2.2 姿态指令发布的程序讲义

```
/**
 * author:kaidi wang
 * mail    :1055080765@qq.com
 * date    :2020.12.8
 * describe: px4 in offboard mode and publish attitude control
**/
#include <ros/ros.h>
#include <std_msgs/Float32.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
#include <mavros_msgs/AttitudeTarget.h>
#include <geometry_msgs/Quaternion.h>
#include <geometry_msgs/Point.h>
#include <stdio.h>
#include <math.h>
#include <tf/tf.h>

#define PI 3.14159
//define thrust MAX
float THRUST_FULL=60;




//declear function list
geometry_msgs::Point ned2enu_angle(geometry_msgs::Point ned_angle);
float thrust_scale(float thrust);
geometry_msgs::Quaternion euler_to_quat(geometry_msgs::Point euler_angle_enu);

//ned to enu
geometry_msgs::Point ned2enu_angle(geometry_msgs::Point ned_angle)
{
    geometry_msgs::Point enu_angle;
    enu_angle.x = ned_angle.x;
    enu_angle.y = -ned_angle.y;
    enu_angle.z = -ned_angle.z + PI/2;
    return enu_angle;
}


//callback function list
mavros_msgs::State current_state;
```

```
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}

//eluer to quad
geometry_msgs::Quaternion euler_to_quat(geometry_msgs::Point euler_angle_enu)
{
    tfScalar yaw,pitch,roll;
    tf::Quaternion q;
    geometry_msgs::Quaternion quat;

    roll = euler_angle_enu.x;
    pitch = euler_angle_enu.y;
    yaw = euler_angle_enu.z;

    q.setEulerZYX(yaw,pitch,roll);
    quat.x = q.x();
    quat.y = q.y();
    quat.z = q.z();
    quat.w = q.w();
    return quat;
}

//calc thrust_scale
float thrust_scale(float thrust)
{
    float thrust_scale = 0.0;
    if (thrust>=THRUST_FULL)
    {
        /* code */
        thrust= THRUST_FULL;
    }
     thrust_scale = thrust/ THRUST_FULL;
}

char tmp[1024];
int main (int argc,char** argv)
{
    ros::init(argc,argv,"pub_setpoints");
    ros::NodeHandle nh;

    ros::Subscriber state_sub =
nh.subscribe<mavros_msgs::State>("mavros/state",10,state_cb);
```

```
ros::Publisher raw_pub_att = nh.advertise<mavros_msgs::AttitudeTarget>
("mavros/setpoint_raw/attitude",100);

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");

mavros_msgs::AttitudeTarget raw_att;
ros::Rate rate(100.0);
while (ros::ok() && !current_state.connected)
{
      /* wait until px4 connected */
      ROS_INFO_STREAM("wait until px4 connected ");
      ros::spinOnce();
      rate.sleep();
}



//ros::Rate loop_rate(100);

//set offboard mode and then arm the vechicl
mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";

mavros_msgs::CommandBool arm_cmd;
arm_cmd.request.value=true;

geometry_msgs::Point enu_euler;
geometry_msgs::Point ned_euler;

geometry_msgs::Quaternion q;
float thr;
ros::Time last_request = ros::Time::now();
while (ros::ok())
{
      /* set offboard and then arm the vechile */
      if (!current_state.armed && current_state.mode != "OFFBOARD")
      {
            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(3.0)))
                    {
                        ROS_INFO("arm enabled or not ");
                        if( arming_client.call(arm_cmd)&&arm_cmd.response.success)
                        {
                            ROS_INFO("Vehicle armed");
                        }
                        last_request = ros::Time::now();
                            // raw_pub_att.publish(raw_att);

                    }
                    else
                    {
                        //ROS_INFO_STREAM("don't enter the if armed....");
                    }
                }
            }

            ned_euler.x = 0;
            ned_euler.y = 0;
            ned_euler.z = PI/6;
            enu_euler = ned2enu_angle(ned_euler);

            q = euler_to_quat(enu_euler);
            thr = thrust_scale(30);

            raw_att.header.stamp = ros::Time::now();

            raw_att.thrust=thr;
            raw_att.orientation.x=q.x;
            raw_att.orientation.y=q.y;
            raw_att.orientation.z=q.z;
            raw_att.orientation.w=q.w;
```

```
            raw_pub_att.publish(raw_att);

            ros::spinOnce();
            loop_rate.sleep();
        }

        return 0;
}
```