

- 绪论

- ▼ 无人系统的平台、硬件结构

- 四旋翼无人机
- 固定翼无人机
- vtol
- 无人车
- 无人船
- 电子系统（以四旋翼为例，不含大脑）
- "大脑"
- 多旋翼平台设计总体思路
- 固定翼平台设计总体思路
- 无人船平台设计总体思路

- ▼ 无人系统软件结构与常用软件

- windows: GCS (c#)
- linux平台 + 传感器（任务计算单元）
- 嵌入式平台（底层飞控单元）
- 问题：上层中的控制与底层中的控制的区别
- 常用的开发软件
- ROS基本概念（一个管家，你要什么就跟管家说，要传达命令也跟管家说）
- linux命令与常见问题
- ros 需要知道什么？我会启动小海龟，然后呢？
- pcb电路板、单片机软件（思想都是通的）

- ▼ 仿真的逻辑与分模块仿真

- 问题引入
- 实验 vs 仿真
- 仿真适用场景
- 如何让仿真也令人信服？
- 公认的飞行仿真软件？
- 主流的仿真软件
- sitl vs hitl

- ▼ 数学、物理基础

- 刚体运动
- 模型
- 曲线几何
- 向量的运算
- 微分平坦

- ▼ 飞控的使用

- 注意事项

- 飞控设置与试飞步骤
- ▼ 飞行模式、规划、控制
 - 高级指令--> 低级指令
 - 飞行模式
 - 规划
 - 控制 (反馈)
 - 路径规划 (固定翼) vs 轨迹规划(多旋翼)
 - 规划与制导 (个人理解)

- ▼ 规划算法简介
 - 全局规划与局部规划
 - 全局规划算法 (解决连通性)
 - 局部规划算法 (解决可执行性、光滑路径生成)

- 编程的逻辑

- ▼ rpg工程示例

- ▼ 使用flightmare进行算法仿真
 - 测试环境是否安装成功
 - 修改门框的图案(简要过程)
 - 从flightmare中订阅图像
- ▼ 控制算法: rpg_quadrotor_control
 - 基础概念
 - ▼ autopilot
 - Subscribed Topics
 - Published Topics
 - ▼ sbus_Bridge
 - Subscribed Topics
 - Published Topics
 - Trajectory Generation Framework 运动规划模块
 - ▼ Manual flight assistant
 - Subscribed Topics
 - Published Topics
 - 测试controller环境是否安装成功 (没有flightmare)
- 实验相关基础
- ▼ mpc控制算法: rpg_mpc (文档不全)
 - 软件安装
 - 工程结构
- 总结

绪论

1. 无人系统：

车模、船模、航模(rc) + 自主驾驶系统(飞控) => 无人车、无人船、无人机

2. 功能： 娱乐、爱好 => 完成特定功能、任务

3. 自主导航系统架构：

感知（决策（任务分配、组合优化））规划（全局规划、局部规划）控制（大小脑结构） vs 导航制导控制（传统飞控）

- 感知：状态估计，知道我在哪？
- 决策：我该去哪？我该怎么去？
- 规划：提供一个合理的、可执行的、尽量优的参考（不合理的参考无法被跟踪） 知道我该去哪，可我该怎么去？
- 控制：收敛到规划的结果 既然要这么去，我的执行器应该怎么动？

4. 传感器（内感知、外感知）：

相对自身（机体系）：摄像头、激光（需要大量的处理，里程计）

相对地面（惯性系）：gps、动捕（只需要有限的处理）

惯性测量单元（imu） 平移和转动（加速度计 + 陀螺仪）

注意：正是由于各种传感器不在同一坐标系下，所以才需要各种坐标转换

定位规划通常是在惯性系下的，控制通常是在机体系下的。

5. 优化，数学规划，线性规划、非线性规划、二次规划（二次型）、凸优化（非凸优化）

优化问题的数学表达：

工科生应该会到什么程度？工科生的重点是什么？（问题的构建（建模））

6. 线性系统、非线性系统（线性化）

线性系统的特征：叠加性、其次性

7. 模型，控制模型，物理模型（根据第一性原理建立的模型 vs 简化模型（简化到什么程度？））

8. 完整约束、非完整约束、运动学约束、动力学约束

9. 约束（限制）：力、力矩、速度、加速度、角度、角速度、能量转换（固定翼飞机失速）、曲率约束、障碍物约束……

10. 欠驱动、全驱动

11. 仿真，软件在环仿真、硬件在环仿真（飞行模拟器的练习，也是一种仿真）

实际操作飞机之前必须要练习模拟器！！

12. 什么是控制（收敛）

13. 什么是（有限）状态机？**state machine**

if else、switch case

增稳模式、自主飞行模式、跟踪模式

举例：自主起飞->任务执行->自主降落

因此，无人系统领域的工作逻辑：

场景（问题） --> 环境 + 对象 --> 抽象出问题（建模） --> 算法及证明 --> 仿真与实验

- 适应某个场景的新平台（布局等）
- 适应某个场景的新方法

无人系统的平台、硬件结构

四旋翼无人机

- 四个通道 + mixer -> 四个电机 -> 三个平移 + 三个旋转
- 原理：俯仰（前后）、滚转（左右）、偏航（转头）、油门（上下）

固定翼无人机

- 4个通道（忽略其它（不直接参与姿态控制））一个电机 + 3个舵机 -> 一个速度 + 3个旋转
- 原理：俯仰（升降舵）、滚转（副翼）、偏航（转头）、油门（速度）
- 油门控高（类似TECS） vs 油门控速）
- 协调转弯原理

vtol

1. 四旋翼 + 固定翼
2. 倾转
3. tailsitter

无人车

- 两个通道一个舵机 + 一个电机 -> 前进 + 转弯

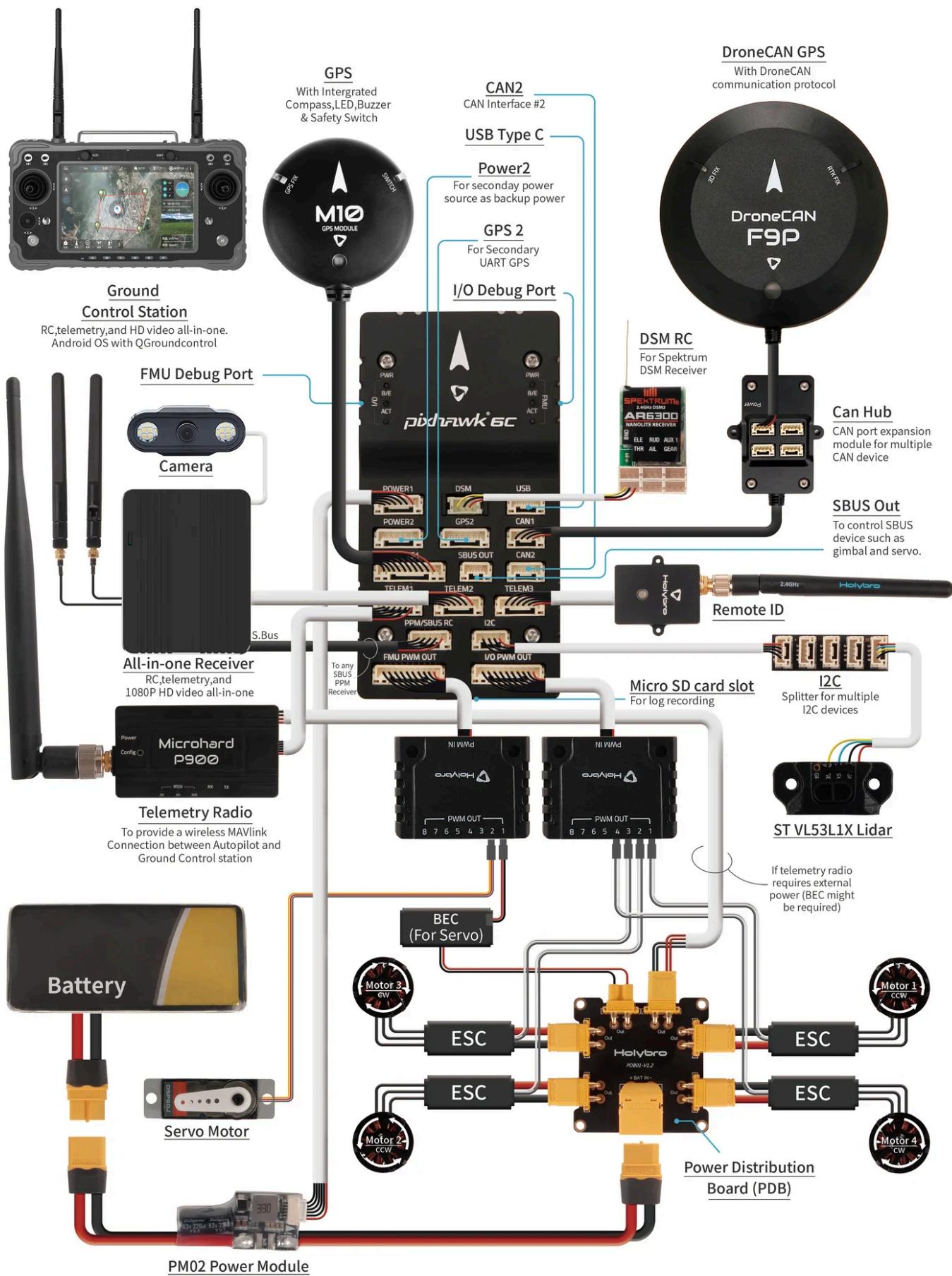
无人船

1. 差速电机
2. 船外机（推进+转弯）推进电机不固定
3. 固定电机+舵

电子系统 (以四旋翼为例，不含大脑)



pixhawk® 6c



"大脑"

linux + ros 上层任务计算单元

视觉等传感器

大脑：有意识的思考

小脑：动作执行（条件反射，来一个命令执行一个命令）

在ros架构中：飞控是ros的一个外设（驱动肌肉运动），小脑和肌肉是封装的，上层任务不用关心底层是如何执行的

多旋翼平台设计总体思路

机架设计要点：刚性（强度足够，连接牢靠）

1. 功能 -> 负载（传感器+任务载荷）
2. 续航 <-> 动力选配（拉力需求（电机、电调）+电池）

起飞重量 = 50% * 所有电机最大推力（一般不超过60%）

(水多了加面，面多了加水) 但最终，限于锂电池的能量密度等因素，旋翼机的续航上限并不高，并非一个大电池就能飞很久很久

因此，在长航时环境中，更多选用固定翼

3. 根据螺旋桨尺寸设计机架

固定翼平台设计总体思路

符合“美学”的基本都好飞

1. 任务场景：（续航、载重、动力形式、发射与回收方式、存储方式）--> 结构与启动外形
2. 发射回收：vtol、皮筋弹射伞降、压缩气体筒射
3. 气动外形：大展弦比-->低速、长续航
4. 长续航：内燃机
5. 高速靶机：涡喷
6. 验证机：缩比、木头蒙皮 --> 全尺寸手糊、复合材料 --> 量产机：模具复合材料

无人船平台设计总体思路

主要考虑：坚固、防水

续航 + 抗波性 ——> 外形 + 尺寸

无人系统软件结构与常用软件

windows: GCS (c#)

GCS: 监控无人机状态、发送指令

功能模块: 地图控件、航空仪表控件、控制按键(面板)、任务面板、日志模块、通信模块

linux平台 + 传感器 (任务计算单元)

ros 功能包: 感知、状态估计、规划、**控制**

嵌入式平台 (底层飞控单元)

控制算法 + GPS以及惯性器件的处理

问题: 上层中的控制与底层中的控制的区别

- offboard模式有很多种控制方式, 要看控制的内外环分别放到哪个平台上执行
(不仅有px4预定义的, 只要开发能力足够强, 不同通道也可以任意组合, 比如升降通道用角度, 滚转通道用角速度)
- 发的是位置、速度还是加速度? 角度还是角速度?
- 如果直接“规划”到舵机的输出, 那就相当于直接开环控制了 (大闭环, mpc)

常用的开发软件

vscode, github , plotjuggler, markdown

- vscode
 - 安装插件可以扩展功能
 - ctrl+ 增大字体, ctrl-减小字体
 - ctrl + F, ctrl + f +shift
 - 思考题: 怎么用vscode打开pdf?
- git 要知道几个基础的命令: git init, git status, git add、git commit -m
 - 在vscode中, 知道自己刚改过什么东西, 各种颜色的标记是什么意思? (演示基本特征)

- .git 是隐藏文件，在windows下要显示隐藏文件才能看到
- 在linux下 按ctrl+H可以显示和隐藏
- 直接下载安装包或者使用命令安装
 - 初次使用需要进行简单设置：（不用是你的github账号）
 - git config --global user.email 你的邮箱
 - git config --global user.name 你的名称
- plotjuggler用于绘图（也可以使用matlab, Python等工具进行日志分析）
 - PX4飞控的日志
 - ros话题消息
- 使用markdown记录你的开发日志，方便日后回忆，好处：跨平台（演示基本操作）

一级

二级

加粗

这是内嵌代码 code

这是一段代码：

```
code  
code
```

插入图片？

插入pdf文档链接？

ROS基本概念（一个管家，你要什么就跟管家说，要传达命令也跟管家说）

- “进程间进行通信的管理软件” + 一些基础组件（工具） + “生态”
- 有了ros的几种通信机制，大家就可以比较容易地进行协作，关注自己关注的“模块”（功能包）其它的功能使用别人开发的功能包

linux命令与常见问题

- 我要不要重装系统？会不会被我弄坏了？不会
- 执行完了没有反应？
- 脚本执行一半，网络连接错误，失败了怎么办？多执行几次，网络搜索错误，通常不会导致严重故障
- 常用的几个指令：pwd, ls, cd, mkdir 别的好像基本不需要，可以直接用鼠标操作
都可以直接百度，比如linux 如何查看ip？：ifconfig
- 常用按键：tab(自动命令补齐)

ros 需要知道什么？我会启动小海龟，然后呢？

- roscore,rosrun, roslaunch
- catkin_make, source ./devel/setup.bash
- rqt_graph
- rostopic list, rostopic info, rostopic type, rostopic echo
- rosnode list, rosnode info
- ros的工作空间与文件存放路径，可以在任何地方

pcb电路板、单片机软件（思想都是通的）

- 封装--布板--后处理（丝印等）--加工--底层驱动--上层应用
- 零件--组合体--加工--组装
- 库：lib,封装库、函数库.....

仿真的逻辑与分模块仿真

问题引入

1. 我研究了一个算法，它到底行不行？
2. 你研究的算法，你说行，你能不能证明给我看？

实验 vs 仿真

实验：行就是行，不行就是不行。

仿真：仿真行，实验也不一定行，为什么？

1. 仿真环境理想，你没有延时、扰动、测量误差等

2. 模型是不是也过度简化了？甚至是不合理

- 用小车的模型来描述船
- 用多旋翼来模拟固定翼

仿真适用场景

实验：

物料、交通等实验条件低（如，摄像头识别一个茶杯）

仿真：

有些场景只能先仿真，最后再实验：（甚至只能仿真）

导弹、火箭、车辆、飞机、无人机等，各方面成本高

如何让仿真也令人信服？

1. 使用公开的数据集
2. 使用**公认的仿真软件**（你说你的算法会开飞机，我给你一个飞机，让你的算法开给我看）

公认的飞行仿真软件？

1. 前提是什么？

能与“我的控制系统”构成完整的闭环，也就是说，能“平替”“我的飞行器”，既能接收我的控制信号，又能给我发送我要的状态。

例如，你可以通过游戏摇杆来操纵游戏中的“F22战斗机”，如果你的“飞控”也能向电脑发送控制指令，也能飞好“F22”。

2. 同时我还希望：

能够直观地体现控制效果（最好有图形界面，日志记录等功能）

因此，**仿真软件 = 渲染引擎 + 动力学模块（模型） + 输入输出接口（UDP?TCP?串口? 其它形式?）**

主流的仿真软件

airsim
flightgear
gazebo
flightmare
等等 (见apm官方网站)

sitl vs hitl

软件在环仿真：

上层应用更多使用软件在环

硬件在环仿真 (半实物仿真) :

底层更多使用硬件在环 (飞控)

数学、物理基础

刚体运动

- 平动+转动
- 坐标系与坐标变换：举例： v_b ? v_e
- 牛顿方程： $f = ma$
- 欧拉方程： $J\dot{w} + w \times Jw = M$
- $so(3)$ 、 $se(3)$

模型

- 运动学模型： 描述运动 (质点)
- 动力学模型： 描述力与运动 (力、 力矩与运动) 加速度， 角加速度

曲线几何

- 曲线是如何描述的？
- 参数连续性 与 几何连续性 (更严格)
- 折线段 vs dubins路径 vs 贝塞尔曲线等

向量的运算

- 点乘-->标量 (求向量夹角) (投影)
- 叉乘-->向量(右手系中，两个基叉乘得到第三个基)
- 向量求导 (出现奇奇怪怪的东西，主要是因为坐标系)
一个点p在机体系中，机体系在运动 (转动)，这个点相对于惯性系怎么动?
 - i. 在机体系下看p怎么运动
 - ii. 机体系的运动对p的影响

微分平坦

1. 四旋翼的状态和输入之间是耦合关系 (微分约束)，并非独立。
因此，这些状态中的有些状态之间是“紧紧绑定”的。
2. 牛顿欧拉方程就是系统的“约束”，就是推导和证明的前提和基础。
3. 选出几个有代表性的 (独立自由的、通过他们能把其他状态推导计算出来的，形象直观的，好计算的) 来连接规划和控制模块。

x,y,z,psi

4. 证明和推导的思路：
有了这四个状态，其他的状态都是怎么表示的，如果表示出来了，那就是证明完成了。
对牛顿欧拉方程进行求导，看得到什么。
5. 主要基础：向量求导，向量叉乘

飞控的使用

注意事项

调试的基本前提：**连接无误、调试过程卸除螺旋桨**

- 电机-电调-飞控 (尤其是电调与飞控的通道连接)
- 电池-电源模块-分电板-电调-飞控
- 遥控器接收机-飞控 (协议，协议不对检测不到遥控器的动作)
- 安全：电池正负极与螺旋桨
- 接线正负极、电压 (24v、12v、**5v、3.3v**) 、接口(引脚)定义等
- 电池有过放问题 (不用不要一直插着，用的时候要经常检查电池电压)
- 初次试飞要栓绳，调试的时候要卸下螺旋桨

注意(仅涉及目前的比赛方案，即，接收机不插飞控，插nx机载电脑)：

sbus信号的电平与是常规串口信号是反的，因此，直接将sbus接收机的信号插到nx电脑上收不到信号，两种处理方式：

- 对于传统的sbus接收机，要中间接一个反向电路
- 对于一些支持sbus 反向输出的接收机，需要在软件上进行设置
- 同样地，nx输出的串口信号如果直接插飞控也收不到，需要将飞控上也设置长sbus反向输出

飞控设置与试飞步骤

1. 刷固件--选择平台类型 (对于不同的飞控，有不同的方式)

- 先刷固件，再选择无人系统类型
- 先选类型，再刷固件

2. 传感器校准 (有些情况需要选择飞控安装方向、gps安装方向)

3. 遥控器设置

- 通道对应 (日本、美国手)
- 通道行程校准
- aux通道选择与飞行模式选择

4. 电机转向检查

如果反向，两种处理方式

- (多个飞机 (可能换飞控)、协作，建议) 改硬件，调换任意两根线
- (自己玩建议) 改软件

5. 解锁、上锁，失控保护检查

6. 反馈检查 (确保移除螺旋桨)，手持无人机进行转动，检查是否具有正常的电机加减速反馈

7. 需要有经验的操控人员进行试飞

8. 拴上绳子进行试飞

9. 飞控需要调参

不同的飞机具有不同的动力配置等，默认的参数并不能完全控制好飞机，因此需要调参

注意 如果是成熟的机型，可以使用别人调整好的参数

如果是自己设计的机型 (机架尺寸、桨尺寸、电机参数等不同)，需要自己调参

10. 确保遥控模式没有问题，再考虑自动飞行等飞行模式

- 确保平台是好的
- 危险状态下能及时挽救

11. 确保日志的记录与分析过程流畅，否则炸机是没有意义的，经济、劳动、时间成本都是白费的，只能得到一个结论:**不行**，而无助于debug和改进

尤其是实验比较困难的情况下，只能回来分析数据，更要注重“原始数据”的记录，传感器读回来的是什么，自己经过处理融合、坐标变换，这个过程中可能数据就已经错了，则必须查看原始数据。debug、修改程序算法之后再进行外场实验。

tips: 如果仿真环境下，**规划和控制算法**已经充分验证过了，出问题的基本也只能是**传感器数据的读取、处理、传输!!**

可能问题：

- 赋值（读取）出错
- 大小端出错（高低位）
- 类型出错（精度）
- 位数出错（4字节 8字节）
- 放大、缩小系数出错

飞行模式、规划、控制

高级指令--> 低级指令

思考：描述无人机智能化程度（自主性程度）

1. 无人机避开雷达去北京
2. 沿着航线去北京
3. 下降100米
4. 推杆、加油门

飞行模式

- 手动模式 (rc)：
手动遥控（时刻高度集中）
- 增稳模式(飞控)：
自动保持姿态（可以暂时离开视线）
- 航线模式（传统的自主飞行模式）：
人规划航线（位置），无人机负责执行（解放人力）

趋势：自主规划：

targetP(生成航线) -->p (航线) --> theta (增稳) --> omega (手动)

规划

提供合适的参考值

控制 (反馈)

跟踪给定的参考值

路径规划 (固定翼) vs 轨迹规划(多旋翼)

区别：是否带有时间信息

规划与制导 (个人理解)

- 规划的概念范围更大，可以规划位置、速度、加速度等，一般是一个区间的都给出来
- 制导的概念更明确具体，生成当下时刻的动作指令，算力消耗通常更低

规划算法简介

全局规划与局部规划

global planning vs local planning

全局规划算法 (解决连通性)

- 栅格地图路径搜索(a^*)
深度优先、广度优先搜索算法
- 采样算法
prm vs rrt
- lattice(主要针对非完整约束)
DWA
sl坐标系 (frenet坐标系, 道路坐标系)

局部规划算法 (解决可执行性、光滑路径生成)

- 基于bezier曲线
- 基于dubins曲线
- 基于多项式曲线

控制点参数<-->多项式系数的关系 (最终都是多项式)

- 安全走廊中的路径生成

minimumsnap(经典算法)

核心就是如何构建优化问题，达到我期望的指标（代价）最优

- 目标：snap最小
- 约束：pva的约束
 - 边界条件（起始点、最终目标点的状态约束）
 - 中间连续性约束（位置及位置导数连续）
- 求解：数值解还是闭式解（解析解）

编程的逻辑

1. 问题引入：编程是干什么的？

- 让计算机**听懂**人类的命令
- 执行命令
- 输出结果

2. hello_world.cpp

- .h
- main
- std::cout
- 源程序-->加工-->可执行文件

一个文件太大了！要分成一个模块一个模块的

3. .h

- 头文件
- 声明
- include .h
- 源程序-->翻译-->链接--> 可执行文件

4. 函数重名怎么办？

- 名字空间：：
- 名字相同的看路径（类似文件夹中的文件不能重名）

5. 合成类型（自定义类型）

- struct

- odom(里程计中包含：位置、速度、角度、角速度)
6. 类型不一样怎么办?
 - 模板 template
 7. 面向对象 vs 面向过程
 - 炒菜 (洗锅-烧火-放菜-盛出来)
 - 人，有锅碗瓢盆这些变量，还有炒菜的方法 (只对外暴露接口，点菜)
 8. 构造函数
 9. 库
 - 标准库
 - 三方库
 10. cmake
 - 如何组织那么多的文件、库等
 11. 在别人的基础上修改，换言之，不要自己从头写 (除非很简单的小程序)

rpg工程示例

日期：2024.12.8 本文档随时更新，如有问题直接联系：15201614600（微信同号）

12.10更新比赛门框

12.13更新比赛四个门框

12.14更新官方文档阅读

flightmare 仿真环境及其组件的使用说明

官方代码及文档：

<https://github.com/uzh-rpg/flightmare>

https://github.com/ethz-asl/rotors_simulator

<https://github.com/uzh-rpg/flightmare/wiki/Basic-Usage-with-ROS>

详细使用文档：

<https://flightmare.readthedocs.io/en/latest/>

安装方式：

ROS

遇到的主要问题：

1. 代码clone不下来，所以只要有 git clone 的地方就会报错（命令行窗口设置过梯子的忽略这个问题）

解决办法：应该是梯子设置的问题，只能直接从浏览器下载，下载之后解压到工作空间中，主要是注意命名和路径问题!!

2. c++ 11 => c++ 17 (别人好像没有这个问题?)

- 一般编译期的报错，才会出现大量的奇奇怪怪的报错，一般百度可解决
- 执行期的报错一般是程序自身问题

3. error: 'mavlink_status_t' has not been declared

解决办法：

找到rotors_simulator/rotors_hil_interface/include/rotors_hil_interface/hil_interface.h

```
##ifndef MAVLINK_H
typedef mavlink::mavlink_message_t mavlink_message_t;
##include <mavlink/v2.0/common/mavlink.h>
##endif
```

增加一行

```
##ifndef MAVLINK_H
typedef mavlink::mavlink_message_t mavlink_message_t;
typedef mavlink::mavlink_status_t mavlink_status_t;
##include <mavlink/v2.0/common/mavlink.h>
##endif
```

使用方法：

1. catkin_make 或者 catkin build
2. source ./devel/setup.bash
3. roslaunch flightros rotors_gazebo.launch (手动控制飞机)

遥控器设置方法：

1. 遥控器插到电脑上 (游戏摇杆，或者sbus_bridge需要设置，顺藤摸瓜，下面以模拟器手柄为例)
2. 查看节点列表: rosnode list (joy_node)
3. 查看节点信息: rosnode info (joy_node)
4. 查看话题信息，谁发谁收 rostopic info /hummingbird/joy

订阅者: /hummingbird/manual_flight_assistant

5. 显示话题内容: rostopic echo /hummingbird/joy :

```
header:  
seq: 1304  
stamp:  
secs: 721  
nsecs: 410000000  
frame_id: "/dev/input/js0"  
axes: [-0.0, -0.0, 0.6169444918632507, -0.0, -0.2660137712955475,  
0.0, 0.0, 0.0] (摇杆)  
buttons: [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 0] (开关)  
动摇杆看数组中的哪个数字动了，就是对应的通道
```

6. 修改节点相关文件：

manual_flight_assistant

摇杆和开关需要根据操控手的操作习惯设定（美国手，或者日本手）

代码中使用了4个摇杆通道和2个开关通道

spectrum模拟器手柄： 摆杆： yaw: 0, pitch:1, roll:3, throttle:4. 开关： A: 0 H:11

修改示例：

/src/rpg_quadrotor_control/utils/manual_flight_assistant/include/manual_flight_assistant/joypad_a
xes_buttons.h

```

##pragma once

namespace manual_flight_assistant {

namespace joypad {

namespace axes {

static constexpr uint32_t kX = 3;
static constexpr uint32_t kY = 1;
static constexpr uint32_t kZ = 4;
static constexpr uint32_t kYaw = 0;
} // namespace axes

namespace buttons {

static constexpr uint32_t kGreen = 0; //start
static constexpr uint32_t kRed = 1;
static constexpr uint32_t kBlue = 11; //land
static constexpr uint32_t kYellow = 3;

static constexpr uint32_t kLb = 4;
static constexpr uint32_t kRb = 5;
} // namespace buttons

} // namespace joypad

} // namespace manual_flight_assistant

```

7. 还有一个正反舵问题，在manual_flight_assistant.cpp中修改正负号
8. 如果感觉响应迟钝，在yaml文件中修改速度最大值：

```

vmax_xy: 5.0
vmax_z: 3.0
rmax_yaw: 3.0

```

9. 对于sbus信号，可参考代码进行设置
10. 起飞步骤：

- roslaunch 启动
- 依次点击图形化界面的 connect, arm_bright, start 按钮（瞥了一眼，感觉多伦多参赛队用的就是这个界面）
- 可进行手柄操纵飞行

使用flightmare进行算法仿真

测试环境是否安装成功

运行钻门框的示例： rosrun flightros racing.launch

示例中，直接生成了一个minimum snap轨迹，然后直接从轨迹上取点赋值给无人机和门框，然后发给flightmare显示，没有控制模块。

如何放置门框： setPosition

更多门框函数：https://flightmare.readthedocs.io/en/latest/cpp_references/gate.html##cpp-gate-ref

修改门框的图案(简要过程)

文档：https://flightmare.readthedocs.io/en/latest/building_flightmare_binary/standalone.html

- 常见安装问题解决（不专门做场景的话，不需要安装Unity和建模软件blender）：
 - 使用hub软件下载极慢，过一会还停止（应该是网络环境问题，校园网应该好些，只能多试几次，使用浏览器下载不会停止，或者可以刷新）
 - hub找不到指定的版本2020.1.10，最后装了2020.3.x
 - 如果是windows平台，要选择linux 平台支持
如果在windows平台直接装了软件2020.1.10，然后用locate定位安装位置，则编译选项无法选择linux平台（可以编译windowds平台应用）
 - 无法启动hub软件 (.deb不会发生 .appimage发生了)
Managed a little workaround:
. /UnityHubBeta.AppImage --appimage-extract
this will create a “squashfs-root” folder from there, you can start the unityhub script with the redirect link as a parameter:
. /unityhub unityhub://login/?code=...
The unityhub:// part is attainable from the html page source code when the xdg-open fails after the first sign in attempt.
- rpg给的编译好的二进制包是有四个场景的，但是他们开源的工程里面只有一个他们自己做的场景
原因：warehouse等几个场景是收费的(商业软件组件)，所以需要单独买仓库场景

换门步骤：

- 从Unity购买仓库场景(付费插件)，从rpg给的工程导入场景

https://flightmare.readthedocs.io/en/latest/building_flightmare_binary/faq.html

<https://assetstore.unity.com/packages/3d/environments/industrial/hangar-building-modular-142104>

- 美图秀秀抠门框，并替换rpg给的工程里面的门框图片

3. 在blender中编辑门框的面板尺寸，生成新的门框，门框文件放入Unity对应目录
4. 在Unity中，进行编译
5. 将编译结果放入ros工作空间中的对应目录：/flightrender/.....
https://flightmare.readthedocs.io/en/latest/building_flightmare_binary/standalone.html##add-scene
6. 运行ros查看结果
 运行钻门框的示例： rosrun flightros racing.launch
 这里面的门框有三个，两个固定一个运动，直接改成四个固定，就变成我们资格赛的场景了！！！
注意：门框的坐标也变了（x轴减去20，否则出现穿墙现象）！！（应该是仓库的初始位置跟rpg编译的设置不一样）

7. a2r 仿真环境搭建：文件名：flight_pilot_a2r.cpp

- i. 首先由于仓库坐标系与赛方给示意图的不同，要进行适当转换

仓库原点 (0, 0) 在门框后面，门框的x轴坐标为负。

四个门框的坐标分别为：

飞机起始点： -2.5, -38.5,

门框1： -2.5, -35.5,

门框2： 2.5, -28.5,

门框3： -2.5, -21.5,

门框4： 2.5, -14.5

- ii. 在launch文件中直接修改无人机初始位置：

```
<arg name="x_init" default="-2.5"/>
<arg name="y_init" default="-38.5"/>
```

- iii. 无人机为无头模式，需要修改为“有头模式”：

摇杆被映射到世界系下的速度，适合新手，但不符合真实的操作场景（除非飞控也设置成无头模式，一般很少有人用）

```
reference_state_.heading += desired_velocity_command_.twist.angular.z * dt;
reference_state_.heading =
    quadrotor_common::wrapMinusPiToPi(reference_state_.heading);
reference_state_.heading_rate = desired_velocity_command_.twist.angular.z;

reference_state_.position[0] = reference_state_.position[0] +
    (reference_state_.velocity[0] * cos(reference_state_.heading)
     - reference_state_.velocity[1] * sin(reference_state_.heading)) * dt;
reference_state_.position[1] = reference_state_.position[1] +
    (reference_state_.velocity[0] * sin(reference_state_.heading)
     + reference_state_.velocity[1] * cos(reference_state_.heading)) * dt;
reference_state_.position[2] += reference_state_.velocity[2] * dt;
```

注意：以上的操作不太真实，应该直接写一个节点把摇杆的角度发布出去，直接接到底层飞控上，用角度模式，应该就比较真实了！！！

iv. 在主循环中显示摄像头的图像

```
void FlightPilot::mainLoopCallback(const ros::TimerEvent &event) {  
    cv::Mat img;  
    rgb_camera_->getRGBImage(img);  
    if (img.empty()) {  
        std::cerr << "Failed to receive image or image is empty!" << std::endl;  
    } else {  
        cv::imshow("img",img);  
        cv::waitKey(1);  
    }  
}
```

从flightmare中订阅图像

官方文档：https://flightmare.readthedocs.io/en/latest/first_steps/sensors_and_data.html
图像组：可以先采用遥控方式测试识别算法，优化延时等问题

控制算法：rpg_quadrotor_control

官方总的文档：https://github.com/uzh-rpg/rpg_quadrotor_control/wiki

基础概念

- **High-Level Control:** p,v position_controller => mode + quaternion + bodyrate + angular_acc + collective_thrust
- **Low-Level Control:** betaflight
- **Autopilot:** position_controller + **state machine**
- **bridge:** autopilot <=> low-level controller i.e. sbus_bridge
- **State Estimate:** => pose + twist
 - pose : x y z + 四元数 (位置 + 姿态)
 - twist : vx vy vz + wx wy wz (线速度 + 角速度)

[nav_msgs/Odometry Message](#)

File: [nav_msgs/Odometry.msg](#)

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.  
# The pose in this message should be specified in the coordinate frame given by header.frame_id.  
# The twist in this message should be specified in the coordinate frame given by the child_frame_id  
Header header  
string child_frame_id  
geometry_msgs/PoseWithCovariance pose  
geometry_msgs/TwistWithCovariance twist
```

Compact Message Definition

```
std_msgs/Header header  
string child_frame_id  
geometry_msgs/PoseWithCovariance pose  
geometry_msgs/TwistWithCovariance twist
```

topic: autopilot/state_estimate

autopilot

position_controller论文: [RAL18_Faessler](#)

[theory_and_math](#)

Subscribed Topics

- autopilot/state_estimate (nav_msgs/Odometry)
里程计
- low_level_feedback (quadrotor_msgs/LowLevelFeedback)
rpg这个有用的信息不多，都是电池，控制模式，没有imu相关的
- autopilot/pose_command (geometry_msgs/PoseStamped)
接收世界系下的目标位置（生成多项式轨迹）
- autopilot/velocity_command (geometry_msgs/TwistStamped)
接收世界系下的目标速度（加了滤波）
- autopilot/reference_state (quadrotor_msgs/TrajectoryPoint)

rpg_quadrotor_common / quadrotor_msgs / msg / TrajectoryPoint.msg 



mfaessle moved packages from rpg_quadrotor_control

b169c9c

Code

Blame

20 lines (12 loc) · 384 Bytes

```
1 duration time_from_start
2
3 geometry_msgs/Pose pose
4
5 geometry_msgs/Twist velocity
6
7 geometry_msgs/Twist acceleration
8
9 geometry_msgs/Twist jerk
10
11 geometry_msgs/Twist snap
12
13 # Heading angle with respect to world frame [rad]
14 float64 heading
15
16 # First derivative of the heading angle [rad/s]
17 float64 heading_rate
18
19 # Second derivative of the heading angle [rad/s^2]
20 float64 heading_acceleration
```

接收从曲线上插值得到的参考点，pvaj等，最好用下面这个，它自己插值

- autopilot/trajectory (quadrotor_msgs/Trajectory)

[rpg_quadrotor_common](#) / [quadrotor_msgs](#) / msg / **Trajectory.msg** 



mfaessle implemented trajectory data structure with corresponding i...

...

6

Code

Blame

21 lines (15 loc) · 480 Bytes

```
1      # Trajectory type enums
2
3      # Undefined trajectory type
4      uint8 UNDEFINED=0
5
6      # General trajectory type that considers orientation from the
7      # neglects heading values
8      uint8 GENERAL=1
9
10     # Trajectory types that compute orientation from acceleration
11     # values and consider derivatives up to what is indicated by t
12     uint8 ACCELERATION=2
13     uint8 JERK=3
14     uint8 SNAP=4
15
16     Header header
17
18     # Trajectory type as defined above
19     uint8 type
20
21     quadrotor_msgs/TrajectoryPoint[] points
```

- autopilot/control_command_input (quadrotor_msgs/ControlCommand)

[rpg_quadrotor_common / quadrotor_msgs / msg / ControlCommand.msg](#) 

mfaessle moved packages from rpg_quadrotor_control

b169c9d · 6 years ago

[Code](#)[Blame](#)

38 lines (28 loc) · 991 Bytes

[Raw](#)  

```
1 # Quadrotor control command
2
3 # control mode enums
4 uint8 NONE=0
5 uint8 ATTITUDE=1
6 uint8 BODY_RATES=2
7 uint8 ANGULAR_ACCELERATIONS=3
8 uint8 ROTOR_THRUSTS=4
9
10 Header header
11
12 # Control mode as defined above
13 uint8 control_mode
14
15 # Flag whether controller is allowed to arm
16 bool armed
17
18 # Time at which this command should be executed
19 time expected_execution_time
20
21 # Orientation of the body frame with respect to the world frame [-]
22 geometry_msgs/Quaternion orientation
23
24 # Body rates in body frame [rad/s]
25 # Note that in ATTITUDE mode the x-y-bodyrates are only feed forward terms
26 # computed from a reference trajectory
27 # Also in ATTITUDE mode, the z-bodyrate has to be from feedback control
28 geometry_msgs/Vector3 bodyrates
29
30 # Angular accelerations in body frame [rad/s^2]
31 geometry_msgs/Vector3 angular_accelerations
32
33 # Collective mass normalized thrust [m/s^2]
34 float64 collective_thrust
35
36 # Single rotor thrusts [N]
37 # These are only considered in the ROTOR_THRUSTS control mode
38 float64[] rotor_thrusts
```

This enables to compute control commands and have the autopilot feed them through to the bridge . In the **default parameters**, this feature is disabled.
用这个应该就可以直接把遥控器的发出去



foehnx added start acceleration

23

[Code](#)[Blame](#)

31 lines (24 loc) · 890 Bytes

```
1 state_estimate_timeout: 0.2 # [s]
2 velocity_estimate_in_world_frame: true
3 control_command_delay: 0.0 # [s]
4
5 start_land_velocity: 0.5 # [m/s]
6 start_land_acceleration: 1 # [m/s^2]
7 start_idle_duration: 2.0 # [s]
8 idle_thrust: 2.0 # [m/s]
9 optitrack_start_height: 1.0 # [m]
10 optitrack_start_land_timeout: 5 # [s]
11 optitrack_land_drop_height: 0.3 # [m]
12 propeller_ramp_down_timeout: 1.5 # [s]
13
14 breaking_velocity_threshold: 0.2 # [m/s]
15 breaking_timeout: 0.5 # [s]
16
17 go_to_pose_max_velocity: 1.5 # [m/s]
18 go_to_pose_max_normalized_thrust: 12.0 # [m/s^2]
19 go_to_pose_max_roll_pitch_rate: 0.5 # [rad/s]
20
21 velocity_command_input_timeout: 0.1 # [s]
22 tau_velocity_command: 0.8 # []
23
24 reference_state_input_timeout: 0.1 # [s]
25
26 emergency_land_duration: 4 # [s]
27 emergency_land_thrust: 9.0 # [m/s^2]
28
29 control_command_input_timeout: 0.1 # [s]
30 enable_command_feedthrough: false
31 predictive_control_lookahead: 2.0 # [s]
```

- autopilot/start (std_msgs/Empty)
- autopilot/force_hover (std_msgs/Empty)
- autopilot/land (std_msgs/Empty)
- autopilot/off (std_msgs/Empty)
关闭电机

Published Topics

- control_command (quadrotor_msgs/ControlCommand)
=> mode + quaternion + bodyrate + angular_acc + collective_thrust
Control command that is sent to the bridge.
- autopilot/feedback (quadrotor_msgs/AutopilotFeedback)
(用处不大)
Feedback message that is e.g. subscribed to by the quadrotor gui to display feedback as described in Basic Usage.

sbus_Bridge

Subscribed Topics

- bridge/arm (std_msgs/Bool)
- control_command (quadrotor_msgs/ControlCommand)
- battery_voltage (std_msgs/Float32)

Published Topics

- low_level_feedback (quadrotor_msgs/LowLevelFeedback)

Trajectory Generation Framework 运动规划模块

生成多项式轨迹 (rpg想重新整理一下)

一个经典工作: [minimum_snap](#)

Manual flight assistant

The manual flight assistant receives inputs from a joypad or a remote control to command the autopilot with **velocity commands** as described in Basic Usage.

Subscribed Topics

- joy (sensor_msgs/Joy) (这是个ros官方包)

Published by a joy node and used to manually command velocities to the quadrotor.

- received_sbus_message (sbus_bridge/SbusRosMessage)

Published by the SBUS bridge and can be used to command the quadrotor as with a joypad with a remote control.

Published Topics

- autopilot/start (std_msgs/Empty)

Sends a start command to the autopilot. (Mapped as described in Basic Usage)

- autopilot/land (std_msgs/Empty)

Sends a land command to the autopilot. (Mapped as described in Basic Usage)

- autopilot/velocity_command (geometry_msgs/TwistStamped)

Sends velocity commands to the autopilot.

注意： rpg这个实现不好，直接就是世界系的速度，无头模式，可以重新实现一下，不过涉及其他模块。

应该让他直接发布摇杆量大小，以及期望的模式，然后在autopilot里面进行响应的处理，然后这种模式也保留

测试controller环境是否安装成功（没有flightmare）

文档：https://github.com/uzh-rpg/rpg_quadrotor_control/wiki/Test-the-Framework

1. 使用catkin build命令：

```
catkin run_tests --no-deps rpg_quadrotor_integration_test
```

运行后会出现一个gazebo页面，四旋翼自己飞行一个预设航线

2. 或者也可以使用catkin_make命令：

```
roslaunch rpg_quadrotor_integration_test rpg_quadrotor_integration_test.launch
```

运行后可以直接遥控无人机

3. rosrun rpg_rotors_interface quadrotor_empty_world.launch

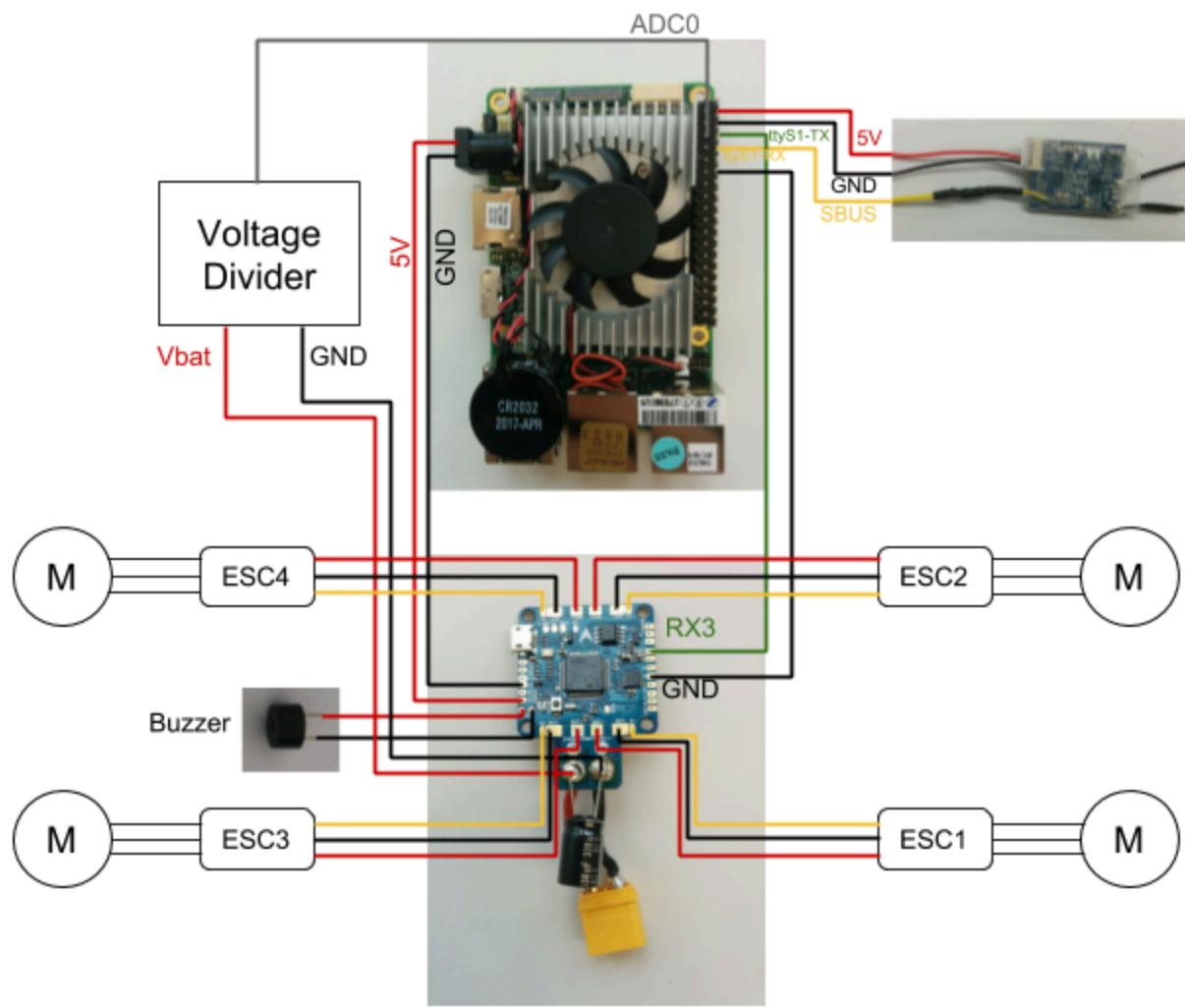
运行后可以直接遥控无人机

实验相关基础

文档：https://github.com/uzh-rpg/rpg_quadrotor_control/wiki/Basic-Usage

1. 硬件组装 (sbus_bridge) (使用了普通的ttl电平，反向sbus电平)

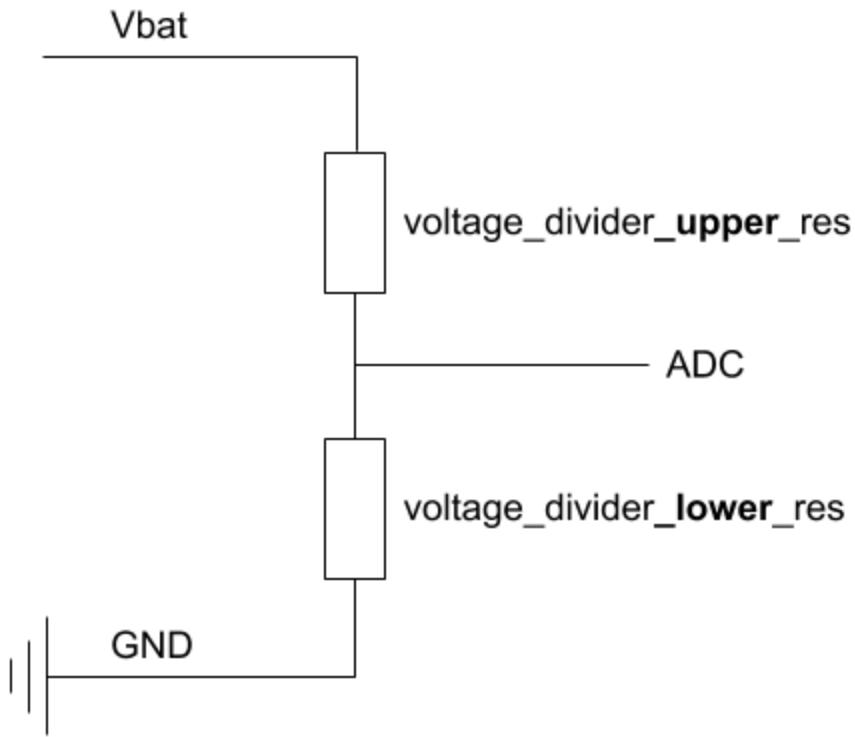
https://github.com/uzh-rpg/rpg_quadrotor_control/wiki/SBUS-Wiring



2. 硬件列表

文档: https://github.com/uzh-rpg/rpg_quadrotor_control/wiki/RPG-Quadrotor-Setup

3. 电压测量: (直接测量电池的总电压, 分压测量)



配置分压参数: `voltage_reader.launch`

```

<?xml version="1.0"?>
<launch>

<node pkg="rpg_single_board_io" type="voltage_reader" name="voltage_reader"
      output="screen">
    <param name="board_name" value="odroid" />
    <param name="adc_id" value="0" />
    <param name="read_voltage_frequency" value="10" />
    <param name="voltage_divider_upper_res" value="11.95" />
    <param name="voltage_divider_lower_res" value="1.21" />
</node>

</launch>
```

4. 启动示例 (launch文件示例) :



foehnx added parameter to allow command feedthrough

Name



..

 base_computer_example.launch

 real_quad_example.launch

The **base_computer_example.launch** launch file starts the gamepad driver and the quadrotor GUI on your base computer. The **real_quad_example.launch** launch file is an example for starting all the necessary nodes on the quadrotor's onboard computer including an autopilot and an SBUS bridge.

mpc控制算法：rpg_mpc（文档不全）

官方文档：https://github.com/uzh-rpg/rpg_mpc/wiki

PAMPC:[IROS18_Falanga](#)

软件安装

http://acado.github.io/install_linux.html.

工程结构

- Solver mpc_solver
- Wrapper mpc_wrapper
- Controller mpc_controller

总结

1. 建立整体的概念，逻辑性十分重要
2. 在某几个点上找一两个例子，仔细体会，一通百通
3. 规律总是相似的，例如，模块化，组织结构，都是一层一层下去的
4. 建立自己的知识图谱和直觉，举例：飞机飞多了，好飞不好飞一眼就看出来了
5. 尽量找到“活字典”，否则，千辛万苦，事倍功半