CONTROL OF A 3DOF BIROTOR HELICOPTER USING ROBUST CONTROL METHODS

A Thesis

by

LUIS ARTURO RUIZ BRITO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2009

Major Subject: Aerospace Engineering

CONTROL OF A 3DOF BIROTOR HELICOPTER USING ROBUST CONTROL METHODS

A Thesis

by

LUIS ARTURO RUIZ BRITO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	Raktim Bhattacharya
	Suman Chakravorty
Committee Member,	Won-Jong Kim
Head of Department,	Dimitris Lagoudas

December 2009

Major Subject: Aerospace Engineering

ABSTRACT

Control of a 3DOF Birotor Helicopter

Using Robust Control Methods. (December 2009)

Luis Arturo Ruiz Brito, B.S., Instituto Politécnico Nacional, Mexico

Co-Chairs of Advisory Committee: Dr. Raktim Bhattacharya
Dr. Suman Chakravorty

The main topic of this thesis is to exhibit how robust control techniques can be applied to real time systems. Presently, the control techniques used in the industry are very simple even when applied to complex systems; these techniques are intuitive and not necessarily systematic. Moreover, the notion of optimality of robustness is absent. Control design procedures are mostly based on SISO techniques, thus, overlooking the intrinsic multivariable aspect of the design that a MIMO system requires.

In this thesis a modern control technique is presented to manipulate a 3DOF birotor helicopter in real time. The objective of this research is to demonstrate the performance of more efficient control algorithms to control these kinds of systems. The robust method proposed in this thesis is an H_∞ controller which exhibits robustness to plant model uncertainties, and good disturbance and noise rejection.

To Esther and Luis Arturo *Para Esther y Luis Arturo*

ACKNOWLEDGMENTS

At the outset, I would like to express my heartfelt thanks to my parents, Esther and Luis Arturo, for all their support and love through the years. Without them, my hopes about aerospace would be but pipe dreams. In addition to the aforementioned, thanks to my sisters, Paulina and Claudia, and my brother, Carlos, and the rest of my family for always being there.

Thanks to Mexico and its people for their support given through CONACYT foundation.

I would like to express my gratitude to Dr. Raktim Bhattacharya, my advisor, for his belief in me and for giving me the opportunity to learn and grow, personally and professionally, by conducting research under his aegis.

Also, a special thanks to Karen Knabe, for all the help given.

I would like to thank Dr. José Alfredo Rosas Flores, someone I greatly admire and to M.S. Miguel Angel Rodríguez Fuentes for always caring for his students.

Thanks also are due to my friends for cheering me up when I needed it the most.

Finally, to the department faculty and staff of Texas A&M University, thanks for their support, specially to the Sponsor Students Office whose people made my life easier in this country.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1

II DYNAMICAL MODEL 2

1. Introduction 2
2. Translational and rotational matrices 3
3. Translational and angular velocities 6
4. Kinetic and potential energy 7
5. Power in 8
6. Equations of motion 9
7. Linearized model 11
8. Controllability and observability 14
9. Conclusion 15

III PROTOTYPE 16

1. Introduction 16
2. Mechanical design 16
3. Electrical design 18
4. Interface 22
5. Conclusion 23

IV PARAMETER ESTIMATION 27

1. Introduction 27
2. Motor operation 27
3. Center of mass 30
4. Nonlinear least squares estimation 31
5. Conclusion 38

V LQR DESIGN 39

1. Introduction 39
2. Feedback model 39
3. LQR problem 40
4. Conclusion 41

VI H_∞ CONTROLLER 44

CHAPTER Page

1. Introduction 44
2. Scaled state space 44
3. Range space and null space 45
4. Norms of systems 46
5. Linear matrix inequalities 48
6. Feedback model 51
7. The H_∞ problem 52
8. Controller loop shape 53
9. H_∞ synthesis 59
10. Conclusion 64

VII IMPLEMENTATION 65

1. Introduction 65
2. MIMO frequency response 66
3. Tustin's method 68
4. Frequency warping 69
5. Conclusion 72

VIII CONCLUSION 74

REFERENCES 75

APPENDIX A 77

APPENDIX B 80

APPENDIX C 86

APPENDIX D 112

APPENDIX E 134

APPENDIX F 148

VITA 149

LIST OF FIGURES

FIGURE		Page
2.1	Euler angles yaw ψ, pitch θ and roll ϕ.	2
2.2	Mechanical model.	3
2.3	Mechanical model with axis and angles.	4
2.4	Open loop simulation 1.	10
2.5	Open loop simulation 2.	11
2.6	Typical state space model.	13
3.1	Ducted fan brushless motor.	16
3.2	Basic brushless motor.	17
3.3	Wireless link diagram.	18
3.4	Bluetooth integrated circuit.	19
3.5	Magnetic encoder.	19
3.6	Analog output operation.	19
3.7	Sensor range.	20
3.8	PIC microcontroller.	20
3.9	Speed controller.	21
3.10	Compact RIO controller.	21
3.11	Li-Po battery.	22
3.12	Ni-MH battery.	22
3.13	User interface 1.	23

FIGURE Page

3.14 Flow of data. 23

3.15 User interface 2. 24

3.16 Fully assembled plant. 25

3.17 Fully assembled plant. Side view. 25

3.18 Fully assembled plant. Back view. 26

3.19 Fully assembled plant. Front view. 26

4.1 Speed controller operation. 28

4.2 Birotor SEC operation. 28

4.3 Motor experiment. 29

4.4 Bytes vs Thrust. 29

4.5 Center of mass experiment. 30

4.6 Center of mass experiment diagram. 31

4.7 Bytes and pitch angle relation. 32

4.8 l'_c location for different values of θ. 32

4.9 Nonlinear Least Squares Algorithm. 36

4.10 Pitch estimates. 37

4.11 Roll estimates. 38

5.1 LQR feedback model. 39

5.2 LQR simulation, yaw ψ. 41

5.3 LQR simulation, pitch θ. 42

5.4 LQR simulation, roll ϕ. 42

5.5 LQR simulation, left thrust. 43

FIGURE Page

5.6 LQR simulation, right thrust. 43

6.1 General feedback model. 51

6.2 Standard H_∞ problem. 52

6.3 Controller K design block. 54

6.4 W model. 55

6.5 W command. 56

6.6 W performance 1. 57

6.7 W performance 2. 57

6.8 W actuator. 58

6.9 H_∞ simulation, errors. 62

6.10 H_∞ simulation, positions. 62

6.11 H_∞ simulation, roll angle and velocities. 63

6.12 H_∞ simulation, thrust. 63

7.1 H_∞ implementation diagram. 65

7.2 H_∞ controller frequency response. 67

7.3 H_∞ controller pole-zero map. 71

7.4 H_∞ implemented, ψ yaw. 72

7.5 H_∞ implemented, θ pitch 72

7.6 H_∞ implemented, ϕ roll 73

7.7 H_∞ implemented, thrust 73

CHAPTER I

INTRODUCTION

The main topic of this thesis is to exhibit how robust control techniques can be applied to real time systems. Nowadays, the control techniques used in the industry are very simple even when applied to complex systems, these techniques are intuitive and not necessarily systematic. More over, the notion of optimality of robustness is absent.

In this thesis a modern control technique is presented to manipulate a 3DOF birotor in real time. The objective of this research is to demonstrate the performance of more efficient control algorithms to control these kind of systems. The required tasks to achieve this goal, are to develop a mathematical model of the system that incorporates all its characteristics, to generate an adequate robust control algorithm that can be used with conventional hardware and show that it works in real time applications.

The journal model is *IEEE Transactions on Automatic Control.*

CHAPTER II

DYNAMICAL MODEL

1. Introduction

In order to make a good controller for a given system, equations of motion that describe the dynamics of the system must be formulated, taking in account all the characteristics that affect the system. This birotor helicopter is a three degree of freedom (3DOF) system that does not perform translational motion, for this reason, the dynamics will be focused only on the rotational equations of motion.

There are three angles commonly used in aerodynamics applications which are called the Euler angles; yaw ψ, pitch θ and roll ϕ[1]. Fig. 2.1 shows a schematic of these angles for any aircraft. Yaw is the heading about the vertical axis (body-z), pitch is known as the rotation about an axis (body-y) perpendicular to the longitudinal plane of symmetry and roll is the angle about the longitudinal axis (body-x).

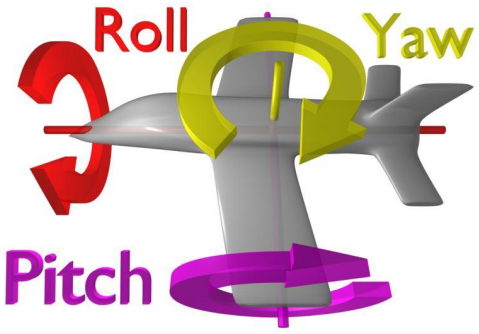


Fig. 2.1. Euler angles yaw ψ, pitch θ and roll ϕ.

Fig. 2.2 shows a computer aided design (CAD) of the mechanical model. The fan located at the back is used to generate disturbances. The Euler angles are controlled by the thrust provided by the front rotors, specifically left and right.

The pitch motion is controlled by increasing the thrust from the rotors, the roll and yaw motions are controlled by the difference in the thrust between the rotors.

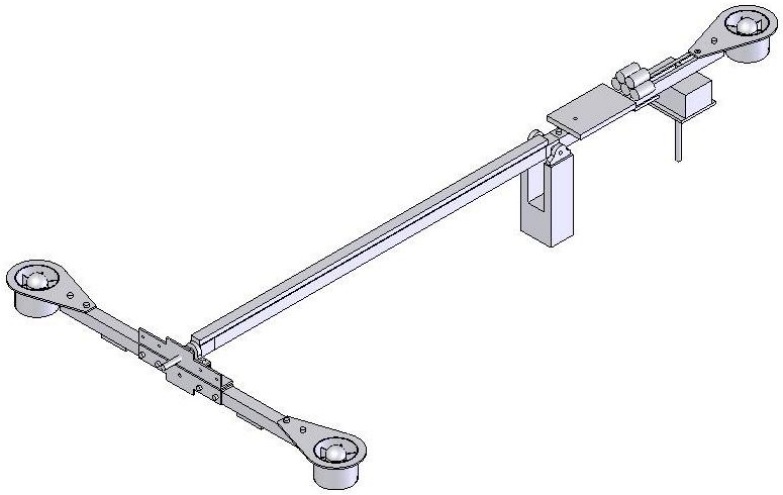


Fig. 2.2. Mechanical model. A CAD drawing of the actual system displaying the main aspects of the plant.

In this chapter the equations of motion will be derived. These equations describe the motion of the system due to changes in the forces applied from the rotors.

2. Translational and rotational matrices

In fig. 2.3 a simple model is shown displaying the axis and direction of the angles used to derive the equations of motion. Let us denote l_2 as the distance from O_2 to O_3, l_3 as the distance from O_3 to F_r or F_l, l_c as the distance from O_2 to the pitch center of mass (COM), F_l is the left force, and F_r is the right force.

Let O_0 be the origin of the stationary frame and O_3 the point where the third reference frame is located and also the middle point between the two rotors. Let O_1 and O_2 be the origins of the intermediate links, these are located in the same place as O_0. A vector $q = [\xi\ \eta]^T$ denotes the generalized coordinates, where $\xi = [x\ y\ z]^T \in \mathcal{R}^3$ denotes the translational coordinates with respect to the origin, and $\eta = [\psi\ \theta\ \phi]^T \in \mathcal{R}^3$

describes the orientation given by the Euler angles[2]. The values for ψ, θ and ϕ depicted in fig. 2.3 are the zero position. That is, when l_3 is aligned with positive body-x, and l_2 is parallel to body-y the system is considered to be in the origin, $[\psi\ \theta\ \phi] = [0\ 0\ 0]$.

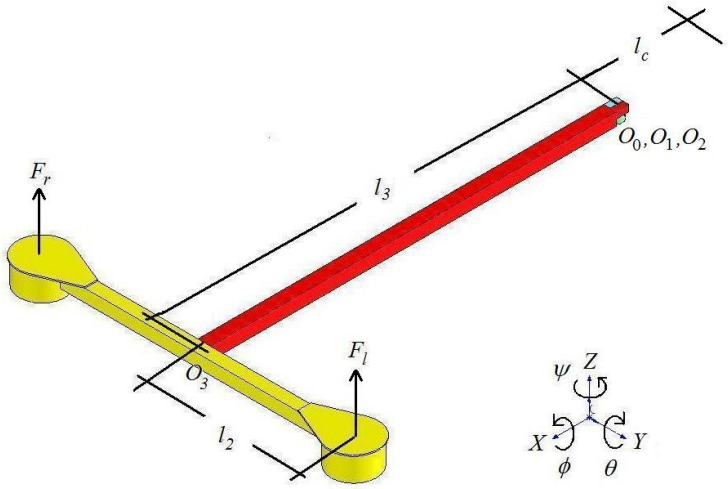


Fig. 2.3. Mechanical model with axis and angles. This picture displays the plant, the coordinate system (x,y,z), and the positive direction of the Euler angles.

The translational and rotational matrices are given by transformation matrices of the form:

$$\mathcal{T}^{i\to j} = \left[\begin{array}{c|c} \mathcal{R}^{i\to j} \in \mathcal{R}^3 & \mathcal{P}^{i\to j} \in \mathcal{R}^{3\times 1} \\ \hline 0 \in \mathcal{R}^{1\times 3} & 1 \end{array}\right] \tag{2.1}$$

where:

i	index of the reference frame O_i.
j	index of the reference frame O_j.
$\mathcal{R}^{i\to j} \in \mathcal{R}^3$	Rotational matrix $O_i \to O_j$.
$\mathcal{P}^{i\to j} \in \mathcal{R}^{3\times 1}$	Translational matrix $O_i \to O_j$.

Given this, the tranformation matrices for this system are:

Roll to pitch[1]:

$$\mathcal{T}_{rp} = \begin{bmatrix} 1 & 0 & 0 & l2 \\ 0 & c_\phi & -s_\phi & 0 \\ 0 & s_\phi & c_\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Pitch to yaw:

$$\mathcal{T}_{py} = \begin{bmatrix} c_\theta & 0 & s_\theta & 0 \\ 0 & 1 & 0 & 0 \\ -s_\theta & 0 & c_\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Yaw to ground:

$$\mathcal{T}_{yg} = \begin{bmatrix} c_\psi & -s_\psi & 0 & 0 \\ s_\psi & c_\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Other transformation matrices can be obtained by combinations of eq. 2.2, eq. 2.3 and eq. 2.4.

$$\mathcal{T}_{pg} = \mathcal{T}_{yg}\mathcal{T}_{py} \text{ Pitch to Ground} \tag{2.5}$$

$$\mathcal{T}_{rg} = \mathcal{T}_{pg}\mathcal{T}_{rp} \text{ Roll to Ground} \tag{2.6}$$

$$\mathcal{T}_{ry} = \mathcal{T}_{py}\mathcal{T}_{rp} \text{ Roll to Yaw} \tag{2.7}$$

$$\mathcal{T}_{Frg} = \mathcal{T}_{rg}\mathcal{T}_{Fl} \text{ Left force to Ground} \tag{2.8}$$

$$\mathcal{T}_{Flg} = \mathcal{T}_{rg}\mathcal{T}_{Fr} \text{ Rigth force to Ground} \tag{2.9}$$

[1] $c_i = \cos(i)$ and $s_i = \sin(i)$.

And:

$$\mathcal{T}_{Fl} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & l_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ Left force to roll} \tag{2.10}$$

$$\mathcal{T}_{Fr} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -l_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ Right force to roll} \tag{2.11}$$

The translational matrices are:

$$\mathcal{P}_p = \mathcal{T}_{pg} \begin{bmatrix} l_c & 0 & 0 & 1 \end{bmatrix}^T \text{ Pitch COM Position} \tag{2.12}$$

$$\mathcal{P}_r = \mathcal{T}_{pg} \begin{bmatrix} l_2 & 0 & 0 & 1 \end{bmatrix}^T \text{ Roll COM Position} \tag{2.13}$$

$$\mathcal{P}_{Fl} = \mathcal{T}_{rg} \begin{bmatrix} 0 & l_3 & 0 & 1 \end{bmatrix}^T \text{ Left Force Position} \tag{2.14}$$

$$\mathcal{P}_{Fr} = \mathcal{T}_{rg} \begin{bmatrix} 0 & -l_3 & 0 & 1 \end{bmatrix}^T \text{ Right Force Position} \tag{2.15}$$

Eq. 2.12 to eq. 2.15 are vectors in $\mathcal{R}^{4\times1}$, but only the first 3 outputs corresponding to $\xi = [x\ y\ z]^T$ are needed, the fourth element will be neglected.

3. Translational and angular velocities

The velocities or rates are the derivatives of the translational and rotational matrices with respect of time, for translational movement:

$$\dot{\xi} = \frac{d\mathcal{P}^{i\to j}}{dt} \tag{2.16}$$

And for rotation, the angular velocity is:

$$\Omega = W\dot{\eta} = \begin{bmatrix} W_x \\ W_y \\ W_z \end{bmatrix} \in \mathcal{R}^3 \tag{2.17}$$

where $\dot{\eta} = (\dot{\psi}, \dot{\theta}, \dot{\phi})^T$. The values of W_x, W_y and W_z are obtained from:

$$\begin{bmatrix} 0 & -W_z & W_y \\ W_z & 0 & -W_x \\ -W_y & W_x & 0 \end{bmatrix} = (\mathcal{R}^{i\to j})^{-1}\frac{d\mathcal{R}^{i\to j}}{dt} \tag{2.18}$$

The translational velocity must be calculated for all the points described in eq. 2.12 to eq. 2.15. The angular velocities for $\mathcal{R}_{py}$ (eq. 2.3) and $\mathcal{R}_{ry}$ (eq. 2.7) are evaluated.

4. Kinetic and potential energy

The kinetic and potential energy are considered for the two COM points in the system (pitch and roll), see eq. 2.12 and eq. 2.13. The kinetic energy is the sum of the translational kinetic energy and rotational kinetic energy, $KE = K_{translational} + K_{rotational}$.

$$KE_p = \frac{1}{2}(M_p\dot{\xi}_p^T\dot{\xi}_p + \Omega_p^T I_c \Omega_p) \tag{2.19}$$

$$KE_r = \frac{1}{2}(M_r\dot{\xi}_r^T\dot{\xi}_r + \Omega_r^T I_{c2} \Omega_r) \tag{2.20}$$

$$KE = KE_p + KE_r \tag{2.21}$$

The subindexes p and r represent the pitch COM and roll COM respectively. M_p is the mass of the link that goes from O_2 to O_3 and M_r is the mass of the link that goes from F_l to F_r passing through O_3. I_c and I_{c2} are the inertia tensor matrices which are calculated at the COM points of pitch and roll.

The parallel axis theorem must be used to get the values of the inertias at the "total" COM of the system, eq. 2.23, m is the COM point mass and R is the distance from the COM point to the "total" COM.

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \tag{2.22}$$

$$I_{displaced} = I_{center} + mR^2 \tag{2.23}$$

The potential energy is the energy that is stored within a system. It exists when there is a force that tends to pull an object back towards some original position when the object is displaced. In this case it only depends on the z coordinate affected by g, the gravity.

$$PE = M_p g \mathcal{P}_p^T \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T + M_r g \mathcal{P}_r^T \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{2.24}$$

5. Power in

The power administered is generated by the input forces. These forces are always perpendicular to the surface of the fans, thus, the position of them at any point is considered. As with eq. 2.12, only the first 3 elements of these vectors are required.

$$F_{left} = F_l \left(\mathcal{T}_{rg} \begin{bmatrix} 0 & l_3 & 1 & 1 \end{bmatrix}^T - \mathcal{T}_{rg} \begin{bmatrix} 0 & l_3 & 0 & 1 \end{bmatrix}^T \right) \tag{2.25}$$

$$F_{right} = F_r \left(\mathcal{T}_{rg} \begin{bmatrix} 0 & -l_3 & 1 & 1 \end{bmatrix}^T - \mathcal{T}_{rg} \begin{bmatrix} 0 & -l_3 & 0 & 1 \end{bmatrix}^T \right) \tag{2.26}$$

The power that will be provided to the system is:

$$P_{in} = F_{left} \dot{\xi}_{Fl} + F_{right} \dot{\xi}_{Fr} \tag{2.27}$$

6. Equations of motion

Applying Euler-Lagrange formalism, the equations of motion are:

$$\frac{d}{dt}\frac{d\mathrm{L}}{d\dot{q}} - \frac{d\mathrm{L}}{dq} + D = F \tag{2.28}$$

From eq. 2.28, L is the Lagrangian, defined by:

$$\mathrm{L} = KE - PE \tag{2.29}$$

$\dot{q}$ are the derivatives with respect of time of q, $\dot{q} = [\dot{x}\ \dot{y}\ \dot{z}\ \dot{\psi}\ \dot{\theta}\ \dot{\phi}]$. Since the model is focused on the rotational equations of motion we can discard the 3 first equations corresponding to translational movement.

D are the damping forces and F are the input forces applied on the system, d is the damping factor vector $d = [d_\psi\ d_\theta\ d_\phi]^T$ and $\mathcal{I}$ is the identity in $\mathcal{R}^3$. Later it will be shown how to determine the damping coefficients performing parameter estimation.

$$D = \mathcal{I}d(\Omega_p + \Omega_r) \tag{2.30}$$

$$F = \frac{dP_{in}}{d\dot{\eta}} \tag{2.31}$$

Eq. 2.28 can be written as:

$$\mathrm{EL} = \frac{d}{dr}\frac{d\mathrm{L}}{d\dot{q}} - \frac{d\mathrm{L}}{dq} + D - F \tag{2.32}$$

The inertia matrix is defined as:

$$M(\eta) = \frac{d\mathrm{EL}}{d\ddot{\eta}} \tag{2.33}$$

An auxiliary matrix H is specified as:

$$H = M(\eta)\ddot{\eta} - \mathrm{EL} \tag{2.34}$$

Finally, the equations of motion for $\ddot{\eta} = [\ddot{\psi}\ \ddot{\theta}\ \ddot{\phi}]^T$ are:

$$\ddot{\eta} = \begin{bmatrix} \ddot{\psi} \\ \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} = M(\eta)^{-1}H \tag{2.35}$$

An open loop simulations were done to verify the equations of motion, as shown in fig. 2.4 and fig. 2.5, the system oscillates towards an equilibrium position[2] as $t \to \infty$.

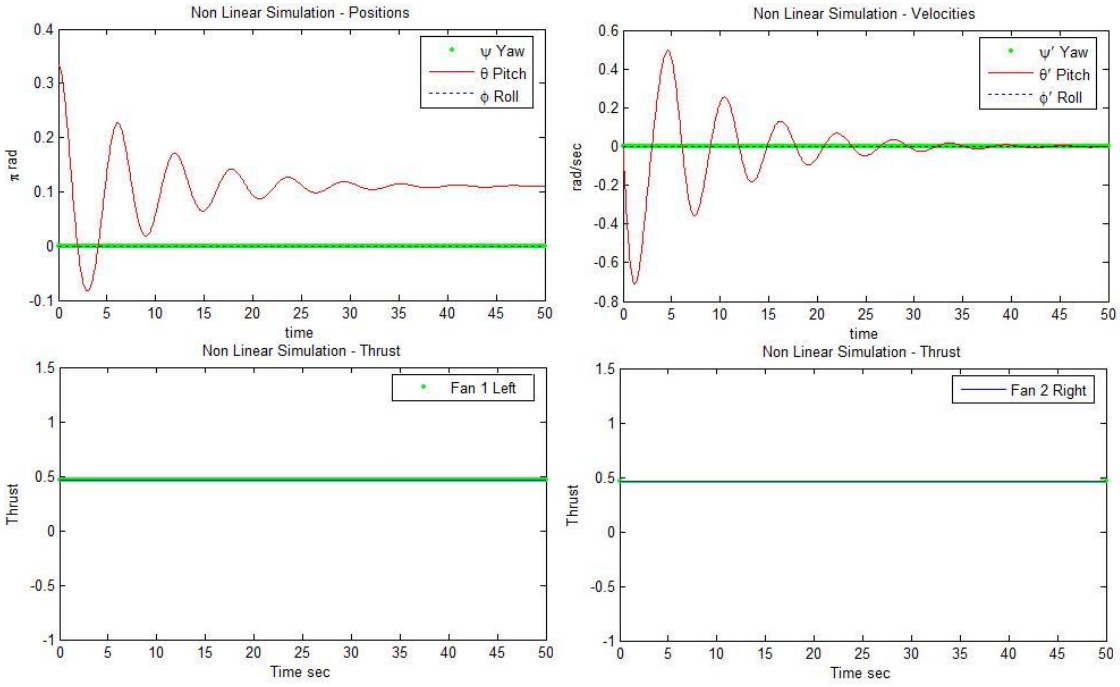


Fig. 2.4. Open loop simulation 1. As $t \to \infty$ the system returns to an equilibrium position. $[\psi_0, \theta_0, \phi_0] = [0, pi/3, 0]$

[2]Equilibrium position or stationary point, is the condition of a system in which competing influences are balanced, is an input to a function where the derivative is zero, that means, the gradient is zero, the function stops increasing or decreasing, hence the name.

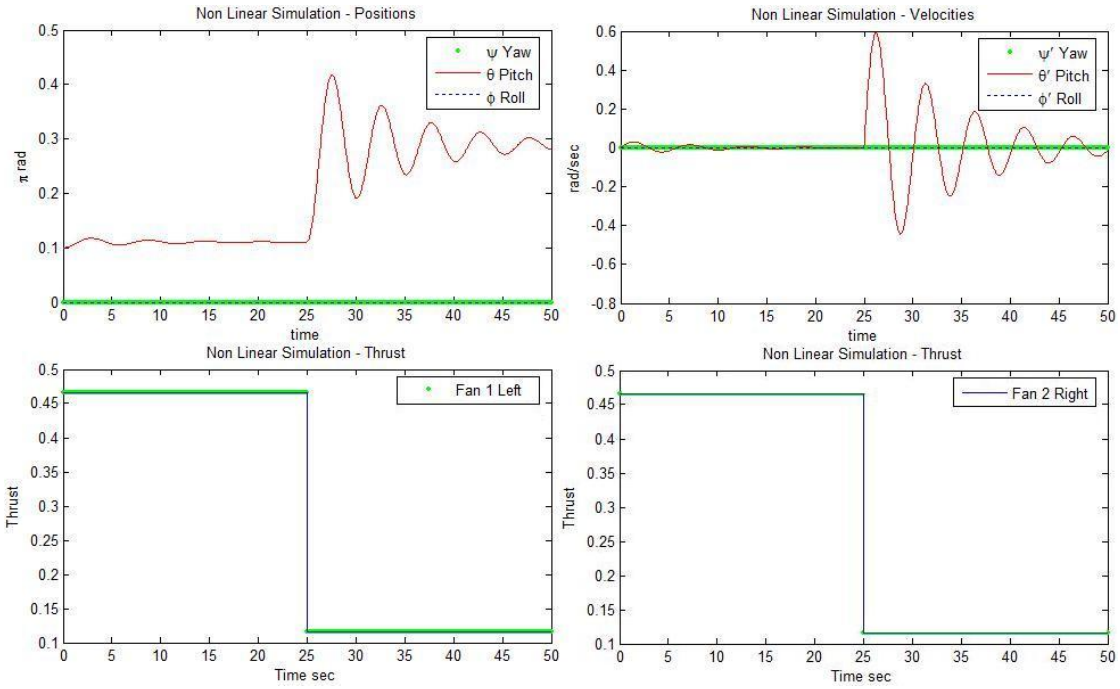


Fig. 2.5. Open loop simulation 2. $[\psi_0, \theta_0, \phi_0] = [0, pi/10, 0]$

7. Linearized model

The nonlinear model of the system is given by eq. 2.35. Nonlinear models are difficult to solve and complex for real time applications, simple changes in one part of the system can produce complex effects in the output, thus, for control purposes linear models are assumed to model the dynamics of the system to an acceptable extent. Due to the inherent nonlinearity of real world phenomena, the nonlinear models[3] might be linearized.

The linearization is done while considering an equilibrium point as the datum. The performance of this linear model depends on the accuracy of the nonlinear representation.

Eq. 2.35 can be transformed into an equation of the form $\dot{x} = f(x, u)$ were x is the states vector $[\eta\ \dot{\eta}]^T = [\psi\ \theta\ \phi\ \dot{\psi}\ \dot{\theta}\ \dot{\phi}]^T$ and u is the input vector $[Fl\ Fr]^T$.

$$\dot{x} = \begin{bmatrix} \dot{\eta} \\ f(x,u) \end{bmatrix} \tag{2.36}$$

$\dot{x}$ is linearized around an equilibrium or trim point, and for any point c in space, the linearization is:

$$f(x) \cong\ f(c) + \nabla f|_{x=c}(x - c) \tag{2.37}$$

where:

$f(x)$	linearized function.
$f(c)$	function evaluated at the equilibrium point c.
∇	$\nabla = \left[\frac{\partial}{\partial x_1}\ \cdots\ \frac{\partial}{\partial x_n}\right]$.

The equilibrium point $[\psi\ \theta\ \phi\ \dot{\psi}\ \dot{\theta}\ \dot{\phi}]^T = [0\ 0\ 0\ 0\ 0\ 0]^T$ is selected. Also the input forces are taken in account when calculating the linear model. From eq. 2.35 there are three equations, setting $[\psi\ \theta\ \dot{\psi}\ \dot{\theta}\ \dot{\phi}]^T = [0\ 0\ 0\ 0\ 0]^T$ and solving for ϕ, F_l and F_r.

ϕ is taken as an unknown to have a set of three unknowns in three equations and finally the trim point is the vector $[\psi\ \theta\ \phi\ \dot{\psi}\ \dot{\theta}\ \dot{\phi}\ F_l\ F_r]^T$.

Once linearized, $\dot{x}$ can be represented in state space form:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{2.38}$$

x is the state vector $[\psi\ \theta\ \phi\ \dot{\psi}\ \dot{\theta}\ \dot{\phi}]^T$. y is the output vector defined by the variables and/or properties desired from the system. u, the input vector $[F_l\ F_r]^T$. A is the state matrix. The input matrix B. C is called the output matrix. And finally D is the feedfoward matrix.

The state space is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations. The state variables are the smallest possible subset of system variables that can represent the entire state of the system at any given time. The interconnection of the state space is depicted in fig. 2.6.

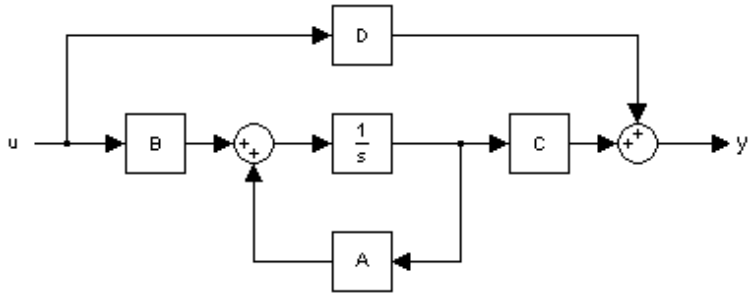


Fig. 2.6. Typical state space model. The relation between the states, matrices, inputs and outputs is shown.

Commonly, the state space representation of the equations of motion of a given system is also named the plant G.

$$G = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{2.39}$$

The values of the matrices A, B, C and D for the birotor helicopter are depicted from eq. 2.40 to eq. 2.43.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1.3763 & 0.004 & 0 & -0.0034 \\ 0 & -0.7548 & 0 & 0 & -0.2283 & 0 \\ 0 & 0 & 0.8084 & -0.2941 & 0 & 0.002 \end{bmatrix} \tag{2.40}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -0.269 & 0.269 \\ -1.1301 & -1.1301 \\ 19.8274 & -19.8274 \end{bmatrix} \tag{2.41}$$

$$C = \mathcal{I}^{6\times 6} \tag{2.42}$$

$$D = 0^{6\times 2} \tag{2.43}$$

From here, the rank of the plant G is defined as the order of the system, the number of states, so $\text{rank}(G) = n_x = 6$.

8. Controllability and observability

A important property of a stable controller is that the closed loop transfer function is asymptotically stable, all the eigenvalues have negative real part, as long as the following two conditions hold:

1. The plant is controllable.

2. The plant is observable.

A system is said to be controllable if and only if it is possible, to transfer the system from any state x_0 to any other state x_t in a finite time $0 \leq t < \infty$. If the system is linear and time invariant, then the system is controllable if and only if the controllability matrix (eq. 2.44), is full rank.

$$\mathcal{Q} = \left[\; B \;\middle|\; AB \;\middle|\; \cdots \;\middle|\; A^{n_x-1}B \; \right] \tag{2.44}$$

A system is said to be observable if and only if its state x_0, at any time t, can be determined from knowledge of the input and output over a finite period of time t_f, that is, $u(t)$ and $y(t)$, where $0 \leq t \leq t_f$. A linear, time invariant system is observable if and only if the observability matrix (eq. 2.45), is full rank.

$$\mathcal{N} = \begin{bmatrix} \frac{C}{CA} \\ \hline \cdots \\ \hline CA^{n_x-1} \end{bmatrix} \tag{2.45}$$

In both cases, controllability and observability, the corresponding matrices are full rank, so the controllers to be designed can be stable.

9. Conclusion

In this section, the equations of motion were derived. Also, the state space representation of the equivalent linear model was obtained. Finally, it was proved that the linear model is controllable and observable, thus, a stable controller can be designed. The mathematical model of any system is a crucial step towards control it.

CHAPTER III

PROTOTYPE

1. Introduction

The prototype was built based on a existing model but with the advantage of full 360 degrees rotation in the yaw ψ direction. This was accomplished by establishing a wireless link between the control device and the mechanical plant. Speed and accurary were considered at the time of making a choice of the components to be impelmented in the system. The control algorithm has to be executed on a real time device to show the perfomance of the controller designed[1], so the prototype must be able to execute it efficiently.

2. Mechanical design

The mechanical plant was built mainly with aluminium, with joint elements that would allow the system to perform in a very acceptable manner compared to the mathematical model.

Fig. 3.1. Ducted fan brushless motor.

[1]For more information that the one provided in this section please refer to the apendix.

The actuators selected are ducted fan brushless motors (Fig. 3.1). These motors are essentially AC synchronous motors with permanent magnet rotors and can maintain a load on them more efficiently that brushed motors. The motors are controlled with a speed controller (SEC, see next section).

Brushless motor rotation relies on the same theory as for AC and DC motors. That is, two magnetic fields interact, which result in motion.

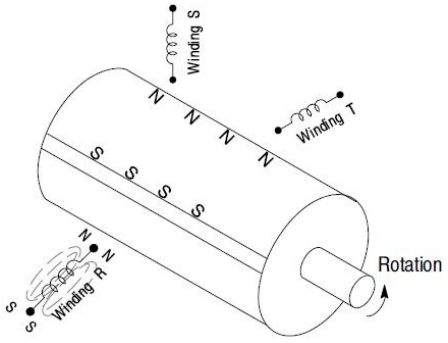


Fig. 3.2. Basic brushless motor. Movement is produced by changing the polarity in the poles, north (N) and south (S).

In the case of AC motors, the stator winding sets up one magnetic field while inducing the second interacting field onto the squirrel cage rotor. With DC motors, the permanent magnet stator sets up the first magnetic field, and the rotor windings produce the second field. These two magnetic fields interacting, results in rotation. In the DC motor, the two fields try to align. However the commutator continually switches power from winding to winding. Thus, maintaining the two magnetic fields at a 90 degree relationship. If they did indeed align, motor rotation would not occur.

Compared to DC motors, brushless technology has been termed an "inside out" design. That is, the permanent magnets are on the rotor, and the stator consists of windings. The design still consists of two magnetic fields interacting[4]. To begin to understand how brushless motor operate, refer to fig. 3.2. Power is applied to winding

"R" and current flow sets up a 'north' pole which the permanent magnet will react to, and begin movement. This movement will cease when the "south' pole of the magnet aligns it.

Typically the motors have conections for power supply and one connection for control input. This control input is a square wave sending the frequency at which the motor has to operate. The control of the motor can be implemented almost effortless with the aid of software and electronics.

3. Electrical design

Since there is not commercial products satisfying all the electrical requieriments of the system, embeded systems had to be built. The electrical diagrams of these embeded systems can be found in the apendix.

As mention above, there is a wireless link between the controller and the plant. A sketch showing the signals affected by the wireless link is depicted in fig. 3.3. The control signal u and the sensor signal y are broken and transmited via serial communication in a bidirectional channel.

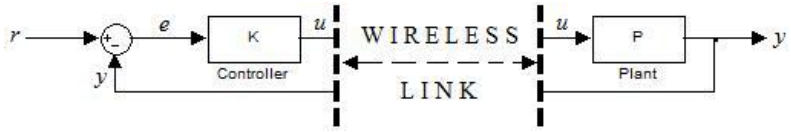


Fig. 3.3. Wireless link diagram. The signals u and y are broken and transmited wirelessly.

The wireless connection was achieved using Bluetooth® technology, see fig. 3.4. These kind of products granted the ability to have bidirectional connection between the control device and the mechanical part using only one accessory on each side.

The sensors operating are absolute encoders, more especific, magnetic encoders, see fig. 3.5. These enjoy the advantage of transmit the information in a single line as

a 10 bit resolution analog output that is linearly proportional to the absolute shaft position, thus saving space and complexity as shown in fig. 3.6.

Fig. 3.4. Bluetooth integrated circuit. This device is the heart of the communication between the controller and the plant.

Fig. 3.5. Magnetic encoder.

Fig. 3.7 depicts a circle ilustrating the working range of the sensor. Now, if the signal from the sensor were to transit from the red area to the blue area in one control time step, the system has done 1 revolution in the positive direction and one revolution in the negative direction if it transits from the blue to the red area.

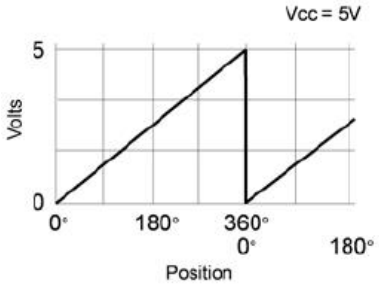


Fig. 3.6. Analog output operation. The operation of the encoder is linear as shown in this picture, with the aid of software the range can be extended practically to infinity.

Also, being the sensors absolute encoders they must be initialized, that is, the initial signal read, called the offset, has to be zero or some reference point to make the system agree with the coordinate system in the mathematical model.

$$Signal_{reference} = Signal_{real} - Signal_{offset} \tag{3.1}$$

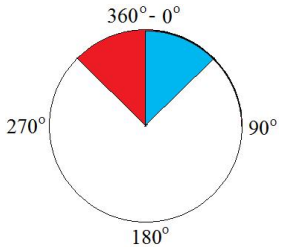


Fig. 3.7. Sensor range. This picture helps to understand how the angle measured from the sensor can be extended by sensing the transition from 0 to 360 degrees in one time step.

Microchip® PIC micontrollers[5], see fig. 3.8, were used to extract the data from the encoders, and recieve the data from the control device to apply input voltage to the actuators in the system. Also the PICs communicate with the Bluetooth modules.

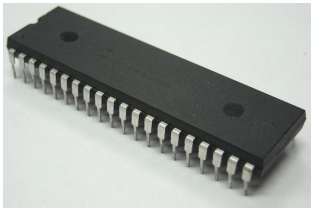


Fig. 3.8. PIC microcontroller.

The PIC sends a signal to a speed controller (fig. 3.9), that generates a square signal to move the motors. This controller has a working period of 20 ms, thus being the device that specifies the sampling time for real time applications.

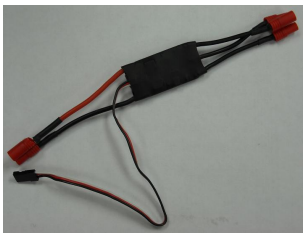


Fig. 3.9. Speed controller. The working period of the SEC is the base period for real time applications.

The control device is an emmbeded control National Intruments® Compact Rio® controller model 9014 depicted in fig. 3.10. This device supports real time capabilities as well as computer based interfaces with acceptable speed. The Compact RIO communicates with the plant via serial protocol by means of the Bluetooth connection.

Fig. 3.10. Compact RIO controller.

The birotor uses 3 lithium polymer (LiPo) batteries (fig. 3.11) in which the lithium-salt electrolyte is not held in an organic solvent as in the lithium-ion (Li-ion) design, but in a solid polymer composite such as polyethylene oxide or polyacrylonitrile. The advantages of LiPo over the Li-ion design include lower cost manufacturing and being more robust to physical damage. Li-poly has a greater life cycle degradation rate. These batteries are used to turn on the motors.

Since no metal battery cell casing is needed, the battery can be lighter and it can be specifically shaped to fit the device it will power. Because of the denser packaging

without intercell spacing between cylindrical cells and the lack of metal casing, the energy density of Li-poly batteries is over 20% higher than that of a classic Li-ion batteries.

Fig. 3.11. Li-Po battery.

The helicopter also uses 2 nickel-metal hydride cell (Ni-MH) batteries (fig. 3.12) to power up the electronic boards responsible for the wireless connection. Both kind of batteries guaratee that the current rate delivered at all times will be enough to keep the helicopter running efficiently.

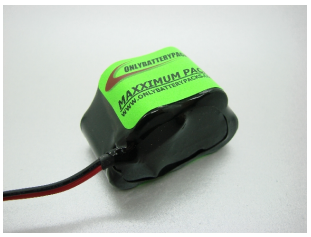


Fig. 3.12. Ni-MH battery.

4. Interface

The user interface was done keeping in mind that every person can test the device, with the safety requirements necessary for the good perfomance of the model. LabView® software is the main plataform of the interface[6]. It is widely aproved in robotics applications.

A shot of the user interface can be seen in fig. 3.13 and fig. 3.15.

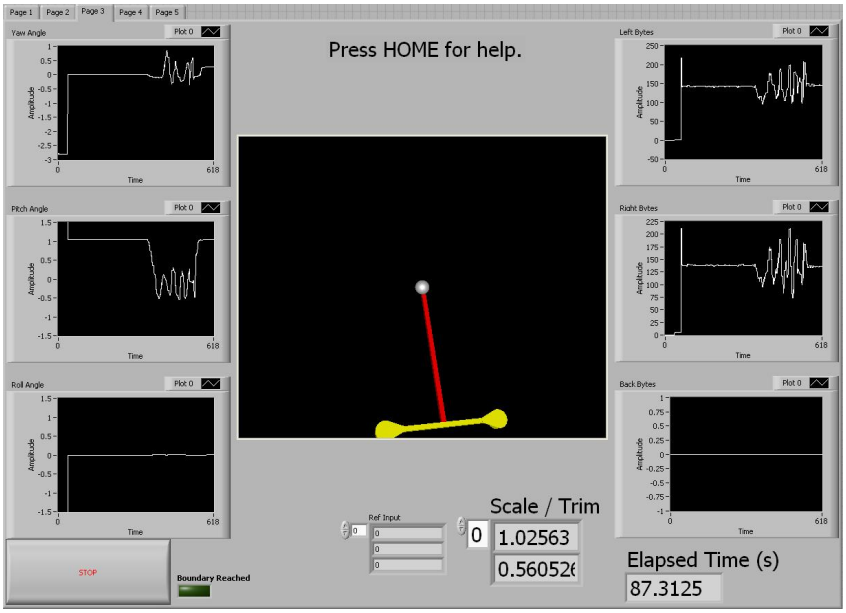


Fig. 3.13. User interface 1. User interface with 3D graph used for close loop simulations.

Fig. 3.14 depicts the flow of data.

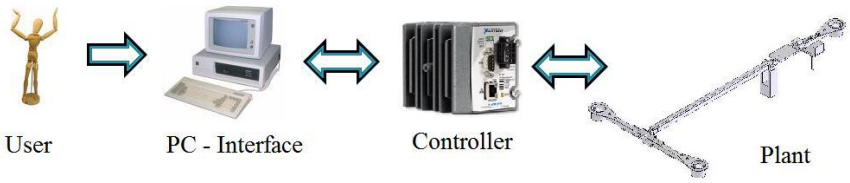


Fig. 3.14. Flow of data. The user inputs commands to a PC-interface, the PC communicates with the real time controller which sends and recieves information from the plant.

5. Conclusion

The good performance of the mathematical model and controls algorithms depends tremendously in how good the prototype represents the model. If a mechanical or

electrical part of the system does not work properly the control techniques will never achieve the performance aimed.

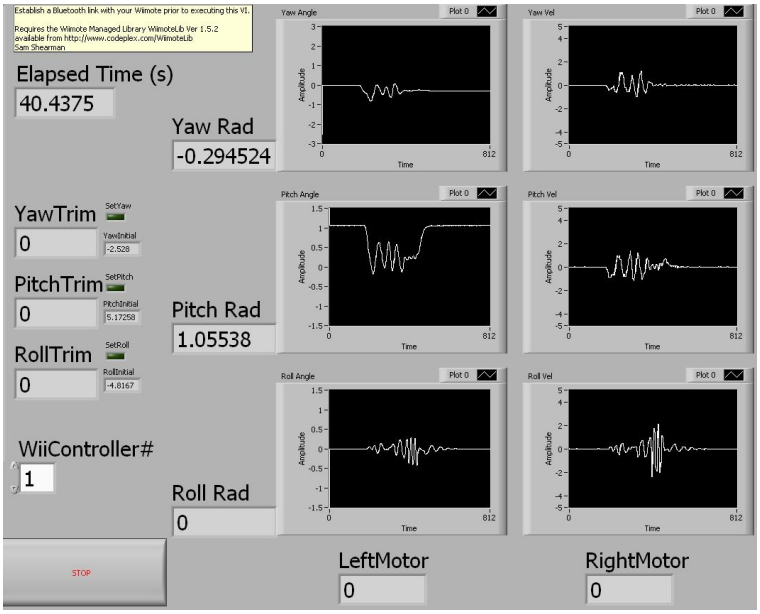


Fig. 3.15. User interface 2. User interface used to get data from sensors and open loop simulations.

Fig. 3.16 to fig. 3.19 show the plant fully assembled.

Fig. 3.16. Fully assembled plant. Above one can see the fully assembled plant, even though there are some issues like loose cables, the system performs in a very acceptable manner and according to the requirements.

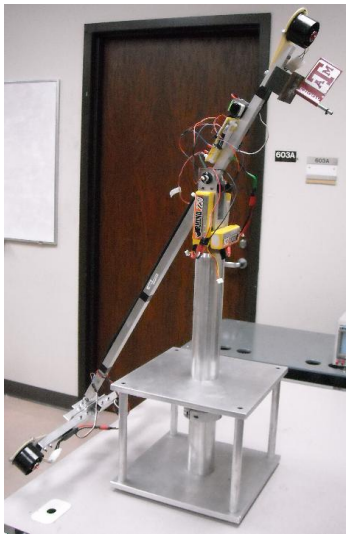


Fig. 3.17. Fully assembled plant. Side view.

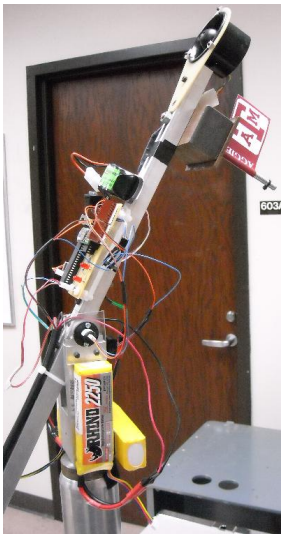


Fig. 3.18. Fully assembled plant. Back view.

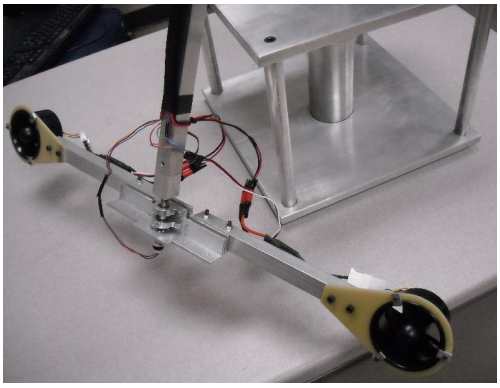


Fig. 3.19. Fully assembled plant. Front view.

CHAPTER IV

PARAMETER ESTIMATION

1. Introduction

Once the dynamical model has been developed, some parameters need to be estimated. In this section parameter estimation is done, which determines the "best" estimates of all poorly known parameters so that the mathematical model provides an accurate representation of the system's actual behavior[7].

For the dynamical system, a mathematical model is hypothesized based upon the experience of the investigator, which is consistent with whatever physical laws known to govern the systems behavior, the number and nature of the available measurements, and the degree of accuracy desired. Such mathematical models almost invariably embody a number of poorly known parameters.

Also the model is verified with some straightfoward experiments.

2. Motor operation

The motors used in the system are ducted fan brushless motors, these motors are controlled by sending a byte signal from 0 to 255 to a PIC, then the PIC generates a PWM signal that is sent to a speed controller and this one genetares a square wave to control the motors.

The speed controller (SEC) operation is depicted in fig. 4.1, the base period of the SEC is 20 ms, and increments of 1μs are allowed. The fisrt 1ms is required to turn on the motor. And then by increments of 1μs the SEC increases the PWM, the maximum number of increments is 1000. The birotor does not require all 1000 increments to make the system functional.

A zero is set at 107 and a maximum of 107 + 255 is set, see fig. 4.2. As seen in fig. 4.1 there are free 18ms which are used to send signal to other SECs in the system, thus making possible to control up to 8 motors at one time.

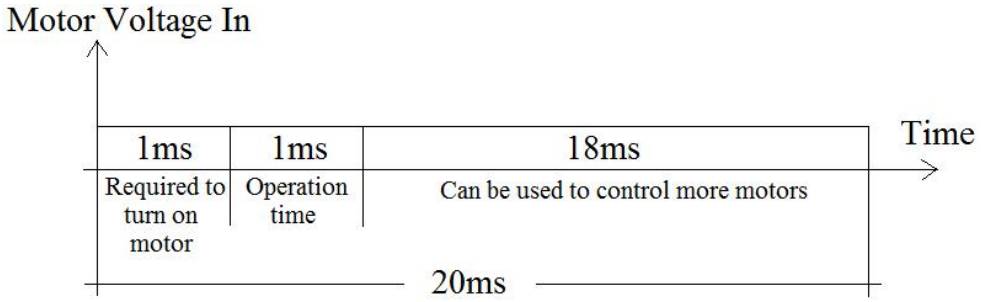


Fig. 4.1. Speed controller operation. 1ms is needed to turn on the motor and increments of 1μs set the output of the PWM signal.

A relation between the bytes sent to the PIC and the thrust generated by the motor is built up. For this purpose the experiment illustrated in fig. 4.3 was done. With the aid of a scale by turning off one of the motors and applying force in the other the relation bytes-thrust is acquired.

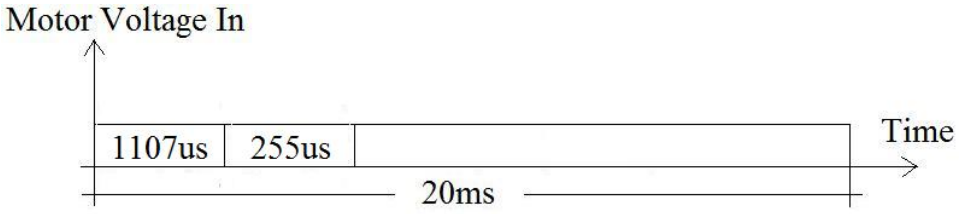


Fig. 4.2. Birotor SEC operation. A zero position is set at 1107μs and a maximum of 255 increments are allowed.

The results are interpolated to get a linear function between the values, see fig. 4.4. This relation is used later to get a transfer function from bytes sent to the PIC to thrust applied in the system.

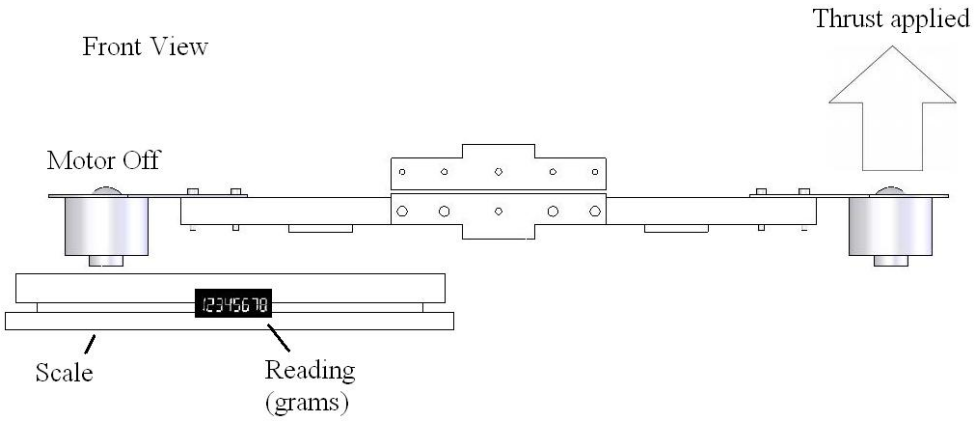


Fig. 4.3. Motor experiment. One motor is turned off and with the help of a scale the thrust provided for every byte increment is recorded.

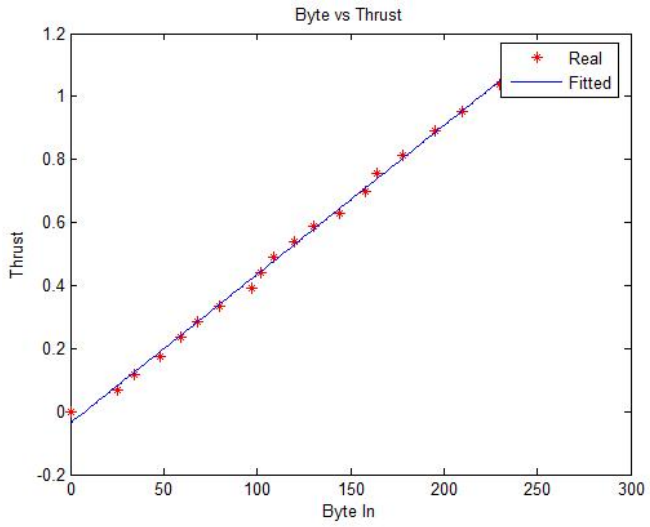


Fig. 4.4. Bytes vs Thrust. The linear relation input - output can be seen on this plot.

3. Center of mass

With the aid of CAD software the center of mass is calculated, name it l_c and l'_c, to check that this value is correct the experiment in fig. 4.5 was performed. By changing both values at the motors at the same time and with the same proportion, the pitch angle θ changes.

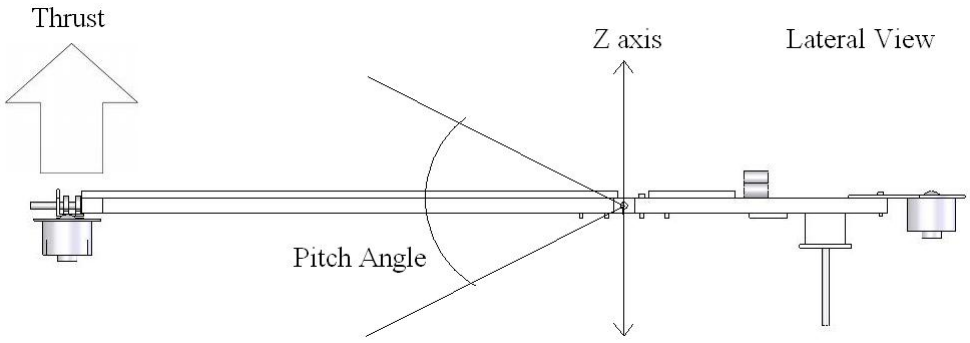


Fig. 4.5. Center of mass experiment. By changing the pitch angle θ and using eq. 4.1 the values of the center of mass can be derived.

The equation that relates pitch and force is:

$$2FL = mg(l_c \cos(\theta) - l'_c \sin(\theta)) \tag{4.1}$$

As depicted in fig. 4.6, F is the force applied in each motor, L is the length from origin to the point where forces are applied, m is the total mass of the system, the gravity is represented by g, l_c is the distance perpendicular to the force from O to cm and l'_c is the distance parallel to the force from O to cm. θ is the pitch angle and cm the center of mass.

A set of values are obtained for different inputs. Then when $\theta = 0$ eq. 4.1 becomes:

$$2FL = mgl_c \tag{4.2}$$

And the value of l_c can be obtained.

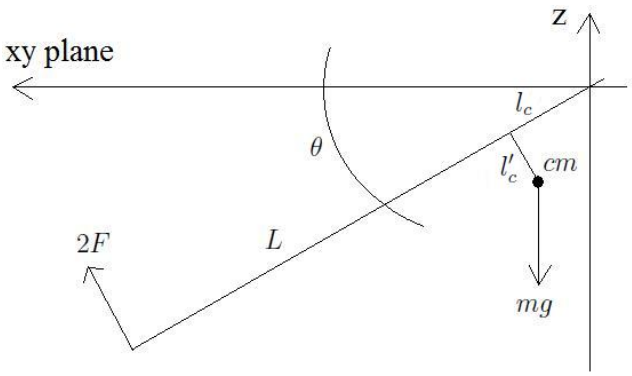


Fig. 4.6. Center of mass experiment diagram. At any given angle $\sum(\text{Torques}) = 0$.

Then, from eq. 4.1:

$$l'_c = \frac{-\frac{2FL}{mg} + l_c \cos(\theta)}{\sin(\theta)} \tag{4.3}$$

Fig. 4.7 shows a graph of bytes input vs pitch angle θ. Knowing the value of l_c the value of l'_c is acquired for some values of θ, then the mean is taken to aproximate the value of l'_c, see fig. 4.8.

Finally the real values of the center of mass and the values got from model are presented, in meters:

	Real	Model
l_c	0.0340	0.0338
l'_c	-0.0184	-0.0203

The error is around 2mm, which is good enough.

4. Nonlinear least squares estimation

The damping factors of the system are the variables to be estimated. Due to the nonlinear nature of the system, nonlinear parameter estimation is needed. The most widely used successive approximation procedure, nonlinear least squares; otherwise known as Gaussian least squares differential correction is used.

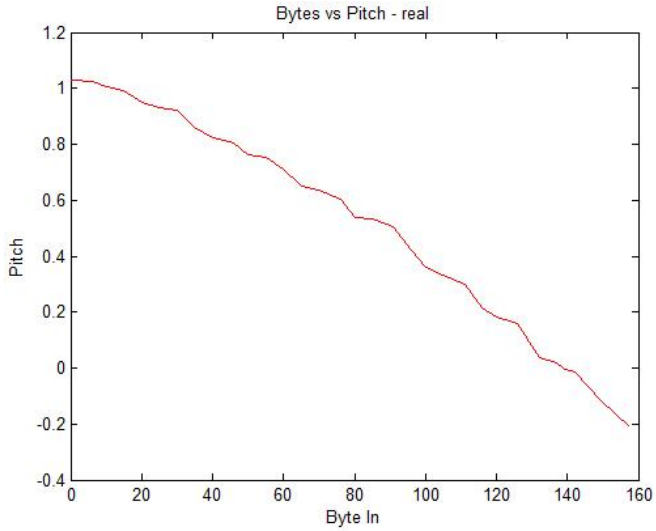


Fig. 4.7. Bytes and pitch angle relation. This graph was obtained from experiments, the effect of trigonometric functions can be seen.

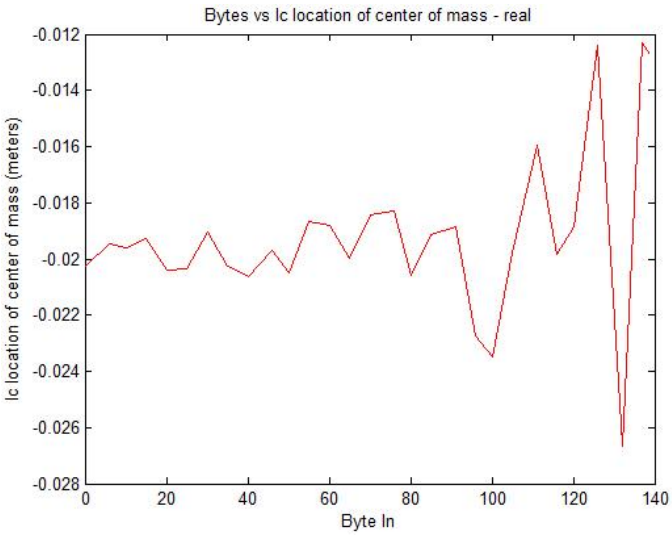


Fig. 4.8. l'_c location for different values of θ.

The method to be developed here is an m x n generalization of Newton's root solving method for finding x-values satisfying $y - f(x) = 0$. As with Newton's method convergence of the multi-dimensional generalization is guaranteed only under rather strict requirements on the functions and their first two partial derivatives as well as on the closeness of the starting estimates.

Assume m observable quantities modelled as:

$$y_j = f_j(x_1, x_2, ..., x_n);\ j = 1, 2, ..., m;\ m \geq n \tag{4.4}$$

where f_j are m arbitrary independent functions of the unknown parameters x_i.

f_j and at least its first partial derivatives are required to be single-valued, continuous and at least once differentiable. Additionally, suppose that a set of observed values of the variables y_j are available. $y_j \in \{y_1, y_2, ..., y_n\}$.

An estimate $\hat{x}$ for x that minimizes eq. 4.5 is the goal of the estimation algorithm. e is the residual errors $\tilde{y} - f(\hat{x}) = \Delta y$. W is a m x m weighting matrix, relative importance of measurements. The measured values $\tilde{y}$ and the vector of estimated parameters $\hat{x}$.

$$J = \frac{1}{2} e^{\mathrm{T}} W e \tag{4.5}$$

The current estimates of the unknown x-values are supposed to be available, denoted by:

$$x_c = [x_{1c}, x_{2c}, ..., x_{nc}]^T \tag{4.6}$$

Whatever the unknown objective x-values $\hat{x}$ are, they are related to their respective current estimates, x_c, by an also unknown set of corrections, Δx, as:

$$\hat{x} = x_c + \Delta x \tag{4.7}$$

If the components of x are sufficiently small, it may be possible to solve for approximations to them and thereby update x_c with an improved estimate of x from eq. 4.7. $f(\hat{x})$ is linearized about x_c using a first-order Taylor series expansion as:

$$f(\hat{x}) = f(x_c) + H\Delta x \tag{4.8}$$

where:

$$H = \left.\frac{\partial f}{\partial x}\right|_{x_c} \tag{4.9}$$

The gradient matrix H is known as a Jacobian matrix. The measurement residual "after the correction" can now be linearly approximated as:

$$\Delta y = \tilde{y} - f(\hat{x}) \approx \ \tilde{y} - f(x_c) - H\Delta x = \Delta y_c - H\Delta x \tag{4.10}$$

The residual "before the correction" is:

$$\Delta y_c = \tilde{y} - f(x_c) \tag{4.11}$$

Recall that the objective is to minimize the weighted sum squares, J. The local strategy for determining the approximate corrections ("differential corrections") in x is to select the particular corrections that lead to the minimum sum of squares of the linearly predicted residuals J_p:

$$J = \frac{1}{2}\Delta y^T W \Delta y - f(\hat{x}) \approx \ J_p \equiv \frac{1}{2}(\Delta y_c - H\Delta x)^T W (\Delta y_c - H\Delta x) \tag{4.12}$$

The minimization of J_p is equivalent to the minimization of J. If the process is convergent, then Δx determined by minimizing would be expected to decrease on successive iterations until (on the final iteration) the linearization is an extremely good approximation.

Any algorithm for solving the weighted least squares problem directly applies to solving for Δx in eq. 4.12. Therefore:

$$\Delta x = (H^T W H)^{-1} H^T W \Delta y_c \tag{4.13}$$

The complete nonlinear least squares algorithm is summarized in fig. 4.9. An initial guess x_c is required to begin the algorithm. A stopping condition with an accuracy dependent tolerance for the minimization of J is given by:

$$\delta J \equiv \frac{|J_i - J_{i-1}|}{J_i} < \frac{\epsilon}{\|W\|} \tag{4.14}$$

ϵ is a prescribed small value. If eq. 4.9 is not satisfied, then the update procedure is iterated with the new estimate as the current estimate until the process converges, or unsatisfactory convergence progress is evident, a maximum allowed number of iterations is exceeded, or J increases on successive iterations.

The experiment to collect the measured data for nonlinear least squares consisted in making the system oscillate around an equilibrium position, several data sets were gathered for pitch and roll, the damping factor for yaw will be taken as the same for roll. The function to be estimated is a damped cosine function:

$$f(x) = A + Be^{-\zeta t}\cos(w_n t + \Phi) \tag{4.15}$$

From eq. 4.15, A is the dc gain, B is the cosine amplitude, ζ is the damping ratio, w_n is the frequency and Φ is the phase angle.

Fig. 4.10 shows one of the plots of the real $\tilde{y}$ values and the estimated function $f(x)$ for pitch and fig. 4.11 is a plot for roll.

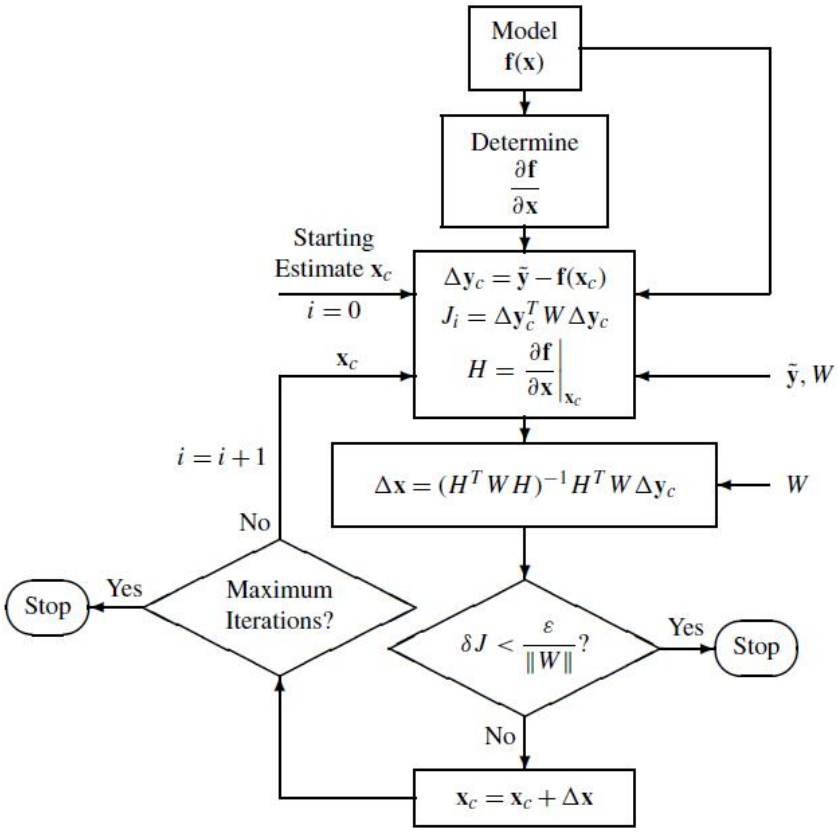


Fig. 4.9. Nonlinear Least Squares Algorithm. The algorithms is very sequential and easy to implement. The performance of the algorithm depends heavily on the initial conditions and the good performance of the plant which administers the measurements.

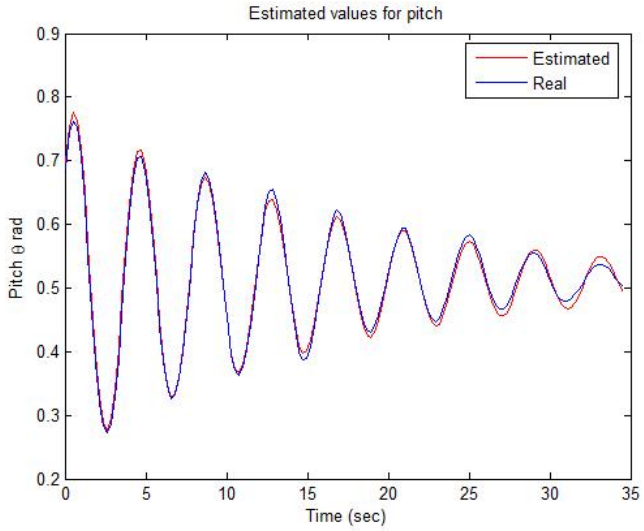


Fig. 4.10. Pitch estimates. Estimates of parameters and real values. The behavior of the plant can be reproduced with the estimated parameters of a damped cosine function as seen in the picture above.

Now, to get the damping factors is known that the equation of motion for angular movement is:

$$\mathrm{I}\ddot{\theta} + \mathrm{C}\dot{\theta} + \mathrm{K}\theta = 0 \tag{4.16}$$

I is the inertia, C is the damping factor and K is the spring factor.

Eq. 4.16 can be represented as a second order transfer function:

$$s^2 + 2\zeta w_n s + w_n^2 \tag{4.17}$$

Then:

$$w_n = \sqrt{\frac{\mathrm{K}}{\mathrm{I}}} \tag{4.18}$$

And finally, the dampinbg factor C is derived from:

$$\zeta = \frac{\mathrm{C}}{2\sqrt{\mathrm{KI}}} \tag{4.19}$$

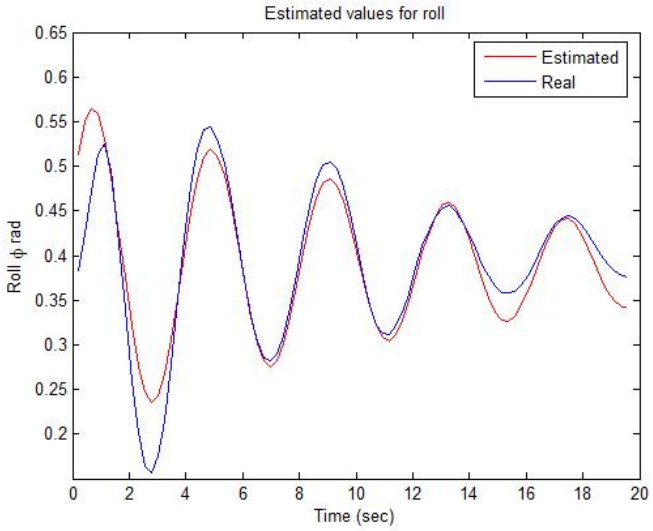


Fig. 4.11. Roll estimates.

5. Conclusion

Parameter estimation is the last step before designing controllers, it tunes the mathematical model so it matches the real performance of the system as close as possible, with this any controller designed will give a grade of authenticity to the work done. It was demostrated that one can certify some aspects of the model very easilly like the center of mass, and other parameters like damping factor required a more advance approach to achieve the results aimed.

CHAPTER V

LQR DESIGN

1. Introduction

When designing a controller it is desired one that provides the best possible performance with respect to some measurements. It can be a controller that uses the least amount of control input energy to make the output zero or one that guarantees stability of the closed loop system, good gain and phase margins, robustness with respect to unmodeled dynamics, or other desirable properties[8].

The minimization procedure used in Linear Quadratic Regulator (LQR) design produces controllers that are stable. The controllers obtained are generally good, even when optimizing for energy is not an objective. Moreover, this procedure is applicable to MIMO processes for which classical designs are difficult to apply.

In this section the LQR methodology is explained, so the differences between this method and the H_∞ can be compared.

2. Feedback model

The feedback model for the LQR control is shown in fig. 5.1.

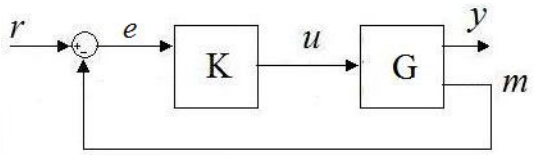


Fig. 5.1. LQR feedback model.

r is the reference input, u is the control input, y is the output of the plant, e is the error and m are the mearurements, thus, the signals available for feedback.

The measurement m is substracted from the reference input r, the controller K tries to make the error e zero, and transmits the control signal u to the plant G.

3. LQR problem

The LQR problem consists in finding a controller K that minimizes eq. 5.1.

$$J_{LQR} = \int_0^\infty y(t)^T Q y(t) + \rho u(t)^T R u(t) dt \tag{5.1}$$

The term $\int_0^\infty y(t)^T Q y(t) dt$ is the energy of the controlled output and the term $\int_0^\infty u(t)^T R u(t) dt$ is the energy of the control signal. Even do LQR tries to minimize both energies, decreasing the energy of the controlled output will require a large control signal and a small control signal will lead to large controlled outputs. So:

1. If ρ is very large, to decrease J_{LQR} little control signals has to be used, but large controlled output will be expected.

2. If ρ is very small, to decrease J_{LQR} very small controlled output has to be obtained, at the expense of a large control signal.

Q and R are symmetric positive definite matrices and ρ is a positive constant. Usually Q and R are diagonal matrices with the maximum values of the states and inputs.

The simplest LQR controller is a matrix gain of the form $u = -Kx$, where K is a matrix given by:

$$K = (D^T Q D + \rho R)^{-1}(B^T P + D' Q C) \tag{5.2}$$

And P is a positive definite solution to the Algebraic Riccati Equation (ARE), eq. 5.3.

$$A^T P + P A + C^T Q C - (P B + C^T Q D)(D^T Q D + \rho R)^{-1}(B^T P + D^T Q C) \tag{5.3}$$

The state feedback control results in a closed loop system of the form:

$$\dot{x} = (A - BK)x \tag{5.4}$$

4. Conclusion

A LQR simulation was done, is it depicted from fig. 5.2 to fig. 5.6. It can be seen that the controller is good, also the H_∞ controller is depicted, which will be explained later.

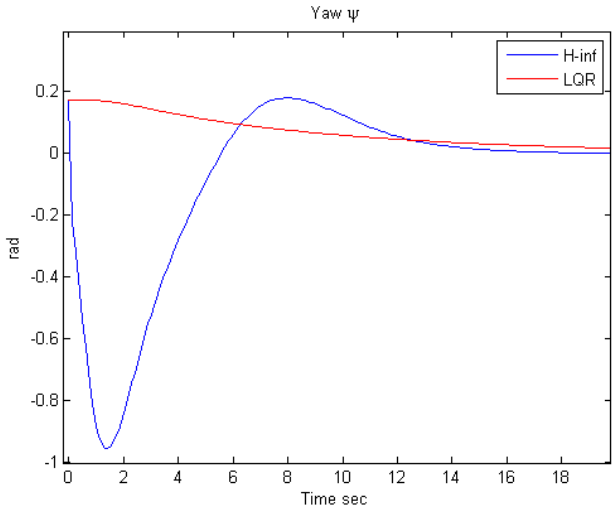


Fig. 5.2. LQR simulation, yaw ψ.

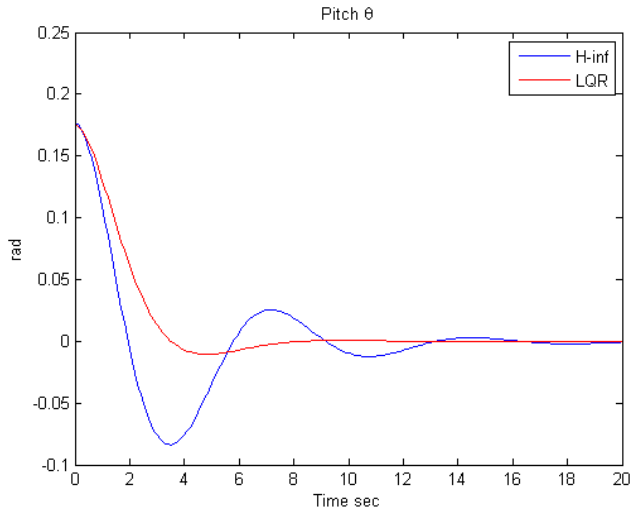


Fig. 5.3. LQR simulation, pitch θ.

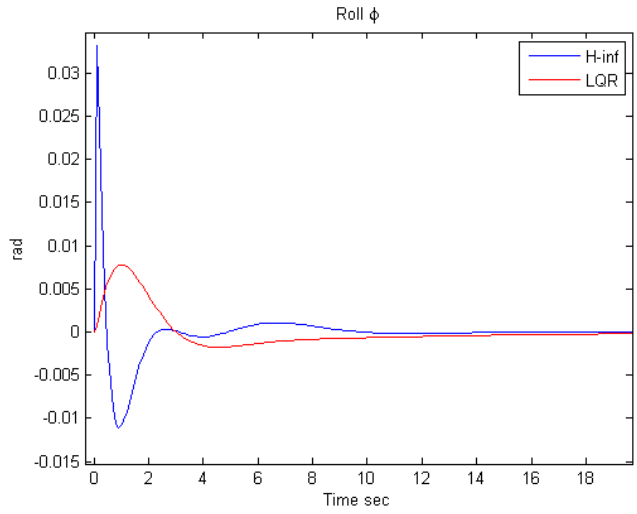


Fig. 5.4. LQR simulation, roll ϕ.

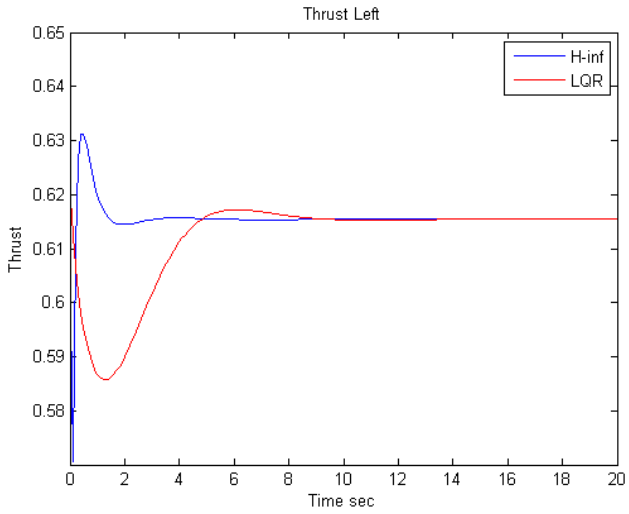


Fig. 5.5. LQR simulation, left thrust.

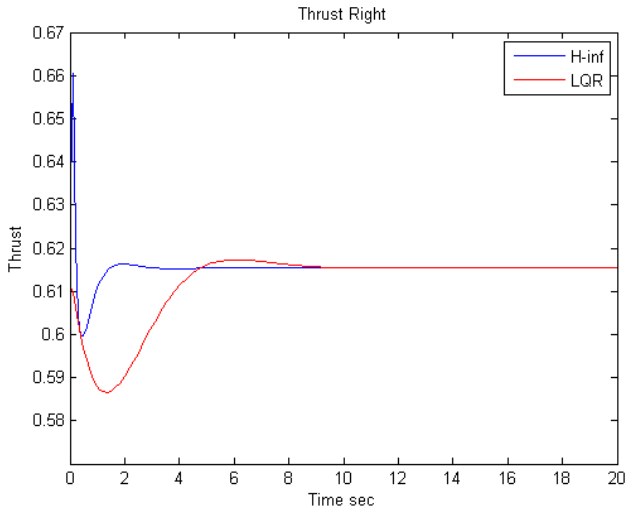


Fig. 5.6. LQR simulation, right thrust.

CHAPTER VI

H_∞ CONTROLLER

1. Introduction

The H_∞ control theory provides a theoretical framework to design a multivariable feedback controller which meets desired performance criteria along with robustness objectives. H_∞ control has turned out more attractive due their robustnes to plant model uncertainties and good disturbance and noise rejection.

Primarily, control design procedures were based on SISO techniques. With this, the intrinsic multivariable aspect of the design was overlooked making incompatibility of systems a big issue[9]. MIMO systems required a more efficient control method.

2. Scaled state space

The birotor state space plant G calculated previously from the nonlinear equations of motion has to be scaled before solving the H_∞ problem. This is acomplished by linear transformations given by eq. 6.1 where N is a diagonal matrix which function is to scale the values of the matrix X.

$$\overline{X} = N^{-1} X N \tag{6.1}$$

.

So for the plant G:

$$\overline{G} = \left[\begin{array}{c|c} \overline{A} & \overline{B} \\ \hline \overline{C} & \overline{D} \end{array}\right] = \left[\begin{array}{c|c} N_x^{-1} A N_x & N_x^{-1} B N_u \\ \hline N_y^{-1} C N_x & N_y^{-1} D N_u \end{array}\right] \tag{6.2}$$

Now, the state space representation of the scaled system is:

$$\begin{aligned} \dot{\overline{x}} &= \overline{A}\overline{x} + \overline{B}\overline{u} \\ \overline{y} &= \overline{C}\overline{x} + \overline{D}\overline{u} \end{aligned} \tag{6.3}$$

By eq. 6.2:

$$\begin{aligned} \dot{\overline{x}} &= N_x^{-1}AN_x\overline{x} + N_x^{-1}BN_u\overline{u} \\ \overline{y} &= N_y^{-1}CN_x\overline{x} + N_y^{-1}DN_u\overline{u} \end{aligned} \tag{6.4}$$

And:

$$\begin{aligned} N_x\dot{\overline{x}} &= AN_x\overline{x} + BN_u\overline{u} \\ N_y\overline{y} &= CN_x\overline{x} + DN_u\overline{u} \end{aligned} \tag{6.5}$$

Finally, from eq. 6.5 it is obvious that:

$$\begin{aligned} x &= N_x\dot{\overline{x}} \\ u &= N_u\overline{u} \\ y &= N_y\overline{y} \end{aligned} \tag{6.6}$$

Eq. 6.2 and eq. 6.6 give the relations between the plant G and $\overline{G}$ as well as the states x, inputs u and outputs y to $\overline{x}$, $\overline{u}$ and $\overline{y}$ respectively. The scaled plant of G is the one used to calculate the H_∞ controller.

The values chosen for the matrices N_x, N_u and N_y are:

$$\begin{aligned} N_x &= \text{diag}(\ 10/\pi \quad 10/\pi \quad 10/\pi \quad 8\pi/180 \quad 5\pi/180 \quad 5\pi/180\ ) \\ N_u &= \text{diag}(\ 1.2 \quad 1.2\ ) \\ N_y &= N_x \end{aligned} \tag{6.7}$$

3. Range space and null space

The concept of linear mapping means that the transformation $A : V \rightarrow W$ is linear if:

$$A(\alpha v_1 + \beta v_2) = \alpha Av_1 + \beta Av_2 \tag{6.8}$$

For all $v_1, v_2 \in V$, and all scalars α and β. V and W are vector spaces with the same associated fiel $\mathbb{F}$. The space V is called the domian of the mapping and W the codomian[10].

Associated with any linear map $A : V \rightarrow W$ is its image or range space $\mathcal{R}(A)$. defined by:

$$\mathcal{R}(A) = \{w \in W \ : \exists v \in V \Rightarrow Av = w\} \tag{6.9}$$

This set contains all the elements of W which are the image of some point in V.

The null space or kernel $\mathcal{N}(A)$ is defined by:

$$\mathcal{N}(A) = \{v \in V \ : Av = 0\} \tag{6.10}$$

In words, $\mathcal{N}(A)$ is the set of vectors in V which get mapped by A to the zero element in W.

The dimensions of the range and null space are linked by the relantionship:

$$dim(V) = dim(\mathcal{R}(A)) + dim(\mathcal{N}(A)) \tag{6.11}$$

4. Norms of systems

The performance of a system can be measured in terms of the size of certain signals of interest, as the size of an error signal in tracking problems. The signals considered map $(-\infty, \infty) \rightarrow \mathcal{R}$.

These signals are assumed to be piecewise continuous. A signal may be zero for $t < 0$, it may start at $t = 0$[11].

A norm of a signal u must have the following properties:

i) $\|u| \geq 0$.

ii) $\|u| = 0 \Leftrightarrow u(t) = 0, \forall t$.

iii) $\|au\| = |a|\|u\|, \forall a \in \mathcal{R}$.

iv) $\|u = v\| \leq \|u\| + \|v\|$.

The 1-norm of $u(t)$ is the integral of its absolute value:

$$\|u\|_1 := \int_{-\infty}^{\infty} |u(t)|dt \tag{6.12}$$

The 2-norm of $u(t)$ is related to power and energy in this statement: the instantaneous power of a signal $u(t)$ is defined to be $u(t)^2$ and its energy is defined to be the square of its 2-norm.

$$\|u\|_2 := \left(\int_{-\infty}^{\infty} u(t)^2 dt\right)^{\frac{1}{2}} \tag{6.13}$$

The ∞-norm of a signal is the least upper bound of its absolute value:

$$\|u\|_\infty := \sup_t |u(t)| \tag{6.14}$$

Systems that are linear, time-invariant, causal, and finite-dimensional are considered. In the time domain an input-output model for such a system has the form of a convolution equation $y = G * u$, or the same $y(t) = \int_{-\infty}^{\infty} G(t-\tau)u(\tau)d\tau$.

Causality means that $G(t) = 0$ for $t < 0$. Let $\hat{G} = G(s)$ denote the Laplace transform of G. Then $\hat{G}$ is rational with real coefficients. We say that $\hat{G}$ is stable if it is analytic in the closed right half-plane ($\mathrm{Re}(s) = 0$), proper if $\hat{G}(jw)$ is finite (degree of denominator = degree of numerator), strictly proper if $\hat{G}(jw) = 0$ (degree of denominator > degree of numerator), and biproper if $\hat{G}$ and $\hat{G}^{-1}$ are both proper (degree of denominator = degree of numerator).

The 2-norm and the ∞-norm of a system are:

$$\|\hat{G}\|_2 := \left(\frac{1}{2\pi}\int_{-\infty}^{\infty}|\hat{G}(jw)|^2 dw\right)^{\frac{1}{2}} \tag{6.15}$$

$$\|\hat{G}\|_\infty := \sup_w|\hat{G}(jw)| \tag{6.16}$$

The ∞-norm of $\hat{G}$ equals the distance in the complex plane from the origin to the farthest point on the Nyquist plot of $\hat{G}$. It also appears as the peak value on the Bode magnitude plot of $\hat{G}$. An important property of the ∞-norm is that it is submultiplicative:

$$\|\hat{G}\hat{H}\|_\infty \leq |\hat{G}\|_\infty|\hat{H}\|_\infty \tag{6.17}$$

The 2-norm of $\hat{G}$ is finite iff $\hat{G}$ is strictly proper and has no poles on the imaginary axis; the ∞-norm is finite iff $\hat{G}$ is proper and has no poles on the imaginary axis.

5. Linear matrix inequalities

The history of linear matrix inequalities (LMI) in the analysis of dynamical systems goes back more than 100 years. The story begins in about 1890, when Lyapunov published his seminal work introducing the Lyapunov theory[12]. He showed that the diferential equation eq. 6.18 is stable, all trajectories converge to zero, if and only if there exists a positive-definite matrix P such that eq. 6.19 is true.

$$\frac{\mathrm{d}}{\mathrm{dt}}x(t) = Ax(t) \tag{6.18}$$

$$A^TP + PA < 0 \tag{6.19}$$

The requirement $P > 0$, $A^TP + PA < 0$ is called a Lyapunov inequality on P, which is a special form of a LMI. Lyapunov also showed that this first LMI could be explicitly solved, selecting $Q = Q^T > 0$ and solving the linear equation $A^TP + PA = -Q$ for the matrix P, which is guaranteed to be positive-definite if

eq. 6.18 is stable. The first LMI used to analyze stability of a dynamical system was the Lyapunov inequality eq. 6.19, which can be solved analytically, by solving a set of linear equations.

A linear matrix inequality has the form:

$$F(x) \triangleq F0 + \sum_{i=1}^{m} x_i F_i > 0 \tag{6.20}$$

$x \in \mathcal{R}^m$ is the variable and the symmetric matrices $F_i = F_i^T \in \mathcal{R}^{nxn}, i = 0, \cdots, m$, are given. The inequality symbol in eq. 6.20 means that $F(x)$ is positive-definite, $u^T F(x) u > 0$ for all nonzero vectors $u \in \mathcal{R}^n$.

The LMI in eq. 6.20 is a convex[1] constraint on x, the set $\{x|F(x)\} > 0$ is convex. Linear inequalities, quadratic inequalities, matrix norm inequalities, and constraints that arise in control theory, such as Lyapunov and convex quadratic matrix inequalities, can all be cast in the form of an LMI. Multiple LMIs $F(1)(x) > 0, \cdots, F(p)(x) > 0$ can be expressed as the single LMI $\text{diag}(F(1)(x), \cdots, F(p)(x)) > 0$.

When the matrices F_i are diagonal, the LMI $F(x) > 0$ is just a set of linear inequalities. Nonlinear (convex) inequalities are converted to LMI form using Schur complements. The basic idea is as follows, the LMI in eq. 6.21 where $Q(x) = Q(x)^T$, $R(x) = R(x)^T$ and $S(x)$ depend affinely on x, is equivalent to eq. 6.22.

$$\begin{bmatrix} Q(x) & S(x) \\ S(x)^T & R(x) \end{bmatrix} > 0 \tag{6.21}$$

$$R(x) > 0, Q(x) - S(x)R(x)^{-1}S(x)^T > 0 \tag{6.22}$$

Problems in which the variables are matrices are very common, like the Lyapunov inequality (eq. 6.19). $A \in \mathcal{R}^{nxn}$ is given and $P = P^T$ is the variable. In this case the

[1]Let $\mathcal{I}$ be a real interval. A function f is said to be convex, if $f(\lambda t + (1-\lambda)s) \leq \lambda f(t) + (1-\lambda)f(s)$ for all t, $s \in \mathcal{I}$ and every $\lambda \in [0,1]$[13].

LMI is not written explicitly in the form $F(x) > 0$, but instead is made clear which matrices are the variables. The phrase "the LMI $\mathrm{A^TP + PA < 0}$ in P" means that the matrix P is a variable. Leaving LMIs in a condensed form such as eq. 6.19, in addition to saving notation, may lead to more efficient computation.

As another related example, consider the quadratic matrix inequality $A^TP + PA + PBR^{-1}B^TP + Q < 0$ where A, B, $Q = Q^T$, $R = R^T > 0$ are given matrices of appropriate sizes, and $P = P^T$ is the variable. It can be expressed as the linear matrix inequality:

$$\left[\begin{array}{c|c} -ATP - PA - Q & PB \\ \hline B^TP & R \end{array}\right] > 0 \tag{6.23}$$

This representation also displays that the quadratic matrix inequality is convex in P, which is not obvious.

In some problems linear equality constraints on the variables may be found.

$$P > 0, A^TP + PA < 0, \mathrm{Tr}(P) = 1 \tag{6.24}$$

From eq. 6.24 $P \in \mathcal{R}^{k\times k}$ is the variable. The equality constraint in eq. 6.24 can be written as $F(x) > 0$. Let $P_1, \cdots, P_m$ be a basis for symmetric $k \times k$ matrices with trace zero ($m = (k(k+1) = 2) - 1$) and let P_0 be a symmetric $k \times k$ matrix with $\mathrm{Tr}(P_0) = 1$. Then take $F_0 = \mathrm{diag}(P_0, -A^TP_0 - P_0A)$ and $Fi = \mathrm{diag}(P_i, -A^TP_i - P_iA)$ for $i = 1, \cdots, m$.

The problem is, given an LMI $F(x) > 0$, the corresponding LMI Problem (LMIP) is to find x^{feas} such that $F(x^{feas}) > 0$ or determine that the LMI is infeasible. This means, find a nonzero $G \geq 0$ such that $\mathrm{Tr}(GF_i) = 0$ for $i = 1, \cdots, m$ and $\mathrm{Tr}(GF_0 \leq 0$. This is a convex feasibility problem. So solving the LMI $F(x) > 0$ means solving the corresponding LMIP.

In the 1980's H_∞ controllers were solved with necessary and sufficient conditions for optimality which were obtained in terms of highly nonlinear coupled matrix equations, which are not easy to solve. But in 1994 Iwasaki and Skelton[14] found that necessary and sufficient conditions for the existence of an H_∞ controller of some (unspecified) order are given in terms of three Linear Matrix Inequalities (LMIs). Positive definite solutions to the LMIs form a convex set. The controller order can be fixed by imposing an additional rank condition (at the expense of convexity) on the solutions to the LMIs. Moreover, the set of all H_∞ controllers is characterized explicitly in the state space representation. As mention by Iwasaki and Skelton, the main advantage of the LMI formulation lies in the computational aspects.

6. Feedback model

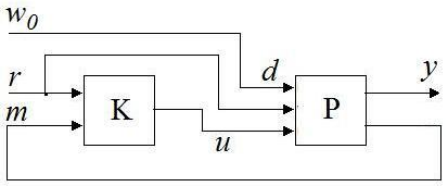


Fig. 6.1. General feedback model. For H controllers usually the input to the controller is the reference and the feedback from sensors. The controller calculates the errors internally.

A general feedbacl model is shown in fig. 6.1. The disturbance w_0 is the vector of inputs that are not generated by the control system, noise is part of this vector. The reference input r is a vector that specifies the desired behavior of the outputs, only the inputs with nonzero desired values are included.

w_0 and r are external inputs that be combined into a single input:

$$d = \left[\begin{array}{c} r \\ \hline w_0 \end{array}\right] \tag{6.25}$$

d is called the generalized disturbance input. u is the vector of control inputs to the plant generated by the controller K. The reference output y is the plant P outputs that are of interest. These outputs may include the errors between plant states and desired values. Additionally, the control input can be incorporated into y. The measured output m is the vector of plant outputs that can be directly measured and therefore available for feedback.

The relation input-output in the plant P can be defined as:

$$\left[\begin{array}{c} \dot{x} \\ \hline y \\ \hline m \end{array}\right] = \left[\begin{array}{c|c|c} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ \hline C_2 & D_{21} & D_{22} \end{array}\right] \left[\begin{array}{c} x \\ \hline d \\ \hline u \end{array}\right] \tag{6.26}$$

7. The H_∞ problem

The H_∞ problem formulation is shown in fig. 6.2. P represents a linear system plant. K is a controller, d is the generalized disturbance input composed of reference inputs and disturbances, e are errors, u is the control signal and y are the measurements.

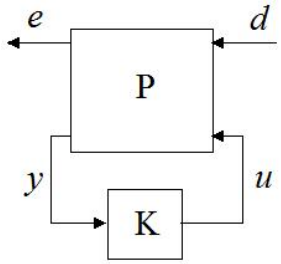


Fig. 6.2. Standard H_∞ problem. The problem formulation is very simple, find a controller K able to estabilize the plant P.

Deriving out of small gain theorem, for unstructured perturbations, robust stability depends on the ∞-norm of the close loop system from the perturbation input to the perturbation output[15]. Thus the minimization of the ∞-norm can be used as means of maximizing robustness.

Frequency dependent weighting functions are used to separate the reference inputs bandwidths as desired. The same designing plataform allows us to design controllers robustly stable to modelling errors.

In a standard H_∞ problem an internally stabilizing controller K must be found such that close loop transfer function from d to e defined as the lower linear fractional transformation $\mathcal{T}_{de}$ given by $\mathcal{F}_\mathcal{L}(P,K)$ has ∞-norm $\|\mathcal{F}_\mathcal{L}(P,K)\|_\infty < \Gamma$ for a given $\Gamma > 0$.

If γ is defined as the ∞-norm of $\mathcal{T}_{de}$, then the objective in H_∞ is to satisfy $\gamma < 1$.

$$\gamma = \|\mathcal{T}_{de}\|_\infty < 1 \tag{6.27}$$

The close loop performance objetives are formulated as weighted close loop transfer functions which are to be made small through feedback. The weighting functions scale the input-output transfer functions so the relation between disturbance and error is suitable and the desired performance objectives are met.

8. Controller loop shape

To design the H_∞ controller for the birotor system, a feedback block with weights is planned as illustrated in fig. 6.3. r, y, u and m were explained in the previous sections, $\hat{u}$ is the weighted control signal, $\hat{m}$ is the measured outputs affected by the noise n, e_1 and e_2 are the weigthed tracking error and weighted plant error respectively. G is the scaled plant and K is the controller.

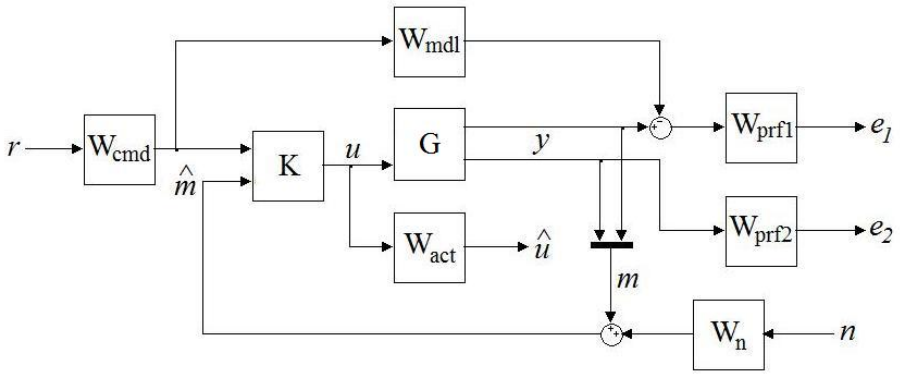


Fig. 6.3. Controller K design block. To synthetize a good controller is necessary to formulate a close loop performance, weight functions are included to scale and shape the performance of the input/output transfer functions.

The weighting functions transform and scale physical units into normalized output signals. W_{cmd} is included in H_∞ control problems that require tracking of a reference command, it shapes the normalized reference command into the expected reference signal. The performance weighting function, W_{prf1}, has only diagonal entries which indicate a tracking error on the reference inputs. W_{prf2} is also a diagonal matrix penalizing variables internal to G and not included in the tracking objetive.

The actuation weighting function W_{act} normalize signals passed through a first-order filter from the controller ouput to the plant input. A noise weighting function, W_n, is a diagonal scaling matrix on the noise disturbances affecting the system.

W_{mdl} represents a desired ideal model for the closed-looped system and is often included in problem formulations with tracking requirements[16]. Inclusion of an ideal model, W_{mdl}, for tracking, is often called a model matching problem.

The objective of the closed-loop system is to match the defined model. For good command tracking response, the closed-loop system might respond like a well-damped second-order system:

$$W_{mdl} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{6.28}$$

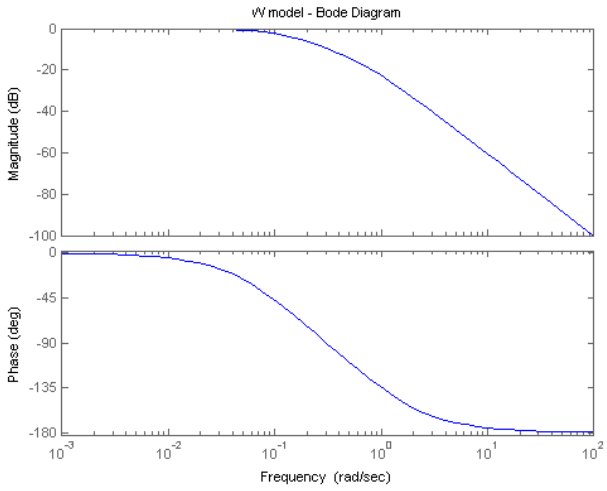


Fig. 6.4. W model.

For yaw (ψ), a damping factor $\zeta = 1.5$ is chosen and $\zeta = 1.5$ for pitch (θ). The natural frequency ω_n, can be extracted from the settling time $T_s|_{=10sec}$ formula:

$$T_s = \frac{4.6}{\zeta \omega_n} \tag{6.29}$$

So, finally:

$$W_{mdl} = \begin{bmatrix} W_{mdl\psi} & 0 \\ 0 & W_{mdl\theta} \end{bmatrix} \tag{6.30}$$

Fig. 6.4 shows the bode plot of W_{mdl}.

A command weighting function, fig. 6.5, W_{cmd}, is selected as a first order butterworth filter which rolls off at high frequencies. Eq. 6.31 shows that W_{cmd} is a diagonal matrix penalizing the two observed references and $w_{c1} = 2\pi f_{w_{c1}}$, $f_{w_{c1}} = 1$.

$$W_{cmd} = \begin{bmatrix} \frac{w_{c1}}{s+w_{c1}} & 0 \\ 0 & \frac{w_{c1}}{s+w_{c1}} \end{bmatrix} \tag{6.31}$$

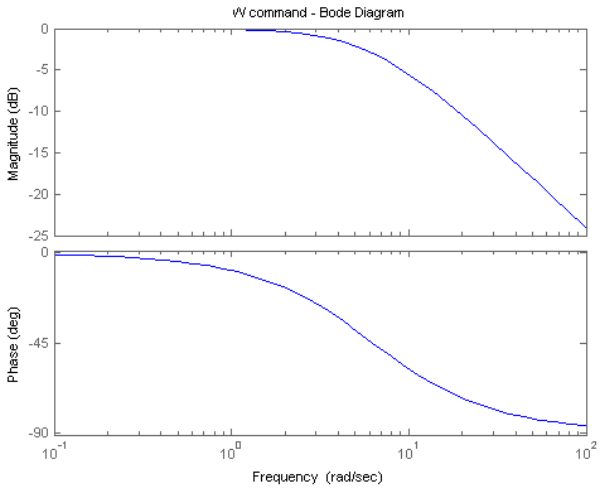


Fig. 6.5. W command.

The perfomance weighting function of the tracking error has a roll off frequency of 1 Hz and then it flats again at 1000 Hz, it is depicted in fig. 6.6.

$$W_{prf1} = \begin{bmatrix} W_\psi & 0 \\ 0 & W_\theta \end{bmatrix} \tag{6.32}$$

$$W_\psi = \frac{\frac{1}{2\pi 1000}s + 1}{\frac{1}{2\pi}s + 1} \tag{6.33}$$

$$W_\theta = \frac{\frac{1}{2\pi 1000}s + 1}{\frac{1}{2\pi}s + 1} \tag{6.34}$$

Each entry of the perfomance weighting function of the non tracking error, W_{prf2}, is selected as a low pass filter, with $w_{c2} = 2\pi f_{w_{c2}}$, $f_{w_{c2}} = 5$, see fig. 6.7.

$$W_{prf2_i} = \frac{w_{c2}}{s + w_{c2}} \tag{6.35}$$

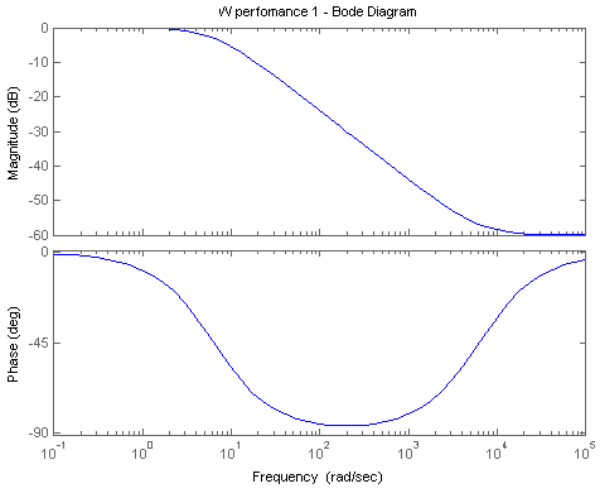


Fig. 6.6. W performance 1.

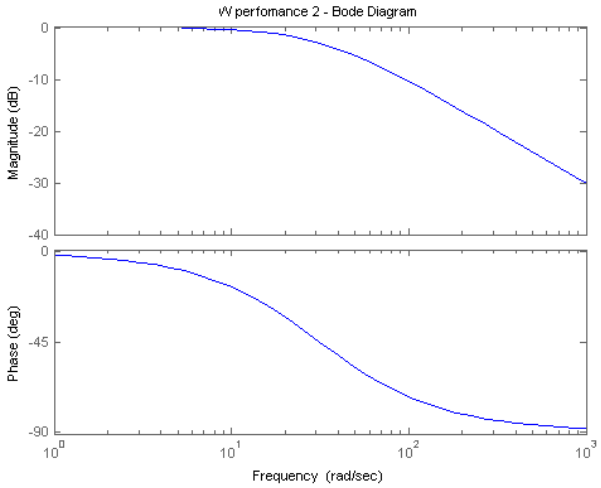


Fig. 6.7. W performance 2.

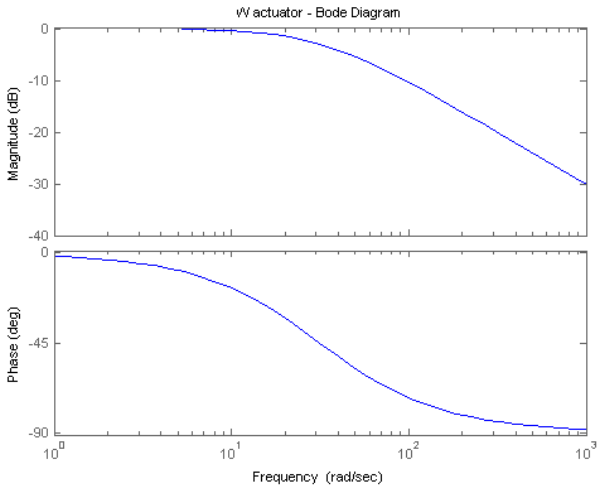


Fig. 6.8. W actuator.

W_{act} normalizes the plant input, fig. 6.8. The cut off frequency chosen is $w_a = 2\pi 5$.

$$W_{acti} = \frac{w_a}{s + w_a} \tag{6.36}$$

Eq. 6.36 is one entry of a diagonal matrix $\in \mathcal{R}^i$, where i denotes the number of control signals.

$$W_{act} = \text{diag}([W_{act1} \ \cdots \ W_{acti}]) \tag{6.37}$$

The noise weighting function, W_n, is displayed in eq. 6.38 to penalize the positions ψ, θ and ϕ. The maximum resolution of the sensors is 10 bits, that means that the minimum perturbation that can be measured is 360/1024 degrees, based on this the noise wieghting function is choosen in radians as:

$$W_n = \mathcal{I}^{3\times 3}\frac{360}{1024 * 57.3} \tag{6.38}$$

9. H_∞ synthesis

First, a state space realization that define P and K in fig. 6.2 is introduced[10], the state space dimensions of the plant and controller will be important, $A \in \mathcal{R}^{n_P \times n_P}$ and $A_K \in \mathcal{R}^{n_K \times n_K}$.

$$P = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & 0 \end{array}\right] \tag{6.39}$$

$$K = \left[\begin{array}{c|c} A_K & B_K \\ \hline C_K & D_K \end{array}\right] \tag{6.40}$$

Then, consider the close loop in fig. 6.2 which can be described as:

$$\Sigma_{CL} = \left[\begin{array}{c|c} A_{CL} & B_{CL} \\ \hline C_{CL} & D_{CL} \end{array}\right] = \left[\begin{array}{cc|c} A + B_2 D_K C_2 & B_2 C_K & B_1 + B_2 D_K D_{21} \\ B_K C_2 & A_K & B_K D_{21} \\ \hline C_1 + D_{12} D_K C_2 & D_{12} C_K & D_{11} + D_{12} D_K D_2 1 \end{array}\right] \tag{6.41}$$

With the controller K, the closed-loop states becomes $\hat{x}^T = [x^T; x_K^T]$ and the plant matrices and the controller are replaced with the following matrices:

$$\left[\begin{array}{ccc} \overline{A} & \overline{B} & \underline{B} \\ \overline{C} & D_{11} & \underline{D}_{12} \\ \underline{C} & \overline{D}_{21} & \overline{K}^T \end{array}\right] = \left[\begin{array}{cc|c|cc} A & 0 & B_1 & 0 & B_2 \\ 0 & 0 & 0 & I & 0 \\ \hline C_1 & 0 & D_{11} & 0 & D_{12} \\ \hline 0 & I & 0 & D_K^T & B_K^T \\ C_2 & 0 & D_{21} & C_K^T & A_K^T \end{array}\right] \tag{6.42}$$

The H_∞ control problem is converted to a problem of solving a LMI.

For that, the following set of matrices are defined[14][2]:

$$\mathcal{L}_B := \{X \in \mathcal{R}^{n_p x n_p} : X = X^T > 0,$$
$$\begin{bmatrix} B_2 \\ D_{12} \end{bmatrix}^{\perp} \times \begin{bmatrix} AX + XA^T + B_1B_1^T & XC_1^T + B_1D_{11}^T \\ C_1X + D_{11}B_1^T & D_{11}D_{11}^T - I \end{bmatrix} \times \begin{bmatrix} B_2 \\ D_{12} \end{bmatrix}^{\perp T} < 0\} \tag{6.43}$$

$$\mathcal{L}_C := \{Y \in \mathcal{R}^{n_p x n_p} : Y = Y^T > 0,$$
$$\begin{bmatrix} C_2^T \\ D_{21}^T \end{bmatrix}^{\perp} \times \begin{bmatrix} YA + A^TY + C_1^TC_1 & YB_1 + C_1^TD_{11} \\ B_1^TY + D_{11}^TC_1 & D_{11}^TD_{11} - I \end{bmatrix} \times \begin{bmatrix} C_2^T \\ D_{21}^T \end{bmatrix}^{\perp T} < 0\} \tag{6.44}$$

$$\begin{bmatrix} X & I \\ I & Y \end{bmatrix} \geq 0 \tag{6.45}$$

Let $\mathcal{L}_B$ and $\mathcal{L}_C$ be defined by replacing the plant matrices in the above definitions with the matrices in eq. 6.39. Note that, in this case, each set is a subset of $(n_P + n_K) \times (n_P + n_K)$ real symmetric matrices.

A synthesis of order n_K exists for the H_∞ problem, if and only if there exist symmetric matrices $X > 0$ and $Y > 0$ satisfying eq. 6.43, eq. 6.44 and eq. 6.45 plus the additional constraint:

$$\text{rank} \begin{bmatrix} X & I \\ I & Y \end{bmatrix} \leq n_P + n_K \tag{6.46}$$

Once X and Y have been found satisfying eq. 6.43, eq. 6.44 and eq. 6.45, then exists a matrix $X_{CL} \in \mathcal{R}^{n_P \times n_K}$ satisfying[10]:

$$X_{CL} = \begin{bmatrix} X & ? \\ ? & ? \end{bmatrix} \tag{6.47}$$

[2] $x^{\perp}$ is a matrix such that $\mathcal{N}(x^{\perp}) = \mathcal{R}(x)$ and $x^{\perp}x^{\perp T} > 0$. $\mathcal{N}(x)$ and $\mathcal{R}(x)$ denote the nullspace and the range space of x, respectively.

$$X_{CL}^{-1} = \begin{bmatrix} Y & ? \\ ? & ? \end{bmatrix} \tag{6.48}$$

This matrix X_{CL} can be constructed by findind a matrix $X_2 \in \mathcal{R}^{n_P \times n_K}$ such that $X - Y^{-1} = X_2 X_2^T$, then eq. 6.49 has the properties desired above.

$$X_{CL} = \begin{bmatrix} X & X_2^T \\ X_2 & I \end{bmatrix} \tag{6.49}$$

Finally, there is a LMI solution to eq. 6.50 for K that provides the state space realization for a feasible H_∞ controller.

$$H_{X_{CL}} + Q^T K^T P_{X_{CL}} + P_{X_{CL}}^T K Q < 0 \tag{6.50}$$

From eq. 6.50:

$$P_{X_{CL}} = \begin{bmatrix} \underline{B}^T X_{CL} & 0 & \underline{D}_{12}^T \end{bmatrix} \tag{6.51}$$

$$Q = \begin{bmatrix} \underline{C} & \underline{D}_{21}^T & 0 \end{bmatrix} \tag{6.52}$$

$$H_{X_{CL}} = \begin{bmatrix} \overline{A}^T X_{CL} + X_{CL} \overline{A} & X_{CL} \overline{B} & \overline{C}^T \\ \overline{B}^T X_{CL} & -I & D_{11}^T \\ \overline{C} & D_{11} & -I \end{bmatrix} \tag{6.53}$$

A H_∞ controller is found using the techniques explained above, using Matlab software. The results are shown in fig. 6.9 to fig. 6.12. A stable controller is achieved with $\gamma = 0.8999$.

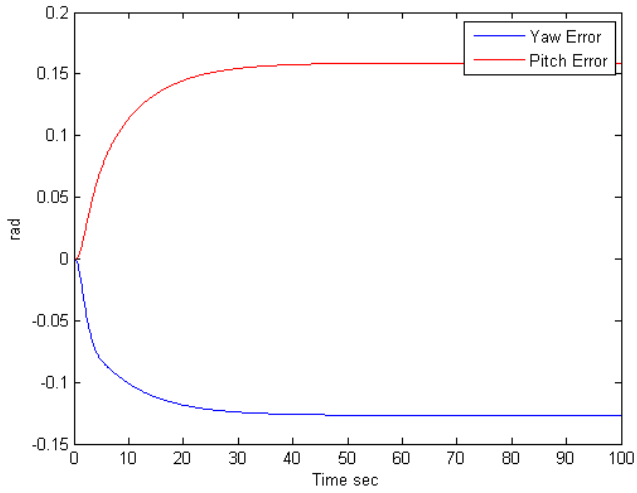


Fig. 6.9. H_∞ implemented simulation, errors. In the simulation the controller takes the birotor from the initial condition ψ_0, θ_0 to a reference point.

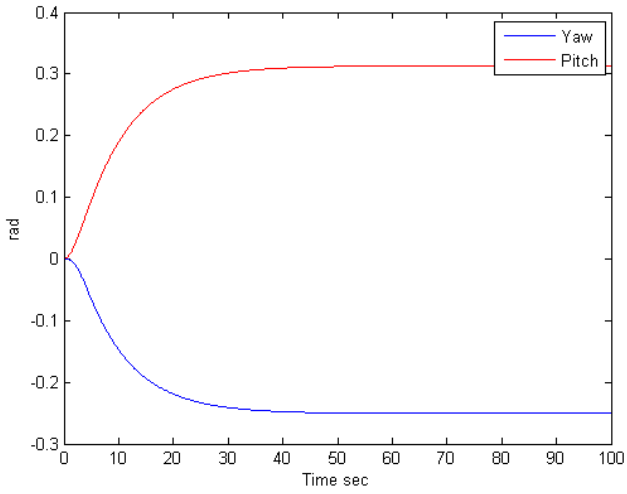


Fig. 6.10. H_∞ implemented simulation, positions.

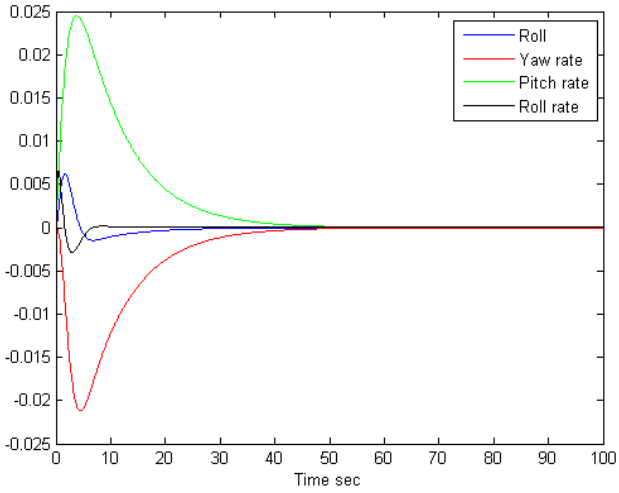


Fig. 6.11. H_∞ implemented simulation, roll angle and velocities.

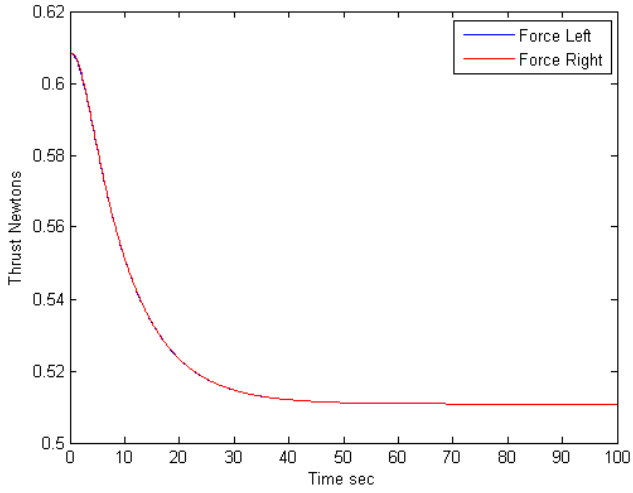


Fig. 6.12. H_∞ simulation, thrust.

10. Conclusion

H_∞ algorithms provide a powerful tool in control theory. The multivariable effects of the mathematical model are not supressed in the process, thus giving a more accurate response of the system. Robust control algorithms prove to be a well done option for applications where perturbations and errors are present.

CHAPTER VII

IMPLEMENTATION

1. Introduction

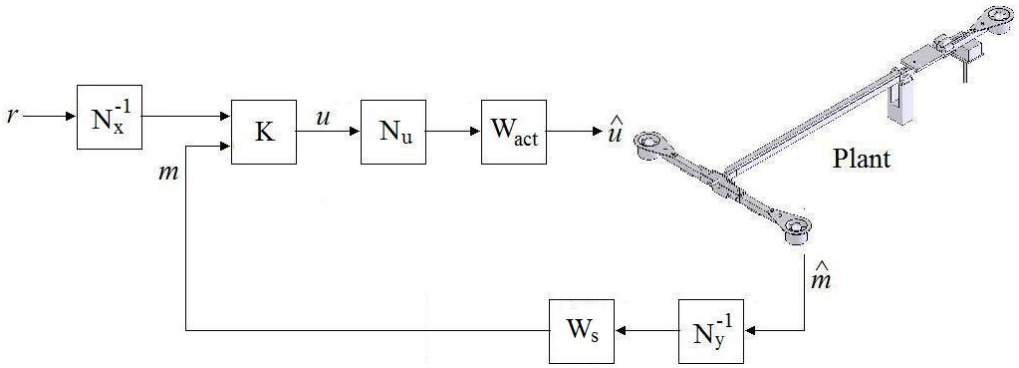


Fig. 7.1. H_∞ implementation diagram. The implementation diagram is just a subset of the controller design block. Noises and disturbances are implicit applied to the system.

Once the H_∞ controller is synthetized and the condition $\gamma < 1$ is satisfied the controller can be implemented. As can be seen in fig. 7.1, the implementation diagram is a subset of the controller design block shown in fig. 6.3, thus only the part compromising the controller and the plant is needed. Also, the references and measurements inputs as well as the controller output must be scaled to be accorded with the controller. In order to implement the controller for real time applications it must be discretized, as explained below.

A low pass filter is included in the sensor output to account for the delay in the system generated due to signal transmission and the wireless link between plant and controller. This is a first order low pass filter given by:

$$W_s = \frac{w_s}{s + w_s}, w_s = 2\pi 0.2243 \tag{7.1}$$

The controller is implemented in a National Instruments® CompactRIO® real time controller using LabView® software. Please refer to the apendix for more information.

2. MIMO frequency response

Classical frequency response methods had been powerful design tools widely used by practicing engineers. There are several reasons for the continued success of these methods for dealing with single-loop problems and multiloop problems arising from some (MIMO) systems[17]. The connection between frequency response plots and data that can be experimentally acquired and trained engineers find these methods relatively easy to learn. Also, their graphical nature provides an important visual aid that is greatly enhanced by modern computer graphics and these methods supply the designer with a rich variety of manipulative and diagnostic aids that enable a design to be refined in a systematic way. Finally, simple rules for standard controller configurations and processes can be developed.

The singular value decomposition (SVD) is the tool used to calculate the frequency reponse of a MIMO system. For any $m \times p$ complex matrix Q, there exist $m \times m$ and $p \times p$ unitary matrices Y and U, and a real matrix Σ, such that:

$$Q = Y \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} U^T \tag{7.2}$$

In which $\Sigma = \text{diag}(\sigma_1, \cdots, \sigma_r)$ with $\sigma_1 \leq \sigma_2 \leq \cdots \sigma_r > 0$ and $\min(m, p) \leq r$. When Q is real, Y and U may be chosen orthogonal. Eq. 7.2 is called a SVD of Q.

The maximum singular value $\overline{\sigma}(Q)$ and the minimum singular value $\underline{\sigma}(Q)$ play a particularly important role in the frequency analysis and are given by the identities:

$$\overline{\sigma}(Q) = \max_{\|u\|=1} \|Qu\| \tag{7.3}$$

$$\underline{\sigma}(Q) = \min_{\|u\|=1} \|Qu\| \tag{7.4}$$

u is a column of the unitary matrix U. The vector norm is the Euclidean norm. Thus $\overline{\sigma}(Q)$ and $\underline{\sigma}(Q)$ are respectively the maximum gain and the minimum gain of the matrix Q.

For a MIMO system given by the transfer function $G(s)$, the frequency response is calculated by eq. 7.5 over all frequencies, $-\infty < w < \infty$. Fig. 7.2 shows the frequency reponse of the controller K previously calculated, the frequency at the maximum gain is 127.6175 rad/sec or which is the same 20.3109 Hz.

$$\text{Frequency response} = \overline{\sigma}(G(jw)) \tag{7.5}$$

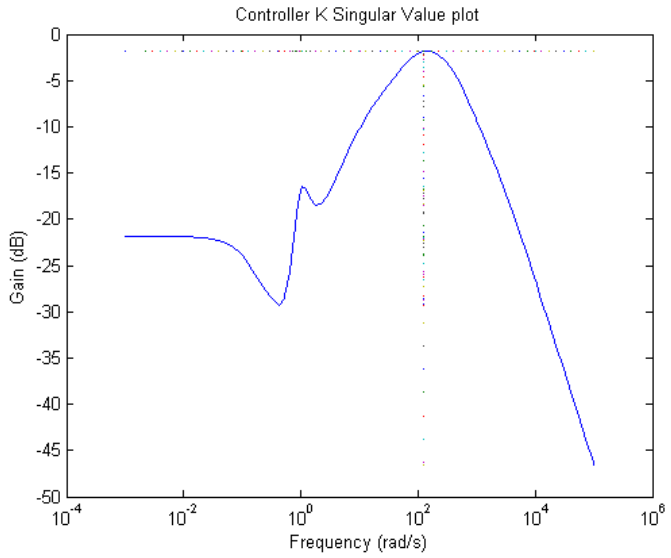


Fig. 7.2. H_∞ controller frequency response. The frequency reponse of the controller K can be seen in this plot. The frequency at the maximum gain is also shown.

3. Tustin's method

The technique used to discretize is the Tustin's method (also called bilinear transformation), which is used in digital signal processing and discrete-time control theory to transform continuous-time system representations to discrete-time and vice versa.

The Tustin's method is a conformal mapping, often used to convert a transfer function $H_a(s)$ of a linear, time-invariant (LTI) system in the continuous-time domain to a transfer function $H_d(z)$ of a linear, shift-invariant filter in the discrete-time domain.

This method maps positions on the jw axis, $Re[s] = 0$, in the s-plane to the unit circle, $|z| = 1$, in the z-plane, is a first-order approximation of the natural logarithm function that is an exact mapping of the z-plane to the s-plane. When the Laplace transform is performed on a discrete-time signal (with each element of the discrete-time sequence attached to a correspondingly delayed unit impulse), the result is the Z transform of the discrete-time sequence with the substitution of:

$$z = e^{sT} = \frac{e^{sT/2}}{e^{-sT/2}} \approx \frac{1 + sT/2}{1 - sT/2} \tag{7.6}$$

T is the sample time (the reciprocal of the sampling frequency) of the discrete-time system.

The above bilinear approximation can be solved for s or a similar approximation for $s = (1/T)\ln(z)$ can be performed. In this case, the sampling time is chosen as $T = 20$ms, the working period of the motors.

The transformation preserves stability and maps every point of the frequency response of the continuous-time system, $H_a(jw_a)$, to a corresponding point in the frequency response of the discrete-time system, $H_d(e^{jw_dT})$, although to a somewhat different frequency, due to frequency warping explained below.

This means that for every feature that one sees in the frequency response in the continous time, there is a corresponding feature, with identical gain and phase shift, in the frequency response in the digital domain, perhaps, at a somewhat different frequency.

This is barely noticeable at low frequencies but is quite evident at frequencies close to the Nyquist frequency of the system.

The inverse of this mapping (and its first-order bilinear approximation) is:

$$\begin{array}{rcccl} s & = & \frac{1}{T}\ln(z) & = & \frac{2}{T}\left[\frac{z-1}{z+1} + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 + \frac{1}{5}\left(\frac{z-1}{z+1}\right)^5 + \frac{1}{7}\left(\frac{z-1}{z+1}\right)^7 + \cdots\right] \\ & \approx & \frac{2}{T}\frac{z-1}{z+1} & \approx & \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \end{array} \tag{7.7}$$

The bilinear transform essentially uses this first order approximation and substitutes into the continuous-time transfer function $H_a(s)$. That is:

$$H_d(z) = H_a(s)\Big|_{s=\frac{2}{T}\frac{z-1}{z+1}} = H_a\left(\frac{2}{T}\frac{z-1}{z+1}\right) \tag{7.8}$$

A continuous-time system is stable if the poles of its transfer function fall in the left half of the complex s-plane. A discrete-time system is stable if the poles of its transfer function fall inside the unit circle in the complex z-plane.

The Tustin's method maps the left half of the complex s-plane to the interior of the unit circle in the z-plane, thus, conserving stability.

4. Frequency warping

To determine the frequency response of a continuous-time system, the transfer function $H_a(s)$ is evaluated at $s = jw$. Likewise, to determine the frequency response of a discrete-time system, the transfer function $H_d(z)$ is evaluated at $z = e^{jwT}$ which is on the unit circle.

When the actual frequency of w is input to the discrete-time system designed by use of the Tustin's method, it is desired to know at what frequency, w_a, for the continuous-time system that this w is mapped to.

$$H_d(z) = H_a\left(\frac{2}{T}\frac{z-1}{z+1}\right) \tag{7.9}$$

$$\begin{aligned} H_d(e^{j\omega T}) &= H_a\left(\tfrac{2}{T}\tfrac{e^{j\omega T}-1}{e^{j\omega T}+1}\right) = H_a\left(\tfrac{2}{T}\cdot\tfrac{e^{j\omega T/2}\left(e^{j\omega T/2}-e^{-j\omega T/2}\right)}{e^{j\omega T/2}\left(e^{j\omega T/2}+e^{-j\omega T/2}\right)}\right) \\ &= H_a\left(j\tfrac{2}{T}\cdot\tfrac{\sin(\omega T/2)}{\cos(\omega T/2)}\right) = H_a\left(j\tfrac{2}{T}\cdot\tan\left(\omega\tfrac{T}{2}\right)\right) \\ &= H_a\left(j\omega_a\right) \end{aligned} \tag{7.10}$$

Eq. 7.10 clarifies that every point on the unit circle in the z-plane is mapped to a point on the jw axis on the s-plane. The discrete-continuous frequency mapping is:

$$\omega_a = \frac{2}{T}\tan\left(\omega\frac{T}{2}\right) \tag{7.11}$$

And the inverse mapping is:

$$\omega = \frac{2}{T}\arctan\left(\omega_a\frac{T}{2}\right) \tag{7.12}$$

The gain and phase shift that the system has at frequency w in the discrete-time is the same gain and phase shift that the continuous has at frequency $(2/T)\tan(\omega T/2)$.

This means that every visiblefeature in the frequency response in continuous-time is also visible in discrete-time but at a different frequency. For low frequencies, that is, when $\omega \ll 2/T$ or $\omega_a \ll 2/T$, $\omega \approx \omega_a$, one can see that the entire continuous frequency range $-\infty < \omega_a < \infty$ is mapped onto the fundamental frequency interval $-\frac{\pi}{T} < \omega < \frac{\pi}{T}$.

The continuous-time frequency $w_a = 0$ corresponds to the discrete-time frequency $w = 0$ and the continuous-time frequency $\omega_a = \pm\infty$ correspond to the discrete-time frequency $\omega = \pm\pi/T$.

There can be seen that there is a nonlinear relationship between w_a and w. This effect of the Tustin's method is called frequency warping. The continuous-time system can be designed to compensate for this frequency warping by setting $\omega_a = \frac{2}{T}\tan\left(\omega\frac{T}{2}\right)$ for every frequency specification that can be manipulated (such as corner or center frequency). This is called pre-warping design.

The main advantage of the warping phenomenon is the absence of aliasing distortion of the frequency response characteristic, such as observed with impulse invariance. It is necessary, however, to compensate for the frequency warping by prewarping the given frequency specifications of the continuous-time system. These prewarped specifications may then be used in the Tustin's method to obtain the desired discrete-time system. Fig. 7.3 shows the pole-zero map of the discretized controller.

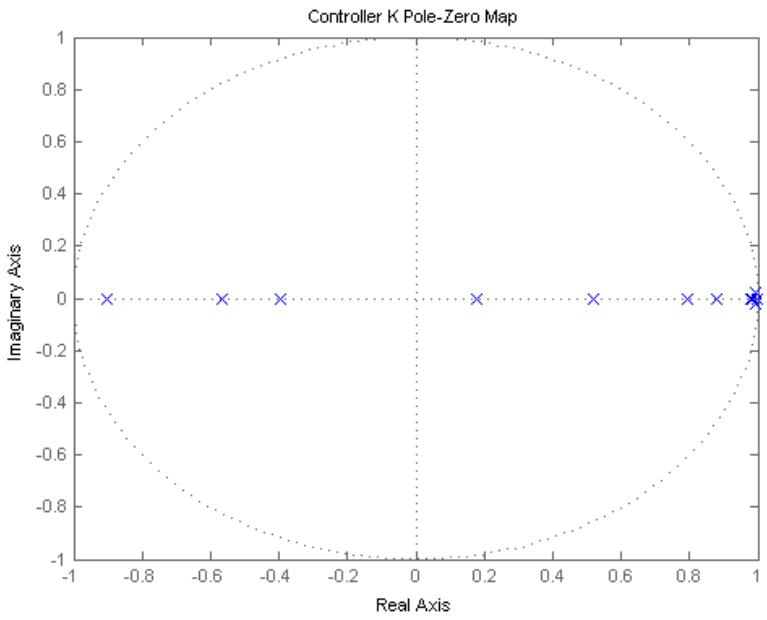


Fig. 7.3. H_∞ controller pole-zero map. Being the controller K stable in continous time the poles of its tranfer function in discrete time rest inside the unit circle in the complex z-plane. The frequency chosen to discretize was $w = 2\pi 5$ rad/sec and the period $T = 0.20$ sec.

5. Conclusion

A simulation is depicted from fig. 7.4 to fig. 7.6. The controller takes the birotor from $\psi_0 = 0$, $\theta_0 = pi/3$ to the zero position.

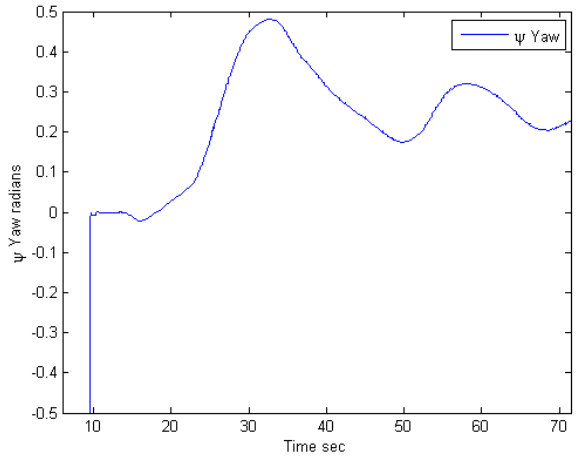


Fig. 7.4. H_∞ implemented, ψ yaw.

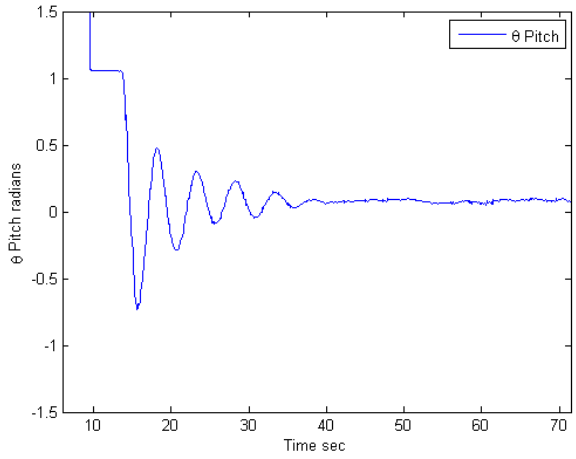


Fig. 7.5. H_∞ implemented, θ pitch.

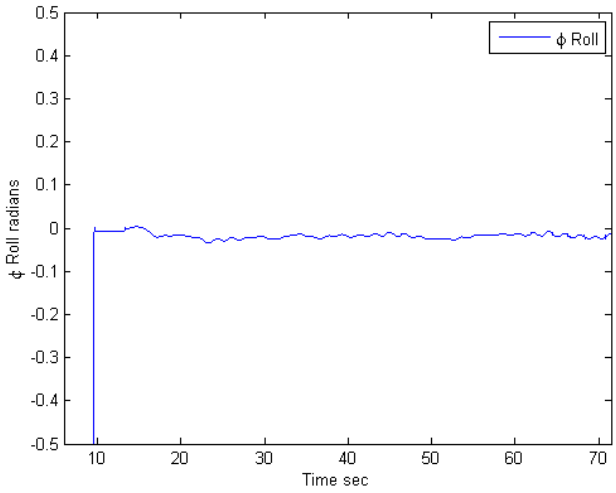


Fig. 7.6. H_∞ implemented, ϕ roll.

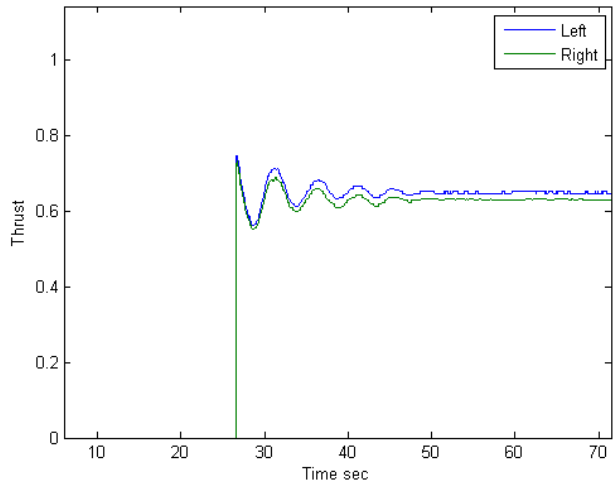


Fig. 7.7. H_∞ implemented, thrust. These values are scaled and rounded to be applied in the plant.

CHAPTER VIII

CONCLUSION

In this thesis a real time application of robust control theory is employed succesfully. It is shown that the H_∞ control methods can be used with tools available in the market.

The growth of tecnological and scientific research have made possible, for techniques like the one presented in this thesis, to be more accesible for real time operations, increasing the confidence in deploying them.

Robust control techniques are more efficient and have tremendous flexibility allowing its use in a wide range of applications. This thesis presents the results and a methodical progression to achieve the goals aimed, in this case, the control of a 3DOF birotor helicopter.

REFERENCES

[1] P. Berner, *Orientation, Rotation, Velocity, and Acceleration and the SRM*, Synthetic Environment Data Representation and Interchange Specification, 2007.

[2] J. Escareño, S. Salazar-Cruz and R. Lozano, "Attitude stabilization of a convertible mini birotor," in *Proc. of the 2006 IEEE International Conference on Control Applications*, Munich, Germany, October 4-6 2006, Institute of Electrical and Electronic Engineers, pp. 2202–2206.

[3] The Johns Hopkins University, *Handout Number 1 about Linearization*, Department of Electrical and Computer Engineering, 2006.

[4] J. Mazurkiewicz, *How a Brushless Motor Operates*, Fort Smith, Arkansas, Baldor Electric.

[5] Microchip Technology Inc., *PIC16F87XA Data Sheet*, Chandler, Arizona, Microchip Technology Inc., 2003.

[6] National Instruments, *LabVIEW User Manual*, Austin, Texas, National Instruments, 2003.

[7] J. L. Crassidis and J. L. Junkins, *Optimal Estimation of Dynamic Systems*, Boca Raton, Florida, Chapman and Hall/CRC, 1st edition, 2004.

[8] A. Emami-Naeini, G. F. Franklin and J. D. Powell, *Feedback Control of Dynamic Systems*, Upper Saddle River, New Jersey, Prentice Hall, 4th edition, 2002.

[9] G. Balas and S. Ganguli, "A TECS Alternative Using Robust Multivariable Control," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Montreal, Canada, August, American Institute of Aeronautics & Astronautics, pp. 4022–4028.

[10] G. E. Dullerud and F. Paganini, *A Course in Robust Control Theory, A Convex Approach*, New York, Springer, 1st edition, 1999.

[11] J. Doyle, B. Francis and A. Tannenbaum, *Feedback Control Theory*, New York, Macmillan Publishing Co., 1st edition, 1991.

[12] V. Balakrishnan, S. Boyd, E. Feron and L. E. Ghaoui, *Linear Matrix Inequalities in System and Control Theory*, Philadelphia, Pennsylvania, Society for Industrial and Applied Mathematics, 1st edition, 1994.

[13] F. Hansen, *Convex Matrix Functions*, Copenhagen, Denmark, Institute of Economics, University of Copenhagen.

[14] T. Iwasaki and R. E. Skelton, "All controllers for the general h-inf control problem: Lmi existence conditions and state space formulas," *Automatica*, vol. 30, no. 8, pp. 1307–1317, 1994.

[15] J. B. Burl, *Linear Optimal Control, H_2 and H_∞ Methods*, Menlo Park, California, Addison-Wesley, 1st edition, 1999.

[16] G. Balas, R. Chiang, A. Packard and M. Safonov, *Robust Control Toolbox 3 User's Guide*, Natick, Massachusetts, The MathWorks Inc., 2009.

[17] M. Green and D. J. N. Limebeer, *Linear Robust Control*, Upper Saddle River, New Jersey, Prentice Hall, Inc., 1st edition, 1995.

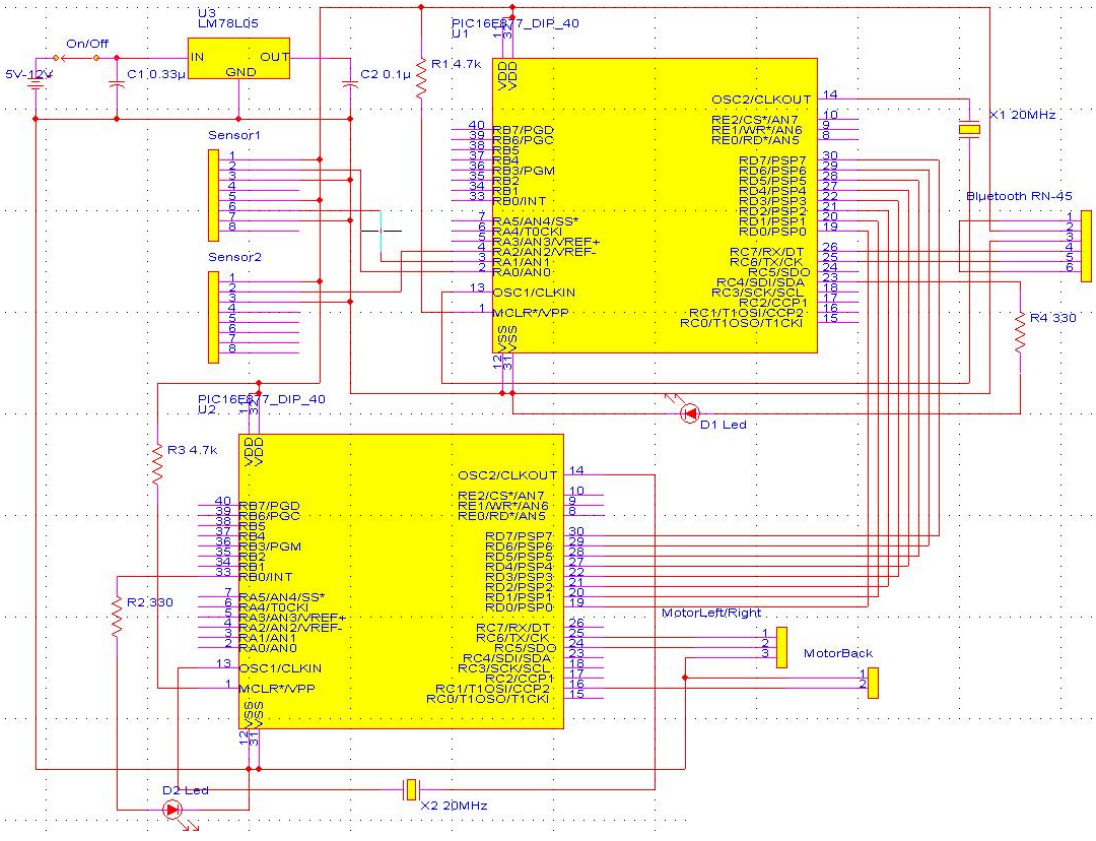
APPENDIX A

ELECTRICAL DIAGRAMS

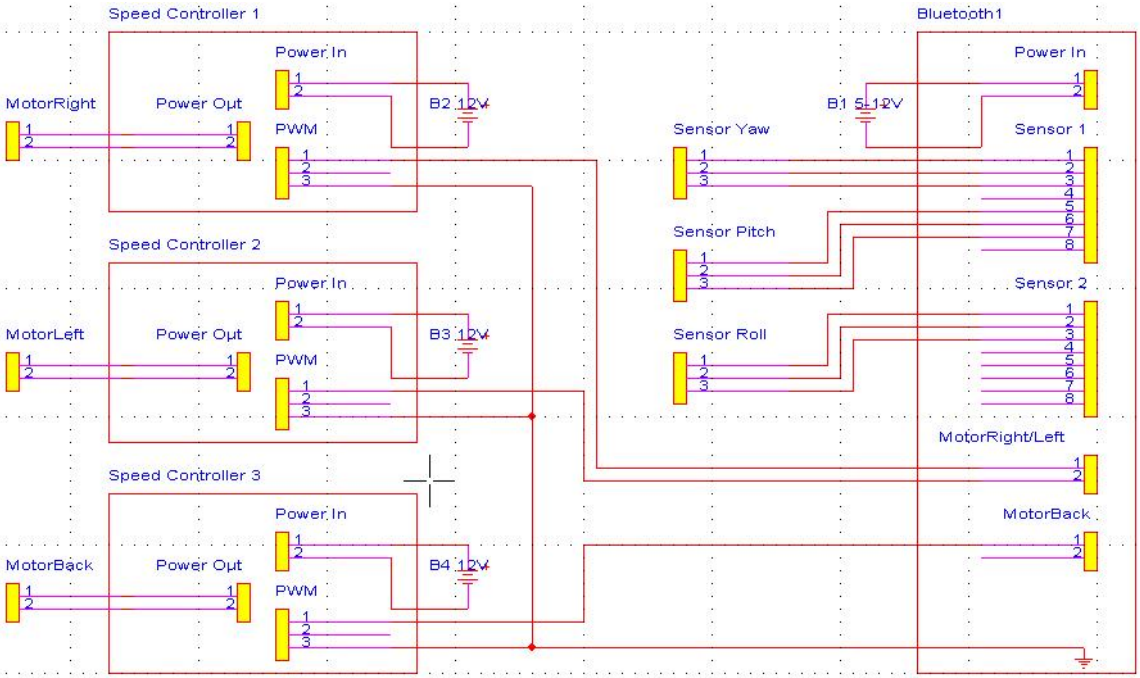
The electrical diagrams were done with PCB123 software.

PIC board on the helicopter.

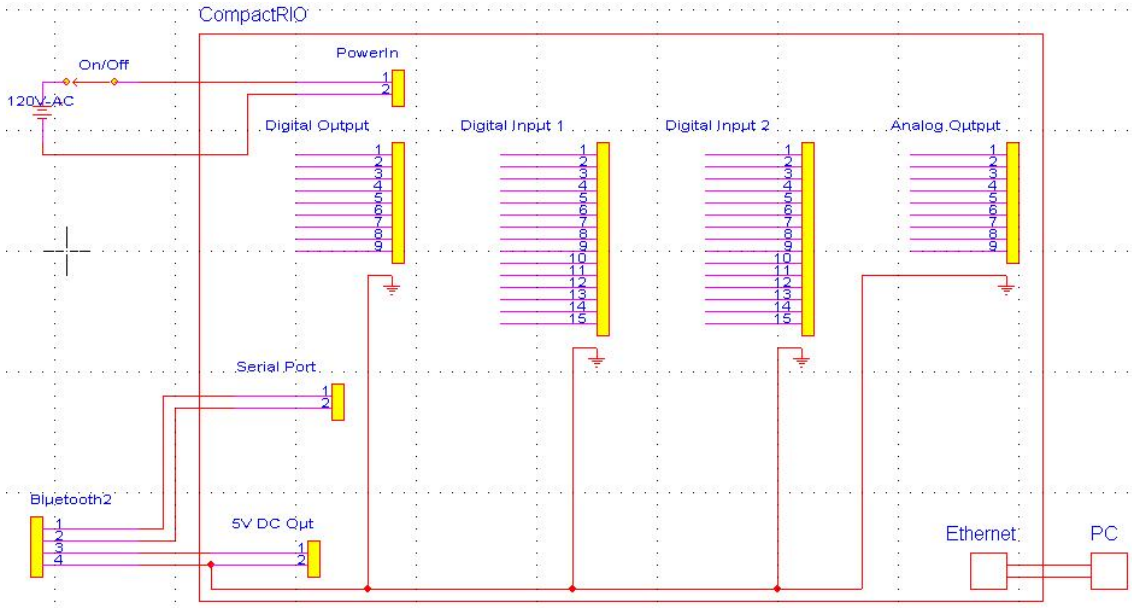
This board reads the sensors and turns on the motors.



Electrical conection onboard the helicopter between sensors and the PIC board.



Conection of the bluetooth module with the CompactRIO controller.



APPENDIX B

PIC CODES

The codes shown are provided to users "as is", without warranty. There is no warranty for the codes, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose and noninfringement of third party rights. The entire risk as to the quality and performance of the codes is with the user. Should the codes prove defective, the user assume the cost of all necessary servicing, repair or correction.

In no event the author will be liable to anyone for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the codes (including but not limited to loss of data or data being rendered inaccurate or losses sustained by the user or third parties or a failure of the codes to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

PIC 1. Code to read sensors.

Send and recieve data from the Compact RIO controller via wireless bluetooth communication. This code also sends the information to PIC 2.

```
#include <16F877A.h>
#device adc=10 //ADC convertion 10 bits
#include <stdlib.h>
#define LED PIN_C4

#use delay(clock=20000000)
#use rs232(baud=57600, xmit=PIN_C6, rcv=PIN_C7)
#byte PORTB=0x06 //Define PORTB in memory location
#byte PORTC=0x07 //Define PORTC in memory location
#byte PORTD=0x08 //Define PORTD in memory location
#byte TXREG=0x19 //Define TXREG in memory location
#byte RCREG=0x1A //Define RCREG in memory location

#define BUFFER_SIZE  3
unsigned char PWMX[BUFFER_SIZE];
unsigned char c; // Variable to read character
unsigned long valueY; // Variable to read ADC0
unsigned long valueP; // Variable to read ADC1
unsigned long valueR; // Variable to read ADC2
int8 nextin=0;

#INT_RDA // Recieve interrupt
void RS232C_Recieve_Interrupt(){
        c=getc(); // Read character
        PWMX[nextin++]=c; // Save character in buffer
        if (nextin >= BUFFER_SIZE){ // If last character send ADC
                        set_adc_channel(0); // Set channel 0
                        delay_us(20); // Wait 20 us
                        valueY=read_adc(); // Read convertion

                        set_adc_channel(1); // Set channel 1
                        delay_us(20); // Wait 20 us
                        valueP=read_adc(); // Read convertion

                        set_adc_channel(2); // Set channel 2
                        delay_us(20); // Wait 20 us
                        valueR=read_adc(); // Read convertion

                printf("%4ld%4ld%4ld",valueY,valueP,valueR); // Send all
                nextin=0; // Reset buffer position
        }
}

void conf(){ // Function that configures ports
        set_tris_a(0xFF); // Port A all input
        setup_port_a(ALL_ANALOG); // Port A all analog
        setup_adc(ADC_CLOCK_DIV_32); // Turn on adc
        set_tris_c(0b11000000); //LED PORT Output
                                        // USART TX & RX
        set_tris_b(0xFF); // Port B all input
        set_tris_d(0x00); // Port D all output

        enable_interrupts(INT_RDA); // Enable recieve interrupt
        enable_interrupts(GLOBAL);

        output_high (LED); // Turn on LED

        PWMX[0]=0; // Initialize variables
        PWMX[1]=0;
```

```
        PWMX[2]=0;
         valueY=0;
         valueP=0;
         valueR=0;
}

void main(){
int a;
a=1;

         conf();

         while(a==1){
                 PORTD=240; //Send value to motors
                  delay_ms(2);
                 PORTD=PWMX[0];
                  delay_ms(2);

                 PORTD=245;
                  delay_ms(2);
                 PORTD=PWMX[1];
                  delay_ms(2);

                 PORTD=250;
                  delay_ms(2);
                 PORTD=PWMX[2];
                  delay_ms(2);
         }
}
```

PIC 2. Code to move the brushless motors.

```
#include <16F877A.h>
#include <stdlib.h>
#define LED     PIN_B0
#define MOTORB PIN_C1 // Motor Back Port
#define MOTORR PIN_C5 // Motor Right Port
#define MOTORL PIN_C6 // Motor Left Port

#use delay(clock=20000000)
#byte PORTB=0x06 //Define PORTB in memory location
#byte PORTC=0x07 //Define PORTC in memory location
#byte PORTD=0x08 //Define PORTD in memory location

unsigned char CHANN,PWMX;
unsigned char VALUER, VALUEL, VALUEB; // PWM values
void Interrupt()
{
   CHANN=PORTD;

   // Store which channel
   if((CHANN>=239)&&(CHANN<=241)) // Store Channel
      PWMX=1;
   else if((CHANN<238)&&(PWMX==1)) // Do PWM
      VALUEL=CHANN;
   else if((CHANN>=244)&&(CHANN<=246))
      PWMX=2;
   else if((CHANN<238)&&(PWMX==2))
      VALUER=CHANN;
   else if((CHANN>=249)&&(CHANN<=251))
      PWMX=3;
   else if((CHANN<238)&&(PWMX==3))
      VALUEB=CHANN;
}

void main()
{
unsigned long duty;
int a, i;

   set_tris_b(0x00); // Port B all Output
                     //LED PORT Output
   set_tris_c(0x00); //PWM2 Output
   set_tris_d(0xFF); // Port D all Input

  PWMX=0; // Initialize these variables
   VALUEL=0;
   VALUER=0;
   VALUEB=0;

   output_high (LED); // Turn on LED

   a=1;
   i=1;
   duty=4500-1107;
   for( i=1 ; i<=250 ; i++ )
   {
   output_high(MOTORL);
   delay_us(999+1);
   output_low(MOTORL);
   delay_us(1001-1);

   output_high(MOTORR);
```

```
delay_us(999+1);
output_low(MOTORR);
delay_us(1001-1);

output_high(MOTORB);
delay_us(999+1);
output_low(MOTORB);
delay_us(1001-1);

delay_us(1300);
delay_us(1001-2/2);

delay_us(500);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(300);
delay_us(500);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(300);
delay_us(500);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(300);
delay_us(500);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
delay_us(501-2/2);
 Interrupt();
}

while(a==1)
{
output_high(MOTORL);
delay_us(1107+VALUEL);
output_low(MOTORL);
delay_us(893-VALUEL);
```

```
    output_high(MOTORR);
    delay_us(1107+VALUER);
    output_low(MOTORR);
    delay_us(893-VALUER);

    output_high(MOTORB);
    delay_us(1107+VALUEB);
    output_low(MOTORB);
    delay_us(893-VALUEB);

    delay_us(1300);
    delay_us(1001-2/2);

    delay_us(500);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(300);
    delay_us(500);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(300);
    delay_us(500);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(300);
    delay_us(500);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    delay_us(501-2/2);
     Interrupt();
    }
}
```

APPENDIX C

MATLAB CODES

The codes shown are provided to users "as is", without warranty. There is no warranty for the codes, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose and noninfringement of third party rights. The entire risk as to the quality and performance of the codes is with the user. Should the codes prove defective, the user assume the cost of all necessary servicing, repair or correction.

In no event the author will be liable to anyone for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the codes (including but not limited to loss of data or data being rendered inaccurate or losses sustained by the user or third parties or a failure of the codes to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

Code to calculate densities.

```
clear all; close all; clc; fig=1;

%% Aluminium UNION PIECES

Volume=1.785*0.75^2+2*0.6^2-0.25^2*pi/4*1.565-0.16^2*pi/4*.625; % Inches
Volume=0.000016387064*Volume; % Meters
Mass=0.073; % Kilograms
rhoUnion=Mass/Volume

%% CounterWieght

Volume=0.0755*0.0477*0.037-0.008^2*pi/4*0.037; % Meters
Mass=1.004; % Kilograms
rhoCounter=Mass/Volume

%% ScrewL

Volume=(0.008^2*pi/4*0.140-(0.008-2*.0006)^2*pi/4*0.140)/2; % Meters
Volume=0.008^2*pi/4*0.140-Volume; % Meters
Mass=0.04; % Kilograms
rhoScrewL=Mass/Volume

%% Screw

Volume= 0.00000047; % Meters
Mass=0.003; % Kilograms
rhoScrew=Mass/Volume

%% Mounting

Volume=0.00001231; % Meters
Mass=0.019; % Kilograms
rhoMounting=Mass/Volume

%% FanPlastic

Volume=0.00000597; % Meters
Mass=0.005; % Kilograms
rhoFan=Mass/Volume

%% Motor

Volume=0.00001771; % Meters
Mass=0.034; % Kilograms
rhoMotor=Mass/Volume

%% Link

Volume=(11+7/8)*(0.75*0.75-0.6*0.6); % Inches
Volume=0.000016387064*Volume; % Meters
Volume=Volume-0.008^2*pi/4*2*0.0016*4;
Mass=0.087; % Kilograms
rhoLink=Mass/Volume

%% Battery

Volume=0.00003292; % Meters
Mass=0.105; % Kilograms
rhoBattery=Mass/Volume

%% Electronics
```

```
Volume=0.00005052; % Meters
Mass=0.098; % Kilograms
rhoElectronics=Mass/Volume

%% SpeedCntr

Volume=0.00000517; % Meters
Mass=0.017; % Kilograms
rhoSpeedCntr=Mass/Volume

%% Cables

Volume=0.00003960; % Meters
Mass=0.035; % Kilograms
rhoCables=Mass/Volume
```

Code to calculate COM points and motor gains.

```
clear all; close all; clc; fig=1;

%% Motor Gain

BitIn=[0 25 34 48 59 68 80 97 102 109 120 130 144 158 164 178 195 ...
                                        210 230 235];
Thrust=[0 7 12 18 24 29 34 40 45 50 55 60 64 71 77 83 91 ...
                                        97 106 110]*9.81/1000;

norder=1;
Kgain=polyfit(BitIn,Thrust,norder); %Get linear equation that relates
                                    %Bytes and Thrust
Kvr=Kgain(1);                   %Gain motor right
Kvl=Kgain(1);                   %Gain motor left
Tvr=0.3;
Tvl=0.3;
Bit0=-Kgain(2)/Kgain(1); %No thrust speed

%% Plot fitted curve

% P(1)*X^N + P(2)*X^(N-1) +...+ P(N)*X + P(N+1).
FitByte=0:255; % Byte vector
FitThrust=zeros(size(FitByte));
for i=0:norder
   FitThrust=Kgain(i+1).*FitByte.^(norder-i)+FitThrust;
end

figure(fig); fig=fig+1;
plot(BitIn,Thrust,'r'); hold on;
plot(FitByte,FitThrust); hold off; title 'Byte vs Thrust';
legend('Real','Fitted'); xlabel 'Byte In'; ylabel 'Thrust';

%% Bytes to pitch

% save('datapitch','datapitch.data');
datapitch=load('datapitch.mat'); % Load data
data=datapitch.data;
xt=[5.2 12 24 38 60 79 98 123 142 160 180 192 215 231 248 269 287 300 ...
    316 336 350 368 387 403 417 436 454 495 516]; % Initial times
xt=[xt 525 556 575];
xf=xt+0.2; xf=[xf 552 572 590]; % Final times

[y1,tim1]=labviewdata(data,10,0,fig,xt(1:26),xf(1:26),2);     % sort data
[y2,tim2]=labviewdata(data,10,0,fig,xt(27:end),xf(27:end),2);

for i=fig+1:length(xt)+1 % close figures
    close(figure(i))
end

pitch=zeros(size(xt)); % make vectors
byte=zeros(size(xt));
for i=1:26
    pitch(i)=y1.(char(i+64))(end,2);
    byte(i)=mean(mean(y1.(char(i+64))(:,7:8)));
end
for i=1:length(xt)-26;
    pitch(i+26)=y2.(char(i+64))(end,2);
    byte(i+26)=mean(mean(y2.(char(i+64))(:,7:8)));
end
pitch=pitch-0.0184; % trim applied due to fans with no bytes applied
```

```
figure(fig); fig=fig+1;
plot(byte,pitch,'r'); title 'Bytes vs Pitch - real';
xlabel 'Byte In'; ylabel 'Pitch';

%% Model parameters

Mp=2.02270866;    % Link2 mass kilograms
Mr=0.45695968;    % Link3 mass

lc=[-0.11010108,0,-0.01657261]';    % Link2 center of mass meters
L=0.67098;                          % Link2 length meters
lc2=[0.00079903+L,0,-0.02671306]'; % Link3 center of mass meters

lmodel=(Mp*lc+Mr*lc2)/(Mp+Mr); % Model total center of mass

g=9.81;                          % Gravity m/s^2
ThrustModel=(Mp+Mr)*g*lmodel(1)/2/L; % Model Thrust at pitch zero

%% Real parameters

ByteReal=interp1(pitch,byte,0);            % Real bytes at pitch 0
ThrustReal=interp1(BitIn,Thrust,ByteReal); % Real thrust at pitch 0

% 2*F*L=m*g*(l*cos(x)-l'*sin(x)); % Torque formula
F=ThrustReal;
m=Mp+Mr;

lreal=zeros(size(lmodel));
lreal(1)=2*F*L/m/g;

thrust=interp1(BitIn,Thrust,byte);
lcda=(2.*thrust.*L./m./g-lreal(1).*cos(pitch))./(sin(pitch));

plot(byte,lcda,'r'); title 'Bytes vs Z location of center of mass - real';
xlabel 'Byte In'; ylabel 'Z location of center of mass (meters)';

lreal(3)=mean(lcda);

%% Display data

disp('Kgain');
disp(Kgain);

disp('ThrustModel  ThrustReal');
disp([ThrustModel ThrustReal]);

disp('  lmodel       lreal');
disp([lmodel lreal]);
```

Code to read data from LabView.

```
function data = lvm_import(filename)
%% lvm_import
% DATA = LVM_IMPORT(FILENAME)
%
% LVM_IMPORT returns the data from a .lvm text file created by LabView.
% Introduction
% DATA = LVM_IMPORT(FILENAME) returns the data from a .lvm text
% file created by LabView.
%
% FILENAME    The name of the .lvm file
%
% DATA        The data is returned as a MxN array: M columns, N data
%               points.
%
% This code will import the contents of a text-formatted LabView .lvm file.

%% open the data file
fid=fopen(filename);
if fid ~= -1, %then file exists
    fclose(fid);
else
    filename=strcat(filename,'.lvm');
    fid=fopen(filename);
    if fid ~= -1, %then file exists
        fclose(fid);
    else
        error(['File not found in current directory! (' pwd ')']);
    end
end
fid=fopen(filename); % open the validated file
fprintf(1,'\nImporting %s:\n\n',filename);

%% read the file
linein=fgetl(fid); % process the file header
% first, is it really a LVM file?
if ~strcmp(sscanf(linein,'%s'),'LabVIEWMeasurement')
    try
    data.Segment1.data = dlmread(filename,'\t');
    fprintf(1,'This file appears to be an LVM file with no header.\n');
    fprintf(1,'Data was copied, but no other information is available.\n');
    return
    catch
        error('This does not appear to be a text-format LVM file.');
    end
end

%% process file header
while 1
    linein=fgetl(fid); % get a line from the file
    % exit when we reach the end of the header
    if strcmp(sscanf(linein,'%s'),'***End_of_Header***')
        break
    end
end

%% process segment
while 1
    linein=fgetl(fid); % get a line from the file
    % exit when we reach the end of the header
    if strcmp(sscanf(linein,'%s'),'***End_of_Header***')
        break
```

```
    end
end % end reading segment header loop

% after header is the row of column labels
linein=fgetl(fid); % Read column label
data = cell2mat(textscan(fid,'%f')); % Read data

%% finish
fprintf(1,...
 'Import complete. File has %s X-Columns and %d data Segments.\n','1',1);
fclose(fid); % close the file
return
```

Code to do parameter estimation.

```
clear all; close all; clc; fig=1;

%% Initial conditions

A=0;      % steady state constant
B=1;      % cosine magnitude
z=0;      % damping factor
phi=0;    % phase angle

We=1/100;    % weighting matrix, relative importance of each measure
eps=1e-7;   % tolerance

imax=20;    % max number of iterations

%% Read LabView data for pitch

data=lvm_import('pichdamp.lvm');      % import data

xt=[47 95 136.2 208 237 282 355 394.5 503 580 623]; % select initial times
xf=[83 125 165 222 270 330 390 435 532 615 660];    % select final times

fig2=fig; smoothsnl=0; nsignal=10;
[ypitch,timpitch,fig]=labviewdata(data,10,smoothsnl,fig,xt,xf,2); % sort

for i=fig2:fig       % close figures
    close(figure(i))
end
fig=fig2;

%% Estimate parameters for pitch

xcpitch=zeros(length(xt),5);
for i=1:length(xt)

    selc=char(i+64); % select set of data to estimate parameters

    pitch=ypitch.(selc)(:,2); pitch=decimate(pitch,10);     % select data
    ytil=pitch; % measured data

    t=timpitch.(selc)-timpitch.(selc)(1); t=decimate(t,10); % select time

    % select initial frequency
    if (selc=='A')||(selc=='B')||(selc=='C')||(selc=='D')||...
            (selc=='J')||(selc=='K')
        f=0.15; % A B C D J K
    elseif (selc=='E')||(selc=='F')
        f=0.2; % E F
    elseif (selc=='G')||(selc=='H')||(selc=='I')
        f=0.25; % G H I
    end

    xcpitch(i,:)=est_states(A,B,f,z,phi,ytil,t,We,eps,imax,0,0);

end
xcpitch

%% Read LabView data for roll

data=lvm_import('rolldamp.lvm');      % import data

xt=[94.3 188.5 203.8 225 266.4 290.2 325.7 337];
```

```
xf=[110 200 215 240 280 310 335 354];

fig2=fig; smoothsnl=0; nsignal=10;
[yroll,timroll,fig]=labviewdata(data,10,smoothsnl,fig,xt,xf,3); % sort

for i=fig2:fig       % close figures
    close(figure(i))
end
fig=fig2;

%% Estimate parameters for roll

xcroll=zeros(length(xt),5);
for i=1:length(xt)

    selc=char(i+64); % select set of data to estimate parameters

    roll=yroll.(selc)(:,2); roll=decimate(roll,10);     % select data
    ytil=roll; % measured data

    t=timroll.(selc)-timroll.(selc)(1); t=decimate(t,10); % select time

    % select initial frequency
    if (selc=='A')||(selc=='B')||(selc=='C')||(selc=='D')
        f=0.15; % A B C D
    elseif (selc=='E')||(selc=='F')||(selc=='G')||(selc=='H')
        f=0.2; % E F G H
    end

    xcroll(i,:)=est_states(A,B,f,z,phi,ytil,t,We,eps,imax,0,0);

end
xcroll

%% Calculate damping coefficients

Ir=0.01541260; % Inertia in roll
Ip=0.35432307; % Inertia in pitch

% for pitch
wpitch=mean(xcpitch(:,3)); % Natural frequency
kpitch=wpitch^2*Ip;        % Spring constant

zpitch=mean(xcpitch(:,4));                    % Damping ratio
format long; dampP=zpitch*2*sqrt(kpitch*Ip) % Damping
wpitch
zpitch
format short

% for roll
wroll=mean(xcroll(:,3));
kroll=wroll^2*Ir;

zroll=mean(xcroll(:,4));
format long; dampR=zroll*2*sqrt(kroll*Ir)
format short
```

Function *est_states*.

```
function xc=est_states(A,B,f,z,phi,ytil,t,We,eps,imax,fig,verbose)

%
%EST_STATES estimates the states of the damped cosine function by
%     nonlinear least squares; otherwise known as Gaussian least squares
%     differential correction.
%     EST_STATES(A,B,F,Z,PHI,YTIL,T,WE,EPS,IMAX)
%
%     XC      = estimated states (A B w,(2*pi*f) z phi)
%     A       = steady state constant
%     B       = cosine magnitude
%     F       = angular velocity
%     Z       = damping factor
%     PHI     = phase angle
%     YTIL    = measured data, column vector
%     T       = time, row vector
%     WE      = weighting matrix, relative importance of each measure
%     EPS     = tolerance
%     IMAX    = max number of iterations
%     FIG     = 0 - no plot, >0 - plot number for answer
%     VERBOSE = 1 - show iteration progress, 0 - don't show
%
%     The damped cosine function is: fx=A+B.*exp(-z.*t).*cos(w.*t+phi)
%
%     Sintax example:
%         xc=est_states(0,1,0.15,0,0,ytil,t,1/100*eye(2),1e-4,15,1)
%
% Estimation algorithm
%
%                                        Model
%                                        f(x)
%                                          |
%                                          |
%                                          V
%                                      Determine
%                                         df/dx
%                                          |
%                                    i=0   |
%                               initial xc |
%                                          |
%                                          V
%                                    delyc=ytil-f(x)
%                    |-------------->  Ji=delyc'*W*delyc
%                    |                    H=df/dx|
%                    |                           |xc
%             i=i+1  |                           |
%                    |                           |
%                    |                           V
%                 no |          delx=inv(H'*W*H)*H'*W*delyc
%                    |                           |
%           yes      max                          |
% STOP <------ iterations?                       V          yes
%                    ^                 delJ<eps/norm(W)? ------> STOP
%                    |                           |
%                    |                       no  |
%                    |                           V
%                    |------------------- xc=xc+delx

%% Initial conditions

w=2*pi*f; % angular velocity
```

```
i=0;        % iteration number

%% Get partial derivatives
%
% syms A B phi t w z
% fx=A+B.*exp(-z.*t).*cos(w.*t+phi);
%
% x=[A B w z phi]'; % vector of estimates
%
% H=jacobian(fx,x')
% H=[1, exp(-z.*t).*cos(w.*t+phi), -B.*exp(-z.*t).*sin(w.*t+phi).*t,...
%                                  -B.*t.*exp(-z.*t).*cos(w.*t+phi)];
% H=[1, exp(-z.*t).*cos(w.*t+phi), -B.*exp(-z.*t).*sin(w.*t+phi).*t,...
%       -B.*t.*exp(-z.*t).*cos(w.*t+phi), -B.*exp(-z.*t).*sin(w.*t+phi)];
%

%% Estimation of Parameters

J=zeros(imax+1,1); J(1)=1000; % initialize
est=zeros(imax,6);
delJ=1000;

while (i<imax)&&(delJ>(eps/norm(We)))

    % Iteration number
    i=i+1;

    % State estimates
    est(i,:)=[i A B w z phi]; % save estimates
    xc=est(i,2:end);          % update estimate vector x
    if verbose==1
      disp('Iteration         A         B         w         z         phi');
      disp(est(1:i,:));         % display estimated
    end

    % Step 1
    fx=A+B.*exp(-z.*t).*cos(w.*t+phi); % function to estimate
    delyc=ytil-fx';                    % residual measurements
    J(i+1)=delyc'*We*delyc;             % optimization function

    H1=1*ones(size(t));                   % df/dx1
    H2=exp(-z.*t).*cos(w.*t+phi);         % df/dx2
    H3=-B.*exp(-z.*t).*sin(w.*t+phi).*t;
    H4=-B.*t.*exp(-z.*t).*cos(w.*t+phi);
    H5=-B.*exp(-z.*t).*sin(w.*t+phi);
    H=[H1' H2' H3' H4' H5'];              % H, jacobian matrix df/dx

    % Step 2
    delx=inv(H'*We*H)*H'*We*delyc; % corrections

    % Step 3
    delJ=abs(J(i+1)-J(i))/J(i+1); % minimization tolerance

    % Step 4
    if delJ>(eps/norm(We)) % update estimate vector x
        xc=xc+delx';
        A=xc(1);
        B=xc(2);
        w=xc(3);
        z=xc(4);
        phi=xc(5);
    end

end
```

```
%% Plot answer

if fig>0
    fx=A+B.*exp(-z.*t).*cos(w.*t+phi); % evaluate f(x) at estimated values
    figure(fig); fig=fig+1;
    plot(t,fx,'r',t,ytil,'b');
    xlabel 'Time (sec)'; ylabel 'Pitch \theta rad';
    legend('Estimated','Real'); title 'Estimated values for pitch';
end
```

Code to design filters.

```
close all; clear all; clc; fig=1;

lowpass=load('lowpass.mat');
data=lowpass.lowpass;
smoothsnl=0;
xt=0;
xf=0;
[ylqr,timlqr,fig]=labviewdata(data,7,smoothsnl,fig,xt,xf,0);
ylqr.A(1:2,:)=[];
timlqr.A(1:2)=[];

%% Real time parameters

x0=ylqr.A(1,5);
T=timlqr.A;
u=ylqr.A(:,5);

%% Filters frequency

f=0.0714*2;
w=2*pi*f;

%% 1st order Low Filter

format long
lpf_1st=tf([w], [1, w])
format short
lpf=tf({[w],0;0,[w]},{[1, w],1;1,[1, w]});

[y,tim]=lsim(lpf_1st,u,T,x0); % Open loop simulation
figure(fig); fig=fig+1; plot(T,u);
hold on; plot(tim,y,'r'); hold off; xlabel 'Time sec'; ylabel 'Output';
legend('input','filter'); title 'First Order Low Pass Filter';

%% 2nd order Low Filter

z=1/sqrt(2);
format long
lpf_2nd=tf([w^2],[1,2*z*w,w^2])
format short

% Simulation

[y,tim]=lsim(lpf_2nd,u,T,x0); % Open loop simulation
figure(fig); fig=fig+1; plot(T,u);
hold on; plot(tim,y,'r'); hold off; xlabel 'Time sec'; ylabel 'Output';
legend('input','filter'); title 'Second Order Low Pass Filter';
```

Code to calculate equations of motion.

```
clear all; close all; clc
digits(5);

%% Drawing

%              Fb
%              -----
%             |-----|         TOP VIEW
%               | |
%               | |         z(out the screen)
%               | |          O ------>y
%               | |          |
%               | |          |    Coordinate System
%   Origin --> |O|          V
%               | |          x
%               | |
%               | |
%               | |
%               | |
%               | |
%               | |
%   Fr          | |             Fl
%   _           | |              _
%  | |----------| |------------| |
%  | |------------------------| |
%  |_|                        |_|

%% Define variables

global pddl tddl phiddl trimeqs
syms p t phi pd td phid pdd tdd phidd % euler angles
syms Fr Fl % Inputs

n=[p;t;phi];              % euler angles vector, psi, theta and phi
nd=[pd;td;phid];          % euler velocities vector
ndd=[pdd;tdd;phidd];      % euler acelerations vector

q=[n;nd];         % state vector
qd=[nd;ndd];      % velocities vector

forces=[Fl;Fr]; % Forces

% Formula to calculate derivative with respect of time
% dF/dt=transpose(simplify(dot(jacobian(function,variables)',
%                                          [variablesdot #equations])));

%% Motor Gain (From experimental results)

BitIn=[0 25 34 48 59 68 80 97 102 109 120 130 144 158 ...
                 164 178 195 210 230 235];
Thrust=[0 7 12 18 24 29 34 40 45 50 55 60 64 71 77 83 ...
                 91 97 106 110]*9.81/1000;

norder=1;
Kgain=polyfit(BitIn,Thrust,norder); % Get linear equation that relates
                                      % Bytes and Thrust
Kvr=Kgain(1);                    % Gain motor right
Kvl=Kgain(1);                    % Gain motor left
Tvr=0.3;
Tvl=0.3;
Bit0=-Kgain(2)/Kgain(1); % No thrust speed
```

```
%% System Parameters

Mp=2.02270866;    % Link2 mass kilograms
Mr=0.45695968;    % Link3 mass

lc=[-0.11010108,0,-0.01657261]';     % Link2 center of mass meters
l3=0.27475;         % Link3 length meters
l2=0.67098;         % Link2 length meters
lc2=[0.00079903+l2,0,-0.02671306]'; % Link3 center of mass meters

dmpY=0.002037386086782; % Yaw damping coefficient Ns/m
dmpP=0.067766320378873; % Pitch damping coefficient
dmpR=0.002037386086782; % Roll damping coefficient
g=9.8;              % Gravity m/s^2

%% Inertia Tensor matrix (From Solid Works drawings)

% Link2 inertia
Ic=[0.13031678 0.00000000 0.00000000;...
        0.00000000 0.12982028 0.00559693;...
        0.00000000 0.00559693 0.00175924];
Ic=Ic([3 1 2],[3 1 2]);
Ic=Ic+Mp*(sum(lc.*lc)*eye(3)-lc*lc');

% Link3 inertia
Ic2=[0.00015069 0.00000000 0.00000000;...
         0.00000000     0.01358558 0.00000468;...
         0.00000000     0.00000468 0.01364239];
Ic2=Ic2([3 1 2],[3 1 2]);
Ic2=Ic2+Mr*(sum(lc2.*lc2)*eye(3)-lc2*lc2');

%% Transformation matrices

% Roll to Pitch
Tpr=[1 0 0 l2;0 cos(phi) -sin(phi) 0;0 sin(phi) cos(phi) 0;0 0 0 1];
% Pitch to Yaw
Typ=[cos(t) 0 sin(t) 0;0 1 0 0;-sin(t) 0 cos(t) 0;0 0 0 1];
% Yaw to Ground
Tgy=[cos(p) -sin(p) 0 0;sin(p) cos(p) 0 0;0 0 1 0;0 0 0 1];

Tgp=Tgy*Typ;    % Pitch to Ground
Tgr=Tgp*Tpr;    % Roll to Ground

Tyr=Typ*Tpr;    % Roll to Yaw

TFr=[1 0 0 0;0 1 0 -l3;0 0 1 0;0 0 0 1];    % Right Force to Roll
TFl=[1 0 0 0;0 1 0 l3;0 0 1 0;0 0 0 1];     % Left Force to Roll

TFgr=Tgr*TFr;   % Right Force to Ground
TFgl=Tgr*TFl;   % Left Force to Ground

%% Rotational and translational matrices Positions

PpH=Tgp*[lc;1];     %Pitch COM Position
PrH=Tgp*[lc2;1];     %Roll COM Position
Pp=PpH(1:3);
Pr=PrH(1:3);

PFrH=Tgr*[0;-l3;0;1];        %Right Force Position
PFlH=Tgr*[0;l3;0;1];         %Left Force Position
PFr=PFrH(1:3);
PFl=PFlH(1:3);
```

```
%% Velocities Rates

Vp=(jacobian(Pp,n))*nd;          %Pitch COM Velocity
Vr=(jacobian(Pr,n))*nd;          %Roll COM Velocity
VFr=(jacobian(PFr,n))*nd;        %Right Force Velocity
VFl=(jacobian(PFl,n))*nd;        %Left Force Velocity

%% Angular Velocities

OmegapH=transpose(Typ)*[0;td;pd;1];
Omegap=OmegapH(1:3);                        %Pitch Angular Velocity

OmegarH=transpose(Tyr)*[0;td;pd;1];
Omegar=OmegarH(1:3)+[phid;0;0];             %Roll Angular Velocity

%% Kinetic Energy

% KE = 1/2 * mass * velocity ^ 2 + 1/2 * inertia * angular velocity ^ 2
KEp=0.5*Mp*transpose(Vp)*Vp+0.5*transpose(Omegap)*Ic*Omegap;     %Pitch KE
KEr=0.5*Mr*transpose(Vr)*Vr+0.5*transpose(Omegar)*Ic2*Omegar;    %Roll KE
KE=KEp+KEr;

%% Potential Energy

% PE = mass * gravity * position in z(0,0,1)
PE=Mp*g*transpose(Pp)*[0;0;1]+Mr*g*transpose(Pr)*[0;0;1];

%% Forces

FleftH=Fl*(Tgr*[0;l3;1;1]-Tgr*[0;l3;0;1]);
Fleft=FleftH(1:3);
FrightH=Fr*(Tgr*[0;-l3;1;1]-Tgr*[0;-l3;0;1]);
Fright=FrightH(1:3);

%% Power In

% Pin = force * velocity
Pin=vpa(simplify(transpose(Fleft)*VFl+transpose(Fright)*VFr));

%% Eqs1 by Euler-Lagrange (EL)

% d/dt dL/dqd - dL/dq + D = F
% L = Lagrangian = KE-PE
% F = external forces
% D = damping forces

% derivatives with respect of velocity, then with time
Term11=vpa(simplify(jacobian(diff(KE,pd),q)*qd));
Term21=vpa(simplify(jacobian(diff(KE,td),q)*qd));
Term31=vpa(simplify(jacobian(diff(KE,phid),q)*qd));

% derivatives with respect of position
Term12=vpa(simplify(diff(KE,p)));
Term22=vpa(simplify(diff(KE,t)));
Term32=vpa(simplify(diff(KE,phi)));

U1=vpa(diff(PE,p));
U2=vpa(diff(PE,t));
U3=vpa(diff(PE,phi));

% external forces = dPin/dqd
Q1=vpa(diff(Pin,pd));
Q2=vpa(diff(Pin,td));
Q3=vpa(diff(Pin,phid));
```

```
% damping forces
D1=dmpY*[1,0,0]*(Omegap+Omegar);
D2=dmpP*[0,1,0]*(Omegap+Omegar);
D3=dmpR*[0,0,1]*(Omegap+Omegar);

%% Equations by Euler-Lagrange (EL)

% EL= d/dt dL/dqd - dL/dq - F + D
Eqn1=simplify(Term11-Term12+U1-Q1+D1);
Eqn2=simplify(Term21-Term22+U2-Q2+D2);
Eqn3=simplify(Term31-Term32+U3-Q3+D3);

%% Inertia Matrix

% M(q) = dEL/dqdd
In=jacobian([Eqn1,Eqn2,Eqn3],ndd);

%%
H1=simplify(jacobian([Eqn1],ndd)*ndd-Eqn1);
H2=simplify(jacobian([Eqn2],ndd)*ndd-Eqn2);
H3=simplify(jacobian([Eqn3],ndd)*ndd-Eqn3);

trimeqs=[H1,H2,H3];
save('Trimmat','trimeqs');

%% State Equations, Equation of motion

Fx=inv(In)*[H1;H2;H3];

pddl=Fx(1); % Finding eqs for pdd tdd and phidd
tddl=Fx(2); % Finding eqs for pdd tdd and phidd
phiddl=Fx(3); % Finding eqs for pdd tdd and phidd

save('Eqsmtn','pddl','tddl','phiddl');

%% Trim condition

xtrim=[0,0,0,0,0,0,0,0]';
trimy=trimp(xtrim) % Calculate trim at (p,t,phi,pd,td,phid)

%% State space representation

[A,B,C,D]=linmod('Gyromdl',trimy(1:6),trimy(7:8)); % Linear system,
% Gyromodel is the model in symulink, it calls the function gyroeqs to
% make the linearization
syslinear=ss(A,B,C,D);
save('SSmatrx','A','B','C','D');

%% Discrete representation

% sysdiscrete=c2d(syslinear,0.02,'zoh');
sysdiscrete=c2d(syslinear,0.02);
[Ad,Bd,Cd,Dd,Ed,Td]=dssdata(sysdiscrete);
save('SSmatrxd','Ad','Bd','Cd','Dd','Ed','Td');

%% Tranfer function V

% T/V=(Kvl/Tvl)/(s+1/Tvl); % Transfer function Bytes --> Thrust
```

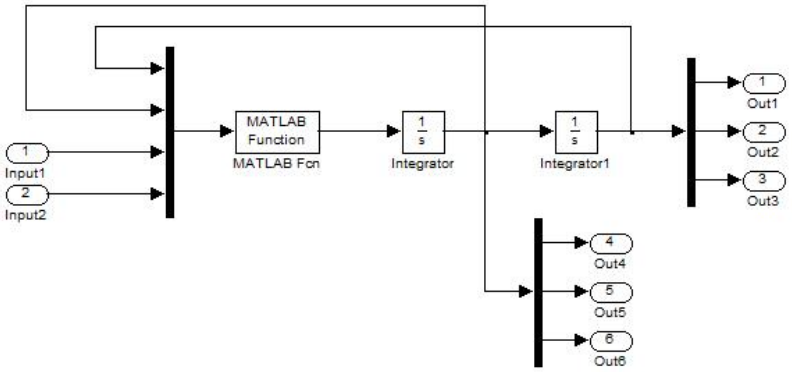
Function *trimp.m*.

```
function trimv=trimp(x)

%
%TRIMP  Finds the trim point of gyrocopter's dynamics.
%     TRIMV = TRIMP(X)
%
%     X       = states, minimum a 2 column vector, containing the values of
%               psi and theta at the desired trim point.
%     TRIMV   = output vector, a 8 column vector containing the 8 states
%               [psi,theta,phi,psid,thetad,phid,Fl,Fr].
%
%     Sintax example:
%          trimv=trimp([pi;0]);

global trimeqs
syms p t phi pd td phid %euler angles
syms Fr Fl %Inputs

p=x(1);
t=x(2);
pd=0;
td=0;
phid=0;

[Fltrim,Frtrim,phitrim]=solve(subs(trimeqs(1)),...
                              subs(trimeqs(2)),...
                              subs(trimeqs(3))); % Trim conditions
trimv=[p;t;zeros(4,1);abs([double(Fltrim(1));double(Frtrim(1))])];
```

Function *gyroeqs.m*.

```
function qdd=gyroeqs(q)
global pddl tddl phiddl

% VARIABLES

p=q(1);
t=q(2);
phi=q(3);
pd=q(4);
td=q(5);
phid=q(6);
Fl=q(7);
Fr=q(8);

% EQS OF MOTION

qdd=zeros(3,1);
qdd(1)=subs(pddl);
qdd(2)=subs(tddl);
qdd(3)=subs(phiddl);
```

File *gyromdl.mdl*.



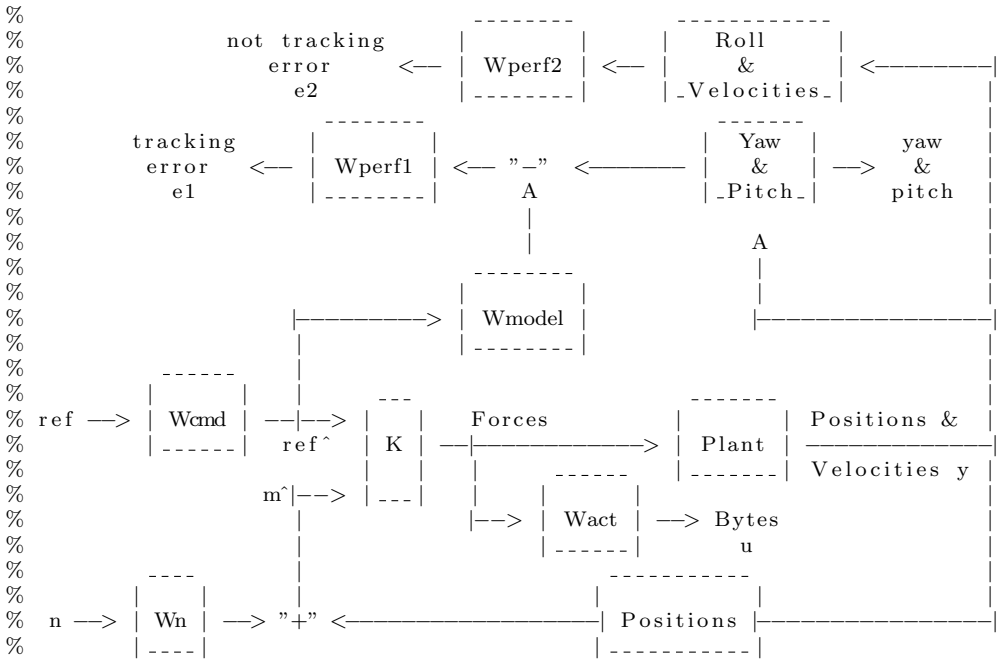
Code to design and simulate the H_∞ controller.

```
clear all; close all; clc; fig=1;

%% Define variables

global pddl tddl phiddl trimeqs
syms p t phi pd td phid %euler angles
syms Fr Fl %Inputs

%% Motor Gain (From experimental results)

BitIn=[0 25 34 48 59 68 80 97 102 109 120 130 144 158 ...
                164 178 195 210 230 235];
Thrust=[0 7 12 18 24 29 34 40 45 50 55 60 64 71 77 83 ...
                91 97 106 110]*9.81/1000;

norder=1;
Kgain=polyfit(BitIn,Thrust,norder); % Get linear equation that relates
                                    % Bytes and Thrust
Kvr=Kgain(1);                   % Gain motor right
Kvl=Kgain(1);                   % Gain motor left
Tvr=0.01;
Tvl=0.01;
Bit0=-Kgain(2)/Kgain(1); % No thrust speed

%% Load Variables

phiddl=load('Eqsmtn.mat');
pddl=phiddl.pddl;
tddl=phiddl.tddl;
phiddl=phiddl.phiddl;

trimeqs=load('Trimmat.mat');
trimeqs=trimeqs.trimeqs;

%% Trim condition

xtrim=[0,0,0,0,0,0,0,0]';
trimy=trimp(xtrim); % Calculate trim at (p,t,phi,pd,td,phid)

%% State space representation

syslinear=load('SSmatrx.mat');
A=syslinear.A;
B=syslinear.B;
C=syslinear.C;
D=syslinear.D;

Nx=eye(6);
Ny=eye(6);
Nu=eye(2);

%% Scale system

xmax=[pi/10;pi/10;pi/10;8*pi/180;5*pi/180;5*pi/180];
Nx=diag(xmax);
umax=[1.2;1.2];
Nu=diag(umax);
ymax=xmax;
Ny=diag(ymax);

A=inv(Nx)*A*Nx; B=inv(Nx)*B*Nu;
```

```
C=inv(Ny)*C*Nx; D=inv(Ny)*D*Nu;

%% Plant G

% Plant G is equal to the state space given by:
% xd = Ax + Bu
%  y = Cx + Du
G=ss(A,B,C,D);

%% Wieghts

s=zpk('s');

%W command, Butterworth filter - High frequency roll off
fwc=1;
wc=2*pi*fwc;
Wcmd=wc/(s+wc)*eye(2);

%W performance 1, tracking objetive
% f1=5; f2=1000;
f1=1; f2=1000;
Wp=2*(1/2/pi/f2*s+1)/(1/2/pi/f1*s+1);
Wt=2*(1/2/pi/f2*s+1)/(1/2/pi/f1*s+1);
Wperf1=[Wp 0;0 Wt];
% figure(fig); fig=fig+1; bode(Wp)

%W performance 2, not in the tracking objetive
fwr=5;
wr=2*pi*fwr;
Wperf2=wr/(s+wr)*eye(4);

%W actuator
fa=5;
wa=2*pi*fa;
Wact=[wa/(s+wa) 0;0 wa/(s+wa)];

%W noises more than 0.3515625 (360/1024) for positions & velocities
Wn=inv(Ny(1:3,1:3))*diag([360 360 360])/1024/57.3;

%W model, desired model to match
ts=10;                                % Setting time of 10 seconds
xi=1.5; wn=4.6/ts/xi;                   % Damping ratio and frequency
Wmodely=wn^2/(s^2+2*xi*wn*s+wn^2);  % Second order system
ts=10;
xi=1.5; wn=4.6/ts/xi;
Wmodelp=wn^2/(s^2+2*xi*wn*s+wn^2);
Wmodel=[Wmodely 0;0 Wmodelp];

%% Close Loop diagram

%                              ------------
%                             |            |
%            Yaw ref  ---->  |            | ---> Yaw error
%          Pitch ref  ---->  |   Plant    | ---> Pitch error
%              noise  ---->  |            | ---> Yaw real    | tracking
%                            |     G      | ---> Pitch real  |
%                            |            | ---> Roll            |   not
%                            |            | ---> Yaw velocity    | tracking
%                            |            | ---> Pitch velocity  |
%                            |            | ---> Roll velocity   |
%                            |            | ---> Bytes Left
%                            |            | ---> Bytes Right
%                            |            |
% ncon | Thrust left  |--->  |            | ---| Yaw ref^      | nmeas
```

```
%      | Thrust right |    |------------|    | Pitch ref^   |
%                         |                  | Yaw sensor   |
%                         |                  | Pitch sensor |
%                         |                  | Roll sensor  |
%                         |                  |
%                         |    --------------    |
%                         |   |              |   |
%                        |-- | Controller K | <-|
%                            |--------------|

%% Signals

% Inputs, external

pref=icsignal(1); % Yaw (psi)
tref=icsignal(1); % Pitch (theta)
   n=icsignal(3); % Noises

% Inputs, internal, from K to G

Tl=icsignal(1); % Thrust motor left
Tr=icsignal(1); % Thrust motor rigth

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Outputs, external

perror=icsignal(1); % Yaw error
terror=icsignal(1); % Pitch error
 preal=icsignal(1); % Yaw real
 treal=icsignal(1); % Pitch real
  phir=icsignal(1); % Roll real
   pdr=icsignal(1); % Yaw velocity real
   tdr=icsignal(1); % Pitch velocity real
phidr=icsignal(1); % Roll velocity real
   Bl=icsignal(1); % Bytes motor left
   Br=icsignal(1); % Bytes motor rigth

% Outputs, internal from G to K

 pref1=icsignal(1); % Wcmd * Yaw ref
 tref1=icsignal(1); % Wcmd * Pitch ref
   pgk=icsignal(1); % Yaw sensor
   tgk=icsignal(1); % Pitch sensor
phigk=icsignal(1); % Roll sensor

%% System M

M=iconnect;

% Define external inputs without K
M.Input=[pref;tref;... % references
         n;...         % noises
         Tl;Tr];       % inputs to G

% Define external outputs without K
M.Output=[perror;terror;...                % tracking errors
          preal;treal;...                  % real values
          phir;pdr;tdr;phidr;...           % not tracking error
          Bl;Br;...                        % input to plant
          pref1;tref1;...                  % references after weighting
          pgk;tgk;phigk];                  % pgk;tgk;phigk;pdgk;tdgk;phidgks
```

```
Ax=[perror;terror;...
    preal;treal;...
    phir;pdr;tdr;phidr;...
    Bl;Br;...
    pref1;tref1;...
    pgk;tgk;phigk];

Bx=[Wperf1*(Wmodel*Wcmd*[pref;tref]-G(1:2,:)*[Tl;Tr]);...
    G(1:2,:)*[Tl;Tr];...
    Wperf2*G(3:6,:)*[Tl;Tr];...
    Wact*[Tl;Tr];...
    Wcmd*[pref;tref];...
    Wn*n+G(1:3,:)*[Tl;Tr]];

M.Equation{1}=equate(Ax,Bx);

Tsys=M.system;
Tsys=minreal(Tsys);

%% H-inf Block Diagram
```




```
%% Hinf Controller Synthesis

ncon=2;  % Number of control signals
nmeas=5; % Number of signals into the controller

[K,CL,GAM,INFO]=hinfsyn(Tsys,nmeas,ncon,'GMIN',0,'GMAX',inf,'METHOD','lmi');
[Ka,Kb,Kc,Kd]=ssdata(K);
if mean(eig(K)<0)==1
    disp('K stable');
else
    disp('K not stable');
end
disp('GAM Close Loop');disp(GAM);

%% Singular value, maximum frequency
```

```
figure(fig);
h_norm_inf(K,'Controller K',0.02,2*pi*5,fig)
fig=fig+2;

%% Reference Input Parameters

T=0:0.1:100; % Simulation time
z=zeros(1);

p0=z; t0=z;                          % Initial conditions
phi0=z; pd0=z; td0=z; phid0=z;
fr10=z; fr20=z;
x0=[p0,t0,phi0,pd0,td0,phid0,fr10,fr20,zeros(1,size(CL.a,1)-8)];

pr=-pi/10; tr=pi/8; % Reference inputs
xr=[pr,tr];
xr=(inv(Nx(1:2,1:2))*xr')'; % Scale references

      u1=xr(1)*ones(size(T));
      u2=xr(2)*ones(size(T));
   noise=zeros(size(n,1),length(T));

u=[u1;u2;noise]; % Input vector

%% Close loop simulation

[y,tim]=lsim(CL,u',T,x0);

y(:,1:2)=(Ny(1:2,1:2)*y(:,1:2)')';
figure(fig); fig=fig+1;
plot(tim,y(:,1),'b',tim,y(:,2),'r');
legend('Yaw Error','Pitch Error');
xlabel('Time sec'); ylabel('rad');

y(:,3:8)=(Ny*y(:,3:8)')';
figure(fig); fig=fig+1;
plot(tim,y(:,3),'b',tim,y(:,4),'r');
legend('Yaw','Pitch');
xlabel('Time sec'); ylabel('rad');

figure(fig); fig=fig+1;
plot(tim,y(:,5),'b',tim,y(:,6),'r',tim,y(:,7),'g',tim,y(:,8),'k');
legend('Roll','Yaw rate','Pitch rate','Roll rate');
xlabel('Time sec');

y(:,9:10)=(Nu*y(:,9:10)')';
figure(fig); fig=fig+1;
plot(tim,y(:,9),'b',tim,y(:,10),'r');
legend('Force Left','Force Right');
xlabel('Time sec'); ylabel('Thrust Newtons');

%% Display data

format long
disp('trimy');disp(trimy(7:8));
disp('Kgain');disp(Kgain);
disp('Wcmd frequency');disp(wc);
format short

%% Unscale Plant

A=Nx*A*inv(Nx); B=Nx*B*inv(Nu);
C=Ny*C*inv(Nx); D=Ny*D*inv(Nu);
```

```
format long
disp('Nx');disp(diag(Nx)');
disp('Ny');disp(diag(Ny)');
disp('Nu');disp(diag(Nu)');
disp('Nx^(-1)');disp(diag(inv(Nx))');
disp('Ny^(-1)');disp(diag(inv(Ny))');
disp('Nu^(-1)');disp(diag(inv(Nu))');
format short

%% Save Matrices

save matA.lvm A -ASCII -TABS;
save matB.lvm B -ASCII -TABS;
save matC.lvm C -ASCII -TABS;
save matD.lvm D -ASCII -TABS;
save matKa.lvm Ka -ASCII -TABS;
save matKb.lvm Kb -ASCII -TABS;
save matKc.lvm Kc -ASCII -TABS;
save matKd.lvm Kd -ASCII -TABS;
```

Function *h_norm_inf*.

```
function h_norm=h_norm_inf(sys,sysname,Ts,Wc,fig)

%
%H_NORM_INF computes the h-inf norm of SYS.
%     H_NORM=H_NORM_INF(SYS,SYSNAME) computes the h-inf norm of SYS. Plots
%     the MIMO frequency reponse and computes the Nyquist frequency wc of
%     SYS at 0 dB.
%
%     SYS     = system
%     SYSNAME = system name
%     H_NORM  = h-inf norm of the system
%
%     Sintax example:
%          h_norm=h_norm_inf(sys,'sysname');

[sv,w]=sigma(sys); % Gains and frequencies
sv1=max(sv,[],1);  % Maximum gains

h_norm=max(sv1); % H-inf norm

% modulus=(max(eig(sys)))

semilogx(w,20*log10(sv1)); % Plot singular values
title([sysname,' Singular Value plot']);
xlabel 'Frequency (rad/s)'; ylabel 'Gain (dB)';

wsvd=interp1(20*log10(sv1),w,0); % Nyquist frequency, at 0 dB

if isnan(wsvd)==1 % Plot below 0 dB
    [wsvd,wsvdind]=max(sv1);
    disp('Bode plot below 0 dB.');
    disp([sysname,' wc at max gain']);disp(w(wsvdind));
    disp([sysname,' frequency at max gain']);disp(w(wsvdind)/2/pi);
    hold on; semilogx(w,20*log10(wsvd));
    semilogx(w(wsvdind),20*log10(sv1)); hold off;
else
    disp([sysname,' wc']);disp(wsvd);
    disp([sysname,' Nyquist frequency']);disp(wsvd/2/pi);
    hold on; semilogx(w,0); semilogx(wsvd,20*log10(sv1)); hold off;
end

if nargin > 2
   figure(fig+1);
   sysd=c2d(sys,Ts,'prewarp',Wc);
   pzmap(sysd);
   title([sysname,' Pole-Zero Map']);
end
```

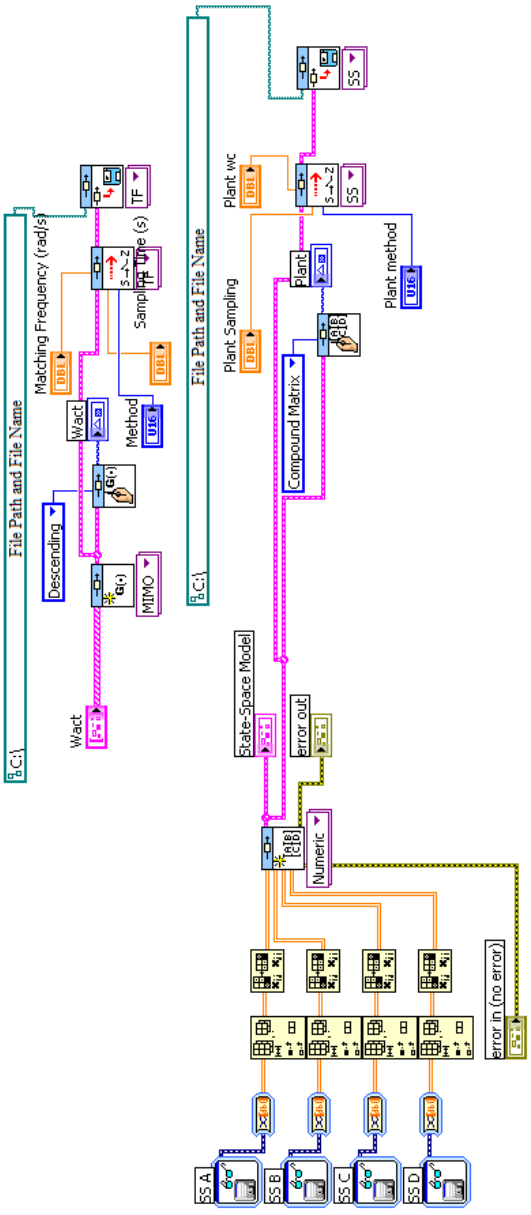
APPENDIX D

LABVIEW CODES

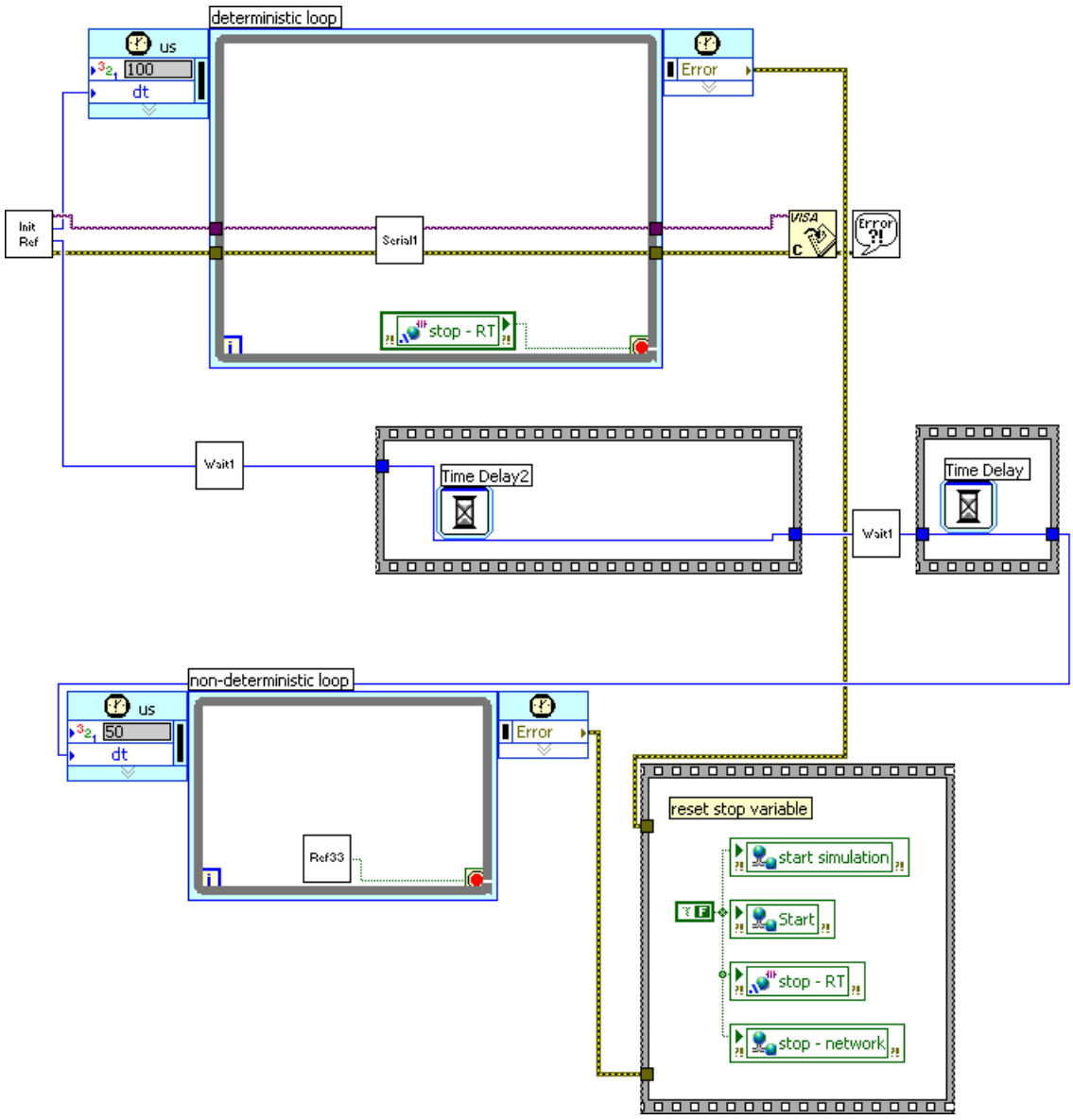
The codes shown are provided to users "as is", without warranty. There is no warranty for the codes, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose and noninfringement of third party rights. The entire risk as to the quality and performance of the codes is with the user. Should the codes prove defective, the user assume the cost of all necessary servicing, repair or correction.

In no event the author will be liable to anyone for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the codes (including but not limited to loss of data or data being rendered inaccurate or losses sustained by the user or third parties or a failure of the codes to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

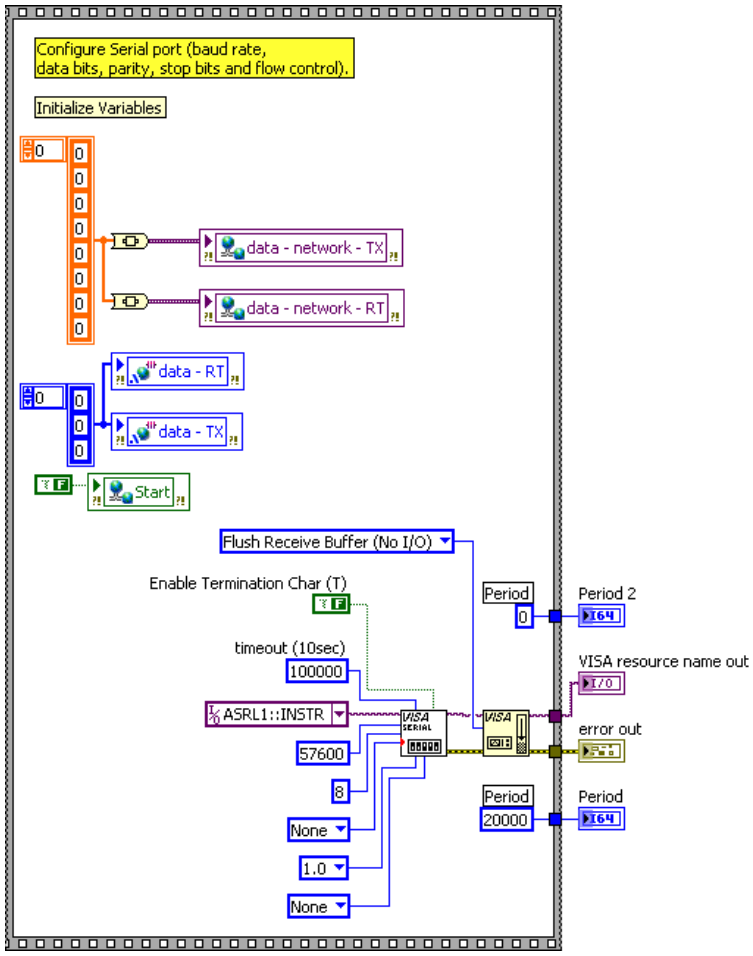
Code to read data from Matlab.



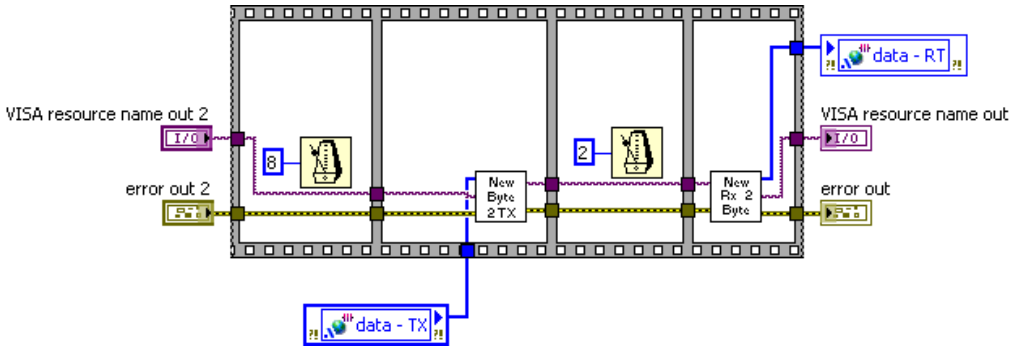
Main code in CompactRIO for H_∞.



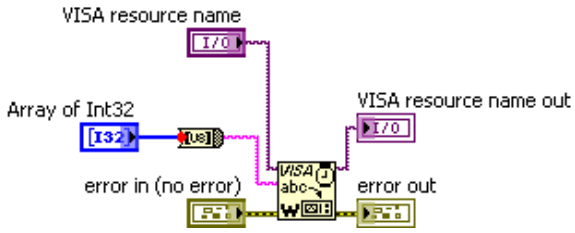
InitRef.vi


Configure Serial port (baud rate,
data bits, parity, stop bits and flow control).
Initialize Variables
data - network - TX
data - network - RT
data - RT
data - TX
Start
Flush Receive Buffer (No I/O)
Enable Termination Char (T)
timeout (10sec)
100000
ASRL1::INSTR
57600
8
None
1.0
None
Period
0
Period 2
VISA resource name out
error out
Period
20000
Period

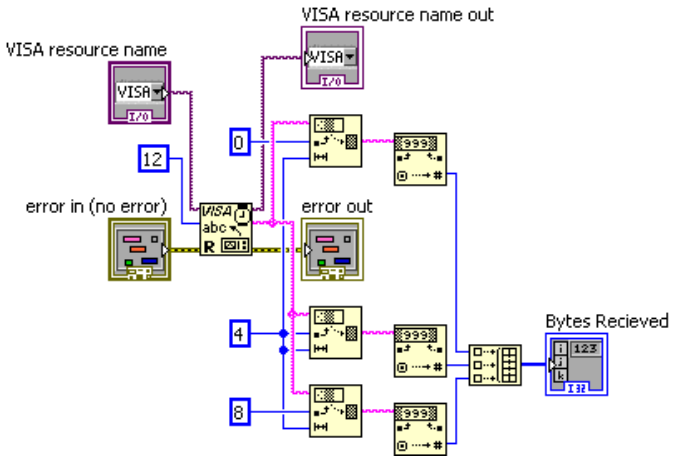
Serial1.vi



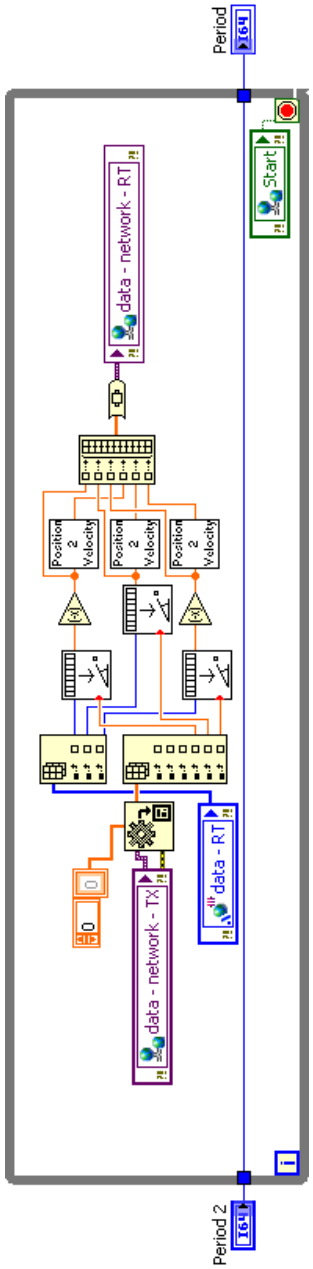
NewByte2Tx.vi



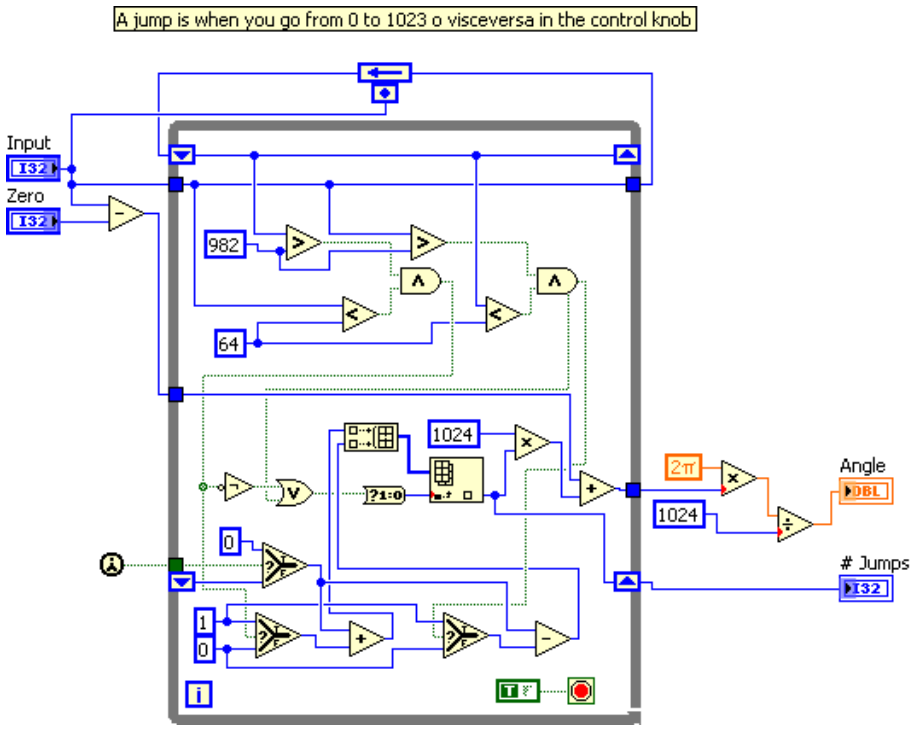
NewRx2Byte.vi



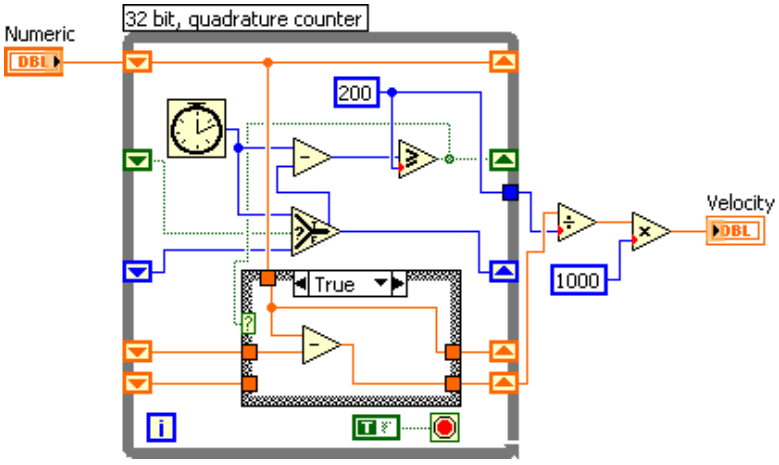
Wait1.vi


Period
I64
Start
data - network - RT
Position 2 Velocity
Position 2 Velocity
Position 2 Velocity
0
data - network - TX
data - RT
Period 2
I64

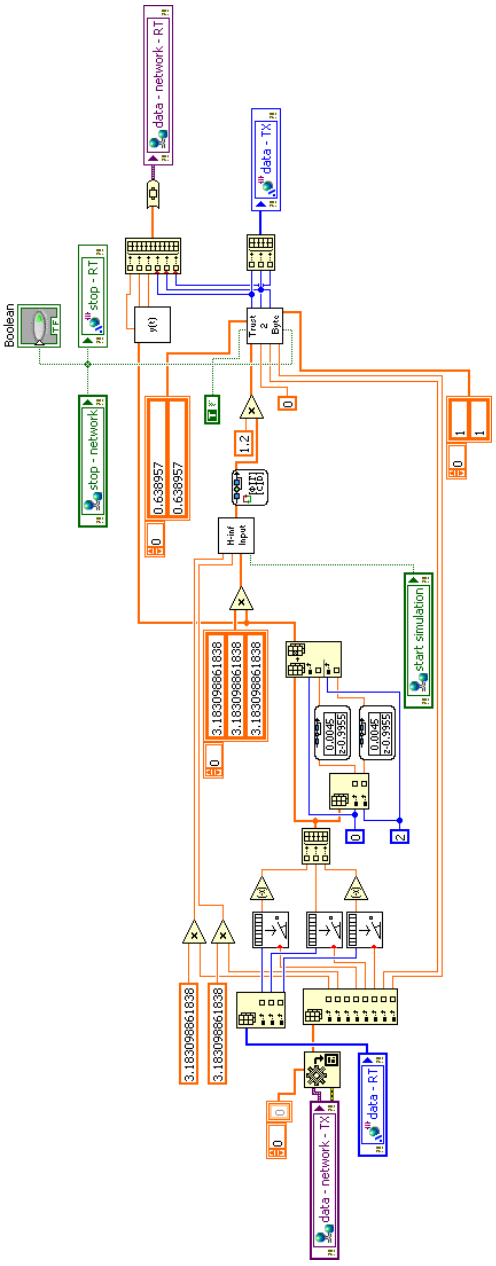
Byte2Angle.vi



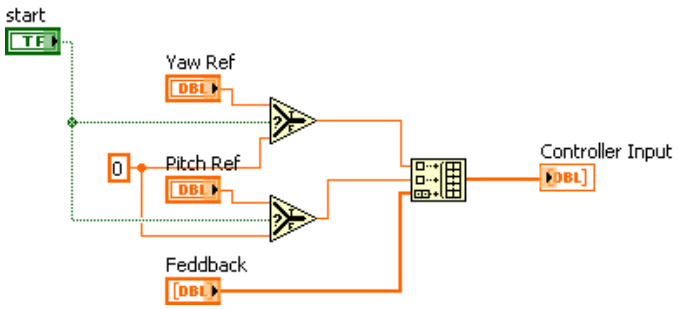
Position2Velocity.vi



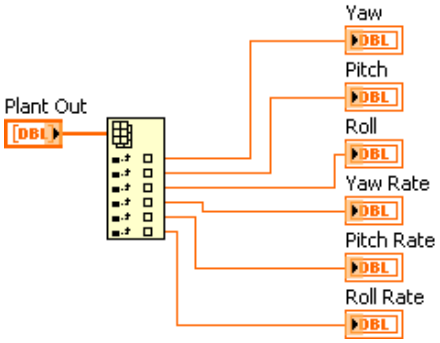
Ref33.vi


data - network - RT
data - TX
Boolean
stop - RT
stop - network
y(t)
Trust 2 Bytes
0.638957
0.638957
3.18309886 1838
start simulation
data - network - TX
data - RT

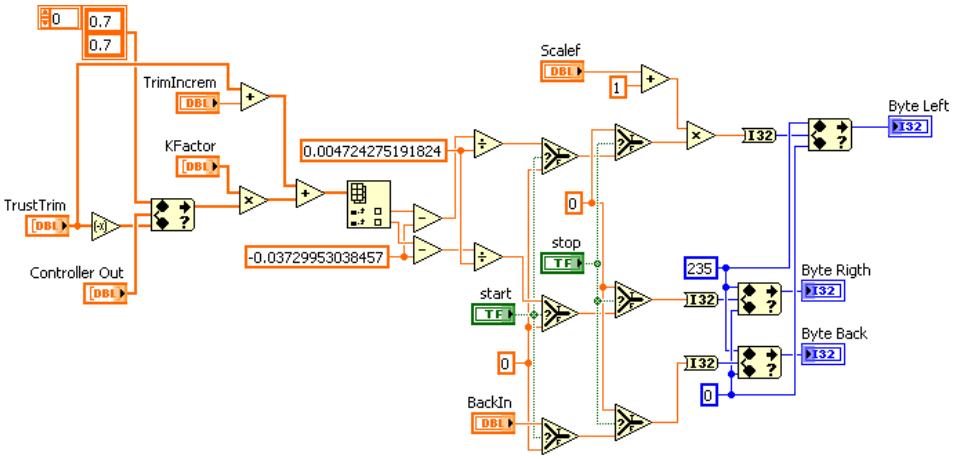
H-infInput.vi



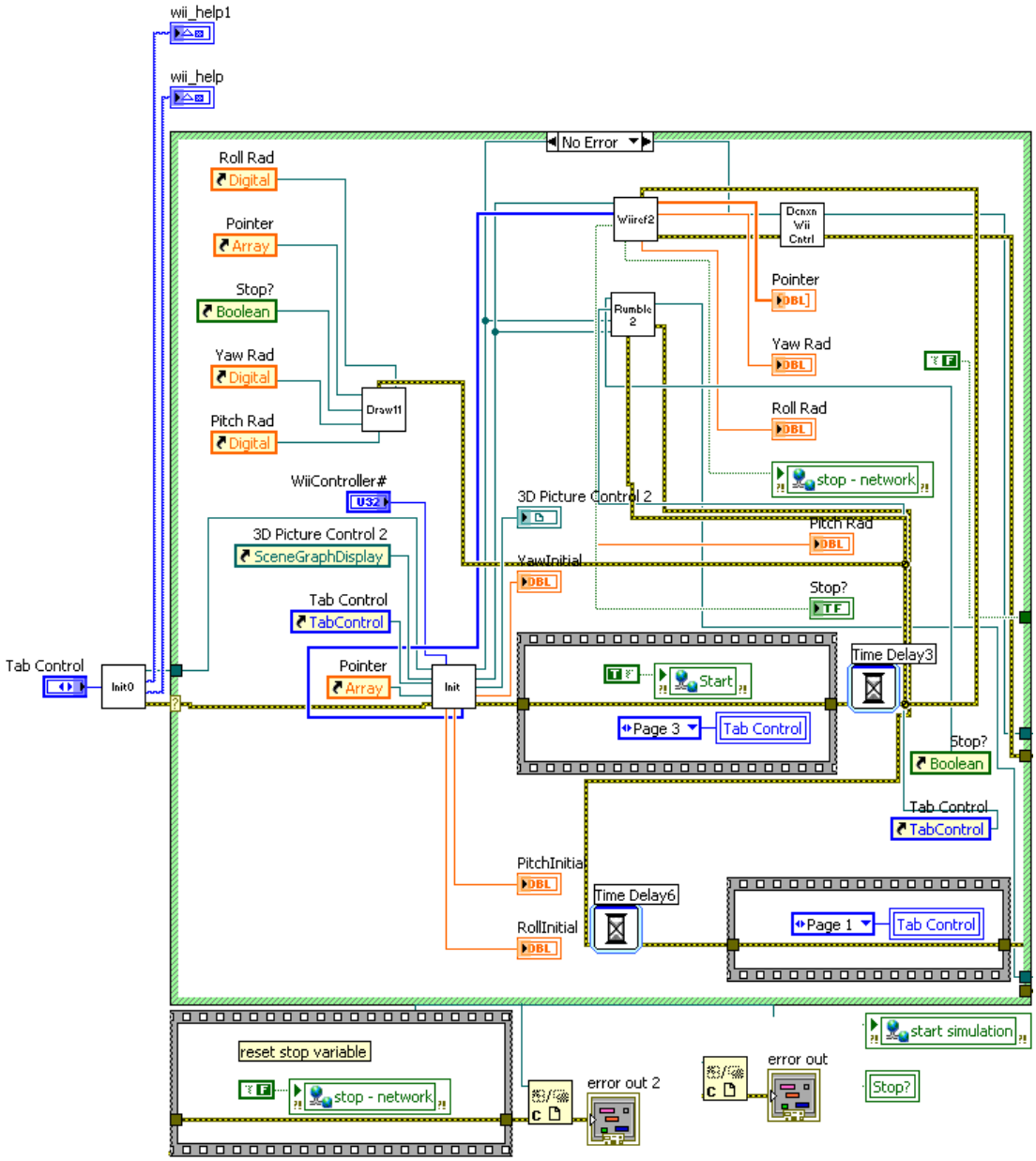
y-t.vi



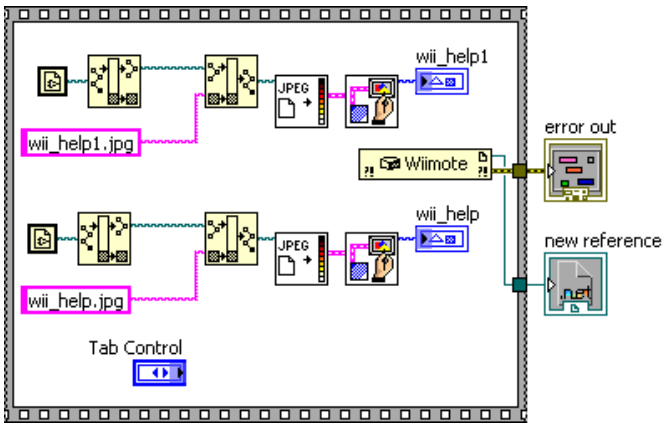
Thrust2Byte.vi



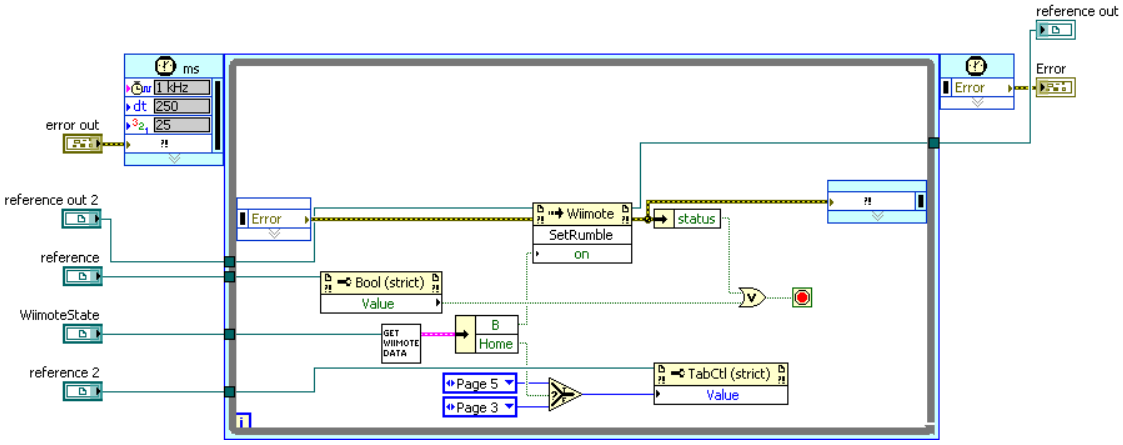
Main code for H_∞ user interface.



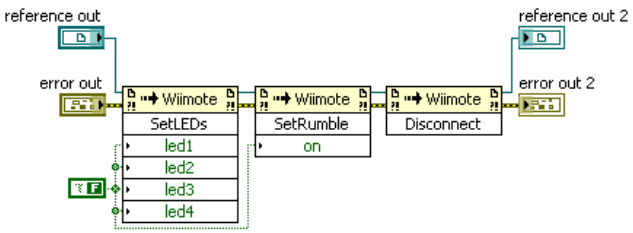
Init0.vi



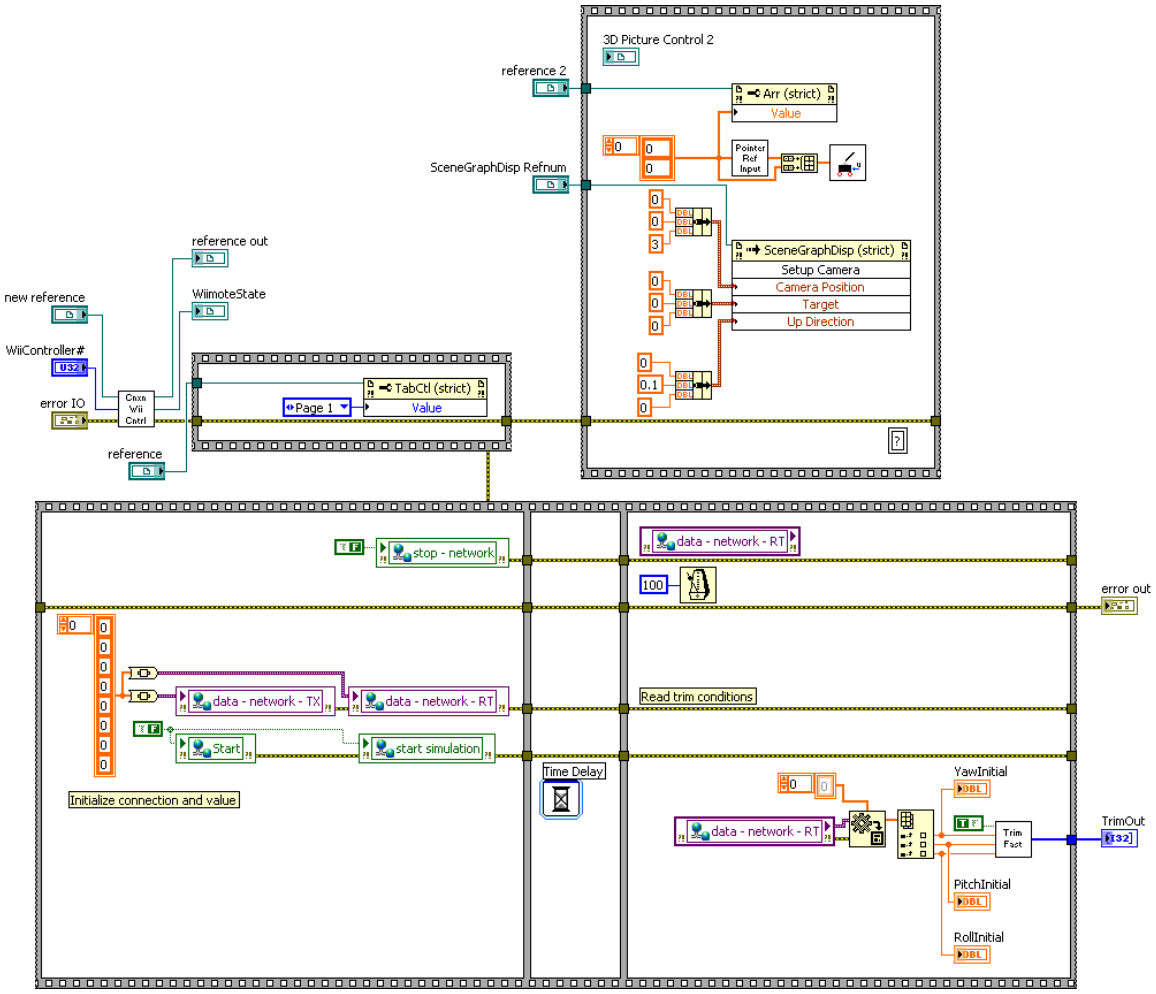
Rumble2.vi



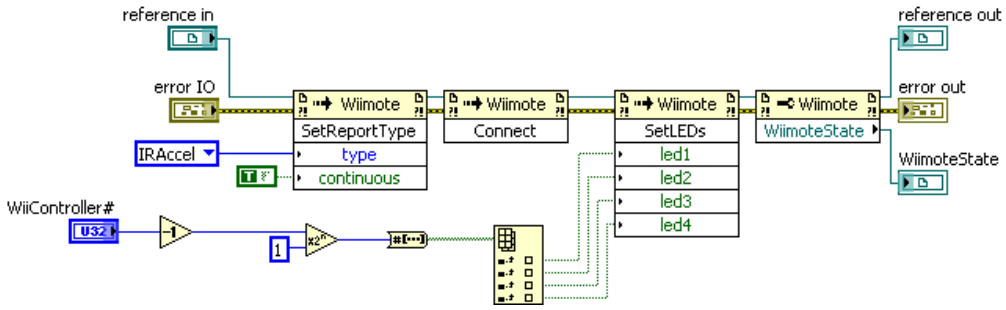
DcnxnWiiCntrl.vi



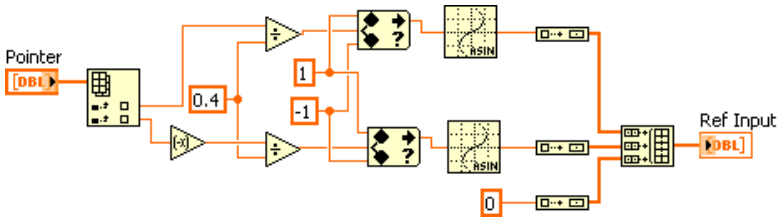
Init.vi


3D Picture Control 2
reference 2
Arr (strict)
Value
SceneGraphDisp Refnum
SceneGraphDisp (strict)
Setup Camera
Camera Position
Target
Up Direction
reference out
WiimoteState
new reference
WiiController#
error IO
reference
Cnxn Wii Cntrl
Page 1
TabCtl (strict)
Value
stop - network
data - network - RT
100
error out
data - network - TX
data - network - RT
Read trim conditions
Start
start simulation
Time Delay
Initialize connection and value
YawInitial
PitchInitial
RollInitial
TrimOut
Trim Fact

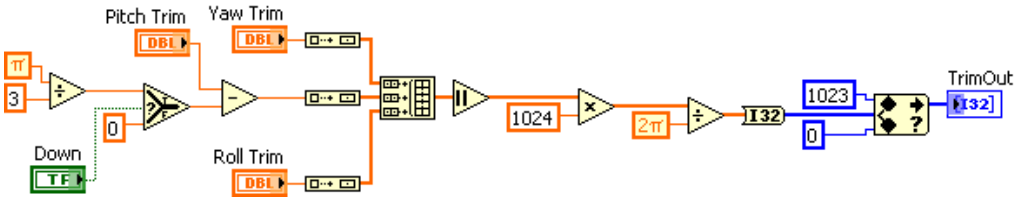
CnxnWiiCntrl.vi



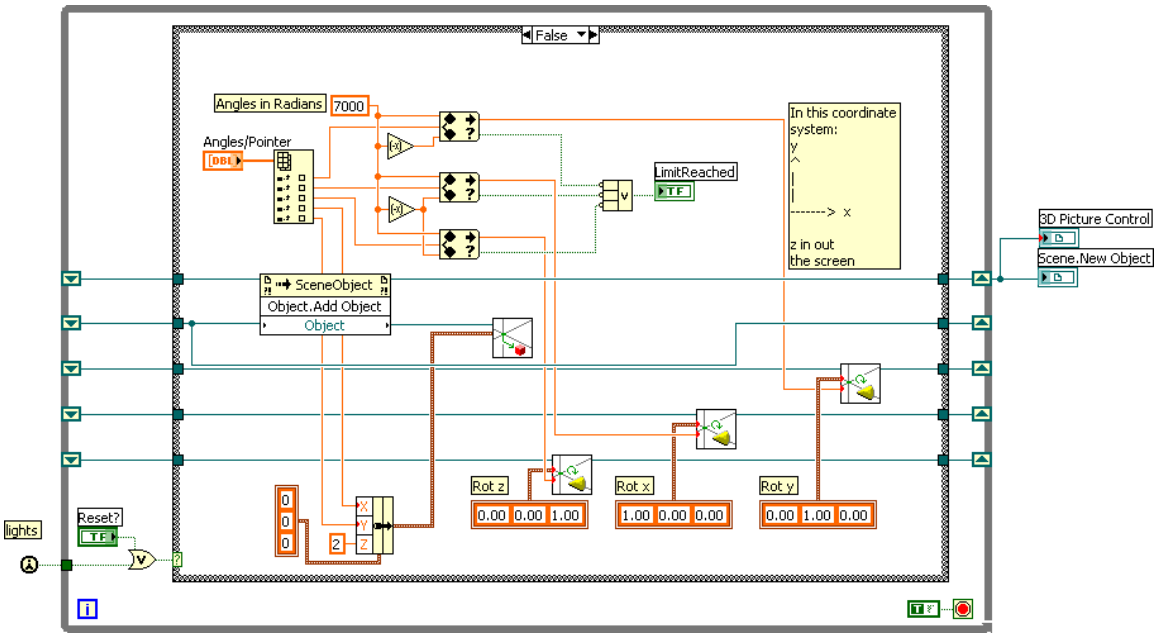
PointerRefInput.vi



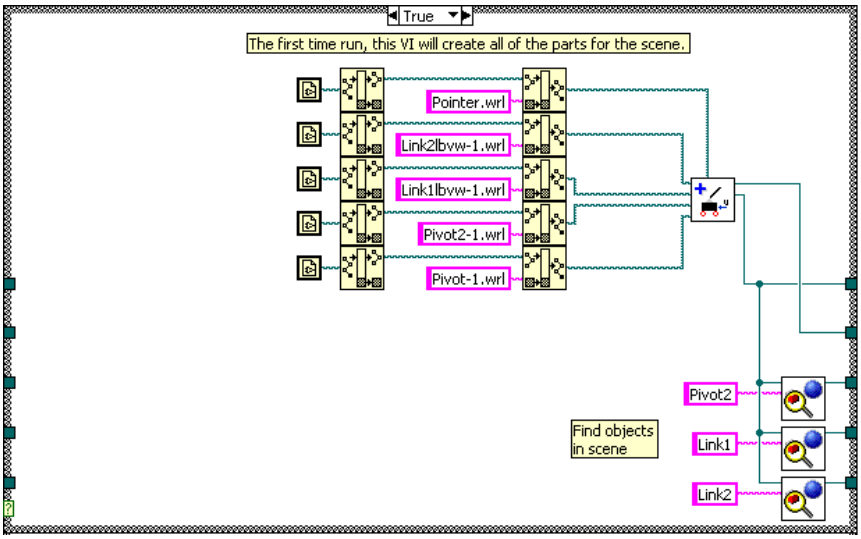
TrimFast.vi



Drawsim.vi

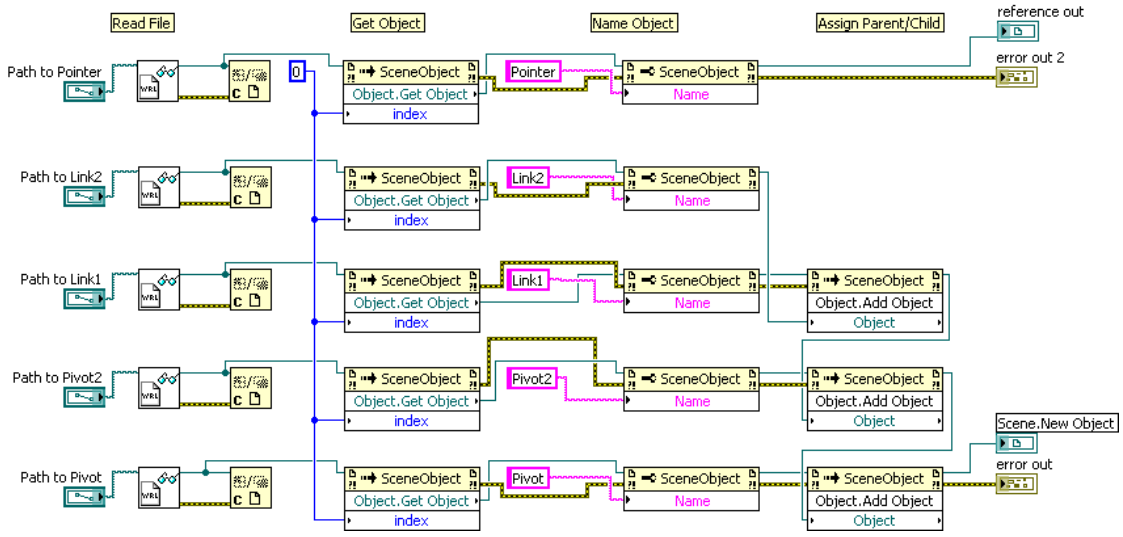


Hide panel of Drawsim.vi

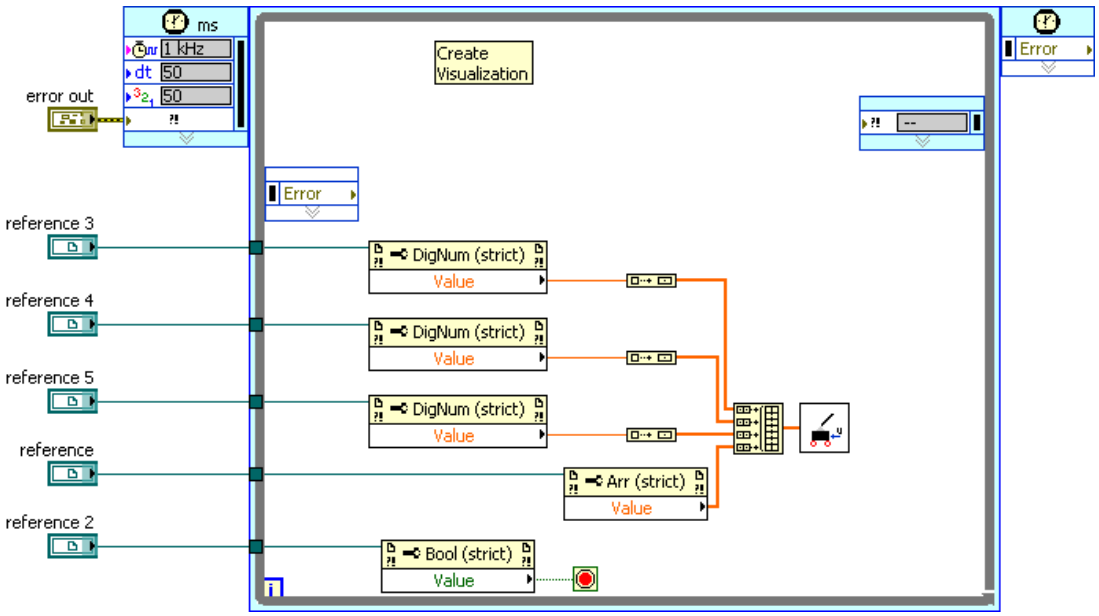


To generate the 3D graphs the models were done in SolidWorks and exported as VRML models.

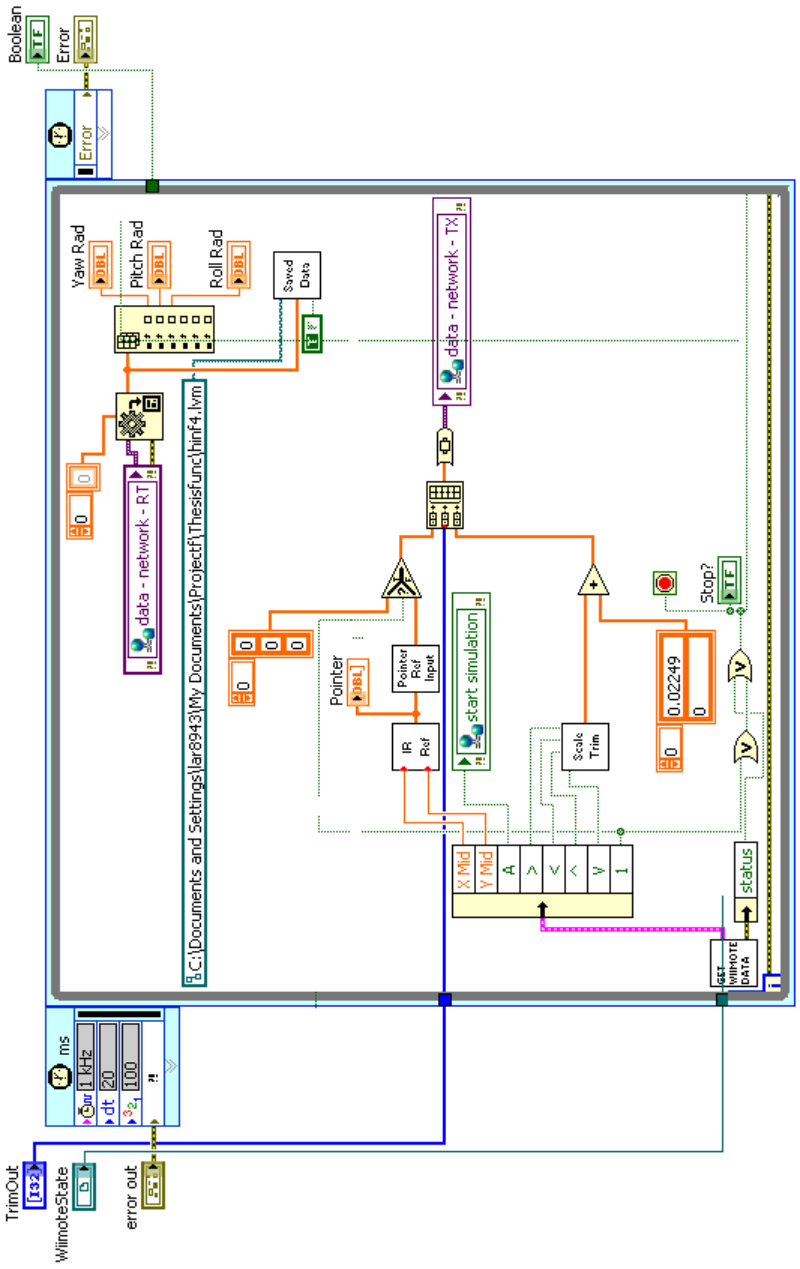
DrawVRML.vi



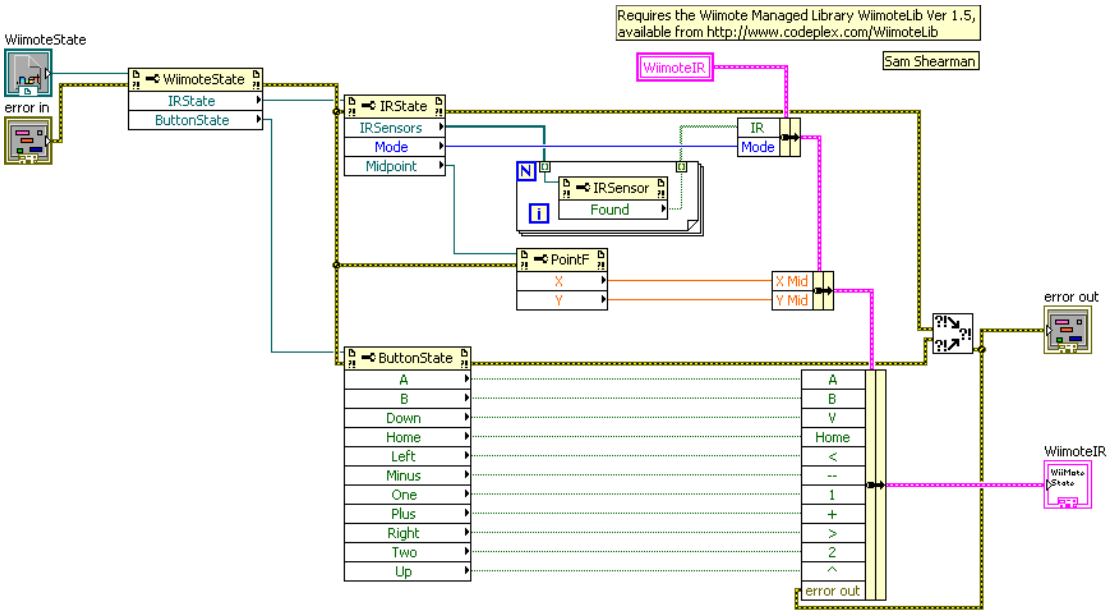
Draw11.vi



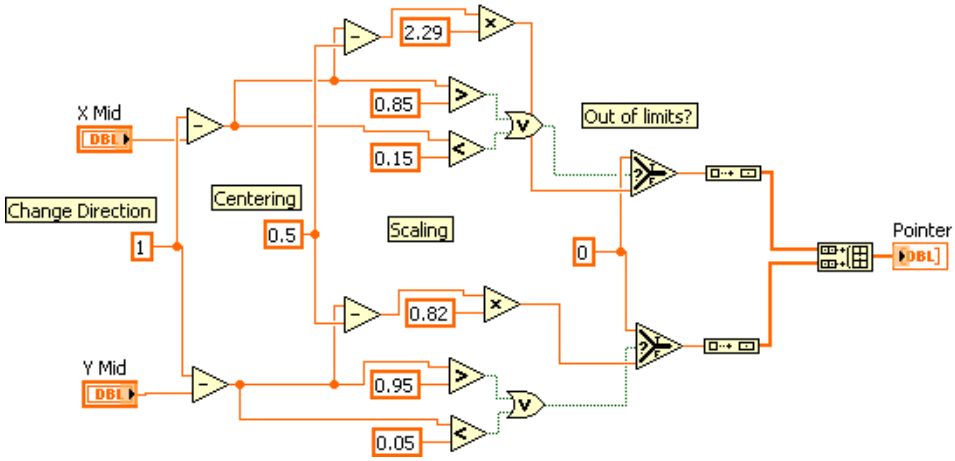
Wiiref2.vi


Boolean
Error
Error
Yaw Rad
Pitch Rad
Roll Rad
Saved Data
data - network - RT
data - network - TX
C:\Documents and Settings\lar8943\My Documents\Project1\Thesisfunc\hinf4.lvm
Pointer
Pointer Ref Input
IR Ref
start simulation
Scale Trim
0.02249
Stop?
X Mid
Y Mid
status
GET WIIMOTE DATA
ms
1 kHz
dt
20
100
TrimOut
WiimoteState
error out

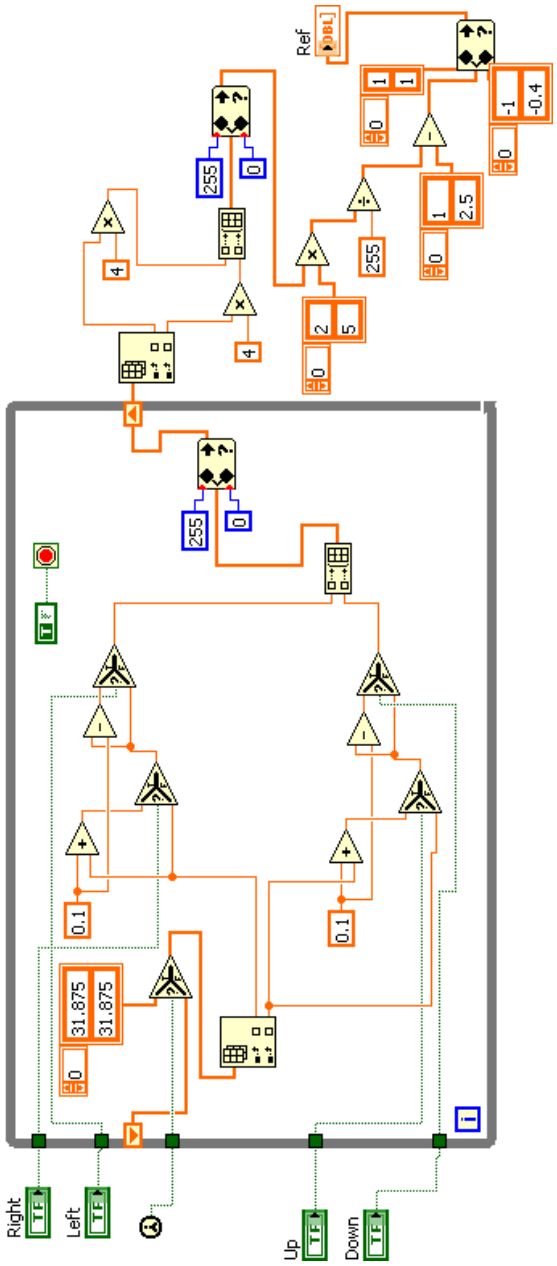
GetWiiMoteData.vi



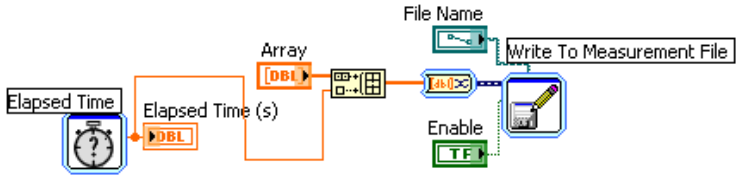
IRRef.vi



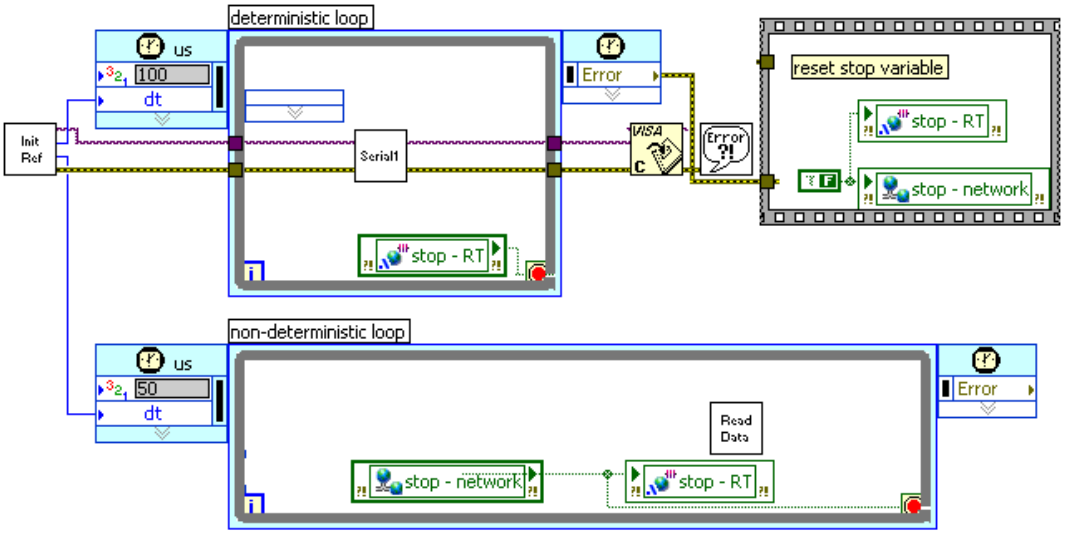
ScaleTrim.vi



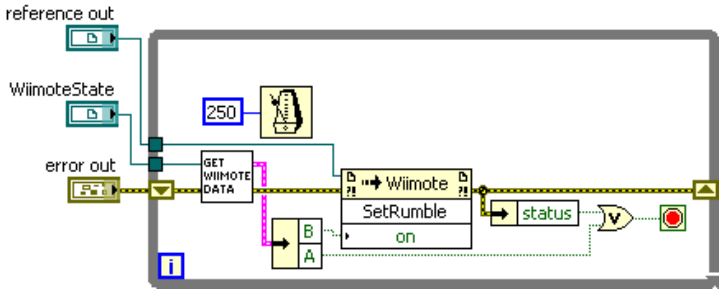
SavedData.vi



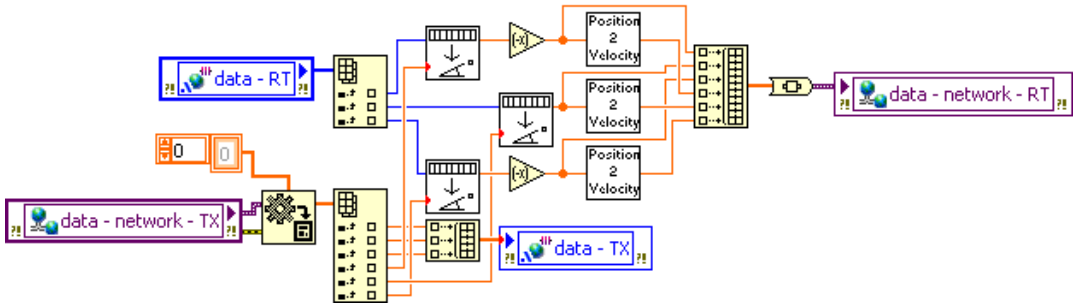
Main code in CompactRIO for Open Loop.



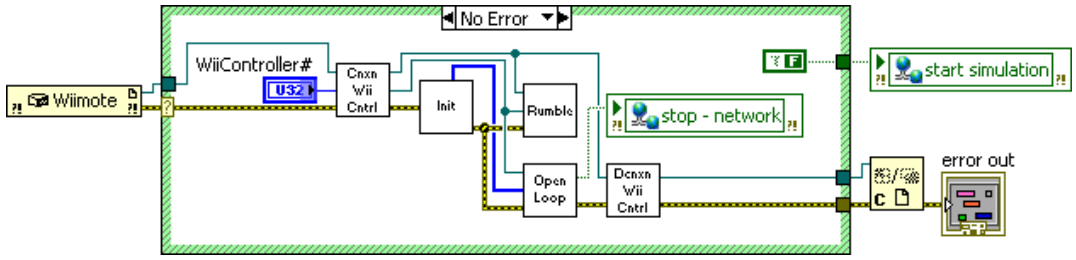
Rumble.vi



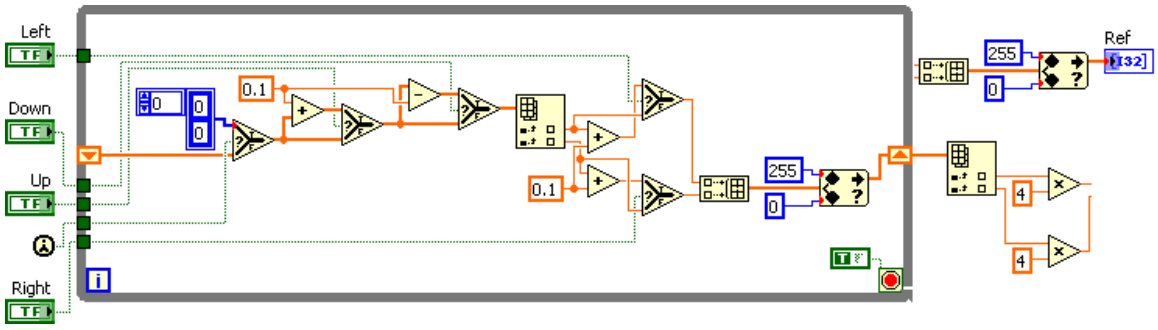
ReadData.vi



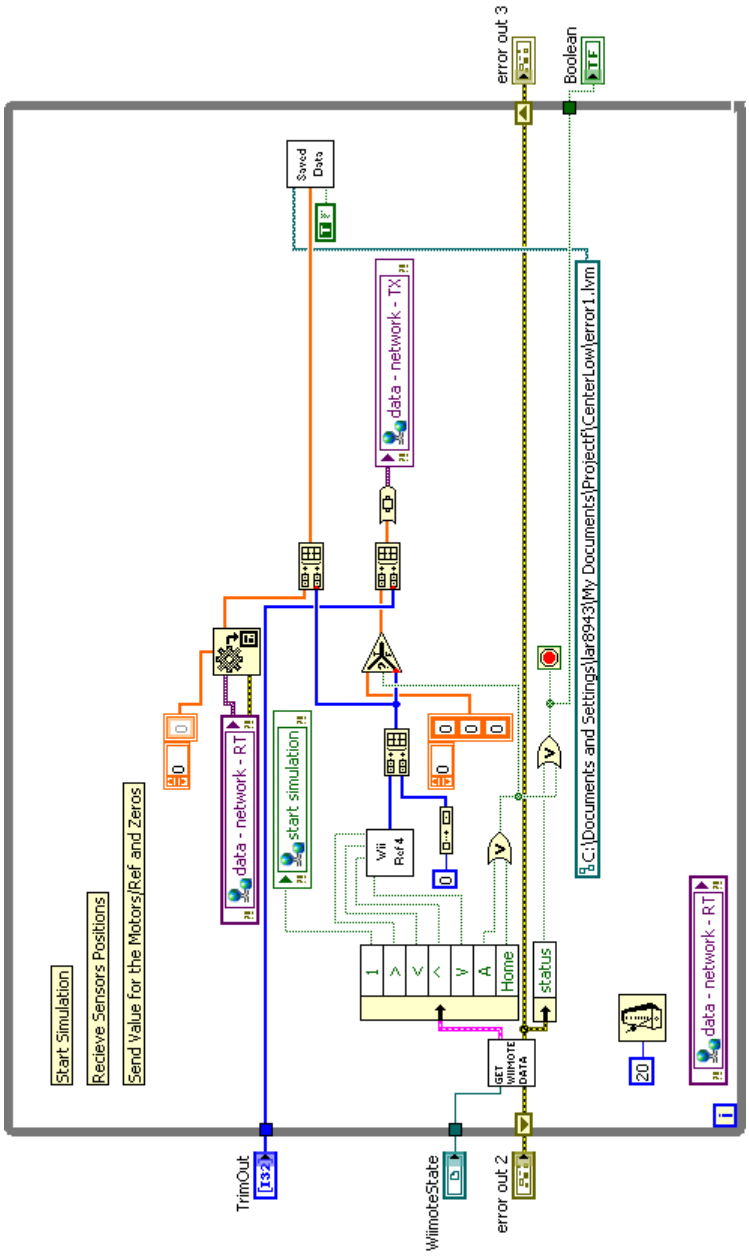
Main code for Open Loop user interface.



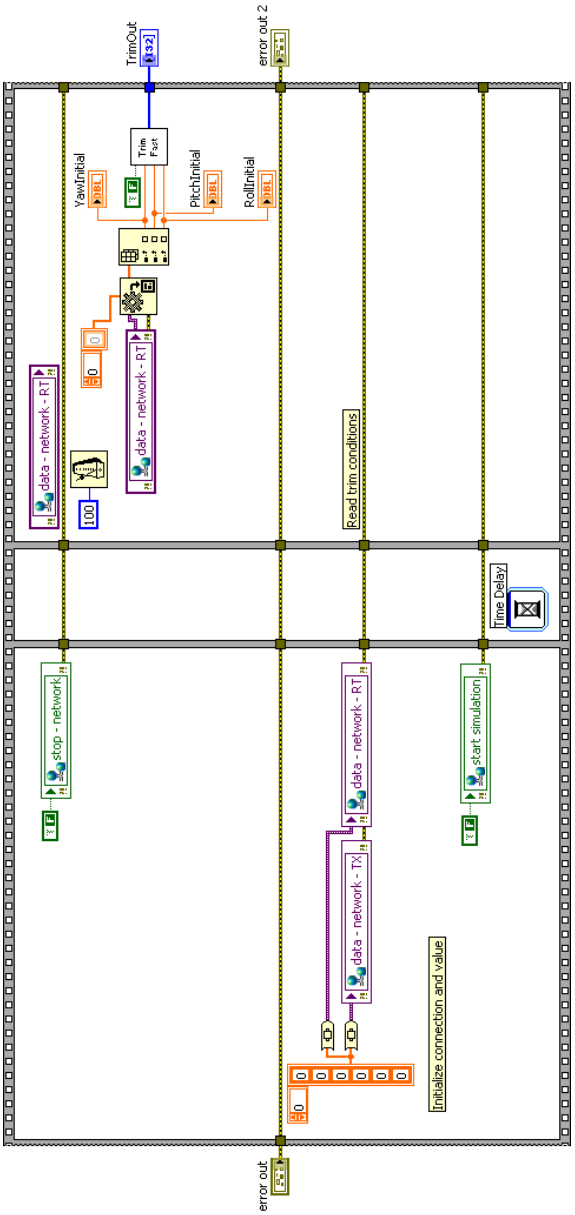
WiiRef4.vi



OpenLoop.vi


error out 3
Boolean
Saved Data
data - network - TX
C:\Documents and Settings\lar8943\My Documents\Project1\CenterLow\error1.lvm
data - network - RT
start simulation
Wii Ref 4
1
^
v
<
>
A
Home
status
GET WIIMOTE DATA
20
data - network - RT
i
Start Simulation
Recieve Sensors Positions
Send Value for the Motors/Ref and Zeros
TrimOut
WiimoteState
error out 2

Init.vi


TrimOut
I32
error out 2
Trim Feat
YawInitial
PitchInitial
RollInitial
DBL
data - network - RT
100
Read trim conditions
Time Delay
stop - network
data - network - RT
data - network - TX
start simulation
Initialize connection and value
error out

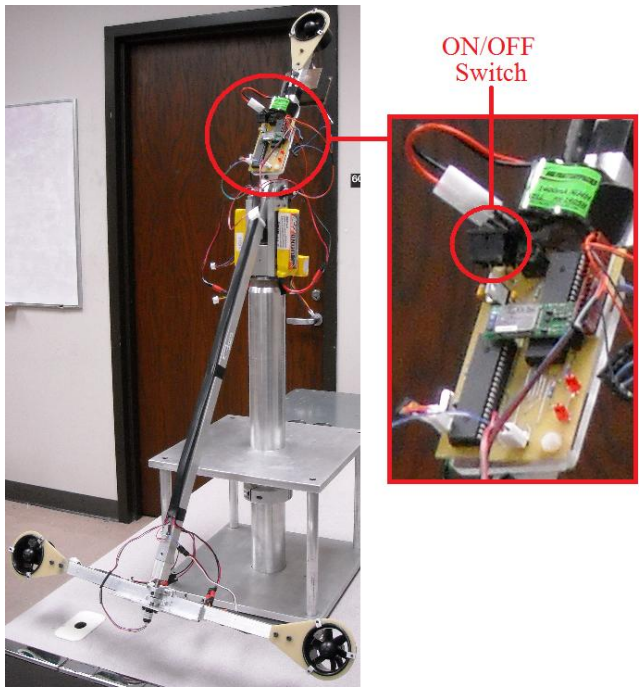
APPENDIX E

MANUAL

THE BIROTOR HELICOPTER IS NOT A TOY.

Before using the birotor helicopter be sure to read this instructions. Even if children are allowed to use it (exposition, scientific fairs, etc.), they never be left alone with the helicopter. Ignoring the instructions in this manual can result in a serious injury.

The helicopter can be stoped at any moment, by turning off the ON switch. If something happens, grab the helicopter and turn it off.



ON/OFF switch location.

1. Be sure that the batteries are fully charged.

   Before tunrning on the helicopter make sure the batteries are fully charged. One charger is needed for the Li-Po batteries and one for the Ni-MH batteries. If the batteries are not fully charged, the performance of the helicopter will not be good, and there is risk of an accident. Explosions can also occur if the battery is short-circuited or if the cell or pack is punctured.

Li-Po battery charger.

Ni-MH battery charger.

2. Carrefully connect all the devices, power supply and batteries, DO NOT TURN ON YET.

   Make sure everything is connected: serial cables, batteries, PC, CompactRIO and electronic boards.

3. Turn on the CompactRIO controller and the bluetooth board attached to it.

   Turn on the CompactRIO, so the PC can recognize it.

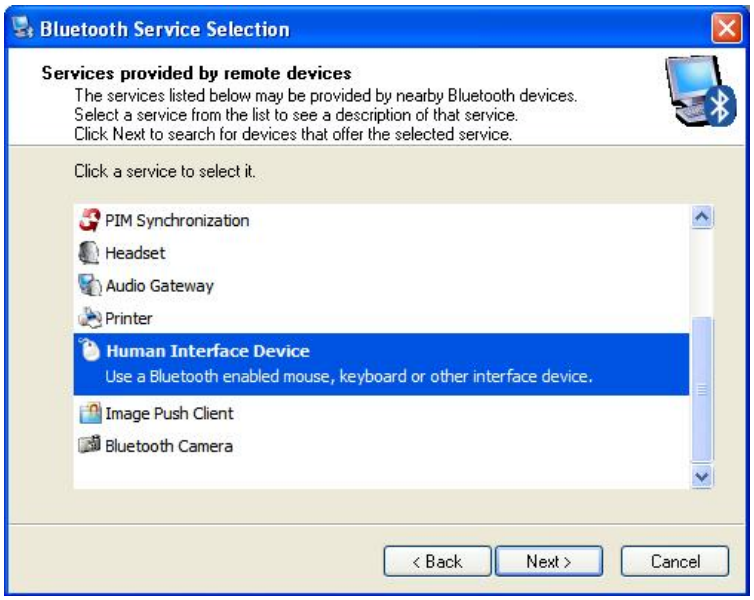
4. Make the connection between the PC and the Wii controller.

   If the Wii controller is not connected before running the VIs, an error message will be displayed and the VIs will not run, you may need to use a blutooth dongle, or a blutooth device on the host PC.

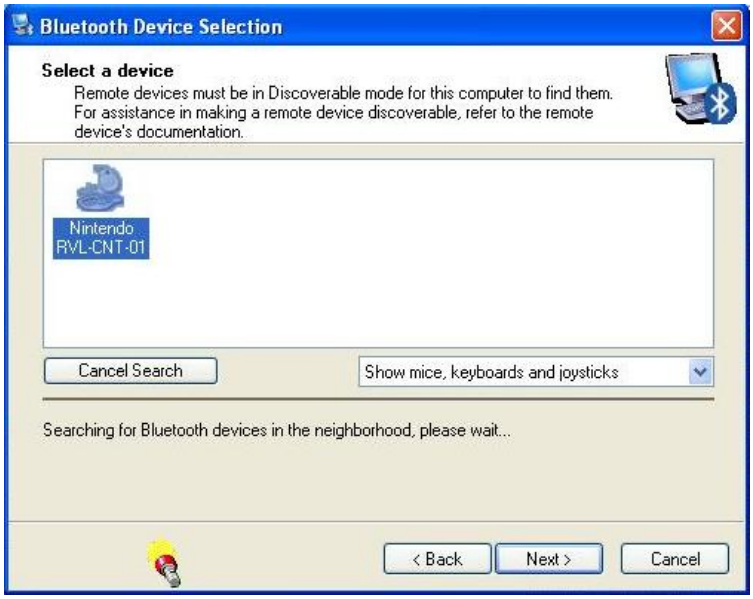
   -> Right click on the bluetooth icon -> Start the Bluetooth Device

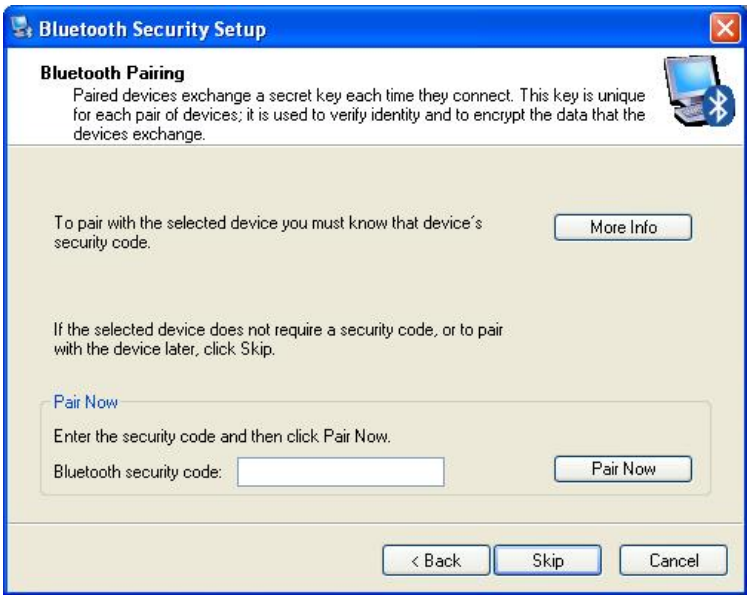
   -> Right click on the bluetooth icon -> Bluetooth Setup Wizard

-> I know the service... -> Next -> Human Interface Device -> Next



-> Nintendo RVL-CNT-01 -> Next -> Skip

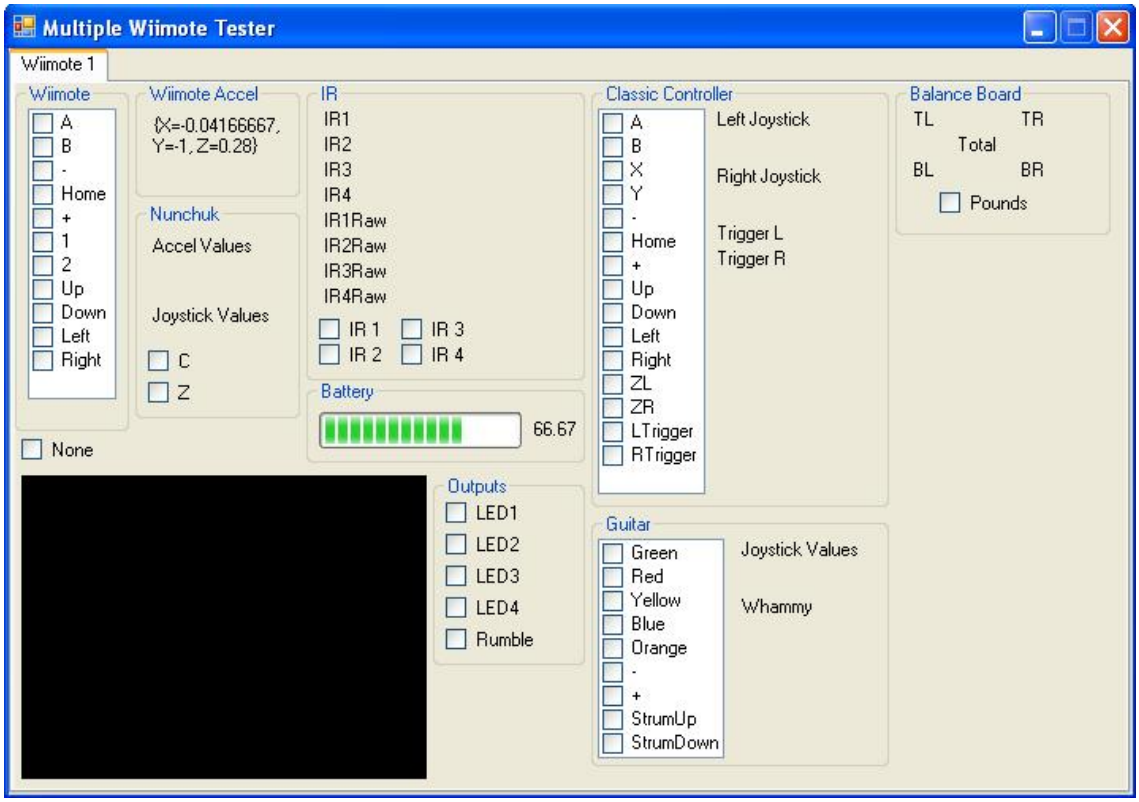




-> Finish

After these steps the next application must be used:

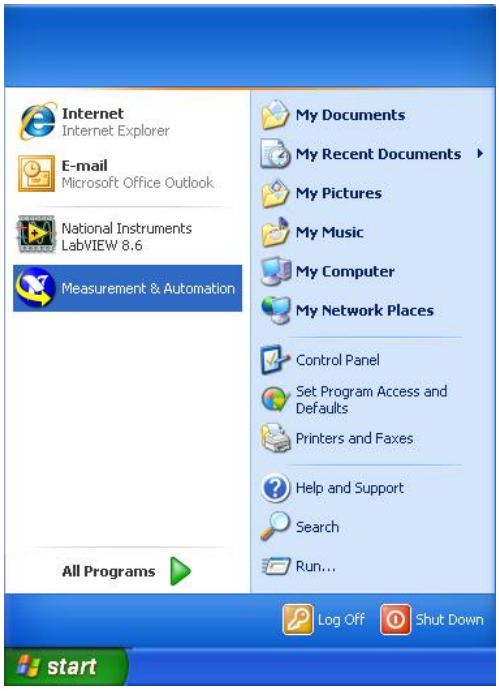
-> Close


Multiple Wiimote Tester
Wiimote 1
Wiimote
A
B
-
Home
+
1
2
Up
Down
Left
Right
None
Wiimote Accel
{X=-0.04166667, Y=-1, Z=0.28}
Nunchuk
Accel Values
Joystick Values
C
Z
IR
IR1
IR2
IR3
IR4
IR1Raw
IR2Raw
IR3Raw
IR4Raw
IR 1
IR 2
IR 3
IR 4
Battery
66.67
Outputs
LED1
LED2
LED3
LED4
Rumble
Classic Controller
A
B
X
Y
-
Home
+
Up
Down
Left
Right
ZL
ZR
LTrigger
RTrigger
Left Joystick
Right Joystick
Trigger L
Trigger R
Guitar
Green
Red
Yellow
Blue
Orange
-
+
StrumUp
StrumDown
Joystick Values
Whammy
Balance Board
TL
TR
Total
BL
BR
Pounds

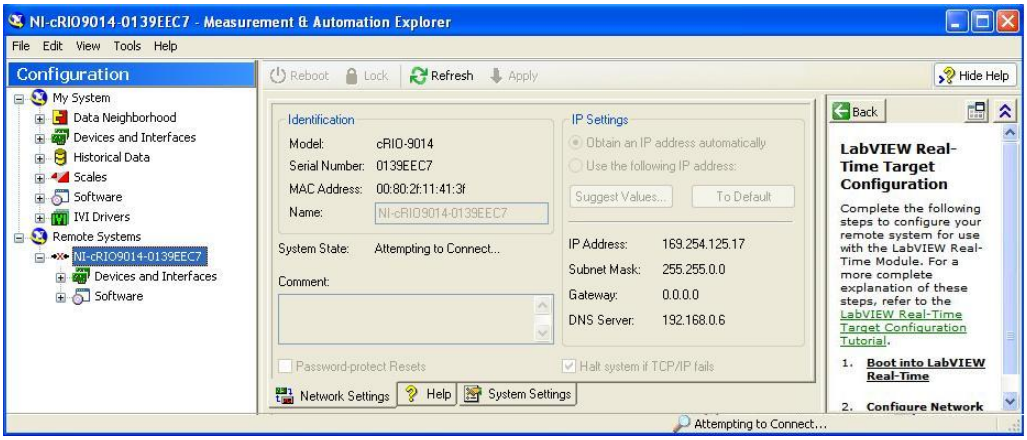
5. Make the connection between the PC and CompactRIO.

   The PC needs to know that the CompactRIO is connected to it.

   -> Start -> Measurement & Automation



   -> Remote Systems -> NI-cRIO9014



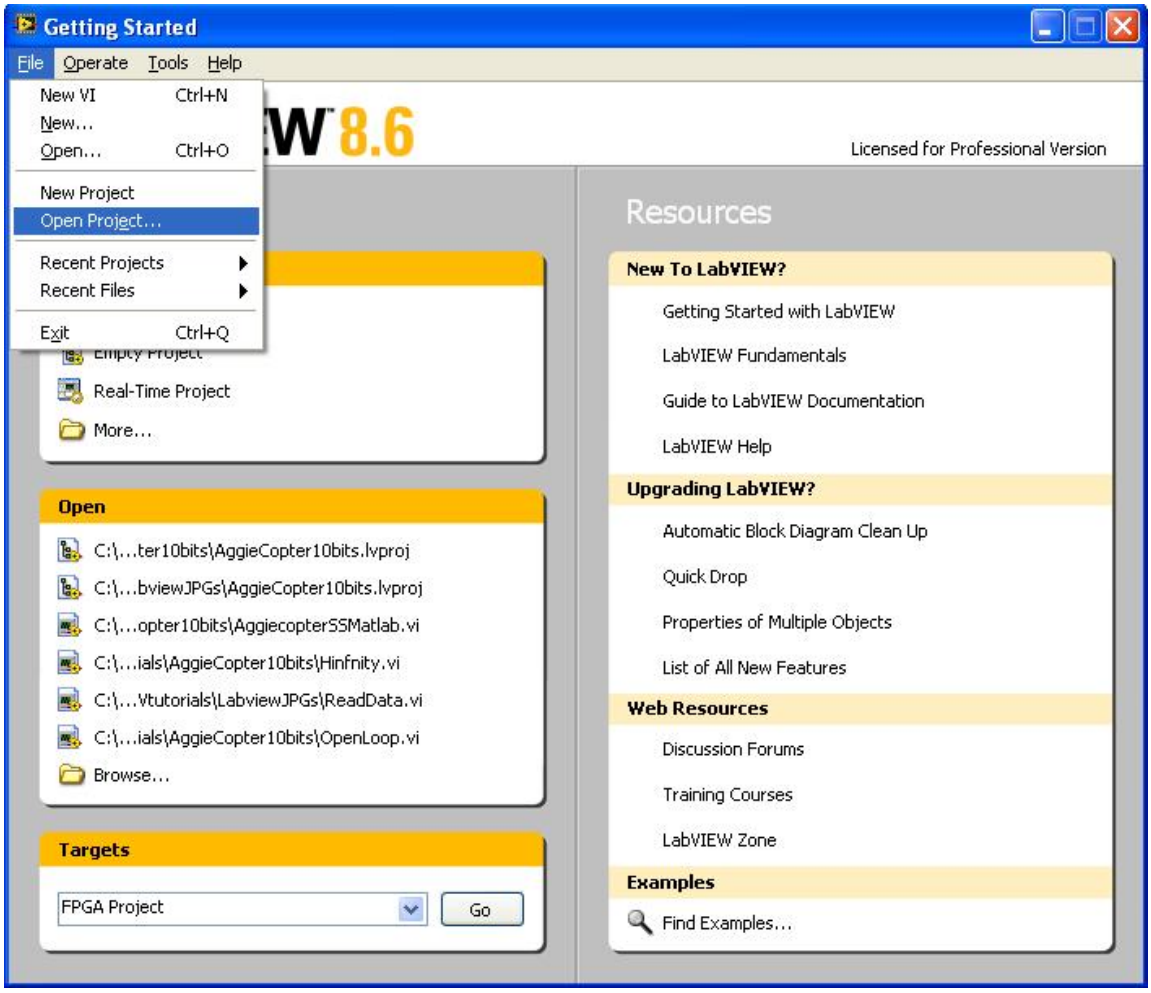
6. Open LabView and VIs.

   Open the project VI and chose all the VIs needed for a simulation based on the next chart:

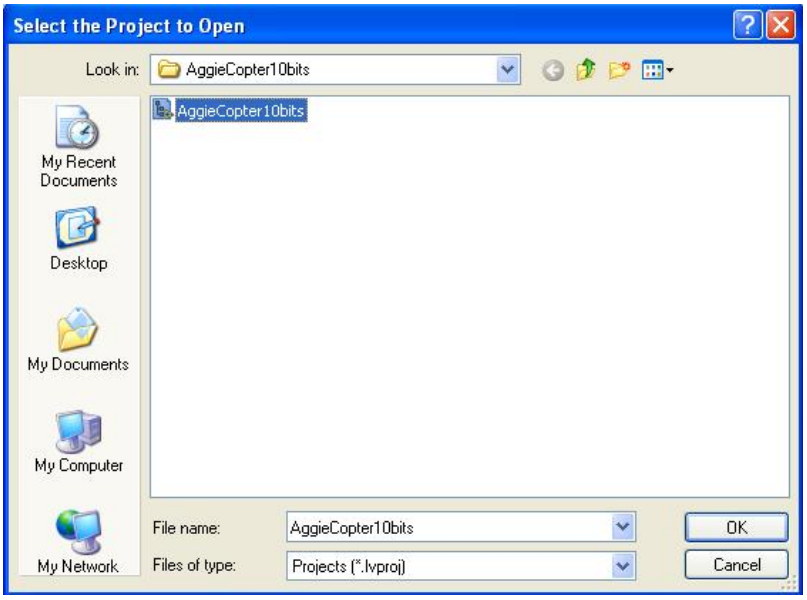
Open Loop	Close Loop
OpenLoop.vi	Hinfinity.vi
WiiPC.vi	WiiPCref.vi

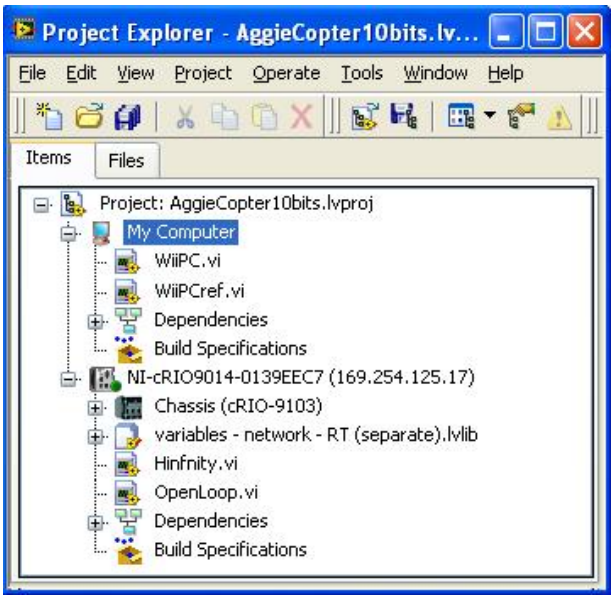
-> Start -> National Instrument LabVIEW 8.6

-> File -> Open Project...



-> AggieCopter10bits -> OK





7. Change any parameter you need on VIs.

   Change any parameter needed for the simulation (name of saved files, trim conditions, etc.).

8. Turn on helicopter.

9. CHECK NOTHING GOES WRONG.

   Turn the helicopter off and anything else if something weird happens.

10. Run the VI on the CompactRIO (controller).

11. CHECK NOTHING GOES WRONG.

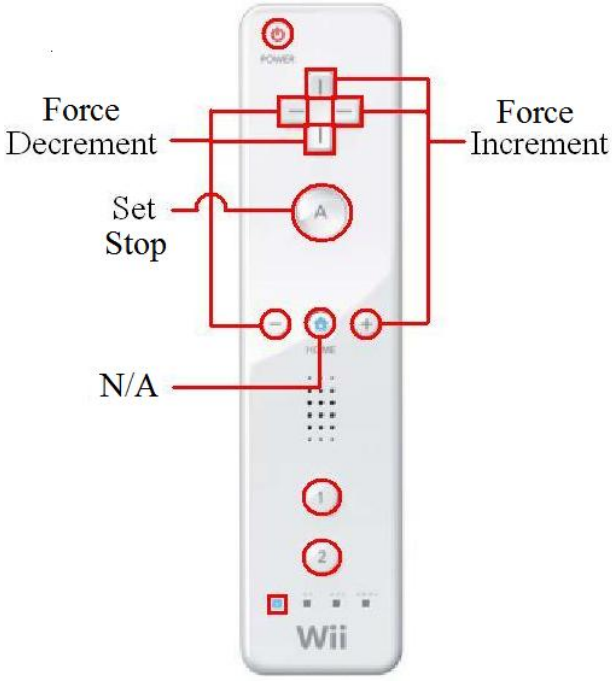
    The sensors are absolute encoders, thus, they need to be initialized before starting the program. This is the reason why the VIs on the CompactRIO have to be run first.

12. Run the VI on the PC (user interface).

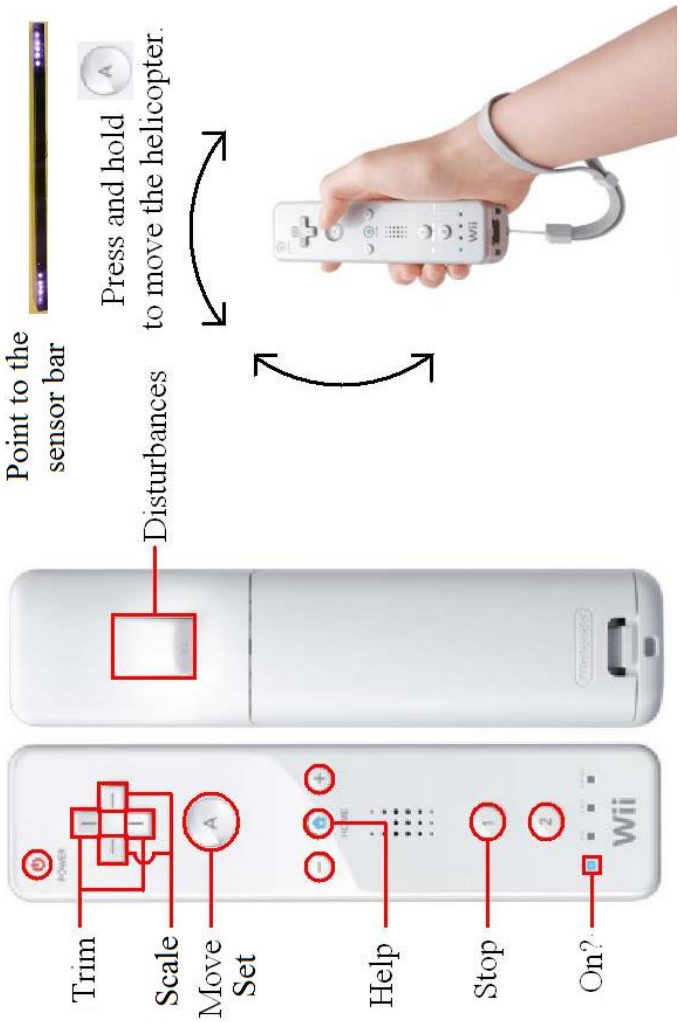
13. CHECK NOTHING GOES WRONG.

    Once you run the VI on the PC make sure everything is OK. Then the helicopter is ready to use.

14. Open Loop Controls



15. Close Loop Controls



16. When finish, turn off everything and disconnect everything. NEVER LEAVE THE BATTERIES CONNECTED TO THE HELICOPTER.

APPENDIX F

NOMENCLATURE

3DOF	3 degrees of freedom
ARE	Algebraic Ricatti Equation
CAD	Computer Aided Design
COM	Center of Gravity
diag(x)	matrix whose diagonal elements are the inputs of vector x
eye(x)	identity matrix of size x
LiPo	Lithium Polymer
LMI	Linear Matrix Inequalities
LQR	Linear Quadratic Regulator
MIMO	Multi Input, Multi Output
Ni-MH	nickel-metal hydride cell
PIC	Programmable
PWM	Pulse Width Modulator
SEC	Speed Electronic Controller
SISO	Single Input, Single Output
SVD	Singular Value Decomposition
Tr(x)	Trace of matrix x

VITA

Luis Arturo Ruiz Brito received his Bachelor of Science degree in Mechatronics Engineering from Instituto Politécnico Nacional at Mexico City in 2005. His research interests include control alghoritms and real time applications. Mr. Ruiz may be reached at Texas A&M University, Department of Aerospace Engineering, H.R. Bright Building, Rm. 603, Ross Street - TAMU 3141 College Station TX 77843-3141. His email is britonet@neo.tamu.edu.