2023光电设计竞赛小车设计思路

**2024光电设计竞赛小车系统设计方案**

**目录**

1. 系统概述
2. 系统架构

2.1 主要功能模块

2.2 系统工作流程

1. 环境配置

3.1 树莓派环境搭建

3.2 视觉Python代码撰写工具

3.3 摄像头选择与参数调整

3.4 树莓派与单片机的硬件连接

1. 代码详解

4.1 主函数（main.py）

4.2 路径规划（FindNearestTreasure.py）

4.3 宝藏识别（JudgeTreasure.py）

4.4 运动控制（MoveControl.py）

4.5 宝藏坐标检测（FindTreasureXY.py）

1. 附录

5.1 参考文献

5.2 经验与杂谈

**1. 系统概述**

本文档详细介绍了2023年光电设计竞赛小车系统的设计和实现。该系统利用树莓派作为中央处理单元，结合OpenCV库进行图像处理，实现小车在复杂迷宫环境中的自主导航和宝藏识别。系统通过摄像头实时捕获迷宫图像，分析宝藏位置，规划最优路径，并控制小车移动以收集宝藏。

**2. 系统架构**

2.1 主要功能模块

* **主函数（main.py）**：作为系统的入口点，负责初始化各个模块，启动宝藏检测、路径规划和运动控制流程。
* **路径规划（FindNearestTreasure.py）**：包含迷宫地图的表示和动态规划算法，用于寻找从当前位置到宝藏的最短路径。
* **宝藏识别（JudgeTreasure.py）**：使用OpenCV进行图像处理，识别宝藏的颜色和形状，判断宝藏的真伪。
* **运动控制（MoveControl.py）**：通过GPIO控制树莓派，发送指令给单片机，实现小车的移动和转向。
* **宝藏坐标检测（FindTreasureXY.py）**：从摄像头捕获的视频中提取宝藏坐标，为路径规划提供起点和终点。

2.2 系统工作流程

1. **初始化**：系统启动，加载迷宫地图和摄像头参数。
2. **宝藏检测**：主函数调用宝藏坐标检测模块，获取宝藏位置。
3. **路径规划**：根据宝藏位置，路径规划模块计算到达宝藏的最优路径。
4. **宝藏识别**：在接近宝藏时，宝藏识别模块分析图像，确定宝藏类型。
5. **运动控制**：运动控制模块根据路径规划结果和宝藏类型，发送指令给单片机，控制小车移动。
6. **循环执行**：系统不断循环执行上述步骤，直至所有宝藏被收集。

**3. 环境配置**

3.1 树莓派环境搭建

* 安装Raspbian操作系统，配置网络和SSH服务。
* 安装OpenCV库，确保版本与代码兼容。

3.2 视觉Python代码撰写工具

* 使用PyCharm或VS Code进行代码开发，利用其调试功能进行错误排查。

3.3 摄像头选择与参数调整

* 选择支持全局快门的摄像头，以减少运动模糊。
* 使用摄像头软件调整白平衡、清晰度、曝光度等参数，确保图像质量。

3.4 树莓派与单片机的硬件连接

* 使用USB连接摄像头到树莓派。
* 通过串口连接树莓派和单片机，确保通信稳定。

**4. 代码详解**

4.1 主函数（main.py）

**功能描述：** 主函数（main.py）是整个智能小车视觉导航系统的入口点，负责初始化系统，协调各个模块的工作，并执行整个寻宝流程。它导入了藏宝图识别、路径规划、宝藏识别和运动控制等模块，通过这些模块的协同工作，实现小车在迷宫中的自主导航和宝藏收集。

**完整程序：**

import FindNearestTreasure  
import os  
import MoveControl  
from FindTreasureXY import findTreasureXY  
from FindNearestTreasure import TreasureDetector  
from JudgeTreasure import judgeTreasure  
import time  
import cv2  
import numpy as np  
  
  
  
pathList4Draw = list()  
END = 0  
numPart = 1  
  
  
FfindTreasureXY = findTreasureXY()  
treasure\_list = FfindTreasureXY[0]  
teamColor = FfindTreasureXY[1]  
  
  
  
TD = TreasureDetector(0)  
TD.classifyXY(treasure\_list)  
  
  
treasurePath, orientPath = TD.findNearestTreasurePoint(numPart, FindNearestTreasure.BEGIN\_POINT\_X,  
 FindNearestTreasure.BEGIN\_POINT\_Y)  
pathList4Draw.append(treasurePath)  
MoveControl.move(orientPath)  
#time.sleep(2)  
cvResult = judgeTreasure()  
print(cvResult)  
  
MoveControl.cvResultMove(cvResult, teamColor=teamColor)  
carResult = TD.updateMazeMap(numPart, cvResult, treasurePath[len(treasurePath) - 1], teamColor, treasurePath[len(treasurePath) - 2])  
  
  
while 1:  
 numPart, directX, directY, path, convertPath = carResult  
 pathList4Draw.append(path)  
 MoveControl.move(convertPath)  
 #time.sleep(2)  
 cvResult = judgeTreasure()  
 print(cvResult)  
 MoveControl.cvResultMove(cvResult, teamColor=teamColor)  
 carResult = TD.updateMazeMap(numPart, cvResult, path[len(path) - 1], teamColor, path[len(path) - 2])  
  
 # 最后一段路  
 if carResult[0] == END:  
 \_, \_, \_, path, convertPath = carResult  
 pathList4Draw.append(path) # 画图功能，可以注释掉。  
 MoveControl.move(convertPath)  
 break

**代码详解：**

1. **导入模块：**
2. import FindNearestTreasure  
   import os  
   import MoveControl  
   from FindTreasureXY import findTreasureXY  
   from FindNearestTreasure import TreasureDetector  
   from JudgeTreasure import judgeTreasure  
   import time  
   import cv2  
   import numpy as np
   * FindNearestTreasure：路径规划模块，负责寻找最近的宝藏路径。
   * os：操作系统模块，用于处理文件和目录。
   * MoveControl：运动控制模块，负责发送指令给小车进行移动。
   * FindTreasureXY：宝藏坐标检测模块，从摄像头捕获的视频中提取宝藏坐标。
   * JudgeTreasure：宝藏识别模块，判断宝藏的真伪和类型。
   * time：时间模块，用于控制程序执行的延时。
   * cv2：OpenCV库，用于图像处理。
   * numpy：数值计算库，用于处理数组和矩阵。
3. **全局变量初始化：**

pathList4Draw = list()  
END = 0  
numPart = 1

* + pathList4Draw：用于存储路径的列表，用于后续的路径绘制。
  + END：定义一个结束标志，用于标识寻宝流程的结束。
  + numPart：当前部分的编号，用于路径规划。

1. **宝藏坐标检测：**

FfindTreasureXY = findTreasureXY()  
treasure\_list = FfindTreasureXY[0]  
teamColor = FfindTreasureXY[1]

* + 使用findTreasureXY()函数获取宝藏列表（treasure\_list）和团队颜色（teamColor）。

1. **宝藏分类：**

TD = TreasureDetector(0)  
TD.classifyXY(treasure\_list)

* + 创建TreasureDetector实例（TD），并调用classifyXY()方法对宝藏进行分类。

1. **路径规划与运动：**

treasurePath, orientPath = TD.findNearestTreasurePoint(numPart, FindNearestTreasure.BEGIN\_POINT\_X,  
 FindNearestTreasure.BEGIN\_POINT\_Y)  
pathList4Draw.append(treasurePath)  
MoveControl.move(orientPath)  
#time.sleep(2)  
cvResult = judgeTreasure()  
print(cvResult)  
  
MoveControl.cvResultMove(cvResult, teamColor=teamColor)  
carResult = TD.updateMazeMap(numPart, cvResult, treasurePath[len(treasurePath) - 1], teamColor, treasurePath[len(treasurePath) - 2])

* + 使用TD.findNearestTreasurePoint()方法规划到最近宝藏的路径（treasurePath）和运动方向（orientPath）。
  + 将路径添加到pathList4Draw列表中。
  + 调用MoveControl.move()方法移动小车。
  + 调用judgeTreasure()方法识别宝藏。
  + 根据识别结果（cvResult），调用MoveControl.cvResultMove()方法调整小车运动。
  + 更新迷宫地图（TD.updateMazeMap()）。

1. **循环寻宝：**

while 1:  
 numPart, directX, directY, path, convertPath = carResult  
 pathList4Draw.append(path)  
 MoveControl.move(convertPath)  
 #time.sleep(2)  
 cvResult = judgeTreasure()  
 print(cvResult)  
 MoveControl.cvResultMove(cvResult, teamColor=teamColor)  
 carResult = TD.updateMazeMap(numPart, cvResult, path[len(path) - 1], teamColor, path[len(path) - 2])  
  
 # 最后一段路  
 if carResult[0] == END:  
 \_, \_, \_, path, convertPath = carResult  
 pathList4Draw.append(path) # 画图功能，可以注释掉。  
 MoveControl.move(convertPath)  
 break

* + 在一个无限循环中，不断执行路径规划、运动控制和宝藏识别。
  + 每次循环都会更新当前部分的编号（numPart）、运动方向（directX, directY）、路径（path）和运动方向（convertPath）。
  + 如果寻宝流程结束（carResult[0] == END），则执行最后一段路径，移动小车到终点，并结束循环。

**注意事项：**

* time.sleep(2)：在实际运行中，这个延时可能需要根据小车的实际运动速度进行调整。
* pathList4Draw列表在实际应用中可能用于可视化路径，如果不需要，可以注释掉相关代码。
* END标志的值需要与迷宫地图的实际终点坐标相匹配。

主函数（main.py）是整个系统的中枢，它确保了各个模块按照预定的逻辑顺序执行，从而实现了小车的自主寻宝功能。在实际部署时，可能需要根据实际环境和硬件条件对代码进行适当的调整。

**4.2 路径规划（FindNearestTreasure.py）**

**功能描述：** FindNearestTreasure.py模块负责实现迷宫中的路径规划功能。它通过分析迷宫地图，使用**BFS算法**，找出从起点到宝藏点的最短路径，并提供路径上的坐标点以及相应的运动方向。该模块还包括了对迷宫地图的更新逻辑，以反映小车的运动和宝藏的收集状态。

**完整程序：**

#从MazeMap和MoveControl模块中导入内容。  
#从Python的内建库中导入deepcopy  
#常量定义:  
#有关迷宫的起点、终点、宝藏类别、方向和旋转方向的常量定义。  
#迷宫地图:  
#使用deepcopy复制MazeMap模块中定义的MAP。  
#查找函数:  
#findImportantPoints(): 用于找到迷宫中的岔路口。  
#find2WallsPoints(): 查找有两面墙的点。  
#findSymmetricalPoint(): 返回输入点的中心对称点。  
  
import MazeMap  
import MoveControl  
from copy import deepcopy  
  
BEGIN\_POINT\_X = 19  
BEGIN\_POINT\_Y = 1  
END\_POINT\_X = 1  
END\_POINT\_Y = 19  
# 象限  
FIRST = 1  
SECOND = 2  
THIRD = 3  
FOURTH = 4  
# 宝藏类别  
TRUE\_RED = 1  
TRUE\_BLUE = 2  
FALSE\_RED = 3  
FALSE\_BLUE = 4  
# 方向  
UP = 0  
RIGHT = 1  
DOWN = 2  
LEFT = 3  
# 旋转方向  
GO\_STRAIGHT = 0  
TURN\_RIGHT = 1  
TURN\_LEFT = 3  
  
mazeMap = deepcopy(MazeMap.MAP) # 迷宫地图  
Red = 1  
Blue = 2  
  
  
# 找出岔路口的点  
def findImportantPoints():  
 MAP = MazeMap.MAP  
 importantList = list()  
 for x in range(1, 20):  
 for y in range(1, 20):  
 count = 0  
 if MAP[x][y] == 1:  
 continue  
 if MAP[x - 1][y] == 0:  
 count = count + 1  
 if MAP[x][y + 1] == 0:  
 count = count + 1  
 if MAP[x + 1][y] == 0:  
 count = count + 1  
 if MAP[x][y - 1] == 0:  
 count = count + 1  
 if count > 2:  
 importantList.append([x, y])  
 return importantList  
  
  
# 找出两面墙的点  
def find2WallsPoints():  
 MAP = MazeMap.MAP  
 twoWallsPointsList = list()  
 for x in range(1, 20):  
 for y in range(1, 20):  
 count = 0  
 if MAP[x][y] == 1:  
 continue  
 if MAP[x - 1][y] == 0:  
 count = count + 1  
 if MAP[x][y + 1] == 0:  
 count = count + 1  
 if MAP[x + 1][y] == 0:  
 count = count + 1  
 if MAP[x][y - 1] == 0:  
 count = count + 1  
 if count == 2:  
 twoWallsPointsList.append([x, y])  
 return twoWallsPointsList  
  
  
def findSymmetricalPoint(input\_x, input\_y): # 找到中心对称的点  
 return 20 - input\_x, 20 - input\_y  
  
  
def setAnotherPathBlock(xy, nearXY):  
 x = xy[0]  
 y = xy[1]  
 # anotherX = nearXY[0]  
 # anotherY = nearXY[1]  
 MAP = MazeMap.MAP  
 if MAP[x - 1][y] == 0 and [x - 1, y] != nearXY:  
 mazeMap[x - 1][y] = 1  
 elif MAP[x][y + 1] == 0 and [x, y + 1] != nearXY:  
 mazeMap[x][y + 1] = 1  
 elif MAP[x + 1][y] == 0 and [x + 1, y] != nearXY:  
 mazeMap[x + 1][y] = 1  
 elif MAP[x][y - 1] == 0 and [x, y - 1] != nearXY:  
 mazeMap[x][y - 1] = 1  
  
  
class TreasureDetector:  
 def \_\_init\_\_(self, count):  
 self.count = count  
 self.Intersection\_coordinates = findImportantPoints()  
 self.twoWallsPoints = find2WallsPoints()  
 self.firstQuadrantList = list()  
 self.secondQuadrantList = list()  
 self.thirdQuadrantList = list()  
 self.fourthQuadrantList = list()  
 # 尝试寻找最短路径的存储数据结构  
 self.searchPath = list()  
 self.orientPath = list()  
  
 def classifyXY(self, treasure\_list): # 对于找到的宝藏坐标list进行象限的分类  
 for xy in treasure\_list:  
 x, y = xy  
 if x > 10 > y:  
 self.firstQuadrantList.append(xy)  
 if x > 10 and y > 10:  
 self.secondQuadrantList.append(xy)  
 if x < 10 and y < 10:  
 self.thirdQuadrantList.append(xy)  
 if x < 10 < y:  
 self.fourthQuadrantList.append(xy)  
  
 def findOneQuadrantXY(self, num): # 返回某个象限的XY坐标的list  
 if num == 1:  
 return self.firstQuadrantList  
 if num == 2:  
 return self.secondQuadrantList  
 if num == 3:  
 return self.thirdQuadrantList  
 if num == 4:  
 return self.fourthQuadrantList  
  
 def updateMazeMap(self, numPart, cvResult, xy, teamColor, nearXY): # 更新mazeMap  
 x, y = xy  
 mazeMap[x][y] = 1  
 if self.twoWallsPoints.count(xy) == 1:  
 setAnotherPathBlock(xy, nearXY)  
 if cvResult is None:  
 print("该点宝物识别失败或无宝物")  
 MoveControl.turnBack()  
 convX, convY = findSymmetricalPoint(x, y)  
 if teamColor == Red:  
 if numPart == FIRST:  
 if cvResult == TRUE\_RED:  
 self.firstQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if len(self.firstQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.firstQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.fourthQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 if cvResult == FALSE\_RED or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 if len(self.firstQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.firstQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.fourthQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_BLUE or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 anotherX, anotherY = self.firstQuadrantList[0]  
 fourthX, fourthY = self.fourthQuadrantList.pop(1 - self.fourthQuadrantList.index([convX, convY]))  
 mazeMap[fourthX][fourthY] = 1  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [FIRST, anotherX, anotherY, path, convertPath]  
 if cvResult == FALSE\_BLUE or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 anotherX, anotherY = self.firstQuadrantList[0]  
 fourthX, fourthY = self.fourthQuadrantList.pop(1 - self.fourthQuadrantList.index([convX, convY]))  
 mazeMap[fourthX][fourthY] = 1  
 #fourthX, fourthY = self.fourthQuadrantList.pop(0)  
 #mazeMap[fourthX][fourthY] = 1  
 #fourthX, fourthY = self.fourthQuadrantList.pop(0)  
 #mazeMap[fourthX][fourthY] = 1  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [FIRST, anotherX, anotherY, path, convertPath]  
 if numPart == SECOND:  
 if cvResult == TRUE\_RED:  
 self.secondQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if len(self.secondQuadrantList) == 0:  
 if len(self.thirdQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 else:  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 else:  
 anotherX, anotherY = self.secondQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.thirdQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 nextX, nextY = self.thirdQuadrantList[0]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 return [THIRD, nextX, nextY, path, convertPath]  
 if cvResult == FALSE\_RED:  
 self.secondQuadrantList.remove([x, y])  
 if len(self.secondQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 else:  
 anotherX, anotherY = self.secondQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.thirdQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 nextX, nextY = self.thirdQuadrantList[0]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 return [THIRD, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_BLUE or cvResult is None:  
 self.secondQuadrantList.remove([x, y])  
 thirdX, thirdY = self.thirdQuadrantList.pop(1 - self.thirdQuadrantList.index([convX, convY]))  
 mazeMap[thirdX][thirdY] = 1  
 anotherX, anotherY = self.secondQuadrantList[0]  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [SECOND, anotherX, anotherY, path, convertPath]  
 if cvResult == FALSE\_BLUE or cvResult is None:  
 self.secondQuadrantList.remove([x, y])  
 #mazeMap[convX][convY] = 1  
 #thirdX, thirdY = self.thirdQuadrantList.pop(0)  
 #mazeMap[thirdX][thirdY] = 1  
 # 获取 self.thirdQuadrantList 中与 [convX, convY] 这个坐标不同的第一个坐标 [thirdX, thirdY]  
 thirdX, thirdY = self.thirdQuadrantList.pop(1 - self.thirdQuadrantList.index([convX, convY]))  
 # 在迷宫地图的坐标 [thirdX, thirdY] 处设置为 1  
 mazeMap[thirdX][thirdY] = 1  
 anotherX, anotherY = self.secondQuadrantList[0]  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [SECOND, anotherX, anotherY, path, convertPath]  
 if numPart == THIRD:  
 if cvResult == TRUE\_RED:  
 self.thirdQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if self.count == 3:  
 path, convertPath = self.findPath(x, y, END\_POINT\_X, END\_POINT\_Y)  
 return [0, END\_POINT\_X, END\_POINT\_Y, path, convertPath]  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_BLUE or cvResult == FALSE\_RED or cvResult == FALSE\_BLUE or cvResult is None:  
 self.thirdQuadrantList.remove([x, y])  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 if numPart == FOURTH or cvResult is None:  
 if cvResult == TRUE\_RED:  
 self.fourthQuadrantList.remove([x, y])  
 path, convertPath = self.findPath(x, y, END\_POINT\_X, END\_POINT\_Y)  
 return [0, END\_POINT\_X, END\_POINT\_Y, path, convertPath]  
 if teamColor == Blue:  
 if numPart == FIRST:  
 if cvResult == TRUE\_BLUE:  
 self.firstQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if len(self.firstQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.firstQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.fourthQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 if cvResult == FALSE\_BLUE or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 if len(self.firstQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.firstQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.fourthQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_RED or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 anotherX, anotherY = self.firstQuadrantList[0]  
 fourthX, fourthY = self.fourthQuadrantList.pop(1 - self.fourthQuadrantList.index([convX, convY]))  
 mazeMap[fourthX][fourthY] = 1  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [FIRST, anotherX, anotherY, path, convertPath]  
 if cvResult == FALSE\_RED or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 anotherX, anotherY = self.firstQuadrantList[0]  
 #fourthX, fourthY = self.fourthQuadrantList.pop(0)  
 fourthX, fourthY = self.fourthQuadrantList.pop(1 - self.fourthQuadrantList.index([convX, convY]))  
 mazeMap[fourthX][fourthY] = 1  
 #fourthX, fourthY = self.fourthQuadrantList.pop(0)  
 # mazeMap[fourthX][fourthY] = 1  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [FIRST, anotherX, anotherY, path, convertPath]  
 if numPart == SECOND:  
 if cvResult == TRUE\_BLUE:  
 self.secondQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if len(self.secondQuadrantList) == 0:  
 if len(self.thirdQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.secondQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.thirdQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 nextX, nextY = self.thirdQuadrantList[0]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 return [THIRD, nextX, nextY, path, convertPath]  
 if cvResult == FALSE\_BLUE:  
 self.secondQuadrantList.remove([x, y])  
 if len(self.secondQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.secondQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.thirdQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 nextX, nextY = self.thirdQuadrantList[0]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 return [THIRD, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_RED or cvResult is None:  
 self.secondQuadrantList.remove([x, y])  
 thirdX, thirdY = self.thirdQuadrantList.pop(1 - self.thirdQuadrantList.index([convX, convY]))  
 mazeMap[thirdX][thirdY] = 1  
 anotherX, anotherY = self.secondQuadrantList[0]  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [SECOND, anotherX, anotherY, path, convertPath]  
 if cvResult == FALSE\_RED or cvResult is None:  
 self.secondQuadrantList.remove([x, y])  
 mazeMap[convX][convY] = 1  
 #thirdX, thirdY = self.thirdQuadrantList.pop(0)  
 #mazeMap[thirdX][thirdY] = 1  
 #thirdX, thirdY = self.thirdQuadrantList.pop(0)  
 thirdX, thirdY = self.thirdQuadrantList.pop(1 - self.thirdQuadrantList.index([convX, convY]))  
 mazeMap[thirdX][thirdY] = 1  
 anotherX, anotherY = self.secondQuadrantList[0]  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [SECOND, anotherX, anotherY, path, convertPath]  
 if numPart == THIRD:  
 if cvResult == TRUE\_BLUE:  
 self.thirdQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if self.count == 3:  
 path, convertPath = self.findPath(x, y, END\_POINT\_X, END\_POINT\_Y)  
 return [0, END\_POINT\_X, END\_POINT\_Y, path, convertPath]  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_RED or cvResult == FALSE\_BLUE or cvResult == FALSE\_RED or cvResult is None:  
 self.thirdQuadrantList.remove([x, y])  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 if numPart == FOURTH:  
 if cvResult == TRUE\_BLUE or cvResult is None:  
 self.fourthQuadrantList.remove([x, y])  
 path, convertPath = self.findPath(x, y, END\_POINT\_X, END\_POINT\_Y)  
 return [0, END\_POINT\_X, END\_POINT\_Y, path, convertPath]  
  
 def findNearestTreasurePoint(self, numPart, beginX, beginY): # 针对num象限进行搜索最近的路径的坐标点  
 index = 0 # 坐标点存储在list中的索引参数  
 father\_point = -1 # 广度优先搜索的父节点索引参数  
 self.searchPath.clear()  
 self.searchPath.append([beginX, beginY, index, father\_point]) # 将起点加入list  
 index += 1  
 newMap = deepcopy(mazeMap)  
 newMap[beginX][beginY] = 1 # 将加入list中的坐标对应的迷宫地图设置为1，表示此路已经经历过，不允许再次遍历  
 p = 0 # 表示队列中首指针的索引，用于出队（并非真出队，只是访问该点），同时也是广度优先子节点的父节点索引  
 chosenQuadrantList = self.findOneQuadrantXY(numPart)  
  
 while p != index: # 队列不空就循环广度优先搜索  
 node = self.searchPath[p]  
 x, y, \_, \_ = node  
 # 遍历四周：若是路径可以走，则加入队列，若是目标点，则break跳出循环  
 if newMap[x - 1][y] == 0:  
 self.searchPath.append([x - 1, y, index, p])  
 index += 1  
 newMap[x - 1][y] = 1  
 if [x - 1, y] in chosenQuadrantList:  
 break  
 if newMap[x][y + 1] == 0:  
 index += 1  
 self.searchPath.append([x, y + 1, index, p])  
 newMap[x][y + 1] = 1  
 if [x, y + 1] in chosenQuadrantList:  
 break  
 if newMap[x + 1][y] == 0:  
 index += 1  
 self.searchPath.append([x + 1, y, index, p])  
 newMap[x + 1][y] = 1  
 if [x + 1, y] in chosenQuadrantList:  
 break  
 if newMap[x][y - 1] == 0:  
 index += 1  
 self.searchPath.append([x, y - 1, index, p])  
 newMap[x][y - 1] = 1  
 if [x, y - 1] in chosenQuadrantList:  
 break  
 p += 1 # 继续遍历下一个坐标  
 index -= 1 # 此时已经找到目标点  
 path = list() # 用于保存真正的路径  
  
 node = self.searchPath[index] # 反向去寻找父坐标，直到找到初始坐标  
 while node[3] != -1:  
 path.append([node[0], node[1]])  
 node = self.searchPath[node[3]]  
 path.append([node[0], node[1]]) # 初始坐标也加进来  
 path.reverse() # 反转list，正向输出  
 print(path)  
 print("路径长度：", len(path), "到达点：", path[len(path) - 1])  
 convertPath = self.convert2OrientPath(deepcopy(path))  
 return path, convertPath # , convertPath  
  
 def findPath(self, begin\_x, begin\_y, end\_x, end\_y):  
 index = 0 # 坐标点存储在list中的索引参数  
 father\_point = -1 # 广度优先搜索的父节点索引参数  
 self.searchPath.clear()  
 self.searchPath.append([begin\_x, begin\_y, index, father\_point]) # 将起点加入list  
 index += 1  
 newMap = deepcopy(mazeMap)  
 newMap[begin\_x][begin\_y] = 1 # 将加入list中的坐标对应的迷宫地图设置为1，表示此路已经经历过，不允许再次遍历  
 p = 0 # 表示队列中首指针的索引，用于出队（并非真出队，只是访问该点），同时也是广度优先子节点的父节点索引  
  
 while p != index: # 队列不空就循环广度优先搜索  
 node = self.searchPath[p]  
 x = node[0]  
 y = node[1]  
 # 遍历四周：若是路径可以走，则加入队列，若是目标点，则break跳出循环  
 if newMap[x - 1][y] == 0:  
 self.searchPath.append([x - 1, y, index, p])  
 index += 1  
 newMap[x - 1][y] = 1  
 if x - 1 == end\_x and y == end\_y:  
 break  
 if newMap[x][y + 1] == 0:  
 index += 1  
 self.searchPath.append([x, y + 1, index, p])  
 newMap[x][y + 1] = 1  
 if x == end\_x and y + 1 == end\_y:  
 break  
 if newMap[x + 1][y] == 0:  
 index += 1  
 self.searchPath.append([x + 1, y, index, p])  
 newMap[x + 1][y] = 1  
 if x + 1 == end\_x and y == end\_y:  
 break  
 if newMap[x][y - 1] == 0:  
 index += 1  
 self.searchPath.append([x, y - 1, index, p])  
 newMap[x][y - 1] = 1  
 if x == end\_x and y - 1 == end\_y:  
 break  
 p += 1 # 继续遍历下一个坐标  
 index -= 1 # 此时已经找到目标点  
 path = list() # 用于保存真正的路径  
  
 node = self.searchPath[index] # 反向去寻找父坐标，直到找到初始坐标  
 while node[3] != -1:  
 path.append([node[0], node[1]])  
 node = self.searchPath[node[3]]  
 path.append([node[0], node[1]]) # 初始坐标也加进来  
 path.reverse() # 反转list，正向输出  
 print(path)  
 print("路径长度：", len(path), "到达点：", path[len(path) - 1])  
 convertPath = self.convert2OrientPath(deepcopy(path))  
 return path, convertPath # , convertPath  
  
 def convert2OrientPath(self, path):  
 orientationList = list()  
 result = -1  
 i = 0  
 n = len(path)  
 while i + 1 < n:  
 runX = path[i + 1][0] - path[i][0]  
 runY = path[i + 1][1] - path[i][1]  
 if runX == 1 and runY == 0:  
 result = DOWN  
 if runX == 0 and runY == 1:  
 result = RIGHT  
 if runX == -1 and runY == 0:  
 result = UP  
 if runX == 0 and runY == -1:  
 result = LEFT  
 self.orientPath.append(result)  
 i = i + 1  
 print("路径方向：", self.orientPath)  
 path.reverse()  
 locationList = path  
 destination = locationList[0]  
 self.orientPath.reverse()  
 reversedOrientPath = self.orientPath  
 nowLocation = locationList.pop()  
 nowOrientation = reversedOrientPath.pop()  
 nextLocation = locationList.pop()  
 nextOrientation = reversedOrientPath.pop()  
 while nextLocation != destination:  
 if nextOrientation - nowOrientation == 1 or nextOrientation - nowOrientation == -3:  
 orientationList.append(TURN\_RIGHT)  
 nowOrientation = nextOrientation  
 if len(reversedOrientPath) != 0:  
 nextOrientation = reversedOrientPath.pop()  
 nowLocation = nextLocation  
 nextLocation = locationList.pop()  
 continue  
 if nextOrientation - nowOrientation == -1 or nextOrientation - nowOrientation == 3:  
 orientationList.append(TURN\_LEFT)  
 nowOrientation = nextOrientation  
 if len(reversedOrientPath) != 0:  
 nextOrientation = reversedOrientPath.pop()  
 nowLocation = nextLocation  
 nextLocation = locationList.pop()  
 continue  
 if nextOrientation - nowOrientation == 0 and self.Intersection\_coordinates.count(nextLocation) == 1:  
 orientationList.append(GO\_STRAIGHT)  
 nowOrientation = nextOrientation  
 if len(reversedOrientPath) != 0:  
 nextOrientation = reversedOrientPath.pop()  
 nowLocation = nextLocation  
 nextLocation = locationList.pop()  
  
 print("旋转方向：", orientationList)  
 print("--------------------")  
  
  
 return orientationList

**广度优先搜索（Breadth-First Search, BFS）算法介绍：** BFS是一种用于遍历或搜索树或图的算法，这种算法从根节点开始，逐层访问节点，直到找到目标节点或遍历完所有可达节点。BFS在图的搜索中特别有用，尤其是在寻找最短路径时。它使用队列（Queue）来实现，确保按照层次顺序访问节点。

算法步骤：

1. 创建一个队列，将起始节点加入队列。
2. 创建一个集合（Set）来存储已访问过的节点，避免重复访问。
3. 当队列不为空时，执行以下步骤： a. 从队列中取出一个节点（称为当前节点）。 b. 检查当前节点是否为目标节点。如果是，返回路径。 c. 否则，遍历当前节点的所有邻接节点（未访问过的）：
   * 将邻接节点加入队列。
   * 将邻接节点加入已访问集合。 d. 如果所有节点都已访问，且未找到目标节点，搜索失败。

特点：

* BFS保证找到的路径是最短的（在无权图中）。
* 在图中，BFS可以找到从起始节点到其他所有节点的最短路径。
* BFS的时间复杂度为O(V+E)，其中V是顶点数，E是边数。

例子：

假设我们有一个简单的图，表示为邻接列表，我们想要找到从节点1到节点4的最短路径。

图的邻接列表表示：

1: [2, 3]

2: [1, 4, 5]

3: [1, 6]

4: [2]

5: [2]

6: [3]

BFS过程：

1. 初始化队列：[1]，已访问集合：{}
2. 取出节点1，访问它，加入已访问集合：{1}。
3. 节点1的邻接节点是2和3，将它们加入队列：[2, 3]，已访问集合：{1, 2, 3}。
4. 取出节点2，访问它，加入已访问集合：{1, 2, 3, 4}。
5. 节点2的邻接节点是4和5，将它们加入队列：[4, 5]，已访问集合：{1, 2, 3, 4, 5}。
6. 取出节点4，访问它，这是目标节点，返回路径。

路径： 1 -> 2 -> 4

在这个例子中，BFS找到了从节点1到节点4的最短路径。在实际应用中，BFS可以用于解决如迷宫寻路、社交网络分析、网络爬虫等问题。

**完整程序：**

**代码详解：**

1. **导入模块与常量定义：**

import MazeMap  
import MoveControl  
from copy import deepcopy

* + 导入MazeMap和MoveControl模块，以及deepcopy函数，用于复制迷宫地图。
  + 定义迷宫的起点、终点、宝藏类别、方向和旋转方向的常量。

1. **迷宫地图初始化：**

BEGIN\_POINT\_X = 19  
BEGIN\_POINT\_Y = 1  
END\_POINT\_X = 1  
END\_POINT\_Y = 19  
# 象限  
FIRST = 1  
SECOND = 2  
THIRD = 3  
FOURTH = 4  
# 宝藏类别  
TRUE\_RED = 1  
TRUE\_BLUE = 2  
FALSE\_RED = 3  
FALSE\_BLUE = 4  
# 方向  
UP = 0  
RIGHT = 1  
DOWN = 2  
LEFT = 3  
# 旋转方向  
GO\_STRAIGHT = 0  
TURN\_RIGHT = 1  
TURN\_LEFT = 3  
  
mazeMap = deepcopy(MazeMap.MAP) # 迷宫地图  
Red = 1  
Blue = 2

* + 使用deepcopy复制MazeMap模块中的迷宫地图（MAP），以便在路径规划时不影响原始地图。

1. **查找函数：**

# 找出岔路口的点  
def findImportantPoints():  
 MAP = MazeMap.MAP  
 importantList = list()  
 for x in range(1, 20):  
 for y in range(1, 20):  
 count = 0  
 if MAP[x][y] == 1:  
 continue  
 if MAP[x - 1][y] == 0:  
 count = count + 1  
 if MAP[x][y + 1] == 0:  
 count = count + 1  
 if MAP[x + 1][y] == 0:  
 count = count + 1  
 if MAP[x][y - 1] == 0:  
 count = count + 1  
 if count > 2:  
 importantList.append([x, y])  
 return importantList  
  
  
# 找出两面墙的点  
def find2WallsPoints():  
 MAP = MazeMap.MAP  
 twoWallsPointsList = list()  
 for x in range(1, 20):  
 for y in range(1, 20):  
 count = 0  
 if MAP[x][y] == 1:  
 continue  
 if MAP[x - 1][y] == 0:  
 count = count + 1  
 if MAP[x][y + 1] == 0:  
 count = count + 1  
 if MAP[x + 1][y] == 0:  
 count = count + 1  
 if MAP[x][y - 1] == 0:  
 count = count + 1  
 if count == 2:  
 twoWallsPointsList.append([x, y])  
 return twoWallsPointsList  
  
  
def findSymmetricalPoint(input\_x, input\_y): # 找到中心对称的点  
 return 20 - input\_x, 20 - input\_y  
  
  
def setAnotherPathBlock(xy, nearXY):  
 x = xy[0]  
 y = xy[1]  
 # anotherX = nearXY[0]  
 # anotherY = nearXY[1]  
 MAP = MazeMap.MAP  
 if MAP[x - 1][y] == 0 and [x - 1, y] != nearXY:  
 mazeMap[x - 1][y] = 1  
 elif MAP[x][y + 1] == 0 and [x, y + 1] != nearXY:  
 mazeMap[x][y + 1] = 1  
 elif MAP[x + 1][y] == 0 and [x + 1, y] != nearXY:  
 mazeMap[x + 1][y] = 1  
 elif MAP[x][y - 1] == 0 and [x, y - 1] != nearXY:  
 mazeMap[x][y - 1] = 1

* + findImportantPoints()：找出迷宫中的岔路口，即连接多个路径的点。
  + find2WallsPoints()：找出有两面墙的点，这些点可能是路径规划的关键点。
  + findSymmetricalPoint()：计算输入点的中心对称点，用于处理宝藏点的对称性。

1. **TreasureDetector类：**

class TreasureDetector:  
 def \_\_init\_\_(self, count):  
 self.count = count  
 self.Intersection\_coordinates = findImportantPoints()  
 self.twoWallsPoints = find2WallsPoints()  
 self.firstQuadrantList = list()  
 self.secondQuadrantList = list()  
 self.thirdQuadrantList = list()  
 self.fourthQuadrantList = list()  
 # 尝试寻找最短路径的存储数据结构  
 self.searchPath = list()  
 self.orientPath = list()  
  
 def classifyXY(self, treasure\_list): # 对于找到的宝藏坐标list进行象限的分类  
 for xy in treasure\_list:  
 x, y = xy  
 if x > 10 > y:  
 self.firstQuadrantList.append(xy)  
 if x > 10 and y > 10:  
 self.secondQuadrantList.append(xy)  
 if x < 10 and y < 10:  
 self.thirdQuadrantList.append(xy)  
 if x < 10 < y:  
 self.fourthQuadrantList.append(xy)  
  
 def findOneQuadrantXY(self, num): # 返回某个象限的XY坐标的list  
 if num == 1:  
 return self.firstQuadrantList  
 if num == 2:  
 return self.secondQuadrantList  
 if num == 3:  
 return self.thirdQuadrantList  
 if num == 4:  
 return self.fourthQuadrantList  
  
 def updateMazeMap(self, numPart, cvResult, xy, teamColor, nearXY): # 更新mazeMap  
 x, y = xy  
 mazeMap[x][y] = 1  
 if self.twoWallsPoints.count(xy) == 1:  
 setAnotherPathBlock(xy, nearXY)  
 if cvResult is None:  
 print("该点宝物识别失败或无宝物")  
 MoveControl.turnBack()  
 convX, convY = findSymmetricalPoint(x, y)  
 if teamColor == Red:  
 if numPart == FIRST:  
 if cvResult == TRUE\_RED:  
 self.firstQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if len(self.firstQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.firstQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.fourthQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 if cvResult == FALSE\_RED or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 if len(self.firstQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.firstQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.fourthQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_BLUE or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 anotherX, anotherY = self.firstQuadrantList[0]  
 fourthX, fourthY = self.fourthQuadrantList.pop(1 - self.fourthQuadrantList.index([convX, convY]))  
 mazeMap[fourthX][fourthY] = 1  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [FIRST, anotherX, anotherY, path, convertPath]  
 if cvResult == FALSE\_BLUE or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 anotherX, anotherY = self.firstQuadrantList[0]  
 fourthX, fourthY = self.fourthQuadrantList.pop(1 - self.fourthQuadrantList.index([convX, convY]))  
 mazeMap[fourthX][fourthY] = 1  
 #fourthX, fourthY = self.fourthQuadrantList.pop(0)  
 #mazeMap[fourthX][fourthY] = 1  
 #fourthX, fourthY = self.fourthQuadrantList.pop(0)  
 #mazeMap[fourthX][fourthY] = 1  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [FIRST, anotherX, anotherY, path, convertPath]  
 if numPart == SECOND:  
 if cvResult == TRUE\_RED:  
 self.secondQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if len(self.secondQuadrantList) == 0:  
 if len(self.thirdQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 else:  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 else:  
 anotherX, anotherY = self.secondQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.thirdQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 nextX, nextY = self.thirdQuadrantList[0]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 return [THIRD, nextX, nextY, path, convertPath]  
 if cvResult == FALSE\_RED:  
 self.secondQuadrantList.remove([x, y])  
 if len(self.secondQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 else:  
 anotherX, anotherY = self.secondQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.thirdQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 nextX, nextY = self.thirdQuadrantList[0]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 return [THIRD, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_BLUE or cvResult is None:  
 self.secondQuadrantList.remove([x, y])  
 thirdX, thirdY = self.thirdQuadrantList.pop(1 - self.thirdQuadrantList.index([convX, convY]))  
 mazeMap[thirdX][thirdY] = 1  
 anotherX, anotherY = self.secondQuadrantList[0]  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [SECOND, anotherX, anotherY, path, convertPath]  
 if cvResult == FALSE\_BLUE or cvResult is None:  
 self.secondQuadrantList.remove([x, y])  
 #mazeMap[convX][convY] = 1  
 #thirdX, thirdY = self.thirdQuadrantList.pop(0)  
 #mazeMap[thirdX][thirdY] = 1  
 # 获取 self.thirdQuadrantList 中与 [convX, convY] 这个坐标不同的第一个坐标 [thirdX, thirdY]  
 thirdX, thirdY = self.thirdQuadrantList.pop(1 - self.thirdQuadrantList.index([convX, convY]))  
 # 在迷宫地图的坐标 [thirdX, thirdY] 处设置为 1  
 mazeMap[thirdX][thirdY] = 1  
 anotherX, anotherY = self.secondQuadrantList[0]  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [SECOND, anotherX, anotherY, path, convertPath]  
 if numPart == THIRD:  
 if cvResult == TRUE\_RED:  
 self.thirdQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if self.count == 3:  
 path, convertPath = self.findPath(x, y, END\_POINT\_X, END\_POINT\_Y)  
 return [0, END\_POINT\_X, END\_POINT\_Y, path, convertPath]  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_BLUE or cvResult == FALSE\_RED or cvResult == FALSE\_BLUE or cvResult is None:  
 self.thirdQuadrantList.remove([x, y])  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 if numPart == FOURTH or cvResult is None:  
 if cvResult == TRUE\_RED:  
 self.fourthQuadrantList.remove([x, y])  
 path, convertPath = self.findPath(x, y, END\_POINT\_X, END\_POINT\_Y)  
 return [0, END\_POINT\_X, END\_POINT\_Y, path, convertPath]  
 if teamColor == Blue:  
 if numPart == FIRST:  
 if cvResult == TRUE\_BLUE:  
 self.firstQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if len(self.firstQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.firstQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.fourthQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 if cvResult == FALSE\_BLUE or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 if len(self.firstQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.firstQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.fourthQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 path, convertPath = self.findNearestTreasurePoint(SECOND, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [SECOND, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_RED or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 anotherX, anotherY = self.firstQuadrantList[0]  
 fourthX, fourthY = self.fourthQuadrantList.pop(1 - self.fourthQuadrantList.index([convX, convY]))  
 mazeMap[fourthX][fourthY] = 1  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [FIRST, anotherX, anotherY, path, convertPath]  
 if cvResult == FALSE\_RED or cvResult is None:  
 self.firstQuadrantList.remove([x, y])  
 anotherX, anotherY = self.firstQuadrantList[0]  
 #fourthX, fourthY = self.fourthQuadrantList.pop(0)  
 fourthX, fourthY = self.fourthQuadrantList.pop(1 - self.fourthQuadrantList.index([convX, convY]))  
 mazeMap[fourthX][fourthY] = 1  
 #fourthX, fourthY = self.fourthQuadrantList.pop(0)  
 # mazeMap[fourthX][fourthY] = 1  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [FIRST, anotherX, anotherY, path, convertPath]  
 if numPart == SECOND:  
 if cvResult == TRUE\_BLUE:  
 self.secondQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if len(self.secondQuadrantList) == 0:  
 if len(self.thirdQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.secondQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.thirdQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 nextX, nextY = self.thirdQuadrantList[0]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 return [THIRD, nextX, nextY, path, convertPath]  
 if cvResult == FALSE\_BLUE:  
 self.secondQuadrantList.remove([x, y])  
 if len(self.secondQuadrantList) == 0:  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [THIRD, nextX, nextY, path, convertPath]  
 anotherX, anotherY = self.secondQuadrantList.pop()  
 mazeMap[anotherX][anotherY] = 1  
 self.thirdQuadrantList.remove([convX, convY])  
 mazeMap[convX][convY] = 1  
 nextX, nextY = self.thirdQuadrantList[0]  
 path, convertPath = self.findNearestTreasurePoint(THIRD, x, y)  
 return [THIRD, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_RED or cvResult is None:  
 self.secondQuadrantList.remove([x, y])  
 thirdX, thirdY = self.thirdQuadrantList.pop(1 - self.thirdQuadrantList.index([convX, convY]))  
 mazeMap[thirdX][thirdY] = 1  
 anotherX, anotherY = self.secondQuadrantList[0]  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [SECOND, anotherX, anotherY, path, convertPath]  
 if cvResult == FALSE\_RED or cvResult is None:  
 self.secondQuadrantList.remove([x, y])  
 mazeMap[convX][convY] = 1  
 #thirdX, thirdY = self.thirdQuadrantList.pop(0)  
 #mazeMap[thirdX][thirdY] = 1  
 #thirdX, thirdY = self.thirdQuadrantList.pop(0)  
 thirdX, thirdY = self.thirdQuadrantList.pop(1 - self.thirdQuadrantList.index([convX, convY]))  
 mazeMap[thirdX][thirdY] = 1  
 anotherX, anotherY = self.secondQuadrantList[0]  
 path, convertPath = self.findPath(x, y, anotherX, anotherY)  
 return [SECOND, anotherX, anotherY, path, convertPath]  
 if numPart == THIRD:  
 if cvResult == TRUE\_BLUE:  
 self.thirdQuadrantList.remove([x, y])  
 self.count = self.count + 1  
 if self.count == 3:  
 path, convertPath = self.findPath(x, y, END\_POINT\_X, END\_POINT\_Y)  
 return [0, END\_POINT\_X, END\_POINT\_Y, path, convertPath]  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 if cvResult == TRUE\_RED or cvResult == FALSE\_BLUE or cvResult == FALSE\_RED or cvResult is None:  
 self.thirdQuadrantList.remove([x, y])  
 path, convertPath = self.findNearestTreasurePoint(FOURTH, x, y)  
 nextX, nextY = path[len(path) - 1]  
 return [FOURTH, nextX, nextY, path, convertPath]  
 if numPart == FOURTH:  
 if cvResult == TRUE\_BLUE or cvResult is None:  
 self.fourthQuadrantList.remove([x, y])  
 path, convertPath = self.findPath(x, y, END\_POINT\_X, END\_POINT\_Y)  
 return [0, END\_POINT\_X, END\_POINT\_Y, path, convertPath]  
  
 def findNearestTreasurePoint(self, numPart, beginX, beginY): # 针对num象限进行搜索最近的路径的坐标点  
 index = 0 # 坐标点存储在list中的索引参数  
 father\_point = -1 # 广度优先搜索的父节点索引参数  
 self.searchPath.clear()  
 self.searchPath.append([beginX, beginY, index, father\_point]) # 将起点加入list  
 index += 1  
 newMap = deepcopy(mazeMap)  
 newMap[beginX][beginY] = 1 # 将加入list中的坐标对应的迷宫地图设置为1，表示此路已经经历过，不允许再次遍历  
 p = 0 # 表示队列中首指针的索引，用于出队（并非真出队，只是访问该点），同时也是广度优先子节点的父节点索引  
 chosenQuadrantList = self.findOneQuadrantXY(numPart)  
  
 while p != index: # 队列不空就循环广度优先搜索  
 node = self.searchPath[p]  
 x, y, \_, \_ = node  
 # 遍历四周：若是路径可以走，则加入队列，若是目标点，则break跳出循环  
 if newMap[x - 1][y] == 0:  
 self.searchPath.append([x - 1, y, index, p])  
 index += 1  
 newMap[x - 1][y] = 1  
 if [x - 1, y] in chosenQuadrantList:  
 break  
 if newMap[x][y + 1] == 0:  
 index += 1  
 self.searchPath.append([x, y + 1, index, p])  
 newMap[x][y + 1] = 1  
 if [x, y + 1] in chosenQuadrantList:  
 break  
 if newMap[x + 1][y] == 0:  
 index += 1  
 self.searchPath.append([x + 1, y, index, p])  
 newMap[x + 1][y] = 1  
 if [x + 1, y] in chosenQuadrantList:  
 break  
 if newMap[x][y - 1] == 0:  
 index += 1  
 self.searchPath.append([x, y - 1, index, p])  
 newMap[x][y - 1] = 1  
 if [x, y - 1] in chosenQuadrantList:  
 break  
 p += 1 # 继续遍历下一个坐标  
 index -= 1 # 此时已经找到目标点  
 path = list() # 用于保存真正的路径  
  
 node = self.searchPath[index] # 反向去寻找父坐标，直到找到初始坐标  
 while node[3] != -1:  
 path.append([node[0], node[1]])  
 node = self.searchPath[node[3]]  
 path.append([node[0], node[1]]) # 初始坐标也加进来  
 path.reverse() # 反转list，正向输出  
 print(path)  
 print("路径长度：", len(path), "到达点：", path[len(path) - 1])  
 convertPath = self.convert2OrientPath(deepcopy(path))  
 return path, convertPath # , convertPath  
  
 def findPath(self, begin\_x, begin\_y, end\_x, end\_y):  
 index = 0 # 坐标点存储在list中的索引参数  
 father\_point = -1 # 广度优先搜索的父节点索引参数  
 self.searchPath.clear()  
 self.searchPath.append([begin\_x, begin\_y, index, father\_point]) # 将起点加入list  
 index += 1  
 newMap = deepcopy(mazeMap)  
 newMap[begin\_x][begin\_y] = 1 # 将加入list中的坐标对应的迷宫地图设置为1，表示此路已经经历过，不允许再次遍历  
 p = 0 # 表示队列中首指针的索引，用于出队（并非真出队，只是访问该点），同时也是广度优先子节点的父节点索引  
  
 while p != index: # 队列不空就循环广度优先搜索  
 node = self.searchPath[p]  
 x = node[0]  
 y = node[1]  
 # 遍历四周：若是路径可以走，则加入队列，若是目标点，则break跳出循环  
 if newMap[x - 1][y] == 0:  
 self.searchPath.append([x - 1, y, index, p])  
 index += 1  
 newMap[x - 1][y] = 1  
 if x - 1 == end\_x and y == end\_y:  
 break  
 if newMap[x][y + 1] == 0:  
 index += 1  
 self.searchPath.append([x, y + 1, index, p])  
 newMap[x][y + 1] = 1  
 if x == end\_x and y + 1 == end\_y:  
 break  
 if newMap[x + 1][y] == 0:  
 index += 1  
 self.searchPath.append([x + 1, y, index, p])  
 newMap[x + 1][y] = 1  
 if x + 1 == end\_x and y == end\_y:  
 break  
 if newMap[x][y - 1] == 0:  
 index += 1  
 self.searchPath.append([x, y - 1, index, p])  
 newMap[x][y - 1] = 1  
 if x == end\_x and y - 1 == end\_y:  
 break  
 p += 1 # 继续遍历下一个坐标  
 index -= 1 # 此时已经找到目标点  
 path = list() # 用于保存真正的路径  
  
 node = self.searchPath[index] # 反向去寻找父坐标，直到找到初始坐标  
 while node[3] != -1:  
 path.append([node[0], node[1]])  
 node = self.searchPath[node[3]]  
 path.append([node[0], node[1]]) # 初始坐标也加进来  
 path.reverse() # 反转list，正向输出  
 print(path)  
 print("路径长度：", len(path), "到达点：", path[len(path) - 1])  
 convertPath = self.convert2OrientPath(deepcopy(path))  
 return path, convertPath # , convertPath  
  
 def convert2OrientPath(self, path):  
 orientationList = list()  
 result = -1  
 i = 0  
 n = len(path)  
 while i + 1 < n:  
 runX = path[i + 1][0] - path[i][0]  
 runY = path[i + 1][1] - path[i][1]  
 if runX == 1 and runY == 0:  
 result = DOWN  
 if runX == 0 and runY == 1:  
 result = RIGHT  
 if runX == -1 and runY == 0:  
 result = UP  
 if runX == 0 and runY == -1:  
 result = LEFT  
 self.orientPath.append(result)  
 i = i + 1  
 print("路径方向：", self.orientPath)  
 path.reverse()  
 locationList = path  
 destination = locationList[0]  
 self.orientPath.reverse()  
 reversedOrientPath = self.orientPath  
 nowLocation = locationList.pop()  
 nowOrientation = reversedOrientPath.pop()  
 nextLocation = locationList.pop()  
 nextOrientation = reversedOrientPath.pop()  
 while nextLocation != destination:  
 if nextOrientation - nowOrientation == 1 or nextOrientation - nowOrientation == -3:  
 orientationList.append(TURN\_RIGHT)  
 nowOrientation = nextOrientation  
 if len(reversedOrientPath) != 0:  
 nextOrientation = reversedOrientPath.pop()  
 nowLocation = nextLocation  
 nextLocation = locationList.pop()  
 continue  
 if nextOrientation - nowOrientation == -1 or nextOrientation - nowOrientation == 3:  
 orientationList.append(TURN\_LEFT)  
 nowOrientation = nextOrientation  
 if len(reversedOrientPath) != 0:  
 nextOrientation = reversedOrientPath.pop()  
 nowLocation = nextLocation  
 nextLocation = locationList.pop()  
 continue  
 if nextOrientation - nowOrientation == 0 and self.Intersection\_coordinates.count(nextLocation) == 1:  
 orientationList.append(GO\_STRAIGHT)  
 nowOrientation = nextOrientation  
 if len(reversedOrientPath) != 0:  
 nextOrientation = reversedOrientPath.pop()  
 nowLocation = nextLocation  
 nextLocation = locationList.pop()  
  
 print("旋转方向：", orientationList)  
 print("--------------------")

* + \_\_init\_\_：初始化TreasureDetector实例，包括岔路口列表、两面墙点列表、四个象限的宝藏列表、搜索路径和运动方向列表。
  + classifyXY()：根据宝藏坐标列表，将宝藏分类到四个象限。
  + findOneQuadrantXY()：根据给定的象限编号，返回该象限的宝藏坐标列表。
  + updateMazeMap()：更新迷宫地图，标记已探索的路径，并处理宝藏点的对称性。
  + findNearestTreasurePoint()：使用广度优先搜索（BFS）算法，找出最近的宝藏点及其路径。
  + findPath()：同样使用BFS算法，找出从起点到终点的路径。
  + convert2OrientPath()：将路径上的坐标转换为运动方向列表，用于控制小车的运动。

**注意事项：**

* 在实际应用中，路径规划模块需要与主函数（main.py）紧密配合，确保小车能够根据规划的路径进行移动。
* updateMazeMap()方法中的逻辑较为复杂，需要根据小车的运动方向和宝藏点的对称性进行适当的路径封锁和更新。
* findNearestTreasurePoint()和findPath()方法中的BFS算法是路径规划的核心，它们能够高效地找到最短路径。
* convert2OrientPath()方法将路径上的坐标转换为运动方向，这是运动控制模块（MoveControl.py）所需的输入。

FindNearestTreasure.py模块是整个寻宝系统的关键部分，它为小车提供了精确的路径信息，使得小车能够在复杂的迷宫中高效地导航。在实际部署时，可能需要根据迷宫的具体布局和障碍物分布对算法进行调整。

**4.3 宝藏识别（JudgeTreasure.py）**

**功能描述：** JudgeTreasure.py模块是智能小车视觉导航系统中的核心组件，负责处理实时视频流并识别宝藏的颜色和真伪。该模块利用OpenCV库的强大图像处理功能，通过分析摄像头捕获的图像帧，识别特定颜色的物体，并根据这些颜色信息判断宝藏的类型。这一过程对于小车的导航策略至关重要，因为它直接影响到小车的运动路径和寻宝效率。

**程序内容：**import cv2  
import numpy as np  
import time  
  
# red1  
redLowHSV1 = np.array([156, 45, 45])  
redHighHSV1 = np.array([180, 255, 255])  
# red2  
redLowHSV2 = np.array([0, 45, 45])  
redHighHSV2 = np.array([12, 255, 255])  
# green  
greenLowHSV = np.array([40 ,60, 30])  
greenHighHSV = np.array([85, 255, 255])  
# blue  
# blueLowHSV = np.array([100, 220, 40])  
blueLowHSV = np.array([90, 95, 75])  
blueHighHSV = np.array([155, 255, 255])  
# yellow  
yellowLowHSV = np.array([20, 30, 50])  
yellowHighHSV = np.array([34, 255, 255])  
  
BLUE\_YELLOW = 2 # 蓝色真  
BLUE\_GREEN = 4 # 蓝色假  
RED\_GREEN = 1 # 红色真  
RED\_YELLOW = 3 # 红色假  
  
  
  
def judgeTreasure():  
 print("WOW")  
 start\_time = time.time()  
 judge\_treasure\_return = False  
 cap = cv2.VideoCapture(0)  
 #cap.release()  
 tag = 0  
 while cap.isOpened():  
 # 逐帧读取视频  
 ret, frame = cap.read()  
 if ret:  
 hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)  
 # 对图片加上hsv # mask是只突出指定颜色的图片  
 picBlueHSV = cv2.inRange(hsv, lowerb=blueLowHSV, upperb=blueHighHSV)  
 picRedHSV = cv2.inRange(hsv, lowerb=redLowHSV1, upperb=redHighHSV1) + cv2.inRange(hsv, lowerb=redLowHSV2,  
 upperb=redHighHSV2)  
 picYellowHSV = cv2.inRange(hsv, lowerb=yellowLowHSV, upperb=yellowHighHSV)  
 picGreenHSV = cv2.inRange(hsv, lowerb=greenLowHSV, upperb=greenHighHSV)  
  
 #cv2.imshow("picBlueHSV", picBlueHSV)  
 #cv2.imshow("picRedHSV", picRedHSV)  
 #cv2.imshow("picYellowHSV", picYellowHSV)  
 #cv2.imshow("picGreenHSV", picGreenHSV)  
  
 # 进行边缘检测  
 edges = cv2.Canny(picBlueHSV, 200, 300)  
 # 轮廓提取  
 contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 外层大色块  
 for contour in contours:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour)  
 w = WH[0]  
 h = WH[1]  
 if h > 40: # 过小直接过滤  
 # 定义腐蚀操作的结构元素  
 kernel = np.ones((3, 3), np.uint8)  
  
 # 进行腐蚀操作  
 eroded\_image = cv2.erode(picYellowHSV, kernel, iterations=1)  
 edges = cv2.Canny(picYellowHSV, 200, 300)  
 contours1, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 内层小色块  
 for contour1 in contours1:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour1)  
 w = WH[0]  
 if w > 20: # 过小直接过滤  
 #cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 now\_time = time.time()  
 return BLUE\_YELLOW  
 edges = cv2.Canny(picGreenHSV, 200, 300)  
 contours2, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 for contour2 in contours2:  
 # 找最小外接矩形  
 center, WH, \_ = cv2.minAreaRect(contour2)  
 w = WH[0]  
 if w > 20: # 过小直接过滤  
 #cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 now\_time = time.time()  
 return BLUE\_GREEN  
 # 进行边缘检测  
 edges = cv2.Canny(picRedHSV, 200, 300)  
 # 轮廓提取  
 contours3, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 外层大色块  
 for contour3 in contours3:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour3)  
 w = WH[0]  
 h = WH[1]  
 if h > 40: # 过小直接过滤  
 edges = cv2.Canny(picGreenHSV, 200, 300)  
 contours5, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 for contour5 in contours5:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour5)  
 w = WH[0]  
 if w > 20: # 过小直接过滤  
 #cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 now\_time = time.time()  
 return RED\_GREEN  
 edges = cv2.Canny(picYellowHSV, 200, 300)  
 contours4, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 内层小色块  
 for contour4 in contours4:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour4)  
 w = WH[0]  
 if w > 20: # 过小直接过滤  
 #cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 now\_time = time.time()  
 return RED\_YELLOW  
   
  
  
 if judgeTreasure is not None:  
 judge\_treasure\_return = True  
 now\_time = time.time()  
  
 if now\_time - start\_time > 4:  
 break  
 cap.release() # 释放摄像头资源  
 cv2.destroyAllWindows()

**代码详解：**

1. **颜色范围定义：**

# red1  
redLowHSV1 = np.array([156, 45, 45])  
redHighHSV1 = np.array([180, 255, 255])  
# red2  
redLowHSV2 = np.array([0, 45, 45])  
redHighHSV2 = np.array([12, 255, 255])  
# green  
greenLowHSV = np.array([40 ,60, 30])  
greenHighHSV = np.array([85, 255, 255])  
# blue  
# blueLowHSV = np.array([100, 220, 40])  
blueLowHSV = np.array([90, 95, 75])  
blueHighHSV = np.array([155, 255, 255])  
# yellow  
yellowLowHSV = np.array([20, 30, 50])  
yellowHighHSV = np.array([34, 255, 255])

* + 在JudgeTreasure.py中，我们定义了五个颜色的HSV范围，这些范围用于在视频流中检测特定颜色的物体。例如，红色（redLowHSV1, redHighHSV1, redLowHSV2, redHighHSV2）和蓝色（blueLowHSV, blueHighHSV）的范围被用来识别红色和蓝色的宝藏。

1. **颜色标记常量：**

BLUE\_YELLOW = 2 # 蓝色真  
BLUE\_GREEN = 4 # 蓝色假  
RED\_GREEN = 1 # 红色真  
RED\_YELLOW = 3 # 红色假

* + 我们定义了四个颜色标记常量：BLUE\_YELLOW（蓝色真）、BLUE\_GREEN（蓝色假）、RED\_GREEN（红色真）、RED\_YELLOW（红色假）。这些常量用于标记在视频帧中检测到的颜色组合，以便后续的逻辑处理。

1. **judgeTreasure函数：**

def judgeTreasure():  
 print("WOW")  
 start\_time = time.time()  
 judge\_treasure\_return = False  
 cap = cv2.VideoCapture(0)  
 #cap.release()  
 tag = 0  
 while cap.isOpened():  
 # 逐帧读取视频  
 ret, frame = cap.read()  
 if ret:  
 hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)  
 # 对图片加上hsv # mask是只突出指定颜色的图片  
 picBlueHSV = cv2.inRange(hsv, lowerb=blueLowHSV, upperb=blueHighHSV)  
 picRedHSV = cv2.inRange(hsv, lowerb=redLowHSV1, upperb=redHighHSV1) + cv2.inRange(hsv, lowerb=redLowHSV2,  
 upperb=redHighHSV2)  
 picYellowHSV = cv2.inRange(hsv, lowerb=yellowLowHSV, upperb=yellowHighHSV)  
 picGreenHSV = cv2.inRange(hsv, lowerb=greenLowHSV, upperb=greenHighHSV)  
  
 #cv2.imshow("picBlueHSV", picBlueHSV)  
 #cv2.imshow("picRedHSV", picRedHSV)  
 #cv2.imshow("picYellowHSV", picYellowHSV)  
 #cv2.imshow("picGreenHSV", picGreenHSV)  
  
 # 进行边缘检测  
 edges = cv2.Canny(picBlueHSV, 200, 300)  
 # 轮廓提取  
 contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 外层大色块  
 for contour in contours:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour)  
 w = WH[0]  
 h = WH[1]  
 if h > 40: # 过小直接过滤  
 # 定义腐蚀操作的结构元素  
 kernel = np.ones((3, 3), np.uint8)  
  
 # 进行腐蚀操作  
 eroded\_image = cv2.erode(picYellowHSV, kernel, iterations=1)  
 edges = cv2.Canny(picYellowHSV, 200, 300)  
 contours1, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 内层小色块  
 for contour1 in contours1:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour1)  
 w = WH[0]  
 if w > 20: # 过小直接过滤  
 #cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 now\_time = time.time()  
 return BLUE\_YELLOW  
 edges = cv2.Canny(picGreenHSV, 200, 300)  
 contours2, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 for contour2 in contours2:  
 # 找最小外接矩形  
 center, WH, \_ = cv2.minAreaRect(contour2)  
 w = WH[0]  
 if w > 20: # 过小直接过滤  
 #cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 now\_time = time.time()  
 return BLUE\_GREEN  
 # 进行边缘检测  
 edges = cv2.Canny(picRedHSV, 200, 300)  
 # 轮廓提取  
 contours3, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 外层大色块  
 for contour3 in contours3:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour3)  
 w = WH[0]  
 h = WH[1]  
 if h > 40: # 过小直接过滤  
 edges = cv2.Canny(picGreenHSV, 200, 300)  
 contours5, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 for contour5 in contours5:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour5)  
 w = WH[0]  
 if w > 20: # 过小直接过滤  
 #cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 now\_time = time.time()  
 return RED\_GREEN  
 edges = cv2.Canny(picYellowHSV, 200, 300)  
 contours4, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 内层小色块  
 for contour4 in contours4:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour4)  
 w = WH[0]  
 if w > 20: # 过小直接过滤  
 #cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 now\_time = time.time()  
 return RED\_YELLOW  
  
  
  
 if judgeTreasure is not None:  
 judge\_treasure\_return = True  
 now\_time = time.time()  
  
 if now\_time - start\_time > 4:  
 break  
 cap.release() # 释放摄像头资源  
 cv2.destroyAllWindows()

* + 这个函数是整个程序的主要逻辑部分。它首先初始化摄像头捕获对象cap，然后进入一个循环，不断读取视频帧。
  + 对于每一帧，函数将其转换为HSV颜色空间，然后使用定义的颜色范围创建掩码图像，突出显示特定颜色的物体。
  + 接着，函数对这些掩码图像进行边缘检测和轮廓提取，以识别物体的轮廓。
  + 对于检测到的轮廓，函数计算最小外接矩形，并根据其尺寸进行过滤，以排除尺寸过小的物体。
  + 最后，根据颜色组合和物体的形状，函数返回相应的颜色标记，如BLUE\_YELLOW或RED\_GREEN。

**实际应用示例：**

* 在main.py中，judgeTreasure()函数被调用，其返回值（cvResult）被用来更新迷宫地图（通过TD.updateMazeMap()）和控制小车的运动（通过MoveControl.cvResultMove()）。
* 例如，如果judgeTreasure()返回BLUE\_YELLOW，这表明小车检测到了一个蓝色的真宝藏，系统将更新地图并规划新的路径以收集这个宝藏。

**注意事项：**

* 在实际应用中，颜色阈值可能需要根据实际环境和光照条件进行调整，以确保准确识别宝藏颜色。
* 边缘检测和轮廓提取的参数（如Canny边缘检测的阈值）可能需要根据图像质量进行优化。
* 在实际部署时，摄像头的稳定性和光照条件对颜色识别的准确性有很大影响，可能需要进行现场调整。

通过JudgeTreasure.py模块，智能小车能够在复杂的迷宫环境中准确识别宝藏，为寻宝任务提供关键的视觉信息，确保小车能够高效地导航并收集所有宝藏。

**4.4 运动控制（MoveControl.py）**

**功能描述：** MoveControl.py模块负责实现智能小车的运动控制逻辑。它通过树莓派发送指令，通过串口与下位机通信，进而控制单片机驱动电机，实现小车的前进、后退、左转、右转等运动。该模块是智能小车导航系统中的关键部分，确保小车能够根据视觉识别的结果和路径规划的指示进行精确移动。

**程序内容：**

import RPi.GPIO as GPIO  
import serial  
import time  
import Distance  
from JudgeTreasure import RED\_GREEN, BLUE\_YELLOW, BLUE\_GREEN, RED\_YELLOW  
#  
  
#  
BLUE\_YELLOW = 2 # 蓝色真  
BLUE\_GREEN = 4 # 蓝色假  
RED\_GREEN = 1 # 红色真  
RED\_YELLOW = 3 # 红色假  
  
  
  
serial\_port = '/dev/ttyACM0'  
  
# 打开串口连接  
ser = serial.Serial(serial\_port, 115200) # 根据需要设置波特率  
  
  
  
Tracking\_1 = 11  
Tracking\_2 = 13  
Tracking\_3 = 15  
Tracking\_4 = 29  
Tracking\_5 = 31  
Tracking\_6 = 33  
Tracking\_7 = 35  
Tracking\_8 = 37  
  
  
#  
GPIO.setmode(GPIO.BOARD)  
GPIO.setwarnings(False)  
  
  
#  
GPIO.setup(Tracking\_1, GPIO.IN)  
GPIO.setup(Tracking\_2, GPIO.IN)  
GPIO.setup(Tracking\_3, GPIO.IN)  
GPIO.setup(Tracking\_4, GPIO.IN)  
GPIO.setup(Tracking\_5, GPIO.IN)  
GPIO.setup(Tracking\_6, GPIO.IN)  
GPIO.setup(Tracking\_7, GPIO.IN)  
GPIO.setup(Tracking\_8, GPIO.IN)  
#  
turnList = list()  
accomplishTag = 1  
#  
  
  
def runn(binary\_code):  
 ser = serial.Serial(serial\_port, 115200)  
 ser.write(binary\_code)  
 ser.close()  
   
def run():  
 ser = serial.Serial(serial\_port, 115200)  
 #binary\_code\_run = b'\x7B\x00\x00\x0F\x00\x00\x00\x00\x00\x74\x7D'  
 binary\_code\_run = b'\x7B\x00\x00\x02\x60\x00\x00\x00\x00\x19\x7D'  
 ser.write(binary\_code\_run)  
 ser.close()  
 return  
  
def back():  
 ser = serial.Serial(serial\_port, 115200)  
 binary\_code\_back = b'\x7B\x00\x00\xFF\xB0\x00\x00\x00\x00\x34\x7D'  
 ser.write(binary\_code\_back)  
 ser.close()  
 return  
  
def stop():  
 ser = serial.Serial(serial\_port, 115200)  
 binary\_code\_stop = b'\x7B\x00\x00\x00\x00\x00\x00\x00\x00\x7B\x7D'  
 ser.write(binary\_code\_stop)  
 ser.close()  
 return  
  
def left():  
 ser = serial.Serial(serial\_port, 115200)  
 binary\_code\_left = b'\x7B\x00\x00\x00\x00\x00\x00\x05\x00\x7E\x7D'  
 #binary\_code\_left = b'\x7B\x00\x00\x00\x00\x00\x00\x08\x00\x73\x7D'  
 #runn(b'\x7B\x00\x00\x00\x00\x00\x00\x03\x00\x78\x7D')  
 #time.sleep(1.15)  
 ser.write(binary\_code\_left)  
 ser.close()  
 return  
  
def right():  
 ser = serial.Serial(serial\_port, 115200)  
 #binary\_code\_right = b'\x7B\x00\x00\x00\x00\x00\x00\xF6\x00\x8D\x7D'  
 binary\_code\_right = b'\x7B\x00\x00\x00\x00\x00\x00\xFB\x00\x80\x7D'  
 ser.write(binary\_code\_right)  
 ser.close()  
 return  
  
#  
def distanceMove():  
 distance = Distance.Distance()  
 while distance > 20:  
 tracking\_function()  
 distance = Distance.Distance()  
 stop()  
 #time.sleep(0.5)  
 return  
  
 #遇到不是直行的路径，就来寻找方向指令，遇到转弯就回来查询转向，并按既定方向指令运动。  
def move(orientList):  
 #处理坐标，转化为控制  
 global turnList, accomplishTag  
 orientList.reverse()  
 turnList = orientList  
 stop()  
 while accomplishTag:  
 tracking\_function()  
 accomplishTag = 1  
 distanceMove()  
#  
#  
def goStraight():  
 #电机驱动代码  
 run()  
 time.sleep(0.2)  
 if len(turnList) == 0:  
 global accomplishTag  
 accomplishTag = 0  
 beginTime = time.time()  
 nowTime = time.time()  
 #最后一个弯道后直行多久  
 while nowTime - beginTime < 0.1:  
 tracking\_function()  
 nowTime = time.time()  
#  
#  
def turnRight():  
 #电机驱动代码  
 run()  
 time.sleep(0.14)  
 right()  
 time.sleep(0.3)  
 while 1:  
 right()  
 time.sleep(0.02)  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 if Tracking\_4Value == 0 or Tracking\_5Value == 0 :  
 stop()  
 #time.sleep(0.2)  
 break  
 if len(turnList) == 0:  
 global accomplishTag  
 accomplishTag = 0  
 beginTime = time.time()  
 nowTime = time.time()  
 #最后一个弯道后直行多久  
 while nowTime - beginTime < 0.3:  
 tracking\_function()  
 nowTime = time.time()  
 #runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 #time.sleep(1.5)  
#  
#  
def turnLeft():  
 #电机驱动代码  
 run()  
 time.sleep(0.14)  
 left()  
 time.sleep(0.3)  
 while 1:  
 left()  
 time.sleep(0.02)  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 if Tracking\_4Value == 0 or Tracking\_5Value == 0:  
 stop()  
 #time.sleep(0.2)  
 break  
 if len(turnList) == 0:  
 global accomplishTag  
 accomplishTag = 0  
 beginTime = time.time()  
 nowTime = time.time()  
 #最后一个弯道后直行多久  
 while nowTime - beginTime < 0.3:  
 tracking\_function()  
 nowTime = time.time()  
 #runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 #time.sleep(1.5)  
#  
def turnBack():  
 right()  
 time.sleep(0.8)  
 print("turn back")  
 while 1:  
 right()  
 time.sleep(0.02)  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 if Tracking\_4Value == 0 or Tracking\_5Value == 0:  
 #left()  
 #time.sleep(0.1)  
 back()  
 time.sleep(0.7)  
 stop()  
 break  
#  
#  
#def turnBack():  
# left()  
# time.sleep(1.35)  
# back()  
# time.sleep(0.8)  
   
  
  
def where2Go():  
 orient = turnList.pop()  
 print("pop orient:", orient)  
 if orient == 1:  
 turnRight()  
 if orient == 3:  
 turnLeft()  
 if orient == 0:  
 goStraight()  
#  
#  
def tracking\_function():  
 #接收输入信号  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
#  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 #四路循迹引脚电平状态  
 #1 0 0 0  
 #1 1 0 0  
 #处理右直角的转动  
#1111 0000 and 1110 0000 and 1100 0000  
 if (Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0) or (Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0)or(Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0)or(Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #0 0 0 1  
 #0 0 1 1  
 #处理左直角的转动  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1) or (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1)or(Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1)or(Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1):  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #0 0 0 0  
 #处理丁字口的转动  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 where2Go()  
 elif (Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 where2Go()  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1):  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #1 0 1 1  
 #0 1 1 1  
 #处理左小弯  
#1110 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x01\x00\x78\x7D')  
 time.sleep(0.01)  
  
#1101 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x02\x00\x7B\x7D')  
 time.sleep(0.01)  
  
#1011 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x03\x00\x7A\x7D')  
 time.sleep(0.01)  
  
#1100 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x02\x00\x7B\x7D')  
 time.sleep(0.01)  
   
#1001 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x03\x00\x7A\x7D')  
 time.sleep(0.01)  
#0011 1111  
 elif Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x04\x00\x7D\x7D')  
 time.sleep(0.01)  
#0111 1111  
 elif Tracking\_1Value == 0 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x05\x00\x7C\x7D')  
 time.sleep(0.01)  
#1100 0111   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x01\x00\x78\x7D')  
 time.sleep(0.01)  
#1000 1111   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x01\x00\x78\x7D')  
 time.sleep(0.01)  
   
#chuliyouxiaowan  
#1111 0111  
  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 time.sleep(0.01)  
  
#1111 1011  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFE\x00\x87\x7D')  
 time.sleep(0.01)  
  
#1111 1101  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFC\x00\x85\x7D')  
 time.sleep(0.01)  
  
#1111 0011  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFE\x00\x87\x7D')  
 time.sleep(0.01)  
   
#1111 1001  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFD\x00\x84\x7D')  
 time.sleep(0.01)  
#1111 1100  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 0 and Tracking\_8Value == 0:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFC\x00\x85\x7D')  
 time.sleep(0.01)  
#1111 1110  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 0:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFB\x00\x82\x7D')  
 time.sleep(0.01)  
#1110 0011   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 time.sleep(0.01)  
#1111 0001   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 time.sleep(0.01)  
  
#  
 #四路循迹引脚电平状态  
 #1 0 0 1  
 #处理直线  
 if Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 run()  
 time.sleep(0.01)  
#  
 #当为1 1 1 1时小车直行  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 run()  
 time.sleep(0.01)  
   
#  
  
def tracking\_function1():  
 #接收输入信号  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
#  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 #四路循迹引脚电平状态  
 #1 0 0 0  
 #1 1 0 0  
 #处理右直角的转动  
#1111 0000 and 1110 0000 and 1100 0000  
 if (Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0)or(Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0)or(Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #0 0 0 1  
 #0 0 1 1  
 #处理左直角的转动  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1)or(Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1)or(Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1):  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #0 0 0 0  
 #处理丁字口的转动  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 where2Go()  
 elif (Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 where2Go()  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1):  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #1 0 1 1  
 #0 1 1 1  
 #处理左小弯  
#1110 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x01\x00\x7B\x7D')  
 time.sleep(0.01)  
  
#1101 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x02\x00\x78\x7D')  
 time.sleep(0.01)  
  
#1011 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x03\x00\x79\x7D')  
 time.sleep(0.01)  
  
#1100 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x02\x00\x78\x7D')  
 time.sleep(0.01)  
   
#1001 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x03\x00\x79\x7D')  
 time.sleep(0.01)  
#0011 1111  
 elif Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x04\x00\x7E\x7D')  
 time.sleep(0.01)  
#0111 1111  
 elif Tracking\_1Value == 0 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x05\x00\x7F\x7D')  
 time.sleep(0.01)  
#1100 0111   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x01\x00\x7B\x7D')  
 time.sleep(0.01)  
#1000 1111   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\x01\x00\x7B\x7D')  
 time.sleep(0.01)  
   
#chuliyouxiaowan  
#1111 0111  
  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x85\x7D')  
 time.sleep(0.01)  
  
#1111 1011  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\xFE\x00\x84\x7D')  
 time.sleep(0.01)  
  
#1111 1101  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\xFC\x00\x86\x7D')  
 time.sleep(0.01)  
  
#1111 0011  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\xFE\x00\x84\x7D')  
 time.sleep(0.01)  
   
#1111 1001  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\xFD\x00\x87\x7D')  
 time.sleep(0.01)  
#1111 1100  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 0 and Tracking\_8Value == 0:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\xFC\x00\x86\x7D')  
 time.sleep(0.01)  
#1111 1110  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 0:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\xFB\x00\x81\x7D')  
 time.sleep(0.01)  
#1110 0011   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\xFF\x00\x85\x7D')  
 time.sleep(0.01)  
#1111 0001   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x01\x00\x00\x00\xFF\x00\x85\x7D')  
 time.sleep(0.01)  
  
#  
 #四路循迹引脚电平状态  
 #1 0 0 1  
 #处理直线  
 if Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 time.sleep(0.01)  
#  
 #当为1 1 1 1时小车直行  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 run()  
 time.sleep(0.01)  
   
  
  
  
  
  
def cvResultMove(cvResult, teamColor):  
 if teamColor == 1:  
 if cvResult == RED\_GREEN:  
 beginTime = time.time()  
 nowTime = time.time()  
#  
 while nowTime - beginTime < 0.8:  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
 #print("imhere")  
 runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 time.sleep(1.8)  
 #if Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 # print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 # run()  
 # #print("1001lo")  
 # time.sleep(0.01)  
 #   
#  
 #当为1 1 1 1时小车直行  
 #elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 # runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 # time.sleep(1)  
 #   
 nowTime = time.time()  
#  
 turnBack()  
 return  
 if cvResult == RED\_YELLOW or cvResult == BLUE\_GREEN or cvResult == BLUE\_YELLOW:  
 turnBack()  
 return  
 if teamColor == 2:  
 if cvResult == BLUE\_YELLOW:  
 beginTime = time.time()  
 nowTime = time.time()  
#  
 while nowTime - beginTime < 0.8:  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
   
 runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 time.sleep(1.4)  
   
 #if Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 # print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 # run()  
 # time.sleep(0.01)  
#  
 #当为1 1 1 1时小车直行  
 #elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 # runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 # time.sleep(1)  
   
 nowTime = time.time()  
#  
 turnBack()  
 return  
 if cvResult == RED\_YELLOW or cvResult == BLUE\_GREEN or cvResult == RED\_GREEN:  
 turnBack()  
 return  
#  
#  
stop()

**代码详解：**

1. **初始化串口连接：**

serial\_port = '/dev/ttyACM0'  
  
# 打开串口连接  
ser = serial.Serial(serial\_port, 115200) # 根据需要设置波特率

* + 使用serial.Serial创建一个串口连接对象ser，用于与下位机通信。

1. **GPIO配置：**

Tracking\_1 = 11  
Tracking\_2 = 13  
Tracking\_3 = 15  
Tracking\_4 = 29  
Tracking\_5 = 31  
Tracking\_6 = 33  
Tracking\_7 = 35  
Tracking\_8 = 37  
  
  
#  
GPIO.setmode(GPIO.BOARD)  
GPIO.setwarnings(False)  
  
  
#  
GPIO.setup(Tracking\_1, GPIO.IN)  
GPIO.setup(Tracking\_2, GPIO.IN)  
GPIO.setup(Tracking\_3, GPIO.IN)  
GPIO.setup(Tracking\_4, GPIO.IN)  
GPIO.setup(Tracking\_5, GPIO.IN)  
GPIO.setup(Tracking\_6, GPIO.IN)  
GPIO.setup(Tracking\_7, GPIO.IN)  
GPIO.setup(Tracking\_8, GPIO.IN)  
#  
turnList = list()  
accomplishTag = 1

* + 使用RPi.GPIO库配置GPIO引脚，用于读取循迹传感器的状态。

1. **运动控制函数：**

def runn(binary\_code):  
 ser = serial.Serial(serial\_port, 115200)  
 ser.write(binary\_code)  
 ser.close()  
   
def run():  
 ser = serial.Serial(serial\_port, 115200)  
 #binary\_code\_run = b'\x7B\x00\x00\x0F\x00\x00\x00\x00\x00\x74\x7D'  
 binary\_code\_run = b'\x7B\x00\x00\x02\x60\x00\x00\x00\x00\x19\x7D'  
 ser.write(binary\_code\_run)  
 ser.close()  
 return  
  
def back():  
 ser = serial.Serial(serial\_port, 115200)  
 binary\_code\_back = b'\x7B\x00\x00\xFF\xB0\x00\x00\x00\x00\x34\x7D'  
 ser.write(binary\_code\_back)  
 ser.close()  
 return  
  
def stop():  
 ser = serial.Serial(serial\_port, 115200)  
 binary\_code\_stop = b'\x7B\x00\x00\x00\x00\x00\x00\x00\x00\x7B\x7D'  
 ser.write(binary\_code\_stop)  
 ser.close()  
 return  
  
def left():  
 ser = serial.Serial(serial\_port, 115200)  
 binary\_code\_left = b'\x7B\x00\x00\x00\x00\x00\x00\x05\x00\x7E\x7D'  
 #binary\_code\_left = b'\x7B\x00\x00\x00\x00\x00\x00\x08\x00\x73\x7D'  
 #runn(b'\x7B\x00\x00\x00\x00\x00\x00\x03\x00\x78\x7D')  
 #time.sleep(1.15)  
 ser.write(binary\_code\_left)  
 ser.close()  
 return  
  
def right():  
 ser = serial.Serial(serial\_port, 115200)  
 #binary\_code\_right = b'\x7B\x00\x00\x00\x00\x00\x00\xF6\x00\x8D\x7D'  
 binary\_code\_right = b'\x7B\x00\x00\x00\x00\x00\x00\xFB\x00\x80\x7D'  
 ser.write(binary\_code\_right)  
 ser.close()  
 return  
  
#  
def distanceMove():  
 distance = Distance.Distance()  
 while distance > 20:  
 tracking\_function()  
 distance = Distance.Distance()  
 stop()  
 #time.sleep(0.5)  
 return  
  
 #遇到不是直行的路径，就来寻找方向指令，遇到转弯就回来查询转向，并按既定方向指令运动。  
def move(orientList):  
 #处理坐标，转化为控制  
 global turnList, accomplishTag  
 orientList.reverse()  
 turnList = orientList  
 stop()  
 while accomplishTag:  
 tracking\_function()  
 accomplishTag = 1  
 distanceMove()  
#  
#  
def goStraight():  
 #电机驱动代码  
 run()  
 time.sleep(0.2)  
 if len(turnList) == 0:  
 global accomplishTag  
 accomplishTag = 0  
 beginTime = time.time()  
 nowTime = time.time()  
 #最后一个弯道后直行多久  
 while nowTime - beginTime < 0.1:  
 tracking\_function()  
 nowTime = time.time()  
#  
#  
def turnRight():  
 #电机驱动代码  
 run()  
 time.sleep(0.14)  
 right()  
 time.sleep(0.3)  
 while 1:  
 right()  
 time.sleep(0.02)  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 if Tracking\_4Value == 0 or Tracking\_5Value == 0 :  
 stop()  
 #time.sleep(0.2)  
 break  
 if len(turnList) == 0:  
 global accomplishTag  
 accomplishTag = 0  
 beginTime = time.time()  
 nowTime = time.time()  
 #最后一个弯道后直行多久  
 while nowTime - beginTime < 0.3:  
 tracking\_function()  
 nowTime = time.time()  
 #runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 #time.sleep(1.5)  
#  
#  
def turnLeft():  
 #电机驱动代码  
 run()  
 time.sleep(0.14)  
 left()  
 time.sleep(0.3)  
 while 1:  
 left()  
 time.sleep(0.02)  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 if Tracking\_4Value == 0 or Tracking\_5Value == 0:  
 stop()  
 #time.sleep(0.2)  
 break  
 if len(turnList) == 0:  
 global accomplishTag  
 accomplishTag = 0  
 beginTime = time.time()  
 nowTime = time.time()  
 #最后一个弯道后直行多久  
 while nowTime - beginTime < 0.3:  
 tracking\_function()  
 nowTime = time.time()  
 #runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 #time.sleep(1.5)  
#  
def turnBack():  
 right()  
 time.sleep(0.8)  
 print("turn back")  
 while 1:  
 right()  
 time.sleep(0.02)  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 if Tracking\_4Value == 0 or Tracking\_5Value == 0:  
 #left()  
 #time.sleep(0.1)  
 back()  
 time.sleep(0.7)  
 stop()  
 break  
#  
#  
#def turnBack():  
# left()  
# time.sleep(1.35)  
# back()  
# time.sleep(0.8)

* + run()：发送指令使小车前进。
  + back()：发送指令使小车后退。
  + stop()：发送指令使小车停止。
  + left()和right()：分别发送指令使小车左转和右转。

1. **路径规划与运动：**

def where2Go():  
 orient = turnList.pop()  
 print("pop orient:", orient)  
 if orient == 1:  
 turnRight()  
 if orient == 3:  
 turnLeft()  
 if orient == 0:  
 goStraight()

* + move(orientList)：根据路径规划结果（orientList），控制小车按照指定的方向移动。
  + goStraight()、turnRight()、turnLeft()、turnBack()：这些函数根据小车的当前位置和路径规划结果，执行相应的运动指令。

1. **循迹逻辑：**

def tracking\_function():  
 #接收输入信号  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
#  
 #print(Tracking\_Left1Value, Tracking\_Left2Value, Tracking\_Right1Value, Tracking\_Right2Value)  
 #四路循迹引脚电平状态  
 #1 0 0 0  
 #1 1 0 0  
 #处理右直角的转动  
#1111 0000 and 1110 0000 and 1100 0000  
 if (Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0) or (Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0)or(Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0)or(Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #0 0 0 1  
 #0 0 1 1  
 #处理左直角的转动  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1) or (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1)or(Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1)or(Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1):  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #0 0 0 0  
 #处理丁字口的转动  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 where2Go()  
 elif (Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 0):  
 where2Go()  
 elif (Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1):  
 where2Go()  
#  
 #四路循迹引脚电平状态  
 #1 0 1 1  
 #0 1 1 1  
 #处理左小弯  
#1110 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x01\x00\x78\x7D')  
 time.sleep(0.01)  
  
#1101 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x02\x00\x7B\x7D')  
 time.sleep(0.01)  
  
#1011 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x03\x00\x7A\x7D')  
 time.sleep(0.01)  
  
#1100 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x02\x00\x7B\x7D')  
 time.sleep(0.01)  
   
#1001 1111  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x03\x00\x7A\x7D')  
 time.sleep(0.01)  
#0011 1111  
 elif Tracking\_1Value == 0 and Tracking\_2Value == 0 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x04\x00\x7D\x7D')  
 time.sleep(0.01)  
#0111 1111  
 elif Tracking\_1Value == 0 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x05\x00\x7C\x7D')  
 time.sleep(0.01)  
#1100 0111   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x01\x00\x78\x7D')  
 time.sleep(0.01)  
#1000 1111   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 0 and Tracking\_3Value == 0 and Tracking\_4Value == 0 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\x01\x00\x78\x7D')  
 time.sleep(0.01)  
   
#chuliyouxiaowan  
#1111 0111  
  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 time.sleep(0.01)  
  
#1111 1011  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFE\x00\x87\x7D')  
 time.sleep(0.01)  
  
#1111 1101  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFC\x00\x85\x7D')  
 time.sleep(0.01)  
  
#1111 0011  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFE\x00\x87\x7D')  
 time.sleep(0.01)  
   
#1111 1001  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFD\x00\x84\x7D')  
 time.sleep(0.01)  
#1111 1100  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 0 and Tracking\_8Value == 0:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFC\x00\x85\x7D')  
 time.sleep(0.01)  
#1111 1110  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 0:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFB\x00\x82\x7D')  
 time.sleep(0.01)  
#1110 0011   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 time.sleep(0.01)  
#1111 0001   
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 0 and Tracking\_6Value == 0 and Tracking\_7Value == 0 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 runn(b'\x7B\x00\x00\x02\x00\x00\x00\xFF\x00\x86\x7D')  
 time.sleep(0.01)  
  
#  
 #四路循迹引脚电平状态  
 #1 0 0 1  
 #处理直线  
 if Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 run()  
 time.sleep(0.01)  
#  
 #当为1 1 1 1时小车直行  
 elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 run()  
 time.sleep(0.01)

* + tracking\_function()和tracking\_function1()：这两个函数根据循迹传感器的输入信号，决定小车的运动方向。它们处理不同的循迹引脚电平状态，以应对不同的迷宫路径。

1. **颜色识别响应：**

def cvResultMove(cvResult, teamColor):  
 if teamColor == 1:  
 if cvResult == RED\_GREEN:  
 beginTime = time.time()  
 nowTime = time.time()  
#  
 while nowTime - beginTime < 0.8:  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
 #print("imhere")  
 runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 time.sleep(1.8)  
 #if Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 # print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 # run()  
 # #print("1001lo")  
 # time.sleep(0.01)  
 #   
#  
 #当为1 1 1 1时小车直行  
 #elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 # runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 # time.sleep(1)  
 #   
 nowTime = time.time()  
#  
 turnBack()  
 return  
 if cvResult == RED\_YELLOW or cvResult == BLUE\_GREEN or cvResult == BLUE\_YELLOW:  
 turnBack()  
 return  
 if teamColor == 2:  
 if cvResult == BLUE\_YELLOW:  
 beginTime = time.time()  
 nowTime = time.time()  
#  
 while nowTime - beginTime < 0.8:  
 Tracking\_1Value = GPIO.input(Tracking\_1)  
 Tracking\_2Value = GPIO.input(Tracking\_2)  
 Tracking\_3Value = GPIO.input(Tracking\_3)  
 Tracking\_4Value = GPIO.input(Tracking\_4)  
 Tracking\_5Value = GPIO.input(Tracking\_5)  
 Tracking\_6Value = GPIO.input(Tracking\_6)  
 Tracking\_7Value = GPIO.input(Tracking\_7)  
 Tracking\_8Value = GPIO.input(Tracking\_8)  
   
 runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 time.sleep(1.4)  
   
 #if Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 0 and Tracking\_5Value == 0 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 # print(Tracking\_1Value,Tracking\_2Value,Tracking\_3Value,Tracking\_4Value,Tracking\_5Value,Tracking\_6Value,Tracking\_7Value,Tracking\_8Value)  
 # run()  
 # time.sleep(0.01)  
#  
 #当为1 1 1 1时小车直行  
 #elif Tracking\_1Value == 1 and Tracking\_2Value == 1 and Tracking\_3Value == 1 and Tracking\_4Value == 1 and Tracking\_5Value == 1 and Tracking\_6Value == 1 and Tracking\_7Value == 1 and Tracking\_8Value == 1:  
 # runn(b'\x7B\x00\x00\x00\x60\x00\x00\x00\x00\x1B\x7D')  
 # time.sleep(1)  
   
 nowTime = time.time()  
#  
 turnBack()  
 return  
 if cvResult == RED\_YELLOW or cvResult == BLUE\_GREEN or cvResult == RED\_GREEN:  
 turnBack()  
 return

* + cvResultMove(cvResult, teamColor)：根据颜色识别的结果（cvResult）和队伍颜色（teamColor），调整小车的运动策略。例如，如果识别到红色宝藏，小车可能会执行特定的运动序列。

**实际应用示例：**

* 在main.py中，cvResultMove()函数会被调用，其返回值（cvResult）用于根据宝藏的颜色类型调整小车的运动。例如，如果小车识别到红色宝藏，且队伍颜色为红色，小车将执行前进动作。

**注意事项：**

* 串口通信的波特率（115200）需要与下位机设置一致。
* GPIO引脚的配置需要根据实际硬件连接进行调整。
* 运动控制指令（如binary\_code\_run、binary\_code\_back等）需要根据实际的电机驱动协议进行设置。
* 循迹逻辑中的电平状态处理需要根据实际的循迹传感器特性进行调整。

MoveControl.py模块为智能小车提供了灵活的运动控制能力，使其能够在复杂的迷宫环境中根据视觉识别和路径规划的结果进行精确导航。在实际部署时，可能需要根据迷宫的具体布局和循迹传感器的性能进行调整。

**4.5 宝藏坐标检测（FindTreasureXY.py）**

**功能描述：** FindTreasureXY.py模块是智能小车视觉导航系统中负责宝藏坐标检测的部分。它通过分析摄像头捕获的视频流，识别出特定形状和颜色的物体（宝藏），并标记出它们的坐标。这些坐标信息对于路径规划和宝藏收集至关重要。

**程序内容：**

import cv2  
import numpy as np  
import os  
  
cornerLocationPoint = list()  
treasureXY = list()  
blueLowHSV = np.array([90, 100, 100])  
blueHighHSV = np.array([124, 255, 255])  
# red1  
redLowHSV1 = np.array([160, 43, 60])  
redHighHSV1 = np.array([180, 255, 255])  
# red2  
redLowHSV2 = np.array([0, 43, 60])  
redHighHSV2 = np.array([5, 255, 255])  
cap = cv2.VideoCapture(0)  
  
  
def findTreasureXY0():  
 while cap.isOpened():  
 # 逐帧读取视频  
 ret, frame = cap.read()  
  
 if ret:  
 # 在这里对每一帧进行处理  
 # 将图像转换为灰度图像  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 进行边缘检测  
 edges = cv2.Canny(gray, 200, 300)  
  
 # 轮廓提取  
 contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 cornerLocationPoint.clear()  
 # 遍历轮廓  
 for contour in contours:  
 # 计算轮廓的边界框  
 center, WH, \_ = cv2.minAreaRect(contour) # 找最小外接矩形  
 w = WH[0]  
 h = WH[1]  
 if w > 40 or h > 40 or w < 20 or h < 20: # 过大过小直接过滤  
 continue  
  
 # 判断边界框是否接近正方形  
 if abs(w - h) < 5:  
 # 绘制边界框  
 x = center[0]  
 y = center[1]  
  
 # NMS  
 tag = 1  
 for point in cornerLocationPoint:  
 x1 = point[0]  
 y1 = point[1]  
 r = pow(pow(x1 - x, 2) + pow(y1 - y, 2), 0.5)  
 if r < 20:  
 tag = 0  
 break  
 if tag:  
 cv2.rectangle(frame, (int(x - w / 2), int(y - h / 2)), (int(x + w / 2), int(y + h / 2)),  
 (0, 255, 0), 2)  
 cornerLocationPoint.append([x, y, w, h])  
  
 # 显示帧  
 cv2.imshow('Camera', frame)  
 cv2.waitKey(2)  
  
 if len(cornerLocationPoint) == 4:  
 # 排序  
 cornerLocationPoint.sort(key=lambda member: member[0], reverse=False)  
 cache = list()  
 if cornerLocationPoint[0][1] < cornerLocationPoint[1][1]:  
 leftTop = cornerLocationPoint.pop(0)  
 leftBottom = cornerLocationPoint.pop(0)  
 cache.append(leftTop)  
 else:  
 leftTop = cornerLocationPoint.pop(1)  
 leftBottom = cornerLocationPoint.pop(0)  
 cache.append(leftTop)  
 cornerLocationPoint.sort(key=lambda member: member[1], reverse=False)  
 rightTop = cornerLocationPoint.pop(0)  
 rightBottom = cornerLocationPoint.pop(0)  
 cache.append(rightTop)  
 cache.append(leftBottom)  
 cache.append(rightBottom)  
  
 # 左上 右上 左下 右下  
 X = [cache[0][0], cache[1][0], cache[2][0], cache[3][0]]  
 Y = [cache[0][1], cache[1][1], cache[2][1], cache[3][1]]  
  
 # 透视变换  
 pic1 = np.float32([[X[0], Y[0]], [X[1], Y[1]], [X[2], Y[2]], [X[3], Y[3]]])  
 pic2 = np.float32([[0, 0], [540, 0], [0, 540], [540, 540]])  
 Matrix = cv2.getPerspectiveTransform(pic1, pic2)  
 perspectivePic = cv2.warpPerspective(frame, Matrix, (540, 540))  
 # cv2.imshow("PerspectiveTransform", perspectivePic)  
  
 # 抵消平均误差  
 width = (cache[0][2] + cache[1][2] + cache[2][2] + cache[3][2]) / 4  
 height = (cache[0][3] + cache[1][3] + cache[2][3] + cache[3][3]) / 4  
 # 截取迷宫区域  
 cropped\_image = perspectivePic[int(height \* 2):int(540 - height \* 2),  
 int(width \* 2):int(540 - width \* 2)]  
  
 # 将彩色图像转换为灰度图像  
 perspectiveGRAYPic = cv2.cvtColor(cropped\_image, cv2.COLOR\_BGR2GRAY)  
 # 应用阈值处理，将灰度图像二值化为黑白图像  
 \_, binary\_image = cv2.threshold(perspectiveGRAYPic, 100, 255, cv2.THRESH\_BINARY) # 光照调整  
 cv2.imshow("THRESH\_BINARY", binary\_image)  
  
 contours, hierarchy = cv2.findContours(binary\_image, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 treasureXY.clear()  
  
 # 获取图像的宽度和高度  
 w, h = perspectiveGRAYPic.shape  
 # 计算每个小块的宽度和高度  
 unitW = int(w/ 10)  
 unitH = int(h/ 10)  
 perspectiveRGBPic = cv2.cvtColor(perspectiveGRAYPic, cv2.COLOR\_GRAY2RGB)  
 # 使用嵌套循环来遍历图像，并划分为100个小块  
 for i in range(0, h, unitH):  
 for j in range(0, w, unitW):  
 # 提取当前块的左上角和右下角坐标  
 x1, y1 = j, i  
 x2, y2 = j + unitW, i + unitH  
 # 在图像上绘制矩形框  
 cv2.rectangle(perspectiveRGBPic, (x1, y1), (x2, y2), (0, 255, 0), 2) # 这里使用绿色框  
  
 countXY = 0  
  
 for contour in contours:  
 (x, y), radius = cv2.minEnclosingCircle(contour)  
 if 8 < radius < 15:  
 center = (int(x), int(y))  
 radius = int(radius)  
 cv2.circle(perspectiveRGBPic, center, radius, (0, 0, 255), 2)  
 countXY = countXY + 1  
 x = int(x / unitW) \* 2 + 1  
 y = int(y / unitH) \* 2 + 1  
 treasureXY.append([int(y), int(x)])  
 cv2.imshow('DONE', perspectiveRGBPic)  
 if countXY == 8:  
 TC = 0  
 teamImage = perspectivePic[300:550, 0:100]  
 hsv = cv2.cvtColor(teamImage, cv2.COLOR\_BGR2HSV)  
  
 picRedHSV = cv2.inRange(hsv, lowerb=redLowHSV1, upperb=redHighHSV1) + cv2.inRange(hsv,  
 lowerb=redLowHSV2,  
 upperb=redHighHSV2)  
 # 进行边缘检测  
 edgesRed = cv2.Canny(picRedHSV, 200, 300)  
 # 轮廓提取  
 contoursRed, \_ = cv2.findContours(edgesRed, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 外层大色块  
 for contour in contoursRed:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour)  
 w = WH[0]  
 h = WH[1]  
 if h > 20: # 过小直接过滤  
 TC = 1  
  
 picBlueHSV = cv2.inRange(hsv, lowerb=blueLowHSV, upperb=blueHighHSV)  
 # 进行边缘检测  
 edgesBlue = cv2.Canny(picBlueHSV, 200, 300)  
 # 轮廓提取  
 contoursBlue, \_ = cv2.findContours(edgesBlue, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 外层大色块  
 for contour in contoursBlue:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour)  
 w = WH[0]  
 h = WH[1]  
 if h > 20: # 过小直接过滤  
 TC = 2  
  
 cv2.imshow('teamImage', teamImage)  
 if TC == 0:  
 print("队伍颜色识别失败")  
 if TC == 1:  
 print("队伍颜色为：红")  
 if TC == 2:  
 print("队伍颜色为：蓝")  
  
 print("识别出的宝藏个数：", len(treasureXY))  
 print("识别出的宝藏坐标：", treasureXY)  
 key = cv2.waitKey(0)  
 # 按下 'Q' 键重新识别  
 if key == ord('Q') or key == ord('q'):  
 # 释放资源  
 cv2.destroyAllWindows()  
 continue  
 # 释放资源  
 cap.release()  
 cv2.destroyAllWindows()  
 return treasureXY,TC  
   
  
  
 else:  
 print("摄像头ret有问题")  
 break  
  
def findTreasureXY():  
 if os.path.exists('treasure\_data.npy'):  
 # 如果文件存在，则直接加载数据并打印坐标  
 treasure\_data = np.load('treasure\_data.npy', allow\_pickle=True).item()  
 treasureXY = treasure\_data['treasure\_list']  
 teamColor = treasure\_data['team\_color']  
 print(treasureXY)  
 print(teamColor)  
 cap.release()  
 cv2.destroyAllWindows()  
 return treasureXY,teamColor  
  
  
 else:  
 # 如果文件不存在，则进行拍照识别等操作  
 treasureXY = []  
 teamColor = 0  
 # findTreasureXY()  
  
 FfindTreasureXY= findTreasureXY0()  
 treasure\_list = FfindTreasureXY[0]  
 teamColor = FfindTreasureXY[1]  
  
 # 根据识别结果生成坐标数据  
 # treasure\_list = ...  
 # teamColor = ...  
 # 保存数据到文件  
 treasure\_data = {  
 'treasure\_list': treasure\_list,  
 'team\_color': teamColor  
 }  
 np.save('treasure\_data.npy', treasure\_data)  
 #print(treasure\_list)  
 #print(teamColor)  
 return treasure\_list,teamColor  
  
  
findTreasureXY()

**代码详解：**

1. **宝藏检测逻辑：**
   * findTreasureXY0()函数包含一个无限循环，用于处理摄像头的视频流。
   * 对每一帧图像，首先将其转换为灰度图像，然后进行Canny边缘检测。
   * 提取图像中的轮廓，并遍历每个轮廓，计算其最小外接矩形。
   * 对于每个矩形，检查其尺寸是否在预设范围内，并判断是否接近正方形。
   * 使用非极大值抑制（NMS）确保不会重复标记相近的矩形。
   * 在满足条件的矩形上绘制绿色矩形框，并记录其位置信息。
2. **宝藏坐标提取：**

def findTreasureXY0():  
 while cap.isOpened():  
 # 逐帧读取视频  
 ret, frame = cap.read()  
  
 if ret:  
 # 在这里对每一帧进行处理  
 # 将图像转换为灰度图像  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 进行边缘检测  
 edges = cv2.Canny(gray, 200, 300)  
  
 # 轮廓提取  
 contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 cornerLocationPoint.clear()  
 # 遍历轮廓  
 for contour in contours:  
 # 计算轮廓的边界框  
 center, WH, \_ = cv2.minAreaRect(contour) # 找最小外接矩形  
 w = WH[0]  
 h = WH[1]  
 if w > 40 or h > 40 or w < 20 or h < 20: # 过大过小直接过滤  
 continue  
  
 # 判断边界框是否接近正方形  
 if abs(w - h) < 5:  
 # 绘制边界框  
 x = center[0]  
 y = center[1]  
  
 # NMS  
 tag = 1  
 for point in cornerLocationPoint:  
 x1 = point[0]  
 y1 = point[1]  
 r = pow(pow(x1 - x, 2) + pow(y1 - y, 2), 0.5)  
 if r < 20:  
 tag = 0  
 break  
 if tag:  
 cv2.rectangle(frame, (int(x - w / 2), int(y - h / 2)), (int(x + w / 2), int(y + h / 2)),  
 (0, 255, 0), 2)  
 cornerLocationPoint.append([x, y, w, h])  
  
 # 显示帧  
 cv2.imshow('Camera', frame)  
 cv2.waitKey(2)  
  
 if len(cornerLocationPoint) == 4:  
 # 排序  
 cornerLocationPoint.sort(key=lambda member: member[0], reverse=False)  
 cache = list()  
 if cornerLocationPoint[0][1] < cornerLocationPoint[1][1]:  
 leftTop = cornerLocationPoint.pop(0)  
 leftBottom = cornerLocationPoint.pop(0)  
 cache.append(leftTop)  
 else:  
 leftTop = cornerLocationPoint.pop(1)  
 leftBottom = cornerLocationPoint.pop(0)  
 cache.append(leftTop)  
 cornerLocationPoint.sort(key=lambda member: member[1], reverse=False)  
 rightTop = cornerLocationPoint.pop(0)  
 rightBottom = cornerLocationPoint.pop(0)  
 cache.append(rightTop)  
 cache.append(leftBottom)  
 cache.append(rightBottom)  
  
 # 左上 右上 左下 右下  
 X = [cache[0][0], cache[1][0], cache[2][0], cache[3][0]]  
 Y = [cache[0][1], cache[1][1], cache[2][1], cache[3][1]]  
  
 # 透视变换  
 pic1 = np.float32([[X[0], Y[0]], [X[1], Y[1]], [X[2], Y[2]], [X[3], Y[3]]])  
 pic2 = np.float32([[0, 0], [540, 0], [0, 540], [540, 540]])  
 Matrix = cv2.getPerspectiveTransform(pic1, pic2)  
 perspectivePic = cv2.warpPerspective(frame, Matrix, (540, 540))  
 # cv2.imshow("PerspectiveTransform", perspectivePic)  
  
 # 抵消平均误差  
 width = (cache[0][2] + cache[1][2] + cache[2][2] + cache[3][2]) / 4  
 height = (cache[0][3] + cache[1][3] + cache[2][3] + cache[3][3]) / 4  
 # 截取迷宫区域  
 cropped\_image = perspectivePic[int(height \* 2):int(540 - height \* 2),  
 int(width \* 2):int(540 - width \* 2)]  
  
 # 将彩色图像转换为灰度图像  
 perspectiveGRAYPic = cv2.cvtColor(cropped\_image, cv2.COLOR\_BGR2GRAY)  
 # 应用阈值处理，将灰度图像二值化为黑白图像  
 \_, binary\_image = cv2.threshold(perspectiveGRAYPic, 100, 255, cv2.THRESH\_BINARY) # 光照调整  
 cv2.imshow("THRESH\_BINARY", binary\_image)  
  
 contours, hierarchy = cv2.findContours(binary\_image, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 treasureXY.clear()  
  
 # 获取图像的宽度和高度  
 w, h = perspectiveGRAYPic.shape  
 # 计算每个小块的宽度和高度  
 unitW = int(w/ 10)  
 unitH = int(h/ 10)  
 perspectiveRGBPic = cv2.cvtColor(perspectiveGRAYPic, cv2.COLOR\_GRAY2RGB)  
 # 使用嵌套循环来遍历图像，并划分为100个小块  
 for i in range(0, h, unitH):  
 for j in range(0, w, unitW):  
 # 提取当前块的左上角和右下角坐标  
 x1, y1 = j, i  
 x2, y2 = j + unitW, i + unitH  
 # 在图像上绘制矩形框  
 cv2.rectangle(perspectiveRGBPic, (x1, y1), (x2, y2), (0, 255, 0), 2) # 这里使用绿色框  
  
 countXY = 0  
  
 for contour in contours:  
 (x, y), radius = cv2.minEnclosingCircle(contour)  
 if 8 < radius < 15:  
 center = (int(x), int(y))  
 radius = int(radius)  
 cv2.circle(perspectiveRGBPic, center, radius, (0, 0, 255), 2)  
 countXY = countXY + 1  
 x = int(x / unitW) \* 2 + 1  
 y = int(y / unitH) \* 2 + 1  
 treasureXY.append([int(y), int(x)])  
 cv2.imshow('DONE', perspectiveRGBPic)  
 if countXY == 8:  
 TC = 0  
 teamImage = perspectivePic[300:550, 0:100]  
 hsv = cv2.cvtColor(teamImage, cv2.COLOR\_BGR2HSV)  
  
 picRedHSV = cv2.inRange(hsv, lowerb=redLowHSV1, upperb=redHighHSV1) + cv2.inRange(hsv,  
 lowerb=redLowHSV2,  
 upperb=redHighHSV2)  
 # 进行边缘检测  
 edgesRed = cv2.Canny(picRedHSV, 200, 300)  
 # 轮廓提取  
 contoursRed, \_ = cv2.findContours(edgesRed, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 外层大色块  
 for contour in contoursRed:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour)  
 w = WH[0]  
 h = WH[1]  
 if h > 20: # 过小直接过滤  
 TC = 1  
  
 picBlueHSV = cv2.inRange(hsv, lowerb=blueLowHSV, upperb=blueHighHSV)  
 # 进行边缘检测  
 edgesBlue = cv2.Canny(picBlueHSV, 200, 300)  
 # 轮廓提取  
 contoursBlue, \_ = cv2.findContours(edgesBlue, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 外层大色块  
 for contour in contoursBlue:  
 # 找最小外接矩形  
 \_, WH, \_ = cv2.minAreaRect(contour)  
 w = WH[0]  
 h = WH[1]  
 if h > 20: # 过小直接过滤  
 TC = 2  
  
 cv2.imshow('teamImage', teamImage)  
 if TC == 0:  
 print("队伍颜色识别失败")  
 if TC == 1:  
 print("队伍颜色为：红")  
 if TC == 2:  
 print("队伍颜色为：蓝")  
  
 print("识别出的宝藏个数：", len(treasureXY))  
 print("识别出的宝藏坐标：", treasureXY)  
 key = cv2.waitKey(0)  
 # 按下 'Q' 键重新识别  
 if key == ord('Q') or key == ord('q'):  
 # 释放资源  
 cv2.destroyAllWindows()  
 continue  
 # 释放资源  
 cap.release()  
 cv2.destroyAllWindows()  
 return treasureXY,TC  
   
  
  
 else:  
 print("摄像头ret有问题")  
 break

* + 当检测到四个角点时，对这些点进行排序，以确定宝藏的相对位置。
  + 使用透视变换将图像校正，然后截取迷宫区域。
  + 对校正后的图像进行二值化处理，提取宝藏的轮廓。
  + 通过轮廓的最小外接圆，确定宝藏的精确坐标。

1. **队伍颜色识别：**
   * 在宝藏坐标检测完成后，对特定区域进行颜色空间转换，识别队伍颜色。
   * 根据颜色识别结果，设置队伍颜色（红色或蓝色）。
2. **数据保存与加载：**
3. def findTreasureXY():  
    if os.path.exists('treasure\_data.npy'):  
    # 如果文件存在，则直接加载数据并打印坐标  
    treasure\_data = np.load('treasure\_data.npy', allow\_pickle=True).item()  
    treasureXY = treasure\_data['treasure\_list']  
    teamColor = treasure\_data['team\_color']  
    print(treasureXY)  
    print(teamColor)  
    cap.release()  
    cv2.destroyAllWindows()  
    return treasureXY,teamColor  
     
     
    else:  
    # 如果文件不存在，则进行拍照识别等操作  
    treasureXY = []  
    teamColor = 0  
    # findTreasureXY()  
     
    FfindTreasureXY= findTreasureXY0()  
    treasure\_list = FfindTreasureXY[0]  
    teamColor = FfindTreasureXY[1]  
     
    # 根据识别结果生成坐标数据  
    # treasure\_list = ...  
    # teamColor = ...  
    # 保存数据到文件  
    treasure\_data = {  
    'treasure\_list': treasure\_list,  
    'team\_color': teamColor  
    }  
    np.save('treasure\_data.npy', treasure\_data)  
    #print(treasure\_list)  
    #print(teamColor)  
    return treasure\_list,teamColor
   * 如果存在之前保存的宝藏数据文件（treasure\_data.npy），则直接加载数据。
   * 如果文件不存在，执行宝藏检测流程，并保存新的宝藏坐标和队伍颜色数据。

**实际应用示例：**

* 在main.py中，findTreasureXY()函数会被调用，用于获取宝藏坐标和队伍颜色。
* 宝藏坐标（treasureXY）和队伍颜色（teamColor）将用于后续的路径规划和宝藏收集。

**注意事项：**

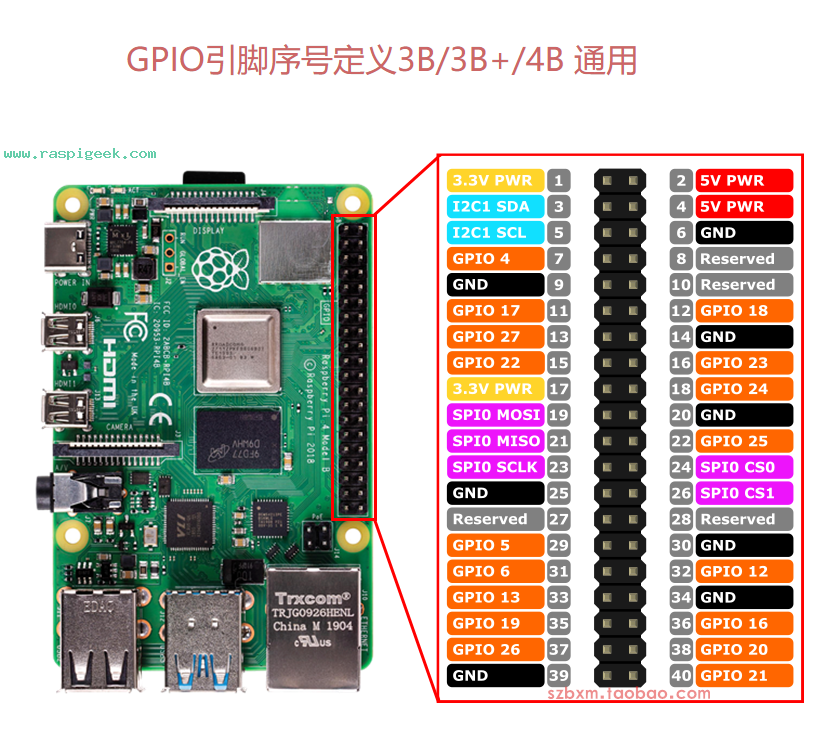
* 宝藏检测的准确性受到摄像头质量和光照条件的影响，可能需要调整颜色阈值和边缘检测参数。
* NMS的参数（如距离阈值）需要根据实际应用场景进行调整，以避免误检或漏检。
* 透视变换和二值化处理的参数（如阈值）可能需要根据迷宫的具体布局进行优化。

FindTreasureXY.py模块通过精确的宝藏坐标检测和队伍颜色识别，为智能小车的导航提供了关键的视觉信息。在实际部署时，可能需要根据迷宫的布局和光照条件进行调整。

**5. 附录**

5.1 参考文献

* 树莓派4B GPIO参考图：



* 树莓派官方网站. 访问于2024年2月26日. 网址：https://www.raspberrypi.org/documentation/gpio/
* OpenCV官方文档. (2023). OpenCV图像处理库. OpenCV官方网站. 访问于2024年2月26日. 网址：[https://docs.opencv.org/master/](https://docs.opencv.org/master/" \t "_blank)
* Python官方文档. (2023). Python编程语言. Python官方网站. 访问于2024年2月26日. 网址：[https://docs.python.org/3/](https://docs.python.org/3/" \t "_blank)的

5.2 经验与杂谈

* 核心思路：能用硬件解决的问题就不要用软件解决，尽量减轻软件压力；
* 关于小车的闭环控制：如果红外循迹不好用，可以考虑使用一对激光雷达或者超声波传感器，测定到左右围墙的距离实现控制；
* 关于小车的电机驱动：目前我们的电机是开环的没有反馈，可以考虑用树莓派直接充当上下位机，用L298N直接连一个12V的直流减速电机，可以编码实现闭环，通过PW波和霍尔编码控制；
* 关于视觉：opencv做灰度化和二值化的时候，可以考虑加黑白滤光片，减少图像处理的压力（相当于用物理方法进行了一个简单的预处理，不过考虑到我们还要进行颜色识别，所以这个还有待商榷）；也可以考虑裁切摄像头范围，减少对算力的要求；可以考虑更换带npu的摄像头，对视频流预处理，目的也是减少对算力的要求；考虑场地环境光线对于图像识别的影响；
* 关于语言：图像处理方面显然python更有优势，但是硬件控制方面c更接近底层，效率会更高，二者可以权衡取舍；
* 其他一些细节：添加散热风扇；各个接口确定不动了之后要上热熔枪打螺丝；要拿理线的扎带理线；场地尽量在类似的胶地上实验，实验室地面太滑没有参考性；考虑场地上环境对于传感器的干扰；增加缓冲泡面或者是下额铁板，增强对抗能力。