强化学习驱动的基于面部追踪的远程会议系统

黄卓彬；沈煜航；祁汝鑫

第一部分 设计概述

1.1 设计目的

当今的世界，依旧没有摆脱疫情对各行各业的冲击，其影响之一就是线上会议这一形式被越来越多的公司或团队采用。 而在视频会中，如何使用摄像头来对与会者的人脸进行全面的摄录，一直都是影响线上会议质量的关键因素。

我们针对这一痛点，基于海思 Taurus & Pegasus 套件与 OpenHarmony 嵌入式系统，设计了一套融合深度强化学习的 6 轴机械臂姿态解算算法的线上会议人脸追焦辅助系统。在这套系统的帮助下，无论与会者采用何种设备参加会议，都能实现全自动的人脸追焦，从而让与会者的面容始终清晰端正地保持在画面中央，显著提升视频会议的通话质量与入会体验。

1.2 应用领域

我们的系统可以配合多种便携式联网设备，如智能手机、平板电脑、掌上电脑等，完成对与会者面部摄录质量的优化，帮助与会者更好地向其他参会者展示摄录画面，并允许与会者在一定空间内自由活动。这套系统也可以应用于传统的摄录场景，如直播过程中的面部捕捉、网络课程的教师追逐等，通过机械臂平台的自动控制，可以实现快速、精准的主讲人面部追逐，避免摄像头录空或录偏。

1.3 主要技术特点

图像识别部分：我们使用 Taurus 套件集成的摄像头采集环境数据，并通过基于 yolo2 的面部检测网络提取人脸位置信息，整个过程使用 Hi3516 的 NNIE 核心进行加速。

网络通信部分：利用 Taurus 与 Pegasus 主板的网络通信模块，我们完成了基于移动边缘计算的物联网系统搭建。我们在接入侧网关边缘部署具有高算力的介入节点，Taurus 开发板、Pegasus 开发板、网络接入点各自是空间上独立的实体，在接入网中通过计算卸载（Computing Offload）的模式进行交互。

深度强化学习训练与决策部分：我们将高算力、高存储要求的深度强化学习模型计算与训练过程卸载到边缘节点上，在保证物联网系统低能耗的同时，满足了在资源受限条件下部署高性能、低延迟的决策模型的需求。

机械臂姿态解算与控制部分：在 Pegasus 开发板侧，它作为独立的实体完成对机械臂的姿态解算与控制。系统运行过程中，Pegasus 开发板将解读深度强化学习的决策，并控制机械臂完成相应的动作。

1.4 关键性能指标

目标检测部分：人脸检测模型的精度在单图从数张人脸到数百张人脸不等的 WIDER FACE 测试集下，拥有超过 90% 的识别准确率，并取决于单图人脸数量多少，召回率取值在 60% 到 90% 之间波动。对于少于五个人脸的照片，模型能检测到照片上极度模糊的目标，并能有效识别出拥有基本面部特征的极小目标（数十个像素组成）。在海思 Hi3516DV300 平台的 Taurus 套件上，由于室内场景大小有限，实测 6m * 8m 的房间内都能有效识别出人脸，并给出精确定位。

```
248208: 9.192521, 10.019512 avg, 0.000100 rate, 2.867938 seconds, 27799296 images
Loaded: 0.000064 seconds
Region Avg IOU: 0.602891, Class: 0.954198, Obj: 0.577088, No Obj: 0.005353, Avg Recall: 0.717557,  count: 131
Region Avg IOU: 0.596216, Class: 0.940298, Obj: 0.584147, No Obj: 0.004512, Avg Recall: 0.716418,  count: 134
Region Avg IOU: 0.660411, Class: 0.991803, Obj: 0.624842, No Obj: 0.004708, Avg Recall: 0.754098,  count: 122
```

图 1-1 人脸检测网络性能指标

机械臂部分：台座高度 7.5 cm，摄像头距离桌面最大可调整高度 32 cm，最低高度 18 cm，机械臂左右可旋转角度均为 60 度。机械臂关节处采用 PWM 舵机控制，输入电压 5V，实测整机待机电流 <0.5A，峰值工作电流 <1.5A，单步行动执行时间 0.25s。通过 Taurus 套件的广角摄像头，在距离机械臂半米的距离内，可以追踪上下半米、左右两米范围内移动的目标。

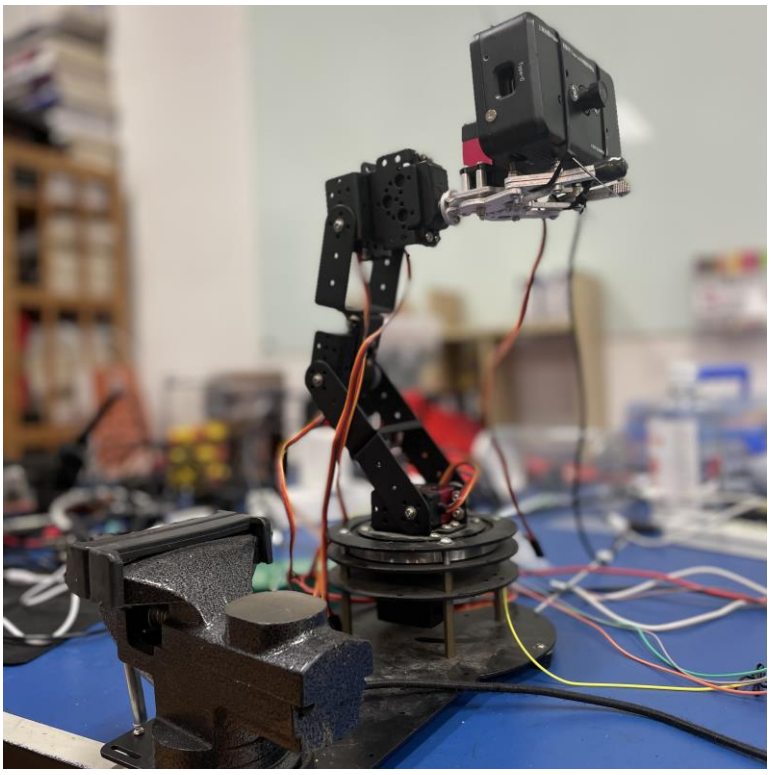


图 1-2 机械臂实物图

决策执行部分：边缘节点接收到的目标定位数据包速率大概为 15-20 报文/秒，机械臂动作执行时间为 0.25s，神经网络单次决策预测时间 <0.1s，神经网络单次训练时间 <0.1s（batch size 为 64），移动边缘计算卸载的总延迟（开始两次决策任务之间的间隔时间）大概为 0.7s，在摄像头补货范围内的面部大幅度运动的平均追踪时间为 2-3s。

1.5 主要创新点

功能创新点：

（1）基于机械臂的高效面部追踪：

我们将自带摄像头的 Taurus 套件以及用于参加网络会议的用户终端固定在改造后的机械臂抓取端，通过深度强化学习输出的行为决策来控制机械臂的姿态，进而能够有效追踪与会者的面部，让人脸始终直面摄像头。

（2）定制的姿态控制模块：

作为嵌入式平台的 Pegasus 套件拥有网络接口与丰富的 IO 引脚，但对于我们所用的机械臂而言，即使复用全部的串口也不足以驱动所有的舵机。同时，Pegasus 开发板的供电芯片也无法为舵机提供足够的供电。因此，我们选用了一块多串口的嵌入式主板作为下位机进行姿态控制，Pegasus 开发板只需要复用一个 IO 口与其通信。同时，我们基于该姿态控制板绘制了一块采用 LM2586S 供电芯片的 PCB，以确保满足 15W 的峰值功率。

（3）低功耗的物联网平台：

我们将决策与行动分离，在控制行动的物联网平台上，通过优化姿态校准算法，确保了舵机同时工作的总功率能够脱离电网供电。同时，我们采用容量 3300mah 额定电压 11.1v 放电倍率 25c 的格氏 ACE 3s 系列航模电池，经过测试我们这套独立系统可以在电池供电的情况下达到六个小时以上的续航，完美满足一天会议可能需要的使用时长。

系统与算法设计创新点：

（4）基于移动边缘计算的系统设计：

高性能的深度神经网络往往都和不低的平台功耗相挂钩，我们巧妙地使用海思嵌入式平台提供的神经网络加速器，部署了经过多次训练和优化的模型，实现了嵌入式平台级功耗下的高精度识别多层网络的部署。至于有高功耗、高性能需求的决策模型，我们采用移动边缘计算（Mobile Edge Computing，MEC）[9] 的模式，将决策计算过程从物联网板端卸载到边缘节点上，从而实现在物联网平台部署高算力模型。

MEC 的具体概念与设计思路详见附录 6.2。

（5）海思平台的本地推理模型部署：

我们充分利用 Taurus 套件的硬件特性与功能，在板端完成本地的图像捕获、处理、识别与判决，在 1GB 的有限运行存储中谨慎控制各程序的内存占用，并利用网络模块将图像的推理结果卸载到边缘节点。

我们的本地模型实现了在 Taurus 开发板广角摄像头可拍摄范围内的面部自动识别与定位，并有较好的识别准确率。

（6）边缘平台的决策模型部署：

MEC 允许我们以极低的硬件成本实现高性能、高负载、高存储的近端计算，在保持低功耗的同时，又能以接近本地运算的延迟获得云端计算的性能。我们利用计算卸载的方式，在边缘节点上部署复杂度高的深度强化学习模型，以实现能够自我学习的控制系统，让控制算法能够自适应外界环境的变化。经过训练，我们的决策模型能够根据识别到的面部定位结果控制舵机动作，将机械臂调整到把用户面部录入屏幕中央的姿态上，实现自动面部追踪的功能。

决策模型的选型以及问题建模相见附录 6.1。

（7）嵌入式平台的姿态解算算法：

在我们的系统中，Pegasus 开发板控制的姿态解算算法是为了配合深度强化学习的神经网络训练而设计的。其由固定的多组舵机位置数据组合而成，可以根据网络输出的需要，选择最合适的一组姿态作为最终指令输出给姿态控制板的微控制器。每一组姿态都经过精心的调试，确保舵机调整时间的和理性的同时，也尽可能地降低了多个舵机同时调整位置时的系统功耗，防止因为短时间多次或大量调整多个舵机导致系统电源过载致使整个系统下线。

结构创新点：

（8）执行传感一体化：

我们使用 Taurus 开发板的摄像头捕获环境数据，通过 Pegasus 开发板控制姿态控制板进行机械臂行为解算与执行，两个子模块都部署在同一个物联网平台上。我们将 Taurus 开发板固定在机械臂抓取端，同时预留放置用户入会终端（如手机、平板电脑等）的位置，将 Pegasus 开发板和姿态控制板放置在机械臂底部的台座上，使用扎带进行固定，整个机械臂的传感与执行单元都作为一体化物联网平台使用。最终搭建好的物联网平台，可以通过 IEEE802.11 与 IEEE802.3 协议与网络接入侧相连，从而接受来自移动边缘计算节点的控制。

（9）模块协同无线化：

在通常的 Taurus 开发板感知、Pegasus 开发板执行的模式下，不再复用 Pegasus 开发板上的 IO 进行与 Taurus 开发板间的串口通信，而是使用一个网络接入点将 Taurus 开发板、Pegasus 开发板以及边缘计算节点连入同一移动接入侧，以实现灵活、高效的物联网部署。

第二部分　系统组成及功能说明

2.1 整体介绍

在机器学习领域，计算机视觉（Computer Vision，CV）提供对可视化信息的理解能力，是智能体的眼睛，自然语言处理（Natural Language Processing，NLP）允许计算机正确识别语法语义，是智能体的耳朵，而强化学习（Reinforcement Learning，RL）[6] 则能有效综合对外界信息的观察结果，制定智能体（Agent）与环境交互的最佳行动策略，它正是智能体的大脑。我们的系统使用了一套完整的 “感知-决策-执行” 流程，通过在 Hi3516 芯片（Taurus 开发板）的 NNIE 单元上运行充分训练的 CV 模型，实现对摄像头捕获图片的特征提取，进而作为对环境状态的感知信息传达给 RL，预测出决策后通告 Hi3861 芯片（Pegasus 开发板）执行对应的机械臂行为。

从**结构**上看，整个系统由三个硬件部分构成：基于 Hi3516DV300 芯片的 Taurus 开发板及其外围套件，基于 Hi3861V100 芯片的 Pegasus 开发版及其外围套件，和移动边缘计算服务器。

其中，Taurus 开发板与边缘服务器之间通过无线局域网以 SOCKET 协议发送人脸识别后处理过的数据包。移动边缘计算节点通过深度强化学习模型进行决策，输出决策行为，通过无线局域网以 socket 协议向 Pegasus 开发板输出对应的行为控制信号。Pegasus 使用经过复用 IO 配置出的 USART 串口，将完成姿态解算的控制信号发送给姿态控制板的微控制器。微控制器最后通过向对应舵机的信号接收端输出不同的 PWM 信号，实现对机械臂上不同舵机转动位置的控制，进而完成了对机械臂整体姿态的调整。

我们的系统结构示意图如下：

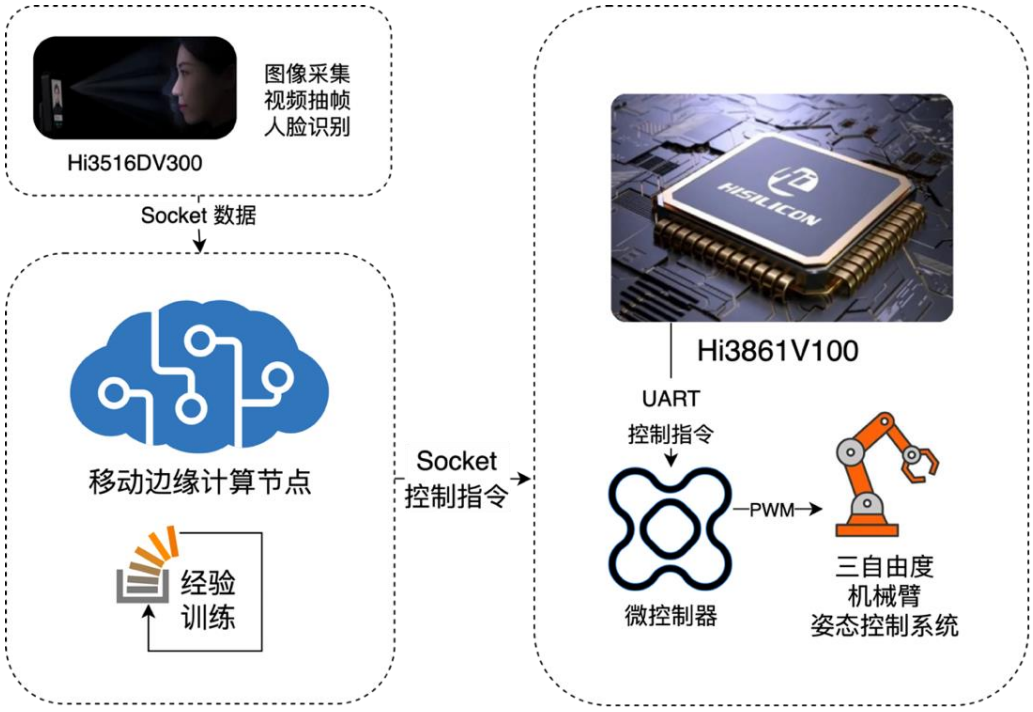


图 2-1 强化学习驱动的基于面部追踪的远程会议系统结构示意图

从**功能**上看，我们的系统主要分为用于感知与执行的**物联网平台**，以及用于决策的**移动边缘计算平台** [9]。

物联网平台中，Taurus 开发板使用自带的摄像头获取外界视觉信息，通过 NNIE 模块完成对面部目标的检测与识别，处理结果将以任务卸载的形式迁移给边缘服务器进行决策。Pegasus 开发板则只负责执行来自边缘服务器的决策结果，将命令解析为对应的 PWM 信号，完成对机械臂的控制。

在整个流程中，串接 Taurus 开发板与 Pegasus 开发板的正是边缘节点。它被部署在网络接入点旁边，以极低的时延处理来自物联网平台的任务请求，将 TAURUS 开发板识别结果输入深度强化学习模型中，预测出对机械臂的控制指令，并将结果下发给 Pegasus 开发板。在整个系统的运行过程中，边缘节点还会不断收集历史经验，并根据经验来训练强化学习模型，因而能够根据环境的变化（如主讲人变更、房间发生变化等），迅速调整出最适应当前环境的决策模型。

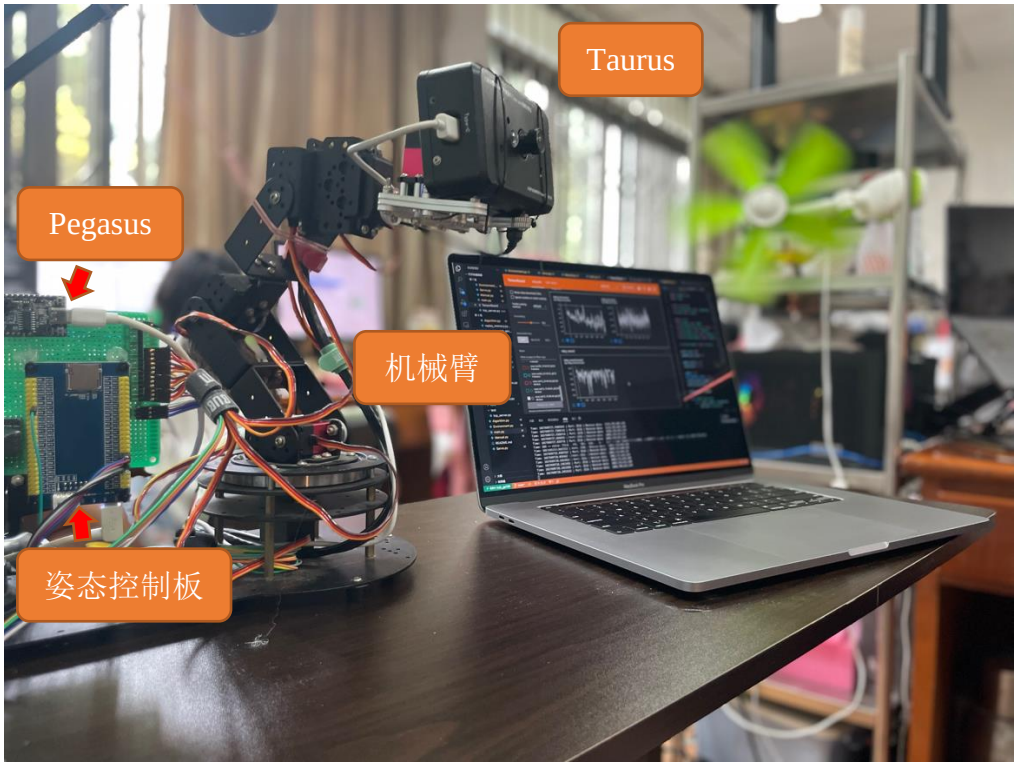


图 2-2 物联网平台（Taurus 套件，Pegasus 套件，姿态控制板，机械臂）

图 2-3 移动边缘计算平台（移动接入点，边缘服务器）

2.2 各模块介绍

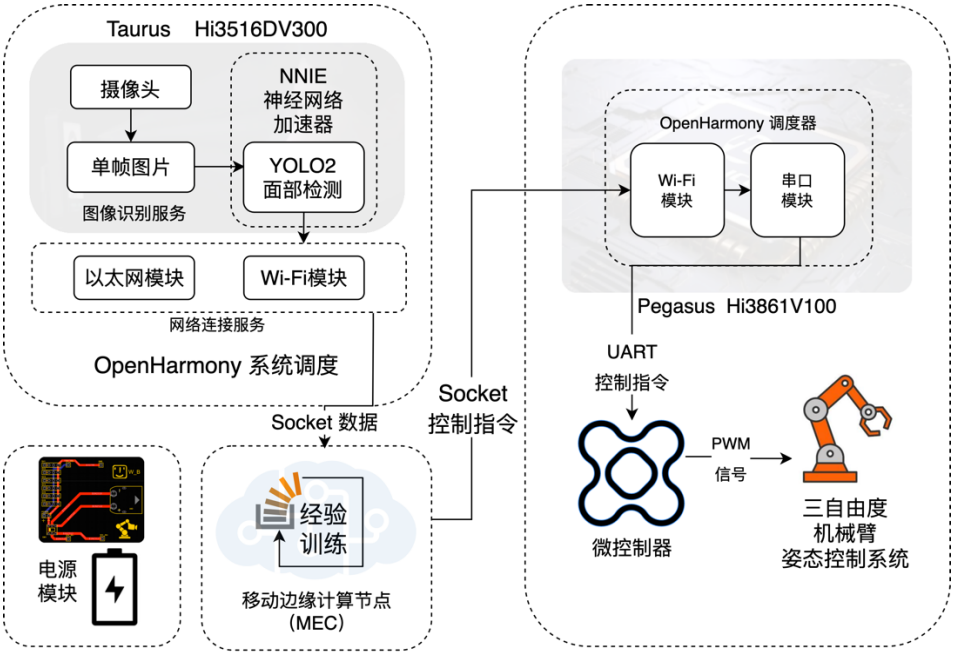


图 2-4 强化学习驱动的基于面部追踪的远程会议系统工作流程

1、物联网平台

（1）Taurus 套件：

Taurus 开发板通过 IMX335 摄像头采集外界视觉信息，从视频流中以接近 20fps 的速率采样出单帧图片，将图片传入 NNIE 单元加速的 YOLO2 面部检测网络中，完成对人脸数据的识别以及面部信息的定位后，一方面将经过浅绿色方框标注的图像输出至套件中的 LCD 显示屏，另一方面将检测结果经网络模块发送给移动边缘节点。

（2）Pegasus 套件：

Pegasus 开发板通过 WIFI 网络模块实现与网络接入侧的连接，从而接受来自移动边缘节点的行动控制。收到控制指令后，Pegasus 开发板将获得的运动指令进行处理分析，通过串口控制部署在三自由度机械臂旁的姿态控制模块，进而调整机械臂的姿态，从而实现对 Taurus 开发板摄像头捕捉画面的调整。

（3）姿态控制下位机：

姿态控制板分为下位机和供电模块两部分。下位机接收 Pegasus 开发板传来的姿态控制指令，对机械臂上的四组舵机进行控制。我们将每次动作执行的时间设置为 0.25s，将控制指令分为上下调整与左右调整两个维度，其中的每一个维度都量化出了 10 种姿态。从距离桌面 18cm 到 32 cm，有 10 个等距的可调高度位置，从偏离正面 -60 度到 +60 度，也有 10 个等角度的可调水平朝向位置。下位机通过姿态解算算法，得到行动指令对应的 PWM 信号，并从串口发送给指定的舵机，从而实现对机械臂姿态的调整。

（4）姿态控制供电模块：

在我们的物联网平台中，所有嵌入式节点都采用电池供电。我们设计了一块用于驱动机械臂的 PCB 作为姿态控制供电模块，作用是在保证硬件系统可以完全正常工作的基础上，减少额外线材的开销，提升供电系统的鲁棒性。经过测试，物联网平台的整机功耗最大为 15W 左右，因此选用了 LM2586S 为电压转换芯片，同时在 PCB 的设计过程中，考虑到单个舵机驱动电流为最大 3A 左右，对板内走线都进行了加宽处理，确保可以满足每个舵机的动态功耗需求。

2、移动边缘计算平台

（5）移动接入点：

在新一代移动通信网络中，大量的云资源下沉到了边缘，随着近年来第五代移动通信技术（5G）的大规模应用，以及 6G 随缘共享理念的引入，移动边缘计算（Mobile Edge Computing，MEC）逐渐成为移动通信领域的焦点。我们搭建了一套移动边缘计算平台，使用一台无线 AP 作为移动边缘接入点，通过在服务器上部署 ESXI 系统实现网络功能虚拟化（Network Function Virtualization，NFV），并用 LEDE 固件的 OPENWRT 系统作为基站的控制面与数据面服务。

从硬件上看，我们使用了一台服务器作为基站本体，在服务器内部切分出虚拟化的软路由，同时保留剩余资源以提供移动边缘计算服务。服务器连接到一台只提供二层转发能力的无线 AP 上，与 AP 相连的网口虚拟映射给软路由，从而实现对 5G 基站接入侧部署模式的仿真。

（6）移动边缘计算服务器：

在我们的系统中，用于计算的 Hi3516 平台虽然有强劲的 NNIE 神经网络加速核心，但是整机运行存储单元只有 1GB 的容量，在我们的测试中，先不论执行神经网络预测可否，单纯用于渲染桌面系统便几乎吃尽了所有空间。因此，通过重新编译纯 CLI 交互的 OpenHarmony 操作系统，我们成功实现了 Hi3516 板端的 CV 检测模型推理，然而需要在内存中保留大量探索经验的 RL 算法却完全无法嵌入进去。不同于 CV 部分的监督学习，RL 算法是没有训练集的，它需要在环境中进行带有随机性的探索，并不断积累经验，根据探索中的经验来逐步改进 RL 的策略。因此，在嵌入式系统中引入 RL 本身就是一个不小的挑战。

由于 MEC 计算卸载（Computing Offload）的引入，以嵌入式设备为首的物联网实体，可以在边缘节点的帮助下，以极低的硬件成本实现高性能、高负载、高存储的近端计算，在保持低功耗的同时，又能以接近本地运算的延迟获得云端计算的性能，因此 MEC 成为了近年来研究的热点。我们也在系统中选用了 MEC 的计算模式，由 Hi3516 平台执行板端推理，并将决策过程 offload 到作为边缘节点的近端主机上，边缘节点自己维护 Actor 与 Critic 两张神经网络与一个用于训练的经验池（Experience Pool），完成决策后交付回物联网侧终端，由 Hi3861 平台执行决策出的行为。

最终，我们选择 YOLO2 作为板端推理模型，带有层归一化、OUN 探索功能的 DDPG 算法 [8] 作为边缘端决策模型，计算卸载流程通过 socket 实现低功耗的边缘通信。

第三部分　完成情况及性能参数

强化学习驱动的基于面部追踪的远程会议系统可以将参加网络会议的终端挂载到改造后的机械臂抓取端，通过摄像头识别到与会者的面部位置，自主控制机械臂移动到能将人脸录制到屏幕中央的位置，实现对用户面部的自动追踪。以下是系统各部分的细节。

1、面部检测模型训练部署过程

（1）面部检测数据集的获取

我们选择 WIDER FACE 数据集用于面部检测网的训练。在这个数据集中，单张图片的人脸数量从一两个到数十个不等，且识别对象不仅包括正面的人脸，也包括侧脸与后脑勺，单张面部图片的清晰度、数据来源对象的肤色、头部的装饰物等方面也几乎涵盖了所有可能的情况，可以说这是涵盖面最广的数据集之一。

对于我们的需求而言，实际上只需要找出照片中最大的人脸位置即可。因此，我们需要对数据集进行筛选，选出其中人脸少于 5 张的照片进行训练。我们也曾使用完整的数据集进行训练，结果一轮训练在我们搭载 A5000 显卡的服务器上需要花费近两周的时间训练，且训练结果对于人脸较少的测试集，性能还不如只训练 5 张人脸以下的模型。

同时，WIDER FACE 数据集并不原生适配 DARKNET 网络的训练，我们需要将该数据集的标注文件转换成 DARKNET 所需的格式。因此，我们直接使用 PYTHON 写了一个自动化脚本，以进行数据集的挑选和标注文件的转换。

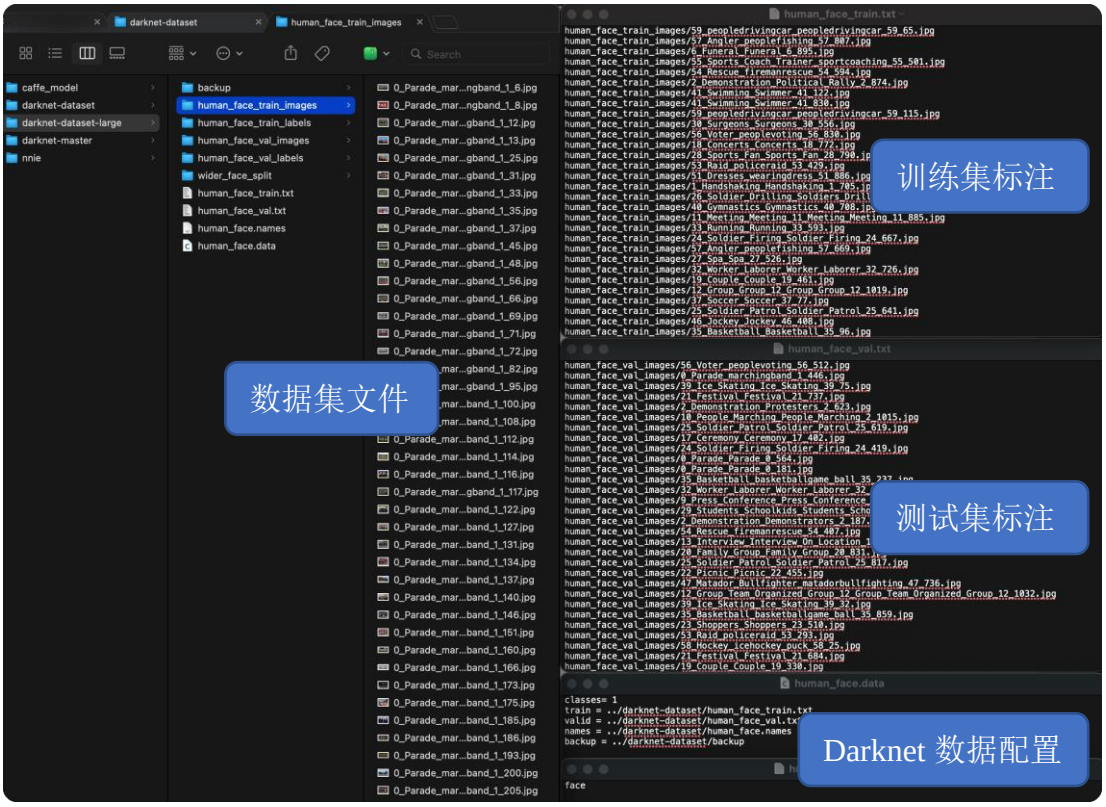


图 3-1 经过处理后的 WIDER FACE 数据集

（2）检测网的训练与转换

我们使用搭载双路志强铂金 CPU、256GB 内存、A5000 显卡的高性能计算服务器完成对检测网模型的训练。考虑到在 NNIE 单元中的算法兼容性，我们选择 YOLOV2 模型实现检测网的搭建。同时，为了方便模型的训练与测试，我们使用基于 cudann 的 darknet 框架来训练网络模型。

在另一台部署 ubuntu 18.04 的 CPU 服务器上，我们借助 darknet2caffe 工具，将训练得到的 .weights 权重文件首先转换成 .caffemodel 与 .prototxt 文件，并将 .prototxt 文件中不适合 NNIE 加速的内容剔除、修改 input_dim 的格式，再通过华为海思提供的 Ruyi Studio 平台，参照指导文档将 caffe 的网络文件转换成 Taurus 板端可用的 .wk 模型文件。

```
yolo2-hisilicon
layer     filters    size              input                output
    0 conv     64  3 x 3 / 2   640 x 384 x   3   ->   320 x 192 x  64  0.212 BFLOPs
    1 max          2 x 2 / 2   320 x 192 x  64   ->   160 x  96 x  64
    2 conv     64  3 x 3 / 1   160 x  96 x  64   ->   160 x  96 x  64  1.132 BFLOPs
    3 conv     64  3 x 3 / 1   160 x  96 x  64   ->   160 x  96 x  64  1.132 BFLOPs
    4 res    1                 160 x  96 x  64   ->   160 x  96 x  64
    5 conv     64  3 x 3 / 1   160 x  96 x  64   ->   160 x  96 x  64  1.132 BFLOPs
    6 conv     64  3 x 3 / 1   160 x  96 x  64   ->   160 x  96 x  64  1.132 BFLOPs
    7 res    4                 160 x  96 x  64   ->   160 x  96 x  64
    8 max          2 x 2 / 2   160 x  96 x  64   ->    80 x  48 x  64
    9 conv    128  1 x 1 / 1    80 x  48 x  64   ->    80 x  48 x 128  0.063 BFLOPs
   10 conv    128  3 x 3 / 1    80 x  48 x 128   ->    80 x  48 x 128  1.132 BFLOPs
   11 conv    128  1 x 1 / 1    80 x  48 x 128   ->    80 x  48 x 128  0.126 BFLOPs
   12 res    9                  80 x  48 x 128   ->    80 x  48 x 128
   13 conv    128  1 x 1 / 1    80 x  48 x 128   ->    80 x  48 x 128  0.126 BFLOPs
   14 conv    128  3 x 3 / 1    80 x  48 x 128   ->    80 x  48 x 128  1.132 BFLOPs
   15 conv    128  1 x 1 / 1    80 x  48 x 128   ->    80 x  48 x 128  0.126 BFLOPs
   16 res   13                  80 x  48 x 128   ->    80 x  48 x 128
   17 max         2 x 2 / 2    80 x  48 x 128   ->    40 x  24 x 128
   18 conv    256  1 x 1 / 1    40 x  24 x 128   ->    40 x  24 x 256  0.063 BFLOPs
   19 conv    256  3 x 3 / 1    40 x  24 x 256   ->    40 x  24 x 256  1.132 BFLOPs
   20 conv    256  1 x 1 / 1    40 x  24 x 256   ->    40 x  24 x 256  0.126 BFLOPs
   21 res   18                  40 x  24 x 256   ->    40 x  24 x 256
   22 conv    256  1 x 1 / 1    40 x  24 x 256   ->    40 x  24 x 256  0.126 BFLOPs
   23 conv    256  3 x 3 / 1    40 x  24 x 256   ->    40 x  24 x 256  1.132 BFLOPs
   24 conv    256  1 x 1 / 1    40 x  24 x 256   ->    40 x  24 x 256  0.126 BFLOPs
   25 res   22                  40 x  24 x 256   ->    40 x  24 x 256
   26 max         2 x 2 / 2    40 x  24 x 256   ->    20 x  12 x 256
   27 conv    512  1 x 1 / 1    20 x  12 x 256   ->    20 x  12 x 512  0.063 BFLOPs
   28 conv    512  3 x 3 / 1    20 x  12 x 512   ->    20 x  12 x 512  1.132 BFLOPs
   29 conv    512  1 x 1 / 1    20 x  12 x 512   ->    20 x  12 x 512  0.126 BFLOPs
   30 res   27                  20 x  12 x 512   ->    20 x  12 x 512
   31 conv    512  1 x 1 / 1    20 x  12 x 512   ->    20 x  12 x 512  0.126 BFLOPs
   32 conv    512  3 x 3 / 1    20 x  12 x 512   ->    20 x  12 x 512  1.132 BFLOPs
   33 conv    512  1 x 1 / 1    20 x  12 x 512   ->    20 x  12 x 512  0.126 BFLOPs
   34 res   31                  20 x  12 x 512   ->    20 x  12 x 512
   35 conv     30  1 x 1 / 1    20 x  12 x 512   ->    20 x  12 x  30  0.007 BFLOPs
   36 detection
mask_scale: Using default '1.000000'
Unused field: 'max_boxes = 300'
Loading weights from ../darknet-dataset/backup/yolo2-hisilicon.backup...Done!
Learning Rate: 0.01, Momentum: 0.9, Decay: 0.0005
Loaded: 0.383483 seconds
Region Avg IOU: 0.372482, Class: 0.842268, Obj: 0.342546, No Obj: 0.005851, Avg Recall: 0.416404,  count: 317
Region Avg IOU: 0.561978, Class: 0.938271, Obj: 0.553950, No Obj: 0.004643, Avg Recall: 0.666667,  count: 162
Region Avg IOU: 0.471948, Class: 0.875432, Obj: 0.465369, No Obj: 0.006683, Avg Recall: 0.557093,  count: 289
Region Avg IOU: 0.410192, Class: 0.870833, Obj: 0.391746, No Obj: 0.005317, Avg Recall: 0.450000,  count: 240
250801: 13.287799, 13.287799 avg, 0.000010 rate, 2.791239 seconds, 28089712 images
Loaded: 0.000079 seconds
Region Avg IOU: 0.590695, Class: 0.932692, Obj: 0.554063, No Obj: 0.004001, Avg Recall: 0.663462,  count: 104
Region Avg IOU: 0.464585, Class: 0.937143, Obj: 0.452231, No Obj: 0.004354, Avg Recall: 0.537143,  count: 175
Region Avg IOU: 0.601379, Class: 0.919999, Obj: 0.602701, No Obj: 0.006006, Avg Recall: 0.720000,  count: 175
Region Avg IOU: 0.434246, Class: 0.973118, Obj: 0.430391, No Obj: 0.004749, Avg Recall: 0.494624,  count: 186
```

图 3-2 面部检测网络（YOLO2）训练过程

（3）检测网的部署与测试

参考开发手册中提供的检测网部署案例，我们成功将生成的 .wk 文件导入并部署到了 Taurus 板端，测试结果如下：

图 3-3 板端面部识别测试

物联网平台检测到人物面部后，会将定位信息通过接入网关发送给边缘节点。我们在图 3-4 中打印了 Taurus 板端的日志信息，通过串口在 PC 平台查看 Taurus 的 CLI 输出，内容为 NNIE 模块对用户面部信息的定位结果。同时，在图 3-5 中打印了移动边缘计算节点中的日志，可以看到边缘节点成功接收到了来自 Taurus 板端的检测数据。

```
[Yolo2HandDetectResnetClassifyCal]-234: biggestBoxIndex:0, objNum:1
[Yolo2HandDetectResnetClassifyCal]-273: crop u32Width = 108, img.u32Height = 142
[GetVpssChnFrameHandClassify]-1707: get vpss frame success, weight:1920, height:1080
[Yolo2HandDetectResnetClassifyCal]-230: yolo2_out: {657, 272, 981, 675}
[Yolo2HandDetectResnetClassifyCal]-234: biggestBoxIndex:0, objNum:1
[Yolo2HandDetectResnetClassifyCal]-273: crop u32Width = 108, img.u32Height = 142
[GetVpssChnFrameHandClassify]-1707: get vpss frame success, weight:1920, height:1080
[Yolo2HandDetectResnetClassifyCal]-230: yolo2_out: {654, 275, 975, 675}
[Yolo2HandDetectResnetClassifyCal]-234: biggestBoxIndex:0, objNum:1
[Yolo2HandDetectResnetClassifyCal]-273: crop u32Width = 106, img.u32Height = 142
[GetVpssChnFrameHandClassify]-1707: get vpss frame success, weight:1920, height:1080
[Yolo2HandDetectResnetClassifyCal]-230: yolo2_out: {651, 275, 978, 675}
[Yolo2HandDetectResnetClassifyCal]-234: biggestBoxIndex:0, objNum:1
[Yolo2HandDetectResnetClassifyCal]-273: crop u32Width = 108, img.u32Height = 142
[GetVpssChnFrameHandClassify]-1707: get vpss frame success, weight:1920, height:1080
[Yolo2HandDetectResnetClassifyCal]-230: yolo2_out: {651, 270, 975, 675}
[Yolo2HandDetectResnetClassifyCal]-234: biggestBoxIndex:0, objNum:1
[Yolo2HandDetectResnetClassifyCal]-273: crop u32Width = 108, img.u32Height = 144
[GetVpssChnFrameHandClassify]-1707: get vpss frame success, weight:1920, height:1080
[Yolo2HandDetectResnetClassifyCal]-230: yolo2_out: {651, 272, 978, 677}
[Yolo2HandDetectResnetClassifyCal]-234: biggestBoxIndex:0, objNum:1
[Yolo2HandDetectResnetClassifyCal]-273: crop u32Width = 108, img.u32Height = 144
[GetVpssChnFrameHandClassify]-1707: get vpss frame success, weight:1920, height:1080
```

图 3-4 板端检测网识别内容 CLI 输出

```
No human face detected. Now reseting the robot arm.
Time: 1655998079.660359 | Port: 3516 | Receive data:  783,522,891,590
Time: 1655998079.7155755 | Port: 3516 | Receive data:  783,528,891,585
Time: 1655998079.7605906 | Port: 3516 | Receive data:  783,528,894,590
Time: 1655998079.8157368 | Port: 3516 | Receive data:  780,528,891,579
Time: 1655998079.8605728 | Port: 3516 | Receive data:  780,525,891,582
Time: 1655998079.9154322 | Port: 3516 | Receive data:  777,519,891,585
Time: 1655998079.967853 | Port: 3516 | Receive data:  777,522,888,582
Time: 1655998080.0154257 | Port: 3516 | Receive data:  774,519,894,585
Time: 1655998080.0606797 | Port: 3516 | Receive data:  771,519,891,582
Time: 1655998080.1153483 | Port: 3516 | Receive data:  774,516,894,585
```

图 3-5 边缘计算节点 SOCKET 接收内容 CLI 输出

2、深度强化学习决策模型训练部署过程

（1）决策模型的训练

通过搭建边缘网络，我们成功打通了物联网平台与边缘计算平台之间的连接，并尝试使用固定的静态图片以及显示器中播放的动态视频两种方式去预训练强化学习模型，如图 3-6 与图 3-7 所示。我们首先采用打印出的静态图片进行了一轮 500 steps 的训练，随后将训练好的模型作为初始参数，再次使用乔布斯的演讲视频作为强化学习的“环境”来进一步强化算法的决策能力。

经过两轮训练后，模型已经有了不错的决策性能，我们最后在真实场景下进行了真人训练，如图 3-8 所示。

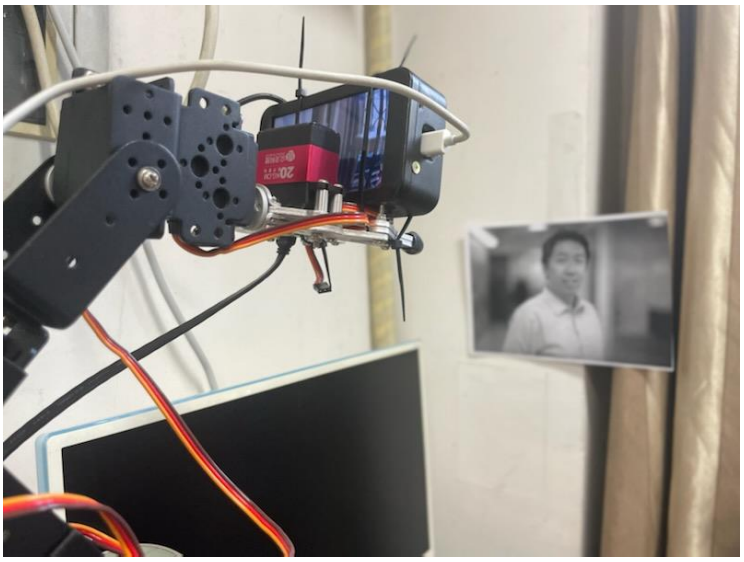
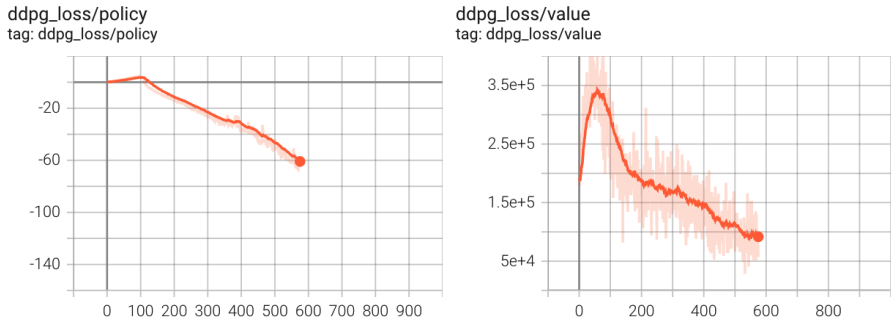


图 3-6 使用静态图片的深度确定性策略梯度算法（DDPG）的训练过程

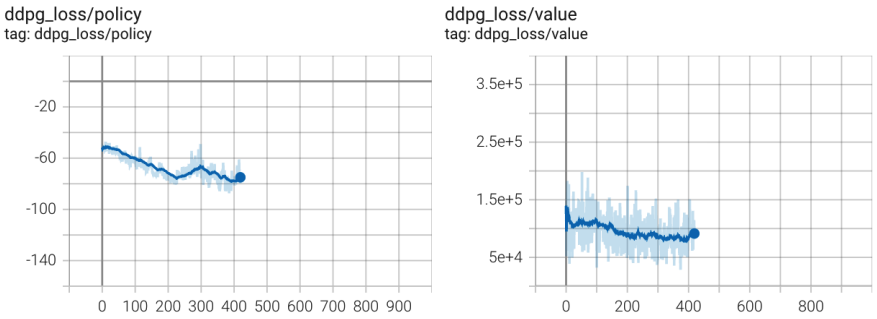
图 3-7 使用动态视频的 DDPG 训练过程

图 3-8 在真实场景下的 DDPG 训练过程

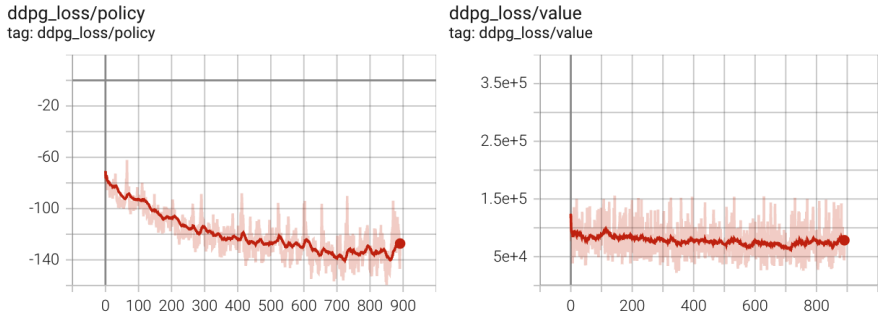
因为一次训练时间过长，我们在训练的过程中对强化学习的神经网络参数以及经验池进行了本地持久化，进而可以实现对同一 DDPG 策略的增量训练。我们在不同时间段进行了三次长时间的训练，可以看到在最后一次训练时，Actor 网络与 Critic 网络的 Loss 曲线都已经接近于收敛。



(a)



(b)



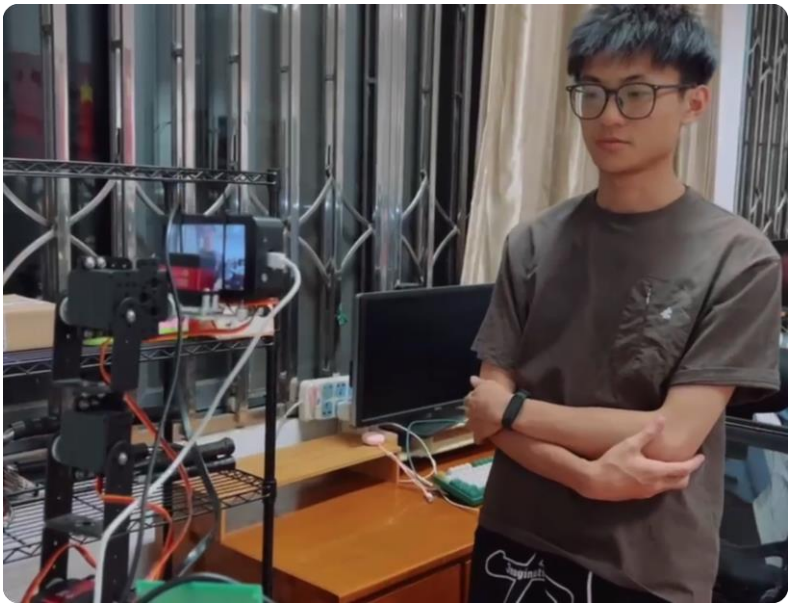
(c)

图 3-9 DDPG 算法的 Actor 网络与 Critic 网络 Loss 曲线图

（2）决策模型的部署

我们将训练好的深度强化学习模型部署在边缘计算服务器上，模仿用户的真实使用情况对整个系统进行了测试。下面我们截取了用户处于坐姿、站姿以及位置较偏的三种情况，可以看见深度强化学习模型都成功完成了正确的行为决策。

(a)



(b)

(c)

图 3-10 系统运行测试与决策模型验证

3、姿态控制模块供电板电路设计

为了驱动机械臂上的舵机，并保证抓取端固定重物的情况下不会让舵机过载，我们特别设计了一套 PCB 板来提供安全稳定的供电。

经测试，我们的姿态控制模块能够完美驱动整个物联网平台，并能通过格氏 ACE 航模电池进行供电，大大增强了硬件平台的移动性，且极低的平台功耗允许整个系统全功率运行五个小时。

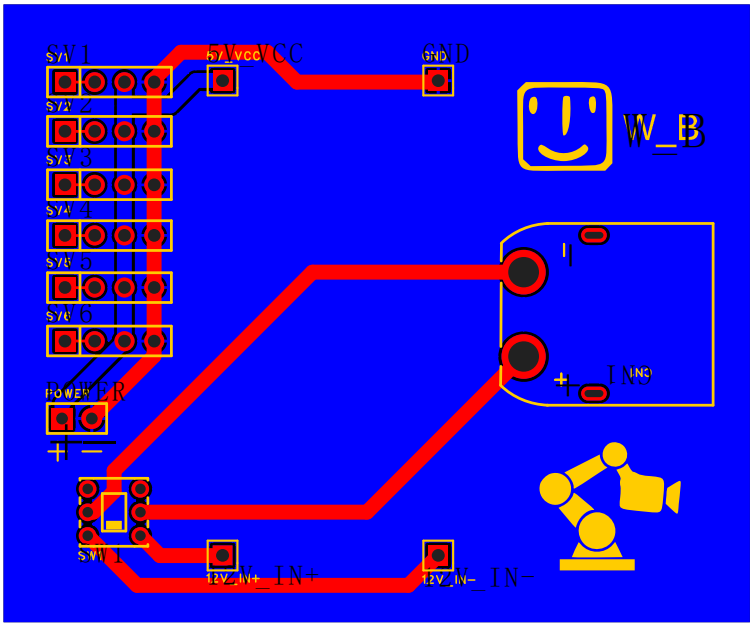


图 3-11 姿态控制供电板 PCB 原理图

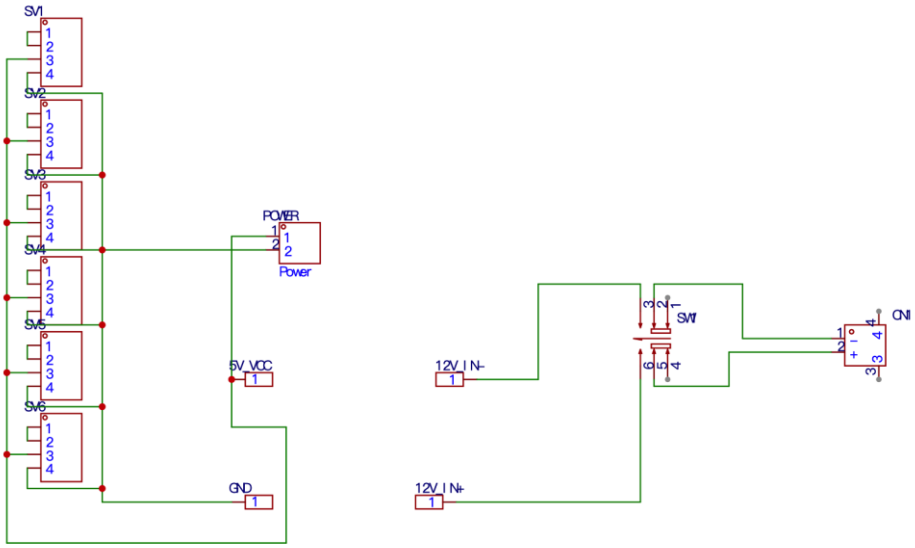


图 3-12 姿态控制供电板 PCB 板图

根据图 3-12 的板图，我们对供电板进行了打样，成品实物图如图 3-13 所示。我们使用 3300mAh 的格氏 ACE 航模电池为其供电，在 PCB 右部设计了侧接的供电输入端口，左部安排了五组供电，使用加粗的 28 芯纯铜杜邦线与舵机相连。通过 LM2586S 电压转换芯片，我们能够为整个机械臂提供稳定的 5V 供电，并能承受所有舵机工作的峰值功率。

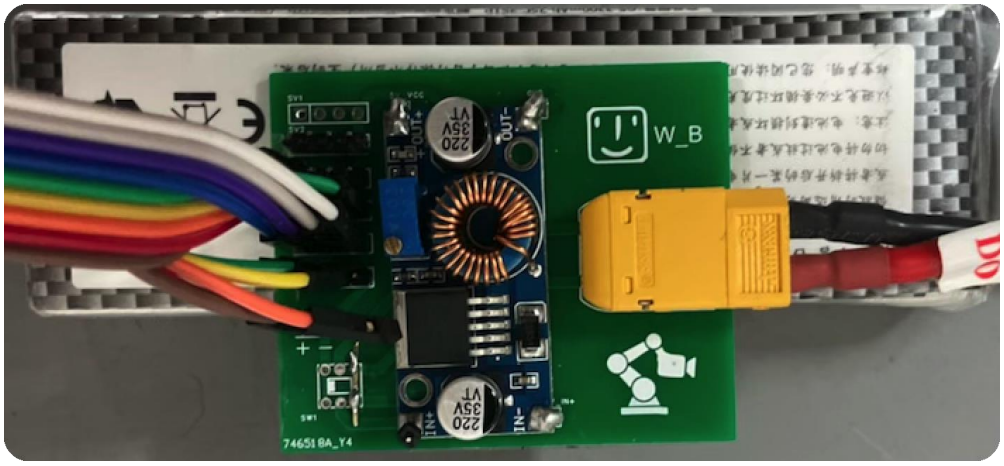


图 3-13 供电板 PCB 成品图（底部为 3300mAh 格氏 ACE 航模电池）

5、物联网平台整体结构

我们按功能将系统分成了两个平台：物联网平台与移动边缘计算平台。在此我们重点讨论集成了 Taurus 和 Pegasus 套件的物联网平台，关于边缘平台的介绍与实物展示参考附录 6.2 与图 2-3。

我们的一体化物联网平台包括 5 个主要部分：Taurus 套件、Pegasus 套件、姿态控制下位机、姿态控制供电板、机械臂，通过自主设计集成在一个桌面机械臂平台上，并预留了移动终端的固定支架。

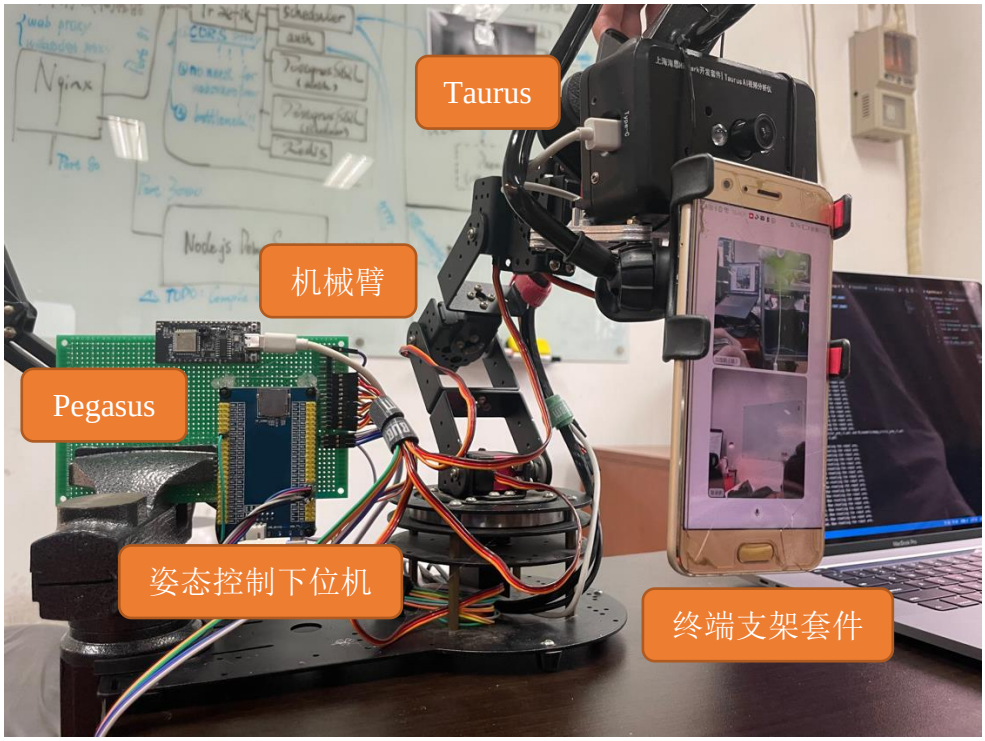


图 3-14 物联网平台整体结构（正面）

我们在图 3-14 中给出了物联网平台的正面结构实拍图。其中，Pegasus 的 Hi3861 核心板、姿态控制下位机、以及供电模块集成在一张主板上，姿态控制供电板与 3300mAh 航模电源组成的供电模块部署在主板背面，如图 3-15 所示。

整个物联网平台的主体是用于控制摄像角度的六舵机机械臂，为了方便用户通过不同移动终端接入会议，或者开启直播自拍，我们在抓取端设置了一个固定支架，可以兼容各类小型移动终端。通过将 Taurus 套件部署在支架上方，利用 IMX335 摄像头捕获用户面部信息，并使用 Hi3516 芯片的 NNIE 单元进行目标检测与定位，从而能够辅助机械臂调整终端的朝向，让用户的面部始终被移动终端的摄像头捕获，且置于拍摄中心。

除此之外，我们的终端支架采用易拆卸的设计，整个模块可以很方便的单独取出，并且支架采用 17mm 万向球头，能够兼容常规的车载支架设备，也能替换为适用于苹果手机的 Magsafe 磁吸模块，如图 3-16 所示。

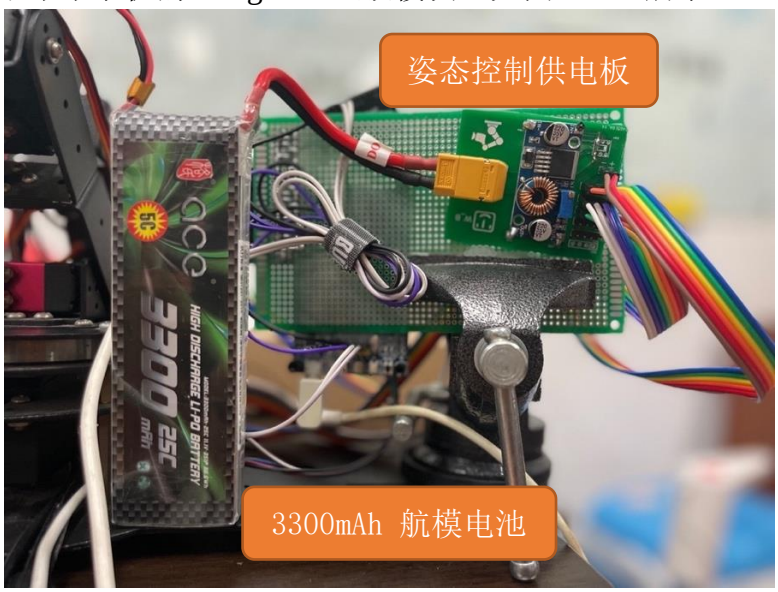


图 3-15 物联网平台供电模块

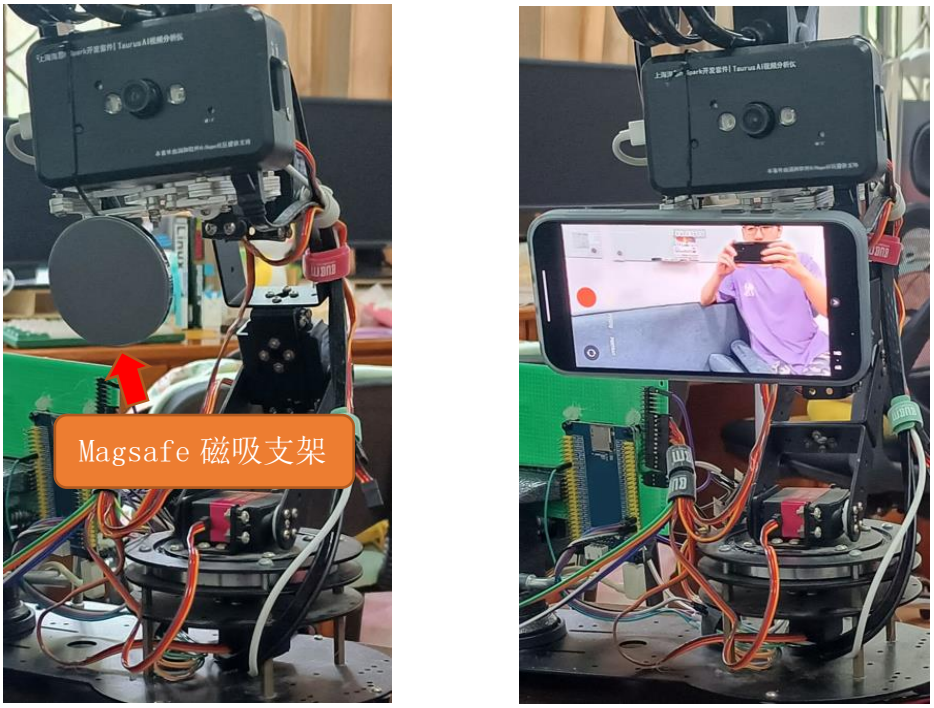


图 3-16 Magsafe 磁吸模式的终端支架（左），挂载 iPhone 实机演示（右）

6、最终实现效果

通过各功能模块的协同工作，我们成功实现了强化学习驱动的基于面部追踪的远程会议系统。在此，我们测试了系统的整个运行流程，图 3-17、3-18、3-19 给出了系统的启动过程。首先需要保证移动边缘网络能够正常工作，然后远程控制边缘节点开启边缘计算服务。随后启动 Taurus 套件与 Pegasus 核心板，二者将自动通过 socket 接入边缘服务。

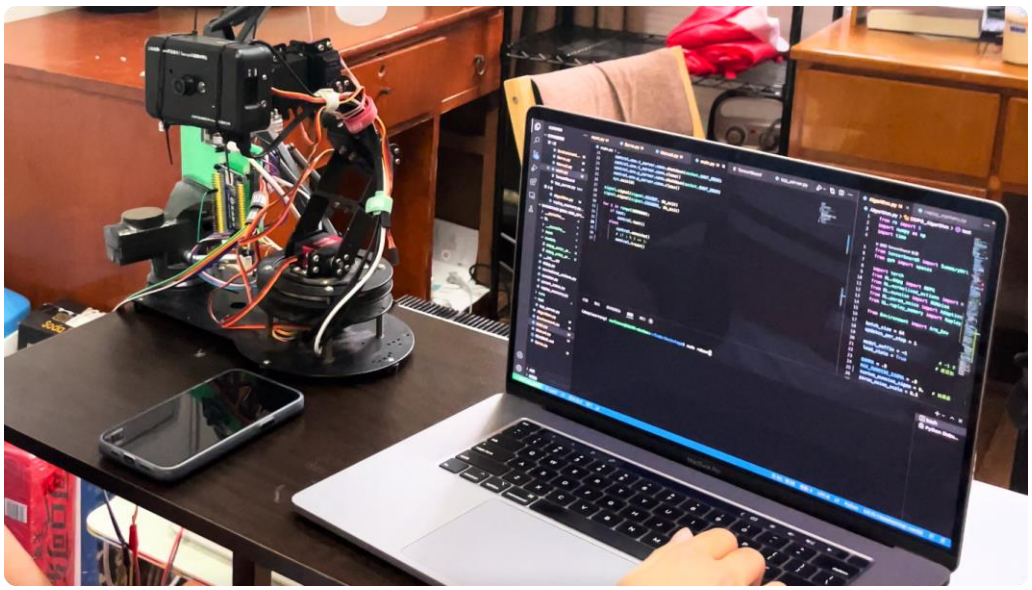


图 3-17 远程启动边缘计算服务

```
# ./ohos_camera_ai_demo 1
load sys for Hi3516CV500...OK!
load chnl.ko for Hi3516CV500...OK!
loEnter hiirq_open
ad region for Hi3516CV500...OK!
load gdc for Hi3516CV500...OK!
load vgs for Hi3516CV500...OK!
load dis for Hi3516CV500...OK!
load vi for Hi3516CV500...OK !
load isp for Hi3516CV500...OK !
load vpss for Hi3516CV500...OK!
load vo for Hi3516CV500...OK!
load vedu for Hi3516CV500...OK!
load rc for Hi3516CV500...OK!
load venc for Hi3516CV500...OK!
load h264e for Hi3516CV500...OK!
load h265e for Hi3516CV500...OK!
load jpege for Hi3516CV500...OK!
load jpegd for Hi3516CV500...OK!
load vdec for Hi3516CV500...OK!
load ive for Hi3516CV500...OK!
load nnie for Hi3516CV500...OK!
SDK init ok...
[SAMPLE_MEDIA_HAND_CLASSIFY]-1938: AIC: snsMaxSize=2592x1536
[SAMPLE_COMM_VI_SetMipiAttr]-606: MipiDev0 SetMipiAttr enWDRMode: 0
linear mode
-------Sony IMX335_init_5M_2592x1944_12bit_linear30 Initial OK!-------
```

图 3-18 串口控制启动 Taurus 面部检测服务

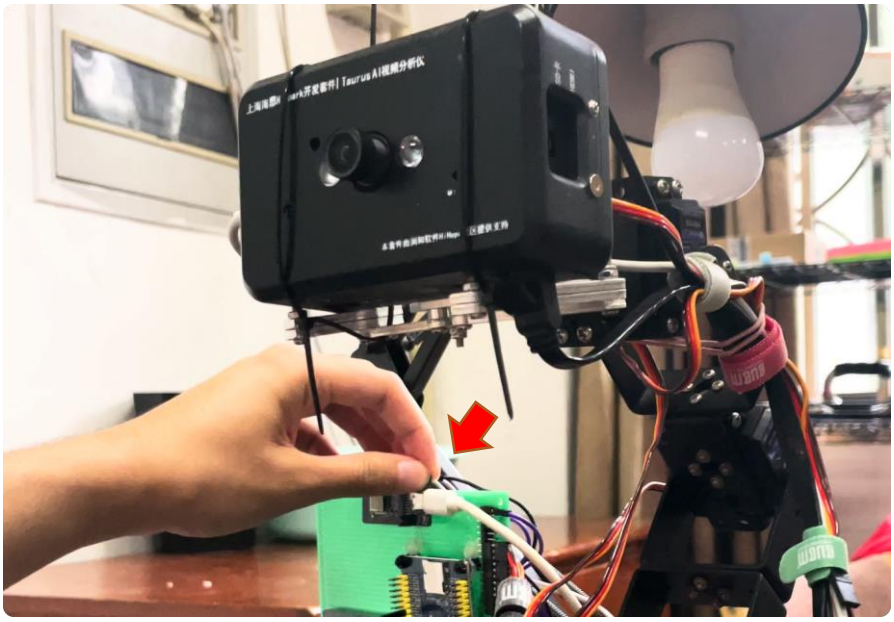


图 3-19 通过 reset 按键重启 Pegasus 核心板控制服务

系统启动完毕后，Taurus 套件将自动获取摄像头捕获范围内的面部信息，并将定位结果发送给边缘节点，边缘节点经过决策后，把控制命令下发给 Pegasus 板端，进而控制机械臂的移动，如图 3-20 所示。

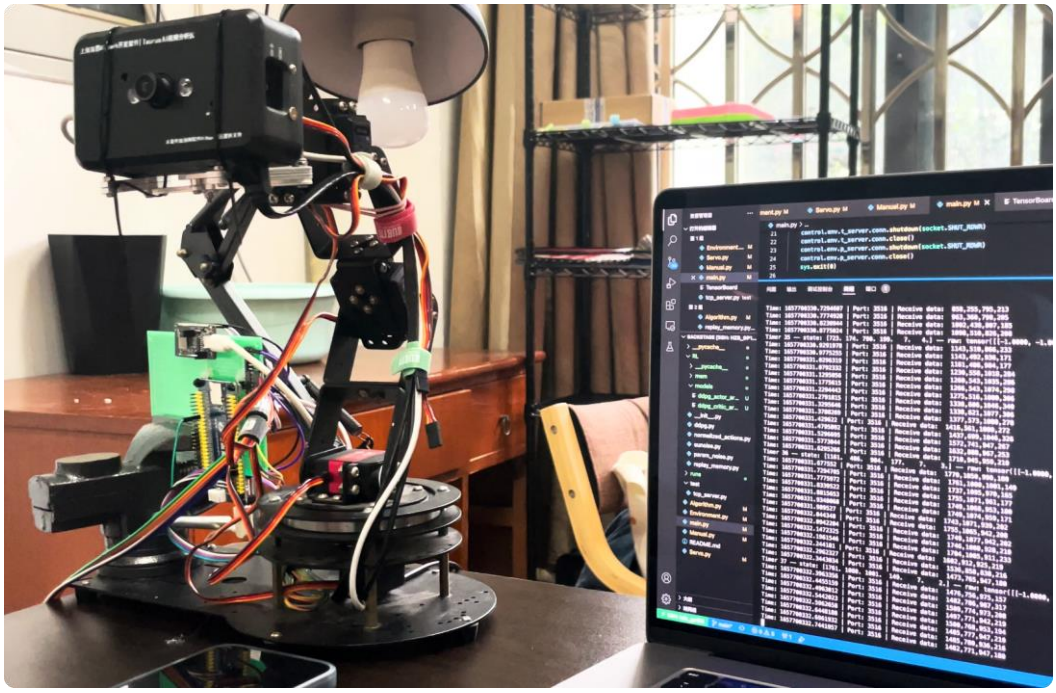


图 3-20 边缘节点接收定位信息并发送控制指令（右），机械臂受控移动（左）

机械臂稳定后，测试用户侧腰移动上半身，头部位置移动向右下方。图 3-21 中，可以看到 Pegasus 主控板在决策模型的控制下，紧跟用户行为调整机械臂姿态，再次将摄像头捕获的用户面部移到图像中央。

图 3-21 用户头部移动后机械臂调整摄像头朝向

进一步，我们测试了系统的鲁棒性，用将面部遮掩一半，模拟戴口罩、风帽等受限条件，测试系统的追逐性能。经验证，面部检测模型仍能准确标记人脸位置，决策模型也能正常控制机械臂行为。图 3-22 给出了遮挡一半面部的其中一种测试情况，机械臂正常完成了人脸追踪。

图 3-22 部分面部遮掩的鲁棒性测试

上述测试表明我们系统能够可靠运行，接下来我们将手机固定到机械臂抓取端的终端支架上，开启飞书会议模拟真实入会场景。图 3-23 测试了前置摄像头的入会场景，因为前摄距离 Taurus 套件主摄的距离较近，用户在会议软件中被很好地录制到了屏幕中心。

但是，由于我们选购的 20kg 舵机在真实使用情况下，实测抓取端无法承受太大压力，否则容易出现下垂或无法抬升的情况，我们难以完成长时间的支架测试。进而，我们额外设计了一种后置摄像头的拍摄模式，能够适用于直播拍摄、学术会议主讲人录制等场景，如图 3-24 所示。我们在后摄拍摄的模式下，进行了长时间的系统运行测试，经实验，该系统能有效完成面部追踪，且能适配以手机作为移动终端与会的场景。

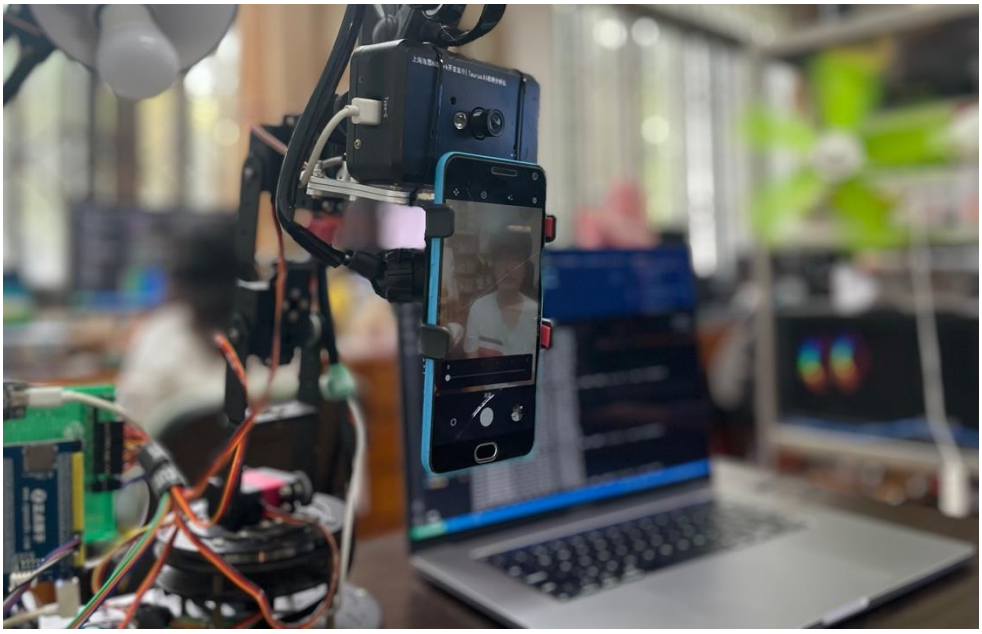


图 3-23 远程会议通过移动终端前摄与会测试

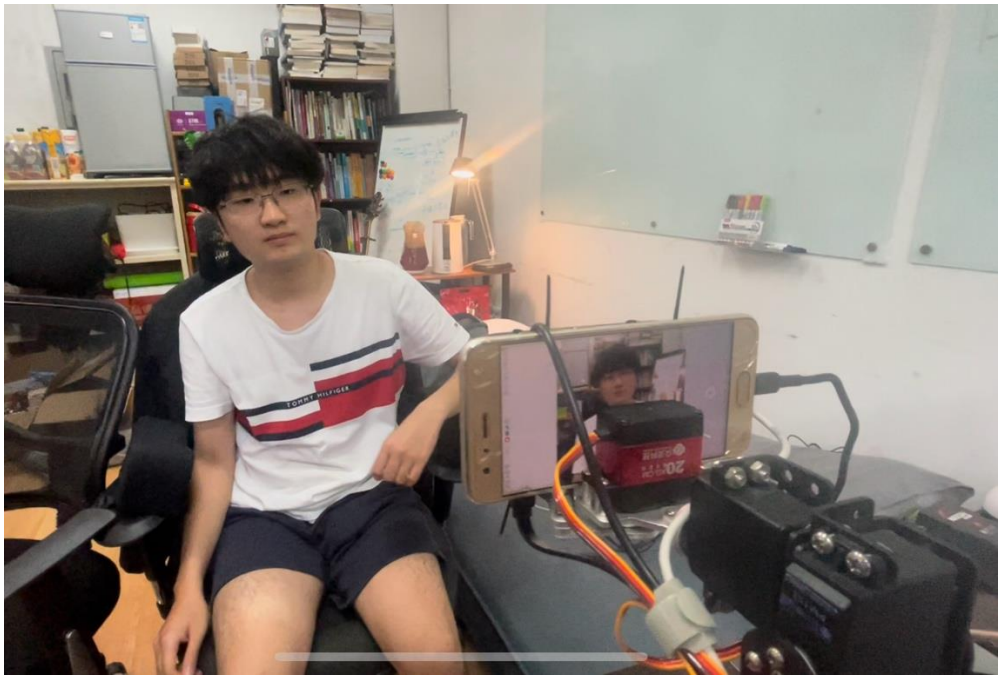


图 3-24 移动终端后摄拍摄与长时间带终端运行测试

第四部分　总结

4.1 可扩展之处

考虑到我们设计 MEC 系统的初衷源自物联网平台的缓存能力受限，未来如果海思嵌入式系统支持内存拓展，我们可以直接将边缘服务器的上年度强化学习运算内容迁移进嵌入式系统中。

在测试与实际使用中，我们发现标称 20kg 的舵机无法支撑我们手中半斤重的 iPhone 13 Pro Max，甚至在安装上我们改装的手机支架后，也会出现机械臂下垂的现象，因此我们选用了一台较轻的入门机型进行演示。在之后的改进中，我们首先会重新考虑舵机的选型，甚至可能重新选购整个机械臂模组，以保证抓取端能够稳定安装我们的全部硬件，并且固定市面上最重的手机。

除此之外，我们还有两个可能的扩展方向，一个是把整个系统和用户手机绑定，另一个是直接使用 Taurus 开发板作为终端设备参会。未来我们可以考虑直接将与会摄像头摄录内容与手机 APP 结合，将整个系统加入线上会议软件中，也可以直接用安卓手机的资源来完成姿态解算，降低嵌入式平台运算需求，进一步优化功耗。

4.2 心得体会

有幸尝试在我国自主研发的芯片上进行嵌入式软件开发，并深入了解到 OpenHarmony 的设计理念，通过整个项目流程，让我们加深了对行业发展的认知。我们看见了国内自研的艰辛与不易，把握优势的同时正视劣势，对我们选择未来的发展方向产生了深刻影响。

在本次项目中，我们第一次体验将神经网络模型部署到嵌入式平台的全过程，也第一次脱离仿真、在处处受限的条件下实现人工智能模型的训练与部署。在功耗、存储资源、计算能力的限制下，我们大胆尝试脱离传统网络部署方式，将论文中广泛讨论的 IoT MEC 模式应用于实践，有效利用物联网设备与边缘网络的特性，将模型复杂、资源要求高的 RL 算法引入物联网系统中，对我们来说是一次极为有趣的探索。

最后，感谢筹备此次大赛的全体工作人员，也祝愿此次大赛能圆满完成。

第五部分 参考文献

[1] Pegasus 开发套件实验指导手册

[2] Taurus & Pegasus 基础开发套件调试指导手册

[3] Taurus 计算机视觉基础开发套件操作指导

[4] 基于 Darknet 框架的 Yolo2 检测网进行 AI 应用开发与部署的指导手册

[5] 基于 Taurus 套件进行训练图片制作、模型训练、模型转换指导手册

[6] Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6), 26-38.

[7] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[8] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[9] Abbas, Nasir, et al. "Mobile edge computing: A survey." IEEE Internet of Things Journal 5.1 (2017): 450-465.

[10] Shi W, Cao J, Zhang Q, et al. Edge computing: Vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

[11] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

第六部分　附录

6.1 问题建模与算法推导

（1）状态机建模

在我们的系统中，核心是部署在边缘计算节点上的决策模型。这里的决策指的是根据用户面部在摄像头中的位置，决定机械臂应该变为何种姿态，以将人脸置于摄像头中央。

因此，整个问题中的决策本身是一个控制问题。对于控制问题，自然而然可以采用状态机进行建模，得到一张状态-行为选择图，如图 6-1 所示。

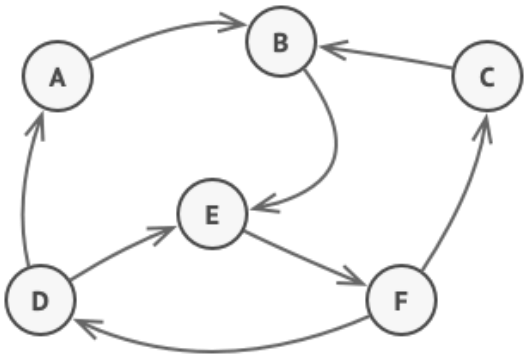


图 6-1 状态机示意图

我们定义状态 $s = (x_1, x_2, y_1, y_2)$，表示人脸在图片中的位置，状态空间 $S: x_1, x_2 \in [0,1920], y_1, y_2 \in [0,1080]$。行为 $a = (pos_{vert}, pos_{hori})$，表示机械臂在竖直方向的位置与在水平方向的位置，行为空间 A: $pos_{vert} \in [0,19]$, $pos_{hori} \in [0,9]$。

因此状态机就是在状态 $s_t \in S$ 下，采取行为 $a_t \in A$ 的一张选择表。基于状态机，我们搭建了一个最简单的决策模型作为对照组：

```
class Manual_Control:
    def execute(self):
        s = self.env.get_state()
        if s is None:
            self.rst_timeout += 1
            time.sleep(1)
            if self.rst_timeout == 5:
                self.env.reset()
                self.rst_timeout = 0
            return
        self.rst_timeout = 0
        right, left, bottom, top, vs, hs = s
        x_center = (left + right)/2
        y_center = (top + bottom)/2
        w = np.abs(right - left)
        h = np.abs(bottom - top)
```

```
        # 根据 x 调整左右
        offset_x = x_center/self.env.screen_width-0.5
        if offset_x > 0.1:
            act2 = hs + 1 + offset_x * 5
        elif offset_x < -0.1:
            act2 = hs - 1 - offset_x * 5
        else:
            act2 = hs
        act2 = np.clip(int(act2), 0, 10)
        # 根据 y 调整上下
        offset_y = y_center/self.env.screen_height-0.5
        if offset_y > 0.1:
            act1 = vs - 1 - offset_y * 5
        elif offset_x < -0.1:
            act1 = vs + 1 + offset_y * 5
        else:
            act1 = vs
        act1 = np.clip(int(act1), 0, 10)
        action = np.array([act1, act2])
        self.env.step(action=action)
```

（2）马尔可夫决策过程建模

状态机的策略是固定，它的好坏取决于设定状态-行为选择表时的考量是否周到。如果要找到一个最优的选择策略，且希望它能和环境共同进步，强化学习算法是近年来最优的解决方案。要使用强化学习算法，我们首先需要将问题建模成马尔可夫决策过过程（Markov Decision Process，MDP）。

与状态机类似地，我们定义状态 $s = (x_1, x_2, y_1, y_2)$，状态空间 $S: x_1, x_2 \in [0,1920], y_1, y_2 \in [0,1080]$。行为 $a = (pos_{vert}, pos_{hori})$，行为空间 A: $pos_{vert} \in [0,19]$, $pos_{hori} \in [0,9]$。

进一步，定义状态-行为转移概率 $P(s_{t+1} \mid s_t, a)$ 表示在状态 s_t 下选择行为 a 有多大的概率转移为状态 s_{t+1}。策略 $\pi(a|s_t)$ 表示在表示在状态 s_t 下选择行为 a 的概率。

回报函数 $R(s_t, a, s_{t+1})$ 示经历一次经验 $\{s_t, a, s_{t+1}\}$ 所带来的收益，回报R_t 是时刻 t 下智能体按照策略 π 为状态 s_t 选择行为 a 后的回报函数值。

于是我们可以定义 MDP 的优化目标为：

$$\max_{\pi} \sum_{t} \gamma^{t-1} R_t,$$

其中， $\gamma \in [0,1]$ 是折扣因子。

（3）深度强化学习算法设计

基于上述 MDP 建模，我们选择深度确定性策略梯度算法（Deep Deterministic Policy Gradient，DDPG）作为决策模型的算法。在策略梯度法

中，通常采用演员-评论家（Actor-Critic）框架。其中的 Actor 就是策略函数 π，它将在环境中根据状态 s_t 选择行为 a 并执行。Critic 则是对价值函数的估计，它根据状态 s_t 以及 Actor 选择的行为 a 来估计价值函数的值，并以此来评价 Actor 当前策略的好坏。其中 Critic 评价 Actor 的过程叫做策略评估 (Policy Evaluation)，Actor 根据评估结果来优化策略的过程叫做策略提升 (Policy Improvement)。

正如 DDPG 的名字所言，该算法使用确定性策略（Deterministic Policy），Actor 网络输出的是 2 个 [-1, 1] 的值，经过简单的映射可以量化成两个 [0, 9] 的整数，分别表示机械臂在竖直方向上与水平方向上的姿态档位，再由姿态解算算法对 DDPG 选择的行为映射成对舵机的控制。

在强化学习中，同策略（On Policy）指的是每次更新完策略后，都要收集一批经验 $\{s_t, a, s_{t+1}\}$（Experience）作为样本，然后使用这些经验进行下一次更新，然后清空经验重新收集；异策略（Off Policy）则不再使用在线收集到的经验，而是维护一个经验回放池 (Experience Reply Pool）以存放大量的历史经验，每次迭代时都从该池中取出一批经验来进行训练。我们所用的 DDPG 算法是一种 Off Policy 的算法，它对经验收集的要求较高，因此无法在 1GB 内存的 Taurus 板端实现决策模型的部署工作。进而，我们选用近年来火热的移动边缘计算（Mobile Edge Computing，MEC）框架，通过计算卸载的方式将物联网平台无法胜任的工作迁移到接入网边缘侧。

我们的经验回放池设计如下：

```
class ReplayMemory(object):
    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0

    def push(self, *args):
        """Saves a transition."""
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = Transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)
```

DDPG 算法决策过程如下：

```
class DDPG_Algorithm:
    def execute(self):
        s = self.env.get_state()
        if s is None:
            self.rst_timeout += 1
            if self.rst_timeout == 4:
                print("No human face detected. Now reseting the robot arm.")
```

```
                    self.env.reset()
                    self.rst_timeout = 0
                    time.sleep(1)
                time.sleep(0.5)
                return
            self.rst_timeout = 0
            try:
                right, left, bottom, top, vs, hs = s
            except:
                return
            x_center = (left + right)/2
            y_center = (top + bottom)/2
            offset_x = x_center/self.env.screen_width-0.5
            offset_y = y_center/self.env.screen_height-0.5
            self.exec_timer += 1
            # 1. 预测 action
            if custom_ounoise_sigma == -1:
                self.ounoise.sigma = max(0.2, MAX_OUNOISE_SIGMA - (self.train_timer/1e5))
            else:
                self.ounoise.sigma = custom_ounoise_sigma
            action_raw = self.agent.select_action(torch.Tensor([s]), action_noise=self.ounoise)    # (-1.,
1.)
            if not type(action_raw) is np.ndarray:
                action = action_raw[0, :].detach().numpy()
            else:
                action = action_raw[0, :]
            action = np.clip(action, -1., 1.)
            # 2. 修改 action
            # action 的输出是偏差量
            if offset_y > 0.05: dy = -1       # 脸在下面，参数越小越下
            elif offset_y < -0.05: dy = 1
            else: dy = 0
            action[0] = dy * ((action[0]+1.)/2.*5 + 1) + vs
            if offset_x > 0.05: dx = 1        # 脸在右边，参数越大越右
            elif offset_x < -0.05: dx = -1
            else: dx = 0
            action[1] = dx * ((action[1]+1.)/2.*5 + 1) + hs
            action = np.clip(action, 0, 10)
            action = np.round(action)
            # 3. 执行 action
            s_, reward, done, info = self.env.step(action=action)
            # 4. 记录经验
            self.push_memory(s, action_raw.detach().numpy(), done, s_, reward)
```

```
def train(self):
        memory = self.memory
        if len(memory) > batch_size:
                for _ in range(updates_per_step):
                        transitions = memory.sample(batch_size)
                        batch = Transition(*zip(*transitions))
                        self.agent.update_parameters(batch)

def push_memory(self, state, action, done, next_state, reward):
        state = torch.Tensor([state])
        action = torch.Tensor(action)
        mask = torch.Tensor([not done])
        next_state = torch.Tensor([next_state])
        reward = torch.Tensor([reward])
        self.memory.push(state, action, mask, next_state, reward)
```

6.2 移动边缘计算

根据 ETSI 的定义，边缘计算系统主要由用户和边缘服务器构成，其典型系统 架构如图 6-2 所示。为了充分发挥边缘计算的架构优势，服务提供商需要在整个网 络中尽可能密集地部署边缘服务器。因此，在当前的实践中，除了基站，无线接 入点、路边单元、路由器等具备无线通信和计算能力的设备都可以充当边缘服务器 [10]。用户端则主要包括智能手机、平板电脑、智能车辆以及各类传感器等多样 化设备。这些终端设备往往自身也具备一定的计算能力，可以根据当前情况灵活地选择是否将计算任务卸载到邻近的边缘服务器上。由于边缘服务器的计算能力有限，面对突发式的任务卸载可能难以应对。为了优化整个系统的表现，边缘服务器也可以将一部分对时延不那么敏感的任务进一步上传到云端进行处理，从而实现用户层——边缘层——云层三端协同的垂直分层架构。因此，边缘计算更像是云计算的扩展，而不是代替。

图 6-2 边缘计算系统架构示意图

本质上，边缘计算是云计算模式从核心网向网络边缘的下沉。由于离用户更近，边缘计算相比于传统云计算在很多方面都具有天生的优势。因此，我们

在进行系统设计时，考虑到物联网平台对决策延迟的高要求，摒弃了传统远端部署的模式，而是选用更加新颖高效的边缘计算，将高算力需求的决策模型卸载到边缘计算节点上，以探索探索高复杂 DRL 在 IoT 中的可能性。

(a)

(b)

图 6-3 我们实现的移动边缘方案

我们使用一台连接无线网关的服务器作为边缘基站，无线网关只实现二层接入与转发功能，服务器中的 Openwrt（KoolShare）提供网络功能虚拟化（Network Function Virtualization，NFV）后的基站接入控制服务，该基站虚拟化出切割出的其他计算存储资源用于提供 MEC 服务。

通过在 Taurus 板端部署 CV 模型进行目标检测，在 MEC 部署 DRL 模型实现行为决策，在 Pegasus 板端解析决策指令进行行为控制，我们实现了一套完整的计算卸载流程。在运行过程中，MEC 将统计所有状态、决策、结果，进行训练，实现让决策模型“自适应”环境的效果。