自动飞行控制系统

目录

1 无人机模型 C 语言实现

1.1 气动数据

1.1.1 大气密度和马赫数

```c
// Return air density and mach number
//====================================
static void uav_density(double H, double VT, double *ru, double *mach) {
    double T, PP, RR, TS, sonic;
    const double K = 34.163195;
    const double RL = 1.225;
    const double C1 = 0.001;
    const double AL = 340.294;
    const double BT = 1.458E-06;

    H = C1 * H / (1 + C1 * H / 6356.766);

    if (H < 11) {
        T = 288.15 - 6.5 * H;
        PP = pow(288.15 / T, -K / 6.5);
    } else if (H < 20) {
        T = 216.65;
        PP = 0.22336 * exp(-K * (H - 11) / 216.65);
    } else if (H < 32) {
        T = 216.65 + (H - 20);
        PP = 0.054032 * pow(216.65 / T, K);
    } else if (H < 47) {
        T = 228.65 + 2.8 * (H - 32);
        PP = 0.0085666 * pow(228.65 / T, K / 2.8);
    } else if (H < 51) {
        T = 270.65;
        PP = 0.0010945 * exp(-K * (H - 47) / 270.65);
    } else if (H < 71) {
        T = 270.65 - 2.8 * (H - 51);
        PP = 0.00066063 * pow(270.65 / T, -K / 2.8);
    } else {
        T = 214.65 - 2 * (H - 71);
        PP = 3.9046e-05 * pow(214.65 / T, -K / 2);
    }

    RR = PP / (T / 288.15);
    TS = T / 288.15;
    sonic = AL * sqrt(TS);
    *ru = RL * RR;
    *mach = VT / sonic;
}
```

1.1.2 阻力系数

```c
// return drag coefficient
//========================
static double uav_CD(double alpha_deg) {
    static double IDX_alpha[9] = {-4.0, -2.0, 0.0, 2.0, 4.0, 8.0, 12.0, 16.0, 20.0};
    static double TBL_CD[9] = {0.026, 0.024, 0.024, 0.028, 0.036,
                               0.061, 0.102, 0.141, 0.173};
```

```
    return uav_interp1(TBL_CD, IDX_alpha, 9, alpha_deg);
}
```

1.1.3 升力系数

```
// return lift coefficient
//=======================
static double uav_CL0(double alpha_deg) {
    static double IDX_alpha[9] = {-4.0, -2.0, 0.0, 2.0, 4.0, 8.0, 12.0, 16.0, 20.0};
    static double TBL_CL0[9] = {-0.219, -0.04, 0.139, 0.299, 0.455,
                                0.766,  1.083, 1.409, 1.743};
    return uav_interp1(TBL_CL0, IDX_alpha, 9, alpha_deg);
}
```

1.1.4 力矩系数

```
// return moment coefficient
//==========================
static double uav_CM(double alpha_deg) {
    static double IDX_alpha[9] = {-4.0, -2.0, 0.0, 2.0, 4.0, 8.0, 12.0, 16.0, 20.0};
    static double TBL_CM0[9] = {0.1161,  0.0777,  0.0393,  0.0009, -0.0375,
                               -0.0759, -0.1527, -0.2295, -0.3063};
    return uav_interp1(TBL_CM0, IDX_alpha, 9, alpha_deg);
}
```

1.2 质量数据

```
// [Aircraft parameters]
static double SA = 1.3536,              // Wing area [m^2]
    b = 3.2,                            // Wingspan [m]
    cbar = 0.423,                       // Mean aerodynamic chord [m]
    Ixx = 1.71, Iyy = 5.54, Izz = 4.15, // Moments of inertia [kg*m^2]
    Ixy = 0, Ixz = 0, Iyz = 0,          // Products of inertia [kg*m^2]
    g = 9.81, mass = 17;                // Gravitational acceleration and mass
```

1.3 动力数据

设定无人机推重比为 0.25。

```
Pow = eng / 100 * (mass * g / 4.0);
```

1.4 微分方程

```
void model6dof(double t, double x[], double u[], double dx[], int dim) {
    double Vt, alpha, beta, phi, theta, psi, P, Q, R, PN, PE, H;
    double alpha_deg, beta_deg;
    double dVt, dalpha, dbeta, dphi, dtheta, dpsi, dP, dQ, dR, dPN, dPE, dH;
    double salpha, sbeta, sphi, stheta, spsi, calpha, cbeta, cphi, ctheta, cpsi;
```

```
double ele, ail, rud, eng, Pow;
double U, V, W;
double dU, dV, dW;
double ru, mach, qs;
double CD, CL0, CM0, D, L, Y;
double Lbar, M, N;

const double CL_ele = 0.00636;
const double CY_beta = -0.00909;
const double CR_beta = -0.00600, CR_ail = -0.003618, CR_rud = 0.000144,
             CR_P = -0.52568, CR_R = 0.01832;
const double CM_ele = -0.02052, CM_Q = -9.3136 / 3.0, CM_dalpha = -4.0258;
const double CN_beta = 0.00235, CN_ail = 0.000132, CN_rud = -0.00111,
             CN_P = 0.01792, CN_R = -0.15844;

Vt = x[0];
alpha = x[1];
beta = x[2];
phi = x[3];
theta = x[4];
psi = x[5];
P = x[6];
Q = x[7];
R = x[8];
PN = x[9];
PE = x[10];
H = x[11];

// control input
ele = u[0];  // elevator deflection angle [deg]
ail = u[1];  // aileron  deflection angle [deg]
rud = u[2];  // aileron  deflection angle [deg]
eng = u[3];  // engine input

uav_density(H, Vt, &ru, &mach);  // [air density] [mach number]
qs = SA * (ru * Vt * Vt / 2);    // [Dynamic pressure](kg/m^2)

alpha_deg = alpha * 180.0 / M_PI;
beta_deg = beta * 180.0 / M_PI;

salpha = sin(alpha);
calpha = cos(alpha);
sbeta = sin(beta);
cbeta = cos(beta);
sphi = sin(phi);
cphi = cos(phi);
stheta = sin(theta);
ctheta = cos(theta);
spsi = sin(psi);
cpsi = cos(psi);

U = Vt * calpha * cbeta;
V = Vt * sbeta;
W = Vt * salpha * cbeta;
double uvw[3] = {U, V, W};  // [u,v,w]  ----- body axis velocity

Pow = eng / 100 * (mass * g / 4.0);

double J[3][3] = {
```

```
    {Ixx, 0, -Ixz}, {0, Iyy, 0}, {-Ixz, 0, Izz}};  // [inertia matrix]
double J_inv[3][3] = {
    {-Izz / (Ixz * Ixz - Ixx * Izz), 0, -Ixz / (Ixz * Ixz - Ixx * Izz)},
    {0, 1 / Iyy, 0},
    {-Ixz / (Ixz * Ixz - Ixx * Izz), 0, -Ixx / (Ixz * Ixz - Ixx * Izz)}};
double S[3][3] = {
    {calpha * cbeta, -calpha * sbeta, -salpha},
    {sbeta, cbeta, 0},
    {salpha * cbeta, -salpha * sbeta, calpha}};  // [rotation matrix]
double B[3][3] = {
    {ctheta * cpsi, ctheta * spsi, -stheta},
    {sphi * stheta * cpsi - cphi * spsi, sphi * stheta * spsi + cphi * cpsi,
     sphi * ctheta},
    {cphi * stheta * cpsi + sphi * spsi, cphi * stheta * spsi - sphi * cpsi,
     cphi * ctheta}};            // [rotation matrix]
double pqr[3] = {P, Q, R};       // [roll rate] [yaw rate] [pitch rate]
double Pow_v[3] = {Pow, 0, 0};  // [thrust vector]

qs = SA * ru * Vt * Vt / 2;  // [Dynamic pressure](kg/m^2)

CD = uav_CD(alpha_deg);  // [drag coefficient]
CL0 = uav_CL0(alpha_deg);
CM0 = uav_CM(alpha_deg);  // [moment coefficient]

D = qs * CD;
Y = qs * CY_beta * beta_deg;
L = qs * (CL0 + CL_ele * ele);

double F0_v[3] = {-D, Y, -L};  // [force vector]
double F_v[3];
double Fxyz[3];
multiply(&S[0][0], F0_v, F_v, 3, 3, 3, 1);  // F = S * [-D; Y; -L]
add(Pow_v, F_v, Fxyz, 3, 1);  // Fxyz = [Pow; 0; 0] + S * [-D; Y; -L];

// duvw = Fxyz / mass - cross(pqr, uvw) + g * [-stheta; sphi * ctheta;
// cphi * ctheta];
double duvw[3] = {Fxyz[0] / mass, Fxyz[1] / mass, Fxyz[2] / mass};
vector3 pqr_v = {P, Q, R};
vector3 uvw_v = {U, V, W};
vector3 cv1 = cross(pqr_v, uvw_v);  // cross(pqr, uvw)
double v1[3] = {cv1.x, cv1.y, cv1.z};
subtract(duvw, v1, duvw, 3, 1);
double c2[3] = {-stheta * g, sphi * ctheta * g, cphi * ctheta * g};
add(duvw, c2, duvw, 3, 1);
dU = duvw[0];
dV = duvw[1];
dW = duvw[2];

dVt = (U * dU + V * dV + W * dW) / Vt;
dbeta = (dV * Vt - V * dVt) / (Vt * Vt * cbeta);
dalpha = (U * dW - W * dU) / (U * U + W * W);

Lbar = qs * b *
       (CR_beta * beta_deg + CR_ail * ail + CR_rud * rud +
        (CR_P * P + CR_R * R) * b / Vt / 2.0);
M = qs * cbar * (CM0 + CM_ele * ele + CM_Q * Q * cbar / Vt / 2.0);
N = qs * b *
    (CN_beta * beta_deg + CN_ail * ail + CN_rud * rud +
     (CN_P * P + CN_R * R) * b / Vt / 2.0);
```

```
    // dpqr = inv(J) * (-cross(pqr, (J * pqr)) + [Lbar; M; N]);
    double dpqr[3];
    double v3[3];
    multiply(&J[0][0], pqr, v3, 3, 3, 3, 1);  // v3 = J*pqr
    vector3 v3_v = {v3[0], v3[1], v3[2]};
    vector3 c4 = cross(pqr_v, v3_v);
    double v4[3] = {-c4.x, -c4.y, -c4.z};  // v4 = -pqr x (J*pqr)
    multiply(&J_inv[0][0], v4, dpqr, 3, 3, 3,
             1);  // J_inv * -(pqr x (J*pqr))
    double v5[3] = {Lbar, M, N};
    add(dpqr, v5, dpqr, 3, 1);  // dpqr = J_inv*(pqr x (J*pqr)) + [Lbar,M,N]
    dP = dpqr[0];
    dQ = dpqr[1];
    dR = dpqr[2];

    dphi = P + (stheta / ctheta) * (Q * sphi + R * cphi);
    dtheta = Q * cphi - R * sphi;
    dpsi = (Q * sphi + R * cphi) / ctheta;

    // dVe = B' * uvw;
    double B_t[3][3];
    transpose(&B[0][0], &B_t[0][0], 3, 3);  // B_t = B^T
    double dVe[3];
    multiply(&B_t[0][0], uvw, dVe, 3, 3, 3, 1);  // dVe = B^T * uvw
    dPN = dVe[0];
    dPE = dVe[1];
    dH = -dVe[2];  // [dVe] = [dPN,dPE,dH]
    ac_dH = dH;

    dx[0] = dVt;
    dx[1] = dalpha;
    dx[2] = dbeta;
    dx[3] = dphi;
    dx[4] = dtheta;
    dx[5] = dpsi;
    dx[6] = dP;
    dx[7] = dQ;
    dx[8] = dR;
    dx[9] = dPN;
    dx[10] = dPE;
    dx[11] = dH;
}
```

1.5 模型验证

根据 MATLAB 配平线性化脚本给出的配平数据，代入平飞控制函数，进行无人机开环控制：

```
void ctrl_level(void) {
    if (t > 10) flag_Stop = 0;
    speed_cmd = 30;
    theta_cmd = THETA_LEVEL;  // 0.993401430622199
    H_cmd = 50;
}
```

观察仿真输出的高度曲线，如图 1所示，说明仿真模型准确无误，和 MATLAB 仿真结果一致。

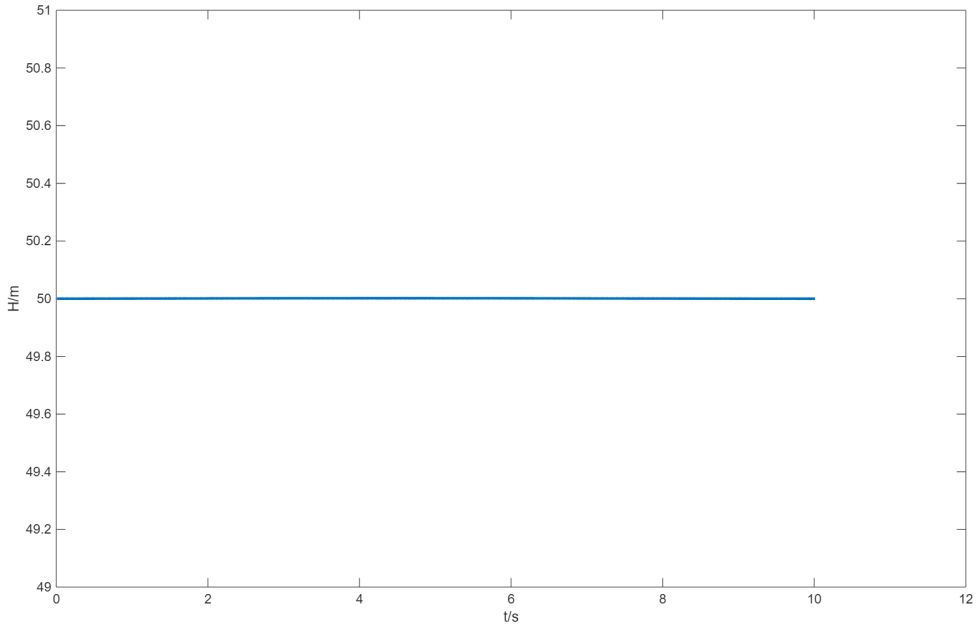

图 1: 平飞控制仿真结果

2 进场着陆控制

2.1 实现过程

根据 Simulink 模型，设计进场着陆控制器，主要包括：

- 高度控制
- 俯仰角控制
- 速度控制

飞行器纵向控制内环为俯仰角控制，外环为高度控制，俯仰角控制器控制升降舵，速度控制器控制油门。其中高度控制和俯仰角控制采用 PID 控制器，速度控制采用 PI 控制器。

2.1.1 高度控制

```
void ctrl_alt(void) {
    static double H_i = 0, H_e = 0, H_prev = 0;

    H_e = H_cmd - ac_H;  // 高度误差

    H_d = (ac_H - H_prev) / DT;  // 高度导数
```

```
    H_prev = ac_H;

    if (H_e > 20)
        H_i += 20 * DT;
    else if (H_e < -20)
        H_i += -20 * DT;
    else
        H_i += H_e * DT;  // 积分项, dt=0.01s

    H_out = KP_H * H_e + KI_H * H_i + KD_H * H_d;  // 高度控制
}
```

2.1.2 俯仰角控制

```
void ctrl_long(void) {  // incremental PID
    static double theta_e = 0, theta_e1 = 0,
                  theta_e2 = 0;  // 当前、上一次、上上次误差
    static double du = 0;

    theta_e = theta_cmd + H_out - ac_theta * Rad2Deg;

    du = KP_THETA * (theta_e - theta_e1) + KI_THETA * theta_e * DT +
         KD_THETA * (theta_e - 2 * theta_e1 + theta_e2) / DT;

    ac_ele -= du;  // 舵量输入相反

    theta_e2 = theta_e1;
    theta_e1 = theta_e;
}
```

2.1.3 速度控制

```
void ctrl_speed() {
    static double speed = 0, speed_e1 = 0,
                  speed_e2 = 0;  // 当前、上一次、上上次误差
    static double du = 0;

    speed = speed_cmd - ac_Vt;

    du = KP_SPEED * (speed - speed_e1) + KI_SPEED * speed * DT +
         KD_SPEED * (speed - 2 * speed_e1 + speed_e2) / DT;

    ac_eng += du;  // 舵量输入相反

    if (ac_eng > 100) ac_eng = 100;
    if (ac_eng < 0) ac_eng = 0;

    speed_e2 = speed_e1;
    speed_e1 = speed;
}
```

2.1.4 控制逻辑

仿照 Stateflow 的状态机设计，进场着陆分为三个阶段，分别是平飞阶段，陡下滑阶段和接地拉飘阶段。控制任务实现如下：

```
void ctrl_approach(void) {
    const double H2 = 2, H1 = 50, path2 = 0.8, path1 = 3.5;
    static double L1, L2;
    L2 = H2 / tan(path2 / Rad2Deg);              // 拉飘开始距离
    L1 = L2 + (H1 - H2) / tan(path1 / Rad2Deg);  // 陡下滑开始距离
    static double Vt_slope, Vt_0;

    switch (ctrl_state) {
        case 0:
            theta_cmd = THETA_LEVEL;
            H_cmd = H1;
            speed_cmd = 30;
            if (ac_PN > -L1) {
                theta_cmd = -1;
                ac_eng = 5;
                Vt_slope = (ac_Vt - 18) / (H1 - H2);
                Vt_0 = ac_Vt;
                ctrl_state++;
            }
            break;
        case 1:
            H_cmd = H2 + (-ac_PN - L2) * tan(path1 / Rad2Deg);
            speed_cmd = Vt_0 - Vt_slope * (H1 - ac_H);
            if (ac_PN > -L2) {
                theta_cmd = 4;
                ac_eng = 0;
                ctrl_state++;
            }
            break;
        case 2:
            H_cmd = -ac_PN * tan(path2 / Rad2Deg);
            speed_cmd = 17;
            if (ac_H <= 0 || t > 50) flag_Stop = 0;
            break;
        default:
            break;
    }
    ctrl_alt();
    ctrl_long();
    ctrl_speed();
}
```

2.2 平飞下滑着陆仿真数据分析

2.2.1 高度控制

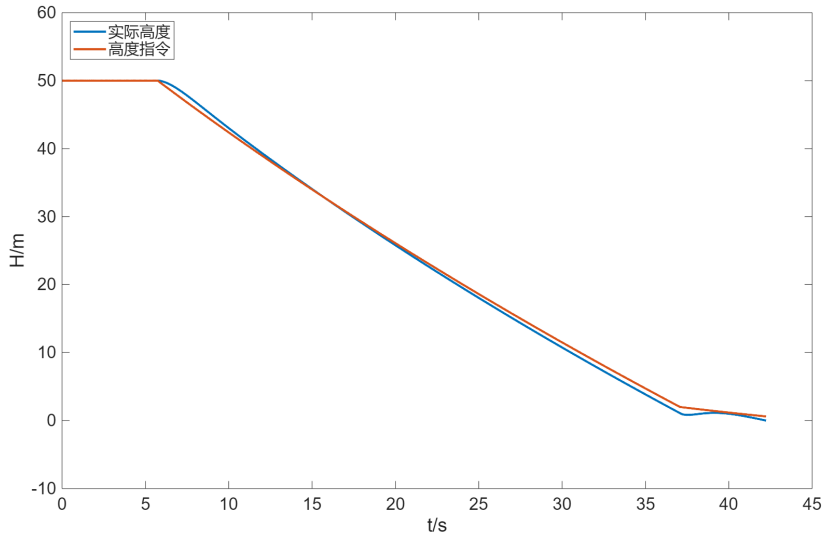

图 2: 高度控制曲线

2.2.2 速度控制

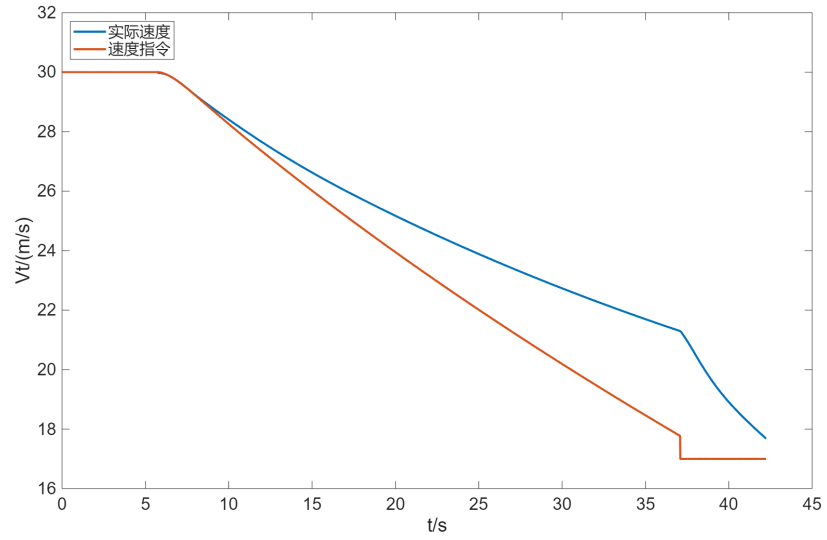

图 3: 速度控制曲线

2.2.3 俯仰角控制

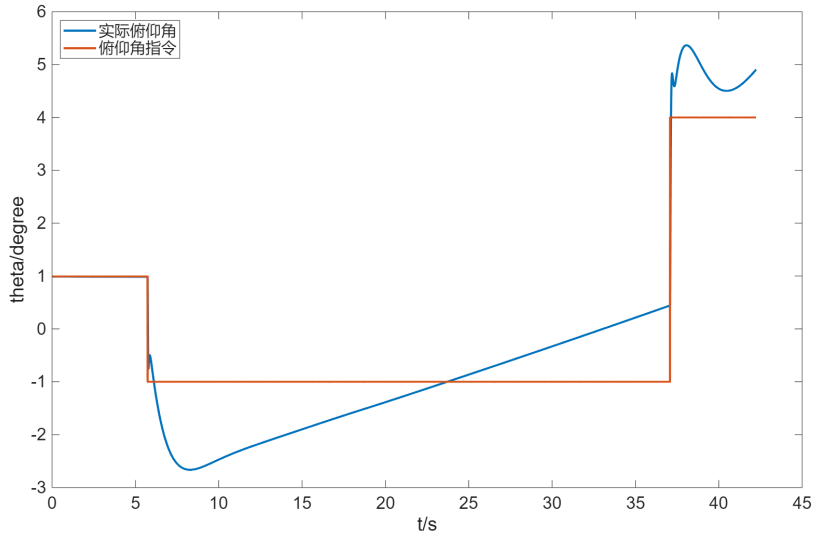

图 4: 俯仰角控制曲线

2.2.4 下沉率

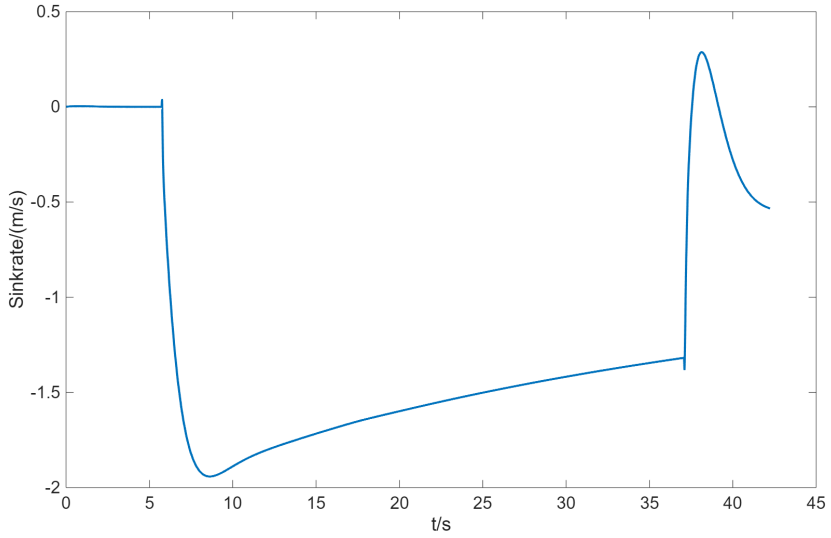

图 5: 下沉率曲线

2.3 与 Matlab 仿真对比

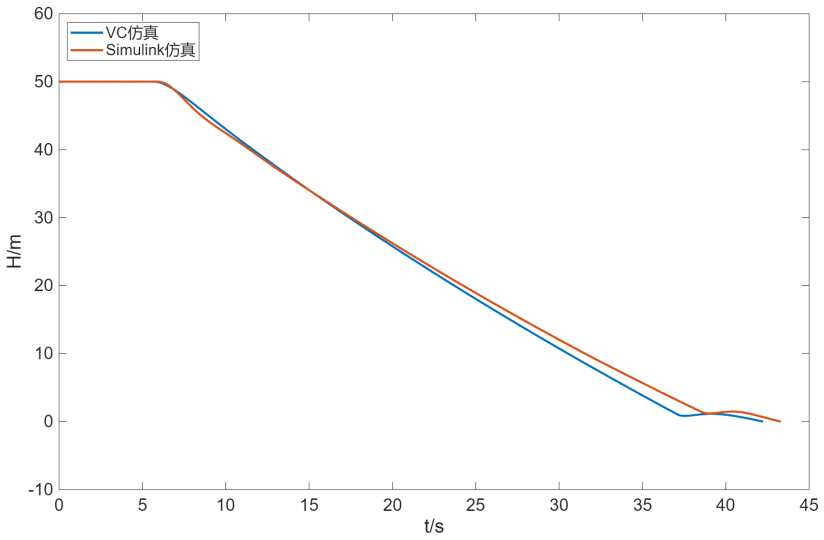

图 6: 高度对比

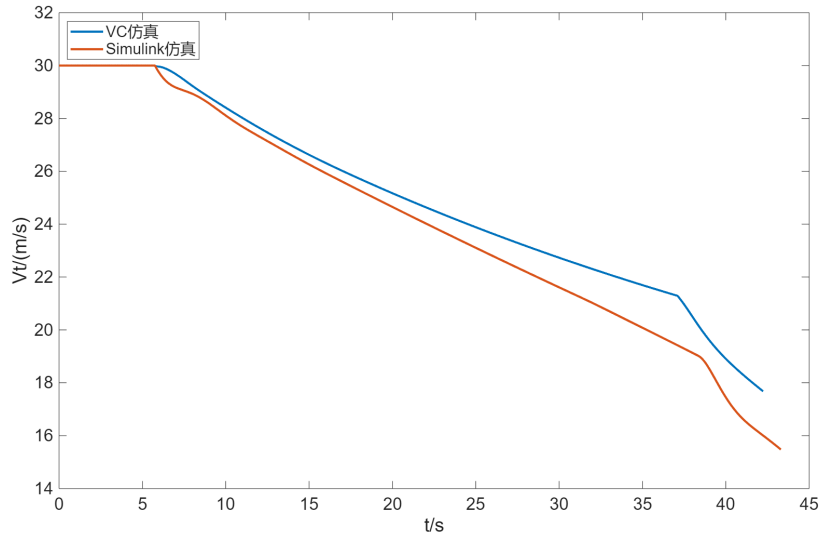

图 7: 速度对比

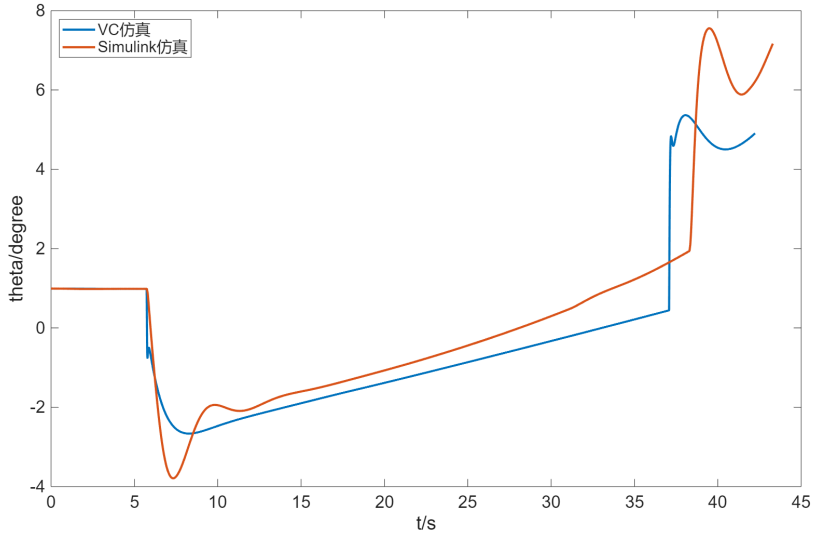

图 8: 俯仰角对比

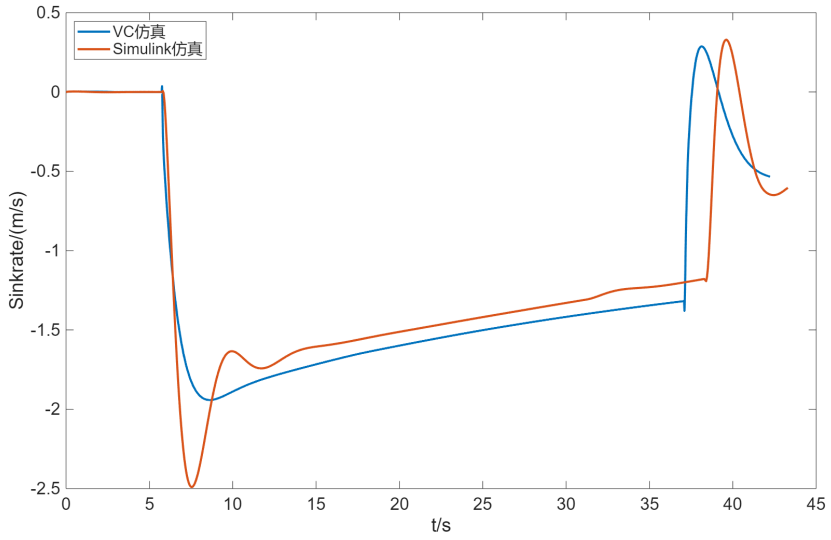

图 9: 下沉率对比

3 四边航路控制

3.1 实现过程

参照 Stateflow 状态机（图 10）设计四边航路控制任务和横侧向控制器，横侧向使用 PID 控制器。

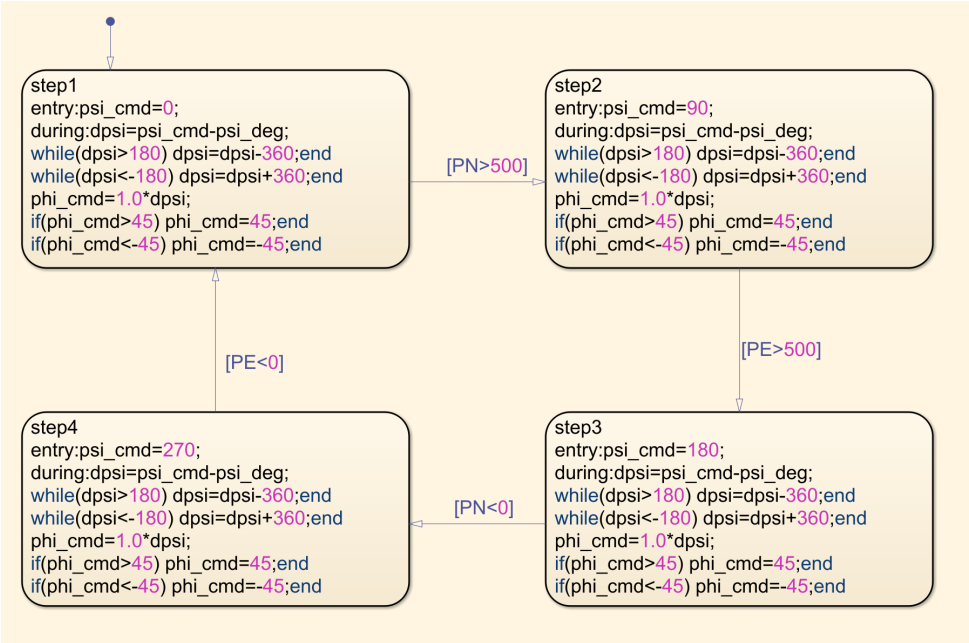

图 10: Statflow 四边航线状态机

3.2 四边航路仿真数据分析

3.2.1 滚转控制

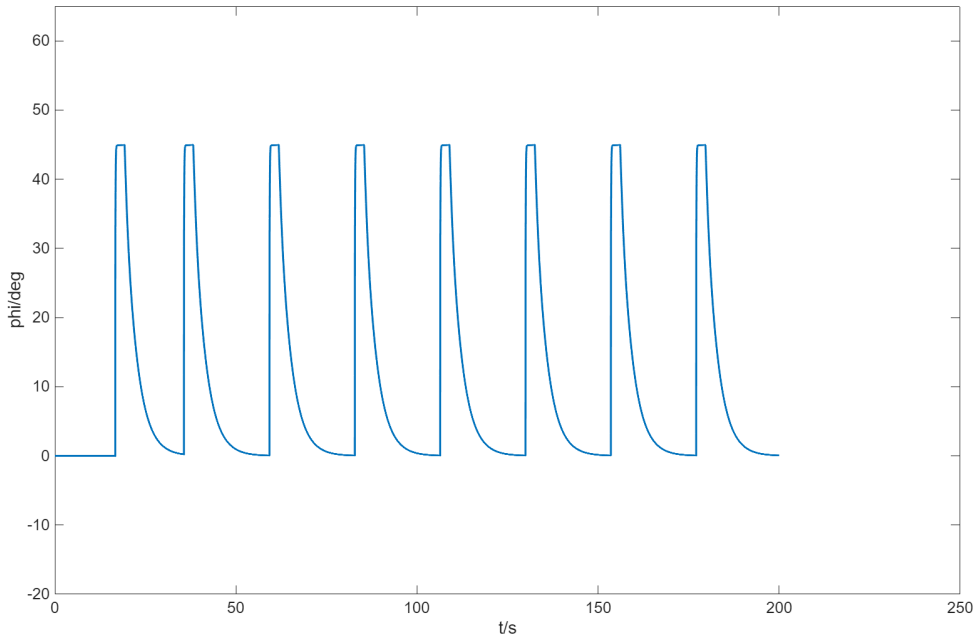

图 11: 滚转角曲线

3.2.2 航向控制

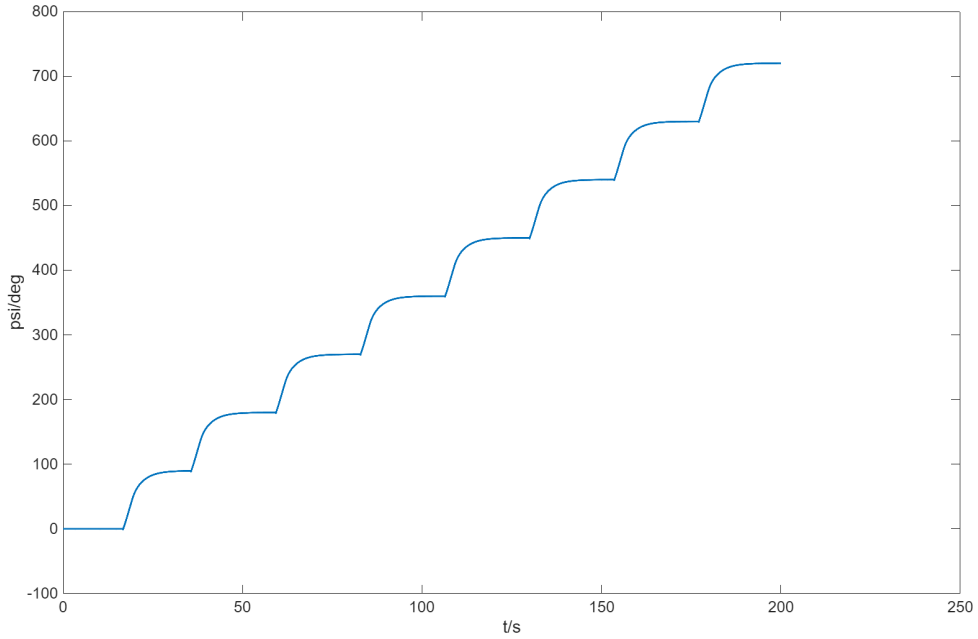

图 12: 航向角曲线

3.2.3 飞行轨迹

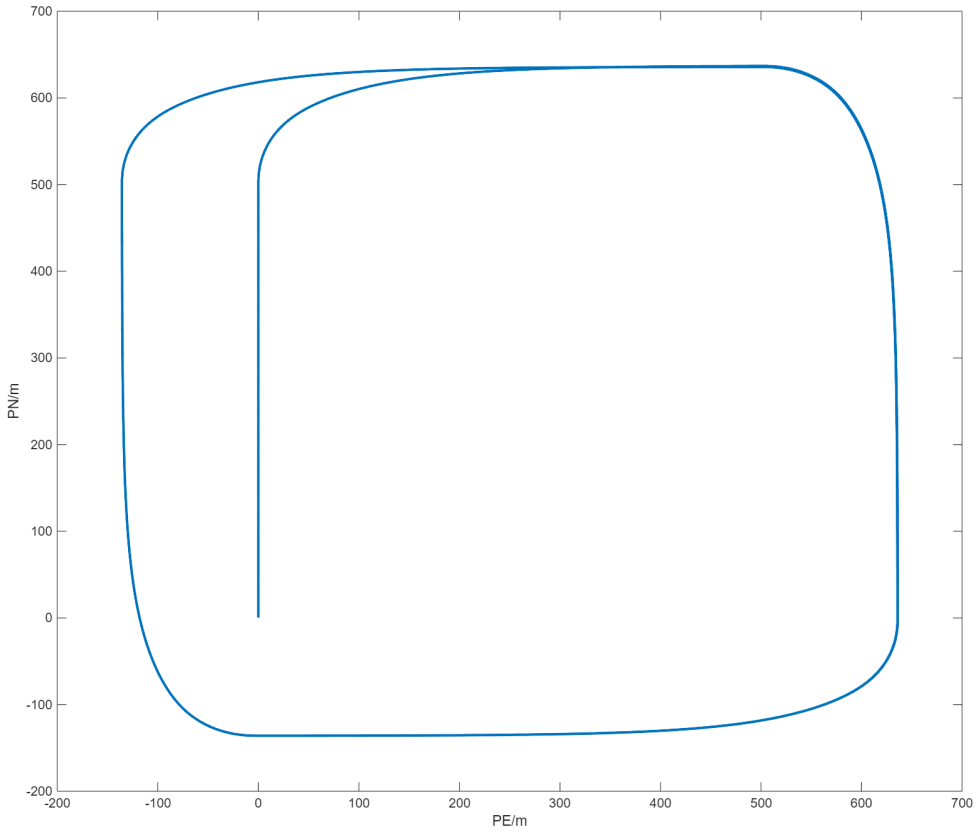

图 13: 飞行轨迹

4 总结

本次实验完成了无人机六自由度模型、进场着陆控制和四边航路控制的 C 语言实现，并对仿真结果进行了详细分析。通过与 MATLAB 仿真结果的对比，验证了 C 语言仿真模型和控制任务的准确性。

在实验过程中，遇到了一些问题，例如在实现 PID 控制器时，如何选择合适的参数以保证系统的稳定性和快速响应。通过不断调试和修改参数，结合理论分析和仿真结果，最终得到了较为满意的控制效果。此外，在实现复杂的状态机逻辑时，也需要仔细设计状态切换条件和控制逻辑，以确保控制器能够在不同阶段正确工作。

通过本次实验，对无人机模型的 C 语言实现和控制算法有了更深入的理解，尤其是对 PID 控制器的设计与调试、状态机的实现以及飞行器动力学模型的构建有了更加系统的认识。同时，本次实验也提高了自己的编程能力、调试能力以及分析问题和解决问题的能力。

未来的工作可以进一步优化控制算法，例如引入自适应控制或鲁棒控制方法，以提高控制系统在复杂环境下的性能。此外，还可以尝试将控制算法移植到嵌入式平台上，进行实际飞行器的硬件在环仿真测试，从而验证算法的工程实用性。

附录 A C 语言矩阵运算库

```c
#include "matrix.h"
#include <stdio.h>

void add(double *a, double *b, double *c, int row, int col) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            c[i * col + j] = a[i * col + j] + b[i * col + j];
        }
    }
}

void subtract(double *a, double *b, double *c, int row, int col) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            c[i * col + j] = a[i * col + j] - b[i * col + j];
        }
    }
}

void multiply(double *a, double *b, double *c, int row1, int col1, int row2, int col2) {
    if (col1 != row2) {
        printf("Matrix multiplication error: incompatible dimensions\n");
        return;
    }
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col2; j++) {
            c[i * col2 + j] = 0;  // c: row1*col2
            for (int k = 0; k < col1; k++) {
                c[i * col2 + j] += a[i * col1 + k] * b[k * col2 + j];
            }
        }
    }
}

void transpose(double *a, double *b, int row, int col) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            b[j * row + i] = a[i * col + j];
        }
    }
}

vector3 cross(vector3 a, vector3 b) {
    vector3 result;
    result.x = a.y * b.z - a.z * b.y;
    result.y = a.z * b.x - a.x * b.z;
    result.z = a.x * b.y - a.y * b.x;
    return result;
}
```