amov_wiki Documentation

发布 *0.0.1*

eason

2020 年 10 月 19 日

Contents

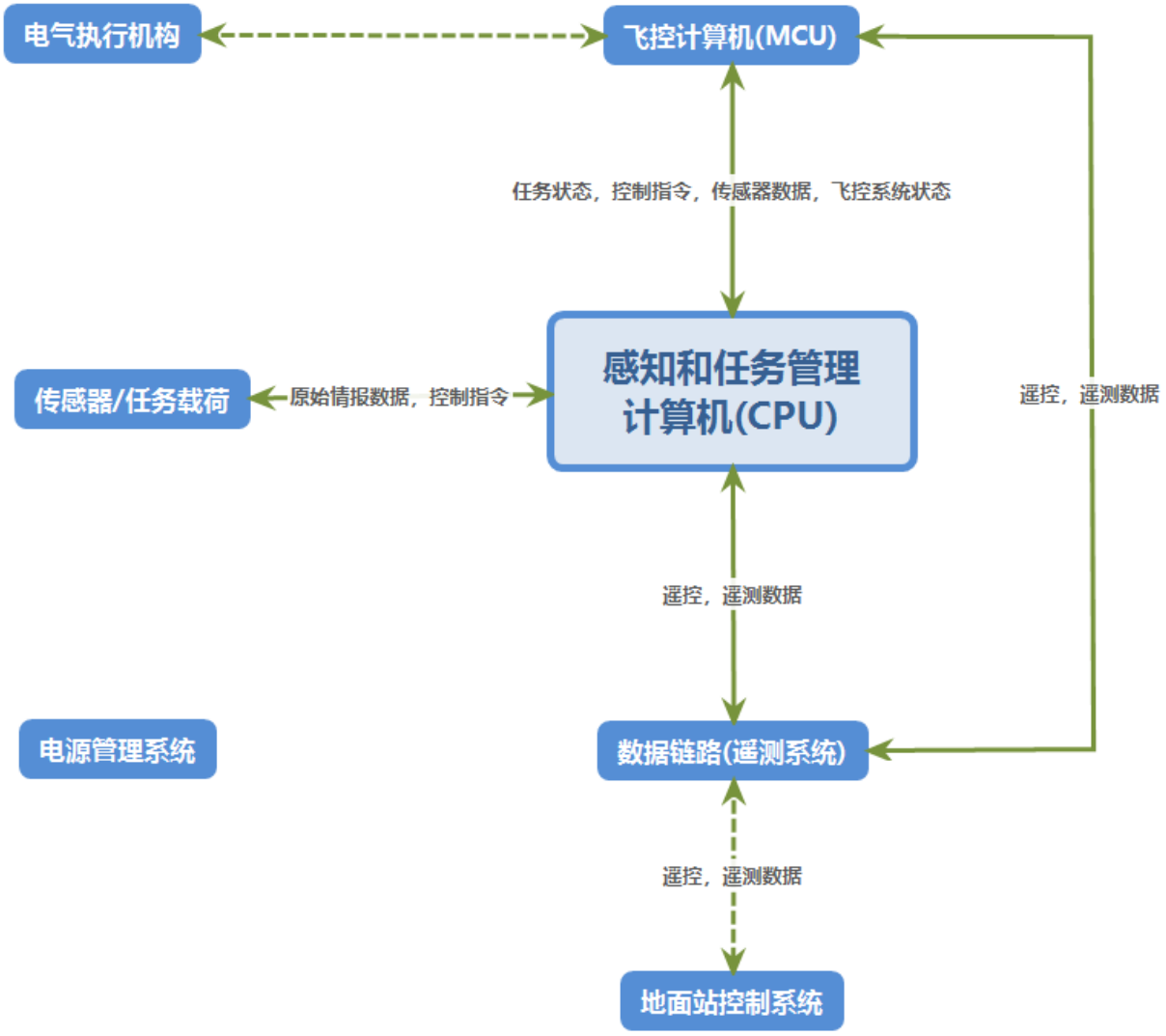
CHAPTER 1

小型无人机系统的计算机系统

相对比较完善的小型无人机计算机系统，包含飞控计算机 (相当于人的脑干，负责完成运动控制系统的平衡性)，感知和任务管理计算机 (相当于人体的大脑，负责处理来自任务载荷的数据，并且做出决策，发送控制指令给飞控计算机)，任务载荷模块 (相当于人体的眼睛等传感器，感知周围环境数据然后给任务管理计算机处理)。电气系统是最底层的执行机构。地面站是人机交互系统。数据链接由无线数据传输系统来完成。各个模块都有对应的软件和硬件设备，相对独立，互相协作，构成一个完整的无人机计算机系统。

飞控计算机由单片机 (MCU) 和对应的实时操作系统担任，要求就是实时性强，任务执行频率上百赫兹，可以快速的控制电器系统，强调响应的实时性和相对简单可靠，确保被控对象的平衡稳定。

感知和任务管理计算机由 PowerPC，DSP，Arm，X86 构架的 CPU 来担任，通常都是计算性能强大，并且运行之上的操作系统可靠如 (VxWorks) 等。负责运算和决策，通常这种任务计算复杂，某些情况下也要求较高的任务执行频率。


电气执行机构
飞控计算机(MCU)
任务状态，控制指令，传感器数据，飞控系统状态
传感器/任务载荷
原始情报数据，控制指令
感知和任务管理
计算机(CPU)
遥控，遥测数据
遥控，遥测数据
电源管理系统
数据链路(遥测系统)
遥控，遥测数据
地面站控制系统

CHAPTER 2

PX4/Ardupilot 的计算机系统

除了 Ardupilot 和 PX4 的飞控计算机之外，他们也有任务计算机系统，但是相对还不是很完善，比如目前的飞控计算机中还做一部分的任务管理计算机的工作，比如航线规划，避障，环境感知等，没有做到模块分离和独立。

Ardupilot 支持的 Companion Computers(任务计算机)，大多以 Linux/ROS/Mavros 为软件构架，硬件上以 Arm 内核,X86 内核为主。下面是 Ardupilot 官方 给出的列表：

Related topics on this wiki include:

- APSync
- DroneKit Tutorial
- FlytOS
- Maverick
- ROS
- Raspberry Pi
- ODroid
- Intel Edison
- NVidia TX2
- NVidia TX1
- BeaglePilot Project
- Turnkey Companion Computer Solutions
- LYCHEE - Carrier board for Cube flight controller with built in Raspberry Pi Compute Module 3+

从功能上，有的是提供一个软件的框架，可以适配很多的不同硬件设备，有的提供一个完整的软件硬件解决

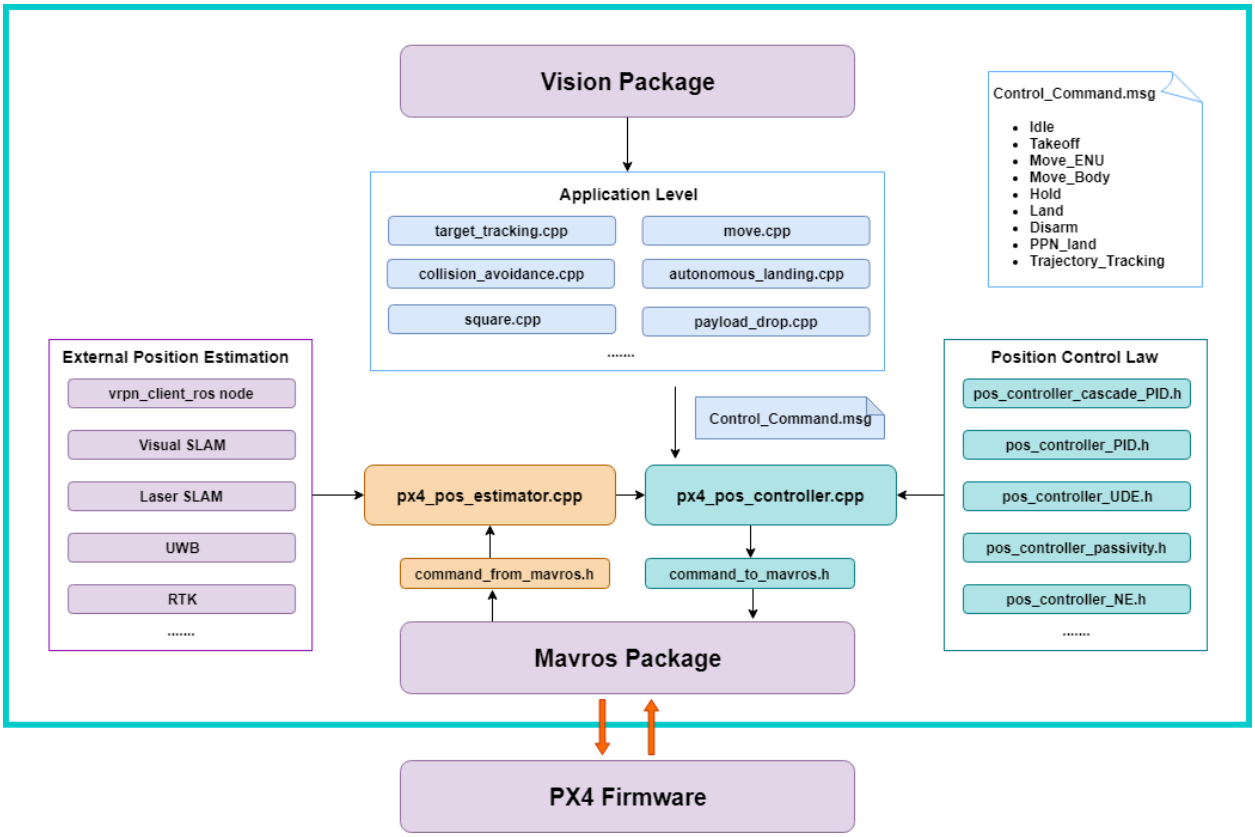
方案提供 4G 通信，图像处理等高级功能。也在向模块独立的方向上发展。相信随着技术的发展，这些板载计算机 (任务计算机) 将会越来越完善，越来越像成熟的软件硬件方向发展 (可以参考大型载人航空器的设计)。随着应用广泛和技术也会出现诸如 PowerPC 和 VxWorks 为代表的任务管理计算机系统，和飞控计算机系统彻底软件硬件分离。

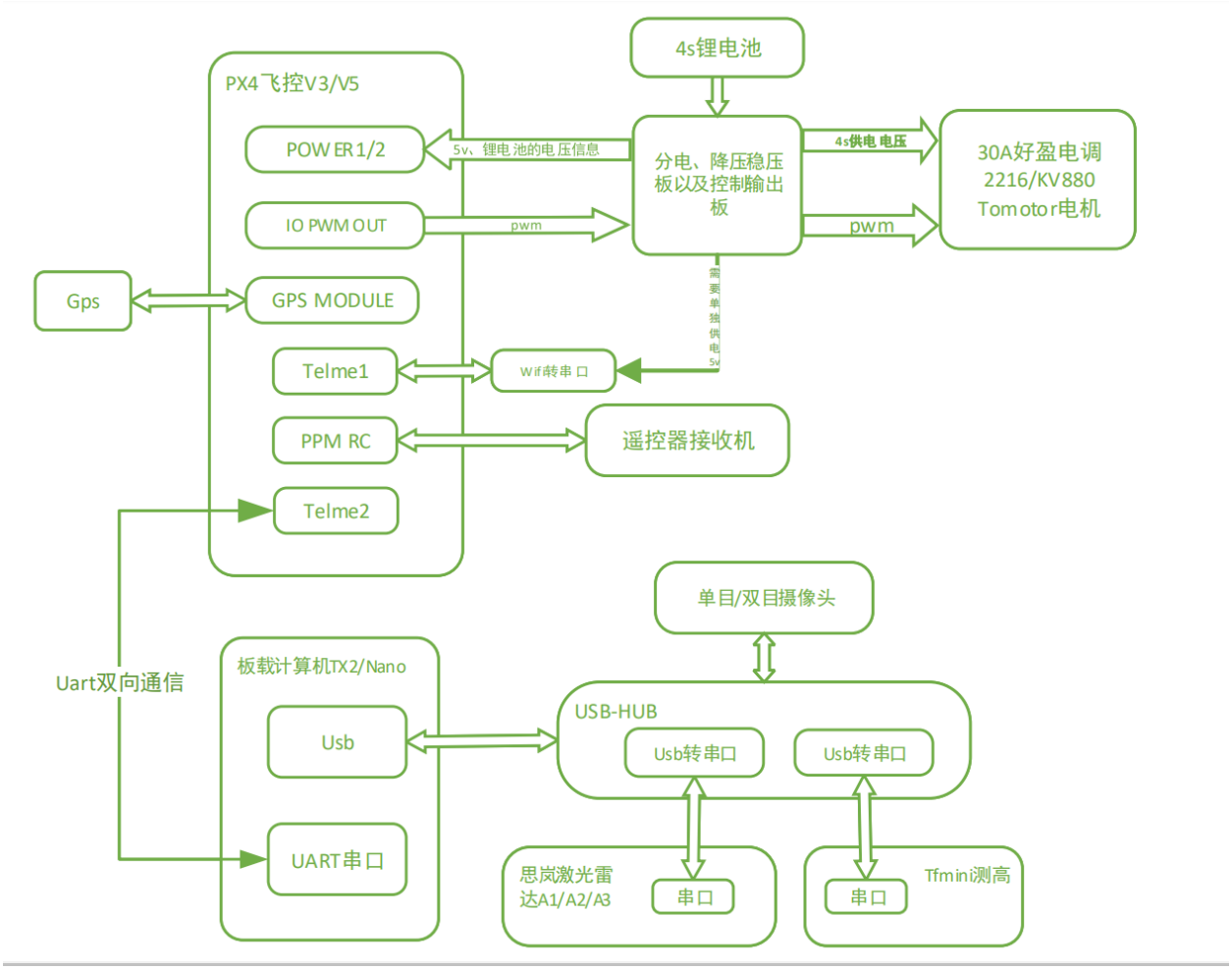
CHAPTER 3

阿木实验室的感知任务管理系统

我们也计划在任务管理系统上做一些工作，我们开源了我们的板载计算机软件框架 (Px4Commander 暂时命名)

px4_command 功能包是一个基于 Ardupilot/PX4 开源固件及 Mavros 功能包的开源项目, 提供一个感知和任务计算机模块。目前已集成无人机外环控制器修改、目标追踪、激光 SLAM 定位、双目 V-SLAM 定位、激光避障等上层开发代码、后续将陆续推出涵盖任务决策、路径规划、滤波导航、单/多机控制等无人机/无人车/无人船科研及开发领域的功能。配合板载计算机 (树莓派、TX2、Nano) 等运算能力比较强的处理器，来实现复杂算法的运行，运行得到的控制指令通过串口或者网口通信发送给底层控制板。


Vision Package
Control_Command.msg
Idle
Takeoff
Move_ENU
Move_Body
Hold
Land
Disarm
PPN_land
Trajectory_Tracking
Application Level
target_tracking.cpp
move.cpp
collision_avoidance.cpp
autonomous_landing.cpp
square.cpp
payload_drop.cpp
.....
External Position Estimation
vrpn_client_ros node
Visual SLAM
Laser SLAM
UWB
RTK
.....
Control_Command.msg
px4_pos_estimator.cpp
px4_pos_controller.cpp
Position Control Law
pos_controller_cascade_PID.h
pos_controller_PID.h
pos_controller_UDE.h
pos_controller_passivity.h
pos_controller_NE.h
.....
command_from_mavros.h
command_to_mavros.h
Mavros Package
PX4 Firmware

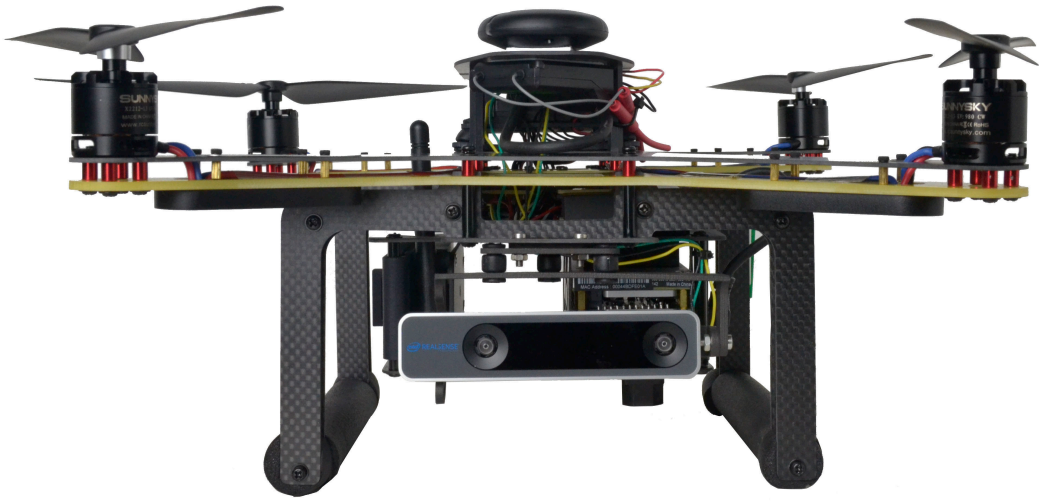

4s锂电池
PX4飞控V3/V5
POWER1/2
5v、锂电池的电压信息
分电、降压稳压板以及控制输出板
4s供电 电压
30A好盈电调
2216/KV880
Tomotor电机
IO PWM OUT
pwm
pwm
需要单独供电5v
Gps
GPS MODULE
Telme1
Wif转串口
PPM RC
遥控器接收机
Telme2
单目/双目摄像头
板载计算机TX2/Nano
Uart双向通信
USB-HUB
Usb
Usb转串口
Usb转串口
UART串口
思岚激光雷达A1/A2/A3
串口
串口
Tfmini测高

AMOVLAB 阿木实验室

P200系列

室内/外

科研无人机开发平台



3.1 感知和任务管理计算机的开发

在 Ardpuliot/PX4 的感知和任务管理计算机开发中，和大型无人机的任务管理计算机开发不同，小型无人机的任务管理计算机集中在环境感知，自动避障路径规划，目标物体识别追踪，自动搜寻，集群控制策略等领域。符合小型化，低空域，在有限空间飞行的特点。

3.1.1 开发重点

环境感知: 以激光雷达，视觉传感器等硬件为主的激光 SLAM/视觉 SLAM 开发方向，获得设备自身的方位，速度和姿态。

路径规划: 以自动避障的路径规划，以集群控制为主的路径规划，以自动搜寻的路径规划，以无人机自动调度系统的路径规划。获得当前空间和时间下的设备自身的最优运动路径。

视觉识别: 以目标物体的识别，感知，追踪，引导。获得目标物体的相对位置。

3.1.2 开发手段

按照完整的开发流程:

- 1 建立数学模型

- 2 算法开发
- 3 仿真
- 4 实机测试

以完成一个 **激光雷达避障**为例：

1 数学建模

业界关于避障的算法，有很多论文，开源的代码。比如可以采用简单高效的 VFH 算法，数学模型相对比较简单，方便构建。

2 算法开发

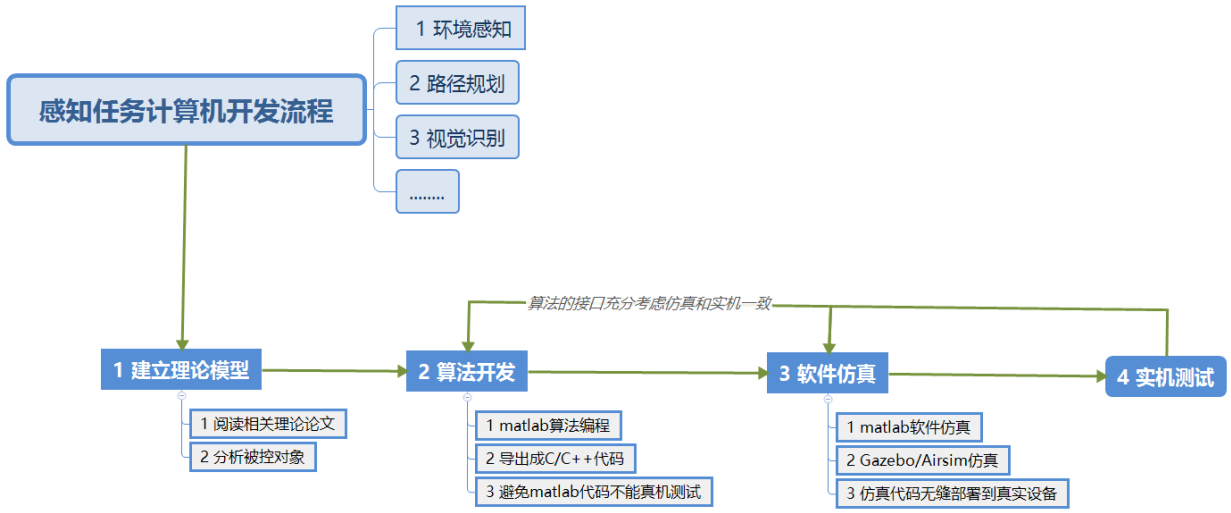
根据数学模型编写代码，涉及复杂的数学公式可以采用 matlab 来开发，VFH 比较简单，开源的代码做一下优化就可以完成。

3 仿真

避障属于场景和逻辑，最好用三维可视环境搭建出虚拟环境，在虚拟环境中仿真，常用的仿真工具有 FlightGear，Gazebo，AirSim 等等，写好的算法在虚拟仿真环境中运行，看一看避障的效果如何。总结就是仿真有问题，实际测试一定有问题。

4 实机测试

仿真测试通过以后，然后实机测试具体功能，在仿真中传感器数据大多为理想值，仿真的代码有时候不能完全适用于真实环境。当然可以在仿真系统中加入噪声数据，和建立仿真模型的时候高度还原真实物理场景，这样也可以提高仿真代码的适应性。



我们在感知任务管理计算机设计的时候，就充分考虑到一个完整的开发流程，来提高开发效率。

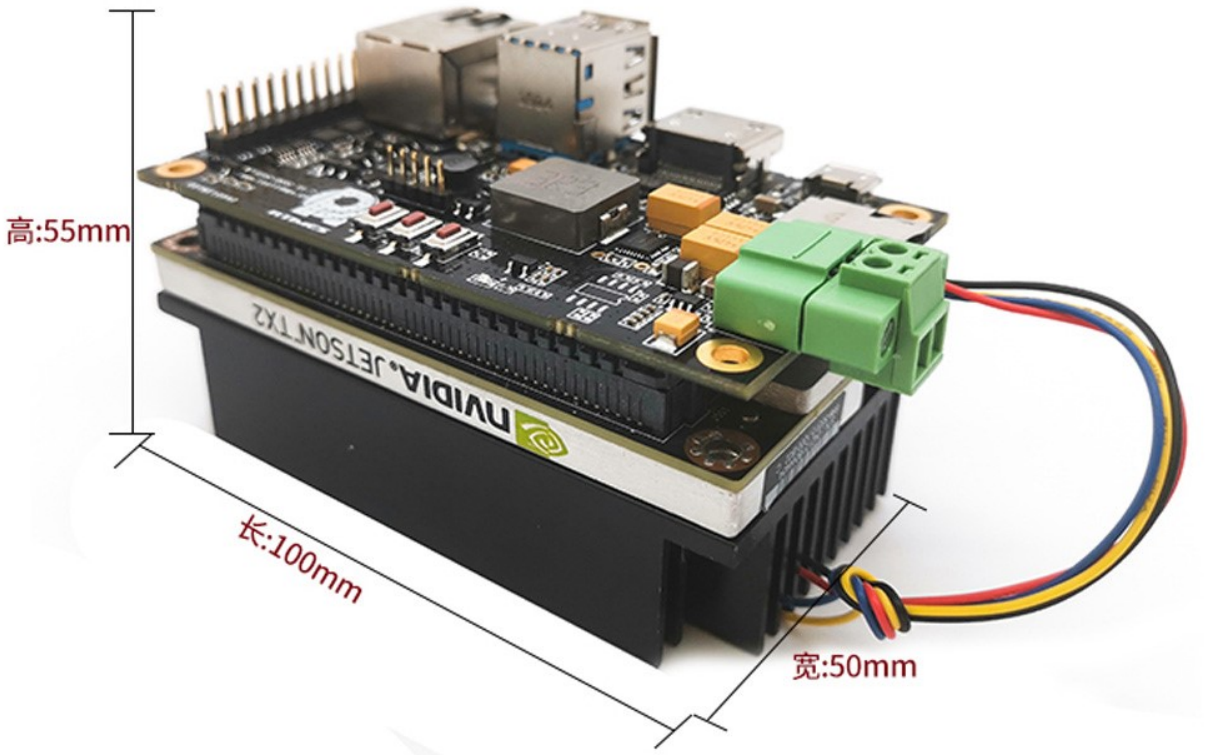
尤其是一些复杂的算法开发，Matlab 接口，三维任务仿真接口必不可少，可以极大的提升开发效率。我们提供的这个任务计算机功能包是基于 ROS/Mavros 开发，三维仿真接口比较完善比如用 Gazebo/Airsim。Matlab 的接口在基于集群的路径规划非常有用。

项目介绍

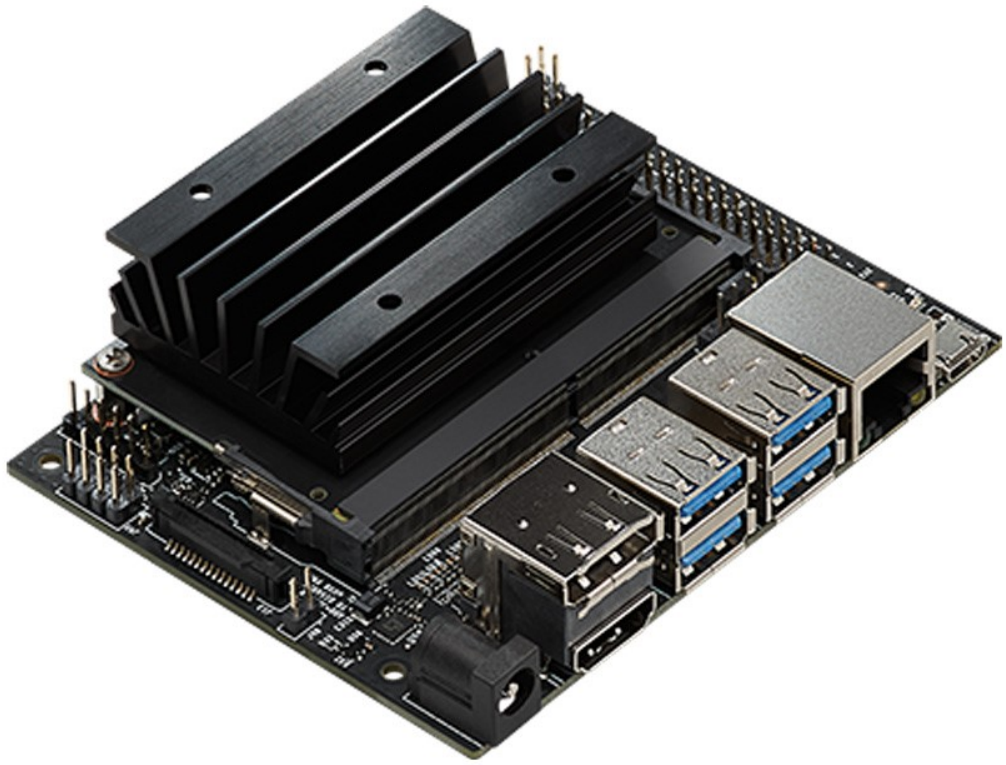
Prometheus200(简称 P200) 是一款专为科研工作者及无人机开发者设计的无人机开发实验平台, 适用于无人机专业应用研究和开发. 本开发平台提供丰富的 demo 例程, 涵盖路径规划, 导航滤波, 建模控制, 目标识别, 深度学习, 视觉/激光 SLAM, 编队控制等多个无人机及机器视觉相关研究方向, 为无人机科研开发助力.

P200 是科研无人机开发平台, 它由流行的开源组件以及与之配合使用的必要文档和库组成.

该套件包括有 PX4 组件包的飞控 pixhawk 硬件平台, 作为板载计算机的 NVIDIA TX2/Nano, 用于视觉导航的相机, 以及一系列其他的外部传感器.



更多 tx2 的性能信息请移步 淘宝 tx2



更多 Nano 的性能信息请移步 淘宝 Nano

P200 科研无人机平台包含了基于 NVIDIA TX2/Nano 板载计算机对飞控的 mavros 逻辑控制程序, 改程序的源代码是开放的, 可以在 Github 上获得, 如果有疑惑, 有问题的欢迎提 **issue** .

如果在文档教程中没有解决你想要的答案, 加入我们研发者维护的平台, 我们开发者很乐意为您解答: amov 社区论坛!

新手入门

专业名词解释

aircraft: 任何可以飞或者可以携带物品还是搭载旅客的飞行器统称为飞机 (航空器).

uav: 无人驾驶飞机

vehicle: 飞行器

airplane/plane/aero-plane : 有机翼和一个或多个引擎的飞行器统称为飞机

Drone: 无人航空器, 典型的有四旋翼, 六旋翼, 飞机模型, 固定翼, 垂起, 直升机模型

aerial: 空中的, 从飞机上的.

四旋翼 (Quadcopter) : 无人驾驶飞行器, 一般带有四个螺旋桨和一个电子稳定系统

多旋翼: 无人驾驶飞行器, 带有一个电子稳定系统和螺旋桨的数量取决于三旋翼, 四旋翼, 六旋翼, 八旋翼或者更多.

飞行控制器/自动驾驶:

- 1. 设计了用于控制多旋翼, 飞机或者是其他飞行器的一种专用电路板, 例如:pixhawk,Ardupilot,Naze32,CC3D
- 2. 多旋翼控制的软件, 例如:PX4,APM,CleanFlight,BetaFlight.

固件: 主要用于嵌入式系统的软件, 例如: 飞控系统, 电调系统 (ESC)

电机: 旋转多轴螺旋桨的电动机, 通常使用的是无刷电动机, 这些电机需要电调.

电调/电机控制器:ESC, 电子调速控制器, 用于控制无刷电机速度的专用电路板. 它使用 PWM 飞行控制器控制.

电池: 无人机航空器的可充电电池. 四旋翼常用的是 LiPo 锂电池

电池芯数: 无人机航空器常用的电芯数 (2s~6s), 每节锂电池的最大电压 4.2V, 正常充电电压 3.7V, 电池的电压是每节锂电池电压串联的总和,P200 无人机使用的是 4s 锂电池

遥控器/无线电控制设备: 远程遥控操控四旋翼的设备, 我们称为遥控器. 远程遥控器控制的的前提是接收机需要连接到飞控中.P200 无人机使用的是富斯 i6s.

遥测设备:

解锁:Arming, 飞机准备飞行的状态, 当遥控器慢慢推动油门摇杆时或者发送带有目标点的外部命令时, 飞机将开始飞行. 通常解锁的操作是, 油门最低 + 偏航角最大, 一般情况下, 解锁之后螺旋桨会以怠速转动, 与之相反的状态是上锁,Disarmed

PX4: 非常流行的开源飞控代码, 它可以在 pixhawk 系列嵌入式主板中运行.P200 所使用的就是 pixhawk4 嵌入式主板.

NVIDIA TX2/NANO: 性能较强大的板载计算机, 也是 P200 中所使用的板载计算机.

APM/Ardupilot: 最开始的时候是为 Arduino 开发板创建的开源飞控. 现在也可以兼容 pixhawk 系列嵌入式主板.

Mavlink: 无人航空器, 地面站以及通过无线信道的其他设备相互间通信协议,Mavlink 是一个用于无人机的通讯协议，在这个通讯协议下有很多很多消息类型。外部与飞控建立连接，绝大部分是利用 Mavlink 协议，传递的是 Mavlink 消息。比如：地面站、mavros

ROS: 编写复杂机器人应用程序的流行框架.Robot Operating System (ROS) 是一种得到广泛使用的机器人操作与控制系统软件框架。其提供了一个标准的操作系统环境，包括硬件抽象、底层设备控制、通用功能的实现、进程间消息转发和功能包管理等.

mavros: 连接 ROS 和 mavlink 协议之间的库.

uORB:uORB 是一种类似 ROS 主题的发布, 订阅机制, 但大大简化了其复杂性并将其应用到嵌入式环境之中.

硬件介绍

主要向介绍三款科研无人机开发平台, 第一款是型号为 P200-A2-TX2 的 P200 系列无人机

该无人机包含有 pixhawk4 基础套件, 完成 px4 最基本的飞行功能; 其次有激光雷达 rplidar A2, 在室内完成激光雷达 slam 定位; 单目视觉相机基于 openCV 的图像识别, 视觉引导降落等; 板载计算机 NVIDIA TX2 作为核心来处理这些外部设备获取到的原始数据, 最后通过 mavros 发送至飞控之中. 在板载计算机中包含有视觉处理 ROS 包,rplidarROS 包,cartag 算法 ROS 包, 以及核心控制功能包 px4_command.

更多详细参数, 功能可以查看 淘宝 产品详情.

第二款型号为 JCV4-410 无人机

机体	P200 含电机电调	台	1
主控	pixhawk4	个	1
GPS	M8N	个	1
螺旋桨	9 寸	支	8
遥控器 (含接收机)	富斯 I6S	台	1
遥控器电池	镍氢	个	1
镍氢电池充电器	镍氢充电器	个	1
Intel 双目相机	T265	个	1
动力电池	4200mAh/4S	个	1
数传	WIFI 数传 (AP 模式)	个	1
平衡充电器	A400	台	1

选配:

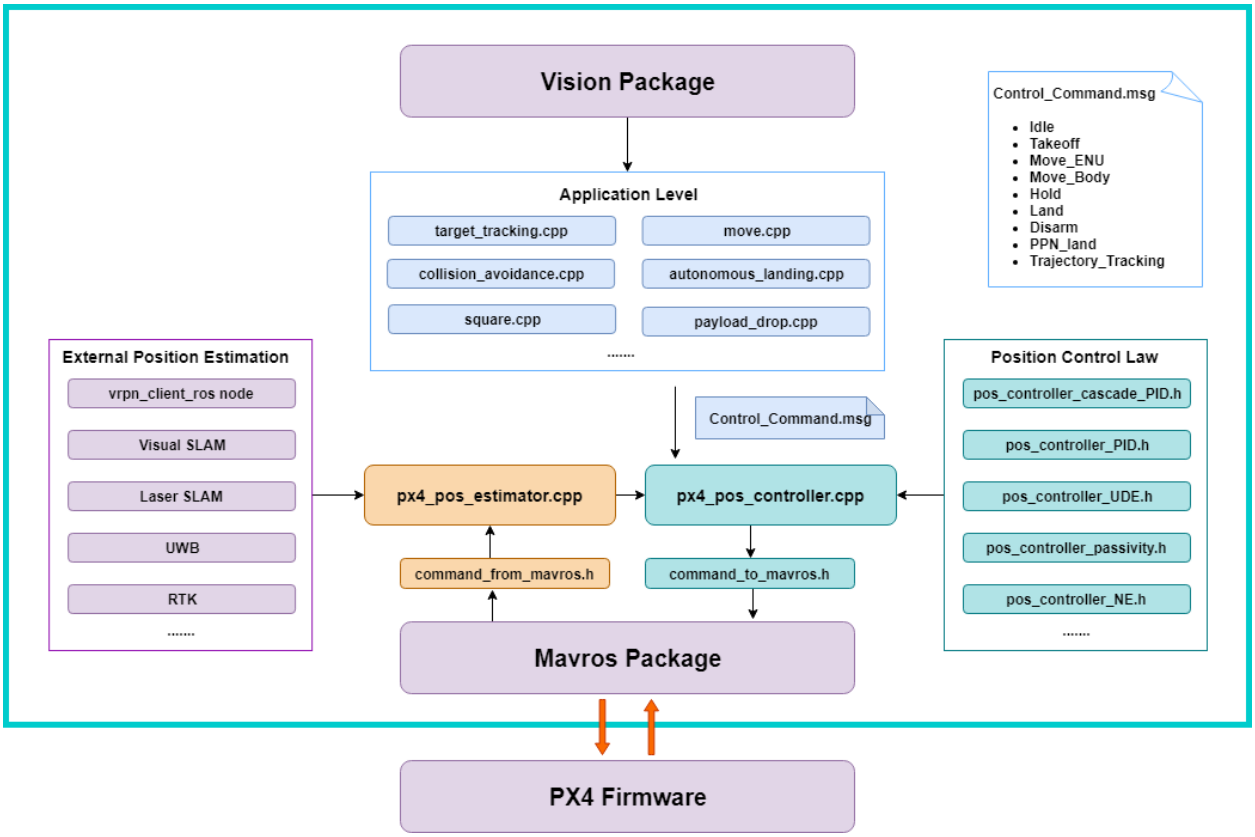
板载计算机 (二选一)	NVIDIA Nano	个	1
板载计算机 (二选一)	NVIDIA TX2	个	1
激光定高雷达	TFmini Plus	个	1

无人机参数:

无人机外形尺寸	322 X 316 X 197 mm
轴距	410 mm
整机重量飞控最大倾角	5’
电机	SunnSky X2213 KV980 或者 T-Motor Air 2216 KV880
螺旋桨	9 寸,10 寸自锁桨
电调	Hobbywing 30A
电池	4200mAh/4S

本科研无人机的更多详细, 可以直接去 淘宝 查看更多详细参数.

软件介绍



px4_command 功能包是一个基于 PX4 开源固件及 Mavros 功能包的开源项目，旨在为 PX4 开发者提供更加简洁快速的开发体验。目前已集成无人机外环控制器修改、目标追踪及避障等上层开发代码，后续将陆续推出涵盖任务决策、路径规划、滤波导航、单/多机控制并且提供完整的三维软件仿真 (Gazebo/MATLAB) 等无人机科研及开发领域的功能。

PX4: 非常流行的开源飞控代码,PX4 Firmware 可以烧写 pixhawk 系列嵌入式主板中并运行. 它的官方 Github 固件在 这里 . 在我们的 Github 中, 也 fork 了官网的固件, 之后我们修改过的 PX4 Firmware 会放到该 github 上面.

ROS 是一个平台，不仅提供各种开源代码，还提供开源代码间互相通信的机制。比如要搭建一个机器人，电机、力传感器、摄像头这些设备的底层驱动 ROS 都可以提供；还提供给你 PID 控制代码，SLAM 算法等等；还写了一整套通讯协议，让你实现各个程序间的通信，你只需要下载这些功能包，写一个简单的启动脚本，就能运行起来一个机器人系统。

Mavros: 就是 mavlink 与 ROS 连接的库, 无人机与板载计算机就是通过这种方式相互交互的.

我们可以简单的将本无人机中包含的代码分为以下几个部分：**视觉端代码**、**Mavros 代码**、**飞控代码**. 每一块代码各有分工，一个人也可能可以完全掌握这三部分代码，单需要很长时间。所以根据自己的需求，选择自己要学习的部分，事半功倍。

视觉端代码: 视觉端代码这里不叙述，基本一个功能对应一个文件夹或者 cpp 文件。比如做追踪，我们关心的是目标机体系下的位置，那么视觉端最后的输出就是目标的位置，如果不探讨视觉的具体代码，知道这些

就足够了。

Mavros 代码: Mavros 代码分为上层开发代码，位置环代码，底层代码。底层代码是下载 mavros 功能包时就有的（但我们也有一些小修改，具体看配置文件夹），位置环代码和上层开发代码是我们提供的核心代码。其中，位置环代码是我们移植了 PX4 中的位置环串级 PID 算法，然后二次封装了一些接口供用户使用，上层代码则是针对具体应用（如追踪、降落、避障等）。

飞控代码: 飞控中代码的修改和优化没有太大的必要，除非有对底层修改的必要，本无人机中只是针对特定的部分进行了修改。但是需要了解：飞控中的数据流

安全指引

飞行试验前你要做的：

1. 稳固的机架。做实验，特别是开发性质的实验，炸机是难免的，即使不炸机也常常会有不正常降落（紧急情况下从 offboard 切换回手动，或者直接 kill 电机），从个人开发经验来看，机架要耐摔耐撞比较重要。由于是室内飞行，一般飞行高度会在 0.5-2 米之内，P100 直接断电降落也不会对飞机造成太大了影响，可能螺丝会松以及粘贴的零部件会松，不会造成大的损伤。所以飞行前要检查飞机各个部件是否稳固，飞机螺旋桨是否会割到某些线。

2. 手动试飞。一个无人机要想能够自主飞行（内环 + 外环），首先要手动飞行稳定（内环）。如果你不是天天做实验，保证飞机的连续的一个良好的状态，建议每次实验前手动试飞检查一下飞机是否正常。

3. 熟悉程序。前两步都没问题，要进行任务级别的开发了（比如追踪等等），首先就是要熟悉程序，带着注释看代码还是很快的，至少了解一下程序的运行逻辑。在不上桨的情况下先测试下程序的运行是否正常。

4. 离线测试。基本上每一个程序都有详细的打印界面，会输出传感器或者飞机的状态信息。以追踪为例，在不解锁的情况下，运行所有程序，遥控器切 offboard 模式，拿一个目标标志物在镜头前面左右摆动，查看程序运行的情况，观察机载电脑给飞控发送的指令是否正确。

以上四步是真正飞行试验前必做的。虽然 P200 具备一部分到手飞的功能，但是你还是需要去熟悉这台飞机和代码，不然出了问题你根本不知道怎么去解决和优化。

说明:

- 1.P100 仍属于开发性质的平台，并不像大疆或者其他商业开源平台那样稳定。所以实验中会遇到各种各样的问题，大部分问题如果通过重启飞控或者程序解决了，就不需要咨询售后了，因为我们也天天遇到。
- 2. 做实验是有一定危险的，注意安全第一，紧急情况直接切手动降落.
- 3. 我们会陆续提供各个 demo 的教学视频，更加直观的指导大家操作
- 4. 我们也会定期开设自主无人机及 mavros 培训课程，课上会具体讲授开发流程及注意事项，敬请关注。
- 5. 祝大家飞行顺利，愉快!

手动飞行

> **警告：** 本实验室出厂的飞机都是校准正确的, 如果您拿到的是到手的飞机, 可以略过前三点. 直接看第四点手动飞行.

1. 前提准备

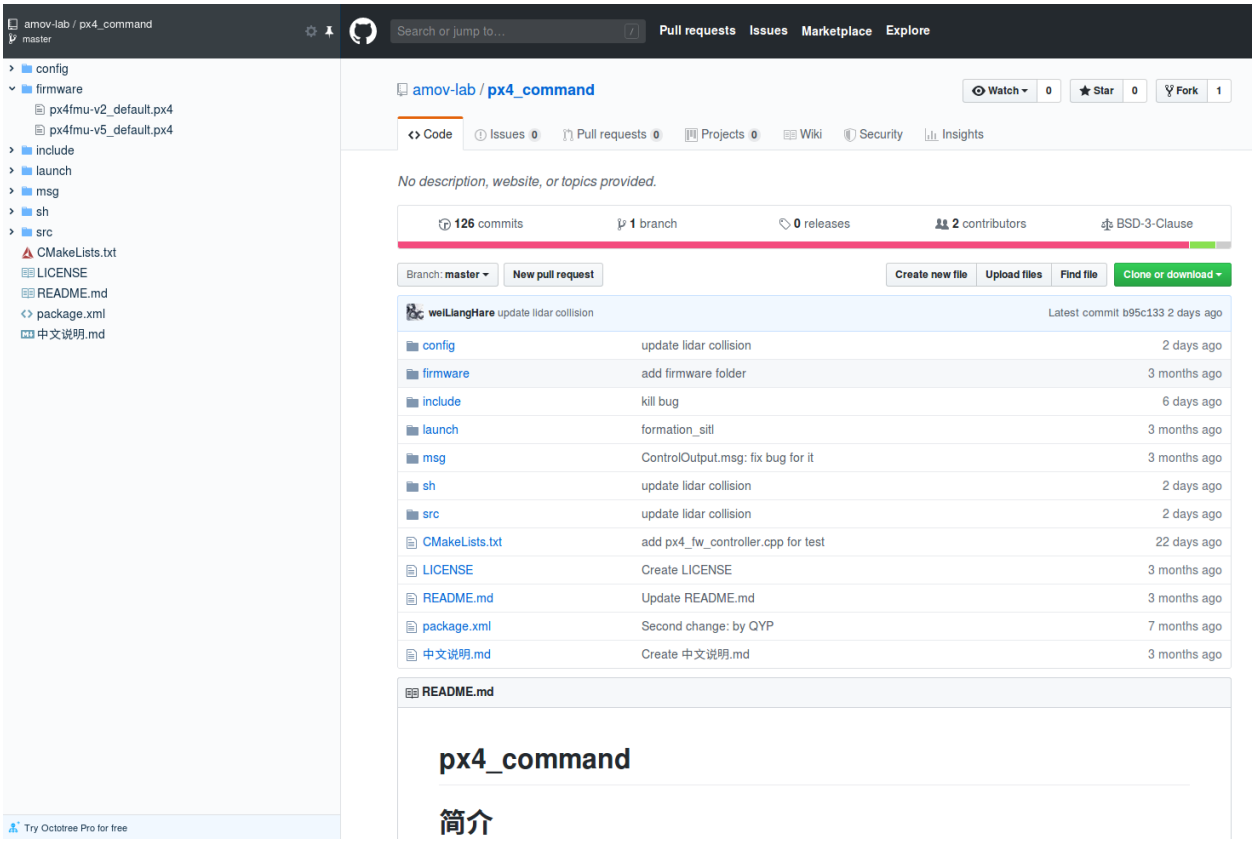
a.QGroundControl 的安装

QGC 是个可以烧写固件, 配置和校准的飞控地面站应用软件.

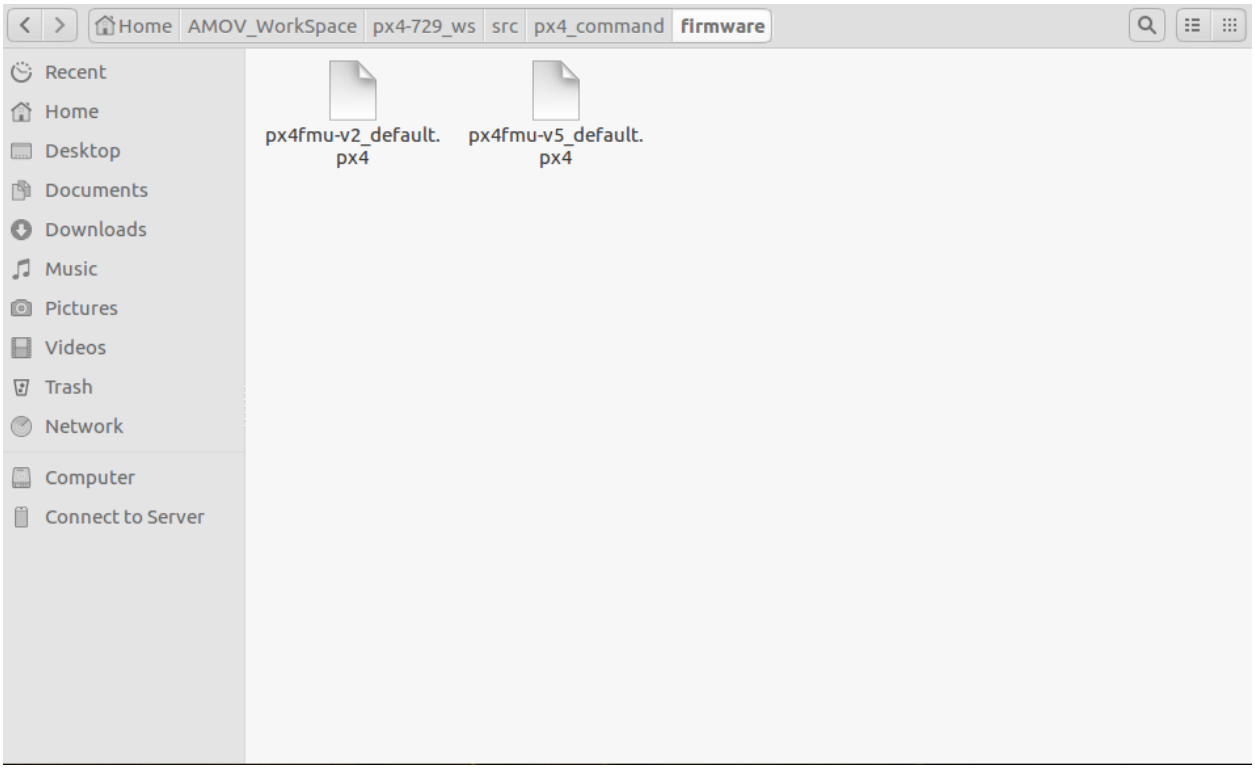
下载适合你电脑操作系统的 QGC 版本, 请参考 QGC 官网下载

b. 固件的下载

固件的下载有 v2 和 v5 版本的,github 上的 px4_command 中 firmware 中有我们提供的两种固件.



直接下载到本地就可以烧写了.



c. 遥控器与接收机使用

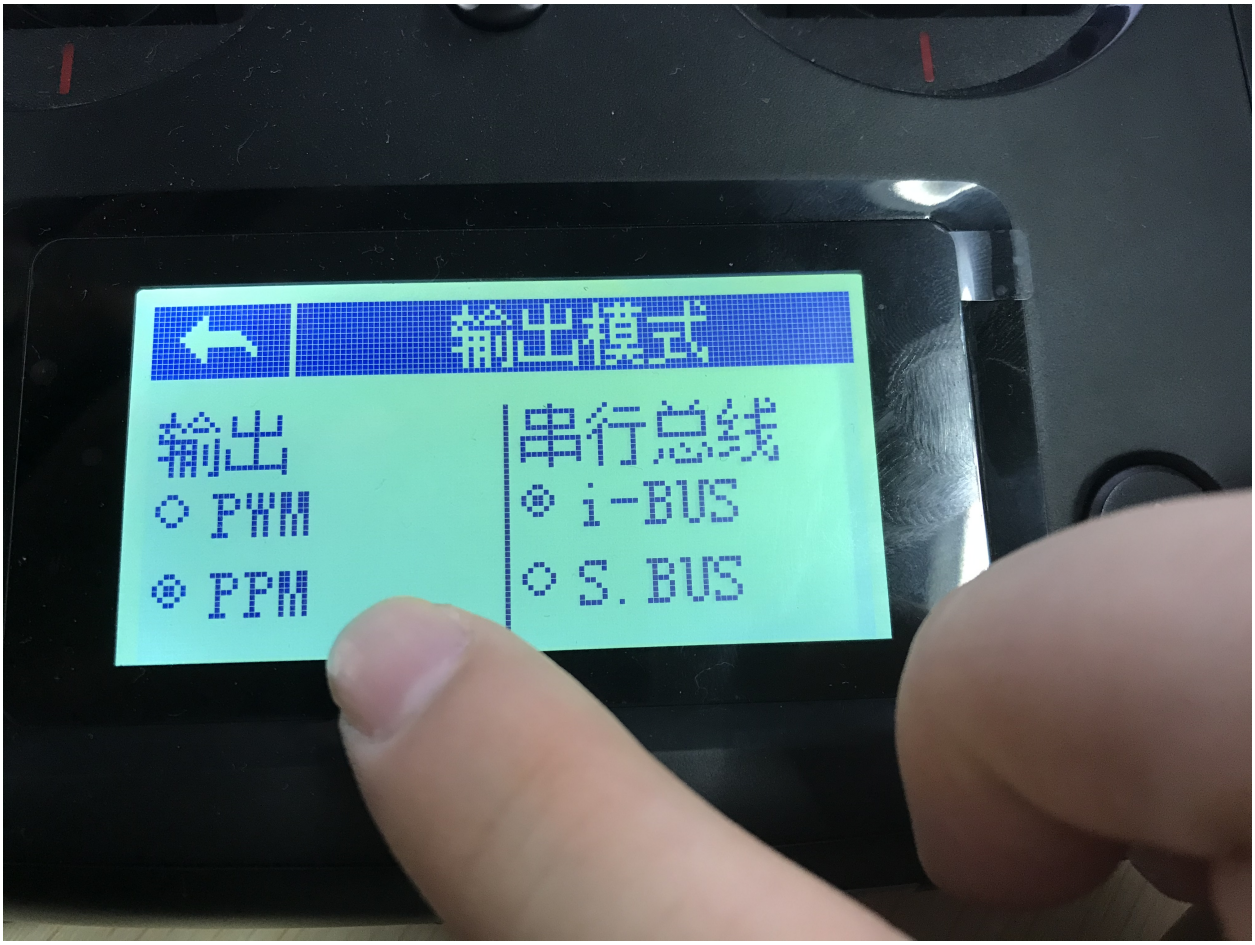
遥控器使用左手油门, 两个摇杆均回中. 使用了遥控器 6 个通道,5 通道设置为 SWC 三段开关用来控制飞行模式, 飞行模式分别为自稳模式 (stabilized), 定点模式 (position) 和降落模式 (land). 6 通道设置为 SWD 两段开关用来激活/关闭 offboard 模式. 接下来, 将介绍富斯 I6S 该款遥控器常用的配置流程.

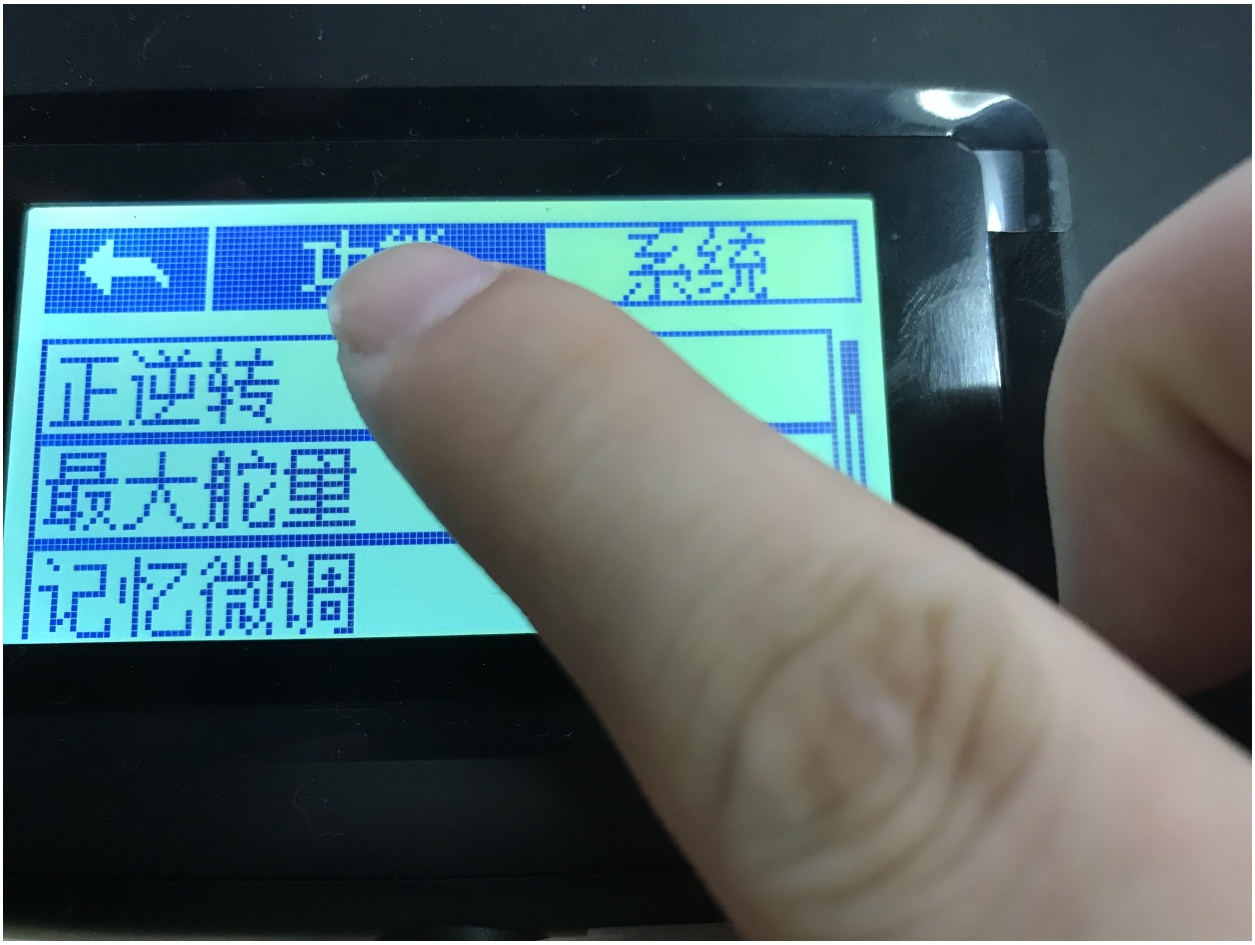
在未对频时, 遥控器开机之后如下图所示, 没有 RX. 表示该遥控器还未与接受机对频.

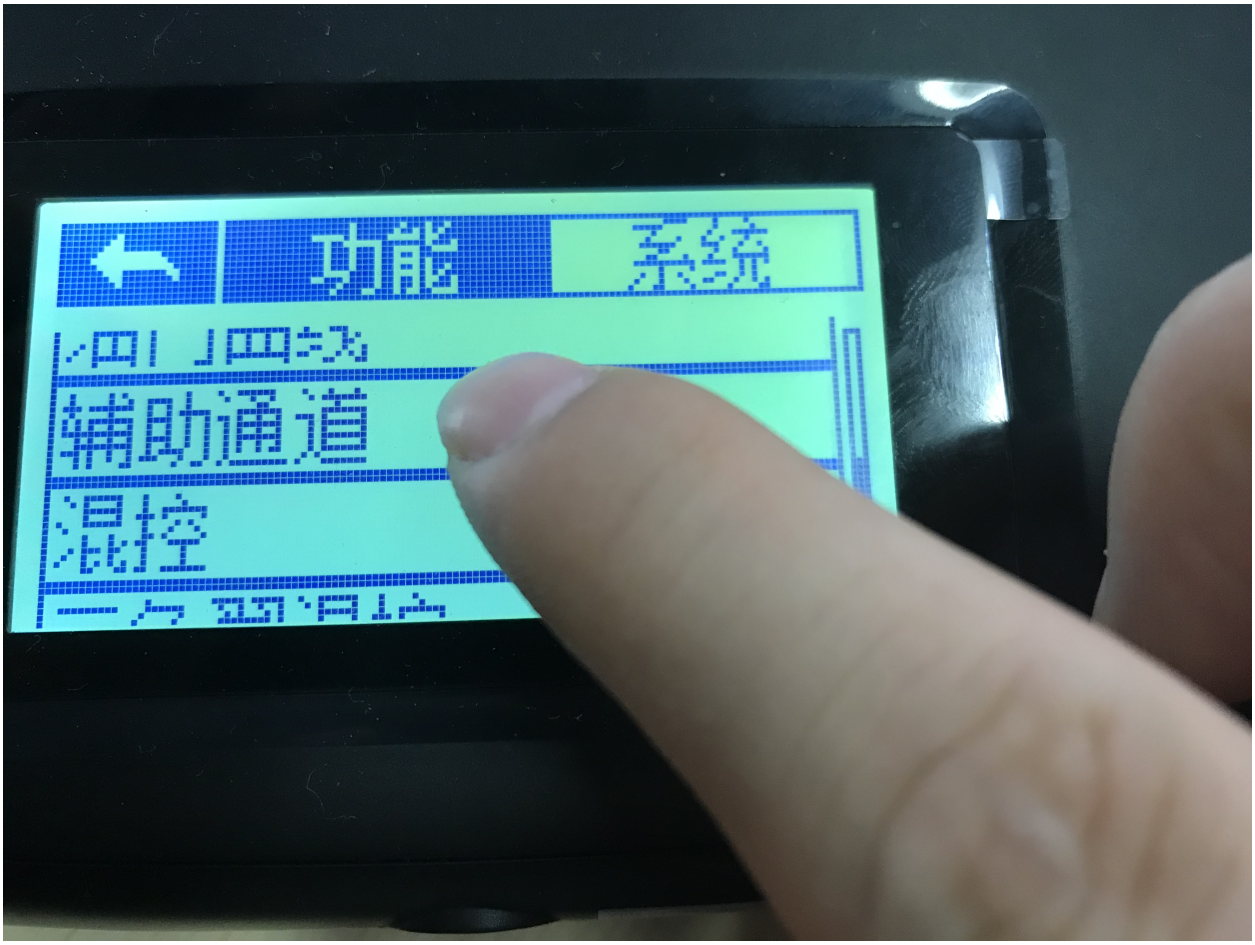

T1 00:00:00
T2 00:00:00
TX
RX
飞行模式
关闭


T1 00:00:00
T2 00:00:00
TX
RX
飞行模式
关闭


功能
对码
模型
输出模式


输出模式
输出
PWM
PPM
串行总线
i-BUS
S.BUS


系统
正逆转
最大能量
记忆微调


功能
系统
辅助通道


辅助通道
通道 5
SwC


辅助通道
通道 6
SWD


LED
UPDATE
FLY SKY
4.0-6.5V/DC
2.4GHz 10 CHANNEL RECEIVER
FS-iA10B
AFHDS 2A
B/VCC
CH10
CH9
CH8
CH7
CH6
CH5
CH4
CH3
CH2
PPM/CH1
i-BUS
SERVO
SENS

2. 地面站设置

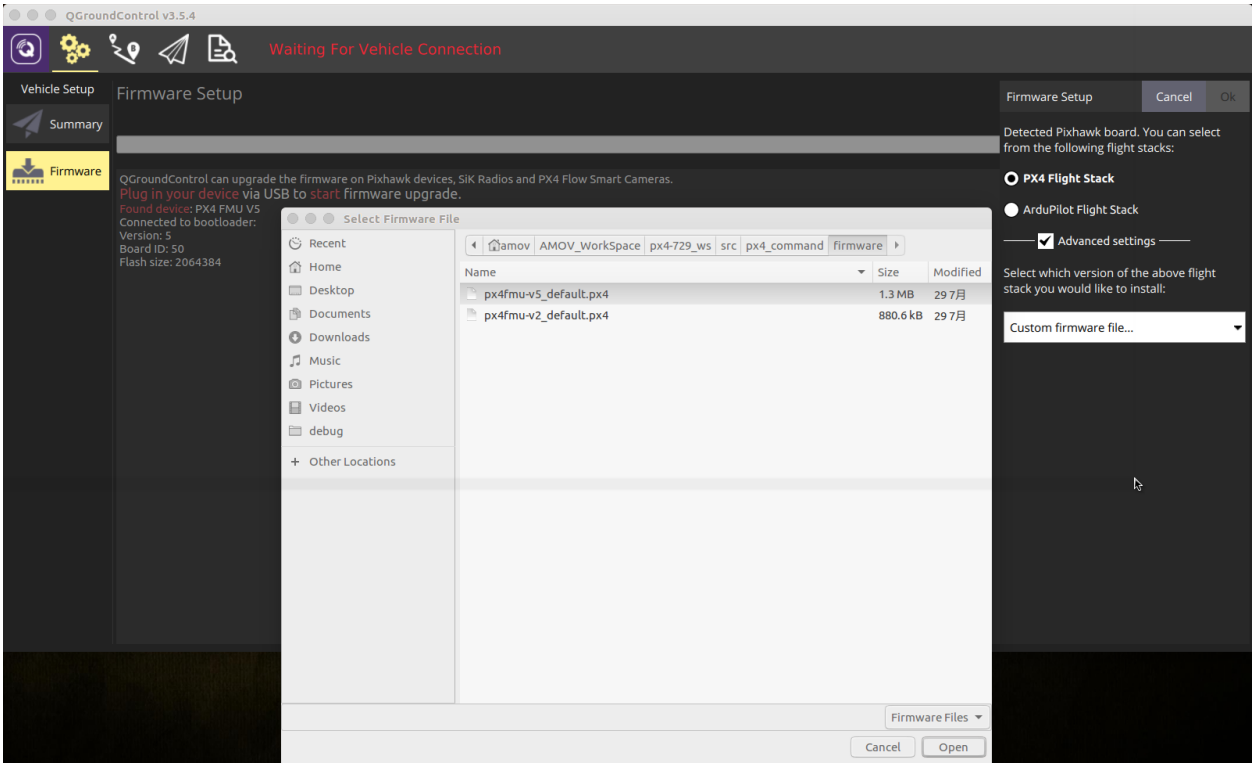
参数的设置, 包括: 固件烧写, 机型选择, 传感器校准, 遥控器校准, 飞行模式设置, 电源设置之后的话, 可能会配合视频使用.

主要参考文章: https://docs.qgroundcontrol.com/en/SetupView/SetupView.html

a. 固件烧写

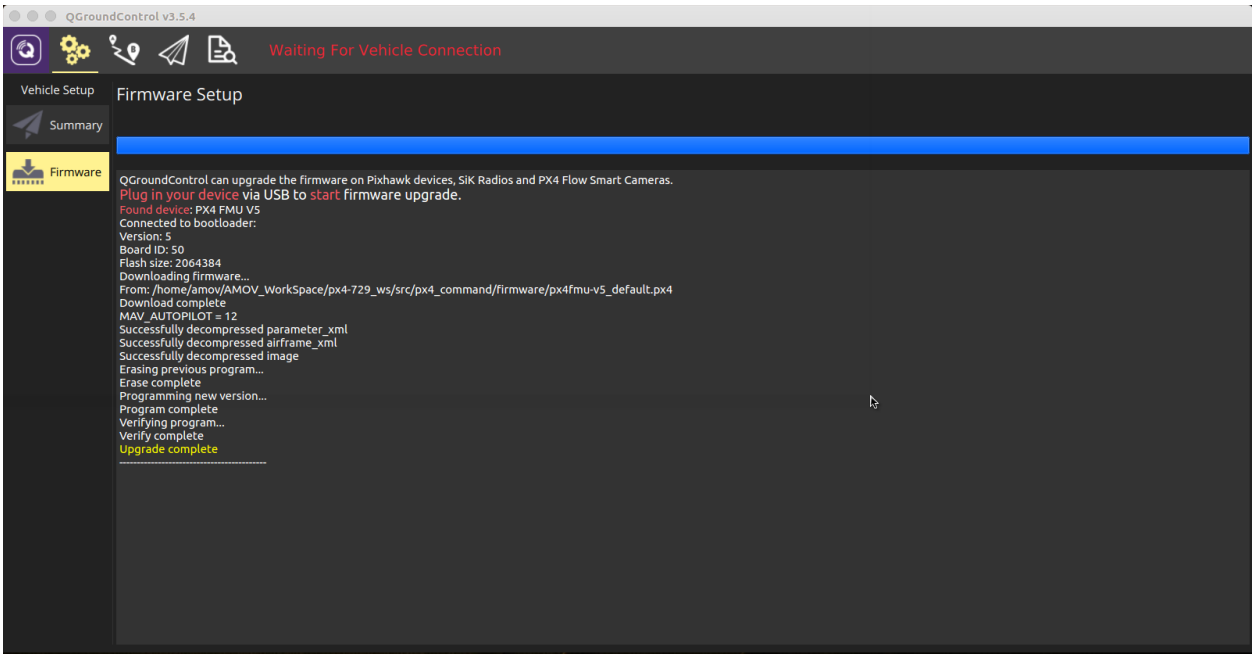
- 第一步. 打开 QGC, 点击飞行器设置图标
- 第二步. 选择固件菜单栏
- 第三步. 通过 USB 连接飞控与电脑端
- 第四步. 等待飞控连接地面站
- 第五步. 在右边选择框选择 PX4 飞行控制栈

在这里, 我们使用提供的固件:



- 点击高级设置
- 选择我们下载好的固件文件 (v2 或者 v5)
- 选择 OK 烧写固件

如果要用官网最新的固件, 直接点击 OK 就行. 等待烧写完成.



b. 机型选择

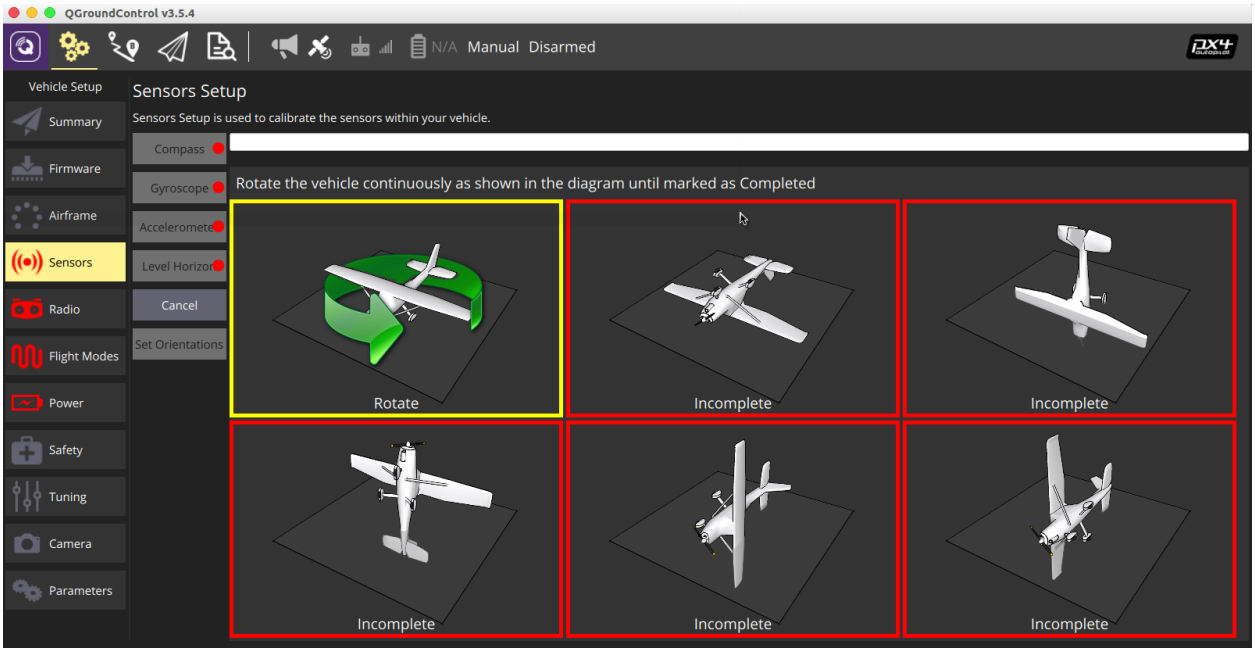
- 1. 点击飞行器设置图标
- 2. 选择机型菜单栏
- 3. 选择四旋翼机架类型,P200 使用的是 DJI450 机架. 如果没有合适的轴距, 就选择通用机架.
- 4. 选择完成之后, 点击右上角应用并重启
- 5. 等待飞控重启

c. 传感器校准

传感器的校准, 首先点击飞行器设置图标; 然后选择传感器菜单

罗盘校准:

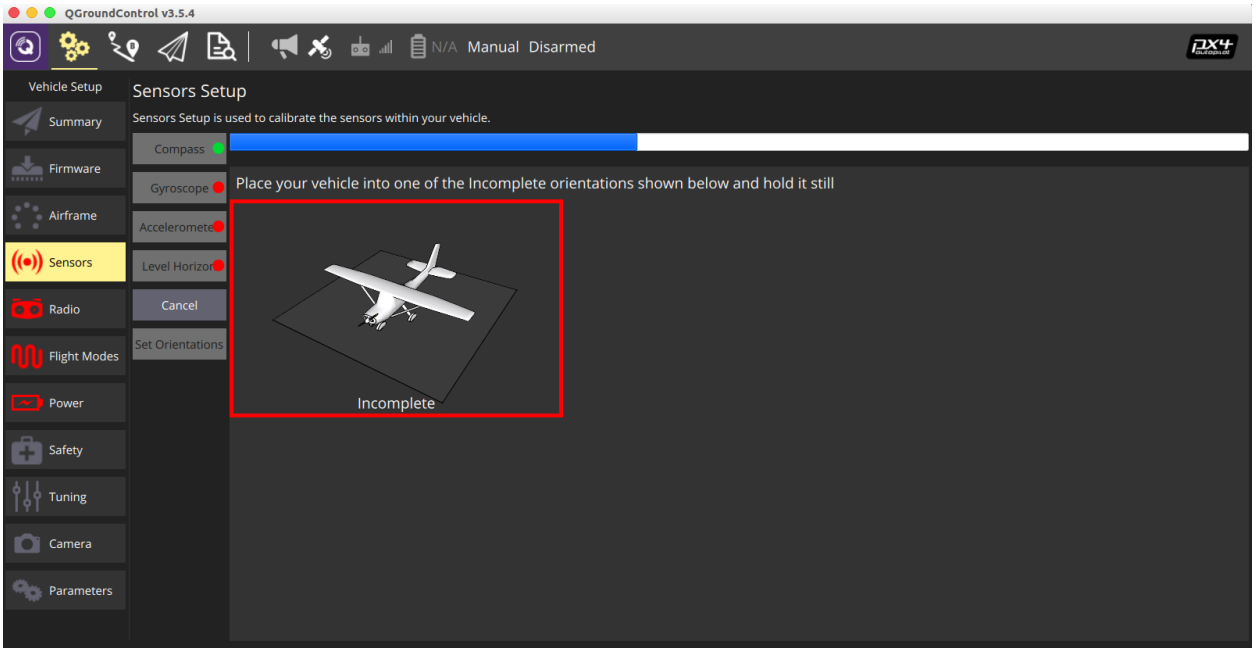
- 选择罗盘子菜单
- 选择飞控的方向 (如果飞控上的箭头方向与机架方向的箭头一致对齐, 则选择默认 ROTATION_NONE)
- 点击 OK 继续
- 共有六个面需要校准, 分别为前后左右上下, 红色轮廓表示未校准该面, 黄色表示该面正在校准, 绿色表示该面已校准, 将无人机以红色轮廓标记的任意方向放置, 等待轮廓变为黄色
- 根据需要旋转无人机, 直到轮廓变成绿色. 对每个面都这样做.



文章参考至 PX4 QGC 用户手册 https://docs.qgroundcontrol.com/en/SetupView/sensors_px4.html

陀螺仪校准:

- 选择陀螺仪子菜单
- 无人机放置与水平面上 (正常放在地面站即可, 如果追求极致, 可以用水平仪放在无人机调节至水平)
- 点击 OK 继续
- 等待校准进度条完成



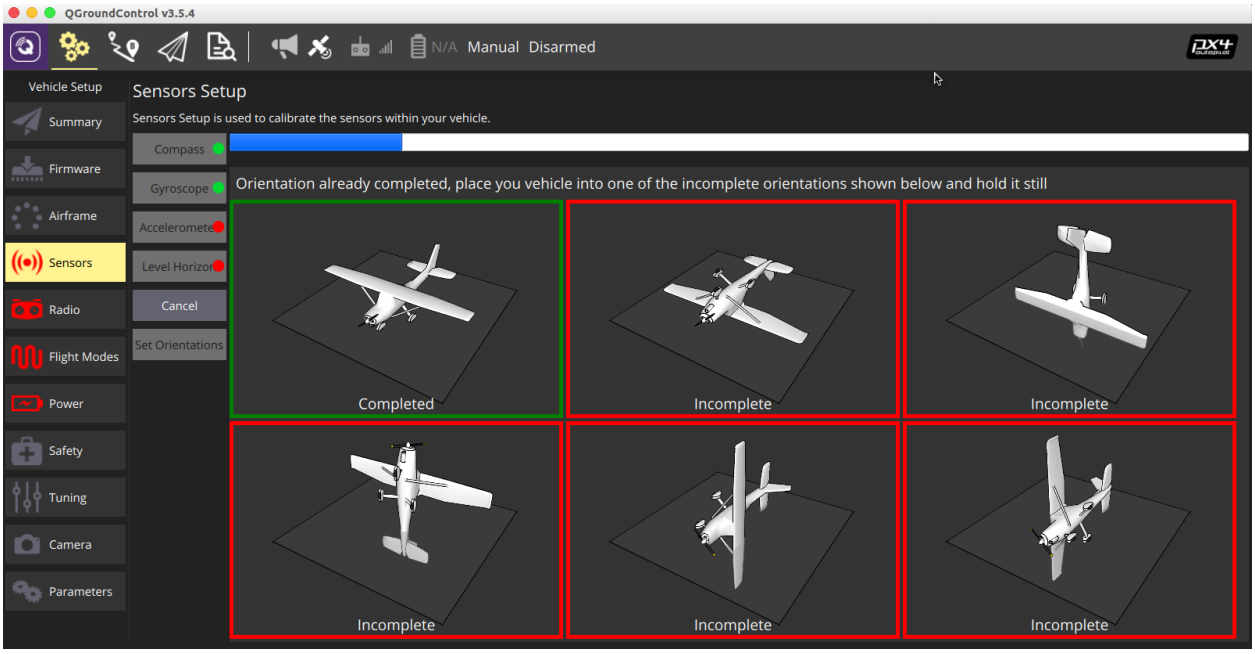
> **警告:** 无人机在校准过程中应当静止不动

文章参考至 PX4 QGC 用户手册 https://docs.qgroundcontrol.com/en/SetupView/sensors_px4.html

加速度计校准:

基本和罗盘校准类似, 不需要像罗盘校准那样旋转无人机.

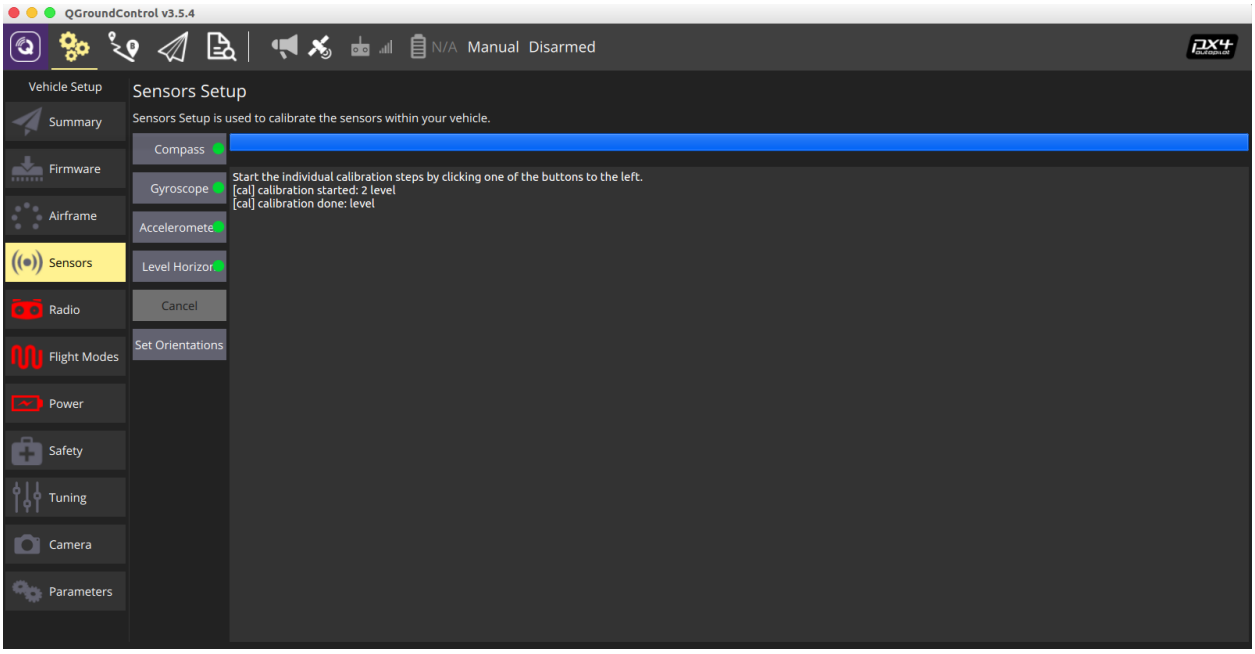
- 选择加速度计子菜单
- 选择飞控的方向 (如果飞控上的箭头方向与机架方向的箭头一致对齐, 则选择默认 ROTATION_NONE)
- 共有六个面需要校准, 分别为前后左右上下, 红色轮廓表示未校准该面, 黄色表示该面正在校准, 绿色表示该面已校准, 将无人机以红色轮廓标记的任意方向放置, 等待轮廓变为黄色
- 保持无人机在该方向不动, 知道轮廓变为绿色, 对每个面都这样做.



文章参考至 PX4 QGC 用户手册 https://docs.qgroundcontrol.com/en/SetupView/sensors_px4.html

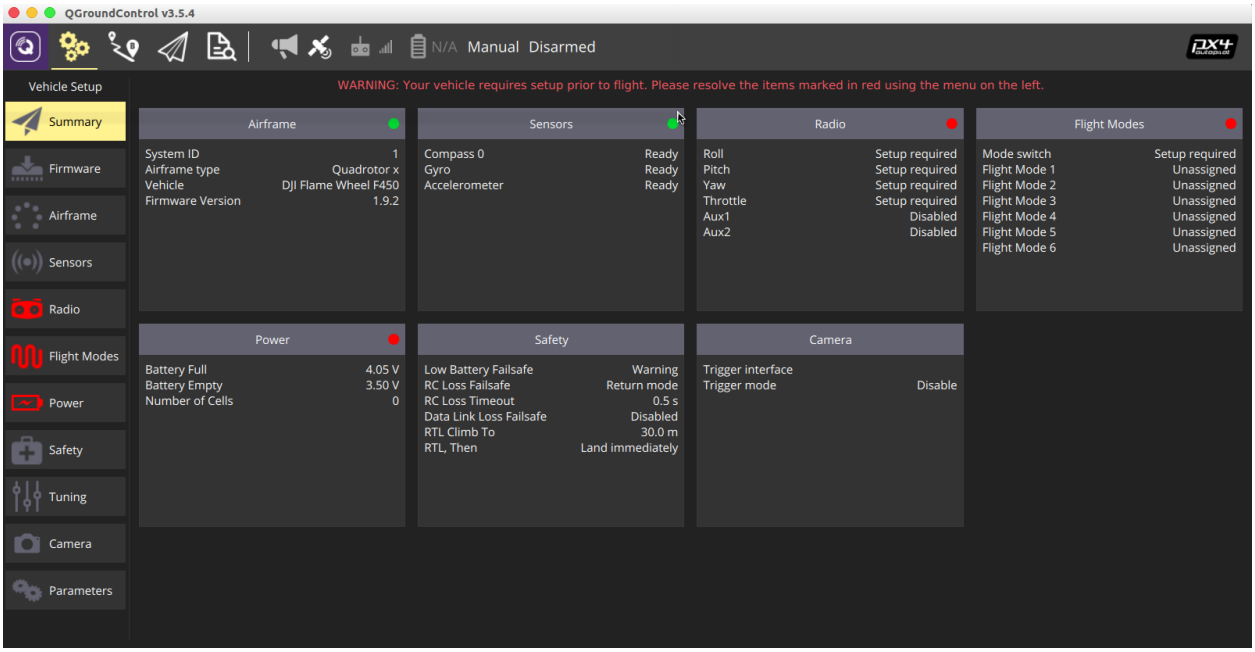
水平校准:

- 选择水平仪子菜单
- 选择飞控的方向 (如果飞控上的箭头方向与机架方向的箭头一致对齐, 则选择默认 ROTATION_NONE)
- 无人机放置与水平面上 (正常放在地面站即可, 如果追求极致, 可以用水平仪放在无人机调节至水平)
- 点击 OK 继续
- 等待校准完成



文章参考至 PX4 QGC 用户手册 https://docs.qgroundcontrol.com/en/SetupView/sensors_px4.html

所有传感器校准完成:



d. 遥控器校准

提前将油门, 偏航, 俯仰, 横滚等摇杆的微调设置为零, 这样遥控器更加精准的控制无人机.

(有两种模式, 模式 1 是日本手, 右手油门; 模式 2 是美国手, 左手油门,P200 所配置的的遥控器是美国手, 选择模式 2)

在模式二中, 左手摇杆油门拉至最低, 偏航中间. 右手摇杆回中即可

- 1. 点击飞行器设置图标
- 2. 选择遥控器菜单栏
- 3. 点击校准按钮, 继续 next, 开始校准
- 4. 按照右边的图示, 打相应的摇杆至相应的位置
- 5. 摇杆校准完成之后, 会拨动校准一下遥控器两边的 2 或 3 段开关
- 6. 上述校准完成之后, 点击 next, 会提示 “所有设置校准完成, 点击 next 把新的参数写入到你的飞控板里面”
- 7. 点击 next 即可完成遥控器校准

文章参考至 PX4 QGC 用户手册 https://docs.qgroundcontrol.com/en/SetupView/Radio.html

e. 飞行模式设置

提前在遥控器上设置号一个三段开关 (用作飞行模式开关, 一般对应的是 5 通道), 两段开关 (用作 offboard 模式开关, 一般对应的是 6 通道)

QGC 提供的默认飞行模式通道有 6 种飞行模式 (设置 6 种飞行模式需要一个二段开关 + 一个三段开关, 组合 2X3=6, 需要在遥控器中设置混控来实现此功能),P200 用的三段开关, 对应的是飞行模式 1, 飞行模式 4 和飞行模式 6 是三种飞行模式.

- 1. 点击飞行器设置图标
- 2. 选择飞行模式菜单栏
- 3. 选择通道 5 作为飞行模式开关
- 4. 飞行模式 1 设置为” 自稳模式 (stabilized)” , 飞行模式 4 设置为” 定点模式 (position)” , 飞行模式 6 设置为” 降落模式 (land)”
- 5. 设置 6 通道为 offboard 模式开关

文章参考至 PX4 QGC 用户手册 https://docs.qgroundcontrol.com/en/SetupView/FlightModes.html

飞行模式的介绍:

手动控制: 有三种, 一种是自稳模式, 一种是半自主模式, 一种是特技模式.

辅助飞行模式: 姿态 (定高) 模式和位置 (定点) 模式

自主飞行模式:offboard 模式, 自主任务模式, 自主返航模式, 自主降落模式.

f. 电源设置

电源设置中, 我们需要设置目前电池电压量, 所以需要 bb 响, 实测一下电池电压量. 电调的校准也是在这完成的.

校准电源传感器

- 1. 点击飞行器设置图标
- 2. 选择电源菜单栏
- 3. 输入电池芯数为 3 或 4, 满电电压 (每芯) 为 4.2V, 空电电压 (每芯) 为 3.7V (P200 无人机为 4S 电池)
- 4. 准备校准电压分压器, 提前实测测好电池总电量
- 5. 点击电压分压器旁边的”校准”按钮, 输入实测的总电池电压, 点击校准即可
- 6. 等待飞行器的电压与实测电压一致时, 点击右上方”关闭”, 完成电源模块校准

文章参考至 PX4 QGC 用户手册 https://docs.qgroundcontrol.com/en/SetupView/Power.html

电调校准 (ESC):

> **警告:** 永远不要尝试在装桨叶的时候校准电调, 因为在校准过程中电机会以最大转速旋转的.

- 1. 确保电池当前与没有接通, 且使用 USB 连接上飞控与 QGC 地面站
- 2. 点击”校准”
- 3. 当提示你接上电池, 这时候再接上电池
- 4. 等待校准完成

文章参考至 PX4 QGC 用户手册 https://docs.qgroundcontrol.com/en/SetupView/Power.html

3.PX4 入门

a. 日志分析

有关 PX4 固件飞行过程中的详细数据, 可以查看飞行日志分析. 飞行日志是 uORB 主题中的消息, 后缀名为.ulg 的文件. 首先可以用 QGC 通过数传 (速度慢) 或者 USB 在”分析图标”下面的”日志下载”菜单栏中下载对应的日志文件. 另外一种获取日志方式, 拔出飞控中的 SD 卡, 然后用读卡器获取到相应的日志文件.

日志分析

日志分析工具推荐使用 flightplot, 在 github 下载与电脑操作系统一直的安装包, 该软件的使用需要有 Java 环境.

在 flightplot 软件中, 你可以查看飞控相关的所有主题, 列表出你所需要查看的主题, 然后就可以将该主题显示在图表上.

PX4 中主要的主题

主题完整的列表可以在固件下的 msg 文件 中找到. 下面是比较重要的几个主题 topic:

- vehicle_status 无人机的各个状态 (导航状态, 解锁状态, 系统状态), 飞行模式

- vehicle_local_position 无人机本地状态
- vehicle_attitude 无人机姿态角
- vehicle_local_position_setpoint 无人机位置的相对目标点
- vehicle_global_position 无人机全球位置
- vehicle_vision_position 无人机视觉位置
- att_pos_mocap
- actuator_controls 电机的信号控制
- vehicle_land_detected 无人机降落检测

b.PID 调参

4. 手动飞行

如果是到手的飞机, 你可以不用执行上面第二点. 在飞机出厂之后我们都已经校准好了, 你可以直接开始手动飞行. 在上面的遥控器对频使用过程中, 也讲过遥控器上面设置三种飞行模式, 分别为自稳模式, 定点模式, 降落模式. 如果你对 px4 的飞行模式还不了解的话, 请参考 飞行模式介绍

纯手动飞行模式: 在飞行模式为 stabilized 下, 手动控制飞行, 室内中没有 GPS 情况下,GPS 的指示灯为蓝色闪烁, 此时, 可以手动解锁, 控制无人机飞行. 如果没有飞行经历的话, 建议现在模拟器上熟练了遥控器, 然后在实际飞行

辅助飞行模式: 定高或者定点飞行, 定高飞行不需要使用 GPS, 定点模式飞行需要 GPS, 在室外可以测试飞行, 定点模式 (position) 有油门阈值, 在油门量的 40%~60% 是油门死区. 高于 60% 或者低于 40% 油门摇杆才会有向上或者向下的动作.

自主飞行模式:mission 模式中, 可以在 QGC 地面站上面规划预先规划好了的路径, 该飞行模式也是需要在室外有 GPS 的地方测试.

文章参考至 PX4 用户使用手册 https://docs.px4.io/master/en/flying/missions.html

下面介绍到手飞机的手动飞行说明:

- 首先清楚遥控器的 SWC 三段开关代表的是飞机的三种飞行模式, 向外 (远离摇杆) 飞行模式为自稳模式 (也就是手动模式), 中间 (开关位置在中间) 飞行模式为定点模式 (在手动飞行中无需使用), 向内 (靠近摇杆) 飞行模式为降落模式 (在手动飞行中无需使用)
- 遥控器是美国手. 左手油门. 左手上下表示油门大小, 左手左右表示偏航, 右手上下表示俯仰, 右手左右表示横滚.

手动飞行之前将 WIFI 数传配置为 AP 模式 (也可以配置为网卡模式), 利用同一局域网下,QGroundControl 使用 TCP 连接到飞控, 查看飞控当前状态.

手动飞行步骤:

- 上电, 连接 WIFI 数传至 QGroundControl, 查看飞控当前状态有无报错

- 遥控器解锁, 解锁方式为内八式解锁, 解锁之后飞机有怠速
- 遥控器控制飞机, 油门慢慢推起至飞机飞起来, 如果飞机有明显倾斜, 可以一遍慢慢推油门, 一遍稍微打一点俯仰或者横滚 (根据飞机倾斜位置, 反方向修正)
- 正常用遥控器控制飞机各个姿态变化.

自主飞行之入门

1.ROS

ROS 官网: 维基

ROS 是用于开发复杂的分布式机器人系统的广泛使用的框架.

a. 安装

主要参考: 官网安装说明

在飞机上的机载计算机 TX2 上已经安装好了 ROS.

如果要在 PC 上面使用 ROS, 我们建议使用 Ubuntu Linux 系统.Windows 系统下面可以使用 VMware 虚拟机,Mac 系统可以使用 **VirtualBox** 或者是 **Parallels Desktop Lite**

小技巧: 我们建议使用 Ubuntu16.04 与之相对应的 ROS 版本 Kinetic.

---

b. 基本概念

节点 (nodes):

主要参考来源: ROS 节点

ROS 节点是一个特殊的程序, 通常使用 Python 或者 C++ 编写, 可以通过 ROS 主题和 ROS 服务与其他节点进行通信. 将复杂的机器人系统划分为孤立的节点具有某些优点: 减少代码的耦合性, 提高可重用性和可靠性

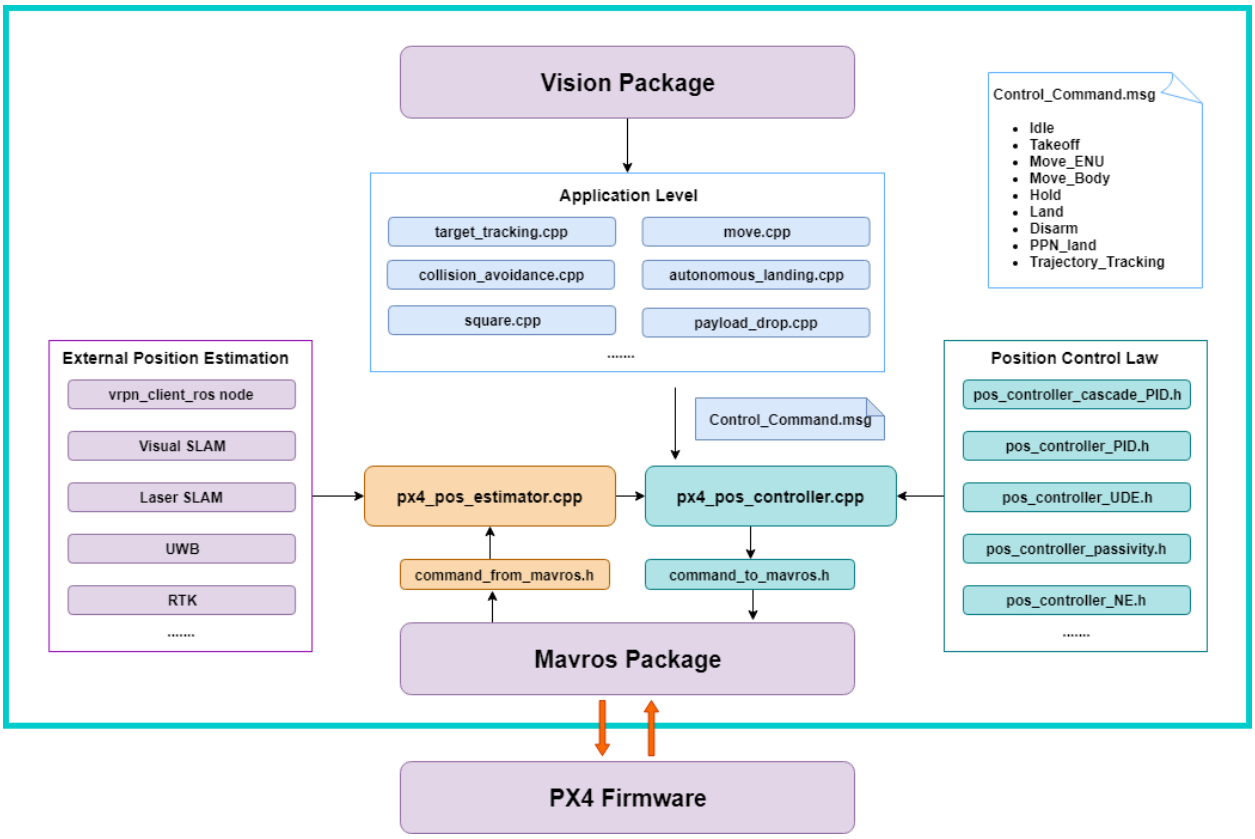
许多机器人库和驱动程序都是以 ROS 节点的形式执行.

为了将普通程序变成 ROS 节点, 请包含 rospy 或 roscpp 库, 并插入初始化代码.

2.MAVROS

3.px4_command 介绍

px4_command 功能包是一个基于 PX4 开源固件及 Mavros 功能包的开源项目，旨在为 PX4 开发者提供更加简洁快速的开发体验。目前已集成无人机外环控制器修改、目标追踪、激光 SLAM 定位、双目 V-SLAM 定位、激光避障等上层开发代码、后续将陆续推出涵盖任务决策、路径规划、滤波导航、单/多机控制等无人机/无人车/无人船科研及开发领域的功能。配合板载计算机 (树莓派、TX2、Nano) 等运算能力比较强的处理器，来实现复杂算法的运行，运行得到的控制指令通过串口或者网口通信发送给底层控制板。



- **state_from_mavros.h**: 订阅飞控状态，包括无人机当前的状态 (/mavros/state)，当前位置 (/mavros/local_position/pose)，当前速度 (/mavros/local_position/velocity_local)，和当前角度，角速度 (/mavros/imu/data)
- **command_to_mavros.h**: 发布 px4_command 功能包生成的控制量至 mavros 功能包，可发送期望位置，速度 (本地系与机体系)，角度，角速度，底层控制 (遥控器输入)
- **px4_pos_estimator.cpp**: 订阅激光雷达或者 mocap 发布的位置信息，并进行坐标转换，在 state_from_mavros.h 中已订阅飞控发布的位置，速度，欧拉角信息，此处直接使用，根据订阅的数据，发布相应的位置，偏航角给飞控
- **px4_pos_controller.cpp**: 订阅由位置估计发布的 DroneState，初始化当前飞机状态的时间. 订阅 ControlCommand(不知从何发布的数据). 发布 topic_for_log 主题. 在选择控制率，检查参数正确后，初始化完成. 对 move 节点中,takeoff,Move_ENU,Move_Body,Hold,Land,Disarm,PPN_land 和 Trajectory_Tracking 等进行逻辑处理.

- **ground_station.cpp**: 订阅自定义日志主题 (/px4_command/topic_for_log), 订阅视觉系统位置估计 PoseStamped 主题 (/vrpn_client_node/UAV/pose, 非 mavlink 消息, 数据包括 point 位置 (x,y,z), 四元数方向 (w,x,y,z)), 订阅飞控姿态四元数 AttitudeTarget 主题 (/mavros/setpoint_raw/target_attitude,#82 号 mavlink 消息). 不断的更新视觉传感器状态, 并打印当前飞机的状态.
- **px4_sender.cpp**: 订阅自定义消息控制指令主题 (/px4_command/control_command), 机体系到惯性系坐标转换,move 中控制命令的具体实现 (0 表示位置控制,3 表示速度控制)
- **autonomous_landing.cpp**: 降落识别使用 xyz 均为速度控制. 订阅数据包括降落板与无人机的相对位置, 降落板与无人机的相对偏航角, 视觉 flag 来自视觉节点. 最后发布位置控制指令
- **collisiom_avoidance_streo.cpp**: 订阅/streo_distance 该数据作为计算飞机四个方向的距离判断.
- **formation_control_sitl.cpp**: 多机仿真 SITL, 只适用于 Move_ENU 坐标系下, 若使用 Move_Body, 需自行添加修改.
- **payload_drop.cpp**: 订阅/mavros/local_position/pose 本地位置. 发布遥控器通道值.
- **square.cpp**: 发布/px4_command/control_command 命令. 子模式 xyz 均为位置控制.
- **target_tracking.cpp**:
- **move.cpp**: 发布/px4_command/control_command, 并设置子模式 xy 速度控制 (0b10), 位置控制.z 速度控制 (0b01), 位置控制
- **set_mode.cpp**: 模拟遥控器, 根据 mavros 服务, 进行在 SITL 下解锁, 切换 offboard, 控制飞行器.
- **TFmini.cpp**: 激光定高雷达的处理, 如果需要添加超声波传感器, 可参考此代码.

4. 飞行前准备

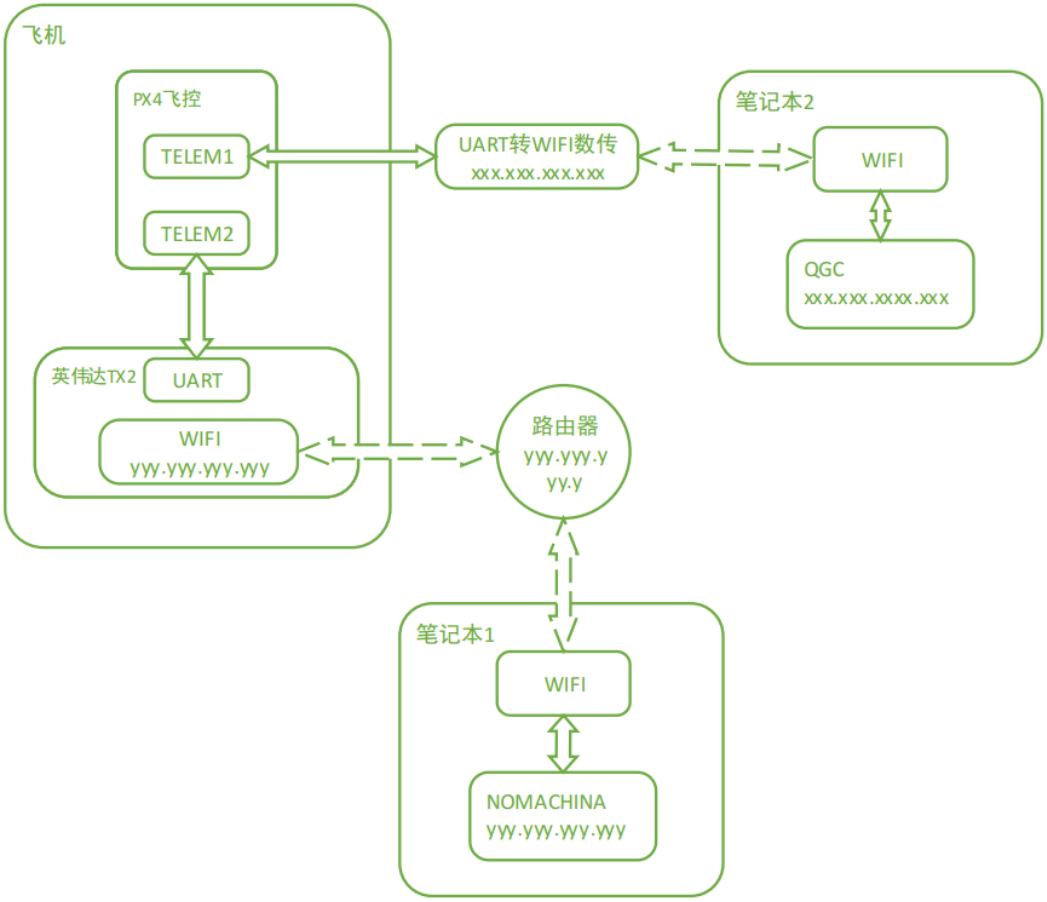
WiFi 数传两种配置模式

小技巧: WiFi 数传的配置工具请在百度网盘自主下载 网盘链接

提取码: **exk3**

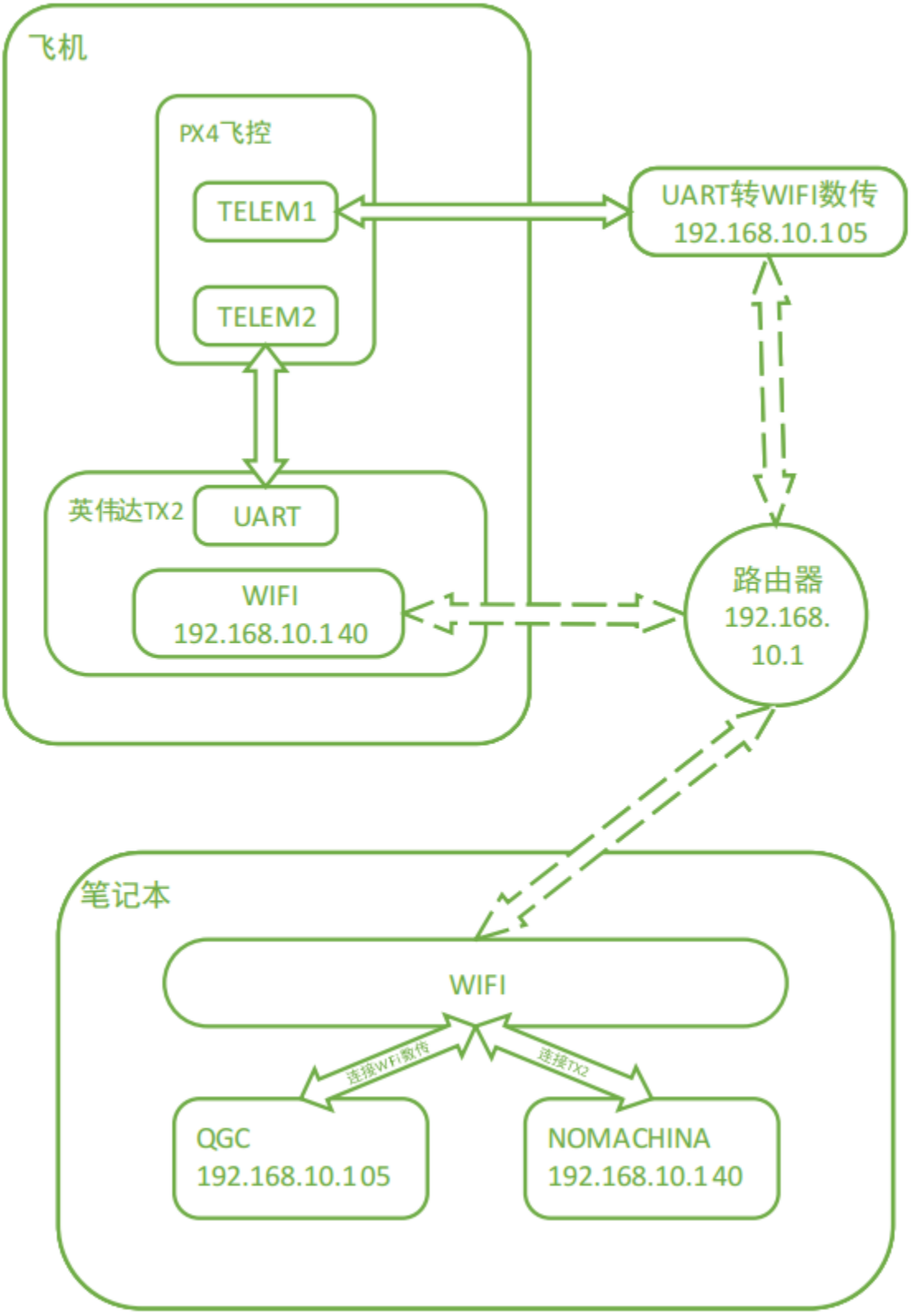
根据 WiFi-LINK 配置模式有两种连接方式：一种是无线 AP 模式, 另外一种无线网卡模式. 出厂默认配置为 AP 模式, 如需配置为无线网卡模式, 可按照无线网卡模式说明文档自行配置

WIFI-LINK 配置成无线 AP 模式



这是我们发货时的默认配置，其中 WiFi-LINK 模块相当与一个热点，笔记本 2 要连接这个热点进而连接 QGC. 他们配置详细步骤请看 WIFI 数传远程连接 QGroundControl 非正常连接篇

WiFi-LINK 配置成无线网卡模式



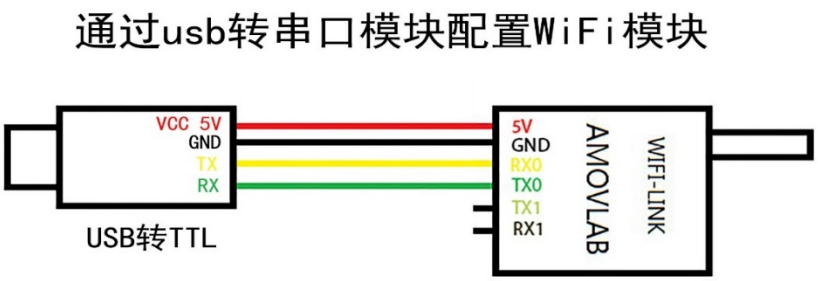
路由器设置

如图为无线 WiFi 路由器默认 IP 地址为 192.168.10.1，登录到此路由器的后台，用已连接此 WiFi 的手机或电脑登陆地址 wifi.wavlink.com（默认密码为 admin）。

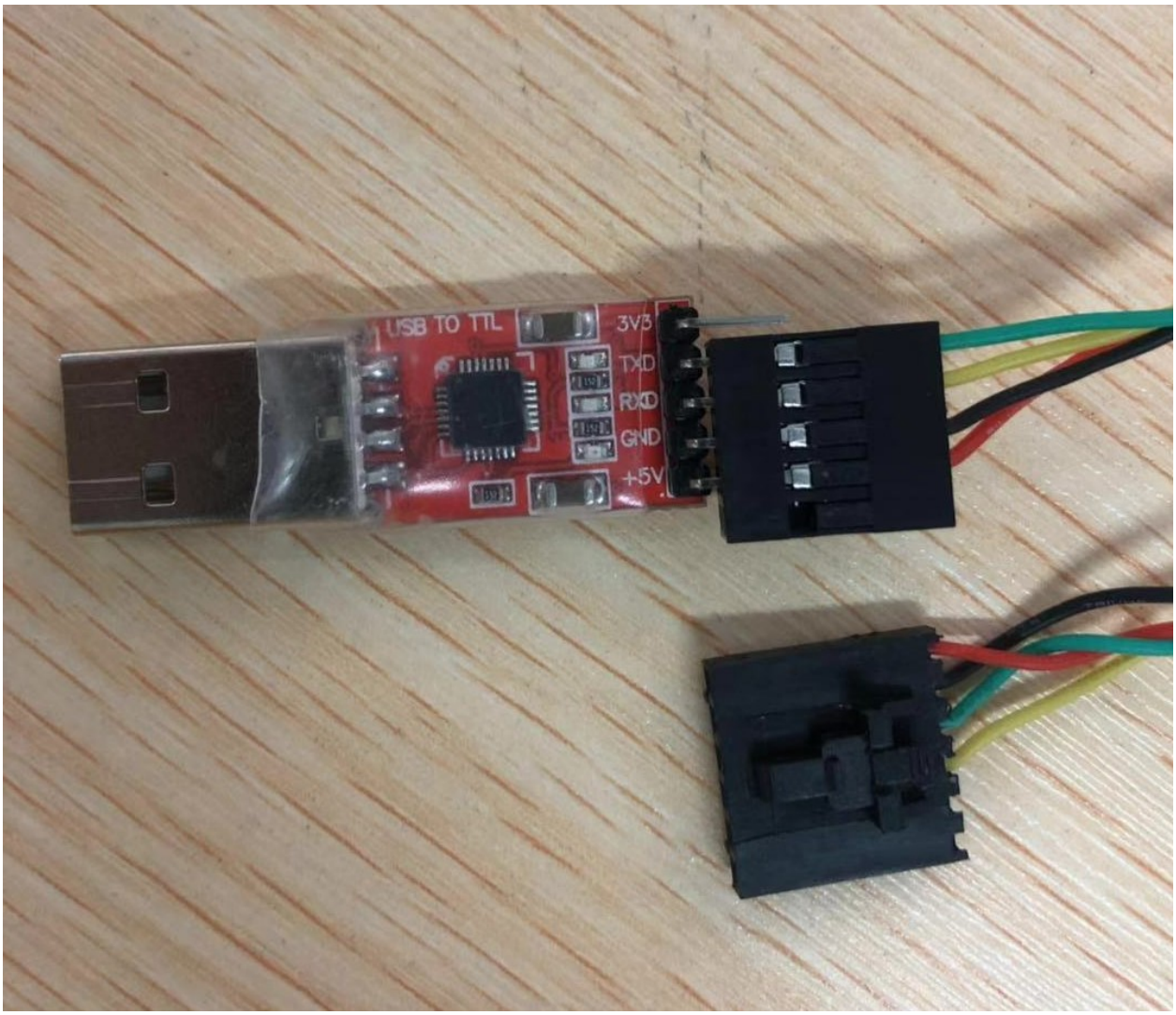
WIFI 设置：设置 WiFi 的名称（此名称会在数传设置中用到）加密方式选择 WPA2-PSK

WiFi 数传设置

- 硬件连接

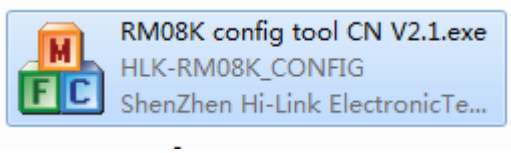


可用四根杜邦线连接 usb 转 TTL 与 WiFi-LINK，下图是线序



- 软件配置

打开配置软件 RM08K config tool CN V2.1.exe



选择相应的 COM 口使用一个细长的金属棒或者牙签点击 WiFi 数传模块的 reset 按钮（模块上远离天线的那个孔），然后点击搜索模块按钮，软件显示如下（Found Device at COMX(57600)）说明连接成功。

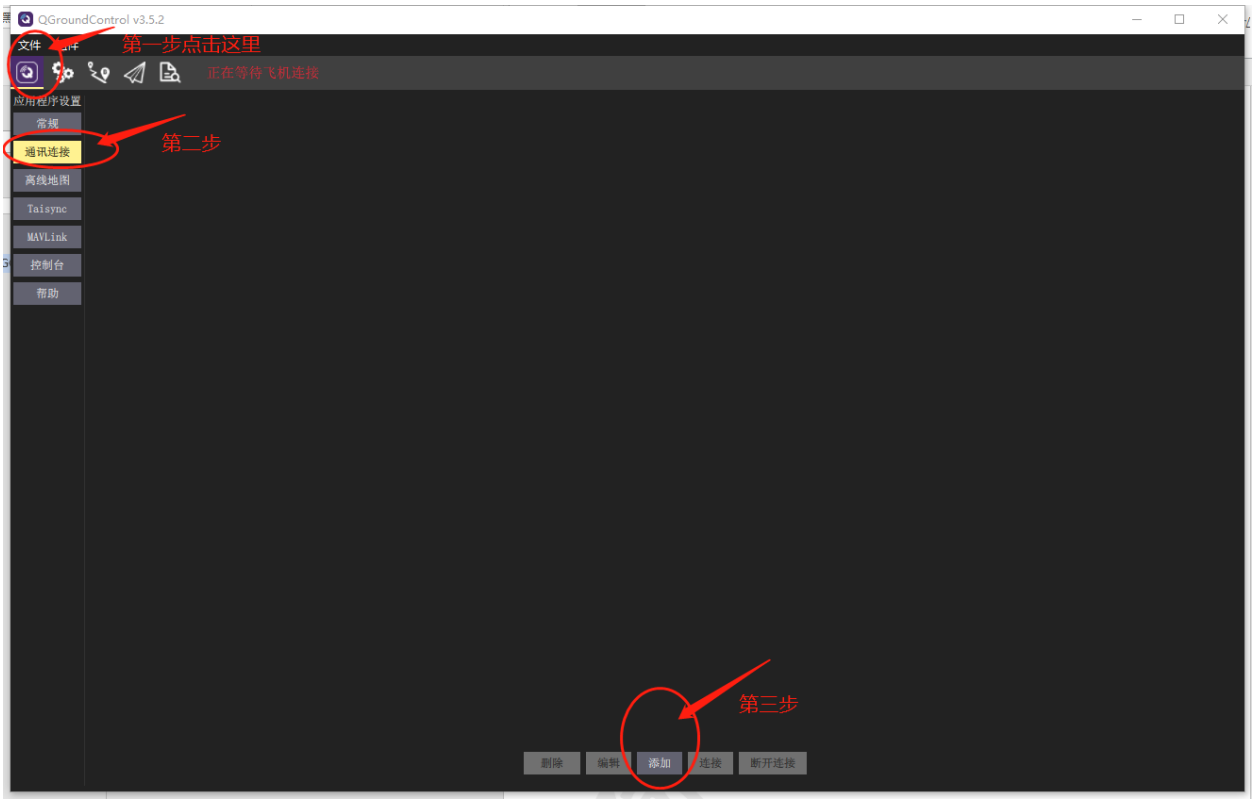
然后下图所示配置你们的 WiFi 数传，其中网关要选择你的路由器的网段，网络名称及密码设置你的 WiFi 路由器的名称与密码，其中加密方式选择 WPA2_AES。

设置完成后点击提交配置即可。

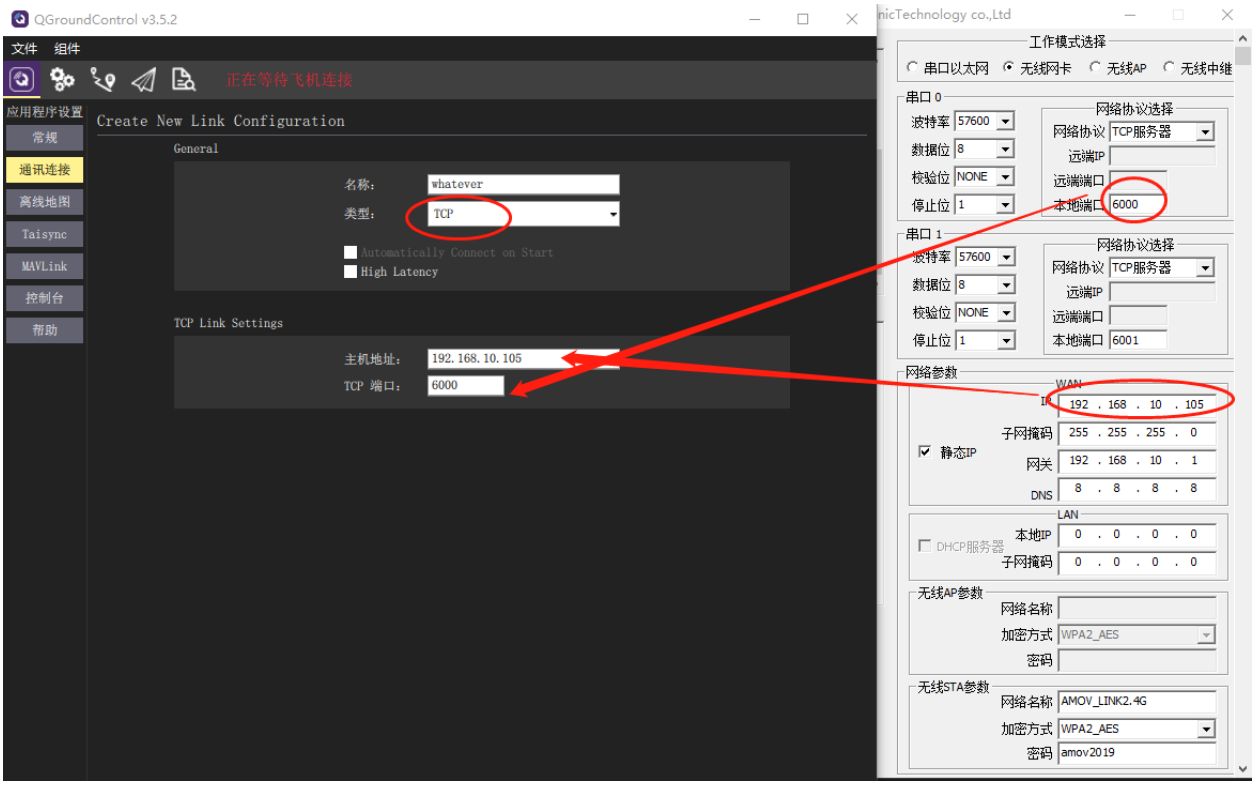
接下来测试是否配置成功：重启 WiFi 数传，假设你配置成功，那么 WiFi 数传会自动连接 WiFi 路由器。那么我们用连接此 WiFi 路由器的笔记本，打开终端 (cmd)，输入 ping 192.168.10.105 尝试 ping 一下我们的 WiFi 数传，ping 通了，那么恭喜你搞定了，ping 不通那么你需要打开 RM08K config tool CN V2.1.exe 重新配置一下，可以点击查询配置，看看配置的是否有出入。

- 地面站连接: 此步骤需要笔记本连接 WiFi 路由器的后才可进行，而且你已经在笔记本上 ping 通了 WiFi 数传。

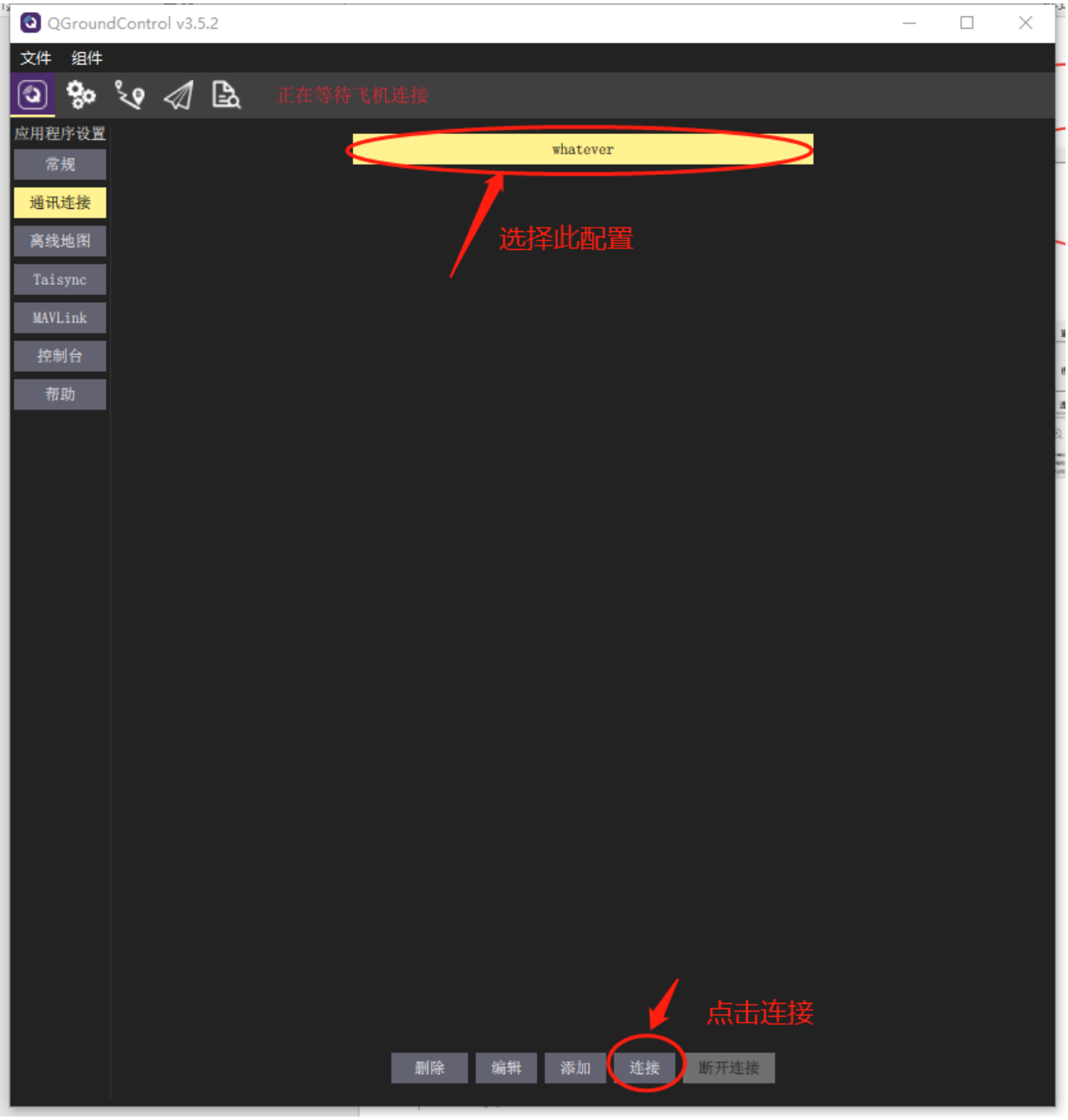
打开 QGC，如下图操作。



然后选择 TCP 类型，名称任意，主机地址为你设置得到 wifi 数传的 IP 地址，端口为 6000，然后确定即可。



点击你的配置名称，然后点击连接，即可连接飞控了。



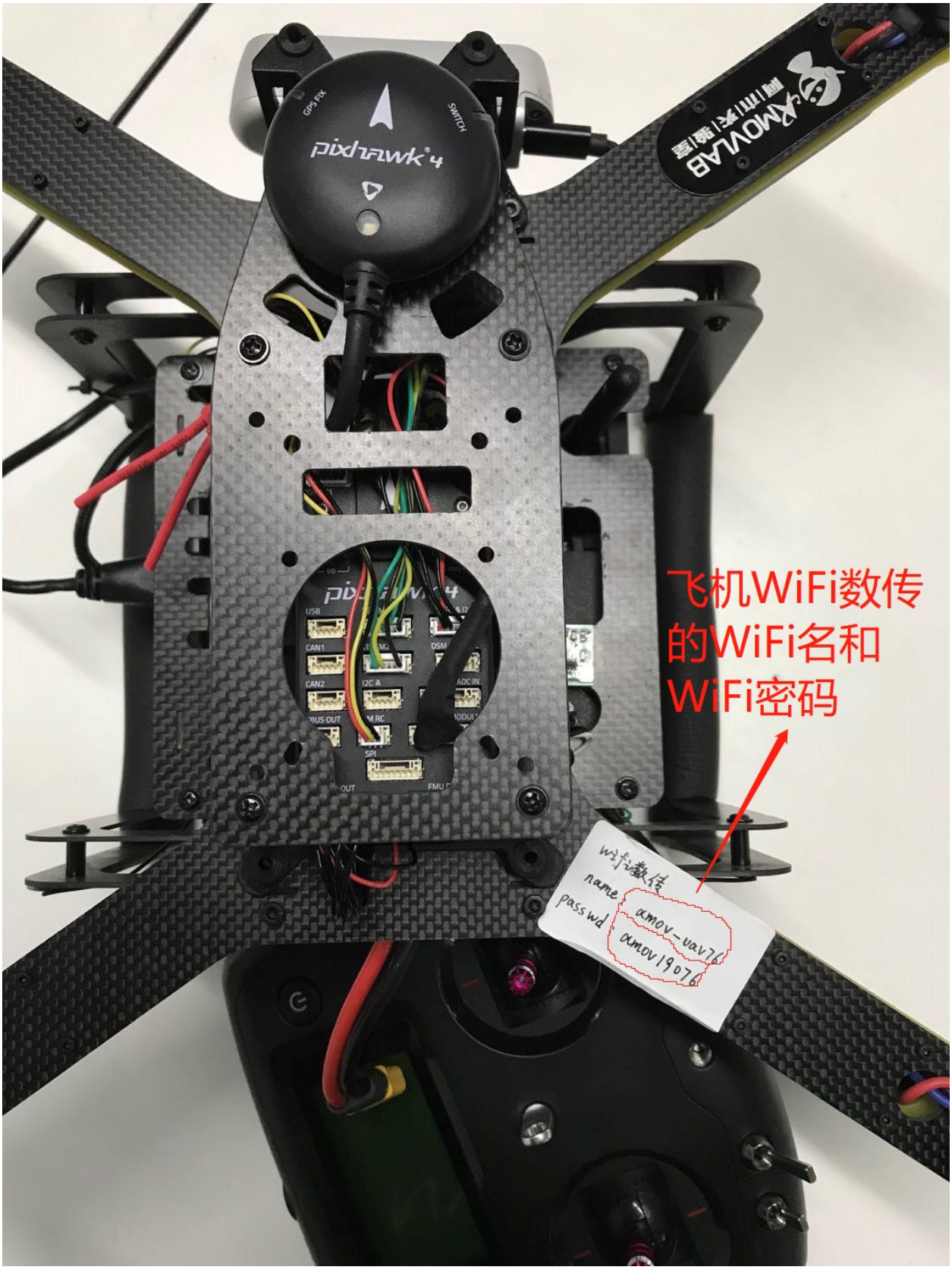
WIFI 数传远程连接 QGroundControl

在 WIFI 数传连接 QGroundControl 的过程中, 本小节文档中会分为两种情况来说明正常连接的过程以及非正常连接的过程.

正常连接

小技巧: 飞机说明: 本教程飞机配置为 TX2+T265+D435I+pixhawk4

飞机整机图:



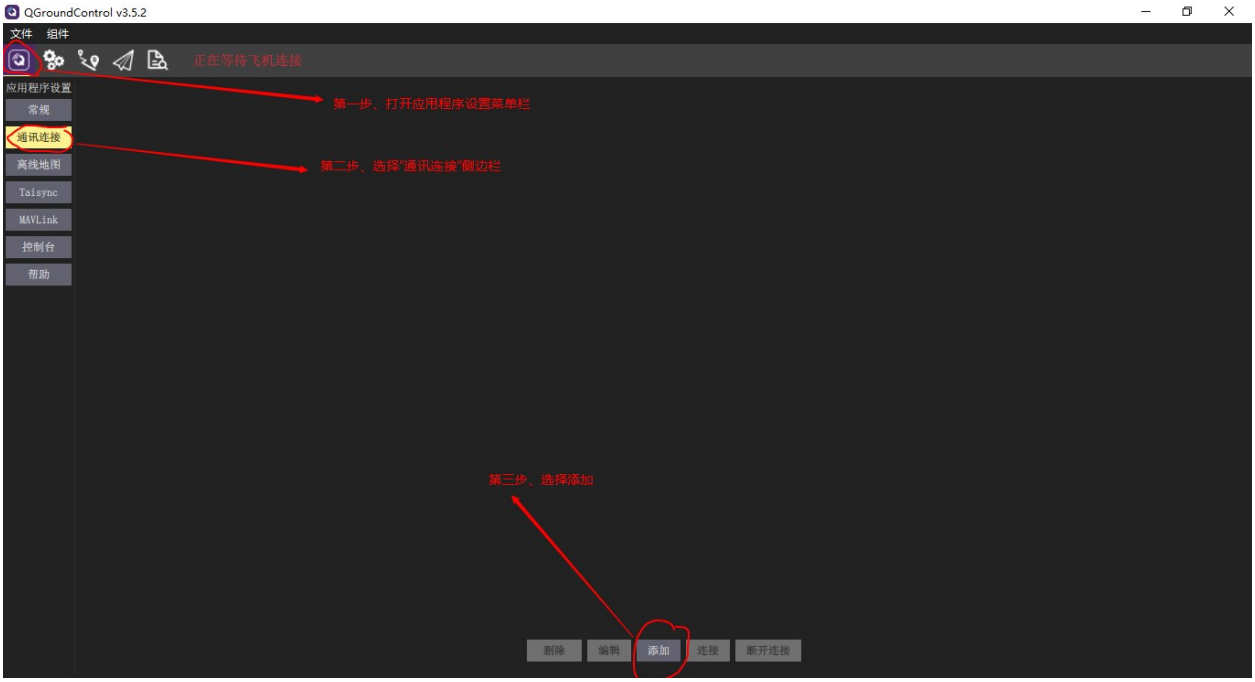
飞机上面贴有标签,WIFI 数传名字为 amov-uav76. 密码为 amov19076.

首先拿到飞机之后, 上电, 打开自己电脑 WIFI, 选择 WIFI 名为 amov-uav76, 并输入密码进行连接.(截图 WIFI 的连接图片)

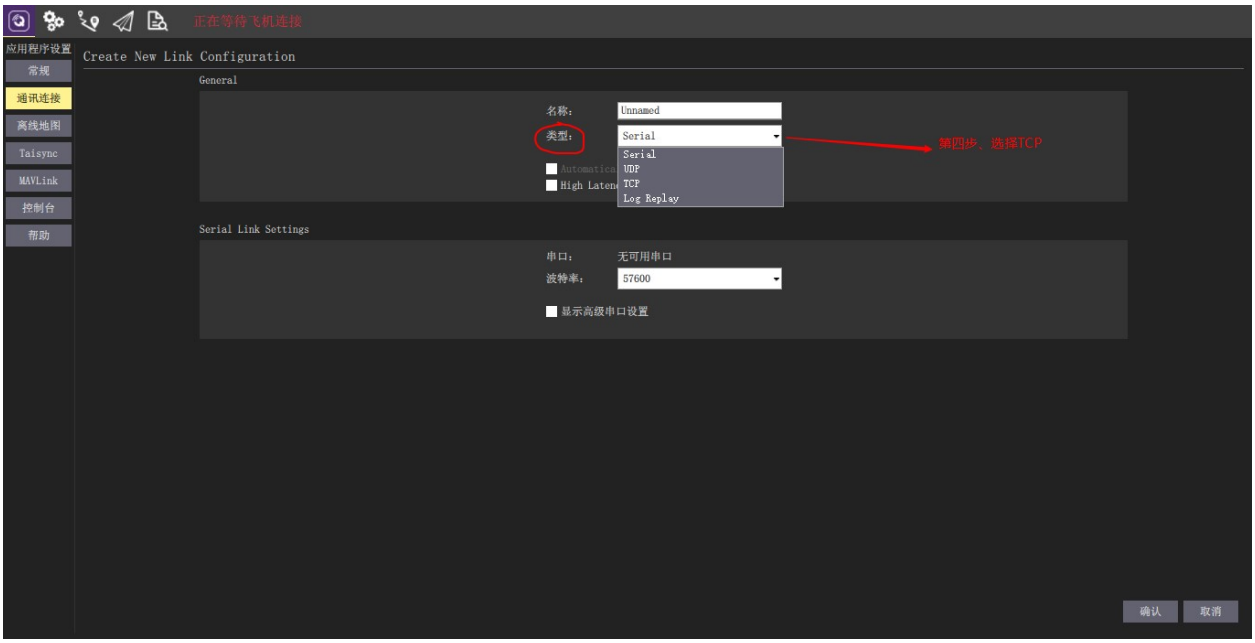
然后打开一个终端,ping 一下 WIFI 数传的 IP,192.168.10.76.(截图为 WIFI 数传配置成功的图片)

发现已经 ping 成功了, 说明我们可以连接到 QGroundControl 地面站上面了.

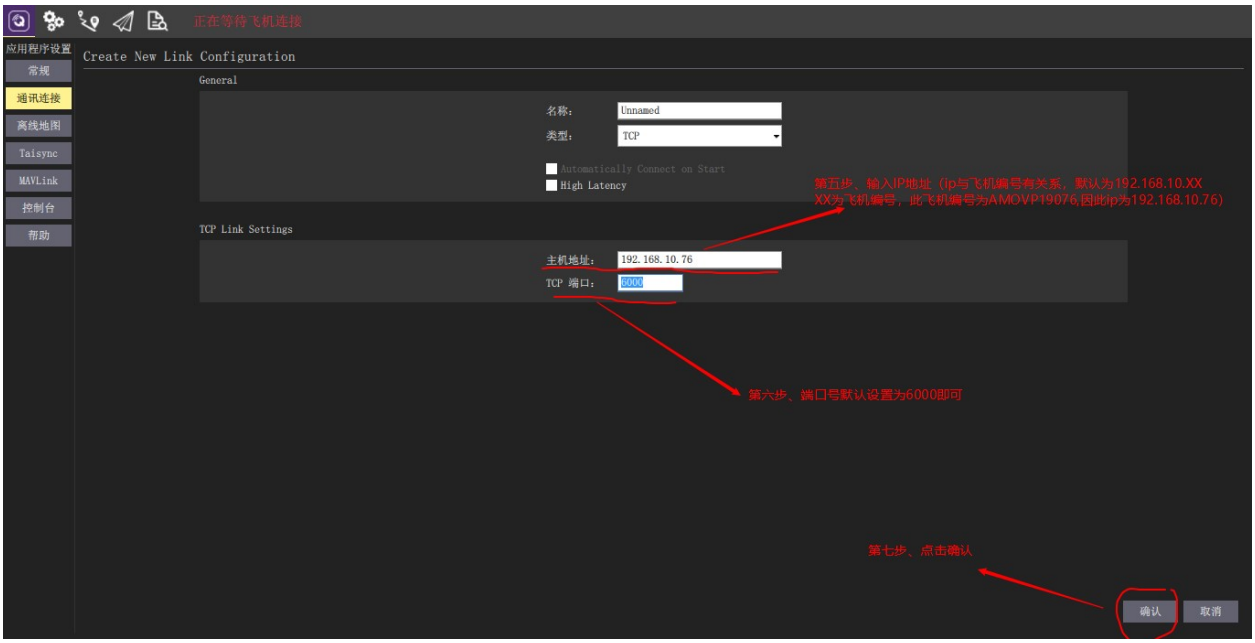
接着, 打开 QGC, 第一步是打开应用程序设置菜单栏; 第二步是选择通讯连接侧边栏; 下面会出现五个可修改功能按钮, 分别为删除, 编辑, 添加, 连接, 断开连接. 第三步是点击添加按钮.(此处图片为 qgc 的连接 1~3)



第四步是在类型旁边将原有的类型 serial 重选为 TCP.(此处图片为第四步)

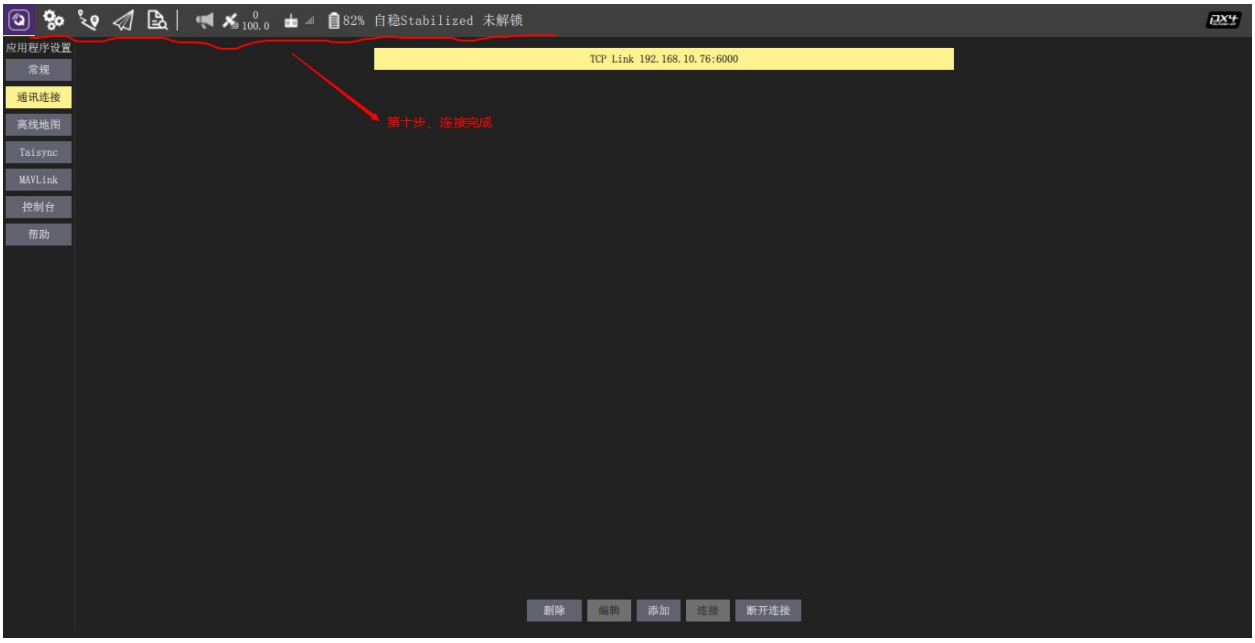


选择 TCP 类型之后, 下面的输入框有所变化, 主机地址就填写为 192.168.10.76; 端口号填写为 6000, 接着点击确认即可.(此处照片 5~7)



第八步是选中刚才添加的通讯连接, 第九步点击下面的连接按钮, 进行连接到 QGC.(此处照片为 8~9)

第十步就可以看到 WIFI 数传连接成功, 地面站上显示飞控的各个信息.(此处照片为 10)



非正常连接

小技巧: 飞机说明: 本教程飞机配置为 Nano+ 双目 T265+A1+pixhawk4

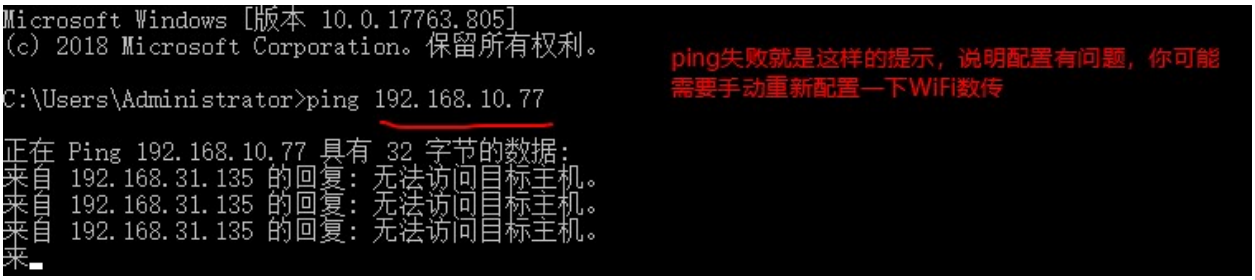
飞机整体图:

飞机上面有标签, 写的是有关 WIFI 数传的默认配置. 例如本架飞机, 我们出厂默认的 WIFI 数传配置为 AP

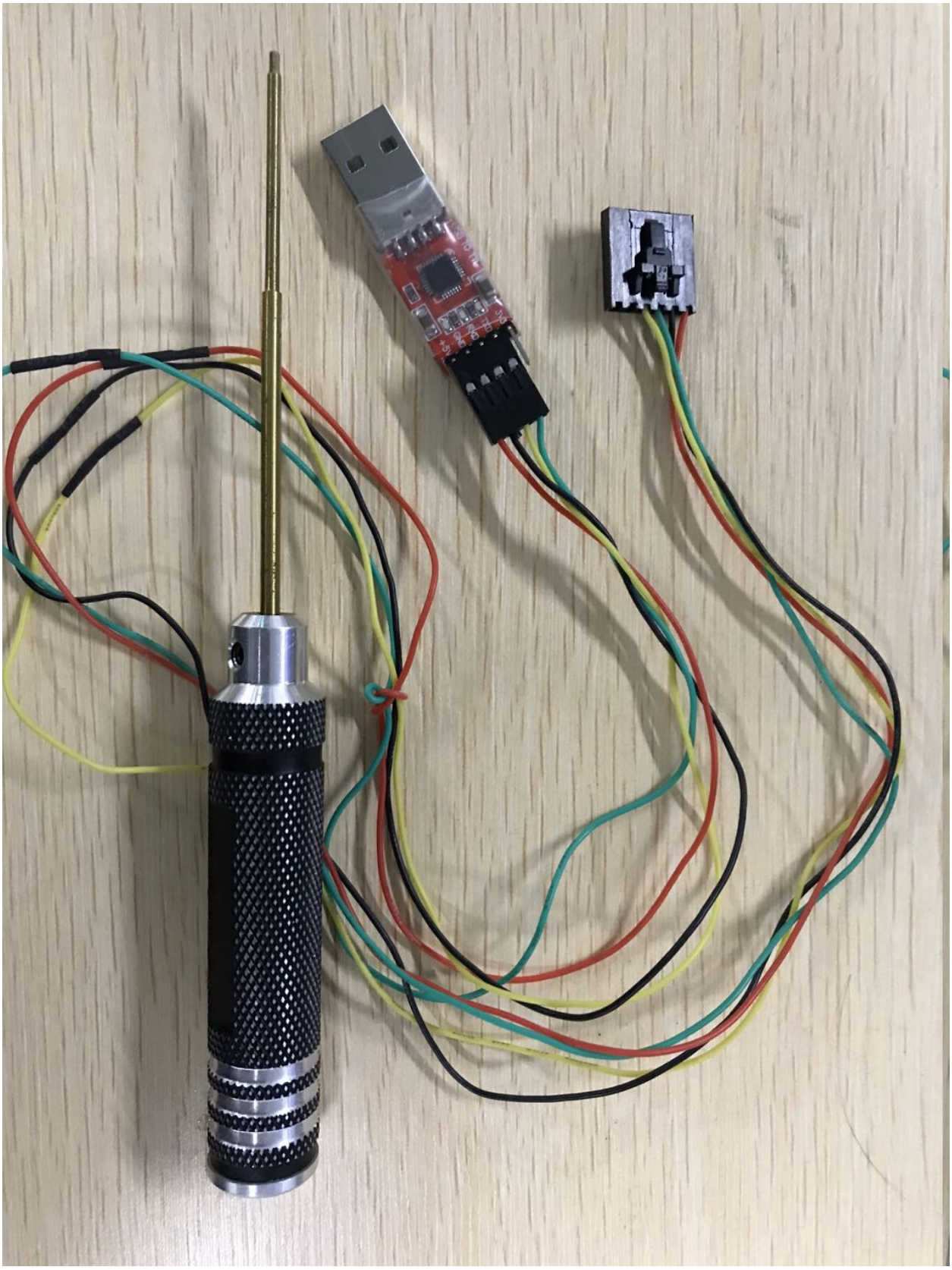
模式 (数传是个热点, 地面站连接热点).

上电之后热点名称就为:amov-uav77, 热点密码为:amov19077, 电脑连接上 WIFI

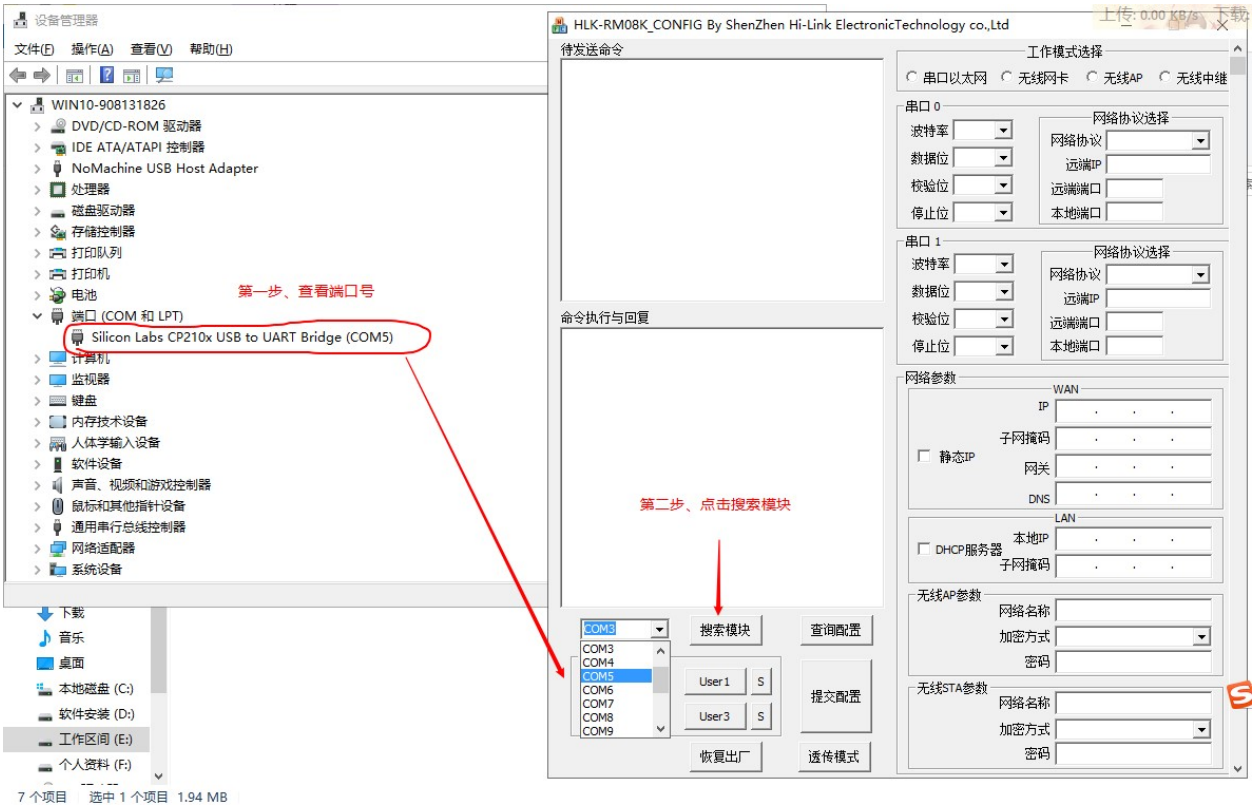
这时候你就可以在你连接热点的电脑打开终端 (Windows 下面 Ctrl+r, 输入 cmd 进入终端,Ubuntu 下面直接右击鼠标打开 termial), 使用 ping 命令看是否能 ping 通热点. 热点的 IP 为 192.168.10.77. 直接输入 ping 192.168.10.77. 如果 ping 通, 说明就正常可以使用了. 如果 ping 不通, 那就是 WIFI 数传没有配置正确. 可以看到是 ping 失败了,WIFI 没有配置成功, 你需要自己重新手动配置一下



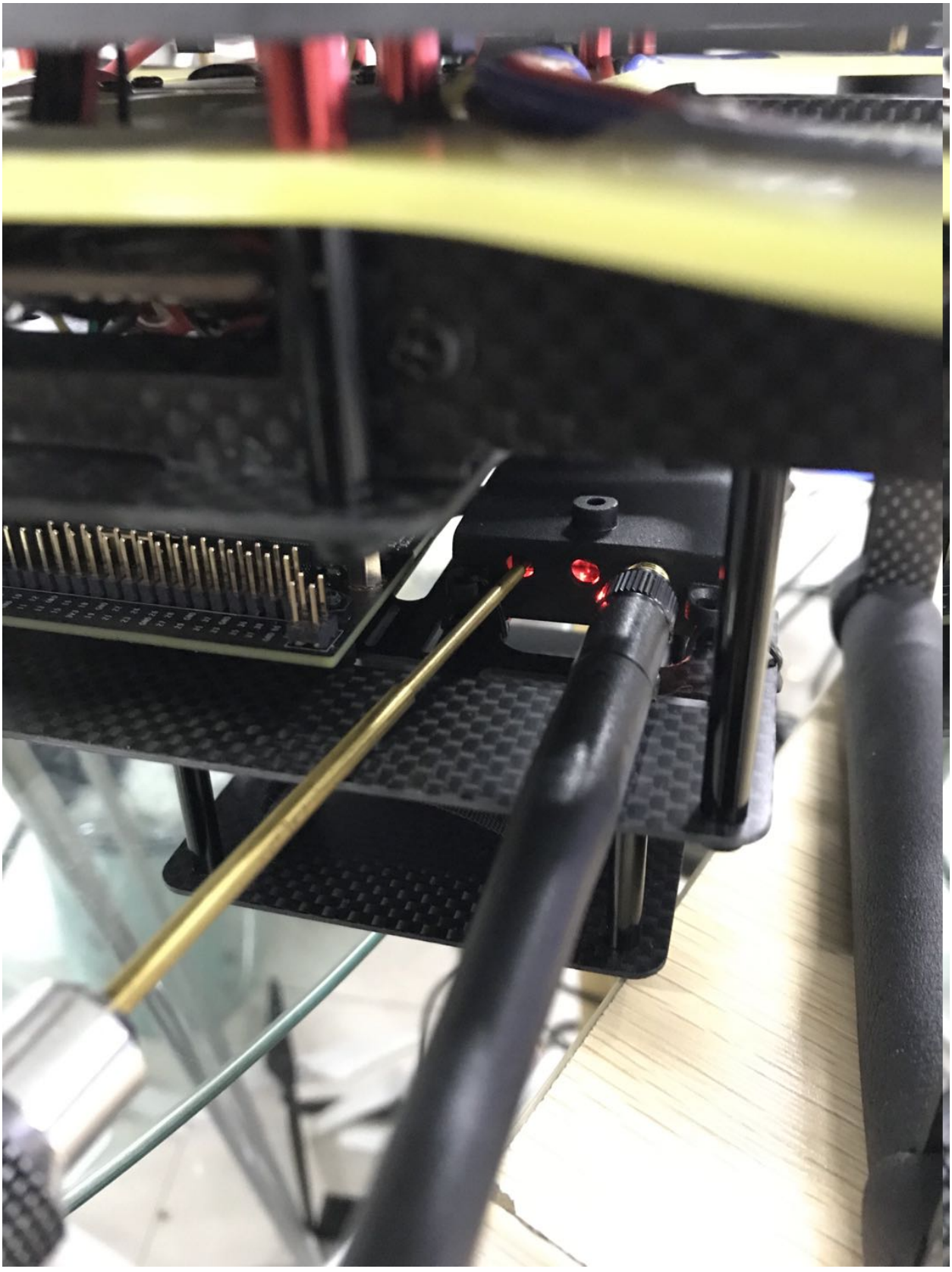
首先需要准备的东西有, 配置软件 RM08K config tool CN V2.1, 和 WIFI 数传配置线以及工具刀 (有的 WiFi 数传的复位按钮地方在里面, 可能需要镊子或者较细的工具)



将 WiFi 数传配置线接上 WIFI 数传和电脑上，然后在电脑设备管理器查看识别到的串口号是多少，再打开配置软件，选择相应的串口号并点击搜索模块. 点击搜索模块之后发现没响应，别急，继续看第三步.

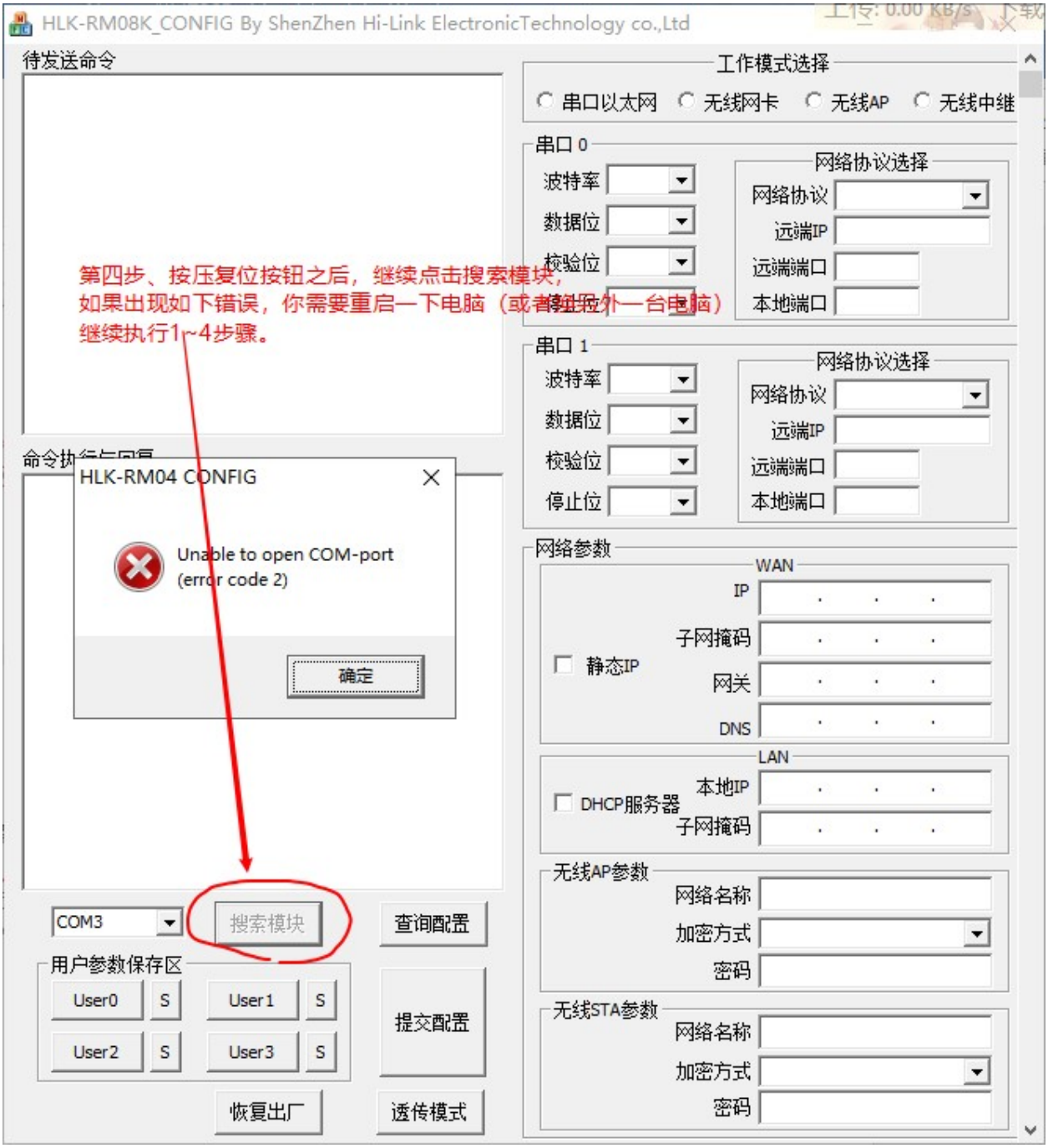


在第三步中就要使用工具刀了，第二步执行点击搜索模块之后，使用工具刀按下 WiFi 数传复位按钮，复位按钮的位置在远离 WiFi 数传信号线的那个孔里面，如下图所示.(有的 WIFI 数传外壳遮挡住了复位按钮，根据实际情况选择适当的工具)

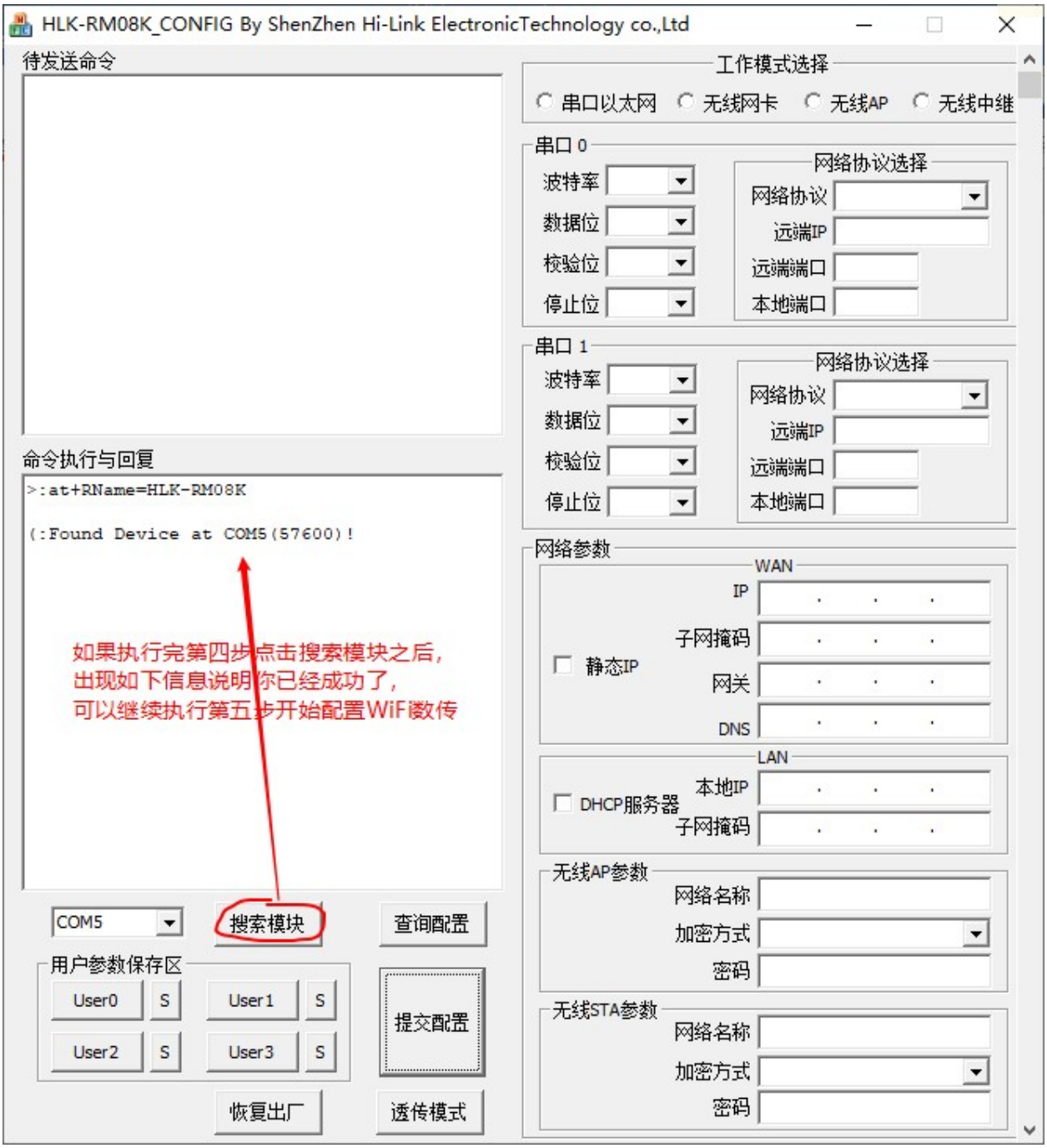


进行第四步, 复位按钮按下之后, 继续点击搜索模块. 这时候会出现两种情况, 一种是下图错误提示无法打开

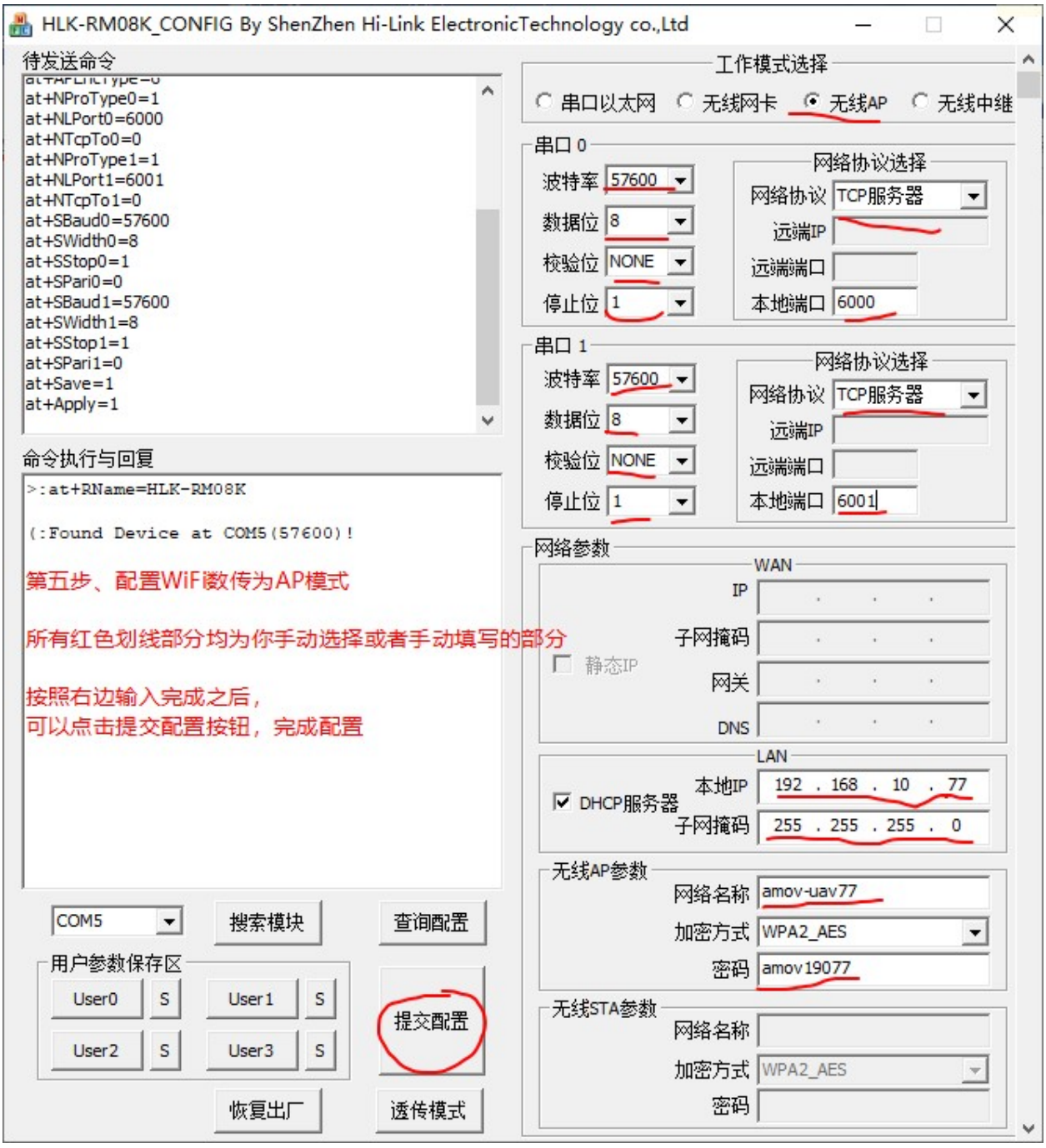
com 口, 解决办法是重启电脑重新执行第一步到第三步.



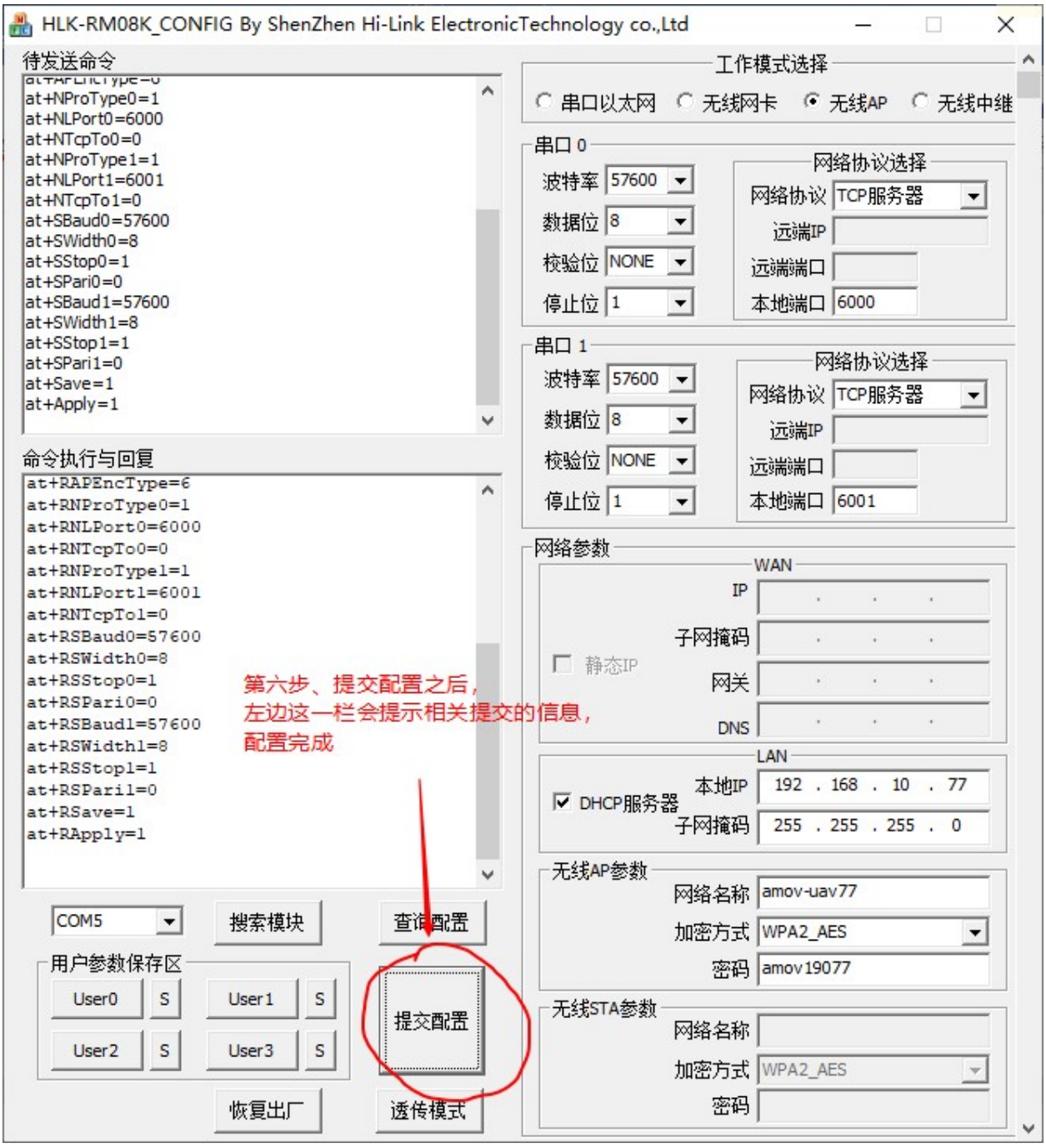
另外一种情况是, 点击搜索模块之后, 出现如图所示找到设备 com 口就说明正确打开了串口, 可以继续下一步配置了.



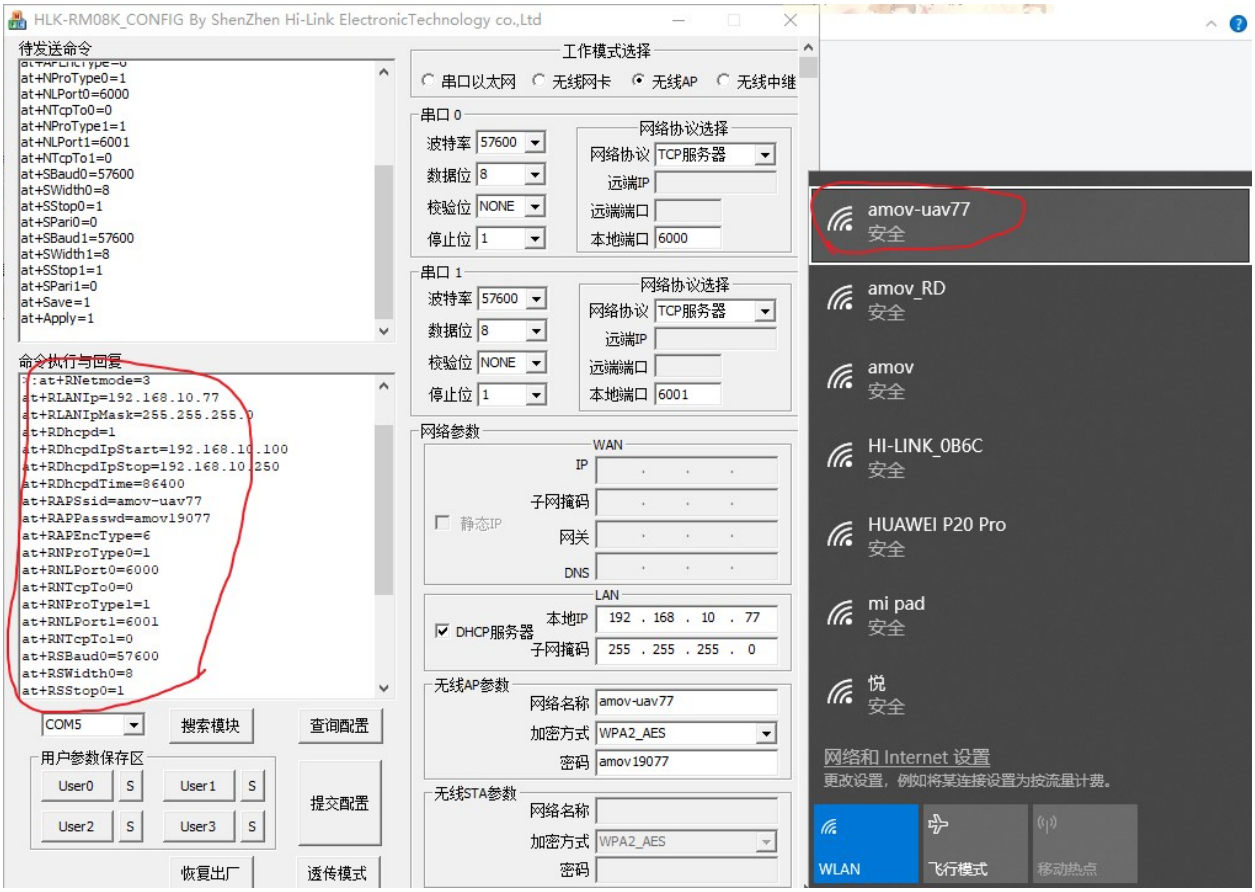
第五步是配置 WIFI 数传为 AP 模式, 如图所示, 所有划红线部分为你手动选择或者手动添加内容, 所有配置按照图片右边的具体配置一致即可, 最后选择提交配置按钮.



第六步, 提交配置之后, 左边这一栏会显示刚才提交配置的信息, 配置就算完成结束, 别着急拔掉配置线, 我们先测试一下配置的成功不成功.



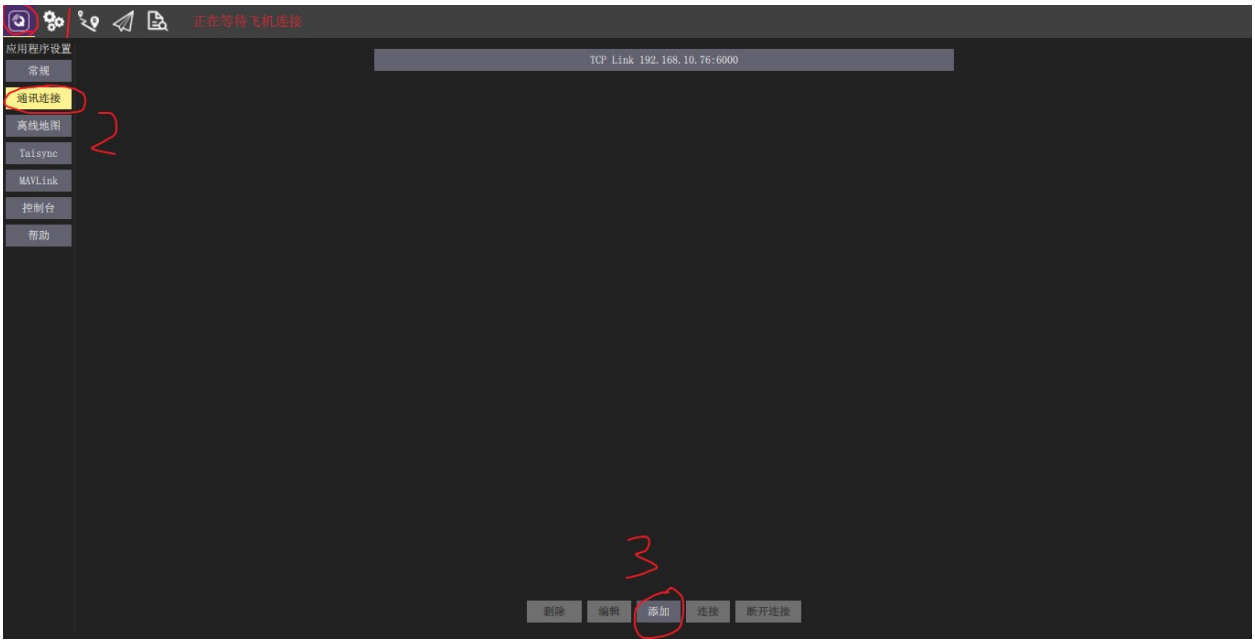
继续连接 WIFI, 如下图所示



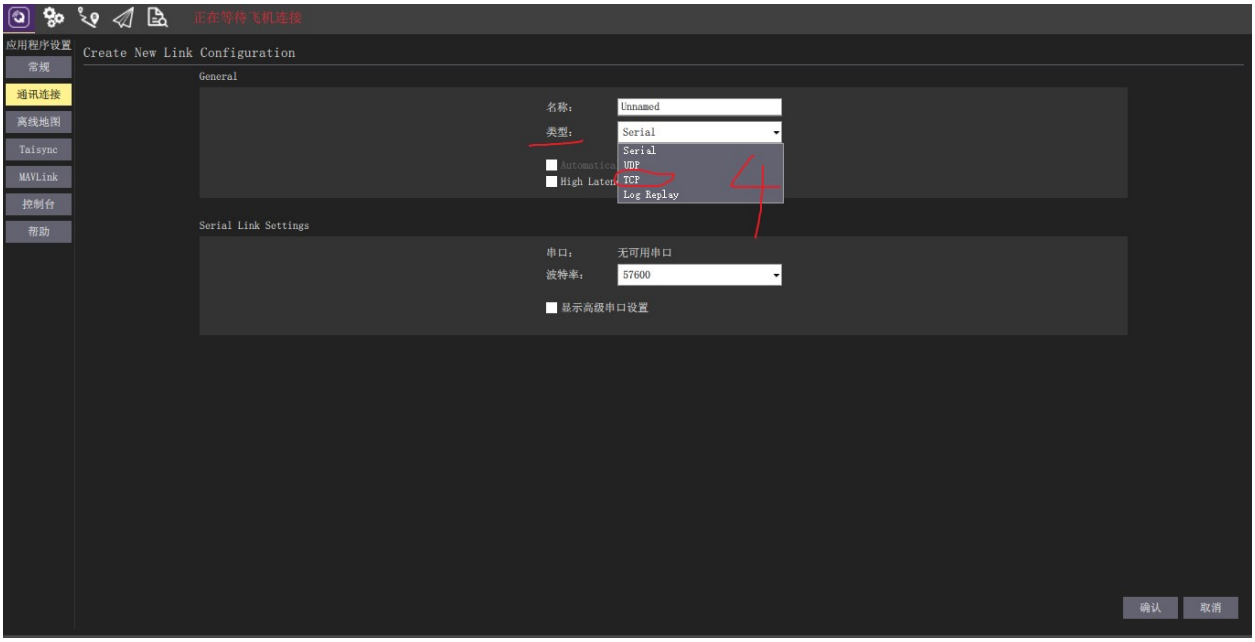
然后打开终端,ping 一下 WIFI 数传的 IP, 直接执行 ping 192.168.10.77

发现 ping 通了之后, 说明刚才配置就算完成, 拔掉配置线, 接到飞控上面. 准备连接地面站 QGroundControl.

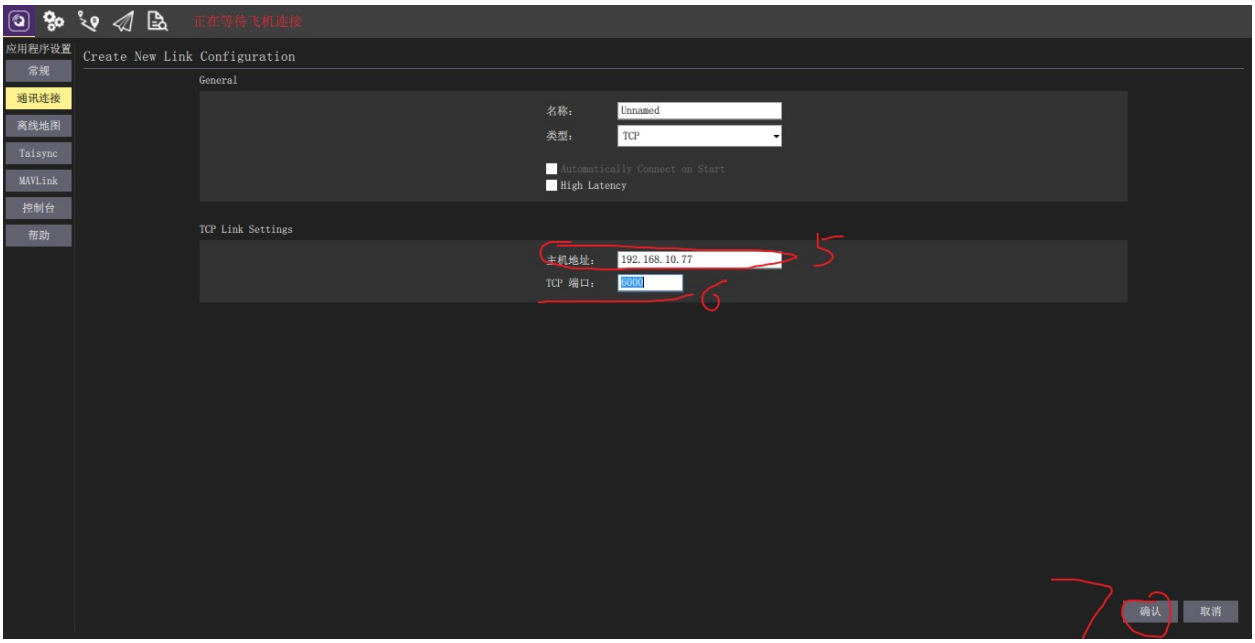
打开 QGC, 选择应用程序设置菜单栏, 继续选择通讯连接子项. 下面有五种操作, 分别为删除, 编辑, 添加, 连接以及断开连接.



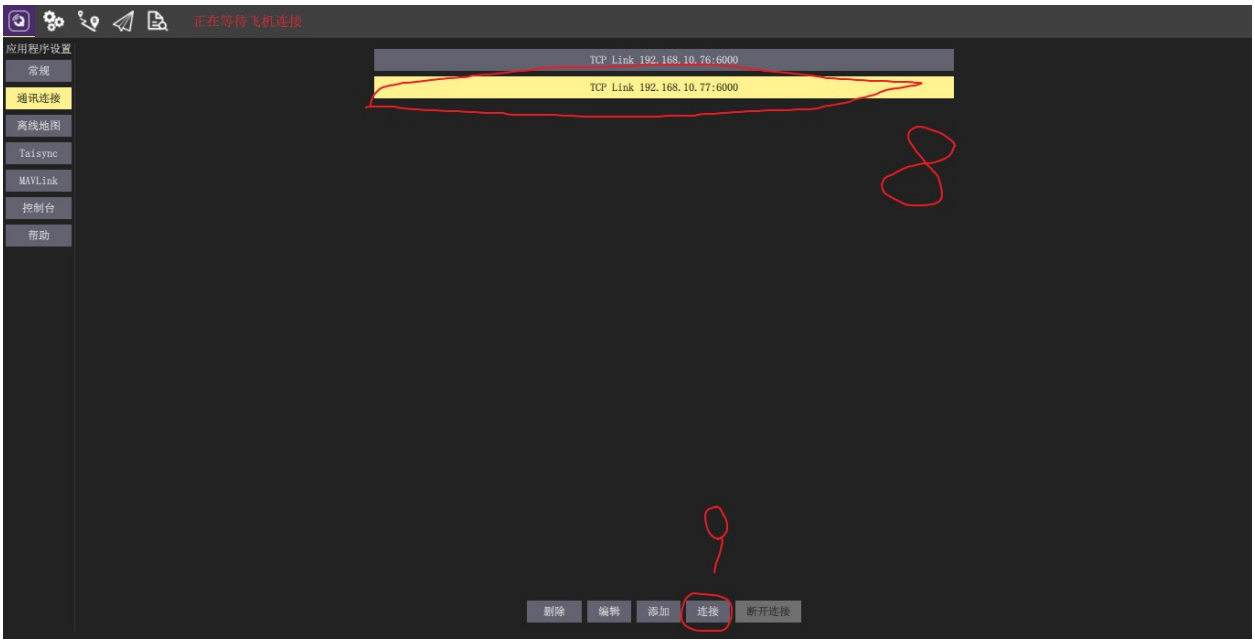
接下来, 选择添加项, 将类型选择由默认的 serial(串口) 改为 TCP, 上面的名称可改可不改.



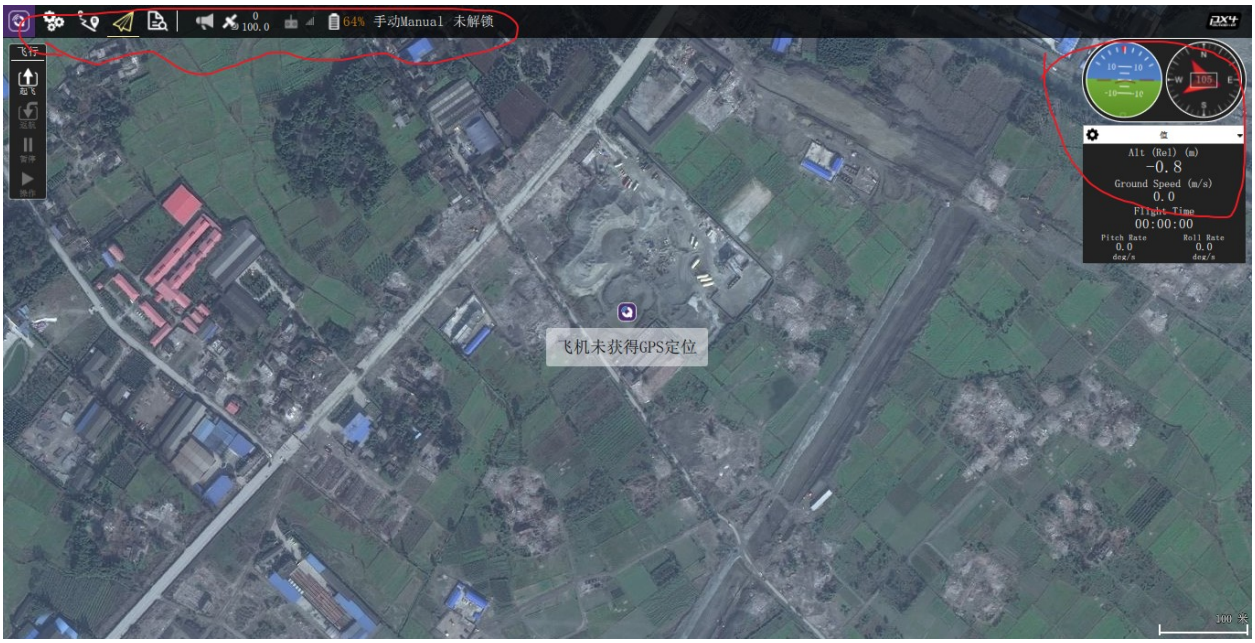
下面的主机地址填写 192.168.10.77(我们刚才 ping 通的地址, 也就是 WIFI 数传配置好的 IP 地址), 端口号填写为 6000(注意和 nomachine 的默认端口号 4000 区分开). 接着, 右下角点击确认即可.



然后, 选择刚添加的通讯连接, 点击第四个连接操作连接到地面站.



最后可以看到 QGC 可以显示飞控相关姿态信息了. 完成 WiFi 数传连接 QGroundControl 的过程.



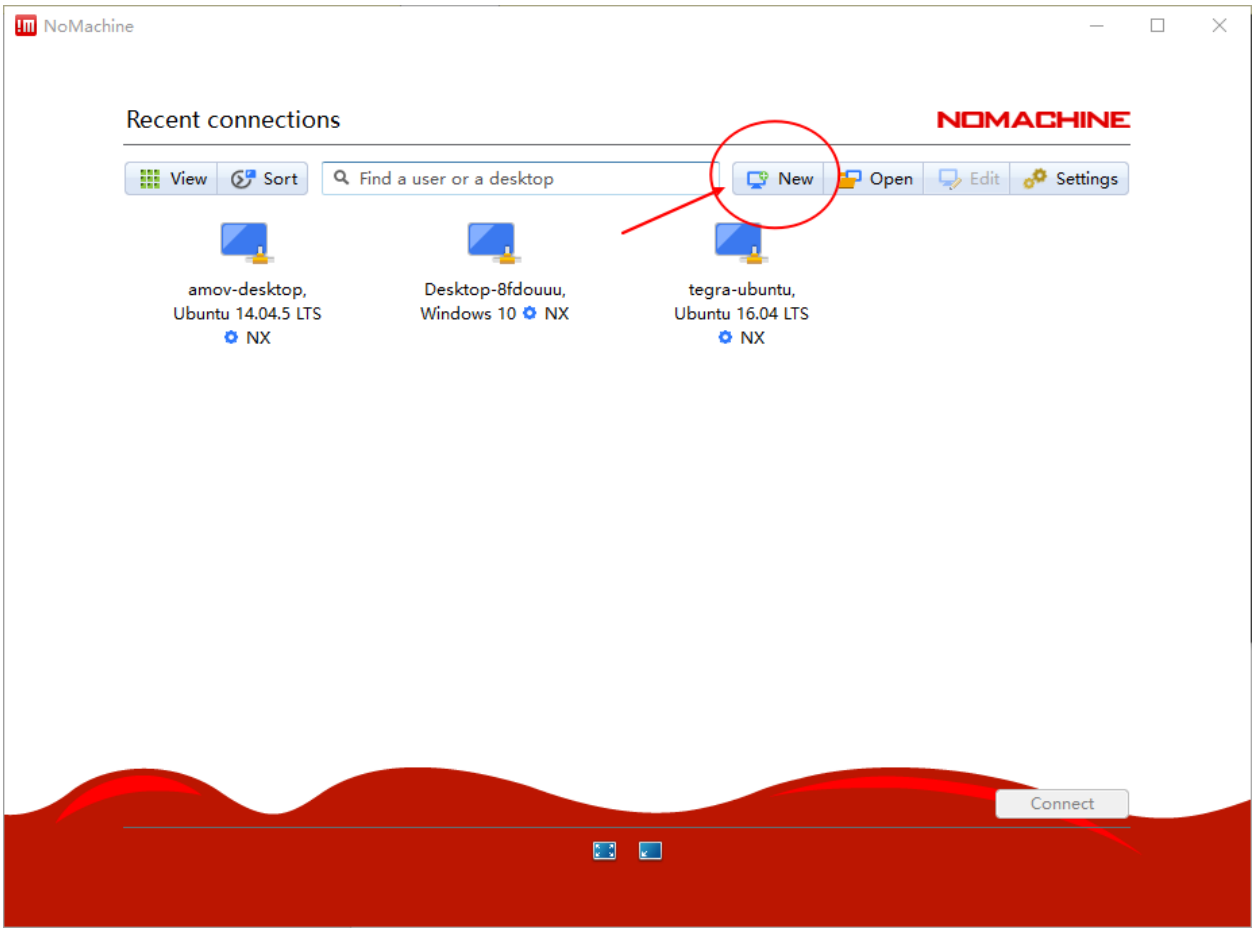
远程连接 TX2

需要的硬件:

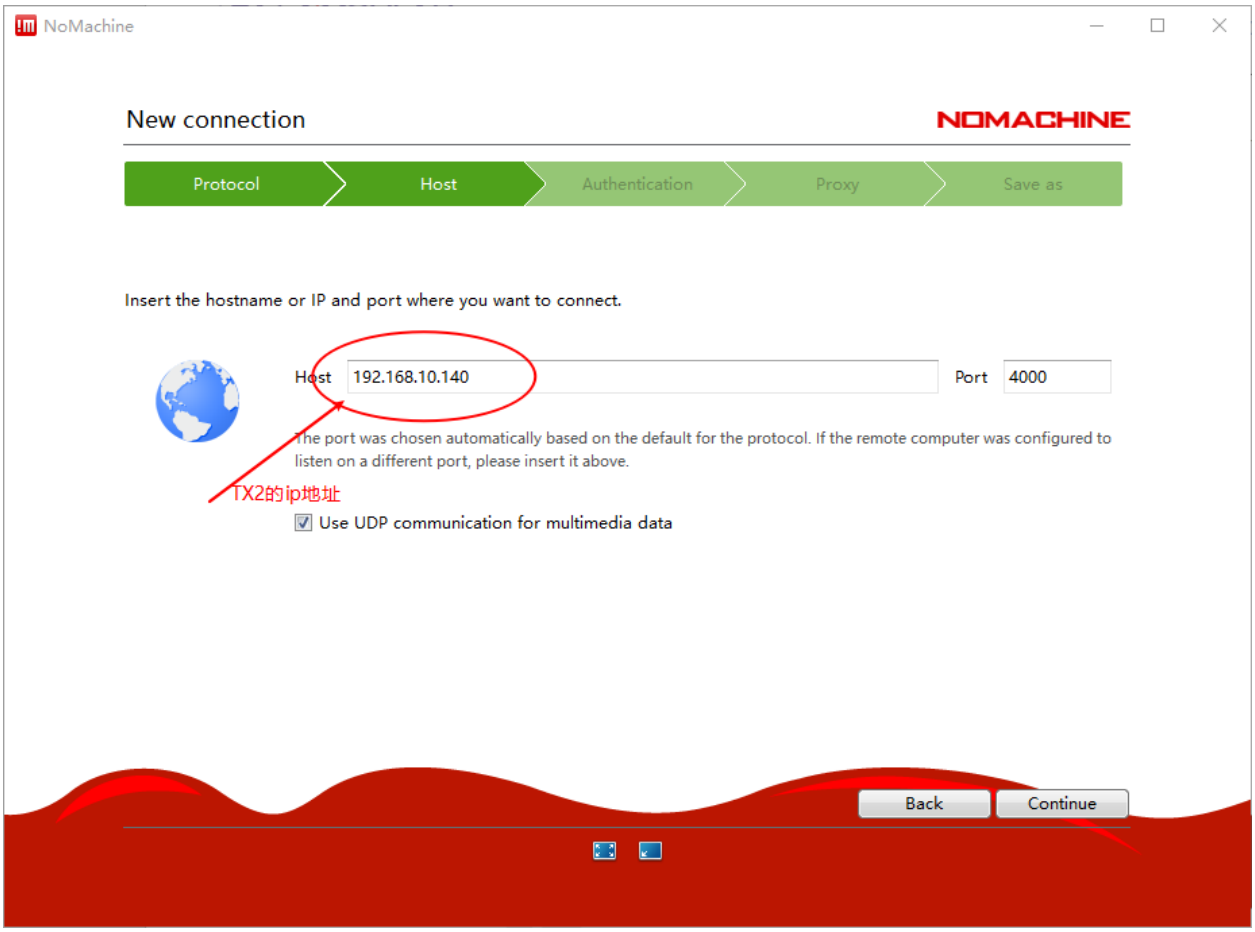
- 1. 有 HDMI 接口的显示器一台 (用于显示 TX2，TX2 连接你准备好的路由器的 WiFi)
- 2. 带宽比较好的路由器一台
- 3. 安装有 QGC 地面站与 NoMachine 电脑一台或两台

NoMachine 配置: 此步骤需要英伟达 TX2 中的 Ubuntu 系统连接成功 WiFi 路由器，并在终端上输入 ifconfig 命令找到 TX2 的 IP（我的 TX2 IP 为 192.168.10.140）。

打开 NOMACHINA 软件，点击 New 新建

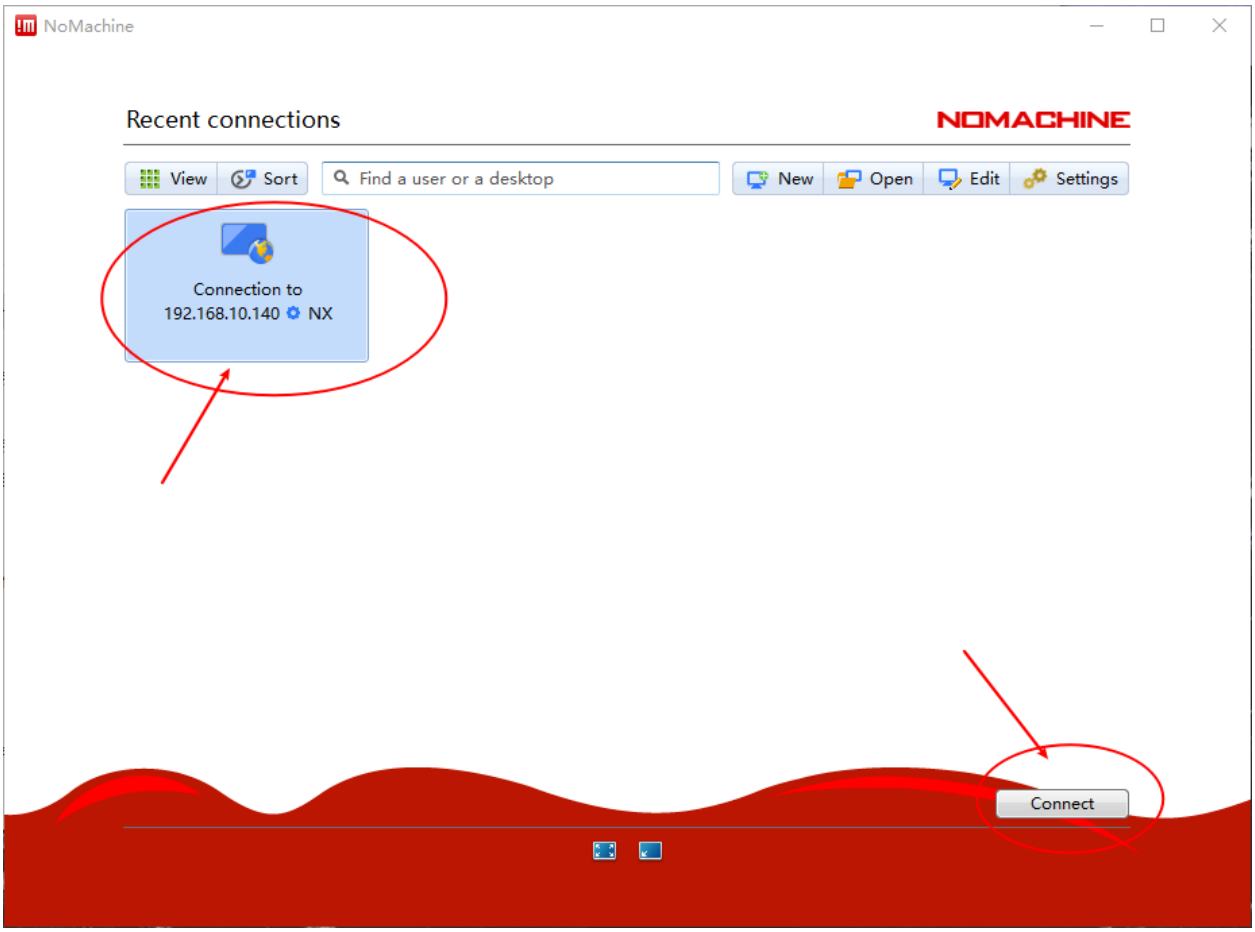


Protrocol 选择 NX 模式，然后输入 TX2 的 IP 地址 192.168.10.140，Port 默认 4000



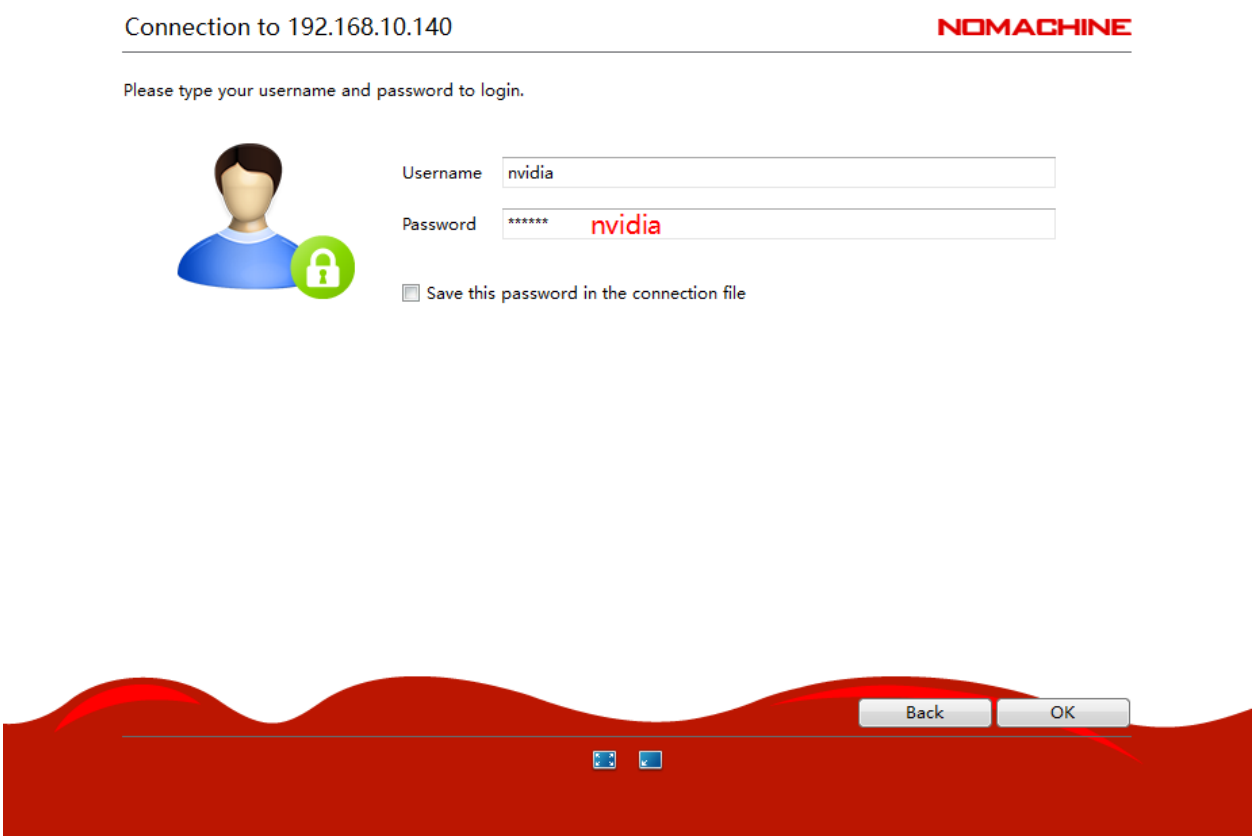
下一步，鉴定方式选择 Password，下一步选择 Do not use proxy，最后 Done

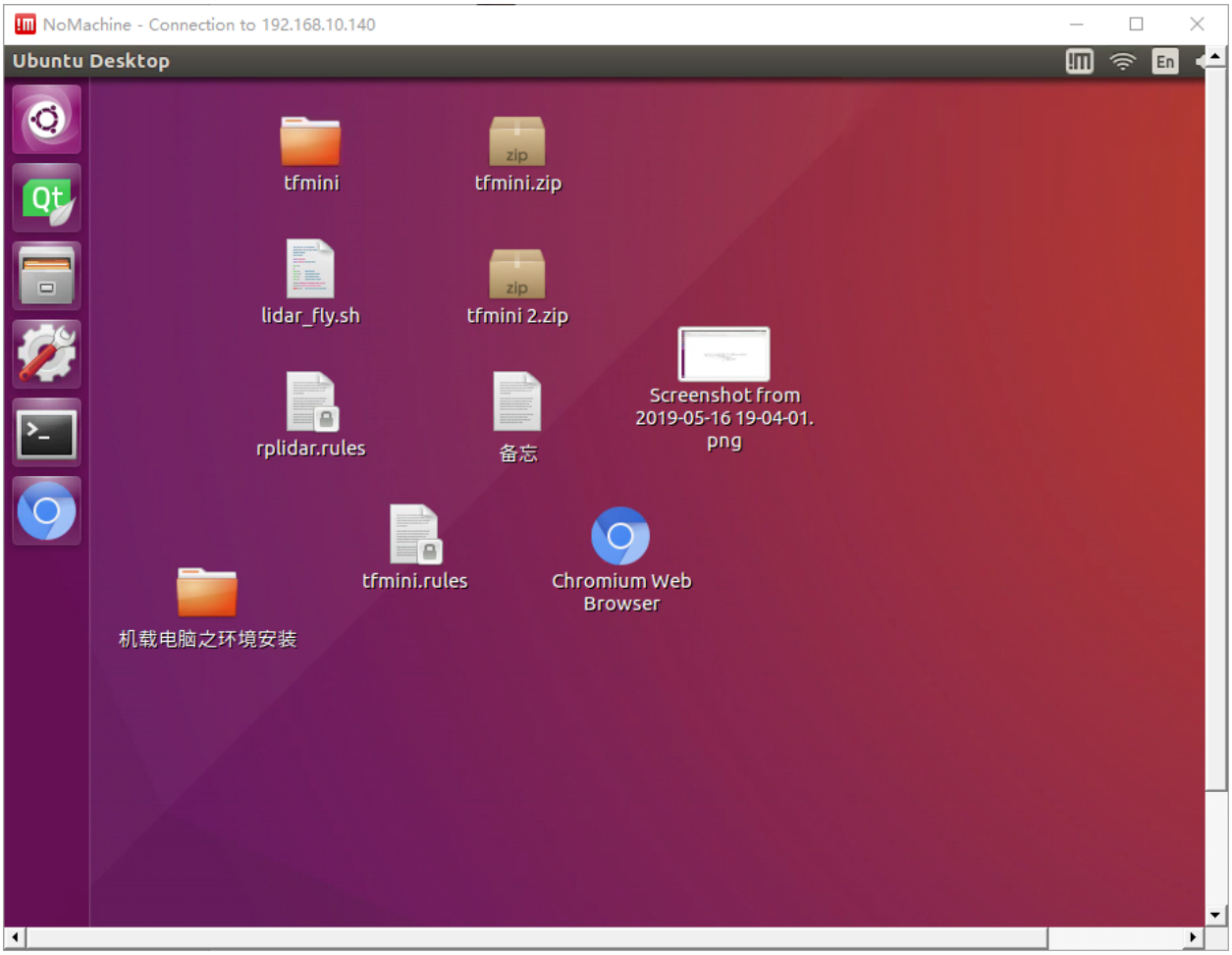
如下选择刚才配置的参数，点击连接



成功后点击 YES

输入 TX2 的用户名以及密码然后一路 OK 即可进行访问。


Connection to 192.168.10.140
NOMACHINE
Please type your username and password to login.
Username
nvidia
Password
******
nvidia
Save this password in the connection file
Back
OK

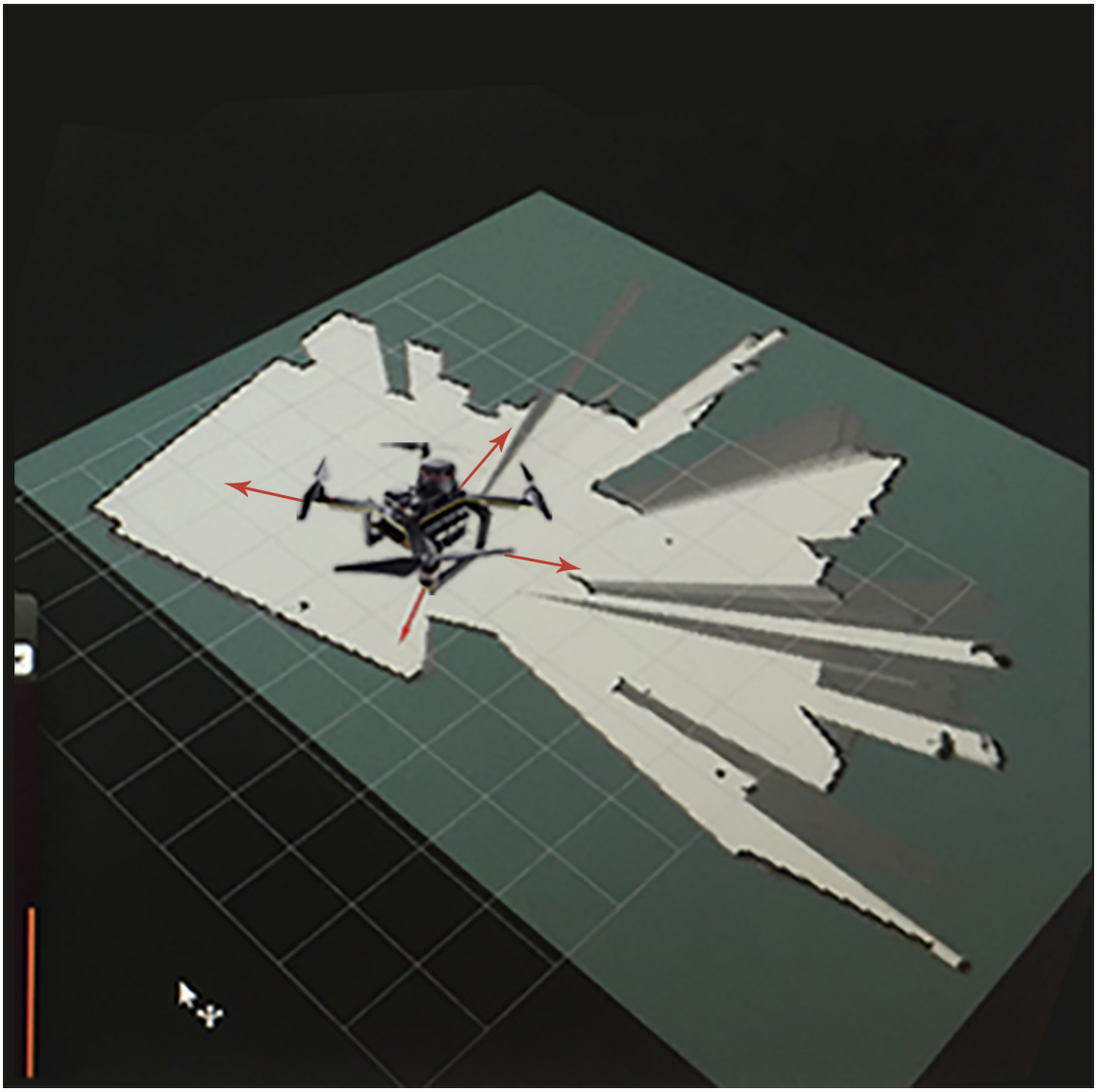


飞行环境说明

室内环境

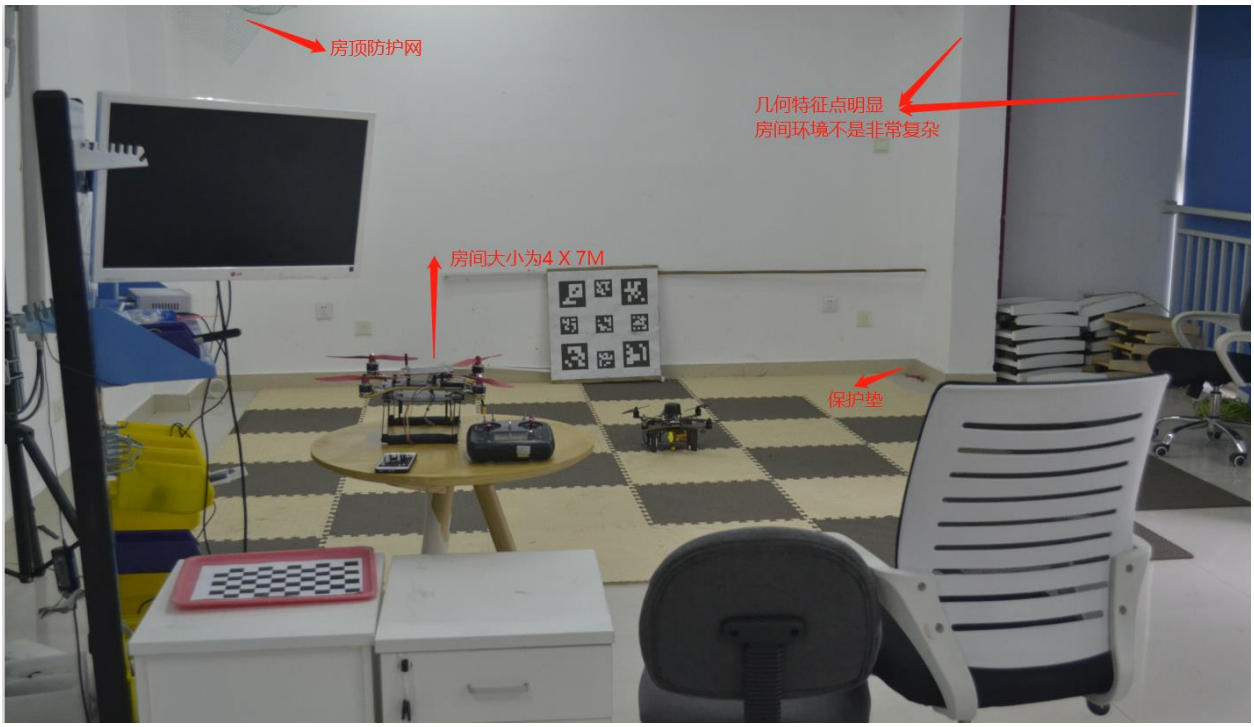
激光雷达

室内我们采用的激光雷达进行定位，因为采用的二维雷达，会在室内构建出如下的平面地图



所以在测试以前要选择一个房间大小合适 (A1/A2 雷达半径 12 米有效，A3 雷达半径 25 米有效)，房间不能地形非常复杂，也不能没有角点和几何特征点，否则不能很好的构建出地图，就不能很好的定位。

我们的测试环境如下：



房间内有一些防护装置，房顶的防护网，地面的保护垫等等最好具备，防止伤人和损坏飞机。

视觉 SLAM

视觉的室内环境也是如上图所示环境, 在室内可以添加一些桌椅板凳放置视觉前方, 以便能有较好的特征点.

室外环境

视觉 SLAM

室外视觉的环境, 可以选择在马路旁边的树下面, 视觉所能覆盖的视野包括树, 马路, 以及楼区, 这样特征点多一点, 效果能好. 在视觉 slam 中就尽可能不要选择空旷, 单一特征点的地方, 如操场. 我们实验室经常测试视觉的环境, 有马路, 有草地, 有房屋, 如果视野只有单纯的草地, 视觉定位就不是很好. 容易漂移.

GPS/RTK

室外使用 GPS/RTK 时, 应当选择较为空旷地方, 不要在房屋旁边或者树的旁边, 搜星信号会很差. 推荐在公园绿地或者操场中.

5. 自主飞行说明

自主飞行可分为室内和室外两种模式, 相应的室内分为激光 SLAM 飞行模式和视觉 SLAM 飞行模式, 室外分为视觉 SLAM 飞行模式和 GPS/RTK 飞行模式. 每一种飞行模式又对应分为 TX2 板载计算机和 Nano 板载计算机. 在每一种飞行模式中, 你就可以进行室内外相应的实现定点飞行, 降落等.

小技巧: 每一种飞行模式的位置来源有所不同, 这里在 QGroundControl 地面站中有两个非常重要的参数, 该参数决定了使用不同的飞行模式需要选择不同的位置数据来源 EKF2_AID_MASK 和 EKF2_HGT_MODE, 前者是位置数据来源参数. 后者是高度数据来源参数.

室内飞行

激光 SLAM

室内激光雷达定位模式

- EKF2_AID_MASK = 24 （选择位置以及偏航来源）; EKF2_HGT_MODE = Vision/Range sensor （选择高度来源）

TX2

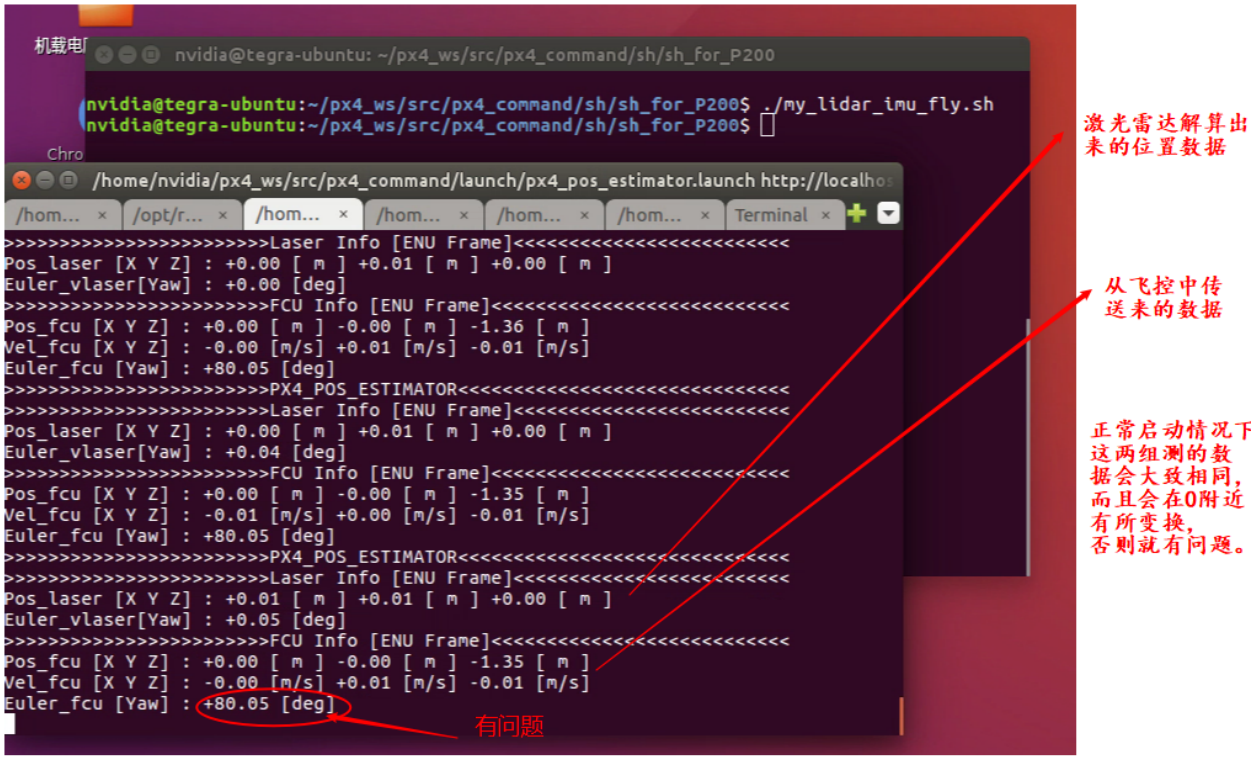
参数确定后尝试解锁，遇到解锁不成功，看看提示是否需要重新校准磁罗盘？解锁成功，电机启转，这时切记不要上桨试飞，因为此时激光雷达没有工作，没有给飞控送入位置以及高度信息，所以此时姿态控制失效，飞机一飞就会偏。这是我们固件的 bug，近期会修复，请注意。

在 TX2 板载计算机系统里打开终端进入如下图所示目录并启动脚本:

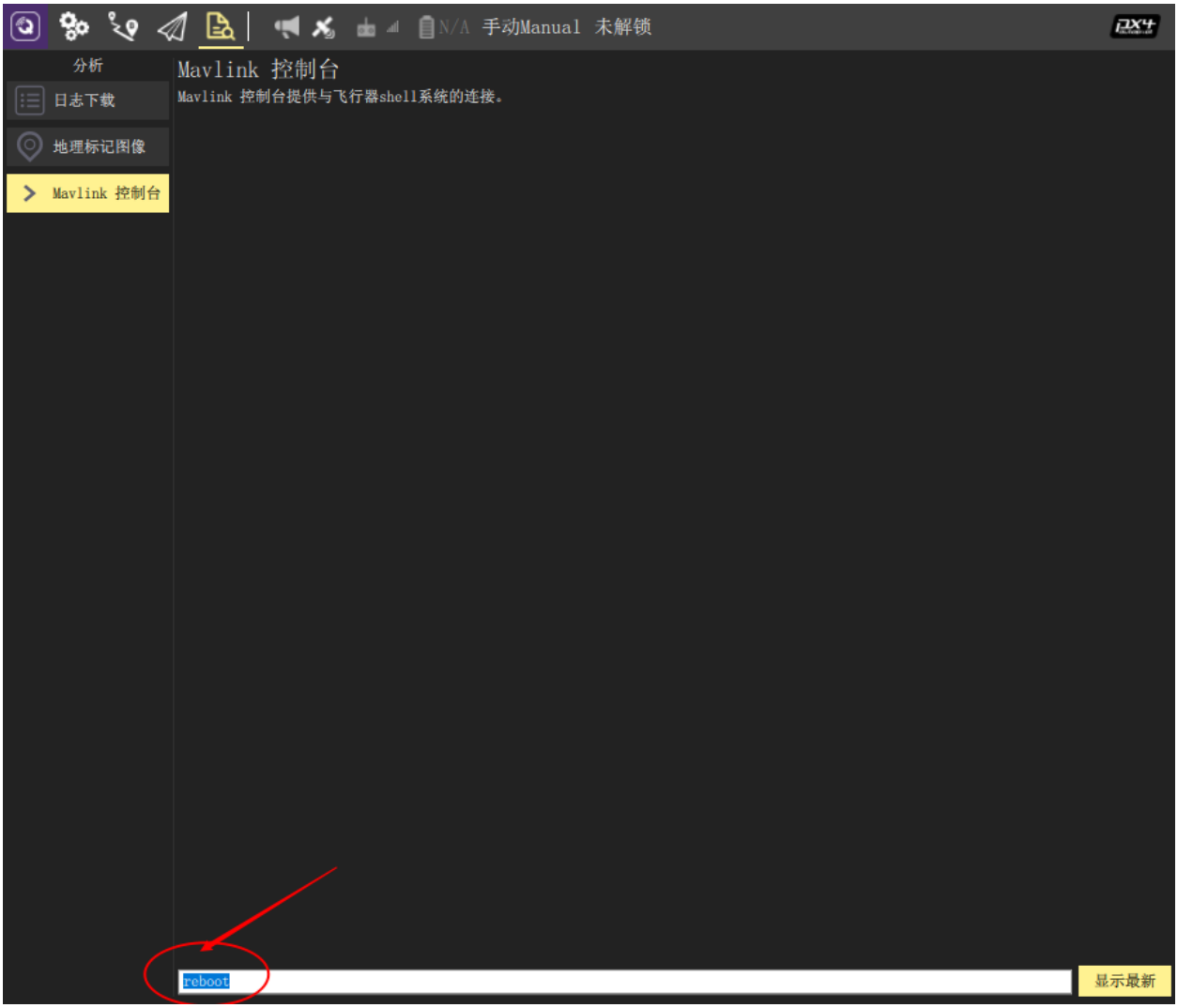
```
./my_lidar_imu_fly.sh
```

脚本会依次启动各个 launch 文件

正常情况下各个窗口会正常工作，其中有个 **px4_pos_estimate** 节点要特别注意，如下图。



当出现上述问题，需重新启动飞控以及脚本，飞控的重启需要在地面站的控制台上，输入 **reboot** 然后回车即可。



NANO

与 TX2 类似, 找其相应的启动脚本即可.

视觉 SLAM

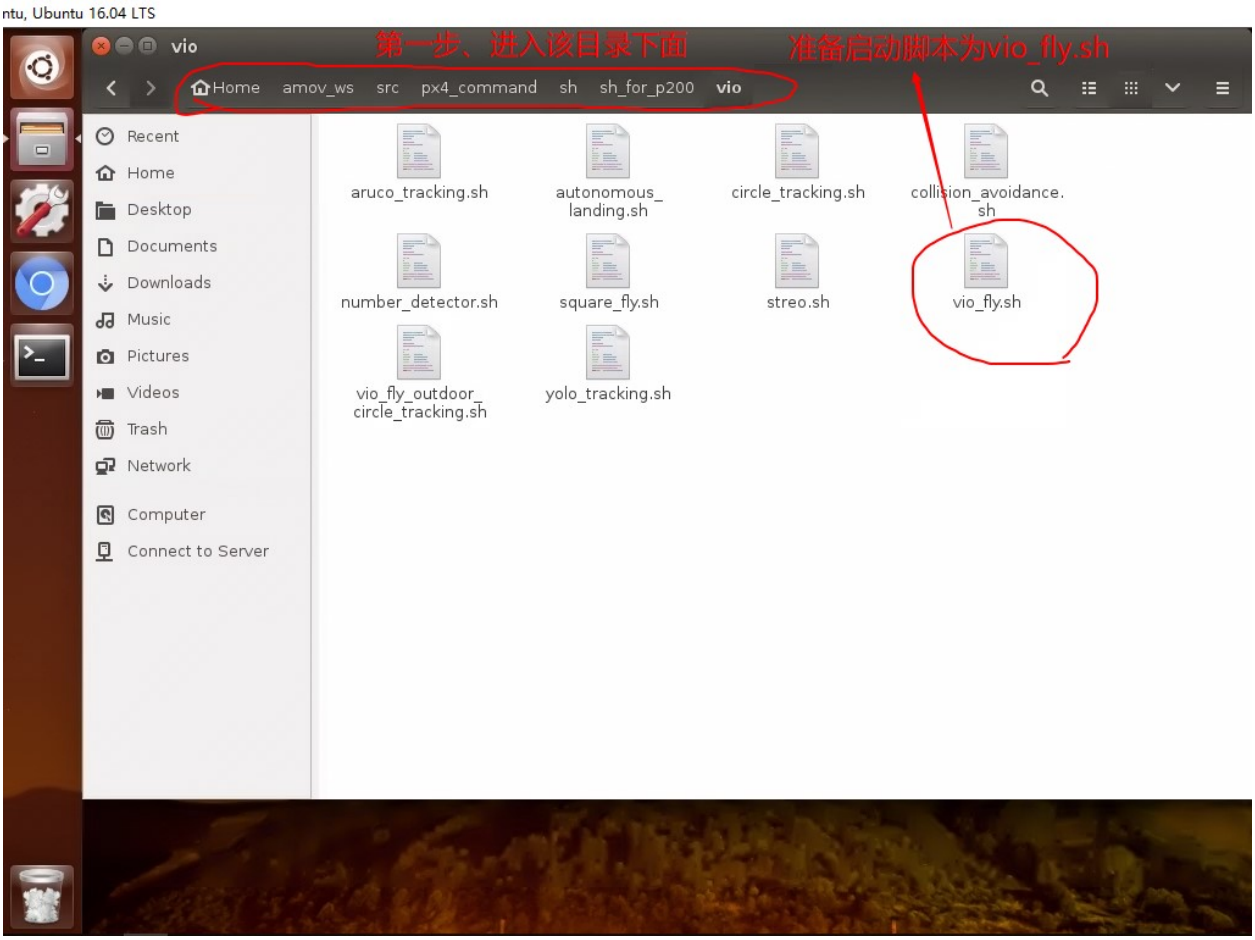
室内视觉 SLAM 定位模式

- EKF2_AID_MASK = 24 （选择位置以及偏航来源）; EKF2_HGT_MODE = Vision/Range sensor （选择高度来源）

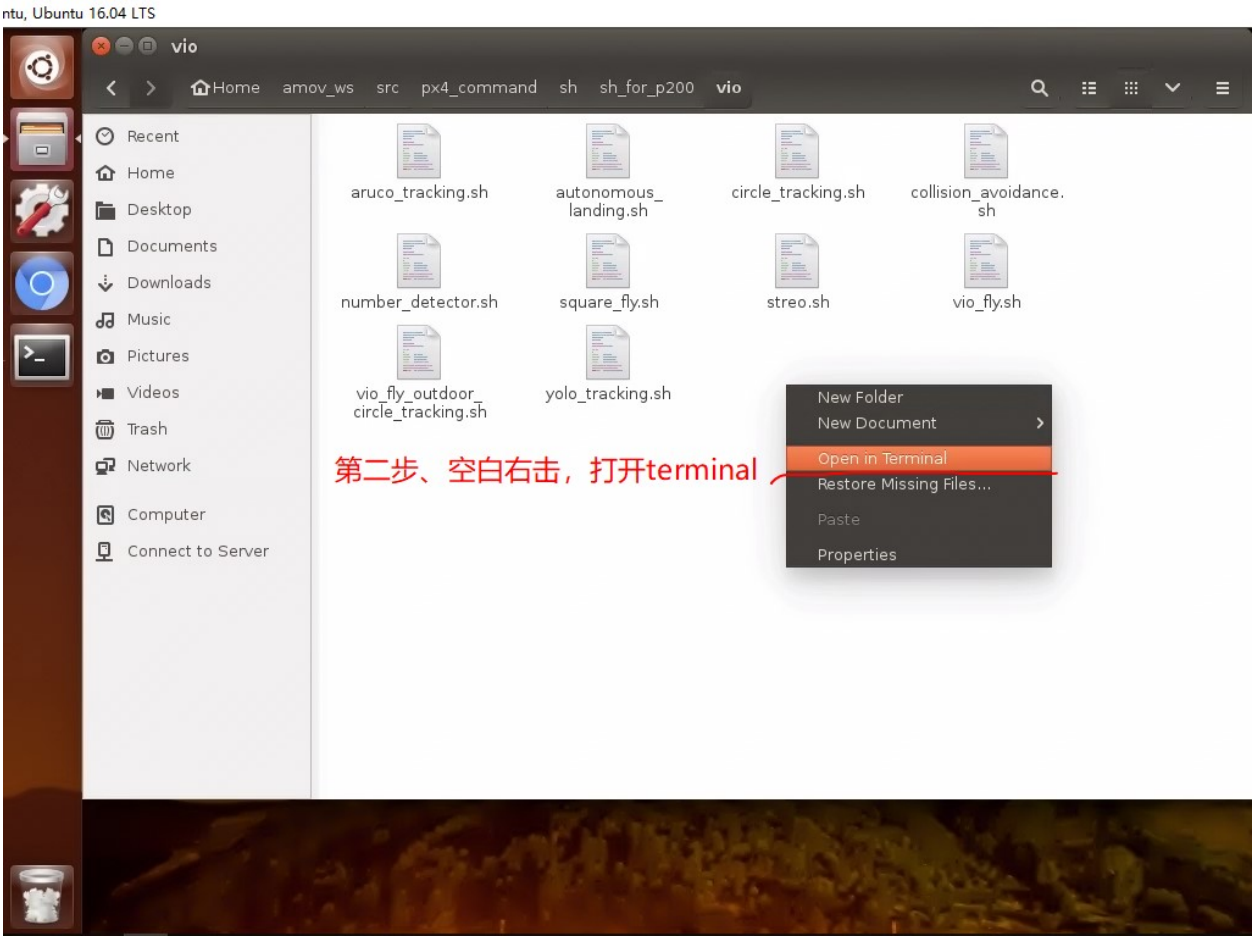
TX2

把飞机放在室内待起飞点，远程已连接好板载计算机与飞控。

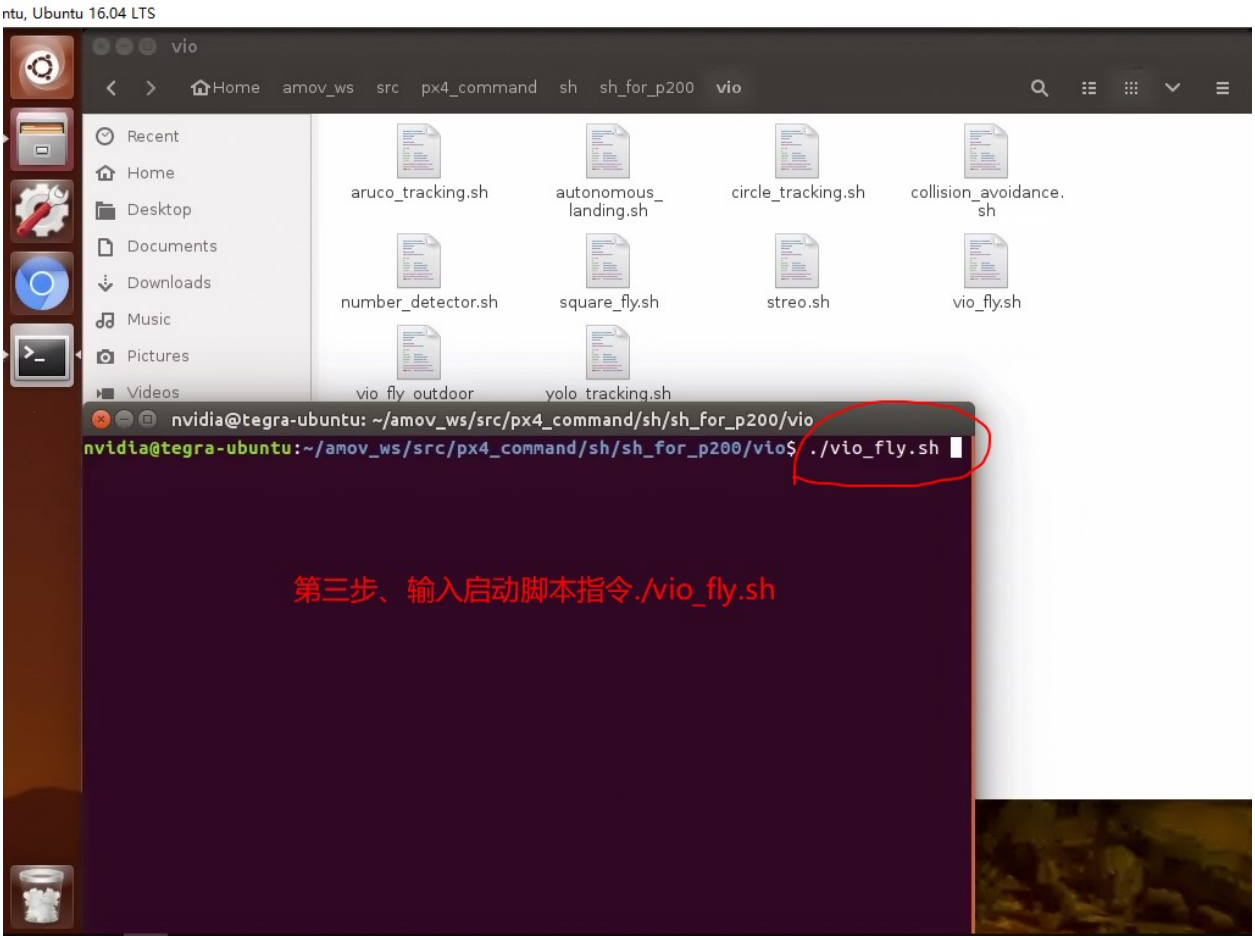
现在已经连接上了 nomachine, 第一步就是打开目录, 进入到 amov_ws/src/px4_command/sh/sh_for_P200/vio 目录下面, 待会准备启动的脚本为 vio_fly.sh.(图片)



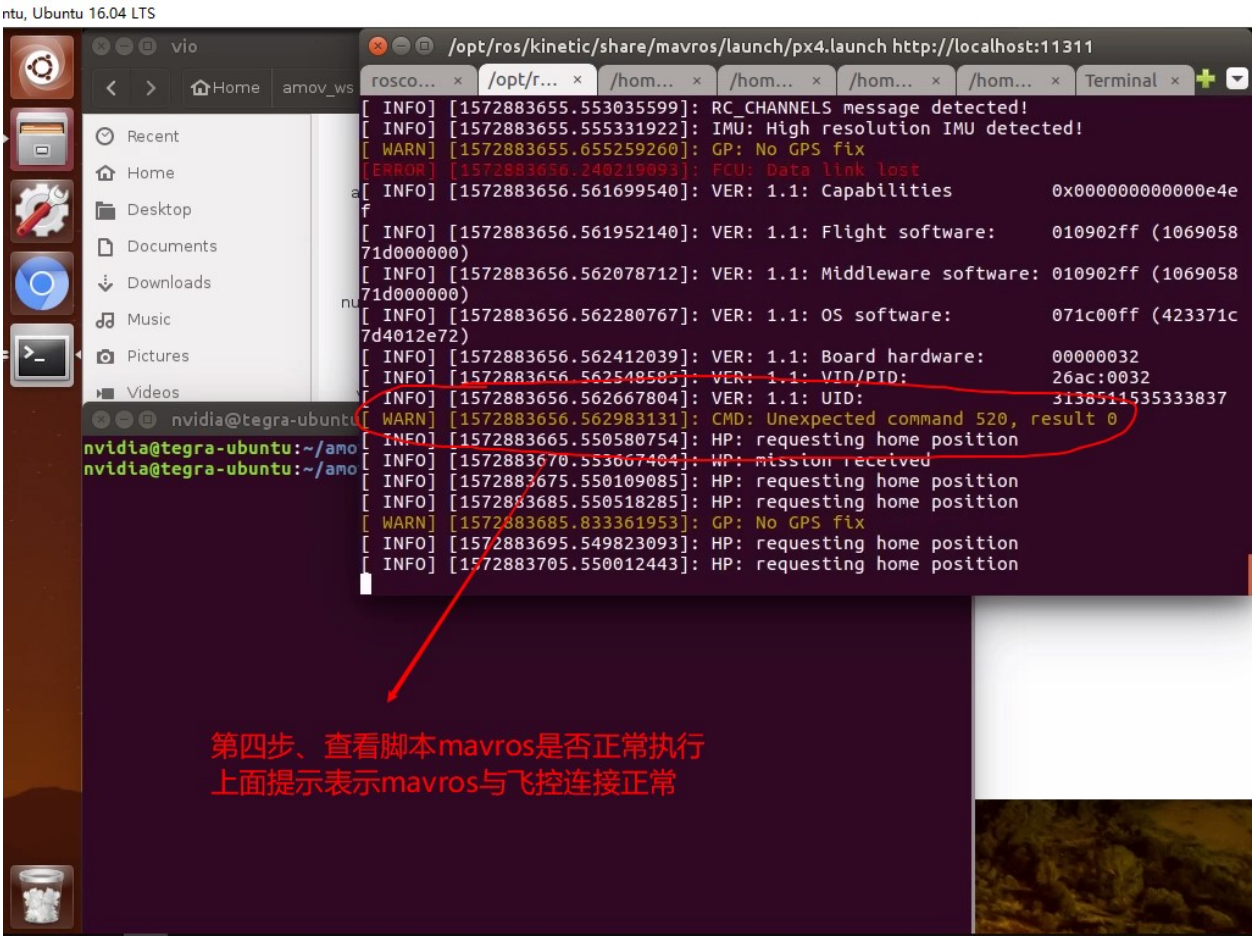
然后进行第二步, 鼠标右击空白处选择打开一个终端 (图片)



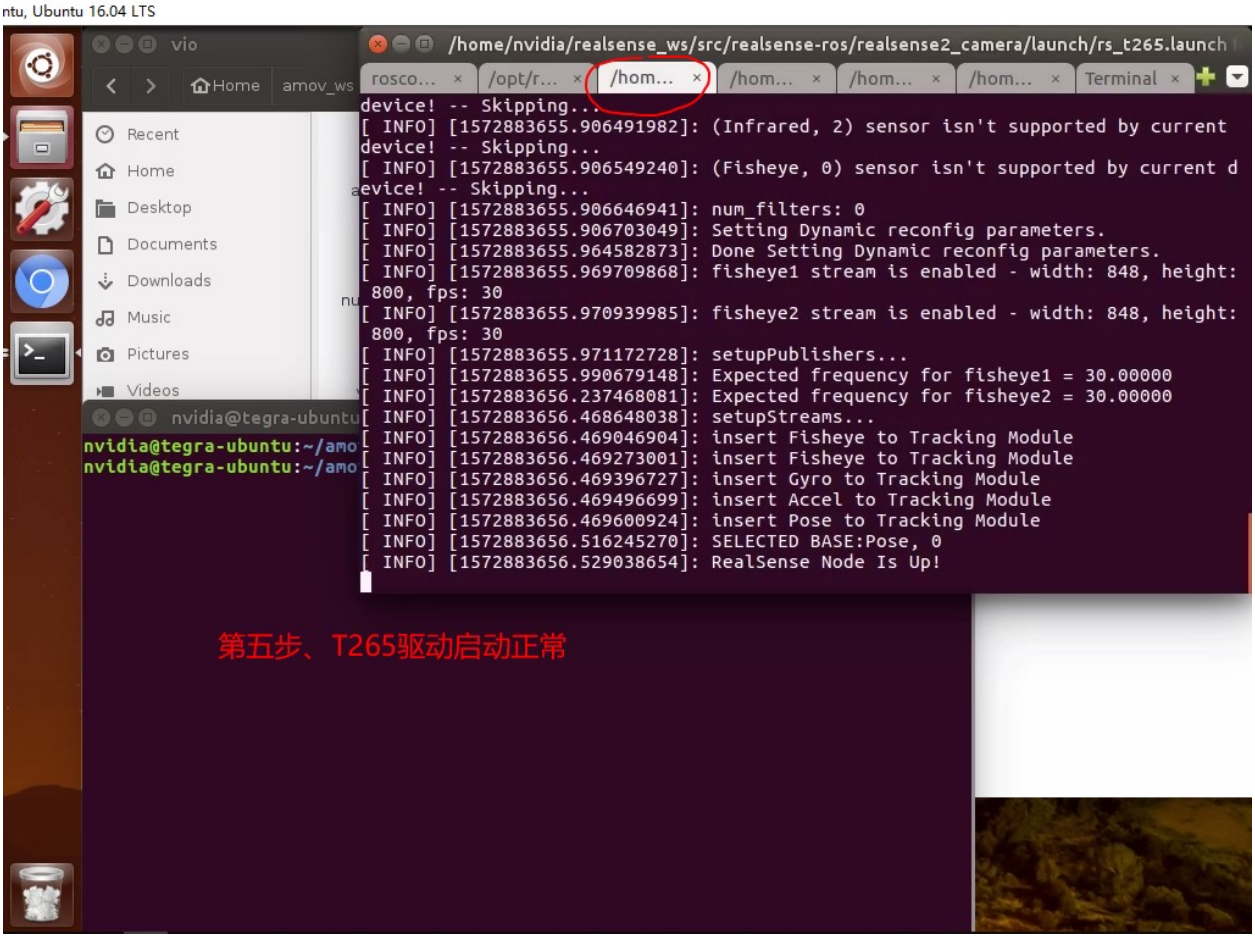
第三步输入启动脚本的执行指令./vio_fly.sh



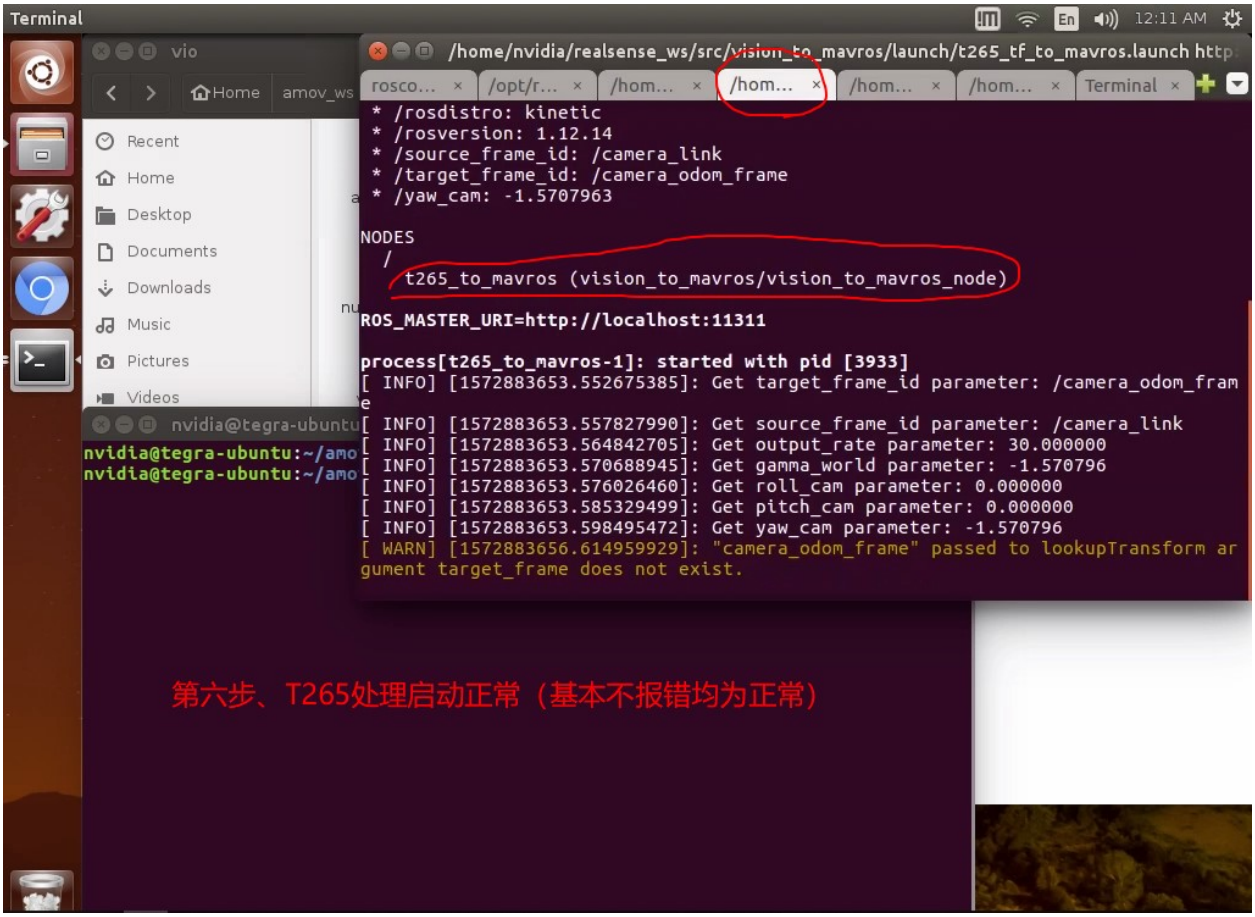
第四步, 启动脚本完成之后, 自动开启脚本终端. 第一个终端为 roscore 此处不做详解. 查看 mavros 脚本是否正常启动, 如果红圈内容说明 mavros 正常与飞控已经建立了通信连接. (如果出现报错的话, 请仔细查看报错的内容原因是什么, 坐标系相关的报错是正常的, 不影响我们飞行使用). 图片



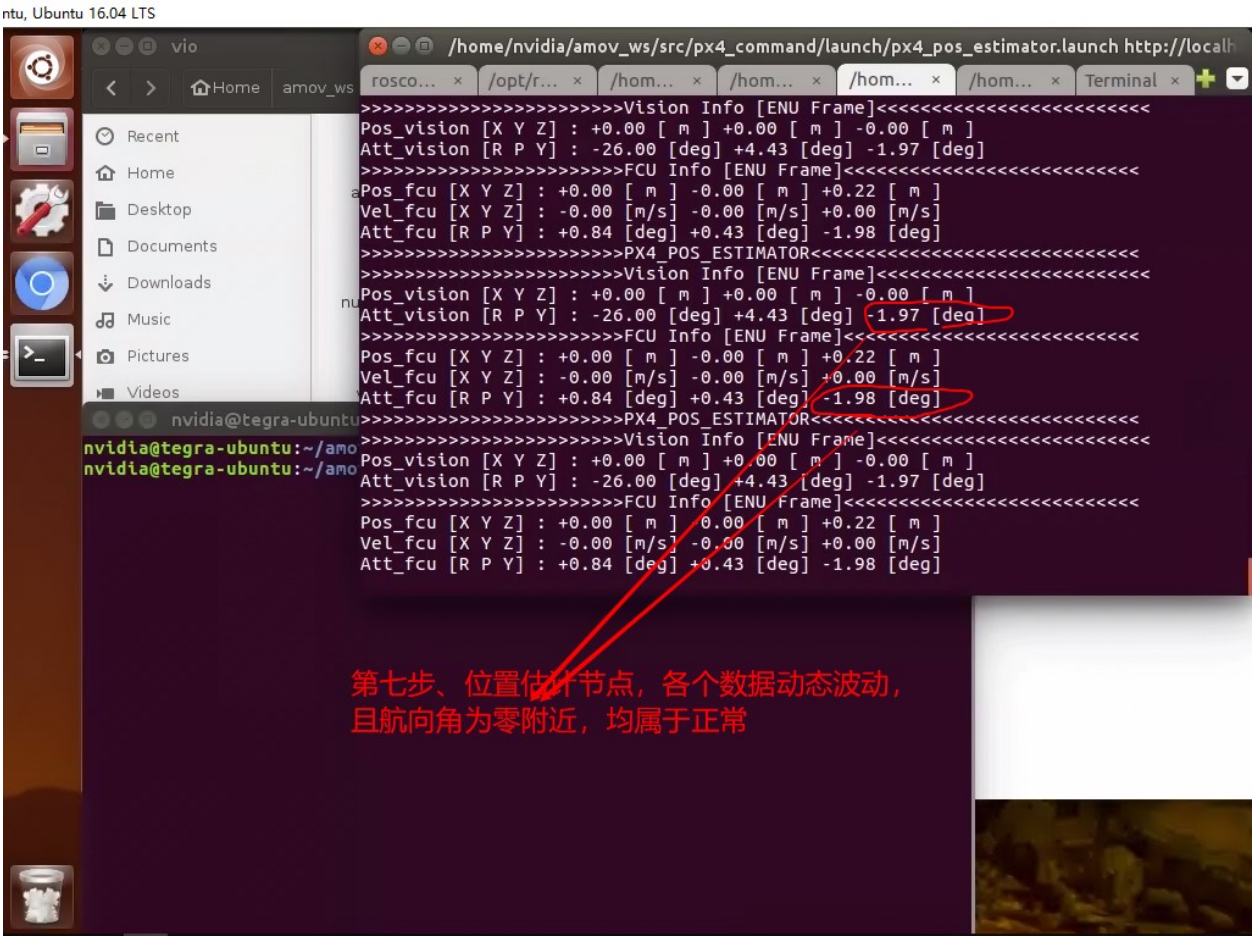
第五步, 该脚本是 T265 驱动启动脚本, 该脚本正常运行, 说明 T265 已经正常启动.



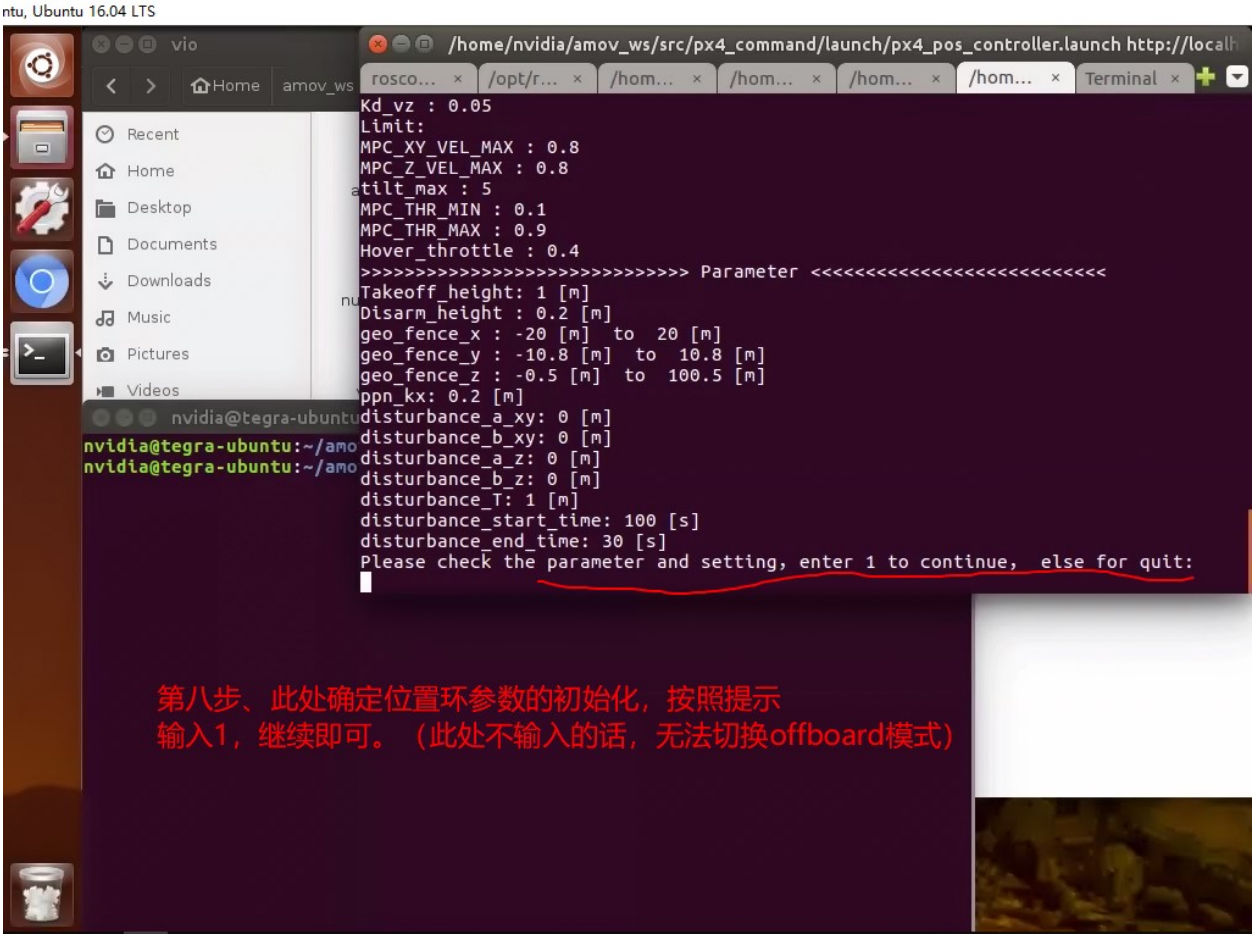
第六步, 是 T265 的处理脚本, 不是异常错误就是正常的, 此脚本角 T265 的位置信息提供给了位置估计节点, 以用作位置估计定位



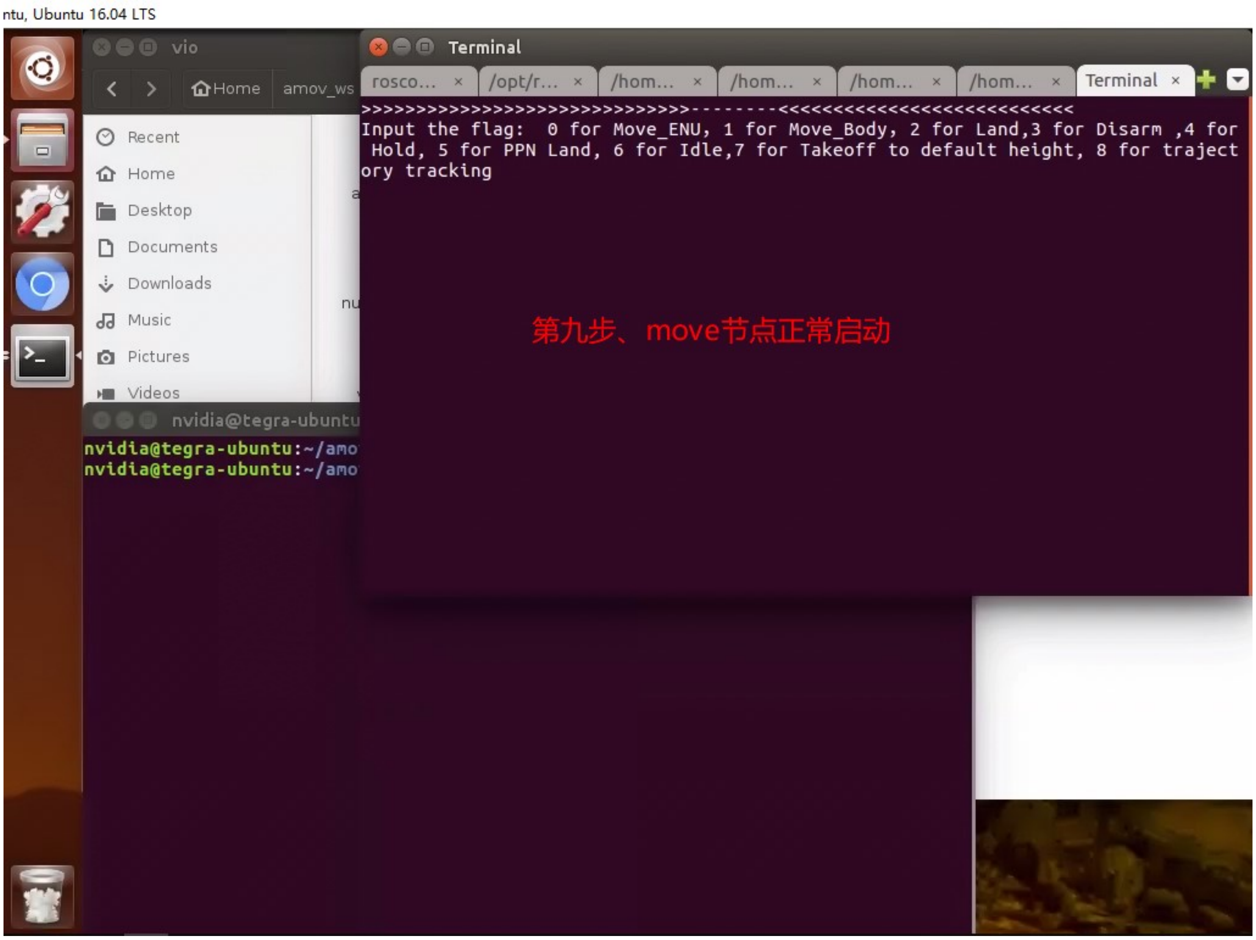
第七步, 位置估计脚本, 此处有多处打印消息, 包含视觉信息, 飞控信息, 两者的各个数据基本保持动态波动就是正常的, 波动很大就是非正常现象. 如下图所示为正常.



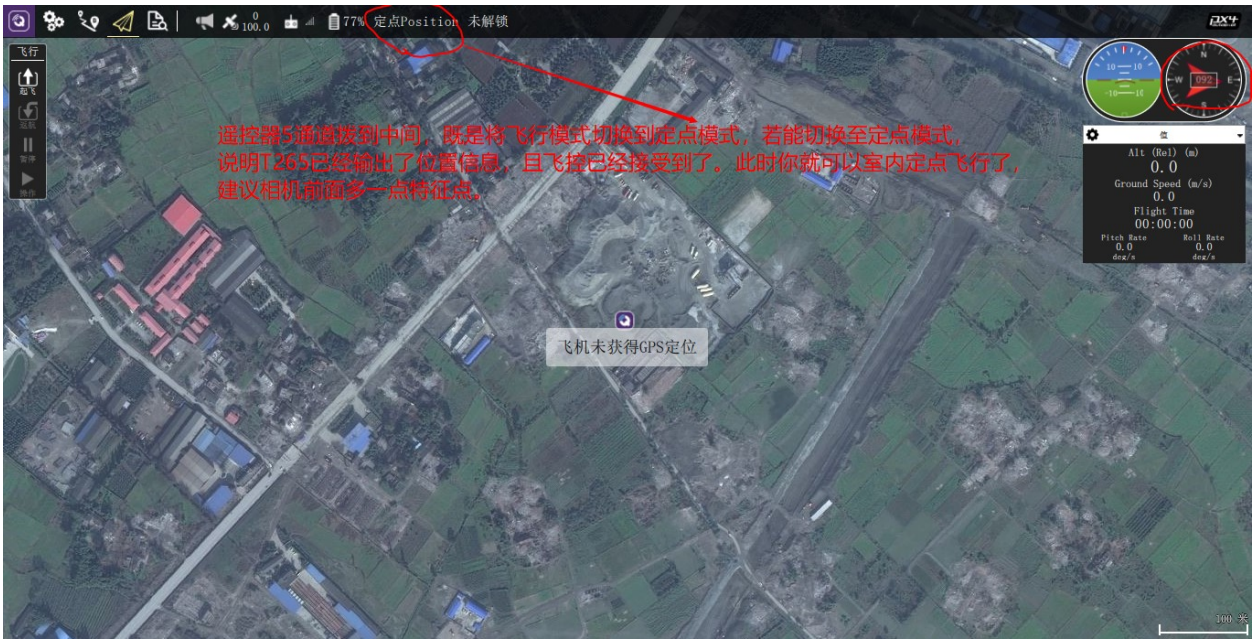
第八步, 是位置控制节点, 按照提示输入 1 继续, 它是位置环参数的初始化. 如果此处不输入 1 继续的话, 你可以在室内进行定位飞行, 但是不能切换到 offboard 模式. 只有输入了 1, 确认位置环参数之后, 才能进入 offboard 模式.



第九步, 为 move 节点, 正常启动. 他可以控制飞机前后左右, 位置速度控制飞行.



此时你也可以用另外一台电脑连接到飞控中去, 按照开箱第一步的教程执行. 上面脚本如果都能正常运行的话, 此时在遥控器上的 5 通道 (三段) 拨到中间, 将飞行模式改为定点模式, 在地面站 QGC 上面可以看到是可以切到定点飞行模式的, 这就说明, 此时可以在室内进行定点飞行了.



NANO

使用操作和 TX2 类似, 区别在于文件目录不一致.

室外飞行

视觉 SLAM

- EKF2_AID_MASK = 24 （选择位置以及偏航来源）；EKF2_HGT_MODE = Barometric pressure/GPS/Vision（选择高度来源）

TX2

待续……

NANO

GPS/RTK

- EKF2_AID_MASK = 1（选择位置以及偏航来源）; EKF2_HGT_MODE = Barometric pressure/GPS（选择高度来源）

TX2

待续……

NANO

经过入门本章内容, 你就可以在室内外进行定点飞行, 若遇到困难请上论坛求助：阿木社区论坛 .

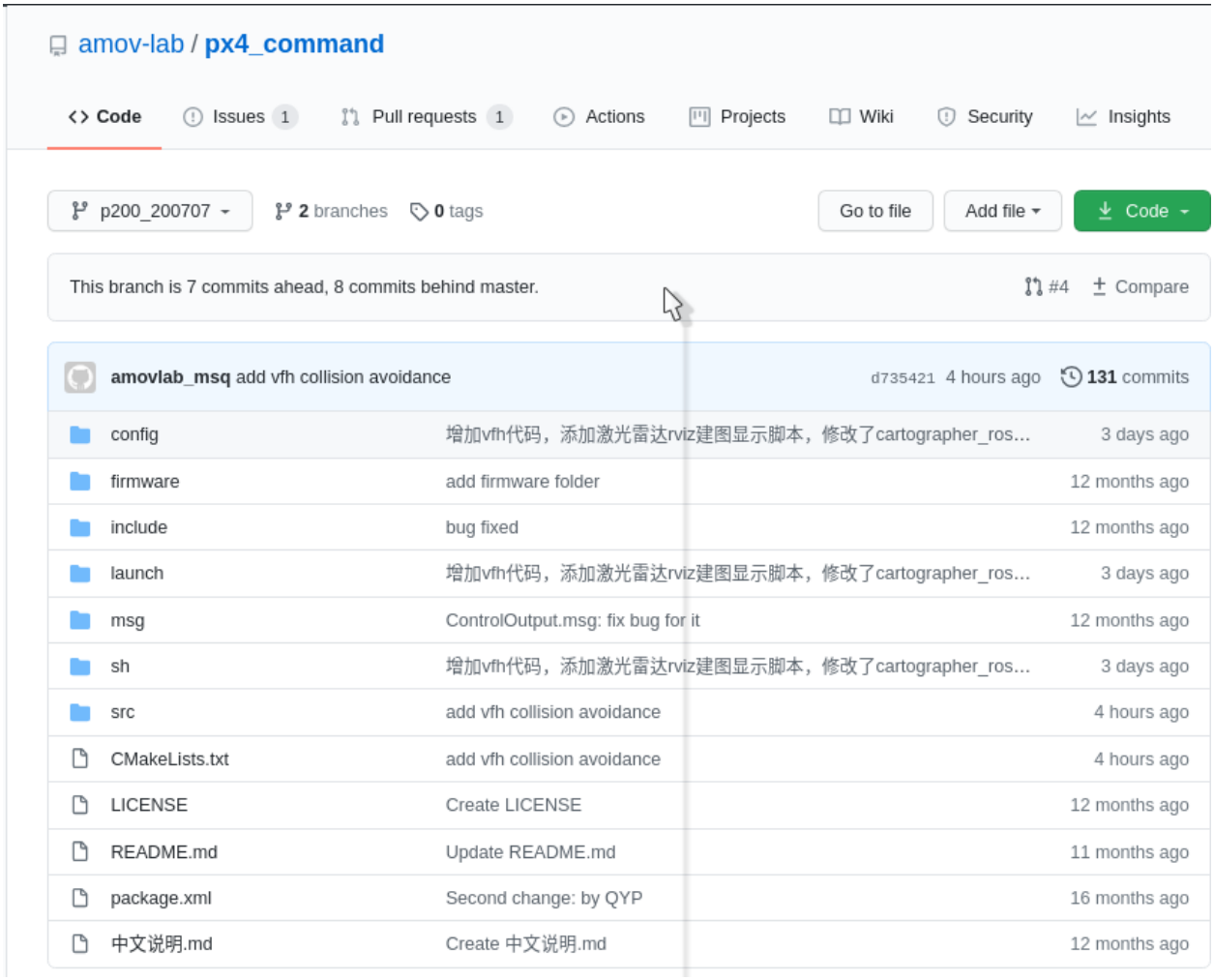
在下一章自主飞行之进阶中, 会详细讲解如何在室内进行 move 控制, 自主跑四边形, 室外圆跟踪等 demo 实现教程.

自主飞行之进阶

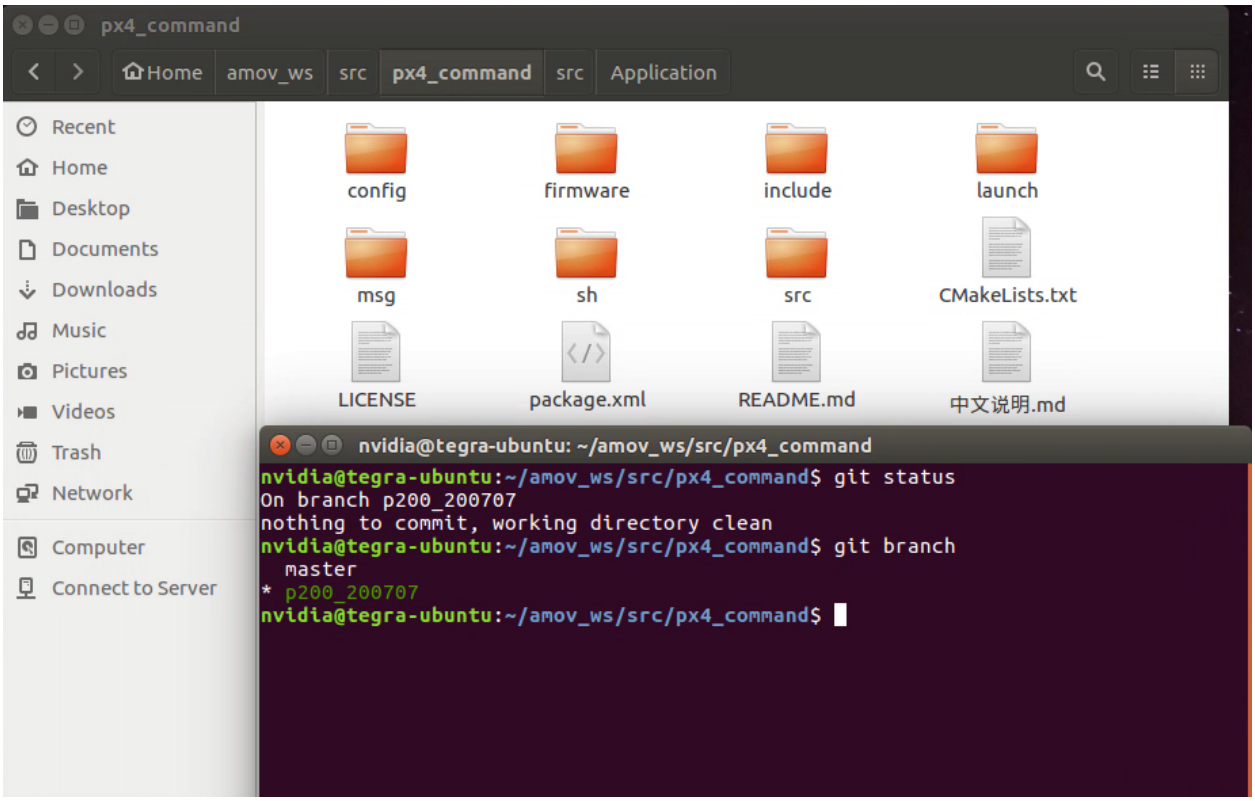
1.vfh 避障 demo

a. 更新 px4_command 功能包

px4_command 最新功能包在 github 上面。请点击链接 px4_command , 我们更新的不是 master，而是分支 p200_200707 , 如下图：



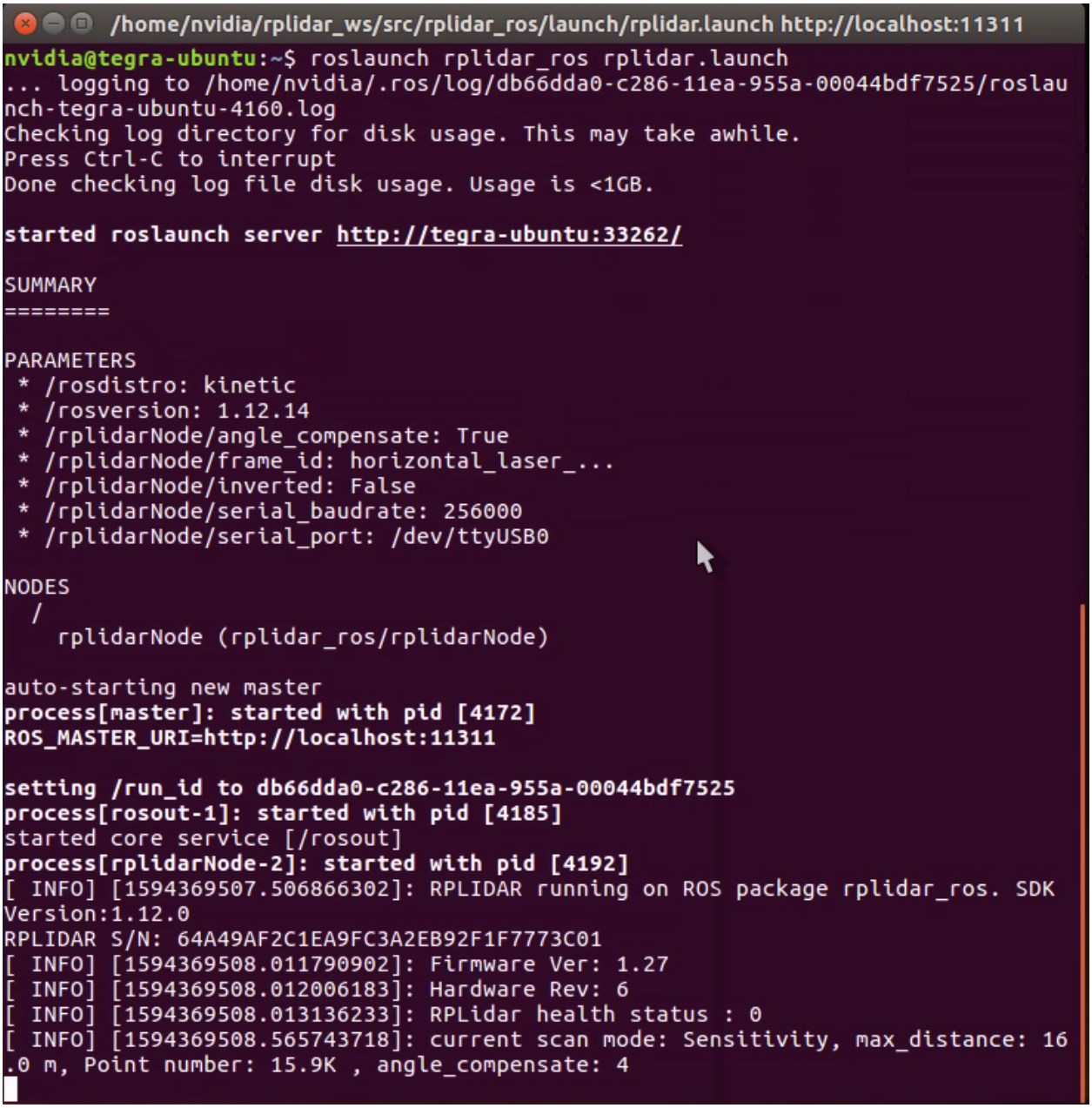
现在进入到 TX2 里面，确保能够正常上网，本身有个 px4_command 功能包，现在我们删除掉原有的 px4_command，重新 git clone 最新的功能包。使用 git clone https://github.com/amov-lab/px4_command.git ，进入到 px4_command，切换 p200_200707 分支下面，使用命令 git checkout p200_200707 . 然后退回到 ~/amov_ws 目录之下，使用 catkin_make 编译 px4_comman 功能包。如下图所示，我们使用的就是 p200_200707 该分支



b. 更新 rplidar_ros 功能包

我们通常使用的是思岚科技的 A2，A3 或者 S1 雷达，不管您是使用 A2，A3 还是 S1 雷达做避障，都建议更新一下激光雷达功能包。rplidar_ros 的源码地址为 https://github.com/amov-lab/rplidar_ros.git

删除之前 rplidar_ws 里面的编译生成文件夹 build 和 devel。删除掉 src 下面的所有文件及文件夹，在 ~/rplidar_ws/src 路径之下下载源码，使用 git clone https://github.com/amov-lab/rplidar_ros.git ，然后进入到上级目录 ~/rplidar_ws 下面，执行 catkin_make 进行编译，编译完成之后，如果之前没有删除 ~/.bashrc 下面的环境变量，你无需添加新的环境变量。打开新的终端，使用 roslaunch rplidar_ros rplidar.launch 启动激光雷达，如下图：



默认的 rplidar.launch 的波特率是 256000 ，我这里使用的是 A2 的雷达，A2 雷达有的是 115200，有的是 256000. 每一款机关雷达底座下面有具体的型号。这里我所使用的是 A2M8R4 系列，对应波特率为 256000.

c. 实际避障前说明

第一点、激光雷达的的正方向是有连接线的一边，我们称之为激光雷达的尾巴，正确的在飞机机架上的安装方式为，激光雷达的尾巴朝向飞机的机尾。这个很重要，不要搞错了!!! 如果您拿到的飞机，激光雷达的尾巴是朝前的如果要使用室外激光雷达避障功能，请将激光雷达尾巴改为朝向飞机的机尾。一定注意！一定注意!! 一定注意!!!

第二点、避障前可熟悉仿真环境中的避障，这个必须要配置一下仿真环境。也可以学习 无人机仿真课程开发

，里面有降到 vfh 的无人机避障教程。这个是为了让你熟悉避障的整个流程和逻辑。

第三点、在避障过程中我们控制的是飞机的水平 xy 速度，高度 z 保持一定值不变。在室外使用地面站 QGroundControl 规划一条航线。正常情况下飞机按照这个规划航线飞行，现在如果中途有障碍物，飞机会自行绕开这个障碍物。我们可以实时打印出 ros topic,rostopic echo /mavros/setpoint_raw/local 这个可以方便我们查看 offboard 模式下，tx2 通过激光雷达给飞控发送了水平方向的速度是怎么变化的。

第四点、飞机整体性能都没问题，怎么判断呢？就是正常的自稳下飞行可控，position 下面飞行的还挺稳定，GPS 飘的不是太大。

第五点、在室内有接显示器的情况下将 tx2 所连接的 WiFi 为飞机上自带的 WiFi 数传或者图传模块，方便室外下直接使用 nomachine 远程连接到 tx2。

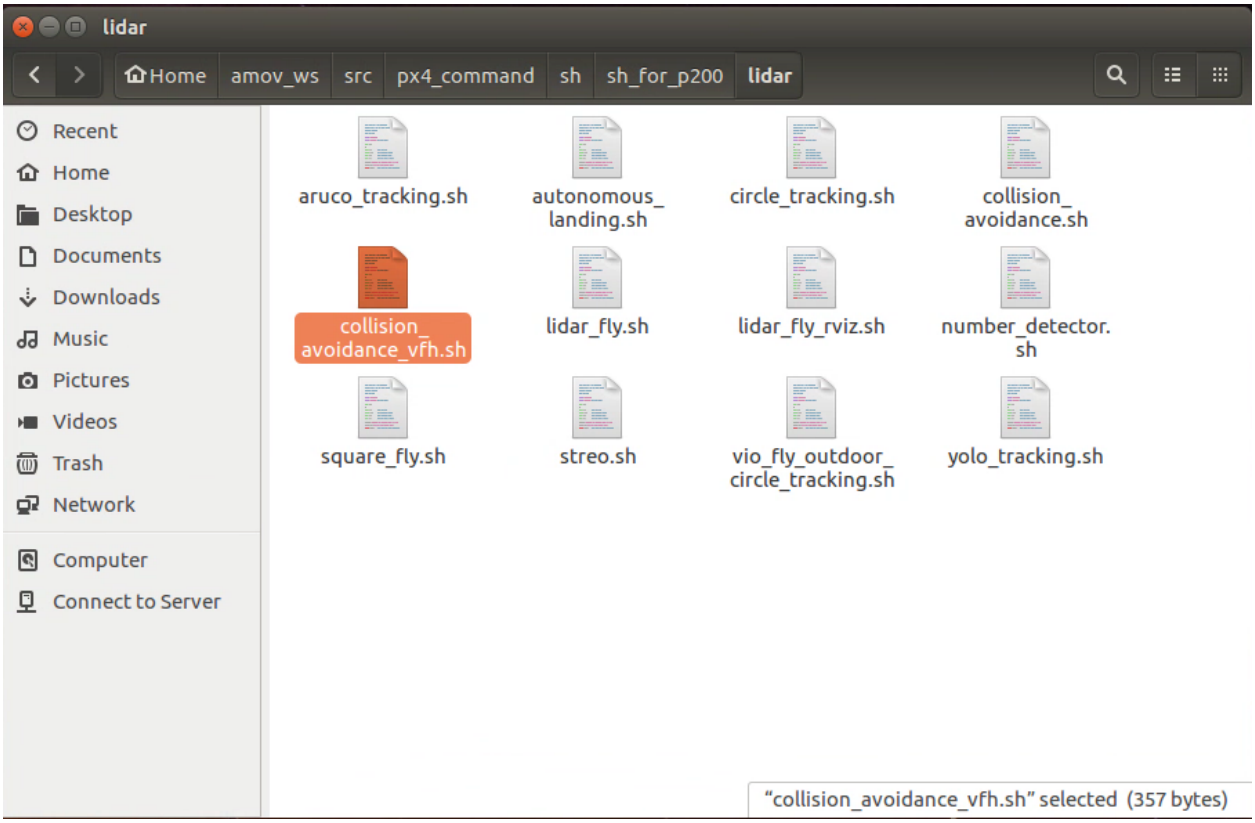
第六点、GPS 的位置不要遮挡住激光雷达扫描处，即便要遮住部分激光雷达数据，要把 GPS 位置放到飞机尾部。

d. 避障过程

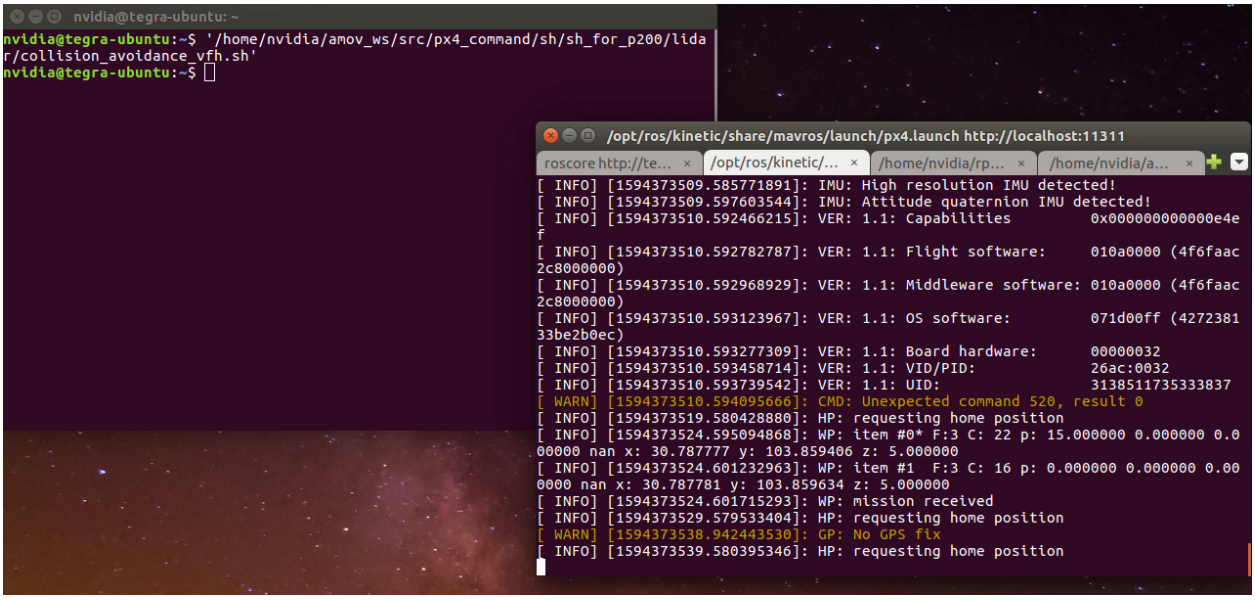
避障可视化过程可仔细查看我们提供的视频 demo 讲解。

室外飞机准备都正常情况下，现在地面站 qgc 上面规划一条航线，先让飞机正常飞行 mission，注意高度不要太高，如果不熟悉飞机，建议使用绳子将飞机牵住。要是飞行 mission 航点没有问题，继续执行下面的避障脚本。注意一下，每次飞完一次飞机，建议重启 reboot 一下，直接在 qgc 中输入 reboot 远程重启。遥控器飞行结束之后保持在自稳模式下。

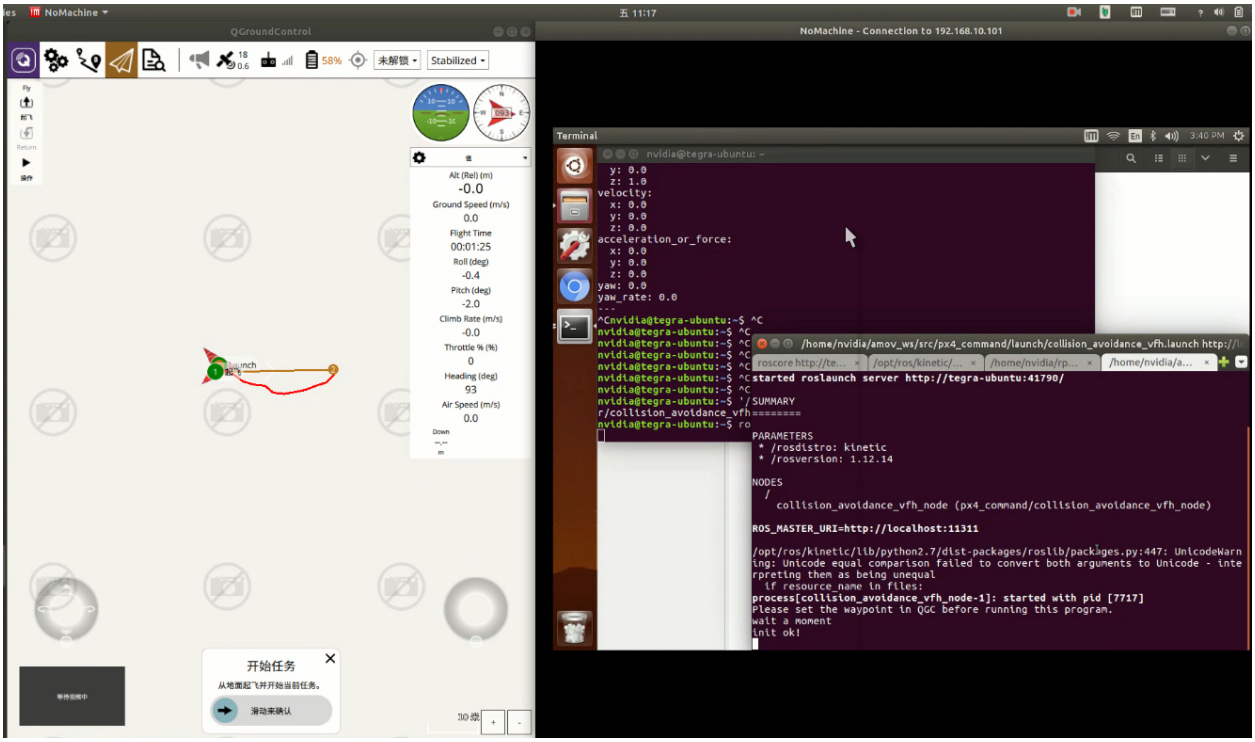
启动避障脚本。在 ~/amov_ws/src/px4_command/sh/sh_for_p200/lidar 路径之下，执行./collision_avoidance_vfh.sh 脚本。



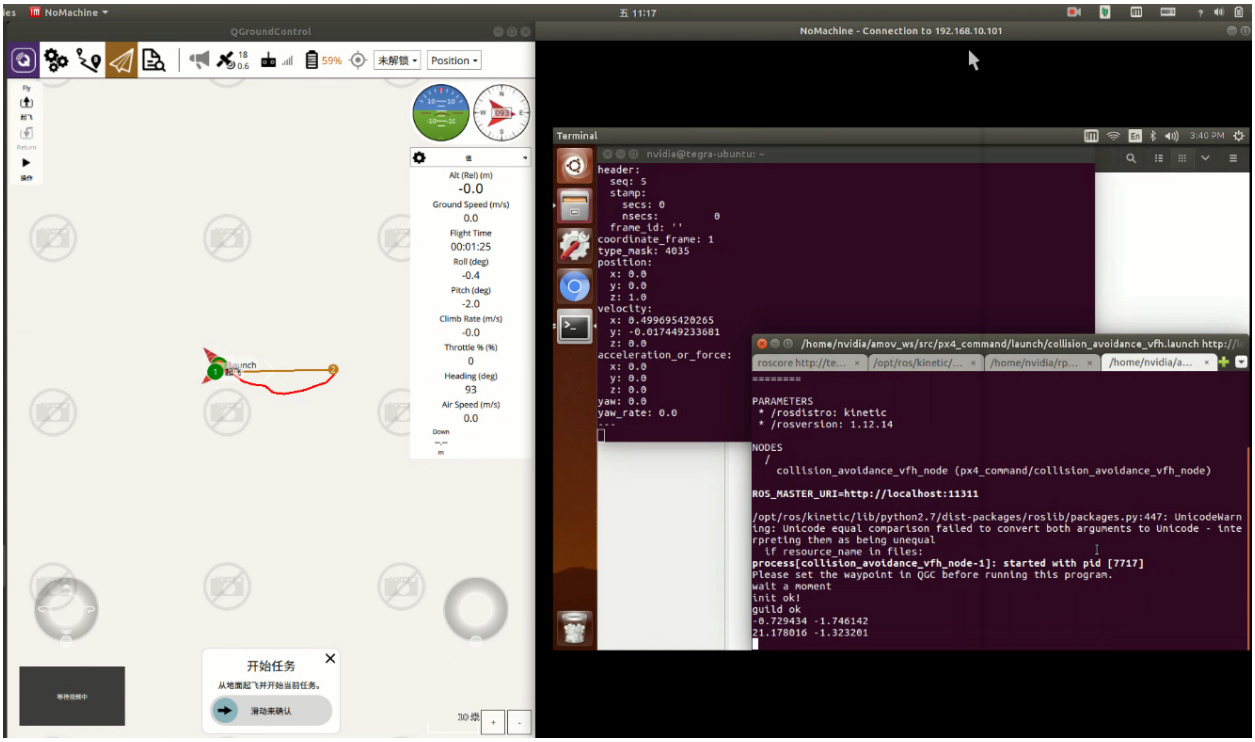
或者直接将该脚本拖动到终端中启动，如下：



等待初始化 OK，可以看到飞机的头是指向正东 E 的。航向角为 93° ，基本满足指向正东。



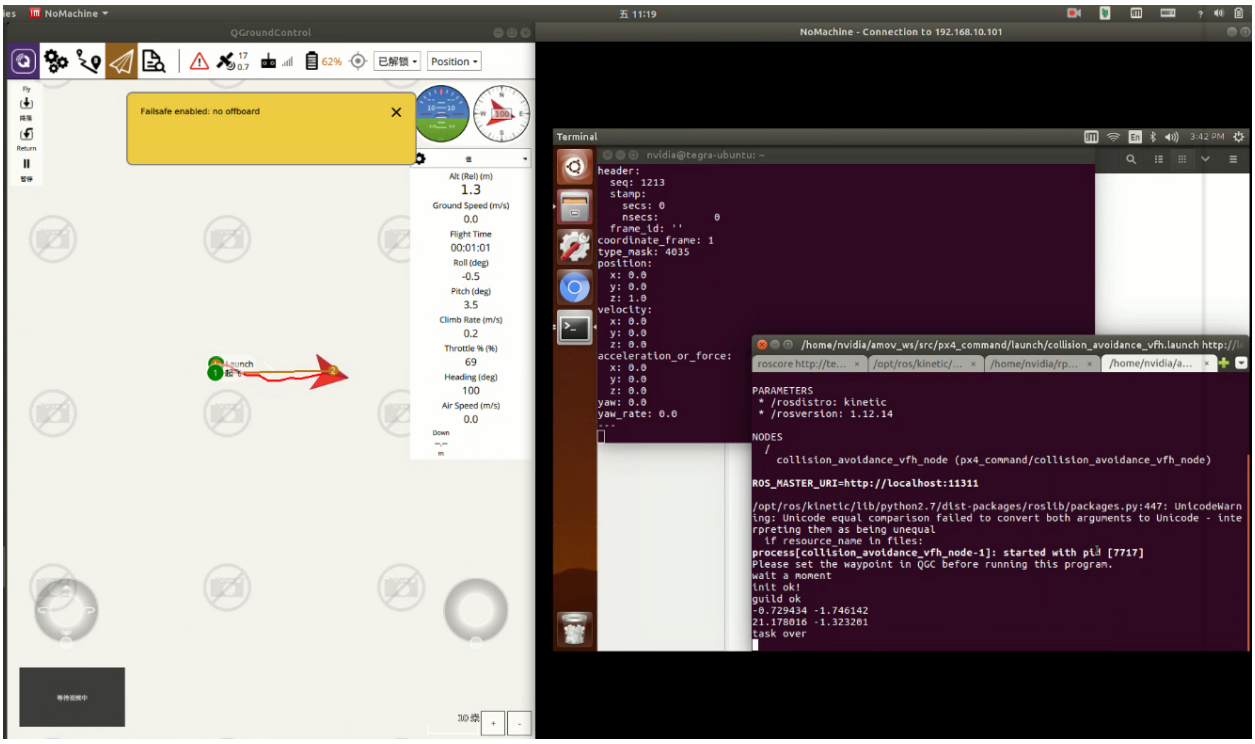
然后遥控器切换 position 模式，就会初始化两个航点，然后可以看到 rostopic echo /mavros/setpoint_raw/local 也有给飞机发送期望的数据，速度有 xy 数值，高度 z 一直发送 1.



此时就可以在 position 下进行起飞，先保持悬停，然后切换遥控器切换到 offboard 模式。遥控器保持在悬停的时候就不要动摇杆了，直接切换两段开关至 offboard 模式即可。飞机就会按照规划的路径飞行，飞行高度

保持 1 米。xy 水平速度会实时变化，有障碍物了 x 速度减少，y 速度负增大或者正增大，绕开障碍物继续前行。实际飞行过程中一定要注意安全!

结束时候，到达规划航点时候，如下图所示，避障节点打印了 task over ，说明避障节点运行结束，飞机此时也会退出 offboard 模式，从而进入到 position 模式，这个时候需要使用遥控器在 position 模式下面缓缓将飞机降落。



因为该平台开源，可变因素很多，所以实机飞行过程中一定要注意安全。飞行完成一次之后，要将飞控 reboot 重启，重新运行避障脚本。

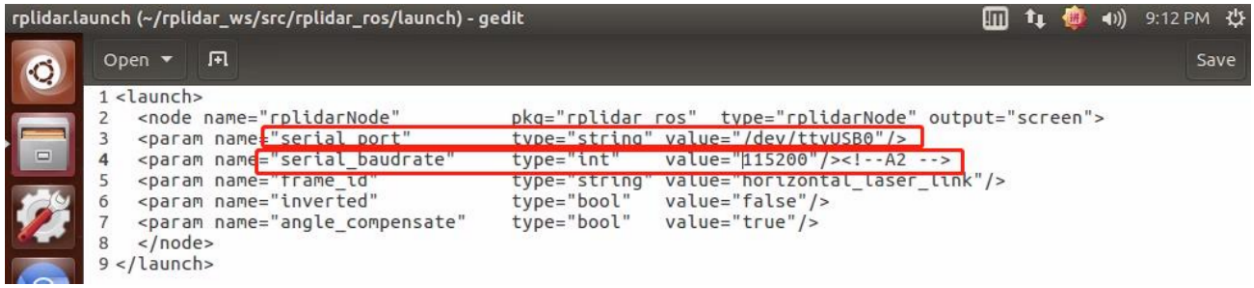
2. 激光雷达室内建图定位

在我们目前的 P200 飞机上搭载的默认激光雷达是可以实现定位并且可以建图。使用的激光 SLAM 算法为谷歌开源的 cartographer。在 p200 真机中，为了节省板载计算机的 cpu 的使用率，我们做了一些设置，只是启动了 rartographer_ros，让这个算法跑了起来，没有为其配置 rviz. 下面描述一下如何使用 rplidar,cartographer_ros 实时建图并且显示 rviz 下的地图。

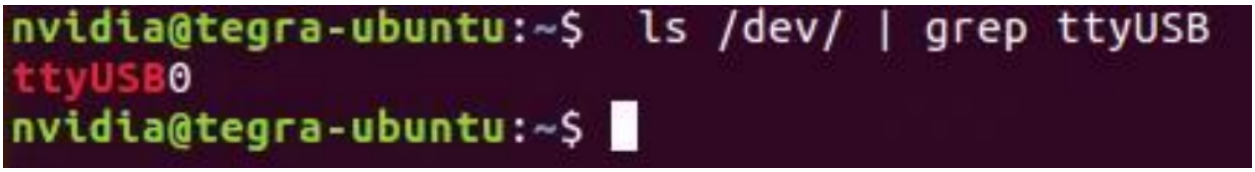
1. 确保 rplidar_ros 能够正常启动激光雷达

首先在终端输入命令 `roslaunch rplidar_ros rplidar`

启动成功需要注意两点，查看 rplidar.launch 文件中的 serial_port（串口设备）和 serial_baudrate(串口波特率)，如下图所示:



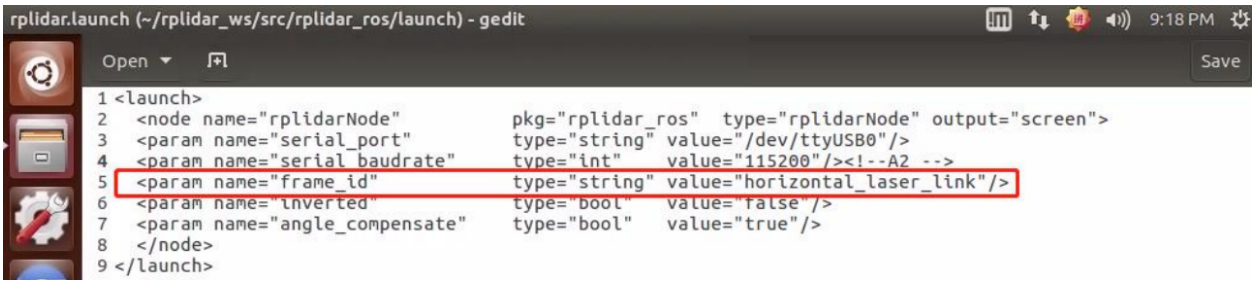
查看鐵口设备可通过命令：`ls /dev/ | grep ttyUSB`



查看设备为 ttyUSB0, 所以上图的 serial_port 为 ttyUSB0。

其次是波特率的问题，激光雷达 A2 的波特率一般为 115200，需要将激光雷达的中间转换模块的波特率和 launch 文件的波特率拨到一致，需要自行检查一下。

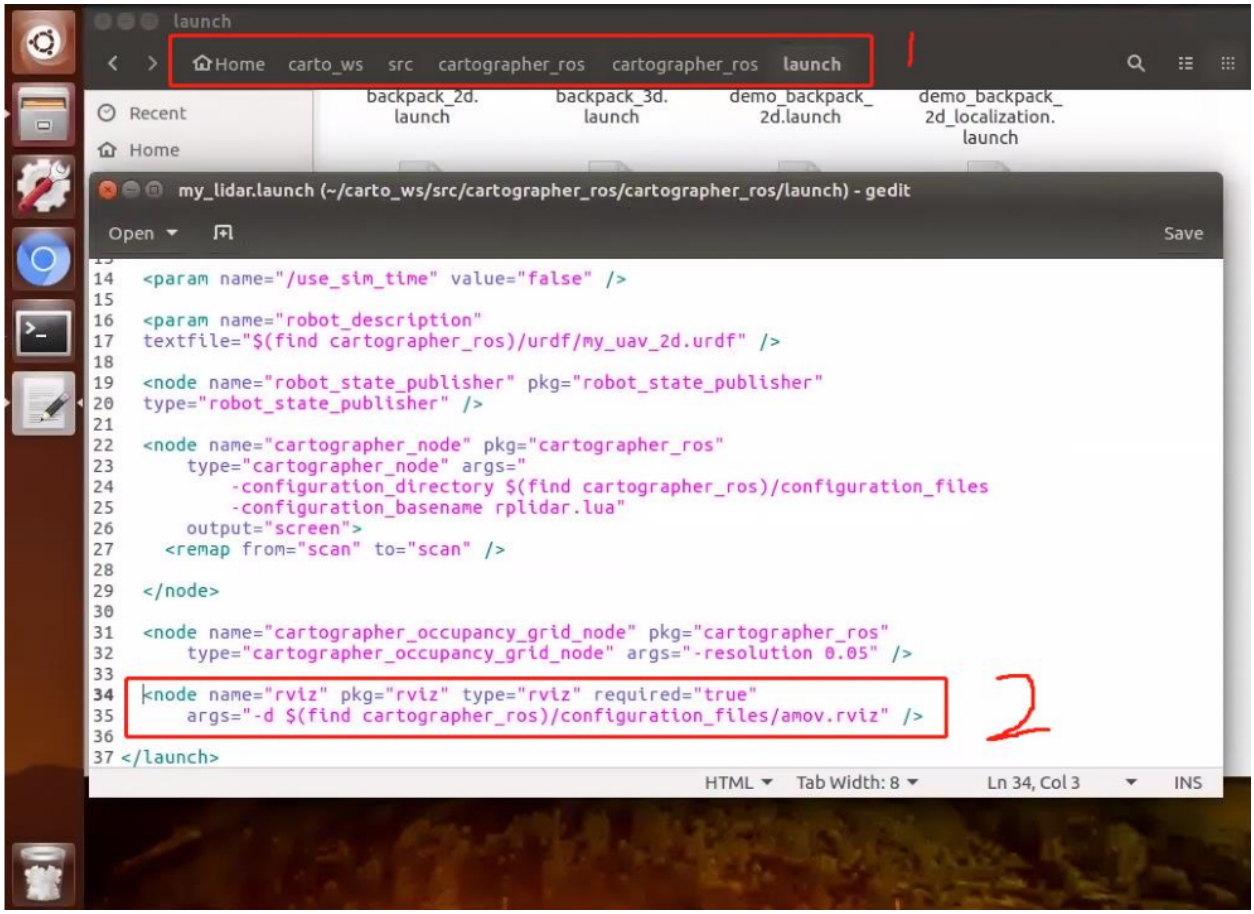
上面两个注意到了，一般会正常启动 rplidar。



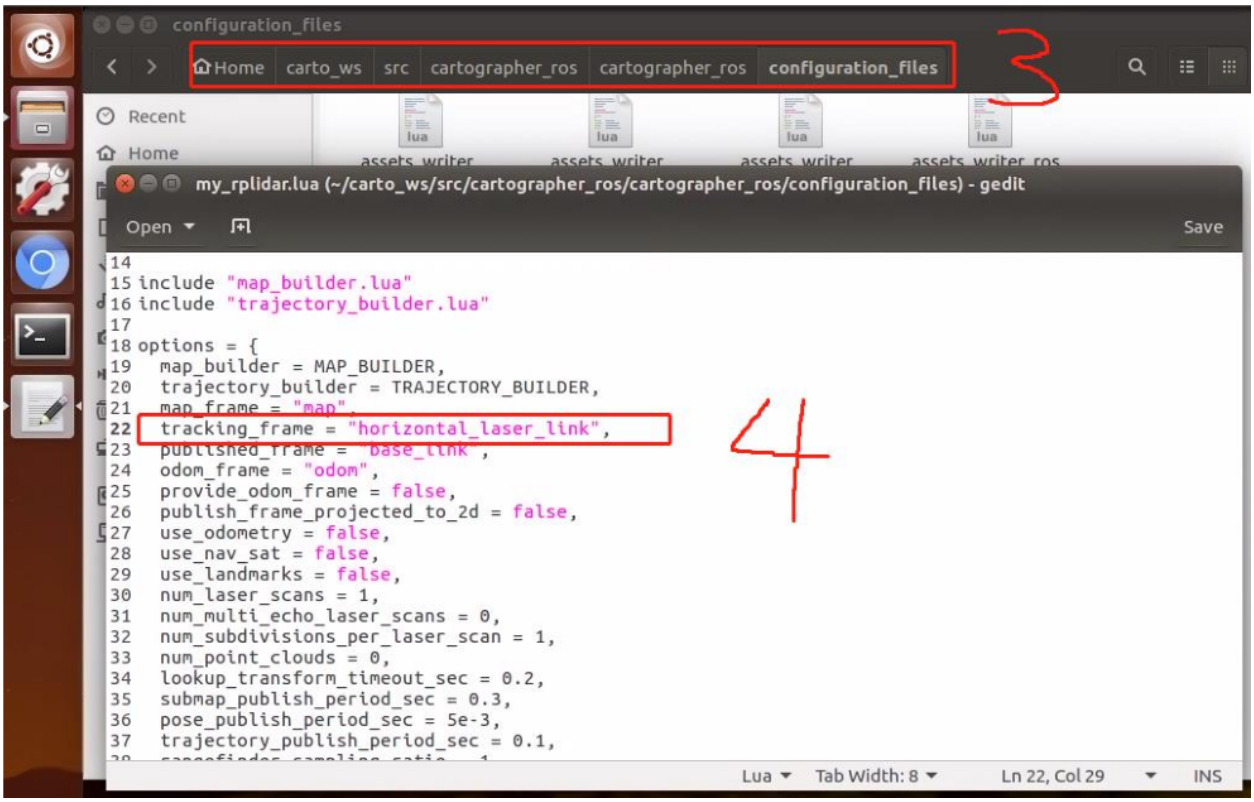
还有一点需要注意的是，frame_id, 在这里将其设置为 horizontal_laser_link, 这个改动后续会在 cartographer_ros 中用到。

2. 修改 cartographer_ros

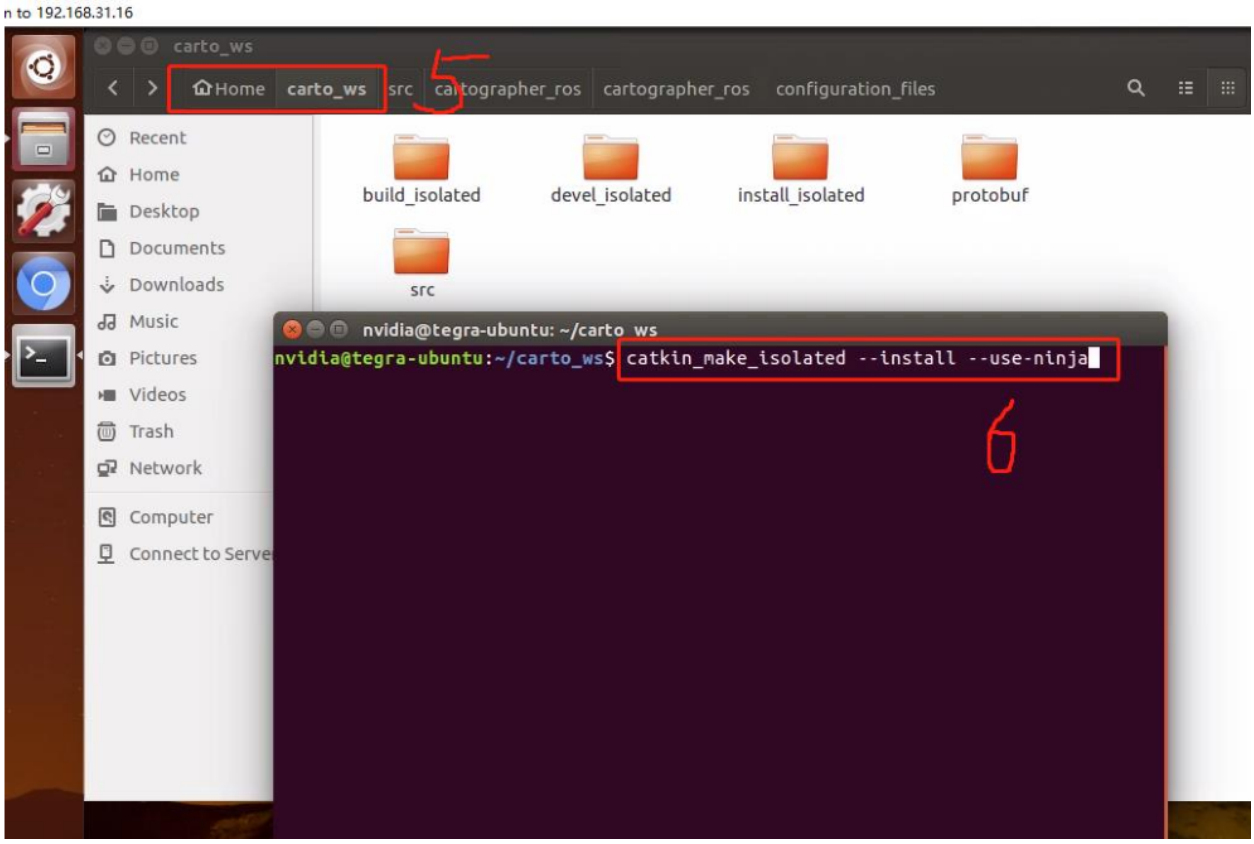
首先进入到如图所 1 标识的路径下，修改编辑 my_lidar.launch，如下所示，添加如下图 2 标识的两段代码，保存。所添加的这两段的代码就是在启动 cartographer_ros 的时候启动一下 rviz，进行显示。



然后在 3 标识的路径下面，打开编辑 my_lidar.lua 文件，将 4 标识的第 22 行 tracking_frame 改为启动激光雷达 launch 文件中的 frame_id，确保是一致的，修改为 horizontal_laser_link。

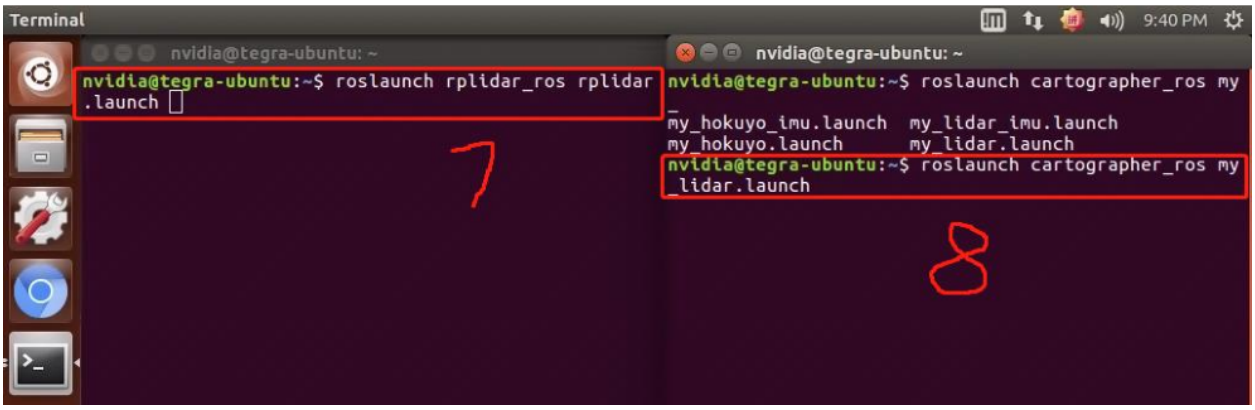


接着在 5 标识的路径下，编译 cartographer_ros，使用如 6 标识的指令：`catkin_make_isolated --install --use-ninja`

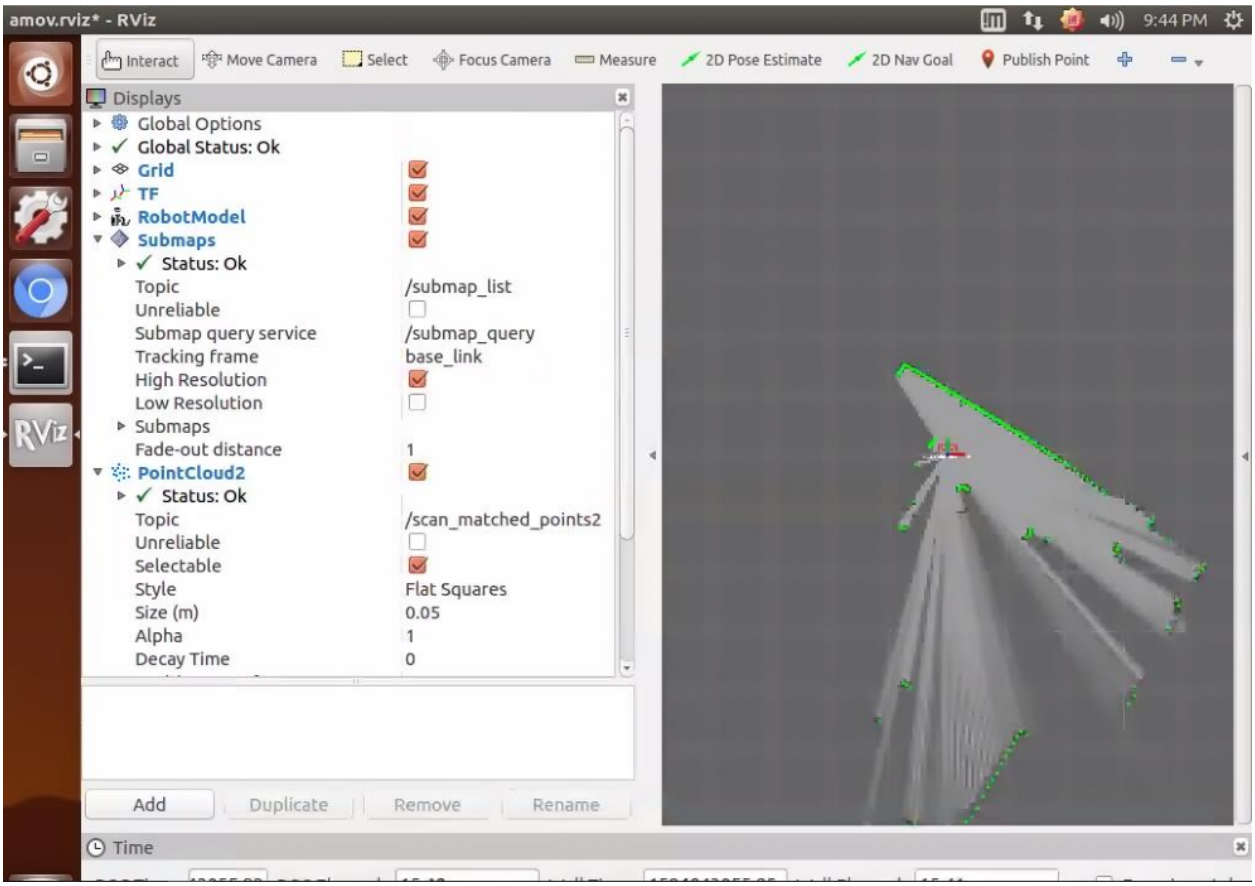


3. 建图演示

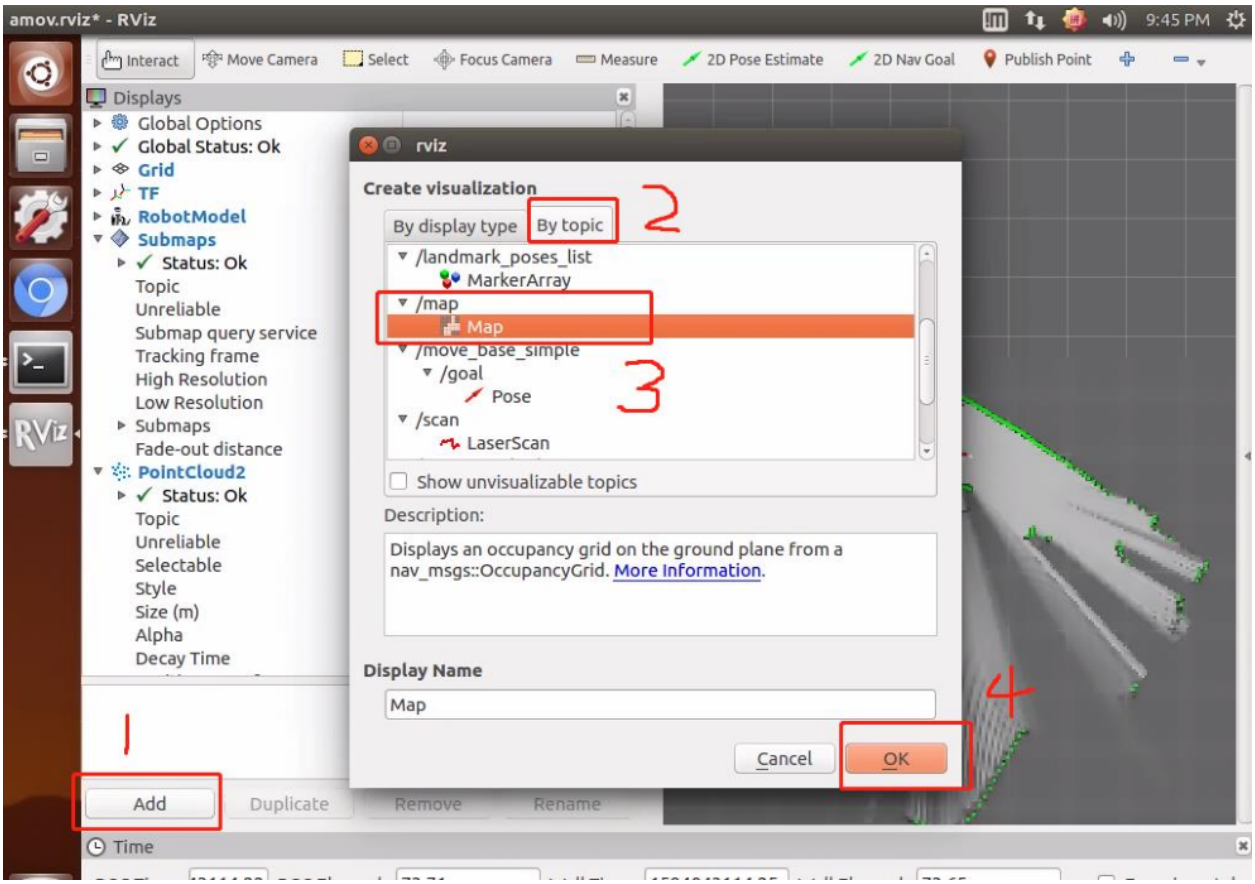
编译完成以后，启动激光雷达节点和 cartographher_ros 节点。打开两个终端 terminal, 一个如 7 标识启动激光雷达：`roslaunch rplidar_ros rplidar.launch`。另一个如 8 标识启动 cartographer_ros: `roslaunch cartorgapher_ros my_lidar.launch`。



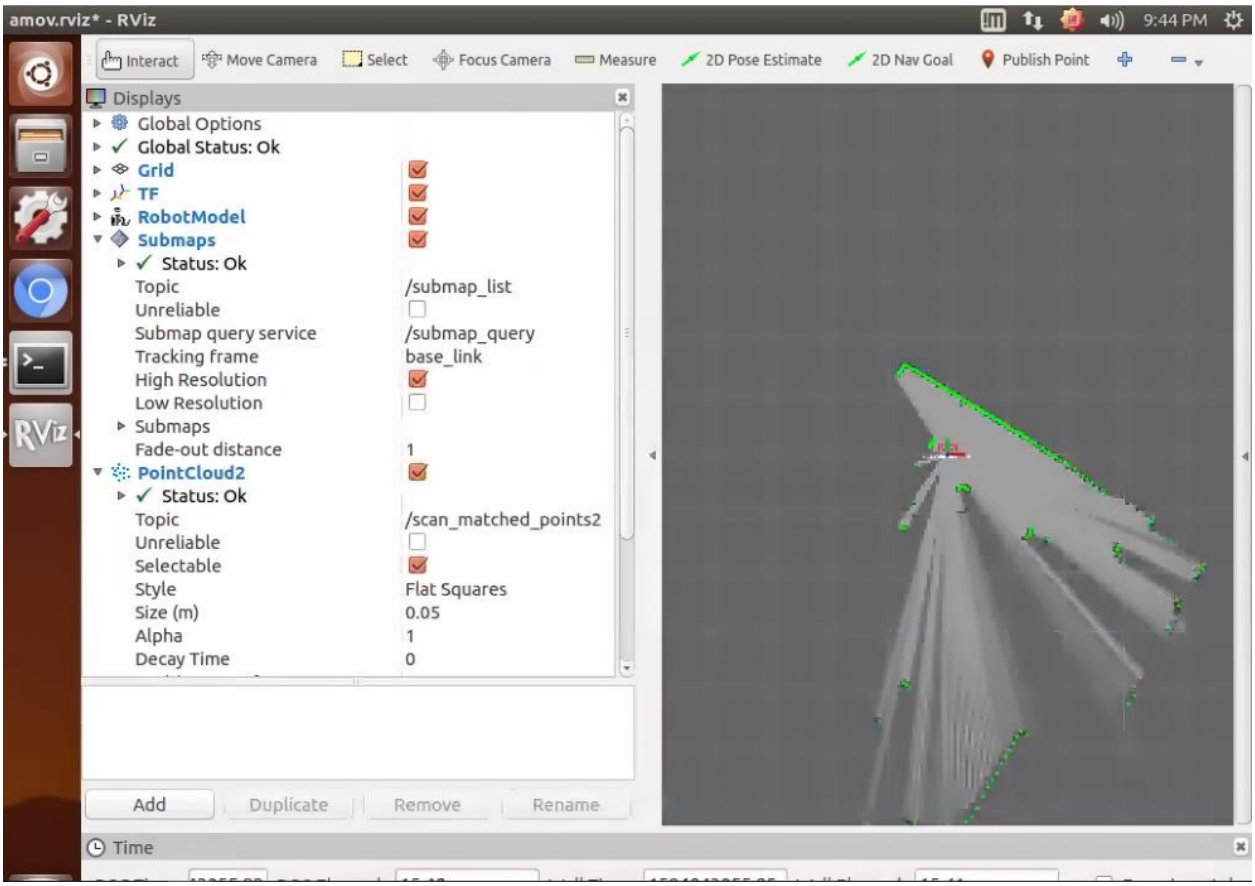
启动完成以后会自动打开 rviz, 并有雷达扫点，这就是正常的。如下图所示：



打开建图 map。1 点击 add 2 点击 topic 3 点击 map 4 点击 OK。

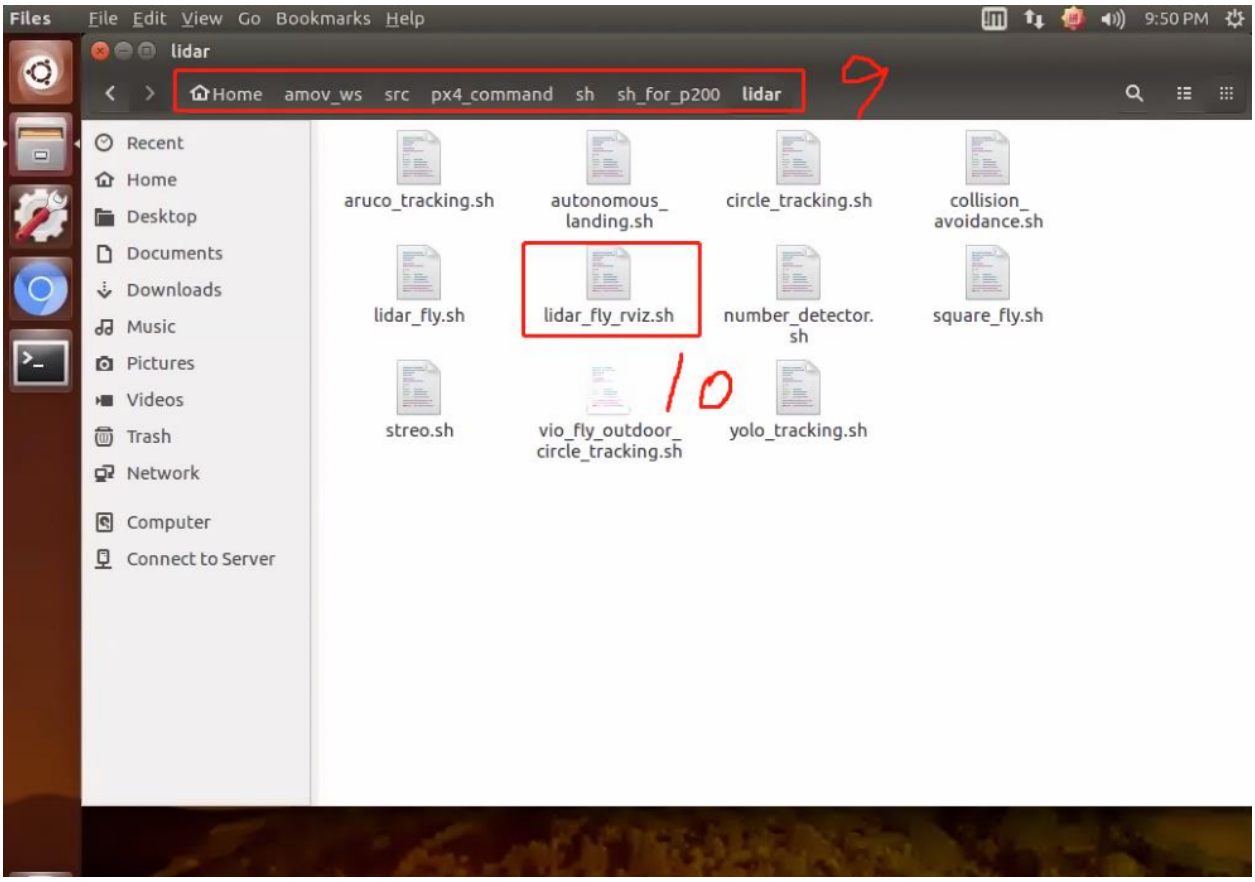


最后呈现的就是实时的地图数据了。

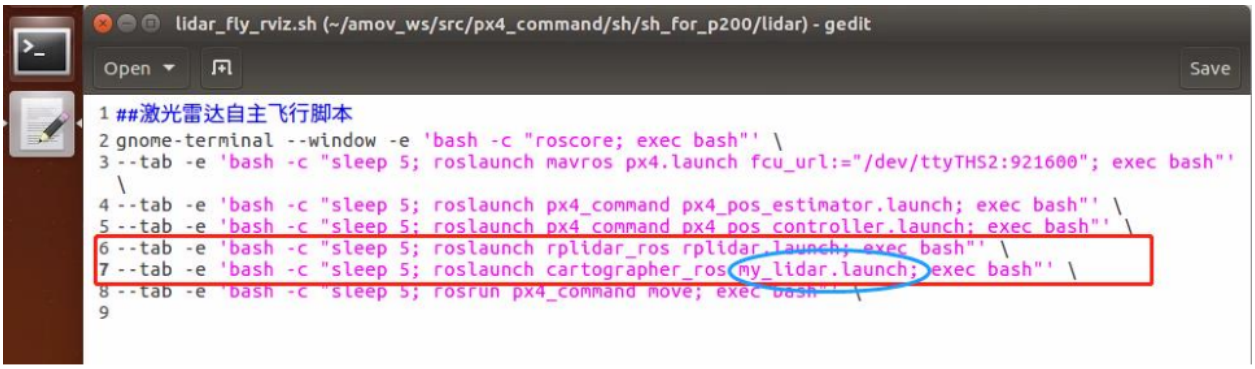


4. 在脚本中直接启动定位 + 建图

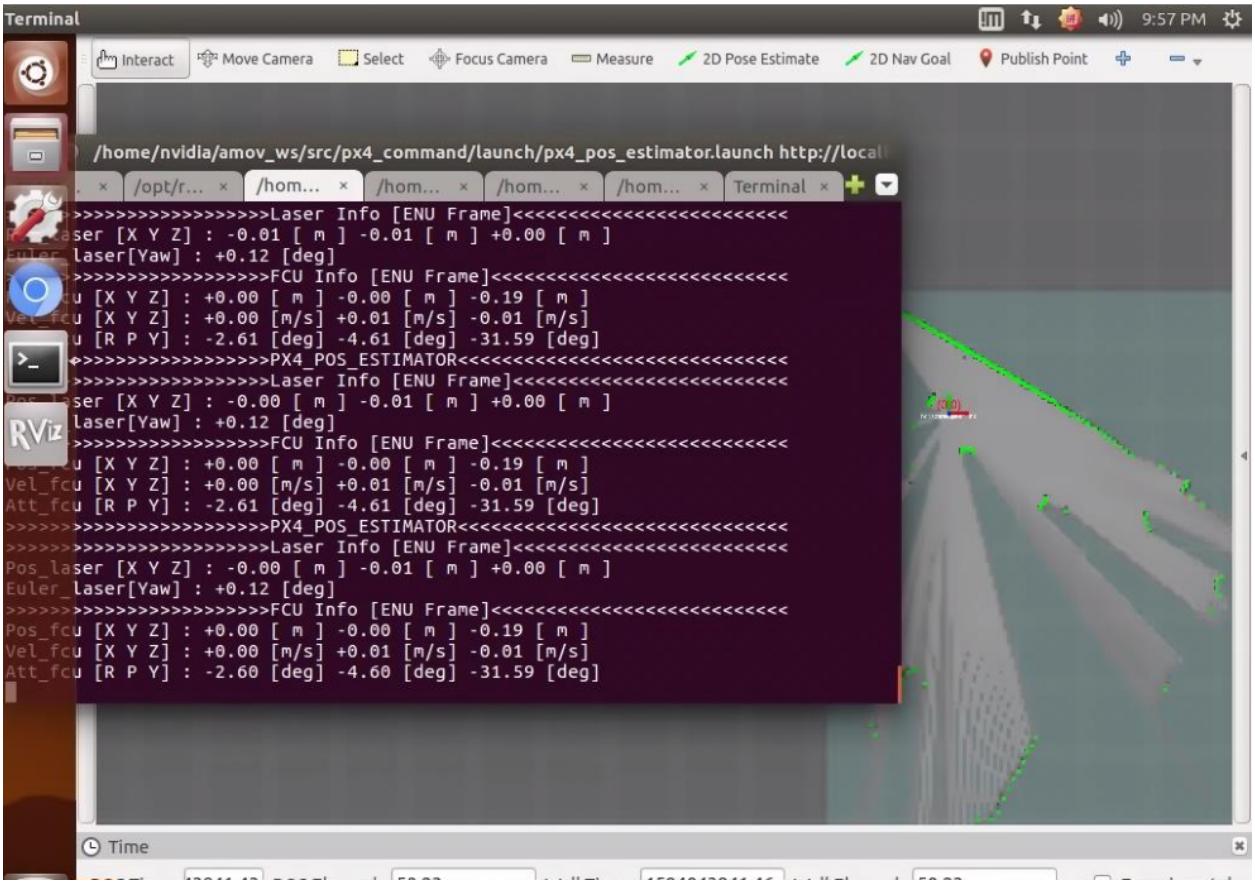
进入到 9 路径之下，拷贝 lidar_fly.sh ，并重命名为 lidar_fly_rviz.sh



脚本内容为如下：



启动该脚本，就可以在飞行中进行建图并查看建图效果了。



启动就完成了。

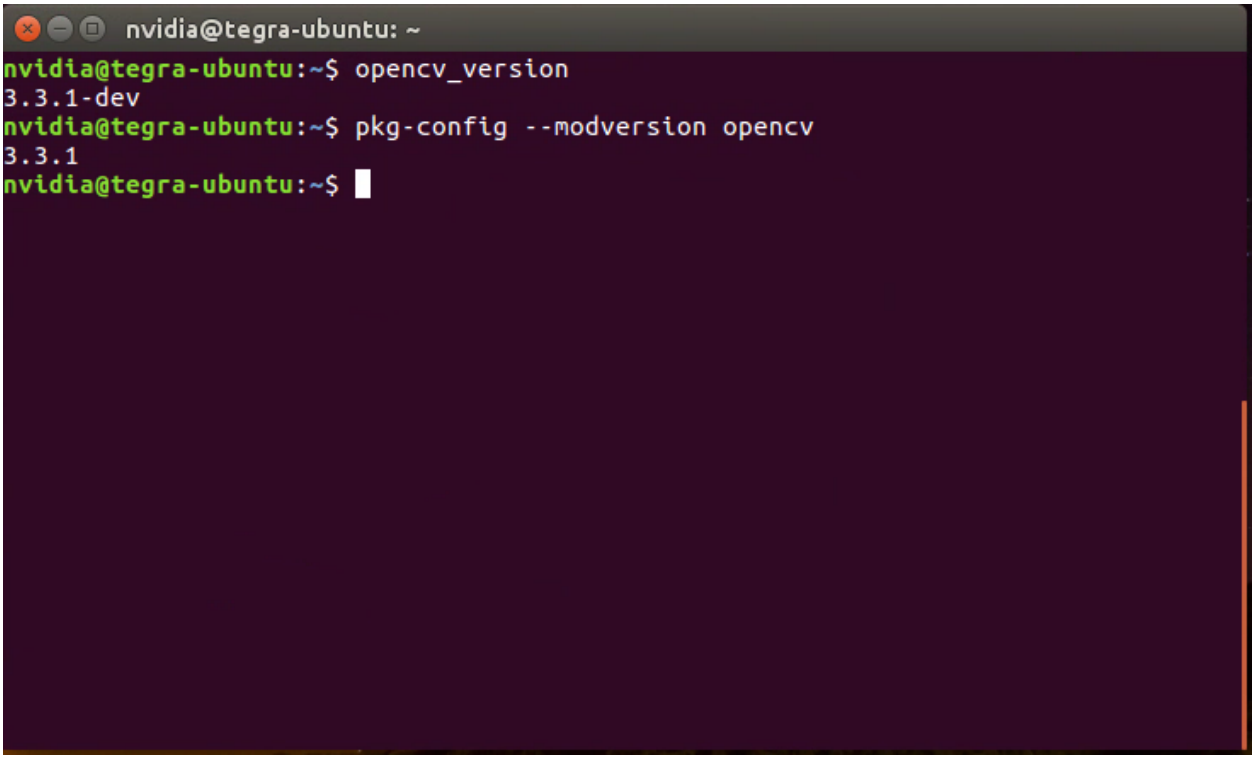
如果存在疑问，欢迎在我们的论坛 <https://bbs.amovlab.com/> 提问

3.yolo 通用目标检测

1、查看各个依赖环境是否正常

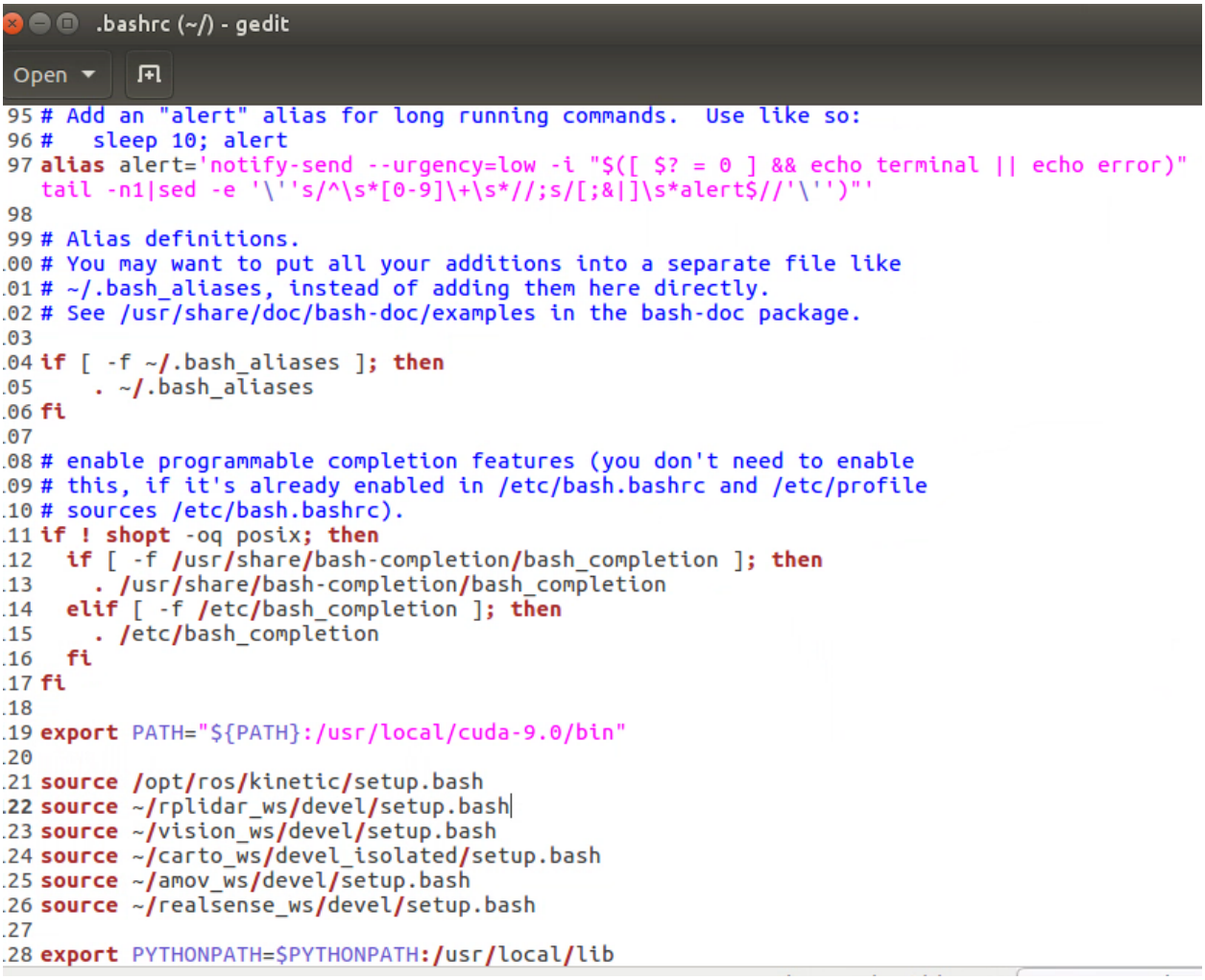
a、opencv

使用 `opencv_version` 或者 `pkg-config --modversion opencv` ，如下图所示，安装的为 opencv3.3.1



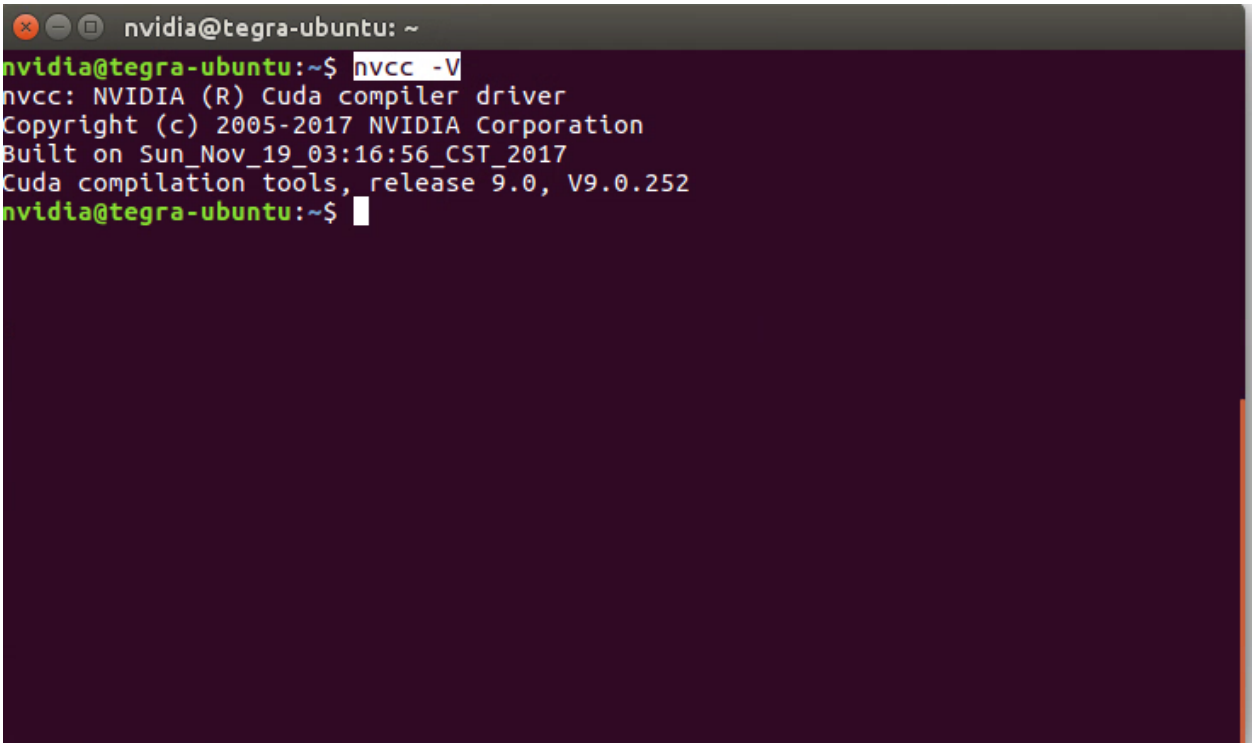
b、cuda 和 cudnn

cuda 和 cudnn 都已经安装成功，只是环境中没有把它使能。修改环境变量文件.bashrc 。修改之前的环境变量，添加如下 `export PATH="${PATH}:/usr/local/cuda-9.0/bin"` ，因为 cuda 的路径就在 /usr/local/cuda-9.0 。



.bashrc 基本上没有太多修改，和上图环境保持一直即可。

打开终端输入 `nvcc -V` ，如下图表示 cuda 是安装好的，并且可以使用。



使用 `dpkg -l | grep -i cuda` 可以查看 cuda 相关的安装情况，如下图可以看出，我们安装的是 cuda9.0+cudvnn7.1.5.14，说明 cuda 和 cudnn 都已经安装成功，并可以正常使用。

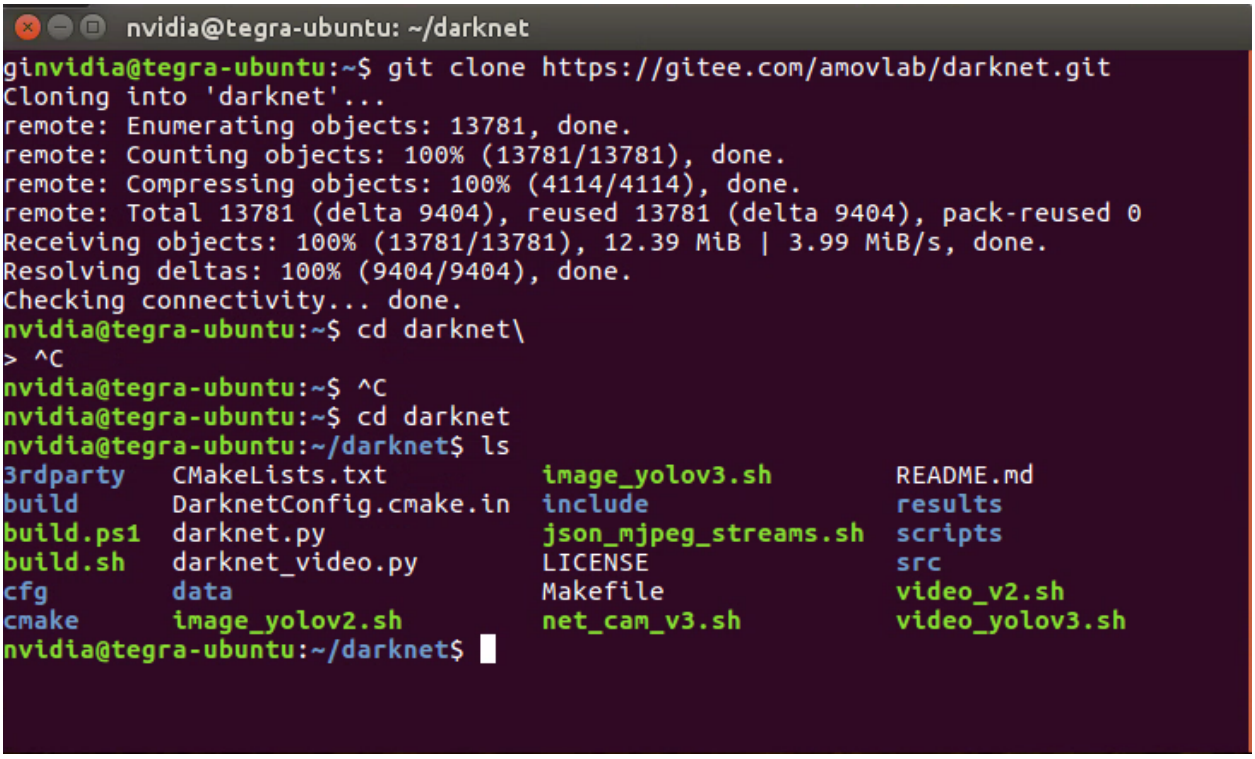
```
ii  cuda-libraries-dev-9-0                       9.0.252-1
m64         CUDA Libraries 9.0 development meta-package
ii  cuda-license-9-0                             9.0.252-1
m64         CUDA licenses
ii  cuda-misc-headers-9-0                        9.0.252-1
m64         CUDA miscellaneous headers
ii  cuda-npp-9-0                                 9.0.252-1
m64         NPP native runtime libraries
ii  cuda-npp-dev-9-0                             9.0.252-1
m64         NPP native dev links, headers
ii  cuda-nvgraph-9-0                             9.0.252-1
m64         NVGRAPH native runtime libraries
ii  cuda-nvgraph-dev-9-0                         9.0.252-1
m64         NVGRAPH native dev links, headers
ii  cuda-nvml-dev-9-0                            9.0.252-1
m64         NVML native dev links, headers
ii  cuda-nvrtc-9-0                               9.0.252-1
m64         NVRTC native runtime libraries
ii  cuda-nvrtc-dev-9-0                           9.0.252-1
m64         NVRTC native dev links, headers
ii  cuda-repo-l4t-9-0-local                      9.0.252-1
m64         cuda repository configuration files
ii  cuda-samples-9-0                             9.0.252-1
m64         CUDA example applications
ii  cuda-toolkit-9-0                             9.0.252-1
m64         CUDA Toolkit 9.0 meta-package
ii  libcudnn7                                    7.1.5.14-1+cuda9.0
m64         cuDNN runtime libraries
ii  libcudnn7-dev                                7.1.5.14-1+cuda9.0
m64         cuDNN development libraries and headers
ii  libcudnn7-doc                                7.1.5.14-1+cuda9.0
m64         cuDNN documents and samples
ii  libgie-dev                                   4.1.3-1+cuda9.0
m64         Transitional package
ii  libnvinfer-dev                               4.1.3-1+cuda9.0
m64         TensorRT development libraries and headers
ii  libnvinfer-samples                           4.1.3-1+cuda9.0
m64         TensorRT samples and documentation
ii  libnvinfer4                                  4.1.3-1+cuda9.0
m64         TensorRT runtime libraries
ii  tensorrt                                     4.0.2.0-1+cuda9.0
m64         Meta package of TensorRT
nvidia@tegra-ubuntu:~$ dpkg -l | grep -i cuda
```

2、下载 darknet 源码

源码地址：github https://github.com/amov-lab/darknet.git

码云 https://gitee.com/amovlab/darknet.git

jetson tx2 Ubuntu16.04 系统里面本身有个 darknet 包，请重新命名或者删除掉，然后下在上面的源码，建议使用码云，速度可能比较快一点。如果给出的码云地址无法克隆，可以将 github 的源码备份到自己的码云库里面，自行下载。

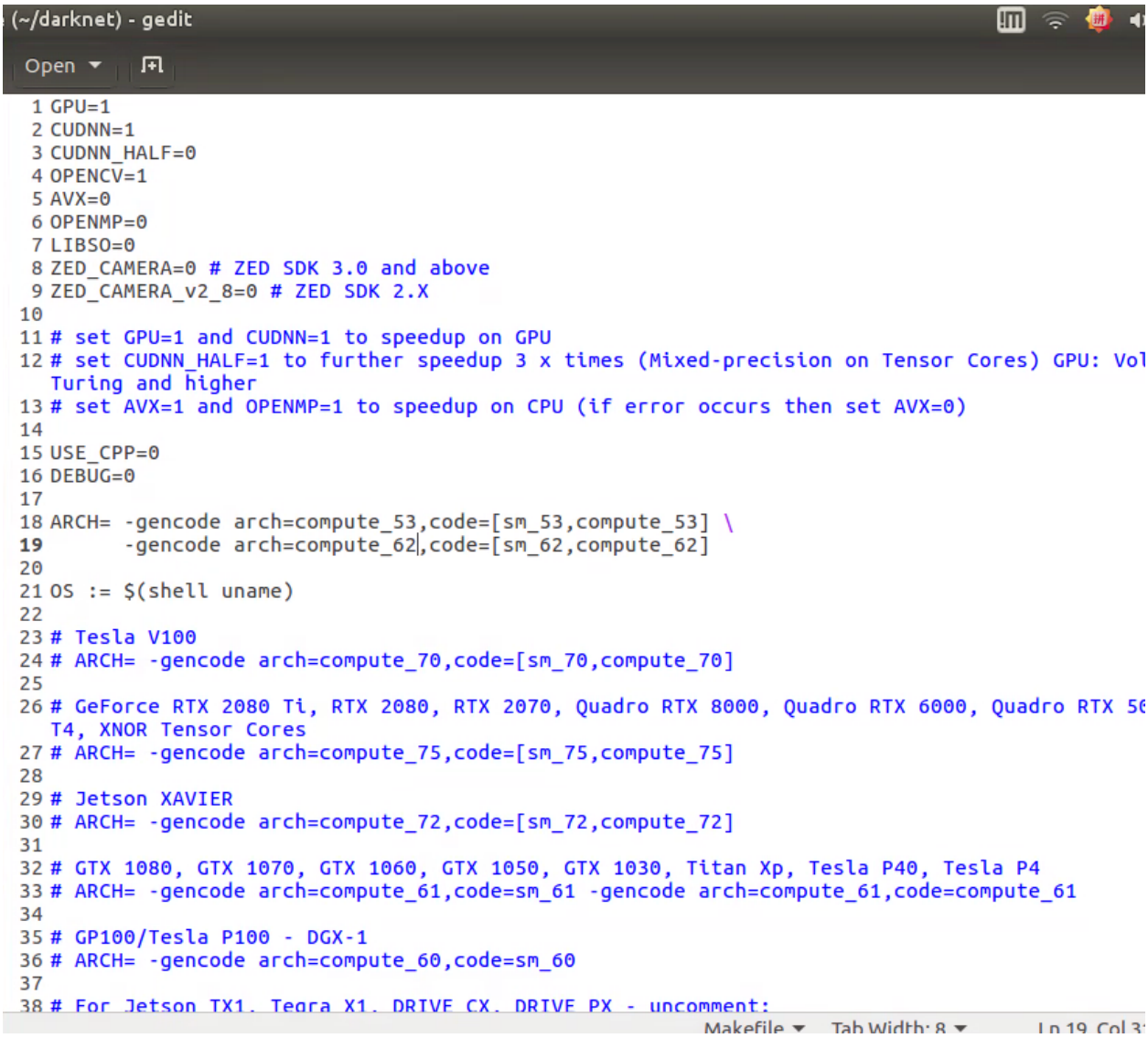


3、修改并编译

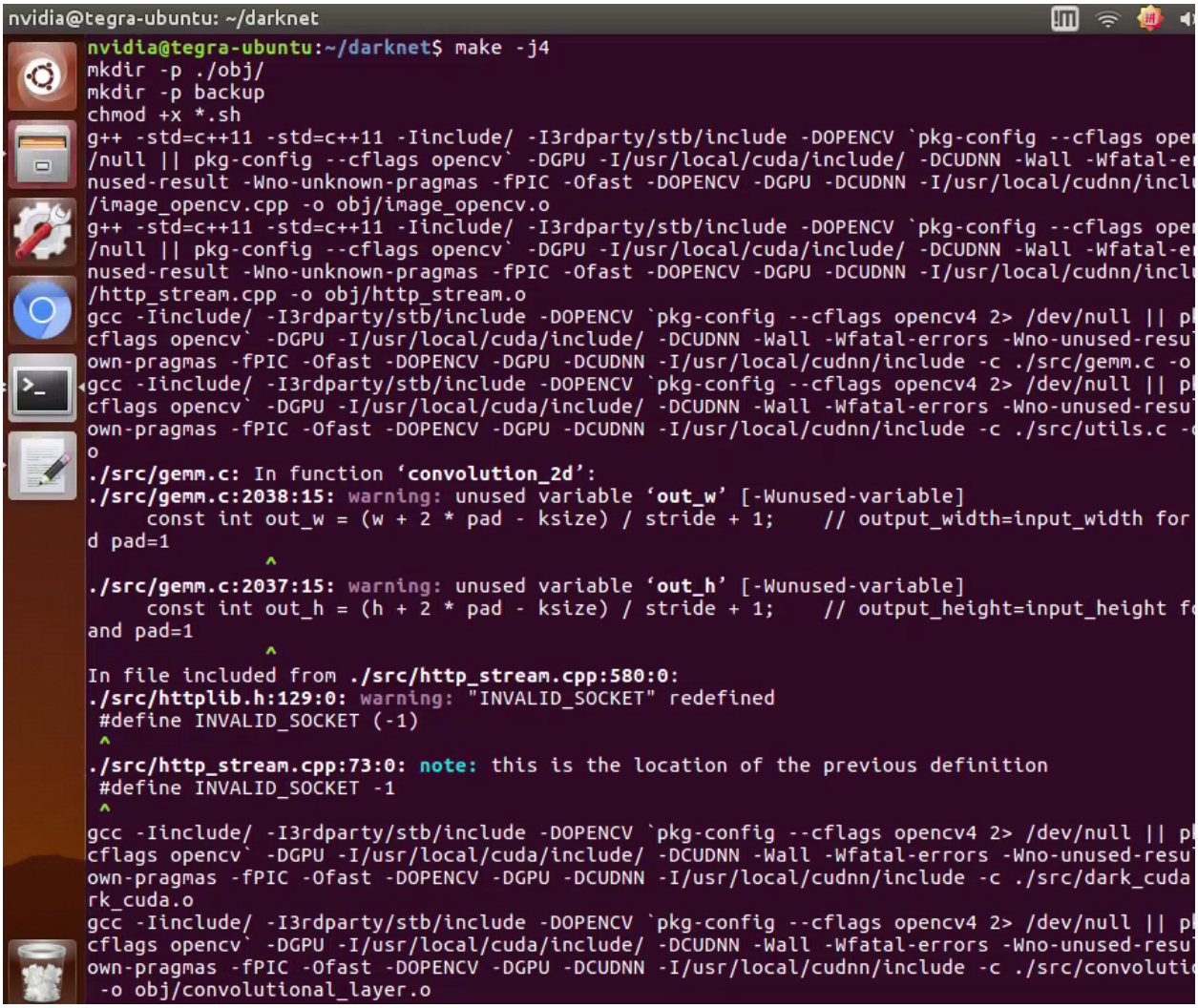
需要修改 *Makefile* 文件，修改的内容如下：

```
nvidia@tegra-ubuntu: ~/darknet
nvidia@tegra-ubuntu:~$ cd darknet
nvidia@tegra-ubuntu:~/darknet$ git diff
diff --git a/Makefile b/Makefile
index 1d9d442..aa16cc3 100644
--- a/Makefile
+++ b/Makefile
@@ -1,7 +1,7 @@
-GPU=0
-CUDNN=0
+GPU=1
+CUDNN=1
 CUDNN_HALF=0
-OPENCV=0
+OPENCV=1
 AVX=0
 OPENMP=0
 LIBSO=0
@@ -15,11 +15,8 @@ ZED_CAMERA_v2_8=0 # ZED SDK 2.X
 USE_CPP=0
 DEBUG=0

-ARCH= -gencode arch=compute_30,code=sm_30 \
-      -gencode arch=compute_35,code=sm_35 \
-      -gencode arch=compute_50,code=[sm_50,compute_50] \
-      -gencode arch=compute_52,code=[sm_52,compute_52] \
-            -gencode arch=compute_61,code=[sm_61,compute_61]
+ARCH= -gencode arch=compute_53,code=[sm_53,compute_53] \
+      -gencode arch=compute_62,code=[sm_62,compute_62]

 OS := $(shell uname)

nvidia@tegra-ubuntu:~/darknet$
```

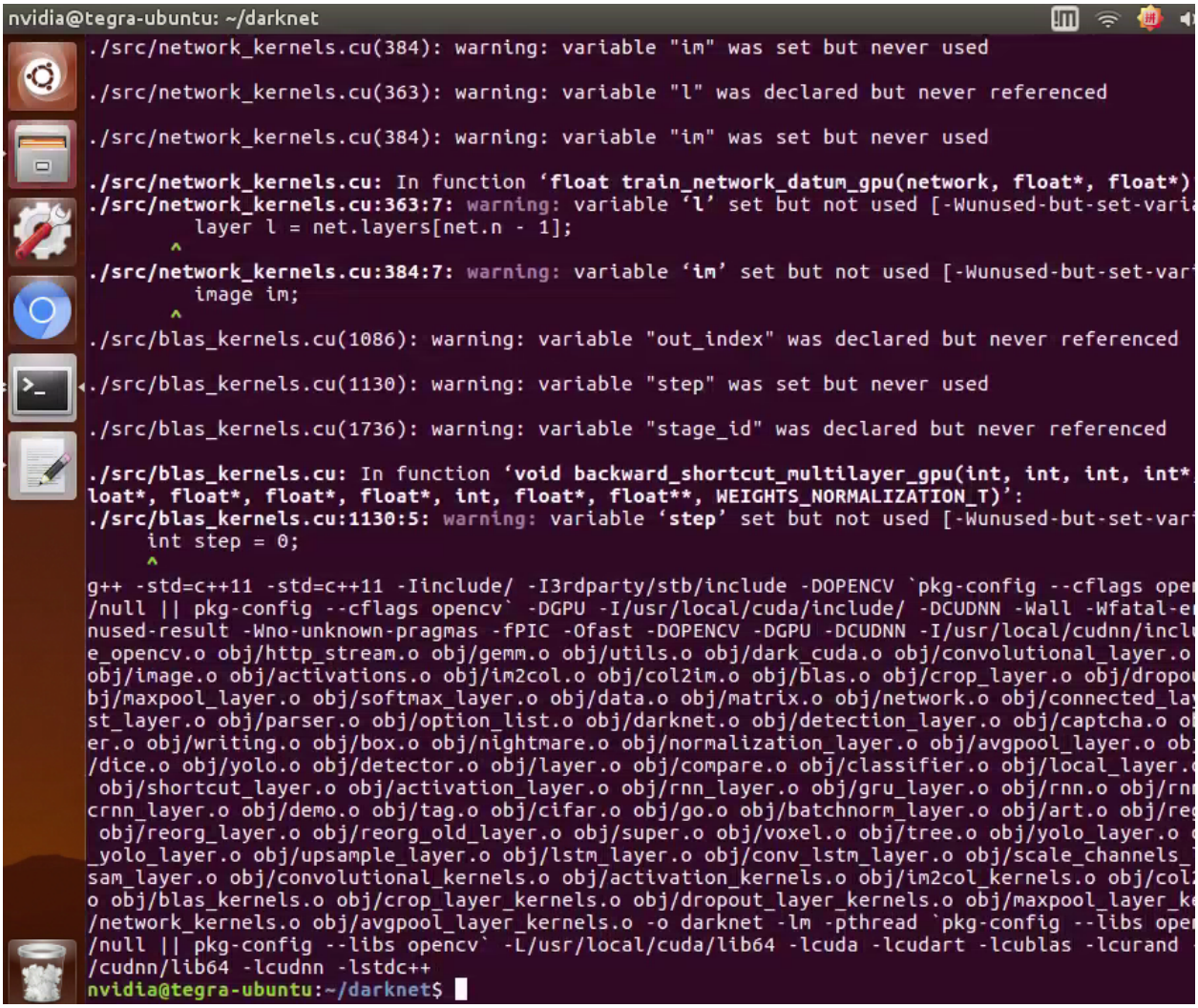
用文本打开修改，修改和下图所示一致



然后执行 `make -j4` ，等待编译



编译完成之后，如下图所示



4、使用 darknet

a、下载 weights 文件

COCO data set (Yolo v2):

wget http://pjreddie.com/media/files/yolov2.weights

wget http://pjreddie.com/media/files/yolov2-tiny.weights

VOC data set (Yolo v2):

wget http://pjreddie.com/media/files/yolov2-voc.weights

wget http://pjreddie.com/media/files/yolov2-tiny-voc.weights

Yolo v3:

wget http://pjreddie.com/media/files/yolov3.weights

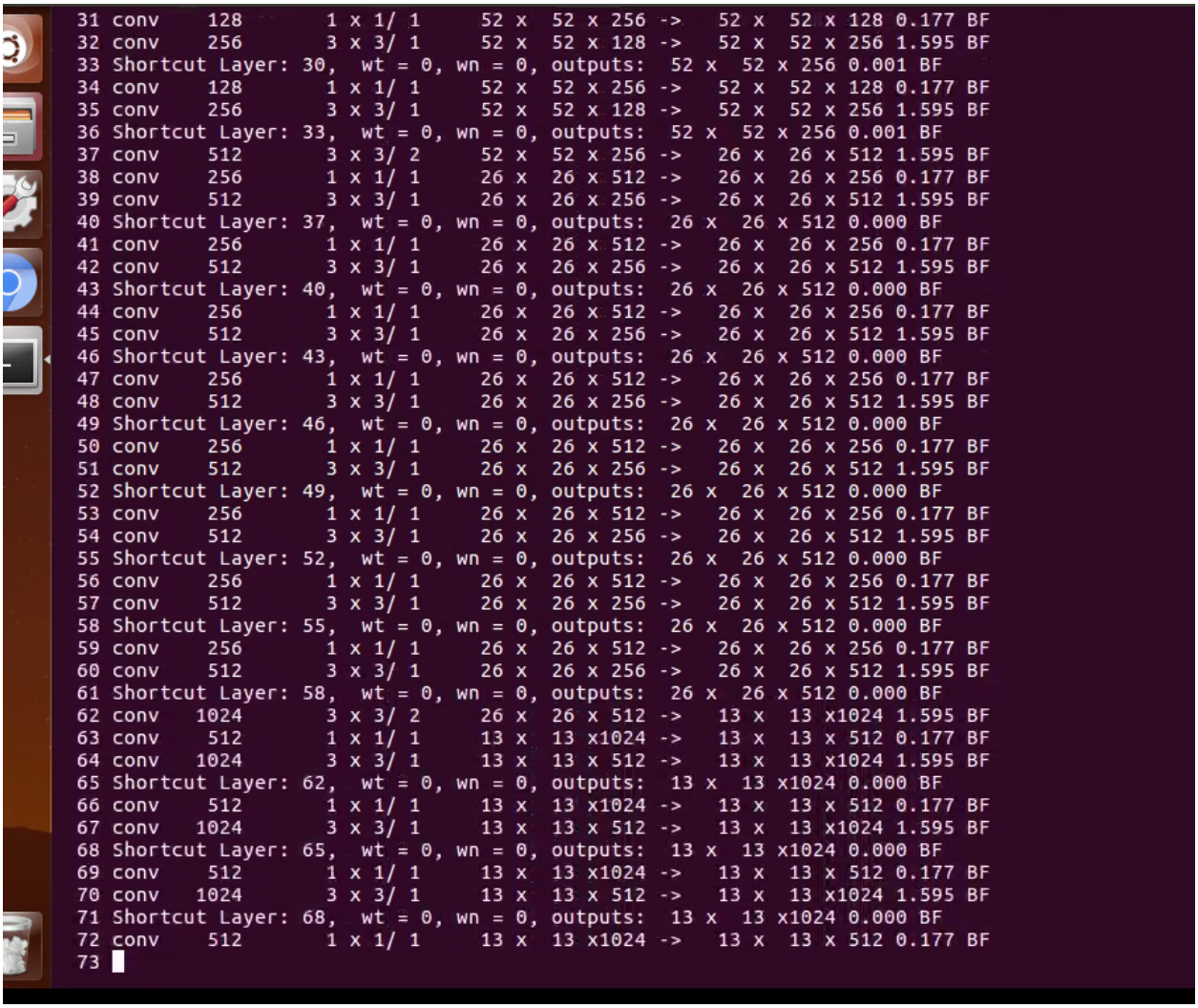
wget http://pjreddie.com/media/files/yolov3-voc.weights

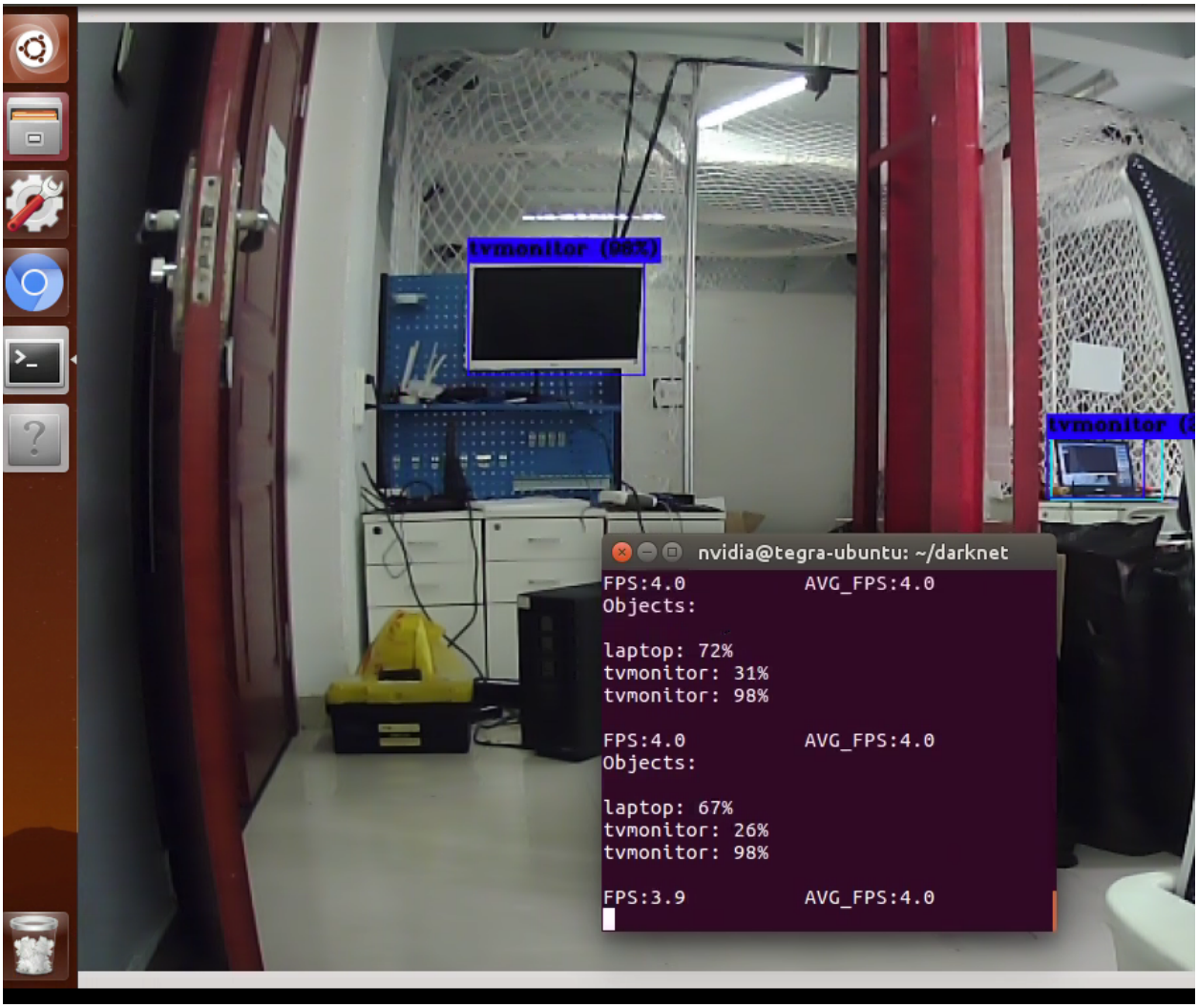
将下载好的 weights 文件放到 darknet 目录之下。

b、使用单目摄像头进行 yolo 检测

进入到 *darknet* 目录下，执行 `./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights /dev/video0`

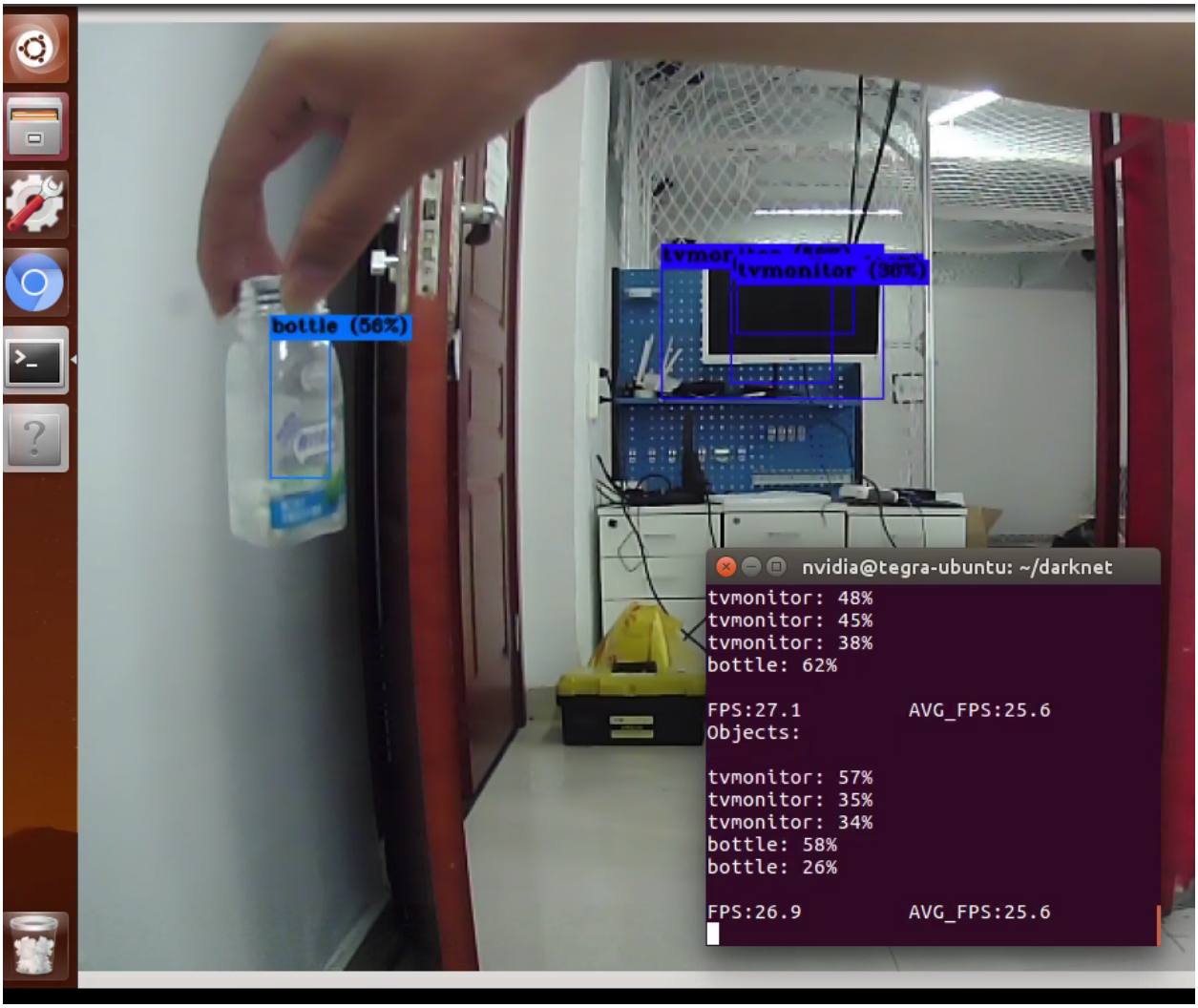
其中 */dev/video0* 就是单目摄像头的检测到的设备驱动。





yolov3 的帧率只有 4 左右。如上图所示

yolov3-tiny 的帧率基本在 23 左右，如下图



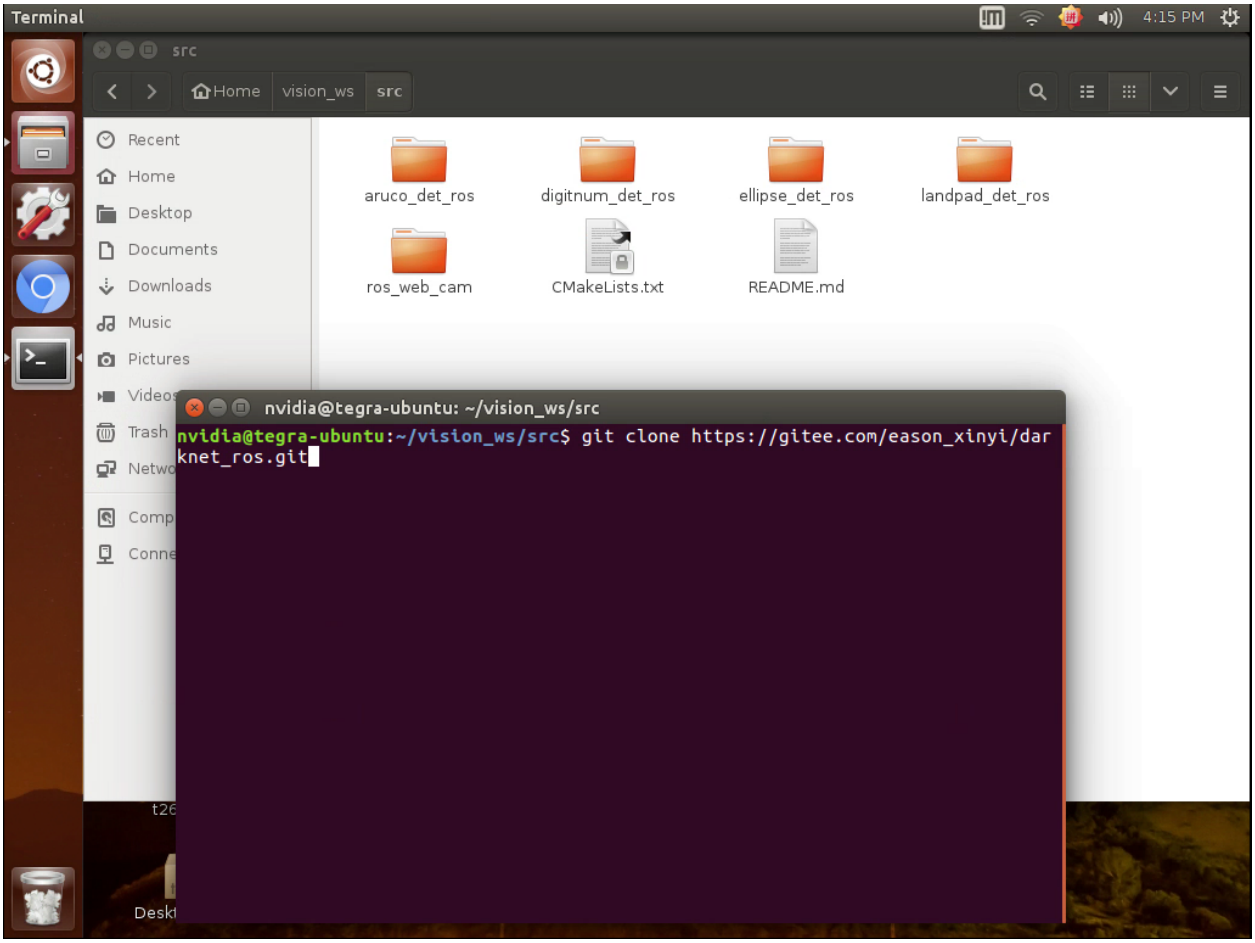
5、使用 darknet_ros

a、下载 darknet_ros 源码

码云地址：https://gitee.com/eason_xinyi/darknet_ros.git

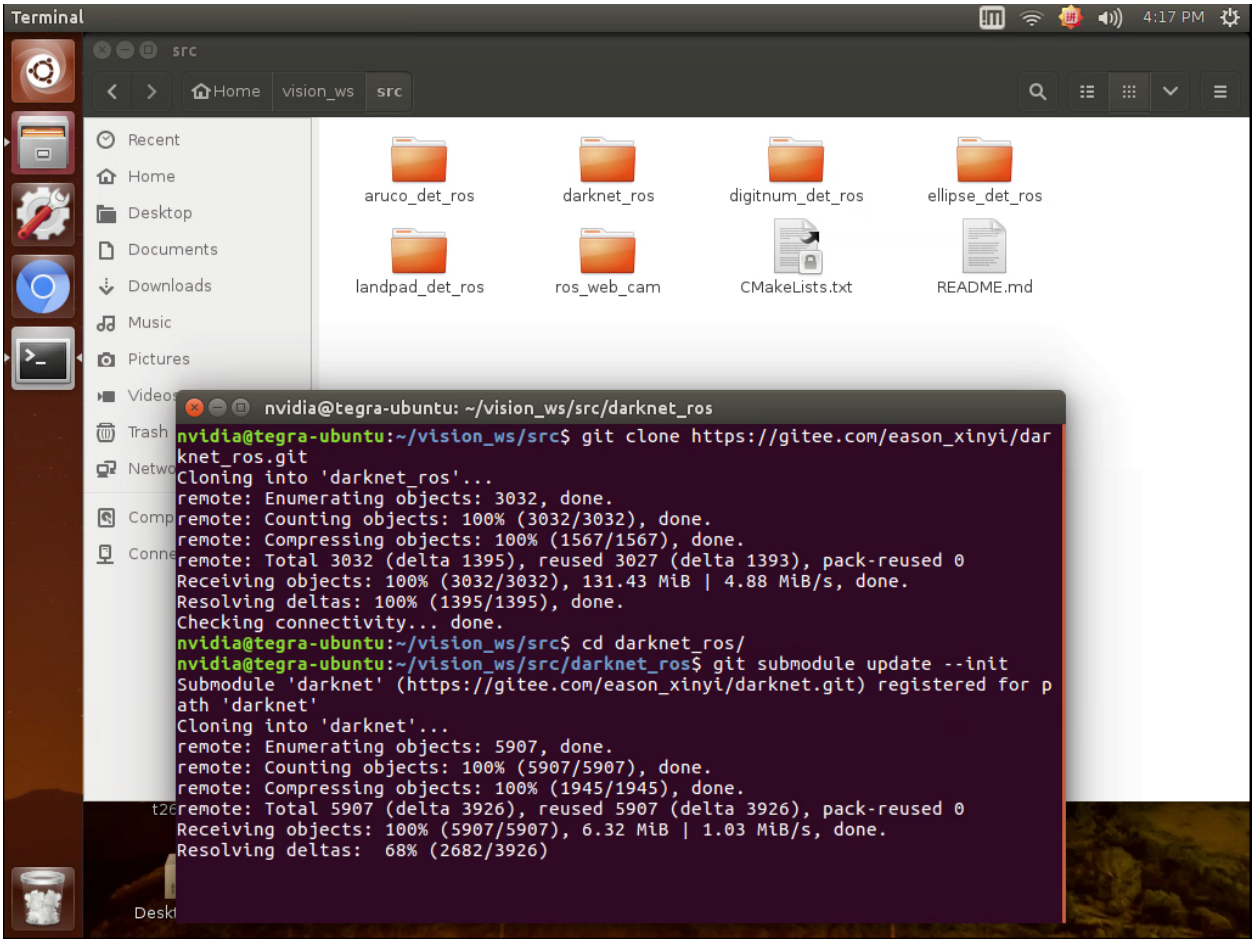
删除 *TX2* 中 *vision_ws/src* 下面的 darkent_ros 功能包。从码云上面地址下载 darkent_ros 源码并更新子模块。执行如下指令：`git clone https://gitee.com/eason_xinyi/darknet_ros.git`

如下图所示：



下载完源码并进行更新，执行指令：`git submodule update --init`

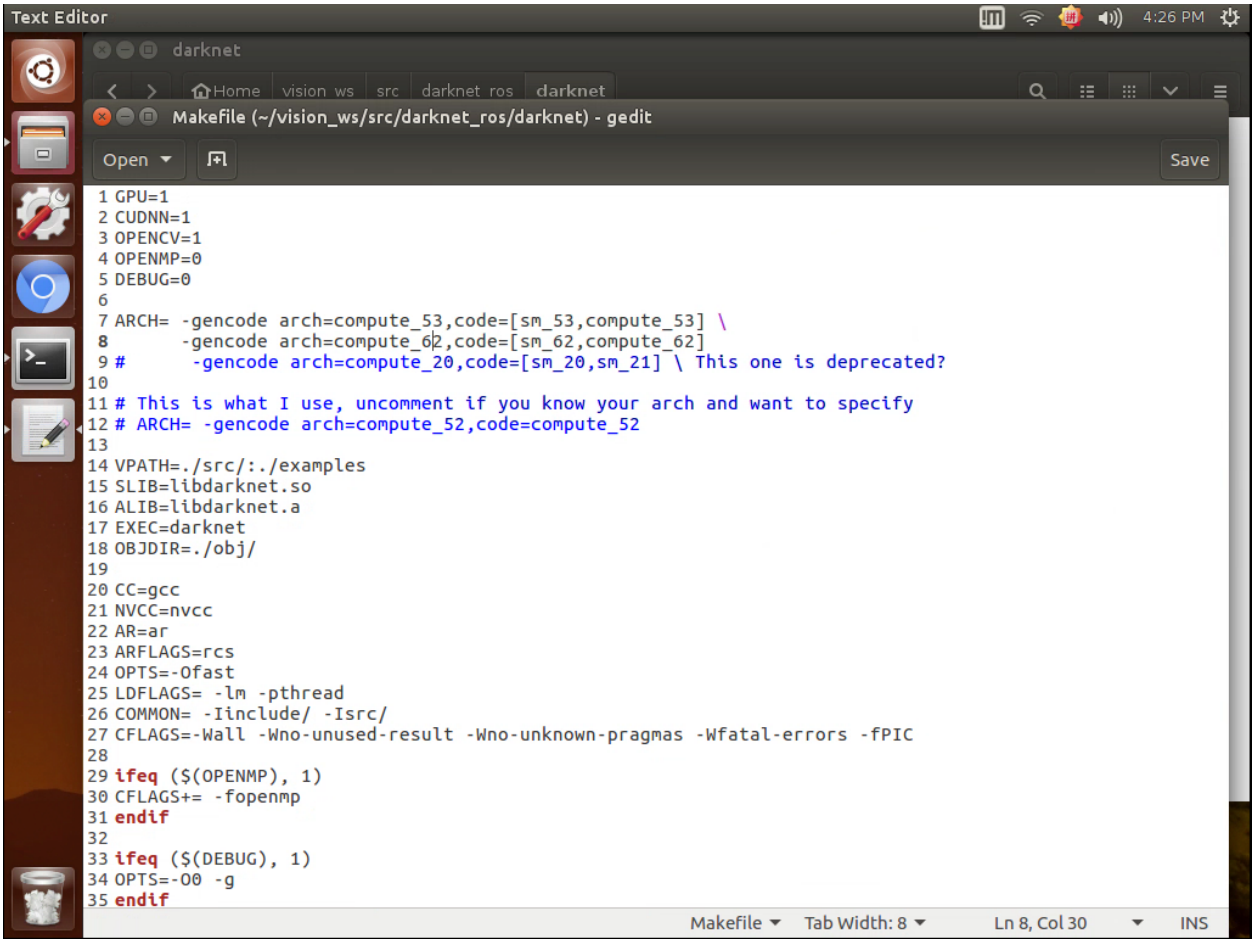
如下图所示：



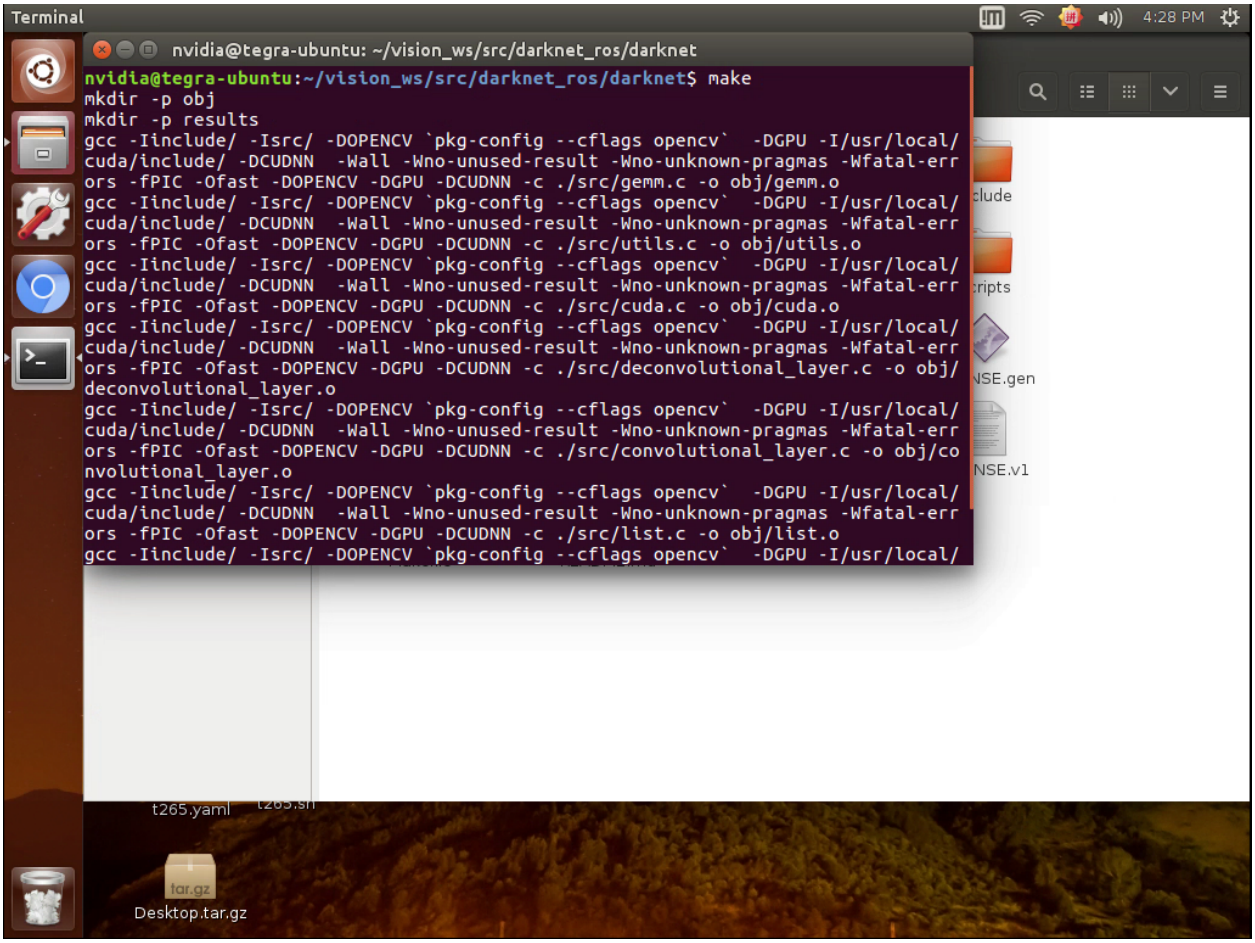
b、修改配置文件并编译

darknet 的 Makefile 文件修改并编译

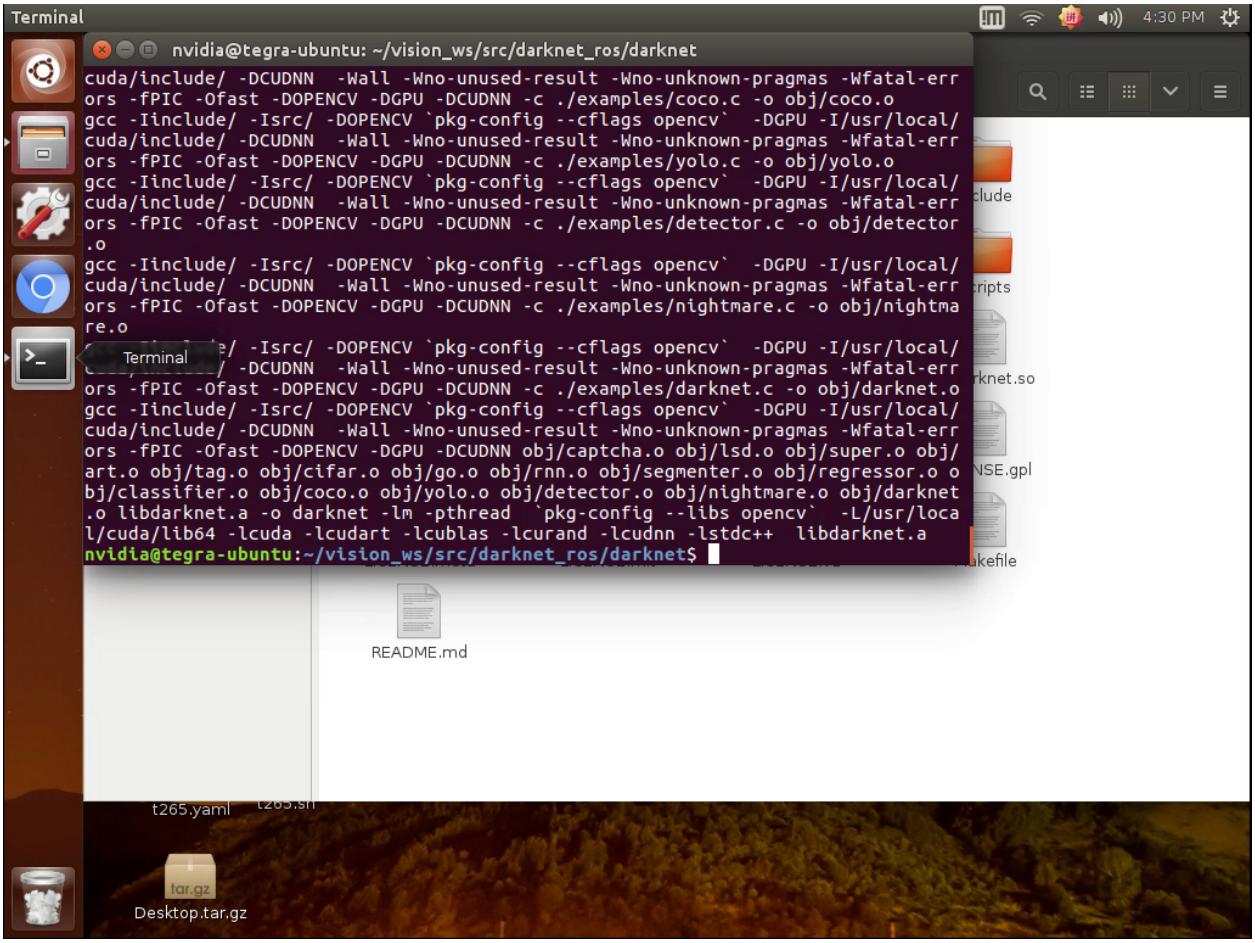
GPU=1，CUDNN=1，OPENCV=1，系统架构保留 53 和 62，修改之后如下图所示：



然后进行 make 编译，如下图：

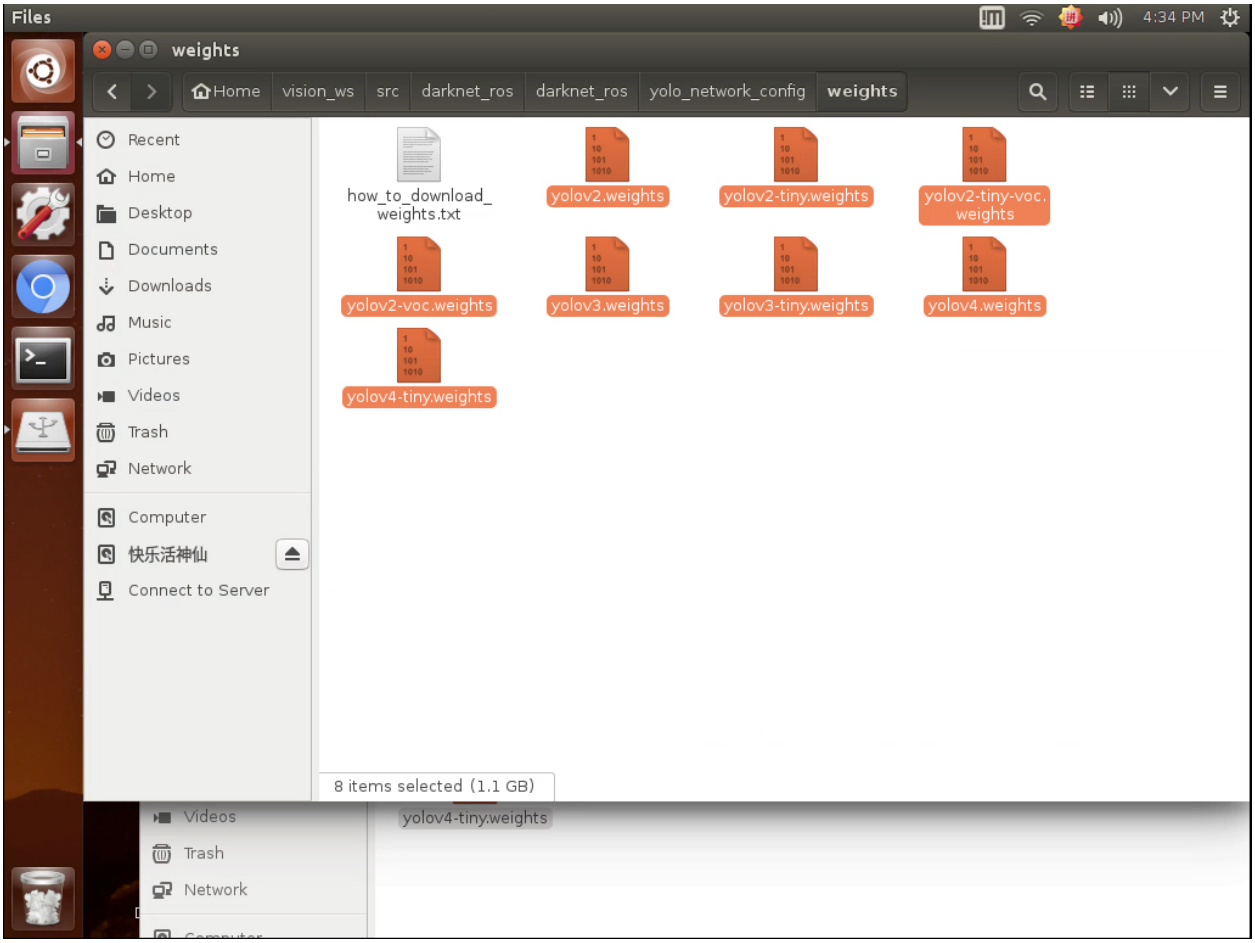

Terminal
4:28 PM
nvidia@tegra-ubuntu: ~/vision_ws/src/darknet_ros/darknet
nvidia@tegra-ubuntu:~/vision_ws/src/darknet_ros/darknet$ make
mkdir -p obj
mkdir -p results
gcc -Iinclude/ -Isrc/ -DOPENCV `pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/include/ -DCUDNN -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -DGPU -DCUDNN -c ./src/gemm.c -o obj/gemm.o
gcc -Iinclude/ -Isrc/ -DOPENCV `pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/include/ -DCUDNN -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -DGPU -DCUDNN -c ./src/utils.c -o obj/utils.o
gcc -Iinclude/ -Isrc/ -DOPENCV `pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/include/ -DCUDNN -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -DGPU -DCUDNN -c ./src/cuda.c -o obj/cuda.o
gcc -Iinclude/ -Isrc/ -DOPENCV `pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/include/ -DCUDNN -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -DGPU -DCUDNN -c ./src/deconvolutional_layer.c -o obj/deconvolutional_layer.o
gcc -Iinclude/ -Isrc/ -DOPENCV `pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/include/ -DCUDNN -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -DGPU -DCUDNN -c ./src/convolutional_layer.c -o obj/convolutional_layer.o
gcc -Iinclude/ -Isrc/ -DOPENCV `pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/include/ -DCUDNN -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -DGPU -DCUDNN -c ./src/list.c -o obj/list.o
gcc -Iinclude/ -Isrc/ -DOPENCV `pkg-config --cflags opencv` -DGPU -I/usr/local/
t265.yaml
t265.sh
tar.gz
Desktop.tar.gz

等待编译完成，如下：

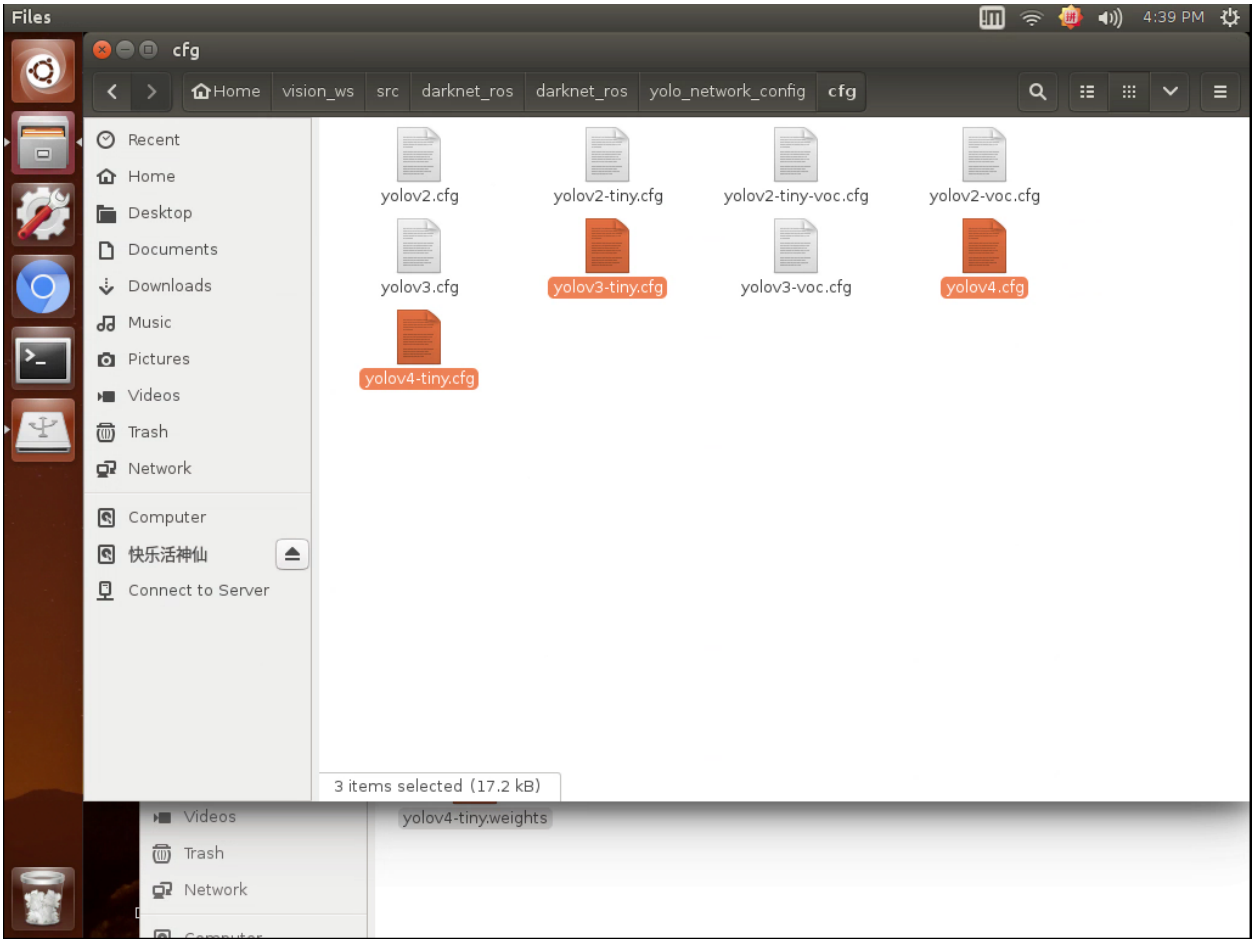


darknet_ros 文件配置

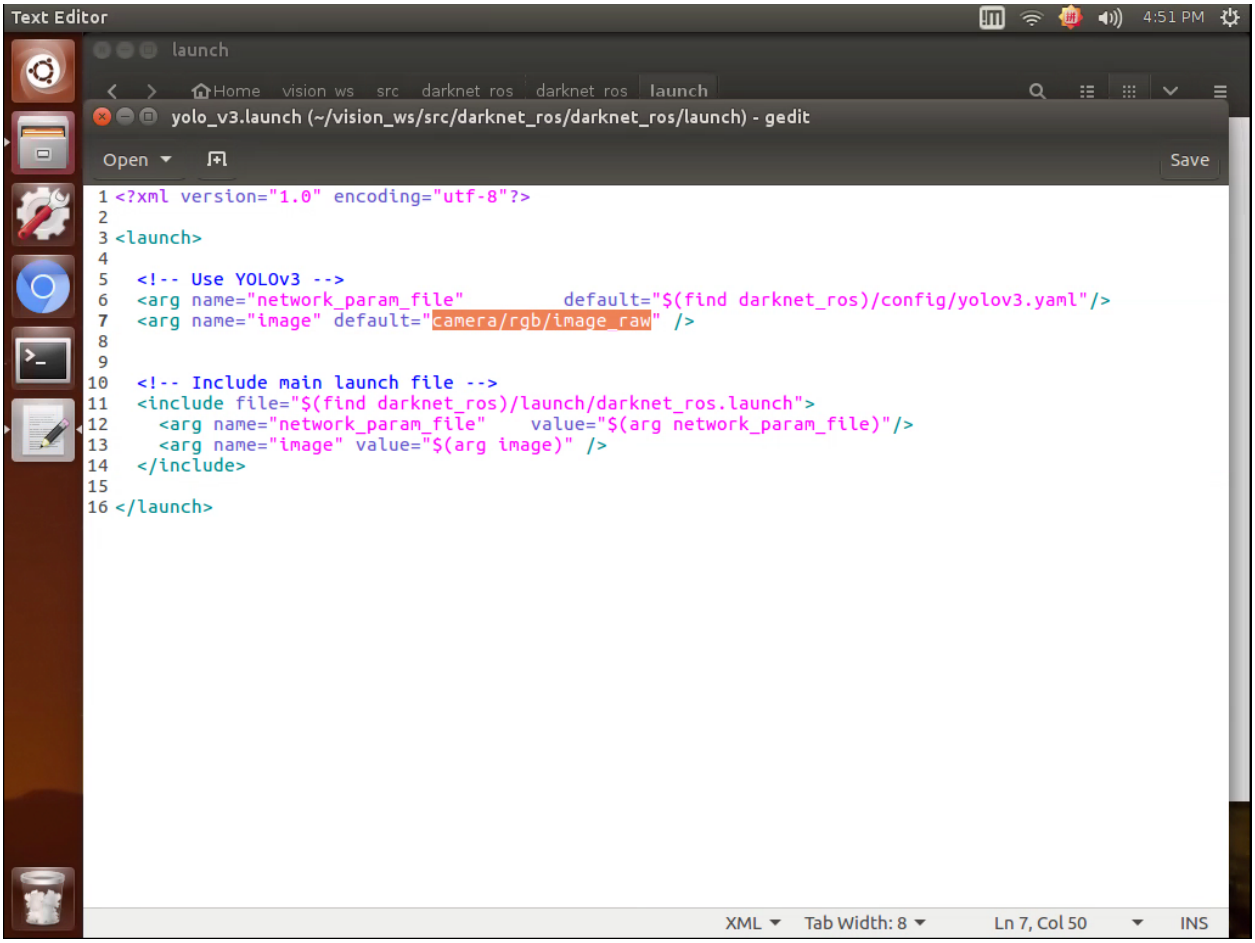
添加 weights 包到 darknet_ros/yolo_network_config/weights 。提前下载好相关的 weights 包存放到 U 盘之内，然后拷贝到该路径之下。如下图:



添加 cfg 文件到 darknet_ros/yolo_network_config/cfg 。从 home 下面的 darknet/cfg 路径下拷贝需要的 cfg 文件到该目录路径之下。

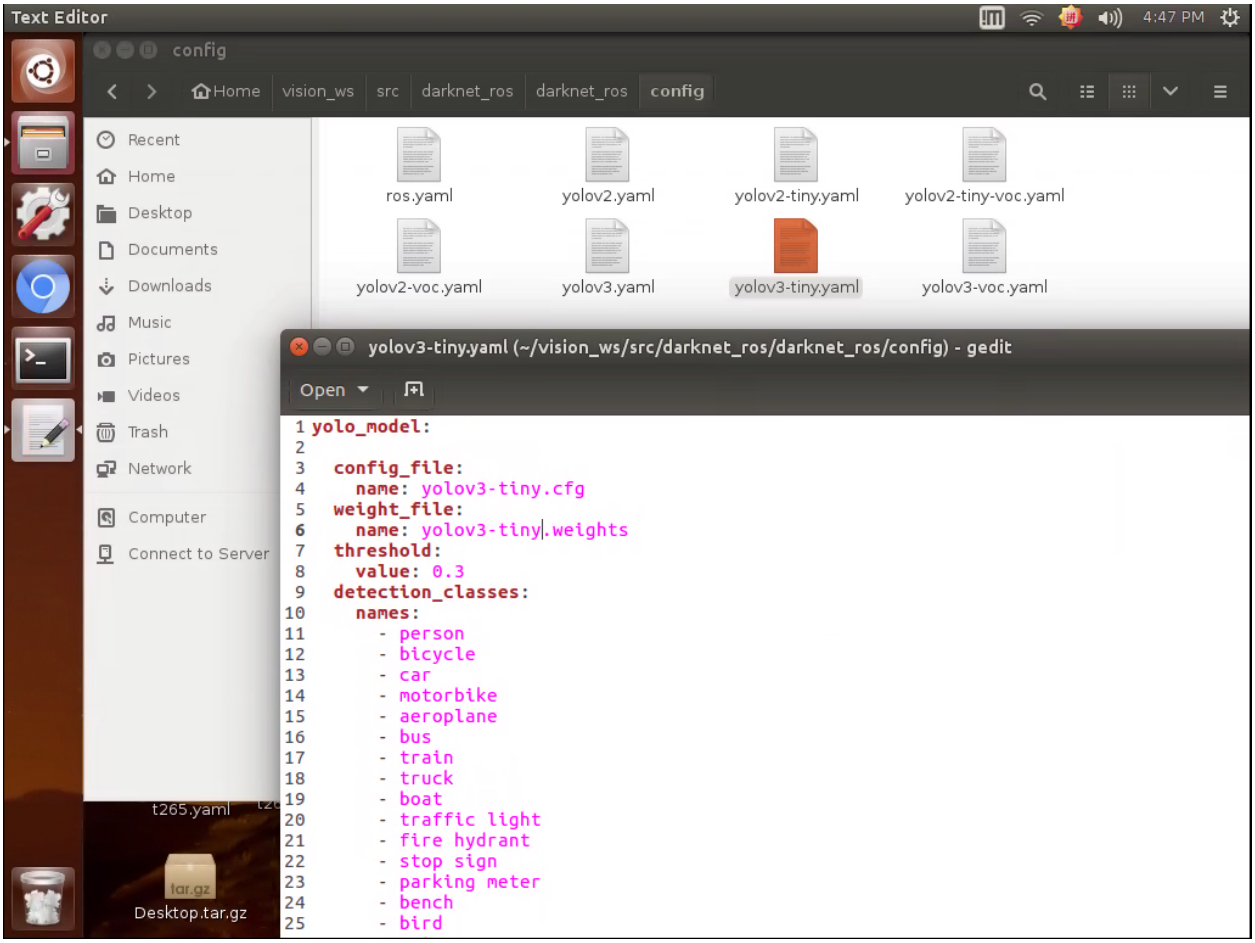


修改 yolov3 的 launch 文件，只要修改订阅相机图像的 topic，修改为单目摄像头发布的 topic。如下图：

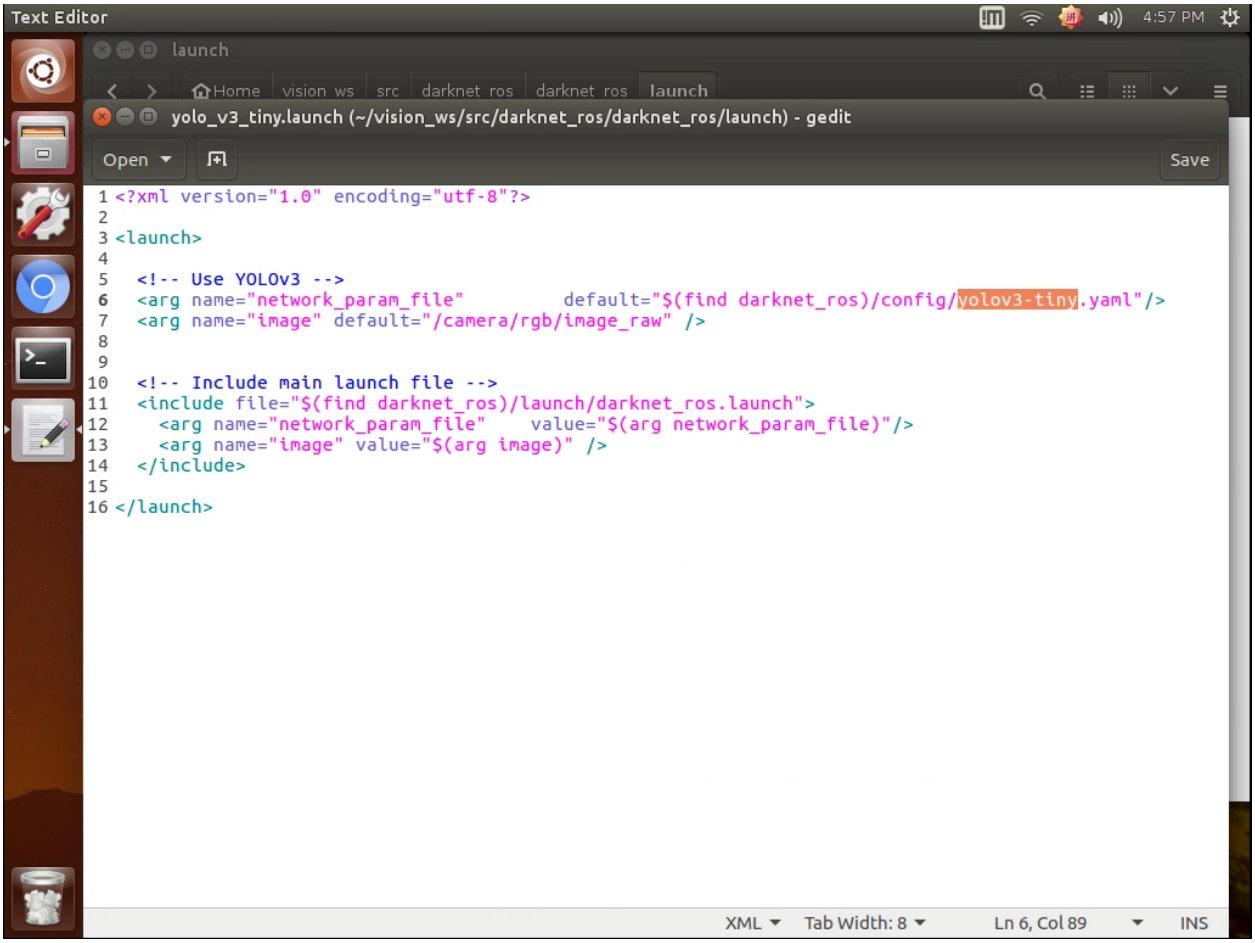


添加 yolov3-tiny 。默认下载的只有 yolov3，但实际上使用 yolov3 帧率很低，所以提前配置下 yolov3-tiny 。

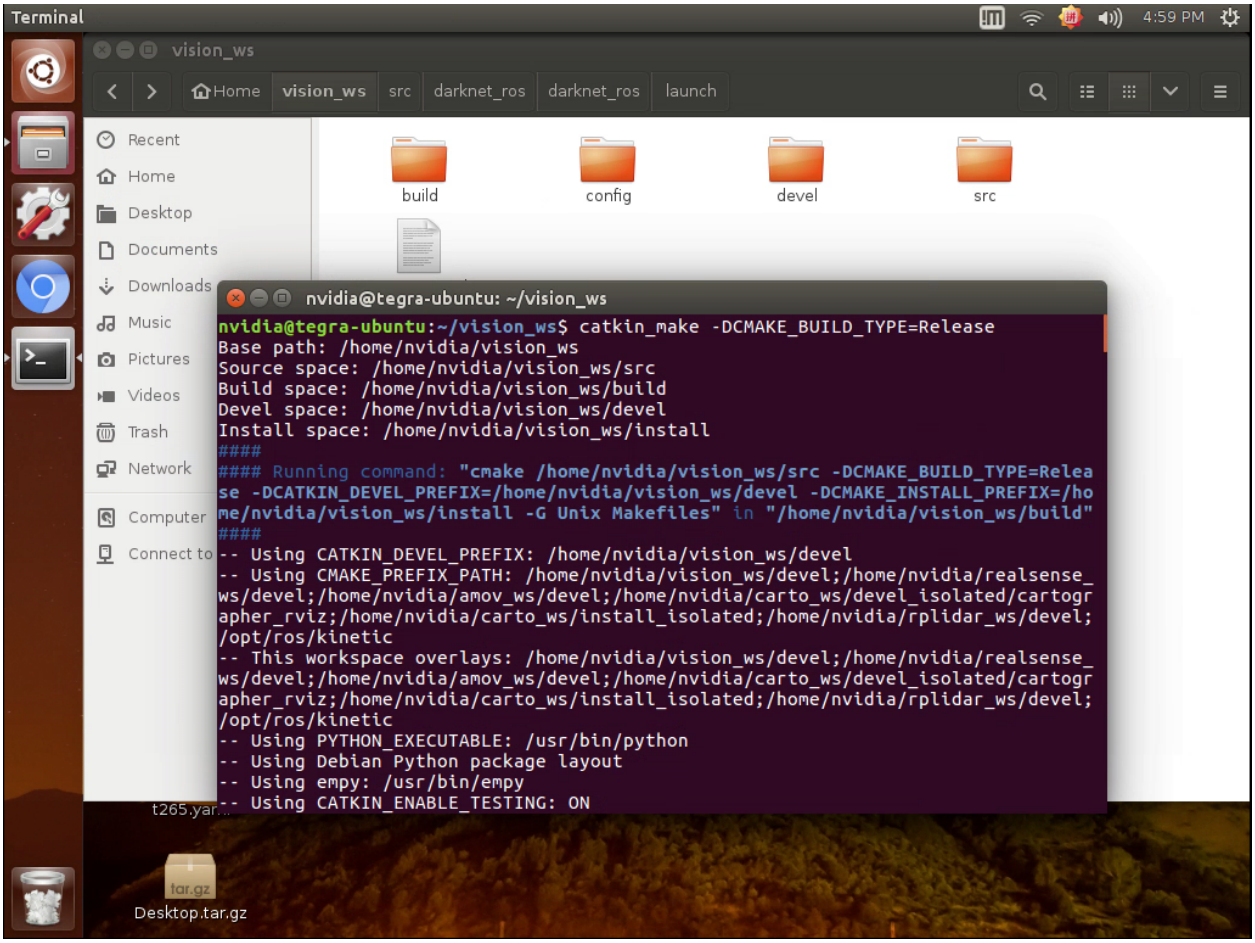
修改 config 文件，添加 yolov3-tiny.yaml 文件。修改方式只是修改 config 文件和 weights 文件，如下图：



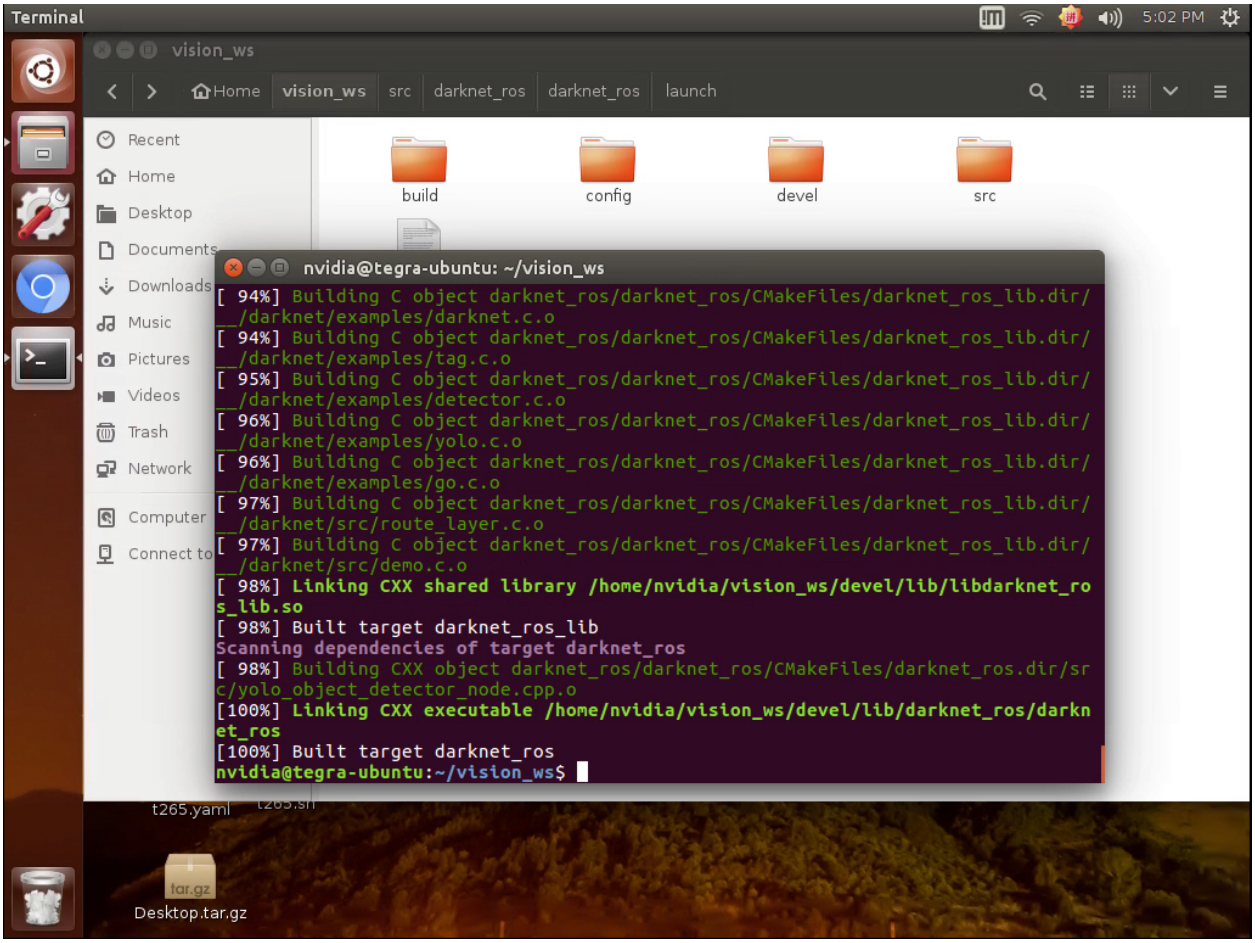
添加 yolov3-tiny 的 launch 文件，可直接拷贝 yolov3 的 launch，需要修改他的参数加载的配置文件，将 yolov3 改为 yolov3-tiny。如下图所示：



然后编译整个 vision_ws 功能包，执行 `catkin_make -DCMAKE_BUILD_TYPE=Release`



等待编译完成，如下图所示：



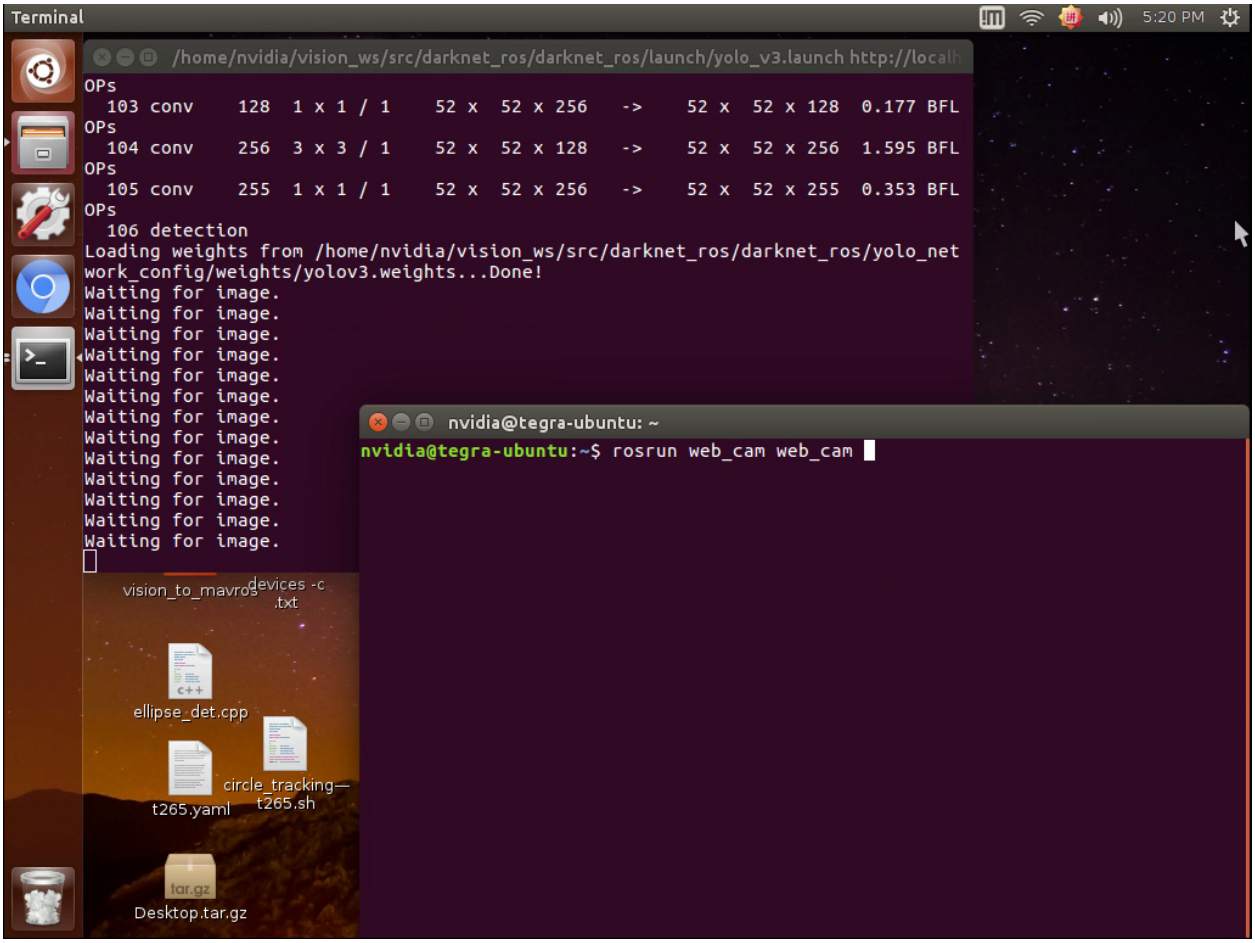
c、单目 darknet_ros 使用

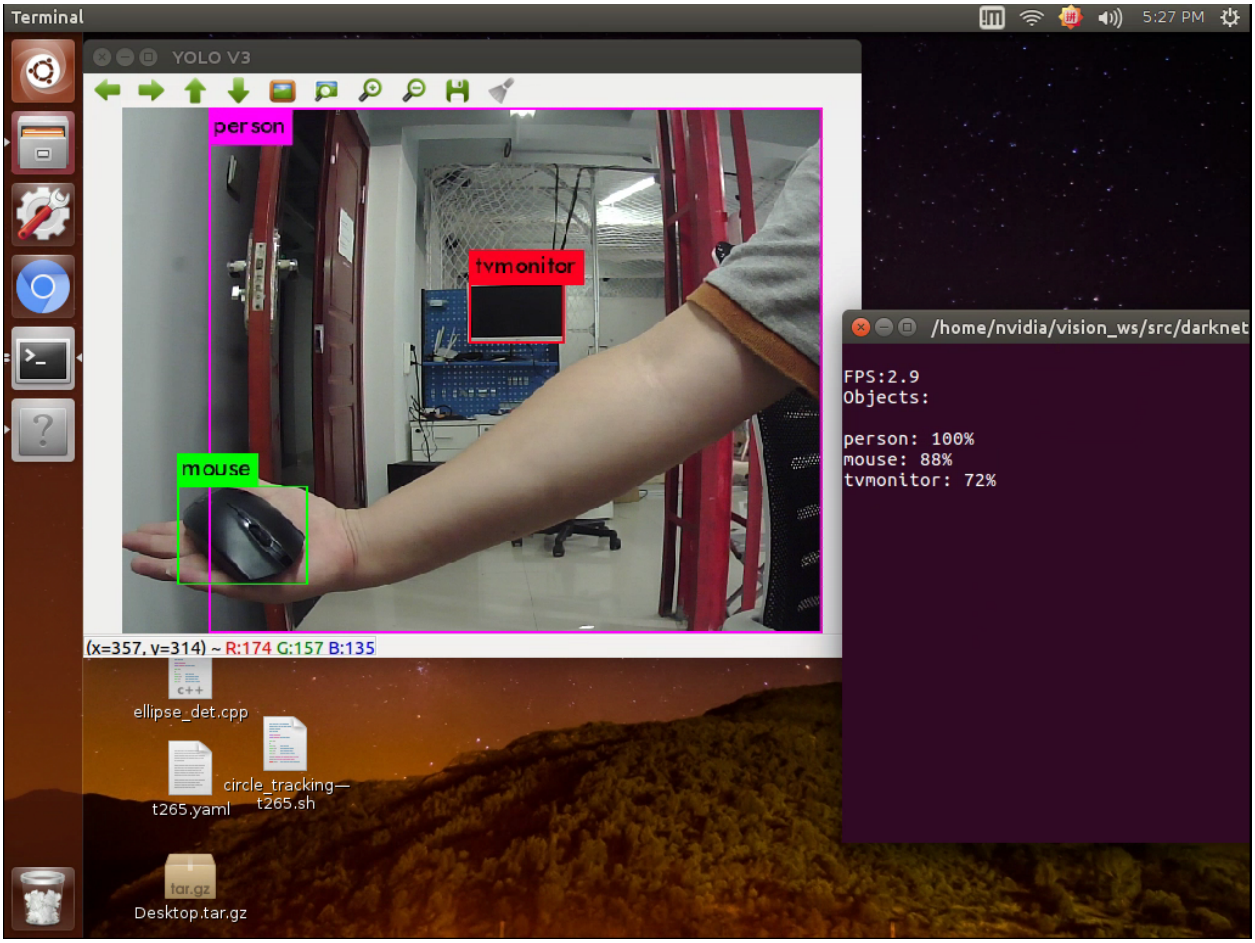
在 vision_ws/src 下面，有 ros_web_cam 功能包，它可以将图像转换为 ros 下面的 topic 形式。上面内容中我们已经修改好，yolov3 和 yolov3-tiny 的 launch 文件中图像的 topic，所以我们直接进行 ros 版的 darknet。

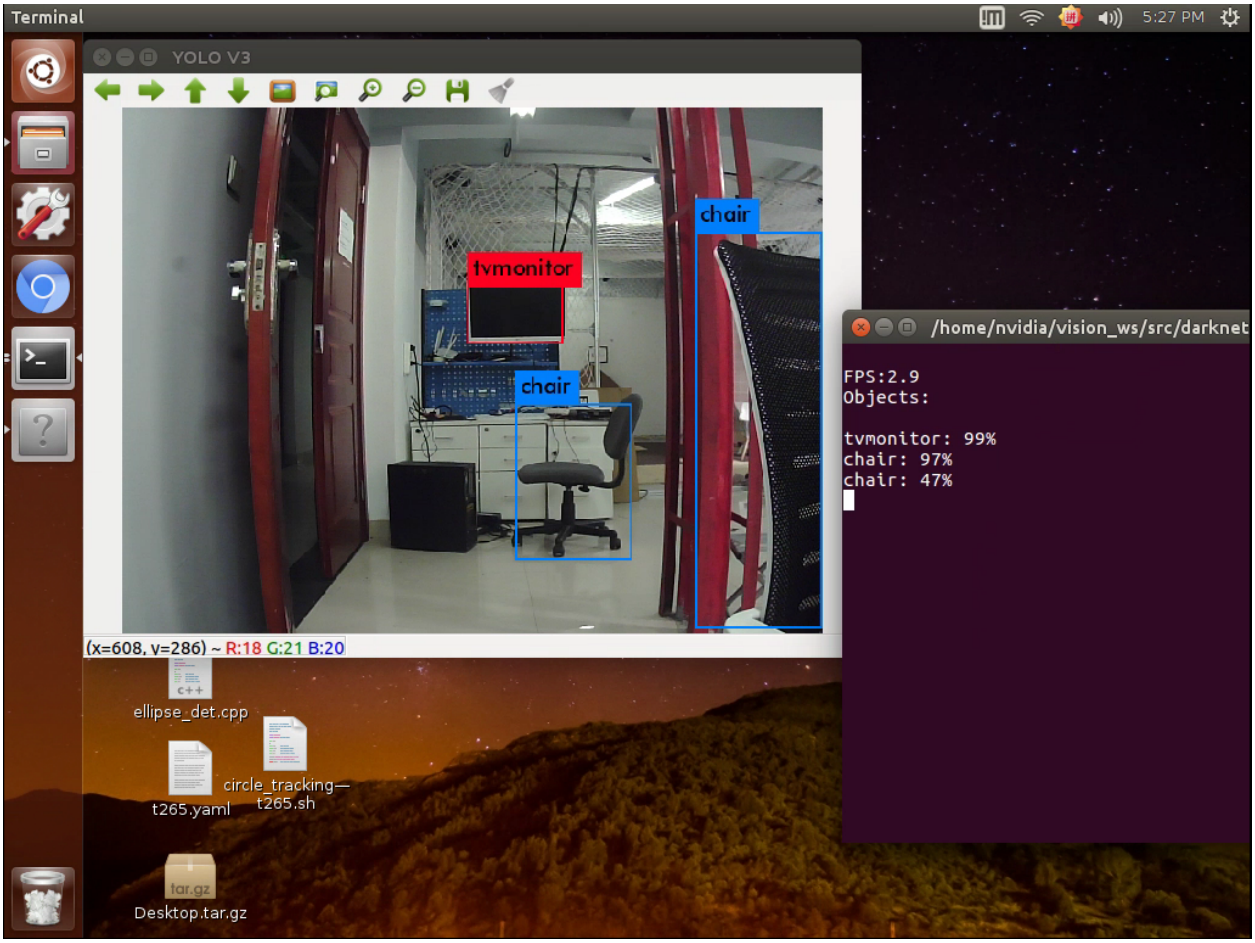
yolov3

先启动 yolov3 的 launch 文件，执行 `roslaunch darknet_ros yolo_v3.launch`

然后启动相机，执行 `rosrun web_cam web_cam`


Terminal
5:20 PM
/home/nvidia/vision_ws/src/darknet_ros/darknet_ros/launch/yolo_v3.launch http://localh
OPs
103 conv 128 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BFL
OPs
104 conv 256 3 x 3 / 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BFL
OPs
105 conv 255 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 255 0.353 BFL
OPs
106 detection
Loading weights from /home/nvidia/vision_ws/src/darknet_ros/darknet_ros/yolo_network_config/weights/yolov3.weights...Done!
Waiting for image.
nvidia@tegra-ubuntu: ~
nvidia@tegra-ubuntu:~$ rosrun web_cam web_cam
vision_to_mavros
devices -c .txt
ellipse_det.cpp
t265.yaml
circle_tracking—t265.sh
tar.gz
Desktop.tar.gz


Terminal
5:27 PM
YOLO V3
person
tvmonitor
mouse
(x=357, y=314) ~ R:174 G:157 B:135
/home/nvidia/vision_ws/src/darknet
FPS:2.9
Objects:
person: 100%
mouse: 88%
tvmonitor: 72%
ellipse_det.cpp
t265.yaml
circle_tracking—t265.sh
tar.gz
Desktop.tar.gz

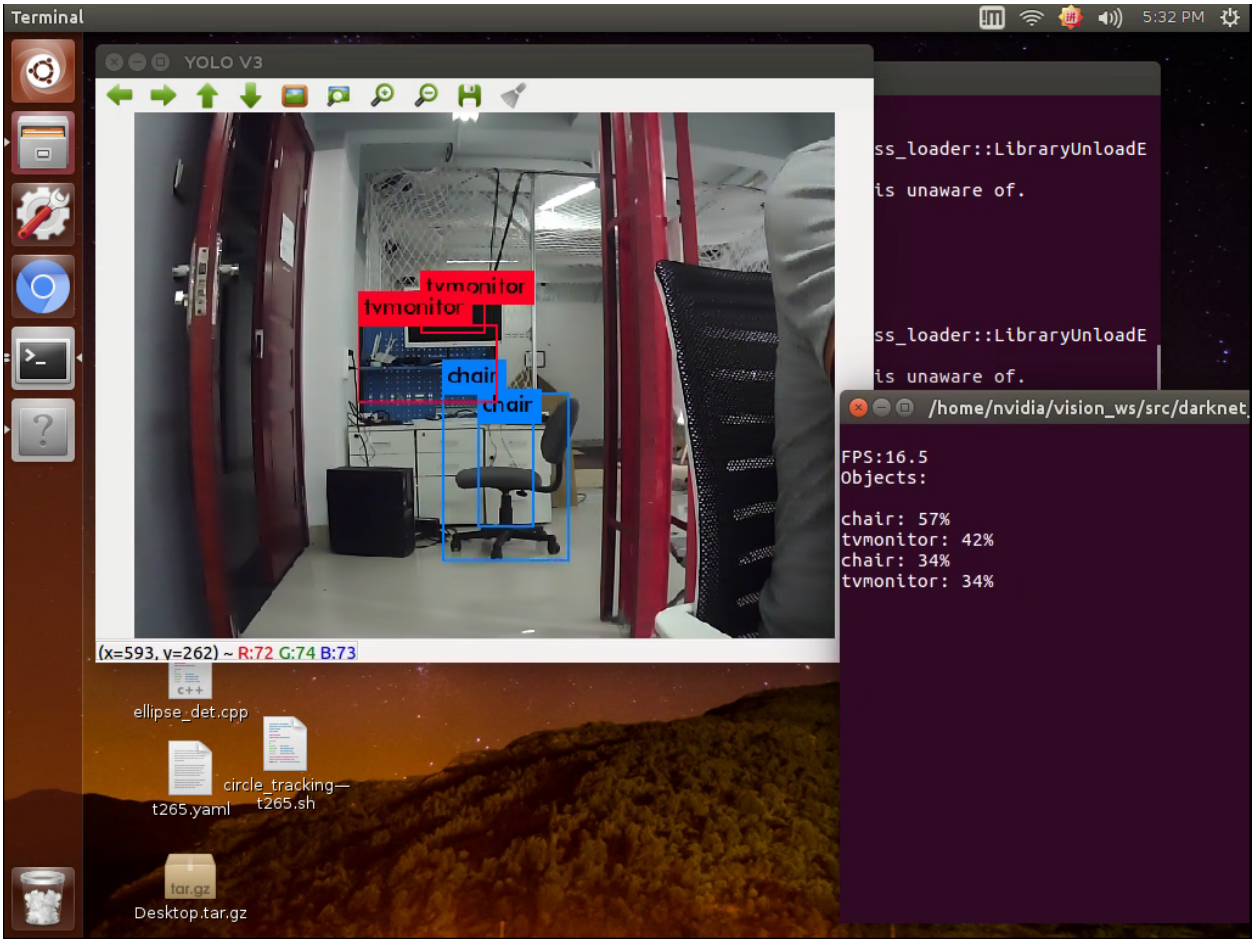


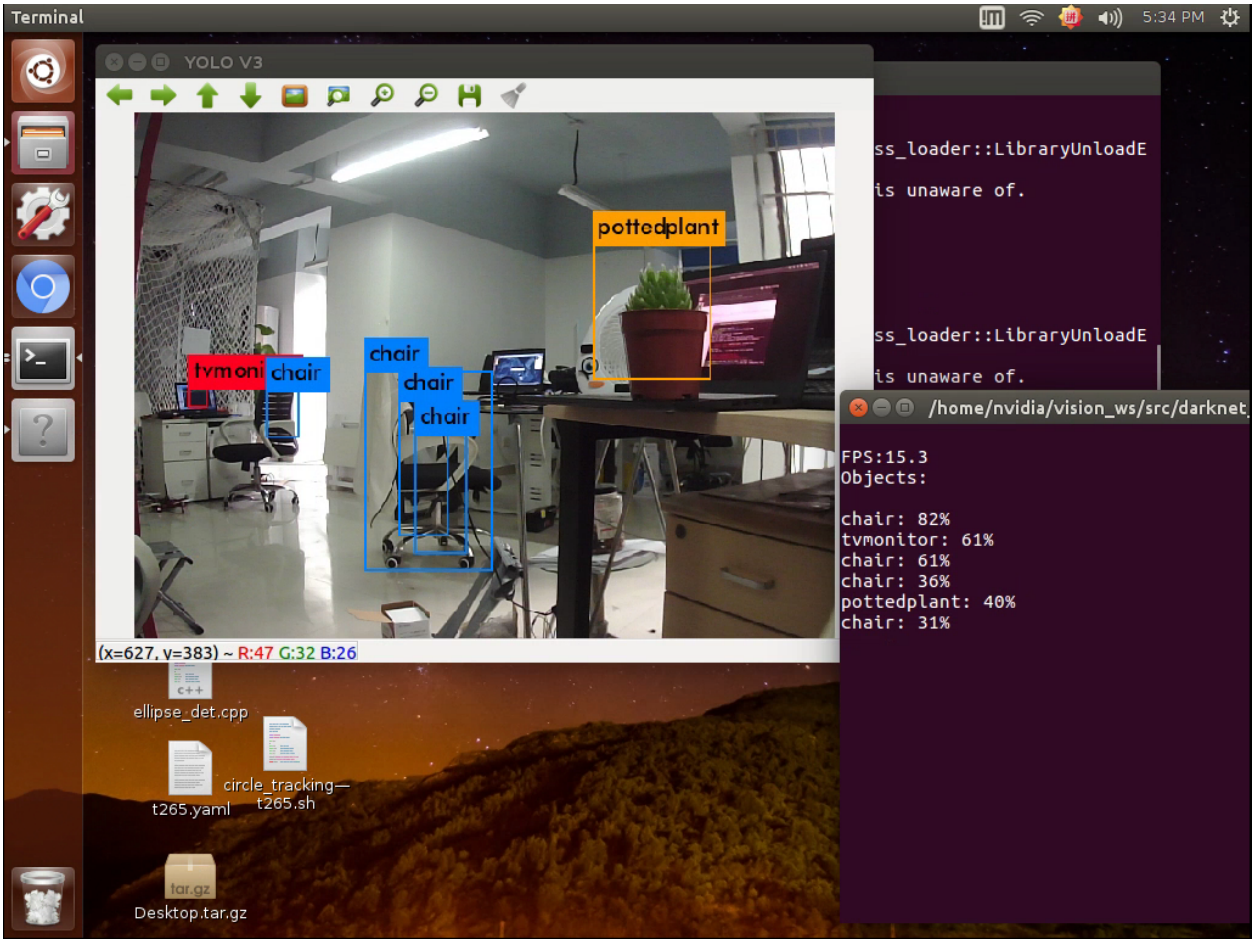
由此可见，帧率只有 3.0 左右。接下来使用 yolov3-tiny 看看。

yolov3-tiny

先运行 `roslaunch darknet_ros yolo_v3_tiny.launch`

接着运行 `rosrun web_cam web_cam`


Terminal
5:32 PM
YOLO V3
tvmonitor
tvmonitor
chair
chair
(x=593, y=262) ~ R:72 G:74 B:73
ss_loader::LibraryUnloadE
is unaware of.
ss_loader::LibraryUnloadE
is unaware of.
/home/nvidia/vision_ws/src/darknet
FPS:16.5
Objects:
chair: 57%
tvmonitor: 42%
chair: 34%
tvmonitor: 34%
ellipse_det.cpp
t265.yaml
circle_tracking—t265.sh
Desktop.tar.gz



帧率明显看出来是提高了很多，延迟还是会有点。

d、双目 t265 使用 darknet_ros

t265 功能包代码更新

更新 vision_to_mavros 功能包，该功能包的路径在 *~/realsense_ws/src* ，可以使用 git remote 也可以直接删除该功能包，重新下载。

vision_to_mavros 功能包地址，码云 gitee：https://gitee.com/eason_xinyi/vision_to_mavros.git

删除之前的包，下载更新，如下 `git clone https://gitee.com/eason_xinyi/vision_to_mavros.git`

切换到 *amov_200902* 分支之下，使用如下命令：`git checkout amov_200902`

cd ../.. 进入到 *~/realsense_ws* 路径之下，执行 `catkin_make` 进行编译，如有报错，请提交 issue。

编译完成之后，代码更新完毕，代码增加了 t265 的畸变处理。

修改畸变处理的配置文件：

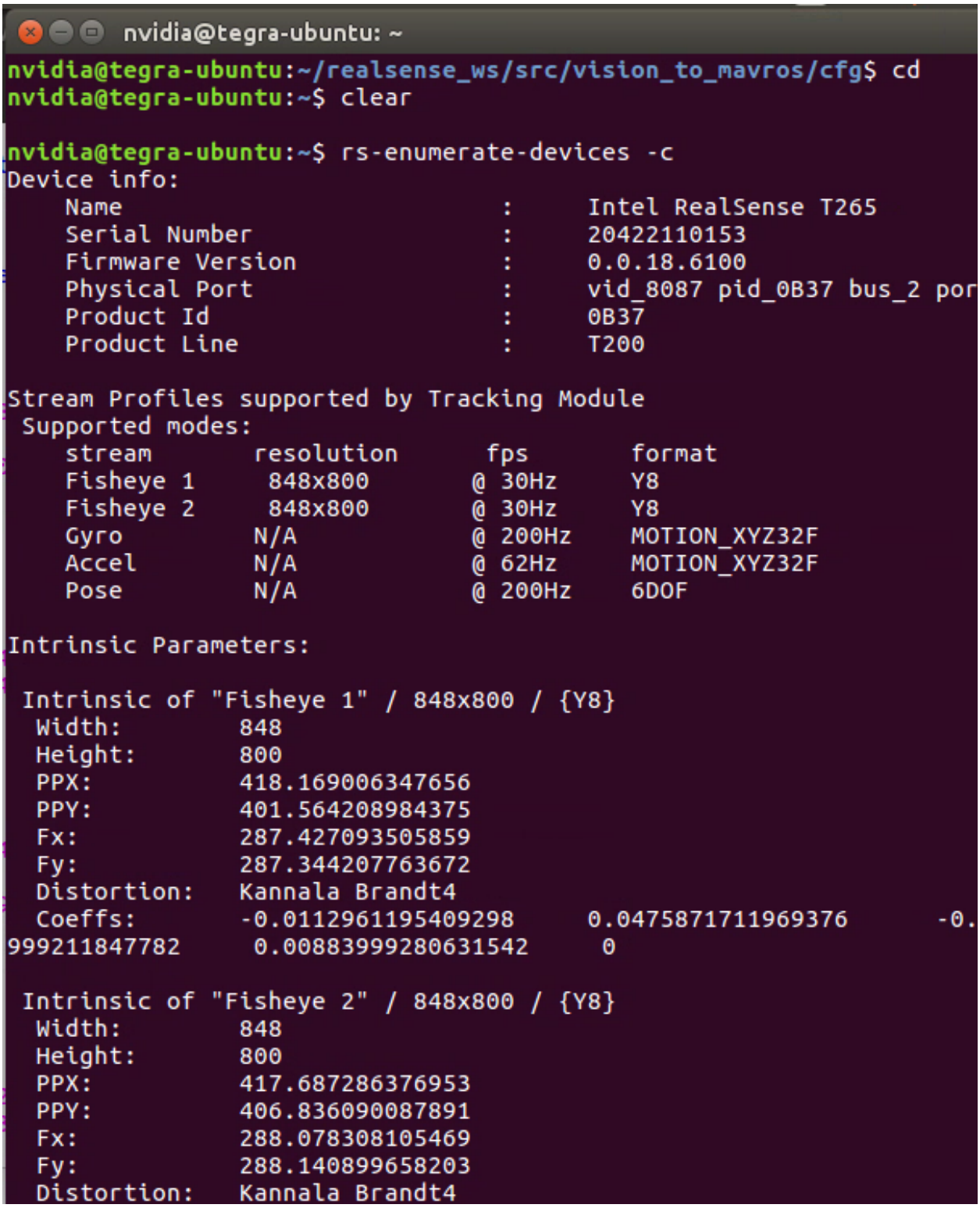
在路径 *~/realsense_ws/src/vision_to_mavros/cfg* 之下的 t265.yaml 文件。

t265.yaml参数描述	描述	通过rs-enumerate-devices -c 获取的输出
K1	左相机内参矩阵	Fisheye1内参矩阵：PPX，PPY，Fx，Fy
K2	右相机内参矩阵	Fisheye2内参矩阵：PPX，PPY，Fx，Fy
D1	左相机畸变系数	Fisheye1前四个畸变系数
D2	右相机畸变系数	Fisheye2前四个畸变系数
R	左到右的旋转矩阵	Fisheye1到Fisheye2的旋转矩阵
T	左到右的平移向量	Fisheye1到Fisheye2的平移向量

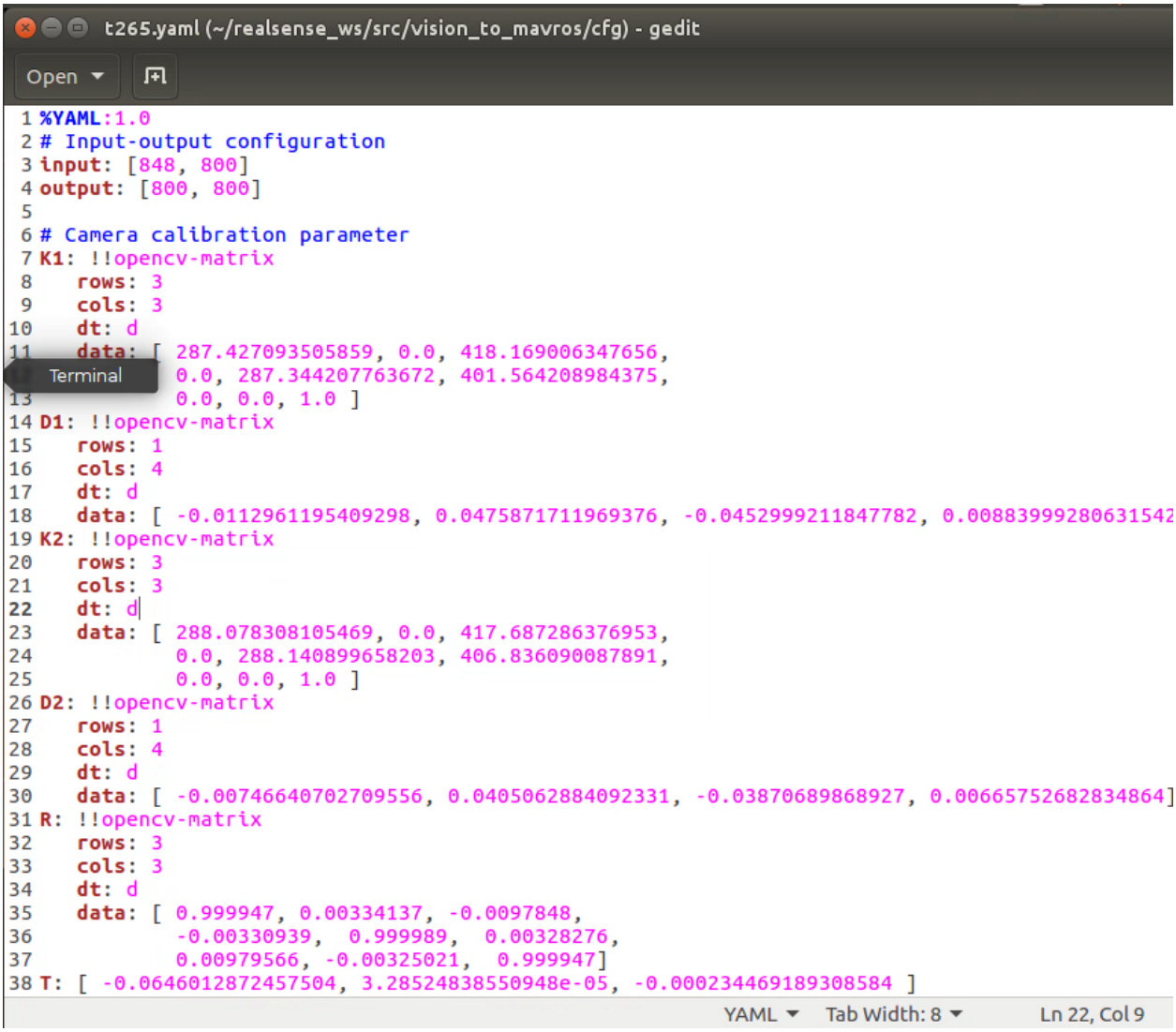
相机的内参矩阵 K：

Fx	0	PPX
0	Fy	PPY
0	0	1

t265 接在 tx2 USB3.0 的接口之上（下面的 USB 口），打开一个终端，输入：`rs-enumerate-devices -c`


nvidia@tegra-ubuntu: ~
nvidia@tegra-ubuntu:~/realsense_ws/src/vision_to_mavros/cfg$ cd
nvidia@tegra-ubuntu:~$ clear
nvidia@tegra-ubuntu:~$ rs-enumerate-devices -c
Device info:
Name : Intel RealSense T265
Serial Number : 20422110153
Firmware Version : 0.0.18.6100
Physical Port : vid_8087 pid_0B37 bus_2 por
Product Id : 0B37
Product Line : T200
Stream Profiles supported by Tracking Module
Supported modes:
stream resolution fps format
Fisheye 1 848x800 @ 30Hz Y8
Fisheye 2 848x800 @ 30Hz Y8
Gyro N/A @ 200Hz MOTION_XYZ32F
Accel N/A @ 62Hz MOTION_XYZ32F
Pose N/A @ 200Hz 6DOF
Intrinsic Parameters:
Intrinsic of "Fisheye 1" / 848x800 / {Y8}
Width: 848
Height: 800
PPX: 418.169006347656
PPY: 401.564208984375
Fx: 287.427093505859
Fy: 287.344207763672
Distortion: Kannala Brandt4
Coeffs: -0.0112961195409298 0.0475871711969376 -0.
999211847782 0.00883999280631542 0
Intrinsic of "Fisheye 2" / 848x800 / {Y8}
Width: 848
Height: 800
PPX: 417.687286376953
PPY: 406.836090087891
Fx: 288.078308105469
Fy: 288.140899658203
Distortion: Kannala Brandt4

```
nvidia@tegra-ubuntu: ~


Motion Intrinsic of "Accel"       MOTION_XYZ32F
Bias Variances:          0.000099999997474  0.000099999997474  0.000099999997474
Noise Variances:         0.000066952452471  0.000066952452471  0.000066952452471
Sensitivity :
     1.011681       0.000000       0.000000      -0.123306
     0.000000       1.021065       0.000000       0.622881
     0.000000       0.000000       1.018397      -0.111889



Extrinsic Parameters:
Extrinsic from "Fisheye 1"        To      "Fisheye 1" :
 Rotation Matrix:
   1   0   0
   0   1   0
   0   0   1

 Translation Vector: 0  0  0

Extrinsic from "Fisheye 1"        To      "Fisheye 2" :
 Rotation Matrix:
   0.999947         0.00334137      -0.0097848
  -0.00330939       0.999989         0.00328276
   0.00979566      -0.00325021       0.999947

 Translation Vector: -0.0646012872457504  3.28524838550948e-05  -0.000234469189308584

Extrinsic from "Fisheye 1"        To      "Gyro" :
 Rotation Matrix:
  -0.999874        -0.00109203       0.0158113
   0.00110183      -0.999999         0.000610681
   0.0158107        0.000628026      0.999875

 Translation Vector: 0.0106986593455076  -1.1789561540354e-05  -0.00016917398897931

Extrinsic from "Fisheye 1"        To      "Accel" :
 Rotation Matrix:
  -0.999874        -0.00109203       0.0158113
   0.00110183      -0.999999         0.000610681
   0.0158107        0.000628026      0.999875
```



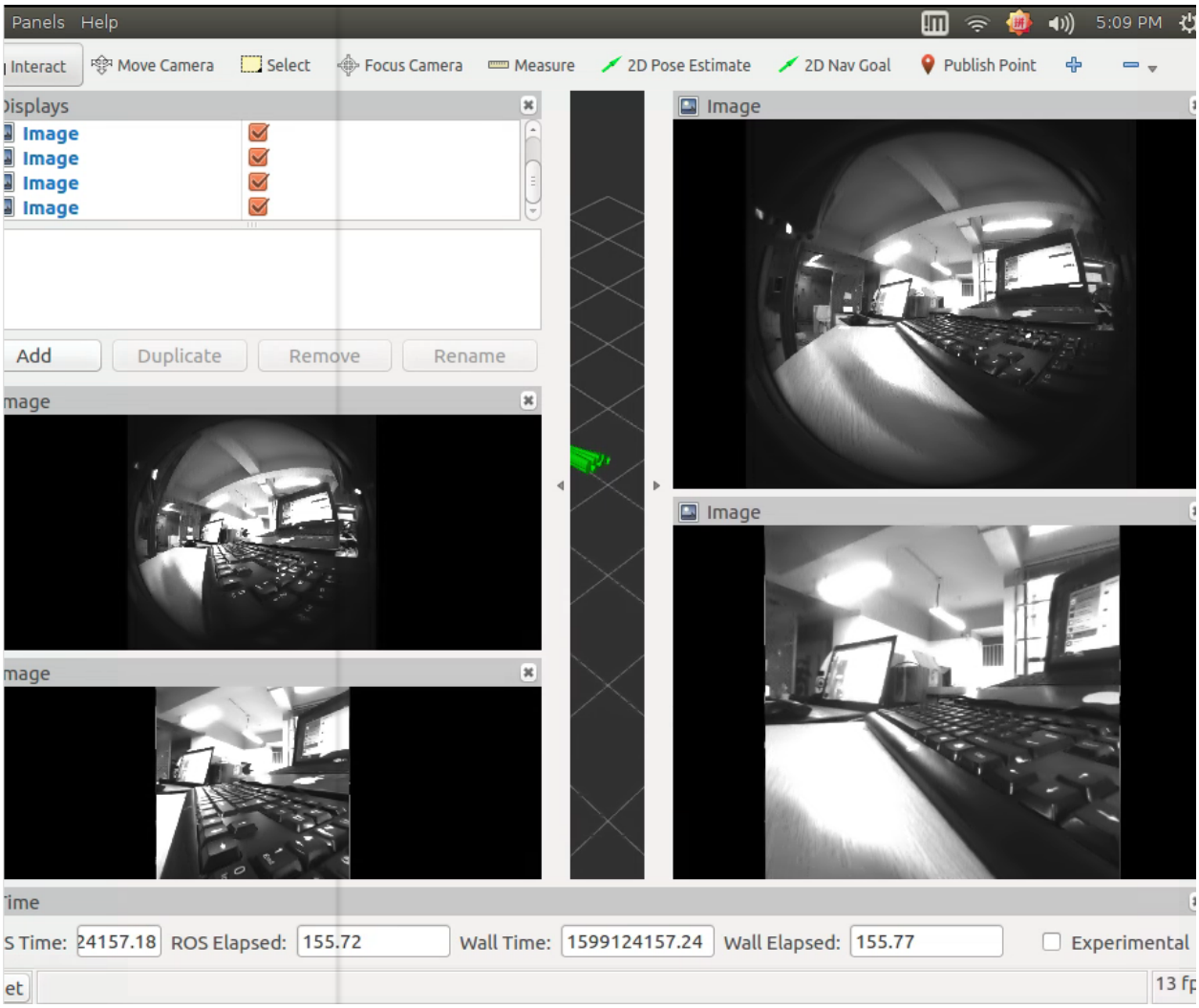
将 t265.yaml 中的所有参数根据 `rs-enumerate-devices -c` 所查到的参数进行一一对应进行填写。

运行 t265 畸变处理的效果：

先运行 t265 的驱动：`roslaunch realsense2_camera rs_t265.launch`

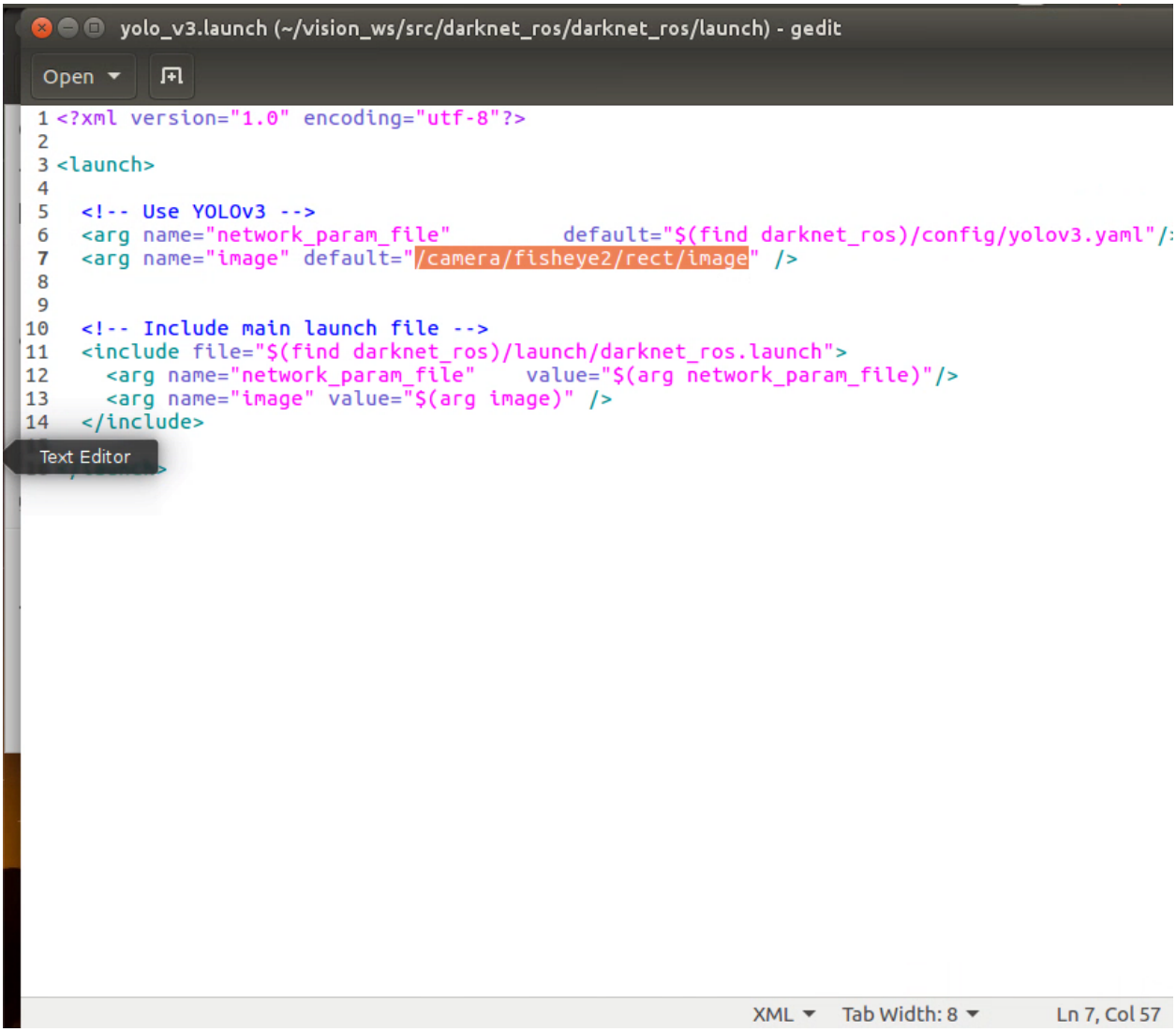
再运行畸变处理程序：`roslaunch vision_to_mavros t265_fisheye_undistort.launch`

如下图所示，为原始图像以及对应的畸变处理之后的图像。



yolov3

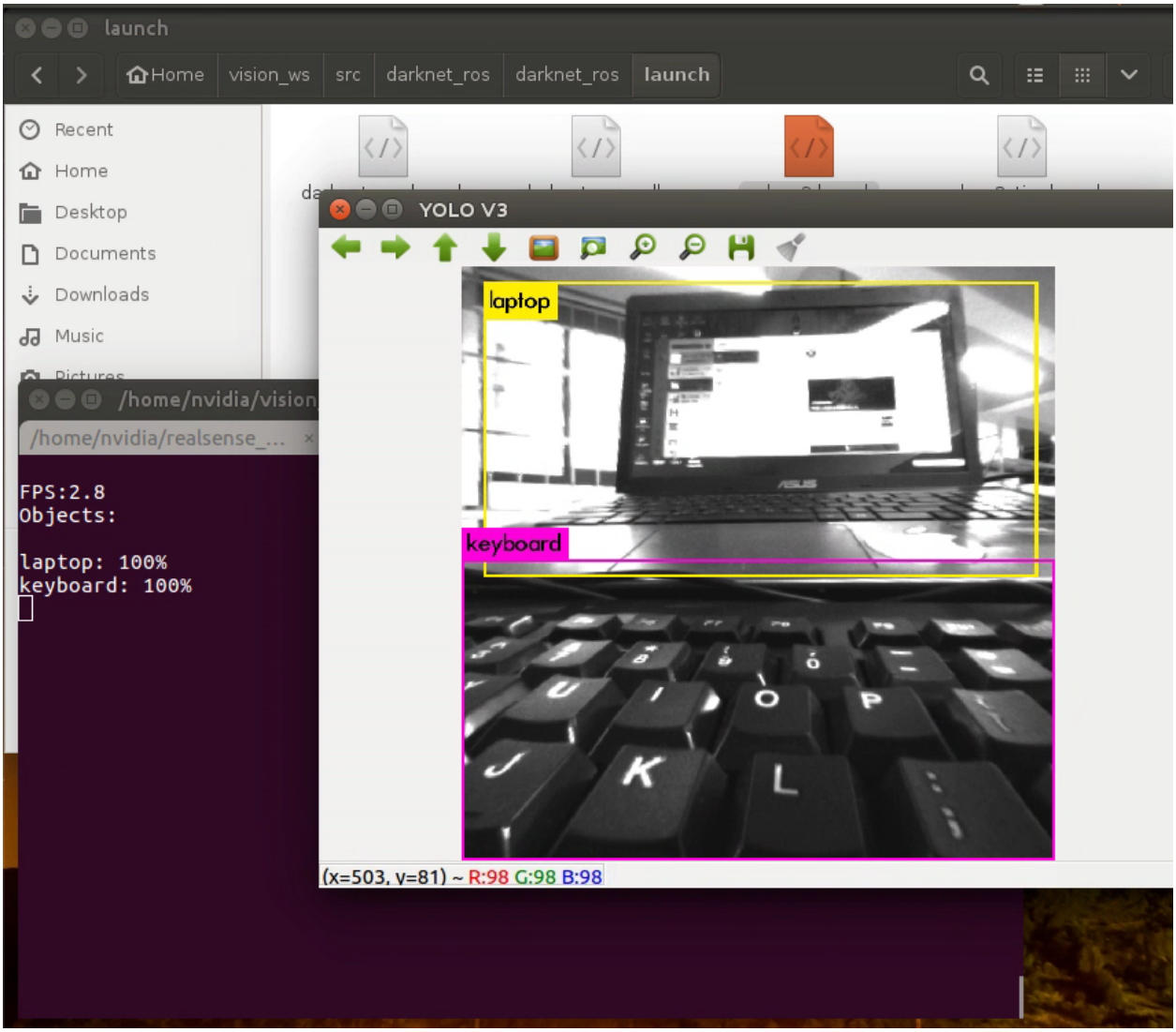
修改 yolo_v3.launch 文件，将 image 参数改为双目的 topic ，我们改为 /camera/fisheye2/rect/image 。

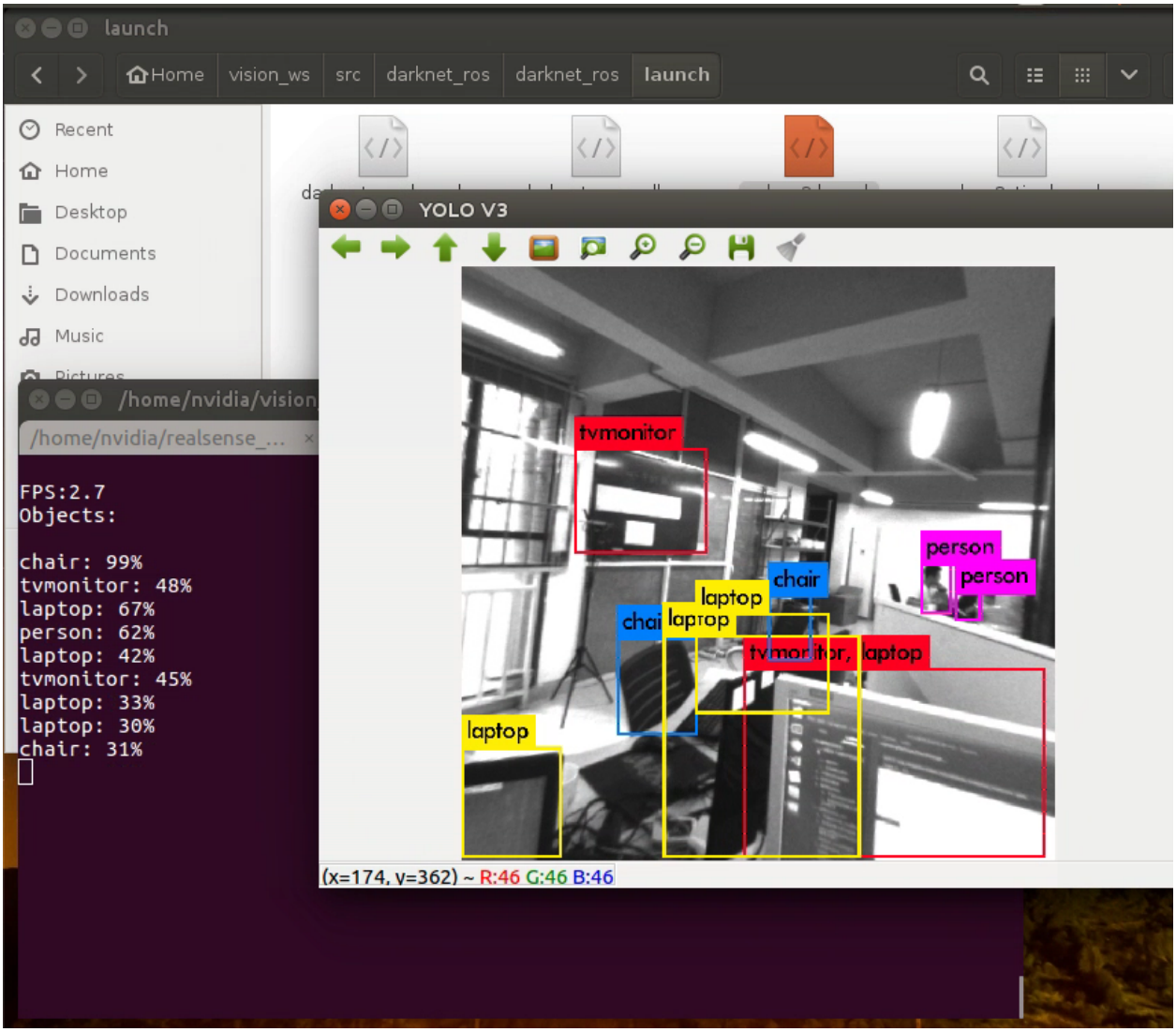


运行如下三个 launch 文件:

```
roslaunch realsense2_camera rs_t265.launch

roslaunch vision_to_mavros t265_fisheye_undistort.launch

roslaunch darknet_ros yolo_v3.launch
```

检测效果如下图:


launch
Home
vision_ws
src
darknet_ros
darknet_ros
launch
Recent
Home
Desktop
Documents
Downloads
Music
YOLO V3
laptop
keyboard
/home/nvidia/realsense_...
FPS:2.8
Objects:
laptop: 100%
keyboard: 100%
(x=503, v=81) ~ R:98 G:98 B:98



yolov3-tiny

与 yolov3 一样的操作流程。

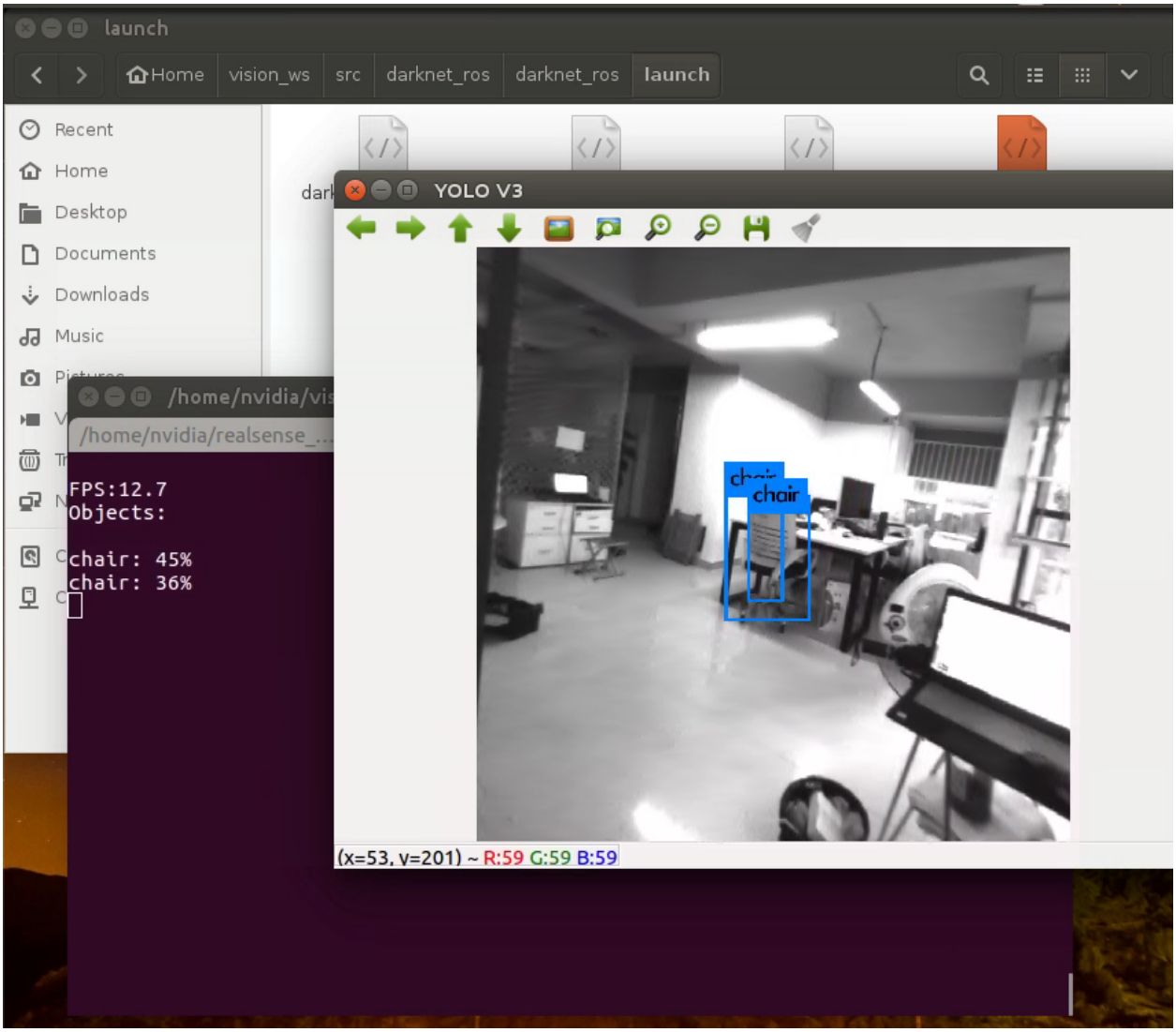
运行如下三个 launch 文件:

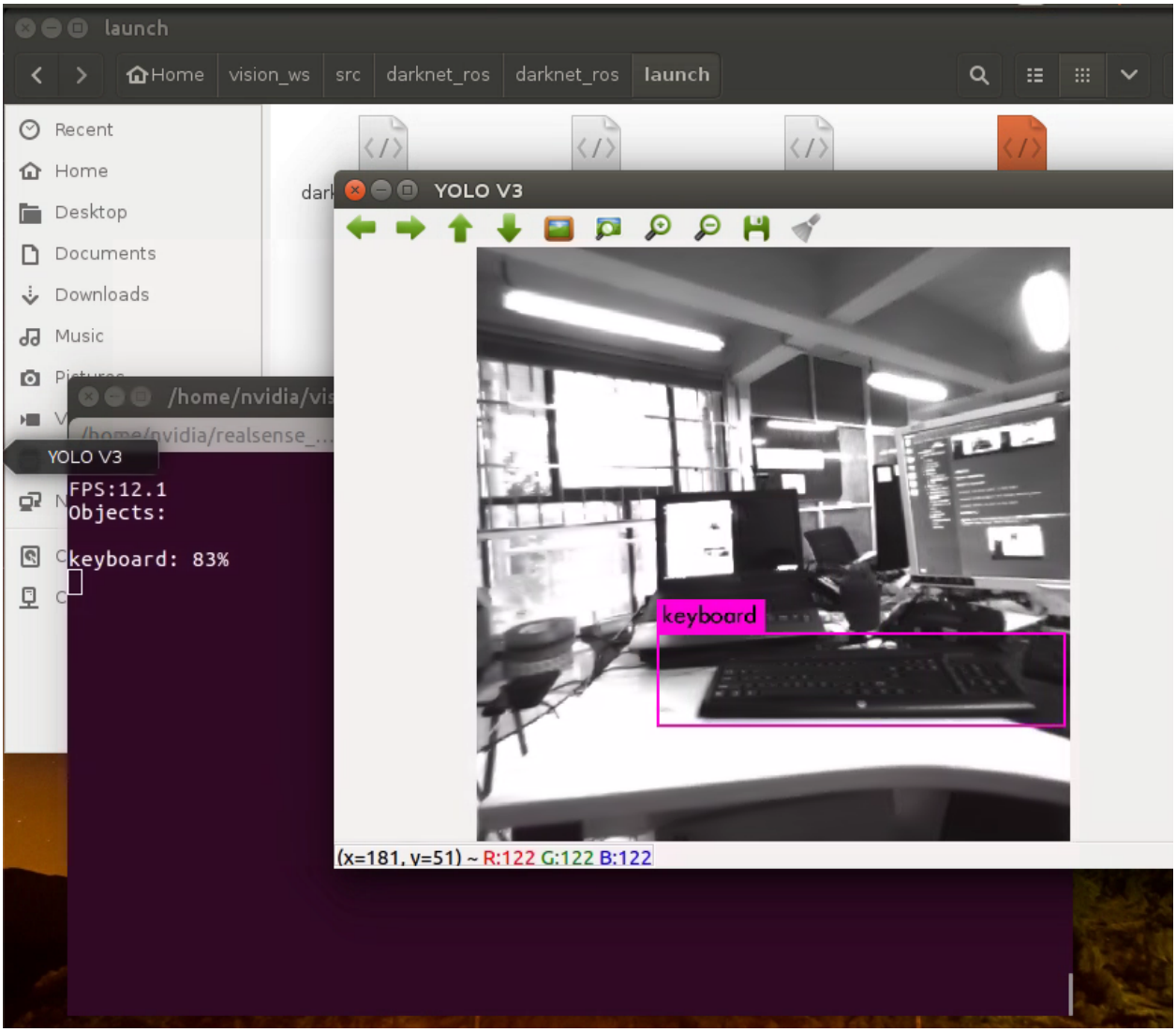
```
roslaunch realsense2_camera rs_t265.launch
```

```
roslaunch vision_to_mavros t265_fisheye_undistort.launch
```

```
roslaunch darknet_ros yolo_v3_tiny.launch
```

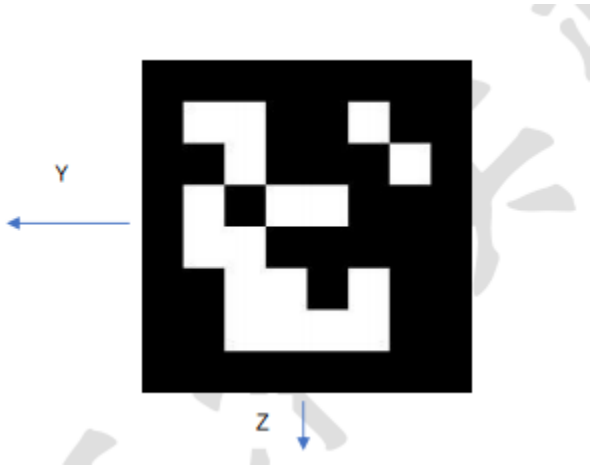
检测效果如下图:


launch
Home
vision_ws
src
darknet_ros
darknet_ros
launch
Recent
Home
Desktop
Documents
Downloads
Music
YOLO V3
chair
chair
/home/nvidia/realsense_...
FPS:12.7
Objects:
chair: 45%
chair: 36%
(x=53, y=201) ~ R:59 G:59 B:59



自主飞行之二次开发

测试用例



待定

Gazebo 仿真

PX4 提供了一种全自主飞行控制方式，offboard 模式。而阿木社区具有一套完整，可靠的系统体系。阿木社区的学员们在购买我们的飞机回去之后发现对我们的系统体系还不是了解，导致操作不当还是会有很多炸机现象，为了让学员们更加了解系统体系的种种情况，现推出仿真环境平台，对阿木系统体系的仿真模拟。有了这一套仿真系统，可以在不用实际飞行情况下理解阿木系统体系中的逻辑，减少实际飞行中炸机的发生。本篇文章中，会讲解如何使用系统体系控制无人机飞行。本套仿真平台在 Ubuntu16.04（16.04.6）LTS，ROS-Kinetic（1.12.14），Firmware（1.9.2），QGroundControl-v3.3.2，交叉编译工具链为 gcc-arm-none-eabi-7-2017-q4-major-linux，mavros 的二进制安装，测试通过。先从搭建环境开始讲起吧。

第一节硬件准备

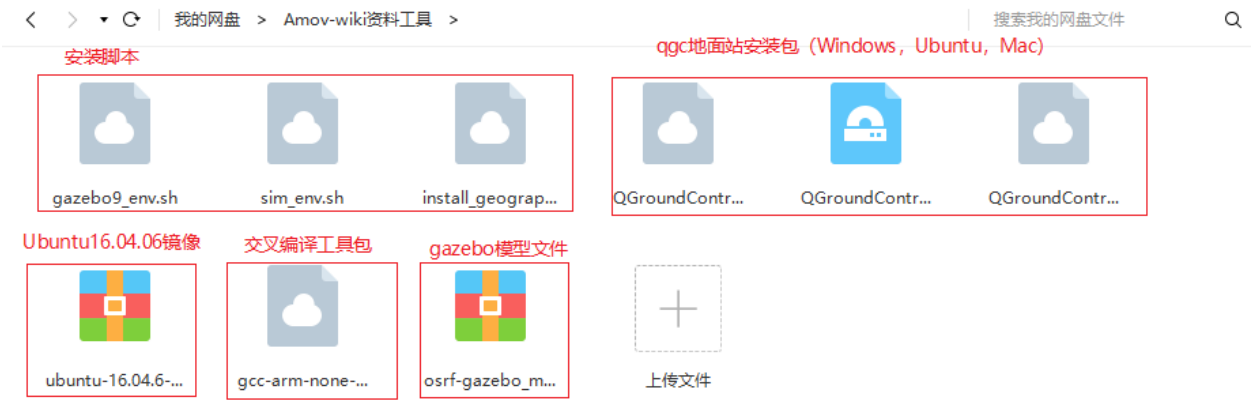
本套仿真环境既可以在 Windows 下面的 VMware 下面搭建，也可以在实体机上面搭建.Windows 虚拟机 VMware 下面我们提供搭建好了的开发环境，包括 PX4 开发环境,mavros 环境,ROS 环境. 但是，需要注意，虚拟机上运行会很卡，如果有显卡的话，打开 3D 加速，不是很卡顿.

本篇文章是在实体机上搭建环境的，电脑配置如下: 显卡 NVIDIA GTX2060;CPU 为 AMD Ryzen 5 3600 6-Core Processor × 12，内存大小为 16G,256 固态硬盘

第二节软件配置

小技巧: 软件相关资料百度网盘链接如下: 内容如下图所示，含有环境安装所需相关脚本,QGC 三大系统的安装包,Ubuntu16.04 长期支持版镜像文件,gcc7.2.0 交叉编译工具包，以及最后的 gazebo 模型文件包.

链接：https://pan.baidu.com/s/1wlLhcF07AAe0zTqcOdTEag 提取码：uv2u



1.Ubuntu16.04 操作系统

用 UltraISO 制作 U 盘启动盘，步骤如下:

- 打开 UltraISO 软件，打开文件，选择要安装的 Ubuntu 版本，安装的镜像是 Ubuntu-16.04.6，偶数版本表示长期 (5 年) 维护的系统镜像，在 2021 年之前是支持的。
- 接着我们点击启动-> 写入磁盘映像，进入下面的界面，在制作 Ubuntu 的 U 盘启动盘的时候，要选择 RAW。之前也尝试过其他的写入方式，偶尔会成功，偶尔会失败。但选择 RAW 之后，装的好几次都是成功了的，所以建议用 RAW 写入方式。
- 之后就是把 U 盘格式化，然后点击写入即可，等待……（长短取决于电脑性能），待完成之后 U 盘启动盘就制作成功了。

安装 Ubuntu 系统

> 插上 U 盘，开机启动选项中设置 U 盘启动即可，一般不同的电脑进入 BIOS 的方式不同。之后的安装很简单，百度上有很多教程，按照教程安装即可完成安装。本次安装是一块新的 256 固态硬盘，全盘直接安装的是 Ubuntu 系统，所以没有什么分区设置。在安装完 Ubuntu 系统的第一件事情就是用户组的添加

```
sudo usermod -a -G dialout $USER
```

Ubuntu 系统的技巧设置

- 技巧一刚装完 Ubuntu 后发现分辨率很低，屏幕看起来很别扭。查看了一下，分辨率只有 640x480，然而我的屏幕是 1920x1080 的。然后通过改 xrandr 和 cvt 都无效。经过一番查找，找到解决方案，修改 grub 默认的分辨率，具体过程如下：sudo gedit /etc/default/grub 找到：#GRUB_GDXMODE=640x480 改为：GRUB_GDXMODE=1920x1080 然后更新一下 grub: sudo update-grub 最后重启电脑即可
- 技巧二建议在安装完系统之后，只留下现在所使用的版本的内核，删除其余多余的内核，并且禁用内核的更新，否则过段时间，系统默认启动更新后的内核。(具体的如何禁用设置上网一搜索便可找到)
- 安装显卡驱动，Ubuntu 默认的显卡驱动是 nouveau，你需要安装与你显卡相匹配的驱动程序。以 NVIDIA 驱动为例，首先是查看自己显卡，发现是设备 ID 为 1f08，通过 NVIDIA 驱动 ID 查看 搜索

发现该驱动是 GTX2060，然后我们到 NVIDIA 驱动程序下载 下载相应的驱动下载相应的驱动安装程序。安装的过程你可以参考这篇文档 NVIDIA 驱动安装 . 在参考该文档的时候, 在安装驱动的时候可以不选择后面的可选配置, 比如 -no-x-check,-no-nouveau-check 等选项. 直接安装也可以, 但是有时候, 安装完成之后出现循环登录问题, 请重新多安装几次 (修改可选配置安装).

```
如下命令就是查看自己电脑当前可用的显卡，获取到 NVIDIA 显卡的设备 ID
amov@amov:~$ lspci | grep VGA
0a:00.0 VGA compatible controller: NVIDIA Corporation Device 1f08 (rev a1)
```

2.PX4 环境安装

参考官方文档 Ubuntu 下 px4 开发环境搭建 .

警告: 该文档链接是在当时环境下的 master 文档, 对应的是 1.8.2 的 wiki 文档, 你现在所在的 master 页面是最新的 wiki 文档, 你需要在左上角由 master 切换到 1.8.2

下载好软件相关资料, 将安装脚本放置 home 下面, 给脚本可执行权限并执行这个脚本。

```
chmod +x sim_env.sh
sudo ./sim_env.sh
```

这个安装的快慢与你的网速有关。这个脚本本身是没有安装交叉编译工具链, 也没有下载 Firmware 固件的, 也没有安装 gazebo. 该脚本如果你正常安装且没有安装错误, 那么 px4 编译环境装好, 然后再安装 gazebo9 软件, 安装交叉编译工具包, 最后下载 px4 固件 Firmware, 编译固件并编译仿真.

上面 sim_env.sh 脚本执行完成之后, 然后安装 gazebo9

```
chmod +x gazebo9_env.sh
sudo ./gazebo9_env.sh
```

接着安装交叉编译工具包，手动安装交叉编译工具链：下载软件相关资料, 找到 gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2, 之后复制 (+sudo) 放到/opt/之下，解压, 下所示本机的 gcc 的路径

```
amov@amov:/opt$ sudo tar -jxvf gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2
amov@amov:/opt/gcc-arm-none-eabi-7-2017-q4-major/bin$ pwd
/opt/gcc-arm-none-eabi-7-2017-q4-major/bin
```

然后打开/etc/profile 文件，如下

```
amov@amov:/opt/gcc-arm-none-eabi-7-2017-q4-major/bin$ sudo gedit /etc/profile
```

在最下面添加一行

```
export PATH=$PATH:/opt/gcc-arm-none-eabi-7-2017-q4-major/bin
```

路径就是 gcc 存放的路径。接着 source 一下刚才修改的/etc/profile

```
source /etc/profile
```

测试安装 gcc 是否成功，输入

```
arm-none-eabi-gcc --version
```

若出现如下类似，说明安装成功

```
arm-none-eabi-gcc (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 7.2.1 20170904␣
↪(release) [ARM/embedded-7-branch revision 255204]
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

若输出是：

```
arm-none-eabi-gcc --version
arm-none-eabi-gcc: No such file or directory
```

需要安装 32 位支持库 此链接查看详细步骤

```
sudo apt-get install libc6:i386 libgcc1:i386 libstdc++5:i386 libstdc++6:i386
```

现在 PX4 环境配置已经完成，之前在运行 ubuntu_sim.sh 脚本中下载过 Firmware，建议重新下载一个 PX4 固件。

```
amov@amov:~/Desktop/px4-src/src-1.8.2$ ls
amov@amov:~/Desktop/px4-src/src-1.8.2$ git clone https://github.com/PX4/Firmware.git
Cloning into 'Firmware'...
remote: Enumerating objects: 278734, done.
```

下载完之后，我们进入到 Firmware 中，下载的还需要更新子模块

```
amov@amov:~/Desktop/px4-src/src-1.8.2$ cd Firmware/
amov@amov:~/Desktop/px4-src/src-1.8.2/Firmware$ git checkout v1.8.2
amov@amov:~/Desktop/px4-src/src-1.8.2/Firmware$ git submodule update --init --recursive
```

漫长等待之后，就可以编译源码了，先试试最基本的能力。首先是编译源代码

```
amov@amov:~/Desktop/px4-src/src-1.8.2/Firmware$ make px4fmu-v5_default
```

若编译成功的话，再执行编译最基本的 gazebo 仿真

```
amov@amov:~/Desktop/px4-src/src-1.8.2/Firmware$ make posix_sitl_default gazebo
```

到此为止，说明你的 PX4 环境配置已经搭建完成了。下来我们会配置与 Ubuntu16.04 系统对应的 ROS Kinetic 版本。

小技巧: 在 px4 固件代码 v1.8.2 之前的编译规则和 v1.8.2 之后的编译规则略有不同,

v1.8.2 中编译 v5 固件命令为 make px4fmu-v5_default.v1.9.2 中编译 v5 固件命令为 make px4_fmu-v5_default

v1.8.2 中编译 gazebo 仿真命令为 make posix_sitl_default gazebo . v1.9.2 中编译 gazebo 仿真命令为 make px4_sitl_default gazebo

---

3.ROS-Kinetic 安装

ROS-Kinetic 的安装参考 ROS-Kinetic 官网安装教程 需要注意的一点是，在安装 ROS 时候，国内最好选择镜像来自中科大的源或者是清华的源，其他就是按照官网提示一步步安装即可。

小技巧: 安装 ROS（有 700MB 到 800MB）完成之后，查看是否安装成功，如下表示安装 ROS 完成。

特别注意, 在上面我们安装好 px4 的编译环境时候, 安装的 gazebo9, 在安装 ROS-Ubuntu16.04-kinetic 的时候, 会默认将之前系统的 gazebo 卸载, 并重新安装 gazebo7. 但在实际过程中,gazebo9 更为好使用, 兼容性也更好, 所以在安装 ROS-kinetic 时候不要选择安装 sudo apt-get install ros-kinetic-desktop-full , 而应该选择 sudo apt-get install ros-kinetic-desktop. 这点需切记.

```
amov@amov:~$ roscore
... logging to /home/amov/.ros/log/d98e04fe-b1ca-11e9-bf5f-e0d55ee7d1ba/roslaunch-amov-
↪23391.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://amov:39279/
ros_comm version 1.12.14
```

(下页继续)

(续上页)

```
SUMMARY
========

PARAMETERS
* /rosdistro: kinetic
* /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [23401]
ROS_MASTER_URI=http://amov:11311/

setting /run_id to d98e04fe-b1ca-11e9-bf5f-e0d55ee7d1ba
process[rosout-1]: started with pid [23414]
started core service [/rosout]
```

4.mavlink 与 mavros 安装

mavlink 与 mavros 的安装参考 mavros 官方安装

最好最清晰的安装过程便是官方提供的步骤, 以安装二进制的方式安装 mavros, 还需要安装 geographiclib, 可别忘了.

5、下载 QGroundControl

本系统的 qgc 版本是 v3.3.2，是通过 Qt5.11.0 编译生成的。建议直接下载可执行程序，可参考开发者手册 QGC 下载与安装

第三节仿真过程

上节中，我们已经搭建好 PX4 仿真的环境了，而本节旨在下载阿木社区的源码，并且建立新的工作空间到个人工作路径下，然后配置仿真所使用的固件版本的选择以及环境配置，最后进行仿真操作。先从如何下载阿木社区源码说起

1. 打开阿木社区的 GitHub

上网进入 amovlab 阿木实验室维护的 GitHub.

2. 下载源码并建立工作区间

详细的建立工作空间请查看阿木社区 GitHub 上的项目 px4_commander. 或者如下链接：px4_command

建立好工作空间之后，笔者的工作空间如下：

```
amov@amov:~/AMOV_WorkSpace$ cd px4_ws/
amov@amov:~/AMOV_WorkSpace/px4_ws$ ls
build  devel  src
amov@amov:~/AMOV_WorkSpace/px4_ws$ cd devel/
amov@amov:~/AMOV_WorkSpace/px4_ws/devel$ ls
cmake.lock  lib               local_setup.zsh  _setup_util.py
env.sh      local_setup.bash  setup.bash       setup.zsh
include     local_setup.sh    setup.sh         share
amov@amov:~/AMOV_WorkSpace/px4_ws/devel$
```

打开.bashrc 文件

```
amov@amov:~/AMOV_WorkSpace/px4_ws/devel$ sudo gedit ~/.bashrc
```

需要在.bashrc 文件最后添加一行如下：

```
source ~/AMOV_WorkSpace/px4_ws/devel/setup.bash
```

3. 添加环境变量.bashrc 文件添加如下

```
source ~/Desktop/px4-src/src-1.8.2/Firmware/Tools/setup_gazebo.bash ~/Desktop/px4-src/
↪src-1.8.2/Firmware/ ~/Desktop/px4-src/src-1.8.2/Firmware/build/px4_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/Desktop/px4-src/src-1.8.2/Firmware
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/Desktop/px4-src/src-1.8.2/Firmware/Tools/
↪sitl_gazebo
```

4. 启动仿真

进入工作区间仿真部分目录下，可以看到有 6 个脚本文件

```
amov@amov:~/AMOV_WorkSpace/px4_ws/src/px4_command/sh/sh_for_simulation$ ls
sitl_gazebo_formation.sh       sitl_gazebo_square.sh
sitl_gazebo_iris.sh            sitl_jMAVSim_pos_controller.sh
sitl_gazebo_pos_controller.sh  sitl_test.sh
```

启动 sitl_gazebo_iris.sh 脚本, 执行如下

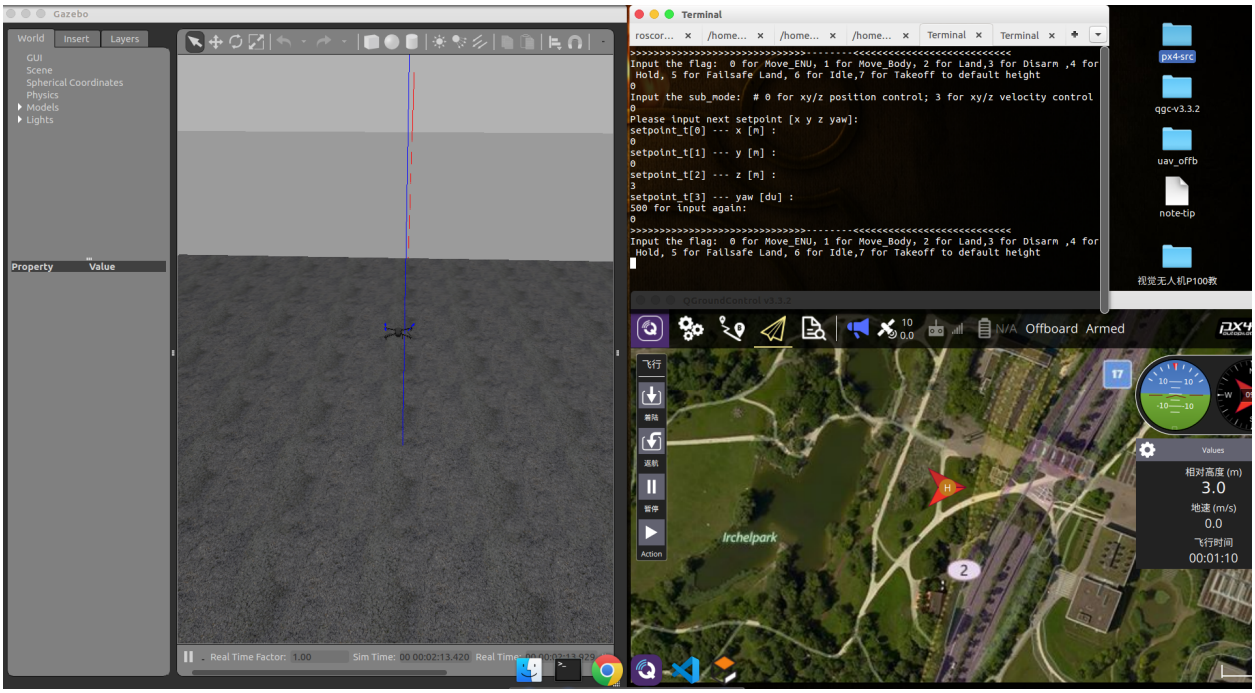
```
amov@amov:~/AMOV_WorkSpace/px4_ws/src/px4_command/sh/sh_for_simulation$ ./sitl_gazebo_
↪iris.sh
```

即可进入仿真界面。

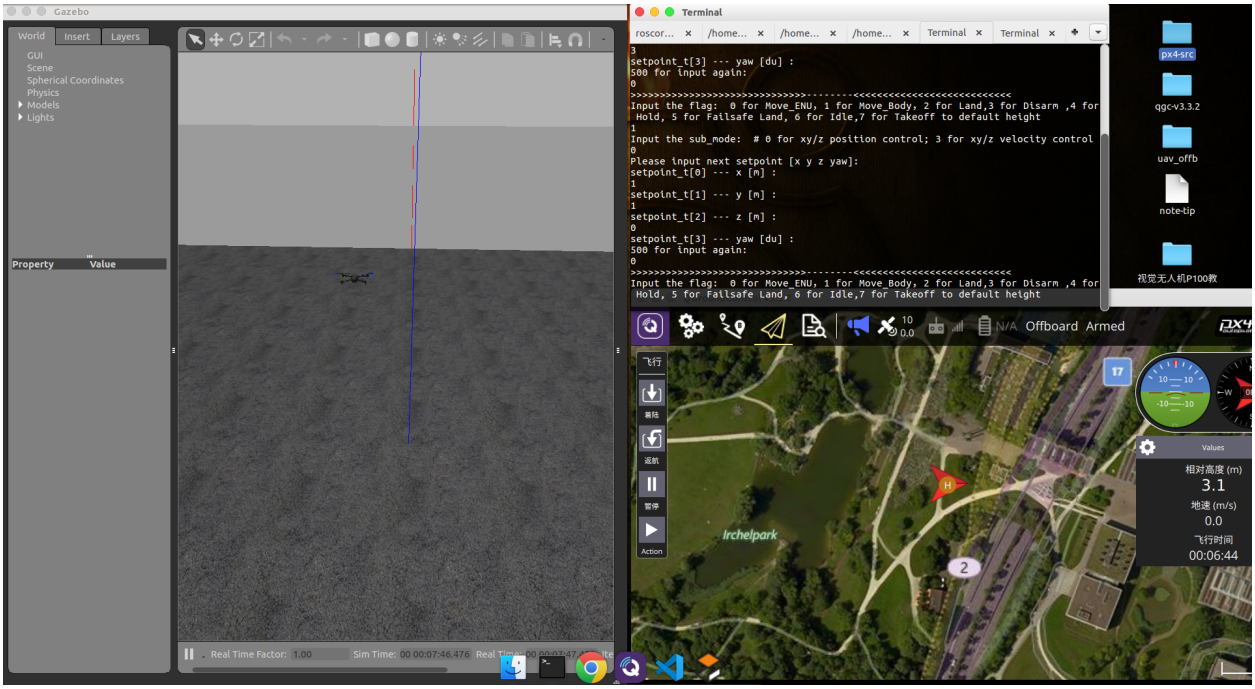
第四节仿真脚本说明

1. 脚本 sitl_gazebo_iris.sh

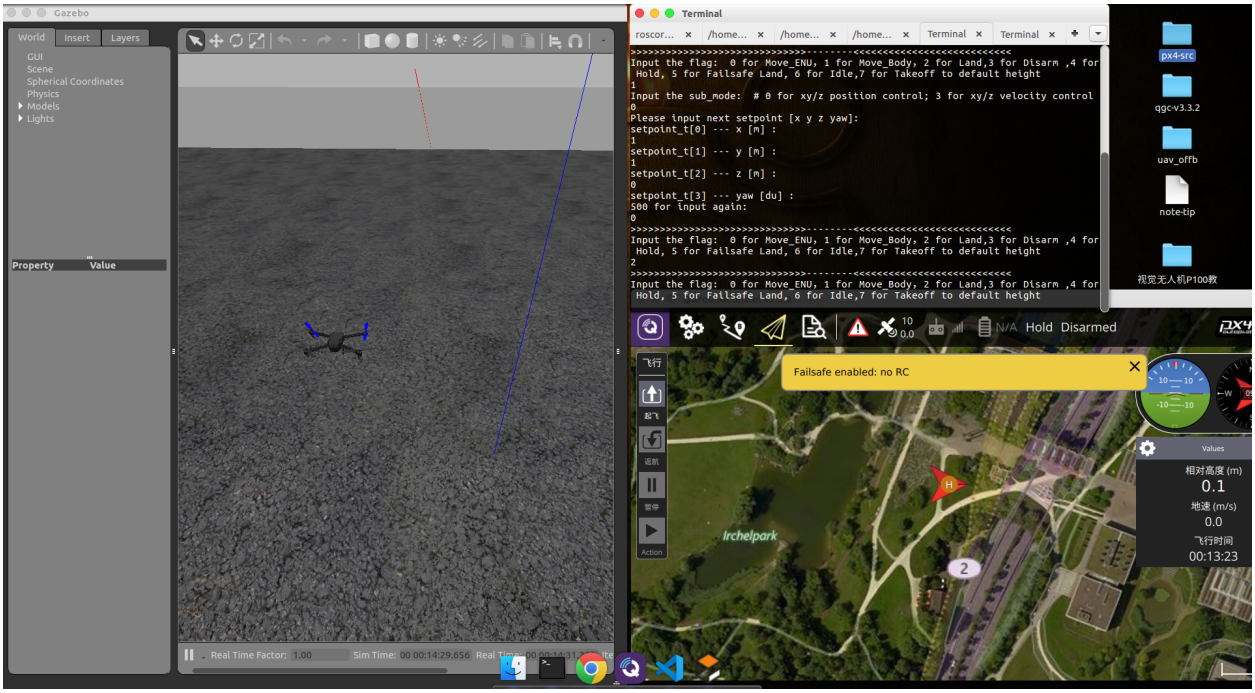
正常启动 sitl_gazebo_iris.sh 🄵本，基本操作流程和实体飞机操作流程一致。先起飞 3m, 如下图:



接着, 我们在 Move_Body 坐标系下,x,y,z 分别为 1,1,0. 飞行轨迹如下图:



最后我们执行一下 land 模式, 如下图:



存在 Bug 描述:

- 若起飞之后飞机降落至地面，无法进行再次起飞。(和实体飞机一致现象)
- 在 ENU 坐标系下，若使用速度控制，进行起飞 2M，飞机一致向上飞，不会停止，在 gazebo 中，飞至 26M，切换至悬停模式，无法成功相应，飞至 30M，切换至 land，正常降落。
- 经过多次测试，move 节点中，按键 4hold 模式无响应，在两种坐标系下的速度控制中，飞机一直向上

飞。

- 在 passivity 控制率下，正常设置起飞 3M，飞机纯粹油门量最大向上直飞，到达 53M 左右之后，有姿态角的迅速降落，直至炸机。
- 在 NE 控制率下，正常设置起飞 3M，飞机纯粹油门量最大向上直飞，一直飞。

2. 脚本 sitl_gazebo_square.sh

注解: 直接下载的 px4_command 是没有 sitl_gazebo_square.sh 该脚本的, 需要手动添加该脚本. 首先可以建立一个新的可执行脚本 sitl_gazebo_square.sh, 添加下面内容:

gnome-terminal –window -e ‘bash -c “roscore; exec bash”’

–tab -e ‘bash -c “sleep 4; roslaunch px4 posix_sitl.launch; exec bash”’

–tab -e ‘bash -c “sleep 2; roslaunch mavros px4.launch fcu_url:=” udp://:14540@127.0.0.1:14557” ; exec bash”’

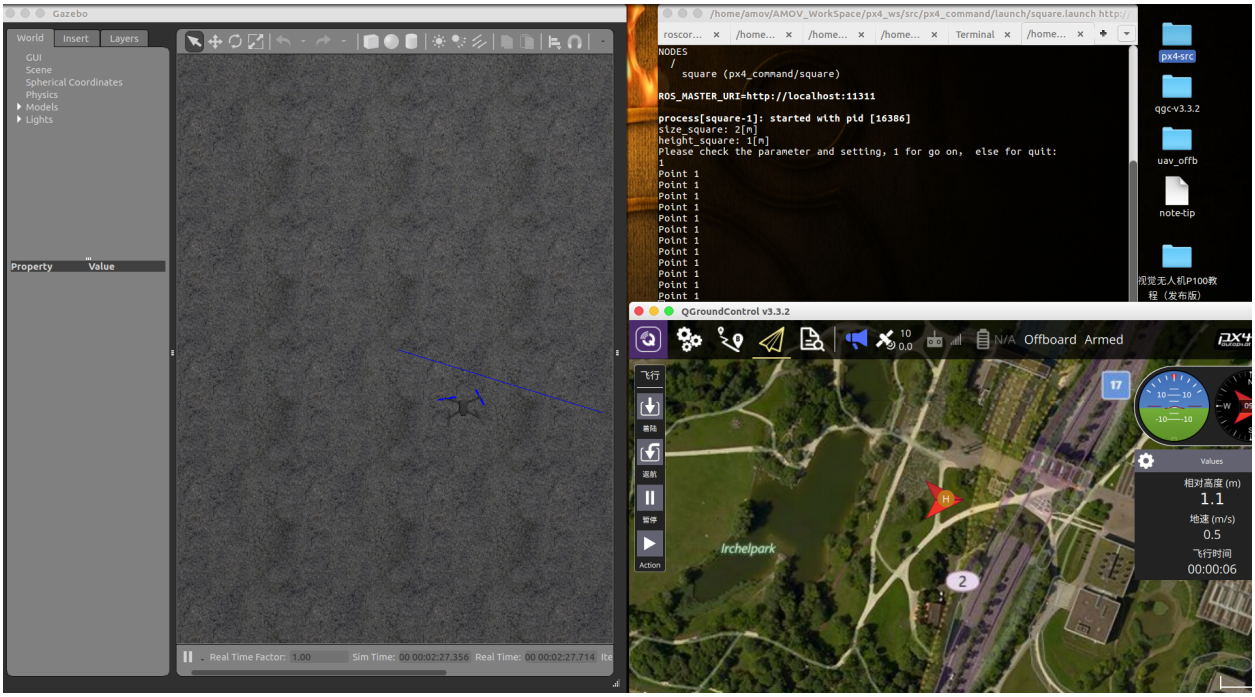
–tab -e ‘bash -c “sleep 2; roslaunch px4_command px4_pos_controller.launch; exec bash”’

–tab -e ‘bash -c “sleep 2; rosrun px4_command set_mode; exec bash”’

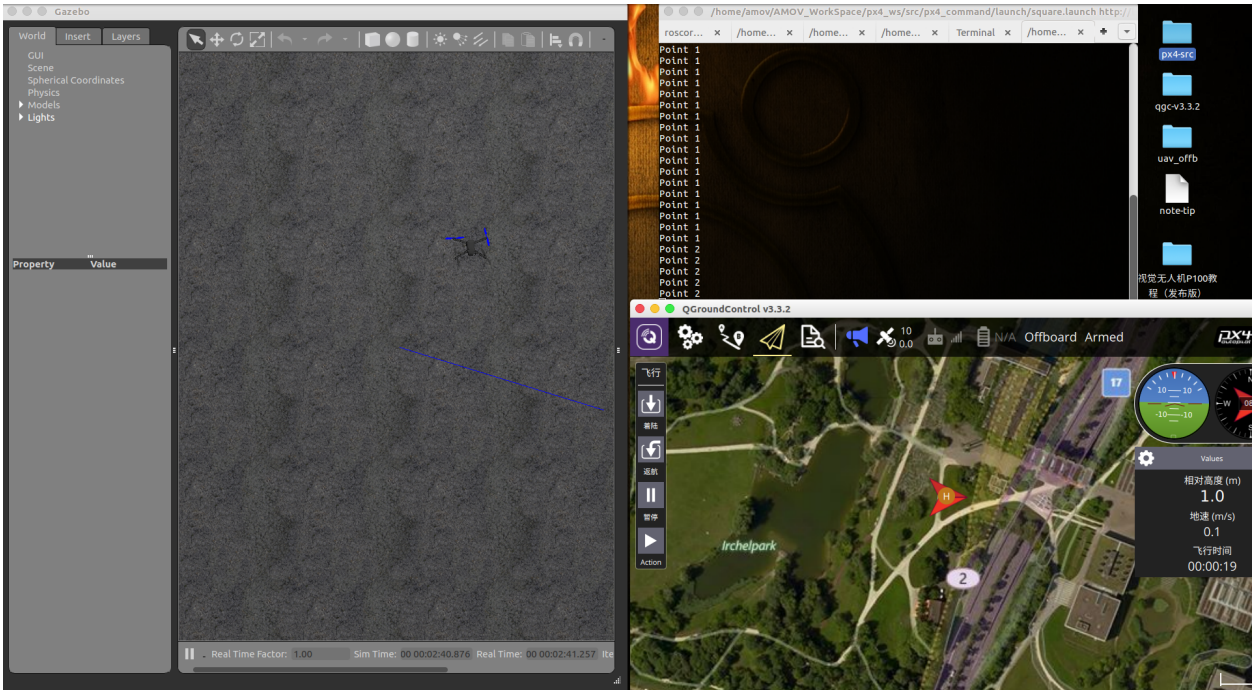
–tab -e ‘bash -c “sleep 2; roslaunch px4_command square.launch; exec bash”’

正常启动 sitl_gazebo_square.sh 脚本。确定并初始化 px4_pos_controller 节点。然后在 set_mode 节点中切换至 offboard 模式。检查 square 节点中，按键 1 执行飞正方形。最后在 qgc 中解锁飞机，飞机正常按照 Point 点进行飞行。

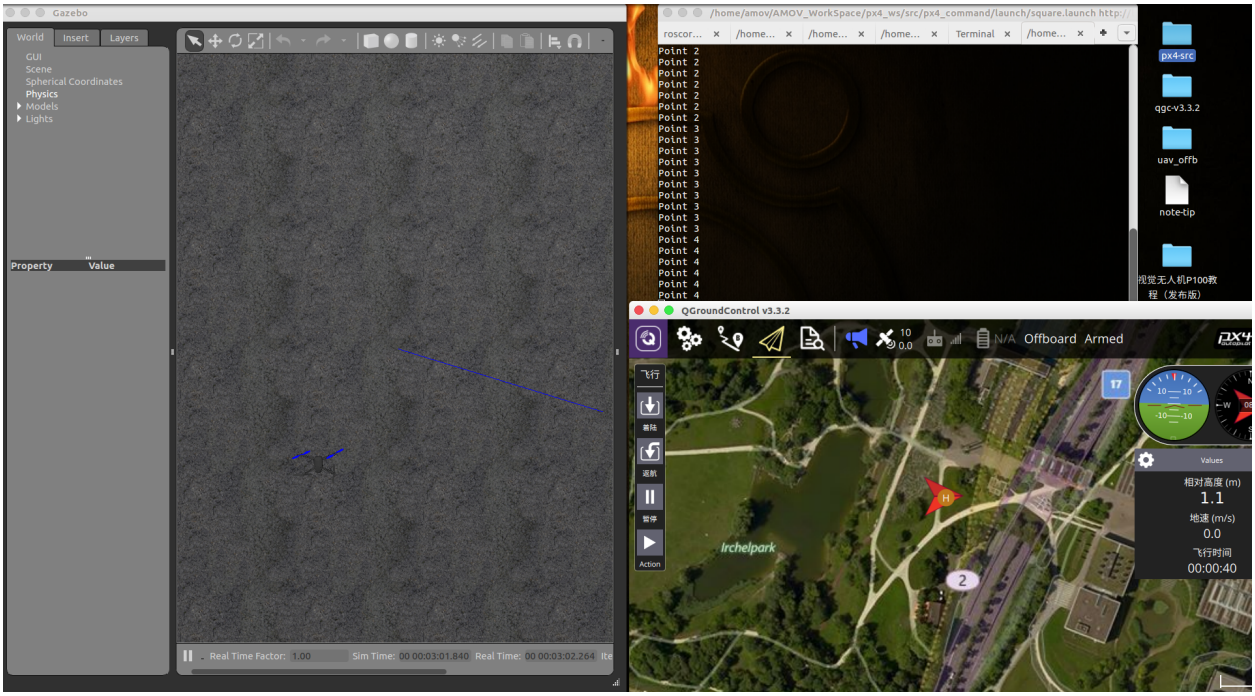
在飞机飞正方形的时候, 有 5 个 point 点的设置, 飞行过程部分截图如下 point1:



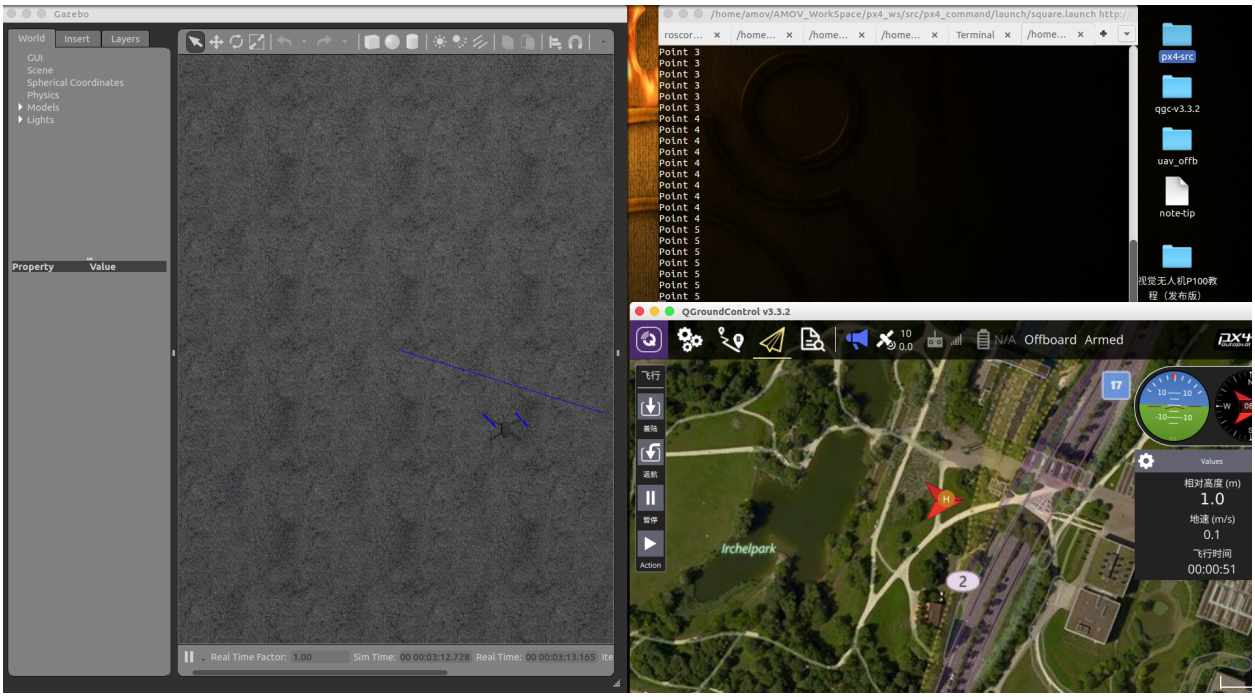
point2:



point4:



point5:

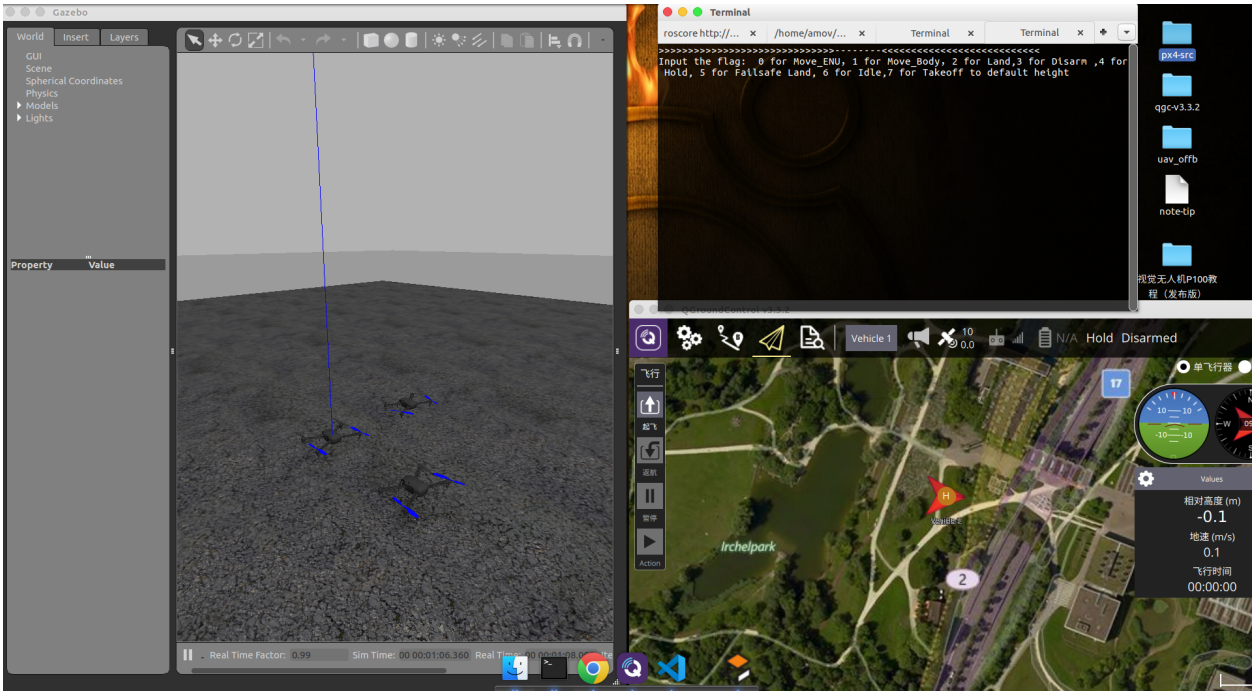


3. 脚本 sitl_gazebo_formation.sh

下载下来的 px4_command 也可能不能直接进行多机仿真, 在自己本机下面的固件代码中的 launch 文件需要改一下名称, 可能没有 three_uav_mavros_sitl.launch. 需要将现有的 multi_uav_mavros_sitl.launch 改为 three_uav_mavros_sitl.launch . 运行仿真之后可能只出现两架飞机, 原因是, 在 px4_command 中的多

机仿真用的是 uav0,uav1,uav2, 而在你下载的固件代码中只有 uav1,uav2. 没有 uav0, 这时候你需要手动添加一个 uav0 出来, 才能多机 (3 架飞机) 仿真跑起来.

正常启动 sitl_gazebo_formation.sh，在启动正常的情况下（qgc 可以连接上三个飞机），此时确认 formation_control 节点并初始化，按照 ENU 坐标系下，设置坐标点，三架飞机同步执行动作。如下图:



存在 Bug 描述:

- 启动脚本失败（已将时间由 2 改为 4 ，成功启动概率增大）
- 确认初始化 formation_control 节点之后，打印信息有问题。UAV2 显示未连接，解锁状态无响应，飞行模式无显示
- 飞机解锁之后，设置好第一个坐标点，飞机起飞，相互位置会有所调换，然后悬停至稳定
- 使用 land 模式之后，有的飞机会直接失控，有的会缓缓降落。
- 飞机执行 land 落地之后飞行模式在 pos 与 RTL 之间频繁切换

AirSim 仿真

第一节 AirSim 介绍

AirSim 是一款为飞行器, 汽车, 甚至更多的一种仿真器, 而 UE4 则是为这款仿真器提供完整的环境, 比如森林, 道路等. 它是一款开源的, 跨平台的, 且支持现有的 px4 飞行控制器硬件在环仿真, 提供了物理和视觉逼真的模拟. 它是作为 UE4 插件开发的, 先从仿真环境搭建开始.

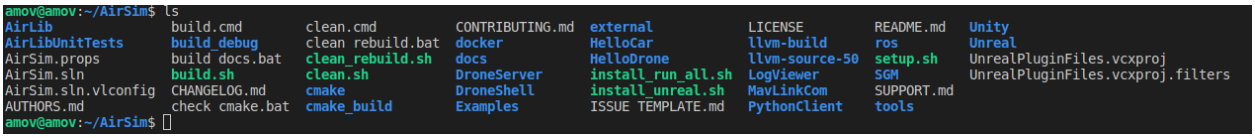
第二节基于 Ubuntu16.04 下的环境搭建

1 AirSim 环境

a AirSim 环境 github 安装

```
git clone https://github.com/microsoft/AirSim.git
cd AirSim
./setup.sh
./build.sh
```

下载好 git 库之后, 可以看到里面有 setup 和 build 脚本. 安装完成即可.



b UE4 环境

小技巧: 注意要去 epic game 官网绑定你的 epic game 账号和 github 账号, 一定要在留的邮箱中点击确认, 如果没有收到邮件, 则去 github UE4 首页手动确认

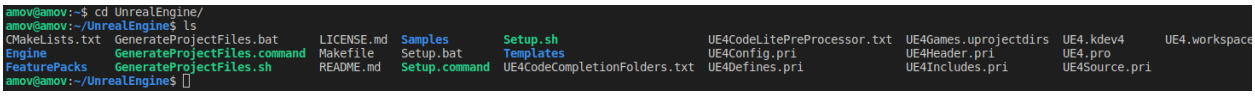
- 具体绑定方法可以去 UE4 与 github 教程
- 网络环境一定要好, 不然可能下载不下来

```
git clone -b v4.18 https://github.com/epicGames/UnrealEngine.git
```

- 这里输入的是 github 的用户名而不是账号

```
cd UnrealEngine
./Setup.sh
./GenerateProjectFiles.sh
make
```

同样的, 下载好 git 库之后, 可以看见 Setup 和 GenerateProjectFiles 这两个可执行脚本, 依次运行即可.



2 PX4 环境

px4 的环境在之前有讲过, 安装的顺序也是先安装完成了 gazebo 那篇的仿真环境搭建, 之后才开始安装 airsim.

在完成 px4 环境之后, 进入 px4 原生固件准备编译, 本机下的路径:

```
amov@amov:~/Desktop/px4-src/src-1.8.2-sim$ cd Firmware/
amov@amov:~/Desktop/px4-src/src-1.8.2-sim/Firmware$ ls
build           CODE OF CONDUCT.md  Documentation     Firmware.sublime-project  launch    mavlink      platforms      ROMFS   test
cmake           CONTRIBUTING.md     eclipse.cproject  integrationtests          LICENSE   msg          posix-configs  rootfs  test_data
CMakeLists.txt  CTestConfig.cmake   eclipse.project   Jenkinsfile               Makefile  package.xml  README.md      src     Tools
amov@amov:~/Desktop/px4-src/src-1.8.2-sim/Firmware$
```

```
make posix_sitl_ekf2 none_iris
```

编译完成最后出现如下图, 等待 UDP 连接

```
amov@amov:~/Desktop/px4-src/src-1.8.2-sim/Firmware$ make posix sitl ekf2 none iris
ninja: Entering directory `/home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/build/posix sitl ekf2'
[2/2] cd /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/build/posix sitl ekf2/tmp && /home...rc-1.8.2-sim/Firmware /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/build/posix sitl ekf2
args: /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/build/posix sitl ekf2/px4 posix-configs/SITL/init/ekf2 none none iris /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware /home/amov/Des
ktop/px4-src/src-1.8.2-sim/Firmware/build/posix sitl ekf2
SITL ARGS
sitl bin: /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/build/posix sitl ekf2/px4
rcS dir: posix-configs/SITL/init/ekf2
debugger: none
program: none
model: iris
src path: /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware
build path: /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/build/posix sitl ekf2
SITL COMMAND: /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/build/posix sitl ekf2/px4 /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/
posix-configs/SITL/init/ekf2/iris
data path: /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware
commands file: /home/amov/Desktop/px4-src/src-1.8.2-sim/Firmware/posix-configs/SITL/init/ekf2/iris
50 WARNING: setRealtimeSched failed (not run as root?)


|    \ \ \ / /   /   |
| | / /  \ V /   / /| |
|   /   /   \  /  / | |
| |    / /^\ \ \    | |
\ |    \/   \/     | /

px4 starting.

INFO  [dataman] Unknown restart, data manager file 'rootfs/fs/microsd/dataman' size is 11405132 bytes
+ COM OBL ACT: curr: 2 -> new: 1
  LPE LAT: curr: 47.3977 -> new: 47.6415
  LPE LON: curr: 8.5456 -> new: -122.1402
INFO  [simulator] Waiting for initial data on UDP port 14560. Please start the flight simulator to proceed..
```

注解: 之前已经编译过, 所以这次编译会直接出现结果, 没有编译过程

3 mavros 安装

mavros 的安装之前在 gazebo 仿真搭建文章也都有讲过, 如果是第一次直接安装 airsim 仿真环境, 则需要看看之前的 gazebo 仿真环境搭建过程; 如果您已经完成了 gazebo 仿真环境的搭建, 那么该步骤可以忽略.

第三节使用说明

在使用之前, 关闭所有与 airsim,UE4,px4 有关的窗口

1 修改 json 文件 (在 document/AirSim 文件夹中)

```
amov@amov:~$ cd Documents/
amov@amov:~/Documents$ ls
AirSim  QGroundControl  Unreal Projects
amov@amov:~/Documents$ cd AirSim/
amov@amov:~/Documents/AirSim$ ls
settings.json  settings ros.json  settings shanghaijiaoda.json  settings S.json
amov@amov:~/Documents/AirSim$
```

修改为如下即可

```
{
    "SeeDocsAt": "https://github.com/Microsoft/AirSim/blob/master/docs/settings.md",
    "SettingsVersion": 1.2,
    "SimMode":"Multirotor",
    "Vehicles":{
            "PX4":{
                    "VehicleType":"PX4Multirotor",
                    "UseSerial":false
        }
    }
}
```

2 SITL 连接 UE4,QGC

a 进入到 PX4/Firmware 目录下, 编译

```
make posix_sitl_ekf2 none_iris
```

修改配置文件 PX4/posix-config/SITL/init/ekf2/iris

编译 iris 文件, 添加以下几点:

- 设置 GPS 原点与 AirSim 中一致

```
param set LPE_LAT 47.641468
param set LPE_LON -122.140165
```

- 在 offboard 模式下执行完指令后自动悬停

```
param set COM_OBL_ACT 1
```

- 在仿真下需要添加下面参数, 但在实机飞行中, 需要禁用掉

```
param set NAV_RCL_ACT 0
param set NAV_DLL_ACT 0
```

接下来是打开 SITL, 进入到 PX4/Firmware 目录下, 如下图

```
./build/posix_sitl_ekf2/px4 ./posix-configs/SITL/init/ekf2/iris
```

```
amov@amov:~/Desktop/px4-src/src-1.8.2-sim/Firmware$ ./build/posix sitl ekf2/p
platforms/ px4
amov@amov:~/Desktop/px4-src/src-1.8.2-sim/Firmware$ ./build/posix sitl ekf2/px4 ./posix-configs/SITL/init/ekf2/iris
commands file: ./posix-configs/SITL/init/ekf2/iris
60 WARNING: setRealtimeSched failed (not run as root?)


|     \ \ \ / /   /   |
| | / /  \ V /   / /| |
|    /   /   \  / / | |
| |     / /^\ \ \     |
\ |     \/   \/     | /

px4 starting.

INFO  [dataman] Unknown restart, data manager file 'rootfs/fs/microsd/dataman' size is 11405132 bytes
INFO  [simulator] Waiting for initial data on UDP port 14560. Please start the flight simulator to proceed..
```

- 编译完成之后, 等待 UDP 连接
- 打开 QGC
- 打开 UE4Editor
- 查看 PX4 状态

b 控制飞行

进入 px4_command/sh/sh_for_simulation 目录下

```
amov@amov:~/AMOV_WorkSpace/px4_ws/src/px4_command/sh/sh_for_simulation$ ls
airsim_siumulation.sh  run_px4.sh              sitl_gazebo_iris.sh           sitl_gazebo_square.sh          sitl_test.sh
depth.jpg              sitl_gazebo_formation.sh  sitl_gazebo_pos_controller.sh  sitl_jMAVSim_pos_controller.sh
amov@amov:~/AMOV_WorkSpace/px4_ws/src/px4_command/sh/sh_for_simulation$
```

自己新建一个 airsim_simulation.sh 脚本, 添加如下启动文件

```
##sitl_airsim
gnome-terminal --window -e 'bash -c "roscore; exec bash"' \
--tab -e 'bash -c "sleep 2; roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.
↪1:14557"; exec bash"' \
--tab -e 'bash -c "sleep 2; roslaunch px4_command px4_pos_controller.launch; exec bash"'␣
↪\
--tab -e 'bash -c "sleep 5; rosrun px4_command move; exec bash"' \
--tab -e 'bash -c "sleep 2; rosrun px4_command set_mode; exec bash"' \
```

在该目录下, 启动该脚本即可

```
./airsim_simulation.sh
```

之后的飞行操作和 gazebo 下的操作是一样的.

可选配件

可选配件 1–RTK

第一节 px4-rtk 使用介绍

1 参数说明

2 建议飞控参数

第二节市面常用产品对比

1 参数

2 关键参数

3 视频

第三节 RTK 产品介绍

1 产品图片

2 产品参数

- 接收卫星信号 GPS L1 / L2 BDS B1 / B2 GLONASS L1 / L2
- 单频定位 (RMS) 平面 1.5m 高程 3.0m
- 双频定位 (RMS) 平面 1.2m 高程 2.5m
- DGNSS(RMS) 平面 0.4m 高程 0.8m
- RTK (RMS) 平面 10mm +1ppm 高程 15mm +1ppm
- 定向精度 (RMS) 0.2° / 1m 基线
- 速度精度 (RMS) 0.03m/s
- 时间精度 (RMS) 20ns
- 惯性导航精度 < 5% x 行驶距离 (无 GNSS 信号 30s 内)
- 测量精度 (RMS) BDS GPS GLONASS
  - B1/L1 C/A 码 10cm 10cm 10cm
  - B1/L1 载波相位 1mm 1mm 1mm

  - B2/L2P(Y) 码 10cm 10cm 10cm
  - B2/L2 载波相位 1mm 1mm 1mm
- 初始化时间小于 10 秒（典型值）
- 首次定位时间
  - 冷启动：40s（典型值）
  - 温启动：30s（典型值）
  - 热启动：5s（典型值）
- 差分数据 RTCM2.x/3.x CMR CMR+
- 数据格式
  - NMEA-0183
  - Femtomes ASCII 及 binary 格式
- 数据更新 1Hz / 5Hz / 10Hz / 20Hz（可选）
- 组合导航 /IMU 原始数据更新最高支持 200Hz

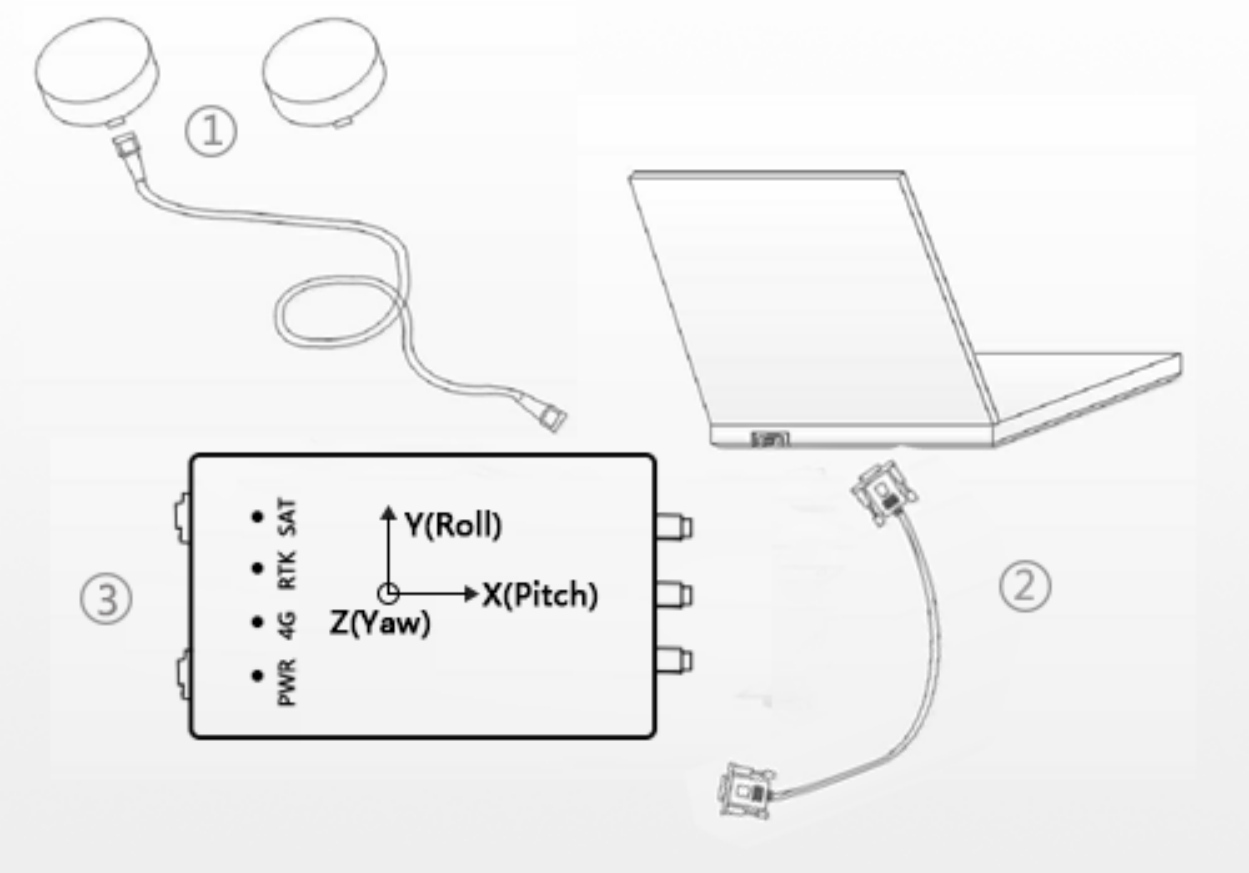
3 使用视频

4 安装要求

a 器材清单

名称	数量	备注
MINI2-D 接收机	1 台	
4G 天线	1 根	
GNSS 天线	2 个	
SIM 卡	1 个	
10 芯电源数据线	1 根	
12 芯电源数据线	1 根	
GNSS 天线馈线	2 个	
网口线	1 根	客户自备
串口线	1 根	客户自备
电脑	1 台	客户自备

b 硬件连接



5 价格，购买链接

6 配套软件

FemtoMonitor.exe

第四节使用方法

1 使用说明

2 配件说明

第五节产品分析方法

第六节讨论区

第七节 github 源码

可选配件 2–网口图传

网口图传分为地面端图传设备与空中端图传设备, 传输距离实际测试过有 2 公里左右.

资料后续有待补充