机器人导论第三次作业

李钰 19335112

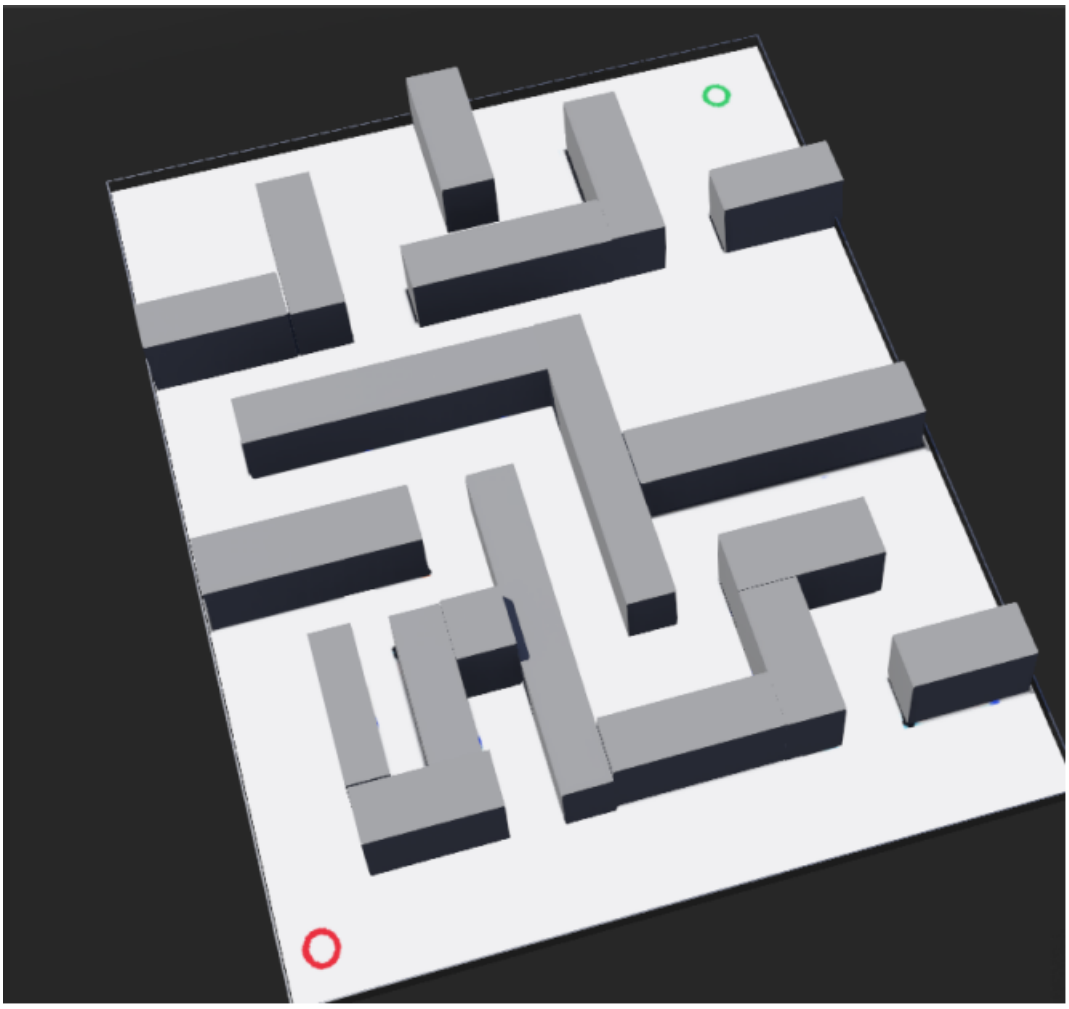
作业题目：使用PRM算法进行路径规划

实验要求

绿色方块代表起始位置，红色方块代表目标位置，要求在已知地图全局信息的情况下，规划一条尽可能短的轨迹，控制机器人从绿色走到红色

实验场景

给定了迷宫webots模型，地图的全局信息通过读取maze.png这个图片来获取



实验思路

本次实验可以分为两个步骤

图像处理——PRM算法进行路径规划

我们可以预先对地图进行处理，首先将地图转换为灰度图像，再将该图像转换为二维矩阵存储，之后利用PRM算法对路径进行规划，避开黑色障碍，绘制出一条从起点走向终点的路径。本块内容代码由python实现，可以直接调用python的一些库，对图像的处理简单些。

机器人巡线

经过上一步骤之后，用黑色点描出了机器人的路线。结合上一实验中我们已经做好的利用照相机视觉巡线的机器人小车，针对本环境中做一些细节的调整即可。

具体过程（含代码说明）

图像处理

- PRM算法
  - 初始化。设 $G(V,E)$ 为一个无向图，其中顶点集 V 代表无碰撞的构型，连线集 E 代表无碰撞路径。初始状态为空。
  - 构型采样。从构型空间中采样一个无碰撞的点 $\alpha(i)$ 并加入到顶点集 V 中。
  - 领域计算。定义距离 ρ，对于已经存在于顶点集 V 中的点，如果它与 $\alpha(i)$ 的距离小于 ρ，则将其称作点 $\alpha(i)$ 的邻域点。
  - 边线连接。将点 $\alpha(i)$ 与其领域点相连，生成连线 τ。
  - 碰撞检测。检测连线 τ 是否与障碍物发生碰撞，如果无碰撞，则将其加入到连线集 E 中。
  - 结束条件。当所有采样点（满足采样数量要求）均已完成上述步骤后结束，否则重复2-5。
- 全局变量

```
BLACK='B'       #用来标记障碍物
WHITE='W'       #用来标记白色可行走区域
BORDER = '@'    #用来膨胀障碍物，标记障碍物边界
```

- DrawMap()类，用来将图片读入，并将其转换为有障碍物的二维网格化地图，并提供计算欧几里得距离、曼哈顿距离等操作。
  - 初始化函数：先使用 `img.convert('L')` 将图形转换为灰度图片，黑色值为0，白色值为255。之后将灰度图片的数值转换为二维数组，并对该数组进行扫描，对于值为255的元素，将其替换为"WHITE"，否则替换为“BLACK”，转换到map中。因为在webots世界中，每个墙体都会存在一些阴影，所以这里为了减少阴影对小车巡线的影响，我们对其进行一定值的膨胀。我的膨胀方法是，对于原来是“WHITE”的元素，如果它左边5个像素点或右边5个像素点的范围内是"BLACK"的话，我将其原来的"WHITE"值改为"BORDER"。

```
def __init__(self,imgage_file):
    temp_map = []
    img = Image.open(imgage_file)
    img_gray = img.convert('L')  # 地图灰度化,0-black,255-white
    img_arr = np.array(img_gray)
    img_binary = np.where(img_arr<127,0,255)

    for x in range(img_binary.shape[0]):
        temp_row = []
        for y in range(img_binary.shape[1]):
            status = WHITE if img_binary[x,y] == 255 else BLACK
            temp_row.append(status)
            temp_map.append(temp_row)

            self.map = temp_map
            self.cols = len(self.map[0])
            self.rows = len(self.map)

            '''对障碍物稍做膨胀，'''
```

```
            for x in range(0, self.rows):
                for y in range(0, self.cols):
                    if self.map[x][y] == WHITE:
                        for i in range(0,5):
                            if y > i and y < self.cols - i and
(self.map[x][y-i] == BLACK or self.map[x][y+i] == BLACK):
                                self.map[x][y] = BORDER
```

- 计算两个像素点之间的欧几里得距离

```
def e_distance(self, xy1, xy2):
    dis = 0
    for (x1, x2) in zip(xy1, xy2):
        dis += (x1 - x2)**2
        return dis**0.5
```

- 计算两个像素点之间的曼哈顿距离

```
def m_distance(self,xy1,xy2):
    dis = 0
    for x1,x2 in zip(xy1,xy2):
        dis+=abs(x1-x2)
        return dis
```

- 检测两个点的连线是否经过障碍物：如果两点之间的像素值不是"WHITE"，则说明这两点之间经过了障碍物或障碍物边界，需要舍弃

```
def out_black(self, xy1, xy2):
    steps = max(abs(xy1[0]-xy2[0]), abs(xy1[1]-xy2[1]))
    xs = np.linspace(xy1[0],xy2[0],steps+1)
    ys = np.linspace(xy1[1],xy2[1],steps+1)
    for i in range(1, steps - 1):
        if self.map[math.ceil(xs[i])][math.ceil(ys[i])] != WHITE:
            return False
        if self.map[math.ceil(xs[i] - 2)][math.ceil(ys[i])] != WHITE:
            return False
        if self.map[math.ceil(xs[i])][math.ceil(ys[i] - 1)]:
            return False
    return True
```

- 绘制路径：按照map结合已经形成的path重新画图，在"BLACK"处填充黑色，赋值为0；接着对在path中的(x,y)点也赋值为0，填充为黑色并显示

```
def drawpath(self,path):
        out = []
        for x in range(self.rows):
            temp = []
            for y in range(self.cols):
                if self.map[x][y]==BLACK:
                    temp.append(0)
                else:
                    temp.append(255)
            out.append(temp)
        for x,y in path:
```

```
                out[x][y] = 0
                out[x][y - 1] = 0
                out[x - 1][y] = 0
                if x + 1 < self.rows - 1:
                    out[x + 1][y] = 0
                if y + 1 < self.cols - 1:
                    out[x][y + 1] = 0
            out = np.array(out)
            img = Image.fromarray(np.uint8(out))
            img.show()
```

- PRM类，该类继承DrawMap类，主要对刚刚生成的地图进行PRM路径规划
  - 初始函数，相关参数包括起点终点坐标，采样点个数以及领域距离

```
def __init__(self, img_file, **param):
    DrawMap.__init__(self,img_file)
    self.num_sample = param['num_sample'] if 'num_sample' in param else
100
    self.distance_neighbor = param['distance_neighbor'] if
'distance_neighbor' in param else 100
    self.G = nx.Graph() # 无向图
```

  - learn()函数，进行一次，对num_sample个采样点进行学习，随机取点，判断是否在障碍物中，如果不是加入无向图G中，对G中的点进行领域范围内的检验，如果两点之间的连线不经过障碍物，则可以将改变加入到无向图G中，加入的变的权重为欧几里得距离。

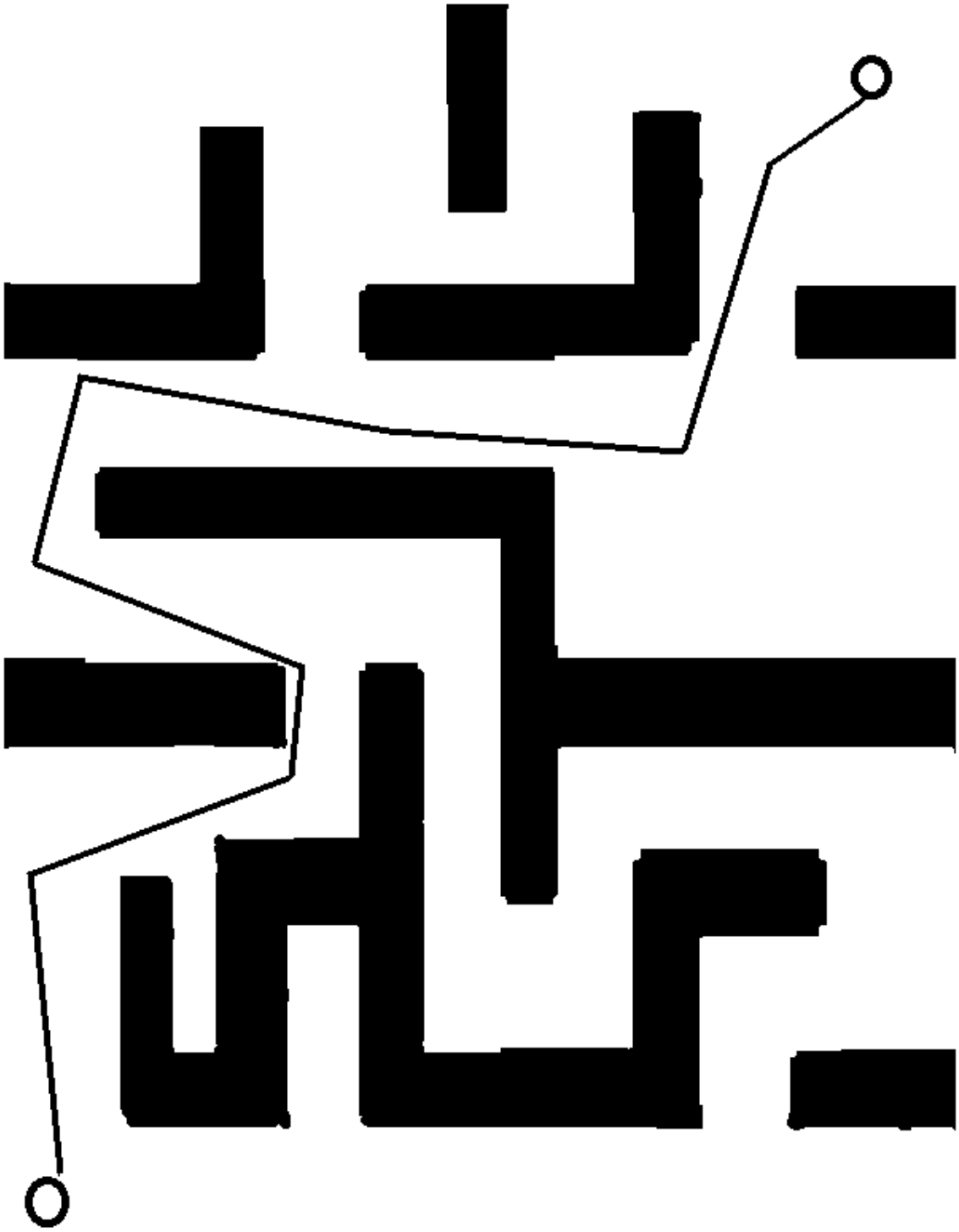
```
def learn(self):
    # 随机采样节点
    while len(self.G.nodes)<self.num_sample:
        XY = (np.random.randint(0, self.rows),np.random.randint(0,
self.cols)) # 随机取点
        if self.is_valid_xy(XY[0],XY[1]) and self.map[XY[0]][XY[1]] ==
WHITE: # 不是障碍物点
            self.G.add_node(XY)
    # 邻域范围内进行碰撞检测，加边
    for node1 in self.G.nodes:
        for node2 in self.G.nodes:
            if node1==node2:
                continue
            dis = self.e_distance(node1,node2)
            if dis<self.distance_neighbor and
self.out_black(node1,node2):
                self.G.add_edge(node1,node2,weight=dis) # 边的权重为 欧几里
得距离
```

  - 规划路径函数:设置起始位置和终点位置为两个圆圈处，并将起点和终点加入到图中。

```
def find_path(self,startXY=None,endXY=None):
    """ 使用学习得到的无障碍连通图进行寻路"""
    # 寻路时再将起点和终点添加进图中，以便一次学习多次使用
    temp_G = copy.deepcopy(self.G)
    startXY = tuple(startXY) if startXY else (60, self.cols-60)
    endXY = tuple(endXY) if endXY else (self.rows-65,35)
    temp_G.add_node(startXY)
    temp_G.add_node(endXY)
```

```
    for node1 in [startXY, endXY]: # 将起点和目的地连接到图中
        for node2 in temp_G.nodes:
            dis = self.e_distance(node1,node2)
            if dis<self.distance_neighbor and
self.out_black(node1,node2):
                temp_G.add_edge(node1,node2,weight=dis) # 权重欧几里得距离
    # 直接调用networkx中求最短路径的方法
    path = nx.shortest_path(temp_G, source=startXY, target=endXY)
    return self.construct_path(path)
```

- 生成的地图如下



机器人巡线

我直接将上一个实验中的小车导出到本次实验的世界中，由于小车相对墙体太大了，我相应的修改了车体、轮子和照相机的scale值进行适当的缩小，并将小车放到起点处，使得照相机中可以拍到他即将要走的路径，并且对新的世界中设置摩擦系数，这里参考实验一中的设置。接着小车就可以巡线啦。

试验过程中遇到的问题

- 编写代码时，对图像转换的几个函数没搞清楚，经常有链表溢出的情况。$img.convert('L')$转换的灰度图像，每个像素用8个bit来表示的。
- 墙体阴影对小车产生影响
    - 解决：在绘制地图时对障碍物进行膨胀处理，为其增加边界状态，如果在边界的话在之后PRM学习时也看作是在障碍物中，并且缩小小车巡线对于黑色的界定范围。
- 更改小车大小时不小心把车轮的位置设错了，导致小车不能走动——不能只简单修改scale值，其他位置参数也要做相应的调整。
- 由于PRM是随机点确定的路径，所以最终生成的还是和实际有误差