

2024 年英特尔杯大学生电子设计竞赛嵌入式系统专题邀请赛

2024 Intel Cup Undergraduate Electronic Design Contest

- Embedded System Design Invitational Contest

作品设计报告

Final Report



Intel Cup Embedded System Design Contest

报告题目：沉浸式智能仪器控制系统

学生姓名：宋羽 段宇乐 肖舒铭

指导教师：庄杰

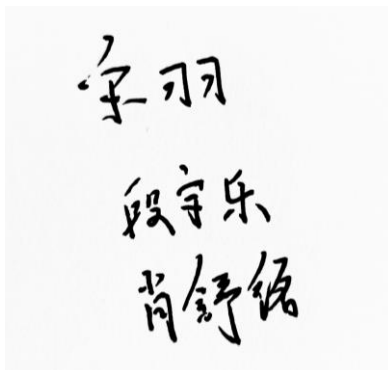
参赛学校：电子科技大学

2024 年英特尔杯大学生电子设计竞赛嵌入式系统专题邀请赛

参赛作品原创性声明

本人郑重声明：所呈交的参赛作品报告，是本人和队友独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果，不侵犯任何第三方的知识产权或其他权利。本人完全意识到本声明的法律结果由本人承担。

参赛队员签名：



余羽
段宇乐
肖舒铭

日期：2024 年 6 月 29 日

沉浸式智能仪器控制系统

摘要

如今嵌入式芯片系统设计开发领域正迅猛发展，电子工程师对于系统自动化测试、提高生产力和测试测量效率的需求迅速增加。为解决电子工程师对于系统和芯片自动测试、仪器自动化运作需求，本项目将借助 Intel 强大的算力和丰富的接口，以哪吒开发套件为中心设备，配合其他通用 VISA 接口的仪器设备，以语音交互为操作方式，利用 Agent 将工程师请求解析成系统标准命令流，实现控制各类仪器；同时利用手势识别代替传统机械按键，实现仪器的“去按键化”，提升操作的便利性和灵活性。该项目提供一站式的系统和芯片调试测量、仪器使用帮助，极大的简化了仪器操作方式以适应不同的测试需求，为工程师提供及时和准确的测试结果。

关键词：仪器控制，数据分析，语音交互，手势识别，大语言模型，开放平台

Immersive Intelligent Instrument Control System

ABSTRACT

Nowadays, the field of embedded chip system design and development is developing rapidly, and electronic engineers' needs for system automation testing, improvement of productivity and test and measurement efficiency are rapidly increasing. This project is dedicated to solving the needs of electronic engineers for system automatic testing and providing comprehensive intelligent and automated testing solutions. With the Nezha Dev.Kit as the central device and other common VISA interface instruments, through voice interaction, the agent is used to parse the engineer's request into the system standard command stream to control various instruments; at the same time, gesture recognition is used to replace traditional mechanical buttons to achieve the "key-free" instrument and fine-tune instrument parameters. The project provides one-stop system and chip debugging measurement, instrument use assistance, greatly simplifies the instrument operation mode to adapt to different test needs, and provides engineers with timely and accurate test results.

Key words: pen Platform, Instrument Control, Speech Interaction, Micromotion Recognition, Large Language Model

目 录

第一章	项目背景	1
1.1	立项背景和意义	1
1.2	国内外发展现状	1
1.3	项目特点及优势	2
第二章	项目概述	3
2.1	研究开发内容	3
2.2	系统工作流程	3
2.3	系统架构	3
2.3.1	哪吒 (Nezha) 开发套件	4
2.3.2	信号发生器	4
2.3.3	示波器	5
2.3.4	直流可编程电源	5
2.3.5	麦克风	6
2.3.6	摄像头	6
2.3.7	自制待测模块。	6
2.4	软件流程	6
第三章	关键技术和特色	8
3.1	基于 Agent 体系的 IVI 命令流	8
3.1.1	Agent 的概念	8
3.1.2	大语言模型	8
3.1.3	规划模块	9
3.1.4	记忆模块	9
3.1.5	工具模块	9
3.1.6	Agent 整个系统	10
3.2	智能语音识别	10
3.2.1	语音识别 ASR	11
3.2.2	语音抗干扰能力	12
3.2.3	热词功能	12
3.2.4	LLM 语义修正	12
3.3	手势识别	13
3.4	进程交互协同	14
第四章	系统测试	16
4.1	硬件测试	16
4.2	功能测试	16
4.2.1	Agent 体系规划	16
4.2.2	语音识别	16
4.2.3	手势识别	17
第五章	附录	18

5.1	程序清单	18
5.1.1	基于 Agent 体系的 IVI 命令流	18
5.1.2	智能语音识别	18
5.1.3	手势识别	20
5.2	程序源码	20
5.2.1	语音识别客户端	20
5.2.2	语音识别服务端	23
5.2.3	ReAct 规划部分	24
5.2.4	记忆模块部分	26
5.2.5	工具链	28
5.2.6	语音识别服务端	34
5.2.7	语音服务客户端	36
5.2.8	旋转手势识别	39
	参考文献	46

表目录

表 1 国内外测试设备一览.....	2
表 2 哪吒（Nezha）开发套件性能参数	4
表 3 LNI-T UTG4202A 参数	5
表 4 LNI-T MSO3504CS-S 参数	5
表 5 RIGOL DP832 参数.....	5
表 6 硬件测试结果.....	16
表 6 系统运行测试结果.....	16
表 7 语音识别测试结果.....	17
表 8 手势识别效果测试.....	17

图目录

图 1 硬件架构	4
图 2 摄像头畸变示意图.....	6
图 3 软件主程序运行流程.....	6
图 4 软件子程序运行流程.....	7
图 5 Agent 系统组成	8
图 6 不同策略的规划模块.....	9
图 7 Agent 工作流程图	10
图 8 Agent 控制仪器流程	10
图 9 Paraformer 模型.....	11
图 10 LLM 辅助语音识别	12
图 11 手势调节流程.....	13
图 12 旋转角度检测示意.....	14
图 13 旋转手势识别.....	14
图 14 子程序交互逻辑.....	15

第一章 项目背景

1.1 立项背景和意义

在当今，嵌入式芯片系统设计开发领域正以惊人的速度迅猛发展。据预测，从 2021 年到 2025 年，我国嵌入式系统市场规模的年复合增长率将达到 21%，智能硬件系列更是呈现出 25% 的增长趋势。在这一背景下，电子工程师对于系统自动化测试、提高生产力和测试测量效率的需求迅速增加、选型和参数调整愈加复杂，但常用仪器软件的迭代更新速度缓慢，计算和存储性能不足，功能单一集约，难以实现自动化测量，个人在系统开发当中进行迅捷测试的需求重要性不断凸显。部分个人开发者对于仪器高级使用仍未完全熟悉，有时难以捕获重要信息波形，造成信息延误。

随着科技的发展，诸如 GPT3.5、文心一言、ChatGLM、Google Bard 等通用大模型的涌现，以及新发展的 AI Agent 技术，已经为解决这一问题提供了新的可能性。这些通用大模型拥有庞大的知识储备，能够从自主决策走向自主行动，丰富了应用场景，提升了用户体验。自 AutoGPT 火爆以来，AI Agent 热度一路飙涨。落地应用离不开 Agent，开会活动聚焦 Agent，创业项目也在走向 Agent。

回顾近几年来智能设备的设计发展，其实体按键和接口逐渐减少，在整体尺寸大小不变的情况下，通过将机械按键改为虚拟按键，尽可能增加屏幕屏占比以致力于增加信息量。然而，使用传统的 QWERTY 键盘或是触摸屏界面，都将不可避免地导致设备磨损。此外，触摸屏界面在潮湿的环境中、用户戴着手套或者很难接触控制面板的情况下都不能使用，为提升用户操作的便利性和灵活性，基于手势识别的虚拟按键应运而生。动作识别技术在控制这方面展现出了广阔的应用前景。

在本次项目中，我们将借助 Intel 强大的算力和丰富的接口，一方面让 Intel AI Box 能够运行通用大模型，让非智能仪器接入模型，实现智能测量。使个人能够轻松获得科学合理完整统一的测试测量报告，降低了对于芯片、系统测试的使用门槛，另一方面，本项目致力于仪器“去按钮化”，通过手势识别来实现无接触控制仪器，进一步提升用户操作的便利性和灵活性。同时，本项目能够在任意仪器间轻松放置和迁移，降低仪器使用门槛，实现了从“学习仪器”到“仪器主动展示其功能”的转变。本项目将为新手入门到老手调试提供一站式的系统和芯片调试测量、仪器使用帮助或简化操作服务。

1.2 国内外发展现状

国内外对于芯片或系统测试的发展，主要集中于两个类别当中，一是：集中于生产环境当中对于芯片出厂时测试的专业设备，二是：个人使用针对小器件的测试设备。这两种设备中，专业设备主要的服务场景并非以开发为核心的工程师，同时价格高、难以获取、需要专人操作，而且专业设备自身整合了部分实验室设备，与嵌入式工程师现有设备功能存在重合；但专业性操作较多、个人工程师学习曲线陡峭。体积较大、功能受限，并非适用于实验室的设备。同时存在售价高昂、调整测试项目不便等问题。

而个人使用的设备，存在算力低、功能有限、功能分散的特点。在构建一套完整的测试体系时，除了接入整个电路之外，还需要进行手工的反复调整、机械劳动才能获得较为规整的数据。而这些调节过程，都可以通过一套完整的远程协议（如 VISA）进行控制。

表 1 国内外测试设备一览

国家	公司	种类	产品	特点	缺点
美国	Agilent Technologies	专业测试设备	Agilent 3070 Series 5 In-Circuit Test System	高度自动化、精准测试、广泛适用于各类芯片生产环境	价格昂贵、使用门槛较高、需要专业维护人员
日本	Advantest Corporation	大型测试设备	Advantest T2000 Series Test System	多功能、高速度、适用于各种器件测试	设备体积较大、维护成本较高、对操作人员技术要求较高
美国	Saleae	便携逻辑分析仪	Saleae Logic Analyzer	高性价比、易于使用、适用于数字信号分析	适用范围较窄、无法进行模拟信号测试、性能相对较低
中国	FNIRSI	便携示波器兼简易电桥	数字示波器晶体管测试仪 LCR 表三合一 DSO-TC3	高性价比、易于使用、适用于简单直插器件分析	可分辨器件种类少、无法进行复杂人物测试、性能极
中国	RIGOL	台式示波器	DS1102Z-E	主流实验室采用、性价比高	功能相对简单

除此之外，目前的手势控制仅能实现比较简单的功能，即便如此，在真正的用户体验过程中还是会出现一些问题。不管是采用 ToF、结构光还是毫米波雷达识别手势，都要求手势指令必须在特定的区域内操作，所有手势还必须要符合系统对动作的标准要求；现有的识别多为单次按键类操作，对于连续的操作控制还处于发展阶段。

1.3 项目特点及优势

(1) 通用性强：与上述示波器等个人设备相比，我们的项目着重于解决系统和芯片自动测试、仪器自动化运作的需求，而不仅仅局限于单一测试功能。我们的系统提供全面的智能且自动化测试解决方案，还涵盖了信号生成、数据采集、分析和报告等方面的功能。提供更加综合和全面的测试、内容讲解支持。

(2) 场景灵活：与专业设备相比，我们的设备功能可以随着连接的设备不同而达到不同的连接形态。自主生成解决方案，采用通用的测试方法和硬件设备，以中心 + 分布的形式，不仅能够提供全流程的决策支持，还能够适应不同的测试需求，自主生成解决方案实现高效的数据提取和分析，为用户提供更加及时和准确的测试结果。通过语音识别和手势识别的方式，提升操作的便利性和灵活性。

(3) 价格优势：我们的项目采用通用的算法和硬件设备，降低用户终端设备成本，能够连接现有设备转化为测试节点和智能测试设备，无需另购置测试仪器，降低了系统总造价。这在市场推广和服务方面具有更好的竞争力。目前大模型加持下的嵌入式 Agent 设备，还处于“蓝海”阶段。

第二章 项目概述

2.1 研究开发内容

本项目聚焦于开发一套用于协助电子工程师解决系统和芯片自动测试、仪器自动化运作的需求的智能辅助系统。系统整体主要有 4 个模块组成：一是语音识别模块。该模块部署 AI 处理算法实现语言识别，进行语音到自然语言转换和语义完善修正；二是 Agent 智能体系模块。该模块进行语义提炼和测试方案的生成决策，进而进行仪器控制标准命令流生成；三是手势识别模块。该模块利用手势识别代替传统机械按键来控制仪器，实现仪器的去“按键化”，提升操作的便利性和灵活性。四是仪器控制模块。该模块执行 Agent 生成的标准命令流，实现控制各类仪器。通过标准接口进行仪器设备的操作，输出激励信号到待测系统，并读取输出信号进一步分析，以及进行最终测试报告的书写。

工程师只需连接系统，提出需求并进行手势操作，即可立即获得测试结果。这一简化流程大大提高了交互性，让工程师享受到智能调试带来的沉浸式体验，进而有利于提高工作效率。整个交互过程更贴近人类的自然行为，实现了更智能、更自然、更高效的用户体验。

2.2 系统工作流程

系统工作时，工程师使用语音输入，在说出唤醒词前，系统进入低功耗阻塞等待；确认唤醒后开始语音识别进行内容提取，确认后再进行以大模型+Agent 生成测试方案并；在主程序中，不断检查解决方案进程是否存在，保证服务质量。通过“无需必要，勿增实体”的设计方式，增加系统鲁棒性和当前执行任务时的性能。也方便构建易用、简洁的用户界面。

同时工程师可以下达调节仪器参数命令，系统将识别工程师的手势动作对仪器进行细微调节，方便细致的观察实际波形和调节参数。当整个测试方案生成后，Agent 将采用完整的工具链来输出控制仪器的标准命令流，实时追踪方案实施情况并输出。当整个测试结束后，系统整合测试数据生成测试报告以便工程师调取或阅读。

2.3 系统架构

本系统实现上需要使用的硬件设备主要为：

- (1) 哪吒（Nezha）开发套件，
- (2) 信号发生器——以 LNI-T UTG4202A 为例，
- (3) 示波器——以 LNI-T MSO3504CS-S 为例，
- (4) 直流可编程电源——以 RIGOL DP832 为例，
- (5) 麦克风，
- (6) 摄像头，
- (7) 显示屏幕，
- (8) 自制用于验证方案的待测模块。

其中信号发生器、示波器以及直流可编程电源型号并非固定，可进行更换。系统工作时，除部分 LLM 模型调用 API 以外，所有运算均部署在板端，旨在降低整体执行带来的

延迟。系统整体设计如下：

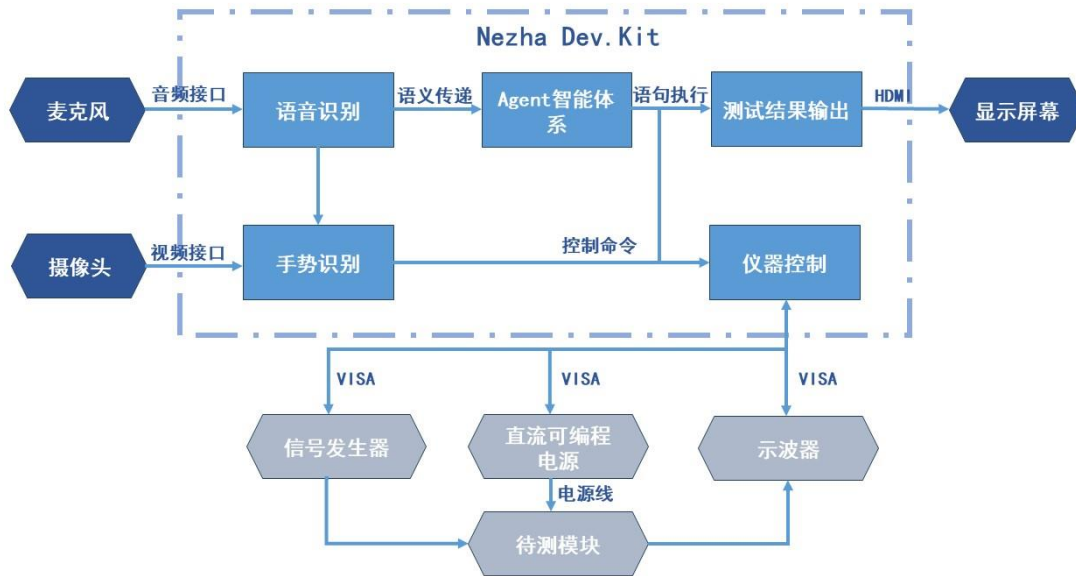


图 1 硬件架构

2.3.1 哪吒（Nezha）开发套件

本设备为参赛的核心设备，提供高性能、低功耗、配置丰富的硬件选型，可满足各类边缘 AI 应用场景对算力、I/O、扩展性的等要求。采用信步边缘 AI 计算平台，搭载 Intel OpenVINO 工具套件，以及来自算法合作伙伴、遍及广泛用例的可商用 AI 算法，帮助开发者快速落地及部署边缘 AI 解决方案。

表 2 哪吒（Nezha）开发套件性能参数

参数	
处理器	Intel® N97 处理器
图形	Intel® UHD Graphics
内存	8GB 双通道 LPDDR5
储存	64GB eMMC
外设接口	HDMI 1.4b/USB 3.2 Gen 2 STACK Connector x 1 (Type-A) 4-pin Front Panel x 1 2-pin Fan Wafer x 1 (12V) 2-pin RTC Battery Wafer x 1 USB 3.2 Gen 2 (Type-A) x 3 10-pin USB 2.0 x 2/UART x 1
USB	40-pin GPIO x 1
扩展	HDMI 1.4b x 1
显示接口	1GbE RJ-45 x 1 (Realtek RTL8111H CG)
以太网	板载 TPM 2.0
安全	有
RTC	Windows® 10 Enterprise LTSC 2021
支持的系统	Linux: Ubuntu 22.04 LTS/Kernel 5.15, Yocto 4.0

2.3.2 信号发生器

选择 LNI-T UTG4202A 作为测试仪器之一。LNI-T UTG4202A 是集函数发生器、任意波形发生器、脉冲发生器、谐波发生器、模拟/数字调制器、频率计等功能于一体的多功能混合信号发生器；1μHz 频率分辨率、以及 1mV 的幅度分辨率，可生成精确、稳定、纯净、低失真的输出信号。

表 3 LNI-T UTG4202A 参数

参数	
最高频率	200MHz
通道数	2
最大采样率	500MSa/s
波形	正弦波、方波。斜波、谐波、脉冲波、噪声、直流电压、任意波形
工作模式	输出选通、持续、调制、扫描、猝发
调制类型	AM、FM、PM、ASK、FSK、PSK、BPSK、QPSK、OSK、PWM、SUM、QAM
幅度范围	1mVpp~11.5Vpp(50Q), 2mVpp~23Vpp(高阻)
频率计	100mHz~800MHz, 7 位
分辨率	1uHz
接口	USB、LAN、10M 时钟源输入输出

2.3.3 示波器

选择 LNI-T MSO3504CS-S 作为测试仪器之一。LNI-T MSO3504CS-S 是集成了 Ultra Phosphor 技术的一款多功能、高性能的示波器，实现了易用性、优异的技术指标及众多功能特性的完美结合，可帮助用户更快地完成测试工作。

表 4 LNI-T MSO3504CS-S 参数

参数	
模拟通道带宽	500MHz,
模拟通道数量	4 通道
模拟通道实时采样率	2.5GSa/s,
数字通道实时采样率	1.25GSa/s(仅 MSO)
输入阻抗	1M Ω 和 50 Ω
存储深度	标配每通道 70Mpts 单次或扫描模式最大存储深度 250Mpts
波形捕获率	率最高 1,000,000wfms/s
内置信号源	内置 50MHz 等性能双通道函数/任意波形发生器，支持实时加载示波器屏幕 数据到 AWG 任意波输出
支持 SCPI 可编程仪器标准命令	是
支持即插即用 USB	可通过 USB 设备与计算机通信

2.3.4 直流可编程电源

选择 RIGOL DP832 作为测试仪器之一。RIGOL DP832 是一款高性能的可编程线性直流电源，拥有清晰的用户界面、优异的性能指标、多种分析功能以及 多种通信接口，可满足多样化的测试需求。

表 5 RIGOL DP832 参数

参数	
额定输入电压	交流 110V/220V \pm 10%，50/60Hz
输出规格	CH1: 电压 0~3V, 电流 0~3A CH2: 电压 0~3V, 电流 0~3A CH3: 电压 3.3V/5V, 电流 3A
电源调节率	电压, \leq 0.01% +2mV 电流, \leq 0.01% +250uA
负载调节率	电压, \leq 0.01% +2mV 电流, \leq 0.01% +250uA
电源纹波(恒压)	\leq 350uVrms
编程分辨率	电压: 10mV, 电流: 1mA
编程接口	USB、RS-232, 支持 SCPI 指令集

2.3.5 麦克风

为了保证系统的良好移动性，选择 USB mini 麦克风来减小不必要的外设体积，其具有高灵敏度，即插即用的特点。

2.3.6 摄像头

为达到良好的手势识别效果，选择无畸变广角且支持自动对焦的摄像头，支持拍摄范围 90°，拍摄范围较大，支持手动变焦。同时其分辨率高达 2592*1944，能够更好的手势识别。

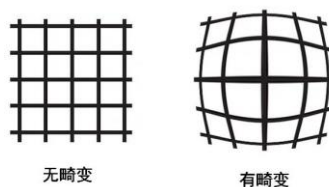


图 2 摄像头畸变示意图

2.3.7 自制待测模块。

为了验证作品可执行性，我们自制了一部分较为典型的测试电路。电路选择如下：

- (1) 运放测试电路，采用自制通用运放模块，选择不同的电路结构来分别测试用运放压的摆率、噪声系数、增益带宽
- (2) 滤波器电路，采用自制有源/无源的滤波电路模块，选择不同的高通、低通、带通
- (3) 滤波电路模块来测试其幅频响应特性曲线和截止频率。
- (4) 比较器电路，采用自制的比较器电路模块，来进行测试其时延。
- (5) 开关电源：采用自制的开关电源模块，结构为 cuk 和 sepic, 测试电源纹波大小。

2.4 软件流程

本系统采用中心分布式的设计，终端各部分有各自的运行逻辑与进程，进程之间相互配合，共同完成整个测试过程。具体过程如下图 3，图 4 所示：

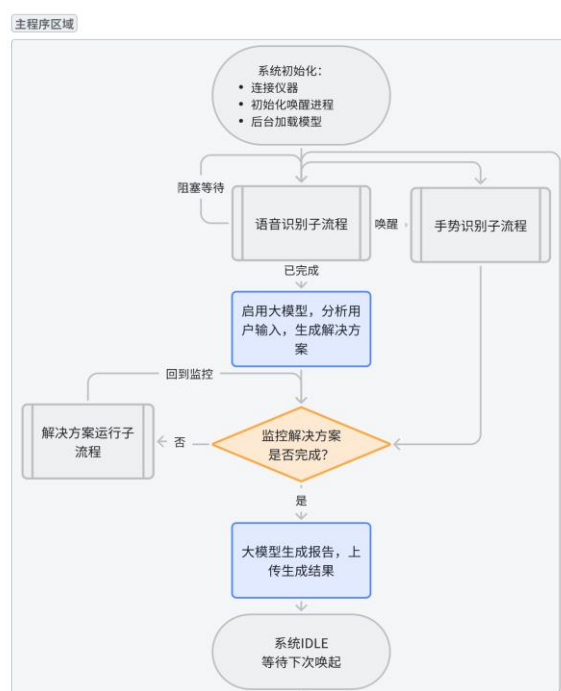


图 3 软件主程序运行流程

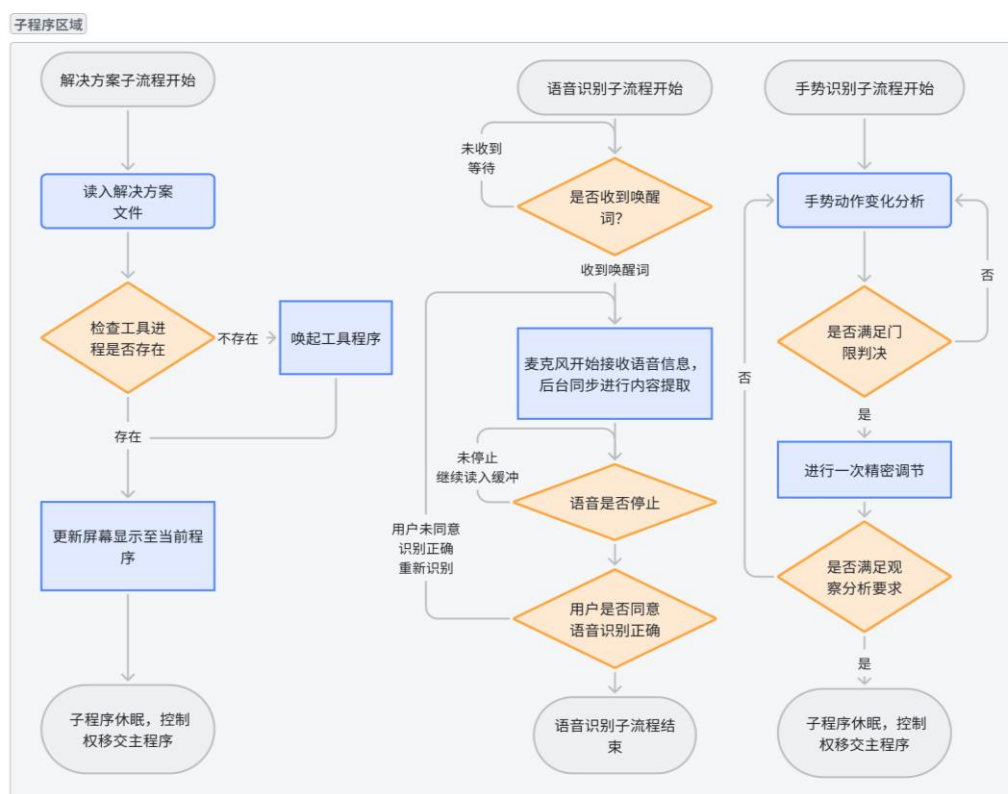


图 4 软件子程序运行流程

由上图可知，系统启动后保持待机状态，待收到关键词唤醒后，进行语音识别，根据用户指令判断当前任务。

如果为测试要求，则系统进行解决方案的子流程的调用。通过 LLM-Agent 将用户请求解析成系统可以理解的标准命令流，进而进行规范性工具链函数的调用来控制仪器完成测量并输出结果和分析报告，最后结束调用并转移至主程序。

如果为修改仪器参数，则系统进行手势识别子流程的调用。通过摄像头进行手势捕捉，进而进行手势动作变化分析，当调节至合适时，结束调用并转移至主程序。

第三章 关键技术和特色

3.1 基于 Agent 体系的 IVI 命令流

在这个部分，我们使用 LLM-Agent 将用户请求解析成系统可以理解的标准命令流，进而调用工具链来实现控制各类仪器。

3.1.1 Agent 的概念

在日常工作中，我们通常将一项任务拆解成几个步骤：制定计划、执行计划、检查结果，然后将成功的作为标准，未成功的则留待下一次循环解决，这种方法已经被证明是高效完成任务的经验总结。而 Agent 的执行过程与人做事的方式类似，以 LLM（大语言模型）作为其核心构建 Agent 是一个很酷的概念，AutoGPT、GPT-Engineer 和 BabyAGI 等几个概念验证演示都是很不错的示例，可以简单的将 Agent 视为一个强大的通用问题解决器。对于一个标准的 Agent 系统，可以由以下核心组件组成：用户请求、Agent、规划、记忆、工具集。

在 LLM 驱动的 Agent 系统中，LLM 充当代理的大脑，处理交互信息；规划模块将用户需求的大型任务分解为更小的、可管理的子目标，从而能够有效处理复杂的任务，同时对过去的行为进行自我批评和自我反思，从错误中吸取教训，并针对未来的步骤进行完善，从而提高最终结果的质量；记忆模块记录着过去的思考、行动及对环境的观察，涵盖了 Agent 和用户之间所有交互；工具集学习调用外部 API 来获取模型权重中缺失的额外信息，包括当前信息、代码执行能力、对专有信息源的访问等。

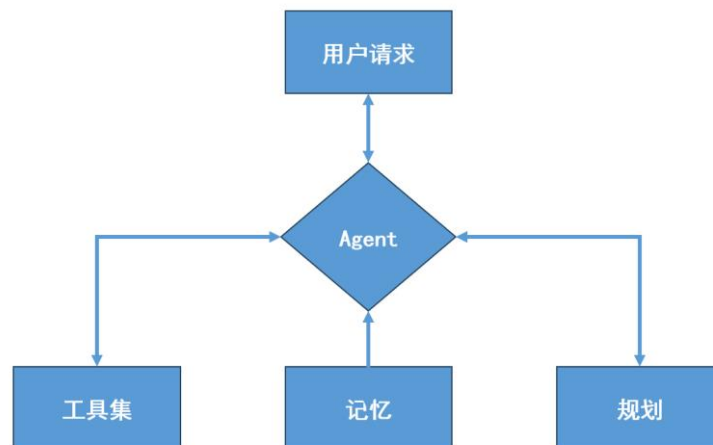


图 5 Agent 系统组成

3.1.2 大语言模型

在 LLM-Agent 选择方面，采用开源的 ChatGLM 模型作为核心引擎，该模型在自然语言处理领域表现优异，能够理解和生成高质量文本内容；同时基于 AgentTuning 技术，利用多个 Agent 任务交互轨迹对 LLM 进行指令调整，以强大泛化能力适应各种仪器调试场景；而且 ChatGLM 原生对中文的强大支持，也便于 Agent 理解我们的需求。同时我们采用嵌入式设备和计算中心两套模型的本地运行和 API 接口整合，确保了其在复杂环境下的稳健性，使其更加高效、精准。

3.1.3 规划模块

而对于规划模块，通常可分为无反馈规划和有反馈规划两种类型。常见的无反馈规划技术包括思维链（COT）和思维树（TOT）；而常见的有反馈规划技术则包括 ReAct 和 Reflexion。本项目采用 ReAct 提示词策略，旨在为每一步操作提供详尽的思维链，以深入理解模型的决策过程。我们采用 LangChain 来构建整个规划模块，它需要处理用户原始的需求，给出一条完整的思维链。这条思维链的格式为：

Thought: ...

Action: ...

Observation: ...

同时为了防止无休止的“思考”下去，我们需要加入 AgentFinish 结束整条链。接着在后面加上一个 parser 解析出每个 Action 传递给 Tools 模块。如此一来，将调试仪器的过程具体化和规范化，同时保证调试过程安全性和正确性。

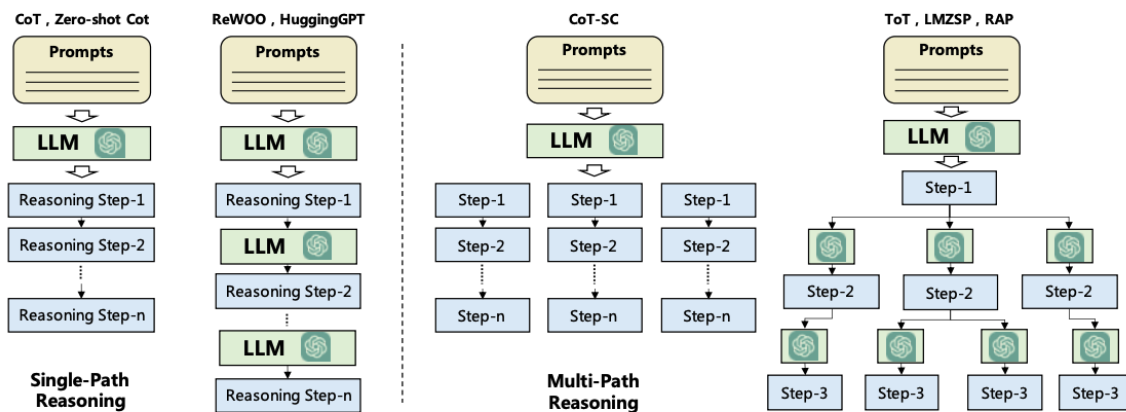


图 6 不同策略的规划模块

3.1.4 记忆模块

一般而言，记忆模块存储 Agent 的内部日志，其中包括过去的思考、行动及对环境的观察，涵盖了 Agent 和用户之间所有交互。主要两种记忆类型是短期记忆和长期记忆。本项目选用混合记忆，综合短期记忆和长期记忆，一方面保证不同场景下调试仪器灵活性，另一方面提高 Agent 长期推理和经验积累能力，让 Agent 记住一些用户指导的仪器调试操作步骤。我们需要实现消息存储与历史记录管理、限制十条上下文长度、消息格式化和输出。规划和记忆模块允许 Agent 在动态环境中操作，并使其能够有效地回忆过去的行为和计划未来的行动。

3.1.5 工具模块

工具集使 LLM-Agent 能够与仪器进行交互。通过 workflow 执行任务，这些 workflow 帮助工具利用方面我们采用 Function Calling，定义一组工具 API 并将其作为请求的一部分提供给模型，使模型能够将规划好的抽象过程转变为现实仪器的操作。我们定义本系统的工具规范为：

- (1) name=name: 这是函数的名称
- (2) func=func: 这是函数的实际执行体
- (3) coroutine=coroutine: 是否为协程
- (4) description=description: 这是函数的描述或说明
- (5) return_direct=return_direct: 是否直接返回

(6) `args_schema=args_schema`: 用于验证和描述函数所需的参数结构。

(7) `kwargs`: 这个是用来接收额外的关键字参数

3.1.6 Agent 整个系统

本次项目中，首先通过语音识别获取用户请求同时将工具集传入 LLM-Agent，再通过规划模块决策整个调试过程，工具利用后返回现实仪器的操作和测试的结果，同时在整个系统中都通过记忆模块来储存 Agent 和用户之间的所有交互。本项目工具集采用 Python 的 PyVISA 库控制设备。IVI 驱动程序标准定义了一个开放的驱动程序架构、一组仪器类和共享软件组件。这些共同提供了仪器互换性所需的关键元素。使用 IVI 驱动程序设计的系统享有标准化代码的好处，这些代码可以互换到其他系统中。该代码还支持测量设备的互换，有助于防止硬件过时。在三个级别上支持可互换性：IVI 架构规范允许架构可互换性——这是可以重复使用的标准驱动程序架构。类规范提供语法互换性，支持以最少的代码更改进行仪器交换。通过使用 IVI 信号规范实现了最高水平的互换性。

Agent 整体工作流程如下所示：

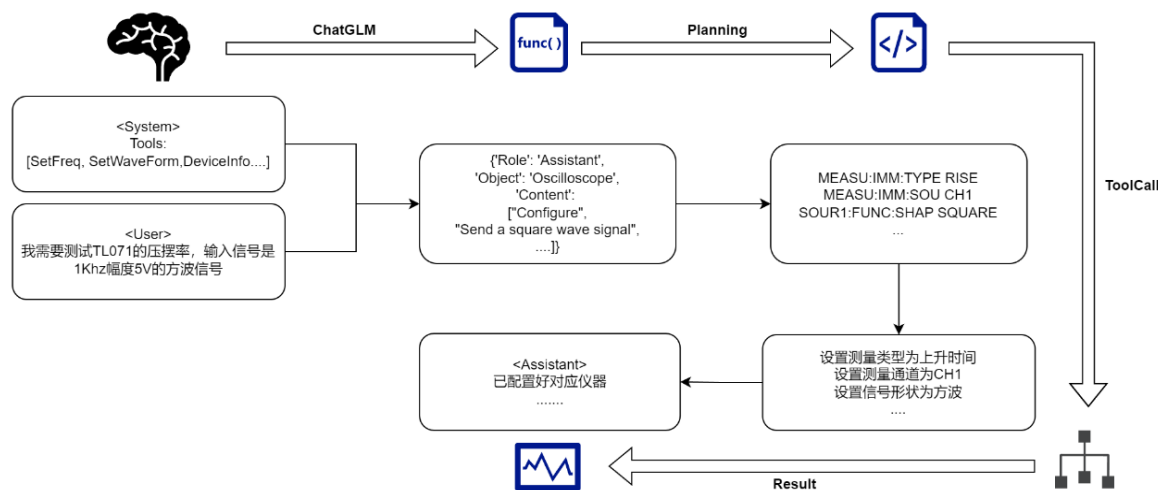


图 7 Agent 工作流程图

我们使用 PyVISA 作为 Python IVI 的通讯接口的封装驱动程序来连接仪器设备。在此基础上进行规范性工具链函数库的封装，以便 Agent 生成的方案转换进一步的调用控制仪器来输出信号以及读取信号，并将信息进行反馈进一步分析后输出分析报告。流程图如下：

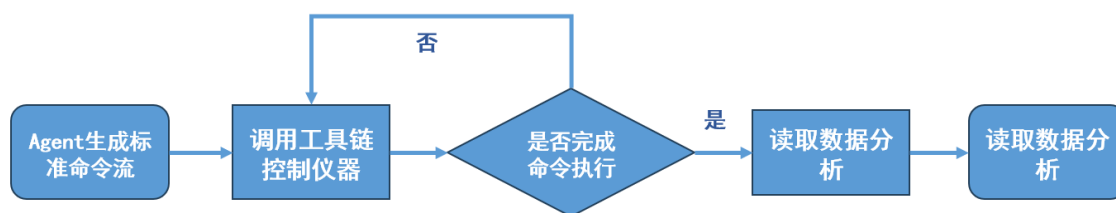


图 8 Agent 控制仪器流程

3.2 智能语音识别

在这部分，我们主要完成对用户语音的识别，以获取用户指令，满足与用户交互的需求。

3.2.1 语音识别 ASR

我们将用户的下达的语音指令通过 AI 处理算法，实现语言识别，进行从语音到自然语言的转换。为指令解析做准备。我们使用阿里巴巴开源的 Paraformer 语音模型和 CT-Transformer 标点模型来进行高性能推理。该模型大大加快了识别速度，同时针对 x86 下的 AI 新架构，采用对 x86 架构的 AVX 支持，适合 N97 神经网络芯片。

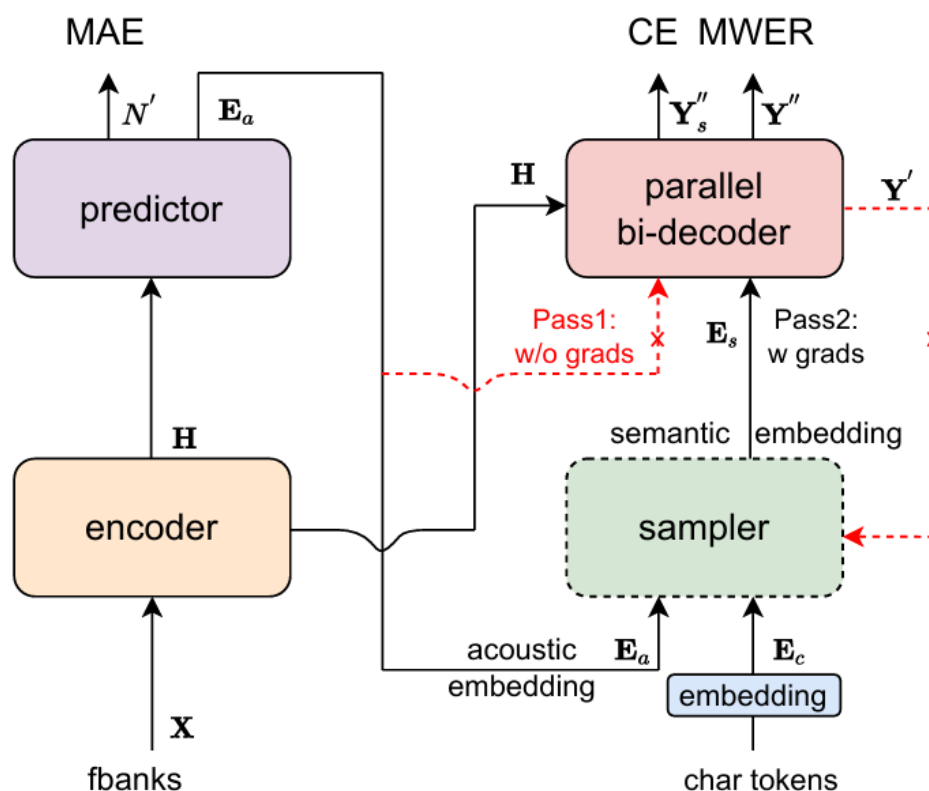


图 9 Paraformer 模型

Paraformer 模型结构如上图所示，由 Encoder、Predictor、Sampler、Decoder 与 Loss function 五部分组成。Encoder 可以采用不同的网络结构，例如 self-attention, conformer, SAN-M 等。Predictor 为两层 FFN，预测目标文字个数以及抽取目标文字对应的声学向量。Sampler 为无学习参数模块，依据输入的声学向量和目标向量，生产含有语义的特征向量。Decoder 结构与自回归模型类似，为双向建模（自回归为单向建模）。Loss function 部分，除了交叉熵（CE）与 MWER 区分性优化目标，还包括了 Predictor 优化目标 MAE。

其核心点主要有：

1. Predictor 模块：基于 Continuous integrate-and-fire (CIF) 的预测器 (Predictor) 来抽取目标文字对应的声学特征向量，可以更加准确的预测语音中目标文字个数。
2. Sampler：通过采样，将声学特征向量与目标文字向量变换成含有语义信息的特征向量，配合双向的 Decoder 来增强模型对于上下文的建模能力。
3. 基于负样本采样的 MWER 训练准则。

Paraformer 属于单轮非自回归模型。院采用一个预测器（Predictor）来预测文字个数并通过 Continuous integrate-and-fire (CIF) 机制来抽取文字对应的声学隐变量；并一个基于 GLM 的 Sampler 模块来增强模型对上下文语义的建模。在离线运行测试下，对于长达 10s 的语音命令，识别的时延也只有 0.2s 左右，这大大提高了交互的效率。

3.2.2 语音抗干扰能力

在复杂噪声环境中，语音信号往往会被环境噪声淹没，导致语音识别效果下降；此外用户的无关交谈内容也可被作为需求输入下一步的 Agent 规划内从而造成不必要的任务。对于本系统，我们的实际需求为对于所需模块性能的测量。为了解决这种可能造成意外的情况。我们采用了两方面来进行语音信号的处理：前端信号处理和后端信号过滤。

在前端信号处理时，通过对静态时麦克风平均功率的获取，来进行基底噪声功率的量化，在此基础上通过对用户发送语音时麦克风平均功率的获取，来进行语音功率的量化，实现动态语音获取。在后端信号过滤时，我们一方面通过中文关键词提取保证有效信息通过。通过使用类似热词一样的词表，来保证有效命令能够正确流向 Agent 以及手势控制参数的选择。

3.2.3 热词功能

热词通常是指对在特定业务领域需要优先准确识别的关键词或短语，在我们的系统中，特别是电子测试当中的专业名词识别准确尤为重要。由于中文语音同音字同音词较多，对于特定词语、特定文字的识别效果不理想，不能满足精度要求，为了解决这个问题，我们采取了热词功能。添加热词可以提升通用引擎在这类专有词汇上的识别准确率。

针对电子测试测量方面，我们针对性的添加了 100 多个专业名词存在本地以提升这类专有词汇上的识别准确率。关键热词部分如下：压摆率、增益带宽积、输入失调电压、等效输入噪声电压、电源电压抑制比、差模开环直流电压增益、输入偏置电流温漂、阻抗匹配、信噪比、电磁兼容性、探头、触发沿。

3.2.4 LLM 语义修正

尽管热词以及解决部分词汇的修正，然而在语音识别中，任然不可避免的会出现模糊发音，在热词功能后我们调用 LLM 模型来进行语义修正和关键信息提取输出以便输出给 Agent 和 3.4 手势识别部分来下达命令。此处我们选择的是阿里巴巴的通义千问。此处先对 LLM 模型预设和调节参数，保证其经可能稳定的输出，通过调用外部 API，来实现与 LLM 的交互。

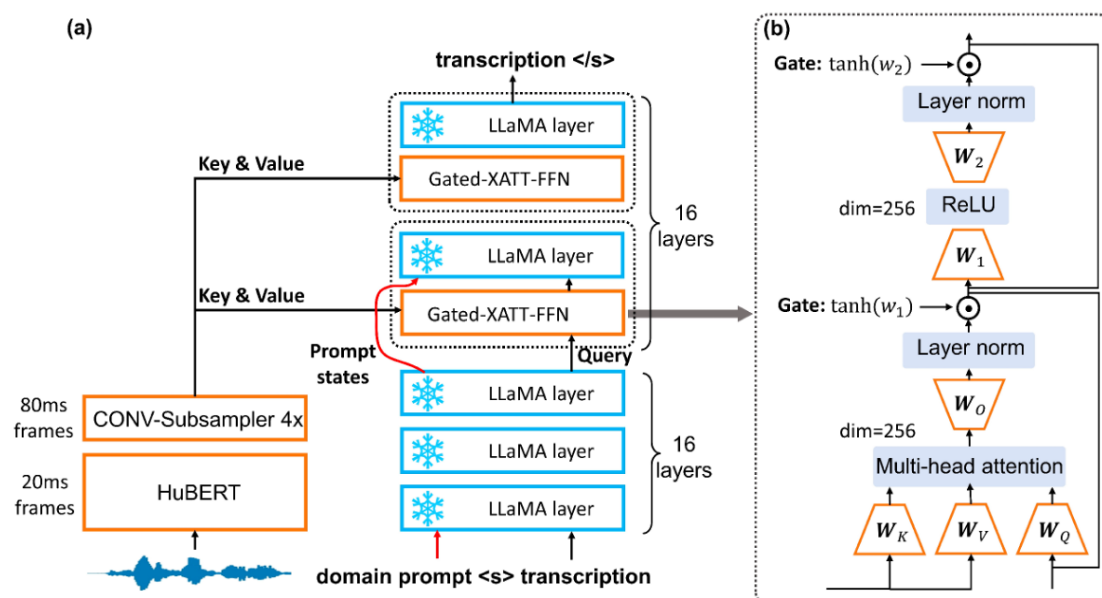


图 10 LLM 辅助语音识别

3.3 手势识别

手势交互是一种更自然、直观的控制方式，在这个部分，我们通过摄像头采集用户的手势动作，通过机器学习模型识别手势动作，来进行进一步的仪器控制和参数精密调节。考虑到用户的手部会在实验桌上大范围活动，我们采用固定放置的广角摄像头来捕捉用户的手势，这样的方案具有更好的可靠性和稳定性。

我们使用 MediaPipe 机器学习模型来识别摄像头采集的视频流中的手势。MediaPipe 是 Google 推出的一款开源机器学习框架，其中包含了许多预训练的模型，可以直接使用，也可以根据需求进行微调训练。MediaPipe 手部识别模型参数量和计算量小，并且提供量化模型，适合在嵌入式设备上部署。对输入图像进行手势识别大致分为三个步骤：

(1) 手掌检测：使用基于卷积神经网络的单阶段目标检测模型，检测输入图像中的手掌位置。在此步骤中，完整的输入图像被缩放到 192×192 尺寸，由卷积神经网络进行特征提取得到表示锚框位置的特征向量，然后通过非极大值抑制算法得到最终的手掌位置。

(2) 手部跟踪：将图像中由手掌检测模型得到的手掌位置部分裁切并缩放到 224×224 尺寸，输入到基于卷积神经网络的回归模型中，预测 21 个关键点的 3D 坐标与置信度。这 21 个关键点包括手掌中心、手指关节等。在上一帧图像存在手部关键点的情况下，后续的预测将改用参数量更小的轻量级模型，以提高实时性。

(3) 手势分类：将手部跟踪模型得到的手部关键点坐标输入到带残差模块的全连接神经网络中得到嵌入向量，并使用全连接分类检测头来得到当前帧中手势的分类结果。MediaPipe 的预训练模型能分类包括伸出食指、握拳、张开手掌动作等 8 种手势，通过结合手部跟踪结果和手势分类结果进行判断，我们还实现了对拧旋钮等手势的识别。

以上的算法流程实现了从单张输入图像中预测其中包含的手势类型。为了实现通过手势控制与系统交互，还需要设计状态机从连续的手势动作中提取出用户的意图，在正确的时机触发系统的相应行为。我们将用户的手势操作分为两类，一是单次交互手势，例如张开手掌、握拳等，我们在一段时间内首次检测到用户做出这类手势时，就触发系统的相应行为，如开始测量、结束测量等；二是连续调节手势，例如拧旋钮、上下滑动等，这类手势允许用户通过手部的运动幅度来连续地调节实验仪器和系统的各项参数，给用户更好的沉浸式交互体验。

我们通过对多帧连续的图像进行分析来提取连续调节手势的信息，流程如下图所示：

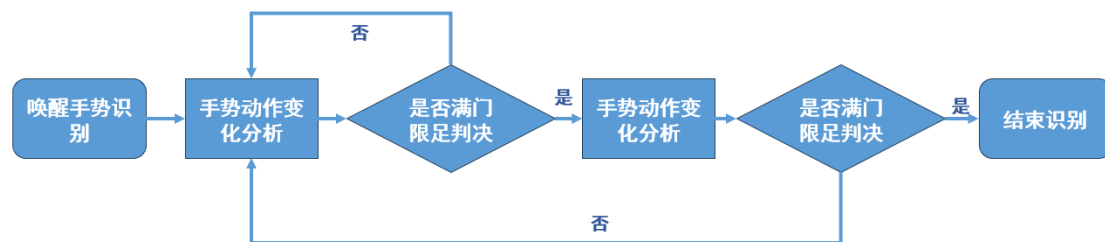


图 11 手势调节流程

首先我们通过手势分类模型识别出连续调节手势的唤醒动作。在用户做出唤醒动作后，我们开始记录用户的手部关键点坐标，并比较用户手势关键点与上一帧的位移。将用户手势的位移大小与预设的阈值相比较，从而区分用户是在退出连续调节手势还是在继续调节。当用户继续调节手势时，根据手指尖关键点的位移换算为待调节量的大小，并实时传递给系统以做出相应的动作。当用户退出连续调节手势时，通过用户最后一帧的手势关键点与唤醒手势的关键点的位移确定最终的调节量，以减小累计误差，带来更高的操作精

度。下图展示了通过检测拇指与食指连线的方位角来实现虚拟旋钮调节。

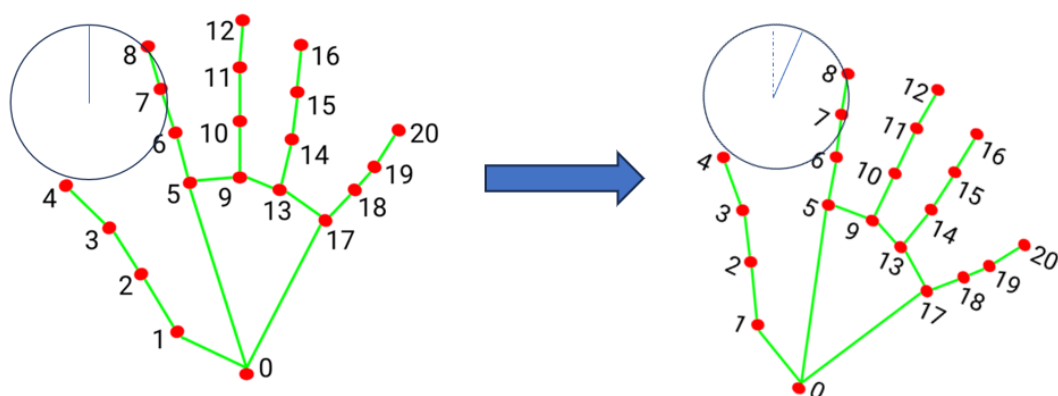


图 12 旋转角度检测示意

旋转检测面临的主要问题在于用户操作的起始和结束点难以准确确定，并且具有一定的可逆性。与具体按键不同，无法直接反馈给用户或设备，这显著降低了其在精密操作调节中的友好性和实用性。为了解决这一挑战，我们进行了算法的改良和优化。

我们采用了经过自主编写验证的算法，并结合了 OpenCV 推理后的优化模型来改进旋转检测的效果。这些改进不仅提高了系统对用户操作的识别精度，还增强了操作的实时响应能力。改良后的算法能够更准确地捕捉用户操作的开始和结束点，从而有效地提升了系统在精密操作调节中的易用性和实用性。为了展示这些改进带来的效果，我们设计了一组运行效果示意图，清晰展示了优化后的旋转检测系统在不同场景下的工作状态和反馈效果。这些示意图不仅直观地展示了系统的操作流程和反馈机制，还验证了改良算法的实际应用价值和效果。



图 13 旋转手势识别

3.4 进程交互协同

哪吒（Nezha）开发套件运行着 3.1、3.2 和 3.3 三个子进程，这些进程在系统启动时同时启动，构成了整个系统的核心功能。为了确保各个子进程在运行过程中环境的独立性和数据的有效交互，我们采用了 socket 通信技术来实现语音识别后的文本数据传输。具体的

交互逻辑如下图所示：语音识别子程序根据命令的性质和需求，有选择性地处理后的数据传递给 Agent 智能体系或手势识别模块，以实现不同的功能和响应用户指令。

在实际操作中，特别是在下达测量命令时，系统需要通过工具链反复调用来控制各种仪器设备，以确保测量过程的顺利进行和数据的准确获取。为了维持整体系统的稳定性和效率，Agent 智能体系被赋予了高优先级，这意味着它在处理命令时会对手势识别模块产生阻塞效应，确保了测量过程中的优先处理和资源分配，从而保证了系统的正常运行和数据的准确性。

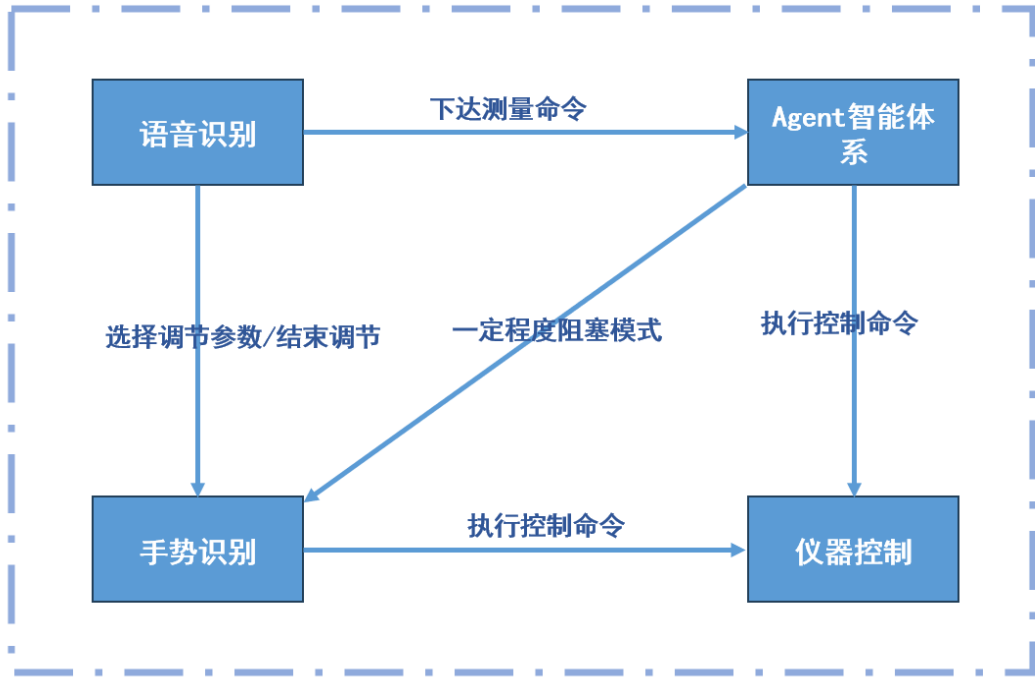


图 14 子程序交互逻辑

第四章 系统测试

4.1 硬件测试

本系统的核心硬件是哪吒（Nezha）开发套件，我们主要使用它的 USB 接口及 HDMI 接口来完成数据的交互，测试内容为多个外设以及核心硬件的运行情况。测试结果如下：

表 6 硬件测试结果

测试项目	测试效果
哪吒（Nezha）开发套件运行测试	正常
摄像头运行测试	正常
麦克风运行测试	正常
HDMI 显示运行测试	正常
信号发生器通信测试	正常
示波器通信测试	正常
可编程直流电源通信测试	正常

4.2 功能测试

由于哪吒（Nezha）开发套件上运行了较多的服务，为测试各个服务的运行情况、精度、速度等指标，我们分别对系统整体和子模块单独进行测试。系统整体运行测试系统整体测试分为以下几个部分：

表 7 系统运行测试结果

测试项目	测试效果
语音识别测试命令	正常
语音识别控制仪器命令	正常
Agent 体系规划	正常
Agent 控制工具链	正常
Agent 控制信号发生器	正常
Agent 控制示波器	正常
Agent 控制直流可编程电源	正常
手势控制仪器参数调节	正常

4.2.1 Agent 体系规划

Agent 体系规划测试主要分为 3 个部分，分别是工具集，记忆和规划的验证。我们通过两种方法来验证其正确性：第 1 种是在无仪器控制的条件下直接输入用户需求，通过其输出工具链调用和规划与正确测试方法的对比来验证其正确性；第 2 种是在连接仪器和待测模块的条件下，下达命令，通过系统测试结果和实际测试结果的对比来验证其正确性。经过我们反复多次测试，Agent 体系的规划具有一定的不确定性，但大部分情况规划和调用工具链较好，这是由于 LLM 自身问题导致的，此外的不足之处的是，在少部分情况调用工具链时，发生了阻塞现象。

4.2.2 语音识别

语音识别测试主要分为 2 个部分，第 1 个部分是语音识别生成文本以及关键词的准确

率；第 2 个部分是语音识别的延时，经过我们反复测试，优化后的语音识别的准确率很高，即使是一些专业名词也能很好的识别出来，且延时非常低，基本做到实时转译。

表 8 语音识别测试结果

测试项目	测试性能
语音识别生成文本以及关键词准确率	98.1%
语音识别延时	≤0.3s
语音抗干扰能力测试	正常

4.2.3 手势识别

手势识别主要分为个部分的测试，第 1 部分是旋转手势识别准确度；第二部分是手势控制各仪器参数的效果。经过我们的反复多次测试，手势控制仪器的交互体验较高，可以在语音下达控制命令后分别实现控制信号发生器的生成波形的频率、幅度、相位、占空比、直流偏置的步进调节；实现控制示波器的耦合方式、X 轴(时基)、Y 轴(幅度)、触发沿电平等的步进调节。

表 9 手势识别效果测试

测试项目	测试性能
旋转手势识别准确度	可以实现 $\pm 5^\circ/360^\circ$ 的精度。
控制信号发生器波形幅度变化	正常
控制信号发生器波形频率变化	正常
控制信号发生器波形相位变化	正常
控制信号发生器波形直流变化	正常
控制信号发生器波形相位变化	正常
控制信号发生器波形占空比变化	正常
控制示波器波形幅度挡位变化	正常
控制示波器波形直流偏置变化	正常
控制示波器波形时间挡位变化	正常
控制示波器波形时间偏移变化	正常
控制示波器波形触发电压变化	正常

第五章 附录

5.1 程序清单

程序清单如下所示：

5.1.1 基于 Agent 体系的 IVI 命令流

```
-- chains.py          # 包含处理思维链和规划的代码文件
-- conversation_log.txt # 存储对话日志的文本文件
-- core               # 核心代码文件夹
| |-- __init__.py      # 核心包初始化文件
| |-- chat_models      # 聊天模型代码文件夹
| | |-- __init__.py    # 聊天模型初始化文件
| | |-- base.py        # 基础聊天模型类
| |-- embeddings       # 嵌入规划模型代码文件夹
| | |-- __init__.py    # 嵌入模型初始化文件
| | |-- base.py        # 基础嵌入模型类
| |-- llm_core.pyc     # 编译后的 LLM 核心代码文件
| |-- llms             # LLM 模块代码文件夹
| | |-- __init__.py    # LLM 模块初始化文件
| | |-- base.py        # 基础 LLM 模块类
| |-- output_parsers   # 输出解析工具代码文件夹
| | |-- __init__.py    # 输出解析工具初始化文件
| | |-- tools.py       # 工具函数集合
-- fb_planning.py      # 包含反馈规划技术的代码文件
-- llmimport.py        # LLM 端口管理代码文件
-- observe_wave.png    # 观察波形图像文件
-- parameters.txt      # 参数文本文件
-- requirements.txt     # 依赖包列表文件
-- templates_data.pkl   # 模板数据文件
-- tools.py            # 工具函数集合
-- util.py             # 实用函数集合\
```

5.1.2 智能语音识别

```
-- models
| |-- paraformer-offline-zh
| |-- punc_ct-transformer_cn-en
-- util
|-- _init_.py # 初始化
|-- asyncio_to_thread.py
|-- chinese_itn.py
```

```
-- |-- clean_assets.py
-- |-- clean_assets.py
-- |-- clean_assets.py
-- |-- client_adjust_srt.py
-- |-- client_check_websocket.py
-- |-- client_cosmic.py
-- |-- client_create_file.py
-- |-- client_file_cosmic.py
-- |-- client_finish_file.py
-- |-- client_finish_file.py
-- |-- client_hot_update.py
-- |-- client_recv_result.py
-- |-- client_rename_audio.py
-- |-- client_send_audio.py
-- |-- client_shortcut_handler.py
-- |-- client_show_tips.py
-- |-- client_stream.py
-- |-- client_strip_punc.py
-- |-- client_transcribe.py
-- |-- client_type_result.py
-- |-- client_write_file.py
-- |-- client_write_md.py
-- |-- empty_working_set.py
-- |-- format_tools.py
-- |-- hot_kwds.py
-- |-- hot_sub_en.py
-- |-- hot_sub_rule.py
-- |-- hot_sub_zh.py
-- |-- my_status.py
-- |-- server_check_model.py
-- |-- server_classes.py
-- |-- server_cosmic.py
-- |-- server_init_recognizer.py
-- |-- server_recognize.py
-- |-- server_ws_recv.py
-- |-- server_ws_send.py
-- |-- srt_from_txt.py
-- |-- text_change.py # 文本修正
-- hot-en.txt
-- hot-rule.txt
-- hot-zh.txt
-- config.py # 配置文件
-- core_client.py    #语音识别客户端
-- core_server.py    # 语音识别服务端
```

```
|-- key.py # 语音识别流式控制
```

```
...
```

5.1.3 手势识别

```
|-- gesture_recognizer.task
```

```
|-- hand_landmarker.task
```

```
|-- connet.py # 语音获取
```

```
|-- cotrol.py # 手势控制仪器参数选择
```

```
|-- gesture_base.py
```

```
|-- hand_trace.py
```

```
|-- knob.py # 旋转手势识别
```

5.2 程序源码

由于篇幅限制，对于上图中的程序清单，我们此处仅展示部分的关键代码。

5.2.1 语音识别客户端

```
# coding: utf-8

import os
import sys
import asyncio
import signal
from pathlib import Path
from platform import system
from typing import List

import time
import pyautogui
import typer
import colorama
import keyboard

from config import ClientConfig as Config
from util.client_cosmic import console, Cosmic
from util.client_stream import stream_open, stream_close
from util.client_shortcut_handler import bond_shortcut
from util.client_recv_result import recv_result
from util.client_show_tips import show_mic_tips, show_file_tips
from util.client_hot_update import update_hot_all, observe_hot

from util.client_transcribe import transcribe_check, transcribe_send,
transcribe_recv
from util.client_adjust_srt import adjust_srt

from util.empty_working_set import empty_current_working_set
```

```
# 确保根目录位置正确，用相对路径加载模型
BASE_DIR = os.path.dirname(__file__); os.chdir(BASE_DIR)

# 确保终端能使用 ANSI 控制字符
colorama.init()

# MacOS 的权限设置
if system() == 'Darwin' and not sys.argv[1:]:
    if os.getuid() != 0:
        print('在 MacOS 上需要以管理员启动客户端才能监听键盘活动，请 sudo 启动')
        input('按回车退出'); sys.exit()
    else:
        os.umask(0o000)

async def main_mic():
    Cosmic.loop = asyncio.get_event_loop()
    Cosmic.queue_in = asyncio.Queue()
    Cosmic.queue_out = asyncio.Queue()

    show_mic_tips()

    # 更新热词
    update_hot_all()
    # 实时更新热词
    observer = observe_hot()

    # 打开音频流
    Cosmic.stream = stream_open()

    # Ctrl-C 关闭音频流，触发自动重启
    signal.signal(signal.SIGINT, stream_close)

    # 绑定按键,改函数
    bond_shortcut()
    # 清空物理内存工作集
    if system() == 'Windows':
        empty_current_working_set()

    # 接收结果
    while True:
```

```
        await recv_result()

async def main_file(files: List[Path]):
    show_file_tips()

    for file in files:
        if file.suffix in ['.txt', '.json', '.srt']:
            adjust_srt(file)
        else:
            await transcribe_check(file)
            await asyncio.gather(
                transcribe_send(file),
                transcribe_recv(file)
            )

    if Cosmic.websocket:
        await Cosmic.websocket.close()
    input('\n 按回车退出\n')

def init_mic():
    try:
        asyncio.run(main_mic())
    except KeyboardInterrupt:
        console.print(f'再见! ')
    finally:
        print('...')

def init_file(files: List[Path]):

    try:
        asyncio.run(main_file(files))
    except KeyboardInterrupt:
        console.print(f'再见! ')
        sys.exit()

if __name__ == "__main__":
    # 如果参数传入文件，那就转录文件
    # 如果没有多余参数，就从麦克风输入
    if sys.argv[1:]:
        typer.run(init_file)
    else:
```

```
init_mic()
```

5.2.2 语音识别服务端

```
import os
import sys
import asyncio
from multiprocessing import Process, Manager
from platform import system

import websockets
from config import ServerConfig as Config
from util.server_cosmic import Cosmic, console
from util.server_check_model import check_model
from util.server_ws_recv import ws_recv
from util.server_ws_send import ws_send
from util.server_init_recognizer import init_recognizer
from util.empty_working_set import empty_current_working_set

BASE_DIR = os.path.dirname(__file__); os.chdir(BASE_DIR)    # 确保
os.getcwd() 位置正确，用相对路径加载模型

async def main():

    # 检查模型文件
    check_model()

    console.line(2)
    console.rule('[bold #d55252]CapsWriter Offline Server');
    console.line()
    console.print(f'项目地址: [cyan
underline]https://github.com/HaujetZhao/CapsWriter-Offline', end='\n\n')
    console.print(f'当前基文件夹: [cyan underline]{BASE_DIR}', end='\n\n')
    console.print(f'绑定的服务地址: [cyan
underline]{Config.addr}:{Config.port}', end='\n\n')

    # 跨进程列表，用于保存 socket 的 id，用于让识别进程查看连接是否中断
    Cosmic.sockets_id = Manager().list()

    # 负责识别的子进程
    recognize_process = Process(target=init_recognizer,
                                args=(Cosmic.queue_in,
                                       Cosmic.queue_out,
                                       Cosmic.sockets_id),
                                daemon=True)
```

```
recognize_process.start()
Cosmic.queue_out.get()
console.rule('[green3]开始服务')
console.line()

# 清空物理内存工作集
if system() == 'Windows':
    empty_current_working_set()

# 负责接收客户端数据的 coroutine
recv = websockets.serve(ws_recv,
                        Config.addr,
                        Config.port,
                        subprotocols=["binary"],
                        max_size=None)

# 负责发送结果的 coroutine
send = ws_send()
await asyncio.gather(recv, send)

def init():
    try:
        asyncio.run(main())
    except KeyboardInterrupt:          # Ctrl-C 停止
        console.print('\n 再见! ')
    except OSError as e:                # 端口占用
        console.print(f'出错了: {e}', style='bright_red');
console.input('...')
    except Exception as e:
        print(e)
    finally:
        Cosmic.queue_out.put(None)
        sys.exit(0)
        # os._exit(0)

if __name__ == "__main__":
    init()
```

5.2.3 ReAct 规划部分

```
from __future__ import annotations

from typing import List, Optional, Sequence, Union
```

```
from langchain_core.language_models import BaseLanguageModel
from langchain_core.prompts import BasePromptTemplate
from langchain_core.runnables import Runnable, RunnablePassthrough
from langchain_core.tools import BaseTool

from langchain.agents import AgentOutputParser
from langchain.agents.format_scratchpad import format_log_to_str
from langchain.agents.output_parsers import
ReActSingleInputOutputParser
from langchain.tools.render import ToolsRenderer,
render_text_description

def create_react_agent(
    llm: BaseLanguageModel,
    tools: Sequence[BaseTool],
    prompt: BasePromptTemplate,
    output_parser: Optional[AgentOutputParser] = None,
    tools_renderer: ToolsRenderer = render_text_description,
    *,
    stop_sequence: Union[bool, List[str]] = True,
) -> Runnable:
    missing_vars = {"tools", "tool_names",
"agent_scratchpad"}.difference(
        prompt.input_variables + list(prompt.partial_variables)
    )
    if missing_vars:
        raise ValueError(f"Prompt missing required variables:
{missing_vars}")

    prompt = prompt.partial(
        tools=tools_renderer(list(tools)),
        tool_names=", ".join([t.name for t in tools]),
    )
    if stop_sequence:
        stop = ["\nObservation"] if stop_sequence is True else
stop_sequence
        llm_with_stop = llm.bind(stop=stop)
    else:
        llm_with_stop = llm
    output_parser = output_parser or ReActSingleInputOutputParser()
    agent = (
        RunnablePassthrough.assign(
```



```
        agent_scratchpad=lambda x:
format_log_to_str(x["intermediate_steps"]),
    )
    | prompt
    | llm_with_stop
    | output_parser
)
return agent
```

5.2.4 记忆模块部分

```
import warnings
from abc import ABC
from typing import Any, Dict, Optional, Tuple

from langchain_core.chat_history import (
    BaseChatMessageHistory,
    InMemoryChatMessageHistory,
)
from langchain_core.memory import BaseMemory
from langchain_core.messages import AIMessage, HumanMessage
from langchain_core.pydantic_v1 import Field

from langchain.memory.utils import get_prompt_input_key

from typing import Any, Dict, List, Optional

from langchain_core.messages import BaseMessage, get_buffer_string
from langchain_core.pydantic_v1 import root_validator

from langchain.memory.chat_memory import BaseChatMemory, BaseMemory
from langchain.memory.utils import get_prompt_input_key

class LimitedHistoryMemory(BaseChatMemory):
    human_prefix: str = "Human"
    ai_prefix: str = "AI"
    memory_key: str = "history"
    max_history_length: int = 10
    log_file: str = "conversation_log.txt"

    def __init__(self, **data):
        super().__init__(**data)
        with open(self.log_file, 'a') as f:
            pass
```

```

@property
def buffer(self) -> Any:
    return self.buffer_as_messages if self.return_messages else
self.buffer_as_str

    async def abuffer(self) -> Any:
        return await self.abuffer_as_messages() if self.return_messages
else await self.abuffer_as_str()

def _buffer_as_str(self, messages: List[BaseMessage]) -> str:
    return "\n".join(
        f"{self.human_prefix}: {msg.content}" if isinstance(msg,
HumanMessage) else f"{self.ai_prefix}: {msg.content}"
        for msg in messages
    )

@property
def buffer_as_str(self) -> str:
    return self._buffer_as_str(self.chat_memory.messages[-
self.max_history_length:])

    async def abuffer_as_str(self) -> str:
        messages = await self.chat_memory.aget_messages()
        return self._buffer_as_str(messages[-self.max_history_length:])

@property
def buffer_as_messages(self) -> List[BaseMessage]:
    return self.chat_memory.messages[-self.max_history_length:]

    async def abuffer_as_messages(self) -> List[BaseMessage]:
        messages = await self.chat_memory.aget_messages()
        return messages[-self.max_history_length:]

@property
def memory_variables(self) -> List[str]:
    return [self.memory_key]

    def load_memory_variables(self, inputs: Dict[str, Any]) -> Dict[str,
Any]:
        return {self.memory_key: self.buffer}

    async def aload_memory_variables(self, inputs: Dict[str, Any]) ->
Dict[str, Any]:
        buffer = await self.abuffer()
    
```

```
        return {self.memory_key: buffer}

    def save_context(self, inputs: Dict[str, Any], outputs: Dict[str, str]) -> None:
        super().save_context(inputs, outputs)
        self._truncate_memory()
        self._log_conversation(inputs, outputs)

    async def asave_context(self, inputs: Dict[str, Any], outputs: Dict[str, str]) -> None:
        await super().asave_context(inputs, outputs)
        self._truncate_memory()
        self._log_conversation(inputs, outputs)

    def _truncate_memory(self) -> None:
        if len(self.chat_memory.messages) > self.max_history_length:
            self.chat_memory.messages = self.chat_memory.messages[-self.max_history_length:]

    def _log_conversation(self, inputs: Dict[str, Any], outputs: Dict[str, str]) -> None:
        input_str, output_str = self._get_input_output(inputs, outputs)
        with open(self.log_file, 'a') as f:
            f.write(f"{self.human_prefix}: {input_str}\n")
            f.write(f"{self.ai_prefix}: {output_str}\n")
```

5.2.5 工具链

这部分内容比较多，这里只展示部分工具函数，以及工具定义规范：

```
import time
import pyvisa as visa
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
from pyvisa.errors import VisaIOError
from PIL import Image

from typing import Literal

from langchain.agents import Tool
from langchain_core.tools import StructuredTool
```

```
import functools

print("正在连接仪器.....")
rm = visa.ResourceManager()
sources = rm.list_resources()
if not sources:
    print("未找到仪器")
print(sources)
inst = rm.open_resource("USB0::0x5656::0x0832::AMOL323130008::INSTR")
awg = rm.open_resource("USB0::0x6656::0x0834::AWG4422490001::INSTR")

inst.timeout = 10000
response = inst.query("*IDN?")
print("已连接到仪器: " + response)

#####for debug#####
def save_parameters_to_file(parameters):
    with open("parameters.txt", "a") as f:
        f.write(f"{datetime.now()}: {parameters}\n")

#####for debug#####

# decorator
def parse_params(param_string):
    if param_string is None or param_string == "":
        return {}
    param_dict = {}
    params = param_string.split(", ")
    for param in params:
        key, value = param.split("=")
        value = value.strip("'") # 去除单引号
        if value.strip(): # 确保值不为空字符串
            try:
                value = float(value)
            except ValueError:
                pass
        param_dict[key] = value
    return param_dict

def param_decorator(func):
    @functools.wraps(func)
```

```

def wrapper(param_string=None):
    param_dict = parse_params(param_string)
    return func(**param_dict)

return wrapper

@param_decorator
def initialize_oscilloinst(initial_state: str = "ON") -> str:
    """初始化示波器"""
    # 重置并清除仪器
    inst.write(":RST")
    inst.write(":SYSTem:CLEAr")

    save_parameters_to_file(f"initialize_oscilloinst")
    return f"\n 示波器初始化完成。 \n"

@param_decorator
def set_oscilloinst_channel(
    channel: str = "CHAN1",
    state: Literal["ON", "OFF"] = "ON",
    coupling: Literal["DC", "AC", "GND"] = "DC",
    resistance: float = 50,
    invert: Literal["ON", "OFF"] = "OFF",
    probe: str = "1X",
    offset: float = 0.0,
    scale: float = 1.0,
    units: Literal["VOLTs", "AMPeres", "WATTs", "UNKNown"] = "VOLTs",
    vernier: Literal["ON", "OFF"] = "OFF",
) -> str:
    """
    设置示波器的指定通道，包括通道标识符、显示状态、耦合方式、电阻值、是否反转
    信号、探头衰减因子、波形垂直偏移量、垂直缩放因子、测量单位和精细调节选项。
    """
    inst.write(f":{channel}:DISP {state}")
    inst.write(f":{channel}:COUP {coupling}")
    inst.write(f":{channel}:LOAD {resistance}")
    inst.write(f":{channel}:INVert {invert}")
    inst.write(f":{channel}:PROBe {probe}")
    inst.write(f":{channel}:OFFSet {offset}")
    inst.write(f":{channel}:SCALE {scale}")
    inst.write(f":{channel}:UNITs {units}")
    inst.write(f":{channel}:VERNier {vernier}")
    
```

```

    save_parameters_to_file(
        f"set_channel: {channel}/ {state}/ {coupling}/ {resistance}/
{invert}/ {probe}/ {offset}/ {scale}/ {units}/ {vernier}"
    )

    return (
        f"\n 通道 {channel} 设置已完成: \n"
        f"显示状态: {state}\n"
        f"耦合方式: {coupling}\n"
        f"电阻值: {resistance}Ω\n"
        f"反相: {invert}\n"
        f"探头衰减: {probe}\n"
        f"垂直位移: {offset}\n"
        f"伏格档位: {scale}\n"
        f"单位: {units}\n"
        f"微调: {vernier}\n"
    )

@param_decorator
def configure_awg(
    channel: str = "CHANnel1",
    mode: Literal["CONTinue", "MODulation", "SWEEp", "BURSt"] =
"CONTinue",
    waveform: Literal[
        "SINe", "SQUare", "PULSe", "RAMP", "ARB", "NOISe", "DC"
    ] = "SQUare",
    frequency: float = 2e5,
    amplitude: float = 2,
    offset: float = 0,
    phase: float = 0,
    duty: int = 50,
    invert: bool = False,
    sync_invert: bool = False,
    limit_enable: bool = False,
    limit_lower: float = None,
    limit_upper: float = None,
    amplitude_unit: Literal["VPP", "DBM", "VRMS"] = "VPP",
    load: float = 50,
    psk_code: Literal["PN7", "PN9", "PN15", "PN21"] = "PN7",
    qam_code: Literal[
        "PN7", "PN9", "PN11", "PN15", "PN17", "PN21", "PN23", "PN25"
    ] = "PN7",

```

```

        trigger_source: Literal["INTernal", "EXTRise", "MANual"] =
"INTernal",
        trigger_output: Literal["CLOSe", "RISe", "FALL"] = "RISe",
    ) -> str:
        """配置信号发生器的通道、工作模式、波形类型、频率、幅值、偏移量、相位角
        度、占空比、反向输出、同步反向输出、幅值限制、幅值单位、负载阻抗、PSK 编码、
        QAM 编码、触发源和触发输出极性参数"""
        awg.write(f":{channel}:MODE {mode}")
        awg.write(f":{channel}:BASE:WAVe {waveform}")
        awg.write(f":{channel}:BASE:FREQuency {frequency}")
        awg.write(f":{channel}:BASE:AMPLitude {amplitude}")
        awg.write(f":{channel}:BASE:OFFSet {offset}")
        awg.write(f":{channel}:BASE:PHase {phase}")
        awg.write(f":{channel}:BASE:DUTY {duty}")
        awg.write(f":{channel}:INVersion {'ON' if invert else 'OFF'}")
        awg.write(f":{channel}:OUTPut:SYNc:INVersion {'ON' if sync_invert
else 'OFF'}")
        awg.write(f":{channel}:LIMit:ENABle {'ON' if limit_enable else
'OFF'}")
        if limit_lower is not None:
            awg.write(f":{channel}:LIMit:LOWer {limit_lower}")
        if limit_upper is not None:
            awg.write(f":{channel}:LIMit:UPPer {limit_upper}")
        awg.write(f":{channel}:AMPLitude:UNIT {amplitude_unit}")
        awg.write(f":{channel}:LOAD {load}")
        awg.write(f":{channel}:PSK:PNCode {psk_code}")
        awg.write(f":{channel}:QAM:PNCode {qam_code}")
        awg.write(f":{channel}:TRIGger:SOURce {trigger_source}")
        awg.write(f":{channel}:TRIGger:OUTPut {trigger_output}")
        awg.write(f":{channel}:OUTPut ON")
        save_parameters_to_file(
            f"configure_awg: {channel}/ {mode}/ {waveform}/ {frequency}/
{amplitude}/ {offset}/ {phase}/ {duty}/ {invert}/ {sync_invert}/
{limit_enable}/ {limit_lower}/ {limit_upper}/ {amplitude_unit}/ {load}/
{psk_code}/ {qam_code}/ {trigger_source}/ {trigger_output}"
        )
        return (
            f"\n 通道 {channel} 设置已完成: \n"
            f"模式: {mode}\n"
            f"波形: {waveform}\n"
            f"频率: {frequency} Hz\n"
            f"幅度: {amplitude} {amplitude_unit}\n"
            f"偏置: {offset}\n"
            f"相位: {phase}\n"

```

```

        f"占空比: {duty}%\n"
        f"反相: {'ON' if invert else 'OFF'}\n"
        f"同步反向输出: {'ON' if sync_invert else 'OFF'}\n"
        f"限幅: {'ON' if limit_enable else 'OFF'}\n"
        f"限幅下限: {limit_lower if limit_lower is not None else '未设置'}\n"
    '} \n'
    f"限幅上限: {limit_upper if limit_upper is not None else '未设置'}\n"
    '} \n'

    f"负载: {load} Ω\n"
    f"PSK 码: {psk_code}\n"
    f"QAM 码: {qam_code}\n"
    f"触发源: {trigger_source}\n"
    f"触发输出: {trigger_output}\n"
)

@param_decorator
def configure_sweep_mode_awg(
    channel: str = "CHANnel1",
    sweep_type: Literal["LINE", "LOG"] = "LINE",
    start_frequency: float = 2000,
    stop_frequency: float = 4000,
    sweep_time: float = 2,
    trigger_sweep: bool = False,
) -> str:
    """配置信号发生器的通道、扫描类型、起始频率、终止频率、扫描时间和触发扫描参数"""
    awg.write(f":{channel}:SWEep:TYPe {sweep_type}")
    awg.write(f":{channel}:SWEep:FREQuency:START {start_frequency}")
    awg.write(f":{channel}:SWEep:FREQuency:STOP {stop_frequency}")
    awg.write(f":{channel}:SWEep:TIME {sweep_time}")
    if trigger_sweep:
        awg.write(f":{channel}:SWEep:TRIGger")
    return (
        f"\n 通道 {channel} 扫频配置完成: \n"
        f"扫频类型: {sweep_type}\n"
        f"起始频率: {start_frequency} Hz\n"
        f"截止频率: {stop_frequency} Hz\n"
        f"扫频时间: {sweep_time} s\n"
        f"扫频触发: {'是' if trigger_sweep else '否'}\n"
    )

....
# 这里定义其他工具函数
....

```



```
# 定义工具列表
tools = [
    StructuredTool.from_function(
        name="initialize_oscilloinst",
        func=initialize_oscilloinst,
        description="初始化示波器。重置并清除仪器，未设置自动设置。",
    ),
    StructuredTool.from_function(
        name="set_oscilloinst_channel",
        func=set_oscilloinst_channel,
        description="设置示波器的指定通道，包括通道标识符、显示状态、耦合方式、电阻值、是否反转信号、探头衰减因子、波形垂直偏移量、垂直缩放因子、测量单位和精细调节选项。",
    ),
    StructuredTool.from_function(
        name="configure_awg",
        func=configure_awg,
        description="配置信号发生器的通道、工作模式、波形类型、频率、幅值、偏移量、相位角度、占空比、反向输出、同步反向输出、幅值限制、幅值单位、负载阻抗、PSK 编码、QAM 编码、触发源和触发输出极性参数",
    ),
    StructuredTool.from_function(
        name="configure_sweep_mode_awg",
        func=configure_sweep_mode_awg,
        description="配置信号发生器扫频模式下的通道、扫描类型、起始频率、终止频率、扫描时间和触发扫描参数",
    ),
    ...
]
```

5.2.6 语音识别服务端

```
import os
import sys
import asyncio
from multiprocessing import Process, Manager
from platform import system

import websockets
from config import ServerConfig as Config
from util.server_cosmic import Cosmic, console
from util.server_check_model import check_model
from util.server_ws_recv import ws_recv
```

```
from util.server_ws_send import ws_send
from util.server_init_recognizer import init_recognizer
from util.empty_working_set import empty_current_working_set

BASE_DIR = os.path.dirname(__file__); os.chdir(BASE_DIR)    # 确保
os.getcwd() 位置正确，用相对路径加载模型

async def main():

    # 检查模型文件
    check_model()

    console.line(2)
    console.rule('[bold #d55252]CapsWriter Offline Server');
console.line()
    console.print(f'项目地址: [cyan
underline]https://github.com/HaujetZhao/CapsWriter-Offline', end='\n\n')
    console.print(f'当前基文件夹: [cyan underline]{BASE_DIR}', end='\n\n')
    console.print(f'绑定的服务地址: [cyan
underline]{Config.addr}:{Config.port}', end='\n\n')

    # 跨进程列表，用于保存 socket 的 id，用于让识别进程查看连接是否中断
    Cosmic.sockets_id = Manager().list()

    # 负责识别的子进程
    recognize_process = Process(target=init_recognizer,
                                args=(Cosmic.queue_in,
                                       Cosmic.queue_out,
                                       Cosmic.sockets_id),
                                daemon=True)

    recognize_process.start()
    Cosmic.queue_out.get()
    console.rule('[green3]开始服务')
    console.line()

    # 清空物理内存工作集
    if system() == 'Windows':
        empty_current_working_set()

    # 负责接收客户端数据的 coroutine
    recv = websockets.serve(ws_recv,
                            Config.addr,
                            Config.port,
                            subprotocols=["binary"],
```

```
max_size=None)

# 负责发送结果的 coroutine
send = ws_send()
await asyncio.gather(recv, send)

def init():
    try:
        asyncio.run(main())
    except KeyboardInterrupt:          # Ctrl-C 停止
        console.print('\n 再见! ')
    except OSError as e:                # 端口占用
        console.print(f'出错了: {e}', style='bright_red');
console.input('...')
    except Exception as e:
        print(e)
    finally:
        Cosmic.queue_out.put(None)
        sys.exit(0)
        # os._exit(0)

if __name__ == "__main__":
    init()
```

5.2.7 语音服务客户端

```
# coding: utf-8

import os
import sys
import asyncio
import signal
from pathlib import Path
from platform import system
from typing import List

import time
import pyautogui
import typer
import colorama
import keyboard

from config import ClientConfig as Config
```

```
from util.client_cosmic import console, Cosmic
from util.client_stream import stream_open, stream_close
from util.client_shortcut_handler import bond_shortcut
from util.client_recv_result import recv_result
from util.client_show_tips import show_mic_tips, show_file_tips
from util.client_hot_update import update_hot_all, observe_hot

from util.client_transcribe import transcribe_check, transcribe_send,
transcribe_recv
from util.client_adjust_srt import adjust_srt

from util.empty_working_set import empty_current_working_set

# 确保根目录位置正确，用相对路径加载模型
BASE_DIR = os.path.dirname(__file__); os.chdir(BASE_DIR)

# 确保终端能使用 ANSI 控制字符
colorama.init()

# MacOS 的权限设置
if system() == 'Darwin' and not sys.argv[1:]:
    if os.getuid() != 0:
        print('在 MacOS 上需要以管理员启动客户端才能监听键盘活动，请 sudo 启动')
        input('按回车退出'); sys.exit()
    else:
        os.umask(0o000)

async def main_mic():
    Cosmic.loop = asyncio.get_event_loop()
    Cosmic.queue_in = asyncio.Queue()
    Cosmic.queue_out = asyncio.Queue()

    show_mic_tips()

    # 更新热词
    update_hot_all()
    # 实时更新热词
    observer = observe_hot()

    # 打开音频流
    Cosmic.stream = stream_open()
```

```
# Ctrl-C 关闭音频流, 触发自动重启
signal.signal(signal.SIGINT, stream_close)

# 绑定按键, 改函数
bond_shortcut()

# 清空物理内存工作集
if system() == 'Windows':
    empty_current_working_set()

# 接收结果
while True:
    await recv_result()

async def main_file(files: List[Path]):
    show_file_tips()

    for file in files:
        if file.suffix in ['.txt', '.json', '.srt']:
            adjust_srt(file)
        else:
            await transcribe_check(file)
            await asyncio.gather(
                transcribe_send(file),
                transcribe_recv(file)
            )

    if Cosmic.websocket:
        await Cosmic.websocket.close()
    input('\n 按回车退出\n')

def init_mic():
    try:
        asyncio.run(main_mic())
    except KeyboardInterrupt:
        console.print(f'再见! ')
    finally:
        print('...')

def init_file(files: List[Path]):

    try:
        asyncio.run(main_file(files))
```

```
except KeyboardInterrupt:
    console.print(f'再见! ')
    sys.exit()

if __name__ == "__main__":
    # 如果参数传入文件, 那就转录文件
    # 如果没有多余参数, 就从麦克风输入
    if sys.argv[1:]:
        typer.run(init_file)
    else:
        init_mic()
```

5.2.8 旋转手势识别

```
import time
import cv2
import numpy as np
import mediapipe as mp
import pyvisa

from gesture_base import HandLandmarker, draw_landmarks_on_image
from typing import Literal

def count_fingers_raised(
    rgb_image, detection_result: mp.tasks.vision.HandLandmarkerResult
):
    """Iterate through each hand, checking if fingers (and thumb) are
    raised.
    Hand landmark enumeration (and weird naming convention) comes from
    https://developers.google.com/mediapipe/solutions/vision/hand_landm
    arker."""
    try:
        # Get Data
        hand_landmarks_list = detection_result.hand_landmarks
        # Counter
        numRaised = 0
        # for each hand...
        for idx in range(len(hand_landmarks_list)):
            hand_landmarks = hand_landmarks_list[idx]
            # for each fingertip... (hand_landmarks 4, 8, 12, and 16)
            for i in range(8, 21, 4):
                # make sure finger is higher in image the 3 proceeding
                values (2 finger segments and knuckle)
                tip_y = hand_landmarks[i].y
```

```

        dip_y = hand_landmarks[i - 1].y
        pip_y = hand_landmarks[i - 2].y
        mcp_y = hand_landmarks[i - 3].y
        if tip_y < min(dip_y, pip_y, mcp_y):
            numRaised += 1

    # for the thumb
    # use direction vector from wrist to base of thumb to
determine "raised"
    tip_x = hand_landmarks[4].x
    dip_x = hand_landmarks[3].x
    pip_x = hand_landmarks[2].x
    mcp_x = hand_landmarks[1].x
    palm_x = hand_landmarks[0].x
    if mcp_x > palm_x:
        if tip_x > max(dip_x, pip_x, mcp_x):
            numRaised += 1
    else:
        if tip_x < min(dip_x, pip_x, mcp_x):
            numRaised += 1

    # display number of fingers raised on the image
    annotated_image = np.copy(rgb_image)
    height, width, _ = annotated_image.shape
    text_x = int(hand_landmarks[0].x * width) - 100
    text_y = int(hand_landmarks[0].y * height) + 50
    cv2.putText(
        img=annotated_image,
        text=str(numRaised) + " Fingers Raised",
        org=(text_x, text_y),
        fontFace=cv2.FONT_HERSHEY_DUPLEX,
        fontScale=1,
        color=(0, 0, 255),
        thickness=2,
        lineType=cv2.LINE_4,
    )
    return annotated_image
except:
    return rgb_image

def get_angle(dx1, dy1, dx2, dy2):
    dot_product = dx1 * dx2 + dy1 * dy2
    magnitude1 = (dx1**2 + dy1**2) ** 0.5
    magnitude2 = (dx2**2 + dy2**2) ** 0.5

```

```

    if magnitude1 == 0 or magnitude2 == 0:
        return 0
    cosine = dot_product / (magnitude1 * magnitude2)
    cosine = np.clip(cosine, -1, 1)
    angle = np.arccos(cosine) * 180 / np.pi
    return angle

def is_clockwise(dx1, dy1, dx2, dy2):
    return dx1 * dy2 - dx2 * dy1 > 0

class KnobRecognizer:
    last_knob: tuple | None

    def __init__(self, width, height):
        self.total_value = 0
        self.dangle = 0
        self.last_knob = None
        self.last_timestamp = None
        self.starting_knob = None
        self.width = width
        self.height = height

    def _find_knob(self, hand_landmarks):
        def get_finger_angle(a, b, c, d):
            dx1 = hand_landmarks[b].x - hand_landmarks[a].x
            dy1 = hand_landmarks[b].y - hand_landmarks[a].y
            dx2 = hand_landmarks[d].x - hand_landmarks[c].x
            dy2 = hand_landmarks[d].y - hand_landmarks[c].y
            return abs(get_angle(dx1, dy1, dx2, dy2))

        thumb_angle = get_finger_angle(1, 2, 3, 4)
        index_angle = get_finger_angle(5, 6, 6, 7)
        open_angle = get_finger_angle(2, 3, 5, 6)
        if (
            thumb_angle < 45
            # and index_angle > 20
            and index_angle < 90
            and open_angle < 30
        ):
            # draw circle by diameter from [4] to [8]
            x1 = int(hand_landmarks[4].x * self.width)
            y1 = int(hand_landmarks[4].y * self.height)
            x2 = int(hand_landmarks[8].x * self.width)

```



```

        y2 = int(hand_landmarks[8].y * self.height)
        return x1, y1, x2, y2
    else:
        return None

    def update(self, detection_result:
mp.tasks.vision.HandLandmarkerResult, timestamp):
        if timestamp == self.last_timestamp:
            return
        current_knob = None
        if hasattr(detection_result, "hand_landmarks"):
            hand_landmarks_list = detection_result.hand_landmarks
            for hand_landmarks in hand_landmarks_list:
                current_knob = self._find_knob(hand_landmarks)
                if current_knob:
                    break
        if current_knob is not None:
            if self.last_knob is not None:
                def length(x1, y1, x2, y2):
                    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5

                dt = timestamp - self.last_timestamp
                d_last = length(*self.last_knob)
                d_current = length(*current_knob)
                drate = (d_current - d_last) / d_last / dt
                # print(drate)
                if abs(drate) < 1:
                    # the movement is slow enough to be considered as a
knob rotation

                    if self.starting_knob is None:
                        self.starting_knob = self.last_knob
                    d = (
                        self.starting_knob[2] - self.starting_knob[0],
                        self.starting_knob[3] - self.starting_knob[1],
                        current_knob[2] - current_knob[0],
                        current_knob[3] - current_knob[1],
                    )
                    angle = get_angle(*d) // 5
                    if is_clockwise(*d):
                        angle = -angle
                    self.dangle = angle
                else:
                    self.total_value += self.dangle
                    self.dangle = 0

```

```

        self.starting_knob = None
        # print("reset")
        self.last_knob = current_knob
        self.last_timestamp = timestamp
    else:
        self.last_knob = None
        self.total_value += self.dangle
        self.dangle = 0
        self.starting_knob = None

    def annotate_image(self, rgb_image):
        annotated_image = np.copy(rgb_image)
        current_value = self.total_value + self.dangle
        if self.starting_knob is not None and self.last_knob is not
None:
            cx = (self.last_knob[0] + self.last_knob[2]) // 2
            cy = (self.last_knob[1] + self.last_knob[3]) // 2
            r = int(((self.last_knob[2] - self.last_knob[0]) ** 2 +
(self.last_knob[3] - self.last_knob[1]) ** 2) ** 0.5 / 2)
            cv2.circle(annotated_image, (cx, cy), r, (0, 0, 255), 2)
            cv2.line(annotated_image, (cx, cy), (self.last_knob[2],
self.last_knob[3]), (0, 0, 255), 2)
            cv2.putText(
                img=annotated_image,
                text=f"Total Value: {current_value:.2f}",
                org=(10, 30),
                fontFace=cv2.FONT_HERSHEY_DUPLEX,
                fontScale=1,
                color=(0, 0, 255),
                thickness=2,
                lineType=cv2.LINE_4,
            )

        return annotated_image

    def detect_knob_gesture(
        rgb_image, detection_result: mp.tasks.vision.HandLandmarkerResult
    ):
        if not hasattr(detection_result, "hand_landmarks"):
            return rgb_image
        # Get Data
        hand_landmarks_list = detection_result.hand_landmarks
        annotated_image = np.copy(rgb_image)

```

```

for hand_landmarks in hand_landmarks_list:

    def get_finger_angle(a, b, c, d):
        dx1 = hand_landmarks[b].x - hand_landmarks[a].x
        dy1 = hand_landmarks[b].y - hand_landmarks[a].y
        dx2 = hand_landmarks[d].x - hand_landmarks[c].x
        dy2 = hand_landmarks[d].y - hand_landmarks[c].y
        return get_angle(dx1, dy1, dx2, dy2)

    thumb_angle = get_finger_angle(1, 2, 3, 4)
    index_angle = get_finger_angle(5, 6, 6, 7)
    open_angle = get_finger_angle(2, 3, 5, 6)
    if (
        thumb_angle < 45
        and index_angle > 20
        and index_angle < 90
        and open_angle < 30
    ):
        # draw circle by diameter from [4] to [8]
        height, width, _ = annotated_image.shape
        x1 = int(hand_landmarks[4].x * width)
        y1 = int(hand_landmarks[4].y * height)
        x2 = int(hand_landmarks[8].x * width)
        y2 = int(hand_landmarks[8].y * height)
        cv2.circle(
            annotated_image,
            ((x1 + x2) // 2, (y1 + y2) // 2),
            int(((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5 / 2),
            (0, 0, 255),
            2,
        )
        cv2.line(
            annotated_image,
            ((x1 + x2) // 2, (y1 + y2) // 2),
            (x2, y2),
            (0, 0, 255),
            2,
        )
    return annotated_image

def main():
    # access webcam
    
```

```
cap = cv2.VideoCapture(0)

# create landmarker
hand_landmarker = HandLandmarker()
ret, frame = cap.read()
knob_recognizer = KnobRecognizer(frame.shape[1], frame.shape[0])

while True:
    # pull frame
    ret, frame = cap.read()
    # mirror frame
    frame = cv2.flip(frame, 1)
    # update landmarker results
    hand_landmarker.detect_async(frame)
    time.sleep(0)
    # draw landmarks on frame
    frame = draw_landmarks_on_image(frame, hand_landmarker.result)

    knob_recognizer.update(hand_landmarker.result,
hand_landmarker.result_timestamp)
    frame = knob_recognizer.annotate_image(frame)

    react = knob_recognizer.dangle

    # display image
    cv2.imshow("frame", frame)
    if cv2.waitKey(1) == ord("q"):
        break

# release everything
hand_landmarker.close()
cap.release()
cv2.destroyAllWindows()

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

参考文献

- [1] Gao, Z., Zhang, S., McLoughlin, I., and Yan, Z., “Paraformer: Fast and Accurate Parallel Transformer for Non-autoregressive End-to-End Speech Recognition”, *<i>arXiv e-prints</i>*, 2022. doi:10.48550/arXiv.2206.08317.
- [2] Xu, H., Han, L., Yang, Q., Li, M., and Srivastava, M., “Penetrative AI: Making LLMs Comprehend the Physical World”, *<i>arXiv e-prints</i>*, 2023. doi:10.48550/arXiv.2310.09605.
- [3] Yao, S., “ReAct: Synergizing Reasoning and Acting in Language Models”, *<i>arXiv e-prints</i>*, 2022. doi:10.48550/arXiv.2210.03629.
- [4] Xu, H., Han, L., Yang, Q., Li, M., and Srivastava, M., “Penetrative AI: Making LLMs Comprehend the Physical World”, *<i>arXiv e-prints</i>*, 2023. doi:10.48550/arXiv.2310.09605.
- [5] Boiko, D. A., MacKnight, R., and Gomes, G., “Emergent autonomous scientific research capabilities of large language models”, *<i>arXiv e-prints</i>*, 2023. doi:10.48550/arXiv.2304.05332.
- [6] Park, J. S., O'Brien, J. C., Cai, C. J., Ringel Morris, M., Liang, P., and Bernstein, M. S., “Generative Agents: Interactive Simulacra of Human Behavior”, *<i>arXiv e-prints</i>*, 2023. doi:10.48550/arXiv.2304.03442.
- [7] Wuhuang Huang, Aijun Chen, Zhixiang Pan, Song Zhang, Kuojun Yang, Duyu Qiu, Peng Ye, Shulin Tian, Houjun Wang, Design of portable high-speed oscilloscope analyzer for multifunctional integrated signal testing, Measurement, Volume 209,2023,112490, ISSN 0263-2241, <https://doi.org/10.1016/j.measurement.2023.112490>.
- [8] 姜斌, 李丰璞, 包建荣, 刘超, and 郭春生, “低成本功能可扩展虚拟示波器研制及验证,” 实验室研究与探索, vol. 38, no. 7, pp. 62 – 67, 2019.
- [9] 孟芳芳, “基于 usb2.0 的一体化虚拟仪器的软件设计与实现,” Thesis, 2007.