list[i].parent_x)-1):
    cv2.line(img2, (int(node_list[i].parent_x[j]),int(node_list[i].parent_y[j])), (int(node_list[i].parent_x[j+1]),int(node_list[i].par
```

Discuss about the robot navigation results

the other results is in the .mp4





- In load.py, I change the semantic color in color101

```
colors = loadmat('color101.mat')['colors']
```

- And change the **def transform_semantic** below：

```
def transform_semantic(semantic_obs):
    semantic_img = Image.new("P", (semantic_obs.shape[1], semantic_obs.shape[0]))
    semantic_img.putpalette(colors.flatten())
    semantic_img.putdata(semantic_obs.flatten().astype(np.uint8))
    semantic_img = semantic_img.convert("RGB")
    semantic_img = cv2.cvtColor(np.asarray(semantic_img), cv2.COLOR_RGB2BGR)
    return semantic_img
```

- Load the RRT point that I save in the last step (RRT_position.npy) to get the camera postion in the habitat.

```
RRT_position = np.load('./RRT_position.npy')
```

- I use the **def dis_and_angel** to calculate the angel and distance in the fisrt 3 point. Because each move_forward step is 0.01m, and each turn_left and right is 1 degree, so I use the distance and angel to find out how many step that I need to implement.

```
current_norm, angle, rotate_action = dis_and_angel(np.array([RRT_position[0][0] + 1, RRT_position[0][1]]), RRT_position[0], RRT_positio
angle_step = int(angle)
for i in range(angle_step):
    cv2.waitKey(0)
    navigateAndSee(rotate_action)

# append RRT's shape because of the end point to use the dis_and_angel must have 3 point
# can use to find the target roientation by define the last point
print(RRT_position)
RRT_position = np.append(RRT_position, values = [RRT_position[-1]], axis = 0)
print(RRT_position)

x, y, z = navigateAndSee()
for step in range(RRT_position.shape[0]-1):

    current_norm, angle, rotate_action = dis_and_angel(RRT_position[step], RRT_position[step+1], RRT_position[step+2])
    forward_step = int(current_norm // 0.01)  # 取整數可以知道要走幾步
    for i in range(forward_step):
        cv2.waitKey(0)
        navigateAndSee("move_forward")
    # print(RRT_position.shape[0]-1)
    # print(step)
    if ((RRT_position.shape[0]-3)==step):
        break
    for j in range(int(angle)):
        cv2.waitKey(0)
        navigateAndSee(rotate_action)
```

- Use the dictionary to give the target color in the semantic image (check in color_coding_semantic_segmentation_classes.xlsx)

```
target_color = {'refrigerator':(255, 0, 0), 'rack':(0, 255, 133), 'cushion':(255, 9, 92), 'lamp':(160, 150, 20), 'cooktop':(7, 255, 224
```

- Use np.where to mask the target

```
chosen_color = np.where((semantic_img[:,:,2] == target_color[Input_target][0])* (semantic_img[:,:,1] == target_color[Input_target][1])*
```

- Use the image that I save before to create the video

```
path = "./video/" + Input_target + ".mp4"
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
videowriter = cv2.VideoWriter(path, fourcc, 100, (512, 512))
print(numer_of_step)
for i in range(1, numer_of_step):
    img = cv2.imread("./my_data/image"+str(i)+".png")
    videowriter.write(img)
videowriter.release()
```