2.1 讲义：对正方形轨迹代码进行改写

1、头文件代码

```
#ifndef    OFFBOARD_CLASS_TIMER_H_
#define    OFFBOARD_CLASS_TIMER_H_

#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
#include <geometry_msgs/PoseStamped.h>
#include <ros/ros.h>

class offboard_class_timer
{
private:
        ros::Publisher local_pos_pub;
        ros::Subscriber local_pos_sub ;
        ros::Subscriber state_sub ;
        ros::ServiceClient arming_client;
        ros::ServiceClient set_mode_client;



        ros::NodeHandle nh;//we will need this, to pass between "main" and constructor
        void init_publisher();
        void init_subscriber();
        void init_service();

        void local_pos_cb(const geometry_msgs::PoseStamped::ConstPtr& msg);
        void state_cb(const mavros_msgs::State::ConstPtr& msg);
      // void output_publish(geometry_msgs::Point ang1,geometry_msgs::Point
ang2,geometry_msgs::Point ang3,std_msgs::Float64 thu1,std_msgs::Float64
thu2,std_msgs::Float64 thu3);

public:
        offboard_class_timer(ros::NodeHandle* nodehandle);
        ~offboard_class_timer();

        ros::Timer calc_timer;
        void calc_cb(const ros::TimerEvent&);

        geometry_msgs::PoseStamped pose;
        mavros_msgs::State current_state;
        geometry_msgs::PoseStamped local_pos;
        mavros_msgs::SetMode offb_set_mode;
        mavros_msgs::CommandBool arm_cmd;
```

```
    ros::Time last_request;
    int step = 0;
    int sametimes = 0;
};


offboard_class_timer::~offboard_class_timer()
{
}

#endif
```

2、.cpp 文件

```
#include <offboard_class_timer/offboard_class_timer.h>

offboard_class_timer::offboard_class_timer(ros::NodeHandle*
nodehandle):nh(*nodehandle)
{
    calc_timer = nh.createTimer(ros::Duration(0.05), &offboard_class_timer::calc_cb, this);
//timer used to publish state, should be at least for some minimal frequency

}

void offboard_class_timer::init_publisher()
{
    local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
    ("mavros/setpoint_position/local", 10,this);
}


void offboard_class_timer::init_subscriber()
{
    local_pos_sub = nh.subscribe<geometry_msgs::PoseStamped>
        ("mavros/local_position/pose", 10, &offboard_class_timer::local_pos_cb,this);
    state_sub = nh.subscribe<mavros_msgs::State>
        ("mavros/state", 10, &offboard_class_timer::state_cb,this);
}

void offboard_class_timer::init_service()
{
    arming_client = nh.serviceClient<mavros_msgs::CommandBool>
        ("mavros/cmd/arming",this);
    set_mode_client = nh.serviceClient<mavros_msgs::SetMode>
        ("mavros/set_mode",this);
```

```
}

void offboard_class_timer::local_pos_cb(const geometry_msgs::PoseStamped::ConstPtr&
msg)
{
	local_pos = *msg;

}

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

void offboard_class_timer::calc_cb(const ros::TimerEvent&)
{
	 if(current_state.connected)
	 {
		offb_set_mode.request.custom_mode = "OFFBOARD";
		arm_cmd.request.value = true;
		last_request = ros::Time::now();
		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();
			}
			else
			{
				switch (step)
				{
				case 0:
					//take off to 2m
```

```
                    pose.pose.position.x = 0;
                    pose.pose.position.y = 0;
                    pose.pose.position.z = 2;
                    //
                    if (local_pos.pose.position.z > 1.9 && local_pos.pose.position.z <
2.1)
                    {
                        if (sametimes > 100)
                        {

                            sametimes = 0;
                            step = 1;
                            pose.pose.position.x = 2;
                            pose.pose.position.y = 0;
                            pose.pose.position.z = 2;
                        }
                        else
                            sametimes++;
                    }
                    else
                    {
                        sametimes = 0;
                    }
                    local_pos_pub.publish(pose);
                    break;
                case 1:

                    if (local_pos.pose.position.x > 1.9 && local_pos.pose.position.x <
2.1)
                    {
                        if (sametimes > 100)
                        {

                            step = 2;
                            pose.pose.position.x = 2;
                            pose.pose.position.y = 2;
                            pose.pose.position.z = 2;
                        }
                        else
                            sametimes++;
                    }
                    else
                    {
                        sametimes = 0;
```

```
                }
                local_pos_pub.publish(pose);
                break;
            case 2:

                if (local_pos.pose.position.y > 1.9 && local_pos.pose.position.y <
2.1)
                {
                    if (sametimes > 100)
                    {

                        step = 3;
                        pose.pose.position.x = 0;
                        pose.pose.position.y = 2;
                        pose.pose.position.z = 2;
                    }
                    else
                        sametimes++;
                }
                else
                {
                    sametimes = 0;
                }
                local_pos_pub.publish(pose);
                break;
            case 3:

                if (local_pos.pose.position.x > -0.1 && local_pos.pose.position.x <
0.1)
                {
                    if (sametimes > 100)
                    {

                        step = 4;
                        pose.pose.position.x = 0;
                        pose.pose.position.y = 0;
                        pose.pose.position.z = 2;
                    }
                    else
                        sametimes++;
                }
                else
                {
                    sametimes = 0;
```

```
                        }
                        local_pos_pub.publish(pose);
                        break;
                    case 4:

                        if (local_pos.pose.position.y > -0.1 && local_pos.pose.position.y <
0.1)
                        {
                            if (sametimes > 100)
                            {
                                step = 5;
                            }
                            else
                                sametimes++;
                        }
                        else
                        {
                            sametimes = 0;
                        }
                        local_pos_pub.publish(pose);
                        break;
                    case 5:
                        offb_set_mode.request.custom_mode = "AUTO.LAND";
                        if (current_state.mode != "AUTO.LAND" && (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("AUTO.LAND enabled");
                            }
                            last_request = ros::Time::now();
                        }
                        break;
                    default:
                        break;
                    }
                }
            }
        }
        else{
            ROS_INFO_STREAM("unconnected");
        }
```

```
}

int main(int argc, char **argv)
{
        ros::init(argc, argv, "offb_node");
        ros::NodeHandle nh;

        //Constructor
        offboard_class_timer offboard_class_timer_node(&nh);//init some param and then
start the controller
        ros::spin();
        return 0;
}
```