Posture-Controled Camera

IAVI final project

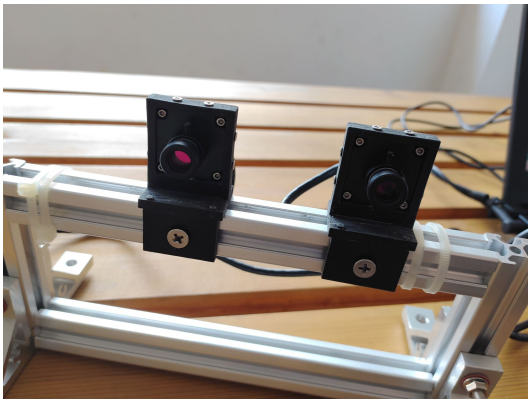
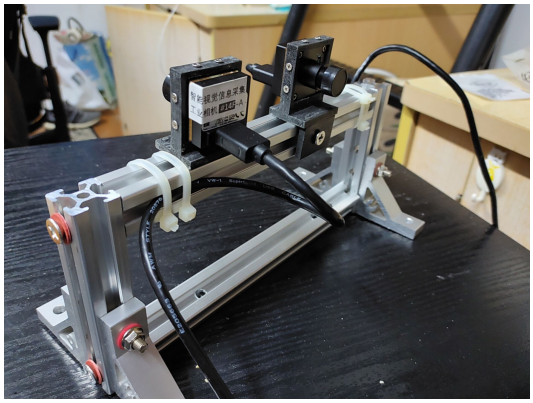
第二组 焦笑然 温子奇 张雯琪

提交文件	说明
report.pdf	提交报告
pre.pptx	展示PPT
\code\Action\	动作识别模块
\code\Action\trainning\	训练出的单标签分类模型
\code\Pose\	姿态识别模块
\code\Pose\graph_models\	封装后的VGG和mobilenet模型
\code\test_out\	保存多组关节数据的txt文件，用于动作识别模型训练
\code\Tracking\	实时追踪模块
\demo\	项目实现效果演示
\demo\selfie\	新增的自拍效果演示

1 项目简介

通过单个相机完成对相机前人物姿势的识别（称为主控相机），从而控制另一个相机（称为被控相机）进行各种操作，包括但不限于启动快门，调整常用的相机参数等操作。

如下图所示，两种相机摆放方式可实现不同的拍摄效果，左侧同向摆放附加延时1.5秒可实现较好地自拍效果（效果图见 \demo\selfie\），右侧不同向摆放可进行实时控制相机拍摄对面物体(效果图见 \demo\)

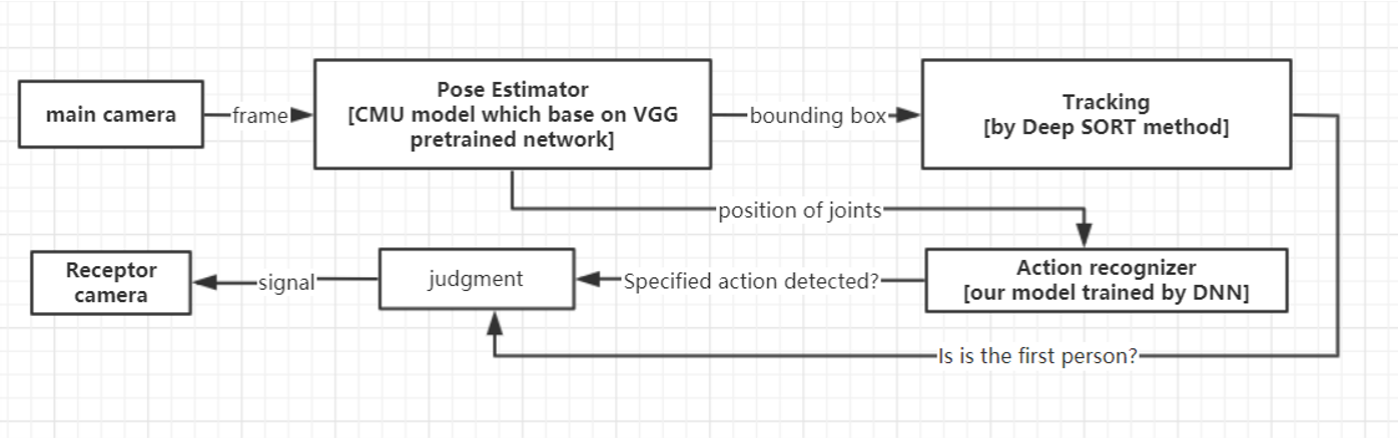



对于这一想法的实际应用，被控设备完全可以不限于相机，比如其他的智能设备（电脑、汽车、智能家居），可以通过这一技术实现质量更好的人机交互，即使是对于相机本身，也可能够实现更为方便

的自拍功能（不再需要提前设置快门启动时间，可以通过手势或姿势直接启动快门）。

2 项目原理和实施

2.1 流水线



项目实施的流程如下：

由**主控相机**读取每一帧的图像数据，并实时传入**姿态估计**模块；

在**姿势识别**模块中，标记出所有骨骼的位置，同时在图像上绘制出骨骼点，将关节位置传给**动作识别**模块；同时，通过关节位置计算出表示对象位置的bounding box信息传给**实时追踪**模块；

动作识别模块通过我们自己实现的一个DNN模型来完成，通过关节点位置作为输入，判断是否检测到了能够驱动相机的特定动作（在本项目中，通过挥手wave这一动作来驱动相机）

实时追踪模块能够区别出在相机中的所有人并实时追踪，为了避免路过的路人可以控制相机，我仅以第一个识别出的人的动作为准（即ID=1的人）；

最后由**动作识别**和**实时追踪**模块共同给出驱动**被控相机**的信号，完成从主控相机到被控相机的姿态控制。

- 关键代码如下（详见main.py文件）：

```
#获取被控相机
cap_Receptor = EasyPySpin.VideoCapture(0)

# 获取主控相机
cap_main = choose_run_mode(args)

# 读写视频文件
video_writer = set_video_writer(cap_main, write_fps=int(7.0))

# 保存多组关节数据的txt文件，用于训练过程(for training),重命名为wave_*.txt/stand_*.txt,数据处理为.csv文件，用于/Action/Training中的动作识别模型训练
# f = open('test_out/origin_data.txt', 'a+')

while cv.waitKey(1) < 0:
    has_frame, show = cap_main.read()
```

```
    if has_frame:
        fps_count += 1
        frame_count += 1

        # 灰度图像转为RGB图像
        show=cv.cvtColor(show, cv.COLOR_GRAY2RGB)
        # 姿势识别,human存储着标记好骨骼的数据（以图像的相对位置表示）
        humans = estimator.inference(show)
        # get pose info, pose中存储着绘制好关键节点的姿势数据，主要目的是计算出关节的实际坐标位置，绘制骨骼，并返回整理完后的数据
        pose = TfPoseVisualizer.draw_pose_rgb(show, humans,)  # return frame, joints, bboxes, xcenter
        # recognize the action framewise
        show = framewise_recognize(pose, action_classifier,cap_Receptor)

        cv.imshow('Gesture control camera based on OpenPose', show)
        video_writer.write(show)

input('The program finish!\n')

video_writer.release()
cap_main.release()
cap_Receptor.release()
```

2.2 姿态估计模块

姿态估计的目标是在RGB图像或视频中标记出人体的形状（在本项目中针对的是视频输入），并将标记了人体骨骼关键点的数据以图片/视频/JSON等多种格式进行输出。如下图所示

姿态估计包括单人/多人姿态估计，2D/3D姿态估计，以及人体姿态追踪等多个子方向。

目前有相当多的人体姿态估计的深度学习开源框架，覆盖了以上的各个方向，并且可以实现实时的姿态估计，例如OpenPose，DensePose等等，其中OpenPose是一个多人、实时，采用2D方法的姿势识别项目，非常符合我们本次project的需求，因此本次实验我们选择了OpenPose框架来实现。

> OpenPose
> OpenPose人体姿态识别项目是美国卡耐基梅隆大学（CMU）基于卷积神经网络和监督学习并以caffe为框架开发的开源库。可以实现人体动作、面部表情、手指运动等姿态估计。适用于单人和多人，具有极好的鲁棒性。是世界上首个基于深度学习的实时多人二维姿态估计应用。

> 论文地址：https://arxiv.org/pdf/1812.08008.pdf
> GitHub地址：https://github.com/CMU-Perceptual-Computing-Lab/openpose

本项目中第一个需要处理的内容就是将主控相机所获取的图像，用相应的方法完成姿态估计后，将输出的带有标记的数据传入下一个部分进行进一步的处理，以具体地判断人物的动作。本次实验我们采用的标记骨骼模式为官网提供的COCO模式，如上图所示，共标记了18个人体关节位置并以相对于图片上的xy坐标来表示，输出18组共36个数据作为动作检测模块的输入。

在OpenPose提供的训练模型中，我们首先用了主流的VGG模型来训练，识别的精度很高但帧率非常低，只有3-6FPS，改用Google提出的移动端模型MobileNet这个相对小一点的模型能实现较高的帧率，但是精确度上又低了很多，骨架抖动现象明显，在某些情况下还会出现骨架的错位，将导致**动作识别**模块的处理的难度增加，因此我们最后还是选择了VGG模型。

2.3 实时追踪模块

实时追踪模块参考了基于多目标跟踪的deep SORT(Simple Online And Realtime Tracking)算法。

多目标跟踪，即Multiple Object Tracking(MOT),其主要任务是给定一个图像序列，找到图像序列中运动的物体，并将不同帧中的运动物体一一对应,然后给出不同物体的运动轨迹。多目标跟踪被视为数据关联问题，在视频帧序列中进行跨检测结果的关联，为了解决数据相关的问题，跟踪器使用了多种方法对运动过程和运动目标的外观特征进行建模。在Deep SORT中，通过使用更加可靠的度量来代替关联度量，同时采用CNN网络在大规模行人数据集进行训练并提取特征，能有效增加网络对遗失和障碍的鲁棒性，符合本次实验要求。

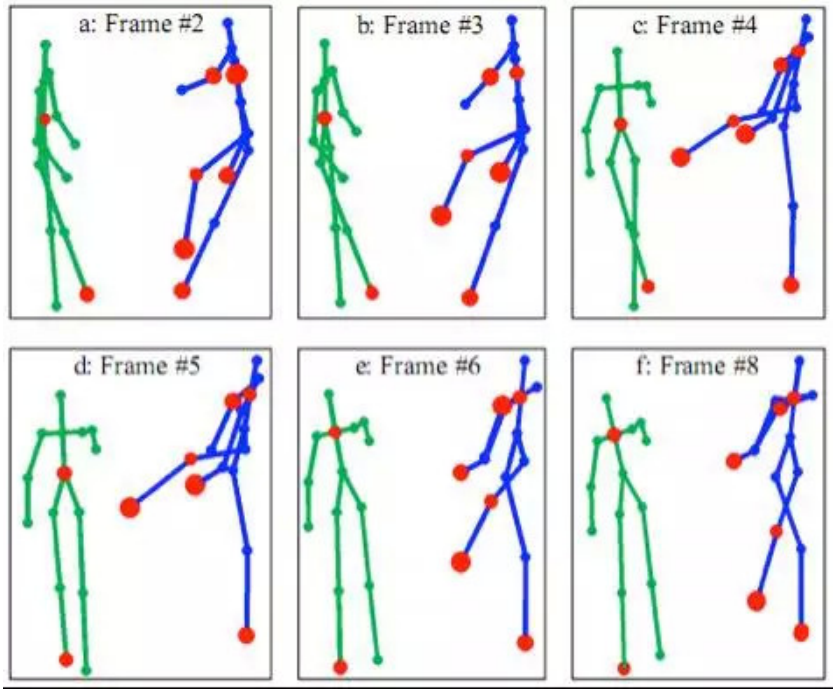
考虑到我们在使用姿态控制相机的过程中，相机容易受到多人姿态改变的干扰，我们引入实时追踪模块来确保同一时间相机只针对一人的姿势改变做出响应。

追踪模块的代码与动作识别模块的代码在同一个py文件内，将在2.4中给出。

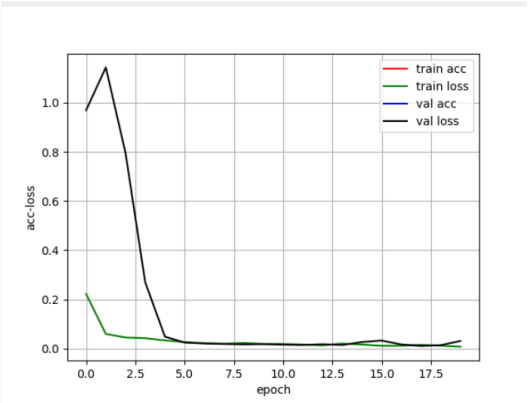
2.4 动作识别模块

动作识别的意义在于判断目标视频中人物行为的类别。

在获取了人物骨架图的基础上，可以通过特定骨骼的运动速度和方向，通过简单的条件判断来识别一些简单的运动（例如下蹲：通过重心向y轴负方向移动超过一定速度就判断为下蹲）。挥拳行为识别示例：



本项目的主要任务是实时地识别出被拍摄对象的特定动作，识别的效率与准确性是最为重要的。本次实验我们首先采集了一定量的数据，训练出一个单标签的多分类模型，通过不断调整模型的参数和训练数据集的大小，最终成功将train loss和test loss控制在0.01以下



随后根据姿态估计模块中给出的18组36个关节数据输入来判断属于哪种动作。
受时间关系的限制，本次实验我们仅定义了wave和stand两种动作，并且在实时追踪模块的基础上完成了指定动作实现相机拍摄的功能。不过在以下代码中还是给出了一部分被注释的代码：是一段驱动相机修改其增益参数的代码，只要模型能够识别出指定的动作，这段代码便能运行。

考虑到为了更方便地进行自拍（本项目的初衷），相机不会在识别到wave的一瞬间就进行拍摄，而是在识别出wave的动作后等待1.5s事件后再进行拍摄，从而给被拍摄对象留出时间选择一个自己的pose。

- 关键代码如下（详见 \Action\Recognizer.py）

```
def framewise_recognize(pose, pretrained_model,camera):
    if bboxes:
        bboxes = np.array(bboxes)
        features = encoder(frame, bboxes)

        # 调用tracker并实时更新
        tracker.predict()
        tracker.update(detections)
```

```
        # 记录track的结果，包括bounding boxes及其ID
        trk_result = []
        for trk in tracker.tracks:
            if not trk.is_confirmed() or trk.time_since_update > 1:
                continue
            bbox = trk.to_tlwh()
            trk_result.append([bbox[0], bbox[1], bbox[2], bbox[3], trk.track_id])
            # 标注track_ID
            trk_id = 'ID-' + str(trk.track_id)
            cv.putText(frame, trk_id, (int(bbox[0]), int(bbox[1]-45)), cv.FONT_HERSHEY_SIMPLEX, 0.8,
trk_clr, 3)
            # 找出第一个进入镜头的人的ID
            if fir=='true':
                fir='false'
                fir_ID=trk.track_id
                print("First person ID:{}".format(fir_ID))

        for d in trk_result:
            xmin = int(d[0])
            ymin = int(d[1])
            xmax = int(d[2]) + xmin
            ymax = int(d[3]) + ymin
            id = d[4]
            try:
                # xcenter是一帧图像中所有human的1号关节点（neck）的x坐标值
                # 通过计算track_box与human的xcenter之间的距离，进行ID的匹配
                tmp = np.array([abs(i - (xmax + xmin) / 2.) for i in xcenter])
                j = np.argmin(tmp)
            except:
                # 若当前帧无human，默认j=0（无效）
                j = 0

            # 进行动作分类
            if joints_norm_per_frame.size > 0:
                joints_norm_single_person = joints_norm_per_frame[j*36:(j+1)*36]
                joints_norm_single_person = np.array(joints_norm_single_person).reshape(-1, 36)
                pred = np.argmax(pretrained_model.predict(joints_norm_single_person))
                init_label = Actions(pred).name
                # 显示动作类别
                cv.putText(frame, init_label, (xmin + 80, ymin - 45), cv.FONT_HERSHEY_SIMPLEX, 1,
trk_clr, 3)
                # 检测到指定动作，拍摄照片,瞬间拍照
                # if (init_label == 'wave' and last_init_label !='wave' and capture_picture_count <= 10
and time.time() - start_time > 2 and id == fir_ID):
                #     ret, frame = camera.read()
                #     filename = "img/getPicutre-{0}.jpg".format(capture_picture_count)
                #     cv.imwrite(filename, frame)
                #     print("Image saved at {}".format(filename))
                #     capture_picture_count=capture_picture_count+1
                #     start_time=time.time()

                # 检测到指定动作，在1.5s后拍照
                if(init_label == 'wave' and getSignal == 'false'):
                    getSignal='true'
                    print("Prepare to get image\n")
                    start_time=time.time()

                if(getSignal == 'true' and time.time()-start_time > 1.5):
                    getSignal='false'
                    ret, frame = camera.read()
                    filename = "img/getPicutre-{0}.jpg".format(capture_picture_count)
```

```
                    cv.imwrite(filename, frame)
                    print("Image saved at {}\n".format(filename))
                    capture_picture_count=capture_picture_count+1

                # 检测到指定动作，改变相机参数，待实现
                # else if(init_label == 'XXX' and last_init_label !='XXX' and time.time() - start_time >
1 and id == fir_ID)
                #     myGain=cap.get(cv2.CAP_PROP_GAIN)
                #     if(myGain < 20)
                #         cap.set(cv2.CAP_PROP_GAIN, myGain+2) #调整增益
                last_init_label=init_label
            # 画track_box
            cv.rectangle(frame, (xmin - 10, ymin - 30), (xmax + 10, ymax), trk_clr, 2)
    return frame
```

3 项目难点

3.1 相机驱动

相机驱动的主要问题是FLIR并没能提供打包过的python驱动代码，我们一开始采用通过C++扩展python模块或者编译dll动态链接库的方式来实现了在python中内嵌C++的相机驱动，例如通过`import ctypes`，`ctypes.cdll.LoadLibrary()`来调用C++生成的dll文件（dll文件的内容是驱动相机的相应代码，例如拍照）。

使用这个方式驱动被控相机很容易，将相关函数封装在dll文件内，并且无需考虑参数传递的问题，无需返回值和传入的参数，直接调用即可；

但是主控相机的输入，通过这一方式较难实现，由于通过C++的驱动代码获取到的相机是一种Spinnaker自己定义的一个相关的数据结构，以及通过驱动代码获取到的图像数据也是Spinnaker自定义的数据结构，无论是二者中的哪一个，传输到python代码中都需要转换数据结构，复杂数据结构的转化存在很大困难。

3.2 接口匹配

本次项目的实现参考了部分现有的模型（例如OpenPose和Deep SORT），但在模型的输入输出接口上面容易出现错误：

比如FILR相机的输入无法契合OpenPose模型中的Placeholder，最后发现是实时输入的灰度图像（单通道）必须转成RGB三通道，再输入给网络。

3.3 训练DNN模型的数据问题

动作识别模块的DNN模型由我们自己采集数据并实现，其中可能存在因为数据偏差而导致的训练出的模型存在偏差，例如训练出的模型会在人物在镜头右半边时，始终将人物识别为stand的状态，原因在于我们在采集数据时，仅在右半边的镜头中进行了挥手的动作，而没有在左侧进行挥手，导致训练的模型有所误差。

同时，整个工程的流水线中的任何一个环节出现改变，在动作识别模块中的模型就最好需要重新采集并训练；例如当我们将OpenPose的基于VGG16的模型改为基于moblileNet的模式时，需要重新采集关节数据并进行训练，这是由于二者采集到的骨骼关节数据仍然存在一定的不同，可能导致更换模型时，动作识别模块的准确性会大幅下降。

4 项目拓展

4.1 模型封装

取消计算机作为中间控制设备，将**主控相机->计算机->被控相机**的间接控制模式转化为**主控相机->被控相机**的直接控制模式。

本次实验项目最初是准备在AI相机上载入神经网络来直接实现动作识别功能，查询相关资料后发现现有的AI相机能支持载入的网络较小，并不能较好达到实验需求。

想要实现模型封装的效果有两种方案，一是简化项目流水线，本次实验我们主要采用了3层网络模型，能较为精准地采集数据和控制相机，可以在保证相机正常控制的条件下进行简化，以达到封装的效果；二是调研功能更加完备的AI相机，达到成功载入模型的效果。

4.2 扩展动作控制

本次实验由于时间限制我们仅进行了wave和stand两种动作的识别来控制相机拍摄，而OpenPose提供的COCO骨架模型共有18组关节数据，支持更多动作的识别，可以实现放大、缩小增益，增加、减少曝光等多种效果，使相机控制更为智能。

4.3 扩展控制对象

实现了动作识别的功能之后可以扩展控制对象，并不局限于调整相机的参数，具体可应用于智能家居等更多人机交互应用。

5 Dependencies

运行本工程的代码需要如下依赖：

- python >= 3.5
- Opencv >= 3.4.1
- sklearn
- tensorflow =1.14.0 & keras=2.1.0
- numpy & scipy
- pathlib

- pyspin

6 参考资料

1 行为识别综述

2 姿态估计与行为识别（行为检测、行为分类）的区别

3 六种人体姿态估计的深度学习模型和代码总结

4 行为动作识别

5 FireFly-DL 相机介绍

6 OpenPose

7 TF-Pose-Estimation

8 Deep SORT