

# AutoDriving：无人机项目第一阶段报告

Author: 邓琛龙 2018202077

Group ID: 7

AutoDriving: 无人机项目第一阶段报告

## 一、第一阶段目标简述

- 1.1. 购买DJI Tello无人机，学习掌握Tello SDK以及相关接口
- 1.2. 实现简单语音识别，并利用其操控无人机执行简单操作
- 1.3. 实现视频流处理，能检测追踪简单目标

## 二、第一阶段技术路线

- 2.1. 语音路线
- 2.2. 视频路线

## 三、阶段性成果展示

## 四、下一阶段展望

注：第一阶段进行时间：**2020.10.9—2020.10.30**

## 一、第一阶段目标简述

### 1.1. 购买DJI Tello无人机，学习掌握Tello SDK以及相关接口

购买Tello无人机后，学习相关SDK以及接口使用，尽量做到熟悉相应操控接口。

### 1.2. 实现简单语音识别，并利用其操控无人机执行简单操作

利用相关语音识别引擎或自训练分类技术，使无人机能够听懂简单指令并作出相关动作。

### 1.3. 实现视频流处理，能检测追踪简单目标

利用Tello上自带的摄像头向电脑终端传输图像，并在电脑上实现图像处理，返回相应指令操控无人机。

## 二、第一阶段技术路线

### 2.1. 语音路线

我们首先利用电脑的麦克风或耳机进行收音，并将一段时间内的音频进行采样处理。

```
def record():
    CHUNK = 1024
    FORMAT = pyaudio.paInt16      #量化位数
```

```

CHANNELS = 1 #采样管道数
RATE = 16000 #采样率
RECORD_SECONDS = 2
WAVE_OUTPUT_FILENAME = "output.wav" #文件保存的名称
p = pyaudio.PyAudio() #创建PyAudio的实例对象
stream = p.open(format=FORMAT, #调用PyAudio实例对象的open方法创建流stream
                 channels=CHANNELS,
                 rate=RATE,
                 input=True,
                 frames_per_buffer=CHUNK)
frames = [] #存储所有读取到的数据
print('* 开始录音 >>>') #打印开始录音
for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
    data = stream.read(CHUNK) #根据需求，调用Stream的write或者read方法
    frames.append(data)
print('* 结束录音 >>>') #打印结束录音
stream.close() #调用Stream的close方法，关闭流
p.terminate() #调用pyaudio.PyAudio.terminate() 关闭会话
wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb') #写入wav文件里面
wf.setnchannels(CHANNELS)
wf.setsampwidth(p.get_sample_size(FORMAT))
wf.setframerate(RATE)
wf.writeframes(b''.join(frames))
wf.close()

```

然后将录制结束的音频文件output.wav批量转送到百度开放平台的语音处理系统中，进行识别处理，进一步得到我们的语音指令内容

```

def cognitive(): #读取文件
    def get_file_content(filePath):
        with open(filePath, 'rb') as fp:
            return fp.read()

    result = client.asr(get_file_content('output.wav'), 'wav', 16000, {
        'dev_pid': 1537, #识别本地文件
    })
    result_text = result["result"][0]

    print("you said: " + result_text)

    return result_text

```

下一步便可以利用解析出来的指令，向Tello发送相关指令信号，指引其执行相关操作：

```

def action():
    if result == "开始。":
        ...
        command = "command"
        send( command )
        ...

        print("开始")
    if result == "起飞。":
        #send("takeoff")
        drone.takeoff()
        print("tello无人机起飞")

.....

```

## 2.2. 视频路线

视频路线我们主要想法是使用opencv中相关方法，对Tello返回的图像进行处理，并检测目标（如人脸）做出相应的动作。

首先我们需要从Tello中取回视频流的关键帧：

```

# 读入Tello图像
def telloGetFrame(myDrone, w= 360,h=240):
    myFrame = myDrone.get_frame_read()
    myFrame = myFrame.frame
    img = cv2.resize(myFrame,(w,h))
    return img

```

然后我们利用已经训练好的人脸识别模型，对关键帧中的图像数据进行人脸识别操作，并返回人脸的坐标位置：

```

def findFace(img):
    faceCascade =
    cv2.CascadeClassifier('./face_recognition/haarcascade_frontalface_default.xml')
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(imgGray, 1.1, 6)

    myFaceListC = []
    myFaceListArea = []

    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,0,255), 2)
        cx = x + w//2
        cy = y + h//2

```

```

area = w*h
myFaceListArea.append(area)
myFaceListC.append([cx,cy])

if len(myFaceListArea) !=0:
    i = myFaceListArea.index(max(myFaceListArea))
    return img, [myFaceListC[i],myFaceListArea[i]]
else:
    return img,[[0,0],0]

```

下一步可以通过查找和上一次分析中人脸图像位置的差，来控制Tello飞行的转向、运行速度，以达到追踪人脸的目的。

```

# 追踪人脸
def trackFace(myDrone,info,w,pError):

    ## PID
    error = info[0][0] - w//2
    speed = pid[0]*error + pid[1]*(error-pError)
    speed = int(np.clip(speed,-100,100))

    print(speed)
    if info[0][0] !=0:
        myDrone.yaw_velocity = speed
    else:
        myDrone.for_back_velocity = 0
        myDrone.left_right_velocity = 0
        myDrone.up_down_velocity = 0
        myDrone.yaw_velocity = 0
        error = 0
    if myDrone.send_rc_control:
        myDrone.send_rc_control(myDrone.left_right_velocity,
                               myDrone.for_back_velocity,
                               myDrone.up_down_velocity,
                               myDrone.yaw_velocity)

    return error

```

### 三、阶段性成果展示

语音方面，已经能够实现对所有常用命令的识别和操控，样例语音识别结果如下：

```
* 开始录音 >>>
* 结束录音 >>>
you said: 起飞。
起飞。
Tello: 20:59:56.424: Info: set altitude limit 30m
Tello: 20:59:56.424: Info: takeoff (cmd=0x54 seq=0x01e4)
tello无人机起飞
Tello: 20:59:56.430: Info: recv: ack: cmd=0x54 seq=0x0000 cc 60 00 27 b0 54 00
```

img1:起飞相应命令的识别、执行

并且，我们录制了相应视频展示。视频存放位置为百度网盘

链接: <https://pan.baidu.com/s/1n9AiKVfvzHp84XZP-VlgWQ>

提取码: i7j0

视频方面，我们已经实现了对人脸的检测，同时无人机会自动对检测到的人脸进行追踪运动。

face\_recognition > main.py > ...

```
1 from utils import *
2
3 w,h = 360,240
4 pid = [0.4,0.4,0]
5 pError = 0
6 startCounter = 0 # for no Flight 1 - for flight 0
7
8
9 myDrone = initializeTello()
10
11 while True:
12
13     ## Flight
14     if startCounter == 0:
15         myDrone.takeoff()
16         startCounter = 1
17
18     ## Step 1
19     img = telloGetFrame(myDrone)
20     ## Step 2
21     img, info = findFace(img)
22     ## Step 3
23     pError = trackFace(myDrone)
24     print(info[0][0])
25     cv2.imshow('Image',img)
```

问题 4 输出 调试控制台 终端

File "/Users/dengchenlong/Documents/大三上学期2020/人工智能导论/Code/face\_recognition/main.py", line 21, in <module>
 img, info = findFace(img)
File "/Users/dengchenlong/Documents/大三上学期2020/人工智能导论/Code/face\_recognition/utils.py", line 30, in findFace
 faces = faceCascade.detectMultiScale(imgGray,1.1,6 )
cv2.error: OpenCV(4.2.0) /Users/travis/build/skvark/opencv-python/opencv/modules/objdetect/src/cascadedetect.cpp:1689: error: (-215:Assertion failed) !src.empty() in function 'cv::cvtColor'

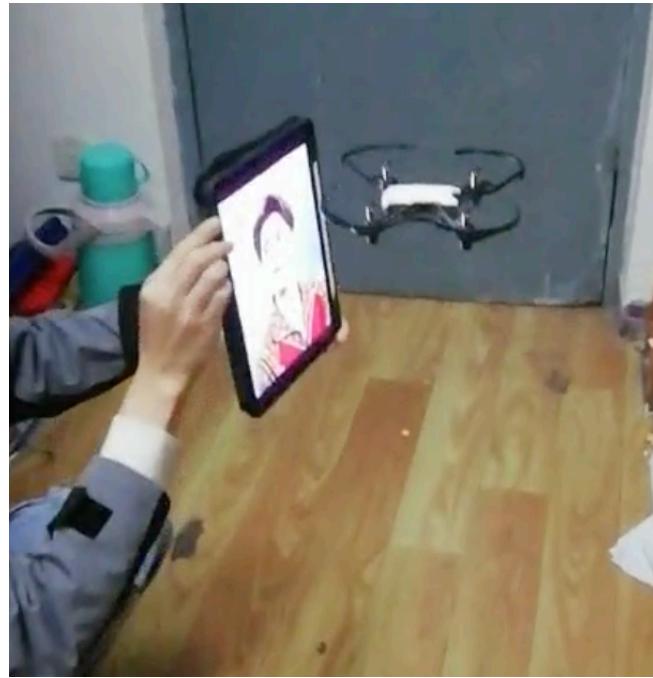
[h264 @ 0x7f9f098f4400] error while decoding MB 16 40, bytestream -8

Tello exited with code=1 in 68.000 seconds

行 1, 列 20 空格: 4 UTF-8 LF Python



人脸追踪识别：电脑端视角



人脸追踪识别：无人机端视角

同样地，我们录制了相应视频作为展示。链接与提取码与上面语音识别相同，为同一个链接下的两个视频。

#### 四、下一阶段展望

下一阶段，我们将在已有的基础上继续实现更高难度的目标寻找、自动飞行功能，并将根据难度赋予 Tello 不同程度的自我创作飞行动作功能。同时，针对第一阶段中存在的部分问题（如识别依赖网络等问题），我们也将寻找方法作出改进！

# AutoDriving：无人机项目第二阶段报告

Author: 邓琛龙 2018202077

Group ID: 7

AutoDriving: 无人机项目第二阶段报告

- 一、第二阶段实现成果概览
- 二、实现功能 & 实现过程详解
  - 2.1. 实现了基于深度学习模型的语音识别
  - 2.2. 手势识别
  - 2.3. 基于YOLO算法的目标检测
  - 2.4. 实现目的地飞行的GUI界面
- 三、下一阶段计划

注：第二阶段进行实践：**2020.10.30-2020.11.20**

## 一、第二阶段实现成果概览

- 1.1. 实现了基于深度学习模型的语音识别，逐步摆脱了对封装API的依赖
- 1.2. 初步实现了基于YOLO算法的实时目标检测，并计划利用此模型后续实现多种操作
- 1.3. 为目标飞行功能制作了地图GUI，方便后续功能实现

## 二、实现功能 & 实现过程详解

### 2.1. 实现了基于深度学习模型的语音识别

在第一阶段中，我们是以实现大致功能为导向。但在第二阶段乃至最后整个项目的完成中，我们会逐步摆脱对各种封装API的依赖，转向使用训练/预训练的模型进行离线操作，增强项目的鲁棒性。

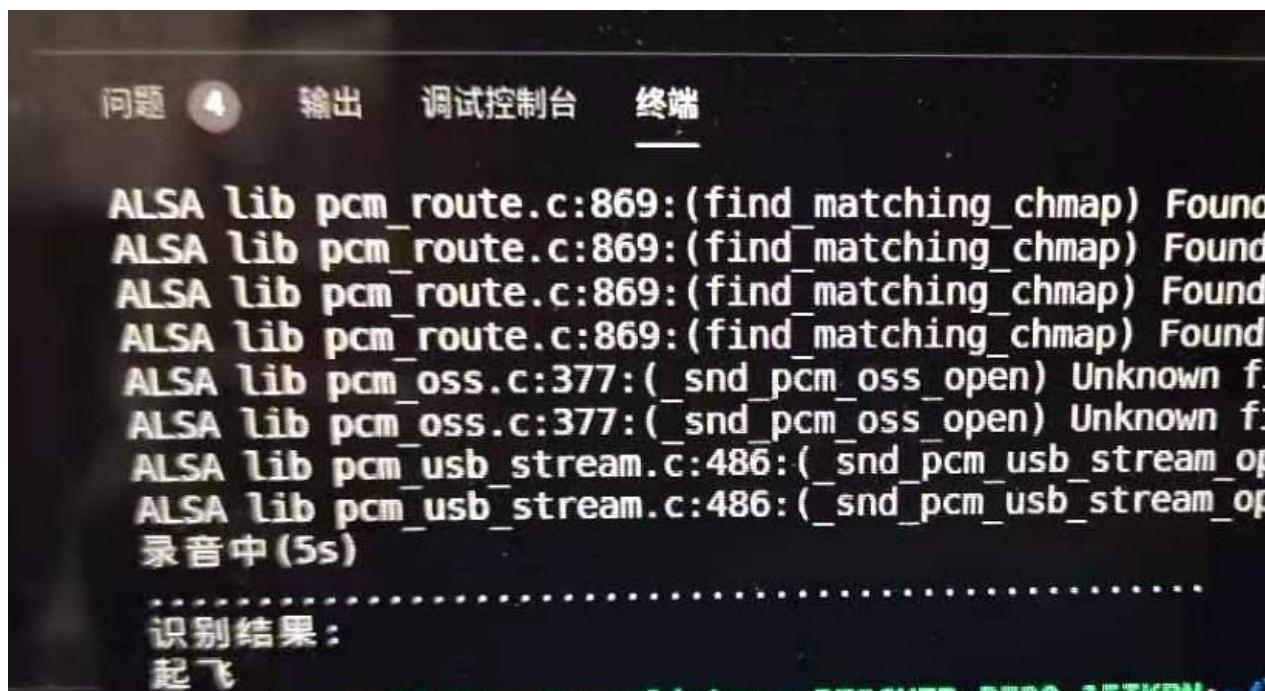
我们首先解决的是语音识别的离线识别实现。原先在语音操控无人机方面，主要依赖百度的语音识别API进行相关功能开发和应用。但在本阶段，我们依赖一个预训练的深度学习模型，较好的解决了离线语音识别的问题。

本模块功能，我们主要是基于一个称为Mandarin Automatic Speech Recognition的中文语音识别模型进行开发。该模型使用的是门控卷积神经网络（Gated Convolutional Network），激活函数使用的是GLU。预训练使用的数据集为AISHELL-1数据集，共150h。

但单纯的深度学习模型还并不能达到一个很好的效果，可能会出现一些滑稽的效果。例如：你说“你好很高兴认识你”，却被模型识别成了“利好很高性任实米”。若按字来评价准确率，一定是一个极其糟糕的模型。但我们会发现，尽管出来的结果十分滑稽，但在发音上和原句还是比较接近的。一个很自然的想法产生——可以通过语言模型来提升准确率。

语言模型的用处在于，当发音近似时模型会更倾向于生成语义相关的句子。我们使用了一个大小约为2G的已训练好的开放语言模型，加入到识别模型中。

模型训练安装好后，我们在一些常用语音短语中测试了效果，部分截图如下：

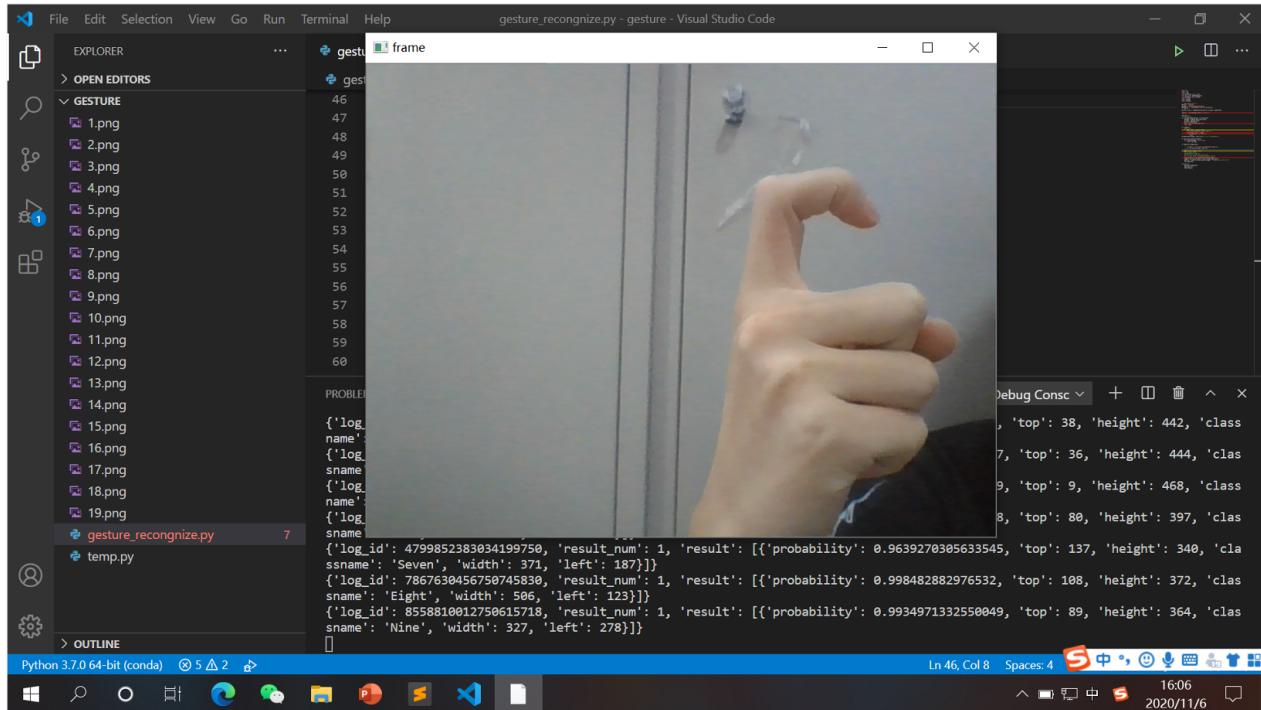


在我们测试的样例中，识别效果还是较为不错的，已经完全可以满足操控无人机的需求。在接下来的模块实现中，我们也要像这样逐步摆脱各项封装API的桎梏，实现完全的自主离线操控。

## 2.2. 手势识别

我们的目标是实现使用语音和手势（以及部分特殊物体）实现自主飞行部分的完全脱机操作。**手势识别**的这一部分，主要目的是让无人机能够分类不同的手势，并作出相应反应。

目前这一部分仍以实现功能的导向为主：在基于在线识别的基础上，检验使用无人机摄像头识别分类的效果。我们利用自己拍摄的照片，检验1-9手势的识别效果，效果图如下：



这一部分的后续离线实现将使用YOLO模型，将会在下一部分详述。

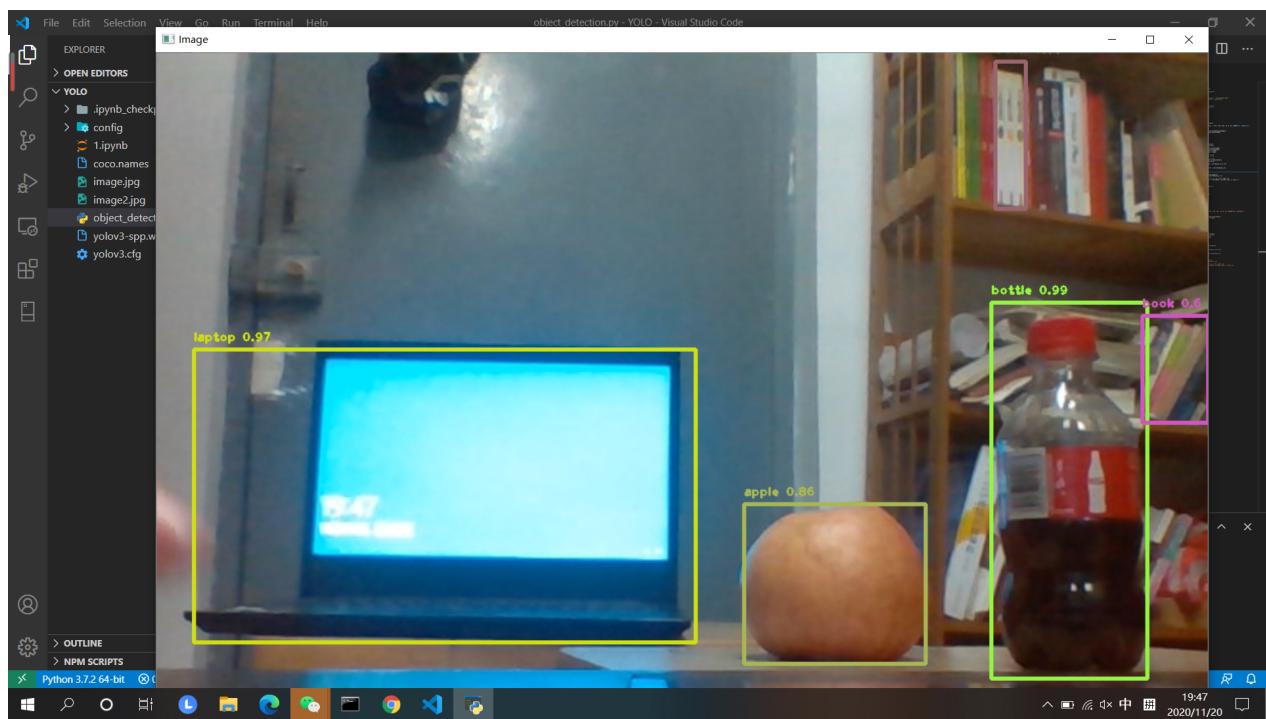
### 2.3. 基于YOLO算法的目标检测

YOLO (You Only Look Once) 是一种基于深度神经网络的对象识别和定位算法，其最大的特点是运行速度很快，可以用于实时系统。现在YOLO已经发展到v4版本(有争议一说是v5版本)。

目前主流的目标检测算法主要是基于深度学习模型，其可以分成两大类：two-stage检测算法；one-stage检测算法。

目标检测模型的主要性能指标是检测准确度和速度，对于准确度，目标检测要考虑物体的定位准确性，而不单单是分类准确度。一般情况下，two-stage算法在准确度上有优势，而one-stage算法在速度上有优势。不过，随着研究的发展，两类算法都在两个方面做改进。而本次我们采用的YOLO算法就是基于one-stage的，更适合用在我们显卡性能并不出众的笔记本上，得到更高的推理速度和较好的检测准确度。

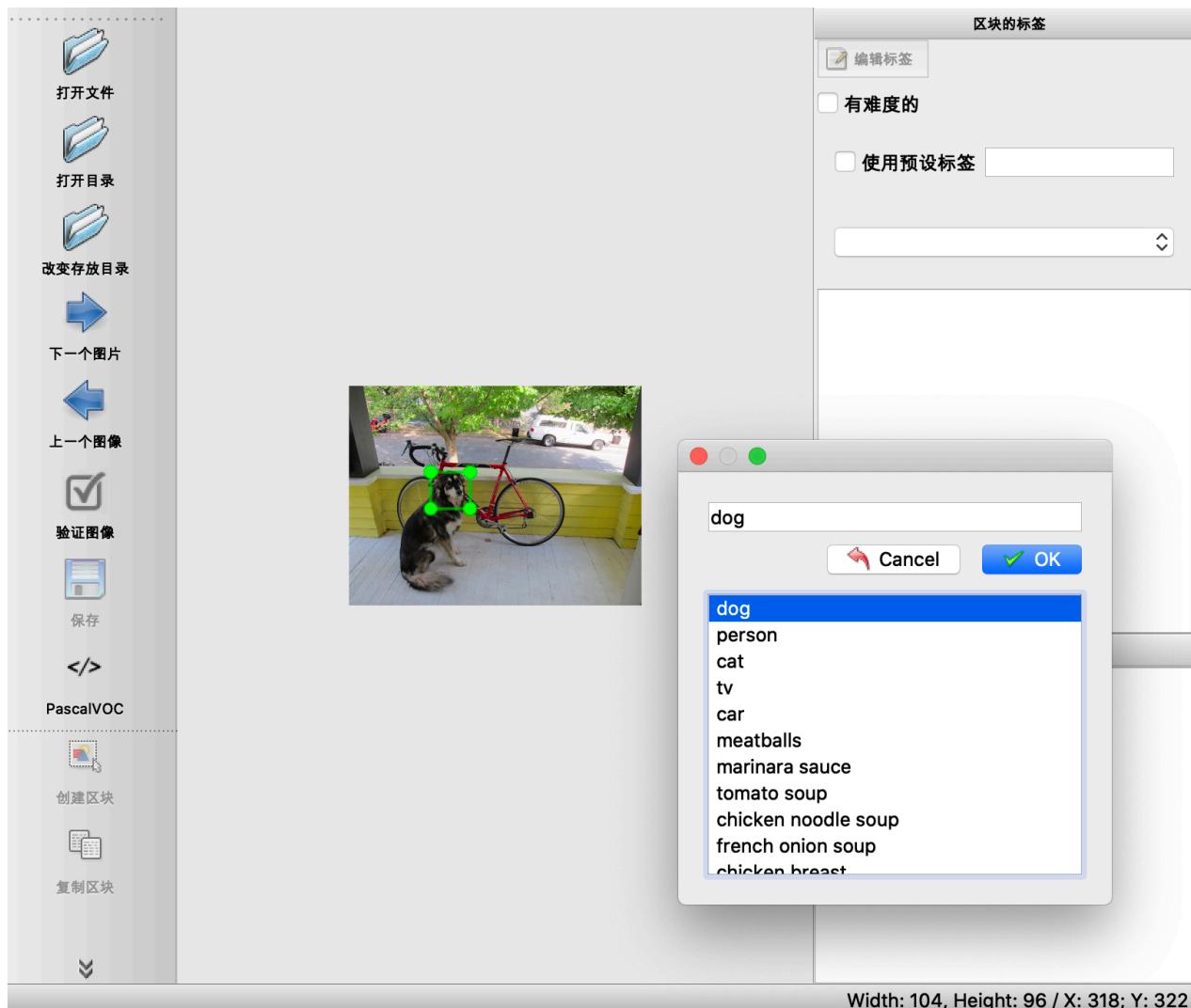
我们下载了相关的YOLOv4网络结构，并使用了基于coco训练集得到的预训练权重进行本次的测试，效果如下：



可以看到，在图像比较模糊的情况下，YOLOv4在预训练权重下仍能将相关类别的物品检测出来，并达到了大致可用的推理速度（后续可能需要使用部分tiny模型进一步加快推理速度以适应笔记本的离线处理速度）

我们期待，未来在YOLO算法的基础上，通过对自行标注数据的训练，进一步实现上述的手势识别功能与自动避障功能。

其中，我们也已经配置好了基于PyQt的标注工具labelimg，可以快速进行相关标注操作，工具效果如图：



## 2.4. 实现目的地飞行的GUI界面

这一部分，我们的目标是通过实现一个GUI界面，让用户可以通过在GUI的地图上点击来向无人机发送飞行的路径，易于操作。

我们从地图软件中截取了一部分地图作为GUI的背景，如下：



然后定义地图上的距离与真实距离的比，来得到应该真实飞行的距离：

```
def get_dist_btwn_pos(pos0, pos1):
    """
    Get distance between 2 mouse position.
    """

    x = abs(pos0[0] - pos1[0])
    y = abs(pos0[1] - pos1[1])
    dist_px = math.hypot(x, y)
    dist_cm = dist_px * MAP_SIZE_COEFF
    return int(dist_cm), int(dist_px)
```

目前已经实现了在python上的GUI实现，接下来会实现目的地点击的自动飞行功能。

### 三、下一阶段计划

对于下一阶段的计划，我们最重要的是基于YOLO算法相关的操作训练出相关模型参数。利用这个实时目标检测的模型，我们就能够进一步实现手势识别与自动避障的功能。

其次是目的地飞行的开发完善，这部分目前还在起步阶段，在第三阶段中将实现较为成熟的功能框架。

为了加快项目的推进速度，我们将集中人力在上述两个功能的开发实现上，尽可能在第三阶段结束前实现一个较为完整的功能框架。

# AutoDriving：无人机项目最终报告

Author: 邓琛龙 2018202077

Group ID: 7

AutoDriving：无人机项目最终报告

- 一、第三阶段成果概览
- 二、实现功能 & 实现过程详解
  - 2.1. 对人脸、双手势、障碍物的实施目标检测
    - 2.1.1. 训练目标 & 训练数据
    - 2.1.2. 训练效果
    - 2.1.3. 无人机人脸追踪算法
  - 2.2. 语音模块的集成方法
  - 2.3. 人脸追踪、手势控制、语音控制的综合程序
- 三、总结与心得

注：第三阶段实践时间：2020.11.20 - 2020.12.17

## 一、第三阶段成果概览

- 1.1. 实现了对人脸、双手势以及障碍物的实时目标检测
- 1.2. 实现了语音模块的集成功能
- 1.3. 实现了人脸追踪、手势控制、语音控制的综合程序

## 二、实现功能 & 实现过程详解

### 2.1. 对人脸、双手势、障碍物的实施目标检测

首先讲下我们遇到的主要硬件困难

#### 1. Tello无人机本身的硬件不足

Tello无人机仅配备一颗500万像素的摄像头，并且只有摄像头这寥寥几种可怜的传感器，难以单凭摄像头实现复杂的行动。

#### 2. 自己电脑图像的实时处理能力不足

拿我自己的设备举例（2019款MacBook Pro 13'），Intel Iris 低压CPU并且无独显，想要实时处理从Tello拿回图像并返回处理结果极为困难，需要追求处理算法的极致性能。

这两大困难很大程度也决定了我们整个项目的技术路线。

## 其次讲下我们的技术选型过程

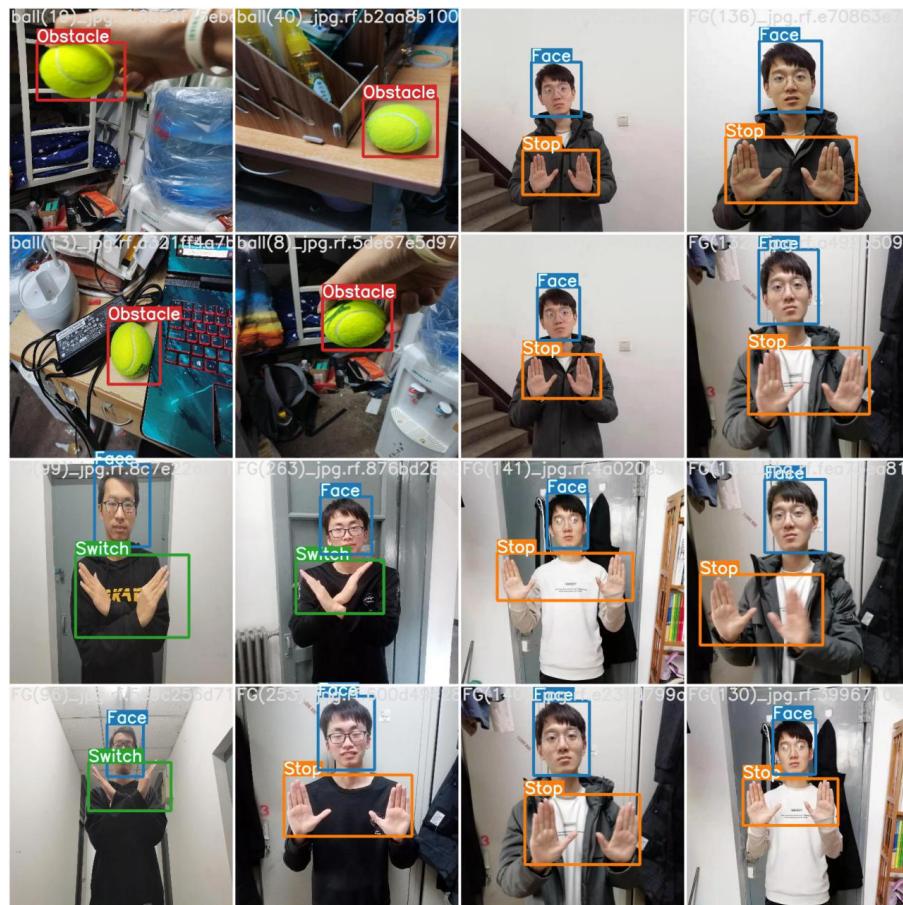
目标检测算法大致可以分为"One-Stage"的算法和"Two-Stage"的算法，一般来说"Two-Stage"的算法准确率更高，但速度相对较慢；而"One-Stage"算法一般来说推理速度较快，但准确率不如"Two-Stage"。鉴于以上提到的两点困难，我们决定采用"One-Stage"的YOLO系列算法来实现目标检测功能。

我们测试了YOLOv4和YOLOv5两个版本的算法，其中YOLOv4在本机的推理速度仅有0.8帧每秒，显然不足以胜任我们的推理速度需求；而采用YOLOv5最小的模型YOLOv5进行测试发现在本机上的推理速度能够达到接近20帧每秒，可以基本胜任我们的需求。因此，我们最终选择YOLOv5s作为我们的目标检测模型。

### 2.1.1. 训练目标 & 训练数据

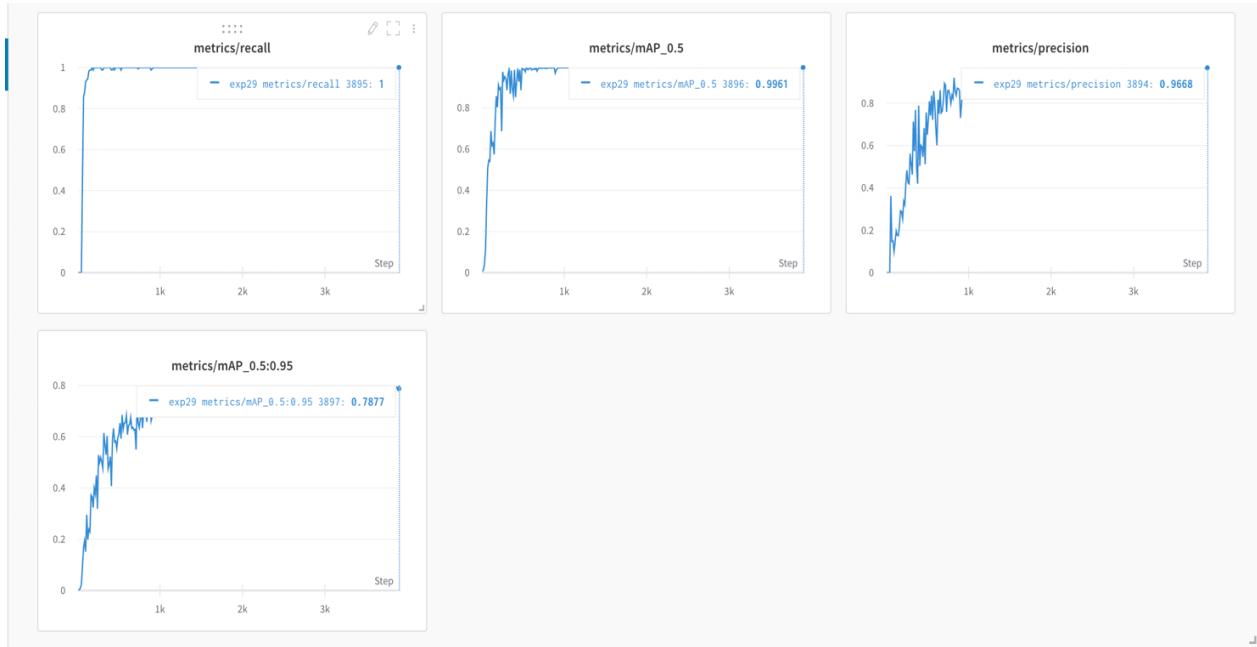
我们的训练目标是识别四种目标，分别是人脸的面部识别、双手在胸前交叉的Switch手势、双手掌在前的Stop手势以及障碍物（一个网球）的识别。利用对这四种目标的检测，我们能够交叉完成很多复杂、有意思的组合命令。

训练数据方面，由于上述的两种手势基本没有可用的现成数据集，故我们选择自己拍摄照片并标注生成相应数据集。最终，经过我们的拍摄以及标注，我们得到了一个共计389张图片的数据集。该数据集包含了其中四种目标的图片，并符合YOLOv5的训练规范，部分训练图片标注如下：



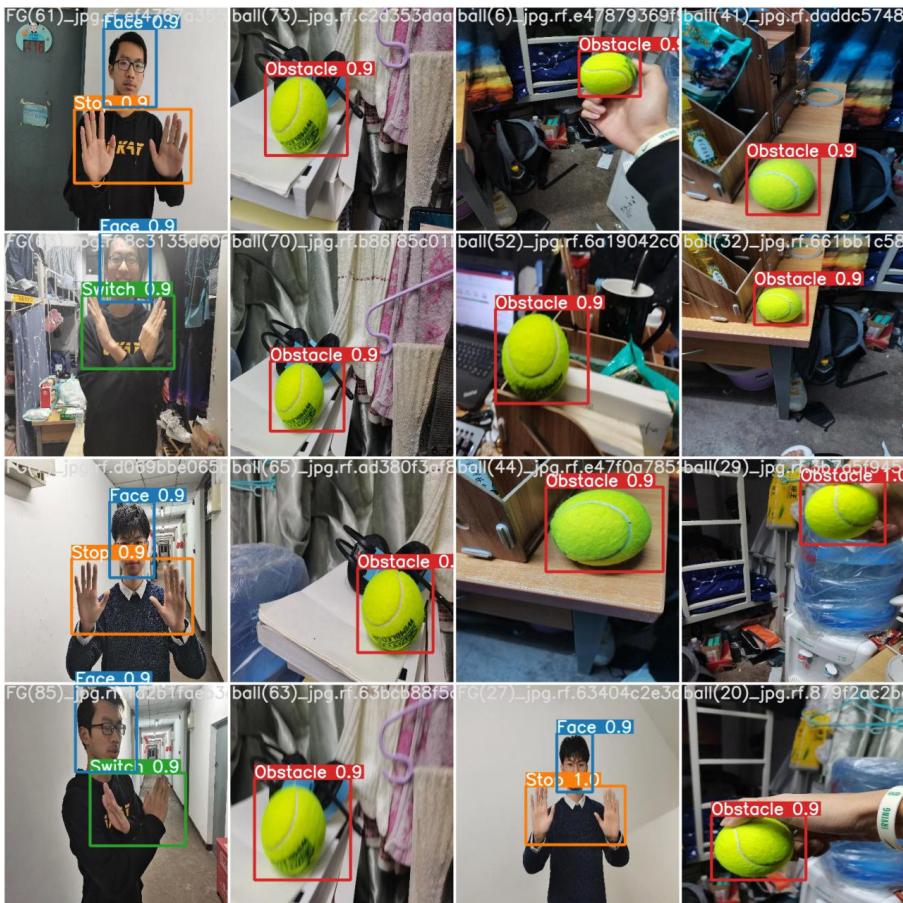
### 2.1.2. 训练效果

我们设定训练的Batch\_size为16，Epochs为300，进行训练后得到模型并进行了性能测试，结果如下：

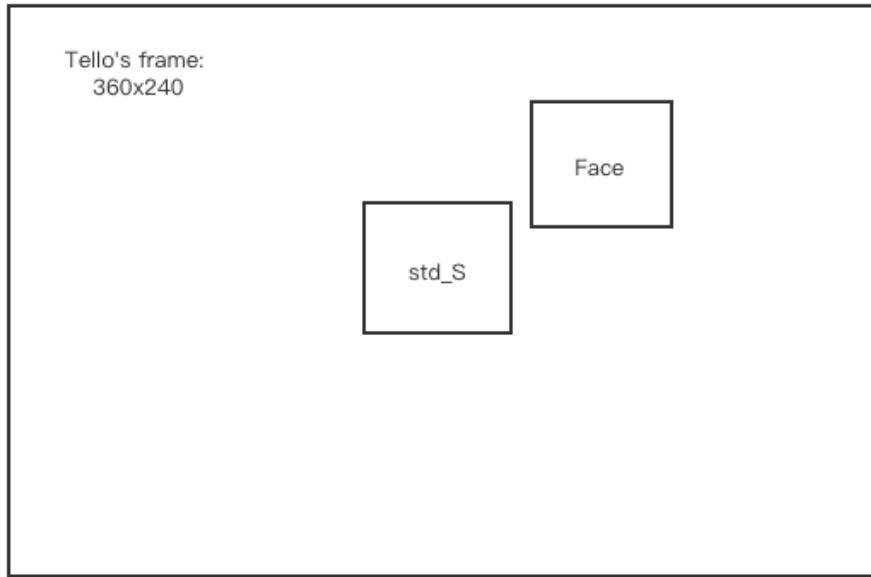


可以看到最后得到的结果中，我们的模型在训练集上的Recall值达到了100%，Precision达到了96.68%，mAP\_0.5达到了99.61%，效果相当不错。

我们也在实际环境中测试了模型的效果，发现在绝大多数环境下该模型都能取得较好的结果，置信度也都较高，部分测试图片如下：



### 2.1.3. 无人机人脸追踪算法



我们用上述图片简单说明无人机人脸追踪算法的大致过程：对于Tello拍摄到的360x240的图像，我们将中心点定位到宽高的中心，并给出一个标准人脸大小的方框std\_S。然后对每一帧的图像进行检测，如图若人脸的中心点在图像中心点的右上方，无人机首先将获得一个向右上方移动的速度；然后对比Face框与std\_S的面积，若Face大于或者小于std\_S超过一定阈值，无人机将会获得一个向前或向后的速度。在三维速度的改变下，无人机能够较好的实现人脸追踪的功能。

部分逻辑代码如下：

```

error_w = target_pos[0] - center_pos[0]
error_h = target_pos[0] - center_pos[0]
error_s = s - std_S

if math.abs(error_w) > w_extrem:
    myDrone.left_right_velocity = -move_speed
elif math.abs(error_h) > h_extrem:
    myDrone.up_down_velocity = -move_speed
elif math.abs(error_s) > s_extrem:
    myDrone.for_back_velocity = -move_speed

```

## 2.2. 语音模块的集成方法

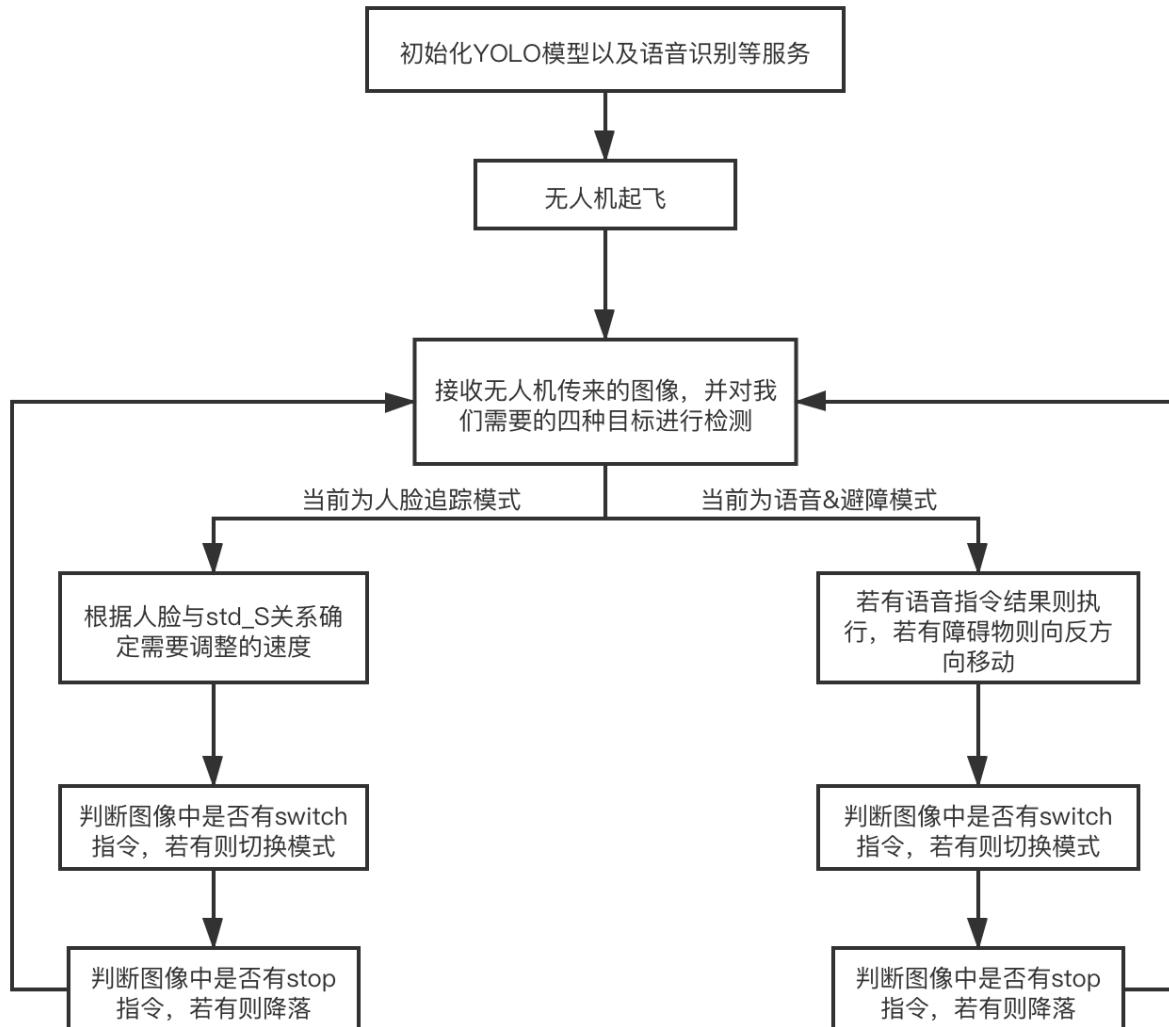
在语音识别的集成过程中，我意识到一个问题：若语音模块直接放在程序中，会使得图像处理过程被阻塞，导致计算机视觉功能无法使用。因此，经过思考后，我将语音识别模块放到了一个额外新开的线程中单独运行，避免了对进程资源的争夺。

但这个做法同时又带来了另外的问题：无人机可能在跟随飞行模式中接收语音，若语音中含有部分关键字，且在识别结果时切换到语音控制&自动避障模式，就会导致出现不可预期的指令执行结果。

因此，我继续改进这种做法。具体解决方案是：记录开始录音时间和结束识别时间的模式状态，只有这两个时间点的状态都是在语音控制的状态才能执行相应的指令。这样就很好的实现了语音识别和指令执行之间的冲突。

### 2.3. 人脸追踪、手势控制、语音控制的综合程序

由于这是一个多功能、多阶段的程序，我们需要一套完整的机制保证无人机的正常飞行。单纯用语言来进行描述过于晦涩，因此我画了一张大致流程图进行说明，图片如下所示：



总体来看，在初始化、加载完必须的几项服务后，我们发出指令让无人机起飞，然后进入一个循环保证无人机的始终飞行和接受命令。在每次接收完无人机传来的图像并进行目标检测后，依照当前模式进入不同决策分支。若当前为人脸追踪模式，则根据上述算法调整飞行速度；若有语音指令则执行，检测到有障碍物则进行避障。然后在结束后都判断一次是否图像中出现了switch和stop手势，若出现switch手势则切换模式，若出现stop则执行降落指令，重复循环。

如此，我们便得到了一个完整的无人机自动飞行程序。

## 三、总结与心得

至此，推进时间为三个月的Tello无人机自动飞行项目就基本完结了。在这个项目中，我们充分调动了Tello能够使用的各项传感器，以求做到更好的体验和更丰富的功能，将计算机视觉、语音处理等功能都在了小小的程序当中，再通过无人机与计算机的交互实现各项“很酷”的功能。

我本人也在项目中学习、实践了大量目标检测算法（如YOLO算法等）和配合硬件的整体工程构建工作，同步提高了对相应领域的认识、理解和自己动手做较大型项目的能力，同时也在团队合作中收获了经验和友情。本次无人机项目的完成需要的能力是多方面的，对我的锻炼也是多方面的，也印证了实践是学习最好的老师这一概念。

最后，感谢老师的辛勤付出和为本项目付出努力的每一个人，推进项目的每一步对我来说都是宝贵的经验，我从中受益良多！