上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

多机器人系统与控制
课程大作业

小组编号：第二组

组员姓名：陈炜昊　王喆隆
易子淇　章雨悠

指导教师：熊振华　董伟　吴建华

2021 年 12 月

多机器人系统与控制大作业

目录

1 Question 1

编写可视化仿真环境，模拟上图所示主动视觉连接下的多机器人系统。同时假定系统还配置有 10Hz 通讯系统，可供机器人间进行状态交换。以下所有仿真均在此仿真环境中依照上述假定进行验证。

1.1 Modeling

在 Gazebo 中建立仿真模型如下，图中红色表示相机视角范围，白色表示相机中心线。

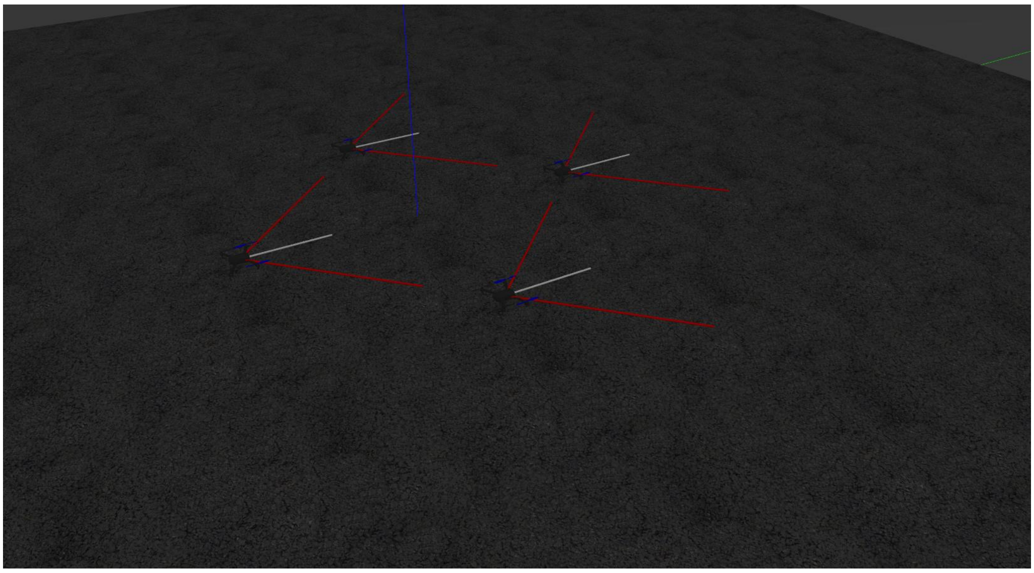

图 1: 仿真模型示意图

2 Question 2

在圆周飞行当中，若去除等间距飞行的要求，且每个相机可以观测任一协同机器人，但限定机器人飞行速度区间为 $[0, 2\omega_r r]$ ，加速度区间为 [-g,g]，且 4 个机器人的平均飞行速度应尽可能接近 $\omega_r r$，相机偏航角最大转速为 $\omega_{\psi_c}^m(\omega_{\psi_c}^m = 10\omega_r)$，试推导保证最佳拓扑连接的运动形式。阅读多机器人拓扑保持相关文献 [1, 2]。如每个机器人可通过通讯系统获知全局连通拓扑，试自行选择几种不同类型的椭圆运动，编写可保障视觉拓扑连通的分布式控制算法。若每一单体机器人可由一个单积分系统描述，试设计合理的控制器以保持上述拓扑连接。

2.1 Modeling and Problem Formulation

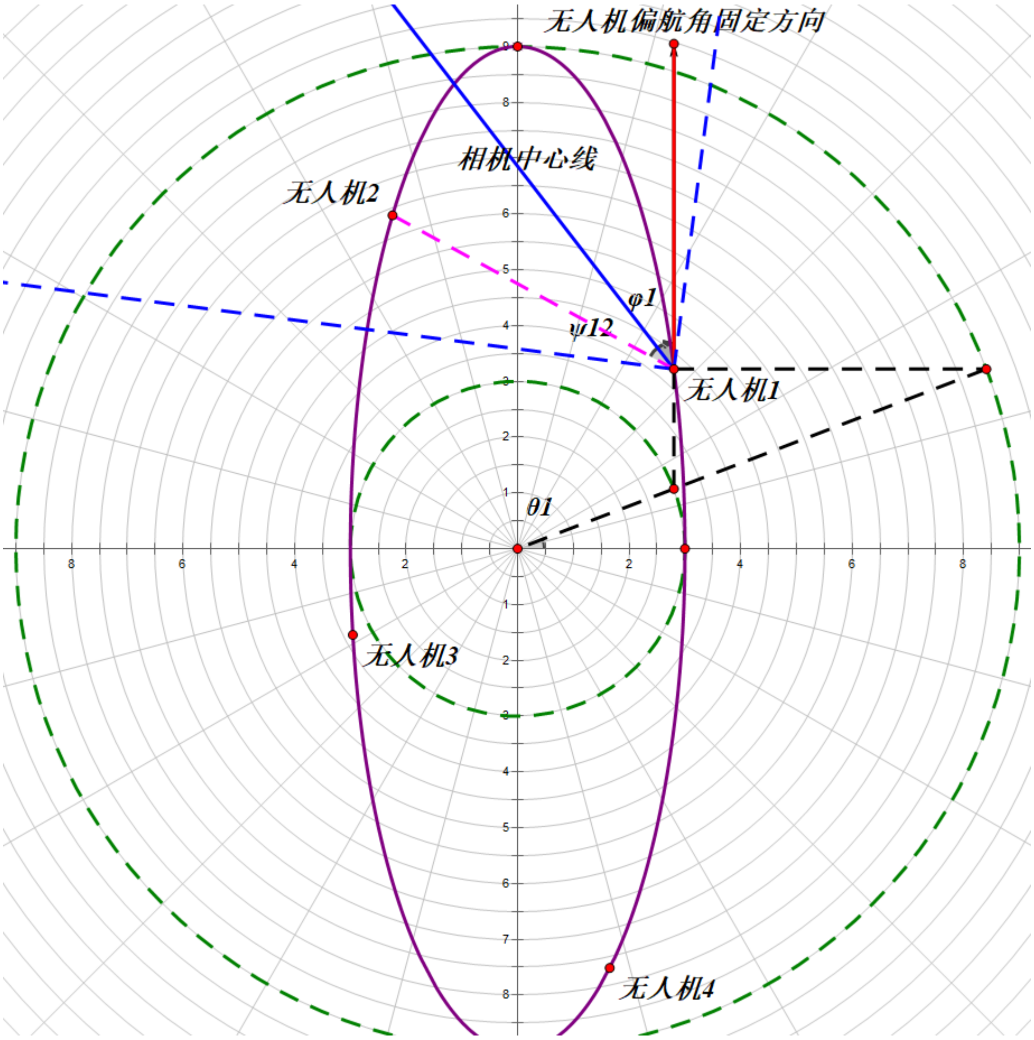

图 2: 问题二示意图

假设椭圆方程 $x^2/a^2+y^2/b^2=1$，机器人 i 在椭圆坐标 $(a\cos\theta_i, b\sin\theta_i), \theta_i\in[0,2\pi)$ 处，其相机偏航角可行范围为 $\psi_c\in(-\pi/2,\pi/2)$，机器人 j 相对于机器人 i 的偏航方向夹角为 $\psi_{ij}\in[-\pi,\pi)$

$$\psi_{ij}=\arctan\left(\frac{y_j-y_i}{x_j-x_i}\right) \tag{1}$$

机器人 j 与机器人 i 连线与机器人 i 距离为

$$l_{ij}=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2} \tag{2}$$

由此设计机器人 i 观察机器人 j 的连通性权重函数为

$$w_{ij}=\exp\left[-c_1(\psi_{ij}-\varphi_i)^2-c_2\varphi_i^2-c_3(l_{ij}-\frac{\sqrt{a^2+b^2}}{2})^2\right] \tag{3}$$

上式指数部分三项的物理含义分别为

(1) $(\psi_{ij}-\varphi_i)^2$ 代表机器人 j 相较于相机 i 中轴的偏转角度性能，机器人位于相机视野中央时观测性能最佳，偏角越大性能越差。

(2) φ_i^2 代表相机偏航角性能，偏航角绝对值越大，相机越容易陷入“旋转死角”不利于后续的相机旋转即连通保持。

(3) $(l_{ij}-\frac{\sqrt{a^2+b^2}}{2})^2$ 代表机器人 j 相较于相机 i 的距离性能，假设距离值等于等距分布时的距离 $2\sqrt{\frac{a^2b^2}{a^2+b^2}}$ 时性能最佳，距离过近时容易发生碰撞，距离过远时相机可视性不好。

(4) c_1, c_2, c_3 分别为各分量的乘子系数。

2.2 Optimal Control

因为机器人间可以进行状态交换，彼此传递观测结果，例如机器人 i 观测到机器人 j 后将相对信息同步给机器人 j 后，j 也可以据此反推出机器人 i 相对于自己的位置等信息。我们认为机器人 i 与机器人 j 的连通性为 i 观察 j 连通性权重与 j 观察 i 之和，表达式如下

$$\begin{aligned}\widetilde{w}_{ij} &= w_{ij}+w_{ji}\\ &= \exp\left[-c_1(\psi_{ij}-\varphi_i)^2-c_2\varphi_i^2-c_3(l_{ij}-\frac{\sqrt{a^2+b^2}}{2})^2\right]\\ &+\exp\left[-c_1(\psi_{ji}-\varphi_j)^2-c_2\varphi_j^2-c_3(l_{ij}-\frac{\sqrt{a^2+b^2}}{2})^2\right]\end{aligned} \tag{4}$$

最终可以列出拉普拉斯矩阵为

$$L=\begin{bmatrix}\widetilde{w}_{12}+\widetilde{w}_{13}+\widetilde{w}_{14} & -\widetilde{w}_{12} & -\widetilde{w}_{13} & -\widetilde{w}_{14}\\ -\widetilde{w}_{12} & \widetilde{w}_{12}+\widetilde{w}_{23}+\widetilde{w}_{24} & -\widetilde{w}_{23} & -\widetilde{w}_{24}\\ -\widetilde{w}_{13} & -\widetilde{w}_{23} & \widetilde{w}_{13}+\widetilde{w}_{23}+\widetilde{w}_{34} & -\widetilde{w}_{34}\\ -\widetilde{w}_{14} & -\widetilde{w}_{24} & -\widetilde{w}_{34} & \widetilde{w}_{14}+\widetilde{w}_{24}+\widetilde{w}_{34}\end{bmatrix} \tag{5}$$

参考文献 [1] 中的梯度控制器，利用 MATLAB 求解 λ_2 的表达式（见附录代码），因结果过于复杂最终采用数值解法，设计控制器如下

$$\begin{aligned}u_i^\theta &= K(\boldsymbol{p})\frac{\partial\lambda_2}{\partial\theta_i}\\ u_i^\varphi &= K(\boldsymbol{p})\frac{\partial\lambda_2}{\partial\varphi_i}\\ s.t.\quad K(\boldsymbol{p}) &= \mathbf{csch}^2(\lambda_2-\epsilon)\end{aligned} \tag{6}$$

3 Question 3

参考上一任务的思路，但若机器人为双积分系统，试设计控制程序，使得所有机器人在作一致性椭圆运动时（长轴对应偏航角为 0 度，且长轴为短轴的 3 倍），并分析视觉系统最佳连通变化情况。

3.1 Optimal Control

在上一题的基础上，对于机器人 i，设计双积分控制器表达式如下

$$\begin{cases} u_i & = \dot{p}_i = \frac{\partial\lambda_2}{\partial p_i} \\ \dot{u}_i & = \ddot{p}_i = \frac{\partial}{\partial t}(\frac{\partial\lambda_2}{\partial p_i}) = \frac{\partial}{\partial p_i}(\frac{\partial\lambda_2}{\partial p_i}) \cdot \frac{\partial p_i}{\partial t} = \frac{\partial}{\partial p_i}(\frac{\partial\lambda_2}{\partial p_i})u_i = \frac{\partial u_i}{\partial p_i}u_i \end{cases} \tag{7}$$

最终完整控制律表达式如下

$$\begin{aligned} u_i^\theta &= \frac{\partial\lambda_2}{\partial\theta_i} \\ \dot{u}_i^\theta &= \frac{\partial u_i^\theta}{\partial\theta_i}u_i^\theta = \frac{\partial}{\partial\theta_i}\frac{\lambda_2}{\partial\theta_i}\cdot\frac{\partial\lambda_2}{\partial\theta_i} \\ u_i^\varphi &= \frac{\partial\lambda_2}{\partial\varphi_i} \\ \dot{u}_i^\varphi &= \frac{\partial u_i^\varphi}{\partial\varphi_i}u_i^\varphi = \frac{\partial}{\partial\varphi_i}\frac{\lambda_2}{\partial\varphi_i}\cdot\frac{\partial\lambda_2}{\partial\varphi_i} \end{aligned} \tag{8}$$

4 Question 4

如果扩大机器人数量（如 10 个机器人），且每一机器人只可获知前后方向各两个机器人的连通拓扑，试编写分布式仿真程序估计表征主动视觉连通的拉氏矩阵次小特征值及其对应特征向量。上述估计是否可保障收敛，试证明你的结论并分析保证收敛的通讯拓朴条件。

4.1 Estimation of the algebraic connectivity of the graph

任一机器人在每次通讯中只能与前后各两个，即一共四个进行通讯。对于机器人 i，无法获知全局的拉普拉斯矩阵 L 和次小特征值 λ_2 的表达式，为了实现连通性的保持，必须对估计表征主动视觉连通的拉氏矩阵次小特征值及其对应特征向量给出定义。由问题2中设计的控制器可知

$$u_i = \frac{\partial\lambda_2}{\partial p_i} = v_2^T\frac{\partial L}{\partial p_i}v_2 \tag{9}$$

参考 [2] 对 λ_2, v_2 进行估计，将 2 估计值记为 $\widetilde{v}_2$ 具体估计及迭代过程定义如下

$$\begin{aligned} \alpha_1^i &= \widetilde{v}_2^i, \quad \alpha_2^i = (\widetilde{v}_2^i)^2 \\ z_1^i &= \mathrm{Ave}(\widetilde{v}_2^i), \quad z_2^i = \mathrm{Ave}((\widetilde{v}_2^i)^2) \\ \dot{z}^i &= \gamma(\alpha^i - z^i) - K_p\sum_{j\in\mathcal{N}_i}(z^i - z^j) + K_i\sum_{j\in\mathcal{N}_i}(\omega^i - \omega^j) \\ \dot{\omega}^i &= -K_i\sum_{j\in\mathcal{N}_i}(z^i - z^j) \\ \dot{\widetilde{v}}_2^i &= -k_1 z_1^i - k_2\sum_{j\in\mathcal{N}_i}a_{ij}(\widetilde{v}_2^i - \widetilde{v}_2^j) - k_3(z_2^i - 1)\widetilde{v}_2^i \end{aligned} \tag{10}$$

需要注意的是，由幂迭代算法 [1]，$\widetilde{v}_2$ 的初始值应当满足条件 $\mathbf{1}^T\widetilde{v}_2 = 0$

得到 v_2 估计值后即可设计控制律如下

$$
\begin{aligned}
u_i &= \widetilde{v}_2^T \frac{\partial L}{\partial p_i} \widetilde{v}_2 = \sum_{j \in \mathcal{N}_i} \frac{\partial a_{ij}}{\partial p_i} (\widetilde{v}_2 - \widetilde{v}_j)^2 \\
a_{ij} &= \widetilde{w}_{ij} \\
&= \exp\left[-c_1(\psi_{ij} - \varphi_i)^2 - c_2\varphi_i^2 - c_3(l_{ij} - 2\sqrt{\frac{a^2b^2}{a^2+b^2}})^2 \right] \\
&+ \exp\left[-c_1(\psi_{ji} - \varphi_j)^2 - c_2\varphi_j^2 - c_3(l_{ij} - 2\sqrt{\frac{a^2b^2}{a^2+b^2}})^2 \right]
\end{aligned}
\tag{11}
$$

在本问题中 a_{ij} 最终控制律需要求解 $\frac{\partial a_{ij}}{\partial \theta_i}$ 和 $\frac{\partial a_{ij}}{\partial \varphi_i}$ 的表达式，利用附录中的 MATLAB 求解得到结果过于复杂，因此这里省略。

为了优化最终的仿真效果，将模型进行简化，修改队形轨迹为圆形，得到新的连通权重函数如下

$$
\begin{aligned}
\widetilde{w}_{ij} &= w_{ij} + w_{ji} \\
&= \exp\left[-c_1(\psi_{ij} - \varphi_i)^2 - c_2\varphi_i^2 - c_3(\theta_j - \theta_i)^2 \right] \\
&+ \exp\left[-c_1(\psi_{ji} - \varphi_j)^2 - c_2\varphi_j^2 - c_3(\theta_j - \theta_i)^2 \right]
\end{aligned}
\tag{12}
$$

由此可以得到控制律为

$$
\begin{aligned}
u_i^\theta &= \sum_{j \in \mathcal{N}_i} \frac{\partial a_{ij}}{\partial \theta_i} (\widetilde{v}_2 - \widetilde{v}_j)^2 \\
&= \sum_{j \in \mathcal{N}_i} \left\{ w_{ij} \left[-c_1(\pi + \theta_j + \theta_i) + 2(\theta_j - \theta_i) \right] + w_{ji} \left[-c_1(\pi + \theta_j + \theta_i) + 2(\theta_j - \theta_i) \right] \right\} (\widetilde{v}_2 - \widetilde{v}_j)^2 \\
u_i^\varphi &= \sum_{j \in \mathcal{N}_i} \frac{\partial a_{ij}}{\partial \varphi_i} (\widetilde{v}_2 - \widetilde{v}_j)^2 \\
&= \sum_{j \in \mathcal{N}_i} \left\{ w_{ij} \left[+c_1(\pi + \theta_j + \theta_i) - 2c_2\varphi_i \right] \right\} (\widetilde{v}_2 - \widetilde{v}_j)^2
\end{aligned}
\tag{13}
$$

参考文献

[1] Peng Yang, Randy A. Freeman, Geoffrey J. Gordon, Kevin M. Lynch, Siddhartha S. Srinivasa, and Rahul Sukthankar. Decentralized estimation and control of graph connectivity in mobile sensor networks. In *2008 American Control Conference*, pages 2678–2683, 2008.

[2] Lorenzo Sabattini, Nikhil Chopra, and Cristian Secchi. On decentralized connectivity maintenance for mobile robotic systems. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 988–993, 2011.

Appendices

附录 A　MATLAB code for solving the expression of λ_2 in Q2

```
% init
clc;clear;
n = 4;
syms a b c1 c2 c3 theta0 theta1 theta2 theta3 phi0 phi1 phi2 phi3
theta = [theta0 theta1 theta2 theta3];
phi = [phi0 phi1 phi2 phi3];

%% calculate position
for i = 1:n
    x(i) = a*cos(theta(i));
    y(i) = b*sin(theta(i));
end
%% calculate psi
for i = 1:n
    for j = 1:n
        psi(i, j) = atan(-(y(j)-y(i))/(x(j)-x(i)));
    end
end
%% calculate l
for i = 1:n
    for j = 1:n
        l(i, j) = sqrt((x(i)-x(j))^2+(y(i)-y(j))^2);
    end
end
%% calculate w
for i = 1:n
    for j = 1:n
        if i == j
            L(i, j) = sym(0);
        elseif j < i
            L(i, j) = L(j, i);
        else
            L(i, j) = -exp(-c1*(psi(i, j)-phi(i))^2-c2*phi(i)^2-c3*(l(i,j)-2*sqrt(a^2*b^2/(a^2+b^2)))^2)...
            - exp(-c1*(psi(j, i)-phi(j))^2-c2*phi(j)^2-c3*(l(j,i)-2*sqrt(a^2*b^2/(a^2+b^2)))^2);
        end
    end
end
for i = 1:n
    for j = 1:n
        if i == j
            continue;
        else
            L(i, i) = L(i, i) - L(i, j);
        end
    end
end
%% calculate lambda2
e = eig(L);
e = arrayfun(@char, e, 'uniform', 0);
writecell(e, 'e.txt');
```

附录 B MATLAB code for solving the expression of u in Q4

```
clc;clear;
syms a b c1 c2 c3 theta_i theta_j phi_i phi_j
x_i = a*cos(theta_i);x_j = a*cos(theta_j);
y_i = b*sin(theta_i);y_j = b*sin(theta_j);
psi_ij = atan((y_j - y_i)/(x_j - x_i));
psi_ji = atan((y_i - y_j)/(x_i - x_j));
l_ij = sqrt((x_i - x_j)^2+(y_i - y_j)^2);
a_ij = - exp(- c1*(psi_ij - phi_i)^2 - c2*phi_i^2-c3*(l_ij - 2*sqrt(a^2*b^2/(a^2+b^2)))^2) ...
    - exp(- c1*(psi_ji - phi_j)^2 - c2*phi_j^2-c3*(l_ij - 2*sqrt(a^2*b^2/(a^2+b^2)))^2);
pretty(diff(a_ij, theta_i))
pretty(diff(a_ij, phi_i))
```

附录 C C++ code for question 2

```
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/Quaternion.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
#include <nav_msgs/Odometry.h>
#include <Eigen/Eigen>
#include <cmath>

#define pi 3.1415926535

using namespace std;
using namespace Eigen;
///>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>全 局 变 量<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

/* 参数定义 */
double a = 2, b = 6;
double c1 = 16/pi/pi, c2 = 16/pi/pi, c3 = 1/sqrt(a*a+b*b)/4;
double z_lock = 2;                                  // 飞行高度
```

```
double k_v = 0.1, k_w = 1.2;  // 比例系数
double epsilon = 1e-3;
double w = 0.15;
//initial position offset
double x_ini[4] = {2.0, 0.0, -2.0, 0.0};
double y_ini[4] = {0.0, 6.0, 0.0, -6.0};
/* 定义变量 */
VectorXd theta(4);
VectorXd phi(4);
VectorXd x(4);
VectorXd y(4);
VectorXd z(4);
VectorXd u_theta(4);
VectorXd u_phi(4);

geometry_msgs::PoseStamped pose0;
geometry_msgs::PoseStamped pose1;
geometry_msgs::PoseStamped pose2;
geometry_msgs::PoseStamped pose3;
nav_msgs::Odometry pose_odom0;
nav_msgs::Odometry pose_odom1;
nav_msgs::Odometry pose_odom2;
nav_msgs::Odometry pose_odom3;
mavros_msgs::State current_state;

///>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>声 明 函 数<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

void update_theta();
void update_state();
geometry_msgs::Vector3 quaternion2euler(geometry_msgs::Quaternion quater);
geometry_msgs::Quaternion euler2quaternion(geometry_msgs::Vector3 euler);
double get_lambda2(VectorXd Theta, VectorXd Phi);
void update_pub();
double get_angle(double angle1, double angle2);

void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}
void plane_pos_cb0(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom0 = *msg;
}
void plane_pos_cb1(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom1 = *msg;
}
void plane_pos_cb2(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom2 = *msg;
}
void plane_pos_cb3(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom3 = *msg;
}

///>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>主 函 数<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

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

    //uav0
    ros::NodeHandle nh0;
    ros::Subscriber state_sub0 = nh0.subscribe<mavros_msgs::State>("uav0/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub0 = nh0.subscribe<nav_msgs::Odometry>("uav0/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub0 = nh0.advertise<geometry_msgs::PoseStamped>("/uav0/mavros/setpoint_position/local", 1(
    ros::ServiceClient arming_client0 = nh0.serviceClient<mavros_msgs::CommandBool>("/uav0/mavros/cmd/arming");
    ros::ServiceClient set_mode_client0 = nh0.serviceClient<mavros_msgs::SetMode>("/uav0/mavros/set_mode");

    //uav1
    ros::NodeHandle nh1;
    ros::Subscriber state_sub1 = nh1.subscribe<mavros_msgs::State>("uav1/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub1 = nh1.subscribe<nav_msgs::Odometry>("uav1/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub1 = nh1.advertise<geometry_msgs::PoseStamped>("/uav1/mavros/setpoint_position/local", 1(
    ros::ServiceClient arming_client1 = nh1.serviceClient<mavros_msgs::CommandBool>("/uav1/mavros/cmd/arming");
    ros::ServiceClient set_mode_client1 = nh1.serviceClient<mavros_msgs::SetMode>("/uav1/mavros/set_mode");

    //uav2
    ros::NodeHandle nh2;
    ros::Subscriber state_sub2 = nh2.subscribe<mavros_msgs::State>("uav2/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub2 = nh2.subscribe<nav_msgs::Odometry>("uav2/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub2 = nh2.advertise<geometry_msgs::PoseStamped>("/uav2/mavros/setpoint_position/local", 1(
    ros::ServiceClient arming_client2 = nh2.serviceClient<mavros_msgs::CommandBool>("/uav2/mavros/cmd/arming");
    ros::ServiceClient set_mode_client2 = nh2.serviceClient<mavros_msgs::SetMode>("/uav2/mavros/set_mode");

    //uav3
    ros::NodeHandle nh3;
    ros::Subscriber state_sub3 = nh3.subscribe<mavros_msgs::State>("uav3/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub3 = nh3.subscribe<nav_msgs::Odometry>("uav3/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub3 = nh3.advertise<geometry_msgs::PoseStamped>("/uav3/mavros/setpoint_position/local", 1(
    ros::ServiceClient arming_client3 = nh3.serviceClient<mavros_msgs::CommandBool>("/uav3/mavros/cmd/arming");
    ros::ServiceClient set_mode_client3 = nh3.serviceClient<mavros_msgs::SetMode>("/uav3/mavros/set_mode");

    ros::Rate rate(10.0);

///>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>解 锁 飞 机<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    // wait for FCU connection
    while(ros::ok() && !current_state.connected){
        ros::spinOnce();
        rate.sleep();
    }
```

```
    pose0.pose.position.x = 0;
    pose0.pose.position.y = 0;
    pose0.pose.position.z = z_lock;
    pose1.pose.position.x = 0;
    pose1.pose.position.y = 0;
    pose1.pose.position.z = z_lock;
    pose2.pose.position.x = 0;
    pose2.pose.position.y = 0;
    pose2.pose.position.z = z_lock;
    pose3.pose.position.x = 0;
    pose3.pose.position.y = 0;
    pose3.pose.position.z = z_lock;

    //send a few setpoints before starting
    for(int i = 10; ros::ok() && i > 0; --i){
        local_pos_pub0.publish(pose0);
        local_pos_pub1.publish(pose1);
        local_pos_pub2.publish(pose2);
        local_pos_pub3.publish(pose3);
        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_client0.call(offb_set_mode) &&
                set_mode_client1.call(offb_set_mode) &&
                set_mode_client2.call(offb_set_mode) &&
                set_mode_client3.call(offb_set_mode) &&
                offb_set_mode.response.mode_sent){
                ROS_INFO("Offboard enabled : 4");
            }
            last_request = ros::Time::now();
        } else {
            if( !current_state.armed &&
                (ros::Time::now() - last_request > ros::Duration(5.0))){
                if( arming_client0.call(arm_cmd) &&
                    arming_client1.call(arm_cmd) &&
                    arming_client2.call(arm_cmd) &&
                    arming_client3.call(arm_cmd) &&
                    arm_cmd.response.success){
                    ROS_INFO("Vehicle armed : 4");
                }
                last_request = ros::Time::now();
            }
        }
        local_pos_pub0.publish(pose0);
        local_pos_pub1.publish(pose1);
        local_pos_pub2.publish(pose2);
        local_pos_pub3.publish(pose3);
        if (pose_odom0.pose.pose.position.z > 1.8) {
            ROS_INFO("plane takeoff !");
            break;
        }
        ros::spinOnce();
        rate.sleep();
    }
///>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>主  循  环<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    while (ros::ok()) {
        update_state();
        VectorXd theta_tmp, phi_tmp;
        for (int i = 0; i < 4; ++i){
            theta_tmp = theta;
            theta_tmp(i) += epsilon;
            u_theta(i) = (get_lambda2(theta_tmp, phi) - get_lambda2(theta, phi))/epsilon;
            phi_tmp = phi;
            phi_tmp(i) += epsilon;
            u_phi(i) = (get_lambda2(theta, phi_tmp) - get_lambda2(theta, phi))/epsilon;
            cout << "uav" << i << ' ' << u_theta(i) << ' ' << u_phi(i) << endl;
        }
        update_pub();
        local_pos_pub0.publish(pose0);
        local_pos_pub1.publish(pose1);
        local_pos_pub2.publish(pose2);
        local_pos_pub3.publish(pose3);
        ros::spinOnce();
        rate.sleep();
    }

    return 0;
}

void update_theta(){
    for (int i = 0; i < 4; ++i){
        if (x(i) > 0){
            theta(i) = atan(a*y(i)/b/x(i));
        }
        else {
            if (y(i) > 0){
```

```
                theta(i) = atan(a*y(i)/b/x(i)) + pi;
            }
            else {
                theta(i) = atan(a*y(i)/b/x(i)) - pi;
            }
        }
    }
}

void update_state(){
    x(0) = pose_odom0.pose.pose.position.x + x_ini[0];
    y(0) = pose_odom0.pose.pose.position.y + y_ini[0];
    z(0) = pose_odom0.pose.pose.position.z;
    phi(0) = quaternion2euler(pose_odom0.pose.pose.orientation).z;
    x(1) = pose_odom1.pose.pose.position.x + x_ini[1];
    y(1) = pose_odom1.pose.pose.position.y + y_ini[1];
    z(1) = pose_odom1.pose.pose.position.z;
    phi(1) = quaternion2euler(pose_odom1.pose.pose.orientation).z;
    x(2) = pose_odom2.pose.pose.position.x + x_ini[2];
    y(2) = pose_odom2.pose.pose.position.y + y_ini[2];
    z(2) = pose_odom2.pose.pose.position.z;
    phi(2) = quaternion2euler(pose_odom2.pose.pose.orientation).z;
    x(3) = pose_odom3.pose.pose.position.x + x_ini[3];
    y(3) = pose_odom3.pose.pose.position.y + y_ini[3];
    z(3) = pose_odom3.pose.pose.position.z;
    phi(3) = quaternion2euler(pose_odom3.pose.pose.orientation).z;
    update_theta();
}

geometry_msgs::Vector3 quaternion2euler(geometry_msgs::Quaternion quater){
    geometry_msgs::Vector3 temp;
    temp.x = atan2(2.0 * (quater.w * quater.x + quater.y * quater.z), 1.0 - 2.0 * (quater.x * quater.x + quater.y * quat
    temp.y = asin(2.0 * (quater.w * quater.y - quater.z * quater.x));
    temp.z = atan2(2.0 * (quater.w * quater.z + quater.x * quater.y), 1.0 - 2.0 * (quater.y * quater.y + quater.z * quat
    return temp;
}

geometry_msgs::Quaternion euler2quaternion(geometry_msgs::Vector3 euler)
{
    geometry_msgs::Quaternion temp;
    temp.w = cos(euler.x/2)*cos(euler.y/2)*cos(euler.z/2) + sin(euler.x/2)*sin(euler.y/2)*sin(euler.z/2);
    temp.x = sin(euler.x/2)*cos(euler.y/2)*cos(euler.z/2) - cos(euler.x/2)*sin(euler.y/2)*sin(euler.z/2);
    temp.y = cos(euler.x/2)*sin(euler.y/2)*cos(euler.z/2) + sin(euler.x/2)*cos(euler.y/2)*sin(euler.z/2);
    temp.z = cos(euler.x/2)*cos(euler.y/2)*sin(euler.z/2) - sin(euler.x/2)*sin(euler.y/2)*cos(euler.z/2);
    return temp;
}

double get_lambda2(VectorXd Theta, VectorXd Phi){
    VectorXd X(4);VectorXd Y(4);
    MatrixXd psi(4, 4);
    MatrixXd l(4, 4);
    MatrixXd lap(4, 4);
    for (int i = 0; i < 4; ++i){
        X(i) = a*cos(Theta(i));
        Y(i) = b*sin(Theta(i));
    }
    for (int i = 0; i < 4; ++i){
        for (int j = 0; j < 4; ++j){
            if (X(j) - X(i) > 0){
                psi(i, j) = atan((Y(j) - Y(i))/(X(j) - X(i)));
            }
            else {
                if (Y(j) - Y(i) > 0){
                    psi(i, j) = atan((Y(j) - Y(i))/(X(j) - X(i))) + pi;
                }
                else {
                    psi(i, j) = atan((Y(j) - Y(i))/(X(j) - X(i))) - pi;
                }
            }
        }
    }
    for (int i = 0; i < 4; ++i){
        for (int j = 0; j < 4; ++j){
            l(i, j) = sqrt((X(j) - X(i))*(X(j) - X(i)) + (Y(j) - Y(i))* (Y(j) - Y(i)));
        }
    }
    for (int i = 0; i < 4; ++i){
        for (int j = 0; j < 4; ++j){
            if (i == j){
                lap(i, j) = 0;
            }
            else {
                if (j < i){
                    lap(i, j) = lap(j, i);
                }
                else {
                    lap(i, j) = -exp(-c1*(psi(i, j) - Phi(i))*(psi(i, j) - Phi(i)) - c2*Phi(i)*Phi(i)\
                     - c3*(l(i, j) - sqrt(a*a+b*b)/2)*(l(i, j) - sqrt(a*a+b*b)/2))\
                    -exp(-c1*(psi(j, i) - Phi(j))*(psi(j, i) - Phi(j)) - c2*Phi(j)*Phi(j)\
                     - c3*(l(i, j) - sqrt(a*a+b*b)/2)*(l(i, j) - sqrt(a*a+b*b)/2));
                }
            }
        }
    }
    for (int i = 0; i < 4; ++i){
        for (int j = 0; j < 4; ++j){
            if (i == j){
```

```
                    continue;
                }
                else {
                    lap(i, i) -= lap(i, j);
                }
            }
        }
        EigenSolver<Eigen::MatrixXd> es(lap);
        MatrixXcd evals = es.eigenvalues();
        VectorXd evalsReal = evals.real();
        double lambda1, lambda2;
        if (evalsReal(0) < evalsReal(1)){
            lambda1 = evalsReal(0);
            lambda2 = evalsReal(1);
        }
        else {
            lambda1 = evalsReal(1);
            lambda2 = evalsReal(0);
        }
        if (evalsReal(2) < lambda1){
            lambda2 = lambda1;
            lambda1 = evalsReal(2);
        }
        else {
            if (evalsReal(2) < lambda2){
                lambda2 = evalsReal(2);
            }
        }
        if (evalsReal(3) < lambda1){
            lambda2 = lambda1;
            lambda1 = evalsReal(3);
        }
        else {
            if (evalsReal(3) < lambda2){
                lambda2 = evalsReal(3);
            }
        }
        return lambda2;
}

void update_pub(){
    geometry_msgs::Vector3 euler;
    euler = quaternion2euler(pose_odom0.pose.pose.orientation);
    euler.z += k_w*u_phi(0);
    pose0.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom1.pose.pose.orientation);
    euler.z += k_w*u_phi(1);
    pose1.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom2.pose.pose.orientation);
    euler.z += k_w*u_phi(2);
    pose2.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom3.pose.pose.orientation);
    euler.z += k_w*u_phi(3);
    pose3.pose.orientation = euler2quaternion(euler);
    pose0.pose.position.x = a*cos(theta(0) + k_v*u_theta(0) + w) - x_ini[0];
    pose1.pose.position.x = a*cos(theta(1) + k_v*u_theta(1) + w) - x_ini[1];
    pose2.pose.position.x = a*cos(theta(2) + k_v*u_theta(2) + w) - x_ini[2];
    pose3.pose.position.x = a*cos(theta(3) + k_v*u_theta(3) + w) - x_ini[3];
    pose0.pose.position.y = b*sin(theta(0) + k_v*u_theta(0) + w) - y_ini[0];
    pose1.pose.position.y = b*sin(theta(1) + k_v*u_theta(1) + w) - y_ini[1];
    pose2.pose.position.y = b*sin(theta(2) + k_v*u_theta(2) + w) - y_ini[2];
    pose3.pose.position.y = b*sin(theta(3) + k_v*u_theta(3) + w) - y_ini[3];
    pose0.pose.position.z = z_lock;
    pose1.pose.position.z = z_lock;
    pose2.pose.position.z = z_lock;
    pose3.pose.position.z = z_lock;
}
```

附录 D C++ code for question 4

```
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/Quaternion.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
#include <nav_msgs/Odometry.h>
#include <eigen3/Eigen/Eigen>
#include <cmath>
#include <stdio.h>
#include <iostream>
#include <time.h>

#define pi 3.1415926535

using namespace std;
using namespace Eigen;
///>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>全 局 变 量<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

/* 参数定义 */
double r = 5;
double c1 = 16/pi/pi, c2 = 16/pi/pi, c3 = 16/pi/pi/3;
```

```
double gama = 100, Kp = 50, Ki = 200, k1 = 18, k2 = 3, k3 = 60;
double z_lock = 2;          // 飞行高度
double k_v = 0.1, k_w = 1;  // 比例系数
double epsilon = 1e−3;
double w = 0.2;
//initial position offset
double x_ini[8];
double y_ini[8];
/* 定义变量 */
VectorXd theta(8);
VectorXd phi(8);
VectorXd x(8);
VectorXd y(8);
VectorXd z(8);
VectorXd u_theta(8);
VectorXd u_phi(8);
MatrixXd v2(8, 8);
MatrixXd z1(8, 8);
VectorXd z2(8);
MatrixXd omega1(8, 8);
VectorXd omega2(8);
MatrixXd psi(8, 8);
MatrixXd lap(8, 8);

geometry_msgs::PoseStamped pose0;
geometry_msgs::PoseStamped pose1;
geometry_msgs::PoseStamped pose2;
geometry_msgs::PoseStamped pose3;
geometry_msgs::PoseStamped pose4;
geometry_msgs::PoseStamped pose5;
geometry_msgs::PoseStamped pose6;
geometry_msgs::PoseStamped pose7;
nav_msgs::Odometry pose_odom0;
nav_msgs::Odometry pose_odom1;
nav_msgs::Odometry pose_odom2;
nav_msgs::Odometry pose_odom3;
nav_msgs::Odometry pose_odom4;
nav_msgs::Odometry pose_odom5;
nav_msgs::Odometry pose_odom6;
nav_msgs::Odometry pose_odom7;
mavros_msgs::State current_state;

//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>声 明 函 数<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

void update_theta(); // 更新机器人在队形中位置角
void update_state(); // 更新机器人状态, 位置+姿态
geometry_msgs::Vector3 quaternion2euler(geometry_msgs::Quaternion quater); // 四元数转欧拉角
geometry_msgs::Quaternion euler2quaternion(geometry_msgs::Vector3 euler);  // 欧拉角转四元数
void update_pub();

void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}
void plane_pos_cb0(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom0 = *msg;
}
void plane_pos_cb1(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom1 = *msg;
}
void plane_pos_cb2(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom2 = *msg;
}
void plane_pos_cb3(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom3 = *msg;
}
void plane_pos_cb4(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom4 = *msg;
}
void plane_pos_cb5(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom5 = *msg;
}
void plane_pos_cb6(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom6 = *msg;
}
void plane_pos_cb7(const nav_msgs::Odometry::ConstPtr &msg){
    pose_odom7 = *msg;
}

//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>主 函 数<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

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

    //uav0
    ros::NodeHandle nh0;
    ros::Subscriber state_sub0 = nh0.subscribe<mavros_msgs::State>("uav0/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub0 = nh0.subscribe<nav_msgs::Odometry>("uav0/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub0 = nh0.advertise<geometry_msgs::PoseStamped>("/uav0/mavros/setpoint_position/local", 10
    ros::ServiceClient arming_client0 = nh0.serviceClient<mavros_msgs::CommandBool>("/uav0/mavros/cmd/arming");
    ros::ServiceClient set_mode_client0 = nh0.serviceClient<mavros_msgs::SetMode>("/uav0/mavros/set_mode");

    //uav1
    ros::NodeHandle nh1;
    ros::Subscriber state_sub1 = nh1.subscribe<mavros_msgs::State>("uav1/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub1 = nh1.subscribe<nav_msgs::Odometry>("uav1/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub1 = nh1.advertise<geometry_msgs::PoseStamped>("/uav1/mavros/setpoint_position/local", 10
    ros::ServiceClient arming_client1 = nh1.serviceClient<mavros_msgs::CommandBool>("/uav1/mavros/cmd/arming");
    ros::ServiceClient set_mode_client1 = nh1.serviceClient<mavros_msgs::SetMode>("/uav1/mavros/set_mode");
```

```

    //uav2
    ros::NodeHandle nh2;
    ros::Subscriber state_sub2 = nh2.subscribe<mavros_msgs::State>("uav2/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub2 = nh2.subscribe<nav_msgs::Odometry>("uav2/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub2 = nh2.advertise<geometry_msgs::PoseStamped>("/uav2/mavros/setpoint_position/local", 10
    ros::ServiceClient arming_client2 = nh2.serviceClient<mavros_msgs::CommandBool>("/uav2/mavros/cmd/arming");
    ros::ServiceClient set_mode_client2 = nh2.serviceClient<mavros_msgs::SetMode>("/uav2/mavros/set_mode");

    //uav3
    ros::NodeHandle nh3;
    ros::Subscriber state_sub3 = nh3.subscribe<mavros_msgs::State>("uav3/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub3 = nh3.subscribe<nav_msgs::Odometry>("uav3/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub3 = nh3.advertise<geometry_msgs::PoseStamped>("/uav3/mavros/setpoint_position/local", 10
    ros::ServiceClient arming_client3 = nh3.serviceClient<mavros_msgs::CommandBool>("/uav3/mavros/cmd/arming");
    ros::ServiceClient set_mode_client3 = nh3.serviceClient<mavros_msgs::SetMode>("/uav3/mavros/set_mode");

    //uav4
    ros::NodeHandle nh4;
    ros::Subscriber state_sub4 = nh4.subscribe<mavros_msgs::State>("uav4/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub4 = nh4.subscribe<nav_msgs::Odometry>("uav4/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub4 = nh4.advertise<geometry_msgs::PoseStamped>("/uav4/mavros/setpoint_position/local", 10
    ros::ServiceClient arming_client4 = nh4.serviceClient<mavros_msgs::CommandBool>("/uav4/mavros/cmd/arming");
    ros::ServiceClient set_mode_client4 = nh4.serviceClient<mavros_msgs::SetMode>("/uav4/mavros/set_mode");

    //uav5
    ros::NodeHandle nh5;
    ros::Subscriber state_sub5 = nh5.subscribe<mavros_msgs::State>("uav5/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub5 = nh5.subscribe<nav_msgs::Odometry>("uav5/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub5 = nh5.advertise<geometry_msgs::PoseStamped>("/uav5/mavros/setpoint_position/local", 10
    ros::ServiceClient arming_client5 = nh5.serviceClient<mavros_msgs::CommandBool>("/uav5/mavros/cmd/arming");
    ros::ServiceClient set_mode_client5 = nh5.serviceClient<mavros_msgs::SetMode>("/uav5/mavros/set_mode");

    //uav6
    ros::NodeHandle nh6;
    ros::Subscriber state_sub6 = nh6.subscribe<mavros_msgs::State>("uav6/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub6 = nh6.subscribe<nav_msgs::Odometry>("uav6/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub6 = nh6.advertise<geometry_msgs::PoseStamped>("/uav6/mavros/setpoint_position/local", 10
    ros::ServiceClient arming_client6 = nh6.serviceClient<mavros_msgs::CommandBool>("/uav6/mavros/cmd/arming");
    ros::ServiceClient set_mode_client6 = nh6.serviceClient<mavros_msgs::SetMode>("/uav6/mavros/set_mode");

    //uav7
    ros::NodeHandle nh7;
    ros::Subscriber state_sub7 = nh7.subscribe<mavros_msgs::State>("uav7/mavros/state", 10, state_cb);
    ros::Subscriber local_pos_sub7 = nh7.subscribe<nav_msgs::Odometry>("uav7/mavros/local_position/odom", 1, plane_pos_c
    ros::Publisher local_pos_pub7 = nh7.advertise<geometry_msgs::PoseStamped>("/uav7/mavros/setpoint_position/local", 10
    ros::ServiceClient arming_client7 = nh7.serviceClient<mavros_msgs::CommandBool>("/uav7/mavros/cmd/arming");
    ros::ServiceClient set_mode_client7 = nh7.serviceClient<mavros_msgs::SetMode>("/uav7/mavros/set_mode");

    ros::Rate rate(10.0);

///>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>解 锁 飞 机<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    for (int i = 0; i < 8; ++i){
        x_ini[i] = r*cos(pi/4*i);
        y_ini[i] = r*sin(pi/4*i);
    }
    // wait for FCU connection
    while(ros::ok() && !current_state.connected){
        ros::spinOnce();
        rate.sleep();
    }

    pose0.pose.position.x = 0;
    pose0.pose.position.y = 0;
    pose0.pose.position.z = z_lock;
    pose1.pose.position.x = 0;
    pose1.pose.position.y = 0;
    pose1.pose.position.z = z_lock;
    pose2.pose.position.x = 0;
    pose2.pose.position.y = 0;
    pose2.pose.position.z = z_lock;
    pose3.pose.position.x = 0;
    pose3.pose.position.y = 0;
    pose3.pose.position.z = z_lock;
    pose4.pose.position.x = 0;
    pose4.pose.position.y = 0;
    pose4.pose.position.z = z_lock;
    pose5.pose.position.x = 0;
    pose5.pose.position.y = 0;
    pose5.pose.position.z = z_lock;
    pose6.pose.position.x = 0;
    pose6.pose.position.y = 0;
    pose6.pose.position.z = z_lock;
    pose7.pose.position.x = 0;
    pose7.pose.position.y = 0;
    pose7.pose.position.z = z_lock;

    //send a few setpoints before starting
    for(int i = 10; ros::ok() && i > 0; --i){
        local_pos_pub0.publish(pose0);
        local_pos_pub1.publish(pose1);
        local_pos_pub2.publish(pose2);
        local_pos_pub3.publish(pose3);
        local_pos_pub4.publish(pose4);
        local_pos_pub5.publish(pose5);
        local_pos_pub6.publish(pose6);
        local_pos_pub7.publish(pose7);
        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_client0.call(offb_set_mode) &&
                set_mode_client1.call(offb_set_mode) &&
                set_mode_client2.call(offb_set_mode) &&
                set_mode_client3.call(offb_set_mode) &&
                set_mode_client4.call(offb_set_mode) &&
                set_mode_client5.call(offb_set_mode) &&
                set_mode_client6.call(offb_set_mode) &&
                set_mode_client7.call(offb_set_mode) &&
                offb_set_mode.response.mode_sent){
                ROS_INFO("Offboard enabled : 8");
            }
            last_request = ros::Time::now();
        } else {
            if( !current_state.armed &&
                (ros::Time::now() - last_request > ros::Duration(5.0))){
                if( arming_client0.call(arm_cmd) &&
                    arming_client1.call(arm_cmd) &&
                    arming_client2.call(arm_cmd) &&
                    arming_client3.call(arm_cmd) &&
                    arming_client4.call(arm_cmd) &&
                    arming_client5.call(arm_cmd) &&
                    arming_client6.call(arm_cmd) &&
                    arming_client7.call(arm_cmd) &&
                    arm_cmd.response.success){
                    ROS_INFO("Vehicle armed : 8");
                }
                last_request = ros::Time::now();
            }
        }
        local_pos_pub0.publish(pose0);
        local_pos_pub1.publish(pose1);
        local_pos_pub2.publish(pose2);
        local_pos_pub3.publish(pose3);
        local_pos_pub4.publish(pose4);
        local_pos_pub5.publish(pose5);
        local_pos_pub6.publish(pose6);
        local_pos_pub7.publish(pose7);
        if (pose_odom0.pose.pose.position.z > 1.8) {
            ROS_INFO("plane takeoff !");
            break;
        }
        ros::spinOnce();
        rate.sleep();
    }
///>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>主 循 环<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    double tmp;
    srand((unsigned)time(NULL));
    for (int i = 0; i < 8; ++i){
        tmp = 0;
        for (int j = 0; j < 7; ++j){
            v2(i, j) = rand() / float(RAND_MAX);
            tmp -= v2(i, j);
        }
        v2(i, 7) = tmp;
        z1.block(i, 0, 1, 8) = v2.block(i, 0, 1, 8);
        MatrixXd tmp = v2.block(i, 0, 1, 8)*v2.block(i, 0, 1, 8).transpose();
        z2(i) = tmp(0, 0);
        for (int j = 0; j < 8; ++j){
            omega1(i, j) = 0;
        }
        omega2(i) = 0;
    }

    while (ros::ok()) {
        update_state();

        for (int i = 0; i < 8; ++i){
            for (int j = 0; j < 8; ++j){
                if (x(j) - x(i) > 0){
                    psi(i, j) = atan((y(j) - y(i))/(x(j) - x(i)));
                }
                else {
                    if (y(j) - x(i) > 0){
                        psi(i, j) = atan((y(j) - y(i))/(x(j) - x(i))) + pi;
                    }
                    else {
                        psi(i, j) = atan((y(j) - y(i))/(x(j) - x(i))) - pi;
                    }
                }
            }
        }
        for (int i = 0; i < 8; ++i){
            for (int j = 0; j < 8; ++j){
```

```
                if (j == i - 1 || j == i - 2 || j == i + 1 || j == i + 2 || j == i + 7 || j == i + 6 || j == i - 7 || j
                    lap(i, j) = -exp(-c1*(psi(i, j) - phi(i))*(psi(i, j) - phi(i)) - c2*phi(i)*phi(i)\
                        - c3*(theta(j) - theta(i))*(theta(j) - theta(i)))\
                        -exp(-c1*(psi(j, i) - phi(j))*(psi(j, i) - phi(j)) - c2*phi(j)*phi(j)\
                        - c3*(theta(j) - theta(i))*(theta(j) - theta(i)));
                }
                else {
                    lap(i, j) = 0;
                }
            }
        }
        for (int i = 0; i < 8; ++i){
            for (int j = 0; j < 8; ++j){
                if (i == j){
                    continue;
                }
                else {
                    lap(i, i) -= lap(i, j);
                }
            }
        }

        for (int i = 0; i < 8; ++i){
            z1.block(i, 0, 1, 8) += gama*(v2.block(i, 0, 1, 8) - z1.block(i, 0, 1, 8))\
             - Kp*(4*z1.block(i, 0, 1, 8) - z1.block(((i-1 >= 0) ? i-1 : 7+i) , 0, 1, 8)\
             - z1.block(((i-2 >= 0) ? i-2 : 6+i) , 0, 1, 8)\
             - z1.block(((i+1 < 8) ? i+1 : i-7) , 0, 1, 8)\
             - z1.block(((i+2 < 8) ? i+2 : i-6) , 0, 1, 8))
             + Ki*(4*omega1.block(i, 0, 1, 8) - omega1.block(((i-1 >= 0) ? i-1 : 7+i) , 0, 1, 8)\
             - omega1.block(((i-2 >= 0) ? i-2 : 6+i) , 0, 1, 8)\
             - omega1.block(((i+1 < 8) ? i+1 : i-7) , 0, 1, 8)\
             - omega1.block(((i+2 < 8) ? i+2 : i-6) , 0, 1, 8));
            omega1.block(i, 0, 1, 8) -= Ki*(4*z1.block(i, 0, 1, 8) - z1.block(((i-1 >= 0) ? i-1 : 7+i) , 0, 1, 8)\
             - z1.block(((i-2 >= 0) ? i-2 : 6+i) , 0, 1, 8)\
             - z1.block(((i+1 < 8) ? i+1 : i-7) , 0, 1, 8)\
             - z1.block(((i+2 < 8) ? i+2 : i-6) , 0, 1, 8));
            MatrixXd tmp1 = v2.block(i, 0, 1, 8)*v2.block(i, 0, 1, 8).transpose();
            z2(i) += gama*(tmp1(0, 0) - z2(i))\
             - Kp*(4*z2(i) - z2((i-1 >= 0) ? i-1 : 7+i)\
             - z2((i-2 >= 0) ? i-2 : 6+i) - z2((i+1 < 8) ? i+1 : i-7) - z2((i+2 < 8) ? i+2 : i-6))
             + Ki*(4*omega2(i) - omega2((i-1 >= 0) ? i-1 : 7+i)\
             - omega2((i-2 >= 0) ? i-2 : 6+i) - omega2((i+1 < 8) ? i+1 : i-7) - omega2((i+2 < 8) ? i+2 : i-6));

            omega2(i) -= Ki*(4*z2(i) - z2((i-1 >= 0) ? i-1 : 7+i)\
             - z2((i-2 >= 0) ? i-2 : 6+i) - z2((i+1 < 8) ? i+1 : i-7) - z2((i+2 < 8) ? i+2 : i-6));
            v2.block(i, 0, 1, 8) += -k1*z1.block(i, 0, 1, 8).array() - (k2*lap.block(i, 0, 1, 0)*v2).array() - (k3*(z2(i
        }

        for (int i = 0; i < 8; ++i){
            u_theta(i) = 0;
            u_phi(i) = 0;
            for (int j = 0; j < 8; ++j){
                if (j == i - 1 || j == i - 2 || j == i + 1 || j == i + 2 || j == i + 7 || j == i + 6 || j == i - 7 || j
                    MatrixXd tmp = -lap(i, j)*(-c1*(pi + theta(j) + theta(i)) + 2*c3*(theta(j)-theta(i)))*(v2.block(i, 0
                    u_theta(i) += tmp(0, 0);
                    tmp = exp(-c1*(psi(i, j) - phi(i))*(psi(i, j) - phi(i)) - c2*phi(i)*phi(i)\
                        - c3*(theta(j) - theta(i))*(theta(j) - theta(i)))*(c1*(pi + theta(j) + theta(i)) - 2*c2*phi(i))*
                    u_phi(i) += tmp(0, 0);
                }
            }
        }

        update_pub();
        local_pos_pub0.publish(pose0);
        local_pos_pub1.publish(pose1);
        local_pos_pub2.publish(pose2);
        local_pos_pub3.publish(pose3);
        local_pos_pub3.publish(pose4);
        local_pos_pub3.publish(pose5);
        local_pos_pub3.publish(pose6);
        local_pos_pub3.publish(pose7);
        ros::spinOnce();
        rate.sleep();
    }

    return 0;
}

void update_theta(){
    for (int i = 0; i < 8; ++i){
        if (x(i) > 0){
            theta(i) = atan(y(i)/x(i));
        }
        else {
            if (y(i) > 0){
                theta(i) = atan(y(i)/x(i)) + pi;
            }
            else {
                theta(i) = atan(y(i)/x(i)) - pi;
            }
        }
    }
}

void update_state(){
    x(0) = pose_odom0.pose.pose.position.x + x_ini[0];
    y(0) = pose_odom0.pose.pose.position.y + y_ini[0];
```

```
    z(0) = pose_odom0.pose.pose.position.z;
    phi(0) = quaternion2euler(pose_odom0.pose.pose.orientation).z;
    x(1) = pose_odom1.pose.pose.position.x + x_ini[1];
    y(1) = pose_odom1.pose.pose.position.y + y_ini[1];
    z(1) = pose_odom1.pose.pose.position.z;
    phi(1) = quaternion2euler(pose_odom1.pose.pose.orientation).z;
    x(2) = pose_odom2.pose.pose.position.x + x_ini[2];
    y(2) = pose_odom2.pose.pose.position.y + y_ini[2];
    z(2) = pose_odom2.pose.pose.position.z;
    phi(2) = quaternion2euler(pose_odom2.pose.pose.orientation).z;
    x(3) = pose_odom3.pose.pose.position.x + x_ini[3];
    y(3) = pose_odom3.pose.pose.position.y + y_ini[3];
    z(3) = pose_odom3.pose.pose.position.z;
    phi(3) = quaternion2euler(pose_odom3.pose.pose.orientation).z;
    x(4) = pose_odom4.pose.pose.position.x + x_ini[4];
    y(4) = pose_odom4.pose.pose.position.y + y_ini[4];
    z(4) = pose_odom4.pose.pose.position.z;
    phi(4) = quaternion2euler(pose_odom4.pose.pose.orientation).z;
    x(5) = pose_odom5.pose.pose.position.x + x_ini[5];
    y(5) = pose_odom5.pose.pose.position.y + y_ini[5];
    z(5) = pose_odom5.pose.pose.position.z;
    phi(5) = quaternion2euler(pose_odom5.pose.pose.orientation).z;
    x(6) = pose_odom6.pose.pose.position.x + x_ini[6];
    y(6) = pose_odom6.pose.pose.position.y + y_ini[6];
    z(6) = pose_odom6.pose.pose.position.z;
    phi(6) = quaternion2euler(pose_odom6.pose.pose.orientation).z;
    x(7) = pose_odom7.pose.pose.position.x + x_ini[7];
    y(7) = pose_odom7.pose.pose.position.y + y_ini[7];
    z(7) = pose_odom7.pose.pose.position.z;
    phi(7) = quaternion2euler(pose_odom7.pose.pose.orientation).z;
    update_theta();
}

geometry_msgs::Vector3 quaternion2euler(geometry_msgs::Quaternion quater){
    geometry_msgs::Vector3 temp;
    temp.x = atan2(2.0 * (quater.w * quater.x + quater.y * quater.z), 1.0 - 2.0 * (quater.x * quater.x + quater.y * quat
    temp.y = asin(2.0 * (quater.w * quater.y - quater.z * quater.x));
    temp.z = atan2(2.0 * (quater.w * quater.z + quater.x * quater.y), 1.0 - 2.0 * (quater.y * quater.y + quater.z * quat
    return temp;
}

geometry_msgs::Quaternion euler2quaternion(geometry_msgs::Vector3 euler)
{
    geometry_msgs::Quaternion temp;
    temp.w = cos(euler.x/2)*cos(euler.y/2)*cos(euler.z/2) + sin(euler.x/2)*sin(euler.y/2)*sin(euler.z/2);
    temp.x = sin(euler.x/2)*cos(euler.y/2)*cos(euler.z/2) - cos(euler.x/2)*sin(euler.y/2)*sin(euler.z/2);
    temp.y = cos(euler.x/2)*sin(euler.y/2)*cos(euler.z/2) + sin(euler.x/2)*cos(euler.y/2)*sin(euler.z/2);
    temp.z = cos(euler.x/2)*cos(euler.y/2)*sin(euler.z/2) - sin(euler.x/2)*sin(euler.y/2)*cos(euler.z/2);
    return temp;
}

void update_pub(){
    geometry_msgs::Vector3 euler;
    euler = quaternion2euler(pose_odom0.pose.pose.orientation);
    euler.z += k_w*u_phi(0);
    pose0.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom1.pose.pose.orientation);
    euler.z += k_w*u_phi(1);
    pose1.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom2.pose.pose.orientation);
    euler.z += k_w*u_phi(2);
    pose2.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom3.pose.pose.orientation);
    euler.z += k_w*u_phi(3);
    pose3.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom4.pose.pose.orientation);
    euler.z += k_w*u_phi(4);
    pose4.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom5.pose.pose.orientation);
    euler.z += k_w*u_phi(5);
    pose5.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom6.pose.pose.orientation);
    euler.z += k_w*u_phi(6);
    pose6.pose.orientation = euler2quaternion(euler);
    euler = quaternion2euler(pose_odom7.pose.pose.orientation);
    euler.z += k_w*u_phi(7);
    pose7.pose.orientation = euler2quaternion(euler);
    pose0.pose.position.x = r*cos(theta(0) + k_v*u_theta(0) + w) - x_ini[0];
    pose1.pose.position.x = r*cos(theta(1) + k_v*u_theta(1) + w) - x_ini[1];
    pose2.pose.position.x = r*cos(theta(2) + k_v*u_theta(2) + w) - x_ini[2];
    pose3.pose.position.x = r*cos(theta(3) + k_v*u_theta(3) + w) - x_ini[3];
    pose4.pose.position.x = r*cos(theta(4) + k_v*u_theta(4) + w) - x_ini[4];
    pose5.pose.position.x = r*cos(theta(5) + k_v*u_theta(5) + w) - x_ini[5];
    pose6.pose.position.x = r*cos(theta(6) + k_v*u_theta(6) + w) - x_ini[6];
    pose7.pose.position.x = r*cos(theta(7) + k_v*u_theta(7) + w) - x_ini[7];
    pose0.pose.position.y = r*sin(theta(0) + k_v*u_theta(0) + w) - y_ini[0];
    pose1.pose.position.y = r*sin(theta(1) + k_v*u_theta(1) + w) - y_ini[1];
    pose2.pose.position.y = r*sin(theta(2) + k_v*u_theta(2) + w) - y_ini[2];
    pose3.pose.position.y = r*sin(theta(3) + k_v*u_theta(3) + w) - y_ini[3];
    pose4.pose.position.y = r*sin(theta(4) + k_v*u_theta(4) + w) - y_ini[4];
    pose5.pose.position.y = r*sin(theta(5) + k_v*u_theta(5) + w) - y_ini[5];
    pose6.pose.position.y = r*sin(theta(6) + k_v*u_theta(6) + w) - y_ini[6];
    pose7.pose.position.y = r*sin(theta(7) + k_v*u_theta(7) + w) - y_ini[7];
    pose0.pose.position.z = z_lock;
    pose1.pose.position.z = z_lock;
    pose2.pose.position.z = z_lock;
    pose3.pose.position.z = z_lock;
```

```
    pose4.pose.position.z = z_lock;
    pose5.pose.position.z = z_lock;
    pose6.pose.position.z = z_lock;
    pose7.pose.position.z = z_lock;
}
```