Question 2)
The changes all make sense in the simulation. Change in yaw angle or yaw rate (r) keep the quadcopter in a steady hover since they do not change the direction of any forces on the aircraft. There is translational movement and loss of altitude when bank or pitch are altered due to the thrust being pointed in a direction other than opposite the force of gravity, altering the rotation rates of bank and pitch (p and q) cause the same changes, and the inertial velocities are non-linear which is expected.

Question 3)
The only significant difference between the non-linearized model and the linearized model is in the inertial Z velocity. The linearized model for Z velocity is dependent only on Z forces, being the control forces from the thrusters. Since there were no changes to the Z control forces, the linearized model calculated an incorrect inertial Z velocity.

Question 4)
The control laws change the control forces to stop the rotation rates of the copter in each axis. These control forces do not instantaneously change the rotation of the copter but do return the copter to equilibrium very quickly. An interesting note is that when p and q are altered, the copter does not return to steady hover but back to its initial condition with the deviation in its respective parameter. This means that the equilibrium that the control forces bring the copter back to is a steady translation in the direction the copter was rotated in. Ideally the control law would have a reference p, q, and r so there can be a control force applied to move the copter back to a steady hover where p=q=r=0.

Question 5)
The control law doesn't help the craft because it doesn't control the translation of the copter. This means that the copter cannot go back to trimmed hovering flight.

Δx | $x = \eta \underbrace{(u^{E^2} + v^{E^2} + w^2)^{1/2}}_{|\vec{v}^E|} u^E$ $\quad u_0^E = v_0^E = w_0^E = 0$

$$\frac{\partial x}{\partial u^E} = \eta |\vec{v}^E| + \frac{\eta u^{E^2}}{|\vec{v}^E|} \qquad \frac{\partial x}{\partial v^E} = \eta \frac{u^E v^E}{|\vec{v}^E|}$$

$$\frac{\partial x}{\partial w^E} = \eta \frac{u^E w^E}{|\vec{v}^E|} \qquad |\vec{v}_0^E| = \sqrt{u_0^{E^2} + v_0^{E^2} + w_0^{E^2}} = 0$$

$$\Delta x = \frac{\partial x}{\partial u^E}\Big|_0 \Delta u^E + \frac{\partial x}{\partial v^E}\Big|_0 \Delta v^E + \frac{\partial x}{\partial w^E}\Big|_0 \Delta w^E$$

$$= \left(\eta |\vec{v}_0^E| + \frac{\eta u_0^{E^2}}{|\vec{v}_0^E|}\right) \Delta u^E + \left(\eta \frac{u_0^E v_0^E}{|\vec{v}_0^E|}\right) \Delta v^E +$$

$$\left(\eta \frac{u_0^E w_0^E}{|\vec{v}_0^E|}\right) \Delta w^E$$

$$\boxed{\Delta x = 0}$$

Δy | $y = \eta (u^{E^2} + v^{E^2} + w^{E^2})^{1/2} v^E$

$$\frac{\partial y}{\partial u^E} = \eta \frac{u^E v^E}{|\vec{v}^E|} \qquad \frac{\partial y}{\partial v^E} = \eta |\vec{v}^E| + \eta \frac{v^{E^2}}{|\vec{v}^E|}$$

$$\frac{\partial y}{\partial w^E} = \eta \frac{v^E w^E}{|\vec{v}^E|}$$

$$\Delta y = \frac{\partial y}{\partial u^E}\Big|_0 \Delta u^E + \frac{\partial y}{\partial v^E}\Big|_0 \Delta v^E + \frac{\partial y}{\partial w^E}\Big|_0 \Delta w^E$$

$$= \left(\eta \frac{u_0^E v_0^E}{|\vec{v}_0^E|}\right) \Delta u^E + \left(\eta |\vec{v}_0^E| + \eta \frac{v_0^{E^2}}{|\vec{v}_0^E|}\right) \Delta v^E + \left(\eta \frac{v_0^E w_0^E}{|\vec{v}_0^E|}\right) \Delta w^E$$

$$\boxed{\Delta y = 0}$$

Δz

$$z = \eta\left(u^{E^2} + v^{E^2} + w^{E^2}\right)^{1/2} w^E + z_c$$

$$\frac{\partial z}{\partial u^E} = \eta \frac{u^E w^E}{|\vec{v}^E|} \qquad \frac{\partial z}{\partial v^E} = \eta \frac{v^E w^E}{|\vec{v}^E|} \qquad \frac{\partial z}{\partial w^E} = \eta |\vec{v}^E| + \frac{w^{E^2}}{|\vec{v}^E|}$$

$$\frac{\partial z}{\partial z_c} = 1$$

$$\Delta z = \frac{\partial z}{\partial u^E}\Big|_0 \Delta u^E + \frac{\partial z}{\partial v^E}\Big|_0 \Delta v^E + \frac{\partial z}{\partial w^E}\Big|_0 \Delta w^E + \frac{\partial z}{\partial z_c}\Big|_0 \Delta z_c$$

$$= \left(\eta \frac{u_0^E w_0^E}{|\vec{v}_0^E|}\right) \Delta u^E + \left(\eta \frac{v_0^E w_0^E}{|\vec{v}_0^E|}\right) \Delta v^E + \left(\eta |\vec{v}_0^E| + \eta \frac{w_0^{E^2}}{|\vec{v}_0^E|}\right) \Delta w^E$$

$$+ (1)\Delta z_c$$

$$\Delta z = \Delta z_c = \Delta F_1 + \Delta F_2 + \Delta F_3 + \Delta F_4$$

$p_0 = q_0 = r_0 = \psi_0 = \theta_0 = \phi_0 = u_0^E = v_0^E = w_0^E = x_0 = y_0 = z_0 = 0$

$\Delta\dot{p}$ $\quad \dot{p} = \dfrac{L}{I_x} - \dfrac{qr(I_z - I_y)}{I_x}$

$\dfrac{\partial \dot{p}}{\partial L} = \dfrac{1}{I_x} \qquad \dfrac{\partial \dot{p}}{\partial q} = -\dfrac{r(I_z - I_y)}{I_x} \qquad \dfrac{\partial \dot{p}}{\partial r} = -\dfrac{q(I_z - I_y)}{I_x}$

$\Delta\dot{p} = \dfrac{\partial \dot{p}}{\partial L}\Big|_0 \Delta L + \dfrac{\partial \dot{p}}{\partial q}\Big|_0 \Delta q + \dfrac{\partial \dot{p}}{\partial r}\Big|_0 \Delta r$

$= \left(\dfrac{1}{I_x}\right)\Delta L + \left(-\dfrac{\cancel{r_0}^{\,0}(I_z - I_y)}{I_x}\right)\Delta q + \left(-\dfrac{\cancel{q_0}^{\,0}(I_z - I_y)}{I_x}\right)\Delta r$

$\Delta\dot{p} = \dfrac{\Delta L}{I_x} \qquad \Delta L = \Delta L_C + \Delta L_A$

$$\boxed{\Delta\dot{p} = \frac{\Delta L_C + \Delta L_A}{I_x}}$$

$\Delta\dot{q}$ $\quad \dot{q} = \dfrac{M}{I_y} - \dfrac{rp(I_x - I_z)}{I_y}$

$\dfrac{\partial \dot{q}}{\partial M} = \dfrac{1}{I_y} \qquad \dfrac{\partial \dot{q}}{\partial r} = -\dfrac{p(I_x - I_z)}{I_y} \qquad \dfrac{\partial \dot{q}}{\partial p} = -\dfrac{r(I_y - I_z)}{I_y}$

$\Delta\dot{q} = \dfrac{\partial \dot{q}}{\partial M}\Big|_0 \Delta M + \dfrac{\partial \dot{q}}{\partial r}\Big|_0 \Delta r + \dfrac{\partial \dot{q}}{\partial p}\Big|_0 \Delta p$

$= \left(\dfrac{1}{I_y}\right)\Delta M + \left(-\dfrac{\cancel{p_0}^{\,0}(I_x - I_z)}{I_y}\right) + \left(-\dfrac{\cancel{r_0}^{\,0}(I_x - I_z)}{I_y}\right)$

$\Delta\dot{q} = \dfrac{\Delta M}{I_y} \qquad \Delta M = \Delta M_C + \Delta M_A$

$$\boxed{\Delta\dot{q} = \frac{\Delta M}{I_y} = \frac{\Delta M_C + \Delta M_A}{I_y}}$$

$\Delta \dot{r}$ $\quad \dot{r} = \frac{N}{I_z} - \frac{pq(I_y - I_x)}{I_z}$

$\frac{\partial \dot{r}}{\partial N} = \frac{1}{I_z} \quad \frac{\partial \dot{r}}{\partial p} = -\frac{q(I_y - I_x)}{I_z} \quad \frac{\partial \dot{r}}{\partial q} = -\frac{p(I_y - I_x)}{I_z}$

$\Delta \dot{r} = \frac{\partial \dot{r}}{\partial N}\Big|_0 \Delta N + \frac{\partial \dot{r}}{\partial p}\Big|_0 \Delta p + \frac{\partial \dot{r}}{\partial q}\Big|_0 \Delta q$

$= \left(\frac{1}{I_z}\right)\Delta N + \left(-\frac{q_0(I_y - I_x)}{I_z}\right)\Delta p + \left(-\frac{p_0(I_y - I_x)}{I_z}\right)\Delta q$

$\Delta \dot{r} = \frac{\Delta N}{I_z} \quad \Delta N = \Delta N_c + \Delta N_A$

$$\boxed{\Delta \dot{r} = \frac{\Delta N}{I_z} = \frac{\Delta N_c + \Delta N_A}{I_z}}$$

$\Delta \dot{\phi}$ $\quad \dot{\phi} = p + \tan(\theta)(q\sin(\phi) + r\cos(\phi))$

$\frac{\partial \dot{\phi}}{\partial p} = 1 \quad \frac{\partial \dot{\phi}}{\partial \theta} = \sec^2(\theta)(q\sin\phi + r\cos\phi)$

$\frac{\partial \dot{\phi}}{\partial q} = \tan(\theta)\sin(\phi) \quad \frac{\partial \dot{\phi}}{\partial r} = \tan(\theta)\cos(\phi)$

$\frac{\partial \dot{\phi}}{\partial \phi} = \tan(\theta)(q\cos\phi - r\sin\phi)$

$\Delta \dot{\phi} = \frac{\partial \dot{\phi}}{\partial p}\Big|_0 \Delta p + \frac{\partial \dot{\phi}}{\partial \theta}\Big|_0 \Delta\theta + \frac{\partial \dot{\phi}}{\partial q}\Big|_0 \Delta q + \frac{\partial \dot{\phi}}{\partial r}\Big|_0 \Delta r + \frac{\partial \dot{\phi}}{\partial \phi}\Big|_0 \Delta\phi$

$= (1)\Delta p + \left[\sec^2(\theta_0)(q_0\sin(\phi_0) + r_0\cos(\phi_0))\right]\Delta\theta +$
$\left[\tan(\theta_0)\sin(\phi_0)\right]\Delta q + \left[\tan(\theta_0)\cos(\phi_0)\right]\Delta r +$
$\left[\tan(\theta_0)(q_0\cos\phi - r_0\sin\phi_0\right]\Delta\phi$

$$\boxed{\Delta \dot{\phi} = \Delta p}$$

$\Delta\dot{\theta}$ $\quad \dot{\theta} = q\cos\phi - r\sin\phi$

$\frac{\partial\dot{\theta}}{\partial q} = \cos\phi \quad \frac{\partial\dot{\theta}}{\partial r} = -\sin\phi \quad \frac{\partial\dot{\theta}}{\partial\phi} = -q\sin\phi - r\cos\phi$

$\Delta\dot{\theta} = \frac{\partial\dot{\theta}}{\partial q}\Big|_0 \Delta q + \frac{\partial\dot{\theta}}{\partial r}\Big|_0 \Delta r + \frac{\partial\dot{\theta}}{\partial\phi}\Big|_0 \Delta\phi$

$= (\cos\phi_0)\Delta q + (-\sin\phi_0)\Delta r + (-q_0\sin\phi_0 - r_0\cos\phi_0)\Delta\phi$

$$\boxed{\Delta\dot{\theta} = \Delta q}$$

$\Delta\dot{u}^E$ $\quad \dot{u}^E = \frac{X}{m} - g\sin\theta - qw^E + rv^E$

$\frac{\partial\dot{u}^E}{\partial X} = \frac{1}{m} \quad \frac{\partial\dot{u}^E}{\partial\theta} = -g\cos\theta \quad \frac{\partial\dot{u}^E}{\partial q} = -w^E$

$\frac{\partial\dot{u}^E}{\partial w^E} = -q \quad \frac{\partial\dot{u}^E}{\partial r} = v^E \quad \frac{\partial\dot{u}^E}{\partial v^E} = r$

$\Delta\dot{u}^E = \frac{\partial\dot{u}^E}{\partial x}\Big|_0 \Delta x + \frac{\partial\dot{u}^E}{\partial\theta}\Big|_0 \Delta\theta + \frac{\partial\dot{u}^E}{\partial q}\Big|_0 \Delta q + \frac{\partial\dot{u}^E}{\partial w^E}\Big|_0 \Delta w^E +$

$\frac{\partial\dot{u}^E}{\partial r}\Big|_0 \Delta r + \frac{\partial\dot{u}^E}{\partial v^E}\Big|_0 \Delta v^E$

$= \left(\frac{1}{m}\right)\Delta x + (-g\cos\theta_0)\Delta\theta + (-w_0^E)\Delta q + (-q_0)\Delta w^E +$

$(v_0^E)\Delta r + (r_0)\Delta v^E$

$$\boxed{\Delta\dot{u}^E = -g\Delta\theta}$$

$\Delta \dot{v}^E$ | $\dot{v}^E = \frac{Y}{m} + g \sin\phi \cos\theta - p w^E + r u^E$

$\frac{\partial \dot{v}^E}{\partial Y} = \frac{1}{m}$ $\frac{\partial \dot{v}^E}{\partial \phi} = g \cos\phi \cos\theta$ $\frac{\partial \dot{v}^E}{\partial \theta} = -g \sin\phi \sin\theta$

$\frac{\partial \dot{v}^E}{\partial p} = -w^E$ $\frac{\partial \dot{v}^E}{\partial w^E} = -p$ $\frac{\partial \dot{v}^E}{\partial r} = u^E$ $\frac{\partial \dot{v}^E}{\partial u^E} = r$

$$\Delta \dot{v}^E = \frac{\partial \dot{v}^E}{\partial Y}\Big|_0 \Delta Y + \frac{\partial \dot{v}^E}{\partial \phi}\Big|_0 \Delta\phi + \frac{\partial \dot{v}^E}{\partial \theta}\Big|_0 \Delta\theta + \frac{\partial \dot{v}^E}{\partial r}\Big|_0 \Delta p + \frac{\partial \dot{v}^E}{\partial w^E}\Big|_0 \Delta w^E + \frac{\partial \dot{v}^E}{\partial r}\Big|_0 \Delta r + \frac{\partial \dot{v}^E}{\partial u^E}\Big|_0 \Delta u^E$$

$$= \left(\frac{1}{m}\right)\Delta Y + (g \cos\phi_0 \cos\theta_0)\Delta\phi + (-g \sin\phi_0 \sin\theta_0)\Delta\theta + (-w_0^E)\Delta p + (-p_0)\Delta w^E + (u_0^E)\Delta r + (r_0)\Delta u^E$$

$$\boxed{\Delta \dot{v}^E = g \Delta\phi}$$

$\Delta \dot{w}^E$ | $\dot{w}^E = \frac{Z}{m} + g \cos\phi \cos\theta - p v^E + q u^E$

$\frac{\partial \dot{w}^E}{\partial Z} = \frac{1}{m}$ $\frac{\partial \dot{w}^E}{\partial \phi} = -g \sin\phi \cos\theta$ $\frac{\partial \dot{w}^E}{\partial \theta} = -g \cos\phi \sin\theta$

$\frac{\partial \dot{w}^E}{\partial p} = -v^E$ $\frac{\partial \dot{w}^E}{\partial v^E} = -p$ $\frac{\partial \dot{w}^E}{\partial q} = u^E$ $\frac{\partial \dot{w}^E}{\partial u^E} = q$

$$\Delta \dot{w}^E = \frac{\partial \dot{w}^E}{\partial Z}\Big|_0 \Delta Z + \frac{\partial \dot{w}^E}{\partial \phi}\Big|_0 \Delta\phi + \frac{\partial \dot{w}^E}{\partial \theta}\Big|_0 \Delta\theta + \frac{\partial \dot{w}^E}{\partial p}\Big|_0 \Delta p + \frac{\partial \dot{w}^E}{\partial v^E}\Big|_0 \Delta v^E + \frac{\partial \dot{w}^E}{\partial q}\Big|_0 \Delta q + \frac{\partial \dot{w}^E}{\partial u^E}\Big|_0 \Delta u^E$$

$$= \left(\frac{1}{m}\right)\Delta Z + (-g \sin\phi_0 \cos\theta_0)\Delta\phi + (-g \cos\phi_0 \sin\theta_0)\Delta\theta + (-v_0^E)\Delta p + (-p_0)\Delta v^E + (u_0^E)\Delta q + (q_0)\Delta u^E$$

$\Delta \dot{w}^E = \frac{\Delta Z}{m}$ | $\Delta Z = \Delta F_1 + \Delta F_2 + \Delta F_3 + \Delta F_4$

$$\boxed{\Delta \dot{w}^E = \frac{\Delta Z}{m} = \frac{\Delta F_1 + \Delta F_2 + \Delta F_3 + \Delta F_4}{m}}$$

ASEN 3128 Homework 3

William Watkins and Ponder Stine

Problem 2.a

Set up the Workspace

```
close all;
clear all;
clc;
```

Declare Constants

```
global g I_x I_y I_z m R G_body g_body fb

g = 9.81; % [m/s^2], acceleration of gravity
m = 0.068; % [kg], mass of the quadcopter
R = 0.060; % [m], distance from CG to motors
I_x = 6.8 * 10 ^ (-5); % [kg*m^2], moment of inertia about x-axis
I_y = 9.2 * 10 ^ (-5); % [kg*m^2], moment of inertia about y-axis
I_z = 1.35 * 10 ^ (-4); % [kg*m^2], moment of inertia about z-axis
eta = 1 * 10 ^ (-3); % [N/(m/s)^2], drag per velocity squared
alpha = 2 * 10 ^ (-6); % [N*m/(rad/s)^2], moment per rate of rotation squared
k = 0.0024; % [N*m/N], moment per force of motors
```

Initialize variables

```
x = 0; % [m], initial North position
y = 0; % [m], initial East position
z = -5; % [m], initial Down position
inertialVelocity = [0; 0; 0];
psi = 0;
theta = 0;
phi = deg2rad(5);
p = 0;
q = 0;
r = 0;
```

Create and Apply Transformation Matrix

```
L_EB = [cos(theta) * cos(psi), sin(phi) * sin(theta) * cos(psi) - cos(phi) * sin(psi), cos(phi)
        cos(theta) * sin(psi), sin(phi) * sin(theta) * sin(psi) + cos(phi) * cos(psi), cos(phi)
        -sin(theta),           sin(phi) * cos(theta),                                  cos(phi)
L_BE = inv(L_EB);
g_body = L_BE * [0; 0; 9.81];
bodyVelocity = L_BE * inertialVelocity;
u = bodyVelocity(1);
v = bodyVelocity(2);
w = bodyVelocity(3);
```

Preparing for the ODE45

```
initialState = [x y z p q r psi phi theta u v w];
tSpan = [0 10];
```

Calling ODE 45

```
[t,y] = ode45('quadcopterSimulation', tSpan, initialState);
[tlin,ylin] = ode45('quadcopterSimulationLin', tSpan, initialState);
```

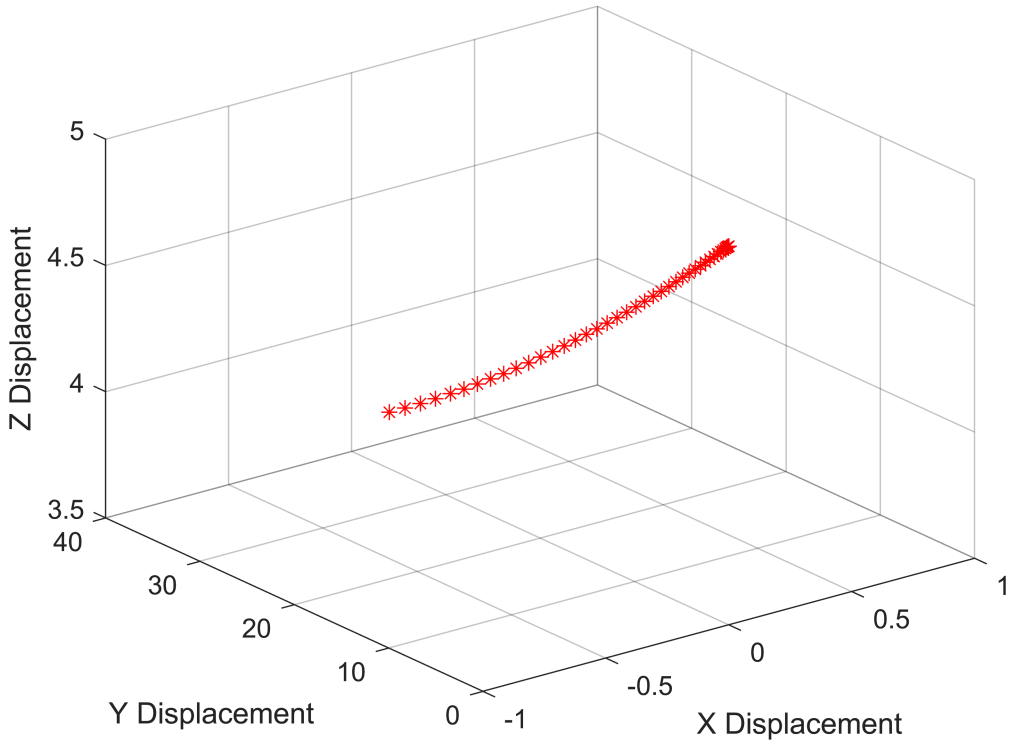
Plotting the results

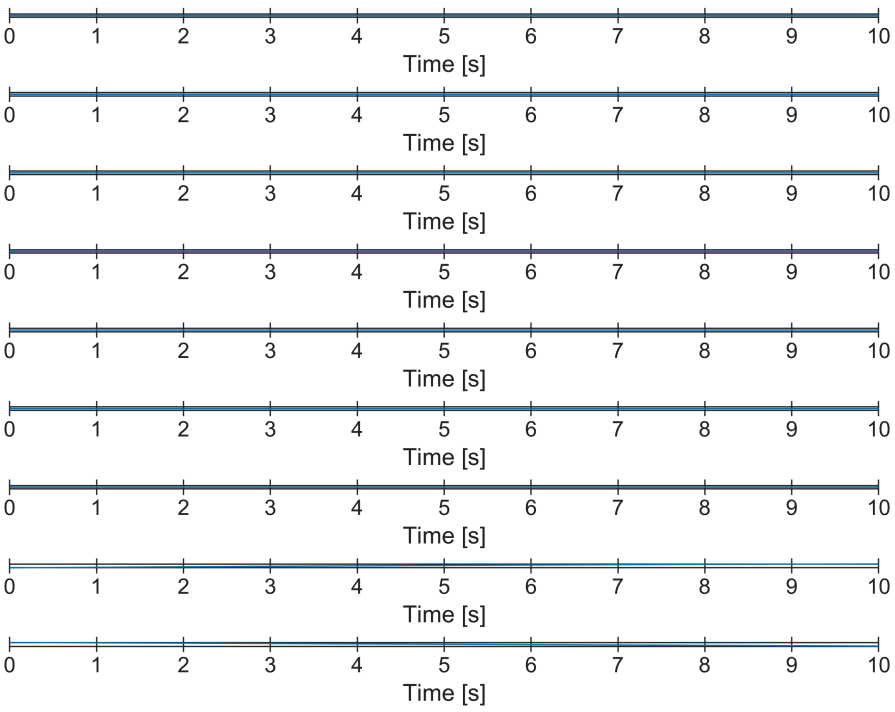
```
bodyVelocityFinal = [y(10); y(11); y(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y(10) = inertialVelocityFinal(1);
y(11) = inertialVelocityFinal(2);
y(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [ylin(10); ylin(11); ylin(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
ylin(10) = inertialVelocityFinal(1);
ylin(11) = inertialVelocityFinal(2);
ylin(12) = inertialVelocityFinal(3);
```

+5 Degree Deviation in Bank

Non Linearized

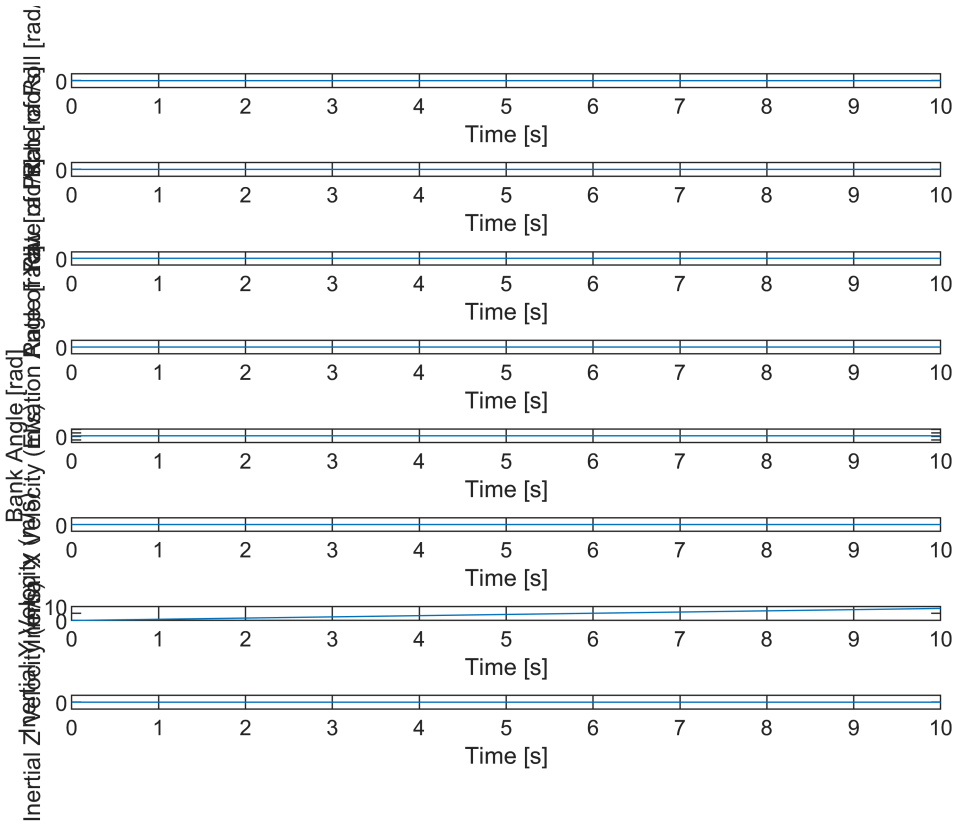
```
plotMotionNonLin(t,y);
```





Linearized

```
plotMotionLin(tlin, ylin);
```



Problem 2.b

Initialize variables

```
x = 0; % [m], initial North position
y = 0; % [m], initial East position
z = -5; % [m], initial Down position
inertialVelocity = [0; 0; 0];
psi = 0;
theta = deg2rad(5);
phi = 0;
p = 0;
q = 0;
r = 0;
```

Create and Apply Transformation Matrix

```
L_EB = [cos(theta) * cos(psi), sin(phi) * sin(theta) * cos(psi) - cos(phi) * sin(psi), cos(phi)
        cos(theta) * sin(psi), sin(phi) * sin(theta) * sin(psi) + cos(phi) * cos(psi), cos(phi)
        -sin(theta),           sin(phi) * cos(theta),                                  cos(phi)
L_BE = inv(L_EB);
g_body = L_BE * [0; 0; 9.81];
bodyVelocity2 = L_BE * inertialVelocity;
u = bodyVelocity2(1);
v = bodyVelocity2(2);
w = bodyVelocity2(3);
```

Preparing for the ODE45

```
initialState2 = [x y z p q r psi phi theta u v w];
tSpan = [0 10];
```

Calling ODE 45

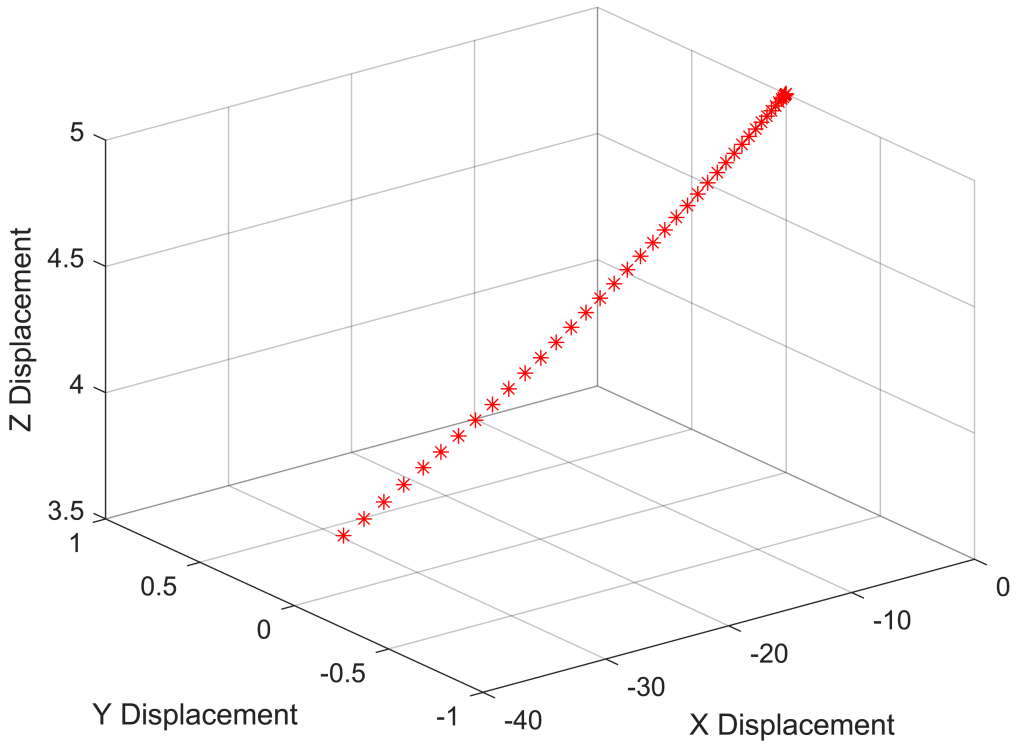
```
[t2,y2] = ode45('quadcopterSimulation', tSpan, initialState2);
[tlin2,ylin2] = ode45('quadcopterSimulationLin', tSpan, initialState2);
```

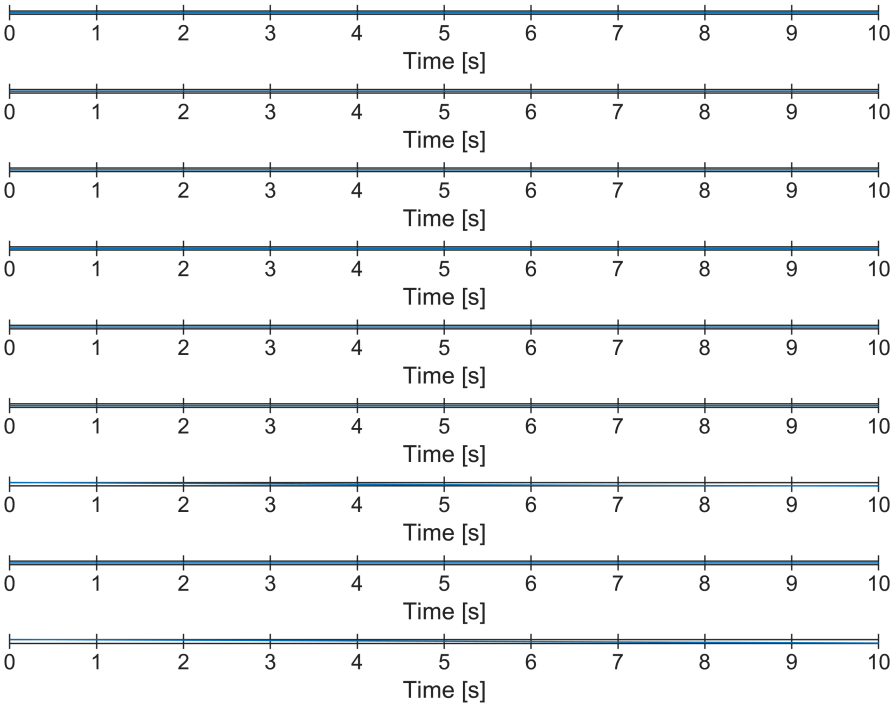
Plotting the results

```
bodyVelocityFinal = [y2(10); y2(11); y2(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y2(10) = inertialVelocityFinal(1);
y2(11) = inertialVelocityFinal(2);
y2(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [ylin2(10); ylin2(11); ylin2(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
ylin2(10) = inertialVelocityFinal(1);
ylin2(11) = inertialVelocityFinal(2);
ylin2(12) = inertialVelocityFinal(3);
```

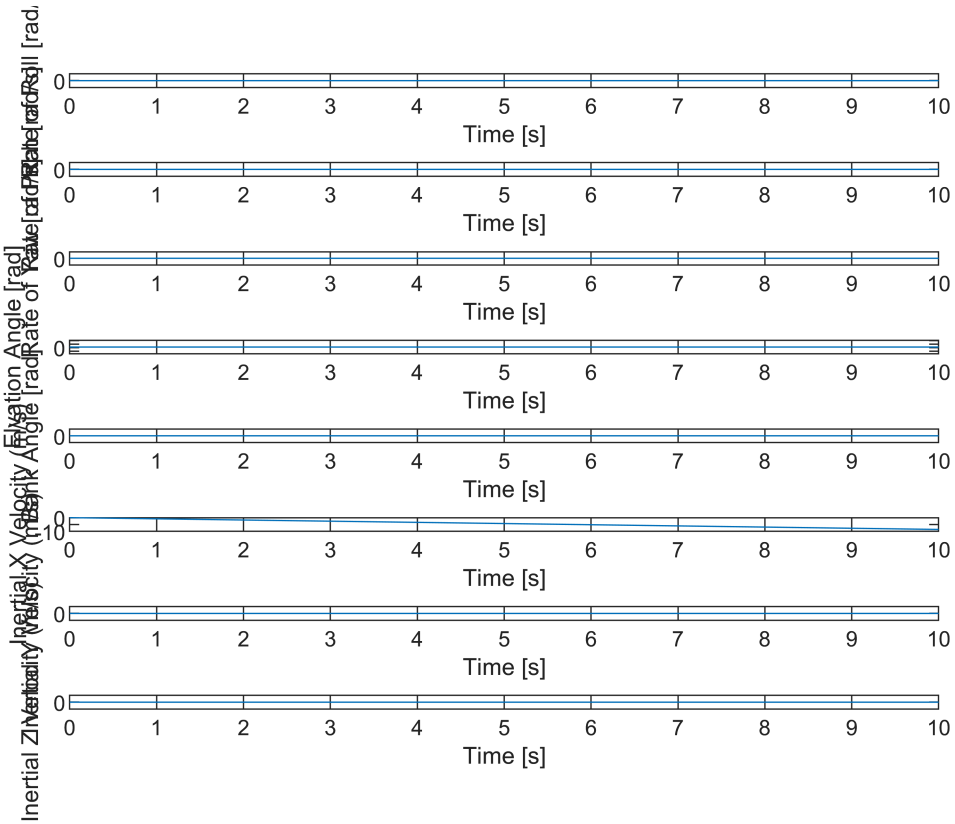
+5 Degree Deviation in Pitch

```
plotMotionNonLin(t2,y2);
```






```
plotMotionLin(tlin2, ylin2);
```



Problem 2.c

Initialize variables

```
x = 0; % [m], initial North position
y = 0; % [m], initial East position
z = -5; % [m], initial Down position
inertialVelocity = [0; 0; 0];
psi = 5 * pi / 180;
theta = 0;
phi = 0;
p = 0;
q = 0;
r = 0;
```

Create and Apply Transformation Matrix

```
L_EB = [cos(theta) * cos(psi), sin(phi) * sin(theta) * cos(psi) - cos(phi) * sin(psi), cos(phi)
        cos(theta) * sin(psi), sin(phi) * sin(theta) * sin(psi) + cos(phi) * cos(psi), cos(phi)
        -sin(theta),           sin(phi) * cos(theta),                                  cos(phi)
L_BE = inv(L_EB);
g_body = L_BE * [0; 0; 9.81];
bodyVelocity3 = L_BE * inertialVelocity;
u = bodyVelocity3(1);
v = bodyVelocity3(2);
w = bodyVelocity3(3);
```

Preparing for the ODE45

```
initialState3 = [x y z p q r psi phi theta u v w];
tSpan = [0 10];
```

Calling ODE 45

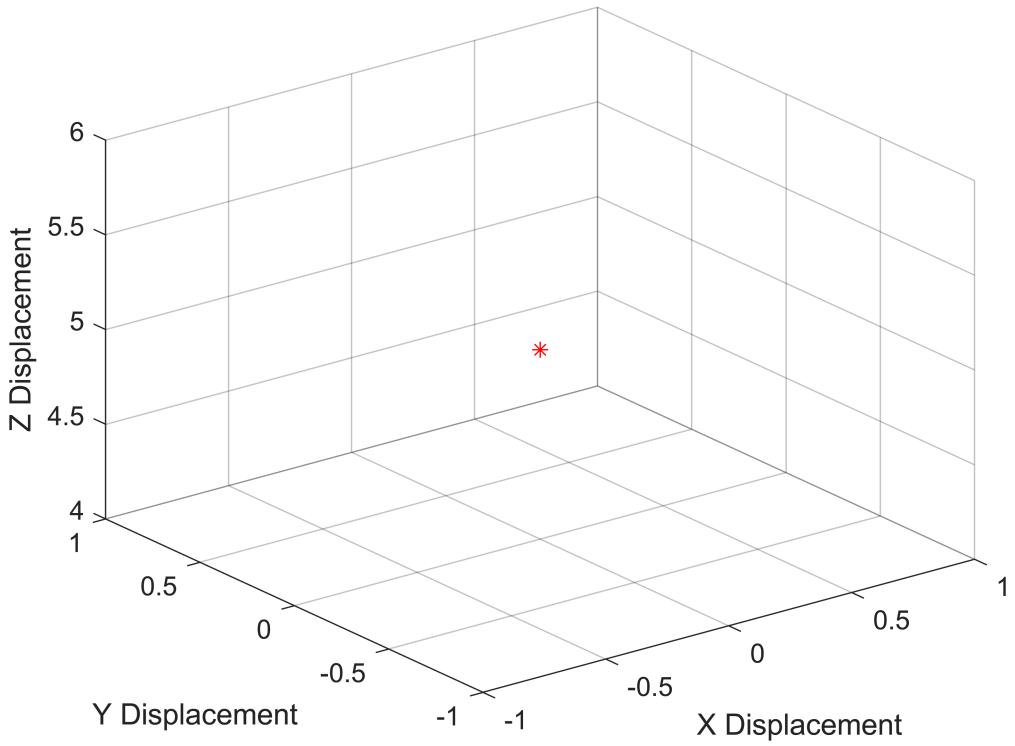
```
[t3,y3] = ode45('quadcopterSimulation', tSpan, initialState3);
[tlin3,ylin3] = ode45('quadcopterSimulationLin', tSpan, initialState3);
```

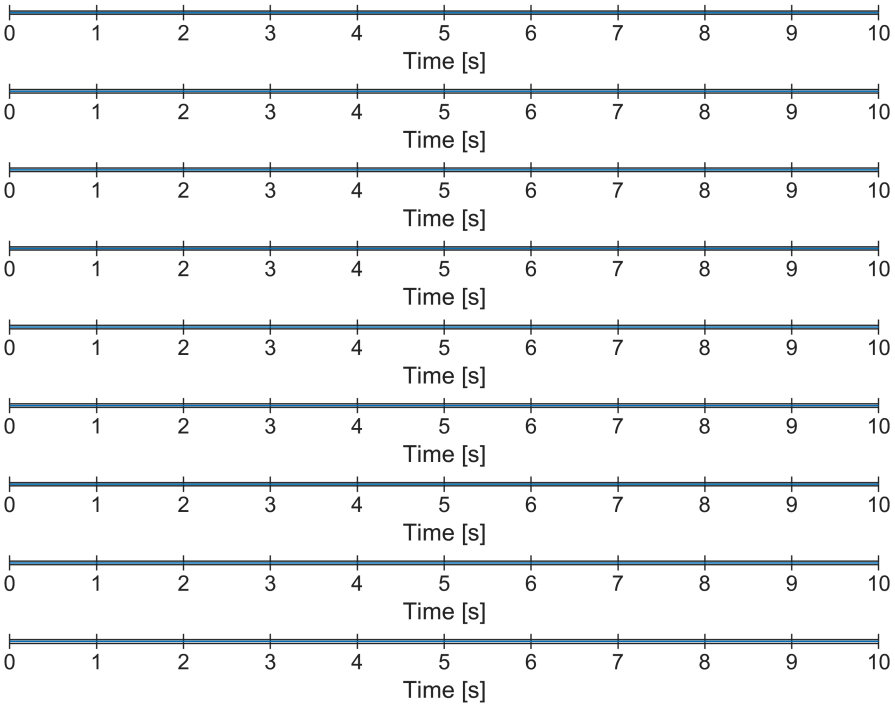
Plotting the results

```
bodyVelocityFinal = [y3(10); y3(11); y3(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y3(10) = inertialVelocityFinal(1);
y3(11) = inertialVelocityFinal(2);
y3(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [ylin3(10); ylin3(11); ylin3(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
ylin3(10) = inertialVelocityFinal(1);
ylin3(11) = inertialVelocityFinal(2);
ylin3(12) = inertialVelocityFinal(3);
```

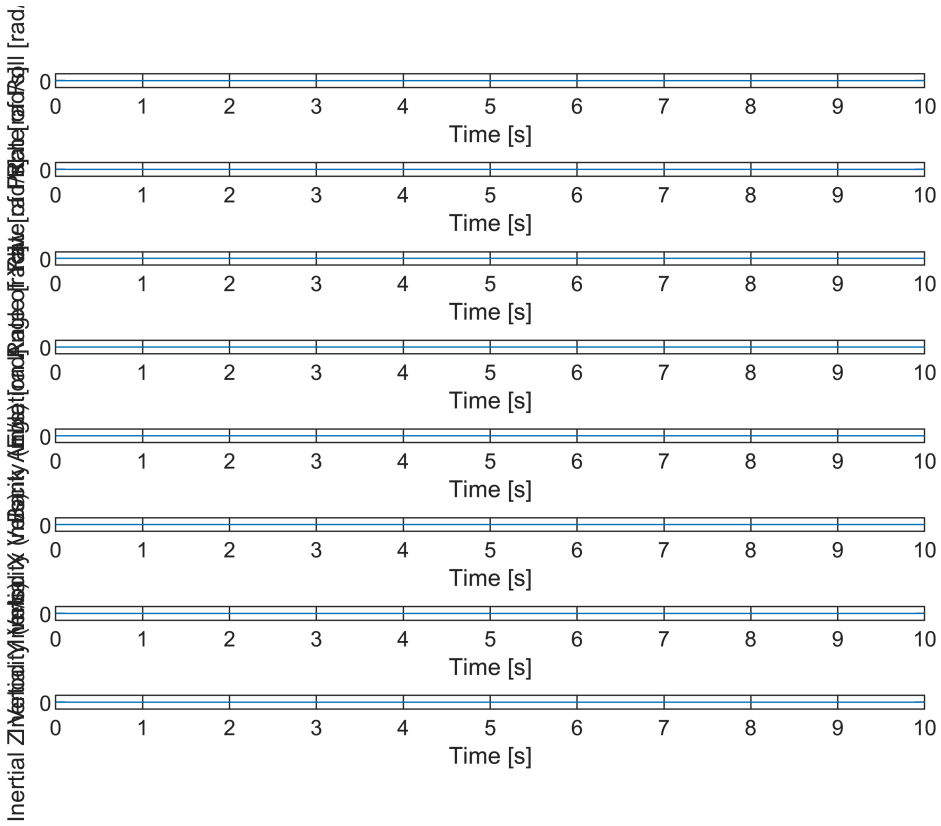
+5 Degree Deviation in Azimuth

```
plotMotionNonLin(t3,y3);
```






```
plotMotionLin(tlin3, ylin3);
```



Problem 2.d

Initialize variables

```
x = 0; % [m], initial North position
y = 0; % [m], initial East position
z = -5; % [m], initial Down position
inertialVelocity = [0; 0; 0];
psi = 0;
theta = 0;
phi = 0;
p = 0.1;
q = 0;
r = 0;
```

Create and Apply Transformation Matrix

```
L_EB = [cos(theta) * cos(psi), sin(phi) * sin(theta) * cos(psi) - cos(phi) * sin(psi), cos(phi)
        cos(theta) * sin(psi), sin(phi) * sin(theta) * sin(psi) + cos(phi) * cos(psi), cos(phi)
        -sin(theta),           sin(phi) * cos(theta),                                  cos(phi)
L_BE = inv(L_EB);
g_body = L_BE * [0; 0; 9.81];
bodyVelocity4 = L_BE * inertialVelocity;
u = bodyVelocity4(1);
v = bodyVelocity4(2);
w = bodyVelocity4(3);
```

Preparing for the ODE45

```
initialState4 = [x y z p q r psi phi theta u v w];
tSpan = [0 10];
```

Calling ODE 45

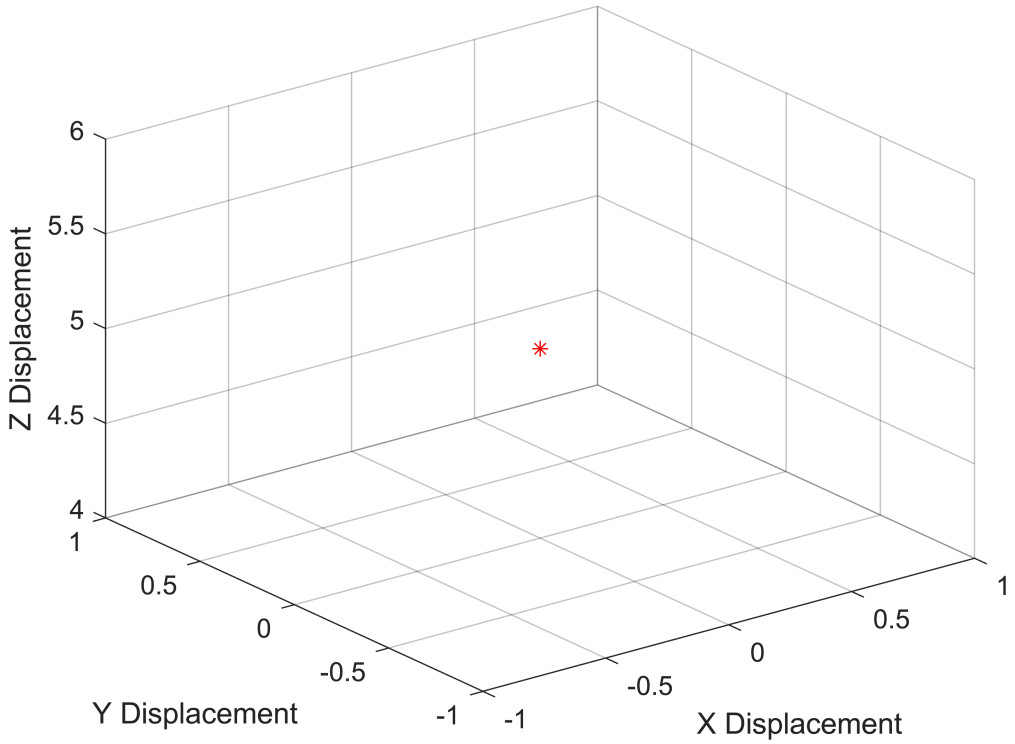
```
[t4,y4] = ode45('quadcopterSimulation', tSpan, initialState4);
[tlin4,ylin4] = ode45('quadcopterSimulationLin', tSpan, initialState4);
[t4F,y4F] = ode45('quadcopterSimulationNonLinFC', tSpan, initialState4);
```

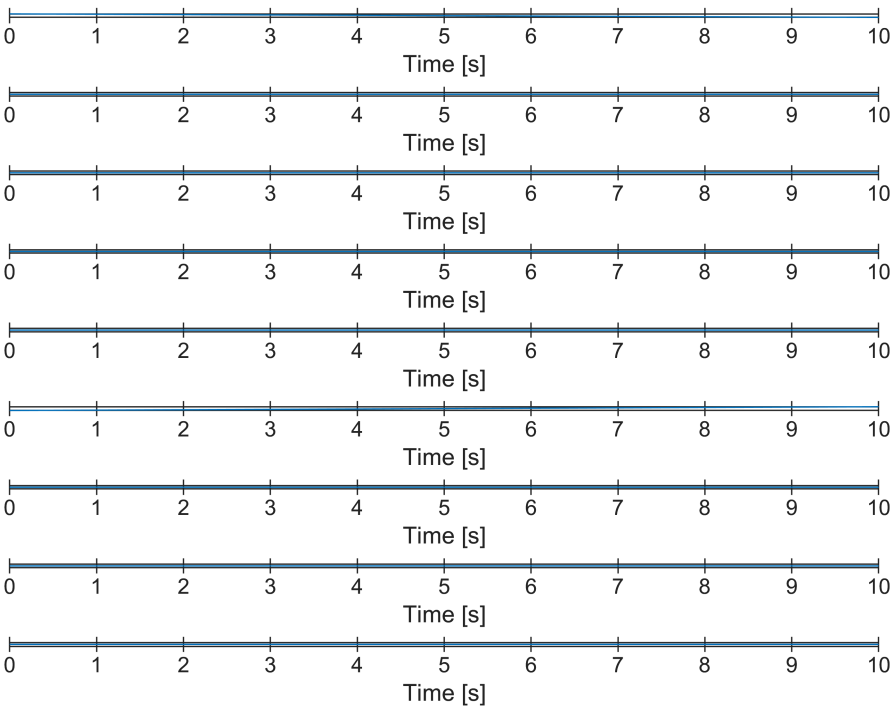
Plotting the results

```
bodyVelocityFinal = [y4(10); y4(11); y4(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y4(10) = inertialVelocityFinal(1);
y4(11) = inertialVelocityFinal(2);
y4(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [ylin4(10); ylin4(11); ylin4(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
ylin4(10) = inertialVelocityFinal(1);
ylin4(11) = inertialVelocityFinal(2);
ylin4(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [y4F(10); y4F(11); y4F(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y4F(10) = inertialVelocityFinal(1);
y4F(11) = inertialVelocityFinal(2);
y4F(12) = inertialVelocityFinal(3);
```

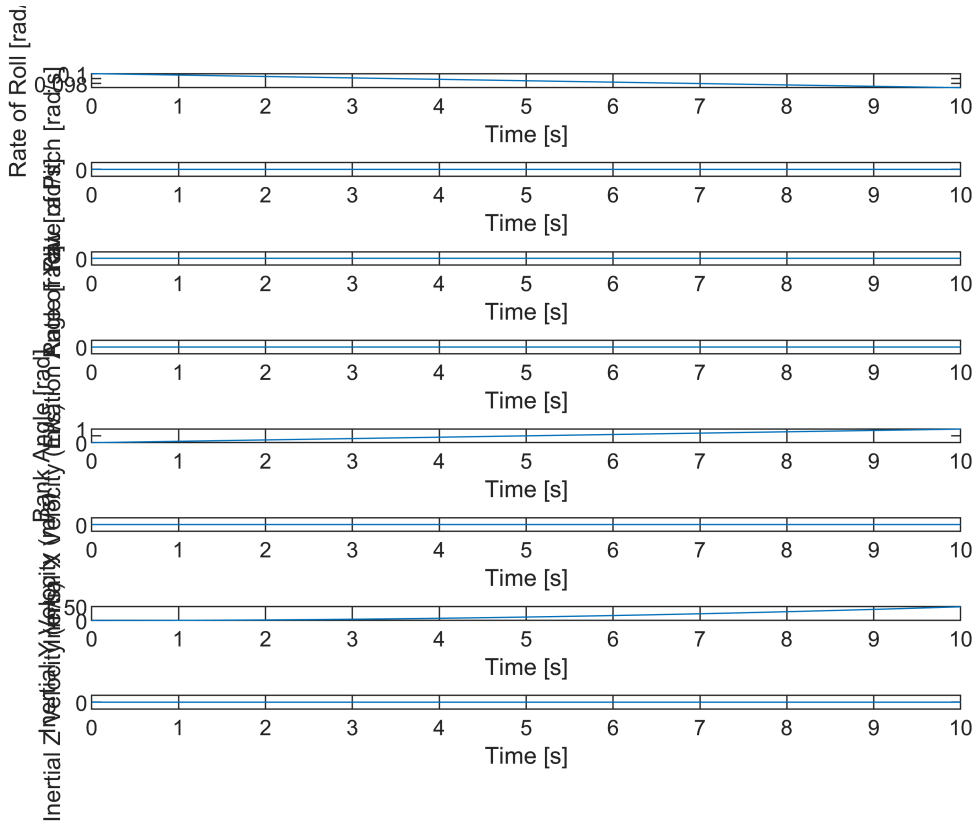
+0.1 rad/s Roll Rate Deviation

```
plotMotionNonLin(t4,y4);
```

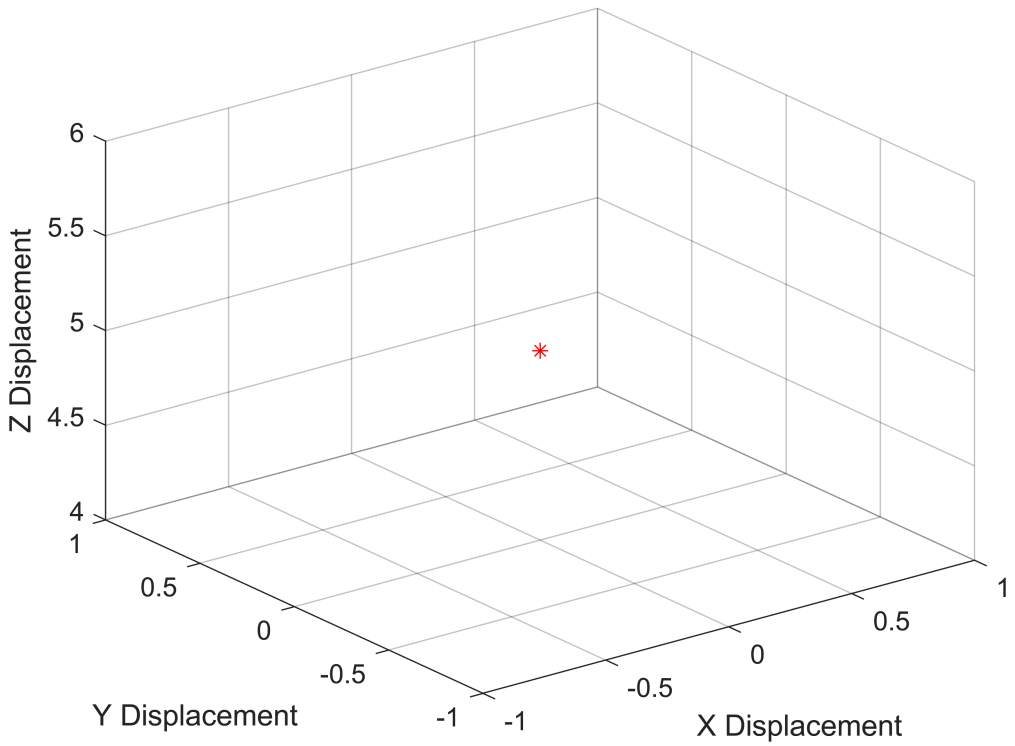


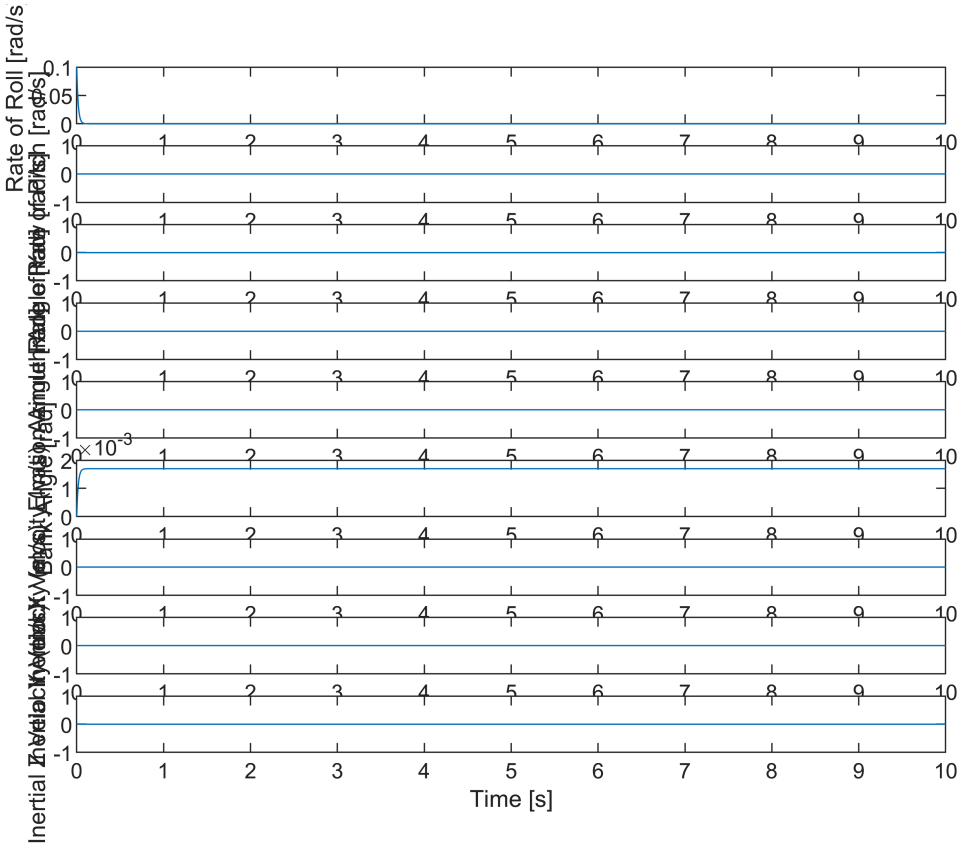



```
plotMotionLin(tlin4, ylin4);
```




```
plotMotionNonLin(t4F,y4F);
```





Problem 2.e

Initialize variables

```
x = 0; % [m], initial North position
y = 0; % [m], initial East position
z = -5; % [m], initial Down position
inertialVelocity = [0; 0; 0];
psi = 0;
theta = 0;
phi = 0;
p = 0;
q = 0.1;
r = 0;
```

Create and Apply Transformation Matrix

```
L_EB = [cos(theta) * cos(psi), sin(phi) * sin(theta) * cos(psi) - cos(phi) * sin(psi), cos(phi)
        cos(theta) * sin(psi), sin(phi) * sin(theta) * sin(psi) + cos(phi) * cos(psi), cos(phi)
        -sin(theta),           sin(phi) * cos(theta),                                  cos(phi)
L_BE = inv(L_EB);
g_body = L_BE * [0; 0; 9.81];
bodyVelocity5 = L_BE * inertialVelocity;
u = bodyVelocity5(1);
v = bodyVelocity5(2);
```

```
w = bodyVelocity5(3);
```

Preparing for the ODE45

```
initialState5 = [x y z p q r psi phi theta u v w];
tSpan = [0 10];
```

Calling ODE 45

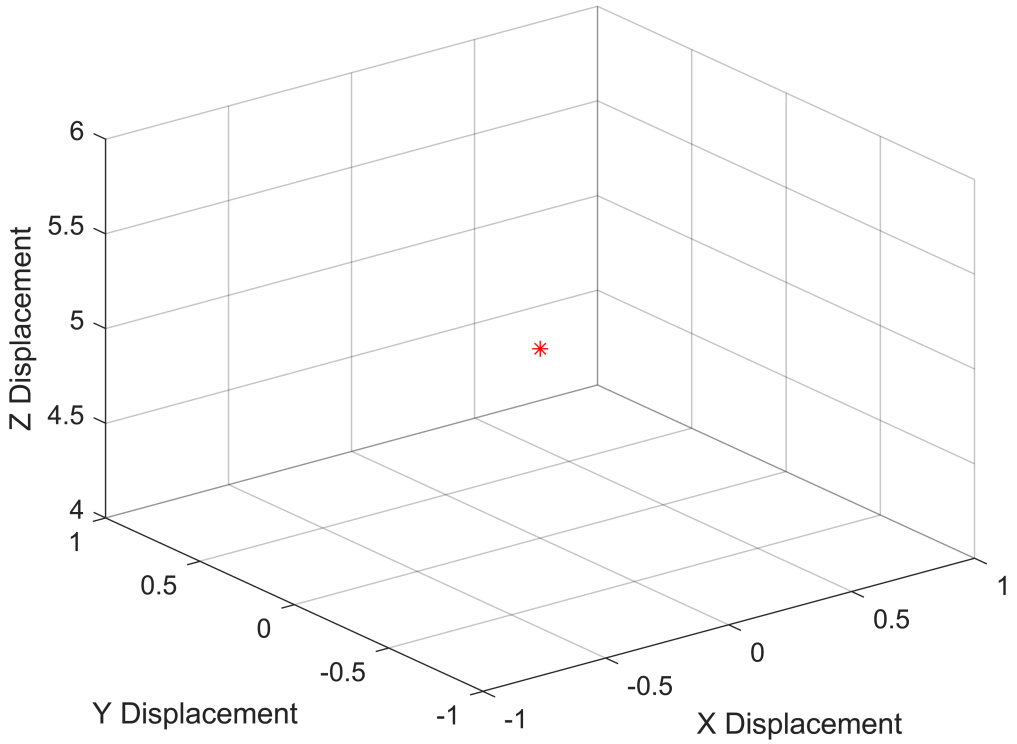
```
[t5,y5] = ode45('quadcopterSimulation', tSpan, initialState5);
[tlin5,ylin5] = ode45('quadcopterSimulationLin', tSpan, initialState5);
[t5F,y5F] = ode45('quadcopterSimulationNonLinFC', tSpan, initialState4);
```

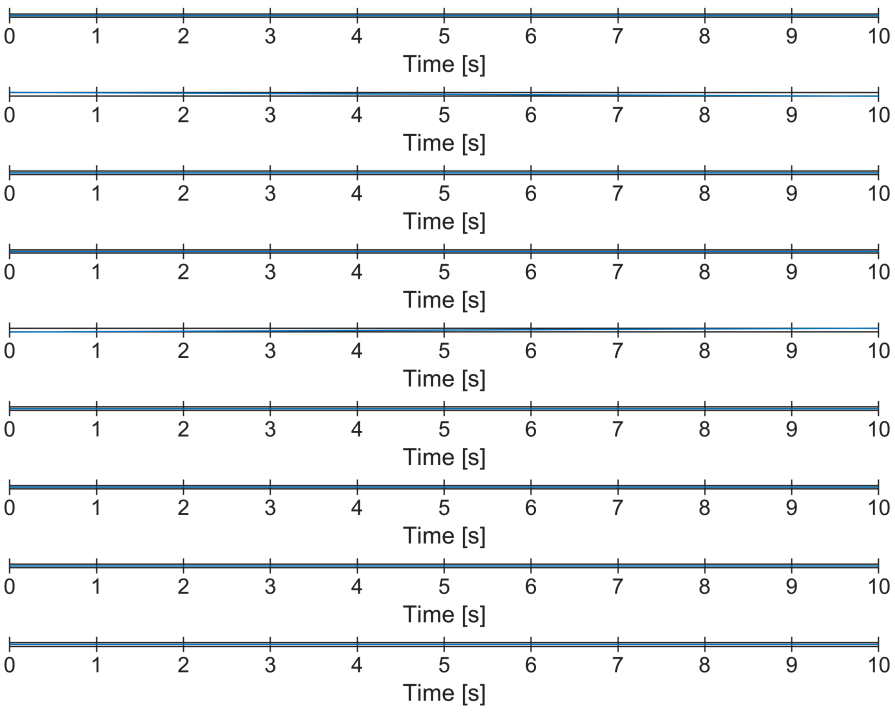
Plotting the results

```
bodyVelocityFinal = [y5(10); y5(11); y5(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y5(10) = inertialVelocityFinal(1);
y5(11) = inertialVelocityFinal(2);
y5(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [ylin5(10); ylin5(11); ylin5(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
ylin5(10) = inertialVelocityFinal(1);
ylin5(11) = inertialVelocityFinal(2);
ylin5(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [y5F(10); y5F(11); y5F(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y5F(10) = inertialVelocityFinal(1);
y5F(11) = inertialVelocityFinal(2);
y5F(12) = inertialVelocityFinal(3);
```

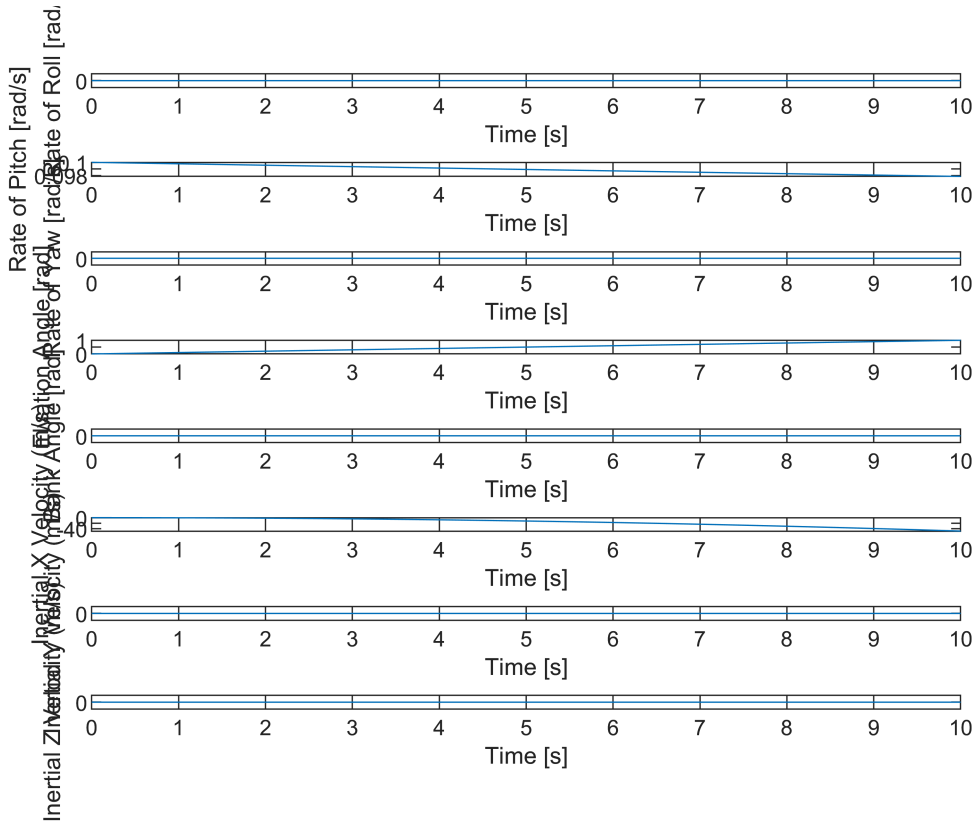
+0.1 rad/s Pitch Rate Deviation

```
plotMotionNonLin(t5,y5);
```

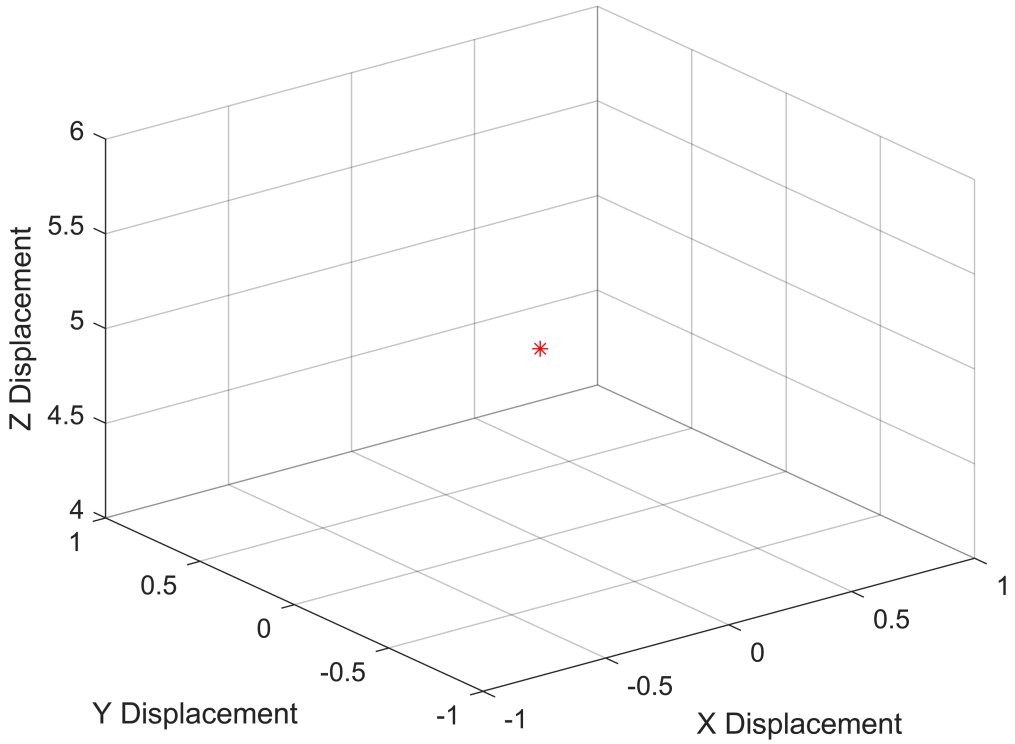


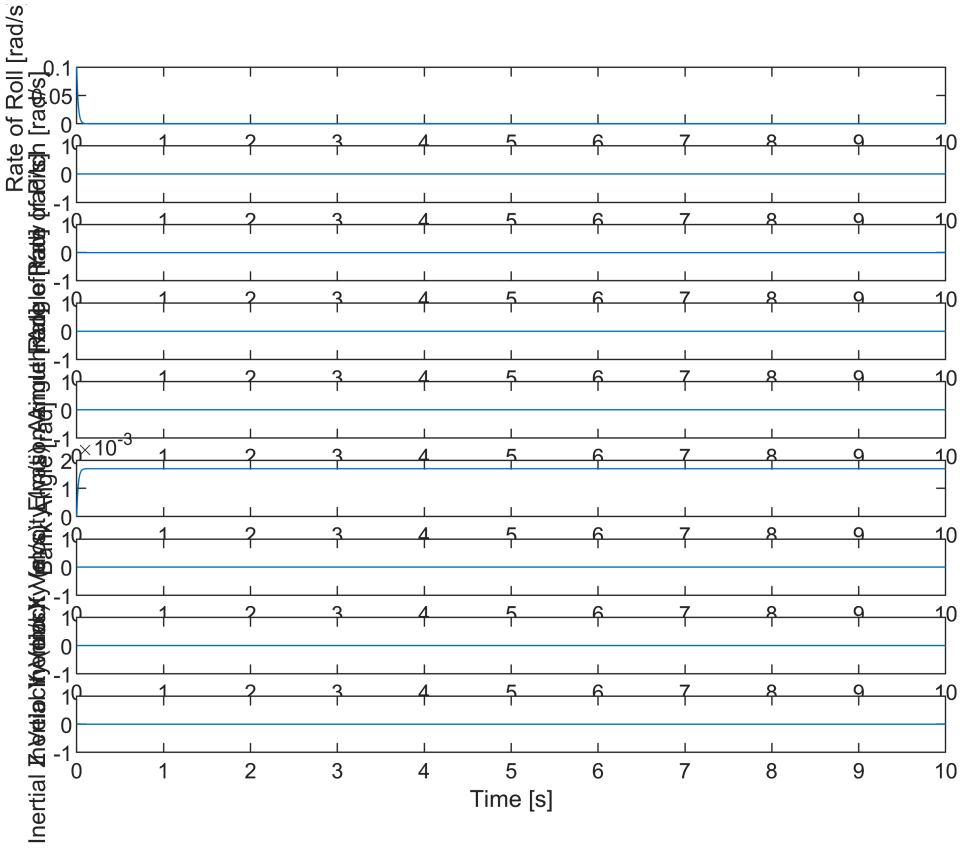



```
plotMotionLin(tlin5, ylin5);
```




```
plotMotionNonLin(t5F,y5F);
```





Problem 2.f

Initialize variables

```
x = 0; % [m], initial North position
y = 0; % [m], initial East position
z = -5; % [m], initial Down position
inertialVelocity = [0; 0; 0];
psi = 0;
theta = 0;
phi = 0;
p = 0;
q = 0;
r = 0.1;
```

Create and Apply Transformation Matrix

```
L_EB = [cos(theta) * cos(psi), sin(phi) * sin(theta) * cos(psi) - cos(phi) * sin(psi), cos(phi)
        cos(theta) * sin(psi), sin(phi) * sin(theta) * sin(psi) + cos(phi) * cos(psi), cos(phi)
        -sin(theta),           sin(phi) * cos(theta),                                  cos(phi)
L_BE = inv(L_EB);
g_body = L_BE * [0; 0; 9.81];
bodyVelocity6 = L_BE * inertialVelocity;
u = bodyVelocity6(1);
v = bodyVelocity6(2);
```

```
w = bodyVelocity6(3);
```

Preparing for the ODE45

```
initialState6 = [x y z p q r psi phi theta u v w];
tSpan = [0 10];
```

Calling ODE 45

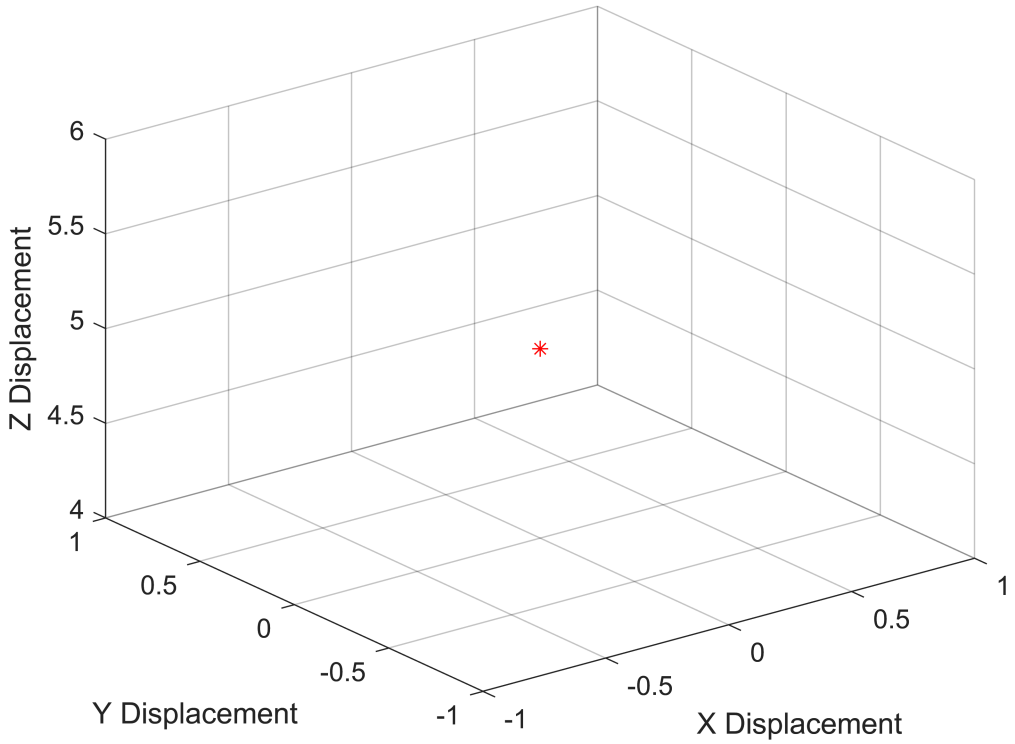
```
[t6,y6] = ode45('quadcopterSimulation', tSpan, initialState6);
[tlin6,ylin6] = ode45('quadcopterSimulationLin', tSpan, initialState6);
[t6F,y6F] = ode45('quadcopterSimulationNonLinFC', tSpan, initialState4);
```

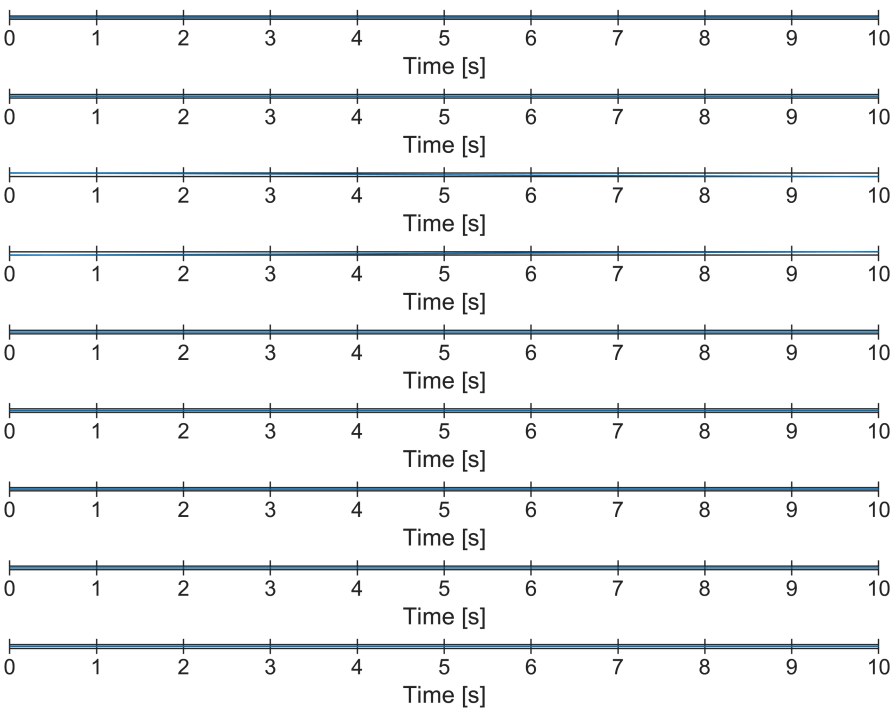
Plotting the results

```
bodyVelocityFinal = [y6(10); y6(11); y6(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y6(10) = inertialVelocityFinal(1);
y6(11) = inertialVelocityFinal(2);
y6(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [ylin6(10); ylin6(11); ylin6(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
ylin6(10) = inertialVelocityFinal(1);
ylin6(11) = inertialVelocityFinal(2);
ylin6(12) = inertialVelocityFinal(3);
bodyVelocityFinal = [y6F(10); y6F(11); y6F(12)];
inertialVelocityFinal = L_EB * bodyVelocityFinal;
y6F(10) = inertialVelocityFinal(1);
y6F(11) = inertialVelocityFinal(2);
y6F(12) = inertialVelocityFinal(3);
```

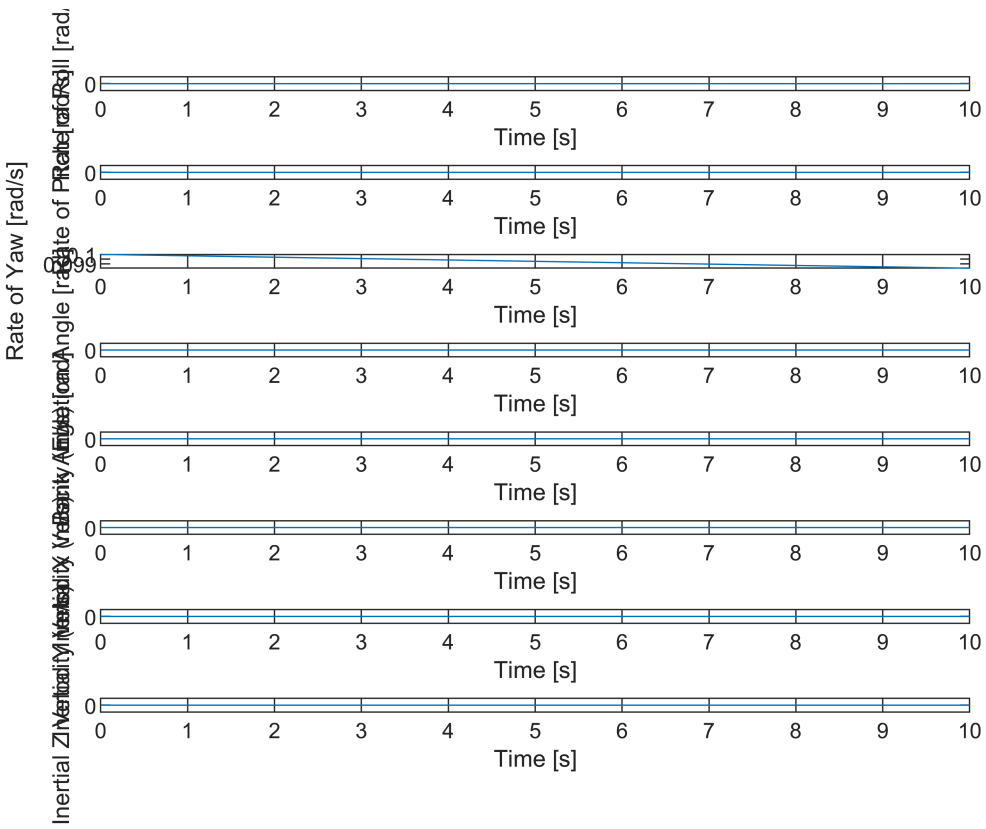
+0.1 rad/s Yaw Rate Deviation

```
plotMotionNonLin(t6,y6);
```

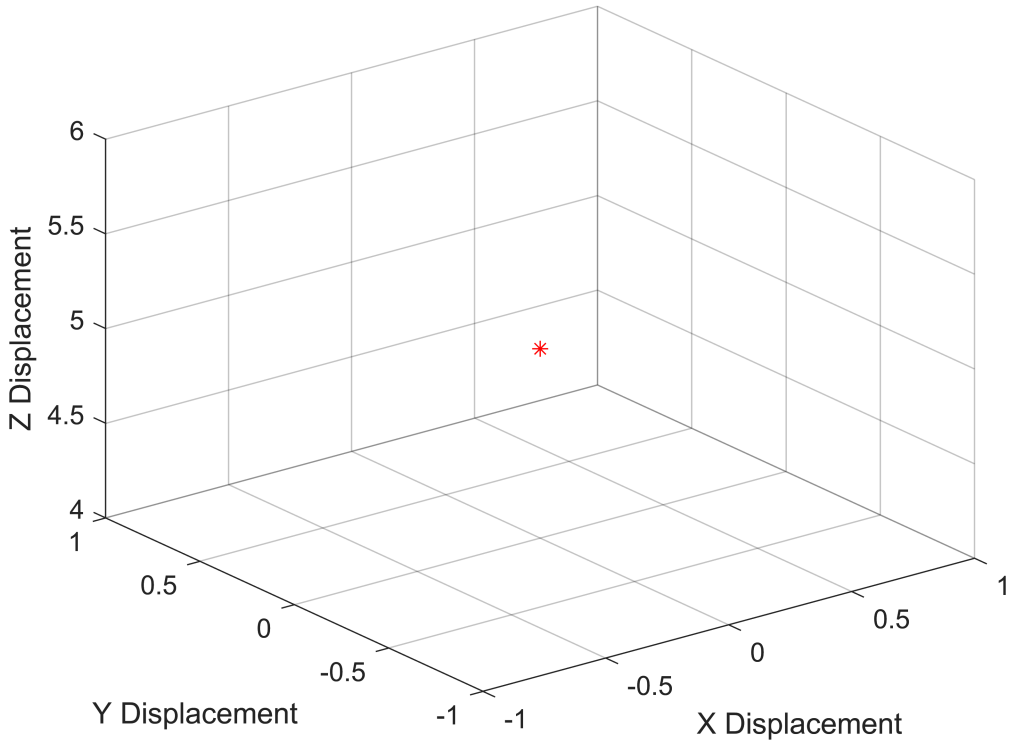


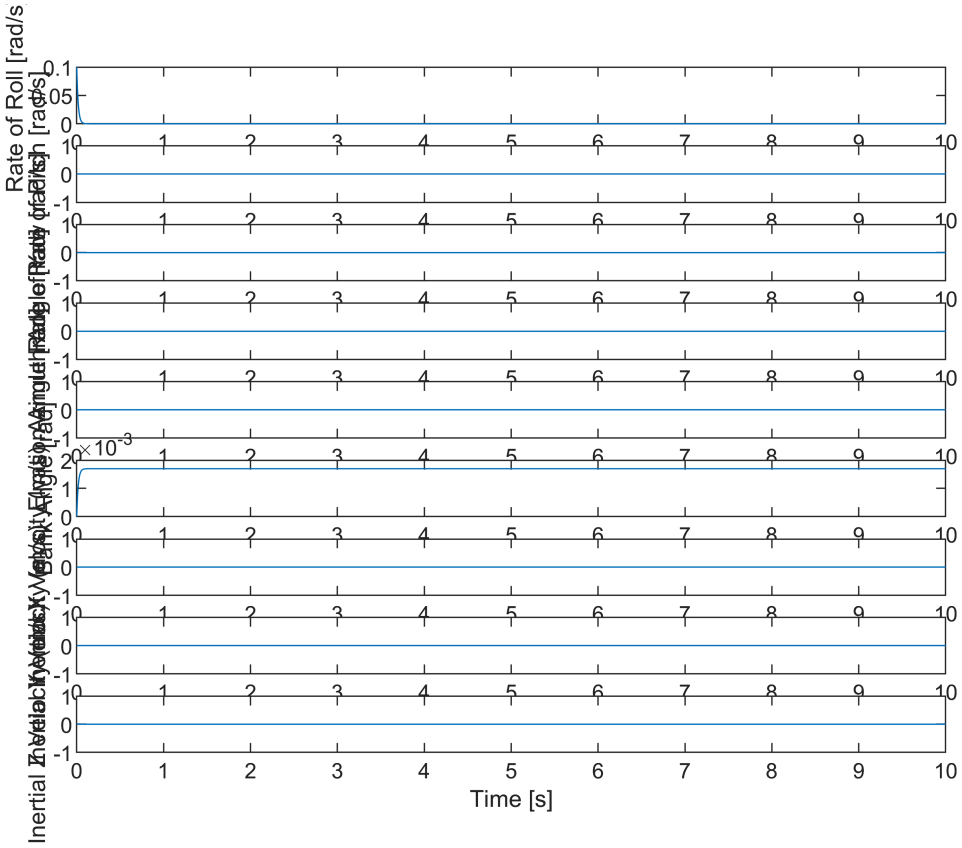



```
plotMotionLin(tlin6, ylin6);
```




```
plotMotionNonLin(t6F,y6F);
```

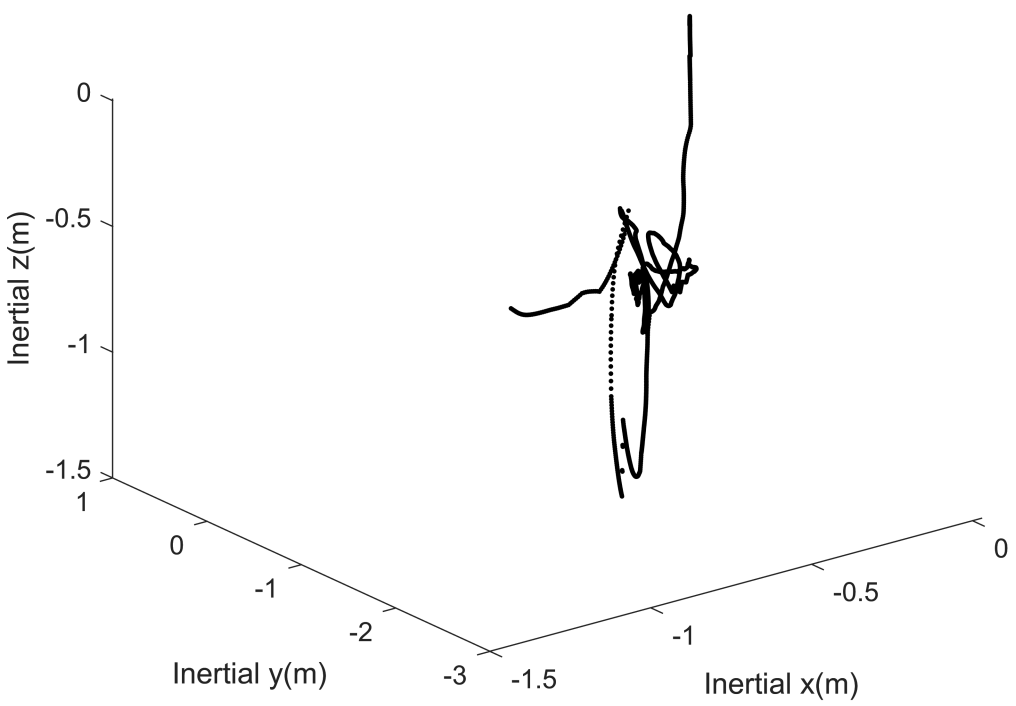




Problem 5

```
load('RSdata_White_0903.mat');
time=rt_estim.time;
x=rt_estim.signals.values(:,1);
y=rt_estim.signals.values(:,2);
z=rt_estim.signals.values(:,3);
yaw=rt_estim.signals.values(:,4);
pitch=rt_estim.signals.values(:,5);
roll=rt_estim.signals.values(:,6);
Vx=rt_estim.signals.values(:,7);
Vy=rt_estim.signals.values(:,8);
Vz=rt_estim.signals.values(:,9);
p=rt_estim.signals.values(:,10);
q=rt_estim.signals.values(:,11);
r=rt_estim.signals.values(:,12);

figure()
plot3(x,y,z,'.k')
title('Flight Path of Rolling Spider Quadcopter')
xlabel('Inertial x(m)')
ylabel('Inertial y(m)')
zlabel('Inertial z(m)')
```

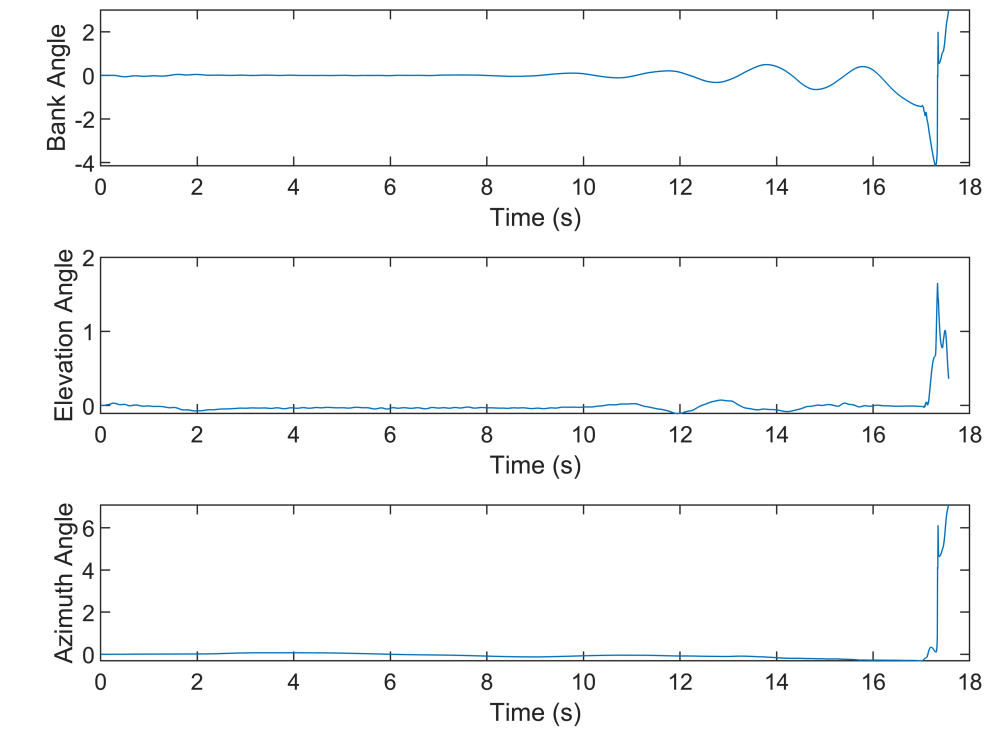



```
figure()
subplot(3,1,1)
plot(time,roll)
xlabel('Time (s)')
ylabel('Bank Angle')

subplot(3,1,2)
plot(time,pitch)
xlabel('Time (s)')
ylabel('Elevation Angle')

subplot(3,1,3)
plot(time,yaw)
xlabel('Time (s)')
ylabel('Azimuth Angle')

sgtitle('Euler Angles of Attitude for the Rolling Spider Quadcopter')
```

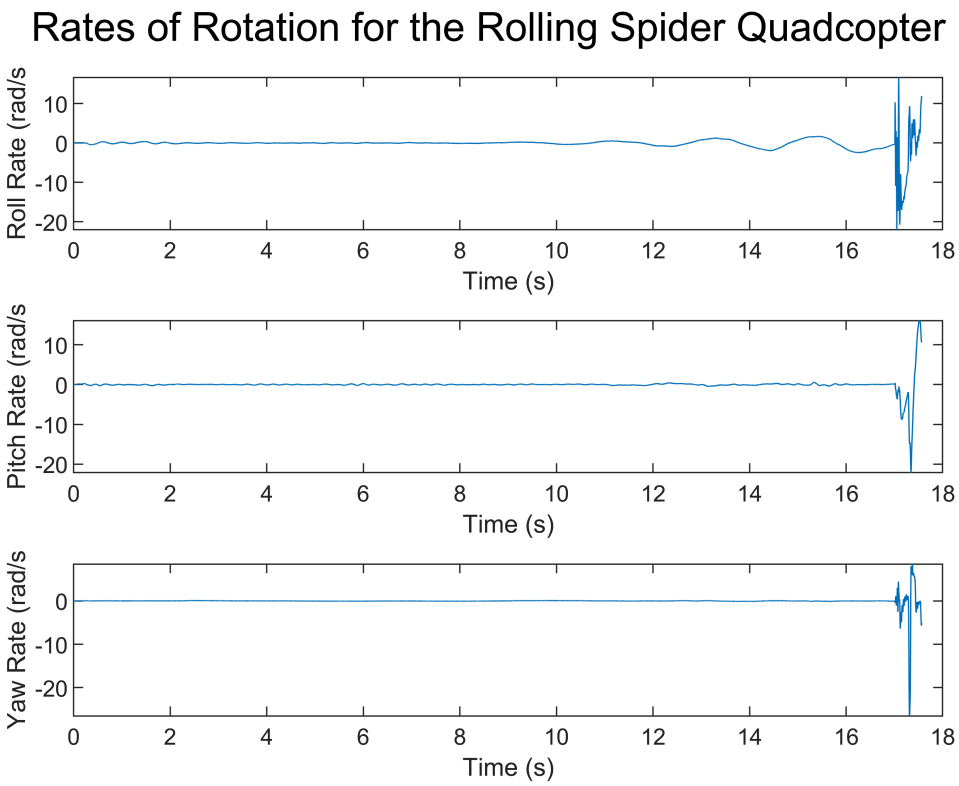



```
figure()
subplot(3,1,1)
plot(time,p)
xlabel('Time (s)')
ylabel('Roll Rate (rad/s')

subplot(3,1,2)
plot(time,q)
xlabel('Time (s)')
ylabel('Pitch Rate (rad/s')

subplot(3,1,3)
plot(time,r)
xlabel('Time (s)')
ylabel('Yaw Rate (rad/s')

sgtitle('Rates of Rotation for the Rolling Spider Quadcopter')
```

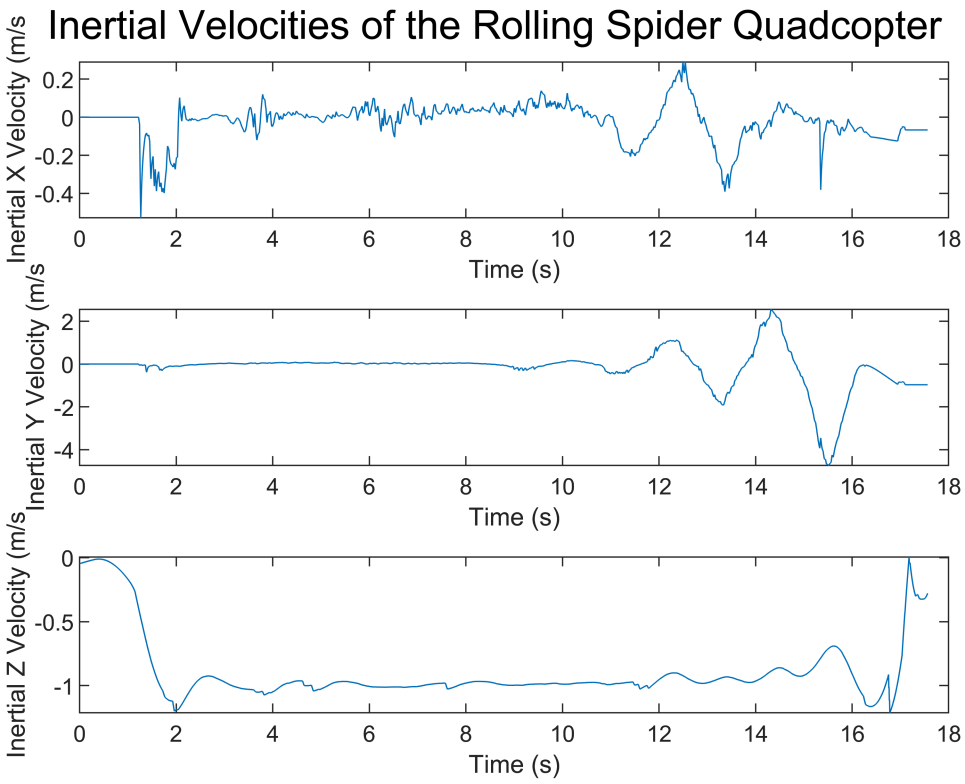



```
figure()
subplot(3,1,1)
plot(time,Vx)
xlabel('Time (s)')
ylabel('Inertial X Velocity (m/s')

subplot(3,1,2)
plot(time,Vy)
xlabel('Time (s)')
ylabel('Inertial Y Velocity (m/s')

subplot(3,1,3)
plot(time,z)
xlabel('Time (s)')
ylabel('Inertial Z Velocity (m/s')

sgtitle('Inertial Velocities of the Rolling Spider Quadcopter')
```



plotMotionNonLin plots the nonlinearized differential equations for a quadcopter

Detailed explanation of this function.

```
function [] = plotMotionNonLin(t, y)

figure;
title("Non-Linearized Responses")
plot3(y(:,1), y(:,2), -y(:,3), "r*");
xlabel("X Displacement");
ylabel("Y Displacement");
zlabel("Z Displacement");
grid on

figure;
subplot(9,1,1);
plot(t, y(:,4));
xlabel("Time [s]");
ylabel("Rate of Roll [rad/s]");

subplot(9,1,2);
plot(t, y(:,5));
xlabel("Time [s]");
ylabel("Rate of Pitch [rad/s]");

subplot(9,1,3);
plot(t, y(:,6));
```

```
xlabel("Time [s]");
ylabel("Rate of Yaw [rad/s]");

subplot(9,1,4);
plot(t, y(:,7));
xlabel("Time [s]");
ylabel("Azimuth Angle [rad]");

subplot(9,1,5);
plot(t, y(:,9));
xlabel("Time [s]");
ylabel("Elvation Angle [rad]");

subplot(9,1,6);
plot(t, y(:,8));
xlabel("Time [s]");
ylabel("Bank Angle [rad]");

subplot(9,1,7);
plot(t, y(:,10));
xlabel("Time [s]");
ylabel("Inertial X Velocity (m/s)");

subplot(9,1,8);
plot(t, y(:,11));
xlabel("Time [s]");
ylabel("Inertial Y Velocity (m/s)");

subplot(9,1,9);
plot(t, y(:,12));
xlabel("Time [s]");
ylabel("Inertial Z Velocity (m/s)");

end
```

plotMotionNonLin plots the nonlinearized differential equations for a quadcopter

Detailed explanation of this function.

```
function [] = plotMotionLin(t, y)

figure;
title("Linearized Responses")
subplot(8,1,1);
plot(t, y(:,4));
xlabel("Time [s]");
ylabel("Rate of Roll [rad/s]");

subplot(8,1,2);
plot(t, y(:,5));
xlabel("Time [s]");
ylabel("Rate of Pitch [rad/s]");

subplot(8,1,3);
```

```
plot(t, y(:,6));
xlabel("Time [s]");
ylabel("Rate of Yaw [rad/s]");

subplot(8,1,4);
plot(t, y(:,9));
xlabel("Time [s]");
ylabel("Elvation Angle [rad]");

subplot(8,1,5);
plot(t, y(:,8));
xlabel("Time [s]");
ylabel("Bank Angle [rad]");

subplot(8,1,6);
plot(t, y(:,10));
xlabel("Time [s]");
ylabel("Inertial X Velocity (m/s)");

subplot(8,1,7);
plot(t, y(:,11));
xlabel("Time [s]");
ylabel("Inertial Y Velocity (m/s)");

subplot(8,1,8);
plot(t, y(:,12));
xlabel("Time [s]");
ylabel("Inertial Z Velocity (m/s)");

end
```

ODE45 function for the motion of a quadcopter

Detailed explanation of this function.

```
function derivativeState = quadcopterSimulation(t,State)
```

Declare global and local constants.

```
global g I_x I_y I_z m R fb f1 f2 f3 f4 L M N G_body g_body
eta = 1 * 10 ^ (-3); % [N/(m/s)^2], drag per velocity squared
alpha = 2 * 10 ^ (-6); % [N*m/(rad/s)^2], moment per rate of rotation squared
k = 0.0024; % [N*m/N], moment per force of motors
```

Extract variable from state vector.

```
x = State(1);
y = State(2);
z = State(3);
p = State(4);
q = State(5);
r = State(6);
psi = State(7);
phi = State(8);
```

```
theta = State(9);
u = State(10);
v = State(11);
w = State(12);
```

Calculate the forces and moments.

```
Thrust = (-eta * sqrt(u^2 + v^2 + w^2) * w) + (m * g);
f1 = -Thrust / 4;
f2 = -Thrust / 4;
f3 = -Thrust / 4;
f4 = -Thrust / 4;

A_aerodynamic_body = -eta * sqrt(u^2 + v^2 + w^2) * [u; v; w]; % [N], aerodynamic forces on veh
A_control_body = [0; 0; f1 + f2 + f3 + f4]; % [N], control forces on vehicle
fb = m * g_body + A_aerodynamic_body + A_control_body; % [N], vector of forces on quadcopter

G_aerodynamic_body = -alpha * sqrt(p^2 + q^2 + r^2) * [p; q; r]; % [N*m], aerodynamic moments o
L = (R/sqrt(2)) * (f1 + f2 - f3 - f4); % [N*m], control moment
M = (R/sqrt(2)) * (-f1 + f2 + f3 - f4); % [N*m], control moment
N = k * (f1 - f2 + f3 - f4); % [N*m], control moment
G_control_body = [L; M; N]; % [N*m], total control moment
G_body = G_aerodynamic_body + G_control_body + 0; % [N*m], total moments on vehicle
```

Calculate the derivatives.

```
xDot = (u * cos(theta) * cos(psi)) + (v * (sin(phi) * sin(theta) * cos(psi) - cos(phi)...
    * sin(psi))) + w * (cos(phi) * sin(theta) * cos(psi) + sin(phi) * sin(psi));
yDot = (u * cos(theta) * sin(psi)) + (v * (sin(phi) * sin(theta) * sin(psi)...
    + cos(phi) * cos(psi))) + w * (cos(phi) * sin(theta) * sin(psi) - sin(phi) * cos(psi));
zDot = (-u * sin(theta))+(v * sin(phi) * cos(theta)) + (w * cos(phi) * cos(theta));

uDot = (fb(1) / m) - (q * w) + (r * v);
vDot = (fb(2) / m) - (r * u) + (p * w);
wDot = (fb(3) / m) - (p * v) + (q * u);

psiDot = (q * sin(phi) + r * cos(phi)) * sec(theta);
phiDot = p + (q * sin(phi) + r * cos(phi)) * tan(theta);
thetaDot =  (q * cos(phi)) - (r * sin(phi));

pDot = (1 / I_x) * (G_body(1) - ((q * r) * (I_z - I_y)));
qDot = (1 / I_y) * (G_body(2) - ((r * p) * (I_x - I_z)));
rDot = (1 / I_z) * (G_body(3) - ((p * q) * (I_y - I_x)));

derivativeState = [xDot yDot zDot pDot qDot rDot psiDot phiDot thetaDot uDot vDot wDot].';

end
```

ODE45 function for the motion of a quadcopter

Detailed explanation of this function.

```
function derivativeState = quadcopterSimulationLin(t,State)
```

Declare global and local constants.

```
global g I_x I_y I_z m R fb f1 f2 f3 f4 L M N G_body g_body
eta = 1 * 10 ^ (-3); % [N/(m/s)^2], drag per velocity squared
alpha = 2 * 10 ^ (-6); % [N*m/(rad/s)^2], moment per rate of rotation squared
k = 0.0024; % [N*m/N], moment per force of motors
```

Extract variable from state vector.

```
x = State(1);
y = State(2);
z = State(3);
p = State(4);
q = State(5);
r = State(6);
psi = State(7);
phi = State(8);
theta = State(9);
u = State(10);
v = State(11);
w = State(12);
```

Calculate the forces and moments.

```
Thrust = (-eta * sqrt(u^2 + v^2 + w^2) * w) + (m * g);
f1 = -Thrust / 4;
f2 = -Thrust / 4;
f3 = -Thrust / 4;
f4 = -Thrust / 4;

A_aerodynamic_body = -eta * sqrt(u^2 + v^2 + w^2) * [u; v; w]; % [N], aerodynamic forces on veh
A_control_body = [0; 0; f1 + f2 + f3 + f4]; % [N], control forces on vehicle
fb = m * g_body + A_aerodynamic_body + A_control_body; % [N], vector of forces on quadcopter

G_aerodynamic_body = -alpha * sqrt(p^2 + q^2 + r^2) * [p; q; r]; % [N*m], aerodynamic moments c
L = (R/sqrt(2)) * (f1 + f2 - f3 - f4); % [N*m], control moment
M = (R/sqrt(2)) * (-f1 + f2 + f3 - f4); % [N*m], control moment
N = k * (f1 - f2 + f3 - f4); % [N*m], control moment
G_control_body = [L; M; N]; % [N*m], total control moment
G_body = G_aerodynamic_body + G_control_body + 0; % [N*m], total moments on vehicle
```

Calculate the derivatives.

```
xDot = (u * cos(theta) * cos(psi)) + (v * (sin(phi) * sin(theta) * cos(psi) - cos(phi)...
    * sin(psi))) + w * (cos(phi) * sin(theta) * cos(psi) + sin(phi) * sin(psi));
yDot = (u * cos(theta) * sin(psi)) + (v * (sin(phi) * sin(theta) * sin(psi)...
    + cos(phi) * cos(psi))) + w * (cos(phi) * sin(theta) * sin(psi) - sin(phi) * cos(psi));
zDot = (-u * sin(theta))+(v * sin(phi) * cos(theta)) + (w * cos(phi) * cos(theta));

uDot = -g * theta;
vDot = g * phi;
```

```
wDot = 0;

psiDot = (q * sin(phi) + r * cos(phi)) * sec(theta);
phiDot = p;
thetaDot = q;

pDot = (1 / I_x) * (G_body(1));
qDot = (1 / I_y) * (G_body(2));
rDot = (1 / I_z) * (G_body(3));

derivativeState = [xDot yDot zDot pDot qDot rDot psiDot phiDot thetaDot uDot vDot wDot].';

end
```

ODE45 function for the motion of a quadcopter

Detailed explanation of this function.

```
function derivativeState = quadcopterSimulationNonLinFC(t,State)
```

Declare global and local constants.

```
global g I_x I_y I_z m R fb f1 f2 f3 f4 L M N G_body g_body
eta = 1 * 10 ^ (-3); % [N/(m/s)^2], drag per velocity squared
alpha = 2 * 10 ^ (-6); % [N*m/(rad/s)^2], moment per rate of rotation squared
k = 0.0024; % [N*m/N], moment per force of motors
Fc = 0.004;
```

Extract variable from state vector.

```
x = State(1);
y = State(2);
z = State(3);
p = State(4);
q = State(5);
r = State(6);
psi = State(7);
phi = State(8);
theta = State(9);
u = State(10);
v = State(11);
w = State(12);
```

Calculate the forces and moments.

```
Thrust = (-eta * sqrt(u^2 + v^2 + w^2) * w) + (m * g);
f1 = -Thrust / 4;
f2 = -Thrust / 4;
f3 = -Thrust / 4;
f4 = -Thrust / 4;

A_aerodynamic_body = -eta * sqrt(u^2 + v^2 + w^2) * [u; v; w]; % [N], aerodynamic forces on veh
```

```
A_control_body = [0; 0; f1 + f2 + f3 + f4]; % [N], control forces on vehicle
fb = m * g_body + A_aerodynamic_body + A_control_body; % [N], vector of forces on quadcopter

G_aerodynamic_body = -alpha * sqrt(p^2 + q^2 + r^2) * [p; q; r]; % [N*m], aerodynamic moments c
L = (R/sqrt(2)) * (f1 + f2 - f3 - f4) - Fc * p; % [N*m], control moment
M = (R/sqrt(2)) * (-f1 + f2 + f3 - f4) - Fc * q; % [N*m], control moment
N = k * (f1 - f2 + f3 - f4) - Fc * r; % [N*m], control moment
G_control_body = [L; M; N]; % [N*m], total control moment
G_body = G_aerodynamic_body + G_control_body + 0; % [N*m], total moments on vehicle
```

Calculate the derivatives.

```
xDot = (u * cos(theta) * cos(psi)) + (v * (sin(phi) * sin(theta) * cos(psi) - cos(phi)...
    * sin(psi))) + w * (cos(phi) * sin(theta) * cos(psi) + sin(phi) * sin(psi));
yDot = (u * cos(theta) * sin(psi)) + (v * (sin(phi) * sin(theta) * sin(psi)...
    + cos(phi) * cos(psi))) + w * (cos(phi) * sin(theta) * sin(psi) - sin(phi) * cos(psi));
zDot = (-u * sin(theta))+(v * sin(phi) * cos(theta)) + (w * cos(phi) * cos(theta));

uDot = (fb(1) / m) - (q * w) + (r * v);
vDot = (fb(2) / m) - (r * u) + (p * w);
wDot = (fb(3) / m) - (p * v) + (q * u);

psiDot = (q * sin(phi) + r * cos(phi)) * sec(theta);
phiDot = p + (q * sin(phi) + r * cos(phi)) * tan(theta);
thetaDot =  (q * cos(phi)) - (r * sin(phi));

pDot = (1 / I_x) * (G_body(1) - ((q * r) * (I_z - I_y)));
qDot = (1 / I_y) * (G_body(2) - ((r * p) * (I_x - I_z)));
rDot = (1 / I_z) * (G_body(3) - ((p * q) * (I_y - I_x)));

derivativeState = [xDot yDot zDot pDot qDot rDot psiDot phiDot thetaDot uDot vDot wDot].';

end
```