1. Mavros的下载与安装

根据网址 https://github.com/SIA-UAVGP/mavros 上的说明
（1）需要首先安装 catkin tool 如下

```
sudo apt-get install python-catkin-tools python-rosinstall-generator -y
```

（2）建立mavros的workspace并下载mavros与mavlink包

```
mkdir mavros_ws/src -p
cd mavros_ws

catkin init

cd src

git clone https://github.com/LiuZhongSIA/UAVGP2016_mavros  mavros （注意分支，使用uavcomp分支）

git clone https://github.com/LiuZhongSIA/UAVGP2016_mavlink mavlink （注意分支，使用master分支）
```

（3）编译也只编译mavros

```
catkin build mavros
```

（4）Qt Creator
建议下载最新的qt，软件的运行体验比较好，不容易出现卡顿。不过其他ros工程不好打开，所以一直使用的qt是4.8.7版本。
使用qt “Open Project”，然后在“Configure Project”中选择“Desktop”，将里面的路径都设置到“mavros_ws/build/mavros”路径下，“Default”设置在此路径，其余的可以是此路径下的文件夹

2. Mavros中添加自定义消息（以attitude消息为例）

（1）添加attitude.msg文件在文件夹mavros_ws/src/mavros/mavros_msgs/msg中

```
std_msgs/Header header

float32 roll
float32 pitch
float32 yaw
float32 rollspeed
float32 pitchspeed
float32 yawspeed
```

在mavros_ws/src/mavros/mavros_msgs文件夹的CMakeLists.txt文件中添加上述msg文件

```
add_message_files(
  ...
  Altitude.msg
  Attitude.msg  #自定义msg文件
  AttitudeTarget.msg
  ...
)
```

（2）每一个自定义消息对应一个“plugin”文件，相当于进行mavlink消息与ros内部消息之间的转化，在此之前，于mavros_ws/src/mavros/mavros文件夹的mavros_plugins.xml文件中添加对于此plugin的声明。根据描述，这个消息是用以发布的，也就是由ros节点订阅飞控消息，该消息的plugin需订阅mavlink消息，发布ros消息；当然消息也可以订阅，意思为ros发布本消息，plugin需订阅ros消息，发布mavlink消息

```
...
<class name="attitude" type="mavros::std_plugins::AttitudePlugin"
base_class_type="mavros::plugin::PluginBase">
    <description>Publish attitude values, as a ROS node</description>
</class>
...
```

在文件夹mavros_ws/src/mavros/mavros/src/plugins中添加该消息对应的plugin文件attitude.cpp。这实际上是定义一个class，之后会进行调用。

```
#include <mavros/mavros_plugin.h>
#include <mavros_msgs/Attitude.h>

namespace mavros {
namespace std_plugins {
```

```cpp
/**
 * @brief Attitude plugin.
 */
class AttitudePlugin : public plugin::PluginBase {
public:
    AttitudePlugin() : PluginBase(),
        nh("~")
    { }
    /**
     * Plugin initializer. Constructor should not do this.
     * Plugin初始化，这一部分内容开发者不要修改
     */
    void initialize(UAS &uas_)
    {
        PluginBase::initialize(uas_);
        nh.param<std::string>("frame_id", frame_id, "map");
        attitude_pub = nh.advertise<mavros_msgs::Attitude>("attitude", 10);
        // 发布是个叫Attitude的消息，所是用的topic名字为“attitude”，10为缓存数量
    }
    Subscriptions get_subscriptions()
    {
        return {
            make_handler(&AttitudePlugin::handle_attitude),
            // 调用处理函数
        };
    }
private:
    ros::NodeHandle nh;
    std::string frame_id;
    ros::Publisher attitude_pub;
    void handle_attitude(const mavlink::mavlink_message_t *msg, mavlink::common::msg::ATTITUDE &attitude)
    // mavlink消息转化为ros消息
    {
        auto ros_msg = boost::make_shared<mavros_msgs::Attitude>(); // 定义一个ros消息要发送的变量
        ros_msg->header = m_uas->synchronized_header(frame_id, attitude.time_boot_ms);
        ros_msg->roll        = attitude.roll; //赋值
        ros_msg->pitch       = attitude.pitch;
        ros_msg->yaw         = attitude.yaw;
        ros_msg->rollspeed   = attitude.rollspeed;
```

```
        ros_msg->pitchspeed  = attitude.pitchspeed;
        ros_msg->yawspeed    = attitude.yawspeed;
        // 发布ros消息
        attitude_pub.publish(ros_msg);
    }
};
}   // namespace std_plugins
}   // namespace mavros
#include <pluginlib/class_list_macros.h>
PLUGINLIB_EXPORT_CLASS(mavros::std_plugins::AttitudePlugin,
mavros::plugin::PluginBase)
```

最后，需要修改mavros_ws/src/mavros/mavros文件夹下的CMakeLists.txt文件，添加上面建立的cpp文件

```
add_library(mavros_plugins
  ...
  src/plugins/attitude.cpp
  ...
)
```

当然，如果mavlink消息需要publish时，所要建立的plugin文件就会不同，以vision_one_num_get消息为例，vision_one_num_get.cpp文件如下

```

#include <mavros/mavros_plugin.h>
#include <mavros_msgs/VisionOneNumGet.h>

namespace mavros {
namespace std_plugins {
/**
 * @brief VisionOneNumGet plugin.
 */
class VisionOneNumGetPlugin : public plugin::PluginBase {
public:
    VisionOneNumGetPlugin() : PluginBase(),
        nh("~")
    { }
    /**
     * Plugin initializer. Constructor should not do this.
     */
    void initialize(UAS &uas_)
    {
```

```
        PluginBase::initialize(uas_);
        nh.param<std::string>("frame_id", frame_id, "map");
        mavros_msg_sub = nh.subscribe("vision_one_num_get", 10, &VisionOneNumGetPlugin::vision_one_num_get_cb, this);
        // 这里需要订阅一个ros消息，topic名字为“vision_one_num_get”，回调函数为后面的“vision_one_num_get_cb”
    }
    Subscriptions get_subscriptions()
    {
        return {
            // 因为不需要消息订阅，所以这里不用处理
        };
    }
private:
    ros::NodeHandle nh;
    std::string frame_id;
    ros::Subscriber mavros_msg_sub;
    void vision_one_num_get_cb(const mavros_msgs::VisionOneNumGet::ConstPtr &req)
    // 回调函数，订阅到的ros消息放在req中
    {
        mavlink::pixhawk::msg::VISION_ONE_NUM_GET test_msg{}; //定义一个mavlink消息
        test_msg.timestamp = ros::Time::now().toNSec() / 1000; //mavlink消息赋值
        test_msg.loop_value     = req->loop_value;
        test_msg.num             = req->num;
        UAS_FCU(m_uas)->send_message_ignore_drop(test_msg); //发布mavlink消息
    }
};
}   // namespace std_plugins
}   // namespace mavros
#include <pluginlib/class_list_macros.h>
PLUGINLIB_EXPORT_CLASS(mavros::std_plugins::VisionOneNumGetPlugin, mavros::plugin::PluginBase)
```

3. 配置lauch文件

lauch文件用于配置运行的内容，需要开启px4.launch文件，该文件内容如下，配置了与pix通讯的相关设置，比如USB口，波特率等。该lauch文件包含了node.launch文件，在这个文件里将开启mavros

```
<launch>
```

```
    <!-- vim: set ft=xml noet : -->
    <!-- example launch script for PX4 based FCU's -->

    <arg name="fcu_url" default="/dev/ttyUSB0:921600" />
    <arg name="gcs_url" default="" />
    <arg name="tgt_system" default="1" />
    <arg name="tgt_component" default="1" />
    <arg name="log_output" default="screen" />

    <include file="$(find mavros)/launch/node.launch">
        <arg name="pluginlists_yaml" value="$(find mavros)/launch/px4_pluginlists.yaml" />
        <arg name="config_yaml" value="$(find mavros)/launch/px4_config.yaml" />

        <arg name="fcu_url" value="$(arg fcu_url)" />
        <arg name="gcs_url" value="$(arg gcs_url)" />
        <arg name="tgt_system" value="$(arg tgt_system)" />
        <arg name="tgt_component" value="$(arg tgt_component)" />
        <arg name="log_output" value="$(arg log_output)" />
    </include>
</launch>
```

运行方式如下

```
roslaunch mavros px4.launch
```

连接Pix后要把TELEM2的mavlink通讯打开，并配置波特率
运行“rosrun rqt_topic rqt_topic”可看到当前所有的topic，方便后面的状态机编写

4. 发送任务指令

上面的mavros仅仅起到一个媒介的作用，仍然需要运行ros节点发送指令消息，充当一个任务状态机的角色。如下为一个简单任务的状态机mission_node.cpp

```
#include <ros/ros.h>
#include <math.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/TwistStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
#include <mavros_msgs/Attitude.h>
```

```cpp
#include <mavros_msgs/CommandTOL.h>
#include <mavros/frame_tf.h>
void state_machine(void); //声明状态机函数

// 定义变量，代表状态机状态
static const int POS_A = 0;
static const int POS_B = 1;
static const int LAND  = 2;
// 当前的位置状态，在点A上
int current_pos_state = POS_A;

// 设置A、B点，msg，将通过ros发送出去
geometry_msgs::PoseStamped pose_a;
geometry_msgs::PoseStamped pose_b;
ros::Publisher local_pos_pub;

// 订阅“状态”消息，以及相应回调函数
mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}
// 订阅“位置消息”，以及相应的回调函数
geometry_msgs::PoseStamped current_pos;
void pos_cb(const geometry_msgs::PoseStamped::ConstPtr& msg){
    current_pos = *msg;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "mission_node"); //节点名称
    ros::NodeHandle nh; //ros节点句柄

    // 配置回调函数，得到飞机的状态信息
    ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>
("mavros/state", 10, state_cb);
    // 配置回调函数，得到飞机的位置信息
    ros::Subscriber pos_sub = nh.subscribe<geometry_msgs::PoseStamped>
("mavros/local_position/pose", 10, pos_cb);
    // 要向飞机发布位置指令
    local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
("mavros/setpoint_position/local", 10);
    // 配置ros service，用于飞机解锁
    ros::ServiceClient arming_client =
nh.serviceClient<mavros_msgs::CommandBool>("mavros/cmd/arming");
```

```
    // 配置ros service, 用于改变模式
    ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>("mavros/set_mode");
    // 配置ros service, 用于降落
    ros::ServiceClient land_client = nh.serviceClient<mavros_msgs::CommandTOL>("mavros/cmd/land");

    // 设置消息发送频率
    ros::Rate rate(20.0);
    // 等待飞行控制器链接
    while(ros::ok() && !current_state.connected){
        // 如果控制器没连接上, 将一直处于循环里
        ros::spinOnce();
        rate.sleep();
    }

    // A点位置
    pose_a.pose.position.x = 0;
    pose_a.pose.position.y = 0;
    pose_a.pose.position.z = 5;
    // B点位置
    pose_b.pose.position.x = 0;
    pose_b.pose.position.y = 5;
    pose_b.pose.position.z = 5;
    // 姿态期望
    auto quat_yaw = mavros::ftf::quaternion_from_rpy(0.0, 0.0, 0.0);
    pose_a.pose.orientation.x = quat_yaw.x();
    pose_a.pose.orientation.y = quat_yaw.y();
    pose_a.pose.orientation.z = quat_yaw.z();
    pose_a.pose.orientation.w = quat_yaw.w();
    pose_b.pose.orientation.x = quat_yaw.x();
    pose_b.pose.orientation.y = quat_yaw.y();
    pose_b.pose.orientation.z = quat_yaw.z();
    pose_b.pose.orientation.w = quat_yaw.w();
    // 开始之前先要进行期望位置的发送！！！
    for(int i = 100; ros::ok() && i > 0; --i){
        local_pos_pub.publish(pose_a);
        ros::spinOnce();
        rate.sleep();
    }

    // 定义模式msg, 写入offboard模式
    mavros_msgs::SetMode offb_set_mode;
    offb_set_mode.request.custom_mode = "OFFBOARD";
```

```
    // 定义解锁msg, 写入解锁
    mavros_msgs::CommandBool arm_cmd;
    arm_cmd.request.value = true;
    // 定义降落msg
    mavros_msgs::CommandTOL landing_cmd;
    landing_cmd.request.min_pitch = 1.0;
    // 时间戳
    ros::Time last_request = ros::Time::now();
    ros::Time landing_last_request = ros::Time::now();

    while(ros::ok()){
        // 下面这几行是针对仿真的
        // 因为仿真中没法解锁并切换模式
        if(current_state.mode != "OFFBOARD" && //如果当前还不是offboard模式
           (ros::Time::now() - last_request > ros::Duration(5.0)))
         {
            // 设置为offboard模式
            if(set_mode_client.call(offb_set_mode) &&
                offb_set_mode.response.success)
             {
                 ROS_INFO("Offboard enabled");
             }
             last_request = ros::Time::now();
         } else
         {
             if(!current_state.armed &&      // 如果还没有解锁, 进进行解锁
               (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();
             }
         }

        // 如果是飞真实飞机, 可以只运行下面这部分
        // 没解锁的情况下, 模式状态在A点
        if(!current_state.armed)
        {
            current_pos_state = POS_A;
        }
```

```
        // 自动起飞显示，仅显示
        if( current_state.mode == "AUTO.TAKEOFF"){
            ROS_INFO("AUTO TAKEOFF!");
        }
        // 如果状态机要求降落，且当前不是自动降落，且上一起降落要求在
        // 发送降落指令
        if((current_pos_state == LAND) && (current_state.mode == "OFFBOARD")){
            if( current_state.mode != "AUTO.LAND" &&
                (ros::Time::now() - landing_last_request > ros::Duration(5.0))){
                if(land_client.call(landing_cmd) &&
                    landing_cmd.response.success){
                    ROS_INFO("AUTO LANDING!");
                }
                landing_last_request = ros::Time::now();
            }
        }
        state_machine();
        ros::spinOnce();
        rate.sleep();
    }
    return 0;
}

// 任务状态机
void state_machine(void){
    switch(current_pos_state){
        case POS_A:
            local_pos_pub.publish(pose_a); //发布A点位置信息
            if((abs(current_pos.pose.position.x - pose_a.pose.position.x) < 0.1) && //到达A点后切换到下一点
               (abs(current_pos.pose.position.y - pose_a.pose.position.y) < 0.1) &&
               (abs(current_pos.pose.position.z - pose_a.pose.position.z) < 0.1))
               {
                    current_pos_state = POS_B;
               }
            break;
        case POS_B:
            local_pos_pub.publish(pose_b); //发布B点位置信息
            if((abs(current_pos.pose.position.x - pose_b.pose.position.x) < 0.1) && //到达B点后准备降落
```

```
                (abs(current_pos.pose.position.y - pose_b.pose.position.y) < 0.1) &&
                (abs(current_pos.pose.position.z - pose_b.pose.position.z) < 0.1))
                {
                    current_pos_state = LAND;
                }
               break;
        case LAND:
            break;
    }
}
```

在添加上述节点后，需要配置一下该package下的CMakeLists.txt，在mavros_ws/src/mavros/mavros文件夹下

```
...
add_executable(mission_node
  src/mission_node.cpp
)
target_link_libraries(mission_node
  mavros
  ${catkin_LIBRARIES}
  ${Boost_LIBRARIES}
)
...
install(TARGETS mavros mavros_plugins mavros_node gcs_bridge offb_node mission_node
  ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
  LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
  RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
...
```

5. Gazebo仿真

```
# Terminal 1
roscore

# Terminal 2
# 开启mavros，不过在电脑上仿真，因而基于UDP通讯
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"

```

```
# Terminal 3
# 开启pix基于gazebo的仿真环境
cd pix的fimware文件夹路径
make posix_sitl_default gazebo
# 第一次开启时，需要等待一段时间
# 出现场景后，继续...

# Terminal 4
# 运行状态机节点
rosrun mavros mission_node
# 下面的语句相当于切换飞行模式
rosrun mavros mavsys mode -c OFFBOARD（如果代码里有相同功能，可省略）
# 下面的语句相当于解锁
rosrun mavros mavsafety arm （如果代码里有相同功能，可省略）
```

在系统Home目录下，有一个隐藏文件夹“.ros/log”，里面有ros运行节点的信息日记rosout.log，可以查看并进行调试。
建议每次用ctrl+c结束roscore，这样可以看到每次生成文件夹的名字。

6. Mavros说明

ROS网站提供了mavros的相关说明 http://wiki.ros.org/mavros
网站中比较有用的是给出了各个msg的Plugin说明，例如setpoint_velocity——

这个msg的作用是发送速度期望点到FCU（飞行控制器），所使用的topic
为“mavros/setpoint_velocity/cmd_vel”，对应变量定义在geometry_msgs::TwistStamped。

7. 问题说明

尝试下载最新的mavlink与mavros包进行修改，使用最新的mavlink包，调用的头文件不
在“mavros_ws/devel/include/mavlink”中，而是在ros的安装文件夹下，不知道是不是自动生成的，但是新加入的字段并没有生成头文件。而且，当把这些文件夹删掉后，编译就无法进行了。。。
使用新的mavros包和老的mavlink包，可以进行所有操作，所以问题还是出在了mavlink里