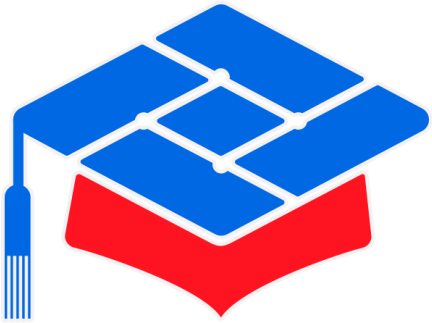


第八届

全国大学生集成电路创新创业大赛

报告类型*：技术报告

参赛杯赛*：飞腾杯

作品名称*：基于飞腾派的视觉机械臂智能分拣系统

队伍编号*：CICC2228

团队名称*：永不言败

目录

一. 应用背景介绍

1.1 系统设计背景

随着科技的迅猛发展，全球各国纷纷将人工智能、机器人技术等前沿领域纳入国家战略发展规划。视觉机器臂技术，作为这一战略中的关键一环，被广泛认为能够显著提升国家的生产力和国际竞争力。

在当今时代，视觉机械臂系统已经成为推动科技创新和社会进步的重要力量。它广泛应用于智能制造、医疗服务、工业生产以及家庭生活等多个领域，为人类社会带来了前所未有的新机遇。通过远程操控和实时交互，视觉机械臂能够在危险环境中安全作业，为医疗领域提供更广泛的服务，提高工业生产的效率与质量，并在日常生活中提供更加智能化的便利。

这些技术背景和社会发展需求的融合，正不断推动视觉机械臂系统的创新与应用，为我们的未来描绘出一幅充满无限可能和便利的图景。

特别是在智能分拣领域，将视觉技术与机械臂的抓取动作相结合，极大地提升了机械臂在各个领域的智能化水平。这不仅使得任务执行更加精准高效，而且为人们的工作和生活带来了更多的便利与安全保障。以物流行业为例，智能分拣系统能够自动识别和分类各种物品，显著提高了分拣速度，同时大幅度减少了人工错误，展现了视觉机械臂技术在实际应用中的卓越潜力。

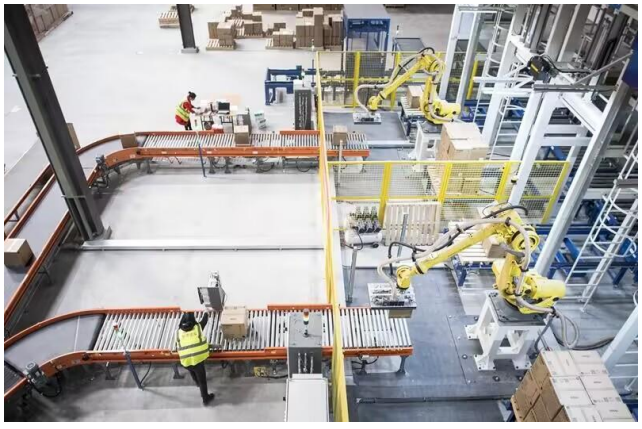


图 1.1 视觉机械臂智能分拣系统

1.2 系统设计思路

飞腾教育开发板基于 ARM 架构，内置四核，在安全性、实时性、可靠性都

有不俗的表现。其支持多核并行处理及核间通信功能，为复杂系统的计算、运行提供了有力的保障。因此，我们选择飞腾开发板作为本系统的处理核心，结合OpenManipulator 机械臂、OAK 智能深度相机、距离传感器等外设，实现视觉机械臂智能分拣系统。

本系统充分利用了飞腾派多核异构的特点，在 CPU0（主核）中实现机械臂的运动控制，在 CPU2（主核）中实现双目深度相机数据的采集，并对识别到的物体进行目标检测和深度计算。同时，本系统在 CPU1（从核）中完成距离传感器数据的采集与处理，利用 OpenAMP 实现与 CPU0（主核）的通信，与 CPU0（主核）中的机械臂运动控制形成联动，实现安全制动等并行处理任务。

此外，机械臂中内置的姿态传感器可以实时检测机械臂运行状态信息，为实现复杂机器人控制系统的精细化管理提供了可靠的数据支撑；系统利用距离传感器，通过设置合适的距离阈值，提供安全制动功能；本项目设计的多核异构和 OpenAMP 核间通信任务，可以充分体现飞腾 CPU 高实时性核和高性能核并行处理任务的能力，进一步提高了系统的应用价值，拓宽了应用场景。

1.3 系统任务目标

本项目开发了一套基于飞腾派开发板的智能视觉机械臂系统，该系统不仅具备高效的抓取和放置功能，还集成了先进的安全制动机制。

系统的任务目标如下：

① 距离感知测量：系统首先通过高精度的 STPL32 测距传感器实时监测物体距离。当物体距离降至预设的安全阈值以下，系统将自动向机械臂发出抓取指令，确保操作的及时性和准确性。

② 深度视觉分析：利用先进的 OAK-D-Lite 深度视觉相机，系统能够精确识别目标物体并实时计算其在 3D 空间中的位置。经过坐标系转换，系统将目标物体的 3D 坐标信息传递给机械臂。

③ 机械臂抓取与移动：在接收到抓取指令和目标物体的 3D 空间位置信息后，系统可以计算出机械臂的最佳运动路径和各关节的最优运动方案，机械臂将根据系统计算的结果执行精确的抓取动作，并沿着预设路径平稳移动，完成目标物品的抓取和移动任务。

④ 设置机械臂的抓取移动操作在 CPU0（主核）上进行，双目相机的目标检测和深度计算操作在 CPU2（主核）上进行，STP-23L 距离传感器的数据采集在 CPU1（从核）上进行。

⑤ 在 OpenAMP 核间通信的部分，CPU1（从核）实时高性能地采集 STP-23L 距离传感器的数据，并利用 OpenAMP 将采集到的距离数据传递给 CPU0（主核）；CPU0（主核）在获取到距离数据后，进一步地处理分析，将结果通过 OpenAMP 再传递给 CPU1（从核），CPU1（从核）根据分析结果提供安全制动功能。

1.4 系统性能指标

1.目标检测：

①准确率（Accuracy）：正确识别目标的比例。

②召回率（Recall）：系统检测到的所有目标中正确识别的比例。

③精确度（Precision）：系统识别为正确的目标中实际正确的比例。

④F1 分数（F1 Score）：精确度和召回率的调和平均值。

⑤mAP(Mean Average Precision)：平均精确率的平均。

⑥FPS：反映目标检测的速度和实时性。

2.深度计算：

①平均绝对误差（Mean Absolute Error, MAE）：预测深度和实际深度之间的平均绝对差异。

②平均平方误差（Mean Squared Error, MSE）：预测深度和实际深度之间的平均平方差异。

3.核间通信时间：

①核间通信时间：主从核收发数据通信所需的时间，包括平均通信时间、最大通信时间和最小通信时间。

4.机械臂运动响应时间：

①机械臂运动响应时间：即从机械臂接收到运动指令开始，抵达不同目标位置所需的时间。

二. 系统工作原理与关键技术原理分析

2.1 U2D2 通信控制机械臂

对于开发板与 OpenManipulator 机械臂之间的通信，我们采用 U2D2 通信转换器和 Power Hub Board 连接，其中 U2D2 通信模块负责 USB 到 UART 的数据转换，而电源集线板用于管理和提供系统所需的电力。通过将这两个组件结合在一起，可以实现在开发板或机械臂控制系统中同时管理通信和电源供应的功能，这种组合可以提高系统的整体效率和可靠性。

我们使用 U2D2 + 电源集线板控制机械臂时，采取了以下步骤：

1.在飞腾派上安置系统：利用 Phytium-Linux-buildroot，通过交叉编译生成支持 OPENAMP 的 20.04 版本 Ubuntu 系统。

1.连接 U2D2 与开发板：通过 USB 接口将 U2D2 连接到飞腾派上，以便进行数据通信。

2.连接电源集线板：将电源集线板连接到电源，为机械臂系统提供稳定的电源。

3.连接机械臂控制器：将 U2D2 连接到机械臂控制器，以使计算机可以通过 U2D2 向机械臂发送控制指令和接收反馈数据。

4.编写控制程序：在飞腾派上编写控制程序，通过 U2D2 与机械臂进行通信和控制。这可以包括控制机械臂的运动、速度、位置和其他参数。

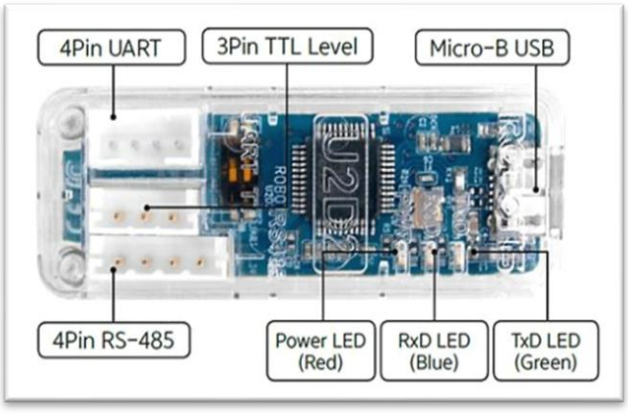


图 2.1 U2D2 转换器

2.2 ROS 系统

机器人操作系统（Robot Operating System，ROS）是一个适用于机器人的开源的元操作系统。ROS 提供了操作系统应有的服务，包括硬件抽象，底层设备控制，常用函数的实现，进程间消息传递，以及包管理。它也提供用于获取、编译、编写、和跨计算机运行代码所需的工具和库函数。在某些方面 ROS 相当于一种“机器人框架（robot frameworks）”，类似的“机器人框架”有：Player，YARP，Orocos，CARMEN，Orca，MOOS 和 Microsoft Robotics Studio 等。

ROS 运行时的“蓝图”是一种基于 ROS 通信基础结构的松耦合点对点进程网络。ROS 实现了几种不同的通信方式，包括基于同步 RPC 样式通信的服务（services）机制，基于异步流媒体数据的话题（topics）机制以及用于数据存储的参数服务器（Parameter Server）。

ROS 并不是一个实时的框架，但 ROS 可以嵌入实时程序。Willow Garage 的 PR2 机器人使用了一种叫做 pr2_etherCAT 的系统来实时发送或接收 ROS 消息。ROS 也可以与 Orocos 实时工具包无缝集成。除此之外，ROS 还有如下特点：

- 小型化：ROS 尽可能设计的很小——不封装 main()函数，所以为 ROS 编写的代码可以轻松的在其它机器人软件平台上使用，可以轻松集成在其它机器人软件平台：ROS 已经可以与 OpenRAVE，Orocos 和 Player 集成。
- 模型独立：ROS 的首选开发模型都是用不依赖 ROS 的干净的库函数编写而成。
- 语言独立：ROS 框架可以简单地使用任何的现代编程语言实现。目前已经实现了 Python 版本，C++版本和 Lisp 版本。同时，ROS 也拥有 Java 和 Lua 版本的实验库。
- 方便测试：ROS 内建一个了叫做 rostest 的单元/集成测试框架，可以轻松安装或卸载测试模块。
- 可扩展：ROS 可以适用于大型运行时系统和大型开发进程。

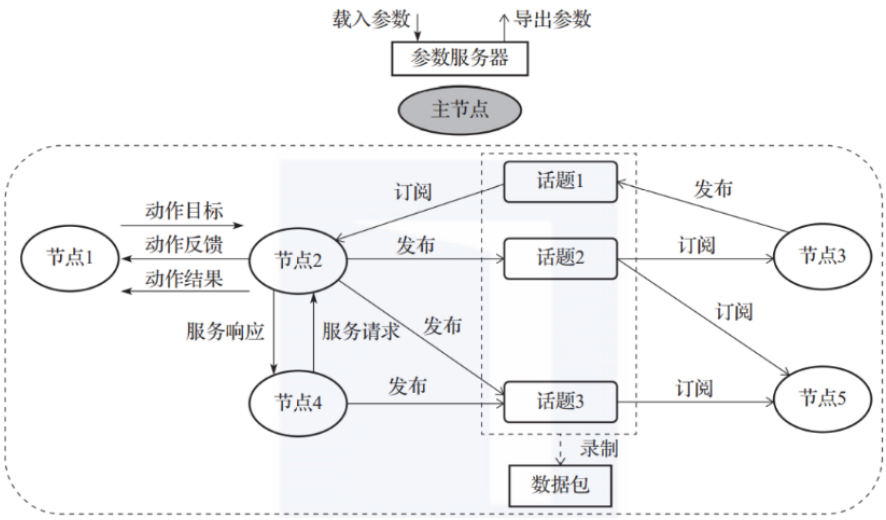


图 2.2 ROS 的计算图结构示意图

ROS 目前兼容性最好的仍然是 Linux 的 Ubuntu 系统，且 Ubuntu 系统版本和 ROS 的版本需要一一对应。由于本项目使用的操作系统是 Ubuntu 20.04，因此需要安装 ROS-Noetic 版本，以此在飞腾派上实现机械臂运动控制功能，处理如运动控制、轨迹监控、断点恢复和联机控制等任务。

2.3 目标检测 YOLO 算法

目标检测（Object Detection）是计算机视觉领域的重要任务之一，旨在识别图像或视频中的特定目标并将其位置标记出来。与图像分类任务不同，目标检测要求不仅能够识别目标类别，还需要精确地定位目标的位置。由于各类物体有不同的外观、形状和姿态，加上成像时光照、遮挡等因素的干扰，目标检测一直是计算机视觉领域最具有挑战性的问题。

YOLO（You Only Look Once）是一种对象检测算法，由 Joseph Redmon 等人于 2015 年提出。YOLO 的核心思想是将对象检测任务转化为一个回归问题，通过一个卷积神经网络直接在图像上进行推理，实现实时对象检测。

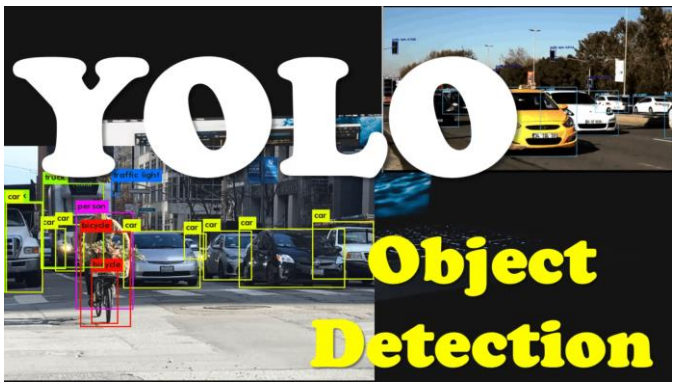


图 2.3 YOLO 目标检测

具体来说，YOLO 算法主要包括以下几个步骤：

①图像划分：将输入图像划分为 S×S 的网格，每个网格负责预测 B 个边界框以及这些边界框的置信度。置信度表示边界框内存在目标的概率以及边界框的准确度。

②特征提取：利用卷积神经网络提取图像特征。这些特征通常包括颜色、纹理、形状等多种信息，对于后续的目标检测至关重要。

③边界框预测：每个网格根据提取到的特征预测 B 个边界框的坐标（x，y，w，h）以及置信度。其中，(x，y)表示边界框中心的坐标，(w，h)表示边界框的宽度和高度。

④类别预测：每个网格还会预测 C 个类别的概率。这些概率表示该网格内存在不同类别目标的可能性。

⑤非极大值抑制：在得到所有网格的预测结果后，通过非极大值抑制算法去除冗余的边界框，保留最佳的检测结果。

自 YOLO 算法提出以来，经过多个版本的迭代和优化，其性能得到了不断提升：

①YOLOv1：YOLO 算法的初始版本。它通过将目标检测任务转化为回归问题，实现了快速的目标检测。然而，由于每个网格只能预测固定数量的边界框，导致对于一些密集或尺寸变化较大的目标检测效果不佳。

②YOLOv2：在 YOLOv1 的基础上进行了多项改进。首先，引入了批量归一化（Batch Normalization）和残差网络（Residual Network）等技巧，提升了模型的训练速度和稳定性。其次，采用了多尺度训练策略，增强了模型对不同尺寸目标的处理能力。此外，还引入了锚框（Anchor Box）机制，提高了边界框预测

的准确性。

③YOLOv3：进一步提升了 YOLO 算法的性能。它采用了更深的卷积神经网络结构（Darknet-53），并引入了特征金字塔网络（Feature Pyramid Network）来融合不同尺度的特征信息。这些改进使得 YOLOv3 在保持高速度的同时，进一步提高了检测的准确性。

④YOLOv4：在 YOLOv3 的基础上进行了更多的优化和创新。它引入了数据增强（Data Augmentation）、自适应锚框（Adaptive Anchor Box）等技巧，进一步提升了模型的泛化能力和边界框预测的准确性。同时，还采用了更高效的硬件加速策略，使得 YOLOv4 在实际应用中具有更高的实用价值。

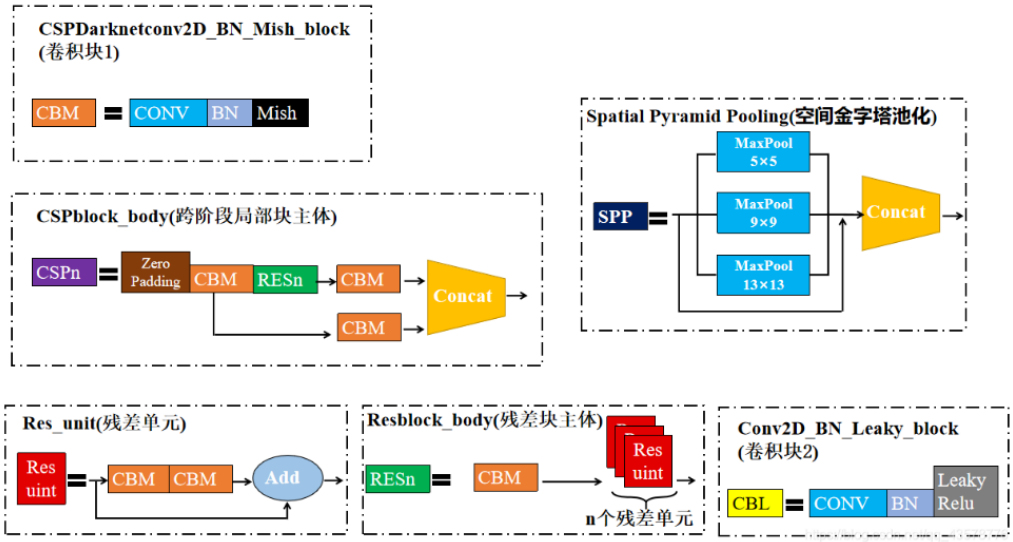


图 2.4 YOLOv4 的网络结构

2.4 双目立体视觉相机的深度信息计算

基于双目立体视觉的深度相机类似人类的双眼，它和基于 TOF、结构光原理的深度相机不同，并不对外主动投射光源，而是完全依靠拍摄的两张图片（彩色 RGB 或者灰度图）来计算深度，因此有时候也被称为被动双目深度相机。

双目立体视觉相机的深度计算原理如下：

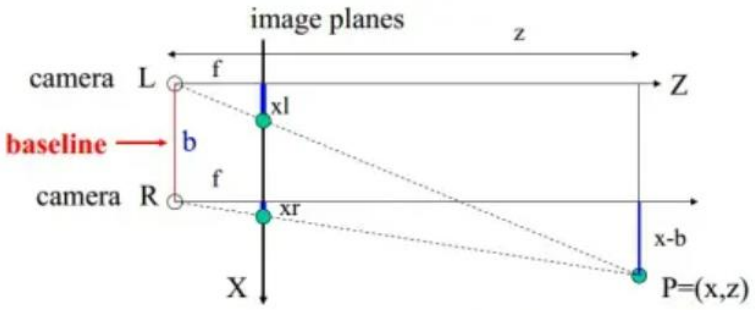


图 2.5 相机视觉深度计算原理图

对于已知的相机焦距 f 和左右相机基线 b，可以利用极线约束得到左右相机的视差 d，也就是得到左相机的每个像素点(x1，y1)和右相机中对应点(xr，yr)之间的对应关系。

其中，所谓极线约束就是指当同一个空间点在两幅图像上分别成像时，已知左图投影点 p1，那么对应右图投影点 p2 一定在相对于 p1 的极线上，这样可以极大的缩小匹配范围。

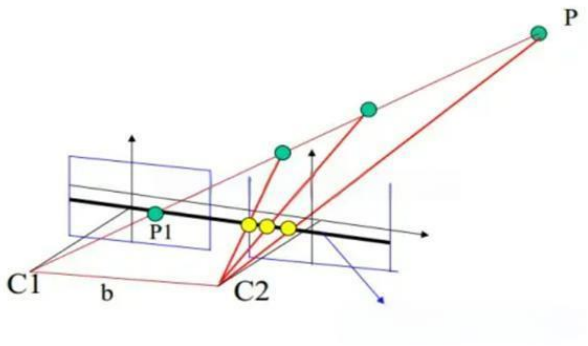


图 2.6 级线约束原理

在本项目中，我们利用双目立体视觉相机可以得到目标物体的深度信息，并通过 ROS 话题发布的形式反馈给机械臂，从而让机械臂精确抓取目标物体。

2.5 手眼标定

手眼标定是确定像素坐标系和空间机械手坐标系的坐标转化关系。在实际控制中，双目相机检测到目标在图像中的像素位置后，通过标定好的坐标转换矩阵将相机的像素坐标变换到机械手的空间坐标系中，然后根据机械臂坐标系计算出各个电机该如何运动，从而控制机械臂到达指定位置。

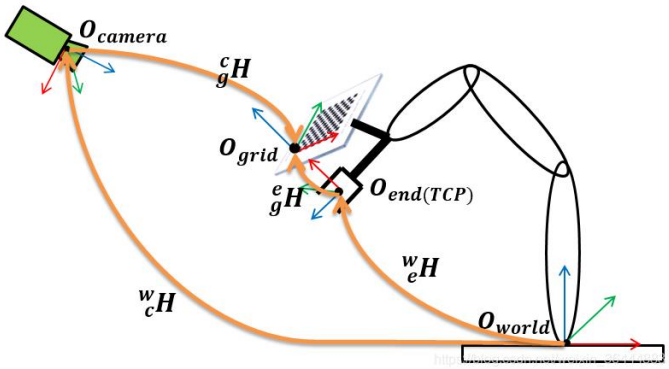


图 2.7 手眼标定（eye-to-hand）

本项目的相机装载位置为 eye-to-hand，即相机固定在一个地方，机械手的运动不会带着相机一起移动。

手眼标定的基本原理可以概括如下：

①标定目标：使用一个已知几何形状的标定板，比如棋盘格或圆点阵列，作为标定目标。

②图像获取：使用摄像头捕捉标定板的图像。

③特征提取：从图像中提取标定板上的特征点，这些点在标定板上的位置是已知的。

④机器人操作：将标定板放置在机器人的工作空间内，并记录机器人末端执行器（手部）的位置和方向。

⑤特征匹配：将摄像头图像中的特征点与标定板上的实际位置进行匹配。

⑥建立模型：利用特征点在图像中的位置和在机器人坐标系中的位置，建立一个数学模型来描述摄像头与机器人手部之间的几何关系。

⑦求解参数：通过最小二乘法或其他优化算法求解模型中的参数，这些参数包括旋转矩阵和平移向量，它们描述了摄像头相对于机器人手部的位置和方向。

⑧验证与应用：使用求解得到的参数进行验证，并将其应用于实际的视觉引导任务中。

手眼标定的数学模型通常可以表示为：$Tee=Twc\cdot TceTee=Twc\cdot Tce$ 其中：

- $TeeTee$ 是机器人手部到摄像头的变换矩阵。
- $TwcTwc$ 是标定板在世界坐标系中的位置和方向的变换矩阵。
- $TceTce$ 是标定板在摄像头坐标系中的位置和方向的变换矩阵。

变换矩阵 TT 通常由旋转矩阵 RR 和平移向量 tt 组成： $T=[R|t]T=[R|t]$

手眼标定的精确性对于机器人视觉应用至关重要，它确保了机械臂在操作时能够准确地感知和响应外部环境的变化，从而实现更高效的自动化操作。

2.6 TOF 测距原理

TOF（Time-Of-Flight）光电技术是一种用于测量物体距离的技术。它基于光飞行时间的原理，通过发射一定波长的光并测量光从发射到被物体反射回来的时间，从而确定物体与光源之间的距离。

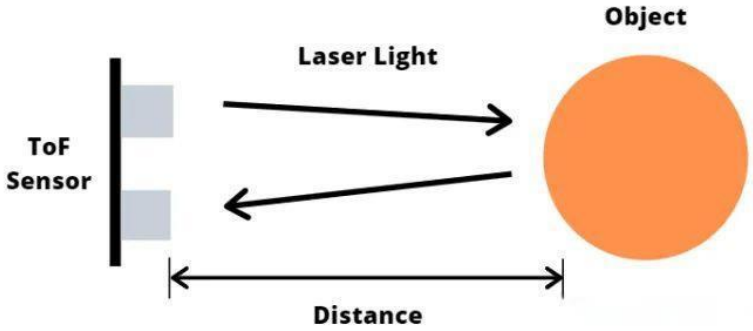


图 2.8 TOF 测距原理图

TOF 传感器能够在短时间内准确地检测物体，并且不受湿度，气压和温度的影响，使其适合于室内和室外使用，同时其具有较高的灵活性，因为它们能够检测各种形状和大小的近距离和远距离物体。

TOF 技术通过测量光脉冲从发射到接收的时间差，可以精确计算出目标物体与传感器之间的距离。这种技术在工业自动化中至关重要，特别是在需要高精度定位和导航的应用中。

2.7 机械臂运动姿态检测

机械臂内嵌了多个姿态传感器模块，其可以用以追踪机械臂的运动轨迹，包含三轴加速度与三轴陀螺仪数据。

三轴加速度传感器是一种基于微机电系统（MEMS）技术的传感器，它能够检测物体在三个方向上的加速度变化。其工作原理是通过微型质量块和微型弹簧，当被测物体发生加速度变化时，微型质量块会相对于传感器的结构产生微小的位移。通过测量这种位移变化，传感器可以确定物体在三个方向上的加速度变化情

况。

而三轴陀螺仪传感器是一种能够测量物体在三个方向上旋转速率或角度变化的传感器。它通常利用旋转惯性的原理来检测物体的旋转，通过测量角速度或角度变化，传感器可以确定物体在三个轴向上的旋转情况。

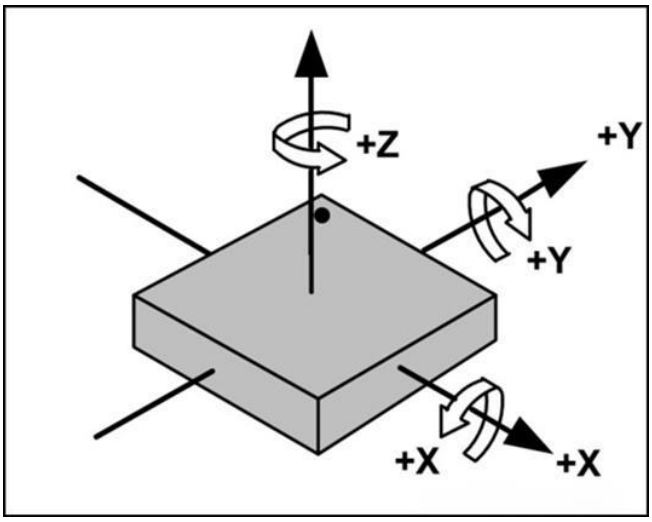


图 2.9 三轴陀螺仪

2.8 多核异构及 OpenAMP 核间通信

随着硬件技术的快速发展，嵌入式芯片的性能日益强大，由单核向多核、再到异构多核的演进，为在单一片上系统（SoC）中同时运行多个操作系统提供了稳固的基础。多核混合部署旨在在多核芯片上同时运行多个操作系统镜像，包括裸机（Baremetal）、高实时性系统（FreeRTOS）和 Linux 等多种实现形式。每种形式都有其独特的特点，例如裸机可实现最佳性能，而嵌入式 Linux 系统则具有更好的易用性和灵活性。

本项目中，CPU0 和 CPU2 作为主核，部署 Linux 操作系统，方便借助 Linux 系统的易用性和丰富的软件资源，实现机械臂运动控制功能（CPU0）、双目相机的目标检测和深度计算功能(CPU2)、以及 ROS 结点间通信传输功能;CPU1 和 CPU3 作为从核，部署高实时系统（FreeRTOS），用于各类传感器数据获取，提供安全制动功能

OpenAMP（Open Asymmetric Multi-Processing）是一种开放式的异构多核处理器架构，旨在实现在多核处理器上运行不同操作系统的需求。它提供了一种通用的方法，使得不同类型的操作系统能够在同一片上系统(SoC)上协同工作。

通过 OpenAMP，用户可以同时运行裸机、实时操作系统（FreeRTOS）和通用操作系统（如 Linux）等不同类型的操作系统，以满足各种应用场景下的需求。

本项目中，CPU1（从核）实时高性能地采集 STP-23L 距离传感器的数据，并利用 OpenAMP 将采集到的距离数据传递给 CPU0（主核）；CPU0（主核）在获取到距离数据后，进一步地处理分析，将结果通过 OpenAMP 再传递给 CPU1（从核），CPU1（从核）根据分析结果提供安全制动功能。

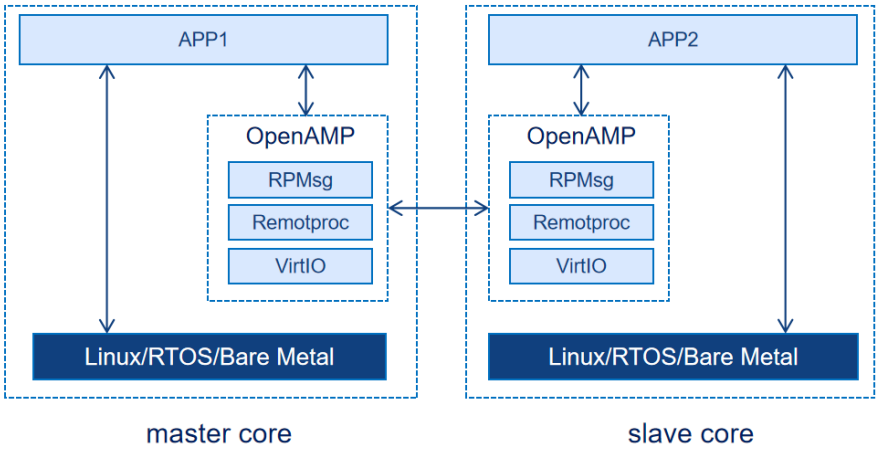


图 2.10 多核异构及 OpenAMP 核间通信

三． 系统体系结构设计

3.1 硬件整体架构

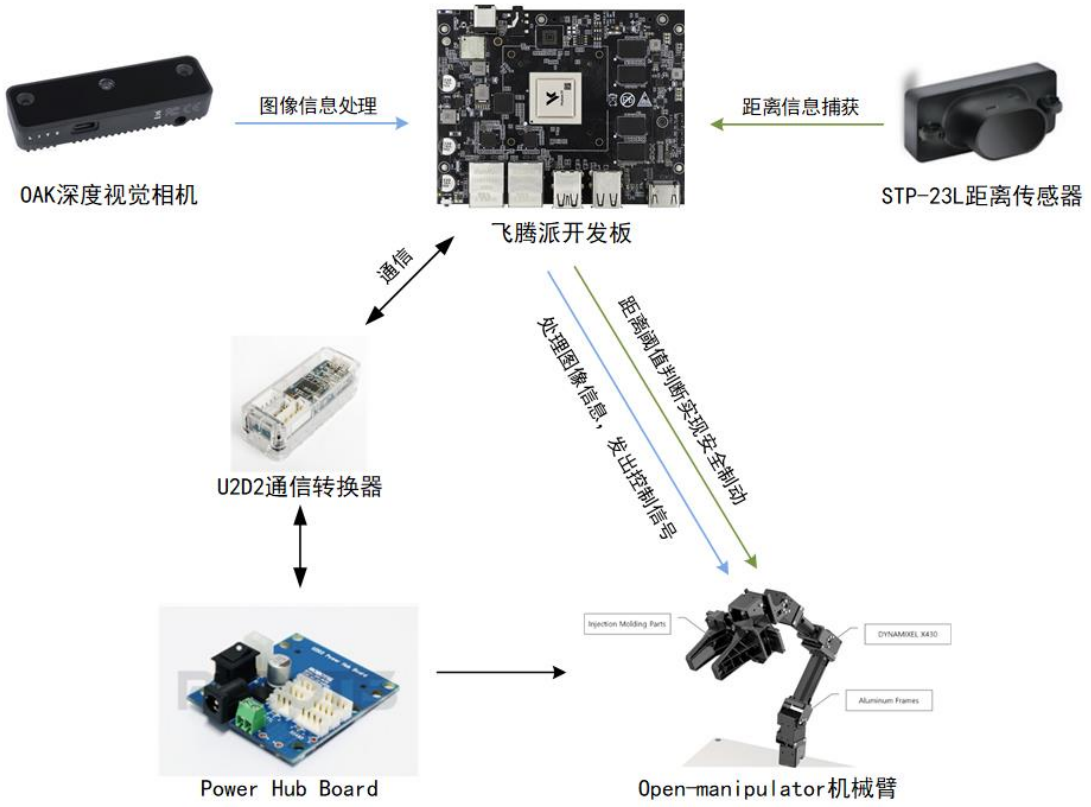


图 3.1 硬件系统设计

本系统以飞腾教育开发板为处理核心，通过 U2D2 及 Power Hub Board 模块连接控制 OpenManipulator 机械臂运动。飞腾教育开发板接收 OAK 深度视觉相机的图像处理结果，经过分析后派遣机械臂执行操作。得益于飞腾教育开发板的高实时性核和高性能核并行处理任务的能力，我们的系统能够达到实时相应的要求，从而能够进一步的为下游任务控制机械臂的一系列运动提供基础条件。

3.2 硬件介绍

3.2.1 飞腾派开发板

板卡正面：

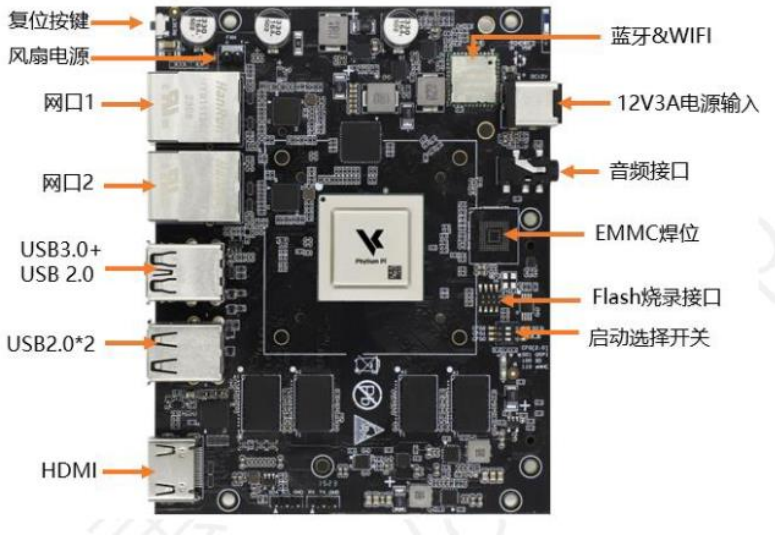


图 3.2 板卡正面图

板卡背面：

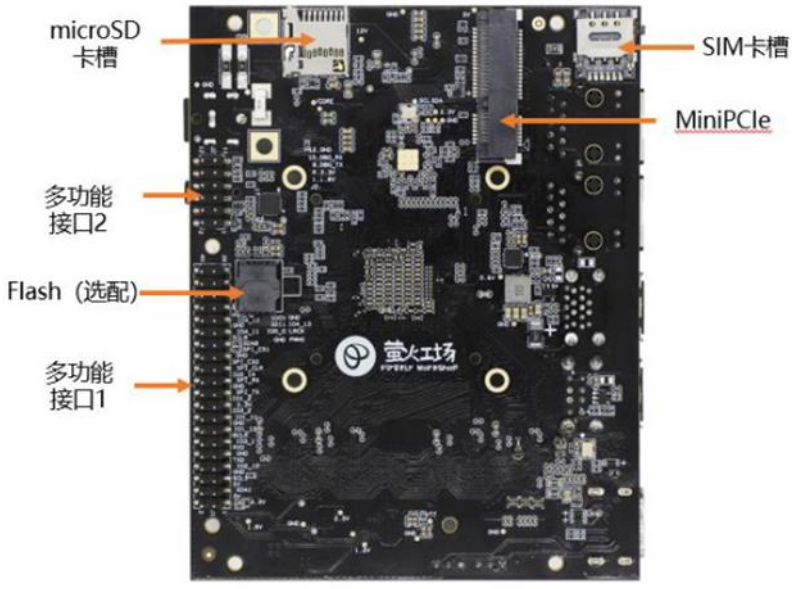


图 3.3 板卡背面图

飞腾派开发板是由萤火工场研发的一款面向行业工程师、学生和爱好者的开源硬件，主板处理器采用飞腾嵌入式四核处理器，该处理器兼容 ARM V8 指令集，包含 2 个 FTC664 核和 2 个 FTC310 核，其中 FTC664 核主频可达 1.8GHz，FTC310 核主频可达 1.5GHz。主板板载 64 位 DDR4 内存，分 2G 和 4G 两个版本，支持 SD 或者 eMMC 外部存储。主板板载 Wifi 蓝牙，陶瓷天线，可快速连

接无线通信。另外还集成了大量外设接口,包括双路千兆以太网、USB.UART、CAN、HDMI、音频等接口，集成一路 miniPCIE 接口，可实现 A1 加速卡与 4G、5G 通信等多种功能模块的扩展。

主板操作系统支持 Ubuntu、Debian 等国外主流开源操作系统，也支持国内 Openkylin、OpenHarmony、SylixOS、RT-Thread 等国产操作系统。

3.2.2 U2D2 + Power Hub Board

我们采用 U2D2 通信转换器与 Power Hub Board 的组合方案，以实现板卡与机械臂间的高效通信。这一系统不仅能够将电脑或控制器发出的数据信号转换为 Dynamixel 系列舵机所需的特定信号，还具备完善的电源管理功能。这种组合常用于机器人控制、教育和研究领域，能够实现对舵机的控制和监测。其主要功能包括数据信号转换、电源管理以及舵机控制和监测，适用于各种机器人应用和实验项目。

图 3.4 U2D2

图 3.5 Power Hub Board

3.2.3 Open Manipulator 机械臂

Open Manipulator 是一个开源的机械臂项目，它旨在提供一个灵活、可定制的机械臂解决方案，可用于各种应用，如教育、研究和工业。该项目由 Robotis（韩国机器人公司）开发，采用 ROS（机器人操作系统）作为其软件框架。

Open Manipulator 机械臂具有以下特点：

①模块化设计：Open Manipulator 机械臂采用模块化设计，可方便地进行定制和扩展。它由多个关节组成，每个关节都具有独立的控制。

②开放式软件框架：Open Manipulator 机械臂使用 ROS 作为其软件框架，这使得用户可以利用 ROS 生态系统中的丰富资源，如导航、感知和运动规划等功能。

③易于集成：Open Manipulator 提供了丰富的文档和示例代码，使得用户可以轻松地将其集成到自己的项目中。

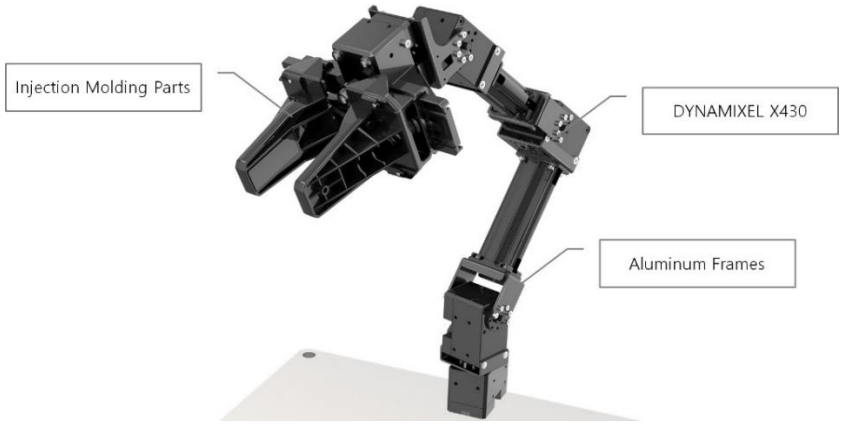


图 3.6 Open Manipulator 机械臂

Open Manipulator-X 机械臂的硬件参数如下：

参数	OpenManipulator-X
驱动器	Dynamixel XM430-W350-T
输入电压	12V
自由度	5 （4 个转动关节 + 1 个夹爪）
最大负载	500g
重复定位精度	±0.2mm
最大关节转速	46 RPM
重量	0.7 kg
可达范围	380mm
夹爪行程	20～75mm
通讯协议	TTL 电平多点总线
软件支持	ROS、Dynamixel SDK，Arduino，Processing
主控器	PC、OpenCR

图 3.7 Open Manipulator 机械臂硬件参数表

3.2.4 OAK D Lite 双目深度相机

OAK D Lite 是一款由 OpenCV 团队开发的小型深度学习计算机视觉套件。包括一个 1300 万高像素 RGB 摄像头和全局快门双目深度摄像头模块，其有 1.4T 的 AI 算力，4K H.265 推流，及厘米级测量精度，支持 H264、h.265、mjpeg- 4k/30fps，1080p/60fps 编码，结合了 AI 推理和计算机视觉功能，提供了高性能的图像处理和深度学习推理能力。OAK D Lite 采用了低功耗、高性能的神经网络加速器，可在嵌入式系统中快速、高效地执行深度学习模型，从而实现实时的目标检测、人脸识别、姿态估计等任务。其小巧的设计和强大的功能使其成为智能摄像头、机器人、智能监控等应用场景的理想选择。

图 3.8 OAK D Lite 深度视觉相机

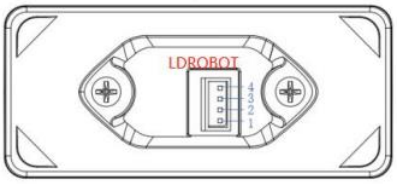
参数	RGB相机	双目相机
图像传感器（Sensor）	IMX214	OV7251
DFOV / HFOV / VFOV	81°D / 69°H / 54°V	86° D/ 73° H/ 58°V
分辨率	13MP (4208×3120)	480P (640×480)
最大帧率	35 FPS	120 FPS
焦距（EFL）	3.37mm	1.3mm
光圈（F.NO）	2.2 ± 5%	2.0± 5%
对焦范围	AF: 8cm – ∞ FF: 50cm – ∞	FF：6.5cm – ∞
镜头尺寸	1/3.1"	1/7.5"
像素大小	1.12μm x 1.12μm	3μm x3μm
快门	卷帘快门	全局快门

图 3.9 OAK D Lite 相机参数表

3.2.5 STP-23L 距离传感器

STP-23L 距离传感器是一款高精度激光测距模块，可实现对物体的位置进行毫米级准确测量。其采用先进的 TOF 测距原理，在实时测量物体位置上具有更高的稳定性与更好的抗光性，最大测距距离达到 13.4m，采样频率高达 4500Hz。此外其全面兼容 ROS1 和 ROS2 系统，更适配我们的机械臂控制模块。

图 3.10　STP-23L 距离传感器



序号	信号名	类型	描述	最小值	典型值	最大值
1	Tx	输出	UART TX	0V	3.3V	3.5V
2	RX	输入	UART RX	0V	-	3.3V
3	GND	供电	电源负极	-	0V	-
4	P5V	供电	电源正极	4.5V	5V	5.5V

图 3.11 传感器接口表

在本项目中，我们采用 STP-23L 测距传感器来检测机械臂的运动情况，并于实际预设情况进行比较，进行差错纠正功能的实现。

3.3　软件整体架构

3.3.1 操作系统

本项目主核采用了 Linux 系统的 20.04 版本的 Ubuntu 系统，相对于其他操作系统，Linux 系统具有开放源代码，用户可以自由地查看、修改和分发系统内核和大部分软件，与此同时它具有丰富的软件资源库，这带来了更高的灵活性和可定制性，为用户实现特定功能提供了很好的代码基础。Linux 系统还对系统资源的管理较为高效，可以更好地利用硬件资源，使得在相同硬件条件下，Linux 系统能够提供更好的性能。对于飞腾派来说，主核上部署 Linux 系统可以更好的使用核内资源，充分发挥主核的高性能性，也为我们实现各种图像处理算法和机器

人遥操控任务提供了很好的软件基础。

对于 20.04 版本的 Ubuntu 系统，它可以保障用户在 Linux 系统上下载 ROS Noetic 资源库，同时 Ubuntu 致力于提供简单、易用的用户界面，使得初学者和普通用户能够轻松上手。它采用了直观的桌面环境，并且拥有大量的图形化工具和应用程序,为我们操作机器人提供了简洁的平台。

本项目从核采用了 FreeRTOS 操作系统，它与传统的虚拟化环境不同，没有虚拟化层和中间件，可以直接访问硬件资源，因此性能更高，这使得 FreeRTOS 操作系统在需要高性能、低延迟和实时响应的应用场景下很有用。对于本项目，在从核上部署 FreeRTOS 并外接传感器设备，可以更好的获取机械臂的环境信息及实时运动参数，接着通过 OPENAMP 核间通信技术，实时对数据进行处理，完成轨迹监控，运动控制等任务。

3.3.2 程序整体流程

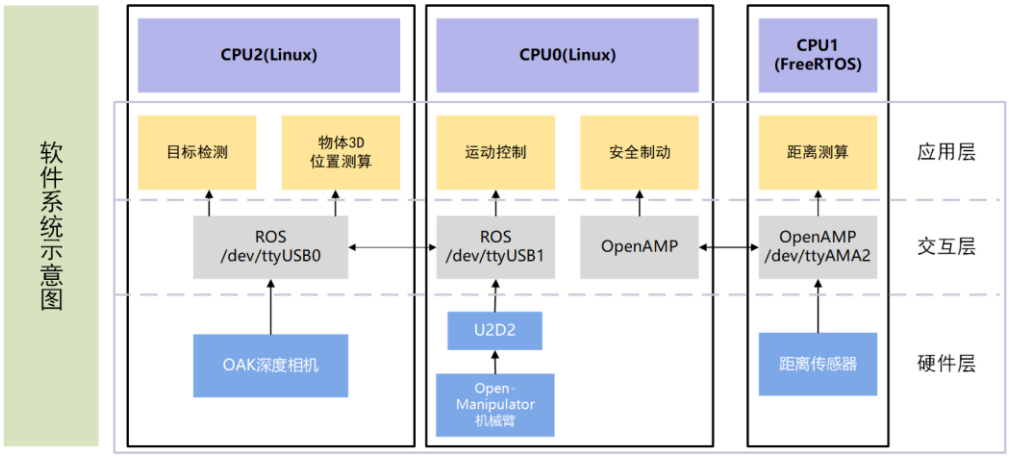


图 3.12 程序软件系统示意图

四． 详细设计与实现

4.1 基于双目相机的目标检测与深度信息的获取

4.1.1 双目相机识别目标物体

本项目所选用的相机为双目相机，原始图像数据流并未对图像进行目标检测。因此需要使用目标检测模型对输入的图像数据流进行处理，得到目标检测结果。我们选用的目标检测模型为 YOLOv4-Tiny，它是一种轻量级的目标检测模型，是 YOLOv4 的简化版本，专为移动设备和嵌入式系统等环境设计，具有高检测速度、高精度保持的特点，非常适合实时的目标检测任务。

核心代码如下：

```
from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
import time

nnPath = str((Path(__file__).parent / Path('../models/yolo-v4-tiny-tf_openvino_2021.4_6shave.blob')).resolve().absolute())

# tiny yolo v4
labelMap = [
    "person",         "bicycle",    "car",           "motorbike",
    "aeroplane",   "bus",           "train",
    "truck",          "boat",       "traffic light", "fire hydrant",
    "stop sign",   "parking meter", "bench",
    "bird",           "cat",        "dog",           "horse",
    "sheep",       "cow",           "elephant",
    "bear",           "zebra",      "giraffe",       "backpack",
    "umbrella",    "handbag",       "tie",
    "suitcase",       "frisbee",    "skis",          "snowboard",
    "sports ball", "kite",          "baseball bat",
    "baseball glove", "skateboard", "surfboard",     "tennis racket",
    "bottle",      "wine glass",    "cup",
    "fork",           "knife",      "spoon",         "bowl",
    "banana",      "apple",         "sandwich",
    "orange",         "broccoli",   "carrot",        "hot dog",
    "pizza",       "donut",         "cake",
```

```
    "chair",          "sofa",       "pottedplant",   "bed",
"diningtable", "toilet",        "tvmonitor",
    "laptop",         "mouse",      "remote",        "keyboard",
"cell phone",  "microwave",     "oven",
    "toaster",        "sink",       "refrigerator",  "book",
"clock",       "vase",          "scissors",
    "teddy bear",     "hair drier", "toothbrush"
]

syncNN = True

# 创建管道
pipeline = dai.Pipeline()

# 定义输入输出
camRgb = pipeline.create(dai.node.ColorCamera)
detectionNetwork = pipeline.create(dai.node.YoloDetectionNetwork)
xoutRgb = pipeline.create(dai.node.XLinkOut)
nnOut = pipeline.create(dai.node.XLinkOut)

xoutRgb.setStreamName("rgb")
nnOut.setStreamName("nn")

camRgb.setPreviewSize(416, 416)
camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1
080_P)
camRgb.setInterleaved(False)
camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
camRgb.setFps(40)

# 检测网络设置
detectionNetwork.setConfidenceThreshold(0.5)
detectionNetwork.setNumClasses(80)
detectionNetwork.setCoordinateSize(4)
detectionNetwork.setAnchors([10, 14, 23, 27, 37, 58, 81, 82, 135, 169
, 344, 319])
detectionNetwork.setAnchorMasks({"side26": [1, 2, 3], "side13": [3, 4
, 5]})
detectionNetwork.setIouThreshold(0.5)
detectionNetwork.setBlobPath(nnPath)
detectionNetwork.setNumInferenceThreads(2)
detectionNetwork.input.setBlocking(False)

# 连接
```

```
camRgb.preview.link(detectionNetwork.input)
if syncNN:
    detectionNetwork.passthrough.link(xoutRgb.input)
else:
    camRgb.preview.link(xoutRgb.input)

detectionNetwork.out.link(nnOut.input)

# 连接设备，开启管道
with dai.Device(pipeline) as device:

    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)

    frame = None
    detections = []
    startTime = time.monotonic()
    counter = 0
    color2 = (255, 255, 255)

    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        color = (255, 0, 0)
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), color, 2)

        cv2.imshow(name, frame)
```

4.1.2 双目相机获取目标物体的深度信息

在双目相机进行目标检测的同时，也会生成二维视差图，其中每个像素的值

代表从相机到场景中相应点的水平距离。对于目标检测框内的所有像素点，可以计算它们在视差图中的视差值，使用相机的内部和外部参数将视差值转换为实际深度，进而得到目标检测框中每个像素点的深度信息，通过计算目标检测框内所有像素点的深度值的加权均值，得到目标物体最终的深度值。

核心代码如下：

```
import cv2
import depthai as dai
import numpy as np
stepSize = 0.05

newConfig = False
pipeline = dai.Pipeline()

monoLeft = pipeline.create(dai.node.MonoCamera)
monoRight = pipeline.create(dai.node.MonoCamera)
stereo = pipeline.create(dai.node.StereoDepth)
spatialLocationCalculator = pipeline.create(dai.node.SpatialLocationCalculator)

xoutDepth = pipeline.create(dai.node.XLinkOut)
xoutSpatialData = pipeline.create(dai.node.XLinkOut)
xinSpatialCalcConfig = pipeline.create(dai.node.XLinkIn)

xoutDepth.setStreamName("depth")
xoutSpatialData.setStreamName("spatialData")
xinSpatialCalcConfig.setStreamName("spatialCalcConfig")

monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoLeft.setCamera("left")
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoRight.setCamera("right")

stereo.setDefaultProfilePreset(dai.node.StereoDepth.PresetMode.HIGH_DENSITY)
stereo.setLeftRightCheck(True)
stereo.setSubpixel(True)

topLeft = dai.Point2f(0.4, 0.4)
bottomRight = dai.Point2f(0.6, 0.6)

```

```
config = dai.SpatialLocationCalculatorConfigData()
config.depthThresholds.lowerThreshold = 100
config.depthThresholds.upperThreshold = 10000
calculationAlgorithm = dai.SpatialLocationCalculatorAlgorithm.MEDIAN

config.roi = dai.Rect(topLeft, bottomRight)

spatialLocationCalculator.inputConfig.setWaitForMessage(False)
spatialLocationCalculator.initialConfig.addROI(config)


monoLeft.out.link(stereo.left)
monoRight.out.link(stereo.right)

spatialLocationCalculator.passthroughDepth.link(xoutDepth.input)
stereo.depth.link(spatialLocationCalculator.inputDepth)

spatialLocationCalculator.out.link(xoutSpatialData.input)
xinSpatialCalcConfig.out.link(spatialLocationCalculator.inputConfig)

with dai.Device(pipeline) as device:

    depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
    spatialCalcQueue = device.getOutputQueue(name="spatialData", maxSize=4, blocking=False)
    spatialCalcConfigInQueue = device.getInputQueue("spatialCalcConfig")

    color = (255, 255, 255)

    while True:
        inDepth = depthQueue.get()
        depthFrame = inDepth.getFrame()
        depth_downscaled = depthFrame[::4]
        if np.all(depth_downscaled == 0):
            min_depth = 0
        else:
            min_depth = np.percentile(depth_downscaled[depth_downscaled != 0], 1)
        max_depth = np.percentile(depth_downscaled, 99)
        depthFrameColor = np.interp(depthFrame, (min_depth, max_depth), (0, 255)).astype(np.uint8)
```

```
70.         depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLO
    RMAP_HOT)
71.
72.         spatialData = spatialCalcQueue.get().getSpatialLocations()
73.         for depthData in spatialData:
74.             roi = depthData.config.roi
75.             roi = roi.denormalize(width=depthFrameColor.shape[1], hei
    ght=depthFrameColor.shape[0])
76.             xmin = int(roi.topLeft().x)
77.             ymin = int(roi.topLeft().y)
78.             xmax = int(roi.bottomRight().x)
79.             ymax = int(roi.bottomRight().y)
80.
81.             depthMin = depthData.depthMin
82.             depthMax = depthData.depthMax
83.
84.             fontType = cv2.FONT_HERSHEY_TRIPLEX
85.             cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax)
    , color, 1)
86.             cv2.putText(depthFrameColor, f"X: {int(depthData.spatialC
    oordinates.x)} mm", (xmin + 10, ymin + 20), fontType, 0.5, color)
87.             cv2.putText(depthFrameColor, f"Y: {int(depthData.spatialC
    oordinates.y)} mm", (xmin + 10, ymin + 35), fontType, 0.5, color)
88.             cv2.putText(depthFrameColor, f"Z: {int(depthData.spatialC
    oordinates.z)} mm", (xmin + 10, ymin + 50), fontType, 0.5, color)
89.         cv2.imshow("depth", depthFrameColor)
90.
91.         calculationAlgorithm = dai.SpatialLocationCalculatorAlgorithm.
    MEAN
```

4.2 基于 Openmanipulator 机械臂的目标物体抓取与移动

4.2.1 双目相机与 Openmanipulator 机械臂的手眼标定

我们通过双目相机可以获取物体在图像中的二维坐标信息，如果想让机械臂末端执行器到达三维空间中的目标位置，就需要知道相机的图像坐标系与机械臂坐标系（一般指机器人底座）的对应关系。

而通过手眼标定的方法，我们可以获得双目相机图像坐标系与机械臂坐标系之间的关系，进而将双目相机得到的二维坐标位置转换为相对于机械臂坐标系的三维空间坐标。

核心代码如下：

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <time.h>
#include <sys/stat.h>
#include <ros/ros.h>
void Calibrate_Hand_Eye::calibrate_handeye() // 实现手眼标定
{
    // 开始求解 AX=XB 方程
    cv::HandEyeCalibrationMethod method;

    std::string str_handeye_result_file = data_file_path + "handeye_result.yaml";
    cv::FileStorage handeye_result = cv::FileStorage(str_handeye_result_file, cv::FileStorage::WRITE | cv::FileStorage::FORMAT_YAML | cv::FileStorage::APPEND);

    for (int method_idx = 0; method_idx < 5; method_idx++)
    {
        switch (method_idx)
        {
        case 0:
            method = cv::CALIB_HAND_EYE_TSAI;
            break;
        case 1:
            method = cv::CALIB_HAND_EYE_PARK;
            break;
        case 2:
            method = cv::CALIB_HAND_EYE_HORAUD;
            break;
        case 3:
            method = cv::CALIB_HAND_EYE_ANDREFF;
            break;
        case 4:
            method = cv::CALIB_HAND_EYE_DANIILIDIS;
            break;
        default:
            method = cv::CALIB_HAND_EYE_TSAI;
```

```
            break;
        }

        cv::Mat R_mc, t_mc;
        cv::calibrateHandEye(world_marker_R_vector, world_marker_t_vector, camera_board_R_vector, camera_board_t_vector, R_mc, t_mc, method);

        cv::Mat Tmc = (cv::Mat_<double>(4, 4) << R_mc.at<double>(0, 0), R_mc.at<double>(0, 1), R_mc.at<double>(0, 2), t_mc.at<double>(0),
                       R_mc.at<double>(1, 0), R_mc.at<double>(1, 1), R_mc.at<double>(1, 2), t_mc.at<double>(1),
                       R_mc.at<double>(2, 0), R_mc.at<double>(2, 1), R_mc.at<double>(2, 2), t_mc.at<double>(2),
                       0, 0, 0, 1);

        int num = world_marker_R_vector.size();
        std::vector<cv::Mat> vT_wb;
        for (int i = 0; i < num; i++)
        {
            cv::Mat Twm = (cv::Mat_<double>(4, 4) << world_marker_R_vector[i].at<float>(0, 0), world_marker_R_vector[i].at<float>(0, 1), world_marker_R_vector[i].at<float>(0, 2), world_marker_t_vector[i].at<float>(0),
                           world_marker_R_vector[i].at<float>(1, 0), world_marker_R_vector[i].at<float>(1, 1), world_marker_R_vector[i].at<float>(1, 2), world_marker_t_vector[i].at<float>(1),
                           world_marker_R_vector[i].at<float>(2, 0), world_marker_R_vector[i].at<float>(2, 1), world_marker_R_vector[i].at<float>(2, 2), world_marker_t_vector[i].at<float>(2),
                           0, 0, 0, 1);

            cv::Mat Tcb = (cv::Mat_<double>(4, 4) << camera_board_R_vector[i].at<double>(0, 0), camera_board_R_vector[i].at<double>(0, 1), camera_board_R_vector[i].at<double>(0, 2), camera_board_t_vector[i].at<double>(0),
                           camera_board_R_vector[i].at<double>(1, 0), camera_board_R_vector[i].at<double>(1, 1), camera_board_R_vector[i].at<double>(1, 2), camera_board_t_vector[i].at<double>(1),
                           camera_board_R_vector[i].at<double>(2, 0), camera_board_R_vector[i].at<double>(2, 1), camera_board_R_vector[i].at<double>(2, 2), camera_board_t_vector[i].at<double>(2),
                           0, 0, 0, 1);
```

```
65.
66.             cv::Mat Twb_tmp, Twb;
67.             Twb = Twm * Tmc * Tcb;
68.             vT_wb.push_back(Twb);
69.         }
70.
71.         std::vector<double> r1, r2, r3, r4, r5, r6, r7, r8, r9;
72.         std::vector<double> t1, t2, t3;
73.
74.         for (int i = 0; i < num; i++)
75.         {
76.             r1.push_back(vT_wb[i].at<double>(0, 0));
77.             r2.push_back(vT_wb[i].at<double>(0, 1));
78.             r3.push_back(vT_wb[i].at<double>(0, 2));
79.             r4.push_back(vT_wb[i].at<double>(1, 0));
80.             r5.push_back(vT_wb[i].at<double>(1, 1));
81.             r6.push_back(vT_wb[i].at<double>(1, 2));
82.             r7.push_back(vT_wb[i].at<double>(2, 0));
83.             r8.push_back(vT_wb[i].at<double>(2, 1));
84.             r9.push_back(vT_wb[i].at<double>(2, 2));
85.
86.             t1.push_back(vT_wb[i].at<double>(0, 3));
87.             t2.push_back(vT_wb[i].at<double>(1, 3));
88.             t3.push_back(vT_wb[i].at<double>(2, 3));
89.         }
90.     }
91.
92.     handeye_result.release();
93. }
```

4.2.2 Openmanipulator 机械臂抓取移动的流程设计

在启动机械臂的抓取任务之前，首先需要对机械臂的位姿进行初始化，以确保机械臂能够快速准确地进入理想的抓取状态。通过优化机械臂的初始位置和姿态，可以显著提高抓取过程的灵活性和效率。

核心代码如下：

```
1.  void OpenManipulatorPickandPlace::publishCallback(const ros::TimerEve
    nt&)
2.  {
3.    printText();
4.    if(kbhit()) setModeState(std::getchar());
5.
```

```
6.    if(mode_state_ == HOME_POSE)
7.    {
8.      std::vector<double> joint_angle;
9.      # 初始位姿
10.     joint_angle.push_back( 0.00);
11.     joint_angle.push_back(-1.05);
12.     joint_angle.push_back( 0.35);
13.     joint_angle.push_back( 0.70);
14.     setJointSpacePath(joint_name_, joint_angle, 2.0);
15.
16.     std::vector<double> gripper_value;
17.     gripper_value.push_back(0.0);
18.     setToolControl(gripper_value);
19.     mode_state_ = 0;
20.   }
21.   else if(mode_state_ == DEMO_START)
22.   {
23.     if(!open_manipulator_is_moving_) demoSequence();
24.   }
25. }
```

在完成机械臂位姿的初始化后，需要定义整个抓取移动的流程，本项目设计的机械臂抓取移动流程如下：

①初始姿态：机械臂移动到初始姿态，关节角度被设置为特定的值，然后调用 setJointSpacePath 函数来移动机械臂到该姿态。

②打开夹爪：机械臂保持当前姿态，夹爪被打开，准备抓取物体。

③移动至目标位置：机械臂根据 yolo_pose 数组中目标物体的三维坐标，移动到对应位置。如果物体检测失败，则重置到等待打开夹爪的状态。

④关闭夹爪：机械臂保持当前姿态，夹爪被关闭以抓紧物体。

移动至放置位置：机械臂移动到特定的放置位置，进入放置姿态，准备放置物体。

⑤打开夹爪：机械臂保持当前姿态，夹爪打开，放下物体。

⑥回到初始姿态：机械臂在放置箱子后向上移动，然后再次回到初始姿态。

核心代码如下：

```
1.  void OpenManipulatorApple::demoSequence()
2.  {
3.    std::vector<double> joint_angle;
4.    std::vector<double> kinematics_position;
5.    std::vector<double> kinematics_orientation;
```

```
6.    std::vector<double> gripper_value;
7.
8.    switch(demo_count_)
9.    {
10.   case 0: // home pose
11.     joint_angle.push_back( 0.00);
12.     joint_angle.push_back(-1.05);
13.     joint_angle.push_back( 0.35);
14.     joint_angle.push_back( 0.70);
15.     setJointSpacePath(joint_name_, joint_angle, 1.5);
16.     demo_count_ ++;
17.     break;
18.   case 1: // initial pose
19.     joint_angle.push_back( 0.01);
20.     joint_angle.push_back(-0.80);
21.     joint_angle.push_back( 0.00);
22.     joint_angle.push_back( 1.90);
23.     setJointSpacePath(joint_name_, joint_angle, 1.0);
24.     demo_count_ ++;
25.     break;
26.   case 2: // wait & open the gripper
27.     setJointSpacePath(joint_name_, present_joint_angle_, 3.0);
28.     gripper_value.push_back(0.010);
29.     setToolControl(gripper_value);
30.     demo_count_ ++;
31.     break;
32.   case 3: // pick the box
33.
34.     for(int i = 0; i < yolo_pose.size(); i ++)
35.     {
36.       if(yolo_pose[i].id == 1)
37.       {
38.         kinematics_position.push_back(yolo_pose[i].position[0]);
39.         kinematics_position.push_back(yolo_pose[i].position[1]);
40.         kinematics_position.push_back(yolo_pose[i].position[2]);
41.         setTaskSpacePath(kinematics_position, kinematics_orientation,
    2.0);
42.         demo_count_ ++;
43.
44.         return;
45.       }
46.
47.     demo_count_ = 2;  // If the detection fails.
48.     break;
```

```
49.   case 4: // wait & grip
50.     setJointSpacePath(joint_name_, present_joint_angle_, 1.0);
51.     // open the gripper
52.     gripper_value.push_back(-0.01);
53.     setToolControl(gripper_value);
54.     demo_count_ ++;
55.     break;
56.   case 5: // initial pose
57.     joint_angle.push_back( 0.01);
58.     joint_angle.push_back(-0.80);
59.     joint_angle.push_back( 0.00);
60.     joint_angle.push_back( 1.90);
61.     setJointSpacePath(joint_name_, joint_angle, 1.0);
62.     demo_count_ ++;
63.     break;
64.   case 6: // place pose
65.     joint_angle.push_back( 1.57);
66.     joint_angle.push_back(-0.21);
67.     joint_angle.push_back(-0.15);
68.     joint_angle.push_back( 1.89);
69.     setJointSpacePath(joint_name_, joint_angle, 1.0);
70.     demo_count_ ++;
71.     break;
72.   case 7: // place the box
73.     kinematics_position.push_back(present_kinematic_position_.at(0));

74.     kinematics_position.push_back(present_kinematic_position_.at(1));

75.     kinematics_position.push_back(present_kinematic_position_.at(2));


76.
77.     setTaskSpacePath(kinematics_position, kinematics_orientation, 2.0
    );
78.     demo_count_ ++;
79.     break;
80.   case 8: // wait & place
81.     setJointSpacePath(joint_name_, present_joint_angle_, 1.0);
82.     gripper_value.push_back(0.010);
83.     setToolControl(gripper_value);
84.     demo_count_ ++;
85.     break;
86.   case 9: // move up after place the box
87.     kinematics_position.push_back(present_kinematic_position_.at(0));
```

```
    kinematics_position.push_back(present_kinematic_position_.at(1));

    kinematics_position.push_back(0.135);
    kinematics_orientation.push_back(0.74);
    kinematics_orientation.push_back(0.00);
    kinematics_orientation.push_back(0.66);
    kinematics_orientation.push_back(0.00);
    setTaskSpacePath(kinematics_position, kinematics_orientation, 2.0
);
    demo_count_ ++;
    break;
  case 10: // home pose
    joint_angle.push_back( 0.00);
    joint_angle.push_back(-1.05);
       joint_angle.push_back( 0.35);
       joint_angle.push_back( 0.70);
       setJointSpacePath(joint_name_, joint_angle, 1.5);
       demo_count_ = 1;
       break;
    } // end of switch-case
  }
```

4.3 距离传感器测量的实现

本项目中，距离传感器也结合了 ROS 系统，保证了传感器与摄像头、机械臂之间的数据共享，这不仅为后续的核间通信提供了保障，同时也为机械臂的安全制动功能打下了基础。

需要注意的是，在 ROS 启动之前，需要对距离传感器的数据处理（包）做出定义，以便后续 ROS 系统的调用，核心代码如下：

```
bool CmdSerialInterfaceLinux::Open(std::string &port_name, uint32_t c
om_baudrate) {
  int flags = (O_RDWR | O_NOCTTY | O_NONBLOCK);

  com_handle_ = open(port_name.c_str(), flags);
  if (-1 == com_handle_) {
    LD_LOG_ERROR("CmdSerialInterfaceLinux::Open open error, errno:%d"
,errno);
    return false;
```

```
8.    }
9.
10.   com_baudrate_ = com_baudrate;
11.
12.   struct asmtermios::termios2 options;
13.   if (ioctl(com_handle_, _IOC(_IOC_READ, 'T', 0x2A, sizeof(struct asm
    termios::termios2)), &options)) {
14.     LD_LOG_ERROR("TCGETS2 first error, errno:%d",errno);
15.     if (com_handle_ != -1) {
16.       close(com_handle_);
17.       com_handle_ = -1;
18.     }
19.     return false;
20.   }
21.
22.   options.c_cflag |= (tcflag_t)(CLOCAL | CREAD | CS8);
23.   options.c_cflag &= (tcflag_t) ~(CSTOPB | PARENB);
24.   options.c_lflag &= (tcflag_t) ~(ICANON | ECHO | ECHOE | ECHOK | ECH
    ONL | ISIG | IEXTEN);  //|ECHOPRT
25.   options.c_oflag &= (tcflag_t) ~(OPOST);
26.   options.c_iflag &= (tcflag_t) ~(IXON | IXOFF | INLCR | IGNCR | ICRN
    L | IGNBRK);
27.
28.   options.c_cflag &= ~CBAUD;
29.   options.c_cflag |= BOTHER;
30.   options.c_ispeed = this->com_baudrate_;
31.   options.c_ospeed = this->com_baudrate_;
32.
33.   options.c_cc[VMIN] = 0;
34.   options.c_cc[VTIME] = 0;
35.
36.   if (ioctl(com_handle_, _IOC(_IOC_WRITE, 'T', 0x2B, sizeof(struct as
    mtermios::termios2)), &options)) {
37.     LD_LOG_ERROR("TCSETS2 error, errno:%d",errno);
38.     if (com_handle_ != -1) {
39.       close(com_handle_);
40.       com_handle_ = -1;
41.     }
42.     return false;
43.   }
44.
45.   if (ioctl(com_handle_, _IOC(_IOC_READ, 'T', 0x2A, sizeof(struct asm
    termios::termios2)), &options)) {
46.     LD_LOG_DEBUG("TCGETS2 second error, errno:%d",errno);
```

```
47.     if (com_handle_ != -1) {
48.       close(com_handle_);
49.       com_handle_ = -1;
50.     }
51.     return false;
52.   }
53.
54.   LDS_LOG_INFO("Actual BaudRate reported:%d",options.c_ospeed);
55.
56.   tcflush(com_handle_, TCIFLUSH);
57.
58.   rx_thread_exit_flag_ = false;
59.   rx_thread_ = new std::thread(RxThreadProc, this);
60.   is_cmd_opened_ = true;
61.
62.   return true;
63. }
64.
65. bool CmdSerialInterfaceLinux::ReadFromIO(uint8_t *rx_buf, uint32_t rx
    _buf_len,
66.                                         uint32_t *rx_len) {
67.   static timespec timeout = {0, (long)(100 * 1e6)};
68.   int32_t len = -1;
69.
70.   if (IsOpened()) {
71.     fd_set read_fds;
72.     FD_ZERO(&read_fds);
73.     FD_SET(com_handle_, &read_fds);
74.     int r = pselect(com_handle_ + 1, &read_fds, NULL, NULL, &timeout,
     NULL);
75.     if (r < 0) {
76.       // Select was interrupted
77.       if (errno == EINTR) {
78.         return false;
79.       }
80.     } else if (r == 0) {  // timeout
81.       return false;
82.     }
83.
84.     if (FD_ISSET(com_handle_, &read_fds)) {
85.       len = (int32_t)read(com_handle_, rx_buf, rx_buf_len);
86.       if ((len != -1) && rx_len) {
87.         *rx_len = len;
88.       }
```

```
    }
  }
  return len == -1 ? false : true;
}

bool CmdSerialInterfaceLinux::WriteToIo(const uint8_t *tx_buf, uint32_t tx_buf_len,
                                        uint32_t *tx_len) {
  int32_t len = -1;

  if (IsOpened()) {
    len = (int32_t)write(com_handle_, tx_buf, tx_buf_len);
      if ((len != -1) && tx_len) {
        *tx_len = len;
      }
    }
    return len == -1 ? false : true;
  }

  void CmdSerialInterfaceLinux::RxThreadProc(void *param) {
    CmdSerialInterfaceLinux *cmd_if = (CmdSerialInterfaceLinux *)param;
    char *rx_buf = new char[MAX_ACK_BUF_LEN + 1];
    while (!cmd_if->rx_thread_exit_flag_.load()) {
      uint32_t readed = 0;
      bool res = cmd_if->ReadFromIO((uint8_t *)rx_buf, MAX_ACK_BUF_LEN, &readed);
      if (res && readed) {
        cmd_if->rx_count_ += readed;
        if (cmd_if->read_callback_ != nullptr) {
          cmd_if->read_callback_(rx_buf, readed);
        }
      }
    }

    delete[] rx_buf;
  }

  }
```

接着，结合 ROS 的核心代码如下所示：

```
if (port_name.empty()) {
    ROS_ERROR("[ldrobot] input <port_name> param is null");
    exit(EXIT_FAILURE);
  }
```

```
5.
6.    lidar_commh->SetProductType(ldlidar::LDType::STP_23L);
7.
8.    cmd_port->SetReadCallback(std::bind(&ldlidar::LiPkg::CommReadCallba
    ck, lidar_commh, std::placeholders::_1, std::placeholders::_2));
9.
10.   if (!cmd_port->Open(port_name, serial_port_baudrate)) {
11.     ROS_ERROR("open lidar serial device:%s is fail.", port_name.c_str
    ());
12.     exit(EXIT_FAILURE);
13.   } else {
14.     ROS_INFO("open lidar serial device:%s is success.", port_name.c_s
    tr());
15.   }
16.
17.   // stop measure
18.   {
19.     if (!lidar_commh->SendCmd(cmd_port, 0x00, ldlidar::PACK_STOP)) {

20.       ROS_ERROR("Send cmd PACK_STOP is fail.");
21.       exit(EXIT_FAILURE);
22.     }
23.     auto st_time = std::chrono::steady_clock::now();
24.     bool wait_ack = false;
25.     while ((!wait_ack) && (std::chrono::duration_cast<std::chrono::mi
    lliseconds>(std::chrono::steady_clock::now()-
    st_time).count() < 1000)) {
26.       if (lidar_commh->IsWorkStopReady()) {
27.         lidar_commh->ResetWorkStopReady();
28.         wait_ack = true;
29.        }
30.        usleep(100);
31.     }
32.   ROS_INFO("Get lidar version information.");
33.   if (!lidar_commh->SendCmd(cmd_port, 0x00, ldlidar::PACK_VERSION)) {

34.     ROS_ERROR("Send cmd PACK_VERSION is fail.");
35.     exit(EXIT_FAILURE);
36.   }
37.   auto start_time = std::chrono::steady_clock::now();
38.   ldlidar::LidarDeviceVerInfoTypeDef lidar_device_inf;
39.   bool wait_result = false;
40.   ROS_INFO("Wait recv lidar version information.");
41.   while ((!wait_result) && \
```

```
42.     (std::chrono::duration_cast<std::chrono::milliseconds>(std::chron
    o::steady_clock::now()-start_time).count() < 1000)) {
43.     if (lidar_commh->IsDeviceInfoReady()) {
44.       lidar_commh->ResetDeviceInfoReay();
45.       lidar_device_inf = lidar_commh->GetLidarDeviceVersionInfo();
46.       wait_result = true;
47.     }
48.     usleep(100);
49.   }
50.
51.   if (wait_result) {
52.     ROS_INFO("Get lidar version information is success.");
53.     ROS_INFO("lidar firmware version:%s", lidar_device_inf.firmware_v
    er.c_str());
54.     ROS_INFO("lidar hardware version:%s", lidar_device_inf.hardware_v
    er.c_str());
55.     ROS_INFO("lidar manufacture times:%s",lidar_device_inf.manufactur
    e_times.c_str());
56.     ROS_INFO("lidar mcu id number:%s", lidar_device_inf.mcu_id.c_str(
    ));
57.     ROS_INFO("lidar sn code:%s", lidar_device_inf.sn.c_str());
58.     ROS_INFO("lidar pitch angle:%d,%d,%d,%d",lidar_device_inf.pitch_a
    ngle[0],
59.       lidar_device_inf.pitch_angle[1],
60.       lidar_device_inf.pitch_angle[2],
61.       lidar_device_inf.pitch_angle[3]);
62.     ROS_INFO("lidar blind area, near:%d,far:%d", lidar_device_inf.bli
    nd_area_most_near, lidar_device_inf.blind_area_most_far);
63.     ROS_INFO("lidar frequence val:%d", lidar_device_inf.frequence);
64.   }
65.   ROS_INFO("Start measure.");
66.   if (!lidar_commh->SendCmd(cmd_port, 0x00, ldlidar::PACK_GET_DISTANC
    E)) {
67.     ROS_ERROR("Send cmd PACK_GET_DISTANCE is fail.");
68.     exit(EXIT_FAILURE);
69.   }
70.
71.   // create a ROS topic
72.   ros::Publisher lidar_pub = nh.advertise<sensor_msgs::Range>(setting
    .topic_name, 10);
73.
74.   ros::Rate r(10); //10hz
75.   auto last_time = std::chrono::steady_clock::now();
```

```
  while (ros::ok()) {
    if (lidar_commh->IsFrameReady()) {
      lidar_commh->ResetFrameReady();
      last_time = std::chrono::steady_clock::now();
      ldlidar::LaserFrameDataType laser_frame_data;
      if (lidar_commh->GetLaserMeasureData(laser_frame_data)) {
        ///// 遍历测量帧数据
        ROS_INFO("get lidar frame Data, timestamp:%d", laser_frame_da
ta.timestamp);
        for (int i = 0; i < 10; i++) {
          ToMessagePublish(laser_frame_data.points[i].distance, setti
ng, lidar_pub);
          ROS_INFO("Measure data: ");
          ROS_INFO("distance:%d,noise:%d,peak:%d,confidence:%d,intg:%
d,reftof:%d",
           laser_frame_data.points[i].distance,
           laser_frame_data.points[i].noise,
           laser_frame_data.points[i].peak,
           laser_frame_data.points[i].confidence,
           laser_frame_data.points[i].intg,
           laser_frame_data.points[i].reftof);
          ROS_INFO("--------------------------------");
           }
      } else {
        ROS_ERROR("get lidar frame data error");
        exit(EXIT_FAILURE);
}
```

4.4 OpenAMP 核间通信的实现

本项目中，设置机械臂的抓取移动操作在 CPU0（主核）上进行，双目相机的目标检测和深度计算操作在 CPU2（主核）上进行，STP-23L 距离传感器的数据采集在 CPU1（从核）上进行。

在 OpenAMP 核间通信的部分，我们实现了 CPU1（从核）实时高性能地采集 STP-23L 距离传感器的数据，并利用 OpenAMP 将采集到的距离数据传递给 CPU0（主核）；CPU0（主核）在获取到距离数据后，进一步地处理分析，将结果通过 OpenAMP 再传递给 CPU1（从核），CPU1（从核）根据分析结果提供安全制动功能。

同时，我们注意到，在 OpenAMP 核间通信过程中，如果想要启动从核程序，

主核需要先主动发送消息通知从核。即由于回调函数的特性，从核程序无法主动运行。因此，我们设计了一套方案：主核发送不同类型的消息，在从核的回调函数中对消息类型进行判断，根据判断结果执行不同的任务。

①从核的核心代码介绍如下：

rpmsg_endpoint_cb 是一个静态定义的回调函数，其主要功能是处理来自远程处理器的消息，这些消息通过 RPMsg（Remote Processor Messaging）通道传递。此函数设计用于在飞腾派开发板中实现主从核之间的通信。

首先，函数通过类型转换和比较检查接收到的数据是否为一个特定的关闭消息（SHUTDOWN_MSG）。如果匹配，函数通过日志输出“Shutdown message is received.”，并将全局变量 shutdown_req 设置为 1，表示收到了关闭请求。之后，函数返回 RPMSG_SUCCESS（通常表示操作成功）。

如果接收到的数据是一个字符串"start"，函数通过 strcmp 函数进行比较确认。确认后，打印消息“CPU1 started collecting laser data.”，表明从核开始执行数据采集任务。随后，调用 STP23L 的内部函数 STP23L_GetLaserDistance 获取激光测距数据。在获取到距离数据后，尝试通过 RPMsg 通道将距离数据发送回主核。如果发送成功，打印“CPU1 has sent laser data.”；若发送失败，则记录错误信息“rpmsg_send failed.”。

如果接收到的是一个整型数据，即主核发送的距离判断结果，从核需要对其进行判断。如果是 0，从核输出“Currently, everything is normal”，表明当前没有需要抓取的物体,需要对机械臂进行制动。反之，如果整型数据非零，会连续输出多条“It is time to grab!”警告信息，强烈提示机械臂该抓取物体了。

从核的核心代码如下：

```
1.  static int rpmsg_endpoint_cb(struct rpmsg_endpoint *ept, void *data,
    size_t len, uint32_t src, void *priv)
2.  {
3.      (void)priv;
4.      (void)src;
5.      /* On reception of a shutdown we signal the application to termin
    ate */
6.      if ((*(unsigned int *)data) == SHUTDOWN_MSG)
7.      {
```

```
        ECHO_DEV_SLAVE_DEBUG_I("Shutdown message is received.\r\n");

        shutdown_req = 1;
        // #define RPMSG_SUCCESS             0 在 openamp/rpmsg.h
        return RPMSG_SUCCESS;
    }

    // 接受主核发送消息的“start”，通知从核开始采集数据
    if (strcmp((char* data), "start") == 0){
        ECHO_DEV_SLAVE_DEBUG_I("CPU1 started collecting laser data.\r\n");
        //调用 STP23_L 的函数，获取距离数据
        float distance = STP23L_GetLaserDistance(dev, 1, false);
        if (rpmsg_send(ept, &distance, sizeof(&distance)) < 0)
        {
            ECHO_DEV_SLAVE_DEBUG_E("rpmsg_send failed.\r\n");
        }
        else{
            ECHO_DEV_SLAVE_DEBUG_I("CPU1 has sent laser data.\r\n");

        }
    }

    // 接受主核发送距离判断结果，打印警告信息
    if(*(int *)data == 0){
        ECHO_DEV_SLAVE_DEBUG_I("Currently, everything is normal.\r\n");
    }
    else{
        ECHO_DEV_SLAVE_DEBUG_I("It is time to grab!.\r\n");
        ECHO_DEV_SLAVE_DEBUG_I("It is time to grab!.\r\n");
        ECHO_DEV_SLAVE_DEBUG_I("It is time to grab!.\r\n");
        ECHO_DEV_SLAVE_DEBUG_I("It is time to grab!.\r\n");
        ECHO_DEV_SLAVE_DEBUG_I("It is time to grab!.\r\n");
    }

    return RPMSG_SUCCESS;
}
```

②主核的核心代码介绍如下：

主核程序是一个主函数实现的控制流程，其核心功能在于利用 RPMsg（Remote Processor Messaging）机制在不同处理器间进行消息传递与处理，具体步骤如下：

i.初始化与设置

打开控制设备：尝试以读写模式(O_RDWR)打开/dev/rpmsg_ctrl0 设备文件，这是用于控制 RPMsg 通道的接口。如果打开失败，则输出错误信息并提前结束程序。

配置端点信息:定义一个 struct rpmsg_endpoint_info 结构体变量 eptinfo，用于存储新创建的 RPMsg 端点信息。这里设置端点名为"xxxx"（实际使用中应为有意义的标识），源和目标地址均为 0。然后通过 memcpy 函数填充端点名称。

创建 RPMsg 端点：使用 ioctl 系统调用，向 ctrl_fd 发送 RPMSG_CREATE_EPT_IOCTL 命令来创建一个新的 RPMsg 端点，如果创建失败则报错并跳转到错误处理标签 err0 处。

打开 RPMsg 设备：尝试打开 RPMsg 设备文件/dev/rpmsg0 用于数据传输，失败则报错并跳转到 err1。

准备轮询描述符：初始化一个 pollfd 结构体 fds，设置其文件描述符为刚打开的 rpmsg_fd，监听的事件为 POLLIN（表示可读事件）。

ii.消息循环处理

发送启动消息：循环内首先调用 write_full 函数向 RPMsg 设备写入字符串"start"，作为消息发送的开始标志，并打印日志信息。

等待并接收数据：使用 poll 函数阻塞等待直到有数据可读或者发生错误。如果因中断导致 poll 返回负值且错误码为 EINTR，则忽略错误继续循环；否则，报错并跳转至错误处理。

处理接收到的数据:从 RPMsg 设备读取一个浮点数表示的激光测距数据到变量 distance 中。根据 distance 的值进行条件判断，设定一个整型的变量 judgement。如果距离小于等于 0.2 米且大于 0，则设置 judgement 为 1；否则设为 0。

反馈判断结果：将 judgement 的值写回 RPMsg 设备，完成一次远程处理结果的反馈，并打印相关日志信息。

iii.结束与异常处理

程序设计为循环发送、接收和处理消息，除非遇到错误导致提前跳出循环并执行相应错误处理逻辑（如关闭已打开的文件描述符等），最终正常结束时返回

0。

主核程序的核心代码如下：

```
int main(int argc, char **argv){
    int ctrl_fd, rpmsg_fd, ret;
    struct rpmsg_endpoint_info eptinfo;
    struct pollfd fds;
    float distance = 0.0;
    int judgement = 0;

    ctrl_fd = open("/dev/rpmsg_ctrl0", O_RDWR);
    if (ctrl_fd < 0) {
        perror("open rpmsg_ctrl0 failed.\n");
        return -1;
    }
    memcpy(eptinfo.name, "xxxx", 32);
    eptinfo.src = 0;
    eptinfo.dst = 0;
    ret = ioctl(ctrl_fd, RPMSG_CREATE_EPT_IOCTL, eptinfo);
    if (ret != 0) {
        perror("ioctl RPMSG_CREATE_EPT_IOCTL failed.\n");
        goto err0;
    }

    rpmsg_fd = open("/dev/rpmsg0", O_RDWR);
    if (rpmsg_fd < 0) {
        perror("open rpmsg0 failed.\n");
        goto err1;
    }

    memset(&fds, 0, sizeof(struct pollfd));
    fds.fd = rpmsg_fd;
    fds.events |= POLLIN;

    while(1){

        ret = write_full(rpmsg_fd, "start", sizeof("start"));
        if (ret < 0) {
            perror("write_full failed.\n");
            goto err1;
        }

        printf("CPU0 sent message: Start! \n");

```

```
        ret = poll(&fds, 1, 0);

        if (ret < 0) {
            if (errno == EINTR)
                continue;
            goto err1;
        }

        ret = read_full(rpmsg_fd, &distance, sizeof(&distance));
        if (ret < 0) {
            perror("read_full failed.\n");
            goto err1;
        }

        printf("CPU0 received laser data\n");

        if(distance <= 0.2 && distance > 0){
            judgement = 1;
        }
        else if(distance > 0.2){
            judgement = 0;
        }

        ret = write_full(rpmsg_fd, &judgement, sizeof(&judgement));
        if (ret < 0) {
            perror("write_full failed.\n");
            goto err1;
        }

        printf("CPU0 sent judgement\n");

    }

    return 0;
}
```

4.5 良好人机交互模块的实现

4.5.1 Open-manipulator 机械臂控制的 GUI 界面

本 GUI 界面基于 ROS RVIZ 编写，旨在方便用户实时操控 Openmanipulator 机械臂，查看相应的运动状态和消息日志，同时提供用户友好的操作界面和直观

的反馈信息，以增强用户体验和提高操作效率。

在Openmanipulator机械臂控制的GUI界面中,用户可以点击"Timer Start"按钮来开启机械臂控制的任务规划。这一功能允许用户设定并启动一系列预定义的动作序列，确保机械臂按照预定的计划执行任务。

在左侧的栏目中，系统会实时打印OpenManipulator机械臂的消息日志，这不仅方便用户跟踪机械臂的操作历史和状态变化,而且也便于进行故障诊断和性能监控。

在右上侧的栏目中，用户可以检查OpenManipulator机械臂的当前状态，包括关节的角度、速度和扭矩等详细信息,以及机械臂末端执行器的精确运动位姿。这些数据对于确保机械臂的精确控制和优化操作流程至关重要。

在右下侧的栏目中，可选择在 joint space、task space 等界面操作机械臂。输入轨迹的关节角度或是目标位置的三维坐标，以及规划的总时间，点击send按钮，规划机械臂的运动。

除此之外，右侧的四个按钮还提供了额外的控制功能，以增强机械臂的灵活性和实用性。这些按钮可以分别执行以下操作：

初始化位姿按钮（Init Pose和Home Pose）：此按钮允许用户将机械臂的当前位姿重置为预设的初始状态。这在机械臂需要重新定位或在任务开始前需要确保机械臂处于特定位置时非常有用。

抓具开关按钮（Gripper open和Gripper close）：用户可以通过这个按钮来控制机械臂的抓具，实现开合动作。这对于执行抓取、搬运或放置物体的任务至关重要，确保机械臂能够准确地与目标物体进行交互。

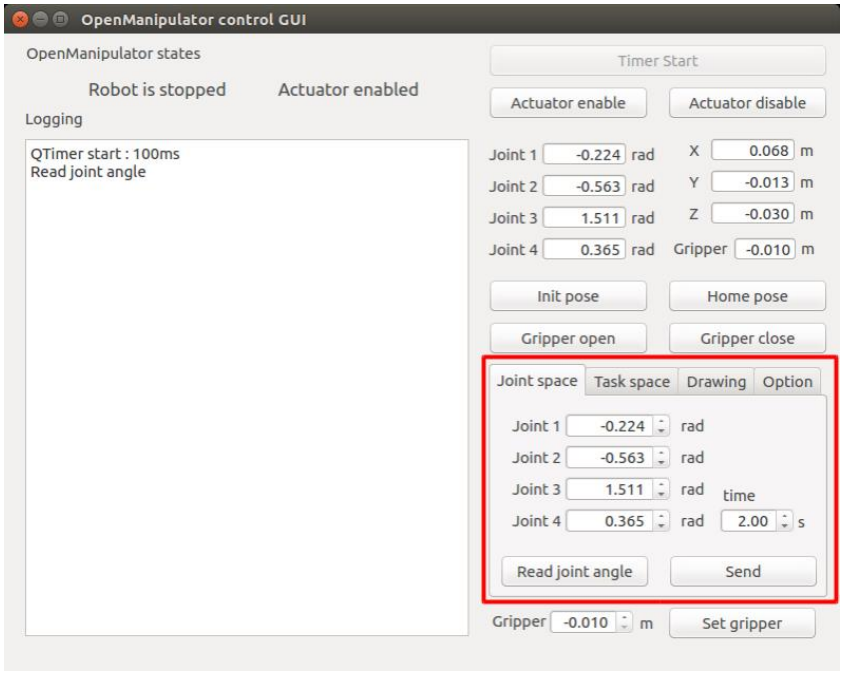


图 4.1 Openmanipulator 机械臂控制的 GUI 界面

4.5.2 三维立体可视化仿真 GUI 界面

本界面是由 ROS 设计的 RVIZ 可视化服务界面，它提供了一个直观、互动性强的三维视图，能够轻松地将 Openmanipulator 机械臂、双目相机以及其他相关设备进行可视化展示。

利用 RVIZ 的强大功能，用户可以在一个统一的三维空间中直观地查看 Openmanipulator 机械臂的每个关节和连杆的实时状态。这种可视化不仅帮助用户理解机械臂的复杂结构，还使得监控其运动变得更加简单。

除了机械臂，界面同样支持双目相机等传感器设备的可视化。用户可以在 RVIZ 界面中看到相机的视角和捕捉到的图像，以及它们与机械臂的相对位置和方向，从而更好地理解整个系统的运作。

同时，界面还提供实时动态反馈功能，当机械臂或相机进行移动或调整时，RVIZ 界面上的模型会同步更新，确保用户始终掌握最新的设备状态。用户不仅可以观察，还可以通过界面与设备进行交互。例如，通过点击或拖动 RVIZ 中的机械臂模型，用户可以发送指令来控制其运动，实现直观的操作体验。

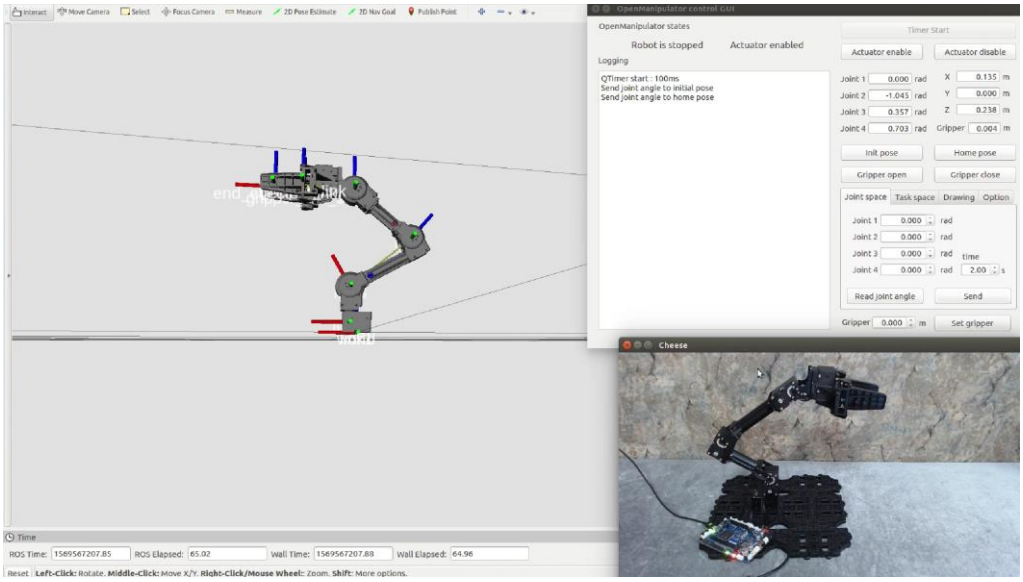


图 4.2 三维立体可视化仿真 GUI 界面

4.5.3 系统性能监控测量 GUI 界面

本界面基于 QT5 编写，旨在提供高效、直观的系统监控可视化服务，不仅能够方便用户实时监控开发板的 CPU 状态、内存占用情况、进程运行情况，还能实时测量核间通信响应时间并显示。

①CPU 状态监控：

用户可以通过本界面实时查看开发板的 CPU 使用率，包括各个核心的负载情况。同时界面还会计算平均 CPU 使用率，使用户能够一目了然地识别出高负载的时间段，从而进行相应的优化和调整。

②内存占用情况：

内存是开发板运行过程中的关键资源。本界面提供了内存使用情况的实时监控，包括总内存、已使用内存、空闲内存等关键指标。用户可以通过直观的图表和数字了解内存的使用效率，及时发现内存泄漏或不足的问题。

③进程运行情况：

进程是操作系统中执行任务的基本单位。本界面允许用户查看当前所有活跃进程的列表，包括进程 ID、名称、状态、父进程、会话等详细信息，方便用户快速定位到关键或异常的进程，进行管理和调整。

④核间通信响应时间：

对于多核处理器的飞腾派开发板，核间通信的效率直接影响到系统的整体

性能。本界面提供了核间通信响应时间的监控功能，用户可以实时了解不同核心之间的通信响应时间，有助于用户优化多线程程序，确保核间通信的高效性。

图 4.3 系统性能监控测量 GUI 界面

五． 系统测试与分析

5.1 目标检测测试分析

COCO（Common Objects in Context）数据集是一个大规模的图像识别用于对象检测、分割和字幕任务的数据集。它包含超过 330,000 张图像，每个都注释有 80 个对象类别和 5 个字幕描述场景。COCO 数据集广泛应用于计算机视觉研究，并已用于训练和评估许多最先进的对象检测和分割模型。

基于飞腾派开发板环境，我们将在 COCO 数据集上训练得到的 YOLOv4-tiny 网络在其测试数据集上进行评测，并与多种现有的目标检测模型进行对比，得到的评测结果如下所示：

模型	FPS	mAP（%）
YOLOv3	49	52.5
YOLOv4	41	64.9
YOLOv3-tiny	277	30.5
YOLOv4-tiny	**270**	**38.1**

可以发现，YOLOv4-tiny 网络在飞腾派上可以达到 270 FPS 的高检测速度，同时保持 mAP 为 38.1 的高检测精度，适合高实时目标检测的应用场景。

5.2 深度计算测试分析

OAK-D-Lite 双目深度相机的基线为 7.5cm，理论最小可感知的距离为 20cm，最大可感知距离为 35m。

我们对 OAK-D-Lite 双目深度相机的深度计算进行了 50 次测试，记录了预测深度和实际深度的差值。其中，所有测试数据的绝对误差之和为 45mm，平方误差之和为 2500mm²，平均绝对误差（MAE）为 0.9mm，平均平方误差（MSE）为 50 mm²。

绝对误差之和	平方误差之和	MAE	MSE
45mm	2500mm²	0.9mm	50 mm²

5.3 核间通信时间测试分析

核间通信时间是衡量多核处理器之间数据交换速度的关键指标。本项目中的 OpenAMP 核间通信，具体实现为 CPU1（从核）实时高性能地采集 STP-23L 距离传感器的数据，并利用 OpenAMP 将采集到的距离数据传递给 CPU0（主核）；CPU0（主核）在获取到距离数据后，进一步地处理分析，将结果通过 OpenAMP 再传递给 CPU1（从核），CPU1（从核）根据分析结果提供安全制动功能

针对 CPU0 与 CPU1 之间的 OpenAMP 核间通信，我们进行了 500 次核间通信测试，测试结果如下：

平均通信时间	最大通信时间	最小通信时间
5ms	10ms	3ms

5.4 从核高实时性测试分析

对于距离传感器的数据采集与处理，我们将其布置在从核上进行处理，为了体现从核的高实时性，对于从核采集处理数据的过程，我们进行了 100 次响应时间测算，结果如下：

	主核采集处理数据	从核采集处理数据
100 次测量平均响应时间	31ms	23ms

5.5 机械臂响应速度测试分析

机械臂的响应速度主要取决于机械臂的运动时间与开发板对于机械臂运动最优方案的解算时间。当机械臂初始位置与目标物体的距离过大时，受机械臂尺寸限制，会出现运动方案解算失败、机械臂无法到达的问题。针对不同的距离，我们各自进行了 100 次测算，得到的平均测算结果如下表所示：

机械臂初始位置与目标物体的距离	是否成功响应	机械臂响应到位时间
5cm	是	2.8s
10cm	是	3.3s

20cm	是	4.1s
30cm	是	4.6s
40cm	否	\

5.6 机械臂运动测试分析

在机械臂运动测试分析层面，我们对比了主机直接控制机械臂运动和飞腾派开发板控制机械臂运动的响应时间。经过 100 次测试后，飞腾派控制机械臂的平均响应时间在 10ms 内，与主机直接控制的差异极小，极大程度上体现了飞腾派的实时高性能性。

	直接控制机械臂	飞腾派控制机械臂
测试次数	100 次	100 次
平均响应时间	4.6ms	6.3ms
机械臂响应动作	第二关节上下 90° 摆动 5 次	第二关节上下 90° 摆动 5 次
动作完成时间	3.95s	4.01s

六、未来展望

在接下来的赛程中，我们计划对用户交互界面进行进一步的优化，以确保系统的可扩展性和可视化效果。此外，我们将进一步提升物体识别和机械臂自动抓取的能力，计划引入先进的目标检测算法，例如 YOLOv8，这将显著增强系统的复杂物体识别和环境适应性。

针对多核架构和 OpenAMP 核间通信，我们将充分利用四个 CPU 核心，精心设计和分配任务，以最大程度地发挥飞腾派开发板的性能潜力。通过这些措施，我们期望为用户提供更加流畅和高效的操作体验。