机器人导论

课程作业: assignment 5

组员：16305204 郑佳豪

16326086 王润锋

提交日期：2019-11-7

Deadline：2019-11-7

1. 任务概要：

- 在给定的迷宫场景中，找到唯一的通路，走出迷宫。
- 路径规划算法要求使用人工势场或 RRT 路径规划算法中的一种或多种算法。
- 作业提示：
    - 使用 Vision Sensor 获取迷宫的全局地图。
    - 构建二维全局地图与迷宫的映射关系。
    - 使用 RRT 或人工势场规划出一条通路。
    - 将通路映射到迷宫之中。
    - 机器人巡线，走出迷宫。

姓名	学号	比例	具体任务
郑佳豪	**16305204**	**80%**	**搭建模型、设计并实现 RRT 路径规划算法、实现机器人通路巡线、完成实验报告**
王润锋	**16326086**	**20%**	**设计并实现通路剪枝算法、完成实验报告、录制实验视频**

2. 完成情况：

总体完成情况如下：

- 已学习并能较为熟悉使用 V-REP Python Remote API 接口。
- 已实现基于 RRT 路径规划算法的迷宫通路计算功能。
- 已实现 RRT 通路的剪枝功能。
- 已实现机器人按通路巡线的功能。

1. V-REP Python Remote API

由于在前面实验中，我们已进行了 V-REP Remote API 接口的使用，积累了部分经验，并且考虑到本次实验任务的复杂性，因此在本次实验中，我们使用 V-REP Python Remote API 实现机器人在路径规划算法下走出迷宫的目标。

首先，我们为机器人模型添加 Non-threaded Script，其具体内容如下。在 sysCall_init 函数中，我们在端口 19999 开启了 Remote API 服务。

```
function sysCall_init()
  simRemoteApi.start(19999)
end
```

1 MyBot Non-threaded 控制脚本

为了成功使用 Python 与 V-REP 交互，我们需要导入 remoteApiBindings 至项目文件夹，具体目录为 V-REP 安装目录下的 programming\remoteApiBindings，我们只需导入 vrep.py、vrepConst.py 和 remoteApi.dll 文件。我们编写简单的 Python 代码，测试 Remote API 是否

调用成功，具体代码如下。

```python
import vrep

# Close all the connections.
vrep.simxFinish(-1)
# Connect the V-REP
clientID = vrep.simxStart("127.0.0.1", 19999, True, True, 5000, 5)

if clientID == -1:
    raise Exception("Fail to connect remote API server.")
```

2　测试 Remote API 是否建立成功

点击 V-REP 仿真运行按钮，随后执行上述脚本，若无发生异常，说明 Remote API 建立成功。

我们可通过 vrep.simxGetObjectHandle 获取 V-REP 仿真环境下的物体句柄。

```python
# Get object handles.
_, bot = vrep.simxGetObjectHandle(clientID, 'MyBot', vrep.simx_opmode_oneshot_wait)
_, vision_sensor = vrep.simxGetObjectHandle(clientID, 'Vision_Sensor', vrep.simx_opmode_oneshot_wait)
_, left_motor = vrep.simxGetObjectHandle(clientID, 'Left_Motor', vrep.simx_opmode_oneshot_wait)
_, right_motor = vrep.simxGetObjectHandle(clientID, 'Right_Motor', vrep.simx_opmode_oneshot_wait)
```

3　获取 V-REP 物体句柄

2. 获取全局地图

按照作业提示，我们需要通过 Vision Sensor 获取全局地图，为此我们实现了 get_image 工具函数，其具体代码如下。

```python
def get_image(sensor):
    """Retrieve a binary image from Vision Sensor.

    :return: a binary image represented by numpy.ndarray from Vision Sensor
    """
    err, resolution, raw = vrep.simxGetVisionSensorImage(clientID, sensor, 0, vrep.simx_opmode_buffer)
    if err == vrep.simx_return_ok:
        img = np.array(raw, dtype=np.uint8)
        img.resize([resolution[1], resolution[0], 3])
        # Process the raw image.
        _, th1 = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)
        g = cv2.cvtColor(th1, cv2.COLOR_BGR2GRAY)
        _, th2 = cv2.threshold(g, 250, 255, cv2.THRESH_BINARY)
        # Find the edges using Canny.
        edge = cv2.Canny(th2, 50, 150)  # type: np.ndarray
        return edge
    else:
        return None
```

4　获取 Vision Sensor 二值化图像

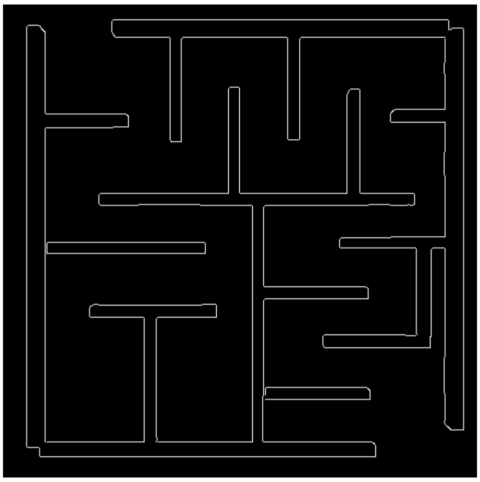
在 get_image 函数中，我们对获取的图像进行了**二值化**、**边缘提取**的处理，目的是减少障碍点的数量，提高后续算法运行性能。

在调用 get_image 函数获取全局地图之前，我们需要对 Vision Sensor 进行初始化，否则，我们可能无法获取图像，初始化函数 init 的具体代码如下。

```python
def init():
    """Initialize the simulation.
    """
    vrep.simxGetVisionSensorImage(clientID, vision_sensor, 0, vrep.simx_opmode_streaming)
    time.sleep(1)
```

5　init 初始化函数

以下是我们初始化后，调用 get_image 获取的迷宫全局地图，其中白色像素表示障碍物的边缘，黑色像素表示无障碍物的通路。



6 迷宫全局地图

3. RRT 路径规划

快速扩展随机树（RRT）算法，通过对状态空间中的采样点（随机采样）进行碰撞检测，将搜索导向空白区域，避免了对空间的建模，能够有效地解决高维空间和复杂约束的路径规划问题，适合解决多自由度机器人在复杂环境下和动态环境中的路径规划，该算法是**概率完备且不最优**的。

算法具体流程大致如下：在状态空间中随机选择一个采样点，然后从随机树中选择与其最近的点进行扩展新的树节点，并进行障碍物检测，若与障碍物发生碰撞，则放弃生长，否则将新节点（以及扩展路径）添加至随机树中。

RRT Algorithm (x_{start}, x_{goal}, step, n)

G.initialize(x_{start})
for $i = 1$ **to** n **do**
    x_{rand} = Sample()
    x_{near}= near(x_{rand}, G)
    x_{new}= steer(x_{rand}, x_{near}, step_size)
    G.add_node(x_{new})
    G.add_edge(x_{new}, x_{near})
    if x_{new}= x_{goal}
        success()

7 RRT 路径规划算法的伪代码实现

在进行 RRT 路径规划前，我们需要在二值化图像中，提取障碍物的坐标信息。在 get_image 获取的二值化图像中，障碍物点为白色像素，其像素值为 255。利用此信息，我们可提取图像中所有的障碍物边界点，具体代码参照以下 get_obstacles 代码（障碍物边界点半径为 15）。

```python
def get_obstacles(img: np.ndarray):
    """Return the list of obstacles.

    :param img: the maze image
    :return: the list of obstacles
    """
    obstacles = []
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i][j] == 255:
                obstacles.append((i, j, 15))
    return obstacles
```

8 获取二值化图像中的障碍物坐标信息

我们在 get_random_node 函数中，实现状态空间下的随机采样点生成，其中为了加快通路的搜索速度，我们根据随机概率来决定采样点是随机点还是目标点：对于范围为 0 到 100 的随机值 rnd，若 rnd 大于 goal_sample_rate，则采样点为随机点，否则为目标点。

```python
def get_random_node(self):
    """Generate a random RRT.Node

    :return: a random RRT.Node
    """
    if random.randint(0, 100) > self.goal_sample_rate:  # Do random sampling.
        rnd = self.Node(random.uniform(self.min_rand, self.max_rand),
                        random.uniform(self.min_rand, self.max_rand))
    else:  # Do goal point sampling.
        rnd = self.Node(self.end.x, self.end.y)
    return rnd
```

9 随机采样点的生成

在生成随机采样点后，我们需要在随机树中查询离其最近的节点，该部分功能由 get_nearest_node_index 函数实现。

```python
@staticmethod
def get_nearest_node_index(node_list: List[Node], rnd_node: Node):
    """Find the nearest node to rnd_node in node_list.

    :param node_list: the candidate nodes
    :param rnd_node: the target node
    :return: the nearest node to rnd_node in node_list
    """
    distances = [(node.x - rnd_node.x) ** 2 + (node.y - rnd_node.y) ** 2 for node in node_list]
    nearest = distances.index(min(distances))
    return nearest
```

10 查询随机树中离采样点最近的节点

我们在需要在“最近点”和采样点之间进行随机树扩展，其具体代码如下。

```python
def steer(self, from_node: Node, to_node: Node, expand_length=float("inf")):
    """Expand the tree from from_node to to_node.

    :param from_node: from which node to expand
    :param to_node: to which node to expand
    :param expand_length: expand length
    :return: the new node
    """
    new_node = self.Node(from_node.x, from_node.y)
    d, theta = self.calc_distance_and_angle(new_node, to_node)
    new_node.path_x = [new_node.x]
    new_node.path_y = [new_node.y]
    if expand_length > d:
        expand_length = d
    n_expand = math.floor(expand_length / self.path_resolution)
    for _ in range(n_expand):
        new_node.x += self.path_resolution * math.cos(theta)
        new_node.y += self.path_resolution * math.sin(theta)
        new_node.path_x.append(new_node.x)
        new_node.path_y.append(new_node.y)
    d, _ = self.calc_distance_and_angle(new_node, to_node)
    if d <= self.path_resolution:
        new_node.path_x.append(to_node.x)
        new_node.path_y.append(to_node.y)
    new_node.parent = from_node
    return new_node
```

11 扩展随机树

在得到扩展后的新节点后，我们需要对其进行障碍物检测，若通过障碍物检测，则添加此节点至随机树中，否则抛弃此节点。

```python
@staticmethod
def check_collision(node: Node, obstacles: List[Tuple[int, int, int]]):
    """Check the node whether collides with obstacles.

    :param node: a RRT.Node
    :param obstacles: obstacles list
    :return: whether the node collides with obstacles
    """
    for (ox, oy, size) in obstacles:
        dx_list = [ox - x for x in node.path_x]
        dy_list = [oy - y for y in node.path_y]
        d_list = [dx * dx + dy * dy for (dx, dy) in zip(dx_list, dy_list)]
        if min(d_list) <= size ** 2:
            return False
    return True
```

12　障碍物检测

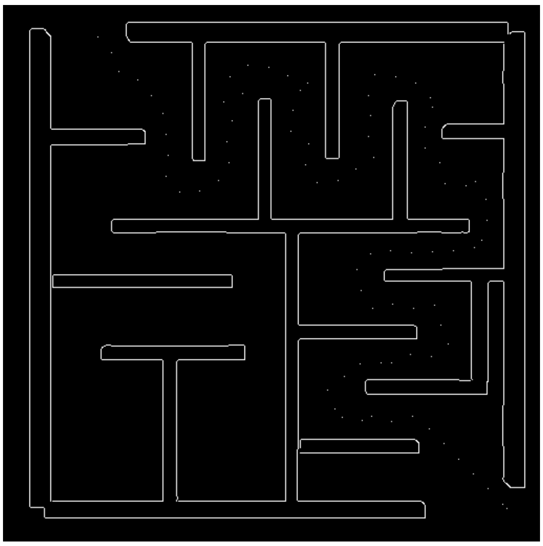
综合上述各个子模块，我们即可实现完整的 RRT 路径规划算法，具体代码如下。

```python
def path_planning(self, img: np.ndarray):
    """Find the solution of the maze.

    :param img: the image of maze
    :return: the solution path
    """
    self.node_list = [self.start]
    for i in range(self.max_iter):
        # Generate a sample node.
        rnd_node = self.get_random_node()
        # Find the nearest node to the rnd_node.
        nearest_ind = self.get_nearest_node_index(self.node_list, rnd_node)
        nearest_node = self.node_list[nearest_ind]
        # Expand the tree.
        new_node = self.steer(nearest_node, rnd_node, self.expand_dis)
        # When there are no collisions, add new_node to the tree.
        if self.check_collision(new_node, self.obstacle_list):
            self.node_list.append(new_node)
            # Show the expanding process.
            tmp = self.node_list[-1]
            x, y = int(tmp.x), int(tmp.y)
            img[x][y] = 255
            cv2.imshow("Processing", img)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        # Check whether it reaches the goal.
        if self.calc_dist_to_goal(self.node_list[-1].x, self.node_list[-1].y) <= self.expand_dis:
            final_node = self.steer(self.node_list[-1], self.end, self.expand_dis)
            if self.check_collision(final_node, self.obstacle_list):
                return self.generate_final_course()
    return None
```

13　RRT 路径规划算法实现

对迷宫的二值化地图执行 RRT 路径规划后，可得以下的路径通路（白色像素点轨迹）。



14　RRT 路径规划算法求取的通路

4. RRT 通路剪枝

通过 RRT 算法，我们能获得一条不错的路径，但实际上我们通过肉眼会发现，由于路径是随机生成的，会有一些点是不需要经过的，我们可以同时跨越几个点，这样我们的小车的拐点就不会太多，小车行驶路线也会更加流畅。

通路剪枝算法大致流程如下：从当前点出发，枚举下一个可行点，两点确定一条直线，判断直线是否与障碍点距离过近，如果过近则不合法，否则即合法。合法后用最远的可行点更新当前点。该剪枝算法属于贪心算法，具体代码如下。

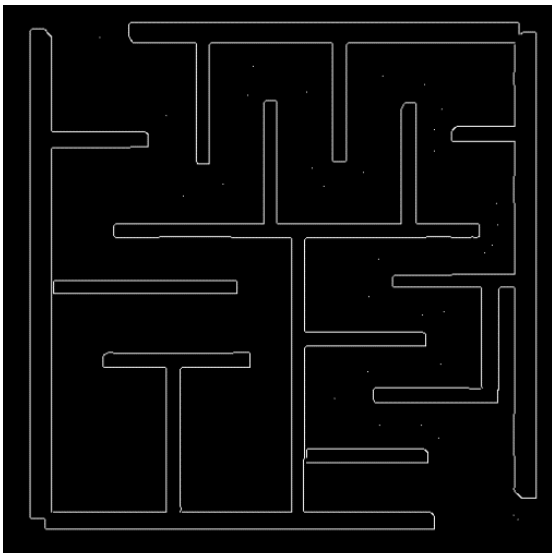
```python
def path_pruning(path: np.ndarray, obstacles: List[Tuple[int, int, int]]):
    """Prune the solution path.

    :param path: the solution path
    :param obstacles: the list of obstacles
    :return: the pruned path
    """
    pruned_path = [path[0]]
    n, m = len(path), len(obstacles)
    cur = 0
    th = obstacles[0][2]
    # 将连线进行等分操作。
    sz = 7
    while True:
        if cur == n - 1:
            break
        to = cur + 1
        for j in range(cur + 1, min(cur + 6, n)):
            x1, y1 = path[cur][0], path[cur][1]
            x2, y2 = path[j][0], path[j][1]
            ok = True
            for h in range(sz):
                mx, my = (h * x1 + (sz - h) * x2) / sz, (h * y1 + (sz - h) * y2) / sz
                min_dist = 99999999
                for k in range(m):
                    x, y = obstacles[k][0], obstacles[k][1]
                    min_dist = min(min_dist, distance(x, y, mx, my))
                if min_dist <= th:
                    ok = False
                    break
            if ok:
                to = max(to, j)
        pruned_path.append(path[to])
        cur = to
    ret = np.array(pruned_path)
    np.savetxt("pruned_solution.txt", ret)
    return ret
```

15　RRT 路径剪枝算法代码

在实验过程中，剪枝算法并未考虑车体大小信息，降低了实验成功率。所以，在实现中，我们保留了原始通路和剪枝后的通路，根据实际情况选取合适的通路进行仿真。

在算法实现中，我们使用了一个 Trick 点：我们不需要计算线段与障碍物点的距离，我们只用取线段上的 n 等分点来计算即可。当 n 取一定大小（约 10）基本上就会有相同的剪枝效果，剪枝效果如下图所示。

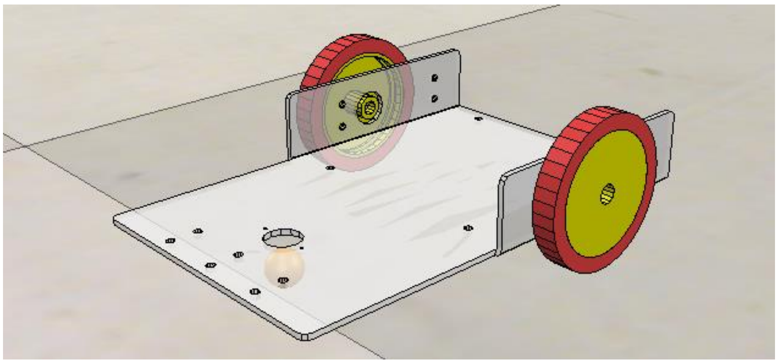


16　RRT 剪枝算法结果

从上图可以看到，剪枝效果还是很不错的。不过实际实验后发现，由于车身大小限制，车旋转时占用空间有点大，导致车体与墙体碰撞。因此，我们要看情况选取原有路径或者剪枝后的路径。

5. 机器人搭建

考虑到迷宫路径的复杂性，我们使用**差速转向**机器人完成实验，静态演示如下图所示。



17　差速转向机器人静态示意图

由于机器人的运动学模型是差速转向，因此我们需要计算其反运动学公式：根据速度和角度推导左右两电机的转速。运动学控制代码如下图所示。

```python
def move(v, o):
    """Move the robot.

    :param v: desired velocity
    :param o: desired angular velocity
    """
    wheel_radius = 0.027
    distance = 0.119
    v_l = v - o * distance
    v_r = v + o * distance
    o_l = v_l / wheel_radius
    o_r = v_r / wheel_radius
    vrep.simxSetJointTargetVelocity(clientID, left_motor, o_l, vrep.simx_opmode_oneshot)
    vrep.simxSetJointTargetVelocity(clientID, right_motor, o_r, vrep.simx_opmode_oneshot)
```

18　差速转向机器人反向运动学模型

6. RRT 通路巡线

在获取到通路路径后，我们需要让机器人按照路径点进行巡线。我们的巡线策略很简单：**获取当前机器人的角度，计算路径点相对机器人的角度，进行原地转向使得路径点位于机器人正前方，随后直线运动至路径点，重复此过程直至抵达目的地**，具体代码如下图所示。

```python
def path_following(path: List[Tuple[int, int]]):
    """Follow the solution path.

    :param path: the solution path
    """
    i = 1
    stage = 0
    goal_angle = None
    tolerance = 2
    while vrep.simxGetConnectionId(clientID) != -1:
        # When it reaches the Goal Position, we stop the scripts.
        if i == len(path):
            break
        if stage == 0:
            # Steer for specific angle.
            # Get current position.
            _, cur = vrep.simxGetObjectPosition(clientID, bot, -1, vrep.simx_opmode_oneshot_wait)
            cur_angle = get_beta_angle()
            if goal_angle is None:
                phi = math.atan2(path[i][0] - cur[0], path[i][1] - cur[1])
                goal_angle = -math.degrees(phi)
            delta = cur_angle - goal_angle
            if delta > tolerance:
                move(0, -0.1)
                continue
            if delta < -tolerance:
                move(0, 0.1)
                continue
            goal_angle = None
            move(0, 0)
            stage = 1
            continue
        if stage == 1:
            # Go straight for specific distance.
            # Get current position.
            _, cur = vrep.simxGetObjectPosition(clientID, bot, -1, vrep.simx_opmode_oneshot_wait)
            dis = distance(cur[0], cur[1], path[i][0], path[i][1])
            if dis < 0.1:
                i += 1
                stage = 0
                move(0, 0)
                continue
            move(0.5, 0)
            continue
```

19 RRT 通路巡线代码

在代码中，我们调用 simxGetObjectPosition 获取机器人当前位置，从而计算机器人和路径点间的相对角度，随后我们调用 get_beta_angle 函数获取机器人对世界坐标系的角度，该函数通过调用 simxGetObjectOrientation 方法获取机器人当前的欧拉角参数，获取 Beta 角度（即为相对于世界坐标的角度），具体代码如下。

```python
def get_beta_angle():
    """Return the degrees of Beta Euler Angle.

    :return: the degrees of Beta Euler Angle
    """
    _, euler_angles = vrep.simxGetObjectOrientation(clientID, bot, -1, vrep.simx_opmode_oneshot_wait)
    ret = math.degrees(euler_angles[1])
    if euler_angles[0] <= 0 < ret:
        return 180 - ret
    if euler_angles[2] <= 0 and ret < 0:
        return -180 - ret
    return ret
```

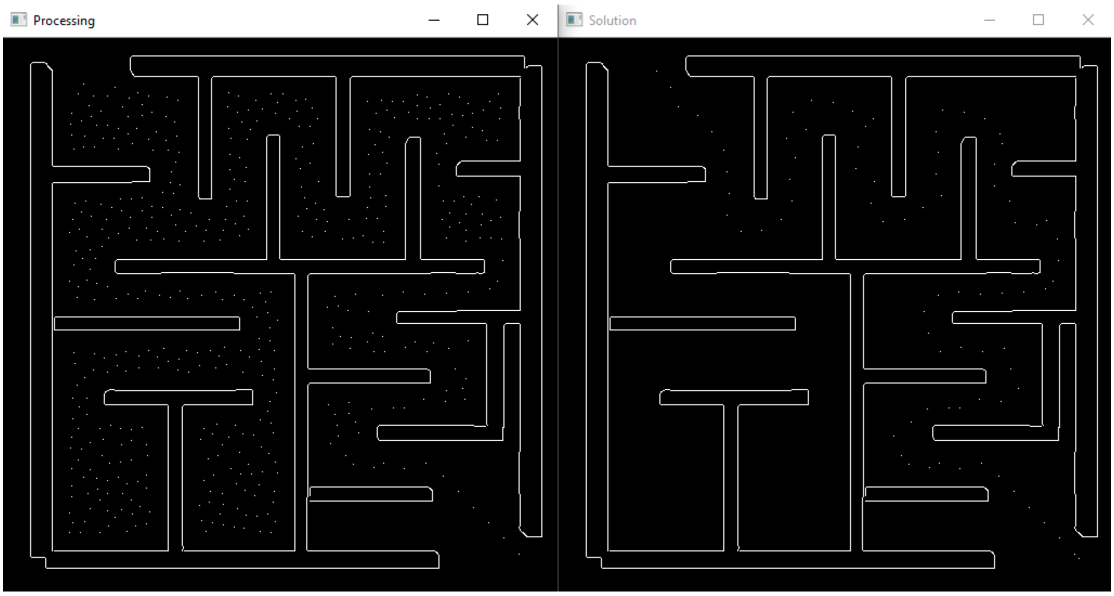
20 获取机器人航向角

3. 效果展示：

演示视频：https://www.bilibili.com/video/av74889189/

- **RRT 路径规划**

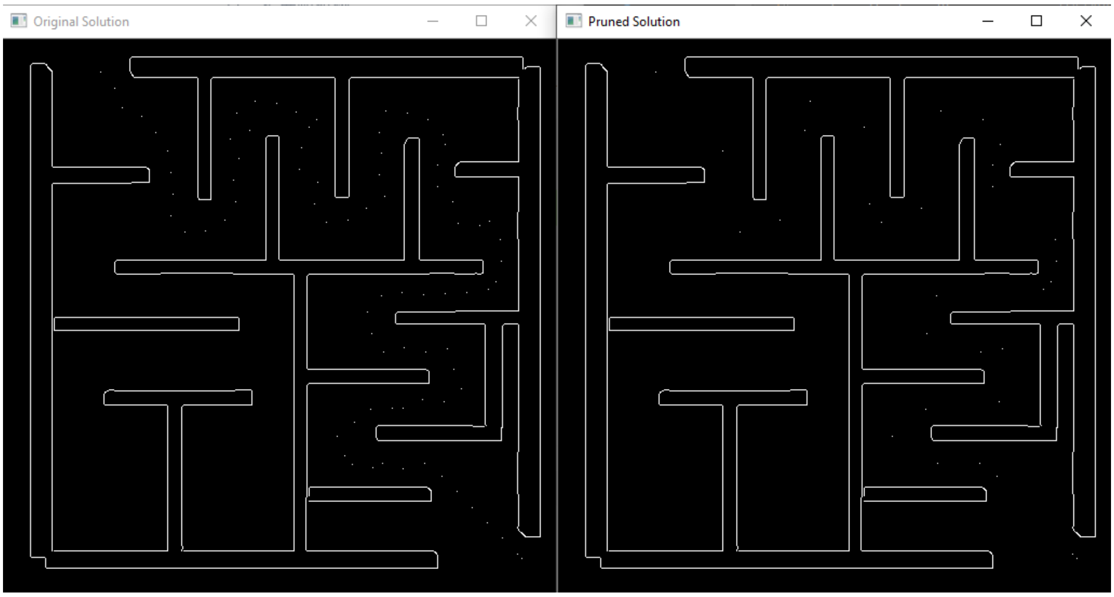
  使用 python 运行 path_planning.py 即可对地图进行 RRT 路径规划，其具体结果如下。



21　RRT 路径规划算法演示

- **RRT 路径剪枝**

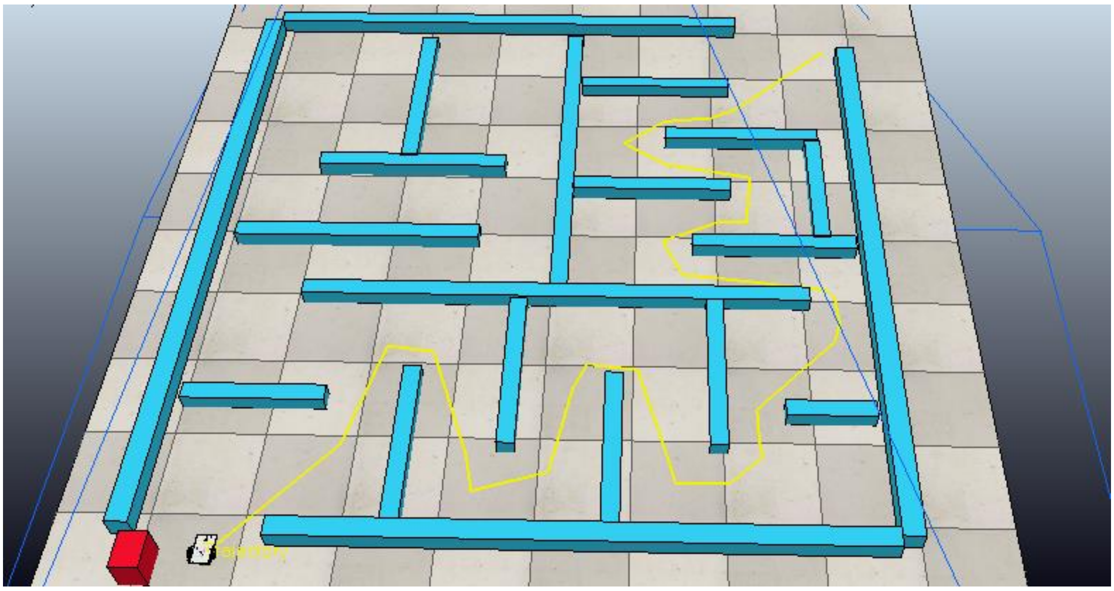
  使用 python 运行 path_pruning.py 即可对 RRT 规划路径进行剪枝操作，其具体结果如下。



22　RRT 路径剪枝算法演示

- **RRT 路径巡线**

执行 path_following.py 文件，即可进行 RRT 路径巡线，巡线效果如下图所示。



23 剪枝路线巡线演示

4. 存在问题：

本次实验，内容是使用路径规划算法求取迷宫通路、并按此通路巡线，具备很强的挑战性。在实验过程中，我们遇到了几个难题，最终大多数都被成功解决。

我们将本次实验划分了三个部分：**路径规划求取通路、通路路径剪枝、通路巡线**。在路径规划求取通路部分中，我们一开始使用的是**人工势场规划算法**（**APF**），但在算法测试环节中，我们发现在 APF 算法规划下，路径容易出现**"局部最小"**的情况：由于各个障碍物对机器人的斥力相互叠加，出现合力为零的情况，且由于目标物的引力不足，导致机器人停止了运动，无法到达目的地。解决此问题的方法是为现有的 APF 算法添加随机行走策略，但出于进度考虑，我们更换了路径规划算法，使用鲁棒性更为优异的 RRT 快速扩张随机树算法。

我们在测试 RRT 规划算法时，出现了**求取通路时间过长**的问题，我们尝试更改扩展步长以及障碍物半径大小，发现并不能很好解决此问题。最终，经过我们的努力，我们将算法运行时间控制在 5 分钟内。或许，我们可以使用更为高效的数据结构（**K-D Tree**）来求取离随机抽样点最近的随机树节点，进而提升算法速度。由于时间的原因，我们并未在程序代码中使用 K-D Tree 数据结构，这是本次实验的一个遗憾。

在求取出通路路径后，我们发现该**路径存在优化空间**，如可利用"两点之间线段最短"来移除不必要的路径点。但在剪枝算法实现中，我们遇到了运行时间过长的问题：由于障碍物边界点数量过多（2 万多个），导致两点连线的障碍物检测运行时间过长，从而增长了算法总体运行时间。针对该问题，我们选择了使用等分点判断碰撞的方法，一定程度减少了算法运行时间。

在实现机器人按通路巡路部分时，我们遇到了**欧拉角**相关的问题。在实验中，欧拉角的 Beta 即为我们需要的转向角度，但我们一开始发现 Beta 存在二义性（其范围为-90 度到+90 度），并不能反映当前机器人相对于世界的转向角度。在耽误了好长时间后，我们发现可通过考虑 Alpha 以及 Gamma 的正负情况，转换 Beta 角度至-180 度至+180 度。

总体来说，本次实验是成功的，我们成功完成了各项实验任务，收获颇丰。

5. 附录：

MyBot Non-threaded 控制代码

```
function sysCall_init()
  simRemoteApi.start(19999)
end
```

RRT 路径规划算法 path_planning.py

```
import time

from RRT import *
from utils import get_obstacles, draw_path


def path_planning(img: np.ndarray):
    """Find the solution of the maze.

    :param img: the image of the maze
    :return: the solution
    """
    # Define Start Position and Goal Position.
    start = (30, 90)
    goal = (480, 480)
    obstacles = get_obstacles(img)
    # Initialize RRT Motion Planner.
    rrt = RRT(start, goal, obstacles, (50, 480), expand_dis=20, goal_sample_rate=35, path_resolution=10, max_iter=50000)
    # Plan the path.
    path = rrt.path_planning(img.copy())
    if path is None:
        raise Exception("Cannot find the path.")
    # Save the solution.
    np.savetxt("solution.txt", np.array(path), fmt="%s")
    return path


def main():
    """Path Planning.
    """
    # Load the maze image.
    img = cv2.imread("maze.png", cv2.THRESH_BINARY)  # type:np.ndarray
    cv2.imshow("Maze", img)

    # Find the solution.
    start = time.time()
```

```python
    solution = path_planning(img)
    cost = time.time() - start
    print(f"It costs {cost} seconds to find the path.")

    # Show the solution.
    cv2.imshow("Solution", draw_path(img, solution))
    cv2.imwrite("solution.png", draw_path(img, solution))

    # When we press "q", the program will exit.
    while True:
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break


if __name__ == '__main__':
    main()
```

RRT 路径剪枝算法 path_pruning.py

```python
from typing import List

import cv2

from utils import *


def path_pruning(path: np.ndarray, obstacles: List[Tuple[int, int, int]
]):
    """Prune the solution path.

    :param path: the solution path
    :param obstacles: the list of obstacles
    :return: the pruned path
    """
    pruned_path = [path[0]]
    n, m = len(path), len(obstacles)
    cur = 0
    th = obstacles[0][2]
    # 将连线进行 n 等分操作。
    sz = 7
    while True:
        if cur == n - 1:
            break
        to = cur + 1
        for j in range(cur + 1, min(cur + 6, n)):
```

```
            x1, y1 = path[cur][0], path[cur][1]
            x2, y2 = path[j][0], path[j][1]
            ok = True
            for h in range(sz):
                mx, my = (h * x1 + (sz - h) * x2) / sz, (h * y1 + (sz -
 h) * y2) / sz
                min_dist = 99999999
                for k in range(m):
                    x, y = obstacles[k][0], obstacles[k][1]
                    min_dist = min(min_dist, distance(x, y, mx, my))
                if min_dist <= th:
                    ok = False
                    break
            if ok:
                to = max(to, j)
        pruned_path.append(path[to])
        cur = to
    ret = np.array(pruned_path)
    np.savetxt("pruned_solution.txt", ret)
    return ret


def main():
    """Path Pruning.
    """
    # Load the maze image.
    img = cv2.imread("maze.png", cv2.THRESH_BINARY)  # type:np.ndarray

    # Load and draw the solution path.
    path = np.loadtxt("solution.txt")
    cv2.imshow("Original Solution", draw_path(img.copy(), path))

    # Prune the solution path.
    pruned_path = path_pruning(path, get_obstacles(img))
    cv2.imshow("Pruned Solution", draw_path(img.copy(), pruned_path))
    cv2.imwrite("pruned_solution.png", draw_path(img.copy(), pruned_pat
h))

    for i in range(len(pruned_path)):
        print(world_coordinate(pruned_path[i]))

    # When we press "q", the program will exit.
    while True:
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

```python
if __name__ == '__main__':
    main()
```

RRT 路径巡线算法 path_following.py

```python
from typing import List

from remote import *
from utils import *


def path_following(path: List[Tuple[int, int]]):
    """Follow the solution path.

    :param path: the solution path
    """
    i = 1
    stage = 0
    goal_angle = None
    tolerance = 2
    while vrep.simxGetConnectionId(clientID) != -1:
        # When it reaches the Goal Position, we stop the scripts.
        if i == len(path):
            break
        if stage == 0:
            # Steer for specific angle.
            # Get current position.
            _, cur = vrep.simxGetObjectPosition(clientID, bot, -
1, vrep.simx_opmode_oneshot_wait)
            cur_angle = get_beta_angle()
            if goal_angle is None:
                phi = math.atan2(path[i][0] - cur[0], path[i][1] - cur[
1])
                goal_angle = -math.degrees(phi)
            delta = cur_angle - goal_angle
            if delta > tolerance:
                move(0, -0.1)
                continue
            if delta < -tolerance:
                move(0, 0.1)
                continue
            goal_angle = None
            move(0, 0)
```

```python
            stage = 1
            continue
        if stage == 1:
            # Go straight for specific distance.
            # Get current position.
            _, cur = vrep.simxGetObjectPosition(clientID, bot, -
1, vrep.simx_opmode_oneshot_wait)
            dis = distance(cur[0], cur[1], path[i][0], path[i][1])
            if dis < 0.1:
                i += 1
                stage = 0
                move(0, 0)
                continue
            move(0.5, 0)
            continue


def main():
    """Path Following.
    """
    # Initialize the simulation.
    init()

    # Load the solution path.
    solution = np.loadtxt("pruned_solution.txt")
    # Convert the coordinates.
    path = []
    for i in range(len(solution)):
        path.append(world_coordinate(solution[i]))

    # Follow the solution path.
    path_following(path)


if __name__ == '__main__':
    main()
```

RRT 快速扩展随机树算法 RRT.py

```python
import random
from typing import List, Tuple

import cv2
import math
import numpy as np
```

```python
class RRT:
    """RRT Motion Planning.
    """

    class Node:
        """RRT Node.
        """

        def __init__(self, x: float, y: float):
            self.x = x
            self.y = y
            self.path_x = []
            self.path_y = []
            self.parent = None  # type:None | RRT.Node

    def __init__(self, start: Tuple[int, int], goal: Tuple[int, int], o
bstacle_list: List[Tuple[int, int, int]],
                 rand_area: Tuple[int, int],
                 expand_dis=3.0, path_resolution=0.5, goal_sample_rate=
5, max_iter=500):
        """Initialize the RRT Motion Planner.

        :param start: Start Position
        :param goal:  Goal Position
        :param obstacle_list: the list of obstacles
        :param rand_area: random sampling area
        :param expand_dis: distance of one expanding
        :param path_resolution: path resolution
        :param goal_sample_rate: gaol sample rate
        :param max_iter: max amount of iterations
        """
        self.start = self.Node(start[0], start[1])
        self.end = self.Node(goal[0], goal[1])
        self.min_rand = rand_area[0]
        self.max_rand = rand_area[1]
        self.expand_dis = expand_dis
        self.path_resolution = path_resolution
        self.goal_sample_rate = goal_sample_rate
        self.max_iter = max_iter
        self.obstacle_list = obstacle_list
        self.node_list = []

    def path_planning(self, img: np.ndarray):
```

```
        """Find the solution of the maze.

        :param img: the image of maze
        :return: the solution path
        """
        self.node_list = [self.start]
        for i in range(self.max_iter):
            # Generate a sample node.
            rnd_node = self.get_random_node()
            # Find the nearest node to the rnd_node.
            nearest_ind = self.get_nearest_node_index(self.node_list, rnd_node)
            nearest_node = self.node_list[nearest_ind]
            # Expand the tree.
            new_node = self.steer(nearest_node, rnd_node, self.expand_dis)
            # When there are no collisions, add new_node to the tree.
            if self.check_collision(new_node, self.obstacle_list):
                self.node_list.append(new_node)
                # Show the expanding process.
                tmp = self.node_list[-1]
                x, y = int(tmp.x), int(tmp.y)
                img[x][y] = 255
                cv2.imshow("Processing", img)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            # Check whether it reaches the goal.
            if self.calc_dist_to_goal(self.node_list[-1].x, self.node_list[-1].y) <= self.expand_dis:
                final_node = self.steer(self.node_list[-1], self.end, self.expand_dis)
                if self.check_collision(final_node, self.obstacle_list):
                    return self.generate_final_course()
        return None

    def steer(self, from_node: Node, to_node: Node, expand_length=float("inf")):
        """Expand the tree from from_node to to_node.

        :param from_node: from which node to expand
        :param to_node: to which node to expand
        :param expand_length: expand length
        :return: the new node
```

```
        """
        new_node = self.Node(from_node.x, from_node.y)
        d, theta = self.calc_distance_and_angle(new_node, to_node)
        new_node.path_x = [new_node.x]
        new_node.path_y = [new_node.y]
        if expand_length > d:
            expand_length = d
        n_expand = math.floor(expand_length / self.path_resolution)
        for _ in range(n_expand):
            new_node.x += self.path_resolution * math.cos(theta)
            new_node.y += self.path_resolution * math.sin(theta)
            new_node.path_x.append(new_node.x)
            new_node.path_y.append(new_node.y)
        d, _ = self.calc_distance_and_angle(new_node, to_node)
        if d <= self.path_resolution:
            new_node.path_x.append(to_node.x)
            new_node.path_y.append(to_node.y)
        new_node.parent = from_node
        return new_node

    def generate_final_course(self):
        """Generate the final path to Goal Position.

        :return: the final path to Goal Position
        """
        path = [(self.end.x, self.end.y)]  # type:List[Tuple[float,floa
t]]
        node = self.node_list[len(self.node_list) - 1]
        while node.parent is not None:
            path.append((node.x, node.y))
            node = node.parent
        path.append((node.x, node.y))
        return path[::-1]

    def calc_dist_to_goal(self, x: float, y: float):
        """Calculate the distance between node (x, y) and Goal Position
.

        :param x: the x-coordinate of the node
        :param y: the y-coordinate of the node
        :return: the distance between node (x, y) and Goal Position
        """
        dx = x - self.end.x
        dy = y - self.end.y
```

```
        return math.sqrt(dx ** 2 + dy ** 2)

    def get_random_node(self):
        """Generate a random RRT.Node

        :return: a random RRT.Node
        """
        if random.randint(0, 100) > self.goal_sample_rate:  # Do random
 sampling.
            rnd = self.Node(random.uniform(self.min_rand, self.max_rand
),
                            random.uniform(self.min_rand, self.max_rand
))
        else:  # Do goal point sampling.
            rnd = self.Node(self.end.x, self.end.y)
        return rnd

    @staticmethod
    def get_nearest_node_index(node_list: List[Node], rnd_node: Node):
        """Find the nearest node to rnd_node in node_list.

        :param node_list: the candidate nodes
        :param rnd_node: the target node
        :return: the nearest node to rnd_node in node_list
        """
        distances = [(node.x - rnd_node.x) ** 2 + (node.y - rnd_node.y)
 ** 2 for node in node_list]
        nearest = distances.index(min(distances))
        return nearest

    @staticmethod
    def check_collision(node: Node, obstacles: List[Tuple[int, int, int
]]):
        """Check the node whether collides with obstacles.

        :param node: a RRT.Node
        :param obstacles: obstacles list
        :return: whether the node collides with obstacles
        """
        for (ox, oy, size) in obstacles:
            dx_list = [ox - x for x in node.path_x]
            dy_list = [oy - y for y in node.path_y]
            d_list = [dx * dx + dy * dy for (dx, dy) in zip(dx_list, dy
_list)]
```

```python
            if min(d_list) <= size ** 2:
                return False
        return True

    @staticmethod
    def calc_distance_and_angle(from_node: Node, to_node: Node):
        """Calculate the distance and angle between two nodes.

        :param from_node: a RRT.Node
        :param to_node: a RRT.Node
        :return: the distance and angle
        """
        dx = to_node.x - from_node.x
        dy = to_node.y - from_node.y
        d = math.sqrt(dx ** 2 + dy ** 2)
        theta = math.atan2(dy, dx)
        return d, theta
```

Remote API 工具函数 remote.py

```python
import time

import cv2
import math
import numpy as np

import vrep

# Close all the connections.
vrep.simxFinish(-1)
# Connect the V-REP.
clientID = vrep.simxStart("127.0.0.1", 19999, True, True, 5000, 5)

# Get object handles.
_, bot = vrep.simxGetObjectHandle(clientID, 'MyBot', vrep.simx_opmode_o
neshot_wait)
_, vision_sensor = vrep.simxGetObjectHandle(clientID, 'Vision_Sensor',
vrep.simx_opmode_oneshot_wait)
_, left_motor = vrep.simxGetObjectHandle(clientID, 'Left_Motor', vrep.s
imx_opmode_oneshot_wait)
_, right_motor = vrep.simxGetObjectHandle(clientID, 'Right_Motor', vrep
.simx_opmode_oneshot_wait)

if clientID == -1:
```

```
    raise Exception("Fail to connect remote API server.")


def init():
    """Initialize the simulation.
    """
    vrep.simxGetVisionSensorImage(clientID, vision_sensor, 0, vrep.simx
_opmode_streaming)
    time.sleep(1)


def get_image(sensor):
    """Retrieve a binary image from Vision Sensor.

    :return: a binary image represented by numpy.ndarray from Vision Se
nsor
    """
    err, resolution, raw = vrep.simxGetVisionSensorImage(clientID, sens
or, 0, vrep.simx_opmode_buffer)
    if err == vrep.simx_return_ok:
        img = np.array(raw, dtype=np.uint8)
        img.resize([resolution[1], resolution[0], 3])
        # Process the raw image.
        _, th1 = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)
        g = cv2.cvtColor(th1, cv2.COLOR_BGR2GRAY)
        _, th2 = cv2.threshold(g, 250, 255, cv2.THRESH_BINARY)
        # Find the edges using Canny.
        edge = cv2.Canny(th2, 50, 150)  # type: np.ndarray
        return edge
    else:
        return None


def move(v, o):
    """Move the robot.

    :param v: desired velocity
    :param o: desired angular velocity
    """
    wheel_radius = 0.027
    distance = 0.119
    v_l = v - o * distance
    v_r = v + o * distance
    o_l = v_l / wheel_radius
    o_r = v_r / wheel_radius
```

```
    vrep.simxSetJointTargetVelocity(clientID, left_motor, o_l, vrep.sim
x_opmode_oneshot)
    vrep.simxSetJointTargetVelocity(clientID, right_motor, o_r, vrep.si
mx_opmode_oneshot)


def get_beta_angle():
    """Return the degrees of Beta Euler Angle.

    :return: the degrees of Beta Euler Angle
    """
    _, euler_angles = vrep.simxGetObjectOrientation(clientID, bot, -
1, vrep.simx_opmode_oneshot_wait)
    ret = math.degrees(euler_angles[1])
    if euler_angles[0] <= 0 < ret:
        return 180 - ret
    if euler_angles[2] <= 0 and ret < 0:
        return -180 - ret
    return ret
```

Utils 工具函数

```
from typing import Tuple

import math
import numpy as np


def get_obstacles(img: np.ndarray):
    """Return the list of obstacles.

    :param img: the maze image
    :return: the list of obstacles
    """
    obstacles = []
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i][j] == 255:
                obstacles.append((i, j, 15))
    return obstacles


def draw_path(img: np.ndarray, path: np.ndarray):
    """Draw the path on the maze image.
```

```python
    :param img: the maze image
    :param path: the solution path
    :return: the maze image with the solution path
    """
    for i in range(len(path)):
        x, y = int(path[i][0]), int(path[i][1])
        img[x][y] = 255
    return img


def distance(x1: float, y1: float, x2: float, y2: float):
    """Calculate the distance between (x1, y1) and (x2, y2).

    :param x1: x-coordinate of (x1, y1)
    :param y1: y-coordinate of (x1, y1)
    :param x2: x-coordinate of (x2, y2)
    :param y2: y-coordinate of (x2, y2)
    :return: the distance between (x1, y1) and (x2, y2)
    """
    dx = x1 - x2
    dy = y1 - y2
    return math.sqrt(dx * dx + dy * dy)


def world_coordinate(coord: Tuple[float, float]):
    """Convert Sensor-Coordinate to World-Coordinate.

    :param coord: Coordinate related to Vision Sensor
    :return: World-Coordinate
    """
    k = 64 * math.sqrt(3)
    x, y = (256 - coord[0]) / k, (coord[1] - 256) / k + 0.8
    return x, y
```