南京理工大学计算机科学与工程学院

人工智能与智能机器人 大作业

题　　目 RRT 算法实现

指导教师 袁 夏

学生姓名 黄海浪

学　　号 9181040G0818

目录

一、系统环境与 GUI 介绍

1. 系统环境与编程语言

系统环境：macOS Big Sir 11.1
编程语言：Python 3.8.6 (default, Oct　8 2020, 14:06:32)

2. 选择 bokeh GUI 的原因

Bokeh 官方介绍：Bokeh 是用于现代 Web 浏览器的交互式可视化库。它提供通用图形的优雅，简洁的构造，并在大型或流数据集上提供高性能的交互性。Bokeh 可以帮助任何想要快速轻松地进行交互式绘图，仪表板和数据应用程序的人。

之前用过 Tkinter 做 GUI，但是在调试的时候总会遇到“程序”计算崩溃的问题。本次实验做的是关于算法的实验，调试过程计算出错的概率非常大，故放弃选择了类似 Tkinter 的 GUI 库。

选择 bokeh 的原因很简单，本次实验刚好绘图与计算的时间多，相较于传统的 plot 库画图更为方便并且相较于传统的绘图能够提供交互，减少了编辑代码求得结果的复杂性。并且 bokeh 提供的与人交户的界面，计算出错的时候界面由浏览器渲染，不会导致崩溃。

二、RRT 算法

本次实现的算法有三个，分别为：RRT 、RRT- Connect、RRT*。

1. RRT 算法（Rapidly-exploring random tree）

RRT 算法倾向于拓展到开放的未探索区域，只要时间足够，迭代次数足够多，没有不会被探索到的区域。

RRT 算法伪代码：

Algorithm BuildRRT
　　Input: Initial configuration q_{init}, number of vertices in RRT K, incremental distance Δq)
　　Output: RRT graph G

　　G.init(q_{init})
　　for k = 1 **to** K **do**
　　　　$q_{rand} \leftarrow$ RAND_CONF()
　　　　$q_{near} \leftarrow$ NEAREST_VERTEX(q_{rand}, G)
　　　　$q_{new} \leftarrow$ NEW_CONF(q_{near}, q_{rand}, Δq)
　　　　G.add_vertex(q_{new})
　　　　G.add_edge(q_{near}, q_{new})
　　return G

- "←" denotes assignment. For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the following value.

2. RRT-Connect

RRT-Connect 基于 RRT 搜索空间的盲目性，节点拓展环节缺乏记忆性的缺点，为了提高空间内的搜索速。在 RRT 算法的基础上加上了两棵树双向抖索的引导策略，并且在生长方式的基础上加上了贪婪策略加快了搜索速度，并且减少了空白区域的无用搜索，节省了搜索时间。

RRT-Connect 算法伪代码

Begin *RRT-Connect* **Procedure**
  T_a ← **Insert** Root Node<q_{start}> to T_a
  T_b ← **Insert** Root Node<q_{goal}> to T_b
  While 1 ← n **to** N **do**
    Generate n-th Random Sample
    q_{rand} ← Position of n-th Random Sample
    If Not *Extend*(T_a, T_b, q_{newB} ← ***Null***, q_{rand}, λ, C) **then**
      If *Connect*(P_{reach} ← ***Null*** Path, T_a, T_b, q_{newB}, λ) **then**
        d_{reach} ← Distance of P_{reach}
        If $d_{shorter}$ = 0 **or** $d_{shorter}$ > d_{reach} **then**
          R ← P_{reach}
          $d_{shorter}$ ← d_{reach}
    Swap(T_a, T_b)

3. RRT* 算法

RRT*算法增加了启发式策略，以及贪婪思想，但 RRT 算法和 RRT-Connect 算法的共同缺点是，他们的路径都不是最优的，没有添加评价路径长短花费的函数，搜索路径策略都是基于随机采样的搜索。渐进最优的 RRT*算法，该算法在原有的 RRT 算法上，改进了父节点选择的方式，采用代价函数来选取拓展节点领域内最小代价的节点为父节点，同时，每次迭代后都会重新连接现有树上的节点，从而保证计算的复杂度和渐进最优解。

RRT* 算法伪代码

$T \leftarrow$ InitializeTree();
$T \leftarrow$ InsertNode(Ø, z_{init}, T);
for i=0 to i=N ***do***
$z_{rand} \leftarrow$ Sample(i);
$z_{nearest} \leftarrow$ Nearest(T, z_{rand});
(z_{new}, U_{new}) ← Steer ($z_{nearest}$, z_{rand});
if Obstaclefree(z_{new}) ***then***
  $z_{near} \leftarrow$ Near(T, z_{new}, $|V|$);
  $z_{min} \leftarrow$ Chooseparent (z_{near}, $z_{nearest}$, z_{new});
  $T \leftarrow$ InsertNode(z_{min}, z_{new}, T);
  $T \leftarrow$ Rewire (T, z_{near}, z_{min}, z_{new});
return T

三、编程实现

本次编程分为两个主要的方面，一个是 GUI 部分，一个是 RRT 计算部分。其中 GUI 使用 bokeh 实现，使用函数对图标、按钮等控件配合全局初始化信息等创建了基于浏览器的 GUI。RRT 计算文件实现了 4 个类，分别为：核心类 RRT、点类 Node、树类 Tree、地图 Map。

1. RRT 文件

RRT 文件提供 genMainRRT 函数返回 GUI 控制的 RRT 对象。以及上述 4 个类，类中提供函数给 MainGUI 调用。

2. Main 文件

Main 文件中的 setInfo 使用了数据驱动对象，减少渲染的计算量。直接调用 RRT 的接口生成图像。

3. obstacles.json 文件

obstacles.json 为手动配置的不可到达范围（满足作业要求：手动配置）。

4. 其他

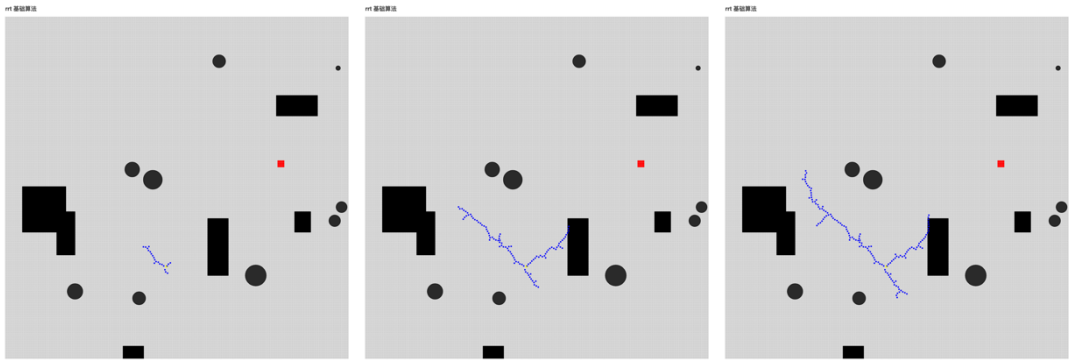
本次编程不能用 python 直接运行文件，需要使用三方库启动 http 服务，使用下面命令：bokeh serve --show main.py
具体请见代码与代码的注释。

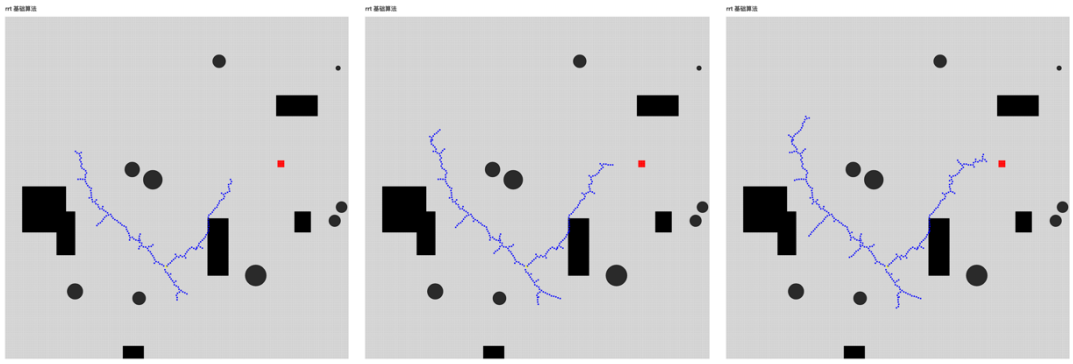
四、实现的结果

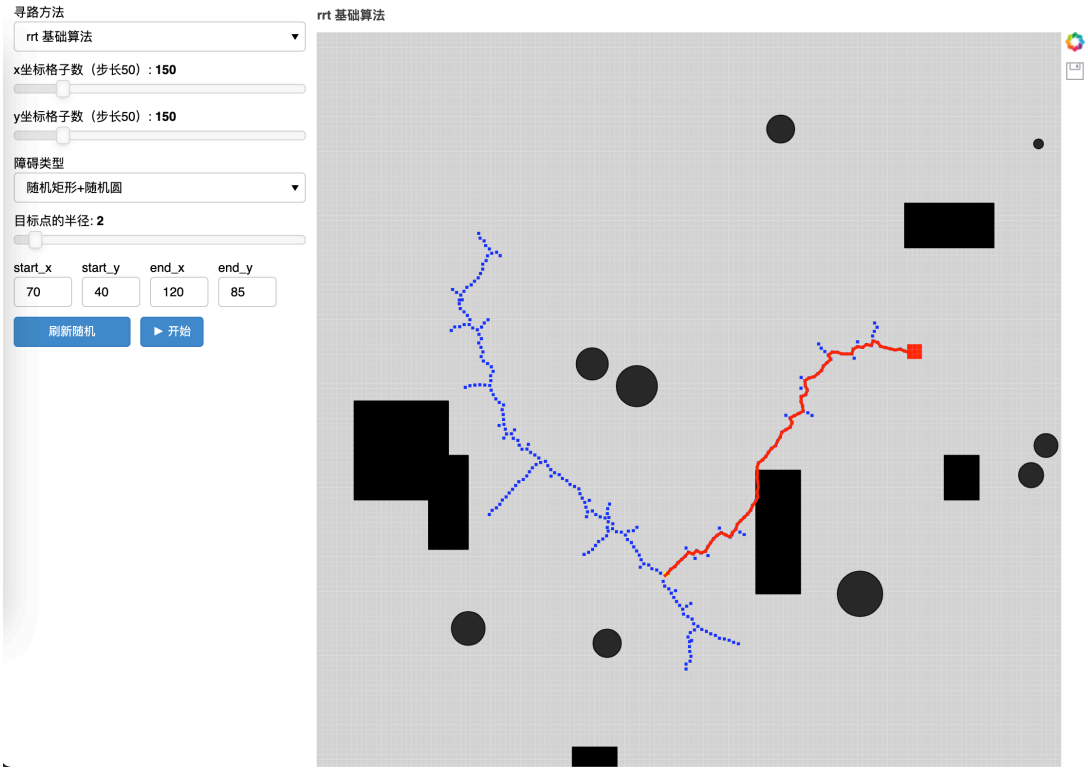
其中 1，2 为 rrt 与 rrt*的横向对比；其中 3 为 rrt- connect 使用手动地图。

1. RRT

起点[70, 40] 、终点[120, 85]，半径 2，使用随机生成地图

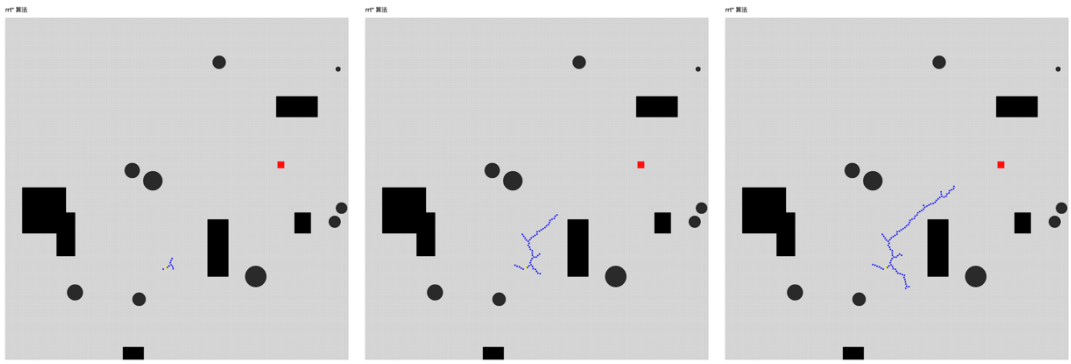


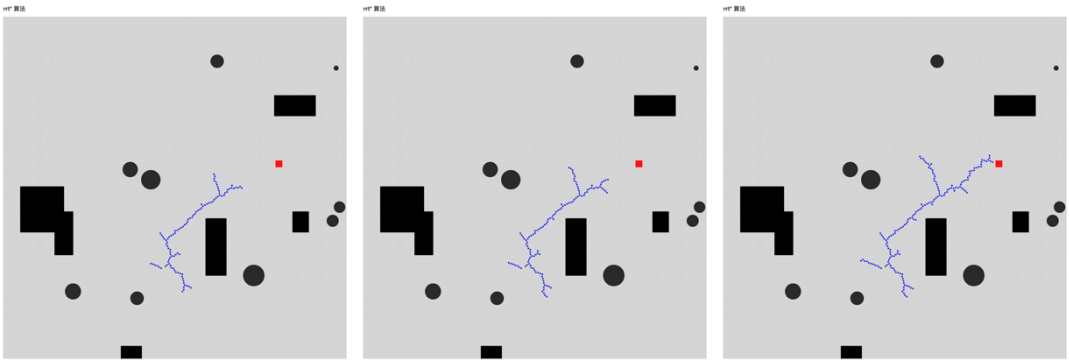


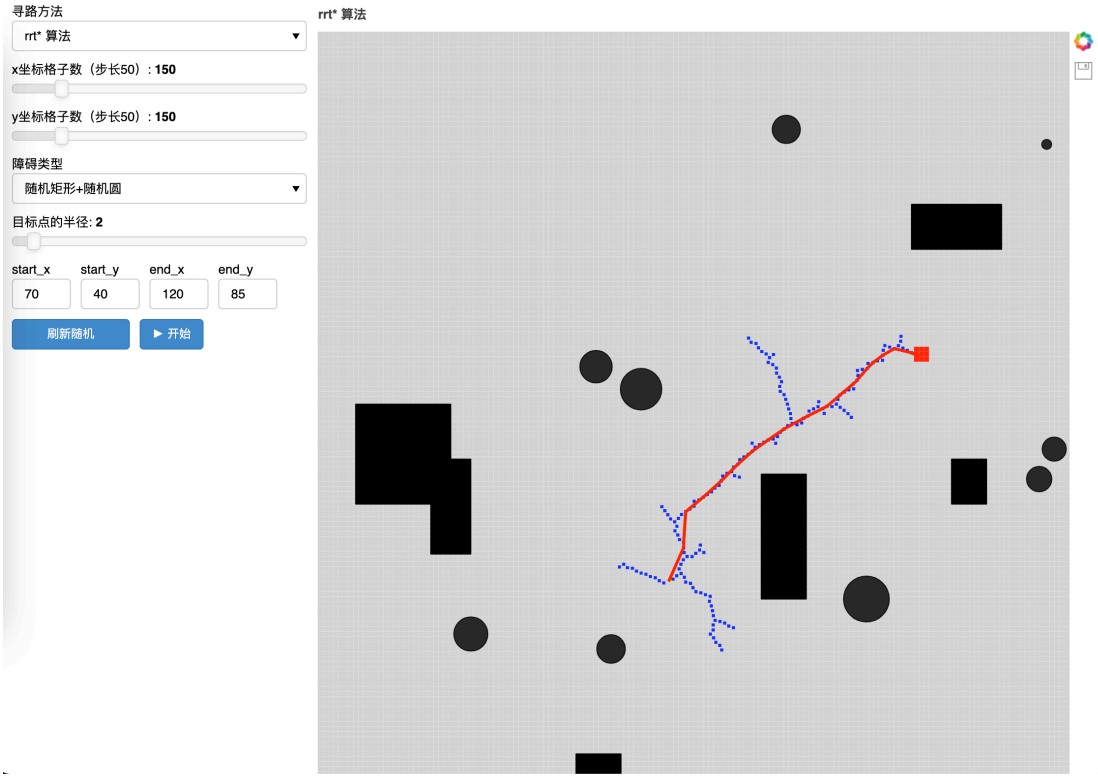


2. RRT*

起点[70, 40] 、终点[120, 85]，半径 2，使用随机生成地图

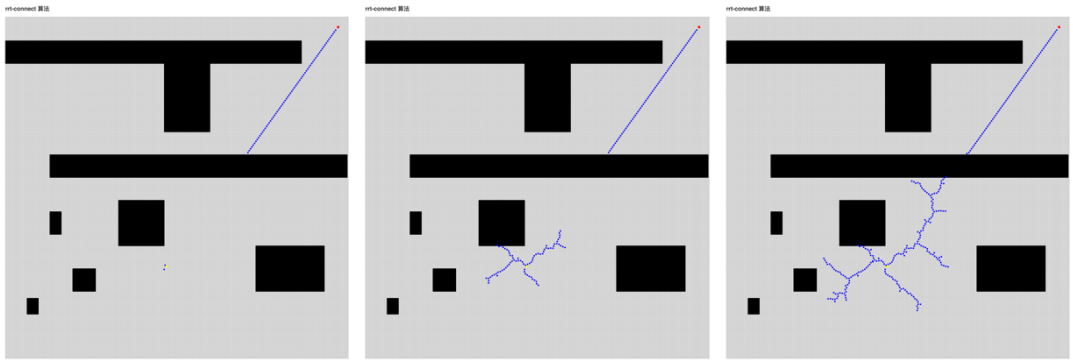


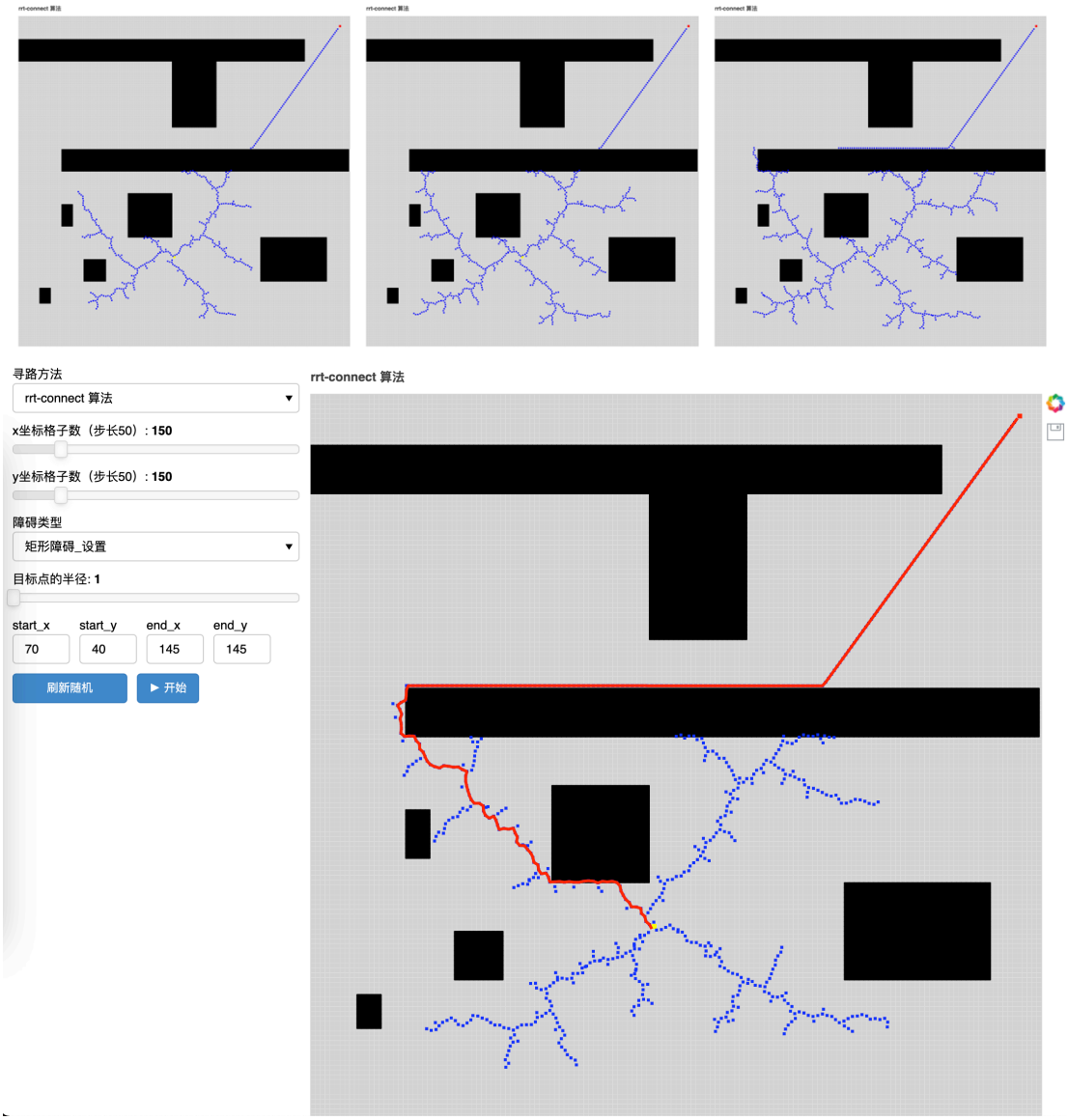




3. RRT- Connect

起点[70, 40] 、终点[145, 145] 使用固定地图





五、遇到的问题与感想

1. 遇到的问题与解决

本次由于第一次使用 bokeh，在弄 GUI 的时候话费了部分时间，并且国内使用 bokeh 的人较少，文档方面较少，查阅英文文档耗时。最终查阅官方文档，使用翻译之后学会基本的 bokeh 使用。

将 RRT 与 bokeh 关联起来的时候，由于 bokeh 的 figure 不能手动强制刷新，导致画图函数耗时长时不知道是否得到结果。可以对绘图数据的结构进行重构，但是时间原因以及复杂程度，就将函数的随机点不做显示处理，得到一个还算不错的效果。

本次写的代码，实现了 3 种方法，部分重复代码没有剪掉写成一个接口，但是注释还算比较全。代码太多，部分区域有一点复杂，但是如果优化算法还算比较方便。需要将来有时间重构部分代码。

2. 感想

本次实验，学习了 bokeh 库的使用，为以后的实验、编程奠定了一定的基础。这次实验给的时间充足，有时间去现学一些没接触过的东西。无论是寻路算法，还是未来可能经常用到的数据可视化，这一次都有接触，除了 RRT 算法，在看 RRT 的算法的同时还学了 A*系列以及 AIT 等等。

在做这个实验的时候，有时熬夜看别人介绍各类寻路算法的视频，有时熬夜读别人写的代码，看原版的论文了解思想等等。总的来说，这次实验锻炼了数据可视化的能力，也锻炼了综合 GUI 和算法类的能力，收获不错。

六、配置文件附录

obstacles.json 配置文件：

```
{
  "矩形障碍_设置": [
    {
      "type": "rectangle",
      "sx": 50,
      "sy": 50,
      "ex": 70,
      "ey": 70
    },
    {
      "type": "rectangle",
      "sx": 20,
      "sy": 55,
      "ex": 25,
      "ey": 65
    },
    {
      "type": "rectangle",
      "sx": 30,
      "sy": 30,
      "ex": 40,
      "ey": 40
    },
    {
      "type": "rectangle",
      "sx": 10,
      "sy": 20,
      "ex": 15,
      "ey": 27
    },
    {
      "type": "rectangle",
```

```
    "sx": 70,
    "sy": 100,
    "ex": 90,
    "ey": 130
  },
  {
    "type": "rectangle",
    "sx": 110,
    "sy": 30,
    "ex": 140,
    "ey": 50
  },
  {
    "type": "rectangle",
    "sx": 0,
    "sy": 130,
    "ex": 130,
    "ey":140
  },
  {
    "type": "rectangle",
    "sx": 20,
    "sy": 80,
    "ex": 150,
    "ey":90
  }
 ]
}
```

没有跑出来的图：

rrt 基础算法