

Mavros培训课 – Mavros源码

戚煜华

Tel: 18611457441

WeChat: qyp0210

目录



1

• Mavros简介

2

• Mavros代码详解

3

• 自定义代码 – 入门操作

4

• 自定义代码 – 自主飞行

5

• 自定义代码 – 上层开发

Mavros简介

Mavros功能包

- 顾名思义，mavros就是mavlink+ros。mavros是PX4官方提供的一个运行于ros下收发mavlink消息的工具，利用mavros可以发送mavlink消息给飞控（可以控制飞机），并且可以从飞控中接受数据（例如：飞控的位置速度 IMU数据等等）。

Mavros源代码安装及运行

详见： **MAVROS功能包的源代码安装.pdf**

谨记：

mavros包的编译命令是 `catkin_build`
而传统ros包的编译命令是`catkin make`
千万别打错！！！！

Mavros Github主页：

<https://github.com/mavlink/mavros>

Mavros简介 - Mavros功能包结构

Mavros功能包结构

首先打开到src\mavros目录

libmavconn

mavros

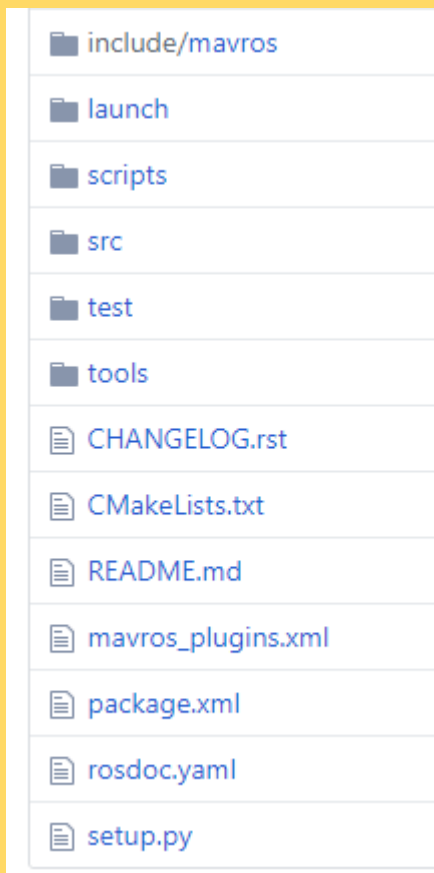
mavros_extras

mavros_msgs

test_mavros

- **libmavconn: 通讯源文件**（飞控板与mavros）
- **mavros: Mavros核心包**（核心源文件，例如读取IMU，发送控制指令）
- **mavros_extras: Mavros附加包**（补充源文件，例如发送mocap消息）
- **mavros_msgs: Mavros自定义消息类型及服务类型**（查看对应消息类型的格式）

Mavros简介 - Mavros功能包结构



打开到src\mavros\mavros目录

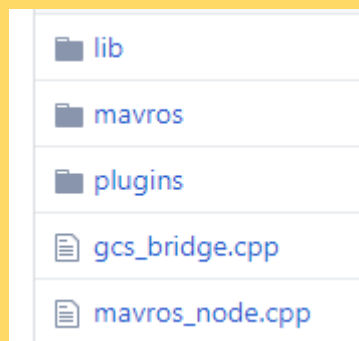
- **include:** 里面是各种头文件的源代码 []
- **launch:** launch启动文件及各种配置文件
- **src:** 核心源文件, 稍后具体介绍
- **CmakeList.txt:** mavros包的配置文件, 相关依赖项的添加及源代码编译都在这里声明 [稍后我所说的修改cmakelist就是修改的这个文件]

此处只介绍我们开发时会用到的一些源文件, 底层的驱动文件及一些编译用的camkelist文件等不做详细介绍, 感兴趣可以自学。

Mavros简介 - Mavros功能包结构

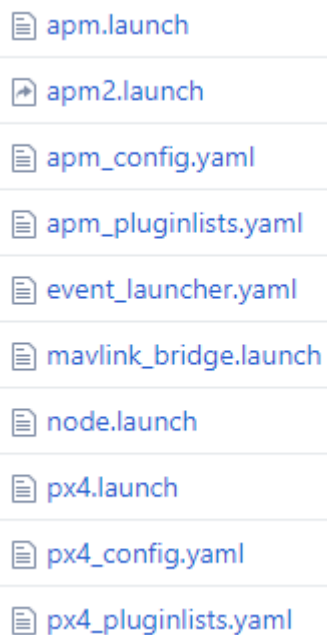


打开到src\mavros\mavros\src目录



- **lib**: 库文件（坐标转换等等）不需要修改
- **mavros**: 自带的一些python脚本，用不到
- **plugins**: 可以理解为是mavros包提供给我们的一些接口，如果需要用到某个plugin，建议可以看看plugin的源码，有的时候它默认的方向定义和一些细节可以自己修改一下，方便使用。有些plugin对于我们的任务是用不到的，可以将它添加到黑名单中禁用。
- 一般可以在这个目录下新建一个cpp文件，开发新功能（需要修改对应的cmakelist然后编译）

Mavros简介 - Mavros功能包结构



apm.launch
apm2.launch
apm_config.yaml
apm_pluginlists.yaml
event_launcher.yaml
mavlink_bridge.launch
node.launch
px4.launch
px4_config.yaml
px4_pluginlists.yaml

打开到src\mavros\mavros\launch目录

- **px4.launch:** 我们会用到的启动文件（修改：端口号和波特率）
- **px4_config.yaml:** px4.launch 调用的配置文件 (plugin的参数配置，基本不需要修改，对应使用某个plugin时，可以自行查看一下这里默认的参数定义)
- **px4_pluginlists.yaml:** px4.launch 调用的配置文件 (plugin的黑白名单，默认启用全部plugin，不想启用某个，可添加到黑名单中去)

Mavros代码详解 – plugin介绍

Plugin的作用

- 读取飞控发送过来的mavlink消息，解码，然后发布为话题，供我们订阅使用
- 订阅我们发布的话题，封装成mavlink消息，编码，然后发送给飞控

飞控mavlink模块的作用

- 读取机载电脑发送过来的mavlink消息，解码，然后发布为uorb消息，供飞控使用
- 订阅飞控中其他模块发布的uorb消息，封装成mavlink消息，编码，然后发送给机载电脑

我们利用plugin发布的ros话题去编写代码，实现功能，然后发布控制相关的ros话题，plugin接收我们的话题，替我们发送给飞控。

Mavros代码详解 – plugin介绍

这里简单介绍每个plugin的用处，用处不大的不提及，**重点的**稍后详细看代码

- **actuator_control**: 舵机控制相关
- **altitude**: 高度相关，用处不大
- **command**: 给飞控发送常规命令，例如解锁、起飞、降落、设置home点。
- **global_position**: 接收飞控的global位置信息
- **home_position**: home点相关
- **imu**: 接收imu信息（原始数据，融合过的数据，磁力计等）
- **local_position**: 接收飞控的local位置信息
- **manual_control**: 接收飞控的遥控器信息
- **safety_area**: 地理围栏设置

- 3dr_radio.cpp
- actuator_control.cpp
- altitude.cpp
- command.cpp
- dummy.cpp
- ftp.cpp
- global_position.cpp
- hil.cpp
- home_position.cpp
- imu.cpp
- local_position.cpp
- manual_control.cpp
- param.cpp
- rc_io.cpp
- safety_area.cpp
- setpoint_accel.cpp
- setpoint_attitude.cpp
- setpoint_position.cpp
- setpoint_raw.cpp
- setpoint_velocity.cpp
- sys_status.cpp
- sys_time.cpp
- vfr_hud.cpp
- waypoint.cpp

Mavros代码详解 – plugin介绍

这里简单介绍每个plugin的用处，用处不大的不提及，**重点的**稍后详细看代码

- **setpoint_position**、**setpoint_velocity**、**setpoint_accel**、**setpoint_attitude**: 发送给飞控期望位置、速度、加速度、姿态（这四个功能都可以用**setpoint_raw**实现，所以有点鸡肋）
- **setpoint_raw**: 发送期望位置、速度、加速度、姿态角给飞控；接收飞控回传的期望位置、速度、加速度、姿态角信息 [offboard模式]
- **sys_status**: 接收系统的状态，例如：连接状态、上/解锁、飞行模式等
- **sys_time**: 飞控时间
- **waypoint**: mission模式下的航点设置

- 3dr_radio.cpp
- actuator_control.cpp
- altitude.cpp
- command.cpp
- dummy.cpp
- ftp.cpp
- global_position.cpp
- hil.cpp
- home_position.cpp
- imu.cpp
- local_position.cpp
- manual_control.cpp
- param.cpp
- rc_io.cpp
- safety_area.cpp
- setpoint_accel.cpp
- setpoint_attitude.cpp
- setpoint_position.cpp
- setpoint_raw.cpp
- setpoint_velocity.cpp
- sys_status.cpp
- sys_time.cpp
- vfr_hud.cpp
- waypoint.cpp

Mavros代码详解 – 源码阅读



setpoint_raw.cpp

- setpoint_raw.cpp 源码阅读
- mavlink_receiver.cpp 源码阅读
- mavlink_message.cpp 源码阅读
- px4位置控制源码
- MAVLINK消息格式 查询 - 网站: <http://mavlink.org/messages/common>
- Ros消息格式查询 – 直接百度查询
- uORB消息格式查询 – 打开至固件的/msg文件夹

这块主要是帮大家梳理消息流向，整体把握代码之间关系，可以在以后开发过程中慢慢熟练掌握

Mavros代码详解 – setpoint_raw.cpp

Topic	Mavlink	uORB
/mavros/setpoint_raw/local	/SET_POSITION_TARGET_LOCAL_NED (#84)	position_setpoint_triplet
/mavros/setpoint_raw/attitude	SET_ATTITUDE_TARGET (#82)	vehicle_attitude_setpoint
/mavros/setpoint_raw/target_local	POSITION_TARGET_LOCAL_NED (#85)	vehicle_local_position_setpoint
/mavros/setpoint_raw/target_attitude	ATTITUDE_TARGET (#83)	vehicle_attitude_setpoint

Mavros代码详解 – 坐标系修改

坐标系修改 ENU系 修改至 NED系

- `imu.cpp`
- `local_position.cpp`
- `setpoint_raw.cpp`
- 和我所提供的代码坐标系匹配
- 方便调试 和 地面站匹配



如何连接飞控和机载电脑？

- **tty转usb模块（数传，安卓线）**
- **地面站参数修改： SYS_COMPANION 修改为 Companion Link(921600,8N1)**
- **修改mavros包中px4.launch中的端口号和波特率：端口号查询得到，波特率为921600**

自定义代码 – 入门操作

如何新建一个.cpp并编译?

- Mavros功能包内部, 依赖mavros现有的功能, 新建一个飞行任务或者一个功能模块
- 打开至/src/mavros/mavros/src, 拷贝mavros_example_1.cpp到此目录
- 打开至/src/mavros/mavros, 打开CMakeList.txt, 声明一下mavros_example_1.cpp
- 编译
- 详见: 如何去新建一个自定义cpp.pdf

如何自定义一个消息类型并编译?

- 现有的消息类型不符合使用需求, 自定义一个消息类型
- 打开至/src/mavros_msgs/msg, 拷贝Command.msg到此目录
- 打开至/src/mavros_msgs, 打开CMakeList.txt, 声明一下Command.msg
- 编译
- 详见: 如何去新建一个自定义msg.pdf

至于其他的一些操作, 新建launch文件, 新建配置文件, 新建服务等, 学会复制粘贴, 部分修改

自定义代码 – 入门操作

mavros_example_1.cpp

- 查询飞控状态（上/解锁，模式）
- 更改模式

mavros_example_2.cpp

- 上锁/解锁
- 订阅飞控的IMU信息（注意坐标系不同）
- 四元数转欧拉角

mavros_example_3.cpp

- 给飞控发送位置命令
- 切换offboard模式
- 读取飞控回传的位置设定值、姿态设定值

给大家十分钟去尝试一下在mavros中新建这三个cpp文件并编译。



type_mask:

- Bitmask to indicate which dimensions should be ignored by the vehicle (1代表忽略, 0代表不忽略; bit 10 特殊)
- Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7: ax, bit 8: ay, bit 9: az, bit 10: is force setpoint, bit 11: yaw, bit 12: yaw rate
- 0000 0000 0000 代表都不忽略
- 100 111 111 000 代表除了 位置xyz和yaw之外都忽略



自定义代码的几点说明:

- 学会修改飞控中mavlink_main.cpp (修改 飞控发送给机载电脑的mavlink消息类型和频率)
- 理解消息流向, 知道自己每条指令是从哪个cpp到了哪个cpp
- 出现问题时, 按照消息流向逐步排查问题。(特别是给飞控发送指令, 飞控没响应时)
- 一定是先发送一串命令给飞控才能切offboard模式 (安全起见, 手动切换)

自定义代码 – 自主飞行

自主飞行 – mavros包修改

1. mavros_msg包中添加Command.msg自定义消息
2. Mavros包include中添加pid.h文件和param.h
3. Mavros包中添加param_pid文件
4. mavros包中添加position_control.cpp 文件、move.cpp文件。
5. px4代码中，确认mavlink_main.cpp和地面站参数

position_control.cpp

- 位置环串级PID控制
- 提供与上层命令接口

move.cpp

- 位置控制测试代码

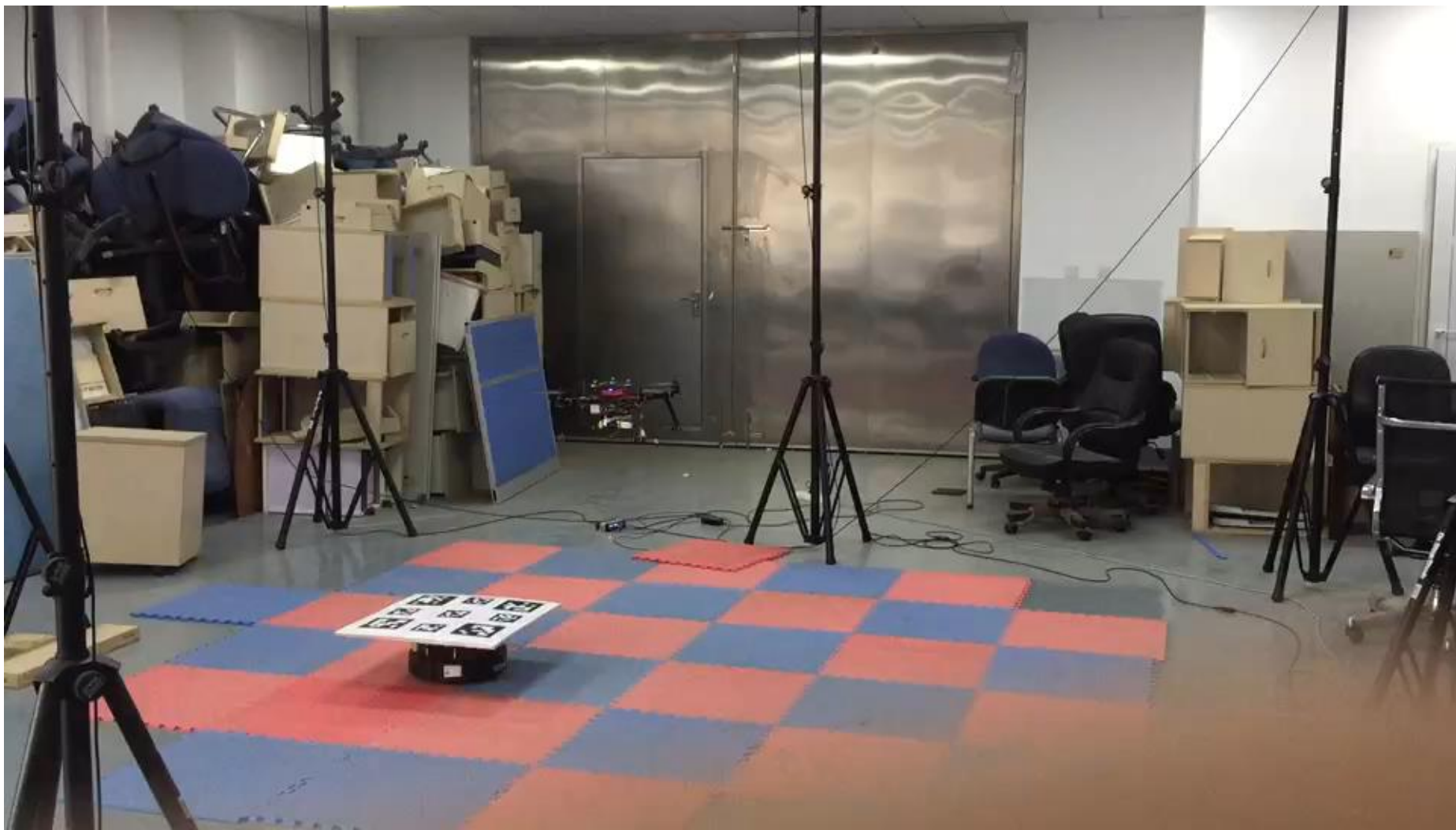


视觉追踪降落 track_land.cpp

- 机体系控制(一般图像提供的信息是基于机体系)
- 位置 or 速度控制
- 死区设置

上层开发就是利用position_control.cpp提供的Command接口去做上层应用

自定义代码 – 上层开发



提问环节