A Robot Navigation Framework

Overview

一開始在**tans_map.py**裡使用助教提供之3D建模圖來做2D地圖的建立，當我們得到2D的RGB地圖之後可便可以用RRT演算法，有初始位置與終點位置後得到其移動路線，那這些演算法與路線資料可以在**birrt_final.py**程式中做實現，最後在**load_fix.py**使用habitat環境裡做導航。

2D semantic map construction

code

```
point = np.load('semantic_3d_pointcloud/point.npy')
color = np.load('semantic_3d_pointcloud/color01.npy')
point = point*10000/255
pcl = o3d.geometry.PointCloud()
pcl.points = o3d.utility.Vector3dVector(point)
pcl.colors = o3d.utility.Vector3dVector(color)
```

在一開始匯入.np檔，並且將其單位轉換到與模擬環境一樣，最後宣告成點雲型態。

```
points = np.asarray(pcl.points)
pcl = pcl.select_by_index(np.where((-1.2<points[:,1])*(points[:,1] < -0.1))[0])
o3d.visualization.draw_geometries([pcl])
point = np.asarray(pcl.points)
color = np.asarray(pcl.colors)
```

利用np.where來濾出天花板與地板，也就是座標裡面的y方向。

```
px = 1/plt.rcParams['figure.dpi']  # pixel in inches
plt.subplots(figsize=(1700*px, 1100*px))
plt.scatter(point[:,2], point[:,0], s=5, c=color)
plt.xlim(-6, 11)
plt.ylim(-4, 7)
plt.axis('off')
plt.savefig('mapp.png', bbox_inches='tight', pad_inches=0)
plt.show()
```

用figsize來規定我要的地圖圖片大小，用plt.scatter把我要的x、z座標資訊都呈現在2D圖上，並且規定好其軸的範圍（因為事先知道地圖大小），最後不要讓座標軸顯示，並可以得到一張2D地圖

Result

RRT Algorithm

code

```
class Nodes:
    def __init__(self, x,y):
        self.x = x
        self.y = y
        self.parent_x = []
        self.parent_y = []
```

Class to store the RRT graph，宣告一種資料型態Node作為tree裡面的節點，每一個節點會儲存自己的座標(x,y)還有其上游的所有node，這樣在找到終點後可以直接回傳路徑。

接下來介紹我的RRTree演算法。

class RRTree

- **init**

```
def __init__(self, stepsize, map_image,iter_num):
        img = cv2.imread(map_image,0) # load grayscale maze image
        img[np.where((img[:,:] != 255))] = 0
        kernel = np.ones((3,3), np.uint8)
        dilation = cv2.erode(img, kernel, iterations = 12)
        # cv2.imshow('dilation',dilation)
        # cv2.waitKey(0)
        # cv2.destroyAllWindows()

        self.img = dilation # load grayscale maze image
        self.img2 = cv2.imread(map_image) # load colored maze image
        self.K = iter_num
        self.start = 0
        self.end = 0
        self.stepsize= stepsize
        self.node_list_start = [0]
        self.node_list_end = [0]
        self.end_final = " "
```

在class的init將匯入我們在前一個task所做出來的map，這個map會有兩種用途因此會被存成img和img2，img2就是單純拿來做路徑顯示的全彩圖，至於img2，在這次撰寫rrt的方法裡面，我使用黑白圖來讓rrt生成路徑，因此在一開始把不是全白的部分都轉為黑色，也就是障礙物，最後會利用openCV的侵蝕，讓障礙物大小變大一圈，讓過程比較不容易撞到障礙物。

- **choice_final**

```
def choice_final(self,end_final):
        final = {"refrigerator":(255, 0, 0),"rack":(0, 255, 133),"cushion":(255, 9, 9
2),"lamp":(160, 150, 20),"cooktop":(7, 255, 224)}
        r = final[end_final][0]
        g = final[end_final][1]
        b = final[end_final][2]
        index = np.where((self.img2[:,:,0] == b)*(self.img2[:,:,1] == g)*(self.img2
[:,:,2] == r))
```

```
        x = int(np.mean(index[0]))
        y = int(np.mean(index[1]))

        h = [x,y+1]
        d = [x, y-1]
        l = [x-1,y]
        r = [x+1,y]
        # print(x,y)
        while(1):
            k = 1
            h[1] = h[1]+k
            d[1] = d[1]-k
            l[0] = l[0]-k
            r[0] = r[0]+k
            if self.img[d[0],d[1]] == 255:
                x,y = d
                break
            elif self.img[h[0],h[1]] == 255:
                x,y = h
                break
            elif self.img[l[0],l[1]] == 255:
                x,y = l
                break
            elif self.img[r[0],r[1]] == 255:
                x,y = r
                break
      self.end = (y,x)
```

選擇目標物的部分，我會先利用一個dictionary把字串相對應的顏色給記下來，那當我們輸入相對應的字串後，就會去計算所有相對應顏色的質心，當得到質心後，會向一個十字一般去擴張，尋找可以到的點也就是白色的piexl，那個點就是我們要到去的終點。

- **compute**

```
def compute(self,end_final):
        if(end_final!=" "):
            self.choice_final(end_final)
            print("find object")
        h,l= self.img.shape # dim of the loaded image

        # insert the starting point in the node class
        self.node_list_start[0] = Nodes(self.start[0],self.start[1])
        self.node_list_start[0].parent_x.append(self.start[0])
        self.node_list_start[0].parent_y.append(self.start[1])
        self.node_list_end[0] = Nodes(self.end[0],self.end[1])
        self.node_list_end[0].parent_x.append(self.end[0])
        self.node_list_end[0].parent_y.append(self.end[1])
```

在一開始先判斷有沒有字串輸入，若有將會生成終點，若沒有則需要自己點選;並且在這個部分會初始化rrtree，將node_list_start放入第一個節點，起點;將.node_list_end放入第一個節點，終點。

```
  flag = -1
  i = 1
  while i<self.K:
      if flag == -1:
          Ta = self.node_list_start.copy()
          Tb = self.node_list_end.copy()
      else:
          Ta = self.node_list_end.copy()
          Tb = self.node_list_start.copy()
      #在地圖中的隨機點
      nx,ny = rnd_point(h,l)
      #if extend1 trap
      nearest_ind = nearest_node(nx,ny,Ta)
      nearest_x = Ta[nearest_ind].x
      nearest_y = Ta[nearest_ind].y
      tx,ty,nodeCon = extrend1(nx,ny,nearest_x,nearest_y,self.stepsize,self.img)
      if nodeCon:
          ##把點加入Ta
          Ta.append(i)
          Ta[i] = Nodes(tx,ty)
          Ta[i].parent_x = Ta[nearest_ind].parent_x.copy()
          Ta[i].parent_y = Ta[nearest_ind].parent_y.copy()
          #parent裡面就是start到自己的路徑
          Ta[i].parent_x.append(tx)
          Ta[i].parent_y.append(ty)
          # display
          cv2.circle(self.img2, (int(tx),int(ty)), 2,(0,0,255),thickness=3, lineType=8)
          cv2.line(self.img2, (int(tx),int(ty)), (int(Ta[nearest_ind].x),int(Ta[nearest_in
d].y)), (0,255,0), thickness=1, lineType=8)
          cv2.imwrite("media/"+str(i)+".jpg",self.img2)
          cv2.imshow("image",self.img2)
          cv2.waitKey(1)
          #if extend2 connect
          directCon,index = extrend2(Ta,Tb,self.img)
          if directCon :
              print("Path has been found")
              path = []
              cv2.line(self.img2, (int(tx),int(ty)), (int(Tb[index].x),int(Tb[index].y)),
 (0,255,0), thickness=1, lineType=8)
              if flag == -1:
                  for i in range(len(Ta[-1].parent_x)):
                      path.append((Ta[-1].parent_x[i],Ta[-1].parent_y[i]))
                      pass
                  Tb[index].parent_x.reverse()
                  Tb[index].parent_y.reverse()
                  for i in range(len(Tb[index].parent_x)):
                      path.append((Tb[index].parent_x[i],Tb[index].parent_y[i]))
```

```
                else:
                    for i in range(len(Tb[index].parent_x)):
                        path.append((Tb[index].parent_x[i],Tb[index].parent_y[i]))
                        pass
                    Ta[-1].parent_x.reverse()
                    Ta[-1].parent_y.reverse()
                    for i in range(len(Ta[-1].parent_x)):
                        path.append((Ta[-1].parent_x[i],Ta[-1].parent_y[i]))
                for i in range(len(path)-1):
                    cv2.line(self.img2, (int(path[i][0]),int(path[i][1])), (int(path[i+1]
[0]),int(path[i+1][1])), (255,0,0), thickness=2, lineType=8)
                cv2.waitKey(1)
                cv2.imwrite("media/"+str(i)+".jpg",self.img2)
                if(self.end_final == " "):
                    cv2.imwrite("birrt.jpg",self.img2)
                else:
                    cv2.imwrite("path/"+self.end_final+".jpg",self.img2)
                break
        else:
            continue
        if flag == -1:
            flag = flag*(-1)
            self.node_list_start = Ta.copy()
            self.node_list_end = Tb.copy()
        else:
            flag = flag*(-1)
            i = i+1
            self.node_list_start = Tb.copy()
            self.node_list_end = Ta.copy()
```

在一開始會先判斷，這次tree是誰要擴張，我的策略是迴圈裡會先選擇一個要擴張的tree，那random出一個點之後，朝random點的方向有一個stepsize的移動，得到新的點之後去判斷這個點會不會發生碰撞，如果不會將點加入這個tree，並且以這個點去判斷與另外一個tree的最近點是否可以連線，若連線代表找到終點，若不行則交換，換成另一個tree來進行擴張，並且執行一樣的步驟。那最後一組if-else的重點就是做交將這次擴張完的ta,tb存回node_list_start,node_list_end，並且調整flag，在迴圈一開始會透過flag來判斷哪一個tree要擴張。

```
if i==self.K:
    print("Can not find the path")
    return 1
print("number of iter: ",i*2)
path = np.asarray(path)
tran1 = l/17
tran2 = h/11
path[:,0] = path[:,0]/tran1-6
path[:,1] = 7-path[:,1]/tran2
```

```
print(path)
return path
```

我會設定一定數量的迭代次數，若超過我所設定的迭代次數，代表找不到路徑，可以直接return 1，那如果可以找到路徑，我會把路徑轉成3D座標的x,z，而這個轉換關係是將piexl之(u,v)除以(寬,高)乘上座標的長寬，最後做平移，即可以找到路徑。

Result

Robot navigation

code

```
test_scene = "apartment_0/habitat/mesh_semantic.ply"
path = "apartment_0/habitat/info_semantic.json"
colors = loadmat('color101.mat')['colors']
colors = np.insert(colors, 0, values=np.array([[0,0,0]]), axis=0)

#global test_pic
#### instance id to semantic id
```

```
with open(path, "r") as f:
    annotations = json.load(f)

id_to_label = []
instance_id_to_semantic_label_id = np.array(annotations["id_to_label"])
for i in instance_id_to_semantic_label_id:
    if i < 0:
        id_to_label.append(0)
    else:
        id_to_label.append(i)
id_to_label = np.asarray(id_to_label)

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
```

在模擬環境中使用之前我們必須先對label過後的照片去做color101的上色，而且我們發現必須先對匯入的顏色資料多加上一格，這樣顏色才會上對。

```
# Set agent state
point = np.load('path.npy') #load the path
start = point[0]
print(start)
agent_state = habitat_sim.AgentState()
agent_state.position = np.array([start[1], 0.0, start[0]])  # agent in world space
```

在一開始初始化agent的部分我把起點設在我所生成路徑點的初始點。

```
def navigateAndSee(action=""):
    if action in action_names:
        observations = sim.step(action)
        #print("action: ", action)
        RGB_img = transform_rgb_bgr(observations["color_sensor"])
        SEIMEN_img =  transform_semantic(id_to_label[observations["semantic_sensor"]])
        index = np.where((SEIMEN_img[:,:,0]==b)*(SEIMEN_img[:,:,1]==g)*(SEIMEN_img[:,:,2]=
=r))
        if len(index[0]) != 0:
            RGB_img[index] = cv2.addWeighted(RGB_img[index], 0.6, SEIMEN_img[index], 0.4,
 50)
        cv2.imshow("RGB", RGB_img)
        cv2.waitKey(1)
        videowriter.write(RGB_img)
        agent_state = agent.get_state()
```

```
        sensor_state = agent_state.sensor_states['color_sensor']
        return sensor_state
```

那每一次尋找目標點的時候我會利用semantic圖來找到我要的物體在2D圖上面的pixel座標，並且利用cv2.addWeighted這個函式可以把rgb圖裡出現目標物的位置用其label後所定義的顏色給label起來。

```
def driver(pre_node,start,end):
    #part1 rotate
    print("ok")
    if pre_node == []:
        v1 =np.array([-1,0])
    else:
        v1 = np.array([start[0]-pre_node[0],start[1]-pre_node[1]])
    v2 = np.array([end[0]-start[0],end[1]-start[1]])
    print(v1,v2)
    flag = v1[0]*v2[1]-v1[1]*v2[0]
    value = v1@v2
    v1 = math.sqrt((v1[0])**2+(v1[1])**2)
    v2 = math.sqrt((v2[0])**2+(v2[1])**2)

    goal_ry = int(math.acos(value/(v1*v2))/math.pi*180)
    print(goal_ry)
    print("rotate number",int(goal_ry))
    if(flag>=0):
        action = "turn_left"
    else:
        action = "turn_right"
    for i in range(int(abs(goal_ry))):
        sensor_state = navigateAndSee(action)
    #part2 goforword
    action = "move_forward"
    forward_distance = math.sqrt((end[0]-start[0])**2+(end[1]-start[1])**2)
    step = int(forward_distance/0.01)
    for i in range(step):
        sensor_state = navigateAndSee(action)
    x = sensor_state.position[0]
    z = sensor_state.position[2]
    return (z,x)
```

我driver的策略是，每一個點與前一個點和後一個點都會有一個向量，那我可以裡用這兩個向量的到一個角度，此角度便是我必須要轉到的角度，當角度轉完後則要開始直走，直走的步數我用現在點與下一個點之間的歐式距離，除以我在環境中所設定的距離。

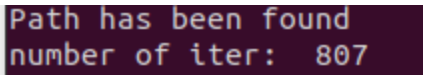
```
pre_node = []
start = point[0]
```

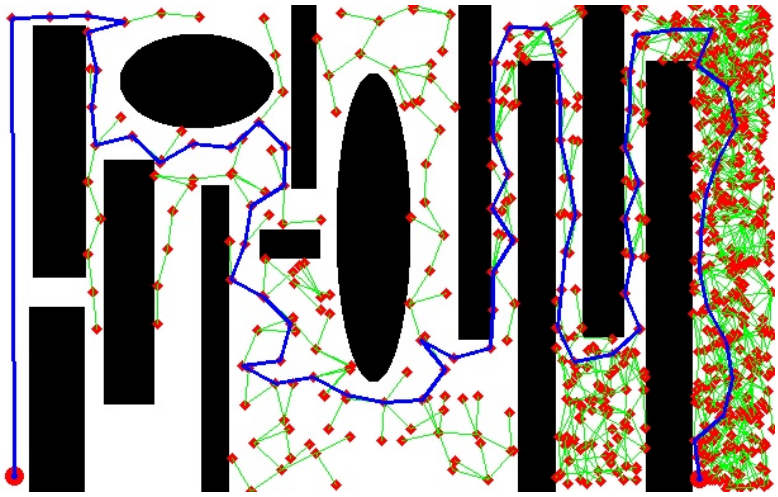
```
final = {"refrigerator":(255, 0, 0),"rack":(0, 255, 133),"cushion":(255, 9, 92),"lamp":(16
0, 150, 20),"cooktop":(7, 255, 224)}
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description = 'Below are the params:')
    parser.add_argument('-f', type=str, default=" ",metavar='END', action='store', dest='E
nd',
                help='Where want to go')
    args = parser.parse_args()

    # save video initial
    path = "video/" + args.End + ".mp4"
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    videowriter = cv2.VideoWriter(path, fourcc, 100, (512, 512))

    end_rgb = final[args.End]
    r = end_rgb[0]
    g = end_rgb[1]
    b = end_rgb[2]
    for i in range(len(point)-1):
        temp = start
        start = driver(pre_node,start,point[i+1])
        pre_node = temp
    videowriter.release()
```

在main裡面一樣要在執行程式之前先輸入要mark起來的物體也是我們所想要到達的終點，並且初始化要存影片的cv2.VideoWriter;在for迴圈裡，每一次的初始點會是下一次的pre_node，目標點會變起點，然在路徑裡找到新的point，直到達到終點。

Bonus

根據單個rrt生成之路徑，在這張圖的情況下，需要生成807個點才可以找到終點





但如果使用bi-rrt，讓終點與起點同時長點，可以在使用比較少得點之情況下找到終點，且也會比較容易收斂。

```
Path has been found
number of iter:  170
```