使用 PRM 算法进行路径规划

19335156 毛羽翎

2021 年 12 月 8 日

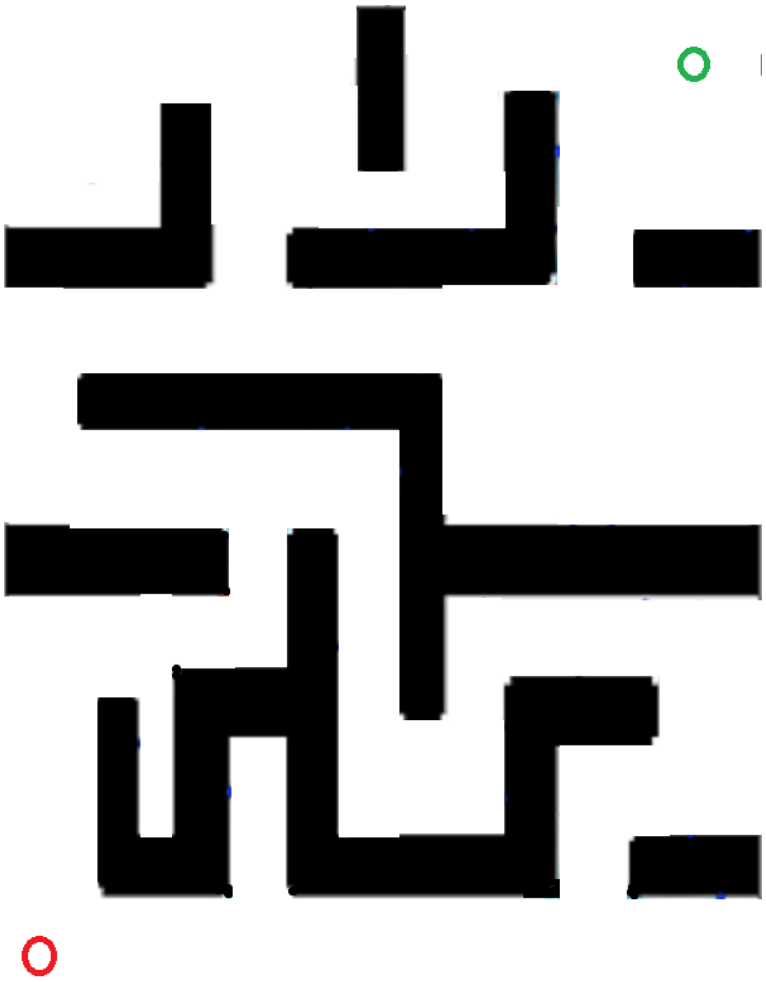
目录

1 实验目的

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

图片如下：



2 实验内容与步骤

我的想法大概是先用 PRM 算法规划最短路径，并将其标记在图中，并用上次的自动巡线代码控制小车沿着线走。

2.1 用 PRM 算法规划最短路径并在图像上标注出来

PRM 是一种基于图搜索的方法，它将连续空间转换成离散空间，再利用搜索算法在路线图上寻找路径，以提高搜索效率。这种方法能用相对少的随机采样点来找到一个解，对多数问题而言，相对少的样本足以覆盖大部分可行的空间，并且找到路径的概率为 1（随着采样数增加，P（找到一条路径）指数的趋向于 1）。显然，当采样点太少，或者分布不合理时，PRM 算法是不完备的，但是随着采用点的增加，也可以达到完备。所以 PRM 是概率完备且不最优的。

PRM 算法的伪代码如下：

Algorithm 6 Roadmap Construction Algorithm

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
  repeat
    $q \leftarrow$ a random configuration in $\mathcal{Q}$
  until q is collision-free
  $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
  $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
  for all $q' \in N_q$ **do**
    if $(q, q') \notin E$ **and** $\Delta(q, q') \neq$ NIL **then**
      $E \leftarrow E \cup \{(q, q')\}$
    end if
  end for
end for

算法主要分为两个步骤，一个是学习阶段，主要通过在图片上随机撒点构建一个无向图。还有一个是查询阶段，即查询从一个起点到一个终点的路径。

我的代码大致如下：

1. 先读入图片并将其灰度化，然后转化为二维数组，并标记出障碍物：

```
img = Image.open(img_file)
img_gray = img.convert('L')
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]):
        if img_binary[x,y] == 0:
            status = '#'
        else:
            status = '.'
        temp_row.append(status)
    test_map.append(temp_row)
```

2. 学习阶段

随机撒点，并将两点之间没有障碍物的边赋给不同的权：

```
while len(self.G.nodes)<self.num_sample:
    XY = (np.random.randint(0, self.rows),np.random.randint(0, self.cols))
    if self.is_valid_xy(XY[0],XY[1]) and self.map[XY[0]][XY[1]] != '#':
        self.G.add_node(XY)
for node1 in self.G.nodes:
    for node2 in self.G.nodes:
        if node1==node2:
            continue
        dis = self.EuclidenDistance(node1,node2)
        if dis<self.distance_neighbor and self.check_path(node1,node2):
            self.G.add_edge(node1,node2,weight=dis)
```

3. 创建路线：

这里为了找出起点的位置我试了很多值，最后终于大致确定了一个：

```
temp_G = copy.deepcopy(self.G)
startXY = tuple(startXY) if startXY else (self.rows-65,30)
endXY = tuple(endXY) if endXY else (60, self.cols-65)
```

然后将其连入图中。

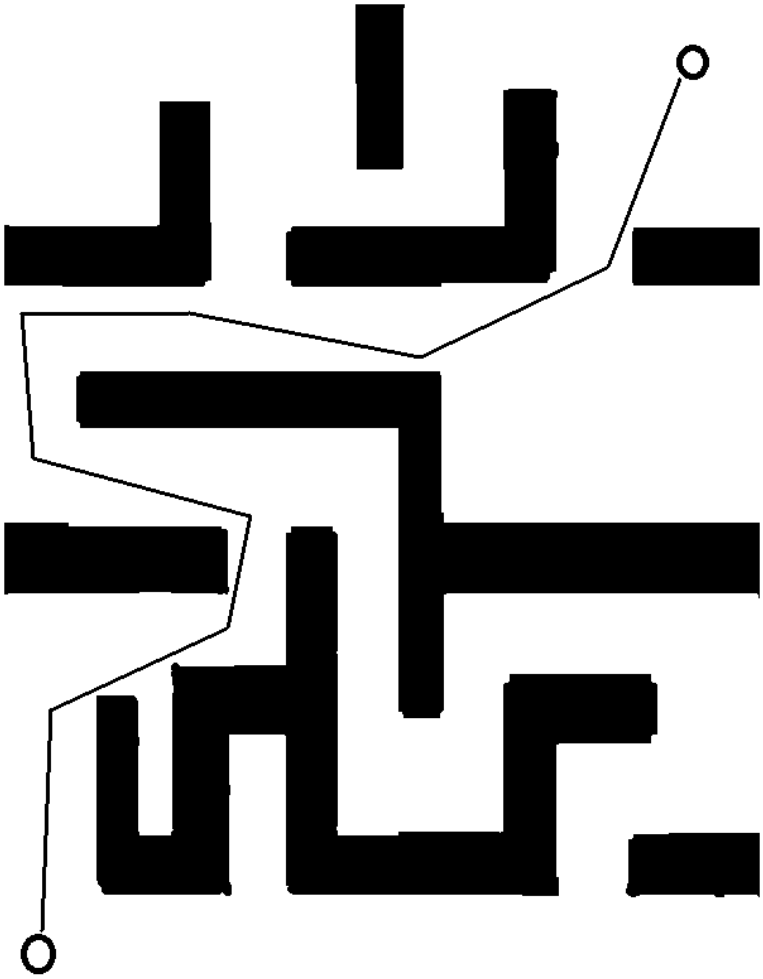
之后最短路径我直接调用了 networkx 中的库函数，这里不过多赘述。

4. 输出图像

这里基本就是读入的逆过程。值得一提的是路线的绘制：

```
        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()
```

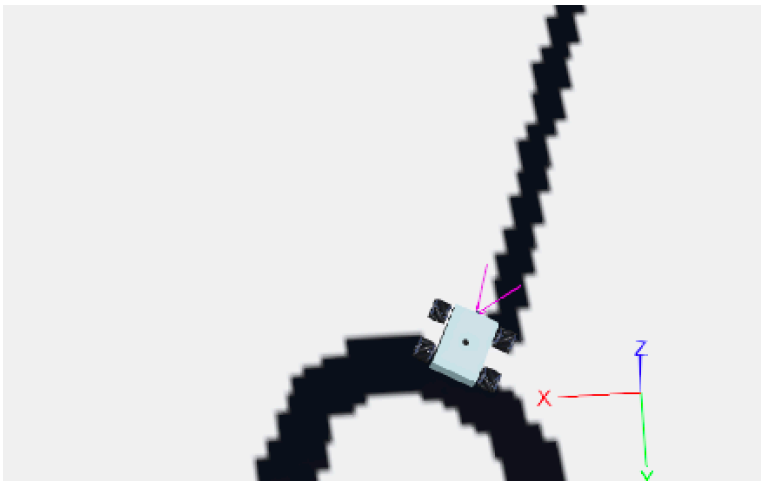
下面是输出的结果：



我们将 webots 的 world 中的背景图替换为这个

2.2 将小车与摄像头到现在的 world 中

这里我偷了个懒，没有重新创建一个小车和摄像头，直接复制粘贴上次的小车并修改尺寸，改小尺寸主要是为了方便在迷宫中行动，之前太大了连弯都拐不过去。具体如下：



前面的线为小车摄像头的视野。

2.3 控制器代码

这里就移植之前的巡线代码就好，用摄像头实现巡线的代码就像上次描述的一样，根据像素不同来判断行驶方向。左边三原色总值比右边大，右转，因为这时说明线在小车右侧，这里注意白色总值比黑色大；反之向左。

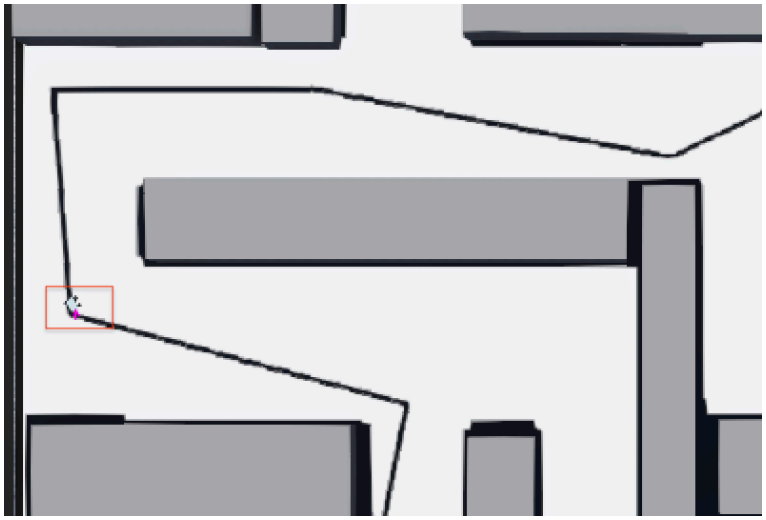
核心代码如下：

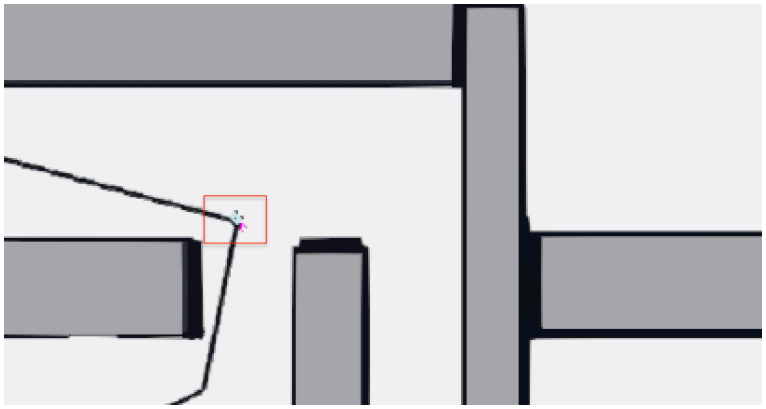
```
const unsigned char* img = camera->getImage();
int right=0,left=0;
for(int i = 0;i < 4*h/5;i++){
for(int j = 0;j<w/2;j++){
right +=camera->imageGetRed(img,w,i,j);
right +=camera->imageGetGreen(img,w,i,j);
right +=camera->imageGetBlue(img,w,i,j);
}}
for(int i = 0;i < 4*h/5;i++){
for(int j = w/2;j<w;j++){
left +=camera->imageGetRed(img,w,i,j);
left +=camera->imageGetGreen(img,w,i,j);
left +=camera->imageGetBlue(img,w,i,j);
}}
if(left-right > w*255){
for(int i = 0;i < 4;i++){
speed2[i] = speed_forward[i];

}
for(int i = 0;i < 4;i++){
speed1[i] = speed_rightCircle[i];}}

if(right-left > w*255){
for(int i = 0;i < 4;i++){
speed2[i] = speed_forward[i];
}
for(int i = 0;i < 4;i++){
speed1[i] = speed_leftCircle[i];}
}
if(right-left == w*255){
for(int i = 0;i < 4;i++){
speed2[i] = speed_forward[i];
}
for(int i = 0;i < 4;i++){
speed1[i] = speed_forward[i];}
}
```

以上步骤完成后本次实验就差不多完成了。

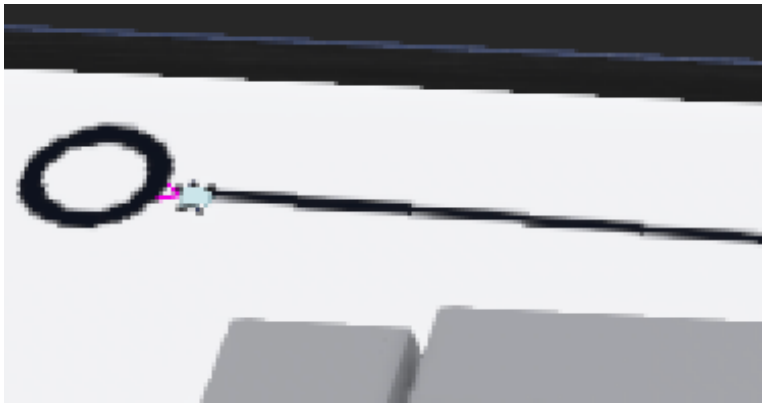
3 实验结果与分析

小车运行的录像已经放在文件夹中。运行过程中我发现我的控制器代码还是实现的比较成功的，小车基本全程没有偏离航线，几个急弯也能成功转过去：





成功到达终点!:



4 实验中的问题与解决方法

由于我预约了 12.11 号的雅思考试，且口语考试就安排在 12.8 也就是 ddl 这天的早上（这次口语又 5.5 预订了，真的不想考了），所以为了备考，这次实验完成比较匆忙，实验报告也是口语考试结束后匆忙完成的，于是中间一些部分，比如小车的控制代码和小车本身的构建，我都直接使用了上次的代码。这次实验遇到的主要问题如下：

1. 图像处理时起点坐标无法确定：开始时由于无法确定我直接让小车从右上角出发到左下角，但是发现这样规划的路线对小车的运行还是有点影响，只能多试了几次，最终还是找到一个比较合适的位置。

2. 摩擦设置：刚开始把小车丢到这个世界并运行，发现根本开不动，想起第一次实验的经验，我发现果然是没设置摩擦。

3. 控制器：刚开始小车经常出现偏航，我把上次控制器代码由：

```
int right=0,left=0;
   for(int i = 0;i < 4*h/5;i++){
   for(int j = w/4;j<w/2;j++){
   right +=camera->imageGetRed(img,w,i,j);
   right +=camera->imageGetGreen(img,w,i,j);
   right +=camera->imageGetBlue(img,w,i,j);
   }}
   for(int i = 0;i < 4*h/5;i++){
   for(int j = w/2;j<3*w/4;j++){
   left +=camera->imageGetRed(img,w,i,j);
   left +=camera->imageGetGreen(img,w,i,j);
   left +=camera->imageGetBlue(img,w,i,j);
   }}
   if(left-right > w*255){
   for(int i = 0;i < 4;i++){
```

修改为：

```
int right=0,left=0;
   for(int i = 0;i < 4*h/5;i++){
   for(int j = 0;j<w/2;j++){
   right +=camera->imageGetRed(img,w,i,j);
   right +=camera->imageGetGreen(img,w,i,j);
   right +=camera->imageGetBlue(img,w,i,j);
   }}
   for(int i = 0;i < 4*h/5;i++){
   for(int j = w/2;j<w;j++){
   left +=camera->imageGetRed(img,w,i,j);
   left +=camera->imageGetGreen(img,w,i,j);
   left +=camera->imageGetBlue(img,w,i,j);
   }}
   if(left-right > w*255){
   for(int i = 0;i < 4;i++){
   speed2[i] = speed_forward[i];

   }
```

之后发现就能正常巡线跑完全程了