**手势识别控制ABB机械臂设计思路**

项目目标：通过手势来远程控制机械臂。

学校隶属：西交利物浦&利物浦大学

项目成员：郭家骏（组长），刘成航, 李羽尧

指导老师：张权，JOSEPH CAMM

1. **项目概述**

**1.1概述**

本项目展示了一个完整的从手势识别到机械臂控制的闭环系统，通过集成计算机视觉mediapipe、网络通信和机器人控制，实现了通过手势远程控制机械臂的位置移动。该项目使用了VScode和RobotStudio平台该系统主要由两个模块组成：Python模块用于手势识别和网络通信，RAPID模块用于机械臂的实际控制。整个系统的设计模块化，且具有较高的扩展性，可以扩展手势控制的更多的机械臂完成更多动作。该系统适应不同种类的手势识别和更加复杂的控制逻辑。

**1.2流程**

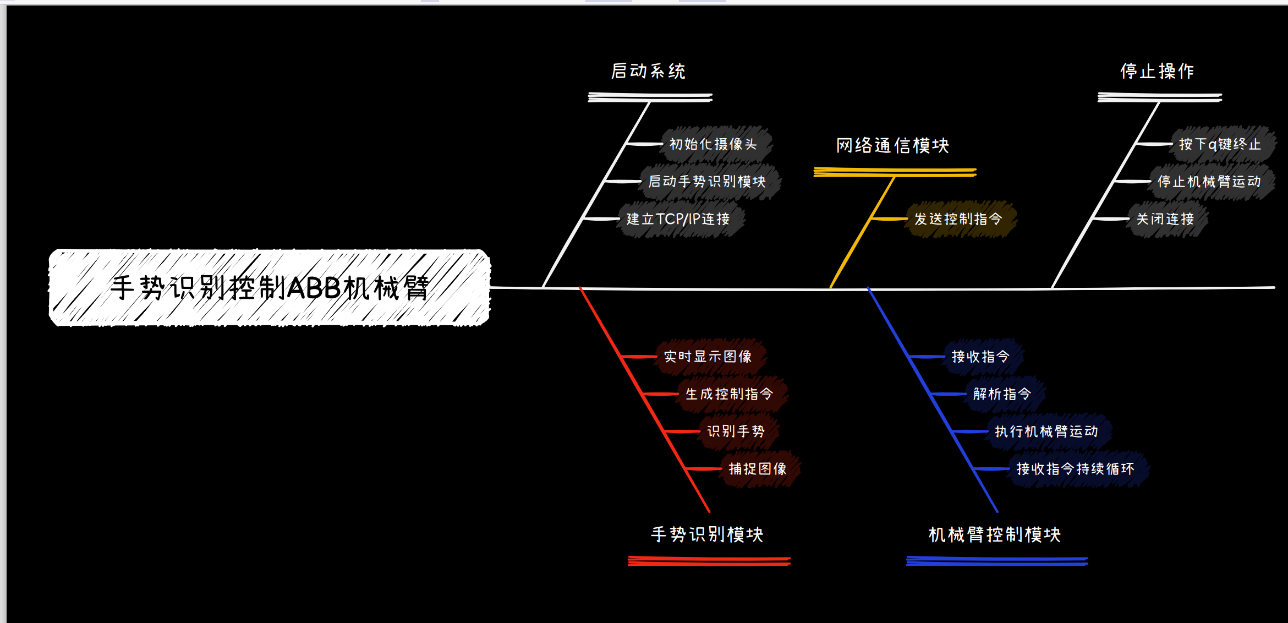
**1.2.1启动系统：**系统初始化，包括摄像头和手势识别模块的启动，并建立与机械臂控制模块的TCP/IP连接。

**1.2.2手势识别模块：**通过摄像头实时捕捉图像，识别手势动作，生成控制指令，并在屏幕上实时显示图像。

**1.2.3网络通信模块：**将生成的控制指令通过网络发送给机械臂控制模块。

**1.2.4机械臂控制模块：**接收并解析指令，然后执行相应的机械臂运动，持续接收和执行指令，直到系统终止。

停止操作：当用户按下“q”键或指令停止时，停止机械臂运动并关闭系统连接。



**2. 系统架构**

**2.1 总体架构**

该系统由以下三个主要部分组成：

* **手势识别模块（Python）：** 通过摄像头捕捉实时图像，并使用MediaPipe库识别手势动作。
* **网络通信模块（Python）：** 负责通过TCP/IP协议将手势识别结果发送给机械臂控制模块。
* **机械臂控制模块（RAPID）：** 接收并解析手势识别结果，控制机械臂的运动。

**2.2 模块间交互**

* Python模块负责通过摄像头捕捉手势图像，并进行实时处理，识别出特定手势动作后，通过TCP连接将动作指令发送给运行RAPID代码的机械臂控制模块。
* RAPID模块接收来自Python模块的指令，解析后控制机械臂按指令执行指定的动作。

**3. 模块设计**

**3.1 手势识别模块（Python）**

* **图像捕捉：** 使用cv2.VideoCapture函数捕捉摄像头的实时图像流。
* **手势识别：** 利用MediaPipe库进行手势识别，通过捕捉手部关键点，识别出预定义的特定手势。
* **指令生成：** 根据识别到的手势生成相应的机械臂移动指令
* **网络通信：** 通过TCP/IP协议将生成的指令发送到机械臂控制模块，使用本地回环地址和端口5000进行通信。
* **界面显示：** 使用cv2.imshow实时显示处理后的图像，并设置按键操作以终止程序运行。

**3.2 网络通信模块（Python）**

* **TCP连接：** 与机械臂控制模块建立TCP连接，通过指定IP地址和端口进行通信。
* **指令发送：** 每当识别出新的手势动作后，立即通过TCP将生成的指令发送到机械臂控制模块。
* **异常处理：** 监控网络连接状态，如连接失败或中断，系统能够及时捕捉并处理异常。

**3.3 机械臂控制模块（RAPID）**

* **模块结构**：使用RAPID语言编写的模块主要用于ABB机械臂的控制，包括创建TCP/IP服务器、接收来自Python客户端的手势指令、解析指令并控制机械臂的运动。
* **指令接收**：模块首先通过SocketCreate、SocketBind、SocketListen等函数创建和绑定一个TCP/IP服务器，然后通过SocketAccept接受来自Python客户端的连接。
* **数据解析**：接收到指令后，通过SocketReceive接收原始数据，并使用UnpackRawBytes将数据解码为字符串格式的控制指令。
* **机械臂控制**：根据接收到的控制指令，通过修改currentPos（表示当前机械臂的位置），并使用MoveL指令实现机械臂的线性插补运动。具体指令如下：
  + **STOP**：停止机械臂的当前动作。
  + **X\_MOVE,10,0,0**：沿X轴正方向移动10单位。
  + **X\_MOVE,-10,0,0**：沿X轴负方向移动10单位。
  + **Y\_MOVE,0,10,0**：沿Y轴正方向移动10单位。
  + **Y\_MOVE,0,-10,0**：沿Y轴负方向移动10单位。
  + **Z\_MOVE,0,0,10**：沿Z轴正方向移动10单位。
  + **Z\_MOVE,0,0,-10**：沿Z轴负方向移动10单位。
* **循环执行**：整个过程在一个无限循环WHILE TRUE DO中不断执行，确保系统可以持续接收和响应来自Python客户端的控制指令，直到系统被手动终止。

**4.难点与解决方案**

**4.1 RAPID编程语言资源匮乏**

问题描述：RAPID编程语言的学习资源相对有限，导致在编写和调试代码时遇到困难，尤其是在控制机械臂的复杂操作中。

解决方法：通过查阅官方文档、利用RobotStudio提供的示例项目以及不断实验，掌握了RAPID语言的用法和特性，确保了代码的正确性。

**4.2 RobotStudio与RobotWare版本兼容性问题**

问题描述：在安装RobotWare时，由于版本不匹配，RobotStudio无法正常运行，影响了仿真和控制的正常进行。

解决方法：通过检查版本兼容性，并确保RobotStudio与RobotWare的版本匹配，成功解决了兼容性问题，使得软件能够正常工作。

**4.3通信问题**

问题描述：初期在RAPID代码中建立TCP/IP通信连接时，频繁出现错误，导致无法与Python模块正常通信。

具体问题及解决方法：

**4.3.1 更改Virtual Controller的Option**：通过在Virtual Controller中选择616-1 PC Interface选项，解决了部分通信设置问题。

**4.3.2 更换代码格式**：**由于使用Open MySocket \IP:="127.0.0.1", \Port:=5000, \IODevice:="TCPIP1";**

格式的RAPID1代码报错，改为RAPID2代码格式，并成功建立了稳定的通信连接。

**4.3.3 设置TCP/IP传输协议**：在RobotStudio的Controller tab > Configuration > Communication > Transmission Protocol中，将传输协议设置为TCP/IP，最终成功实现了与Python模块的网络通信。

**4.4 数据解码问题**

**问题描述：在使用UnpackRawBytes data,1,signalData,\ASCII:=2;解码**

从Python模块接收的数据时，遇到了数据处理不正确的问题。

解决方法：通过查阅相关文档、观看教程视频，并进行多次调试和测试，最终找到了合适的解码方式，确保了数据的正确解析。

**4.5 机械臂运行速度和移动单位调整**

问题描述：最初的机械臂运行速度和移动单位不合适，导致操作不够精确。

解决方法：通过多次测试和调整，优化了机械臂的运行速度和每个单位的移动距离，使机械臂能够按预期精确响应手势控制指令。

**4.6 系统响应延迟和CPU负载问题**

问题描述：系统在执行复杂操作时出现响应延迟，且编程点较密集，导致系统对CPU的负载较高。此外，服务器性能不足，无法满足系统需求。

解决方法：分析系统的计中进行手势检测，并通过cv2.imshow实时显示检测结果，同时通过按键操作（按下算需求，调整代码优化点，以减少CPU负载。后期寻找性能更高的服务器，提升系统整体性能，确保在高负载情况下仍能顺畅运行。

**5总结**

本系统通过整合计算机视觉、网络通信和机器人控制，实现了通过手势远程控制机械臂的功能。系统由两个主要模块组成：Python模块负责手势识别和网络通信，RAPID模块负责机械臂的实际控制。这种设计方式简洁高效，具有较高的扩展性，能够适应不同种类的手势识别和复杂的控制逻辑。

Python模块通过摄像头实时捕捉手部图像，并利用MediaPipe库识别特定的手势动作。每次识别到手势后，Python模块生成相应的控制指令，并通过TCP/IP协议将指令发送至机械臂控制器（RAPID模块）。该模块的核心是在捕获每一帧视频流的过程 “q”键）退出程序循环。

RAPID模块用于控制ABB机械臂，接收并解析来自Python客户端的控制指令，然后根据指令更新机械臂的位置信息。例如，接收到X\_MOVE,10,0,0指令时，机械臂会沿X轴正方向移动10个单位。RAPID模块通过线性插补运动（MoveL指令）来确保机械臂以指定速度（v1000）和精度（z10）移动到目标位置。整个操作过程在一个无限循环中持续进行，确保系统可以不断接收并响应来自Python客户端的控制指令，直到系统被手动终止。

此系统展示了从手势识别到机械臂控制的完整闭环和系统的高效性与易扩展性，适用于更复杂的控制任务和多样的手势识别场景。