pixhawk上下位机通信实验

一、基础概念的明确

刚刚开始接触飞控开发的时候，有若干个概念需要明确。

1、PX4与APM

我们之前做实验室用的pixhawk 2.4.8是一款开源的飞行开发板，在这块硬件开发板上可以安装两套不同的飞控架构，分别是PX4原生固件，和APM固件，就如同是一台计算机既可以安装windows系统，也可以安装linux系统一样。

PX4原生固件的github网址为：<https://github.com/PX4>

APM固件的github网址为：<https://github.com/ArduPilot>

PX4原生固件项目启动得比较晚，特点是代码架构清晰，便于学习，修改和维护。但是可靠性以及功能方面都不如APM固件。APM固件的开发时间较早，特点可靠性好，功能完善，但是代码的架构比较凌乱，对于初学者来说难以快速掌握和维护。在本实验中使用的PX原生固件，在掌握了PX4原生固件的开发流程之后，再去学习APM固件的开发。

2、QGC与Massion Planer

QGC（QGroundControl）和Massion Planer是两款飞行控制地面站软件，不同的是，QGC采用的是QT架构，对应的是PX4原生固件，对QGC进行开发需要精通C++语言；Massion Planer采用的C#架构，对应的是APM固件，对Massion Planer进行开发需要精通C#语言。

3、 mavros与mavlink

MAVLink（Micro Air Vehicle Link，微型空中飞行器链路通讯协议）是无人飞行器与地面站（Ground Control Station ，GCS）之间通讯，以及无人飞行器之间通讯最常用的协议。关于mavlink的详细信息可以参考：<https://mavlink.io/en/>

mavros是mavlink协议在ROS机器人操作系统下的封装，能够非常方便地完成从ROS消息到mavlink消息的转化工作。有关于mavros的更多信息可以参考：<https://dev.px4.io/v1.9.0/en/ros/>

一、 mavros的配置

1、安装和编译

官网链接：<https://dev.px4.io/v1.9.0/en/robotics/>

相关博客的地址：<https://blog.csdn.net/wb790238030/article/details/82316653>

可以参考上述两个链接完成mavros的安装和编译。

2、进行offboard开发的相关配置

官网链接：<https://dev.px4.io/v1.10/en/companion_computer/pixhawk_companion.html>

此部分不建议参考其他的中文资料，因为中文的资料做不到实时更新，我们用的PX4固件是最新版本的，与之前版本的所需要配置的参数不同，盲目参考会把人搞晕。这里仔细去阅读官网的内容就可以了。

二、 px4原生代码的编译

官网链接：<https://dev.px4.io/v1.10/en/setup/getting_started.html>

这部分内容直接参考官网提供的链接即可，需要注意的是，在进行编译和上传固件的过程中，需要根据我们使用的飞控开发板的版本，修改相应的命令。

|  |
| --- |
| cd Firmware  make px4\_fmu-v4\_default  make px4\_fmu-v4\_default upload |

官网给出的是pixhawk4的编译命令，而我们使用的是pixhawk2.4.8飞控板，将上表中的px4\_fmu-v4\_default修改为px4\_fmu-v2\_default。然后才可以编译通过并上传成功。

三、 mavros上位机代码的编写

1、 在mavros中添加一个offb节点

在你的mavros所在文件下路径下的src目录中创建一个新的offb\_node.cpp文件，并粘贴下面内容：

|  |
| --- |
| /\*\*  \* @file offb\_node.cpp  \* @brief offboard example node, written with mavros version 0.14.2, px4 flight  \* stack and tested in Gazebo SITL  \*/    #include <ros/ros.h>  #include <geometry\_msgs/PoseStamped.h>  #include <mavros\_msgs/CommandBool.h>  #include <mavros\_msgs/SetMode.h>  #include <mavros\_msgs/State.h>    mavros\_msgs::State current\_state;  void state\_cb(const mavros\_msgs::State::ConstPtr& msg){  current\_state = \*msg;  }    int main(int argc, char \*\*argv)  {  ros::init(argc, argv, "offb\_node");  ros::NodeHandle nh;    ros::Subscriber state\_sub = nh.subscribe<mavros\_msgs::State>  ("mavros/state", 10, state\_cb);  ros::Publisher local\_pos\_pub = nh.advertise<geometry\_msgs::PoseStamped>  ("mavros/setpoint\_position/local", 10);  ros::ServiceClient arming\_client = nh.serviceClient<mavros\_msgs::CommandBool>  ("mavros/cmd/arming");  ros::ServiceClient set\_mode\_client = nh.serviceClient<mavros\_msgs::SetMode>  ("mavros/set\_mode");    //the setpoint publishing rate MUST be faster than 2Hz  ros::Rate rate(20.0);    // wait for FCU connection  while(ros::ok() && current\_state.connected){  ros::spinOnce();  rate.sleep();  }    geometry\_msgs::PoseStamped pose;  pose.pose.position.x = 0;  pose.pose.position.y = 0;  pose.pose.position.z = 2;    //send a few setpoints before starting  for(int i = 100; ros::ok() && i > 0; --i){  local\_pos\_pub.publish(pose);  ros::spinOnce();  rate.sleep();  }    mavros\_msgs::SetMode offb\_set\_mode;  offb\_set\_mode.request.custom\_mode = "OFFBOARD";    mavros\_msgs::CommandBool arm\_cmd;  arm\_cmd.request.value = true;    ros::Time last\_request = ros::Time::now();    while(ros::ok()){  if( current\_state.mode != "OFFBOARD" &&  (ros::Time::now() - last\_request > ros::Duration(5.0))){  if( set\_mode\_client.call(offb\_set\_mode) &&  offb\_set\_mode.response.mode\_sent){  ROS\_INFO("Offboard enabled");  }  last\_request = ros::Time::now();  } else {  if( !current\_state.armed &&  (ros::Time::now() - last\_request > ros::Duration(5.0))){  if( arming\_client.call(arm\_cmd) &&  arm\_cmd.response.success){  ROS\_INFO("Vehicle armed");  }  last\_request = ros::Time::now();  }  }    local\_pos\_pub.publish(pose);    ros::spinOnce();  rate.sleep();  }    return 0;  } |

该程序的主要功能是：使无人机缓慢飞到2米的高度。

这是一段复制上去就可以直接用的程序，但想要理解深刻，需要对ROS有一定的了解，在了解后，才能看懂下面的内容。下面将对代码的每一部分进行分析，使其能够实现自己想要实现的功能。

需要注意的是：对于用于发送接收的mavros的结构体变量的使用，最好还是查看mavros包中的变量定义，这有助于灵活的对代码进行调整。

2、编辑Cmakelists.txt文件

Cmakelists.txt文件是工程的组织文件，对于工程编译至关重要。想要学好基于Ubuntu的嵌入式开发，Cmakelists相关命令的学习是非常关键的。

详细的内容可以参考其官方网站：<https://cmake.org/>

在本实验中，需要在cmakelists.txt代码中添加一下的两行指令：

|  |
| --- |
| add\_executable(offb\_node src/offb\_node.cpp)  target\_link\_libraries(offb\_node ${catkin\_LIBRARIES}) |

3、代码的编译

在控制台中，回到你的mavros工程所在的工程路径，然后输入以下命令：

|  |
| --- |
| catkin\_make |

在此需要注意的是，mavros的编译可以使用两种不同的命令，catkin build和catkin\_make，两种命令不可以混用，如果你在建立工程的时候，采用的是catkin build命令，那么在此仍需要使用catkin build命令进行编译，否则将会报错。

四、 px4原生代码中应用程序的编写

在Firmware/src/moudle下，创建一个文件夹，例如名为myFirstMoudle，给他起一个你想要的名字，创建一个新的C语言文件，假定他为myFirstMoudle.cpp 。

这里需要说明的是文件名和C语言文件的名字要相同，否则会出现问题。

* 添加main函数声明

|  |
| --- |
| \_\_EXPORT int px4\_myFirstMoudle\_main(int argc, char \*argv[]); |

* 定义一些要用的常量

|  |
| --- |
| static bool thread\_should\_exit = false; /\*\*< 线程是否退出 \*/  static bool thread\_running = false; /\*\*< 线程是否在运行 \*/  static int myFirstMoudle\_task; /\*\*< 线程句柄 \*/ |

* 声明线程函数

|  |
| --- |
| int px4\_myFirstMoudle\_thread\_main(int argc, char \*argv[]); |

* 声明命令的用法打印函数  
  注意此函数申明为了static，那么只在本文件中有效

|  |
| --- |
| static void  usage(const char \*reason)  {  if (reason) {  warnx("%s\n", reason);  }  warnx("usage: myFirstMoudle {start|stop|status} [-p <additional params>]\n\n");  } |

* 主函数

|  |
| --- |
| int px4\_myFirstMoudle\_main(int argc, char \*argv[])  {  if (argc < 2) {  usage("missing command");  return 1;  } //检测命令有效性    //检测到运行命令  if (!strcmp(argv[1], "start")) {    if (thread\_running) {  warnx("myFirstMoudle already running\n");  /\* this is not an error \*/  return 0;  } //判断线程已经在运行  //如果没在运行，准备开始运行线程，更改常量，使线程不可以退出  thread\_should\_exit = false;  //在nuttx系统中创建一个线程  myFirstMoudle\_task = px4\_task\_spawn\_cmd("myFirstMoudle",  SCHED\_DEFAULT, //调度方式，一般就是这个值  SCHED\_PRIORITY\_DEFAULT, //这是优先级，忘了define了  2000, //这是栈的大小  px4\_myFirstMoudle\_thread\_main, //在此处填入线程函数  (argv) ? (char \*const \*)&argv[2] : (char \*const \*)NULL);  return 0;  }  //如果是退出命令  if (!strcmp(argv[1], "stop")) {  thread\_should\_exit = true;  return 0;  }  //如果是查询状态命令  if (!strcmp(argv[1], "status")) {  if (thread\_running) {  warnx("\trunning\n");    } else {  warnx("\tnot started\n");  }    return 0;  }    usage("unrecognized command");  return 1;  } |

* 线程函数的编写

|  |
| --- |
| int px4\_myFirstMoudle\_thread\_main(int argc, char \*argv[])  {    warnx("[myFirstMoudle] starting\n");  /\*\*< 此处可以填入一些启动之初要执行的代码  thread\_running = true;    while (!thread\_should\_exit) {  //此处写循环执行的代码  warnx("Hello!\n");  sleep(5); //注意，这种后台程序必须要延迟，不然比他低优先级的任务永远不会执行  }    warnx("[myFirstMoudle] exiting.\n");  //此处进行任务结束的一些处理  thread\_running = false;    return 0;  } |

* 之后我们把程序加入到系统里  
  修改下面的文件

|  |
| --- |
| [Firmware/boards/px4/fmu-v2/default.cmake](https://github.com/PX4/Firmware/blob/master/boards/px4/fmu-v2/default.cmake) |

* 在后面加上  
  moudles/myFirstMoudle
* 之后就是编译上传到自己的飞控板子上

五、 上下位机的联合调试

1、硬件连接

首先参考链接：<https://dev.px4.io/v1.10/en/companion_computer/pixhawk_companion.html>

使用ch340串口转usb模块，将PC机与pixhawk连接到一起。

同时连接pixhawk的usb数据线。

2、运行启动文件

|  |
| --- |
| roslaunch mavros px4.launch |

3、启动按钮解锁

长按启动爱开关进行解锁操作。



4、运行节点

|  |
| --- |
| rosrun px4\_mavros offb\_node |

这里需要注意的是 px4\_mavros需要根据自己的文件夹名称调整，offb\_node