# 第一阶段总结文档

## 概述

第一阶段从2月24日至3月24日，历时一个月，从最基本的小车硬件系统搭建和TX2开发环境搭建，到后来的逐渐深入工作，基本完成了小车从初始位置到目标点的直线运动任务。

## 上位机与Pixhawk通信--使用DroneKit-Python

DroneKit-Python可以让我们在上位机直接运行Python程序，来方便地实现对Pixhawk各个参数的获取以及对小车的控制。值得注意的是，DroneKit目前只支持Python2，而我们更为熟悉的是Python3，这可能是今后在此基础上继续工作的一个小隐患。

DroneKit的安装十分简单，使用pip可以直接安装。其核心用法是vehicle = connect(‘connect\_string’, wait\_ready=True)，‘connect\_string’可以是虚拟的网络端口，也可以是实际的USB接口，这样vehicle就成为飞控这个大类的一个实例，可以调用其各个属性和内部的函数来获取飞控参数，控制飞控的运动状态。

DroneKit实现对下位机的控制是通过MAVLINK协议的，更具有开发性的用法是vehicle.message\_factory创建一个MAVLINK命令，当然这个命令必须是MAVLINK支持的，因此要熟练使用Dronekit需要对MAVLINK协议有较多的了解。

DroneKit-sitl可以配合mavproxy和地面站软件搭建可视化的仿真系统，最终实现一端运行Python程序，另一端在地面站软件的图形化界面显示程序控制小车的实际效果。可以使用pip方便的安装DroneKit-sitl，地面站软件我们使用的是在Windows下的MissonPlanner，mavproxy按照官网的教程也可以比较容易的安装。全部安装完毕后需要在命令行里运行dronekit-sitl rover，这里是仿真rover的情况，也可以仿真copter、plane等官方支持的系统，也有一些可选参数如dronekit-sitl rover --home=36.364527,120.684733,10,0 设置的初始的位置和朝向。这时就进入待连接状态，再打开另一个命令行，输入mavproxy --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 127.0.0.1:14550 --out 127.0.0.1:14551 即建立本地地址14550端口和14551端口的通信，然后一个端口作为’connect\_string’写入程序，另一个端口在地面站连接，这样就可以实现仿真系统的搭建。

## 硬件清单

|  |  |  |
| --- | --- | --- |
| 元件名称 | 数量 | 参数 |
| NVIDIA Jetson TX2 | 1 |  |
| Pixhawk飞控 | 1 |  |
| GPS模块 | 1 |  |
| 稳压模块 | 2 |  |
| 舵机 | 1 |  |
| 有刷电机 | 1 |  |
| 有刷电机电调 | 1 |  |
| 6通道无线遥控器 | 1 |  |
| 遥控器信号接收器 | 1 |  |
| 18650锂电池 | 1 |  |
| 小车底盘 | 1 |  |

## 硬件连接图



利用NVIDIA TX2的上位机程序对飞控进行控制，Pixhawk接收TX2的命令，根据内嵌的算法和GPS信息相应地控制电调（即后驱动轮）和舵机（及前转向轮）。

## 供电方式

采用18650锂电池对全车进行供电，电源输出电压为12V。电源供电分为两路。一路是电源直接输出12V给TX2供电，第二路是经过降压模块后降至7.2V给飞控和电调供电。