旭日派搭建四旋翼无人机并使用YOLOv5

这篇文章主要是介绍如何快速搭建一个旭日派+PX4无人机平台，并且运行一些检测代码。

为了避免重复造轮子，本文在很多地方附上了链接，根据链接内容，大家就能很好的完成那一部分工作，这篇文章主要是对这些内容进行了一个整合。

1.旭日派装机及远程登陆

使用官网使用用户手册进行安装。

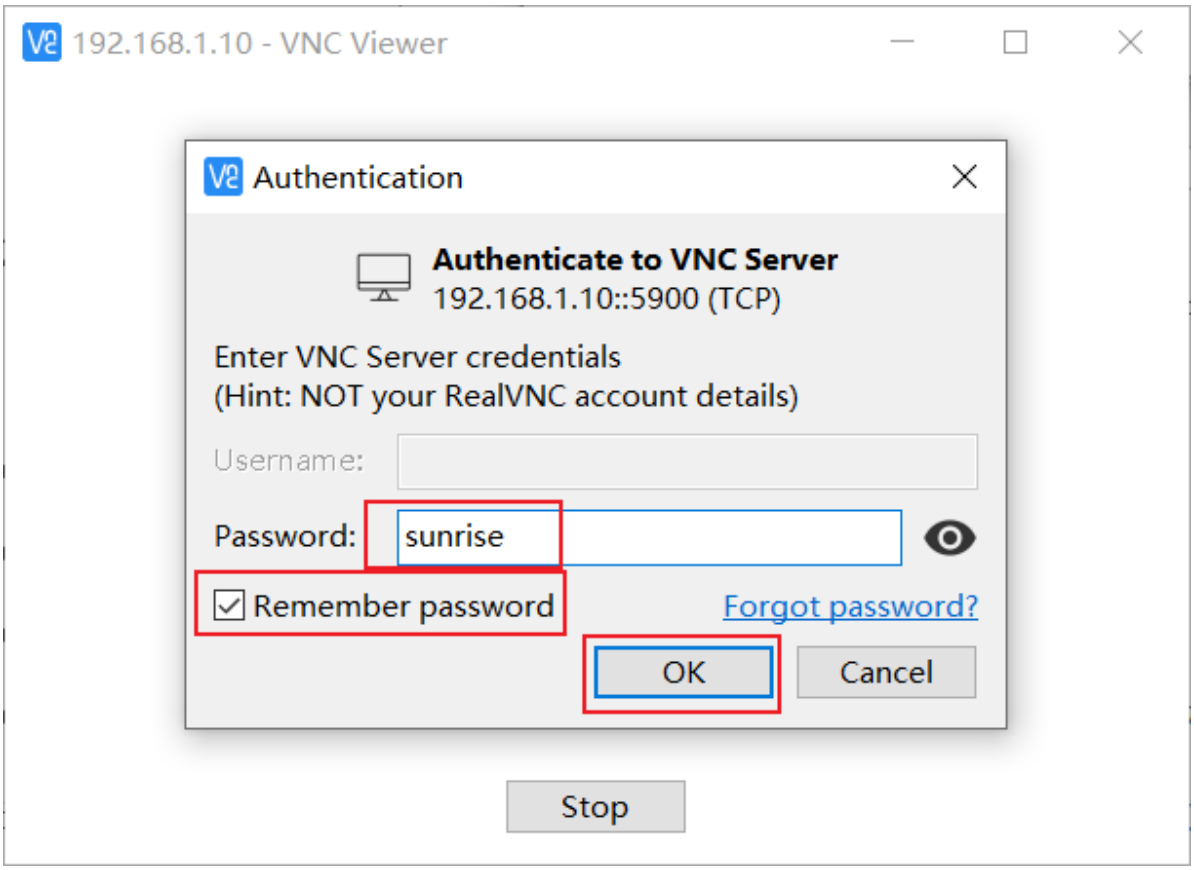
本指南使用是图形界面Ubuntu20.04 Desktop版本。

如果不需要图形化界面并且熟悉linux命令行交互操作模式可以使用Server版本。

https://developer.horizon.ai/api/v1/fileData/documents_pi/Quick_Start/Quick_Start.html#id5#

无人机飞行需要树莓派远程通讯，这里使用VNC登陆的方式，运用局域网使PC机远程连接到旭日派。

VNC登陆在官网的用户手册中也有详细的教程。



2.用pixhawk飞控组装一台S500四轴无人机

想要在无人机上做视觉等模块的二次开发，需要在熟练使用飞控，飞行器能正常操作飞行的基础之上才能进行。

新手无人机爱好者可以在微信公众号苍穹四轴DIY内购买完整的四轴无人机套件和桨叶等配件，并且按照公众号中的教程完成无人机的搭建。

教程：https://mp.weixin.qq.com/mp/appmsgalbum?__biz=MzkyNzI1MDUyNw==&action=getalbum&album_id=1888742813303930881&scene=173&from_msgid=2247484986&from_itemidx=1&count=3&nolastread=1#wechat_redirect

根据教程，需要完成硬件准备、软件准备、硬件组装、飞控调试等步骤，完成这些后就可以使用遥控器操控无人机飞行了。

注意，该公众号软件准备中MissionPlanner地面站的链接失效了，可以在https://firmware.ardupilot.org/Tools/MissionPlanner/archive/中下载MissionPlanner(建议不要下载最新版，不够稳定，可以下载1.3.74版本)

完成无人机搭建、熟悉遥控器飞行后，还不能进行YOLOv5进行高空检测，但我们让飞控与旭日派相连后就可以完成很多任务。

3.旭日派与无人机飞控PX4通讯

准备一根USB2.0的线，将旭日派用USB2.0的线接入飞控，具体方法可以参考视频https://www.youtube.com/watch?v=kB9YyG2V-nA

这个国外的老哥用的树莓派和飞控搭建了一个无人机，我们可以参考来搭建我们的pixhawk旭日派无人机。

4.配置旭日派算法运行环境

安装pytorch

```
pip3 install torch torchvision torchaudio
```

https://pytorch.org/get-started/locally/

安装dronekit

https://blog.csdn.net/weixin_43705900/article/details/120257258

5.YOLOV5检测

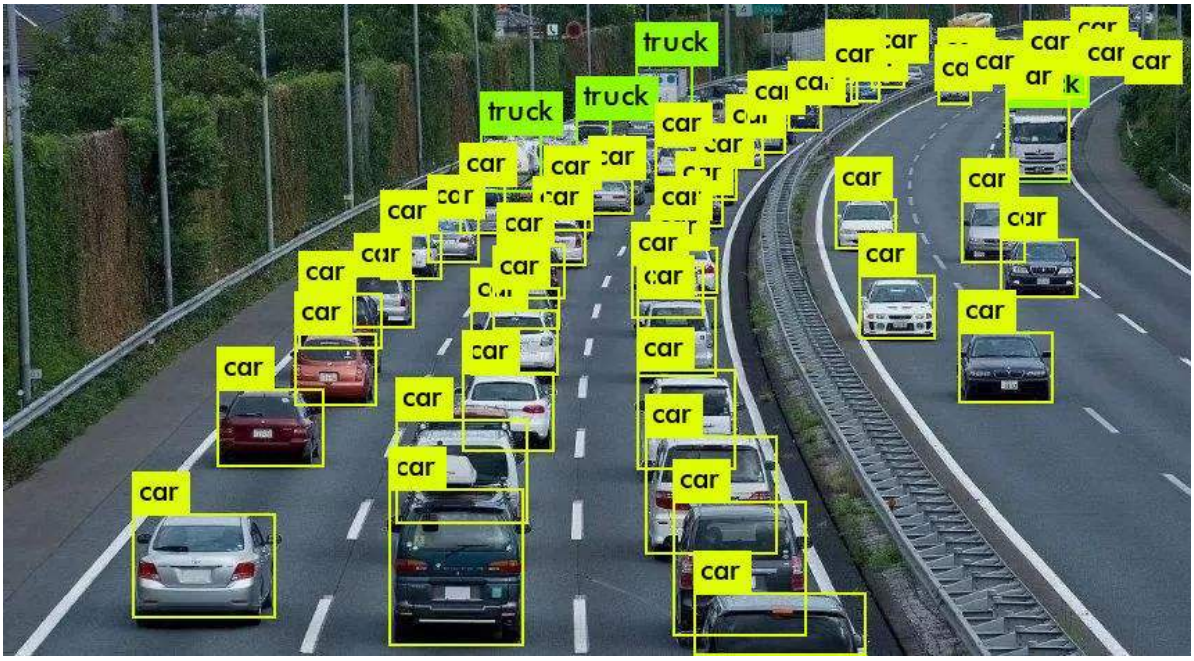
准备一个USB摄像头，插在旭日派上

从这里下好YOLOv5

https://github.com/ultralytics/yolov5

然后在里面把视频源头改成我们自己购买的USB摄像头，把刷新率调到10

用地平线的AI加速可以直升20



6.dronekit控制无人机

dronekit是一个Python中控制无人机的库，在实验中会比mavros更加方便。

这里是一个用键盘控制无人机飞行的程序

```
import time
from dronekit import connect, VehicleMode, LocationGlobalRelative, Command, LocationGlobal
from pymavlink import mavutil

#connection_string = "127.0.0.1:14550"
connection_string = "/dev/ttyACM0"

print('Connecting...')
vehicle = connect(connection_string,wait_ready = True)

gnd_speed =3 # [m/s]
def arm_and_takeoff(altitude):


  while not vehicle.is_armable:
    print("waiting to be armable")
    time.sleep(1)


  print("Arming motors")
  vehicle.mode = VehicleMode("GUIDED")
  vehicle.armed = True

  while not vehicle.armed: time.sleep(1)
```

```
  print("Taking Off")
  vehicle.simple_takeoff(altitude)


  while True:
    v_alt = vehicle.location.global_relative_frame.alt
    print(">> Altitude = %.1f m"%v_alt)
    if v_alt >= altitude * 0.95:
      print("Target altitude reached")
      break
    time.sleep(1)

arm_and_takeoff(10)


def set_velocity_body(vehicle, vx, vy, vz):
  """ Remember: vz is positive downward!!!

  Bitmask to indicate which dimensions should be ignored by the vehicle
  (a value of 0b0000000000000000 or 0b0000001000000000 indicates that
  none of the setpoint dimensions should be ignored). Mapping:
  bit 1: x,  bit 2: y,  bit 3: z,
  bit 4: vx, bit 5: vy, bit 6: vz,
  bit 7: ax, bit 8: ay, bit 9:

  """
  msg = vehicle.message_factory.set_position_target_local_ned_encode(
      0,
      0, 0,
      mavutil.mavlink.MAV_FRAME_BODY_NED,
      0b0000111111000111, #-- BITMASK -> Consider only the velocities
      0, 0, 0,      #-- POSITION
      vx, vy, vz,    #-- VELOCITY
      0, 0, 0,      #-- ACCELERATIONS
      0, 0)
  vehicle.send_mavlink(msg)
  #vehicle.flush()
  vehicle.commands.upload()

while True:

  a = input('enter an sign')

  if a == 'l':
          print("l pressed >> Set the vehicle to LAND")
          vehicle.mode = VehicleMode("LAND")
  elif a == 8 :
          set_velocity_body(vehicle, gnd_speed, 0, 0)
  elif a == 5:
          set_velocity_body(vehicle,-gnd_spepythoned, 0, 0)
  elif a == 4 :
          set_velocity_body(vehicle, 0, -gnd_speed, 0)
  elif a == 6 :
          set_velocity_body(vehicle, 0, gnd_speed, 0)
  elif a == 1 :
          set_velocity_body(vehicle, 0, 0, -gnd_speed)
```

```
    elif a == 3 :
            set_velocity_body(vehicle, 0, 0, gnd_speed)
    elif a == 84 or a == 48 :
            set_velocity_body(vehicle, gnd_speed*2*0.5, -gnd_speed*2*0.5, 0)
    elif a == 86 or a == 68 :
            set_velocity_body(vehicle, gnd_speed*2*0.5, gnd_speed*2*0.5, 0)
    elif a == 45 or a == 54 :
            set_velocity_body(vehicle, -gnd_speed*0.5, -gnd_speed*0.5, 0)
    elif a == 65 or a == 56 :
            set_velocity_body(vehicle, -gnd_speed*2*0.5, gnd_speed*2*0.5, 0)
    elif a == 2 :
            set_velocity_body(vehicle, 0.0.0)
```

你可以参考dronekit官网http://dronekit.io/编写你自己的程序。

7.在ubuntu18.04中搭建仿真平台测试

直接运行dronekit脚本控制无人机，风险很大，容易造成无人机损坏，可以先在仿真环境中测试你的代码。接下来介绍四旋翼无人机 dronekit-STIL +MAVproxy+MIssionPlanner模拟飞行。

1.安装库和MIssionPlanner

```
pip install dronekit
pip install dronekit-sitl
pip install mavproxy
# 有时候要加sudo
#dronekit: 一种用于无人机控制的python库。
#dronekit-sitl：一种本地仿真工具，相当于本地虚拟化了一个无人机。
#mavproxy：数据转发软件
```

安装和启动MIssionPlanner地面站可以参考博客https://blog.csdn.net/TLKids/article/details/122717540

2.启动SITL(Software in the Loop)，并配置home点、机型

打开一个终端，运行如下命令

```
dronekit-sitl  copter --home=31.9386595,118.7898506,3,30 --model=quad
#home为无人机初始化地址，这里我们选了一片操场，可以省略这个参数
#model为模型，这里我们选择为四旋翼（quad）
```

3.启动mavproxy

地面站MissionPlanner中要获取消息，可以通过以下命令连接需要用到的端口。

注意，tcp地址前要加`tcp:`，udp则不用

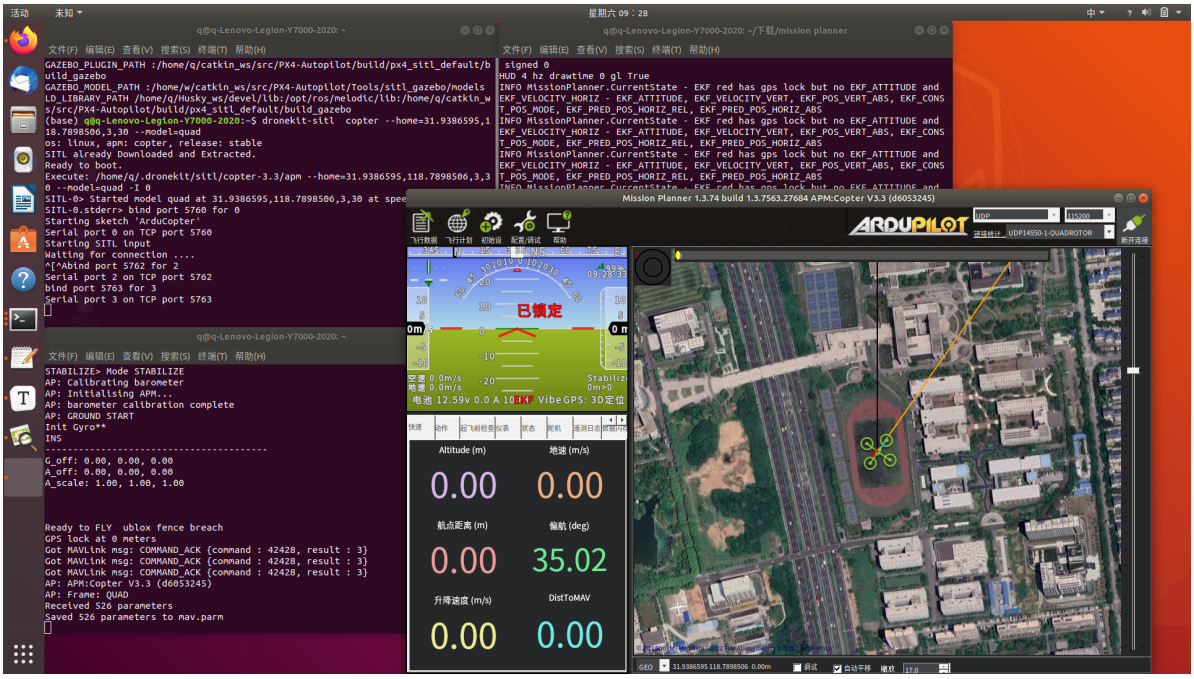
新开一个终端运行如下命令

```
mavproxy.py --master=tcp:127.0.0.1:5760 --out=192.168.0.102:14550 --out=127.0.0.1:14550
```

tcp:127.0.0.1:5760： SITL默认端口，作为mavproxy的输入，把输入数据转发到如下两个端口：
127.0.0.1:14550 ： 本机地址14550端口，供本地python程序连接。
192.168.0.102:14550：供MissionPlanner连接，可用ifconfig命令查看ip地址进行修改!

4.打开MissionPlanner地面站，连接到仿真器

打开地面站，在右上角connect中选择UDP连接，端口号选择14550，仿真器和地面站就会自动连接。



5.运行dronekit脚本

编写dronekit程序，运行。

注意，在仿真时dronekit程序中connect的端口需要改为127.0.0.1:14550

6.仿真程序不能起飞

如果按照上面的步骤仿真，会发现dronekit程序在运行但是无人机高度一直为0m。

这里应该是pymavlink版本的bug，我在pip install dronekit时自动安装了pymavlink-2.4.35，把版本改为pymavlink==2.4.8，有时也会报错，但不影响正常起飞。

```
pip uninstall pymavlink
pip install pymavlink==2.4.8
```