



AIRCRAFT FLIGHT DYNAMICS, CONTROL AND SIMULATION

Using MATLAB and SIMULINK: Cases and Algorithm Approach

SINGGIIH SATRIO WIBOWO

PREFACE

This book is written for students and engineers interesting in flight control design, analysis and implementation. This book is written during preparation of Matlab and Simulink course in UNIKL-MIAT (University of Kuala Lumpur-Malaysian Institute of Aviation Technology) in third week of February 2007. Although this book is still in preparation, I hope that this book will be useful for the readers.

I wish to express my great appreciation to Professor Said D. Jenie for his support. I wish to acknowledge Mr. Kharil Anuar and Mr. Shahrul Ahmad Shah of MIAT for their invitation to the author to give Matlab course in MIAT during the period of 26 February to 2 March 2007. I also wish to acknowledge the support of my colleagues at Institut Teknologi Bandung (ITB): Javensius Sembiring and Yazdi I. Jenie, and also my friends at Badan Pengkajian dan Penerapan Teknologi (BPPT): Dewi Hapsari, Dyah Jatiningrum and Nina Kartika. No words can express the thanks I owe to my parents: Ibunda Sulasmi and Ayahanda Satrolan, and my family for their continuous support through out my life. Finally and the most importantly, I would like to thank The Highest Sweetheart Allah Almighty, The Creator and The Owner of the universe.

Kuala Lumpur, 25 February 2007

Singgih Satrio Wibowo

CONTENTS

Preface	1
Contents	2
List Of Figures	5
List of Tables	7
1 Aircraft Dynamics and Kinematics	9
1.1 Coordinate Systems and Transformation	10
1.1.1 Local Horizon Coordinate Reference System	10
1.1.2 Body Coordinate Reference System	10
1.1.3 Wind Coordinate System	12
1.1.4 Kinematics Equation	15
1.1.5 Direction Cosine Matrix	16
1.1.6 Quaternions	17
1.2 Aircraft equations of motion	21
1.2.1 Translational Motion	21
1.2.2 Angular Motion.....	23
1.2.3 Force and Moment due to Earth's Gravity	25
1.2.4 Aerodynamic Forces and Moments.....	26
1.2.5 Linearization of Equations of Motion	27
1.1 Matlab and Simulink Tools for Flight Dynamics Simulation	30
2 Flight Control	31
2.1 Attitude and Altitude Control using Root Locus Anlysis.....	32
2.2 Optimal Path-Tracking Control for Autonomous Unmanned Helicopter Using Linear Quadratic Regulator	33
2.2.1 Linearized Model	34
2.2.2 Modified Linearized Model	37

2.2.3	Path Generator	39
2.2.4	Path-Tracking Controller Design.....	43
2.2.5	Matlab and Simulink Implementation.....	46
2.2.6	Numerical Results	54
2.2.7	Analysis and Discussion of the Results	63
2.3	Coordinated Turn Using Linear Quadratic Regulator	65
2.3.1	State-Space Equations for an Airframe	65
2.3.2	Problem Definition	65
2.3.3	Matlab and Simulink Implementation.....	66
2.3.4	Results.....	69
2.3.5	Analysis and Discussion of the Results	70
2.4	Adaptive Control for Yaw Damper and Coordinated Turn	71
2.4.1	Yaw Damper and Coordinated Turn: Definition	71
2.4.2	Model Reference Adaptive System	71
2.4.3	State-Space Model of XX-100 Aircraft.....	72
2.4.4	Matlab and Simulink Implementation.....	72
2.4.5	Results.....	72
2.4.6	Discussion of The Results	73
3	Flight Simulation	74
3.1	Matlab and Simulink tool for simulation	75
3.1.1	Matlab command for simulation purpose	75
3.1.2	Simulink toolbox for simulation purpose	75
3.2	Virtual Reality, an advance tool for visualization	76
3.2.1	Introduction to Virtual Reality toolbox: a user guide.....	76
3.2.2	Virtual Reality for transport aircraft.....	88
3.3	Simulation of Aircraft Dynamics: a VirtueAir transport craft	89
	Appendix A	90

Appendix B	93
References	99

LIST OF FIGURES

Figure 1-1 Local horizon coordinate system.....	10
Figure 1-2 Body-coordinate system.....	11
Figure 1-3 Aircraft attitude with respect to local horizon frame: Euler angles.....	12
Figure 1-4 Wind-axes system and its relation to Body axes.....	13
Figure 1-5 Aerodynamic lift and drag	14
Figure 2-1 A small-scale unmanned helicopter, Yamaha R-50.....	33
Figure 2-2 Dimension of the Yamaha R-50 Helicopter	34
Figure 2-3 The complete state-space form of R-50 dynamics.....	35
Figure 2-4 Trajectory for example 1, circular	40
Figure 2-5 Velocity profile for example 1	41
Figure 2-6 Trajectory for example 2, rectangular.....	41
Figure 2-7 Velocity profile for example 2	42
Figure 2-8 Trajectory for example 3, spiral	42
Figure 2-9 Velocity profile for example 3	43
Figure 2-10 Path tracking controller model.....	49
Figure 2-11 Path generator block	49
Figure 2-12 Earth to inertial velocity transform block	50
Figure 2-13 Optimal controller block.....	50
Figure 2-14 Yamaha R50 dynamics model block	50
Figure 2-15 Body to inertial transform block	51
Figure 2-16 Inertial to Earth transform block.....	51
Figure 2-17 Write to file block.....	51
Figure 2-18 Flight trajectory geometry.....	55
Figure 2-19 Trajectory history	55
Figure 2-20 Velocity history.....	56
Figure 2-21 Control input history	56
Figure 2-22 Attitude history	57
Figure 2-23 Trajectory error history	57
Figure 2-24 Flight trajectory geometry.....	58
Figure 2-25 Trajectory history	58
Figure 2-26 Velocity history.....	59
Figure 2-27 Control input history	59
Figure 2-28 Attitude history	60
Figure 2-29 Trajectory error history	60
Figure 2-30 Flight trajectory geometry.....	61
Figure 2-31 Trajectory history	61

Figure 2-32 Velocity history	62
Figure 2-33 Control input history	62
Figure 2-34 Attitude history	63
Figure 2-35 Trajectory error history	63
Figure 2-36 A Body Coordinate Frame for an Aircraft [16]	65
Figure 2-37 Simulink diagram of coordinated turn	67
Figure 2-38 Write to file block	67
Figure 2-39 Attitude history	69
Figure 2-40 Tracking error history	70
Figure 2-41 Control input history	70
Figure 2-42 Block diagram for Turn Coordinator system	71
Figure 2-43 Block diagram for Model Reference Adaptive System	72
Figure 3-1 The 3D AutoCAD model of XW aircraft	77
Figure 3-2 The 3D AutoCAD model of lake and hill	78
Figure 3-3 The V-Realm Builder window	78
Figure 3-4 The 3D studio model of XW craft after imported into the V-Realm Builder	79
Figure 3-5 The 3D studio model of XW craft after a background is added	79
Figure 3-6 Adding four 'Transform'	80
Figure 3-7 Renaming the four 'Transform' and moving the 'Wise'	80
Figure 3-8 Adding a dynamic observer	80
Figure 3-9 Edit rotation (orientation) of the observer	81
Figure 3-10 Edit position of the observer	81
Figure 3-11 Edit description of the observer	82
Figure 3-12 An example of an observer	82
Figure 3-13 An example of an observer, Right Front Observer	82
Figure 3-14 Final results of the Virtual World	83
Figure 3-15 A new SIMULINK model with VR Sink	83
Figure 3-16 Parameter window of VR Sink	84
Figure 3-17 Parameter window of VR Sink after loading "wise8craftVR.wrl"	84
Figure 3-18 The VR visualization window of WiSE-8 craft	85
Figure 3-19 The VR parameter after VRML Tree editing	86
Figure 3-20 The VR Sink after VR parameter editing	87
Figure 3-21 The VR Transform subsystem	88

LIST OF TABLES

Table 1 Physical Parameter of The Yamaha R-50	34
Table 2 Parameter values of matrix A	35
Table 3 Parameter values of matrix B	37

1 AIRCRAFT DYNAMICS AND KINEMATICS

Nature of Aircraft dynamics and kinematics in three-dimensional (3D) space can be described by a set of Equations of Motion (EOM), which contains six degrees of freedom: three translational modes and three rotational modes. In the equations, it needs to define the forces and moments acting on the vehicle since it is the factors responsible for the motion. Therefore, the modeling of the forces and moments is a must. The mathematical model of forces and moments include the aerodynamic, propulsion system and gravity. These models will be discussed in detail in this chapter.

In this chapter, first we briefly overview the coordinate systems that used as the reference frame for the description of aircraft motion. Then, a complete nonlinear model of the aircraft motion will be discussed briefly.

1.1 COORDINATE SYSTEMS AND TRANSFORMATION

A number of coordinate systems will be employed here to be used as a reference for the motion of the aircraft in three-dimensional space,

- Local horizon-coordinate system
- Body-coordinate system
- Wind-coordinate system

1.1.1 LOCAL HORIZON COORDINATE REFERENCE SYSTEM

The local horizon coordinate system is also called the tangent-plane; it is a Cartesian coordinate system. Its origin is located on a pre-selected point of interest and its x_h , y_h , z_h axes align with the north, east and down direction respectively as shown in Figure 1-1.

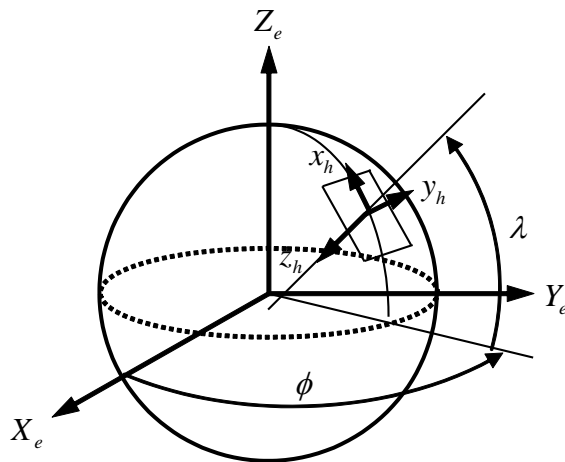


FIGURE 1-1 LOCAL HORIZON COORDINATE SYSTEM

For simulation purposes, the local horizon local will be used as reference (inertial) frame. It is correct since the most of aircraft is flying in low altitude and range relative to the earth surface.

1.1.2 BODY COORDINATE REFERENCE SYSTEM

The body coordinate system is a special coordinate system which represents the aircraft body. Its origin is attached to the aircraft center of gravity, see Figure 1-2. The positive x_b axis lies along the symmetrical axis of the aircraft in the forward direction, its positive y_b axis is perpendicular to the symmetrical axis of the aircraft to the right direction, and the positive z_b is perpendicular to the $ox_b y_b$ plane making the right hand orientation.

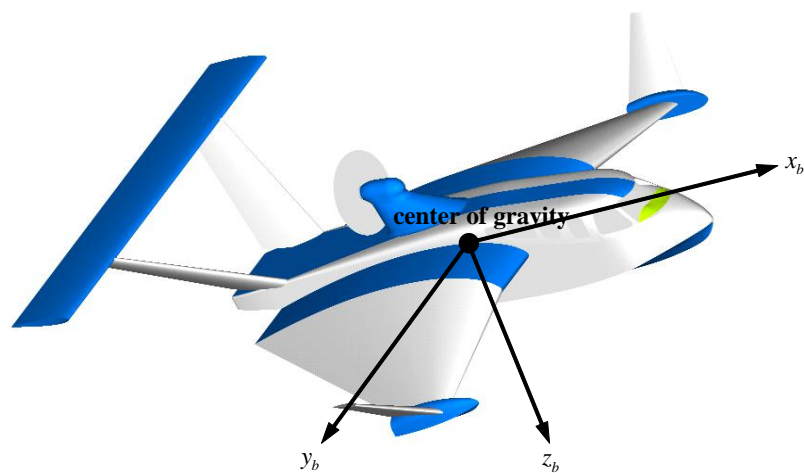


FIGURE 1-2 BODY-COORDINATE SYSTEM

The transformation of body axes to the local horizon frame is carried out using Euler angle orientation procedures. The orientation of the body axes system to the local horizon axes system is expressed by Euler angles as shown in Figure 1-3.

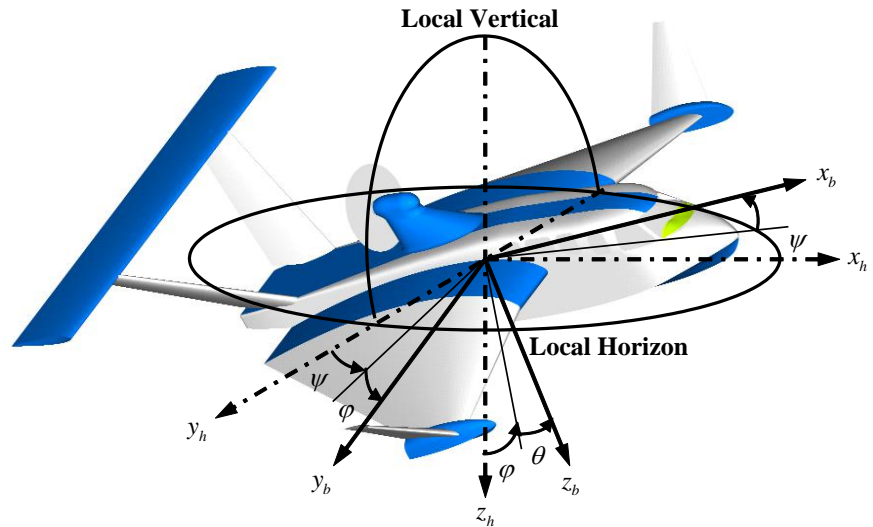


FIGURE 1-3 AIRCRAFT ATTITUDE WITH RESPECT TO LOCAL HORIZON FRAME: EULER ANGLES

The transformation of local horizon coordinate system to body coordinate system can be expressed as [2]

$$C_b^h = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ \sin \varphi \sin \theta \cos \psi - \cos \varphi \sin \psi & \sin \varphi \sin \theta \sin \psi + \cos \varphi \cos \psi & \sin \varphi \cos \theta \\ \cos \varphi \sin \theta \cos \psi + \sin \varphi \sin \psi & \cos \varphi \sin \theta \sin \psi - \sin \varphi \cos \psi & \cos \varphi \cos \theta \end{bmatrix} \quad (1-1)$$

The above formula is very useful for determining the orientation of the aircraft with respect to the earth surface. This matrix is an orthogonal class of matrix, meaning that its inverse can be obtained by transposing the matrix above as $C_h^b = [C_b^h]^{-1} = [C_b^h]^T$.

1.1.3 WIND COORDINATE SYSTEM

Wind coordinate system represents the aircraft velocity vector. This frame defines the flight path of the aircraft. The term ‘wind’ used here is relative wind flowing through the aircraft body as the aircraft fly in the air [2].

Its origin is attached to the center of gravity while its axes define the direction and the orientation of flight path. The positive x_w axis coincides to the aircraft velocity vector V . The z_w axis lies on the symmetrical plane of the aircraft, perpendicular to the x_w axis and positive downward. And the last, positive y_w axis is perpendicular to the ox_wz_w plane obeying the right-hand orientation. These axes definition are shown in Figure 1-4.

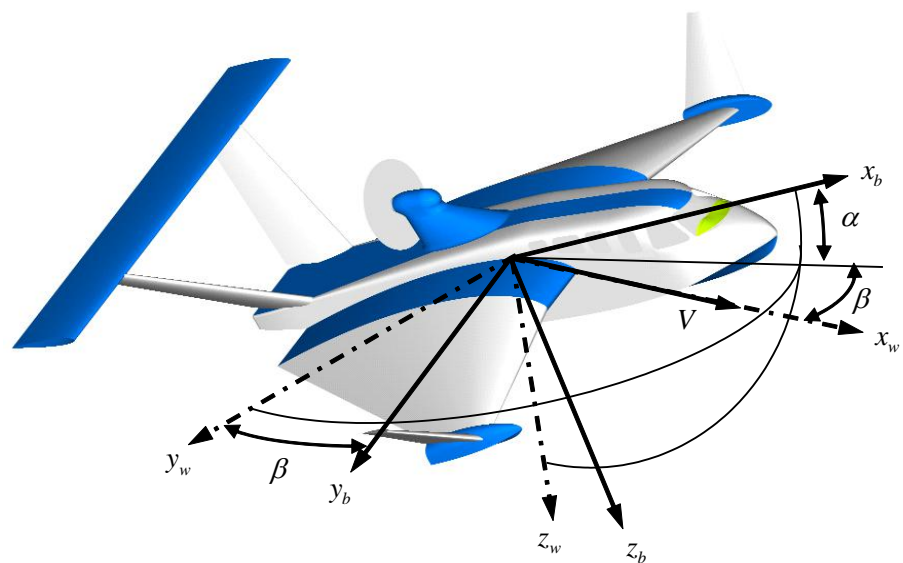


FIGURE 1-4 WIND-AXES SYSTEM AND ITS RELATION TO BODY AXES

Wind axes system can be transformed to the body axes system using the following matrix of transformation,

$$C_b^w = \begin{bmatrix} \cos \alpha \cos \beta & -\cos \alpha \sin \beta & -\sin \alpha \\ \sin \beta & \cos \beta & 0 \\ \sin \alpha \cos \beta & -\sin \alpha \sin \beta & \cos \alpha \end{bmatrix} \quad (1-2)$$

This equation is useful for transforming the aerodynamic lift and drag forces to body axes system. As can be seen in Figure 1-4, the

aerodynamic lift vector is along the negative z_w axis while the aerodynamic drag is along the negative x_w axis. Since the equations of motion are derived in body axes system, it needs to express all forces and moments which acting on the aircraft in the body axes. Therefore the aerodynamic lift and drag vectors should be transformed from wind axes to the body axes.

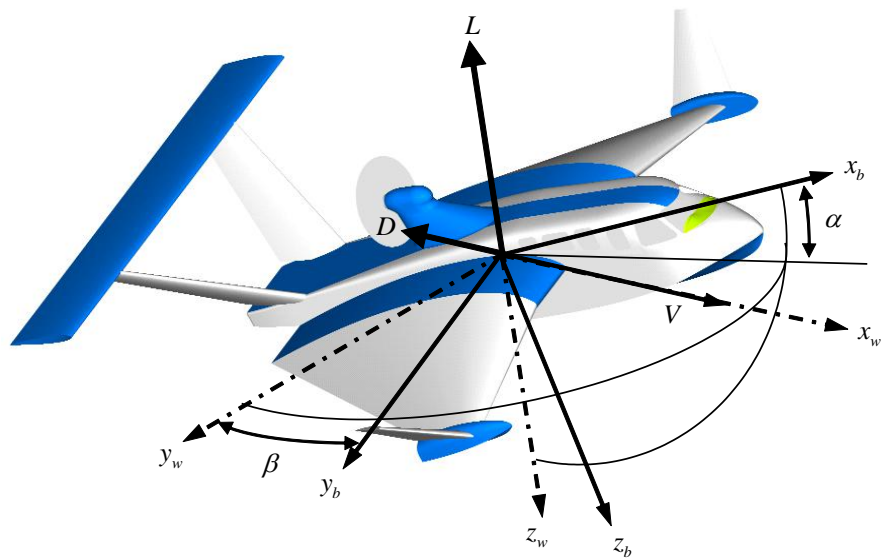


FIGURE 1-5 AERODYNAMIC LIFT AND DRAG

Using Equation (1-2), Aerodynamic lift and drag can be transformed to body axes system by the following relation

$$\begin{Bmatrix} F_{Ax} \\ F_{Ay} \\ F_{Az} \end{Bmatrix} = \begin{bmatrix} \cos \alpha \cos \beta & -\cos \alpha \sin \beta & -\sin \alpha \\ \sin \beta & \cos \beta & 0 \\ \sin \alpha \cos \beta & -\sin \alpha \sin \beta & \cos \alpha \end{bmatrix} \begin{Bmatrix} -D \\ 0 \\ -L \end{Bmatrix} \quad (1-3)$$

Similarly, after dividing Equation (1-3) by $\frac{1}{2} \rho V_T^2 S$, the aerodynamic coefficients can be expressed as

$$\begin{Bmatrix} C_X \\ C_Y \\ C_Z \end{Bmatrix} = \begin{bmatrix} \cos \alpha \cos \beta & -\cos \alpha \sin \beta & -\sin \alpha \\ \sin \beta & \cos \beta & 0 \\ \sin \alpha \cos \beta & -\sin \alpha \sin \beta & \cos \alpha \end{bmatrix} \begin{Bmatrix} -C_D \\ 0 \\ -C_L \end{Bmatrix} \quad (1-4)$$

Equation (1-4) will be used for transforming aerodynamic lift and drag coefficients to body axes aerodynamic coefficients C_X , C_Y , and C_Z .

The translational velocity can also be transformed into the body axes system as follows:

$$\begin{Bmatrix} U \\ V \\ W \end{Bmatrix} = \begin{bmatrix} \cos \alpha \cos \beta & -\cos \alpha \sin \beta & -\sin \alpha \\ \sin \beta & \cos \beta & 0 \\ \sin \alpha \cos \beta & -\sin \alpha \sin \beta & \cos \alpha \end{bmatrix} \begin{Bmatrix} V_T \\ 0 \\ 0 \end{Bmatrix} \quad (1-5)$$

$$= \begin{Bmatrix} V_T \cos \alpha \cos \beta \\ V_T \sin \beta \\ V_T \sin \alpha \cos \beta \end{Bmatrix}$$

in which the total velocity V_T is defined as $V_T = \sqrt{U^2 + V^2 + W^2}$. Angle of attack α , and angle of sideslip β can be derived from equation (2-9) as follows:

$$\alpha = \arctan\left(\frac{W}{U}\right)$$

$$\beta = \arcsin\left(\frac{V}{V_T}\right) \quad (1-6)$$

Equation (2-10) will also be used in the simulation for calculating angle of attack and sideslip angle from body axes velocity.

1.1.4 KINEMATICS EQUATION

Kinematics equation shows the relation of Euler angles and angular velocity $\boldsymbol{\omega}_b = [P \ Q \ R]^T$. The physical definition of Euler angles can be seen in Figure 1-3. The kinematics equations are listed as follows:

$$\begin{aligned}\dot{\phi} &= P + Q \sin \phi \tan \theta + R \cos \phi \tan \theta \\ \dot{\theta} &= Q \cos \phi - R \sin \phi \\ \dot{\psi} &= Q \frac{\sin \phi}{\cos \theta} + R \frac{\cos \phi}{\cos \theta}\end{aligned}\tag{1-7}$$

The above equation can be rewritten in the form of matrix as

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{bmatrix} \begin{Bmatrix} P \\ Q \\ R \end{Bmatrix}\tag{1-8}$$

Equations (2-2) and (2-3) are used to obtain the Euler angles from the angular velocity P , Q , and R . But the above equations have disadvantages, i.e. can be singular for $\theta = \pm 90$ degrees. It motivates to use another way that can avoid the singularity. This can be done using quaternion which will be discussed in the next section.

1.1.5 DIRECTION COSINE MATRIX

Intersection angle θ_i of any two vectors in three-dimensional (3D) space, denoted by \mathbf{r}_1 and \mathbf{r}_2 , can be found by the inner product relationship:

$$\theta_i = \arccos \left[\frac{\mathbf{r}_1 \cdot \mathbf{r}_2}{|\mathbf{r}_1| |\mathbf{r}_2|} \right] \quad (1-9)$$

Using above idea, the transformation coordinate from local horizon axes system $(\mathbf{i}_h, \mathbf{j}_h, \mathbf{z}_h)$ to body axes system $(\mathbf{i}_b, \mathbf{j}_b, \mathbf{z}_b)$ can be cast into the matrix form [48]:

$$\begin{aligned} DCM &= \begin{bmatrix} \mathbf{i}_h \cdot \mathbf{i}_b & \mathbf{i}_h \cdot \mathbf{j}_b & \mathbf{i}_h \cdot \mathbf{z}_b \\ \mathbf{j}_h \cdot \mathbf{i}_b & \mathbf{j}_h \cdot \mathbf{j}_b & \mathbf{j}_h \cdot \mathbf{z}_b \\ \mathbf{z}_h \cdot \mathbf{i}_b & \mathbf{z}_h \cdot \mathbf{j}_b & \mathbf{z}_h \cdot \mathbf{z}_b \end{bmatrix} \\ &= \begin{bmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ s\phi s\theta c\psi - c\phi s\psi & s\phi s\theta s\psi + c\phi c\psi & s\phi c\theta \\ c\phi s\theta c\psi + s\phi s\psi & c\phi s\theta s\psi - s\phi c\psi & c\phi c\theta \end{bmatrix} \end{aligned} \quad (1-10)$$

where symbol $s(\cdot) = \sin(\cdot)$ and $c(\cdot) = \cos(\cdot)$ are used for abbreviation. Equation (1-10) is identical to Equation (1-1). Therefore the term *DCM* will be used together with the transformation matrix C_b^h in the simulation.

1.1.6 QUATERNIONS

Quaternions were discovered by Sir William Hamilton in 1843. He used quaternion for extensions of vector algebras to satisfy the properties of division rings (roughly, quotients exist in the same domain as the operands). It has been widely discussed as interesting topic in algebra and for its amazing applicability in dynamics.

The following paragraphs discuss the application of Quaternion starting with its definition while more detail discussion will be presented in Appendix C. Quaternion is define as

$$\mathbf{q} = 1 \cdot q_0 + \mathbf{i} \cdot q_1 + \mathbf{j} \cdot q_2 + \mathbf{k} \cdot q_3 = [q_0 \quad q_1 \quad q_2 \quad q_3]^T \quad (1-11)$$

where q_0, q_1, q_2, q_3 are reals, 1 is the multiplicative identity element, and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are symbolic elements having the properties:

$$\begin{aligned} \mathbf{i}^2 = -1, \quad \mathbf{j}^2 = -1, \quad \mathbf{k}^2 = -1 \\ \mathbf{ij} = \mathbf{k} \quad \mathbf{ji} = -\mathbf{k} \\ \mathbf{jk} = \mathbf{i} \quad \mathbf{kj} = -\mathbf{i} \\ \mathbf{ki} = \mathbf{j} \quad \mathbf{ik} = -\mathbf{j} \end{aligned} \quad (1-12)$$

The time-derivative of the quaternion can be expressed as follows:

$$\begin{aligned} \dot{\mathbf{q}} &= [\boldsymbol{\psi} - K\boldsymbol{\varepsilon}] \mathbf{q} \\ &= \frac{1}{2} \begin{bmatrix} 0 & R & -Q & P \\ -R & 0 & P & Q \\ Q & -P & 0 & R \\ -P & -Q & -R & 0 \end{bmatrix} \begin{Bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{Bmatrix} - K\boldsymbol{\varepsilon} \begin{Bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{Bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} q_3 & -q_2 & q_1 \\ q_2 & q_3 & -q_0 \\ -q_1 & q_0 & q_3 \\ -q_0 & -q_1 & -q_2 \end{bmatrix} \begin{Bmatrix} P \\ Q \\ R \end{Bmatrix} - K\boldsymbol{\varepsilon} \begin{Bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{Bmatrix} \\ &= \mathbf{Q}\boldsymbol{\omega}_b - K\boldsymbol{\varepsilon}\mathbf{q} \end{aligned} \quad (1-13)$$

where $\boldsymbol{\varepsilon} = 1 - (q_0^2 + q_1^2 + q_2^2 + q_3^2)$ is an error coefficient.

Obviously, integrating equation (1-13) is much more efficient than (1-3) because it does not involve computationally expensive trigonometric functions. This integration can be evaluated using the following relation:

$$\mathbf{q}(t) = \mathbf{q}_0 + \int_{t_0}^t \dot{\mathbf{q}} dt \quad (1-14)$$

where $\mathbf{q}(t)$ denotes quaternion at time t and \mathbf{q}_0 is initial quaternion calculated from initial Euler angles using Eqn. (1-17).

The rotational transformation matrix can be directly found with quaternion:

$$\begin{aligned} C_b^h &= DCM \\ &= \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{bmatrix} \end{aligned} \quad (1-15)$$

Euler angles can be determined from the quaternion by comparing Eqn. (2-15) to Eqn. (2-1) which yields

$$\begin{aligned} \varphi &= \arctan \left[\frac{2(q_2q_3 + q_0q_1)}{q_0^2 + q_3^2 - q_1^2 - q_2^2} \right] \\ \theta &= \arcsin \left[-2(q_1q_3 - q_0q_2) \right] \\ \psi &= \arctan \left[\frac{2(q_1q_2 + q_0q_3)}{q_0^2 + q_1^2 - q_2^2 - q_3^2} \right] \end{aligned} \quad (1-16)$$

This quaternion can also be expressed in terms of Euler angles as [8]:

$$\begin{aligned}
 q_0 &= \pm \left(\cos \frac{\varphi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\varphi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \right) \\
 q_1 &= \pm \left(\sin \frac{\varphi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} - \cos \frac{\varphi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \right) \\
 q_2 &= \pm \left(\cos \frac{\varphi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\varphi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} \right) \\
 q_3 &= \pm \left(\cos \frac{\varphi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} - \sin \frac{\varphi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} \right)
 \end{aligned} \tag{1-17}$$

The above equations will be used in the simulation which will be conducted in this book.

1.2 AIRCRAFT EQUATIONS OF MOTION

The equations of motion are derived based on Newton law. They were first derived by Euler, a great mathematician. It is the reason why the equations of motion are dedicated to Newton and Euler.

The solutions of the complete equations of motion provide the characteristics of motion of any solid body in three-dimensional space, three translational and three angular motions. Therefore they called the six degree of freedom (6-DOF) equations of motion. These equations are very general and apply for all rigid bodies, e.g. aircrafts, rockets and satellites.

The 6-DOF equations of motion consists a set of nonlinear first ordinary differential equations (ODES). They express the motions of the aircraft in terms of external forces and moments, which can be subdivided in a number of categories such as aerodynamics, control surface, propulsion system, and gravity. In this section, the equations of motion will be presented along with all relevant force and moment equations and a large number of output equations of which some are needed to calculate these forces and moments.

1.2.1 TRANSLATIONAL MOTION

Applying the second law of Newton, the net forces acting on the airplane can be found by adding up the force acting on the all parts of the airplane as follows:

$$\sum \Delta \mathbf{F} = \frac{d(m\mathbf{V}_I)}{dt} = m \left(\frac{d\mathbf{V}_b}{dt} + \boldsymbol{\omega}_b \times \mathbf{V}_b \right) + \dot{m}\mathbf{V}_b \quad (1-18)$$

where $\sum \Delta \mathbf{F} = [F_x \quad F_y \quad F_z]^T$ is total force vector along x_b , y_b and z_b axes respectively, $\mathbf{V}_b = [U \quad V \quad W]^T$ is velocity vector coordinated at the body axes frame and $\boldsymbol{\omega}_b = [P \quad Q \quad R]^T$ denotes angular

velocity vector of the aircraft with respect to the inertial space coordinated at the body axes system. Upon decomposition, the resulting three scalar force equations become:

$$\begin{aligned} F_x &= m(\dot{U} + QW - RV) + \dot{m}U \\ F_y &= m(\dot{V} + RU - PW) + \dot{m}V \\ F_z &= m(\dot{W} + PV - QU) + \dot{m}W \end{aligned} \quad (1-19)$$

The above equation then used for calculating the translational acceleration that can be expressed in the following equation:

$$\begin{aligned} \dot{U} &= \frac{F_x - \dot{m}U}{m} - QW + RV \\ \dot{V} &= \frac{F_y - \dot{m}V}{m} - RU + PW \\ \dot{W} &= \frac{F_z - \dot{m}W}{m} - PV + QU \end{aligned} \quad (1-20)$$

The term $(\sum \Delta \mathbf{F} - \dot{m} \mathbf{V}_b)/m$ is defined as translational acceleration \mathbf{a}_b , $= [a_x \ a_y \ a_z]^T = [(F_x - \dot{m}u)/m \ (F_y - \dot{m}v)/m \ (F_z - \dot{m}w)/m]^T$. Forces occurred in (2-19) and (2-20) are caused by the aerodynamics, control surface, propulsion system and Earth's gravity. Hence it can be written as follows:

$$\begin{aligned} F_x &= F_{A_x} + F_{C_x} + F_{P_x} + F_{G_x} \\ F_y &= F_{A_y} + F_{C_y} + F_{P_y} + F_{G_y} \\ F_z &= F_{A_z} + F_{C_z} + F_{P_z} + F_{G_z} \end{aligned} \quad (1-21)$$

where F_{A_x} denotes aerodynamic force acting along x_b axis, F_{H_x} is hydrodynamic force acting along x_b axis, F_{T_x} denotes the propulsion force acting along x_b axis F_{G_x} denotes gravity force acting along x_b axis, and so on.

1.2.2 ANGULAR MOTION

Angular motion of the aircraft is also derived based on the second law of Newton. The net moment acting on the airplane can be found by adding up the moments acting on the all parts of the airplane as:

$$\sum \Delta \mathbf{M} = \frac{d\mathbf{H}_I}{dt} = \mathbf{I}\dot{\boldsymbol{\omega}}_b + \boldsymbol{\omega}_b \times (\mathbf{I}\boldsymbol{\omega}_b) + \dot{\mathbf{I}}\boldsymbol{\omega}_b \quad (1-22)$$

where $\sum \Delta \mathbf{M} = [M_x \quad M_y \quad M_z]^T$ is total moment vector along x_b , y_b and z_b axes respectively, $\boldsymbol{\omega}$ denotes the angular velocity of the aircraft as mentioned before and \mathbf{I} denotes the inertia tensor of the aircraft defined as

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -J_{yx} & -J_{zx} \\ -J_{xy} & I_{yy} & -J_{zy} \\ -J_{xz} & -J_{yz} & I_{zz} \end{bmatrix} \quad (1-23)$$

The angular acceleration can be evaluated using Equation (2-22) as

$$\dot{\boldsymbol{\omega}}_b = \mathbf{I}^{-1} \left(\sum \Delta \mathbf{M} - \boldsymbol{\omega}_b \times (\mathbf{I}\boldsymbol{\omega}_b) - \dot{\mathbf{I}}\boldsymbol{\omega}_b \right) \quad (1-24)$$

Here, \mathbf{I}^{-1} is the inverse of the inertia tensor as shown in Equation (2-24). This inverse has a relatively simple form [8]:

$$\mathbf{I}^{-1} = \frac{1}{\Delta} \begin{bmatrix} k_{11} & k_{21} & k_{31} \\ k_{12} & k_{22} & k_{32} \\ k_{13} & k_{23} & k_{33} \end{bmatrix} \quad (1-25)$$

where

$$\begin{aligned} k_{11} &= (I_{yy}I_{zz} - J_{yz}^2)/\Delta & k_{12} &= k_{21} = (J_{yz}J_{xz} + J_{xy}I_{zz})/\Delta \\ k_{22} &= (I_{zz}I_{xx} - J_{xz}^2)/\Delta & k_{13} &= k_{31} = (J_{xy}J_{yz} + J_{xz}I_{yy})/\Delta \\ k_{33} &= (I_{xx}I_{yy} - J_{xy}^2)/\Delta & k_{23} &= k_{32} = (J_{xy}J_{xz} + J_{yz}I_{xx})/\Delta \end{aligned} \quad (1-26)$$

and

$$\Delta = I_{xx}I_{yy}I_{zz} - 2J_{xy}J_{yz}J_{xz} - I_{xx}J_{yz}^2 - I_{yy}J_{xz}^2 - I_{zz}J_{xy}^2 \quad (1-27)$$

For conventional aircraft which is symmetrical to ox_bz_b plane, the cross inertial products are very small and can be assumed to be zero ($J_{xy} = 0$ and $J_{yz} = 0$). Under this condition, Eqn. (2-27) can be simplified as

$$\Delta = I_{xx}I_{yy}I_{zz} - I_{yy}J_{xz}^2 \quad (1-28)$$

By assuming the inertia tensor is constant which implies $\dot{\mathbf{I}} = 0$, Eqn. (2-24) can be decomposed as:

$$\begin{aligned}
 \dot{P} &= \frac{M_X I_{zz} + M_Z J_{xz}}{\Delta} + \frac{J_{xz} (I_{xx} - I_{yy} + I_{zz}) PQ}{\Delta} - \frac{[I_{zz} (I_{zz} - I_{yy}) + J_{xz}^2] QR}{\Delta} \\
 \dot{Q} &= \frac{M_Y}{I_{yy}} + \frac{(I_{zz} - I_{xx}) PR}{I_{yy}} - \frac{J_{xz} (P^2 - R^2)}{I_{yy}} \\
 \dot{R} &= \frac{M_X J_{xz} + M_Z I_{xx}}{\Delta} + \frac{[I_{xx} (I_{xx} - I_{yy}) + J_{xz}^2] PQ}{\Delta} - \frac{J_{xz} (I_{xx} - I_{yy} + I_{zz}) QR}{\Delta}
 \end{aligned} \tag{1-29}$$

The moments (M_X , M_Y and M_Z) can also be expressed in terms of aerodynamic, control surface and propulsion moments as those for the forces,

$$\begin{aligned}
 M_X &= M_{A_x} + M_{C_x} + M_{P_x} \\
 M_Y &= M_{A_y} + M_{C_y} + M_{P_y} \\
 M_Z &= M_{A_z} + M_{C_z} + M_{P_z}
 \end{aligned} \tag{1-30}$$

where M_{A_x} denotes aerodynamic moment which respect to x_b axis, M_{C_x} is control surface moment which respect to x_b axis, M_{P_x} denotes the propulsion moment which respect to x_b axis, and so on.

1.2.3 FORCE AND MOMENT DUE TO EARTH'S GRAVITY

The gravity force vector can be decomposed along the body axes system as:

$$\begin{aligned}
 F_{G_x} &= -mg \sin \theta \\
 F_{G_y} &= mg \sin \varphi \cos \theta \\
 F_{G_z} &= mg \cos \varphi \cos \theta
 \end{aligned} \tag{1-31}$$

or can be written as

$$\begin{Bmatrix} F_{G_x} \\ F_{G_y} \\ F_{G_z} \end{Bmatrix} = C_b^h \begin{Bmatrix} 0 \\ 0 \\ mg \end{Bmatrix} = DCM \begin{Bmatrix} 0 \\ 0 \\ mg \end{Bmatrix} \quad (1-32)$$

The gravity force produces zero moment because it is acting on the center of gravity. Equation (2-32) is the gravity force equation which will be used in the simulation.

1.2.4 AERODYNAMIC FORCES AND MOMENTS

Aerodynamic forces and moments are function of some parameters. They can be written as:

$$\begin{aligned} F_{A_x} &= C_x(\alpha, \beta, H, M, \delta_a, \delta_e, \delta_r, U, V, W, P, Q, R) \frac{1}{2} \rho V_T^2 S \\ F_{A_y} &= C_y(\alpha, \beta, H, M, \delta_a, \delta_e, \delta_r, U, V, W, P, Q, R) \frac{1}{2} \rho V_T^2 S \\ F_{A_z} &= C_z(\alpha, \beta, H, M, \delta_a, \delta_e, \delta_r, U, V, W, P, Q, R) \frac{1}{2} \rho V_T^2 S \\ M_{A_x} &= C_l(\alpha, \beta, H, M, \delta_a, \delta_e, \delta_r, U, V, W, P, Q, R) \frac{1}{2} \rho V_T^2 S b \\ M_{A_y} &= C_m(\alpha, \beta, H, M, \delta_a, \delta_e, \delta_r, U, V, W, P, Q, R) \frac{1}{2} \rho V_T^2 S \bar{c} \\ M_{A_z} &= C_n(\alpha, \beta, H, M, \delta_a, \delta_e, \delta_r, U, V, W, P, Q, R) \frac{1}{2} \rho V_T^2 S b \end{aligned} \quad (1-33)$$

Equation (1-33) shows that the aerodynamic forces and moments are very complicated. Due to the limitation of methods and tools available for determining the aerodynamic coefficients as function of parameters shown in Eqn. (1-33), the simpler aerodynamic model will be used for the simulation, see Eqns. (1-34) and (1-35). These equations are adopted from aircraft control model.

In aircraft control studies which the interest is laying in the aircraft's response to a (small) deviation from a steady rectilinear symmetrical flight, the aerodynamic forces and moments can be

separated into two uncoupled groups of symmetric and asymmetric equations.

Symmetric equations:

$$\begin{aligned}
 F_{A_x} &= \left(C_X(\alpha, H) + C_{X_q} q \frac{\bar{c}}{2V_R} + C_{X_{\delta_e}} \delta_e \right) \frac{1}{2} \rho V_T^2 S \\
 F_{A_z} &= \left(C_Z(\alpha, H) + C_{Z_q} q \frac{\bar{c}}{2V_R} + C_{Z_{\delta_e}} \delta_e \right) \frac{1}{2} \rho V_T^2 S \\
 M_{A_y} &= \left(C_m(\alpha, H) + C_{m_q} q \frac{\bar{c}}{2V_R} + C_{m_{\delta_e}} \delta_e \right) \frac{1}{2} \rho V_T^2 S \bar{c}
 \end{aligned} \tag{1-34}$$

The aerodynamic coefficients C_X and C_Z which occurred in Eqn. (1-34) were calculated from aerodynamic lift and drag using Equation (1-8). Aerodynamic lift and drag coefficients of the aircraft were predicted using Digital DATCOM as function of angle of attack (α) and altitude (H).

Asymmetric equations:

$$\begin{aligned}
 F_{A_y} &= \left(C_{Y_o} + C_{Y_\beta} \beta + C_{Y_p} p \frac{b}{2V_R} + C_{Y_r} r \frac{b}{2V_R} + C_{Y_{\delta_a}} \delta_a + C_{Y_{\delta_r}} \delta_r \right) \frac{1}{2} \rho V_T^2 S \\
 M_{A_x} &= \left(C_{l_o} + C_{l_\beta} \beta + C_{l_p} p \frac{b}{2V_R} + C_{l_r} r \frac{b}{2V_R} + C_{l_{\delta_a}} \delta_a + C_{l_{\delta_r}} \delta_r \right) \frac{1}{2} \rho V_T^2 S b \\
 M_{A_z} &= \left(C_{n_o} + C_{n_\beta} \beta + C_{n_p} p \frac{b}{2V_R} + C_{n_r} r \frac{b}{2V_R} + C_{n_{\delta_a}} \delta_a + C_{n_{\delta_r}} \delta_r \right) \frac{1}{2} \rho V_T^2 S b
 \end{aligned} \tag{1-35}$$

Aerodynamic coefficient C_Y which occurred in Equation (1-35) was also calculated from aerodynamic lift and drag using Equation (1-8). For sideslip angle $\beta = 0$, the aerodynamic C_{Y_o} , C_{l_o} and C_{n_o} are assumed to be zero.

Stability and control derivatives occurred in equation (1-34) and (1-35) will be calculated using DATCOM and Smetana method. These parameters will be listed in Appendix C.

1.2.5 LINEARIZATION OF EQUATIONS OF MOTION

We rewrite the complete equation of motion for conventional aircraft in the form of

$$\begin{aligned}
 m(\dot{U} + QW - RV) &= -mg \sin \theta + F_{a_x} + F_{p_x} + F_{c_x} \\
 m(\dot{V} + RU - PW) &= mg \cos \theta \sin \varphi + F_{a_y} + F_{p_y} + F_{c_y} \\
 m(\dot{W} + PV - QU) &= mg \cos \theta \cos \varphi + F_{a_z} + F_{p_z} + F_{c_z} \\
 I_{xx} \dot{P} + (I_{zz} - I_{yy})QR - J_{xz} \dot{R} - J_{xz} PQ &= M_{a_x} + M_{p_x} + M_{c_x} \\
 I_{yy} \dot{Q} + (I_{xx} - I_{zz})PR + J_{xz} (P^2 - R^2) &= M_{a_y} + M_{p_y} + M_{c_y} \\
 I_{zz} \dot{R} + (I_{yy} - I_{xx})PQ - J_{xz} \dot{P} - J_{xz} QR &= M_{a_z} + M_{p_z} + M_{c_z}
 \end{aligned} \tag{1-36}$$

Linearization of equations of motion is derived at trim condition, i.e. the condition when all acceleration (translation and rotation) are zero, $\dot{U}_o = \dot{V}_o = \dot{W}_o = \dot{P}_o = \dot{Q}_o = \dot{R}_o = 0$. At this condition, equations of motion become,

$$\begin{aligned}
 m(Q_o W_o - R_o V_o) &= -mg \sin \theta_o + F_{A_{x_o}} + F_{P_{x_o}} + F_{C_{x_o}} \\
 m(R_o U_o - P_o W_o) &= mg \cos \theta_o \sin \varphi_o + F_{A_{y_o}} + F_{P_{y_o}} + F_{C_{y_o}} \\
 m(P_o V_o - Q_o U_o) &= mg \cos \theta_o \cos \varphi_o + F_{A_{z_o}} + F_{P_{z_o}} + F_{C_{z_o}}
 \end{aligned} \tag{1-37}$$

$$\begin{aligned}
 (I_{zz} - I_{yy})Q_o R_o - J_{xz} P_o Q_o &= M_{A_{x_o}} + M_{P_{x_o}} + M_{C_{x_o}} \\
 (I_{xx} - I_{zz})P_o R_o + J_{xz} (P_o^2 - R_o^2) &= M_{A_{y_o}} + M_{P_{y_o}} + M_{C_{y_o}} \\
 (I_{yy} - I_{xx})P_o Q_o - J_{xz} Q_o R_o &= M_{A_{z_o}} + M_{P_{z_o}} + M_{C_{z_o}}
 \end{aligned} \tag{1-38}$$

we introduce small disturbance such that

$$\begin{aligned}
 U &= U_o + u & \theta &= \theta_o + d\theta & P &= P_o + p \\
 V &= V_o + v & \varphi &= \varphi_o + d\varphi & Q &= Q_o + q \\
 W &= W_o + w & \psi &= \psi_o + d\psi & R &= R_o + r
 \end{aligned} \tag{1-39}$$

Where $u, v, w, p, q, r, d\phi, d\theta$ and $d\psi$ is small deviation from its steady state value.

During trim condition, external force and moment can be written as:

$$\begin{aligned}
 F_{a_x} &= F_{a_{x_o}} + dF_{a_x} & F_{p_x} &= F_{p_{x_o}} + dF_{p_x} \\
 F_{c_x} &= F_{c_{x_o}} + dF_{c_x} & & \text{etc} \\
 M_{a_x} &= M_{a_{x_o}} + dM_{a_x} & M_{p_x} &= M_{p_{x_o}} + dM_{p_x} \\
 M_{c_x} &= M_{c_{x_o}} + dM_{c_x} & & \text{etc}
 \end{aligned} \tag{1-40}$$

The trim condition is chosen at symmetrical cruising flight, where $V_0 = P_o = Q_o = R_o = 0$, dan $\theta_o = \phi_o = \psi_o = 0$. Applying Eqn. (1-37) to (1-40) and neglecting product of small variables, yields

$$m(\dot{u} + W_o q) = -(mg \cos \theta_o) d\theta + dF_{A_x} + dF_{P_x} + dF_{C_x} \tag{1-41}$$

$$m(\dot{v} + U_o r - W_o p) = (mg \cos \theta_o) d\phi + dF_{A_y} + dF_{P_y} + dF_{C_y} \tag{1-42}$$

$$m(\dot{w} - U_o q) = -(mg \sin \theta_o) d\theta + dF_{A_z} + dF_{P_z} + dF_{C_z} \tag{1-43}$$

$$I_{xx} \dot{p} - J_{xz} \dot{r} = dM_{A_x} + dM_{P_x} + dM_{C_x} \tag{1-44}$$

$$I_{yy} \dot{q} = dM_{A_y} + dM_{P_y} + dM_{C_y} \tag{1-45}$$

$$I_{zz} \dot{r} - J_{xz} \dot{p} = dM_{A_z} + dM_{P_z} + dM_{C_z} \tag{1-46}$$

1.1 MATLAB AND SIMULINK TOOLS FOR FLIGHT DYNAMICS SIMULATION

2 FLIGHT CONTROL

This chapter deals with control design and analysis using classical and modern techniques. The explanation will be given in examples. First we will give example of classical control applying for longitudinal and lateral control (first example), then continuing by modern control (second to fourth examples).

2.1 ATTITUDE AND ALTITUDE CONTROL USING ROOT LOCUS ANALYSIS

2.2 OPTIMAL PATH-TRACKING CONTROL FOR AUTONOMOUS UNMANNED HELICOPTER USING LINEAR QUADRATIC REGULATOR

This chapter presents tracking control design of a small-scale unmanned helicopter (Yamaha R-50) using Linear Quadratic Regulator (LQR) technique [10]. We proposed scheme involves two steps: (1) generate a path/trajectory off-line and (2) apply a time-invariant LQR to track the path/trajectory. Numerical simulation using MATLAB/Simulink[®] is carried out to demonstrate the feasibility of the control system. Physical parameter of R-50 helicopter is presented in Table 1.



FIGURE 2-1 A SMALL-SCALE UNMANNED HELICOPTER, YAMAHA R-50

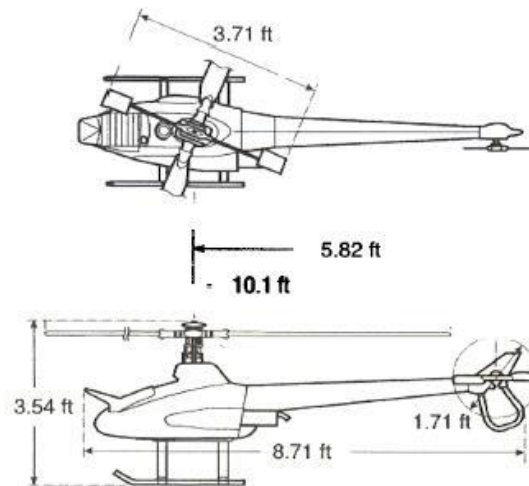


FIGURE 2-2 DIMENSION OF THE YAMAHA R-50 HELICOPTER

TABLE 1 PHYSICAL PARAMETER OF THE YAMAHA R-50

Rotor speed	850 rpm
Tip speed	449 ft/s
Dry weight	97 lb
Instrumented	150 lb
Engine	Single cylinder, 2-stroke

2.2.1 LINEARIZED MODEL

The linearized model of R-50 dynamics can be written in the state-space form as

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \quad (2-1)$$

Where

$$\mathbf{x} = [u \ v \ w \ p \ q \ r \ \phi \ \theta \ a \ b \ w \ r \ r_{fb} \ c \ d]^T \quad (2-2)$$

is state vector, and

$$\mathbf{u} = [\delta_{lat} \quad \delta_{lon} \quad \delta_{ped} \quad \delta_{col}]^T \tag{2-3}$$

is control input. The matrices A and B are shown in the complete state-space form (Figure 1-4).

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{p} \\ \dot{q} \\ \dot{\phi} \\ \dot{\theta} \\ \tau_f \dot{a} \\ \tau_f \dot{b} \\ \dot{w} \\ \dot{r} \\ \dot{r}_{fb} \\ \tau_s \dot{c} \\ \tau_s \dot{d} \end{bmatrix} = \begin{bmatrix} X_u & 0 & 0 & 0 & 0 & -g & X_a & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & g & 0 & 0 & Y_b & 0 & 0 & 0 & 0 & 0 & 0 \\ L_u & L_v & 0 & 0 & 0 & 0 & 0 & L_b & L_w & 0 & 0 & 0 & 0 & 0 \\ M_u & M_v & 0 & 0 & 0 & 0 & M_a & 0 & M_w & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\tau_f & 0 & 0 & -1 & A_b & 0 & 0 & 0 & A_c & 0 & 0 \\ 0 & 0 & -\tau_f & 0 & 0 & 0 & B_a & -1 & 0 & 0 & 0 & 0 & B_d & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & Z_a & Z_b & Z_w & Z_r & 0 & 0 & 0 & 0 \\ 0 & N_v & N_p & 0 & 0 & 0 & 0 & 0 & N_w & N_r & N_{r_{fb}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & K_r & K_{r_{fb}} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\tau_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -\tau_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ p \\ q \\ \phi \\ \theta \\ a \\ b \\ w \\ r \\ r_{fb} \\ c \\ d \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & Y_{ped} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & M_{col} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ A_{lat} & A_{lon} & 0 & 0 \\ B_{lat} & B_{lon} & 0 & 0 \\ 0 & 0 & 0 & Z_{col} \\ 0 & 0 & N_{ped} & N_{col} \\ 0 & 0 & 0 & 0 \\ 0 & C_{lon} & 0 & 0 \\ D_{lat} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_{lat} \\ \delta_{lon} \\ \delta_{ped} \\ \delta_{col} \end{bmatrix}$$

FIGURE 2-3 THE COMPLETE STATE-SPACE FORM OF R-50 DYNAMICS

The parameter values of matrix A and B for hover and cruise flight condition presented in Table 2 and Table 3 below.

TABLE 2 PARAMETER VALUES OF MATRIX A

Parameter	Hover	Cruise
X_u	-0.0505	-0.122
X_θ, X_a	-32.2	-32.2
X_r	0	-11
Y_v	-0.154	-0.155

Y_ϕ, Y_b	32.2	32.2
Y_r	0	-49.2
L_u	-0.144	0
L_v	0.143	0
L_w	0	-0.213
L_b	166	213
M_u	-0.0561	0
M_v	-0.0585	0
M_w	0	0.0728
M_a	82.6	108
B_a	0.368	0.419
B_d	0.71	0.664
A_b	-0.189	-0.176
A_c	0.644	0.577
Z_b	-131	0
Z_a	-9.75	0
Z_w	-0.614	-1.01
Z_r	0.93	0
Z_p	0	11
Z_q	0	49.2
N_p	-3.53	0
N_v	0.0301	0.401
N_w	0.0857	0
N_r	-4.13	-3.9
N_{rpb}	-33.1	-26.4

K_r	2.16	2.18
K_{rfb}	-8.26	-7.79

TABLE 3 PARAMETER VALUES OF MATRIX B

Parameter	Hover	Cruise
B_{lat}	0.14	0.124
B_{lon}	0.0138	0.02
A_{lat}	0.0313	0.0265
A_{lon}	-0.1	-0.0837
Z_{col}	-45.8	-60.3
M_{col}	0	6.98
N_{col}	-3.33	0
N_{ped}	33.1	26.4
D_{lat}	0.273	0.29
C_{lon}	-0.259	-0.225
Y_{ped}	0	11.23
τ_p	0.0991	0.0589
τ_f	0.046	0.0346
h_{cg}	-0.411	-0.321
τ_s	0.342	0.259

2.2.2 MODIFIED LINEARIZED MODEL

We have modified the original dynamic model above for our convenience. We added to the model, the rotation $\dot{\psi} = r$ and then rearrange the state vector as follows

$$\mathbf{x} = [u \ v \ w \ p \ q \ r \ \phi \ \theta \ \psi \ r_{fb} \ a \ b \ c \ d]^T \quad (2-4)$$

Using this new state vector, we have new model with the matrices A and B are as follows (Eqs. 28 and 29),

$$A = \begin{bmatrix} X_u & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & X_a & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & Y_b & 0 & 0 \\ 0 & 0 & Z_w & 0 & 0 & Z_r & 0 & 0 & 0 & 0 & Z_a & Z_b & 0 & 0 \\ L_u & L_v & L_w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L_b & 0 & 0 \\ M_u & M_v & M_w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & M_a & 0 & 0 & 0 \\ 0 & N_v & N_w & N_p & 0 & N_r & 0 & 0 & 0 & N_{fb} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & K_r & 0 & 0 & 0 & K_{fb} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1/\tau_f & A_b/\tau_f & A_c/\tau_f & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & B_a/\tau_f & -1/\tau_f & 0 & B_d/\tau_f \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1/\tau_s & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1/\tau_s \end{bmatrix} \quad (2-5)$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & Y_{ped} & 0 \\ 0 & 0 & 0 & Z_{col} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & M_{col} \\ 0 & 0 & N_{ped} & N_{col} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ A_{lat}/\tau_f & A_{lon}/\tau_f & 0 & 0 \\ B_{lat}/\tau_f & B_{lon}/\tau_f & 0 & 0 \\ 0 & C_{lon}/\tau_f & 0 & 0 \\ D_{lat}/\tau_s & 0 & 0 & 0 \end{bmatrix} \quad (2-6)$$

2.2.3 PATH GENERATOR

The path generator was developed by a simple idea, i.e. setting the trajectory/path in the inertial reference and then finding its velocity profile. This method can be expressed in the following relation:

$$\begin{aligned} x &= x(t) & \dot{x}(t) &= V_x(t) \\ y &= y(t) & \text{and } \dot{y}(t) &= V_y(t) \\ z &= z(t) & \dot{z}(t) &= V_z(t) \end{aligned} \quad (2-7)$$

And the total velocity is

$$V_T = \sqrt{V_x^2 + V_y^2 + V_z^2} \quad (2-8)$$

The total velocity V_T must be less or equal to the maximum velocity of the helicopter, and it is assumed to be constant. We assume that the maximum velocity of the helicopter is $\sqrt{u_0^2 + v_0^2} = \sqrt{49.2^2 + (-11)^2} = 50.4$ ft/s, and therefore we take $V_T = 50$ ft/s for simulation.

The inertial frame, by definition, is chosen such that the positive z-axis is downward. We then set positive x-axis is eastward, and therefore the positive y-axis is southward. But for our convenience, we choose local horizon as inertial frame where the positive x-axis is eastward, the positive y-axis is northward, and the positive z-axis is upward. So we need to transform the original inertial frame to the local horizon frame. The transformation can be expressed as follows:

$$\begin{Bmatrix} X_E \\ Y_E \\ Z_E \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{Bmatrix} X_I \\ Y_I \\ Z_I \end{Bmatrix} \quad (2-9)$$

or inversely:

$$\begin{Bmatrix} X_I \\ Y_I \\ Z_I \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{Bmatrix} X_E \\ Y_E \\ Z_E \end{Bmatrix} \quad (2-10)$$

To give more precise understanding of this method, we present here three examples. The first example is generating horizontal circular trajectory (Figure 2-4). The second example is generating horizontal rectangular trajectory (Figure 2-6). And the third example is generating (3D) spiral trajectory (Figure 2-8).

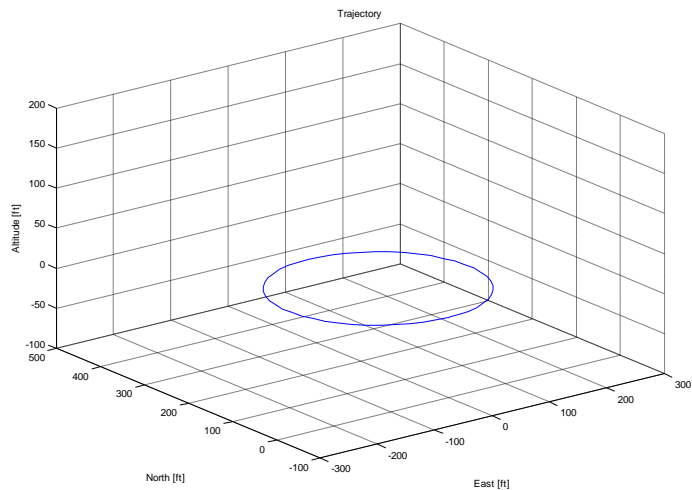


FIGURE 2-4 TRAJECTORY FOR EXAMPLE 1, CIRCULAR

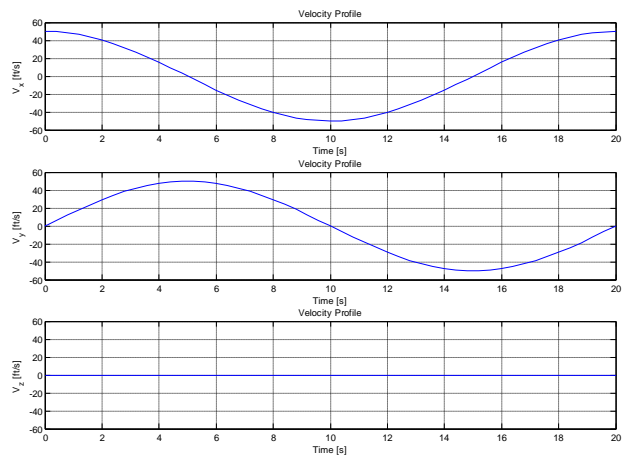


FIGURE 2-5 VELOCITY PROFILE FOR EXAMPLE 1

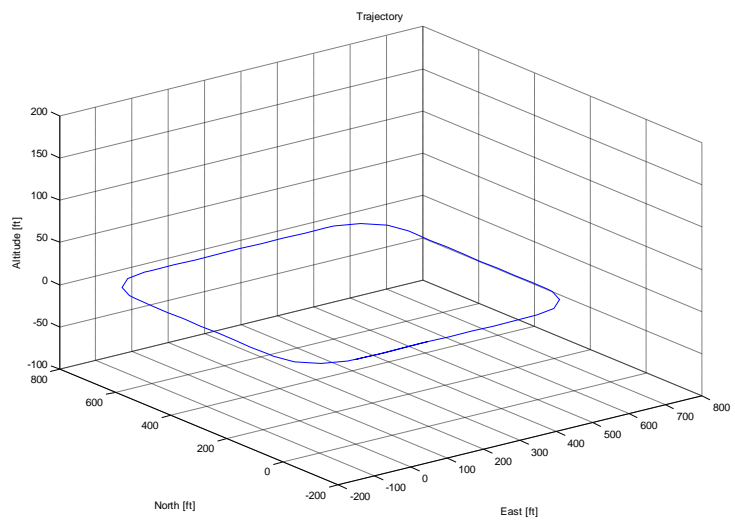
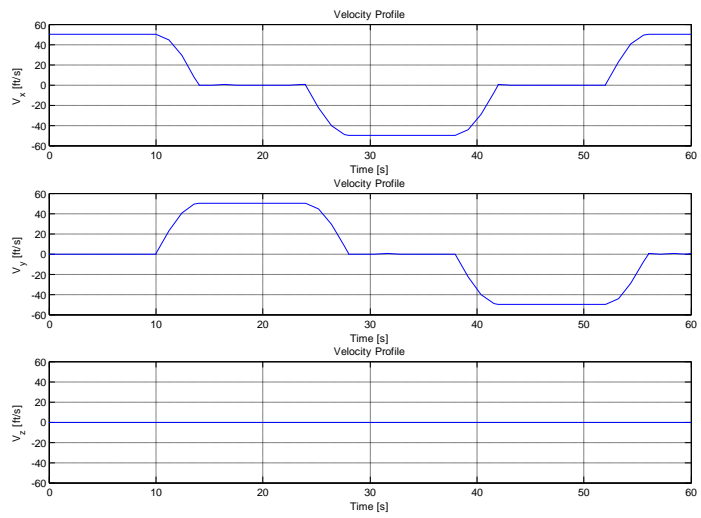
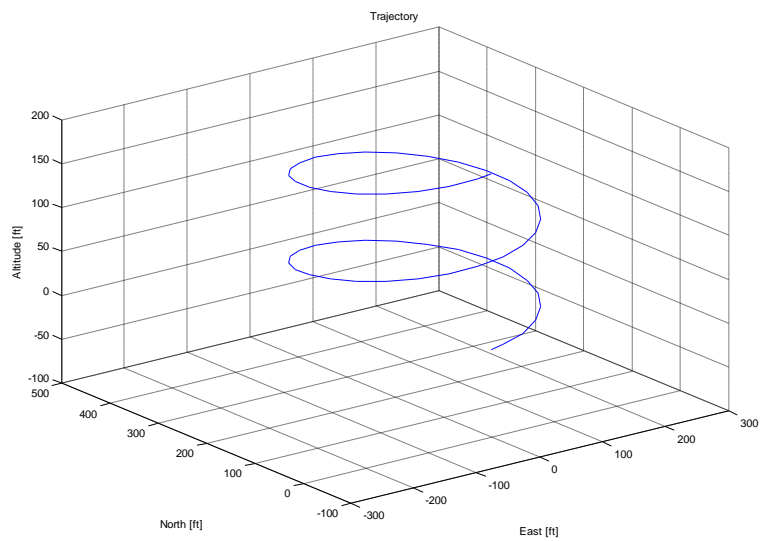


FIGURE 2-6 TRAJECTORY FOR EXAMPLE 2, RECTANGULAR

**FIGURE 2-7** VELOCITY PROFILE FOR EXAMPLE 2**FIGURE 2-8** TRAJECTORY FOR EXAMPLE 3, SPIRAL

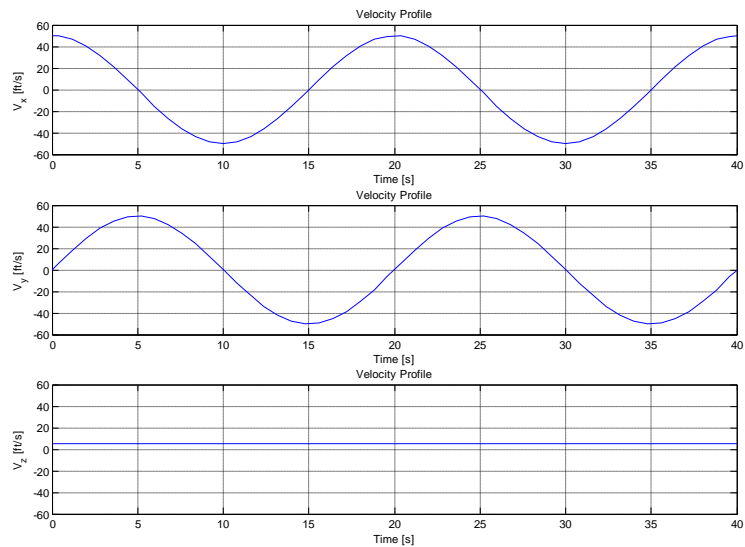


FIGURE 2-9 VELOCITY PROFILE FOR EXAMPLE 3

2.2.4 PATH-TRACKING CONTROLLER DESIGN

2.2.4.1 LINEAR REGULATOR PROBLEM

The controller design is based on LQR problem that is to find the control input that can minimize the performance measure

$$J = \frac{1}{2} \mathbf{x}^T(t_f) \mathbf{H} \mathbf{x}(t_f) + \frac{1}{2} \int_{t_0}^{t_f} [\mathbf{x}^T(t) \mathbf{Q}(t) \mathbf{x}(t) + \mathbf{u}^T(t) \mathbf{R}(t) \mathbf{u}(t)] dt \quad (2-11)$$

Referring to the plant

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t) \mathbf{x}(t) + \mathbf{B}(t) \mathbf{u}(t) \quad (2-12)$$

which have the physical interpretation: it is desired to maintain the state vector close to the origin without an excessive expenditure of control effort.

The solution of this LQR problem can be seen in [6], in the form of optimal gain matrix \mathbf{K} and the optimal control law. The optimal gain

matrix can be found by solving the matrix differential equation as follow

$$\dot{\mathbf{K}}(t) = \mathbf{K}(t)\mathbf{A}(t) - \mathbf{A}^T(t)\mathbf{K}(t) - \mathbf{Q}(t) + \mathbf{K}(t)\mathbf{B}(t)\mathbf{R}^{-1}(t)\mathbf{B}^T(t)\mathbf{K}(t) \quad (2-13)$$

And the optimal control input is

$$\begin{aligned} \mathbf{u}^*(t) &= -\mathbf{R}^{-1}(t)\mathbf{B}^T(t)\mathbf{K}(t)\mathbf{x}(t) \\ &\square -\mathbf{K}_{opt}(t)\mathbf{x}(t) \end{aligned} \quad (2-14)$$

2.2.4.2 PATH-TRACKING FORMULATION

The tracking problem can be expressed in simple mathematics relation as

$$\mathbf{x}_{error}(t) = \mathbf{x}_{ref}(t) - \mathbf{x}(t) \quad (2-15)$$

we then take the derivative of Eqs. (38) respect to time, yields

$$\dot{\mathbf{x}}_{error}(t) = \dot{\mathbf{x}}_{ref}(t) - \dot{\mathbf{x}}(t) \quad (2-16)$$

if we set $\mathbf{x}_{ref}(t) = \text{constant}$ then the time derivative of Eqs. (39) can be simplified to be

$$\dot{\mathbf{x}}_{error}(t) = -\dot{\mathbf{x}}(t) \quad (2-17)$$

equation (40) give us a motivation to design control law for path tracking problem, that is:

$$\dot{\mathbf{x}}_{error}(t) = -\alpha_i \dot{\mathbf{x}}(t); \quad i = 1, 2, \dots, n \quad (2-18)$$

where α_i is arbitrary positive constant.

2.2.4.3 PATH-TRACKING IMPLEMENTATION

Now, we are going to implement the path tracking controller. Our motivation is to minimizing the tracking error matrix \mathbf{x}_{error} . Where the tracking error matrix is

$$\mathbf{x}_{error}(t) = \begin{bmatrix} x_{error}(t) \\ y_{error}(t) \\ z_{error}(t) \end{bmatrix} = \begin{bmatrix} x_{ref}(t) - x(t) \\ y_{ref}(t) - y(t) \\ z_{ref}(t) - z(t) \end{bmatrix} \quad (2-19)$$

where x_{error} , y_{error} , z_{error} are error in x , y , and z position in body axis frame. Applying equation (41) to equation (42) yields

$$\dot{\mathbf{x}}_{error}(t) = \begin{bmatrix} \dot{x}_{error}(t) \\ \dot{y}_{error}(t) \\ \dot{z}_{error}(t) \end{bmatrix} = - \begin{bmatrix} \alpha_1 \dot{x}(t) \\ \alpha_2 \dot{y}(t) \\ \alpha_3 \dot{z}(t) \end{bmatrix} \quad (2-20)$$

Substituting $\dot{x} = u$, $\dot{y} = v$, $\dot{z} = w$ to Eqs. (43) yields

$$\dot{\mathbf{x}}_{error}(t) = - \begin{bmatrix} \alpha_1 u(t) \\ \alpha_2 v(t) \\ \alpha_3 w(t) \end{bmatrix} \quad (2-21)$$

we choose the value such that $\alpha_1 = \alpha_2 = \alpha_3 = \alpha = 0.1$ by trial and error.

Using equation (2-21) we develop the augmented state-space model:

$$\dot{\mathbf{x}}_{aug}(t) = \mathbf{A}_{aug} \mathbf{x}_{aug}(t) + \mathbf{B}_{aug} \mathbf{u}(t) \quad (2-22)$$

where

$$\mathbf{x}_{aug} = [x_{error} \quad y_{error} \quad z_{error} \quad \mathbf{x}]^T \quad (2-23)$$

$$\mathbf{A}_{aug} = \begin{bmatrix} \mathbf{0}_{3 \times 3} & -0.1 \cdot \mathbf{I}_3 & \mathbf{0}_{3 \times 11} \\ \mathbf{0}_{14 \times 3} & & \mathbf{A} \end{bmatrix} \quad (2-24)$$

$$\mathbf{B}_{aug} = \begin{bmatrix} \mathbf{0}_{3 \times 4} \\ \mathbf{B} \end{bmatrix} \quad (2-25)$$

The performance measure is

$$J = \int_{t_0}^{t_f} [\mathbf{x}_{aug}^T(t) \mathbf{Q}(t) \mathbf{x}_{aug}(t) + \mathbf{u}^T(t) \mathbf{R}(t) \mathbf{u}(t)] dt \quad (2-26)$$

with

$$\begin{aligned} \mathbf{Q} &= 0.01 \cdot \mathbf{I}_{17} \\ \mathbf{R} &= 0.01 \cdot \mathbf{I}_4 \end{aligned} \quad (2-27)$$

then we minimizing (2-26) using LQR technique as describe before. The solution is the optimal gain matrix \mathbf{K}_{opt} (2-12).

2.2.5 MATLAB AND SIMULINK IMPLEMENTATION

2.2.5.1 LQR CONTROLLER IMPLEMENTATION: MATLAB CODE

The following code is Matlab implementation of the controller design using LQR.

```

=====
% Simulation Yamaha R-50 Helicopter
% Author : Singgih S. Wibowo
% NIM : 23604003

```

```

% Version 2.1, 18 Des 2004
% 1st modification, 21 Feb 2007
%=====
% References :
% [1] A.Budiyono, H.Y. Sutarto
%     "Multivariable Controller Design for
%     a small scale helicopter using
%     Coefficient Diagram Method"
% [2] B. Mettler, M.B. Tischler, Takeo Kanade
%     "System Identification Modeling of
%     a Small-Scale Unmanned Rotorcraft
%     for Flight Control Design"
%=====

%=====
% Physical Parameter of The Yamaha R-50
%=====
% Rotor speed      850 rpm
% Tip speed        449 ft/s
% Dry weight       97 lb
% Instrumented     150 lb
% Engine           Single cylinder, 2-stroke
% Flight autonomy  30 minutes
%=====

clear;

%=====
% A matrix [Hover mode; Cruise mode]%
%=====
Xu   = [ -0.0505; -0.122 ];
Xth  = [ -32.2 ; -32.2 ];
Xa   = [ -32.2 ; -32.2 ];
Xr   = [ 0 ; -11 ];
Yv   = [ -0.154 ; -0.155 ];
Yph  = [ 32.2 ; 32.2 ];
Yb   = [ 32.2 ; 32.2 ];
Yr   = [ 0 ; -49.2 ];
Lu   = [ -0.144 ; 0 ];
Lv   = [ 0.143 ; 0 ];
Lw   = [ 0 ; -0.213 ];
Lb   = [ 166 ; 213 ];
Mu   = [ -0.0561; 0 ];
Mv   = [ -0.0585; 0 ];
Mw   = [ 0 ; 0.0728];
Ma   = [ 82.6 ; 108 ];
Ba   = [ 0.368 ; 0.419 ];
Bd   = [ 0.71 ; 0.664 ];
Ab   = [ -0.189 ; -0.176 ];
Ac   = [ 0.644 ; 0.577 ];
Zb   = [-131 ; 0 ];
Za   = [ -9.75 ; 0 ];
Zw   = [ -0.614 ; -1.01 ];
Zr   = [ 0.93 ; 0 ];
Zp   = [ 0 ; 11 ];
Zq   = [ 0 ; 49.2 ];
Np   = [ -3.53 ; 0 ];
Nv   = [ 0.0301; 0.401 ];
Nw   = [ 0.0857; 0 ];
Nr   = [ -4.13 ; -3.9 ];
Nrfb = [ -33.1 ; -26.4 ];
Kr   = [ 2.16 ; 2.18 ];
Krfb = [ -8.26 ; -7.79 ];
g     = 32.2; %gravity constant = 32.2 ft/s^2

%=====
% B matrix [Hover mode; Cruise mode]%
%=====
Blat = [ 0.14 ; 0.124 ];
Blon = [ 0.0138; 0.02 ];

```



```

%=====
% X = [u v w p q r phi theta psi rfb a b c d]'
% U = [delta_lat delta_lon delta_ped delta_col]'
%=====
alp = 5;
A_aug = [0 0 0 -alp 0 0 zeros(1,11)
         0 0 0 0 -alp 0 zeros(1,11)
         0 0 0 0 0 -alp zeros(1,11)
         zeros(14,3) A];
B_aug = [zeros(3,4); B];
C = eye(14);
D = B*0;

%=====
% Linear Quadratic Regulator is computed here
% We defined The Performance Cost by
% J = INTEGRAL (X^2 + U^2);
%=====
weight = 1;
Q = weight*eye(17);
R = weight*eye(4);
K = lqr(A_aug,B_aug,Q,R);
%=====

```

2.2.5.2 PATH TRACKING SIMULATION: SIMULINK DIAGRAM

The following figures show the Simulink diagram of the LQR controller design

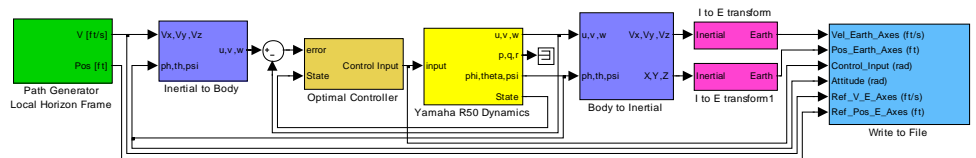


FIGURE 2-10 PATH TRACKING CONTROLLER MODEL

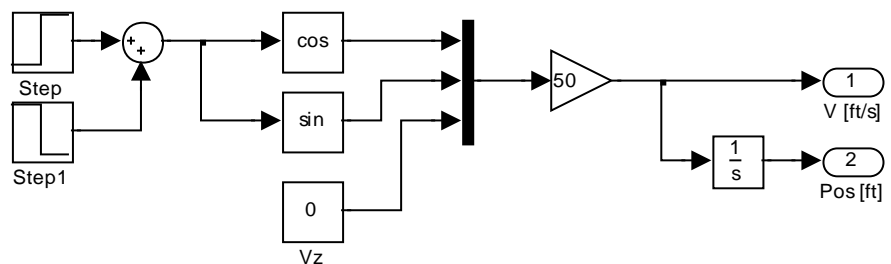


FIGURE 2-11 PATH GENERATOR BLOCK

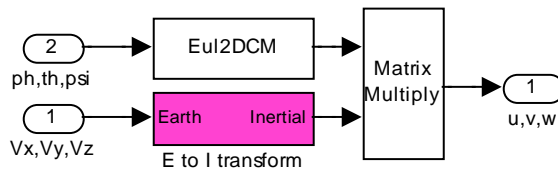


FIGURE 2-12 EARTH TO INERTIAL VELOCITY TRANSFORM BLOCK

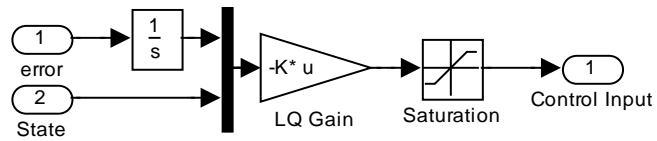


FIGURE 2-13 OPTIMAL CONTROLLER BLOCK

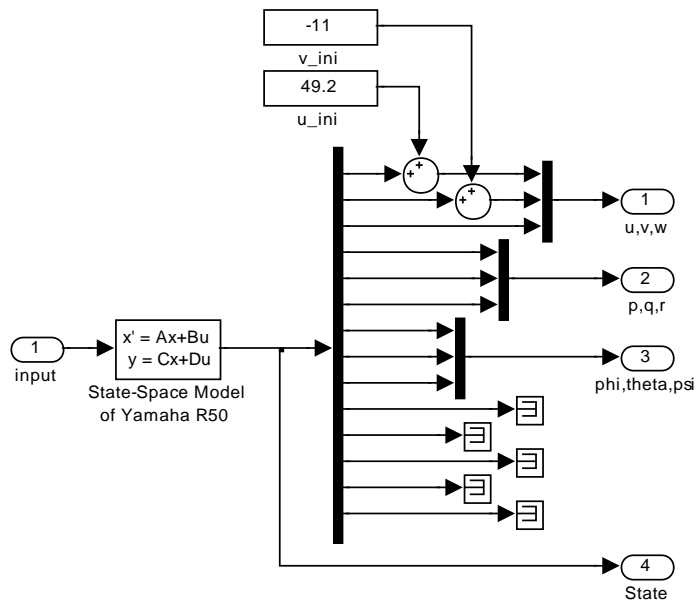


FIGURE 2-14 YAMAHA R50 DYNAMICS MODEL BLOCK

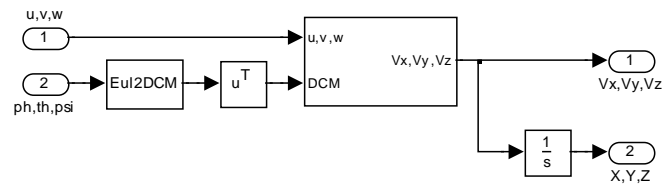


FIGURE 2-15 BODY TO INERTIAL TRANSFORM BLOCK

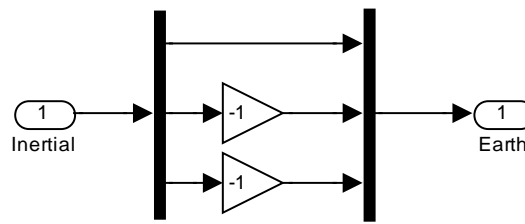


FIGURE 2-16 INERTIAL TO EARTH TRANSFORM BLOCK

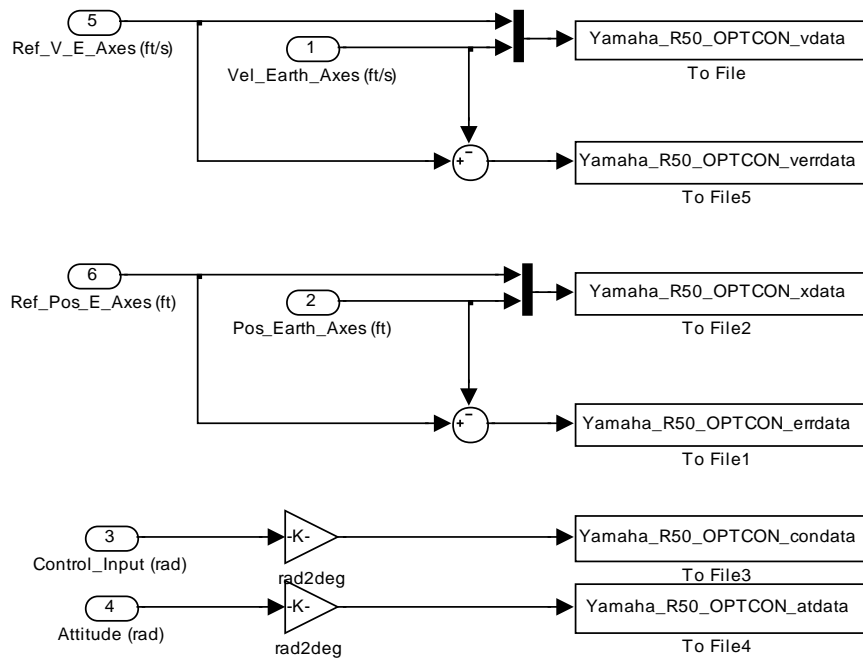


FIGURE 2-17 WRITE TO FILE BLOCK

2.2.5.3 PLOTTING SIMULATION RESULTS: MATLAB CODE

The following Matlab code will plot figures of the simulation results.

```

=====
% This program will plotting simulation results
% of Path-Tracking Controller for R-50 Helicopter
=====
% Loading data From
% [1] Yamaha_R50_OPTCON_vdata.mat
% [2] Yamaha_R50_OPTCON_xdata.mat
% [3] Yamaha_R50_OPTCON_cndata;
% [4] Yamaha_R50_OPTCON_atdata;
% [5] Yamaha_R50_OPTCON_verrdata;
% [6] Yamaha_R50_OPTCON_errdata;
% then plotting them
=====
% Author : Singgih S. Wibowo
% NIM : 23604003
% Version 2.1, 18 Des 2004
% 1st modification, 21 Feb 2007
=====

load Yamaha_R50_OPTCON_vdata;
load Yamaha_R50_OPTCON_xdata;
load Yamaha_R50_OPTCON_cndata;
load Yamaha_R50_OPTCON_atdata;
load Yamaha_R50_OPTCON_verrdata;
load Yamaha_R50_OPTCON_errdata;

Tmax = X(1,end);

%Flight Trajectory Geometry
figure(1);
plot3(X(2,:),X(3,:),X(4,:), 'b',X(5,:),X(6,:),X(7,:), 'r', 'LineWidth',2);
xlabel('East [ft]', 'FontSize',14);
ylabel('North [ft]', 'FontSize',14);
zlabel('Altitude [ft]', 'FontSize',14);
title('Flying Path [3D]', 'FontSize',14);
legend('Path_r_e_f', 'Path_o_u_t');
axis([-200 800 -200 800 -100 100]);
%axes('FontSize',14);
box on;
%grid on;

%Trajectory History
figure(2);
subplot(311);
plot(X(1,:),X(2,:), 'b',X(1,:),X(5,:), 'r', 'LineWidth',2);
xlabel('time [s]'); ylabel('East [ft]'); title('X-position');
legend('X_r_e_f', 'X_o_u_t');
axis([0 Tmax -200 800]);
%grid on;
subplot(312);
plot(X(1,:),X(3,:), 'b',X(1,:),X(6,:), 'r', 'LineWidth',2);
xlabel('time [s]'); ylabel('North [ft]'); title('Y-position');
legend('Y_r_e_f', 'Y_o_u_t');
axis([0 Tmax -200 800]);
%grid on;
subplot(313);
plot(X(1,:),X(4,:), 'b',X(1,:),X(7,:), 'r', 'LineWidth',2);
xlabel('time [s]'); ylabel('Altitude [ft]'); title('Z-position');
legend('Z_r_e_f', 'Z_o_u_t');
axis([0 Tmax -100 100]);
%grid on;

%Velocity History
figure(3);
subplot(311);

```

```

plot(V(1,:),V(2,:), 'b',V(1,:),V(5,:), 'r', 'LineWidth',2);
xlabel('time [s]'); ylabel('V_x [ft/s]'); title('X-Velocity');
legend('V_x_r_e_f', 'V_x_o_u_t');
axis([0 Tmax -60 60]);
%grid on;
subplot(312);
plot(V(1,:),V(3,:), 'b',V(1,:),V(6,:), 'r', 'LineWidth',2);
xlabel('time [s]'); ylabel('V_y [ft/s]'); title('Y-Velocity');
legend('V_y_r_e_f', 'V_y_o_u_t');
axis([0 Tmax -60 60]);
%grid on;
subplot(313);
plot(V(1,:),V(4,:), 'b',V(1,:),V(7,:), 'r', 'LineWidth',2);
xlabel('time [s]'); ylabel('V_z [ft/s]'); title('Z-velocity');
legend('V_z_r_e_f', 'V_z_o_u_t');
axis([0 Tmax -60 60]);
%grid on;

%Control Input History
figure(4);
subplot(221);
plot(Con(1,:),Con(2,:), 'b', 'LineWidth',2);
xlabel('time [s]'); ylabel('\delta_l_a_t [deg]'); title('Control Input 1');
axis([0 Tmax -30 30]);
%grid on;
subplot(222);
plot(Con(1,:),Con(3,:), 'b', 'LineWidth',2);
xlabel('time [s]'); ylabel('\delta_l_o_n [deg]'); title('Control Input 2');
axis([0 Tmax -30 30]);
%grid on;
subplot(223);
plot(Con(1,:),Con(4,:), 'b', 'LineWidth',2);
xlabel('time [s]'); ylabel('\delta_p_e_d [deg]'); title('Control Input 3');
axis([0 Tmax -30 30]);
%grid on;
subplot(224);
plot(Con(1,:),Con(5,:), 'b', 'LineWidth',2);
xlabel('time [s]'); ylabel('\delta_c_o_l [deg]'); title('Control Input 4');
axis([0 Tmax -30 30]);
%grid on;

%Attitude History
figure(5);
subplot(311);
plot(Atd(1,:),Atd(2,:), 'b', 'LineWidth',2);
xlabel('time [s]'); ylabel('\phi [deg]'); title('roll angle');
axis([0 Tmax -60 60]);
%grid on;
subplot(312);
plot(Atd(1,:),Atd(3,:), 'b', 'LineWidth',2);
xlabel('time [s]'); ylabel('\theta [deg]'); title('pitch angle');
axis([0 Tmax -60 60]);
%grid on;
subplot(313);
plot(Atd(1,:),Atd(4,:), 'b', 'LineWidth',2);
xlabel('time [s]'); ylabel('\psi [deg]'); title('yaw angle');
axis([0 Tmax -360 20]);
%grid on;

%Velocity Error History
figure(6);
subplot(311);
plot(verr(1,:),verr(2,:), 'r', 'LineWidth',2);
xlabel('time [s]');
ylabel('u_e_r_r_o_r [ft/s]');
title('X Velocity Error');
axis([0 Tmax -100 100]);
%grid on;
subplot(312);
plot(verr(1,:),verr(3,:), 'r', 'LineWidth',2);

```

```

xlabel('time [s]');
ylabel('v_e_r_r_o_r [ft/s]');
title('Y Velocity Error');
axis([0 Tmax -100 100]);
%grid on;
subplot(313);
plot(verr(1,:),verr(4,:), 'r', 'LineWidth',2);
xlabel('time [s]');
ylabel('w_e_r_r_o_r [ft/s]');
title('Z Velocity Error');
axis([0 Tmax -100 100]);
%grid on;

%Trajectory Error History
figure(7);
subplot(311);
plot(err(1,:),err(2,:), 'r', 'LineWidth',2);
xlabel('time [s]');
ylabel('X_e_r_r_o_r [ft]');
title('X Position Error');
axis([0 Tmax -100 100]);
%grid on;
subplot(312);
plot(err(1,:),err(3,:), 'r', 'LineWidth',2);
xlabel('time [s]');
ylabel('Y_e_r_r_o_r [ft]');
title('Y Position Error');
axis([0 Tmax -100 100]);
%grid on;
subplot(313);
plot(err(1,:),err(4,:), 'r', 'LineWidth',2);
xlabel('time [s]');
ylabel('Z_e_r_r_o_r [ft]');
title('Z Position Error');
axis([0 Tmax -100 100]);
%grid on;

```

2.2.6 NUMERICAL RESULTS

In this section, we present our numerical experiment result using MATLAB/Simulink[®]. The Simulink model as shown in Figure 2-10. We have carried out three experiments as follow

2.2.6.1 EXPERIMENT 1, CIRCULAR TRAJECTORY

Follow the circular trajectory lies on horizontal plane as given previous section, see Figure 2-4.

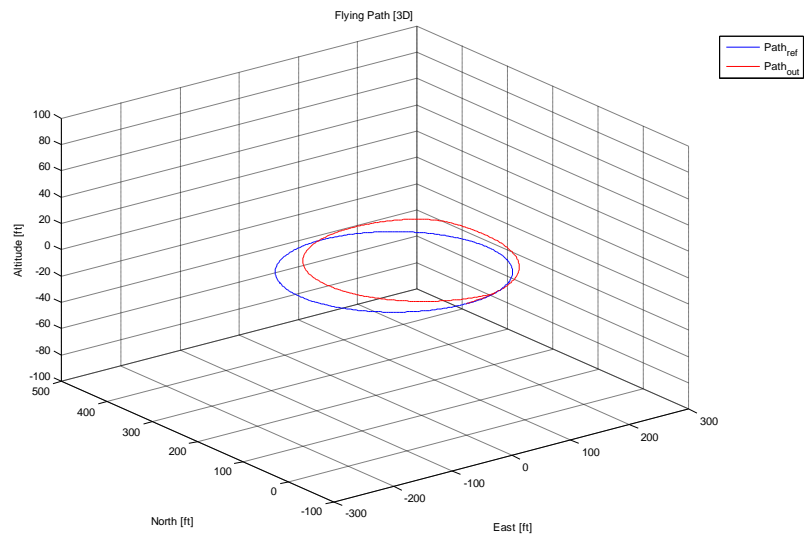


FIGURE 2-18 FLIGHT TRAJECTORY GEOMETRY

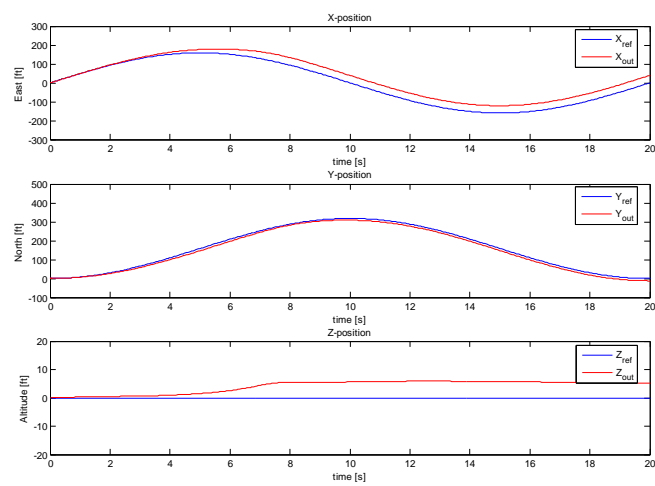


FIGURE 2-19 TRAJECTORY HISTORY

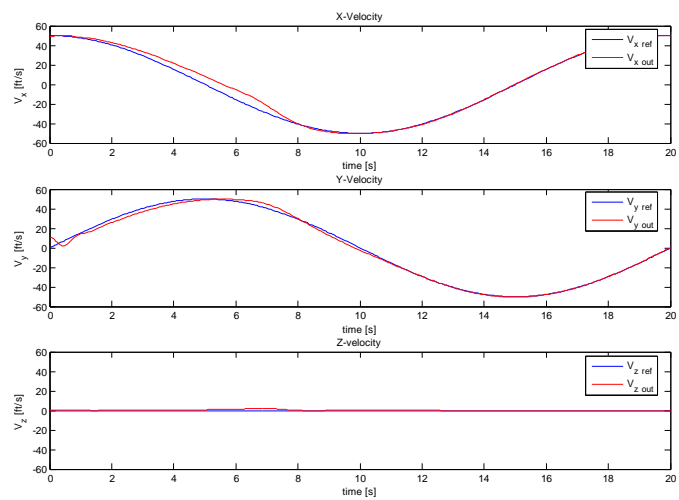


FIGURE 2-20 VELOCITY HISTORY

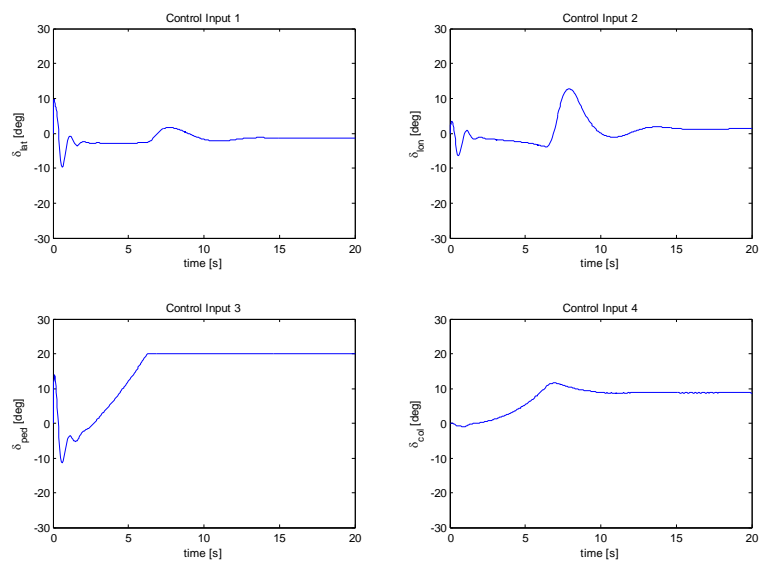
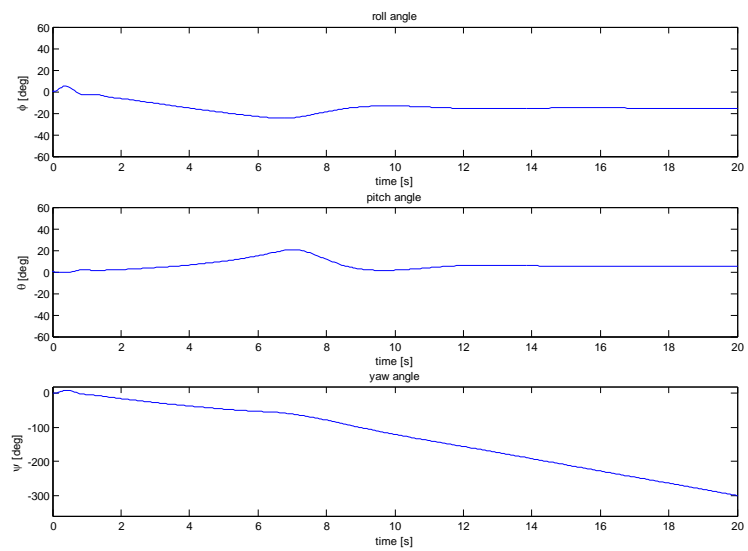
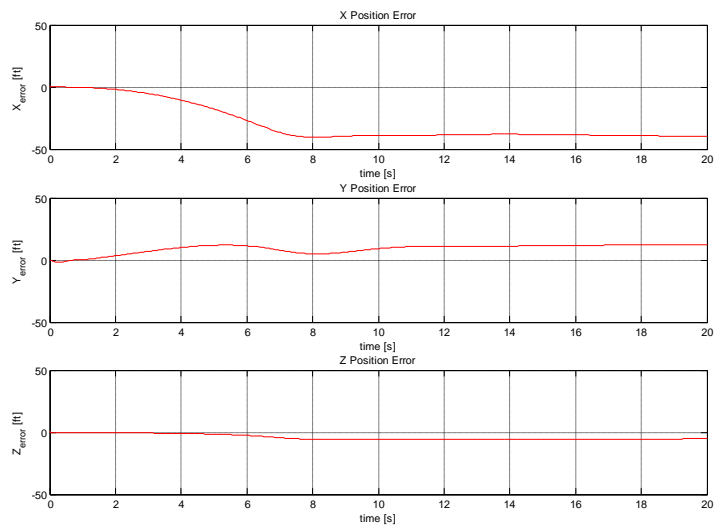


FIGURE 2-21 CONTROL INPUT HISTORY

**FIGURE 2-22** ATTITUDE HISTORY**FIGURE 2-23** TRAJECTORY ERROR HISTORY

2.2.6.2 EXPERIMENT 2, RECTANGULAR TRAJECTORY

Follow the rectangular trajectory as given in Fig.17.

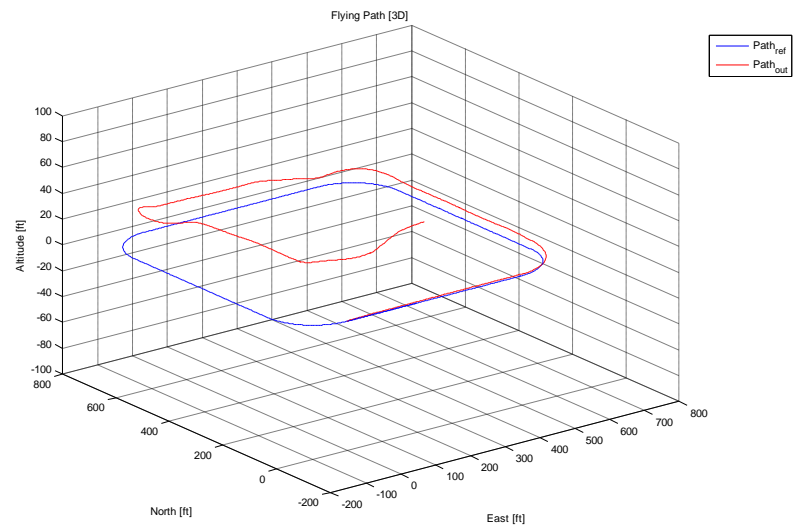


FIGURE 2-24 FLIGHT TRAJECTORY GEOMETRY

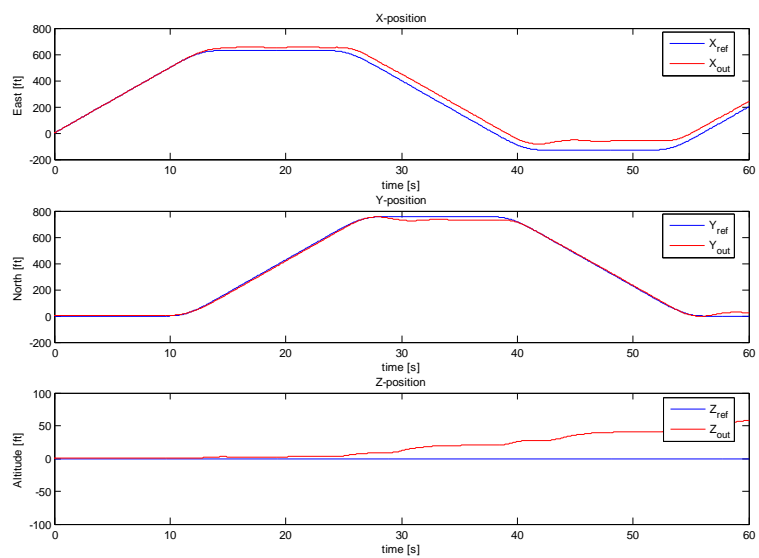


FIGURE 2-25 TRAJECTORY HISTORY

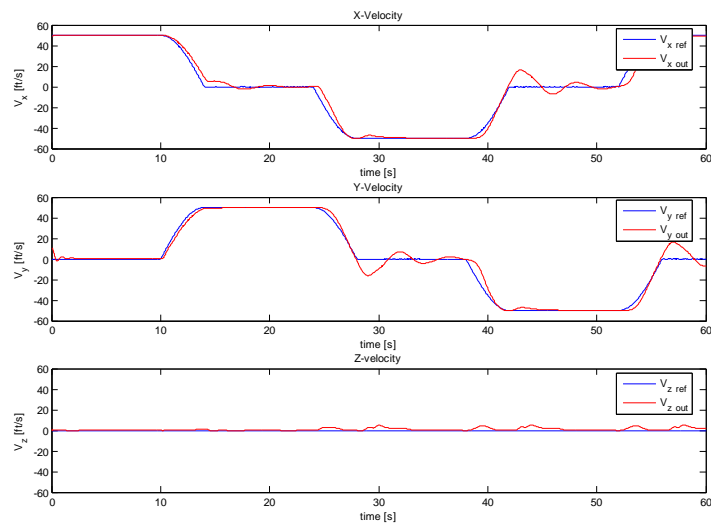


FIGURE 2-26 VELOCITY HISTORY

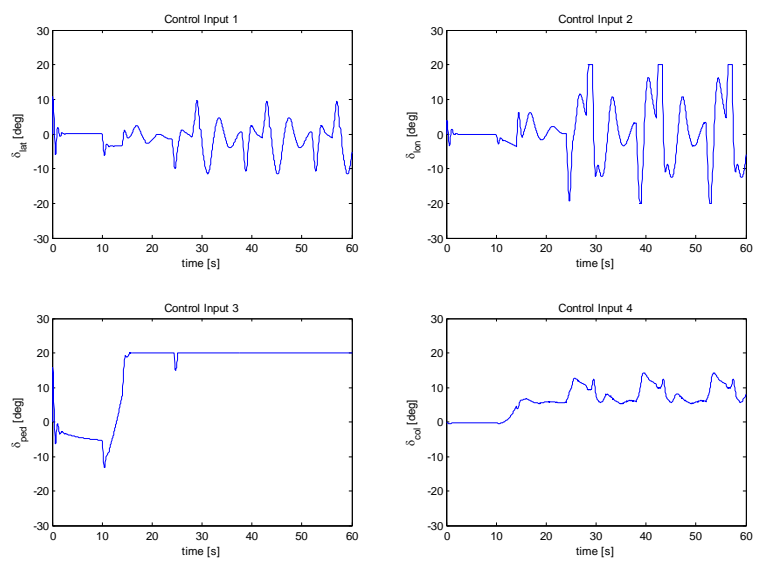


FIGURE 2-27 CONTROL INPUT HISTORY

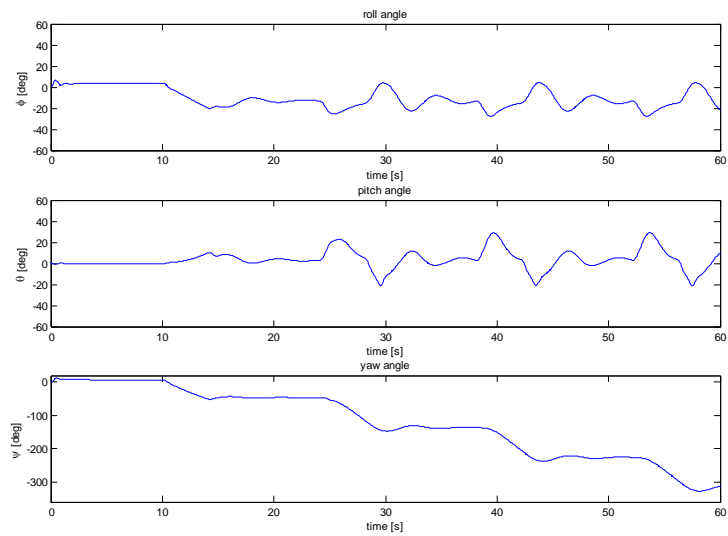


FIGURE 2-28 ATTITUDE HISTORY

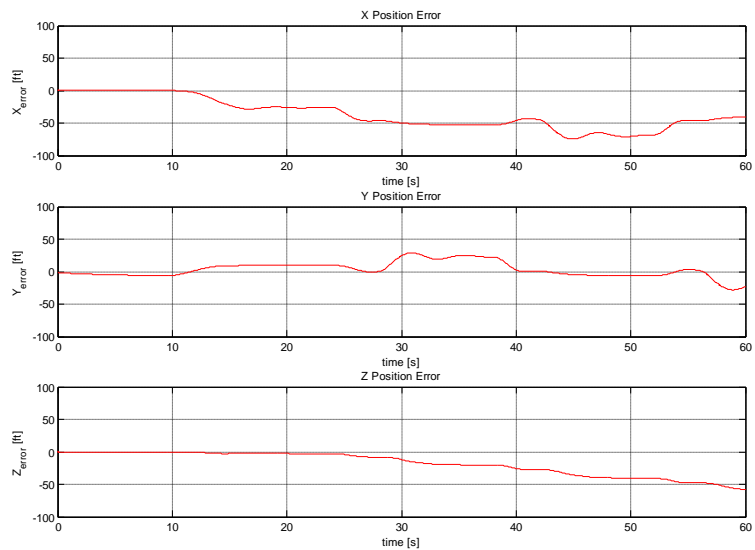


FIGURE 2-29 TRAJECTORY ERROR HISTORY

2.2.6.3 EXPERIMENT 3, SPIRAL TRAJECTORY

Follow the spiral trajectory as given in Fig.23.

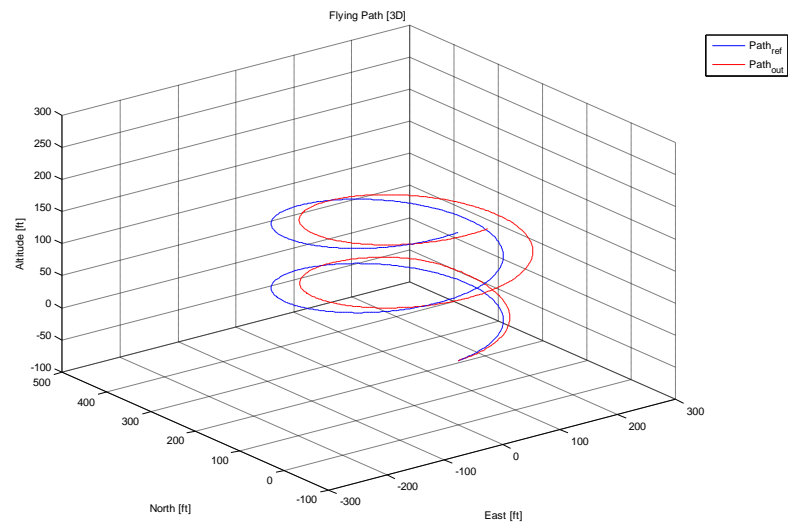


FIGURE 2-30 FLIGHT TRAJECTORY GEOMETRY

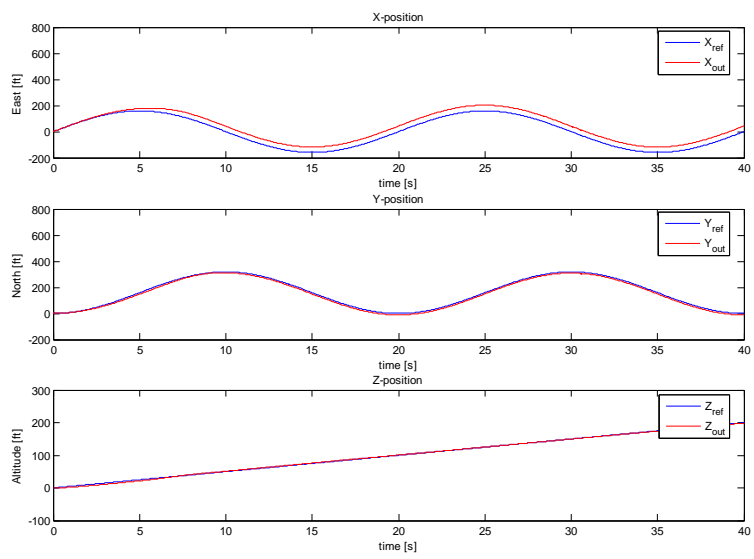


FIGURE 2-31 TRAJECTORY HISTORY

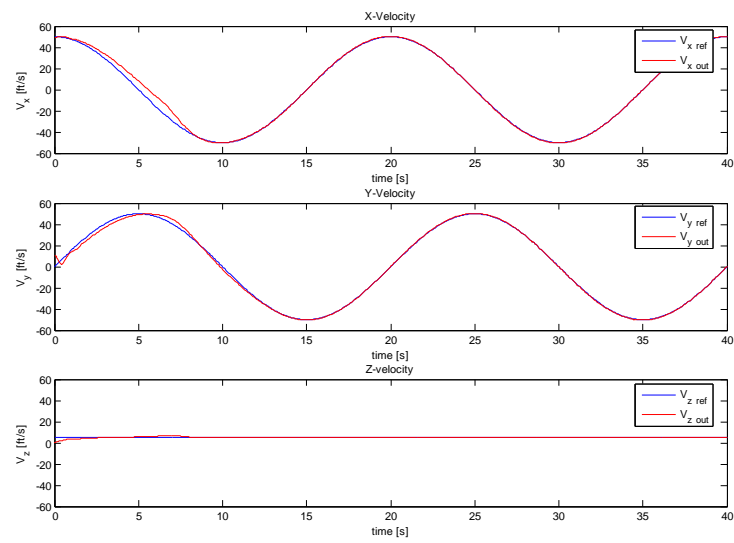


FIGURE 2-32 VELOCITY HISTORY

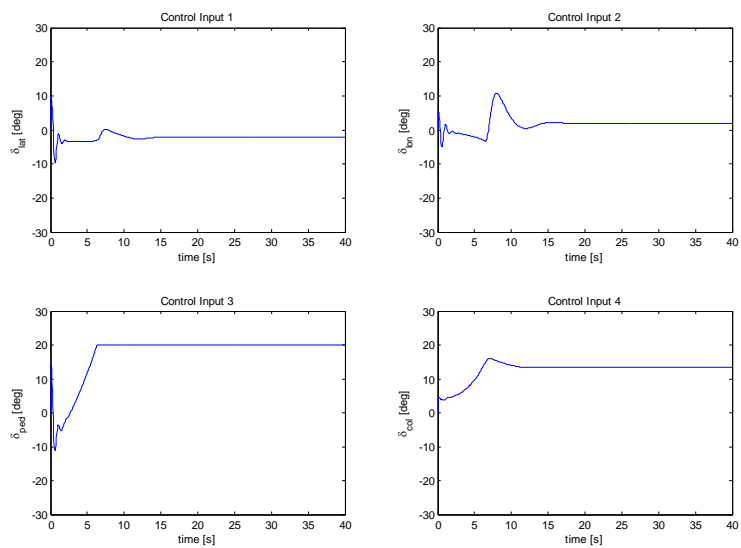


FIGURE 2-33 CONTROL INPUT HISTORY

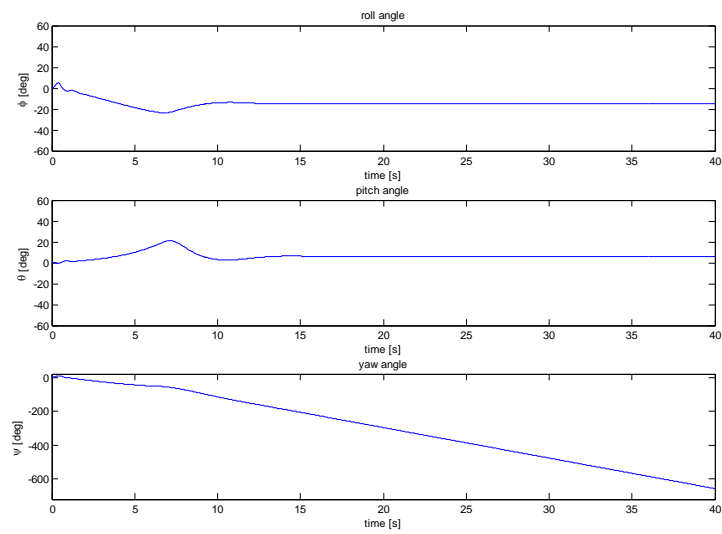


FIGURE 2-34 ATTITUDE HISTORY

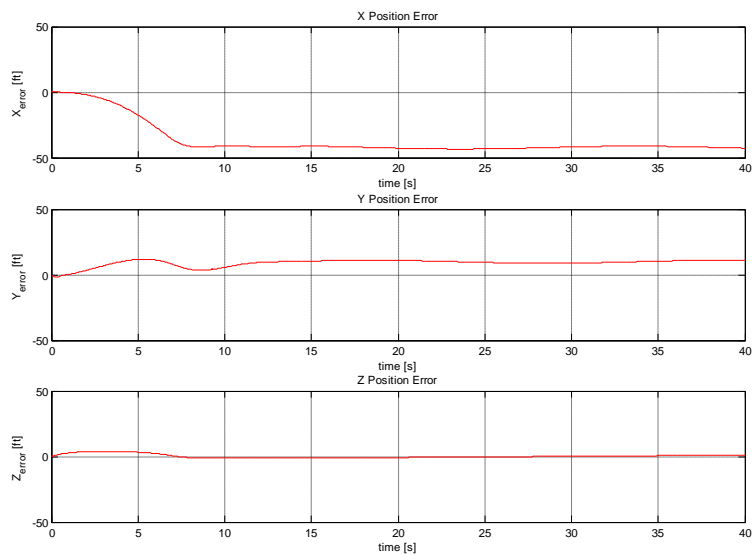


FIGURE 2-35 TRAJECTORY ERROR HISTORY

2.2.7 ANALYSIS AND DISCUSSION OF THE RESULTS

The path tracking controller using LQR applied to our platform (Yamaha R-50) have advantage and disadvantage. The LQR approach can be minimizing error tracking while keep control input low. But it

can be applied for full states feedback where in the real application may not applicable since only a part of states can be fed back.

2.3 COORDINATED TURN USING LINEAR QUADRATIC REGULATOR

The following example is taken from Matlab demo, see [16]. The program is modified for education purpose.

2.3.1 STATE-SPACE EQUATIONS FOR AN AIRFRAME

For this case, the state-space equation is a standard form

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} \quad (2-28)$$

where

$$\mathbf{x} = [u \ v \ w \ p \ q \ r \ \theta \ \varphi]^T \quad (2-29)$$

The variables u , v , and w are the three velocities with respect to the body frame, which is shown in Figure 2-36 below.

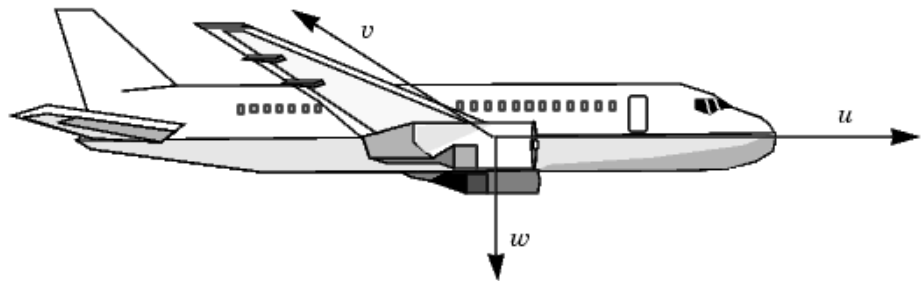


FIGURE 2-36 A BODY COORDINATE FRAME FOR AN AIRCRAFT [16]

The variables p , q , and r are the roll, pitch, and yaw rates, θ and φ are pitch and roll respectively.

2.3.2 PROBLEM DEFINITION

The goal is to perform a steady coordinated turn, as shown in the following figure.

Figure 4-57: An Aircraft Making a 60° Turn

To achieve this goal, we must design a controller that commands a steady turn by going through a 60° roll. In addition, assume that, the pitch angle, is required to stay as close to zero as possible.

2.3.3 MATLAB AND SIMULINK IMPLEMENTATION

2.3.3.1 IMPLEMENTATION OF LQR: MATLAB CODE

The following code is Matlab implementation of the controller design using LQR for aircraft turn.

```

=====
% LQG design for 60 degree aircraft turn
% Copyright 1986-2002 The MathWorks, Inc.
% $Revision: 1.5 $ $Date: 2002/04/10 06:40:38 $
=====
% State vector => x = [u,v,w,p,q,r,theta,phi]
%   u,v,w: linear velocities
%   p,q,r: roll, pitch, yaw rates
%   theta: pitch angle
%   phi : bank angle
% Control vector => u = [u1,u2,u3,u4]
=====
% Modified by SSW, 21 Feb 2007 for education purpose
=====

% Linear dynamics
A = [-0.0404    0.0618    0.0501   -0.0000   -0.0005    0.0000    0    0
      -0.1686   -1.1889    7.6870    0          0.0041    0          0    0
           0.1633   -2.6139   -3.8519    0.0000    0.0489   -0.0000    0    0
      -0.0000   -0.0000   -0.0000   -0.3386   -0.0474   -6.5405    0    0
      -0.0000    0.0000   -0.0000   -1.1288   -0.9149   -0.3679    0    0
      -0.0000   -0.0000   -0.0000    0.9931   -0.1763   -1.2047    0    0
           0          0          0.9056    0          0          -0.0000    0    0
           0          0         -0.0000    0          0.9467   -0.0046    0    0];

B = [ 20.3929   -0.4694   -0.2392   -0.7126
       0.1269   -2.6932    0.0013    0.0033
      -64.6939  -75.6295    0.6007    3.2358
       -0.0000    0          0.1865    3.6625
       -0.0000    0          23.6053    5.6270
       -0.0001    0          3.9462   -41.4112
           0          0          0          0
           0          0          0          0];

=====
% Add integrator state dz/dt = -phi
% Augmented vector => x_aug = [z,u,v,w,p,q,r,theta,phi]
% Augmented control => u_aug = [0,u1,u2,u3,u4]
=====
A_aug = [zeros(1,8) -1;
         zeros(8,1) A];
B_aug = [zeros(1,4) ; B];

```

```

=====
% IQR gain synthesis
=====
Q    = blkdiag(1,0.1*eye(6),1000,1);
R    = diag([10,50,1,1]);
K_lqr = lqr(A_aug,B_aug,Q,R);
    
```

2.3.3.2 COORDINATED TURN SIMULATION: SIMULINK DIAGRAM

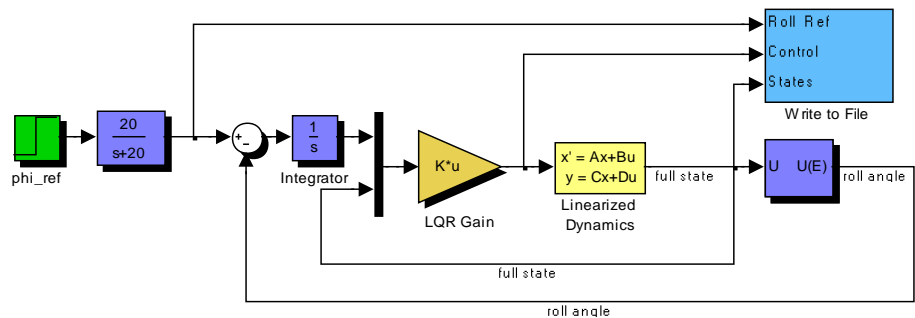


FIGURE 2-37 SIMULINK DIAGRAM OF COORDINATED TURN

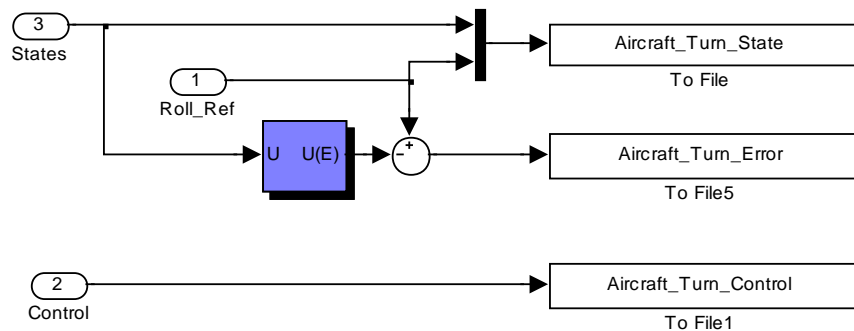


FIGURE 2-38 WRITE TO FILE BLOCK

2.3.3.3 PLOTTING RESULTS: MATLAB CODE

The following Matlab code will plot figures of the simulation results.

```

=====
% This program will plotting simulation results
% of Turn-Coordinator Controller for Aircraft
=====
    
```

```

% Loading data From
% [1] Aircraft_Turn_State.mat
% [2] Aircraft_Turn_Error.mat
% [3] Aircraft_Turn_Control.mat
% then plotting them
%=====
% Author : Singgih S. Wibowo
% NIM : 23604003
% Version 1.0, 21 Feb 2007
%=====

load Aircraft_Turn_State;
load Aircraft_Turn_Error;
load Aircraft_Turn_Control;

Tmax = state(1,end);

%=====
%Aircraft Attitude, Pitch and Roll
%=====
figure(1);
set(1,'Name','Attitude History');
subplot(211);
plot(state(1,:),state(8:9,:)*180/pi,'b','LineWidth',2);
set(gca,'FontSize',14);
xlabel('Time (s)');
ylabel('\theta (deg)');
title('Pitch Attitude History');
grid on;
subplot(212);
plot(state(1,:),state(10:11,:)*180/pi,'r',...
      state(1,:),state(9:10,:)*180/pi,'b','LineWidth',2);
set(gca,'FontSize',14);
xlabel('Time (s)');
ylabel('\phi (deg)');
title('Roll Attitude History');
legend('roll ref','roll actual');
grid on;

%=====
%Aircraft Tracking Error
%=====
figure(2);
set(2,'Name','Tracking Error');
plot(error(1,:),error(2,:)*180/pi,'b','LineWidth',2);
set(gca,'FontSize',14);
xlabel('Time (s)');
ylabel('\phi_r_e_f - \phi (deg)');
title('Roll Tracking-Error History');
grid on;

%=====
%Aircraft Control Input
%=====
figure(3);
set(3,'Name','Control Input');
plot(control(1,:),control(2,:)*180/pi,'b',...
      control(1,:),control(3,:)*180/pi,'g',...
      control(1,:),control(4,:)*180/pi,'r',...
      control(1,:),control(5,:)*180/pi,'m',...
      'LineWidth',2);
set(gca,'FontSize',14);
xlabel('Time (s)');
ylabel('Control Input (deg)');
title('Control Input History');
grid on;
legend('control 1','control 2','control 3','control 4');

```

2.3.4 RESULTS

This figure shows the response of to the 60° step command. Figure 4-58: Tracking the Roll Step Command As you can see, the system tracks the commanded 60° roll in about 60 seconds. Another goal was to keep , the pitch angle, relatively small. This figure shows how well the LQG controller did. Figure 4-59: Minimizing the Displacement in the Pitch Angle, Theta Finally, this figure shows the control inputs. Figure 4-60: The Control Inputs for the LQG Tracking Problem Try adjusting the Q and R matrices in `lqrdes.m` and inspecting the control inputs and the system states, making sure to rerun `lqrdes` to update the LQG gain matrix K. Through trial and error, you may improve the response time of this design. Also, compare the linear and nonlinear designs to see the effects of the nonlinearities on the system performance.

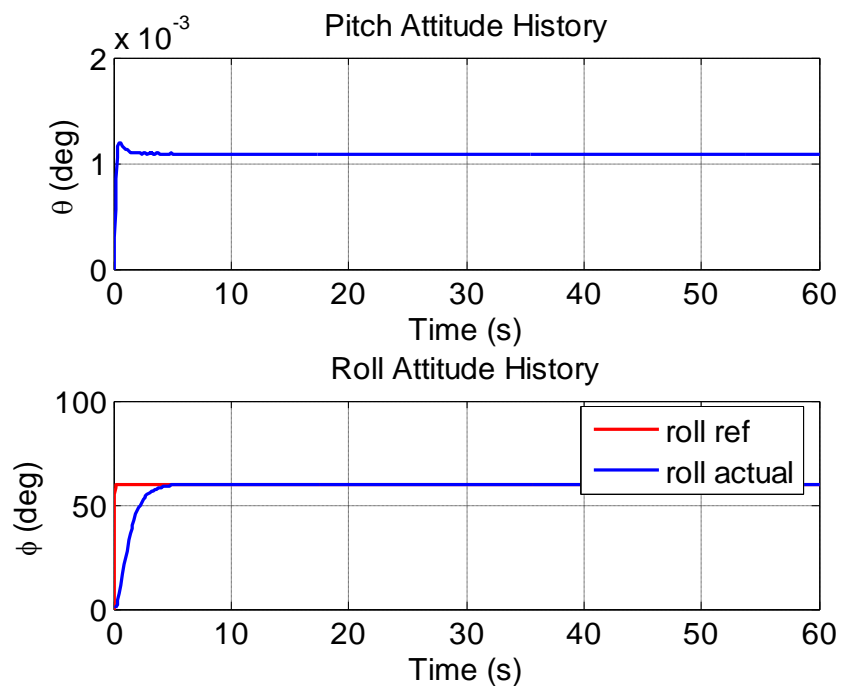
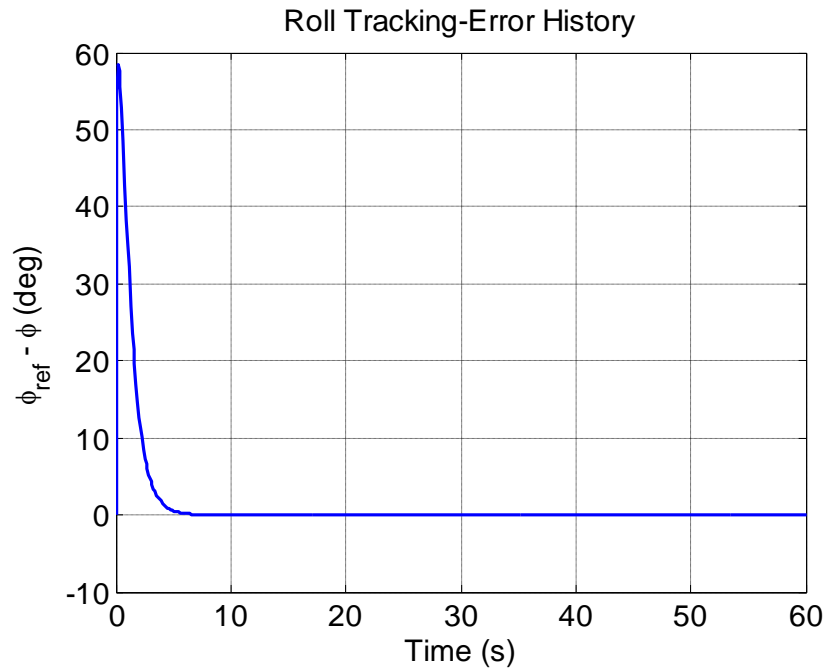
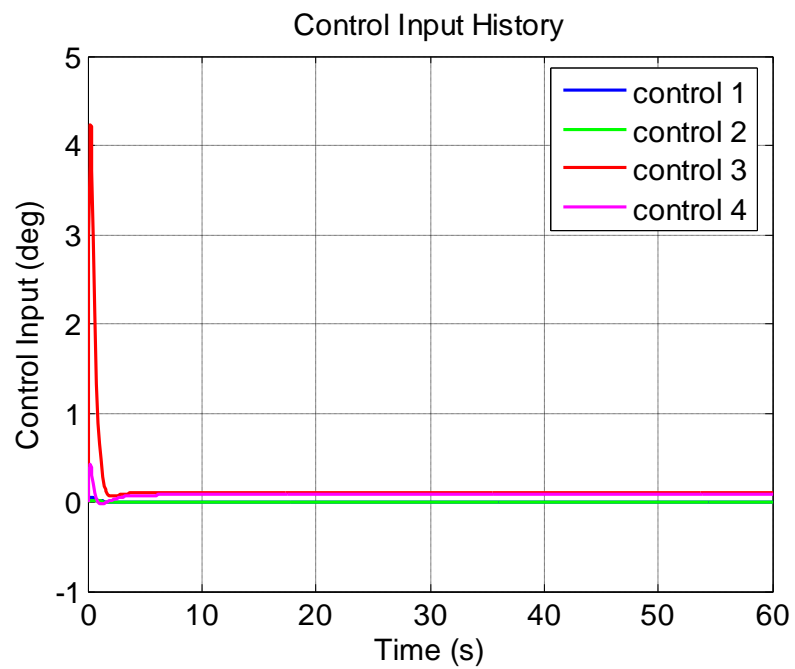


FIGURE 2-39 ATTITUDE HISTORY

**FIGURE 2-40** TRACKING ERROR HISTORY**FIGURE 2-41** CONTROL INPUT HISTORY

2.3.5 ANALYSIS AND DISCUSSION OF THE RESULTS

2.4 ADAPTIVE CONTROL FOR YAW DAMPER AND COORDINATED TURN

This chapter presents implementation of adaptive control for yaw damper and coordinated turn. The major content of this chapter is taken from [9].

2.4.1 YAW DAMPER AND COORDINATED TURN: DEFINITION

Yaw damper is a SAS (Stability Augmentation System) which augment the stability of dutch roll mode of an aircraft. The principle of this control system is giving command to rudder which causes a moment against yaw rate which finally damp the dutch roll. This control system sense yaw rate and use it for feedback. The following figure shows the block diagram of yaw damper system.

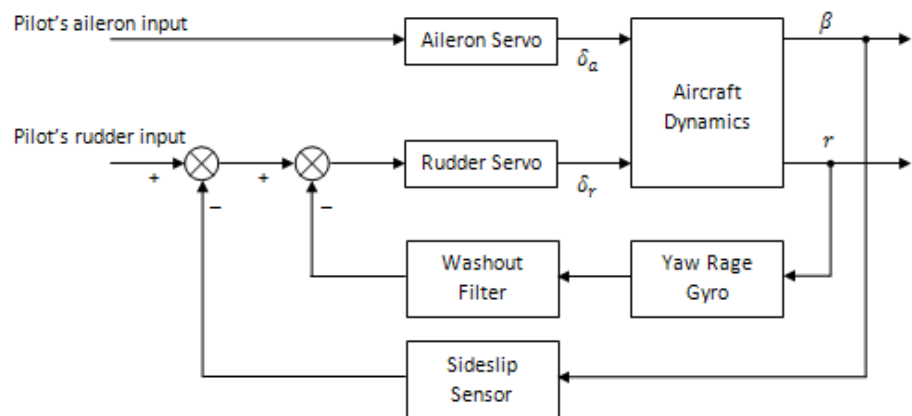


FIGURE 2-42 BLOCK DIAGRAM FOR TURN COORDINATOR SYSTEM

Coordinated turn maneuver is a turn maneuver with zero lateral acceleration at constant altitude. The absence of lateral acceleration makes aircraft passengers feel comfortable during the maneuver. This maneuver is difficult to do since it needs good coordination of control surface (elevator, aileron and rudder) deflection. Therefore an automatic control is needed for it.

2.4.2 MODEL REFERENCE ADAPTIVE SYSTEM

Model reference adaptive system (MRAS) is an adaptive control method by using performance index of reference model. The reference model is a mathematical model of the ideal system. Block diagram of the MRAS shown in figure below:

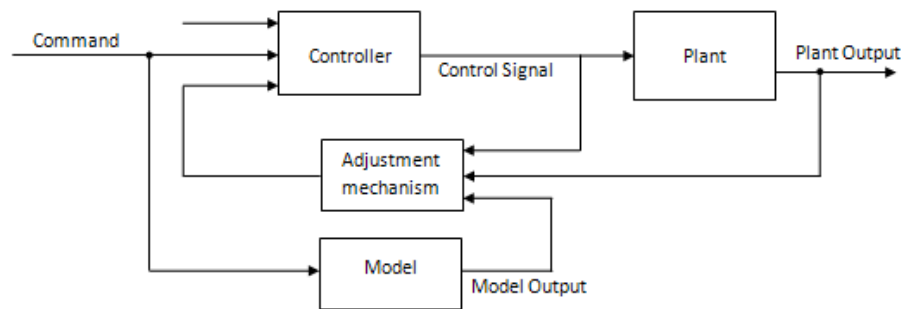


FIGURE 2-43 BLOCK DIAGRAM FOR MODEL REFERENCE ADAPTIVE SYSTEM

2.4.3 STATE-SPACE MODEL OF XX-100 AIRCRAFT

2.4.4 MATLAB AND SIMULINK IMPLEMENTATION

2.4.4.1 IMPLEMENTATION OF MRAS: MATLAB CODE

2.4.4.2 MRAS-COORDINATED TURN SIMULATION: SIMULINK DIAGRAM

2.4.4.3 PLOTTING RESULTS: MATLAB CODE

2.4.5 RESULTS

2.4.6 DISCUSSION OF THE RESULTS

3 FLIGHT SIMULATION

This chapter deals with simulation technique using Matlab and Simulink. We also introduce an advance visualization tools: Virtual Reality toolbox and the application of the tool for visualizing aircraft dynamics related to previous chapter.

3.1 MATLAB AND SIMULINK TOOL FOR SIMULATION

3.1.1 MATLAB COMMAND FOR SIMULATION PURPOSE

3.1.2 SIMULINK TOOLBOX FOR SIMULATION PURPOSE

3.2 VIRTUAL REALITY, AN ADVANCE TOOL FOR VISUALIZATION

3.2.1 INTRODUCTION TO VIRTUAL REALITY TOOLBOX: A USER GUIDE

Virtual Reality (VR) toolbox is already available in MATLAB 6.0 and the higher versions. However, the examples and user guide for the toolbox are available only for simple cases. Therefore, those who want to use the toolbox in advance should study the toolbox in deep themselves. The author has studied the toolbox for hours before using the toolbox for simulating aircraft. MATLAB 7.0 has been used for the simulation. The author suggests the reader to utilize the computer set with minimum specifications of: processor 1 GHz, RAM 256 MB, and Video Memory 32 MB.

The following paragraphs will discuss the detail procedures of using the toolbox, particularly for XW aircraft. The motivation of writing this user guide is to provide comprehensive information to the readers that will give them the skill on using the VR toolbox.

3.2.1.1 DEFINING THE PROBLEM

Our problem is to visualize the motions of XW aircraft in the VR world. The motions defined here are three translational and three rotational motions of the craft. The requirements for this visualization are: the *position* w.r.t. x , y and z axes of VR frame (see Section 3.2) and the *attitude angles* (φ , θ and ψ).

3.2.1.2 CREATE VIRTUAL WORLD

To create a virtual world, the author chose the following steps to achieve the best result:

- (1) Drawing the 3D aircraft and 3D virtual environment in AutoCAD and then export them into 3D Studio file type. It is important to note that the smaller the files size the faster the visualization (simulation) process. The author suggests that to obtain a faster visualization, the files size should not be more than 10 MB.
- (2) Importing the 3D Studio file into V-Realm Builder and then editing the file.
- (3) Saving the virtual world which has been edited using “File\Save As” command. This file will be saved automatically in VRML type (*.wrl). The VRML file is the only file that can be used for VR visualization. **Note: Do not use “Save” command because the file will be saved in original file type (*.3ds) but different format.**

Figure E.1 shows the 3D model of WiSE-8 craft. The 3D model should be drawn in the actual scale and standard dimension (meter) because the VR dimension is in meter. Other important things are: (1) setting the attitude of WiSE craft such that the x_b -axis is coincide with x -axis of UCS world system, y_b -axis is coincide with z -axis of UCS world system and z_b -axis is coincide with negative y -axis of UCS world system. It is important since the UCS world will be defined as the VR frame by V-Realm Builder, (2) setting the origin of UCS coincide with the aircraft center of gravity.

FIGURE 3-1 THE 3D AUTOCAD MODEL OF XW AIRCRAFT

The name for the 3D model of XW craft the author gives is *wise8craft* (.dwg). The author preserves the file name for 3D studio

file. Figure E.2 show the 3D model of virtual lake and hill. The file name of the model is *3D_lake* (.dwg). This name is also preserved for the 3D studio file.

FIGURE 3-2 THE 3D AUTOCAD MODEL OF LAKE AND HILL

3.2.1.3 WORKING IN V-REALM BUILDER

The V-Realm Builder can be executed using one of the following ways:


- (1) Run the `vrbuild2.exe` ( `vrbuild2`) directly. This executable file can be accessed in folder: `MATLAB7\toolbox\vr\vrealm\program`. The V-Realm window will then appear as shown in Figure E.3.
- (2) Open Simulink Library Browser. Then create new model. The new model window will appear. Drag the VR Sink block available in the Virtual Reality Toolbox into the new model window. Double click the block. The parameters window will then appear. Click the new button. This last action will run the `vrbuild2.exe`. Figure E.3 shows the V-Realm window that will be appeared just after the execution.

FIGURE 3-3 THE V-REALM BUILDER WINDOW

After running the V-Realm Builder, the next step is opening the 3D Studio files that have already been created, and then editing the files. The procedures are as follows:



- (1) Click the open button, or choose the menu: File\Open, or push the keyboard buttons: Ctrl + O.
- (2) The open dialog window will then appear. Choose the file type of 3D Studio and file name: wise8craft.3ds.
- (3) The 3D model of WiSE-8 craft will then appear as shown in Figure E.4. Rename 'Group' to 'Wise' by double clicking the word 'Group' and then typing the new name 'Wise'.
- (4) Add a background by clicking the 'Add Background' button (). Figure E.4 shows the result.
- (5) Saving the project using "File\Save As..." command. Name the project as "wise8craftVR.wrl".

FIGURE 3-4 THE 3D STUDIO MODEL OF XW CRAFT AFTER IMPORTED INTO THE V-REALM BUILDER

FIGURE 3-5 THE 3D STUDIO MODEL OF XW CRAFT AFTER A BACKGROUND IS ADDED

- (6) Add four 'Transform' for 'Wise' by clicking the 'Transform' button (). First 'Transform' will be used for translation visualization and the last three 'Transform' will be used for rotation visualization. The 'Transform' should be added such that the second 'Transform' is the child of the first 'Transform' and so on, see Figure E.6.
- (7) Rename each 'Transform' by 'Wise_Translation', 'Wise_Roll', 'Wise_Pitch', and 'Wise_Yaw' as shown in Figure E.7. **Note that**

this action, renaming the transform, is very important because without renaming these parameters will not be identified by SIMULINK.

FIGURE 3-6 ADDING FOUR 'TRANSFORM'

- (8) Move the 'Wise' to the child of the fourth 'Transform', see again Figure E.7. This action can be carried out by the following step: (1) cutting the 'Wise', (2) activate the 'children' by pointing the cursor to 'children' of the fourth 'Transform' or 'Wise_Yaw' and then click once, and (3) click the paste button.

FIGURE 3-7 RENAMING THE FOUR 'TRANSFORM' AND MOVING THE 'WISE'


- (9) Add observer (viewer). An observer can be added into VR world by clicking the Viewpoint button (). It is better to add a 'Transform' first then add an observer as child, see Figure E.8.

FIGURE 3-8 ADDING A DYNAMIC OBSERVER

This action can make the observer become a dynamic observer, in which the observer can be moving and rotating as the aircraft. The observer has six parameters, see again Figure E.8. In this example, only three parameters will be discussed. The parameters are: orientation,

position, and description. The orientation parameter defines the orientation of the observer, see Figure E.9.

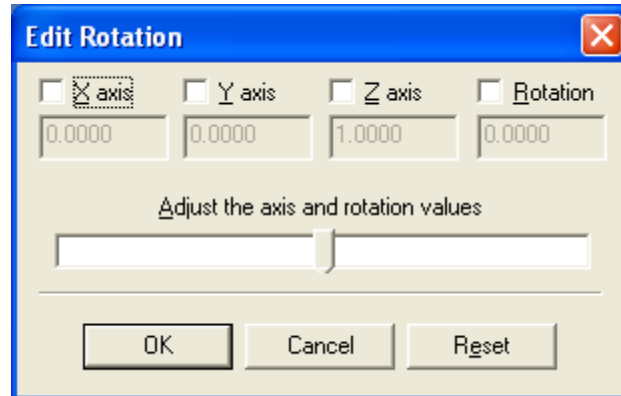


FIGURE 3-9 EDIT ROTATION (ORIENTATION) OF THE OBSERVER

Input for the orientation is X axis, Y axis, Z axis, and Rotation (degree). The X, Y and Z axis define the vector of rotation axes in VR axes system, while the Rotation defines the rotation angle in degree. The position parameter defines the position of the observer. Inputs for the position are X, Y and Z position with respect to VR axes system.

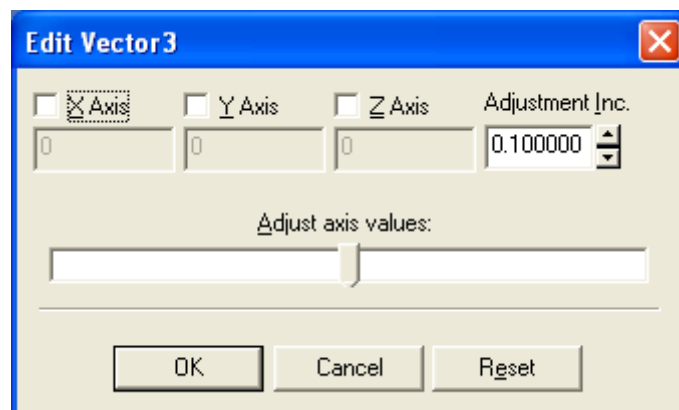


FIGURE 3-10 EDIT POSITION OF THE OBSERVER

The third parameter is description. This parameter defines the observer name, see Figure E.11. Note that the default position is $[0,0,0]$ and the default orientation is along negative Z axis or can be written in vector notation as $[0,0,1,0]$.



FIGURE 3-11 EDIT DESCRIPTION OF THE OBSERVER

In this example, we will show how to make an observer that will flying with the WiSE craft, located at $[20,0,7]$ meter from the craft and the orientation is 73 degrees, see Figure E.12.

FIGURE 3-12 AN EXAMPLE OF AN OBSERVER

To make this observer, set the location as $[20,0,7]$ and orientation as $[0,1,0,73]$. Then name this observer as "Front Right Observer". The result is shown in Figure E.13. Note that the 'Transform' and 'Viewpoint' have been renamed as "RightFront_Observer".

FIGURE 3-13 AN EXAMPLE OF AN OBSERVER, RIGHT FRONT OBSERVER

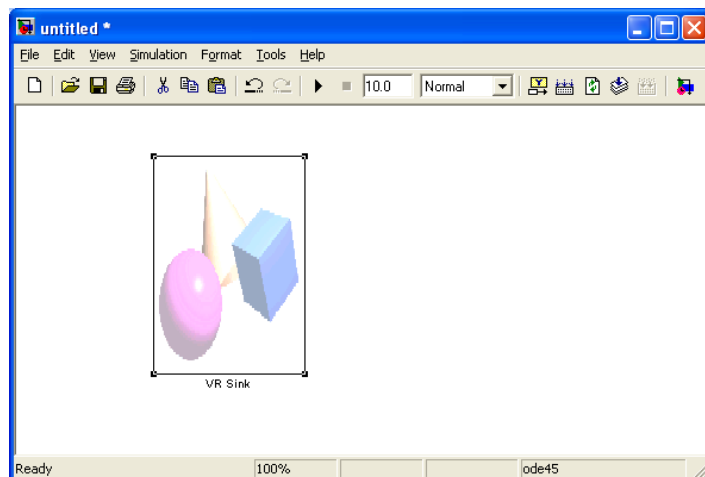
Using the same procedures, we made other observers and add the lake and hill model. The result is shown in Figure E.14.

FIGURE 3-14 FINAL RESULTS OF THE VIRTUAL WORLD

3.2.1.4 PLUGGING THE VR WORLD INTO SIMULINK MODEL

After creating the virtual world (wise8craftVR.wrl), the next step is plugging the world into SIMULINK environment. The following steps show the procedure.

- (1) Open Simulink Library Browser. Then create new model. The new model window will appear. Drag the VR Sink block available in the Virtual Reality Toolbox into the new model window, see Figure E.15.

**FIGURE 3-15** A NEW SIMULINK MODEL WITH VR SINK

- (2) Double click the block. The parameters window will then appear, see Figure E.16. Then click the Browse button. Select the VR file we already made, "wise8craftVR.wrl". The window will show the VRML tree, see Figure E.17.

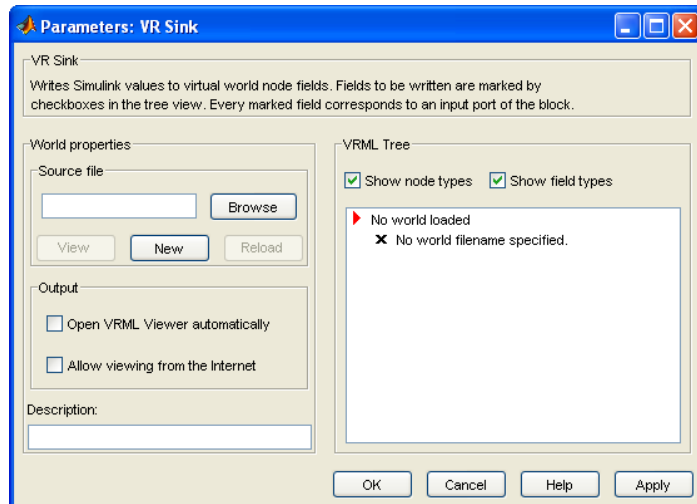


FIGURE 3-16 PARAMETER WINDOW OF VR SINK

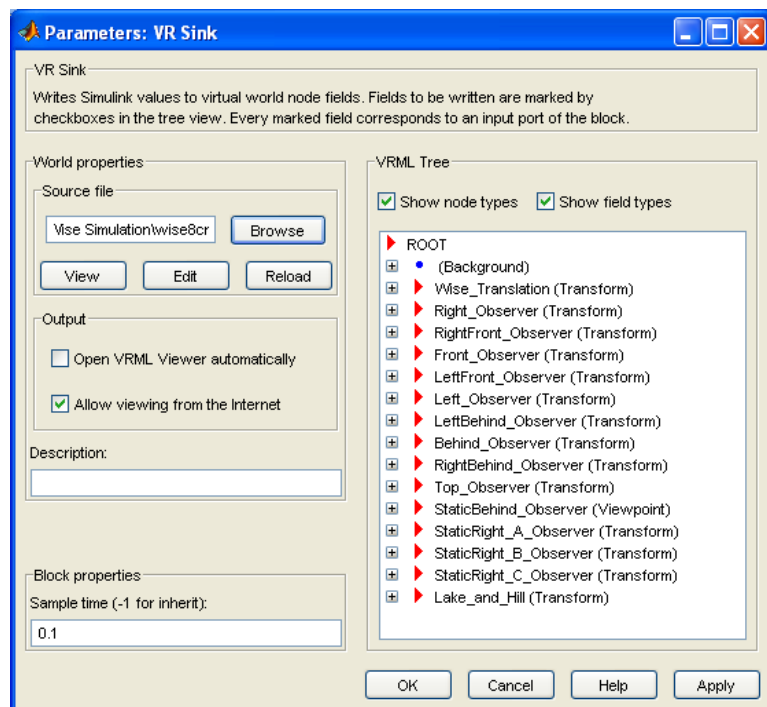



FIGURE 3-17 PARAMETER WINDOW OF VR SINK AFTER LOADING
“WISE8CRAFTVR.WRL”

- (3) Click the OK button. Then we will back to Simulink window as shown in Figure E.15. Double click the VR Sink block, then VR window as shown in Figure E.18 will appear.

FIGURE 3-18 THE VR VISUALIZATION WINDOW OF WISE-8 CRAFT

- (4) Click the Block Parameter button (). This action will show the parameter window as already shown in Figure E.17. In the VRML tree, click the *translation* parameter for Wise_Translation, then click the *rotation* for Wise_Roll, Wise_Pitch, and Wise_Yaw, see Figure E.19. Click also the translation parameter for the Right_Observer, RightFront_Observer, Front_Observer, LeftFront_Observer, Left_Observer, LeftBehind_Observer, Behind_Observer, and RightBehind_Observer. Choose *rotation* parameter for StaticRight_A_Observer, StaticRight_B_Observer, and StaticRight_C_Observer. Finally click OK button.

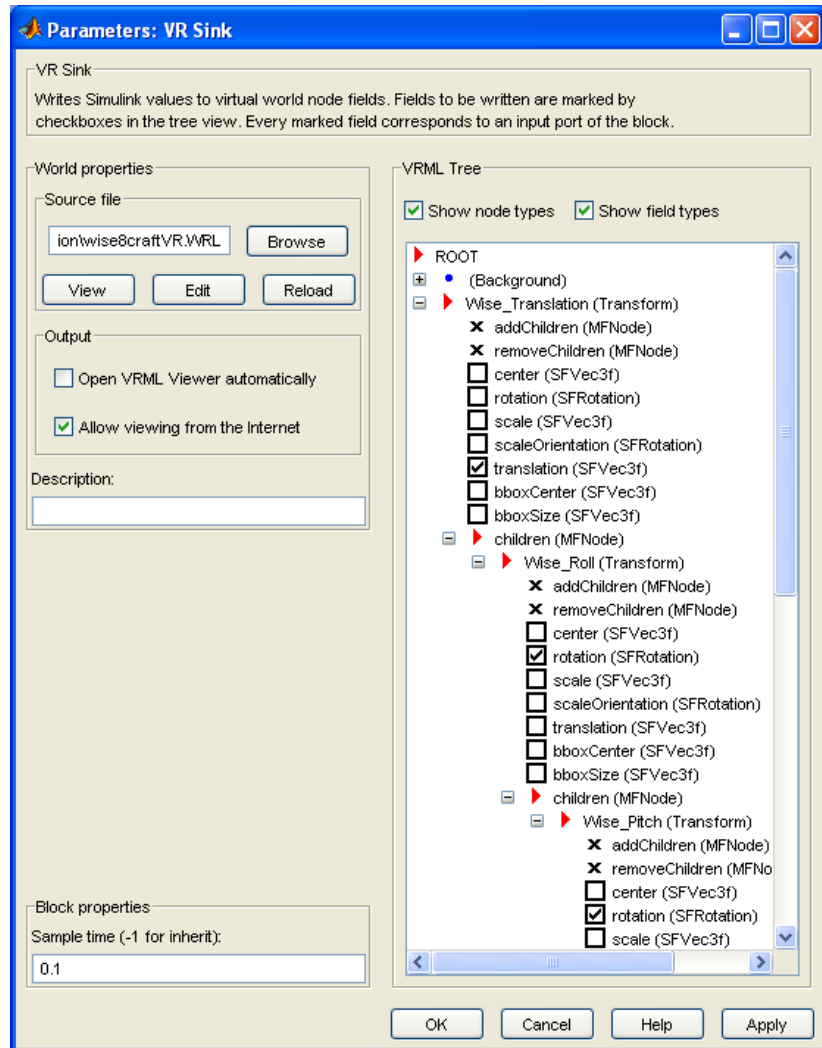


FIGURE 3-19 THE VR PARAMETER AFTER VRML TREE EDITING

- (5) Back to Simulink model window. The VR Sink will show the VR parameter as shown in Figure E.20. Save this Simulink model as Tes_VR_World. Now the VR Sink is ready to be connected to Simulink model of WiSE-8 motion simulation. As already discussed in the beginning of Appendix E, the motion parameters needed for visualization are position and attitude angle.

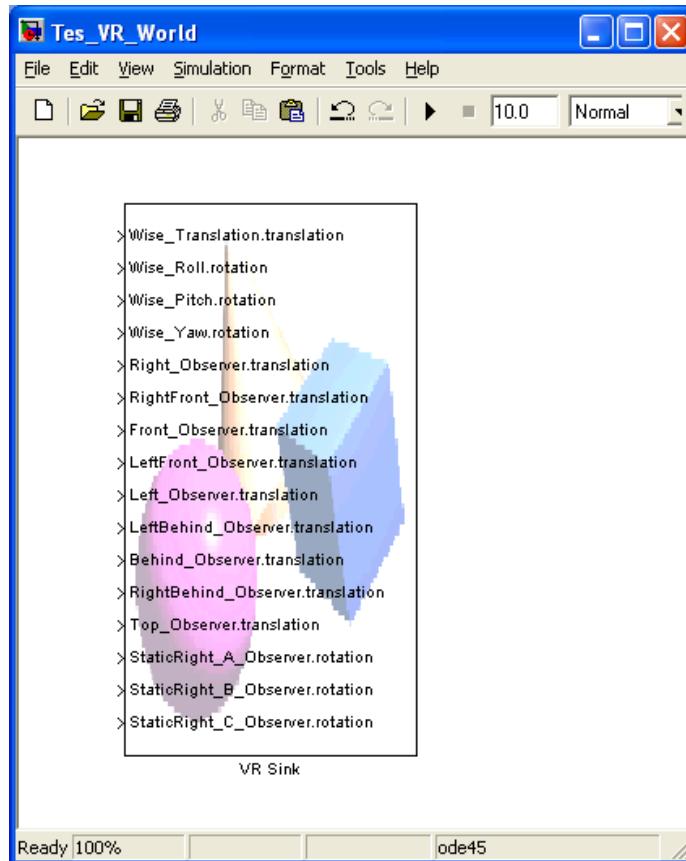


FIGURE 3-20 THE VR SINK AFTER VR PARAMETER EDITING

Before connecting the VR Sink to Simulink model, it is very important to note that (1) the translation input of VR Sink is in the form of vector containing three elements $[X,Y,Z]$, in which each element defines the recent position in meter w.r.t. VR frame, and (2) the rotation input for VR Sink is in the form of vector containing four elements $[X_r,Y_r,Z_r,\phi]$, where X_r , Y_r , and Z_r define the vector of rotation w.r.t. VR frame and ϕ defines the rotation angle in radian. Please note that the unit for rotation input is radian, it is differs from orientation angle input (degree). Since the outputs of WiSE-8 simulation are in local horizon frame, we need to transform the output into VR frame. The transformation matrix from local horizon to VR frame is already shown in Equation (3-1). For convenience, the equation will be rewritten here.

$$C_{VR}^h = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad (3-1)$$

Equation (E-1) is implemented in the VR Transform subsystem as shown in Figure E.21. This figure is similar with Figure 3.28. The different between Figure E.21 and Figure 3.28 is the transformation method. Although the methods are different, they give the same result.

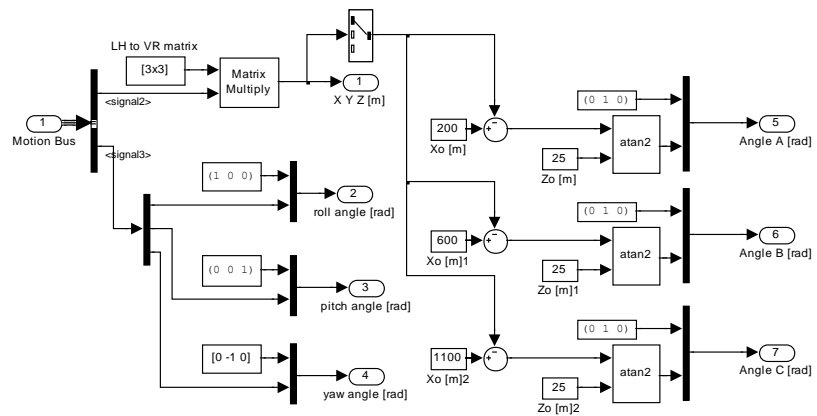


FIGURE 3-21 THE VR TRANSFORM SUBSYSTEM

3.2.1.5 SUMMARY

The procedures of creating VR world have been discussed in detail in this section. The VR world then connected to any aircraft simulation model to visualize the aircraft motion as already discussed in previous chapter. The author hopes that the procedures discussed above give the readers a new knowledge and a guide for using the Virtual Reality toolbox.

3.2.2 VIRTUAL REALITY FOR TRANSPORT AIRCRAFT

3.3 SIMULATION OF AIRCRAFT DYNAMICS: A VIRTUEAIR TRANSPORT CRAFT

APPENDIX A

Quick Matlab Reference: Some Basic Commands

Note: command syntax is case-sensitive!

Help <command>	display the Matlab help for <command>
who	lists all of the variables in matlab workspace
whos	list the variables and describes their matrix size
clear	deletes all matrices (variables) from active workspace
clear u	deletes the matrix or variable u from active workspace
save	saves all the matrices defined in the current session into the file, matlab.mat
load	loads contents of matlab.mat into current workspace
save filename	saves the contents of workspace into filename.mat
save filename x y z	saves the matrices x, y and z into the file titled filename.mat
load filename	loads the contents of filename into current workspace; the file can be a binary (.mat) file or an ASCII file.

Matrix commands

[1 2 3; 4 5 6]	create the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
zeros(n)	creates an nxn matrix whose elements are zero.
zeros(m,n)	creates a m-row, n-column matrix of zeros.
ones(n)	creates a n x n square matrix whose elements are 1's
ones(m,n)	creates a mxn matrix whose elements are 1's.
ones(A)	creates an m x n matrix of 1's, where m and n are based on the size of an existing matrix, A.
zeros(A)	creates an mxn matrix of 0's, where m and n are based on the size of the existing matrix, A.
eye(n)	creates the nxn identity matrix with 1's on the diagonal.
A'	Transpose of A

Plotting commands

plot(x,y)	creates an Cartesian plot of the vectors x & y
plot(y)	creates a plot of y vs. the numerical values of the elements in the y-vector
semilogx(x,y)	plots log(x) vs y
semilogy(x,y)	plots x vs log(y)
loglog(x,y)	plots log(x) vs log(y)
grid	creates a grid on the graphics plot
title('text')	places a title at top of graphics plot
xlabel('text')	writes 'text' beneath the x-axis of a plot
ylabel('text')	writes 'text' beside the y-axis of a plot
text(x,y,'text')	writes 'text' at the location (x,y)
text(x,y,'text','sc')	writes 'text' at point x,y assuming lower left corner is (0,0) and upper right corner is (1,1)
gtext('text')	writes text according to placement of mouse hold on maintains the current

	plot in the graphics window while executing subsequent plotting commands
hold off	turns OFF the 'hold on' option
polar(theta,r)	creates a polar plot of the vectors r & θ where θ is in radians
bar(x)	creates a bar graph of the vector x (Note also the command stairs(y))
bar(x,y)	creates a bar-graph of the elements of the vector y , locating the bars according to the vector elements of ' x ' (Note also the command stairs(x,y))
hist(x)	creates a histogram. This differs from the bargraph in that frequency is plotted on the vertical axis
mesh(z)	creates a surface in xyz space where z is a matrix of the values of the function $z(x,y)$. z can be interpreted to be the height of the surface above some xy reference plane
surf(z)	similar to mesh(z), only surface elements depict the surface rather than a mesh grid
contour(z)	draws a contour map in xy space of the function or surface z
meshc(z)	draws the surface z with a contour plot beneath it
meshgrid	$[X,Y]=\text{meshgrid}(x,y)$ transforms the domain specified by vectors x and y into arrays X and Y that can be used in evaluating functions for 3D mesh/surf plots
print	sends the contents of graphics window to printer
print filename -dps	writes the contents of current graphics to 'filename' in postscript format

Misc. commands

length(x)	returns the number elements in a vector
size(x)	returns the size $m(\text{rows})$ and $n(\text{columns})$ of matrix x
rand	returns a random number between 0 and 1
randn	returns a random number selected from a normal distribution with a mean of 0 and variance of 1
rand(A)	returns a matrix of size A of random numbers
fliplr(x)	reverses the order of a vector. If x is a matrix, this reverse the order of the columns in the matrix
flipud(x)	reverses the order of a matrix in the sense of exchanging or reversing the order of the matrix rows. This will not reverse a row vector!
reshape(A,m,n)	reshapes the matrix A into an $m \times n$ matrix from element (1,1) working column-wise

Some symbolic toolbox commands

syms t	define the variable t to be symbolic. The value of t is now t
$f = t^3 + \sin(t)$	let f be $t^3 + \sin(t)$ symbolically
diff(f)	differentiate f
diff(f,t)	differentiate f with resp. to t
int(f)	integrate f
int(f,t,a,b)	integrate f with resp. to t from a to b
inv(A)	matrix inverse of A
det(A)	determinant of A
rank(A)	rank of A
eig(A)	eigenvalues and eigenvectors.
poly(A)	characteristic polynomial.
expm(A)	matrix exponential
help symbolic	get help on all symbolic toolbox commands.

APPENDIX B

Continuous System Analysis: Some Basic Commands

A. Transfer Function Representation

Commands covered:

```
tf2zp
zp2tf
cloop
feedback
parallel
series
```

Transfer functions are defined in MATLAB by storing the coefficients of the numerator and the denominator in vectors. Given a continuous-time transfer function

$$H(s) = \frac{A(s)}{B(s)}$$

where

$A(s) = a_n s^n + a_{n-1} s^{n-1} + \dots + a_0$ and $B(s) = b_n s^n + b_{n-1} s^{n-1} + \dots + b_0$ written in their coefficients, $A(s)$ in numerator vectors $\text{num} = [1 \ a_{n-1} \dots \ a_0]$ and $B(s)$ is denominator $\text{den} = [b_n \ b_{n-1} \dots \ b_0]$.

In this text,

the names of the vectors are generally chosen to be `num` and `den`, but any other name could be used.

For example,

$$H(s) = \frac{2s + 3}{s^3 + 4s^2 + 5}$$

is defined by

```
num = [2 3];
```

```
den = [1 4 0 5];
```

Note that all coefficients must be included in the vector, even zero coefficients.

A transfer function may also be defined in terms of its zeros, poles and gain:

$H(s) =$

$k(s - z_1)(s - z_2) \dots (s - z_m)$

$(s - p_1)(s - p_2) \dots (s - p_n)$

$1 \ 2 \ m$

$1 \ 2 \ n$

K

K

Error! Switch argument not specified.

To find the zeros, poles and gain of a transfer function from the vectors `num` and `den` which contain the coefficients of the numerator and denominator polynomials, type

```
[z,p,k] = tf2zp(num,den)
```

The zeros are stored in z , the poles are stored in p , and the gain is stored in k . To find the numerator and denominator polynomials from z , p , and k , type

```
11
```

```
G(s)
```

```
G(s)
```

```
H(s)
```

```
unity feedback  
feedback
```

```
-
```

```
-
```

```
[num,den] = zp2tf(z,p,k)
```

The overall transfer function of individual systems in parallel, series or feedback can be found using

MATLAB. Consider block diagram reduction of the different configurations shown in Figure 1.

Store the transfer function G in $numG$ and $denG$, and the transfer function H in $numH$ and $denH$.

To reduce the general feedback system to a single transfer function, $G_{cl}(s) =$

$G(s)/(1+G(s)H(s))$ type

```
[numcl,dencl] = feedback(numG,denG,numH,denH);
```

For a unity feedback system, let $numH = 1$ and $denH = 1$ before applying the above algorithm. Alternately, use the command

```
[numcl,dencl] = cloop(numG,denG,-1);
```

To reduce the series system to a single transfer function, $G_s(s) = G(s)H(s)$ type

```
[numS,denS] = series(numG,denG,numH,denH);
```

To reduce the parallel system to a single transfer function, $G_p(s) = G(s) + H(s)$ type

```
[numP,denP] = parallel(numG,denG,numH,denH);
```

(Parallel is not available in the Student Version.)

```
12
```

```
G(s)
```

```
G(s)
```

```
H(s)
```

```
H(s)
```

```
series  
parallel
```

B. Time Simulations

Commands covered: `residue`

`step`

`impulse`

`lsim`

The analytical method to find the time response of a system requires taking the inverse Laplace

Transform of the output $Y(s)$. MATLAB aides in this process by computing the partial fraction

expansion of $Y(s)$ using the command `residue`. Store the numerator and denominator coefficients of $Y(s)$ in `num` and `den`, then type

```
[r, p, k] = residue(num, den)
```

The residues are stored in `r`, the corresponding poles are stored in `p`, and the gain is stored in `k`.

Once the partial fraction expansion is known, an analytical expression for $y(t)$ can be computed by hand.

A numerical method to find the response of a system to a particular input is available in MATLAB.

First store the numerator and denominator of the transfer function in `num` and `den`, respectively.

To plot the step response, type

```
step(num, den)
```

13

To plot the impulse response, type

```
impz(num, den)
```

For the response to an arbitrary input, use the command `lsim`. Create a vector `t` which contains

the time values in seconds at which you want MATLAB to calculate the response. Typically, this is

done by entering

```
t = a:b:c;
```

where `a` is the starting time, `b` is the time step and `c` is the end time. For smooth plots, choose `b`

so that there are at least 300 elements in `t` (increase as necessary). Define the input `x` as a function of time, for example, a ramp is defined as $x = t$. Then plot the response by typing

```
lsim(num, den, x, t);
```

To customize the commands, the time vector can be defined explicitly and the step response can be

saved to a vector. Simulating the response for five to six time constants generally is sufficient to

show the behavior of the system. For a stable system, a time constant is calculated as $1/\text{Re}(-p)$ where `p` is the pole that has the largest real part (i.e., is closest to the origin).

For example, consider a transfer function defined by

$$H(s) =$$

$$\frac{2}{s+2}$$

$$\frac{4}{s+2}$$

$$\frac{4}{s+2}$$

The step response y is calculated and plotted from the following commands:

```
num = 2; den = [1 2];
```

```
t = 0:3/300:3; % for a time constant of 1/2
```

```
y = step(num, den, t);
```

```
plot(t, y)
```

For the impulse response, simply replace the word `step` with `impz`. For the response to an

arbitrary input stored in `x`, type


```
y = lsim(num, den, x, t);
plot(t, y)
```

C. Frequency Response Plots

Commands covered:

```
freqs
bode
logspace
log10
semilogx
unwrap
```

To compute the frequency response $H(\omega)$ of a transfer function, store the numerator and denominator of the transfer function in the vectors `num` and `den`. Define a vector `w` that contains

the frequencies for which $H(\omega)$ is to be computed, for example `w = a:b:c` where `a` is the lowest

frequency, `c` is the highest frequency and `b` is the increment in frequency. The command `H = freqs(num, den, w)`

returns a complex vector `H` that contains the value of $H(\omega)$ for each frequency in `w`.

To draw a Bode plot of a transfer function which has been stored in the vectors `num` and `den`,

```
type
bode(num, den)
```

To customize the plot, first define the vector `w` which contains the frequencies at which the Bode

plot will be calculated. Since `w` should be defined on a log scale, the command `logspace` is

used. For example, to make a Bode plot ranging in frequencies from 10^{-1} to 10^2 , define `w` by `w = logspace(-1, 2);`

The magnitude and phase information for the Bode plot can then be found by executing:

```
[mag, phase] = bode(num, den, w);
```

To plot the magnitude in decibels, convert `mag` using the following command:

```
magdb = 20*log10(mag);
```

To plot the results on a semilog scale where the y-axis is linear and the x-axis is logarithmic, type

```
semilogx(w, magdb)
```

for the log-magnitude plot and type

```
semilogx(w, phase)
```

for the phase plot. The phase plot may contain jumps of $\pm 2\pi$ which may not be desired. To remove

these jumps, use the command `unwrap` prior to plotting the phase.

```
semilogx(w, unwrap(phase))
```

E. Control Design

Commands covered: `rlocus`

Consider a feedback loop as shown in Figure 1 where $G(s)H(s) = KP(s)$ and K is a gain and $P(s)$

contains the poles and zeros of the controller and of the plant. The root locus is a plot of the roots

of the closed loop transfer function as the gain is varied. Suppose that the numerator and denominator coefficients of $P(s)$ are stored in the vectors `num` and `den`. Then the following command computes and plots the root locus:

```
rlocus(num, den)
```

To customize the plot for a specific range of K , say for K ranging from 0 to 100, then use the following commands:

```
K = 0:100;
```

```
r = rlocus(num, den, K);
```

```
plot(r, 'o')
```

The graph contains dots at points in the complex plane that are closed loop poles for integer values

of K ranging from 0 to 100. To get a finer grid of points, use a smaller increment when defining

K , for example, $K = 0:.5:100$. The resulting matrix `r` contains the closed poles for all of the

gains defined in the vector K . This is particularly useful to calculate the closed loop poles for one

particular value of K . Note that if the root locus lies entirely on the real axis, then using

`plot(r, 'o')` gives inaccurate results.

F. State Space Representation

Commands Covered: `step`

```
lsim
```

```
ss2tf
```

```
tf2ss
```

```
ss2ss
```

The standard state space representation is used in MATLAB, i.e.,

$\dot{x} = Ax + Bu$

$y = Cx$

$= +$

$=$

5

17

where x is $n \times 1$ vector, u is $m \times 1$, y is $p \times 1$, A is $n \times n$, B is $n \times m$, and C is $p \times n$. The response of a system to various inputs can be found using the same commands that are used for transfer function

representations: `step`, `impz`, and `lsim`. The argument list contains the A , B , C , and D

matrices instead of the numerator and denominator vectors. For example, the step response is obtained by typing:

```
[y, x, t] = step(A, B, C, D);
```

The states are stored in `x`, the outputs in `y` and the time vector, which is automatically generated,

is stored in τ . The rows of x and y contain the states and outputs for the time points in τ . Each

column of x represents a state. For example, to plot the second state versus time, type

```
plot( $\tau$ ,  $x(:, 2)$ )
```

To find the response of an arbitrary input or to find the response to initial conditions, use `lsim`.

Define a time vector τ and an input matrix u with the same number of rows as in τ and the number of columns equaling the number of inputs. An optional argument is the initial condition

vector x_0 . The command is then given as

```
[ $y$ ,  $x$ ] = lsim(A, B, C, D, u,  $\tau$ ,  $x_0$ );
```

You can find the transfer function for a single-input/single-output (SISO) system using the command:

```
[num, den] = ss2tf(A, B, C, D);
```

The numerator coefficients are stored in `num` and the denominator coefficients are stored in `den`.

Given a transformation matrix P , the `ss2ss` function will perform the similarity transform.

Store the state space model in A , B , C and D and the transformation matrix in P .

```
[ $A_{bar}$ ,  $B_{bar}$ ,  $C_{bar}$ ,  $D_{bar}$ ] = ss2ss(A, B, C, D, P);
```

performs the similarity transform $z=Px$ resulting in a state space system that is defined as:

$\dot{x} = A_{bar}x + B_{bar}u$

$y = C_{bar}x + D_{bar}u$

$= +$

$= +$

6

where $A_{bar} = PAP^{-1}$, $B_{bar} = PB$, $C_{bar} = CP^{-1}$, $D_{bar} = D$.

REFERENCES

Textbooks

- [1] Brian L. Stevens and Frank L. Lewis, *Aircraft Control and Simulation*, John Wiley and Sons, Inc, 2003.
- [2] Said D. Jenie, *Flight Control*, Lecture Notes, Department of Aeronautics and Astronautics, Bandung Institute of Technology, 2006. (in Bahasa Indonesia)
- [3] Said D. Jenie and Hari Muhammad, *Flight Dynamics*, Lecture Notes, Department of Aeronautics and Astronautics, Bandung Institute of Technology, 2006. (in Bahasa Indonesia)
- [4] John H. Blakelock, *Automatic Control of Aircraft and Missiles*, John Wiley and Sons, Inc, 1991.
- [5] J.A. Mulder, W.H.J.J van Staveren, and J.C. van der Vaart, *Flight Dynamics*, Lecture Notes, Faculty of Aerospace Engineering, TU-Delft, 2000.
- [6] Donald E. Kirk, *Optimal Control Theory an Introduction*, Prentice-Hall, Inc, 1970.

Thesis

- [7] Singgih S. Wibowo, *Virtual Reality of Wing in Surface Effect Craft*, Graduate Thesis, Department of Aeronautics and Astronautics, Bandung Institute of Technology, 2006.
- [8] Singgih S. Wibowo, *Calculation of Aerodynamic Parameter of RX 250 LAPAN Rocket and Analysis of its Dynamics*, Undergraduate Thesis, Department of Aeronautics and Astronautics, Bandung Institute of Technology, 2002. (in Bahasa Indonesia)
- [9] Ony Arifianto, *Adaptive Control Design for Yaw Damper and Turn Coordinator, Case Study: N-250 PA-2 Aircraft*, Undergraduate Thesis, Department of Aeronautics and Astronautics, Bandung Institute of Technology, 1997. (in Bahasa Indonesia)

Paper

- [10] Singgih S. Wibowo, *Optimal Path Tracking Control for Autonomous Helicopter using LQR*, Unpublished Paper, Department of Aeronautics and Astronautics, Bandung Institute of Technology, 2004.

-
- [11] Singgih S. Wibowo, *Effect of Weighing Matrix in Error Tracking for Autonomous Helicopter*, Unpublished Paper, Department of Aeronautics and Astronautics, Bandung Institute of Technology, 2004.
- [12] Singgih S. Wibowo and Hari Muhammad, *Real-Time Simulation with Virtual Reality Visualization*, National Conference in Computational Technology-BPPT, 2006.
- [13] Singgih S. Wibowo, Hari Muhammad and Said D. Jenie, *Simulation with VR Visualization of WiSE Craft during Takeoff Maneuver*, The Sixth Asian Control Conference-Bali, 2006.

Online Resources

- [14] www.control.lth.se/~kursdr/matlab/matlabref.pdf
- [15] www.ee.unlv.edu/kevin/index_files/tutorials/matlab_tutorial.pdf

Other Documents

- [16] MATLAB 7.0 Help Documentation