Modeling & Control of A Quadcopter Under Actuator Fault

Yuyang Xie (yx2979)

ME397 Course Project

1 Introduction

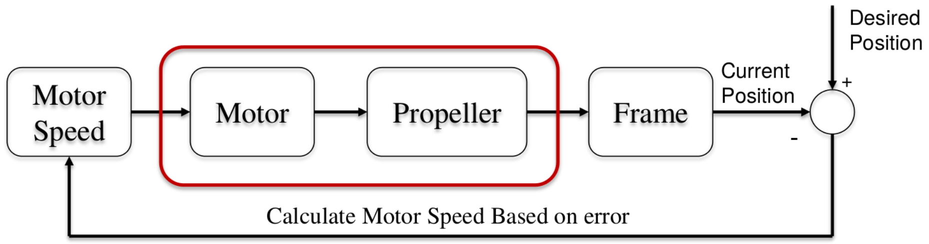

Figure 1: Quadcopter Different Systems

Unmanned Aerial Vehicles is getting more and more popular these days, from aerial photography to city traffic monitoring. As a result, the safety issue has attracted many attentions. The control system in quadcopter operates as shown in 1. Controller collects signal from IMU and compare it with desired postition. Afterward, the controller calculates required motor rotational speed based on motor torque constants as well as difference between current positions and required positions. Then the generated force and torque will drive quadcopter to required position.

Faults in the actuating system (motor + propeller, as shown in the Figure 1) have higher potential in causing quadcopter crash. The project aims at applying what we have learnt during the semester to a quadcopter system. It attempts to provide a fault-tolerant control mechanism to enable quadcopter system to operate robustly when actuator fault happens and a fault diagnosis system to online monitoring key parameters in motor-propeller. The main idea is to adapt to minor fault and calibrate system behavior while execute emergency plan when catastrophic faults happen.

The whole report is divided into three parts — dynamic modeling, controller design and fault tolerant/diagnosis mechanism. More details are given in the following sections.

2 Dynamic Modeling Of Quadcopter System

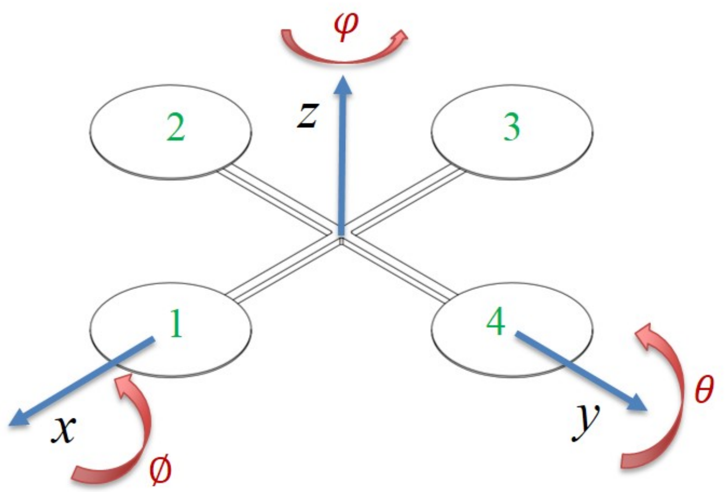

Figure 2: Coordinate Definition of Quadcopter

In this part, the dynamic equations of a quadcopter are derived using Lagrangian approach. Define generalized coordinates as $q's = x, y, z, \phi, \theta, \psi$. Assume symmetricity in the structure, the kinetic energy can be expressed as follows.

2.1 Kinetic Energy

$$T_{tr} = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2)$$

$$T_{ro} = \frac{1}{2}\omega^T I \omega = \frac{1}{2}I_{xx}P^2 + \frac{1}{2}I_{yy}Q^2 + \frac{1}{2}I_{zz}R^2$$

$$T(\dot{q}, q) = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) + \frac{1}{2}I_{xx}P^2 + \frac{1}{2}I_{yy}Q^2 + \frac{1}{2}I_{zz}R^2$$

2.2 Generalized Momentum

$$\omega_b = \begin{pmatrix} P \\ Q \\ R \end{pmatrix} = \begin{pmatrix} -\dot{\psi}sin\theta + \dot{\phi} \\ \dot{\psi}sin\phi cos\theta + \dot{\theta}cos\phi \\ \dot{\psi}cos\phi cos\theta - \dot{\theta}sin\phi \end{pmatrix}$$

$$P_x = \frac{\partial T(\dot{q}, q)}{\partial \dot{x}} = m\dot{x}, P_y = \frac{\partial T(\dot{q}, q)}{\partial \dot{y}} = m\dot{y}, P_z = \frac{\partial T(\dot{q}, q)}{\partial \dot{z}} = m\dot{z}$$

$$P_\phi = \frac{\partial T(\dot{q}, q)}{\partial \dot{\phi}} = I_{xx}(-\dot{\psi}sin\theta + \dot{\phi})$$

$$P_\theta = \frac{\partial T(\dot{q},q)}{\partial \dot{\theta}} = I_{yy}(\dot{\psi} sin\phi cos\theta + \dot{\theta} cos\phi)cos\phi - I_{zz}(\dot{\psi} cos\phi cos\theta - \dot{\theta} sin\phi)sin\phi$$

$$P_\psi = \frac{\partial T(\dot{q},q)}{\partial \dot{\psi}} = -I_{xx}(-\dot{\psi} sin\theta + \dot{\phi})sin\theta + I_{yy}(\dot{\psi} sin\phi cos\theta + \dot{\theta} cos\phi)sin\phi cos\theta + ...$$

$$+I_{zz}(\dot{\psi} cos\phi cos\theta - \dot{\theta} sin\phi)cos\phi cos\theta$$

2.3 Potential Energy

$U_q = mgz$, define $U_q = 0$ when z=0

2.4 Conservative efforts

$$e_x = -\frac{\partial T(\dot{q},q)}{\partial x} + \frac{\partial U_q}{\partial x} = 0, e_y = -\frac{\partial T(\dot{q},q)}{\partial y} + \frac{\partial U_q}{\partial y} = 0, e_z = -\frac{\partial T(\dot{q},q)}{\partial z} + \frac{\partial U_q}{\partial z} = mg$$

$$e_\phi = -\frac{\partial T(\dot{q},q)}{\partial \phi} + \frac{\partial U_q}{\partial \phi} = -[I_{yy}(\dot{\psi} sin\phi cos\theta + \dot{\theta} cos\phi)(\dot{\psi} cos\phi cos\theta - \dot{\theta} sin\phi) + ...$$

$$... + I_{zz}(\dot{\psi} cos\phi cos\theta - \dot{\theta} sin\phi)(-\dot{\psi} sin\phi cos\theta - \dot{\theta} cos\phi)]$$

$$e_\theta = -\frac{\partial T(\dot{q},q)}{\partial \theta} + \frac{\partial U_q}{\partial \theta} = -[I_{xx}(-\dot{\psi} sin\theta + \dot{\phi})(-\dot{\psi} cos\theta) + ...$$

$$... + I_{yy}(\dot{\psi} sin\phi cos\theta + \dot{\theta} cos\phi)(-\dot{\psi} sin\phi sin\theta) + I_{zz}(\dot{\psi} cos\phi cos\theta - \dot{\theta} sin\phi)(-\dot{\psi} cos\phi sin\theta)]$$

$$e_\psi = -\frac{\partial T(\dot{q},q)}{\partial \psi} + \frac{\partial U_q}{\partial \psi} = 0$$

2.5 Net non-conservative power

$$P_{nc} = {F_e}^T \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} + M_1 P + M_2 Q + M_3 R$$

$$F_e = F_T \begin{pmatrix} sin\psi sin\phi - cos\phi cos\psi sin\theta \\ -cos\psi sin\phi + cos\phi sin\psi sin\theta \\ -cos\phi cos\theta \end{pmatrix}, \quad \begin{pmatrix} P \\ Q \\ R \end{pmatrix} = \begin{pmatrix} -\dot{\psi} sin\theta + \dot{\phi} \\ \dot{\psi} sin\phi cos\theta + \dot{\theta} cos\phi \\ \dot{\psi} cos\phi cos\theta - \dot{\theta} sin\phi \end{pmatrix}$$

2.6 State Equations

Translation

$$\dot{P}_x = -e_x + E_x = 0 + F_T(sin\phi sin\phi + cos\phi cos\psi sin\theta) = m\ddot{x}$$

$$\dot{P}_y = -e_y + E_y = 0 + F_T(-cos\psi sin\phi + cos\phi sin\psi sin\theta) = m\ddot{y}$$

$$\dot{P}_z = -e_z + E_z = -mg + F_T(cos\phi cos\theta) = m\ddot{z}$$

Rotation

Assume small Euler angle ($\phi \cong 0, \theta \cong 0, \psi \cong 0$)
We will have $e_\phi \cong (I_{xx} - I_{yy})\dot{\theta}\dot{\psi}, e_\theta = I_{xx}\dot{\phi}\dot{\psi}, e_\psi = 0$ Also we know the following relationship, $\dot{P}_\phi = I_{xx}(\ddot{\phi} - \dot{\psi}\dot{\theta}), \dot{P}_\theta = I_{yy}\ddot{\theta} + (I_{yy} - I_{zz})\dot{\psi}\dot{\phi}, \dot{P}_\psi = I_{zz}\ddot{\psi} + (I_{yy} - I_{zz} - I_{xx})\dot{\phi}\dot{\theta}$

$$\dot{P}_\phi = -e_\phi + E_\phi = (-I_{zz} + I_{yy})\dot{\theta}\dot{\psi} + M_1 = I_{xx}(\ddot{\phi} - \dot{\psi}\dot{\theta})$$

$$\dot{P}_\theta = -e_\theta + E_\theta = -I_{xx}\dot{\phi}\dot{\psi} + M_2 = I_{yy}\ddot{\theta} + (I_{yy} - I_{zz})\dot{\psi}\dot{\phi}$$

$$\dot{P}_\psi = -e_\psi + E_\psi = M_3 = I_{zz}\ddot{\psi} + (I_{yy} - I_{zz} - I_{xx})\dot{\phi}\dot{\theta}$$

3 Controller Design

The main purpose of the designed controller is to perform position tracking. This can be splitted into two parts. One is attitude control and the other one is position control in x-y plane. The controllers are majorly PD controller, as suggested in [1].

3.1 Altitude Control

In altitude control, we hope to keep the quadcopter hovering at certain height. Therefore, $u_z = kp_z(z_d - z) + kp_{zdot}(\dot{z_d} - \dot{z})$. Then thrust can be calculated as $F_T = \frac{m(g+u_z)}{cos\phi cos\theta}$.

3.2 Positon Control

The position control try to regulate quadcopter in a certain position or follow a certain trajectory. It can be defined as

$$u_x = kp_x(x_d - x) + kp_{xdot}(\dot{x_d} - \dot{x}), u_y = kp_y(y_d - y) + kp_{ydot}(\dot{y_d} - \dot{y})$$

In order to achieve u_x and u_y, another layer of PD control is applied to pose angle, which can be expressed as

$$\theta_d = tan^{-1}(\frac{u_x cos\psi + u_y sin\psi}{g + u_z}), \phi_d = tan^{-1}(\frac{u_x sin\phi - u_y cos\phi}{g + u_z})cos\theta$$

After getting required pose angles, the moment around different axises are given as

$$M_\phi = kp_\phi(\phi_d - \phi) + kp_{\dot{\phi}}(\dot{\phi}_d - \dot{\phi})$$

$$M_\theta = kp_\theta(\theta_d - \theta) + kp_{\dot{\theta}}(\dot{\theta}_d - \dot{\theta})$$

$$M_\psi = kp_\psi(\psi_d - \psi) + kp_{\dot{\psi}}(\dot{\psi}_d - \dot{\psi})$$

4 Acuator Fault Diagnosis & Fault Tolerant Mechanism

Denote $s = (x, y, z, \phi, \theta, \psi)^T$ and $u = (u_1, u_2, u_3, u_4)^T$, where $u_i = {o_i}^2$ and o_i is the rotational velocity of motor i

Then the dynamic equations can be written in the following form

$$\frac{d}{dt}\begin{pmatrix} s \\ \dot{s} \end{pmatrix} = \begin{pmatrix} \dot{s} \\ f(s) \end{pmatrix} + \begin{pmatrix} 0 \\ g(s)M_{act}u \end{pmatrix}$$

$$M_{act} = \begin{bmatrix} kL_1 & kL_2 & kL_3 & kL_4 \\ 0 & -kL_2 & 0 & kL_4 \\ -kL_1 & 0 & kL_3 & 0 \\ kD_1 & -kD_2 & kD_3 & -kD_4 \end{bmatrix}, \quad kL_i = LD\_ratio * kD_i$$

The fault to deal with in the project is actuator fault (including motor fault or propeller fault). Assuming that we can get full measurements from IMU through sensor fusion (including position s, velocity $\dot{s}$ and acceleration $\ddot{s}$), we can then use temporal information to online estimate parameters from input (motor angular velocity) to corresponding force and moment applied to propellers. Assume the lift to drag ratio is unchanged during the process, then we have

$$\frac{d}{dt}\begin{pmatrix} s \\ \dot{s} \end{pmatrix} - \begin{pmatrix} \dot{s} \\ f(s) \end{pmatrix} = \begin{bmatrix} g_{T_1}u_1 & g_{T_1}u_2 & g_{T_1}u_3 & g_{T_1}u_4 \\ g_{T_2}u_1 & g_{T_2}u_2 & g_{T_2}u_3 & g_{T_2}u_4 \\ g_{T_3}u_1 & g_{T_3}u_2 & g_{T_3}u_3 & g_{T_3}u_4 \\ 0 & -g_{M_1}u_2 & 0 & g_{M_1}u_4 \\ -g_{M_2}u_1 & 0 & g_{M_2}u_3 & 0 \\ g_{M_3}\frac{u_1}{LD} & g_{M_3}\frac{u_2}{LD} & g_{M_3}\frac{u_3}{LD} & g_{M_3}\frac{u_4}{LD} \end{bmatrix} \begin{pmatrix} kL_1 \\ kL_2 \\ kL_3 \\ kL_4 \end{pmatrix}$$

$$\Rightarrow Y = X\beta$$

where LD is the lift to drag ratio (LD_ratio), $g(s) = [g_{T_1}, g_{T_2}, g_{T_3}, g_{M_1}, g_{M_2}, g_{M_3}]^T$, $\beta = \begin{pmatrix} kL_1 \\ kL_2 \\ kL_3 \\ kL_4 \end{pmatrix}$, and matrices Y,X are measurable.

For the purpose of fault diagnosis, we want to online estimate β vector. In this project, an algorithm called normalized gradient descent is utilized to online update β. It is given as:

$$\beta = \beta + \gamma X(\alpha I + X^T X)^{-1}(Y - X\beta)$$

where γ is the adaptation rate and α is used to guarantee that the inverse of matrix exists.

With the proposed scheme, we can online monitor parameter changes and adjust our control accordingly. It also makes monitoring schemes like Hagglund methods, Favier and Arruda method, etc become feasible.

5 Result & Discussion

In the section, we first let quadcopter follow a trajectory defined below.

$$s(t) = \begin{cases} x = 0, y = 0, z = 0.5, \quad t < 50 \\ x = 0.25, y = 0, z = 0.5, \quad 50 \leq t < 100 \\ x = 0.25, y = 0.25, z = 0.5, \quad 100 \leq t < 150 \\ x = 0, y = 0.25, z = 0.5m, 150 \leq t < 200 \\ x = 0, y = 0, z = 0.5m, \quad 200 \leq t \end{cases}$$

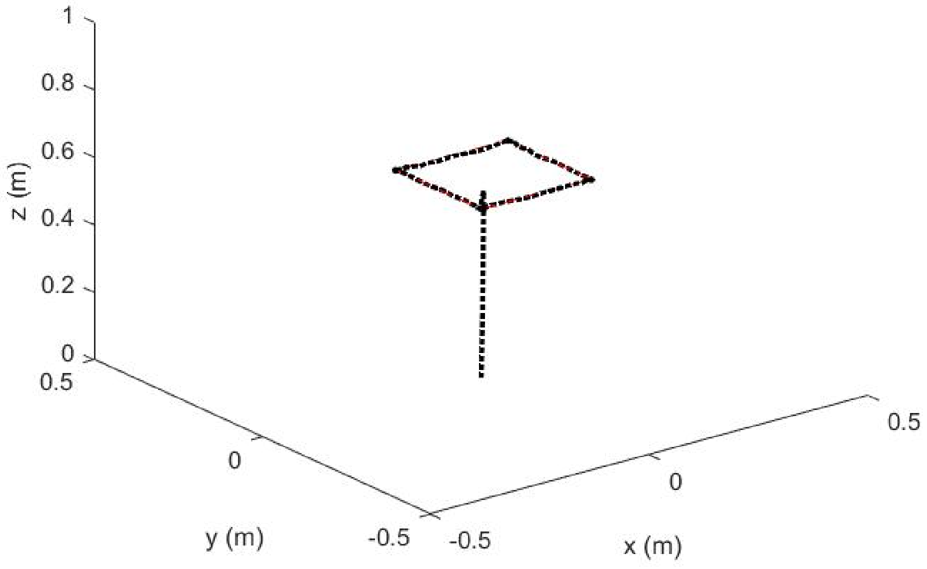

Figure 3: Normal Trajectory (Without Fault)

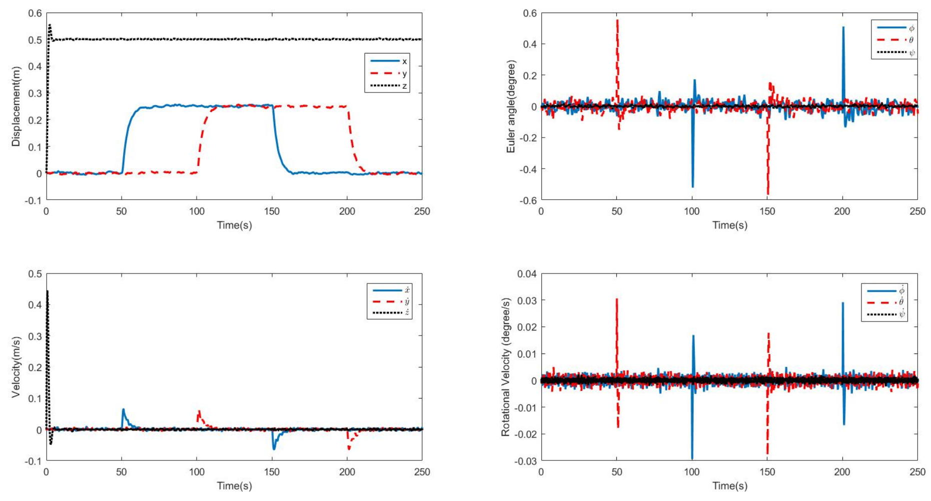

Figure 4: Normal States (Without Fault)

The actuator fault we consider is a sudden loss of 50% thrust in one of the motors. Firstly, let us take a look on how the quadcopter system will respond to this situation with the PD controllers and without parameter learning. As we can see, the quadcopter will go off desired trajectory and not come back to normal orbit.

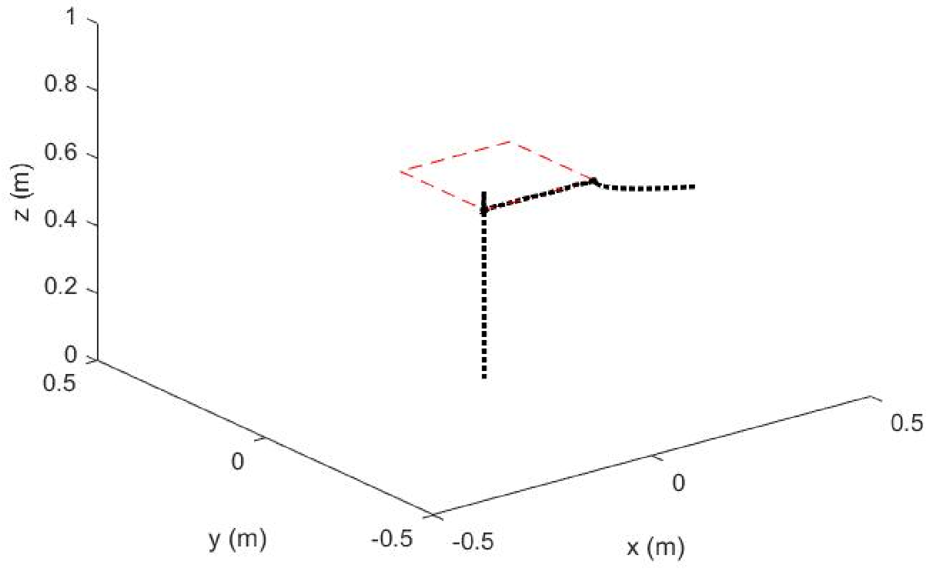

Figure 5: Trajectory With Fault, No Parameter Learning

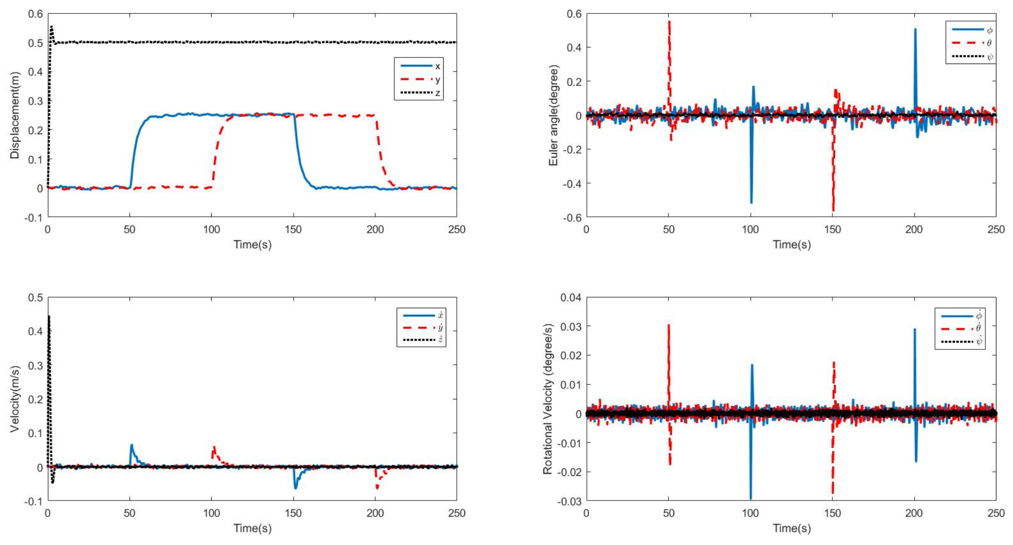

Figure 6: States With Fault, No Parameter Learning

If we can real-time learn the parameters, the motor rotational velocity can be solved accurately. As a result, the quadcopter can recover to desired orbit.

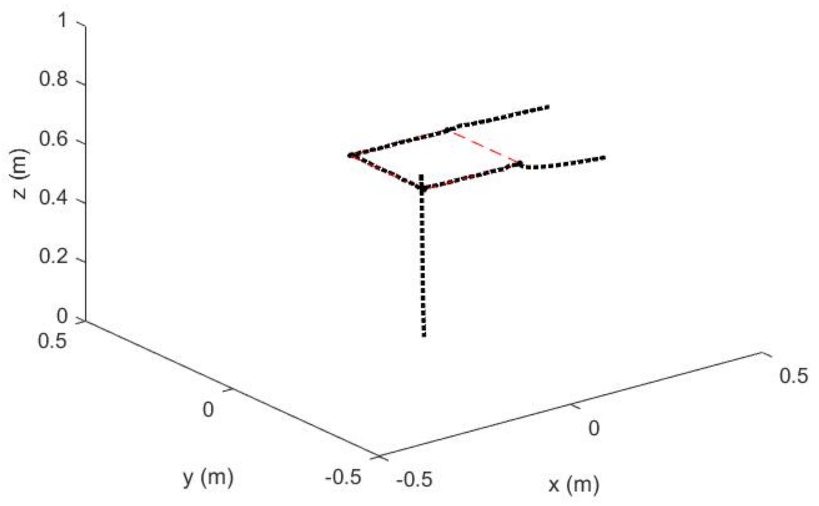

Figure 7: Trajectory With Fault, Parameter Learning

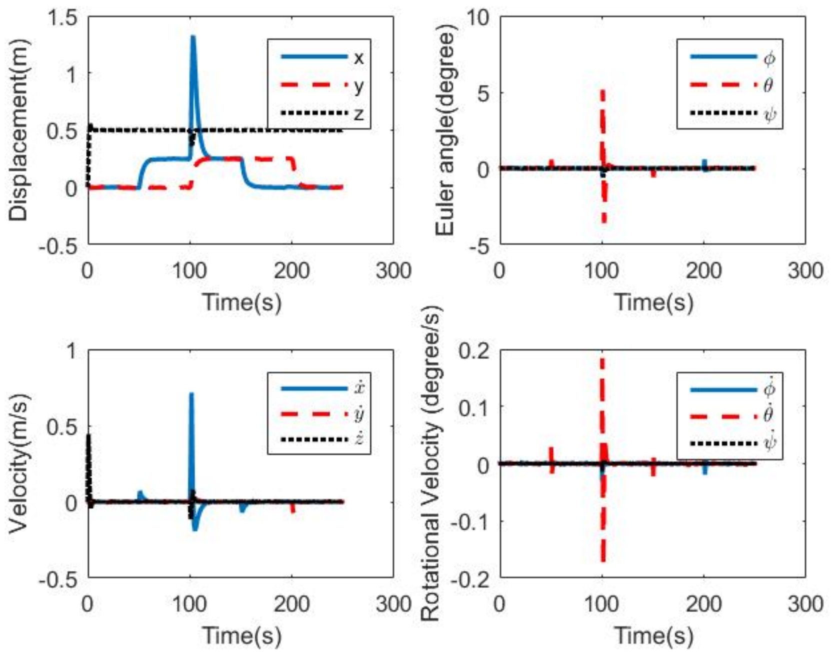

Figure 8: States With Fault, Parameter Learning

Finally, if we take a look at 3D tracking error, we can find out the parameter learning greatly reduce 3D tracking error under actuator fault. Also, the parameter learning can approach to real parameter fastly.

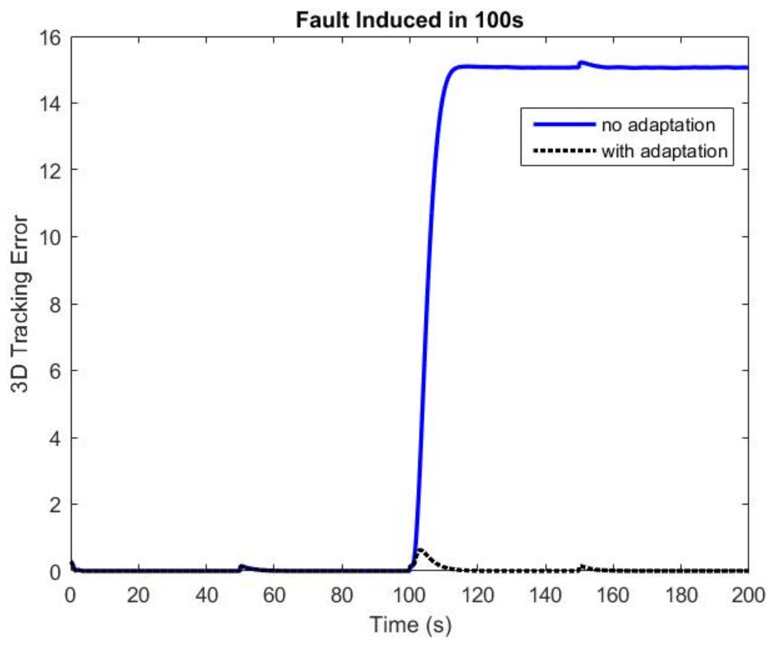

Figure 9: 3D tracking error comparison

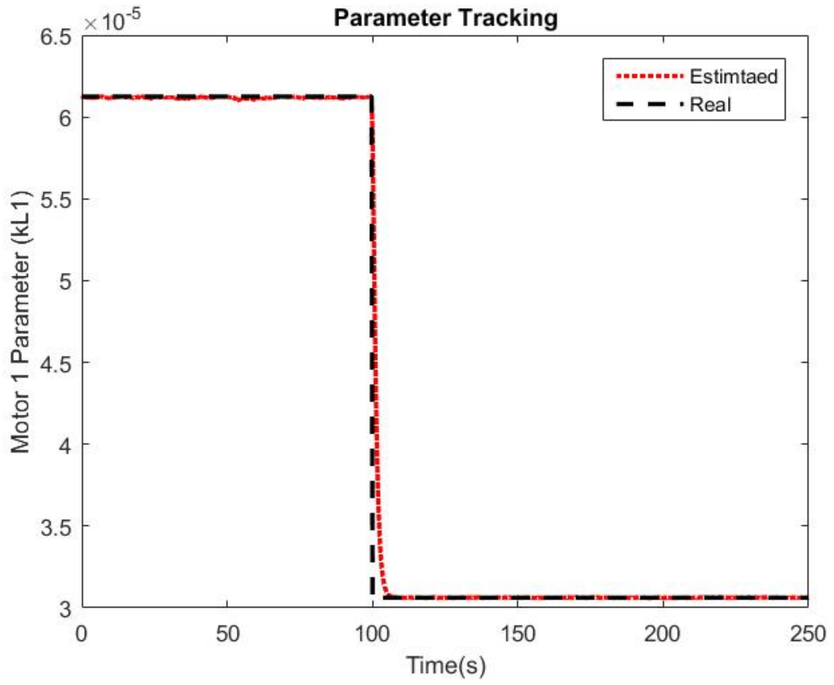

Figure 10: Comparing real parameter and estimated parameter for fault tolerant mechanism

Appendix

5.1 Code for Normalized Gradient Descent Algorithm

```
function [pnew] = projection(p, alpha, gamma, y, dataMtx)
global para constants
dataMtx = dataMtx';
M = alpha*eye(size(dataMtx,2)) + dataMtx'* dataMtx;

if (rcond(M)>10^-15)
    Minv = inv(M);
    dp = (gamma * dataMtx * Minv) * (y - dataMtx' * p);
else
    dp = 0;

end
pnew = p + dp;
% constraint pnew (actuator only losses thrust/moment)
for i = 1:numel(pnew)
    if (pnew(i) > constants.maxKL)
        pnew(i) = constants.maxKL;
    end
end

end
```

5.2 Code for Quadcopter Simulation

```
clc;
clear all;
close;

%% Define initial states
    % Define system states
    x0=0;   y0=0;   z0=0;   phi0=0;    theta0=0;    psi0=0;
    x_dot0=0;   y_dot0=0;   z_dot0=0;   phi_dot0=0; theta_dot0=0; psi_dot0=0;
    sNorm0=[x0,y0,z0,phi0,theta0,psi0,x_dot0,y_dot0,z_dot0,phi_dot0,theta_dot0,psi_dot0]';

    % Define simulation states
    s0 = sNorm0;
```

```
%% set system structure and control parameters

% for detecting parameter change
    global wt st rt gamma1 gamma2 r0 rstat kLMonitor
    kLMonitor = [];
    wt = zeros(4,1);
    rt = zeros(4,1);
    r0 = 1;
    gamma1 = 0.3;
    gamma2 = 0.3;

% define parameters
    global constants para
    constants.g = 9.8;
    para.m=1;
    para.Ixx=10^-1;
    para.Iyy=10^-1;
    para.Izz=10^-1;
    para.kL = para.m*constants.g/(4*200^2) * ones(1,4);
    para.LD_ratio = 15; % assume such ratio does not change

    constants.maxKL = para.m*constants.g/(4*200^2); % known

% PD controller parameter
    global ctlpara
    % pose angle control
    ctlpara.KP_phi=3;
    ctlpara.KP_phi_dot=0.9;

    ctlpara.KP_theta=3;
    ctlpara.KP_theta_dot=0.9;

    ctlpara.KP_psi=3;
    ctlpara.KP_psi_dot=0.9;

    % position control
    ctlpara.KP_x = 0.5;
    ctlpara.KP_xdot = 2;

    ctlpara.KP_y = 0.5;
    ctlpara.KP_ydot = 2;
    % altitude control
    ctlpara.KP_z = 3;
    ctlpara.KP_zdot = 2;


%% simulaiton
    h = 0.005;
    time=0:h:250;
    para.h = h;

    % generate process noise for sensor and disturbance
    para.noise_table = addSensorNoise(time,sNorm0);
    para.disturb_table = addDisturbance(time,sNorm0);

    % find desired trajectory
    ref_traj = zeros(numel(time), numel(sNorm0));
    for i = 1:numel(time)
        ref_traj(i,:) = trajectory(time(i))';
    end

    % use Euler method for solving system
    [t,sout]=Euler(@dyneqn,time,s0);

%% plotting
% plot different states
figure;
subplot(2,2,1);
    plot(t,(sout(:,1)),t,(sout(:,2)),'r--',t,(sout(:,3)),'k:', 'linewidth', 2);
    xlabel('Time(s)');
    ylabel('Displacement(m)');
    legend('x','y','z');

subplot(2,2,3);
    plot(t,(sout(:,7)),t,(sout(:,8)),'r--',t,(sout(:,9)),'k:', 'linewidth', 2);
    xlabel('Time(s)');
    ylabel('Velocity(m/s)');
    l1 = legend('$\dot{x}$','$\dot{y}$','$\dot{z}$');
    set(l1, 'interpreter', 'latex')

subplot(2,2,2);
    plot(t,(sout(:,4))/pi*180,t,(sout(:,5))/pi*180,'r--',t,(sout(:,6))/pi*180,'k:',
        'linewidth', 2);
    xlabel('Time(s)');
    ylabel('Euler angle(degree)');
    legend('\phi','\theta','\psi');

subplot(2,2,4);
    plot(t,(sout(:,10)),t,(sout(:,11)),'r--',t,(sout(:,12)),'k:', 'linewidth', 2);
    xlabel('Time(s)');
    ylabel('Rotational Velocity (degree/s)');
    l2 = legend('$\dot{\phi}$','$\dot{\theta}$','$\dot{\psi}$');
    set(l2, 'interpreter', 'latex')

% 3D trajectory
figure;
    plot3((sout(:,1)),(sout(:,2)),(sout(:,3)),'k:', 'linewidth', 2);
    hold on;
    plot3((ref_traj(:,1)),(ref_traj(:,2)),(ref_traj(:,3)),'r--', 'linewidth', 0.5);
    xlabel('x (m)');
    ylabel('y (m)');
    zlabel('z (m)');
    axis([-0.5 0.5 -0.5 0.5 0 1])

% tracking error
figure;
    distance = sout(:,1:3)-ref_traj(:,1:3);
    distance = sqrt(mean((distance.^2),2));
    plot(t,distance,'k:', 'linewidth', 2);
    xlabel('Time(s)');
    ylabel('3D tracking error (m)');
```

5.3 Code for Dynamic Equations and Controllers

```
% s is the state vector, 12 states in total
% state in following order:
% x,y,z,phi,theta,psi,xdot,ydot,zdot,phi_dot,theta_dot,psi_dot
function [sdot] = dyneqn(time, s)

    [fs, gs_thrust, gs_moment] = calThrustMoment(s);
    % assume only motor parameters will be changed
    cur_para = get_para(time, s);
    kL = cur_para.kL; kD = kL/cur_para.LD_ratio;
    % calculate inputs based on propeller rotational speeds omega
    % support there is a close-loop in omega
    if time==0
       U=[0;0;0;0];
    else
       U = controller(time, s);
    end

    u1=U(1); u2=U(2); u3=U(3); u4=U(4);
    %  denote u1 = o1^2, u2 = o2^2, ....
    FT = (kL(1)*u1 + kL(2)*u2 + kL(3)*u2 + kL(4)*u4);
    M1 = (kL(4)*u4 - kL(2)*u2);
    M2 = (-kL(1)*u1 + kL(3)*u3);
    M3 = (kD(1)*u1 + kD(3)*u3) - (kD(2)*u2 + kD(4)*u4);

    MT = [M1;M2;M3];

    %%
    sdot = zeros(size(s));
    sdot(1:6) = s(7:12);
    sdot(7:12) = fs + [gs_thrust*FT;gs_moment.*MT];

    %% add disturbance
    sdot = sdot + cur_para.disturb_table(:,round(time/cur_para.h+1));

    %% store last states
    global Uold sdot_old s_old
    Uold = U;
    sdot_old = sdot;
    s_old = s;
end

function [U] = controller(time, s)
    global ctlpara constants para
    % add sensor noise
    s = s + para.noise_table(:,round(time/para.h+1));

    x=s(1); y=s(2); z=s(3); phi=s(4); theta=s(5); psi=s(6);
    x_dot=s(7); y_dot=s(8); z_dot=s(9); phi_dot=s(10); theta_dot=s(11); psi_dot=s(12);

    % be careful here, when doing motion planning, need to transform global
    % rotational speed to body rotational speed
```

```
my_map = trajectory(time);

x_d = my_map(1); y_d = my_map(2); z_d = my_map(3);
phi_d = my_map(4); theta_d = my_map(5); psi_d = my_map(6);
x_dot_d = my_map(7); y_dot_d = my_map(8); z_dot_d = my_map(9);
phi_dot_d = my_map(10); theta_dot_d = my_map(11); psi_dot_d = my_map(12);

%% online parameter identification through projection
if(time>0.5)
    global sdot_old Uold s_old kLMonitor
    pmotor = para.kL';
    [fs, gs_thrust, gs_moment] = calThrustMoment(s_old);
    y_id = sdot_old([7,8,9,10,11,12]) - fs([1,2,3,4,5,6]);

    dataMtx = [gs_thrust(1)*Uold';
               gs_thrust(2)*Uold';
               gs_thrust(3)*Uold';
               -gs_moment(1)*[0 Uold(2) 0 -Uold(4)];...
               -gs_moment(2)*[Uold(1) 0 -Uold(3) 0];
               gs_moment(3)*[Uold(1) -Uold(2) Uold(3) -Uold(4)]/para.LD_ratio];
    para.kL = projection(pmotor, 2, 0.01, y_id, dataMtx)';
    kLMonitor = [kLMonitor;para.kL];
end

%% firstly, do an altitude control
uz = ctlpara.KP_z*(z_d - z) + ctlpara.KP_zdot*(z_dot_d-z_dot);
FT_des = (constants.g + uz)*para.m/(cos(phi)*cos(theta));

%% then we do a position control
% calculate desired ux and uy
ux = ctlpara.KP_x*(x_d - x) + ctlpara.KP_xdot*(x_dot_d-x_dot);
uy = ctlpara.KP_y*(y_d - y) + ctlpara.KP_ydot*(y_dot_d-y_dot);

% calculate required theta and phi
theta_d = atan((ux*cos(psi)+uy*sin(psi))/(constants.g + uz));
phi_d = atan((ux*sin(psi)-uy*cos(psi))/(constants.g + uz))*cos(theta);

% regularization, do not overturn the quadcopter
if (abs(theta_d)>(60/180*pi))
    theta_d = 15/180*pi*sign(theta_d);
end
```

```
    if (abs(phi_d)>(60/180*pi))
        phi_d = 15/180*pi*sign(phi_d);
    end

    M1_des = ctlpara.KP_phi*(phi_d-phi)+ctlpara.KP_phi_dot*(phi_dot_d-phi_dot);
    M2_des =
        ctlpara.KP_theta*(theta_d-theta)+ctlpara.KP_theta_dot*(theta_dot_d-theta_dot);
    M3_des = ctlpara.KP_psi*(psi_d-psi)+ctlpara.KP_psi_dot*(psi_dot_d-psi_dot);

    kL = para.kL; kD = kL/para.LD_ratio;
    transMtx = [kL;0,-kL(2),0,kL(4);-kL(1),0,kL(3),0;kD(1),-kD(2),kD(3),-kD(4)];
    U = ((transMtx \ [FT_des;M1_des;M2_des;M3_des]));


end

function [fs, gs_thrust, gs_moment] = calThrustMoment(s)
    global constants para
    cur_para = para;
    phi=s(4); theta=s(5); psi=s(6);
    phi_dot=s(10);
    theta_dot=s(11);
    psi_dot=s(12);

    Ixx = cur_para.Ixx; Iyy = cur_para.Iyy; Izz = cur_para.Izz; m=cur_para.m;

    fs = [0;0;-constants.g; ...
        (Ixx+Iyy-Izz)/Ixx*psi_dot*theta_dot; ...
        (-Ixx-Iyy+Izz)/Iyy*phi_dot*psi_dot;
        (Ixx-Iyy+Izz)/Izz*phi_dot*theta_dot];

    gs_thrust = [sin(phi)*sin(psi)+cos(phi)*sin(theta)*cos(psi);...
            -sin(phi)*cos(psi)+cos(phi)*sin(psi)*sin(theta);...
            cos(phi)*cos(theta)]/m;

    gs_moment = [1/Ixx; 1/Iyy; 1/Izz];
end
```

5.4 Code for Homemade Euler Method

```
% Euler method for solving stochastic system
```

```
function [t, s]=Euler(dyneqn,time,s0)
    s=zeros(numel(time),numel(s0));
    s(1,:)=s0';
    h = time(2)-time(1);
    for n=2:numel(time)
        s(n,:)=s(n-1,:) + (h*dyneqn(time(n),s(n-1,:)'))';
    end
    t = time;
end
```

5.5 Code for Getting System Parameters

```
% get current parameter (assign fault here)
function [cur_para] = get_para(time, s)
    global para constants
    cur_para = para;
    if time>100
        cur_para.kL(1) = constants.maxKL * (0.5 + exp(-(time-100))* 0.5);
    end
end
```

References

[1] Anastasia Razinkova, Igor Gaponov, and Hyun-Chan Cho. Adaptive control over quadcopter uav under disturbances. In *Control, Automation and Systems (ICCAS), 2014 14th International Conference on*, pages 386–390. IEEE, 2014.