

**PYTHON系统功能实现**

姓 名： 何 昂

学 号： 202210310219

班 级： 人工智能221

专 业： 人工智能

学年学期： 2024-2025第一学期

二〇二四 年 10月

**1. 引言**

本报告详细介绍了一个基于 Python 实现的AGV自动导航清舱机器人控制系统软件的设计和实现。该系统由多个模块组成，包括路径规划、卡尔曼滤波、SSH 远程控制、坐标转换、以及实际与预测状态的对比分析。整个系统通过 Tkinter 实现了用户友好的图形界面，支持实时操作与路径展示。项目的核心功能在 robotcontrolmain.py 文件中实现，并通过调用其他辅助模块（ karmanfilter.py、translateenu.py 等）来扩展其功能。本文将对各个模块的技术细节、使用的算法、以及关键代码进行详细的分析和解释。

**2. 系统环境与依赖库**

**2.1 使用的库**

* **tkinter**: Python 标准库中的 GUI 编程库，用于创建系统的图形用户界面（GUI）。本项目中 Tkinter 被广泛用于布局窗口、按钮、输入框等用户界面元素。
* **PIL (Pillow)**: 用于图像处理，主要用于加载、调整大小和展示机器人图像、团队 logo 和软件图标。
* **subprocess**: 提供了一个与外部命令和进程交互的接口，主要用于调用其他 Python 脚本或系统命令，支持 SSH 登录、ENU 坐标转换等功能。
* **heapq**: Python 中的堆队列算法模块，用于实现 A\* 路径规划算法中的优先队列。
* **time**: Python 的时间模块，用于实现机器人路径规划时的延迟效果，模拟机器人实际行进的过程。
* **paramiko**: 提供 SSH 协议的支持，用于远程登录到其他机器执行命令。
* **numpy**: 处理矩阵和数值计算的库，广泛应用于机器人状态的协方差、方差计算和卡尔曼滤波器的实现。
* **matplotlib**: 用于数据的可视化，特别是机器人路径、状态估计与预测结果的可视化展示。
* **scipy**: 提供数学和科学计算工具，特别是在优化算法和卡尔曼滤波等复杂计算中起重要作用。

**2.2 开发环境**

* **Python 版本**：3.8 或更高
* **集成开发环境**：Anaconda + Spyder
* **操作系统**：Windows 10 / Ubuntu 20.04
* **依赖库安装：**

pip install paramiko

pip install numpy

pip install matplotlib

pip install pillow

**3. 模块分析与代码详解**

系统模块化设计，主要功能由多个独立模块负责，每个模块提供不同的功能接口，并通过 robotcontrolmain.py 作为主控程序进行整合。以下将对每个模块的功能和代码进行详细分析。

**3.1 主机器人控制模块 (robotcontrolmain.py)**

robotcontrolmain.py 是该项目的核心，它提供了用户交互界面，允许用户规划机器人的路径，添加障碍物，并展示机器人的行进路线。该模块主要使用了 A\* 算法进行路径规划，并通过画布（Canvas）组件实时展示机器人路径。

**3.1.1 主要功能概述**

* **路径点输入**：通过 GUI 界面，用户可以输入多个路径点，机器人根据这些路径点依次进行路径规划。
* *A 路径规划*\*：利用 A\* 算法在网格中寻找从起点到终点的最优路径，避开障碍物并计算最短路径。
* **障碍物添加**：用户可以通过鼠标点击的方式，在网格上动态添加障碍物，这些障碍物会影响路径的规划结果。
* **实时路径展示**：在界面上以不同颜色的方块标识障碍物、路径点和机器人的行进路径，实时展示规划结果和机器人位置。

**3.1.2 代码详解**

**1. *A 算法的实现*\***

A\* 算法是一种启发式搜索算法，广泛用于路径规划。它结合了从起点到当前节点的实际代价（g 值）和从当前节点到终点的估计代价（h 值），从而找到全局最优路径。

以下是 A\* 算法的核心代码部分：

def astar(start, end, grid):

start\_node = Node(start)

end\_node = Node(end)

open\_list = []

closed\_list = set()

heapq.heappush(open\_list, start\_node)

while open\_list:

current\_node = heapq.heappop(open\_list)

closed\_list.add(current\_node.position)

if current\_node == end\_node:

path = []

while current\_node is not None:

path.append(current\_node.position)

current\_node = current\_node.parent

return path[::-1] # 返回反向路径

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0)]: # 上下左右

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

if node\_position[0] < 0 or node\_position[0] >= len(grid) or node\_position[1] < 0 or node\_position[1] >= len(grid[0]):

continue

if grid[node\_position[0]][node\_position[1]] != 0:

continue

new\_node = Node(node\_position, current\_node)

if new\_node.position in closed\_list:

continue

new\_node.g = current\_node.g + 1

new\_node.h = abs(new\_node.position[0] - end\_node.position[0]) + abs(new\_node.position[1] - end\_node.position[1])

new\_node.f = new\_node.g + new\_node.h

if any(open\_node.position == new\_node.position and open\_node.f <= new\_node.f for open\_node in open\_list):

continue

heapq.heappush(open\_list, new\_node)

return None # 无路径

在这段代码中：

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0)]: # 上下左右

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

if node\_position[0] < 0 or node\_position[0] >= len(grid) or node\_position[1] < 0 or node\_position[1] >= len(grid[0]):

continue

if grid[node\_position[0]][node\_position[1]] != 0:

continue

new\_node = Node(node\_position, current\_node)

if new\_node.position in closed\_list:

continue

new\_node.g = current\_node.g + 1

new\_node.h = abs(new\_node.position[0] - end\_node.position[0]) + abs(new\_node.position[1] - end\_node.position[1])

new\_node.f = new\_node.g + new\_node.h

if any(open\_node.position == new\_node.position and open\_node.f <= new\_node.f for open\_node in open\_list):

continue

heapq.heappush(open\_list, new\_node)

return None # 无路径

* start\_node 和 end\_node 分别表示起点和终点节点。
* open\_list 是优先队列，使用 heapq 实现，优先处理代价最小的节点。
* closed\_list 是一个集合，用于存储已经处理过的节点，避免重复计算。
* A\* 算法通过不断从 open\_list 中弹出代价最小的节点，并检查其周围的邻居节点，直到找到终点。

**2. 路径点输入与障碍物管理**

用户可以通过 GUI 界面输入路径点，并通过点击画布的方式添加障碍物。这些路径点和障碍物将影响机器人的路径规划。

add\_obstacle 函数用于在画布上添加障碍物。用户点击画布时，障碍物的位置被记录到 obstacles 集合中，并在画布上显示为红色方块。

def add\_obstacle(event):

grid\_x = event.x // 5 # 将点击的像素坐标转换为网格坐标

grid\_y = event.y // 5

obstacles.add((grid\_x, grid\_y)) # 将障碍物加入集合

canvas.create\_rectangle(grid\_x \* 5, grid\_y \* 5, grid\_x \* 5 + 5, grid\_y \* 5 + 5, fill="red", outline="black")

**3. 路径实时可视化**

当路径规划完成后，系统会根据规划结果在画布上显示路径点、障碍物和机器人的行进轨迹。

def plan\_path():

total\_path = []

for i in range(len(path\_points) - 1):

start = path\_points[i]

end = path\_points[i + 1]

path = astar(start, end, grid)

if path is None:

messagebox.showerror("错误", f"无法从 {start} 到 {end} 规划路径")

return

total\_path.extend(path)

for (x, y) in total\_path:

canvas.create\_rectangle(x\*5, y\*5, x\*5 + 5, y\*5 + 5, fill="blue", outline="white")

canvas.update()

time.sleep(0.01) # 模拟机器人行进的延迟

在这段代码中：

plan\_path 函数使用 A\* 算法依次规划各个路径点之间的最优路径。

total\_path 存储所有的路径坐标，并在画布上以蓝色方块展示机器人行进的轨迹，time.sleep(0.01) 用于模拟机器人行进的延迟效果。

**3.1.3 图形用户界面（GUI）设计**

该模块的 GUI 设计主要基于 Tkinter 实现，用户可以通过界面轻松地进行路径输入、障碍物添加、路径规划等操作。以下是主要界面元素的设计。

**窗口和按钮设计**

主窗口的标题为 "智航--船舶清舱机器人路径规划系统"，大小为 1000x700 像素。

背景色设置为浅灰色（#f2f2f2），使界面简洁且易于阅读。

界面中提供了路径点输入框和按钮，用户可以通过这些按钮触发相应的功能，如添加路径点、规划路径等。

# 设置窗口图标

root.title("智航--船舶清舱机器人路径规划系统")

root.geometry("1000x700") # 调整主窗口大小

root.configure(bg="#f2f2f2") # 设置背景颜色

# 输入路径点数量

tk.Label(button\_frame, text="输入路径点数量:", font=("Arial", 12), bg="#f2f2f2").pack(side="left", padx=5)

point\_number\_entry = tk.Entry(button\_frame, width=10)

point\_number\_entry.pack(side="left", padx=5)

# 添加路径点按钮

add\_point\_button = tk.Button(button\_frame, text="输入路径点", command=add\_path\_point, bg="#4CAF50", fg="white", font=("Arial", 12, "bold"))

add\_point\_button.pack(side="left", padx=10)

**3.2 卡尔曼滤波与导航仿真 (karmanfilter.py)**

karmanfilter.py 实现了卡尔曼滤波器，用于预测机器人状态并消除传感器噪声。该模块允许用户通过 GUI 调整滤波器的 Q、R 和 P 参数，以优化状态估计。

**3.2.1 卡尔曼滤波器的实现**

卡尔曼滤波器是一种线性二次估计器，用于估计带噪声的系统状态。它通过两个步骤（预测和更新）不断修正对系统状态的估计。

class KalmanFilter:

def \_\_init\_\_(self, A, B, H, Q, R, P, x):

self.A = A # 状态转移矩阵

self.B = B # 控制矩阵

self.H = H # 测量矩阵

self.Q = Q # 过程噪声

self.R = R # 测量噪声

self.P = P # 误差协方差矩阵

self.x = x # 状态估计

def predict(self, u):

self.x = self.A @ self.x + self.B @ u

self.P = self.A @ self.P @ self.A.T + self.Q

return self.x

def update(self, z):

y = z - self.H @ self.x # 测量残差

S = self.H @ self.P @ self.H.T + self.R # 残差协方差

K = self.P @ self.H.T @ np.linalg.inv(S) # 卡尔曼增益

self.x = self.x + K @ y

self.P = (np.eye(len(self.P)) - K @ self.H) @ self.P

return self.x

在这段代码中：

* A 是状态转移矩阵，描述了系统从一个状态到下一个状态的转换规则。
* B 是控制矩阵，描述了外部控制输入对系统状态的影响。
* H 是测量矩阵，用于将系统的状态映射到测量值。
* Q、R 分别表示过程噪声和测量噪声，它们影响系统的状态估计。
* P 是误差协方差矩阵，用于描述估计的不确定性。

卡尔曼滤波的两个主要步骤：

* **预测步骤**：通过状态转移矩阵 A 预测下一个状态，并更新误差协方差 P。

self.x = self.A @ self.x + self.B @ u

self.P = self.A @ self.P @ self.A.T + self.Q

**更新步骤**：根据新的测量数据修正状态估计，使用卡尔曼增益 K 来平衡预测和测量之间的权重。

K = self.P @ self.H.T @ np.linalg.inv(S) # 卡尔曼增益

self.x = self.x + K @ y

**3.2.2 参数调优与仿真**

karmanfilter.py 提供了一个仿真器，模拟 AGV 的运动，并使用卡尔曼滤波对其状态进行估计。用户可以通过界面上的滑块调整 Q、R 和 P 参数，从而优化滤波器性能。

Label(self.root, text="Q 参数").pack()

self.q\_scale = Scale(self.root, from\_=0.001, to=1.0, resolution=0.001, orient='horizontal')

self.q\_scale.pack()

Label(self.root, text="R 参数").pack()

self.r\_scale = Scale(self.root, from\_=0.001, to=1.0, resolution=0.001, orient='horizontal')

self.r\_scale.pack()

Label(self.root, text="P 参数").pack()

self.p\_scale = Scale(self.root, from\_=0.001, to=1.0, resolution=0.001, orient='horizontal')

self.p\_scale.pack()

在界面上，提供了三个滑块，分别用于调整卡尔曼滤波器的 Q、R 和 P 参数。通过这些滑块，用户可以动态调整滤波器的性能，以便找到最佳的状态估计。

**3.3 预测与实际状态对比 (predictandreal.py)**

该模块用于比较机器人在预测状态与实际状态之间的差异，并通过计算协方差矩阵和方差来分析这些差异。

**3.3.1 协方差和方差计算**

协方差矩阵用于衡量两组变量之间的相关性，而方差用于衡量数据的分散程度。通过这些指标，用户可以评估预测值与实际值之间的差异。

# 计算协方差矩阵

covariance\_matrix = np.cov(data.T)

# 计算方差

variance\_pred = np.var(pred\_state)

variance\_real = np.var(real\_state)

**3.3.2 数据可视化**

通过 matplotlib 绘制预测状态和实际状态的散点图，并使用箭头标识两者之间的差异。

plt.scatter([pred\_x], [pred\_y], color='red', label='预测状态')

plt.scatter([real\_x], [real\_y], color='blue', label='实际状态')

plt.quiver(pred\_x, pred\_y, real\_x - pred\_x, real\_y - pred\_y, angles='xy', scale\_units='xy', scale=1, color='green', label='差异')

**3.4 SSH 远程控制 (sshlink.py)**

sshlink.py 模块允许用户通过 SSH 协议远程登录到服务器并执行命令。该模块主要使用 paramiko 库实现 SSH 连接，用户通过 GUI 输入 IP、用户名和密码，建立 SSH 连接并执行远程命令。

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect(hostname=ip, username=username, password=password)

stdin, stdout, stderr = ssh.exec\_command('echo "UTF-8 Check"')

output = stdout.read().decode('utf-8')

在这段代码中：

paramiko.SSHClient() 创建了一个 SSH 客户端对象。

set\_missing\_host\_key\_policy 方法用于自动接受新的主机密钥。

connect 方法使用用户提供的 IP 地址、用户名和密码进行 SSH 登录。

exec\_command 方法执行一条简单的命令并返回结果。

**3.5 ENU 坐标转换 (translateenu.py)**

translateenu.py 模块用于将实际坐标转换为 ENU（East-North-Up）坐标系统。该模块允许用户输入 X、Y、Z 坐标和方位角，通过三角函数进行坐标转换，并在 GUI 中显示转换结果。

east = x \* cos(angle\_rad) - y \* sin(angle\_rad)

north = x \* sin(angle\_rad) + y \* cos(angle\_rad)

up = z

这段代码实现了将实际坐标转换为 ENU 坐标的核心逻辑。方位角的输入由度转换为弧度，再通过正余弦函数进行坐标转换。

**4. 用户界面设计分析**

整个系统的用户界面设计非常简洁直观，所有功能都可以通过按钮、输入框和滑块进行交互。主要的界面元素包括：

**画布（Canvas）**：用于展示路径规划和机器人行进过程。

**文本输入框和按钮**：用于输入路径点、添加障碍物、规划路径等。

**滑块**：用于动态调整卡尔曼滤波器参数。

**4.1 界面布局**

窗口布局采用水平和垂直的布局方式，按钮、输入框、滑块等控件按功能模块分组，使用户可以清晰直观地操作各个功能。

**5 实际演示与功能介绍**

图 1 软件页面演示

root = tk.Tk()

root.title("智航--船舶清舱机器人路径规划系统")

root.geometry("1000x700") # 设置主窗口大小

root.configure(bg="#f2f2f2") # 设置背景颜色

共有六个功能按钮，软件布局左边是机器人图像，中间是路径规划左边是，右边是路径展示。

**5.1路径点与路径规划**

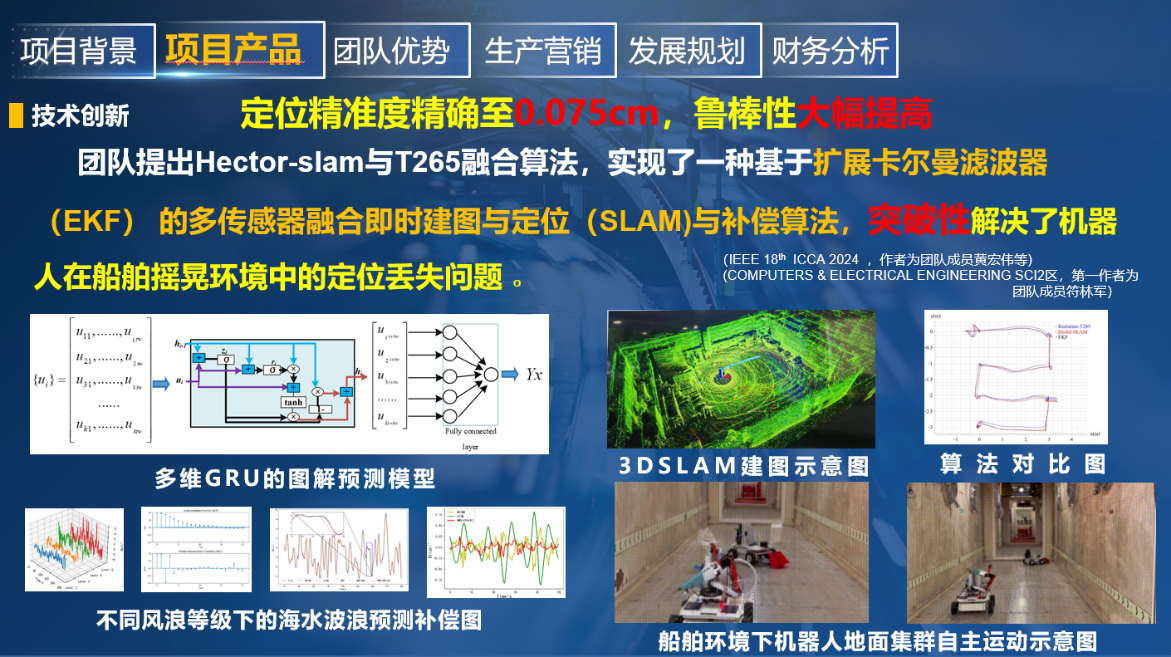
**要**理解软件的用处首先要了解路径规划对于机器人自主导航的重要性。

图 2 SLAM算法与机器人自主导航

在以SLAM算法自主导航的AGV移动机器人中，定点路径规划通常需要耗费时间，由于是2D算法，我们设计的软件中的路径规划框和ROS(机器人操作系统)中的RVIZ仿真的栅格地图类似，但本软件忽略了一些因素，如通常用2d雷达建图需要边缘点与激光，耗费大量资源，我们可以在软件的路径规划中放入障碍物来模拟环境，如下图：

图 3 添加障碍物演示

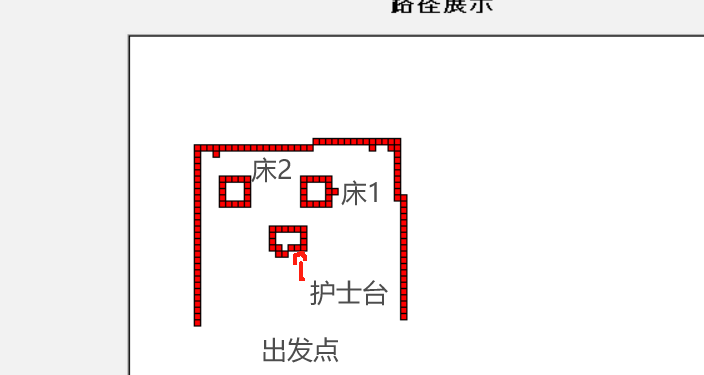
 如某些医疗环境下可作简单的仿真：

图 4 现实环境与模拟

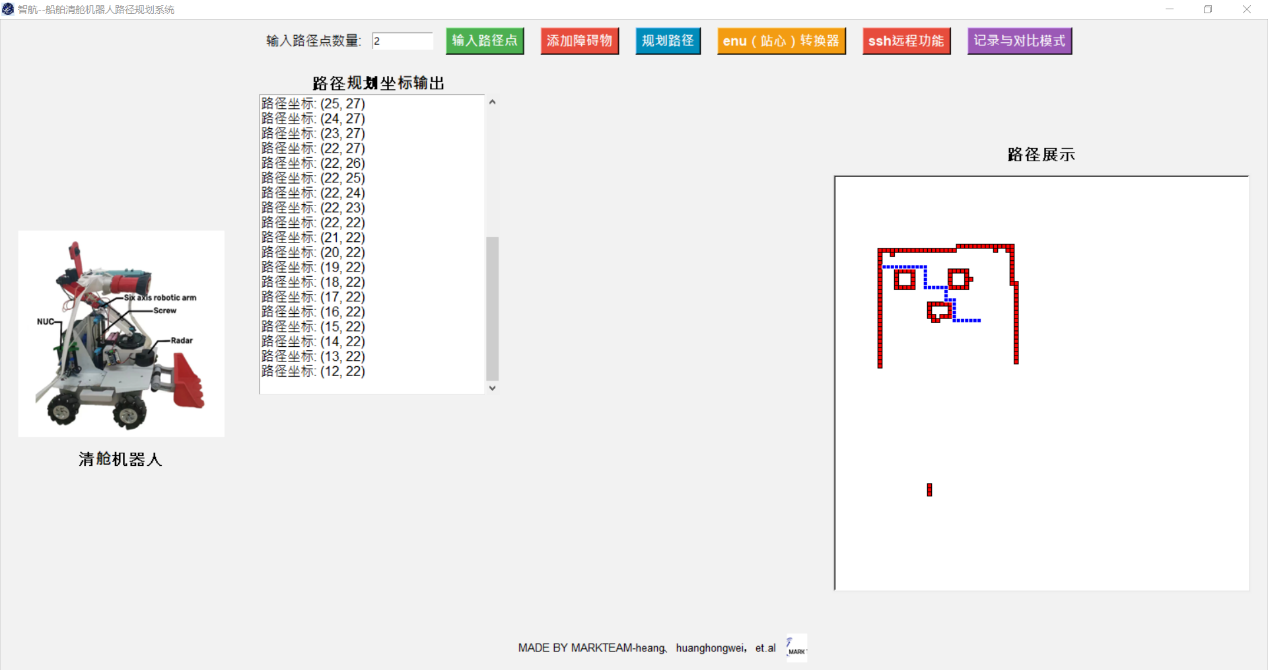
通过软件内置的A\*算法，我们可以得到两点之间基于启发式搜索的最短路径（多点也适用）：

图 5 路径规划演示图

同时中间的路径规划及坐标输出会根据time.sleep(0.01)来实时输出当前路径坐标，也可以根据机器人具体运动轨迹进行调整。

**5.2enu(站心)转换器**

坐标系统转换在地理信息系统（GIS）、导航和无人机（UAV）领域中扮演着至关重要的角色。这种转换不仅对于数据的准确性和一致性至关重要，而且对于跨平台和系统的数据共享与整合也至关重要。下面我们将详细探讨NED到ENU坐标转换的原理，并提供一个扩写的Python代码示例，最后将通过应用场景和案例分析的扩写来展示其重要性。

一、NED到ENU坐标转换原理

（1） 坐标系统概述

- NED坐标系统：这个系统以地理北方为基准，定义了三个相互垂直的坐标轴。北（N）轴指向地球的北极点方向，东（E）轴指向地球的东方向，而下（D）轴则垂直向下，指向地球的质心。在NED系统中，高度通常以负值来表示，这意味着地面上方的点具有正的高度值，而地面下方的点则具有负的高度值。

- ENU坐标系统：与NED系统类似，ENU系统也是一个三维地理坐标系统，但它将垂直轴的方向定义为向上，即远离地球表面。东（E）轴和北（N）轴的方向与NED系统保持一致，但上（U）轴是指向天空的。这种表示方法更符合日常习惯，因为人们通常将向上视为正方向。

（2）转换机制

NED到ENU的坐标转换是一个简单的过程，主要包括以下步骤：

1. 坐标交换：首先，将NED坐标中的北（N）和东（E）分量进行交换。这样做是为了匹配ENU系统中东轴和北轴的顺序。

2. 坐标值取反：接下来，将NED坐标中的下（D）分量取相反数。由于NED系统中的下是负值，而ENU系统中的上是正值，这个步骤确保了高度值的正确表示。

这个过程可以用更直观的方式理解：想象一个坐标系中的点，我们只需将这个点在水平面上顺时针旋转90度，并将垂直方向反转，就可以完成从NED到ENU的转换。

**在工程任务中，我们常常需要将设定的现实中坐标转换成ENU坐标来使得机器人了解自身方位（以hector-slam为例，机器内置坐标系为enu坐标系），这通常是比较复杂且浪费时间的**，通过我们软件内置的程序，通过按钮连接，我们**可以轻松将人类理解的坐标转化为机器人理解的坐标，从而更好的操控机器人：**

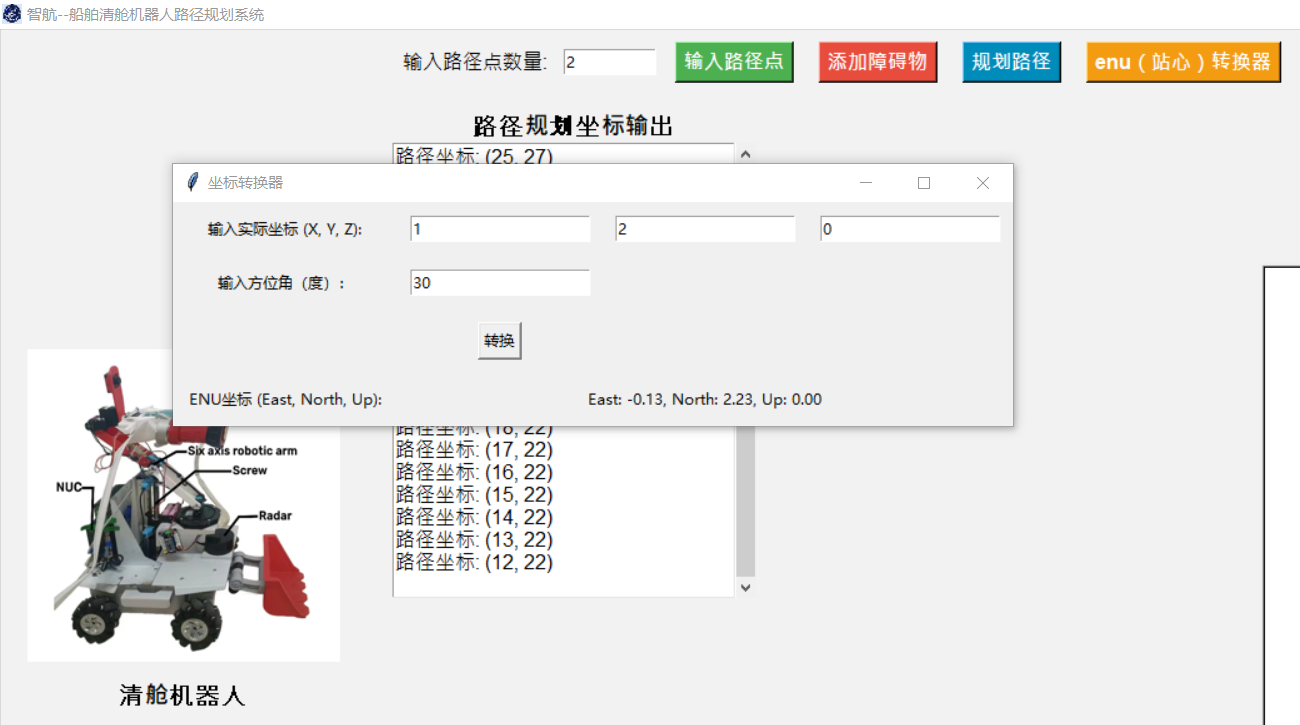
****

图 6 路径规划演示图

**5.3ssh功能**

SSH（Secure Shell）是一种用于安全远程登录和通信的协议，主要功能是在不安全的网络上为两个设备之间提供安全的加密通信。在主机（例如你的电脑）与机器人主控板（例如树莓派、Jetson、Arduino等带有网络连接的嵌入式设备）之间，SSH 的主要作用包括以下几个方面：

**1. 远程访问与控制**

使用 SSH，你可以从主机远程登录到机器人主控板，不需要直接物理连接就能访问和控制主控板。这意味着你可以远程执行命令、修改配置文件、启动或停止程序，而无需在物理上接触机器人。

**2. 安全通信**

SSH 提供了加密通道，确保在网络上传输的数据不会被窃听或篡改。在机器人与主机之间交换敏感信息时（如控制指令或数据流），SSH 保证了通信的安全性。

**3. 文件传输**

使用 SCP（Secure Copy Protocol）或 SFTP（SSH File Transfer Protocol），你可以通过 SSH 安全地在主机和机器人主控板之间传输文件。比如，你可以将程序或配置文件从主机上传到机器人主控板，也可以从主控板下载数据进行分析。

**4. 自动化与调试**

在开发或维护机器人时，SSH 可以帮助你进行远程调试和自动化操作。例如，你可以通过 SSH 脚本化地批量更新机器人系统或运行测试程序，而无需手动操作每个机器人主控板。

**因此ssh功能在远程操控机器人（实时收集数据、遥控控制等功能）具有很大的作用，vscode本身就提供ssh功能，我们就将ssh功能集成到软件中，不需要单独开启如tera tarm等其他支持机器人ssh软件，更为便捷。**

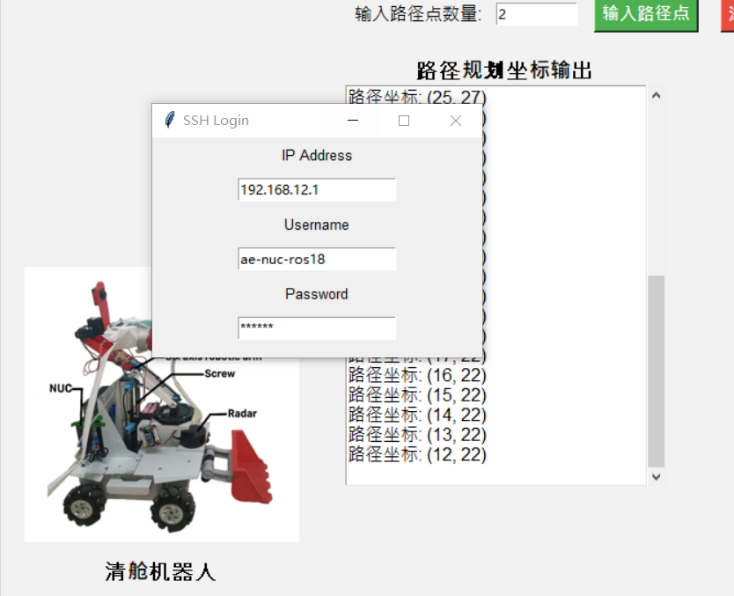


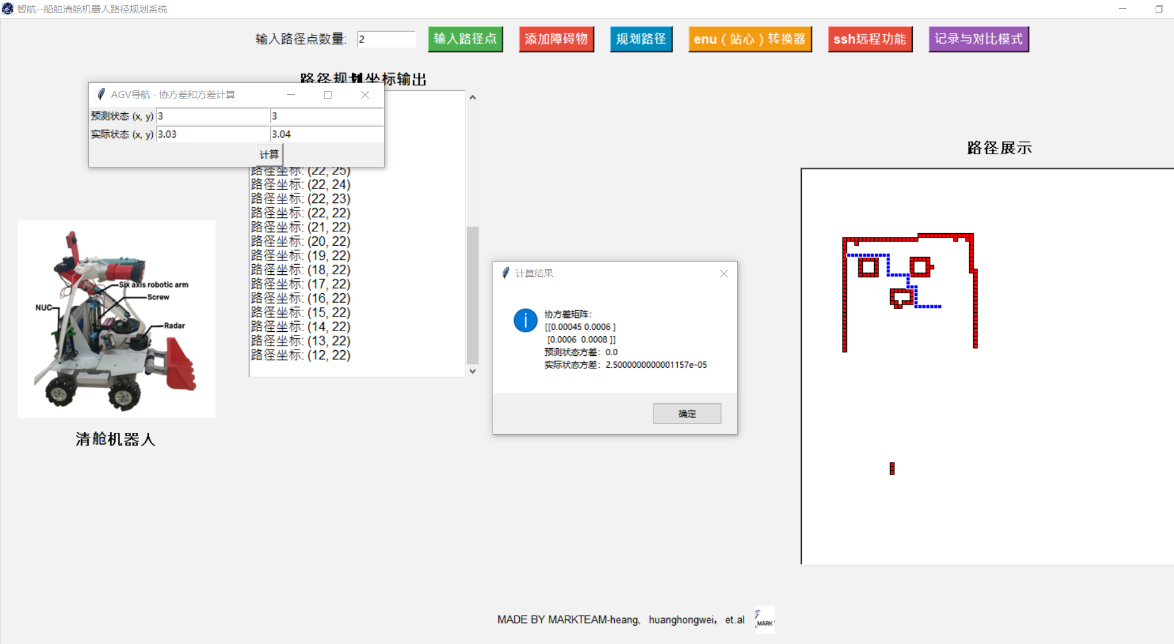
图 7 SSH功能演示图

**5.4记录与对比功能**

 在工程项目或者科研项目中，收集实验数据是做实验的重要一环，我们机器人在设计之初，并不具备实验数据的收集功能（机器人实际到达坐标与设定坐标）。

图 8 数据收集与计算

但是ROS(机器人操作系统)中rostopic功能可通过ssh实时print出来rx（实际X坐标），ry（实际Y坐标），sx（设定X坐标），sy（设定Y坐标）等参数。

因此，在软件中，我们设置了一个记录与对比模式用于记录上面这几个参数，同时计算协方差等一系列参数，可供数据记录或者现实环境参数（卡尔曼滤波等等）调优。

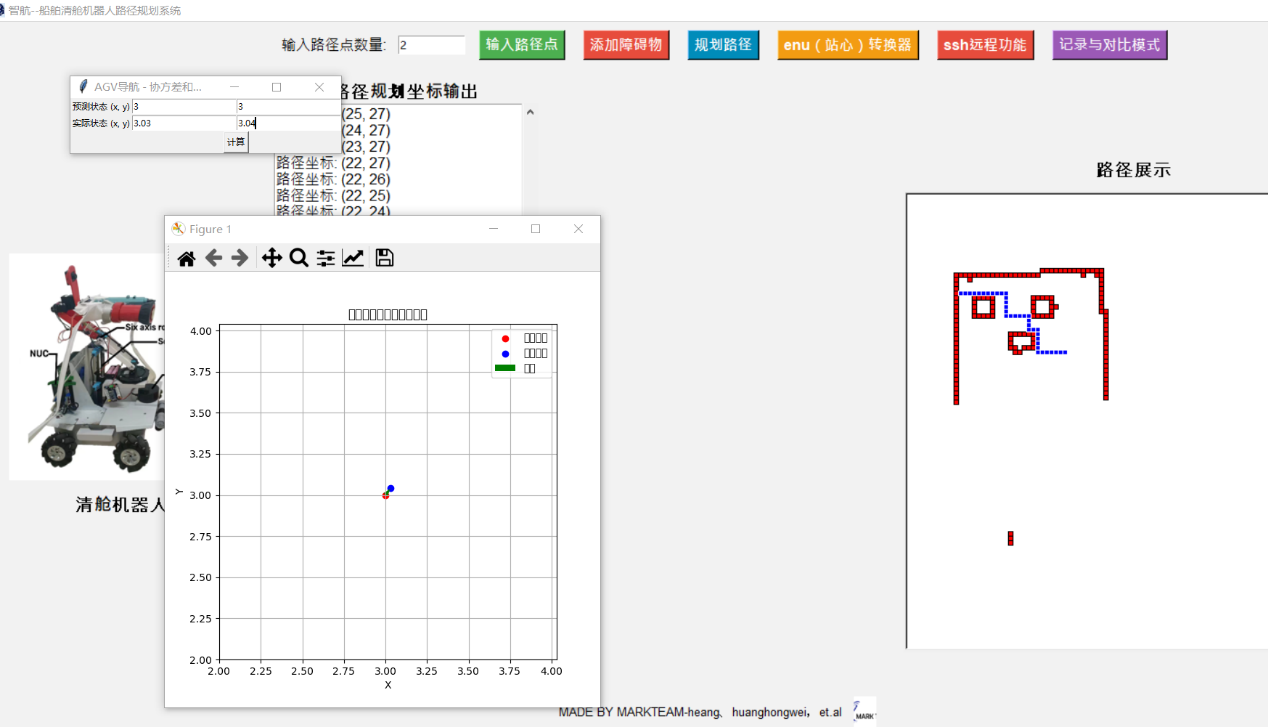
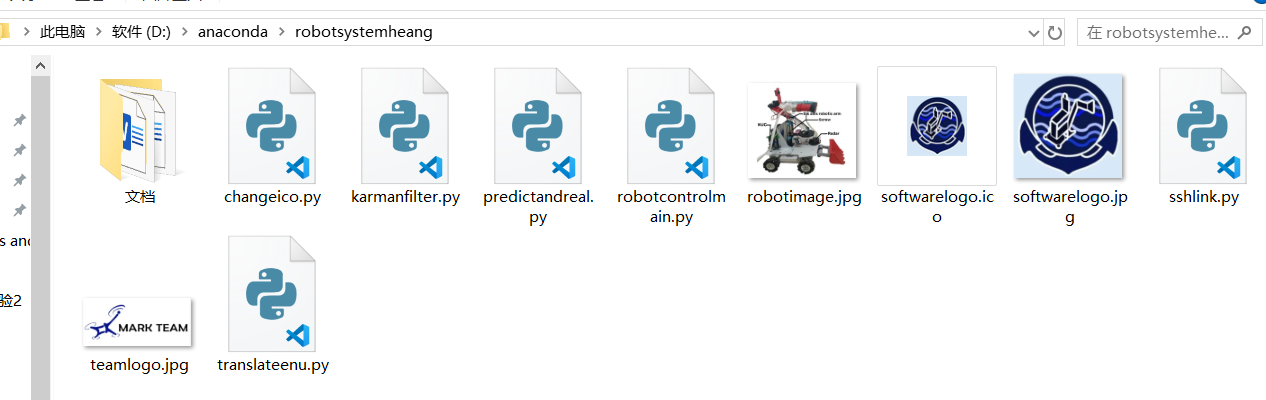
图 9 记录与对比

图 10 坐标对比

****

**源代码清单：**

**Robotcontrolmain.py:**

# -\*- coding: utf-8 -\*-

import tkinter as tk

from tkinter import messagebox, scrolledtext

from PIL import Image, ImageTk

import subprocess

import heapq

import time

# 机器人默认状态

robot\_status = {'电量': 80, '状态': '开'}

# 用于存储路径点和障碍物信息

path\_points = []

obstacles = set() # 障碍物集合

# A\* 算法的节点类

class Node:

def \_\_init\_\_(self, position, parent=None):

self.position = position

self.parent = parent

self.g = 0 # 起点到当前节点的代价

self.h = 0 # 当前节点到终点的估计代价

self.f = 0 # f = g + h

def \_\_eq\_\_(self, other):

return self.position == other.position

def \_\_lt\_\_(self, other):

return self.f < other.f

# A\*算法

def astar(start, end, grid):

start\_node = Node(start)

end\_node = Node(end)

open\_list = []

closed\_list = set()

heapq.heappush(open\_list, start\_node)

while open\_list:

current\_node = heapq.heappop(open\_list)

closed\_list.add(current\_node.position)

# 找到目标点

if current\_node == end\_node:

path = []

while current\_node is not None:

path.append(current\_node.position)

current\_node = current\_node.parent

return path[::-1] # 返回反向路径

# 生成邻居节点

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0)]: # 上下左右

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

# 确保新的位置在网格范围内

if node\_position[0] < 0 or node\_position[0] >= len(grid) or node\_position[1] < 0 or node\_position[1] >= len(grid[0]):

continue

# 确保不是障碍物

if grid[node\_position[0]][node\_position[1]] != 0:

continue

new\_node = Node(node\_position, current\_node)

if new\_node.position in closed\_list:

continue

new\_node.g = current\_node.g + 1

new\_node.h = abs(new\_node.position[0] - end\_node.position[0]) + abs(new\_node.position[1] - end\_node.position[1])

new\_node.f = new\_node.g + new\_node.h

# 如果新节点在 open\_list 中，且当前 f 值大于已存在的 f 值，则跳过

if any(open\_node.position == new\_node.position and open\_node.f <= new\_node.f for open\_node in open\_list):

continue

heapq.heappush(open\_list, new\_node)

return None # 无路径

# 创建并显示每个路径点的速度和期望坐标

def add\_path\_point():

point\_num = point\_number\_entry.get()

try:

point\_num = int(point\_num)

if point\_num <= 0:

raise ValueError

except ValueError:

messagebox.showwarning("输入错误", "请输入有效的路径点数量")

return

# 创建新窗口输入每个路径点的速度和期望坐标

point\_window = tk.Toplevel(root)

point\_window.title(f"输入路径点信息 (共{point\_num}个点)")

entries = []

for i in range(point\_num):

tk.Label(point\_window, text=f"路径点 {i+1}:").grid(row=i, column=0)

tk.Label(point\_window, text="X坐标:").grid(row=i, column=1)

x\_coord\_entry = tk.Entry(point\_window)

x\_coord\_entry.grid(row=i, column=2)

tk.Label(point\_window, text="Y坐标:").grid(row=i, column=3)

y\_coord\_entry = tk.Entry(point\_window)

y\_coord\_entry.grid(row=i, column=4)

entries.append((x\_coord\_entry, y\_coord\_entry))

def save\_points():

global path\_points

path\_points.clear()

for i, entry\_tuple in enumerate(entries):

try:

x\_coord = int(entry\_tuple[0].get())

y\_coord = int(entry\_tuple[1].get())

path\_points.append((x\_coord, y\_coord))

except ValueError:

messagebox.showwarning("输入错误", f"路径点 {i+1} 的数据无效，请输入有效的数值")

return

messagebox.showinfo("路径规划", f"共{len(path\_points)}个路径点已成功添加")

point\_window.destroy()

save\_button = tk.Button(point\_window, text="保存", command=save\_points)

save\_button.grid(row=point\_num, columnspan=5, pady=10)

# 添加障碍物功能

def add\_obstacle\_mode():

messagebox.showinfo("添加障碍物", "点击画布来添加障碍物（红色方块）")

canvas.bind("<Button-1>", add\_obstacle)

# 在画布中添加障碍物

def add\_obstacle(event):

grid\_x = event.x // 5 # 将点击的像素坐标转换为网格坐标

grid\_y = event.y // 5

obstacles.add((grid\_x, grid\_y)) # 将障碍物加入集合

canvas.create\_rectangle(grid\_x \* 5, grid\_y \* 5, grid\_x \* 5 + 5, grid\_y \* 5 + 5, fill="red", outline="black")

# 规划路径并显示结果

def plan\_path():

if len(path\_points) < 2:

messagebox.showwarning("路径规划", "请先输入至少两个路径点")

return

# 清空输出框中的内容

output\_text.delete(1.0, tk.END)

# 创建 100x100 的网格，0 表示可以通过，1 表示障碍物

grid = [[0 for \_ in range(100)] for \_ in range(100)]

# 设置障碍物

for (x, y) in obstacles:

grid[x][y] = 1

total\_path = []

for i in range(len(path\_points) - 1):

start = path\_points[i]

end = path\_points[i + 1]

path = astar(start, end, grid)

if path is None:

messagebox.showerror("错误", f"无法从 {start} 到 {end} 规划路径")

return

total\_path.extend(path)

# 动态展示机器人路径

canvas.delete("all") # 清空之前的内容

# 先绘制障碍物为红色

for (x, y) in obstacles:

canvas.create\_rectangle(x\*5, y\*5, x\*5 + 5, y\*5 + 5, fill="red", outline="black") # 障碍物为红色

# 先绘制路径点为黄色

for (x, y) in path\_points:

canvas.create\_rectangle(x\*5, y\*5, x\*5 + 5, y\*5 + 5, fill="yellow", outline="black") # 路径点为黄色

# 然后绘制路径为蓝色，并在实时输出路径坐标

for (x, y) in total\_path:

# 绘制蓝色路径

canvas.create\_rectangle(x\*5, y\*5, x\*5 + 5, y\*5 + 5, fill="blue", outline="white")

canvas.update()

# 输出坐标到文本框

output\_text.insert(tk.END, f"路径坐标: ({x}, {y})\n")

output\_text.see(tk.END) # 自动滚动到最新行

time.sleep(0.01) # 模拟机器人行进的延迟

messagebox.showinfo("路径规划完成", "机器人已成功规划并展示路径")

# 打开 enu（站心）转换器

def open\_translate\_enu():

try:

subprocess.Popen(["python", r"D:\anaconda\robotsystemheang\translateenu.py"])

except Exception as e:

messagebox.showerror("错误", f"无法打开 enu 转换器: {str(e)}")

# 打开 ssh 远程功能

def open\_ssh\_remote():

try:

subprocess.Popen(["python", r"D:\anaconda\robotsystemheang\sshlink.py"])

except Exception as e:

messagebox.showerror("错误", f"无法执行 SSH 远程功能: {str(e)}")

def open\_predictandreal():

try:

subprocess.Popen(["python", r"D:\anaconda\robotsystemheang\predictandreal.py"])

except Exception as e:

messagebox.showerror("错误", f"无法执行预测和实际对比功能: {str(e)}")

# 创建主窗口

root = tk.Tk()

root.title("智航--船舶清舱机器人路径规划系统")

root.geometry("1000x700") # 调整主窗口大小

# 背景颜色设置

root.configure(bg="#f2f2f2")

# 设置窗口图标

try:

icon\_img = Image.open(r"D:\anaconda\robotsystemheang\softwarelogo.jpg")

icon\_img\_tk = ImageTk.PhotoImage(icon\_img)

root.iconphoto(True, icon\_img\_tk)

except Exception as e:

messagebox.showerror("图标加载失败", f"无法加载图标: {str(e)}")

# 图片加载与展示

image\_frame = tk.Frame(root, bg="#f2f2f2") # 用于放置图片和文字的框架

image\_frame.pack(side="left", padx=20, pady=20)

try:

img = Image.open(r"D:\anaconda\robotsystemheang\robotimage.jpg")

img = img.resize((250, 250)) # 调整图片大小

img\_tk = ImageTk.PhotoImage(img)

image\_label = tk.Label(image\_frame, image=img\_tk, bg="#f2f2f2")

image\_label.pack(pady=5)

# 添加图片下方的说明文字

caption\_label = tk.Label(image\_frame, text="清舱机器人", font=("Arial", 14, "bold"), bg="#f2f2f2")

caption\_label.pack(pady=5)

except Exception as e:

messagebox.showerror("图片加载失败", f"无法加载图片: {str(e)}")

# 创建上方按钮容器

button\_frame = tk.Frame(root, bg="#f2f2f2")

button\_frame.pack(side="top", fill="x", padx=20, pady=10)

# 输入路径点数量

tk.Label(button\_frame, text="输入路径点数量:", font=("Arial", 12), bg="#f2f2f2").pack(side="left", padx=5)

point\_number\_entry = tk.Entry(button\_frame, width=10)

point\_number\_entry.pack(side="left", padx=5)

# 添加路径点按钮

add\_point\_button = tk.Button(button\_frame, text="输入路径点", command=add\_path\_point, bg="#4CAF50", fg="white", font=("Arial", 12, "bold"))

add\_point\_button.pack(side="left", padx=10)

# 添加障碍物按钮

add\_obstacle\_button = tk.Button(button\_frame, text="添加障碍物", command=add\_obstacle\_mode, bg="#e74c3c", fg="white", font=("Arial", 12, "bold"))

add\_obstacle\_button.pack(side="left", padx=10)

# 规划路径按钮

plan\_button = tk.Button(button\_frame, text="规划路径", command=plan\_path, bg="#008CBA", fg="white", font=("Arial", 12, "bold"))

plan\_button.pack(side="left", padx=10)

# 添加 enu（站心）转换器按钮

translate\_enu\_button = tk.Button(button\_frame, text="enu（站心）转换器", command=open\_translate\_enu, bg="#f39c12", fg="white", font=("Arial", 12, "bold"))

translate\_enu\_button.pack(side="left", padx=10)

# 添加 ssh 远程功能按钮

ssh\_button = tk.Button(button\_frame, text="ssh远程功能", command=open\_ssh\_remote, bg="#e74c3c", fg="white", font=("Arial", 12, "bold"))

ssh\_button.pack(side="left", padx=10)

# 添加记录与对比模式按钮

predict\_button = tk.Button(button\_frame, text="记录与对比模式", command=open\_predictandreal, bg="#9b59b6", fg="white", font=("Arial", 12, "bold"))

predict\_button.pack(side="left", padx=10)

# 在中间添加文本框用于实时输出坐标

output\_frame = tk.Frame(root, bg="#f2f2f2")

output\_frame.pack(side="left", fill="y", padx=20, pady=10)

output\_label = tk.Label(output\_frame, text="路径规划坐标输出", font=("Arial", 14, "bold"), bg="#f2f2f2")

output\_label.pack()

output\_text = scrolledtext.ScrolledText(output\_frame, width=30, height=20, font=("Arial", 12), bg="white", wrap=tk.WORD)

output\_text.pack()

# 创建画布用于展示路径，调整为适应 100x100 个 5x5 大小的方格

canvas\_frame = tk.Frame(root, bg="#f2f2f2")

canvas\_frame.pack(side="right", padx=20, pady=20)

canvas\_label = tk.Label(canvas\_frame, text="路径展示", font=("Arial", 14, "bold"), bg="#f2f2f2")

canvas\_label.pack()

canvas = tk.Canvas(canvas\_frame, width=500, height=500, bg="white", bd=2, relief="sunken")

canvas.pack(pady=10)

# 创建底部的版权标志和团队 logo

bottom\_frame = tk.Frame(root, bg="#f2f2f2")

bottom\_frame.pack(side="bottom", fill="x", pady=20)

# 版权标志文本

copyright\_label = tk.Label(bottom\_frame, text="MADE BY MARKTEAM-heang、huanghongwei，et.al", font=("Arial", 10), bg="#f2f2f2")

copyright\_label.pack(side="left")

# 团队 logo 图片

try:

team\_logo\_img = Image.open(r"D:\anaconda\robotsystemheang\teamlogo.jpg")

team\_logo\_img = team\_logo\_img.resize((60, 35)) # 调整图片大小

team\_logo\_img\_tk = ImageTk.PhotoImage(team\_logo\_img)

team\_logo\_label = tk.Label(bottom\_frame, image=team\_logo\_img\_tk, bg="#f2f2f2")

team\_logo\_label.pack(side="right", padx=10)

except Exception as e:

messagebox.showerror("图片加载失败", f"无法加载团队 logo 图片: {str(e)}")

# 运行主循环

root.mainloop()

**predictandreal.py:**

import tkinter as tk

from tkinter import messagebox

import numpy as np

import matplotlib.pyplot as plt

# 创建主窗口

root = tk.Tk()

root.title("AGV导航 - 协方差和方差计算")

# 输入框标签和输入框

tk.Label(root, text="预测状态 (x, y)").grid(row=0, column=0)

pred\_x\_entry = tk.Entry(root)

pred\_x\_entry.grid(row=0, column=1)

pred\_y\_entry = tk.Entry(root)

pred\_y\_entry.grid(row=0, column=2)

tk.Label(root, text="实际状态 (x, y)").grid(row=1, column=0)

real\_x\_entry = tk.Entry(root)

real\_x\_entry.grid(row=1, column=1)

real\_y\_entry = tk.Entry(root)

real\_y\_entry.grid(row=1, column=2)

# 计算协方差和方差的函数

def calculate():

    try:

        # 获取输入的预测状态和实际状态

        pred\_x = float(pred\_x\_entry.get())

        pred\_y = float(pred\_y\_entry.get())

        real\_x = float(real\_x\_entry.get())

        real\_y = float(real\_y\_entry.get())

        # 创建数据矩阵

        pred\_state = np.array([pred\_x, pred\_y])

        real\_state = np.array([real\_x, real\_y])

        data = np.array([pred\_state, real\_state])

        # 计算协方差矩阵

        covariance\_matrix = np.cov(data.T)

        # 计算方差

        variance\_pred = np.var(pred\_state)

        variance\_real = np.var(real\_state)

        # 显示结果

        result\_text = f"协方差矩阵：\n{covariance\_matrix}\n"

        result\_text += f"预测状态方差：{variance\_pred}\n"

        result\_text += f"实际状态方差：{variance\_real}\n"

        messagebox.showinfo("计算结果", result\_text)

        # 画图

        plt.figure(figsize=(6,6))

        plt.scatter([pred\_x], [pred\_y], color='red', label='预测状态')

        plt.scatter([real\_x], [real\_y], color='blue', label='实际状态')

        plt.quiver(pred\_x, pred\_y, real\_x - pred\_x, real\_y - pred\_y, angles='xy', scale\_units='xy', scale=1, color='green', label='差异')

        plt.xlim(min(pred\_x, real\_x)-1, max(pred\_x, real\_x)+1)

        plt.ylim(min(pred\_y, real\_y)-1, max(pred\_y, real\_y)+1)

        plt.xlabel("X")

        plt.ylabel("Y")

        plt.title("预测状态与实际状态对比")

        plt.legend()

        plt.grid(True)

        plt.show()

    except ValueError:

        messagebox.showerror("输入错误", "请输入有效的数值")

# 计算按钮

calc\_button = tk.Button(root, text="计算", command=calculate)

calc\_button.grid(row=2, column=1, columnspan=2)

# 运行主窗口循环

root.mainloop()

**sshlink.py：**

# -\*- coding: utf-8 -\*-

import paramiko

import tkinter as tk

from tkinter import messagebox

# 创建主窗口

root = tk.Tk()

root.title("SSH Login")

# 设置窗口大小

root.geometry("300x200")

# 定义登录逻辑

def ssh\_login():

ip = entry\_ip.get()

username = entry\_username.get()

password = entry\_password.get()

try:

# 创建 SSH 客户端对象

ssh = paramiko.SSHClient()

# 自动添加主机密钥

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

# 连接到远程服务器

ssh.connect(hostname=ip, username=username, password=password)

# 执行简单的命令以验证连接是否成功

stdin, stdout, stderr = ssh.exec\_command('echo "UTF-8 Check"')

output = stdout.read().decode('utf-8')

# 显示成功消息

messagebox.showinfo("Success", f"SSH Login Successful\n{output}")

# 关闭连接

ssh.close()

except Exception as e:

# 显示错误消息，确保错误信息以 UTF-8 编码显示

messagebox.showerror("Error", f"SSH Login Failed: {str(e)}")

# 创建 GUI 界面组件

label\_ip = tk.Label(root, text="IP Address", font=("Arial", 10))

label\_ip.pack(pady=5)

entry\_ip = tk.Entry(root)

entry\_ip.pack(pady=5)

label\_username = tk.Label(root, text="Username", font=("Arial", 10))

label\_username.pack(pady=5)

entry\_username = tk.Entry(root)

entry\_username.pack(pady=5)

label\_password = tk.Label(root, text="Password", font=("Arial", 10))

label\_password.pack(pady=5)

entry\_password = tk.Entry(root, show="\*")

entry\_password.pack(pady=5)

# 创建登录按钮

login\_button = tk.Button(root, text="Login", font=("Arial", 10), command=ssh\_login)

login\_button.pack(pady=20)

# 启动主循环

root.mainloop()

**translateenu.py：**

# -\*- coding: utf-8 -\*-

import tkinter as tk

from math import sin, cos, radians

class CoordinateConverterApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("坐标转换器")

        # 标签和输入框

        self.label1 = tk.Label(root, text="输入实际坐标 (X, Y, Z):")

        self.label1.grid(row=0, column=0, padx=10, pady=10)

        self.x\_entry = tk.Entry(root)

        self.x\_entry.grid(row=0, column=1, padx=10, pady=10)

        self.y\_entry = tk.Entry(root)

        self.y\_entry.grid(row=0, column=2, padx=10, pady=10)

        self.z\_entry = tk.Entry(root)

        self.z\_entry.grid(row=0, column=3, padx=10, pady=10)

        self.label2 = tk.Label(root, text="输入方位角（度）：")

        self.label2.grid(row=1, column=0, padx=10, pady=10)

        self.angle\_entry = tk.Entry(root)

        self.angle\_entry.grid(row=1, column=1, padx=10, pady=10)

        # 转换按钮

        self.convert\_button = tk.Button(root, text="转换", command=self.convert\_coordinates)

        self.convert\_button.grid(row=2, column=1, padx=10, pady=10)

        # 结果显示

        self.result\_label = tk.Label(root, text="ENU坐标 (East, North, Up):")

        self.result\_label.grid(row=3, column=0, padx=10, pady=10)

        self.enu\_label = tk.Label(root, text="")

        self.enu\_label.grid(row=3, column=1, columnspan=3, padx=10, pady=10)

    def convert\_coordinates(self):

        try:

            # 获取输入值

            x = float(self.x\_entry.get())

            y = float(self.y\_entry.get())

            z = float(self.z\_entry.get())

            angle\_deg = float(self.angle\_entry.get())

            # 将角度转换为弧度

            angle\_rad = radians(angle\_deg)

            # 计算ENU坐标 (East, North, Up)

            east = x \* cos(angle\_rad) - y \* sin(angle\_rad)

            north = x \* sin(angle\_rad) + y \* cos(angle\_rad)

            up = z  # Z坐标保持不变

            # 显示结果

            self.enu\_label.config(text=f"East: {east:.2f}, North: {north:.2f}, Up: {up:.2f}")

        except ValueError:

            self.enu\_label.config(text="输入无效，请检查输入的数字")

# 创建主窗口

root = tk.Tk()

app = CoordinateConverterApp(root)

root.mainloop()

**karmanfilter.py：**

# -\*- coding: utf-8 -\*-

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import minimize

from matplotlib.animation import FuncAnimation

import tkinter as tk

from tkinter import Scale, Label

class KalmanFilter:

    def \_\_init\_\_(self, A, B, H, Q, R, P, x):

        self.A = A  # 状态转移矩阵

        self.B = B  # 控制矩阵

        self.H = H  # 测量矩阵

        self.Q = Q  # 过程噪声

        self.R = R  # 测量噪声

        self.P = P  # 误差协方差矩阵

        self.x = x  # 状态估计

    def predict(self, u):

        self.x = self.A @ self.x + self.B @ u

        self.P = self.A @ self.P @ self.A.T + self.Q

        return self.x

    def update(self, z):

        y = z - self.H @ self.x  # 测量残差

        S = self.H @ self.P @ self.H.T + self.R  # 残差协方差

        K = self.P @ self.H.T @ np.linalg.inv(S)  # 卡尔曼增益

        self.x = self.x + K @ y

        self.P = (np.eye(len(self.P)) - K @ self.H) @ self.P

        return self.x

class AGVSimulator:

    def \_\_init\_\_(self, x0, y0, theta0, v, w):

        self.x = x0

        self.y = y0

        self.theta = theta0

        self.v = v  # 线速度

        self.w = w  # 角速度

    def move(self, dt):

        self.x += self.v \* np.cos(self.theta) \* dt

        self.y += self.v \* np.sin(self.theta) \* dt

        self.theta += self.w \* dt

        return np.array([self.x, self.y, self.theta])

class AGVApp:

    def \_\_init\_\_(self):

        self.root = tk.Tk()

        self.root.title('AGV导航参数调优')

        # 卡尔曼滤波参数调优滑块

        Label(self.root, text="Q 参数").pack()

        self.q\_scale = Scale(self.root, from\_=0.001, to=1.0, resolution=0.001, orient='horizontal')

        self.q\_scale.pack()

        Label(self.root, text="R 参数").pack()

        self.r\_scale = Scale(self.root, from\_=0.001, to=1.0, resolution=0.001, orient='horizontal')

        self.r\_scale.pack()

        Label(self.root, text="P 参数").pack()

        self.p\_scale = Scale(self.root, from\_=0.001, to=1.0, resolution=0.001, orient='horizontal')

        self.p\_scale.pack()

        self.fig, self.ax = plt.subplots()

        self.agv\_sim = AGVSimulator(0, 0, 0, 1, 0.1)  # 初始位置和运动

        self.kalman\_filter = KalmanFilter(np.eye(3), np.eye(3), np.eye(3), 0.01\*np.eye(3), 0.1\*np.eye(3), np.eye(3), np.array([0, 0, 0]))

        self.line, = self.ax.plot([], [], 'bo-', lw=2)

        self.ani = FuncAnimation(self.fig, self.update\_plot, init\_func=self.init\_plot, interval=100)

        plt.show()

    def init\_plot(self):

        self.ax.set\_xlim(-10, 10)

        self.ax.set\_ylim(-10, 10)

        return self.line,

    def update\_plot(self, frame):

        # 获取滑块参数

        Q\_value = self.q\_scale.get()

        R\_value = self.r\_scale.get()

        P\_value = self.p\_scale.get()

        # 更新卡尔曼滤波参数

        self.kalman\_filter.Q = Q\_value \* np.eye(3)

        self.kalman\_filter.R = R\_value \* np.eye(3)

        self.kalman\_filter.P = P\_value \* np.eye(3)

        # AGV真实运动

        real\_state = self.agv\_sim.move(0.1)

        # 假设测量数据有一些噪声

        noisy\_measurement = real\_state + np.random.normal(0, 0.1, 3)

        # 卡尔曼滤波预测和更新

        self.kalman\_filter.predict(np.zeros(3))

        estimated\_state = self.kalman\_filter.update(noisy\_measurement)

        # 更新图像显示

        self.line.set\_data([real\_state[0], estimated\_state[0]], [real\_state[1], estimated\_state[1]])

        return self.line,

    def run(self):

        self.root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

    app = AGVApp()

    app.run()