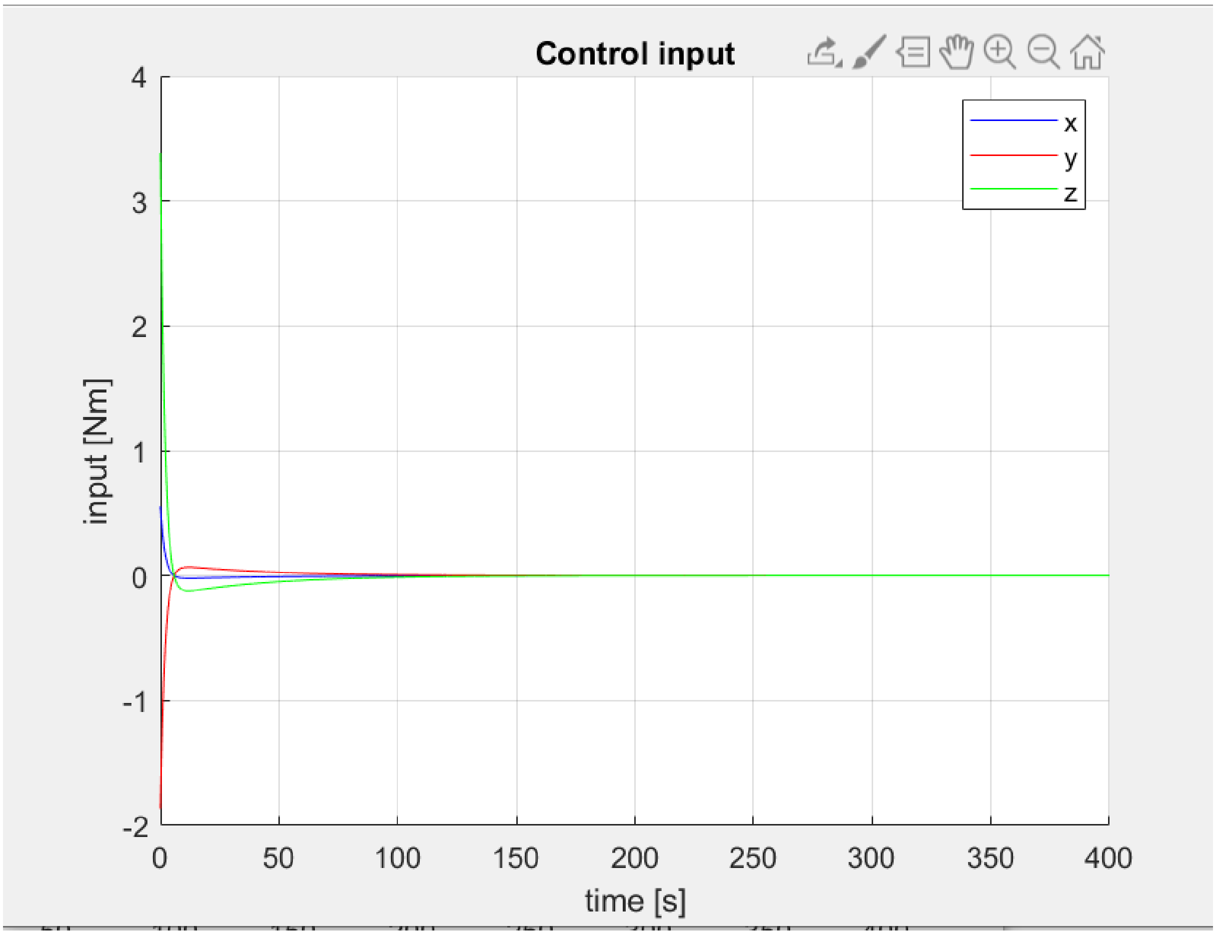
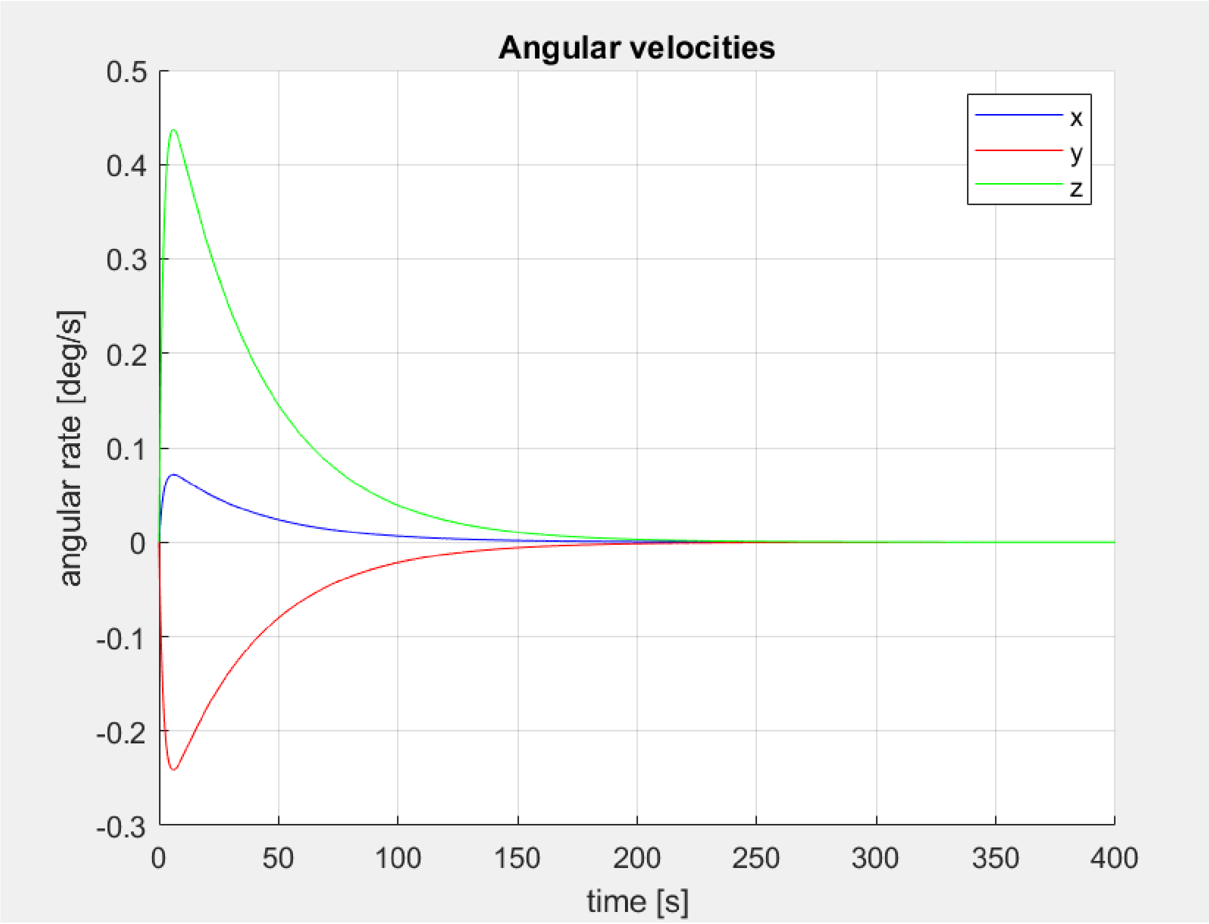
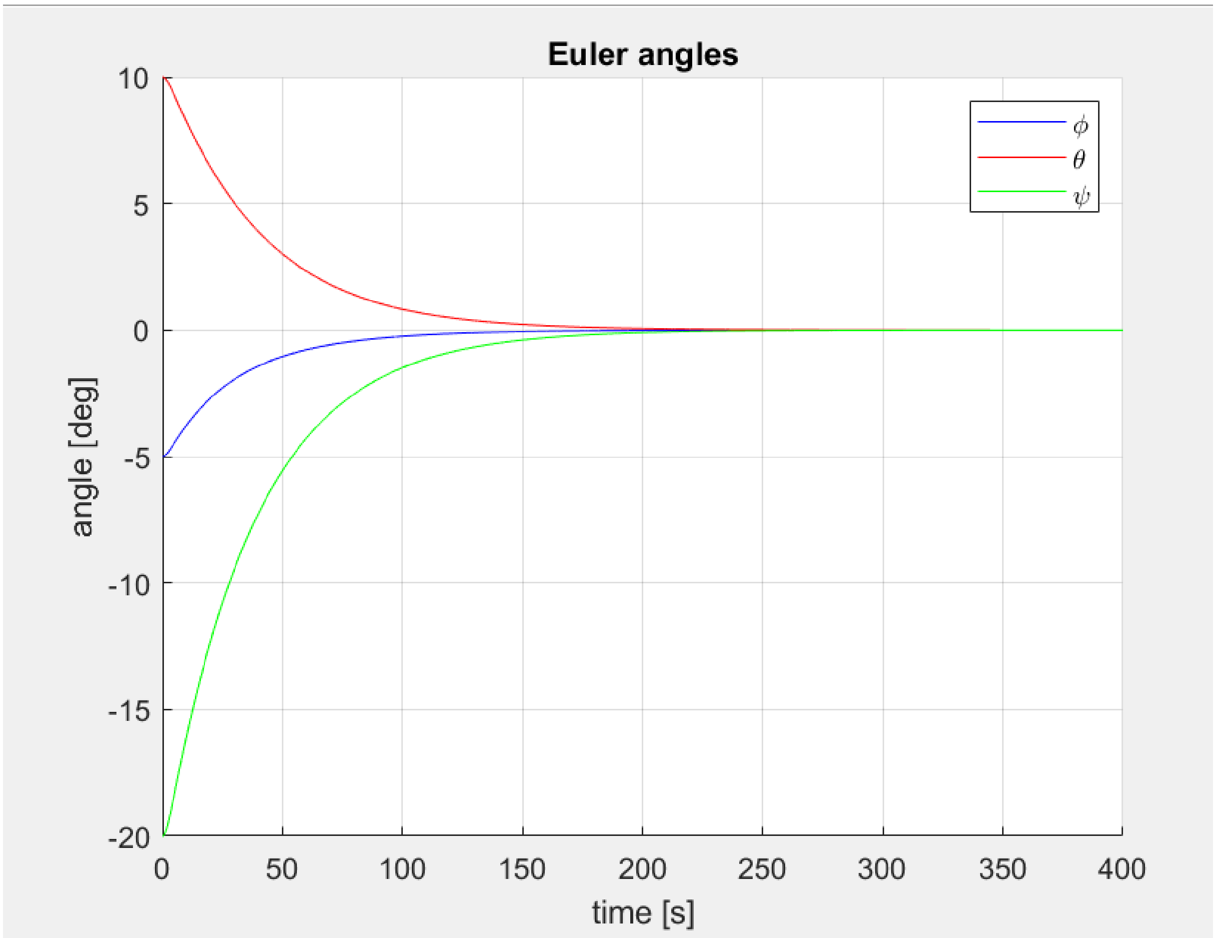
Task 1.3:



Control input
input [Nm]
time [s]
x
y
z
4
3
2
1
0
-1
-2
0
50
100
150
200
250
300
350
400



Angular velocities
angular rate [deg/s]
time [s]
x
y
z
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0
50
100
150
200
250
300
350
400



Code:

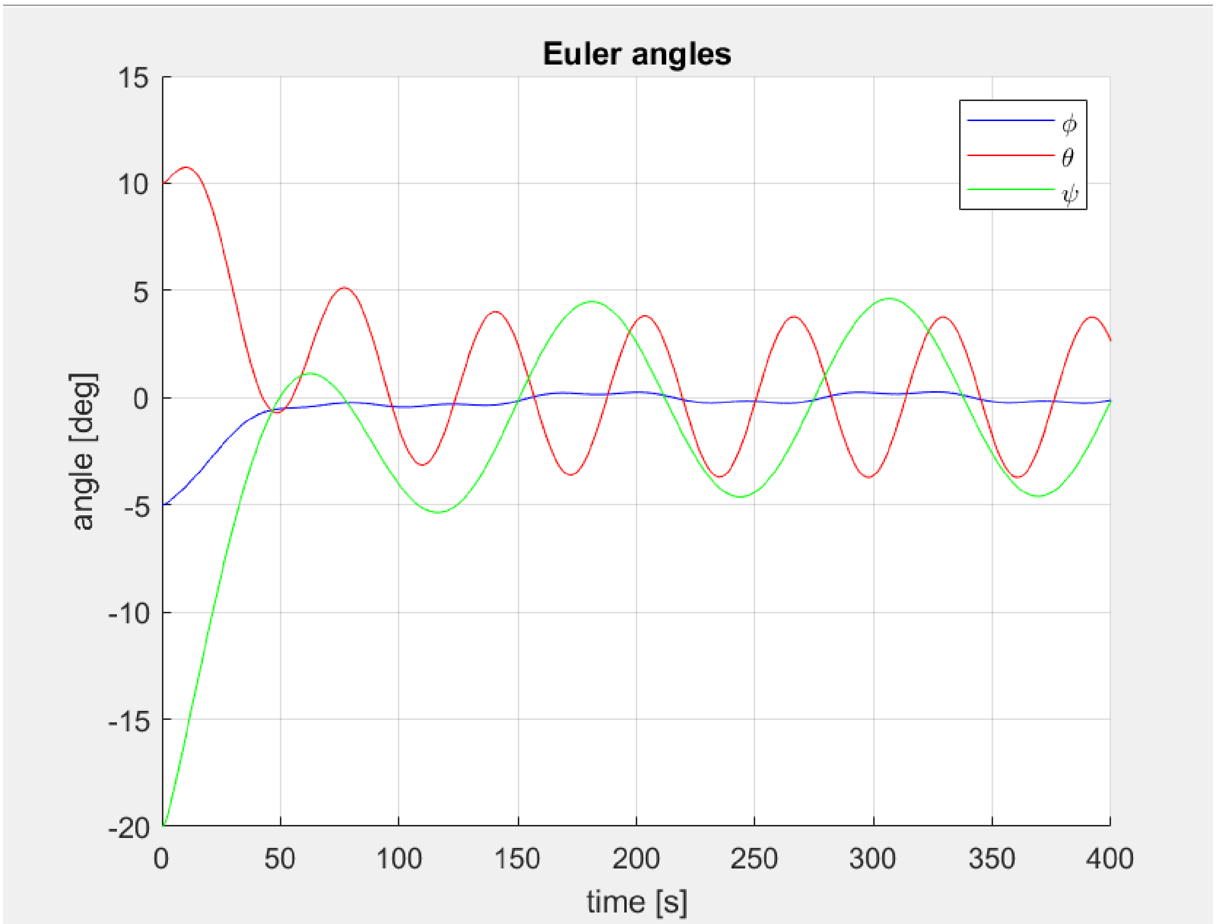
```
%% USER INPUTS
addpath(genpath("C:\Users\jacob\Documents\Student\Fartøy\MSS"));
h = 0.1;                              % sample time (s)
N   = 4000;                            % number of samples. Should be adjusted

% model parameters
m = 180;
r = 2;
I = m*r^2*eye(3);                     % inertia matrix
I_inv = inv(I);

% constants
deg2rad = pi/180;
rad2deg = 180/pi;

phi = -5*deg2rad;                     % initial Euler angles
theta = 10*deg2rad;
psi = -20*deg2rad;

q = euler2q(phi,theta,psi);    % transform initial Euler angles to q

w = [0 0 0]';                          % initial angular rates
```

```
table = zeros(N+1,14);          % memory allocation

%% FOR-END LOOP
for i = 1:N+1
   t = (i-1)*h;                    % time
   x = [q(2:4)' w']';
   Kp = -kp*eye(3);
   Kd = -kd*eye(3);

   K = [Kp, Kd];

   tau = K*x;             % control law

   [phi,theta,psi] = q2euler(q); % transform q to Euler angles
   [J,J1,J2] = quatern(q);        % kinematic transformation matrices

   q_dot = J2*w;                          % quaternion kinematics
   w_dot = I_inv*(Smtrx(I*w)*w + tau);    % rigid-body kinetics

   table(i,:) = [t q' phi theta psi w' tau'];  % store data in table

   q = q + h*q_dot;                % Euler integration
   w = w + h*w_dot;
```

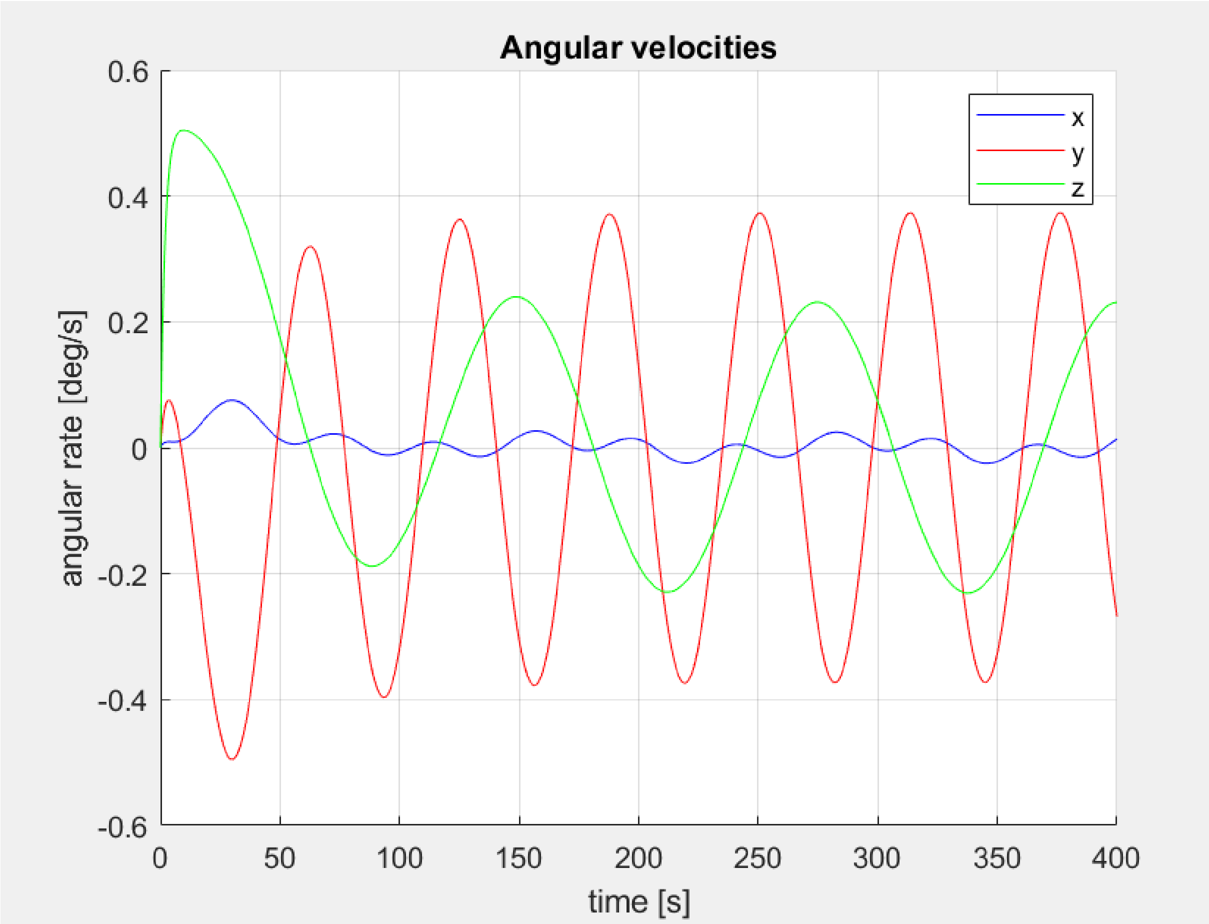
```
   q = q + h*q_dot;                % Euler integration
   w = w + h*w_dot;

   q  = q/norm(q);                 % unit quaternion normalization
end
```

The system-response matches our expectations of such a system with a 0-reference. For a non-zero constant referance we would change the feedback-terms in the control-law to be the difference between the reference-state and the current state.

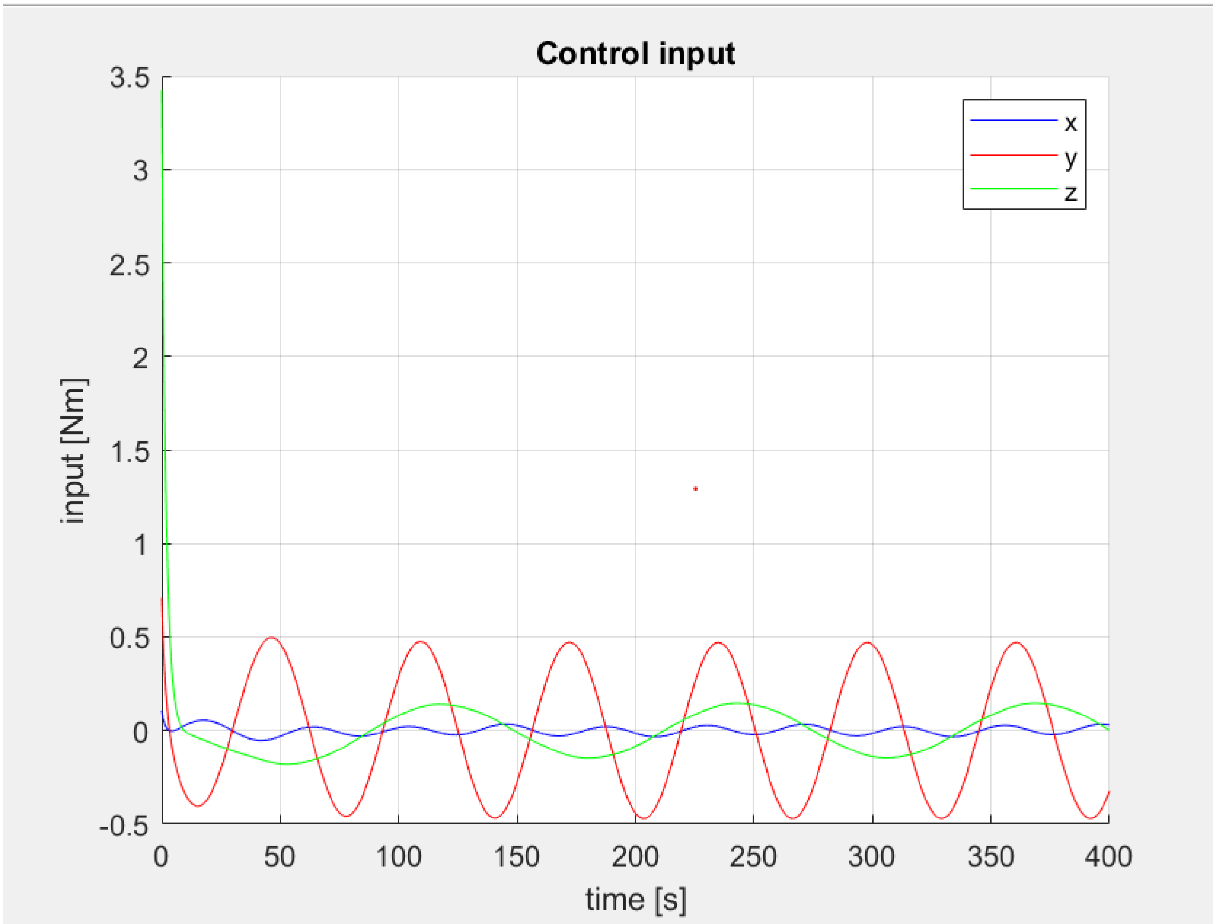
Task 1.5:

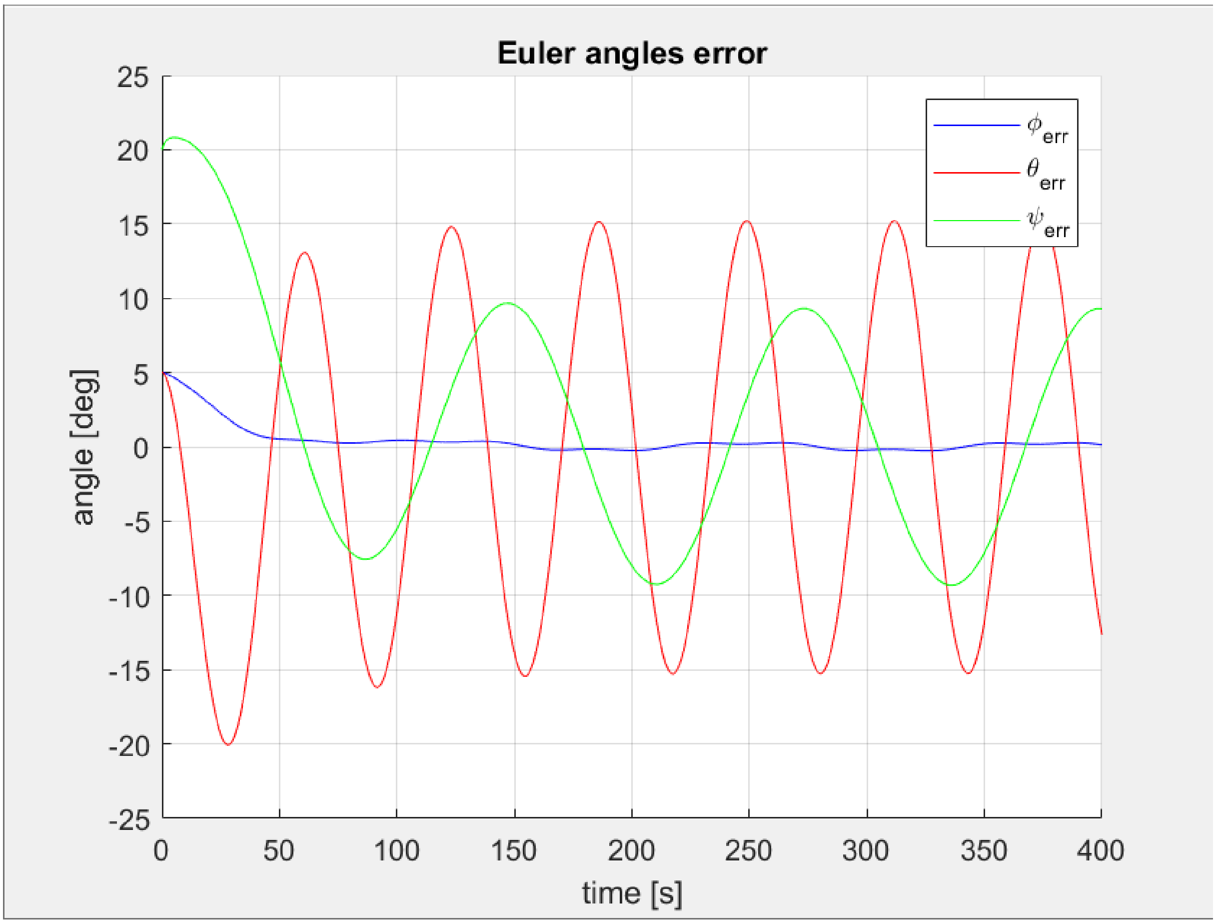


Euler angles
angle [deg]
time [s]
ϕ
θ
ψ
15
10
5
0
-5
-10
-15
-20
0
50
100
150
200
250
300
350
400



Angular velocities
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
angular rate [deg/s]
0
50
100
150
200
250
300
350
400
time [s]
x
y
z


Control input
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
input [Nm]
0
50
100
150
200
250
300
350
400
time [s]
x
y
z


Euler angles error
25
20
15
10
5
0
-5
-10
-15
-20
-25
angle [deg]
0
50
100
150
200
250
300
350
400
time [s]
ϕ_{err}
θ_{err}
ψ_{err}

Code:

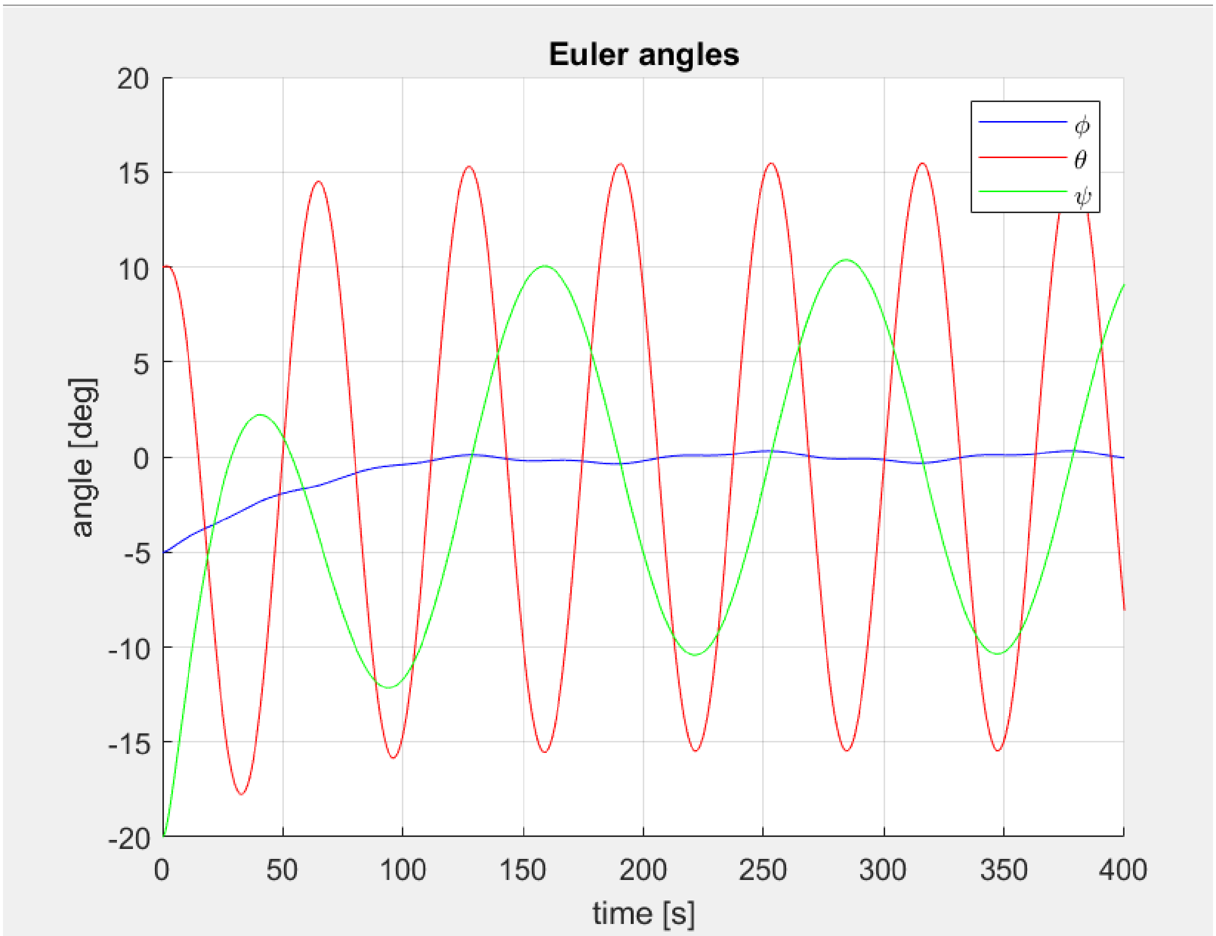
```matlab
%% USER INPUTS
addpath(genpath("C:\Users\jacob\Documents\Student\Fartøy\MSS"));
h = 0.1;                          % sample time (s)
N  = 4000;                         % number of samples. Should be adjusted

% model parameters
m = 180;
r = 2;
I = m*r^2*eye(3);                 % inertia matrix
I_inv = inv(I);

% constants
deg2rad = pi/180;
rad2deg = 180/pi;

phi = -5*deg2rad;                 % initial Euler angles
theta = 10*deg2rad;
psi = -20*deg2rad;

q = euler2q(phi,theta,psi);    % transform initial Euler angles to q

w = [0 0 0]';                      % initial angular rates
```

```matlab

table = zeros(N+1,14);           % memory allocation

kp = 20;                           %proportional gain
kd = 400;                          %derivative gain

table_d = zeros(N+1,3);



%% FOR-END LOOP
for i = 1:N+1
   t = (i-1)*h;                       % time

   phi_d = 0*deg2rad;                          %desired phi
   theta_d = 15*cos(0.1*t)*deg2rad;         %desired theta
   psi_d = 10*sin(0.05*t)*deg2rad;          %desired psi

   q_d = euler2q(phi_d,theta_d,psi_d); %desired quaternion

   inv_matr = eye(4)*-1;                  %matrix for inverting the quaternio
   inv_matr(1,1) = 1;

```

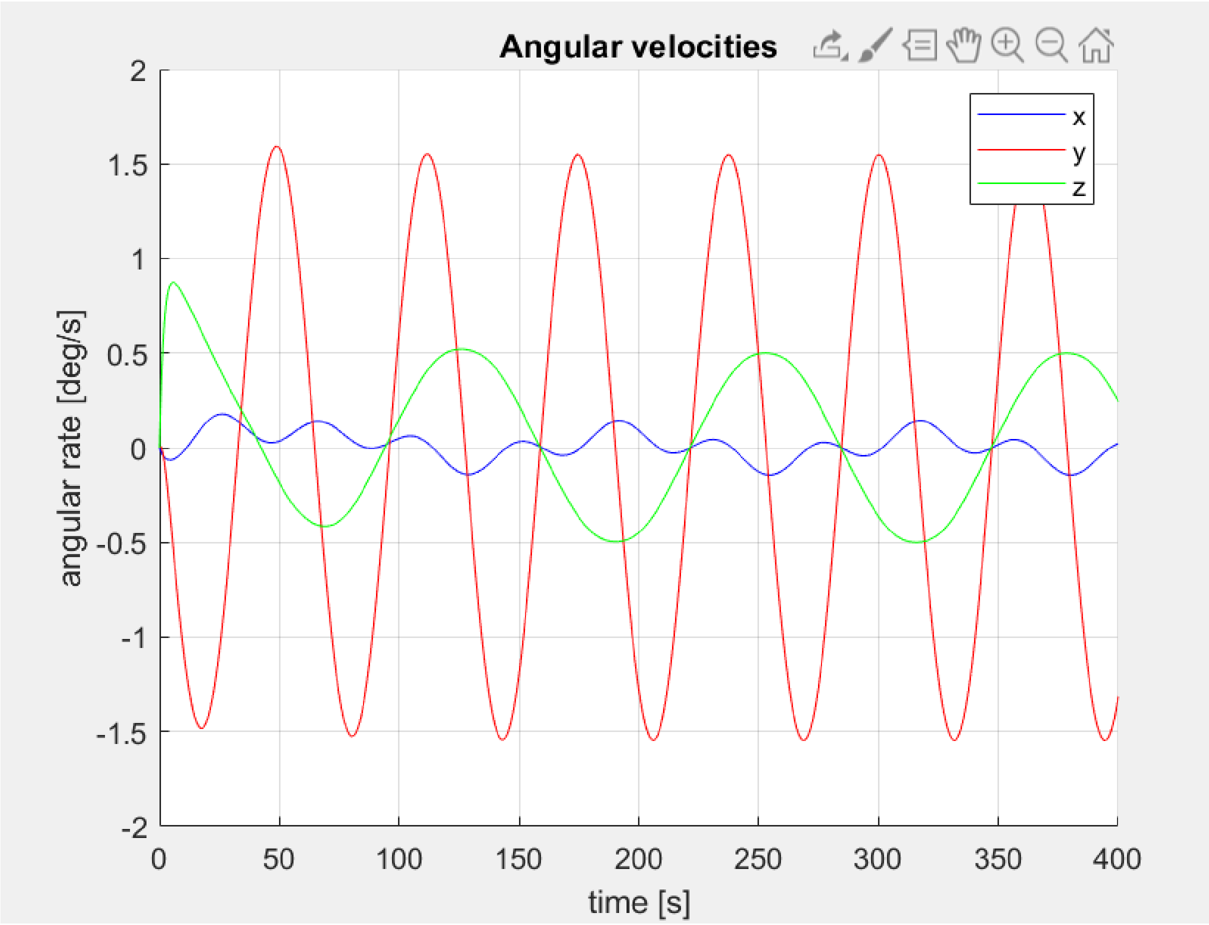
```
    q_d_inv = inv_matr*q_d;                  %inverting quaternion


    q_err = quatprod(q_d_inv,q);             %calculating the error-quaternion
    x = [q_err(2:4)' w']';
    Kp = -kp*eye(3);
    Kd = -kd*eye(3);

    K = [Kp, Kd];

    tau = K*x;             % control law

    [phi,theta,psi] = q2euler(q); % transform q to Euler angles
    [J,J1,J2] = quatern(q);       % kinematic transformation matrices

    q_dot = J2*w;                            % quaternion kinematics
    w_dot = I_inv*(Smtrx(I*w)*w + tau);     % rigid-body kinetics

    table(i,:) = [t q' phi theta psi w' tau'];  % store data in table
    table_d(i,:) = [phi_d,theta_d,psi_d];

    q = q + h*q_dot;                % Euler integration
    w = w + h*w_dot;


    q   = q/norm(q);                % unit quaternion normalization
end

%% PLOT FIGURES
t         = table(:,1);
q         = table(:,2:5);
phi       = rad2deg*table(:,6);
theta     = rad2deg*table(:,7);
psi       = rad2deg*table(:,8);
w         = rad2deg*table(:,9:11);
tau       = table(:,12:14);

phi_d = rad2deg*table_d(:,1);
theta_d = rad2deg*table_d(:,2);
psi_d = rad2deg*table_d(:,3);

phi_err = phi_d - phi;
theta_err = theta_d - theta;
psi_err = psi_d - psi;
```

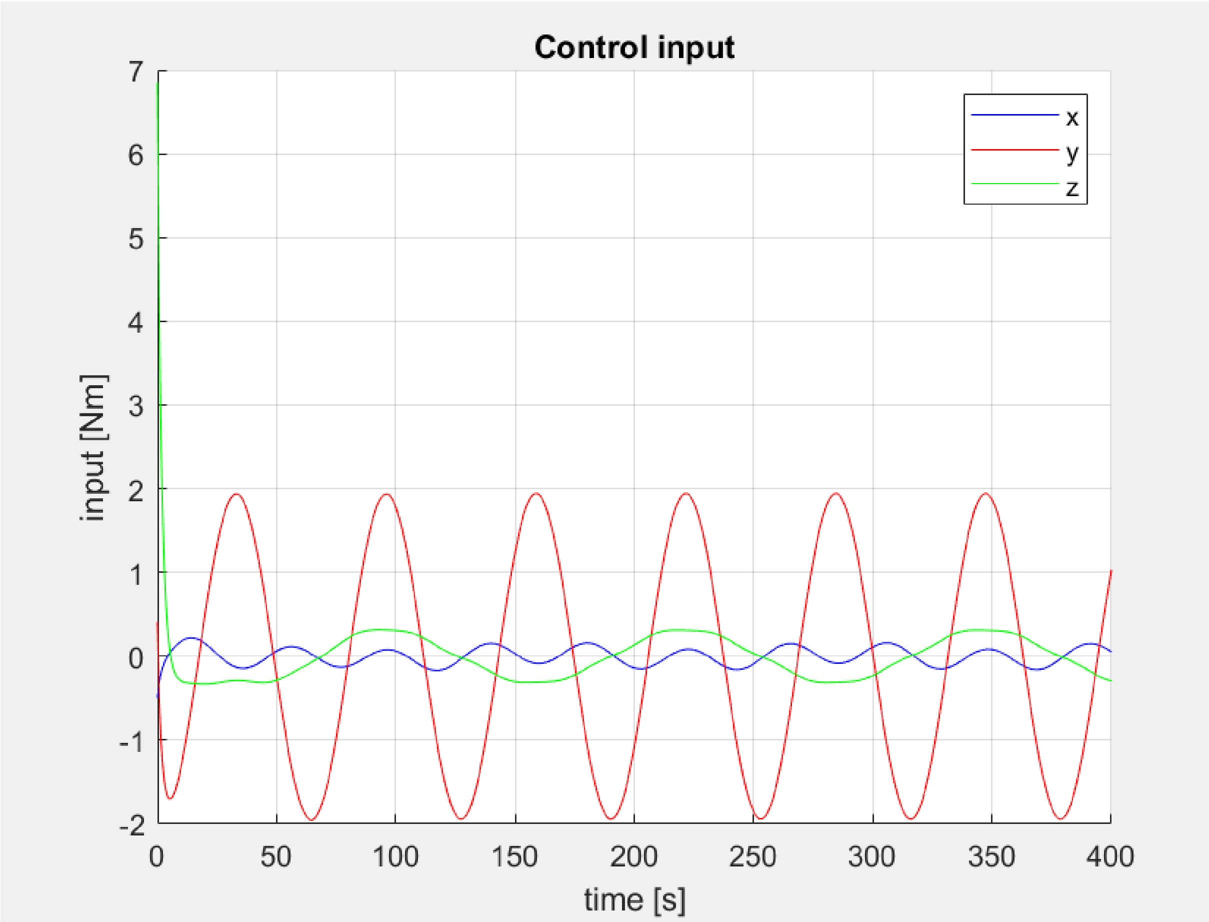
The system response is expected. We have no error-state in the omega so the error is quite large. We see the control input also get a sinusoidal shape which makes sense as we are tracking a sinusoidal reference. Since we switched to quaternions we do not get the issue with a singularity at 90 degrees that we would have had from euler angles, but the simulation shows that no state reaches 90 degrees anyways.

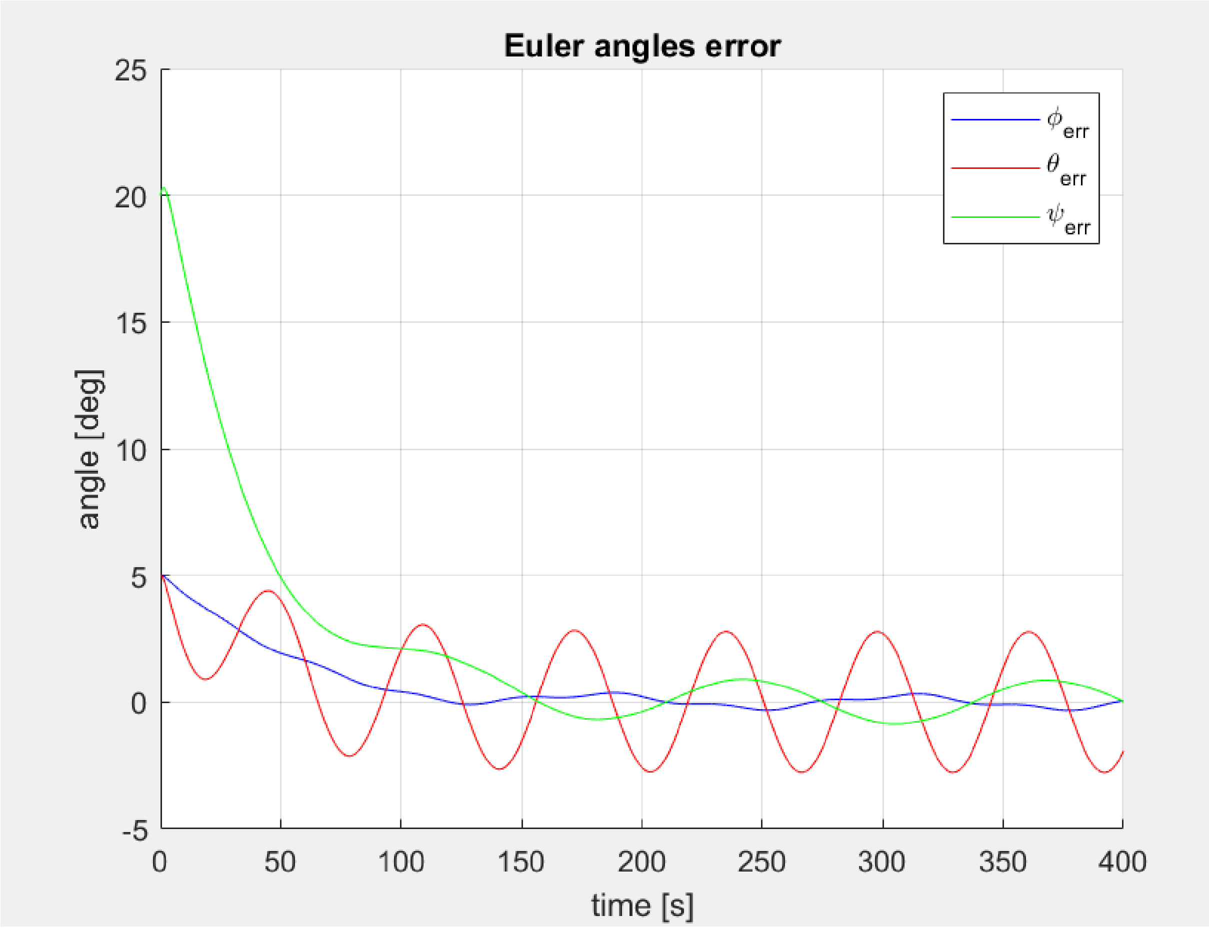
Task 1.6:



Euler angles
angle [deg]
time [s]
20
15
10
5
0
-5
-10
-15
-20
0
50
100
150
200
250
300
350
400
ϕ
θ
ψ


Angular velocities
angular rate [deg/s]
time [s]
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
50
100
150
200
250
300
350
400
x
y
z


Control input
input [Nm]
time [s]
7
6
5
4
3
2
1
0
-1
-2
0
50
100
150
200
250
300
350
400
x
y
z



Code:

```
%% USER INPUTS
addpath(genpath("C:\Users\jacob\Documents\Student\Fartøy\MSS"));
h = 0.1;                        % sample time (s)
N  = 4000;                       % number of samples. Should be adjusted

% model parameters
m = 180;
r = 2;
I = m*r^2*eye(3);               % inertia matrix
I_inv = inv(I);

% constants
deg2rad = pi/180;
rad2deg = 180/pi;

phi = -5*deg2rad;               % initial Euler angles
theta = 10*deg2rad;
psi = -20*deg2rad;

q = euler2q(phi,theta,psi);      % transform initial Euler angles to q

w = [0 0 0]';                    % initial angular rates

```

```
table = zeros(N+1,14);          % memory allocation

kp = 20;                         %proportional gain
kd = 400;                        %derivative gain







%% FOR-END LOOP
for i = 1:N+1
   t = (i-1)*h;                    % time

   phi_d = 0*deg2rad;                           %desired phi
   theta_d = 15*cos(0.1*t)*deg2rad;            %desired theta
   psi_d = 10*sin(0.05*t)*deg2rad;              %desired psi

   phi_d_dot = 0*deg2rad;                       %desired phi_dot
   theta_d_dot = -1.5*sin(0.1*t)*deg2rad;      %desired theta_dot
   psi_d_dot = 0.5*cos(0.05*t)*deg2rad;        %desired psi_dot


   Theta_D = [phi_d_dot, theta_d_dot, psi_d_dot]';

   q_d = euler2q(phi_d,theta_d,psi_d); %desired quaternion

   inv_matr = eye(4)*-1;                %matrix for inverting the quaternio
   inv_matr(1,1) = 1;

   q_d_inv = inv_matr*q_d;              %inverting quaternion

   T_inv = [1, 0, -sin(theta);             %T_inv from 2.41 in Fossen
    0, cos(phi), cos(theta)*sin(phi);
    0, -sin(phi), cos(theta)*cos(phi)];


    w_d = T_inv * Theta_D;                   %desired omega

    w_err = w-w_d;                           %error in omega


   q_err = quatprod(q_d_inv,q);         %calculating the error-quaternion
   x = [q_err(2:4)' w_err']';
   Kp = -kp*eye(3);
   Kd = -kd*eye(3);
```

```
    K = [Kp, Kd];

    tau = K*x;           % control law

    [phi,theta,psi] = q2euler(q); % transform q to Euler angles
    [J,J1,J2] = quatern(q);       % kinematic transformation matrices

    q_dot = J2*w;                         % quaternion kinematics
    w_dot = I_inv*(Smtrx(I*w)*w + tau);   % rigid-body kinetics

    table(i,:) = [t q' phi theta psi w' tau'];  % store data in table

    q = q + h*q_dot;              % Euler integration
    w = w + h*w_dot;

    q  = q/norm(q);               % unit quaternion normalization
end
```

We see that the tracking-error is much lower now that we included the error-term in the omega. Adding integral action is a bad idea with a time-variant reference because of windup. With some anti-windup law it could perhaps improve the system, but as the system is already really slow, more delay with an integral worsens the system.

Since the reference is time-varying we might consider adding gain-scheduling into the control-law to make the response faster and to further reduce the tracking-error.