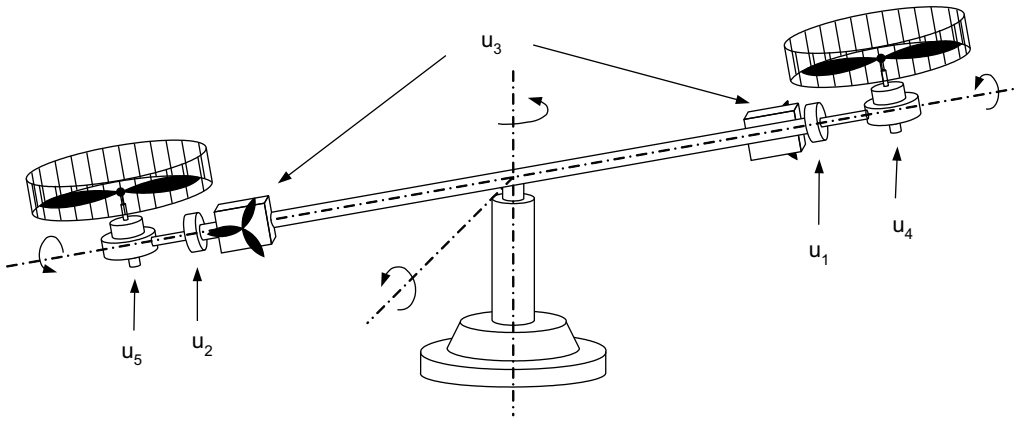
Control Reconfiguration After Actuator And Sensor Faults At The Flight Model

Daniel Rowe Serrano

1st April 2003



Contents

List of Figures

Abstract

Objectives. Any technical system is liable to the occurrence of faults, where typically a fault in a single component affects the whole system. The aim of the project is to implement and verify experimentally a new control structure which will be able to reconfigure the system in response to the occurrence of a severe fault such as breakage of actuators or loss of sensors.

Method. Control reconfiguration is a change of the control structure in response to a detected fault process, so that the resulting control loop is stabilized and reaches acceptable closed loop performance. Thus, the system is kept in operation. The solution tested for the actuator faults is the redesign of the control law which have been calculated using Linear Quadratic Gaussian (LQG) techniques. The nominal weight matrices can be used to obtain a new controller for the faulty situation. Sensor faults are solve by designing a bank of observers which has a nominal observer and as many fault case observers as faults are considered. This is called Dedicated Observer Scheme. The fault case observers do not depend on the faulty output.

Results. It has been demonstrated that the redesign of the state feedback controller and the use of a Dedicated Observer Scheme is a successful way to reconfigure the control loop, keeping the system controllable and with acceptable performances. Moreover, no predesign controllers and no manual intervention is necessary. The changes in the control loop are kept to minimum.

1 Introduction

This study has been carried out in the Institute for Automation and Computer Control of the Ruhr-Universität in Bochum, Germany. It has been supervised by the assistant teacher Ing. Thomas Steffen and the professor Dr. Ing. Jan Lunze.

The Institute has developed different approaches to the reconfiguration control problem including this project and other studies about Hybrid Reconfigurable Control [2]. Several experimental applications have been implemented to verify the theory. This applications include the Flight Model and a Three-Tank System, both with redundant hardware.

1.1 Problem Description

Any technical system is liable to the occurrence of faults, and a fault in a single component may affects the operation of the whole system. Severe faults such as the complete loss of actuators or sensors break the nominal control loop. To keep the system operational, it is necessary to change the control algorithm in response to the fault. This is called reconfiguration.

1.2 Present-Day Approaches

There are mainly three approaches to control reconfiguration. The first one is pointed to systems where there are no structural changes. The considered faults include degraded actuator performance or increased sensor noise. Adaptive methods have been used to adjust the controller parameters.

A second approach solves the reconfiguration problem by a redesign of the controller. Usually, a set of dedicated controllers, one for every fault case, are manually developed and can be automatically selected in every specific fault case using an appropriate supervisor logic.

The third option is based on the use of a reconfiguration block which works as an interface between the plant and the controller, allowing to use the nominal controller at any case.

1.3 Project Aim

The aim of the project is to implement and verify experimentally a new control structure which will be able to reconfigure the system in response to the occurrence of a severe fault. The control structure is changed when a fault happened. The reconfiguration goal is to modify the control loop so that the resulting one is stabilized and reaches acceptable closed loop performance keeping it operational.

The reconfiguration task is similar to a compensator design for the faulty process but there are also two important requirements. Firstly, the reconfiguration has to be carried out completely automatically while the system is in operation. Secondly, a goal of the reconfiguration is to minimize the changes to the control structure.

In this case, the first reconfiguration approach is not valid due to this study considers severe faults with imply structural changes. The general idea of the second approach is followed. However no predesigned controllers are used and they are automatically designed on-line. On the other hand, the Institute is already working on the third approach in order to try to minimize the control structure changes and compare the different reconfiguration approach performances.

1.4 Plant Overview

The plant selected for experimental verification is a flight model with two degrees of freedom (shown in figure 1). This system is chosen because it provides the necessary redundancies for successful reconfiguration. First of all, it has several actuators which control two main rotors, which can change their attack angle and the rotor speed, and another two lateral rotors that work together at the desirable speed. This mean that it is not necessary to use all actuators to control the system. Secondly, it has four position sensors to measure the attack angle and the system pitch and orientation angle and not all of them are necessary to observe the state of the plant.

The actuator faults considered are the blockage of a servo motor that control the main rotor attack angle and the blockage of the lateral rotors. Due to the redundancies, it should be possible to control the system after loosing one of these actuators. However, the breakage of one of the main rotors make

Figure 1: Plant

the system uncontrollable. The loss of sensor signals is also considered. In this case, the reconfiguration task is to observe the value of the lost sensor. It should be possible to observe correctly one of the two attack angles, but other sensors faults lead to an unobservable system.

1.5 Assumptions

- There is a linear model for the nominal process. It is given in state space form:

$$\begin{aligned} \dot{\mathbf{x}}_N &= \mathbf{A}_N\mathbf{x}_N + \mathbf{B}_N\mathbf{u}_N + \mathbf{B}_v\mathbf{v} \\ \mathbf{y}_N &= \mathbf{C}_N\mathbf{x}_N + \mathbf{D}_N\mathbf{u}_N + \mathbf{D}_w\mathbf{w} \\ \mathbf{x}_N(0) &= \mathbf{x}_0 \end{aligned}$$

where $\mathbf{u}, \mathbf{x}, \mathbf{y}$ are the system inputs, states and outputs; $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ the system matrices; $\mathbf{v}, \mathbf{w}$ uncorrelated, normalized white noise; $\mathbf{B}_v$,$\mathbf{D}_w$ the disturbance weighting matrices.

- A nominal controller exists for the nominal plant. The resulting control loop consisting of the nominal plant and the nominal controller is assumed to be stable. Furthermore, it satisfies all requirements concerning disturbance rejection and input tracking.
- The actuator fault detection is already solved by using a *fault detection and isolation* module (FDI).
- A linear model of the faulty process is known. It is also given in state space form and linearized around the same equilibrium as the nominal process:

$$\begin{aligned} \dot{\mathbf{x}}_F &= \mathbf{A}_F\mathbf{x}_F + \mathbf{B}_F\mathbf{u}_F + \mathbf{B}_v\mathbf{v} \\ \mathbf{y}_F &= \mathbf{C}_F\mathbf{x}_F + \mathbf{D}_F\mathbf{u}_F + \mathbf{D}_w\mathbf{w} \\ \mathbf{x}_F(0) &= \mathbf{x}_0 \end{aligned}$$

  where the index F denotes the faulty situation.
- Though some of the actuators or sensor may have lost their function, the number of inputs, outputs and states has not changed.
- It is assumed that the faulty process is still controllable and observable. It follows that a stabilizing controller exists.

1.6 Way of Solution

The nominal control loop consists of a state feedback proportional controller and a state observer. This structure will be called compensator. It is defined by the equations:

$$\begin{aligned} \mathbf{u} &= \mathbf{K}(\mathbf{x}_{ref} - \hat{\mathbf{x}}) \\ \dot{\hat{\mathbf{x}}} &= \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}(\mathbf{y} - \mathbf{C}\hat{\mathbf{x}}) \end{aligned}$$

The control structure is shown in the figure 2.

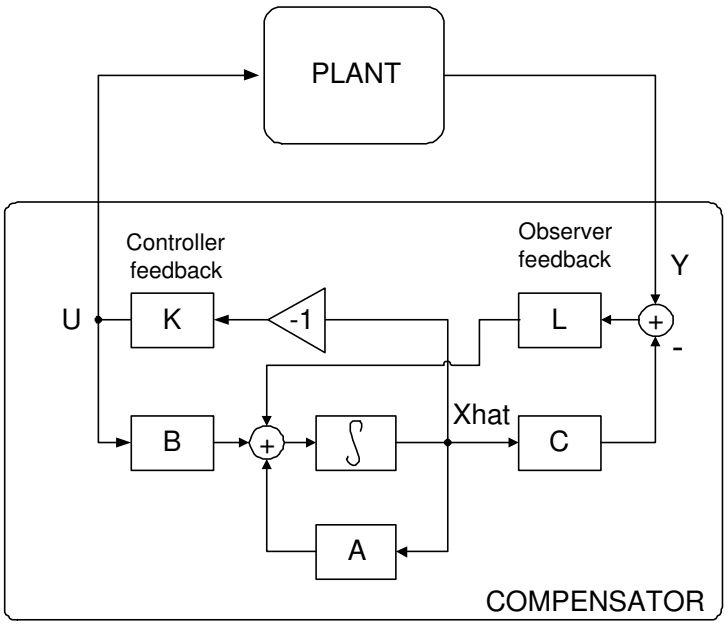


Figure 2: Control loop

The occurrence of a fault implies that it is necessary to use a different set of inputs or outputs for the control task. Thus, a new control configuration must be selected and new controller parameters have to be found.

Once an actuator fault is detected a new state feedback controller is designed using the same design approach as for the nominal controller. The controller is designed using Linear Quadratic Gaussian (LQG) techniques, a stochastic generalization of the Linear Quadratic Regulator [3]. This approach finds a feedback state control law that minimize a quadratic cost index depending on the system states and input signals. Weight matrices are assigned to each variable. Using this approach, the nominal controller is designed. In case of an actuator faults the same cost index can be used to calculate the gain matrix for the faulty situation. Then, the controller is down-load online into the real time system.

In order to treat sensor faults a bank of observers will be used. It includes a nominal observer and one more observer for every fault case. The observer corresponding to a specific fault case does not rely on the sensor affected by this fault case, so the fault does not affect its state observation. This

structure is called Dedicated Observer Scheme. It assumes that it is possible to observe the variable of the faulty sensor using the remaining sensors and the inputs. The fault will be detected by evaluating the difference between the real outputs and the observed ones of each observer. This difference is called innovation. When the Nominal Observer innovation is higher than a threshold, a selection logic will detect the fault and select the Fault Case Observer which has the lower innovation. All observers are designed using the LQG method. Instead of cost functions, assumptions on the variance of state and output error are given. Otherwise the observer design is a dual problem to the state feedback controller design.

The reconfiguration structure used, including both actuator and sensor reconfiguration loops, is shown in the figure 3.

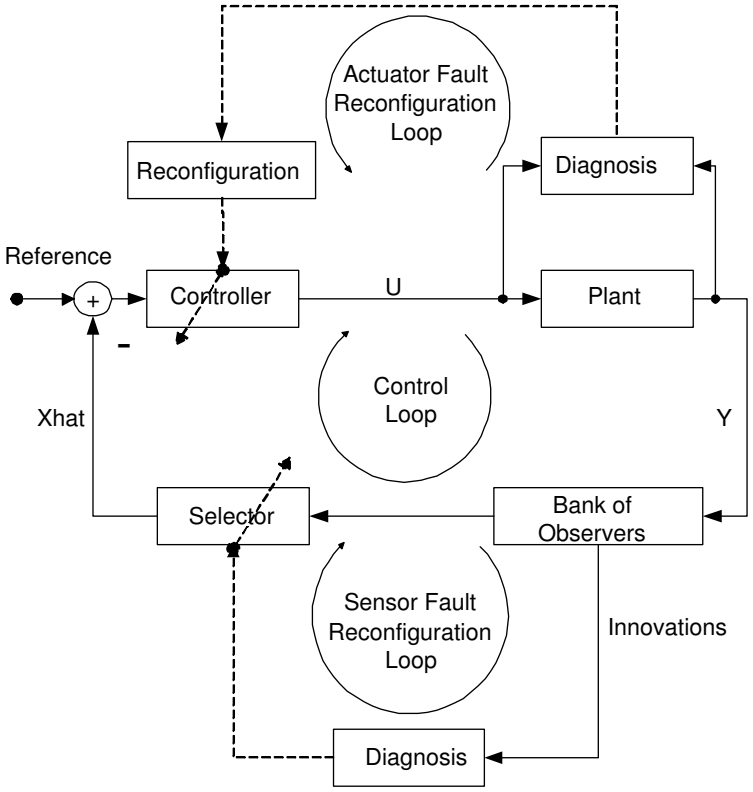


Figure 3: Reconfiguration structure

Due to this approach it is not necessary to predesign controllers and no man-

ual intervention is required for the reconfiguration itself. The system detects the sensor fault and performs the reconfiguration automatically, thus restoring a stable control loop. Of course a *fault detection and isolation* module (FDI) is needed to detect the actuator faults, which is beyond the scope of this document. The quadratic optimization design ensures a reasonable performance under all fault cases. Finally, the changes to control structure are restricted to the part affected by the fault: a selection of the best observer or a recalculation of the control gain matrix using the same weights.

1.7 Design Process

In order to allow the system reconfiguration, it is necessary to perform a previous study. In a first step a detailed system model is constructed. Then several experiments are carried out and in order to identify the model parameters. Once the complete model is implemented, it is linearized and the state space model is obtained. Finally a nominal control loop is designed.

The reconfiguration task is carried out in three steps. Firstly, the controller is redesigned so that it can handle actuator fault cases. Secondly, the bank of observers is designed. Finally, the sensor fault detection logic is implemented.

Both nominal control loop and reconfigured control loop are firstly developed simulating their behavior using Simulink and then are tested on the plant.

1.8 Document Outline

This paper is structured following the logical order given by the design process. Section 2 describes both flight model and real time system in detail. A plant sketch is presented explaining the system degrees of freedom and its actuators and sensors. The hardware implementation of the real time system and the software interface are developed here.

Section 3 depicts the system model, how the experiments which identify the system parameters are carried out and the subsequent data processing. A linear model based on a state-space description is built.

The nominal control loop is implemented in section 4. Both controller and observer are explained in detail and several system simulations are accomplished.

Sections 5 and 6 develop the reconfiguration techniques for both actuator and sensor faults. The problems caused by the occurrence of these severe faults are explained and a way of solution is proposed. Practical problems as the rank deficiency are identified and solved. Finally, several simulations are carried out and results and conclusions are expounded.

Multiple faults simulations are shown in section 7.

Section 8 explains the tests carried out on the plant. The necessary interface to do them is shown and described. Practical problems relative to non modeled noise and non linearities are detected and solved. Results and conclusions are exposed.

A brief summary of the whole project is related in section 9.

Finally, the symbol list description is in the appendix A; the programs codes are placed in appendix B; the remote control used to accomplished the experiments is explained in appendix C; other useful graphics and tables are shown in appendices D and E.

The document concludes with several references to relevant literature.

2 System Description

2.1 Introduction

In order to verify experimentally the reconfiguration techniques, a plant which represents an helicopter was chosen. This system was selected due to it has the necessary redundancies to allow the reconfiguration task. Thus, despite the loss of an actuator or a sensor, the system remains controllable and observable, which is essential to perform the reconfiguration task. A photograph of the plant is shown in the figure 4.

Figure 4: Plant

2.2 Plant Overview

The flight model—shown in the figure 5—has two degrees of freedom given by two axes such as the first one let the system reach different pitch angles and the second one can rotate to reach every position in a flat surface parallel to the floor. The pitch angle is called α and its range is between -45 to 45 degrees. The angle which identifies the rotation is called β. It is considered a range between -180 to 180 degrees, although the system is constructed to turn five turns. There is a rotor at each end of the first axis. These rotors modify their attack angle turning around the axis in a range between -90 and 90 degrees. These attack angles are called γ_1 and γ_2. There are also two smaller lateral rotors in the axis which provides the system with more redundancy.

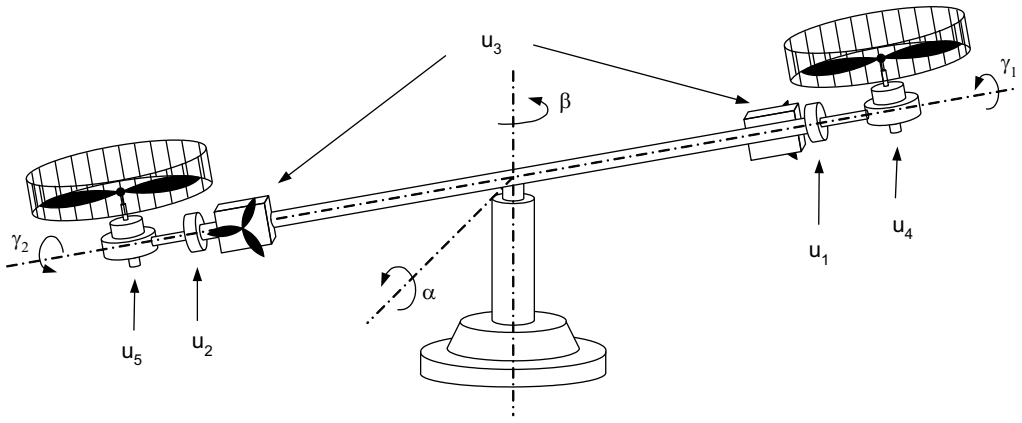


Figure 5: Plant sketch

The system inputs are the voltage applied to each motor: both servo motors which control the attack angle (u_1 and u_2), the lateral rotors (u_3) and the main rotors (u_4 and u_5). The system sensors are the four potentiometers which read the angular position of γ_1 and γ_2, β and α.

The pitch angle α is already controlled with a proportional controller which stabilize the system around α equal to zero using the main rotors. Therefore, inputs u_4 and u_5 are considered constant in this study since the pitch angle α control loop is stable. There are two parameters in this amplifier in order to set the gain and the operating point (value of α and force applied to

the motors). It is also possible to modify this angle using a joystick that is already connected to the amplifier controlling it manually.

2.3 Orientation

The potentiometers wires are connected such that the β rotation angle increase from -180° to 180° when it rotates in the counter-clock wise. The γ rotor attack angle increase their value from -90° (at fifteen minutes) to 90° (at forty five minutes), being zero degrees when the propeller is horizontal in the upside.

The motors wires are connected such that a positive u_1 or u_2 input decrease the gammas angle. A positive u_3 propels the helicopter towards a negative β angle.

Due to the propeller construction, when a positive input to the attack angle rotors is applied, although the γ angle move towards negative values, the rotation angle, β, moves to positive ones.

2.4 Real Time System

2.4.1 Introduction

The plant is digitally controlled. The control configuration is implemented using system modeling software. Thus, in order to convert the digital control signals into analog ones and acquire the data from the plant sensors a real-time system is needed. A hardware configuration and a software interface were implemented to carry out this task.

2.4.2 Hardware Implementation

The chosen solution is known as a host-target combination. The *Math Works* provided a software solution called xPC target [5]. This configuration needs two computers. The first one, the host PC, is used to create models, implement the controller code and run simulations in non real-time. The second one, the target PC, runs the xPC Target real-time kernel which control the plant in real-time. The connections layout is shown in the figure 6

The host PC software requirements include an operating system, *Matlab* (which provide a command line interface and it is used to develop the controller code), *Simulink* (to create the interface and controller models), *Real-Time Workshop* (which converts the *Simulink* model into C code), a *C* language compiler (which creates the executable code) and *xPC Target* (which down loads the code into the Target PC*).*

The target PC don't need even an operating system since it uses a boot disk, created by xPC Target, which loads and runs the xPC Target kernel. This kernel runs the down loaded code and converts the computer screen into an interface between the operator and the plant. That means that there is a scope which trace the signals and some other windows with information about the system state: loading parameters, waiting, etc.

The real time card is connected to the board where the wires to the different actuator amplifiers and the system sensors are connected. The figure 6 shows the layout connections between the plant and the real time processor.

Since there are three plant inputs and the board provided by National Instrument (PCI 6025) has only two outputs (although it has 16 inputs) an additional interface card is needed. The Meilhaus ME-30 provides four analog outputs and three of them are used.

2.4.3 Real-Time Interface

A software interface must be developed to configure the real time processor ports which allow the communication between the plant and the control structure.

These ports connect the analog-digital (A/D) and digital-analog (D/A) converters to the signals coming from the sensors and the signals going to the actuators. The port blocks are found in the *Simulink* libraries provided with the real time system. Once the interface is programmed using Simulink, it is compiled into C and linked using an option situated in the *Tools* menu, and then is down loaded into the target PC. Simulink is used to implement a controller and an observer and down-loading them into the Target PC controlling the system.

The sample time has to be chosen here. According to the heuristic criterion of using a sample time faster than 1/20 of the rising time, and since a rising

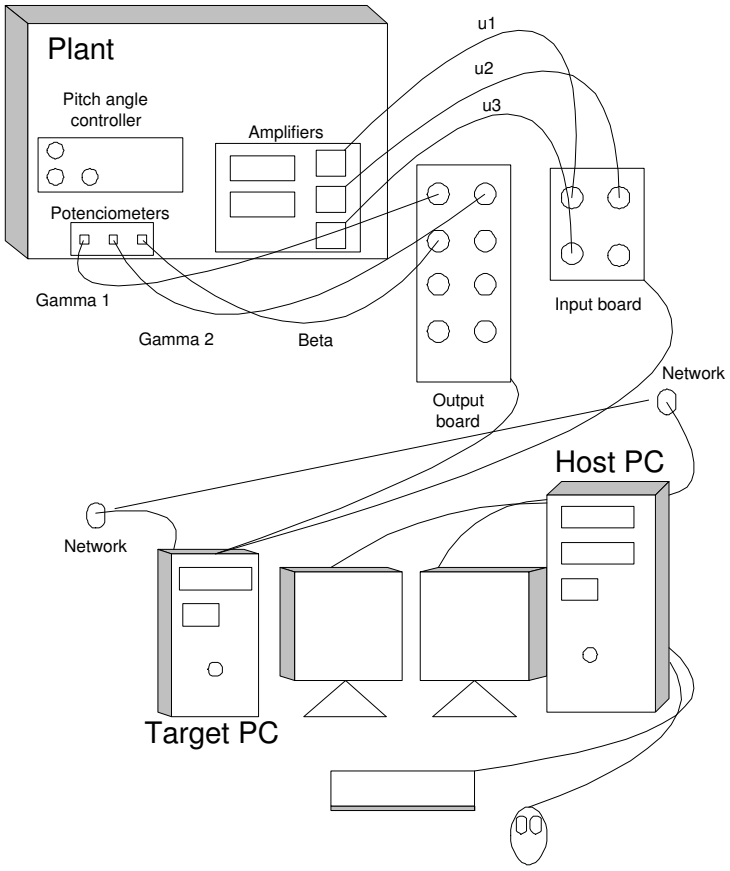


Figure 6: Connections layout

time bigger than 0.2 seconds is expected, a sample time of 0.01 second is chosen. Later, due to the delays introduced while acquiring and processing the data, the sample time has been reduce to 1 millisecond.

The plant is controlled using an input range between -1 to 1. That means three constant blocks set at 1 are selected as inputs and then three slider gain blocks are inserted to modify the values. It is also necessary to connect the ground block to the free input in order to avoid warnings for unconnected ports.

Since the D/A inputs are between -10 to 10 volts, three inputs amplifiers are used to re-scale the range. Besides, since the lateral rotors are not very

powerful, the input u_3 is multiplied by another factor of 2.

The A/D output has a range between -10 and 10 volts. Thus, in order to move this range into a degree scale (-90 to 90 for γ_1 and γ_2, and -180 to 180 for β) another three amplifiers are used. Moreover, a factor of 2.5 is introduced to model the fact that the β sensor allows to measure 5 turns.

Due to the sign criterion used for actuators and sensors the input u_3 and the outputs γ_1 and $\gamma_2$are inverted. The inputs which control the attack angle are not inverted because when a positive input to the attack angle rotors is applied, although the gamma angle move towards negative values, the rotation angle, β, moves to positive ones.

The result outputs are send to workspace so that they can be processed.

The D/A provide two channels with 8-bit resolution (first and second one) and another two ones with 12-bit resolution (third and fourth ones). Since signals u_1 and u_2 are more sensitive, they are connected to the last two ports and u_3 to the first one.

The A/D outputs are connected in the next order: γ_1 to channel one, γ_2 to channel two and β to channel three. The figure 7 shows the interface developed using Simulink.

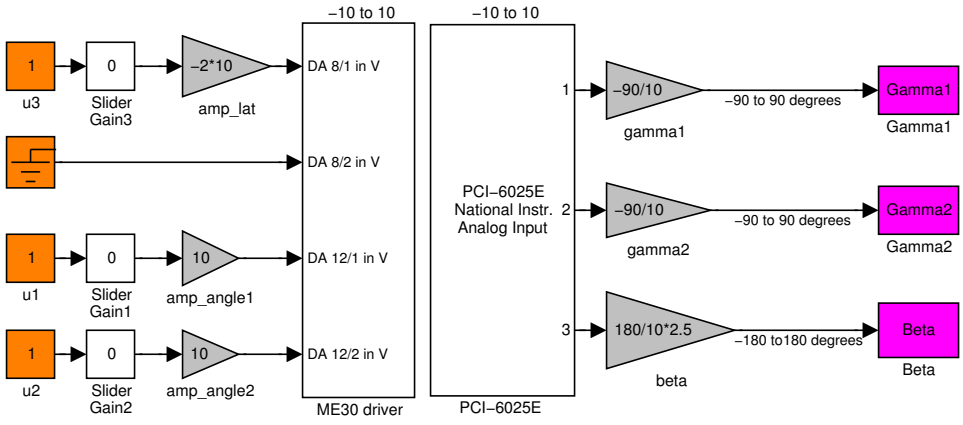


Figure 7: Software interface

3 Modeling

3.1 Introduction

First of all, the plant is modeled as a linear system [1]. This model is implemented using Simulink [4] and stored in a file which is called *flightmodel.mdl* (see figures 8 to 13). The linear model is used to design a nominal controller and observer. Then, a nonlinear model is also implemented and used in order to simulate more realistic experiments.

3.2 Selection of Inputs, Outputs and State-space Variables

The plant is a multiple-input, multiple-output (MIMO) system shown in the figure 8):

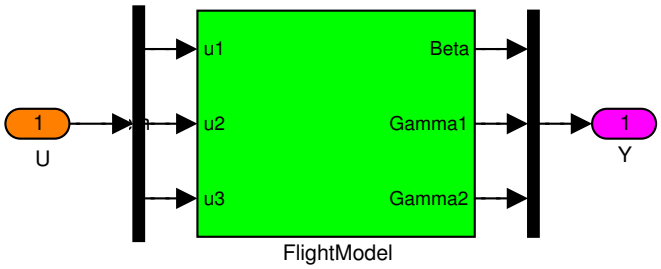


Figure 8: Flight model

The system inputs, $\mathbf{u} = (u_1, u_2, u_3)^T$, are the voltage applied to the different subsystems: *rotor attack angle 1*, *rotor attack angle 2* and *lateral rotors*. The outputs, $\mathbf{y} = (\beta, \gamma_1, \gamma_2)$ are the measured angles, all of them in degrees. Inputs, outputs and subsystems are shown in detail in the next subsection.

The system has been modeled using seven space-state variables which are the three measured angles, their angular speed and the force applied by the lateral rotors. They are ordered as is shown in the state vector $\mathbf{x} = (\dot{\gamma}_1, \dot{\gamma}_2, \gamma_1, \gamma_2, Fl, \dot{\beta}, \beta)^T$. All further considerations apply to this order only.

3.3 Physical Models

The different servo-motors, rotors and the cinematic chain have been modeled as subsystems (see figure 9). Each subsystems are explained thoroughly in the following paragraphs.

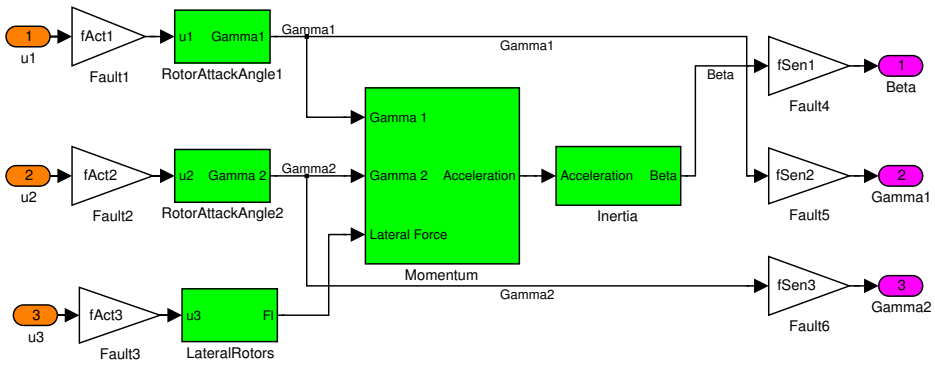


Figure 9: Subsystems

Rotor attack angle. Each servo motor which control the attack angle of each rotor has been modeled as a first order system. Thus, its behavior follows the law given by the equation (1).

$$\tau \dot{y} = ku - y \tag{1}$$

The input is the normalized voltage applied to the servo motor, u_1 or u_2 depends on each servo motor, and the output is the rotor speed. Then, the γ_1 and γ_2 angular position are obtained using an integrator. These angles are the outputs of each subsystem. The whole subsystem is called *Rotor Attack Angle* 1 (or 2) and is a second order system which behavior is described in the equations (2) and (3). Their parameters are the gain (k_1) and the time constant (τ_1) which are supposed to be the same for both subsystems.

$$\tau_1 \ddot{\gamma}_1 = k_1 u_1 - \dot{\gamma}_1 \tag{2}$$
$$\tau_1 \ddot{\gamma}_2 = k_1 u_2 - \dot{\gamma}_2 \tag{3}$$

A dead zone is introduced to account for friction forces in the motor or gear. It is set to zero for the linear model. It will be used later in a more detailed simulation model developed to test the control loop. These subsystems are shown in the figure 10.

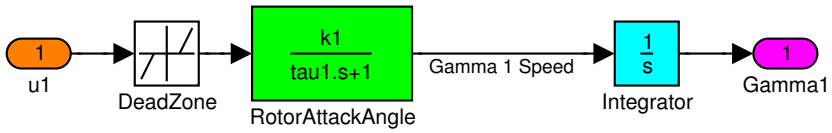


Figure 10: Rotor attack angle

Lateral rotors. The lateral rotors are modeled as a first order system (see equation (1)). The input u_3 is the voltage applied to the pair of motors, normalized to the scale [-1,1]. The output is the force F_l applied to the flight model. The parameters of this subsystem are the gain (k_3) and the time constant (τ_3). The subsystem is called *Lateral Rotors*. It is described by the equation (4). It also has a dead zone to account for friction, which is set to zero here. The figure 11 represent this subsystem.

$$\tau_3 \dot{F}_l = k u_3 - F_l \tag{4}$$

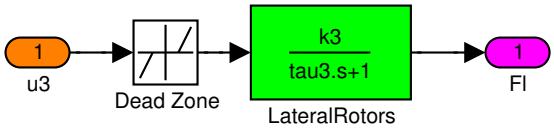


Figure 11: Lateral rotors

Momentum. All the forces modeled above work on the movable part of the flight model, and they create a momentum around the vertical axis. The control aim is to reach a certain angle β around the vertical axis. Therefore this rotational movement has to be carefully modeled.

First of all, the force due to the main rotors is obtained by calculating the sine of both γ_1 and γ_2 angles, multiplied by the forces applied by the main

rotors. These forces are called F_1 and F_2 and they are assumed to be equal and constant. Since these forces are used by a hardware controller to keep the flight model in a horizontal position, this is an approximation.

The momentum generated by the main rotors is added with the momentum generated by the lateral rotors. It is necessary to consider the momentum since the forces are not applied at the same point. The distance to the center differs and so does the momentum. Therefore, these forces must be divided by the constants that content the effects of the system mass and the axis length, called J_m and J_l respectively for the main and lateral rotors. This constant follows the experimental law shown in the equation (5):

$$F = J\ddot{\beta} \tag{5}$$

Then, all momentum components are sum together and the system angular acceleration is calculated. This behavior is described in the equation (6) and modeled in the subsystem *Momentum*, shown in the figure 12.

$$a = \frac{1}{J_m}\left[F_1 \sin(\gamma_1) + F_2 \sin(\gamma_2)\right] + \frac{1}{J_l}F_l \tag{6}$$

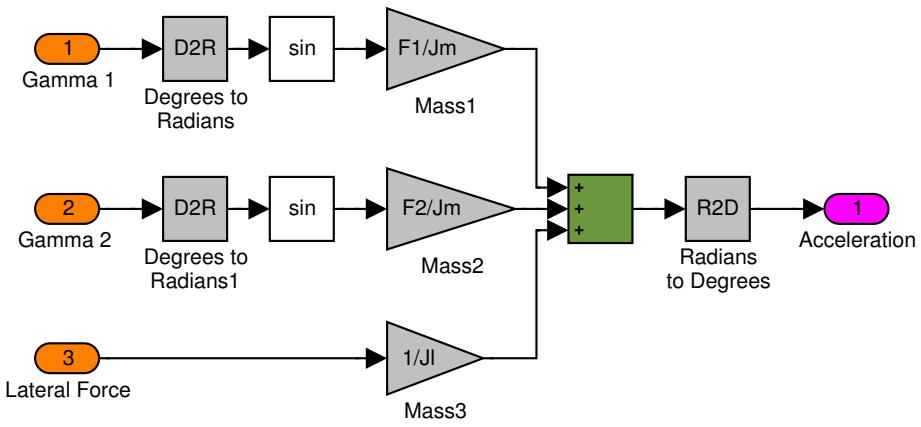


Figure 12: Momentum

Inertia. The momentum is proportional to the angular acceleration, which has to be integrated twice to obtain the angular position β. Speed proportional friction is included in the model to make it more precise. In the autonomous case it is described by:

$$\ddot{\beta} = -F_s\dot{\beta}$$

These phenomena are modeled in the *Inertia block*—shown in the figure 13—as described by this equation:

$$\beta = \int\int \left(a - F_s\dot{\beta}\right) dt^2 \tag{7}$$

The model also contains a dead zone block to account for friction in the bearings, F_b.

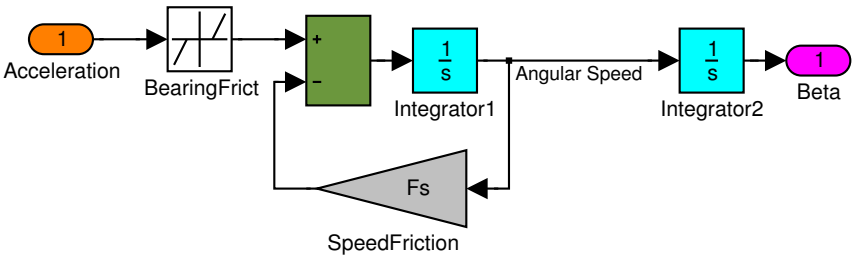


Figure 13: Inertia

Faults. Actuator faults are modeled as amplifiers, inserted after each input, with a nominal value of one. A fault is modeled by changing the amplification to zero and thus breaking the signal path. Sensor faults are model in the same way, inserting amplifiers before each output (see figure 9 on page 26)

3.4 Parameters Identification

3.4.1 Introduction

Once the system is modeled, it is necessary to assign values to the parameters used. Thus, several experiment have been done with the flight model in order

to determinate these values.

In this first stage the real time system and software provide by *DSpace* where used for the experiments. The software to control and acquire data is called *CockPit* and *Trace* (see section C on page 116). The real time system is used to acquire the data from the subsystems where are sensors — both rotor attack angle subsystems and the angular position system. Where there are not sensors —lateral rotors subsystem– an oscilloscope and a force meter were used to identify the parameters. Once the data was acquired they are processed using several programs written in order to identify the parameters of the model.

3.4.2 Data Acquisition and Representation

The function *datamatrix.m* processes and plot the data stored by *Trace*, using as a parameter the name of the file that contains the information. The data consists of a vector containing the time values and an array that contains the angle measured (β, γ_1 or γ_2) and the input (u_1, u_2 or u_3). The angular speed is calculated by differencing consecutive values. However, the result is not immediately useful because of the high amplification of measurement noise.

After this, two plots are made showing the inputs and the results of the experiment. A matrix is constructed containing the values of the input, the angular position and the instant speed at every time. This matrix will be used by other routines in order to identify the different parameters of each subsystem.

The program code is shown in the subsection B.1.1 on page 97.

3.4.3 Experiments

Using *Cockpit*, several excitations have been given to the experiment in order to identify the parameters of the different subsystems. The data is acquired using *Trace* and processed with *Matlab* functions which are developed in order to do this task. Some measurements are not possible using the existing sensors, therefore a force meter and an oscilloscope were used in addition to the existing sensors.

Rotors Attack Angle In order to identify the *rotor attack angle* subsystem parameters, several step functions have been applied to the system (see graphics in appendix D, paragraph D.1 on page 119 where is shown the *attack angle system* γ_1 behavior in response to different inputs). The following paragraphs detail each subsystem experiment.

The figures 60 and 61 show the reaction to different input values. An input above 0.1 is necessary to start turning the rotor. A higher value is necessary to generate a measurable force. There is a saturation level between 0.5 and 0.6 where an increase of the input has no effect on the speed of the rotor.

The figures 63 to 65 show the reaction in the input range from 0.2 to 0.3, from 0.3 to 0.4 and from 0.4 to 0.5. For every range, the differential amplification of the system is identified.

It is shown here that a positive input decreases the angle of attack, that means that the gain is negative. However, a negative angle of attack propels the system with a positive β speed. Therefore, the γ sensor values are negated in the interface block that controls the experiment from the real time system. Thus, positive amplification can be used in all parts of the model.

The data acquire in these experiments is processed in the next subsection in order to identify the system parameters.

Lateral Rotors. This subsystem was modeled as a first order system. Thus, the transfer function for the linear model is shown in the equation (8):

$$G_{lateralRotors}(s) = \frac{k}{1+\tau s} \tag{8}$$

Since there are no sensors in this subsystem other methods must be used to identify its parameters. A force meter is used to measure the force applied in the β direction. When a constant input is applied to the system, the force can be used to identify the system amplification. The time constant is identified from the motor current using an oscilloscope.

Gain Identification. The data acquired using the force meter are plot in the figure 14 (see also the table 4 in section E, on page 126). Thus, the gain is calculated by dividing the force by the input. This force include the effects of the speed proportional friction, thus this result is an approximation.

There is a small dead zone between -0.01 and +0.01 where the rotors do not move. A saturation is reached at a level of -0.60 and +0.80. Inputs below 0.2 move the propellers but not the system because of friction in the bearings. The gain of the system shows strong nonlinearity and it is not symmetric. This is a typical property of a propellers design for efficiency in one direction. For the linear system model only the positive direction is considered, and the friction is neglected for now.

The medium amplification value is 2.8, which is chosen for the linear model. Since with this gain value, the saturation level of 0.8 would give a force equal to 2.2 N but the actual maximum value is 2.0 N, the saturation is modeled at a slightly lower value to match the actual maximum value. This is necessary because the gain is no longer linear near the saturation level. The new saturation level is 0.7.

The gain of this system is negative, that means a positive voltage turns the system in the negative way. The orientation is inverted in the interface block of the real time system. Therefore in the linear model the amplifications are considered positive to make the controller design easier.

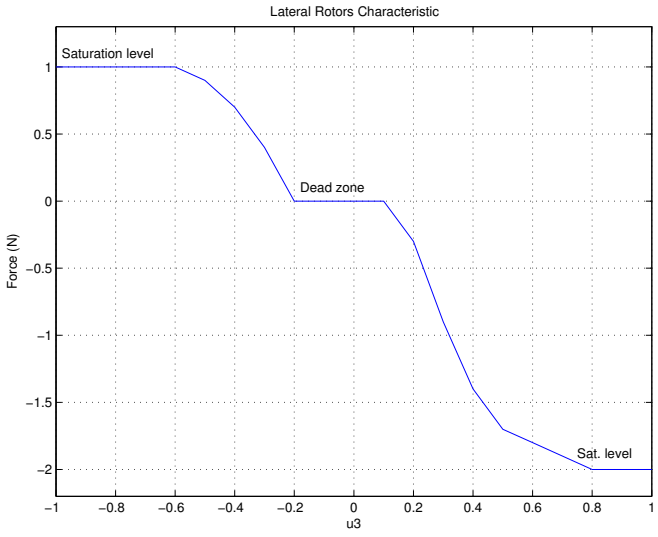


Figure 14: Identification of the *lateral rotors* gain

Time Constant Identification. In order to identify the time constant, the motor current is measured using an oscilloscope. The rotation speed is proportional to the input voltage. While the voltage shows a first order system behavior, the current shows the inverted one. The time constant of both values is identical.

On the plot, 37% of the resulting change are deduced from the final level. The point at which this lower level (63% of the change) is crossed by the plot corresponds to the time constant of the system.

The behavior is shown in the graphic 66 (appendix D on page 123). The calculation goes as follows: the initial voltage once the input is given, at 0.1 seconds, is 0.55 V and the permanent value is 0.1 V. Thus the 37% is 0.17 V and this level occurs at 0.3 seconds. So, the time constant is 0.2 seconds. Therefore, the modeled system has a dead zone between -0.01 and 0.01, a saturation level at 0.7 and the linear transfer function is shown in the equation (9):

$$G_{lateralRotrors}(s) = \frac{2.8}{1 + 0.2s} \tag{9}$$

Angular Position System. The graphics in the figures 67 and 68 (appendix D on page 124) show the behavior of the system when an input of 0.4 is given to the lateral rotors and when the main rotors are used with an angle of $\gamma_1 = 90°$. The flight model starts turning due to the acceleration cause by the force applied. The constant factor between the force applied and the resulting angular acceleration—which depends on the mass of system and on the axis length—is calculated. It corresponds to the angular inertia of the system.

There is a friction in the axis that prevent the system from moving unless the momentum applied is higher than a threshold. It is observed that at least an attack angle of 7° is needed in one of the main rotors or an input of 0.2 to the lateral rotors to start moving the plant in the β direction. Hence, by calculating the corresponding force a dead zone can be set up to model this friction in the bearing.

These data are processed in the next subsection in order to calculate the momentum constant value.

Pitch System. The flight model contains an automatic proportional controller for the main rotor speed. This controller keeps the system in a horizontal position. The gain is fixed to 3. The operation point is set to 7.3 for the first control, so that the system stays horizontally ($\alpha = 0$). The offset is set to 7.0, resulting in a force produced by the main rotors of 1.3 N each. To enhance the system dynamics, this value was later increased to 8.0, which results in a force of 3.6 N.

3.4.4 Data Processing

In the following paragraphs, the data acquired in the previous experiments is processed. This is done using different Matlab functions developed for this purpose.

Rotor Attack Angle Subsystem. The function *identify.m* calculates the parameters of a second order system — the gain (k) and the constant time (τ) — given by the linear transfer function shown in the equation (10).

$$G_{rotorAttackAngle}(s) = \frac{k}{s(1+\tau s)} \tag{10}$$

The function input is the data provided by the previously shown function *datamatrix.m* and the outputs are the parameters which characterize the system.

The sensor range is -90° to 90°, but there are significant errors near the end of the range (see figures 60 and 62 show on page 119). These problems can be avoided when the speed is calculated in the same turn and the measured angles are far enough from the end of the range.

The experiment must contain exactly one jump in the input. Initial and final average speeds are calculated in the previous and next turns. The step must be given at the beginning of the middle turn, allowing the system to reach the nominal speed before being measured. One turn is required before the measurement of the initial speed.

Thus the speed in each interval will be calculated as shown in the next equation

$$\omega = \frac{\theta(t_1) - \theta(t_2)}{t_1 - t_2} \tag{11}$$

where 1 and 2 mean two different measures during the same turn. Repeating this calculation before and after the jump allows to calculate the gain factor using this equation:

$$k = \frac{\theta(t_1) - \theta(t_2)}{u(t_1) - u(t_2)} \tag{12}$$

To calculate the time constant, the system trajectory is approximated by two lines: one before the jump and the second after the jump, according to the next equation

$$\theta(t) = \omega t + \theta_0 \tag{13}$$

Their slopes are the speeds previously measured. New angles are measured to identify θ_0 in both cases. The interval between the jump time and the line intersection time is the time constant of the system.

The dead zone is calculated by finding out the relation between input and speed shown in the next equation

$$\omega = ku + \omega_0 \tag{14}$$

and then extrapolating the input that corresponds to an angular speed of zero:

$$d = -\frac{\omega_0}{k} \tag{15}$$

The program code used to identify the *rotor attack angle* subsystem parameters is shown in the appendix B, subsection B.1.2on page 98.

Three different experiments are performed to identify the parameters: for the first one the input starts at 0.2 and a step is given to 0.3, the second one from 0.3 to 0.4 and the last one from 0.4 to 0.5. These jumps were used because they cover the range between the dead zone at 0.1 and the saturation level, between 0.5 and 0.6, without getting to close to the nonlinear regions.

Table 3 (appendix E, page 126) shows the data of the first of these experiments. Note the amount of noise present in the raw data. The results of *identify.m* are shown in the table 1.

The gain decreases slightly with increasing input levels. The gain is negative because a positive u_1 input leads to a decrease of $\gamma_1$angle. Mean values are chosen for the process model: a gain of 450 and time constant τ equal to 0.12

	$u_3 \in [0.2, 0.3]$	$u_3 \in [0.3, 0.4]$	$u_3 \in [0.4, 0.5]$	Media
Gain k	-462	-454	-433	-450
Time constant τ (s)	0.14	0.11	0.12	0.12
Dead zone	0.086	0.082	0.067	0.078
Initial speed $\omega_1(^\circ s^{-1})$	-53	-99	-144	—
Final speed $(^\circ s^{-1})$	-99	-144	-188	—

Table 1: Identification results

seconds. As with the *lateral rotors subsystem* identification, the gain is not constant near the saturation level. Using a mean gain value, the saturation level would give an angular speed which would exceed the maximum real one. Thus, the saturation is modeled at a slightly lower value to match the actual maximum value. The new saturation level is 0.45. Hence, it is ensured that the angular speed in the model does not exceed 200, which is about the highest value seen in the experiment.

The dead zone values differ significantly between the three experiments. Values of 0.09 for the main rotor 1 and 0.07 for the main rotor 2 were measured experimentally, which are reasonably close to the values seen here. These values are therefore taken for the model.

The resulting linear model is described by equation (16) and the detailed nonlinear model includes a saturation level of 0.45 and a dead zone of 0.09 or 0.07 depend on the rotor.

$$G_{rotorAttackAngle}(s) = \frac{450}{s(1 + 0.12s)} \tag{16}$$

Angular Position System. The script *momentum.m* is used to calculate the constant J between force and angular acceleration using the data from the β experiments (see subsection 3.4.3 on page 33). The force was already calculated in the *lateral rotors* and *pitch* experiments (see subsections 3.4.3 —on page 31— and 3.4.3 —on page 34) and it had a value 1.4 N and 1.3 N respectively.

To calculate the inertia, it is essential to know when the acceleration begins. Analyzing the graphics on figures 67 and 68 on page 124, it is concluded that the movement starts at 1.7 seconds for the first experiment (while the control

input was given at 1.27 seconds) and at 0.8 seconds for the second one. With this data the results are $J_I = 10.9Ns^2/^\circ$ in the first case and $J_m = 6.1Ns^2/^\circ$ in the second one. A difference is to be expected, since the constant includes the length of the lever. The longer the lever, the lower is the constant.

It is observed that a positive input given to the servo motors turns them in the negative way but then the system is propelled in positive direction.

The script is listed in the subsection B.1.3 on page 100.

Now it is possible to calculate the bearing friction, F_b, and its effect on the two different actuators:

$$\begin{aligned} F_l = u_{3min} k_3 &= 0.7N \\ F_m = F \sin \gamma_{min} &= 0.16N \end{aligned}$$

where γ_{min} =7°, $u_{3min} = 0.25$. Since a single block models the friction, both values have to be converted into a momentum. The result is:

$$\begin{aligned} \frac{F_l}{J_l}\frac{180}{\pi} &= 0.36^\circ s^{-2} \\ \frac{F_m}{J_m}\frac{180}{\pi} &= 0.4^\circ s^{-2} \end{aligned}$$

Both values are reasonable close, and the second value $F_b = 0.4^\circ s^{-2}$ is used for the model.

3.4.5 Final Note

The parameter identification of this system is rather difficult for two unrelated reasons. Firstly, the angles are measured with standard potentiometers. They show only moderate precision, a significant amount of noise and strong nonlinearities at the end of the measurement range (see figures 60 and 62 on page 119). These problems could be solved by replacing the potentiometers by digital encoders with a high resolution (at the cost of introducing a quantization). The second problem is that the friction effects are not reproducible. The connection wires can take different positions leading to very different friction forces. The friction in the bearing was also changing from experiment to experiment.

3.5 Linear Model

Once the plant is modeled and all its parameters are identified, the next step is to linearize the model for the controller design. For this purpose a M*atlab* script has been developed. The script is called *modeling.m* (shown in the subsection B.2.1 on page 101 in the appendix B).

First it loads the parameter of the linear system which have been previously identified for the *flightmodel.mdl* model—the time constants, the gain, the force applied by the main rotors, the effect of the mass and length of the axis and the speed proportional friction. At this point, dead zones and the saturation levels of each subsystem are not used, neither the friction in the bearings.

The next step is to identify each state and associate them with the different integrators. This is quite important since *Simulink* assigns the integrators to the different state variable in a random order and it is necessary to know this order to control the system. By calling the *Simulink* file, *Matlab* returns some information about the model. This includes the number of the system inputs, outputs and states and the order of the integrators. Comparing the obtained list with the names of the different integrators, it is possible to calculate a transformation matrix, $\mathbf{T}$, to bring states into a standard order. The transformation is carried out using the next equations:

$$\begin{aligned} \mathbf{A} &= \mathbf{T}^{-1}\mathbf{A}\mathbf{T} \\ \mathbf{B} &= \mathbf{T}^{-1}\mathbf{B} \\ \mathbf{C} &= \mathbf{C}\mathbf{T} \end{aligned}$$

Then, the model is linearized and the state-space representation is obtained, given by the matrix $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ (equations (17) and (18)). For this purpose, it was used the function *linmod.m* provided with Matlab. The operating point is set at zero for all the state-space variables. That means the state-space variables stay within the surroundings of the operating point.

The states of the model are reordered using the transformation matrix previously calculated. The resulting model has a known order of states. This order is the following one: $\dot{\gamma}_1$, $\dot{\gamma}_2$, γ_1, γ_2, F_l, $\dot{\beta}$ and β. All further consideration apply to this transformed model only.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{17}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{18}$$

with

$$\mathbf{A} = \begin{pmatrix} -8.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3750 & -8.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3750 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0.6 & 0.6 & 73.6 & -0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{19}$$

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{20}$$

$$\mathbf{C} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{21}$$

$$\mathbf{D} = \mathbf{0} \tag{22}$$

Before carrying out the linearisation the nonlinear effects were set to zero. In order to perform more realistic simulations, the real values are now loaded. This includes the dead zone and the saturation levels of each subsystem and the friction in the bearings. The design of both the controller and the observer is done using the linear model. The simulation are carried out using the more realistic scheme that includes nonlinear effects and also processing delays.

4 Nominal Control Loop

4.1 Introduction

A standard compensator design is used to control the nominal process. It consists of a state observer and a linear state feedback controller. Both observer and controller are designed using the Linear Quadratic Gaussian (LQG) method [3]. This approach is chosen because it provides good performance in the closed loop and because the design process is mainly automatic. The control law and the observer are built from basic blocks in a Simulink model called *flightmodelcl.mdl*. This model is also used to simulate the controller performance as part of the design cycle, and it is used for verification before trying to control the real system.

The control structure is shown in the figure 15. The controller generates the plant input. The controller input is the difference between the reference and the observed state-space variables. The state observation comes from the state observer.

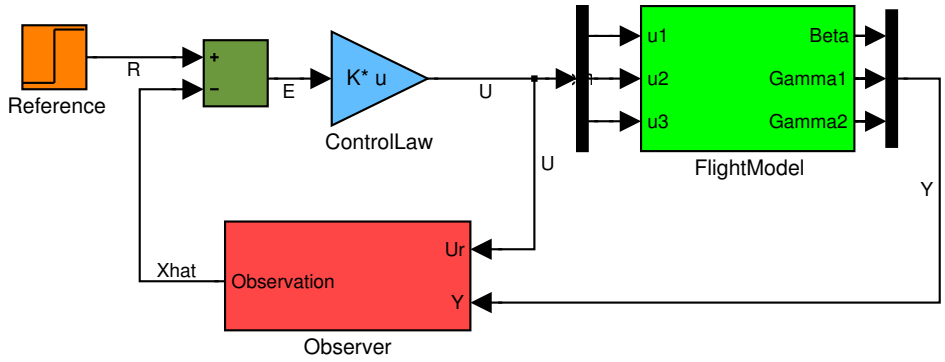


Figure 15: Control structure

4.2 System Blocks

Apart from the observer and the controller, the simulation model includes some other blocks necessaries to care for input limits and to simulate effects of the real system. The complete structure is shown in the figure 16.

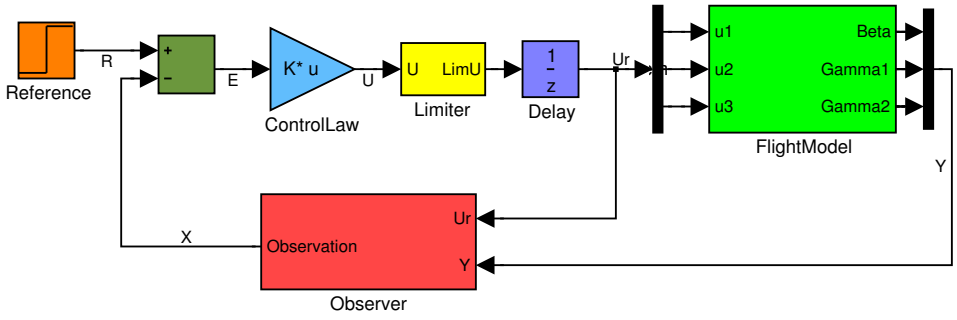


Figure 16: Compensator

Reference. The reference is the desired state of the system. For the experiments, only β is used. All the other state-space variable references are equal to zero. Other values are not used because they are not compatible with a stationary state of the system.

Controller. The controller output is a linear combination of the state deviation. The control law is given by the linear state feedback:

$$\mathbf{u} = \mathbf{K}(\mathbf{x}_{reference} - \hat{\mathbf{x}}) \tag{23}$$

Limiter. The next block in the control chain is a limiter. It cuts input values to the range that can be realized by the system. While this block has no influence on the process or its model, it does increase the accuracy of the observer by avoiding unreasonably high input values (figure 17).

Delay. A delay block is used to model the time needed to evaluate the control law in the real time system. Since the sensor data is read by the real time system and then processed, it takes at least one sample time before the signal reaches process inputs. Due to filters and other blocks in the data acquisition chain the real delay can be slightly higher than one sampling period.

Plant. The next block is the plant mode. The system model used is the detailed nonlinear model described in the last section (see section 3 on

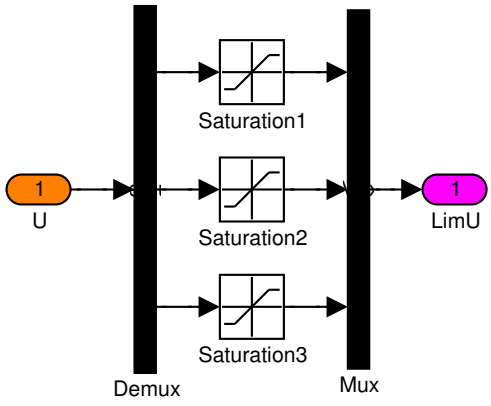


Figure 17: Limiter

page 25, figures 8 to 13). However, the controller and the observer are designed using the linear model given by:

$$\begin{aligned} \dot{\mathbf{x}}_N &= \mathbf{A}_N\mathbf{x}_N + \mathbf{B}_N\mathbf{u}_N + \mathbf{B}_v\mathbf{v} \\ \mathbf{y}_N &= \mathbf{C}_N\mathbf{x}_N + \mathbf{D}_N\mathbf{u}_N + \mathbf{D}_w\mathbf{w} \\ \mathbf{x}_N(0) &= \mathbf{x}_0 \end{aligned}$$

where $\mathbf{u}, \mathbf{x}, \mathbf{y}$ are the system inputs, states and outputs; $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ the system matrices; $\mathbf{v}, \mathbf{w}$ uncorrelated, normalized white noise; $\mathbf{B}_v$,$\mathbf{D}_w$ the disturbance weighting matrices.

Observer. The observer block follows the state observation law:

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}(\mathbf{y} - \mathbf{C}\hat{\mathbf{x}}) \tag{24}$$

This law is based on the system model adding a feedback proportional to the innovation (the difference between the measured output and the observed one). The block model is shown in the figure 18.

4.3 Controller Design

The LQG method is used for the controller design because it is effective and well studied in modern control theory. LQG stands of Linear Quadratic

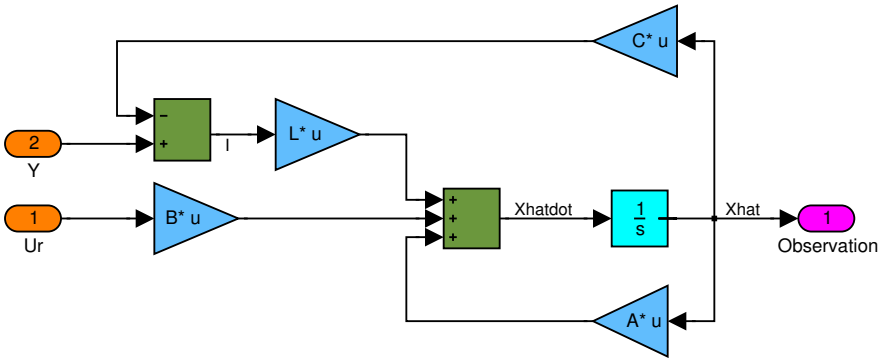


Figure 18: Observer

Gaussian. This technique allows to design the controller automatically and there is no need to find out desirable locations for the system roots. The design finds the optimal control law for minimizing the performance index given by the next equation:

$$J^* = minE\left[\int_0^\infty \left(\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right) dt\right] \tag{25}$$

where the matrix $\mathbf{Q}_c$ is the state costs and $\mathbf{R}_c$ the inputs costs. In this work, $\mathbf{Q}_c$ and $\mathbf{R}_c$ are diagonal and their diagonal terms describe the importance of a state-space variable deviations and the costs of having a high input respectively.

The design technique is implemented in the *Matlab* function *lqr.m* that can be found in the control system tool box. This function works as follows:

1. Because of a proportional feedback state controller is used, the system can be expressed as

$$\dot{x} = \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}\mathbf{x}$$

2. Thus, the performance index is

$$J^* = \frac{1}{2}\int_0^\infty \mathbf{x}^T\left(\mathbf{Q} + \mathbf{K}^T\mathbf{R}\mathbf{K}\right)\mathbf{x}dt \tag{26}$$

3. Assuming that exists a constant matrix, $\mathbf{P}$,such that

$$\frac{d}{dt}\mathbf{x}^T\mathbf{P}\mathbf{x} = -\mathbf{x}^T\left(\mathbf{Q} + \mathbf{K}^T\mathbf{R}\mathbf{K}\right)\mathbf{x} \tag{27}$$

4. the performance index can be written as

$$J^* = \frac{1}{2}\mathbf{x}(0)^T\mathbf{P}\mathbf{x}(\mathbf{0})$$

5. Then, selecting the following gain matrix

$$\mathbf{K} = \mathbf{R}_c^{-1}\mathbf{B}^T\mathbf{P}_c \tag{28}$$

6. the equation 27 leads to the Algebraic Ricatti Equation (ARE)

$$\mathbf{A}^T\mathbf{P}_c + \mathbf{P}_c\mathbf{A} + \mathbf{Q}_c - \mathbf{P}_c\mathbf{B}\mathbf{R}_c^{-1}\mathbf{B}^T\mathbf{P}_c = 0$$

7. The function *lqr.m* solves this equation for $\mathbf{P}_c$and calculates the solution for the optimal control problem given by the equation 28.

Starting with identity matrices, different values for $\mathbf{Q}_c$ and $\mathbf{R}_c$ are tried until an acceptable behavior is found. The simulation involves moving the system in β by 45°. Problematic effects include too fast motion of the system, high frequency swings, high overshoots or saturated input signals for extended periods of time. They are avoided by modifying the cost matrices accordingly. This process is repeated several times until a reasonable behavior is obtained.

A *Matlab* script called *controller.m* was written (see appendix B, subsection B.2.2 on page 104) which implements the controller design.

4.4 Observer Design

The observer design problem is dual to the controller design, therefore the same approach can be used to calculated the observer feedback gain, $\mathbf{L}$. To construct the dual problem it is necessary to replace the matrix $\mathbf{B}$ by the transposed $\mathbf{C}$ matrix and to transpose $\mathbf{A}$. The equivalent cost matrices, called $\mathbf{Q}_o$ and $\mathbf{R}_o$, measure the disturbance to the system states and outputs.

The solution is obtained by solving the next Algebraic Ricatti Equation for $\mathbf{P}_o$:

$$\mathbf{A}\mathbf{P}_o + \mathbf{P}_o\mathbf{A}^T + \mathbf{Q}_o - \mathbf{P}_o\mathbf{C}^T\mathbf{R}_o^{-1}\mathbf{C}\mathbf{P}_o = 0$$

where the cost matrices depends on the disturbance weighting matrices as

$$\begin{aligned} \mathbf{Q}_o &= \mathbf{B}_v\mathbf{B}_v^T \\ \mathbf{R}_o &= \mathbf{D}_w\mathbf{D}_w^T \end{aligned}$$

thus, the optimal observer gain is given by

$$\mathbf{L} = \mathbf{P}_o\mathbf{C}^T\mathbf{R}_o^{-1}$$

The rest of the design process is similar to the controller design. However, the simulation of the resulting observer is difficult because no good model of state and output disturbances exists. Therefore several experiments were necessary to find a good observer.

The *Matlab* script *observer.m* (see appendix B, subsection B.2.3 on page 106) implements the observer design.

4.5 Simulations

In order to design both the controller and the observer, it is necessary to simulate the behavior of the resulting control loop. The faultless case is considered, this means that all fault gains are set to '1'. The cost functions are modified in response to the simulation results till satisfactory system behavior has been reached. Their initial value is set to one.

At first nonlinear effects (friction and dead zones) are not activated. The simulation experiments consists of a reference jump in β by 45° at instant 1 second (see figure 19). The initial state and all other reference elements are set to zero.

It can be seen in the simulation that the inputs have too many oscillations, reaching both saturations levels several times (see figure 20). The γ values

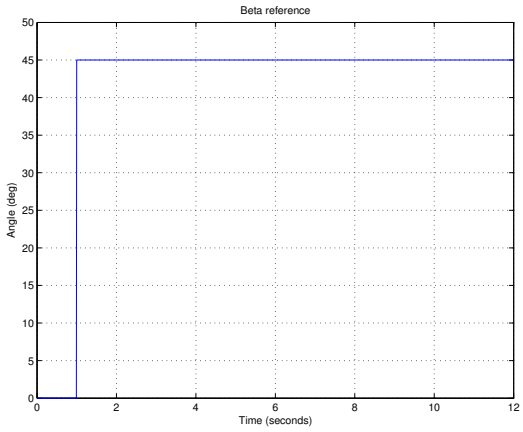


Figure 19: Reference

are low enough, so that mainly linear system behavior can be expected. High values of γ render the system unstable, because they render the pitch controller for the angle α (which is not modeled here) ineffective. Here too oscillation are present. The reference value is reached in less than 10 seconds, which is acceptable.

To solve the oscillation problem the controller amplification is reduced by increased the values of all inputs costs by a factor of 100. The oscillations have disappear (figure 21), and the inputs go into saturation for a short period of time only. The state β shows good performance with no noticeable overshoot. The rising time is less than 5 seconds, which is very fast. The gammas states reach a maximum of less than 20°, which is a reasonable value. To further reduce the input energy, the cost for u_3 was increased again. However, this lead to a slower system response and to slight oscillations around the reference position due to difficulties to precisely control the system via the main rotors.

In a second step, the system is simulated including all nonlinear effects like dead zones, friction and saturation levels. As it is shown in the figure 22 the plant behavior is slightly different with a small overshoot in β (about 3°) and it has a steady state error of 1.5 °. These problems can only be solve using a different control approach, therefore no further changes are made to the cost functions. The resulting performance is acceptable.

Finally, reasonable observer cost functions are chosen. As the graphics (fig-

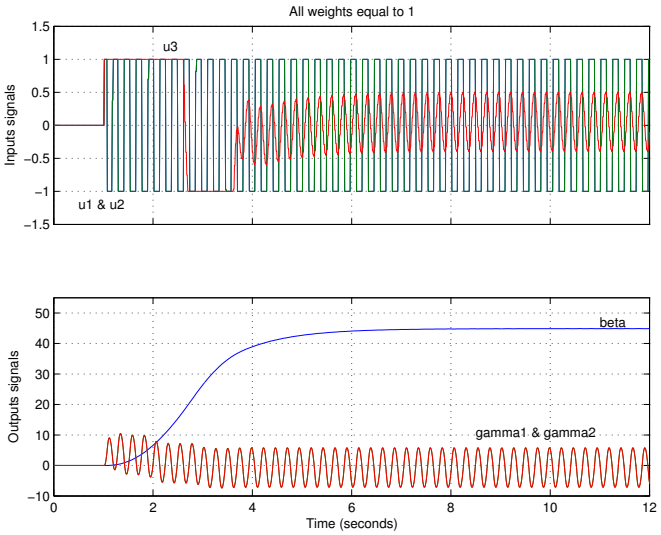


Figure 20: Initial cost values

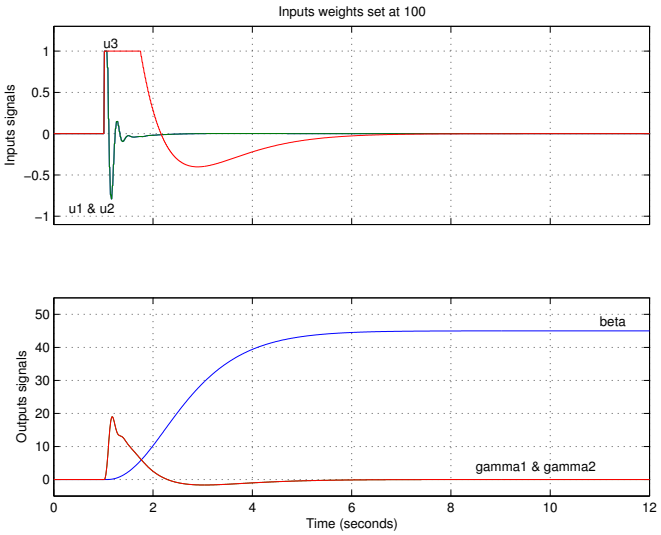


Figure 21: Final cost values

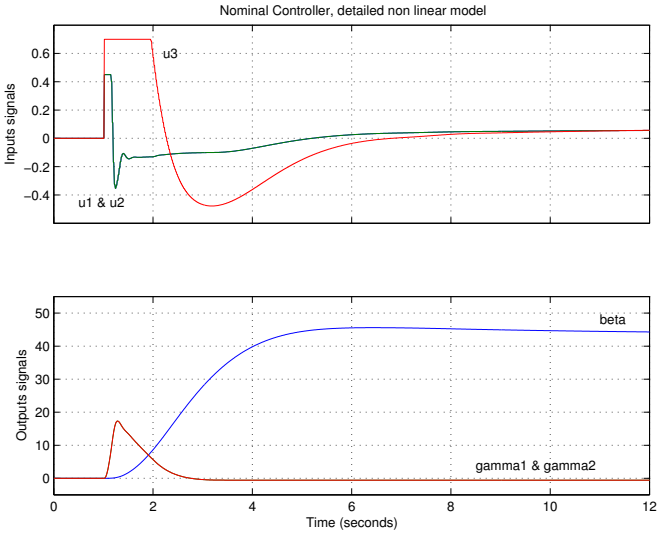


Figure 22: Setting dead zones

ures 22 and 23) show, the behavior of the space state variables and their observed value (figure 24) is very similar. It can be concluded that the observer is working correctly and there is no need to modify the costs. This is due to the lack of noise in the simulation model. Therefore, some of the observer cost functions will be modified after testing the real behavior.

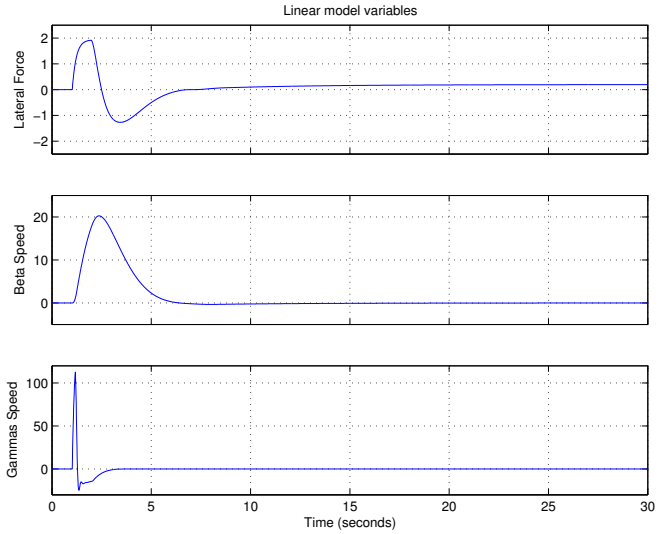


Figure 23: Linear model variables

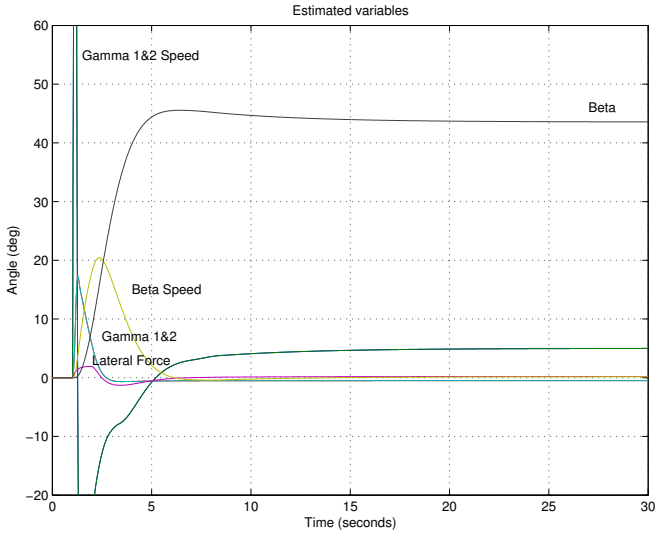


Figure 24: Observed state-space variables

4.6 Results

A nominal control loop, including a proportional state feedback controller and a state observer, have been implemented.

As a result of the design process, the matrices K and $\mathbf{L}$ are calculated. They are used in the nominal control configuration. These matrices are show in the equations (29) and (30).

$$\mathbf{K} = \begin{pmatrix} 20.61 & 0.322 & 0.103 & 0.0025 & 0.680 & 0.058 & 0.041 \\ 0.322 & 20.61 & 0.0025 & 0.103 & 0.680 & 0.058 & 0.041 \\ 0.680 & 0.680 & 0.0053 & 0.0053 & 1.481 & 0.122 & 0.082 \end{pmatrix} \quad (29)$$

$$\mathbf{L} = \begin{pmatrix} -0.0001 & 0.825 & 0 \\ -0.0001 & 0 & 0.825 \\ 0.0019 & 78.68 & 0 \\ 0.0019 & 0 & 78.68 \\ 0.113 & 0 & 0 \\ 12.11 & 0.591 & 0.591 \\ 5.020 & 0.0019 & 0.0019 \end{pmatrix} \quad (30)$$

Only the controller input cost matrix have been modified. This matrix is shown in the equation (31). The controller state cost matrix and both observer cost matrices are the identity matrix.

$$\mathbf{R} = \begin{pmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{pmatrix} \quad (31)$$

4.7 Conclusions

Using the LQG design method, the controller and observer gain matrices have been obtained. They stabilize the system and provide a good reference tracking behavior for the nominal process model. Some of the elements of the controller cost matrices could be modified when testing the plant due to deviations from the linear behavior.

There was no need to modify the observer cost matrices due to the simulation uses the same model for both the plant and the observer. As it is shown in

the **L** matrix, the trust in the sensors γ_1 and γ_2 to calculate the respective states is high. Therefore, the system behavior could be more or less affected depending on the sensor noise. This fact will be considered later. Whether it is necessary, the cost functions for γ_1 and γ_2 (now set to one) will be increased.

5 Reconfiguration After an Actuator Fault Detection

5.1 Introduction

The main part of this project is to test a control structure that can be changed in response to a fault in the process. The idea is to stabilize the system and to achieve reference tracking with the best possible performance even after the loss of an actuator. Thus, the system is kept in operation despite the fault. It is assumed that the relevant states of the faulty system are still controllable, because otherwise reconfiguration is impossible. This means that faults on main rotor actuators are not considered. Because the reconfiguration must be automatically, the same method used for the original controller design is also used for the fault cases, keeping the cost index.

Once the plant is modeled and both the controller and observer are designed, the faulty system behavior can be simulated. The actuator faults are simulated by inserting in each input an amplifier. The nominal gain value is '1' at it changes into '0' when a fault happens. Thus, it breaks the signal path.

Several *Matlab* routines have been written to deal with faults and to perform the steps of the reconfiguration. The system behavior has been simulated for several fault scenarios including different actuator blockages. These simulations are used for verification before trying to control the real system in faulty cases.

The control structure used is the one shown in the figure 16 on page 41.

5.2 Problem Description

An actuator fault affects the process input matrix, $\mathbf{B}$, causing a zero column. All other matrices are unchanged:

$$\begin{aligned} \mathbf{A}_F &= \mathbf{A}_N = \mathbf{A} \\ \mathbf{B}_F &\neq \mathbf{B}_N \\ \mathbf{C}_F &= \mathbf{C}_N \ . \end{aligned}$$

After the fault has occurred, a new control structure has to be found in response to the new situation. The goal is to keep the plant in operation

despite loosing an actuator. Since a state observer/state feedback structure is used for the nominal controller, the state feedback gain matrix has to be recalculated.

5.3 Way of Solution

The idea of the control reconfiguration is to recalculate the state feedback matrix. The same method will be used as for the nominal controller design, which was a Linear Quadratic Gaussian (LQG). Using this design method is simple and effective: the LQG design finds the optimal controller for the new situation. Same weights can be used as in the nominal case, allowing the design process to be completely automatic. Due to the separation principle, there is no need to change the observer.

Once a fault is detected, the system is remodeled calling *modeling.m.* Then, the controller is redesigned and the new feedback gain is down-loaded on line into the target PC.

5.4 Rank Deficiency

Unfortunately, the implementation of LQG design in *Matlab* fails because of singularities in the process model introduced by the fault. The column in $\mathbf{B}$ corresponding to the faulty input is equal to zero, because the input has no effect on the state-space variables. Due to this rank deficiency, which causes numerical problems, the LQG method does not work reliably. Moreover, some states are not input connected in the fault case and therefore not controllable. It is possible that some other states depend only on those which are no longer input connected (such as $\dot{\gamma}_1$ on γ_1). Hence, they are also non controllable.

To solve this problem, a reduction of input and state space is necessary. The reduction eliminates the non-controllable states and the broken inputs. Then, the LQG design can be applied to the reduced process model to obtain the feedback matrix $\mathbf{K}'$. Finally, this matrix has to be expanded so that it can be applied to the actual faulty process. Thus, the rows and columns of $\mathbf{K}$ not present in the reduced design will be filled with zero elements.

In the first step, a transformation matrix $\mathbf{T}_x$ is calculated. It eliminates the non-controllable states. This will be done by the function *ReduceX.m (*see

appendix B, subsection B.3.1on page 110). It calculates the controllability matrix and looks for zero rows that point out the non-controllable states. Then, the matrix which eliminates the corresponding rows in the $\mathbf{A}$ matrix is found and returned. The controllability matrix, $\mathbf{CO} = [\mathbf{B}, \mathbf{AB}, \mathbf{A}^2\mathbf{B}...\mathbf{A}^{n_x-1}\mathbf{B}]$, is calculated calling the *Matlab* function *ctrb.m*.

In the second step, the transformation matrix $\mathbf{T}_u$ is calculated. It reduces the $\mathbf{B}$ matrix eliminating the columns of unconnected inputs. This will be done by the function *ReduceU.m (*see appendix B, subsection B.3.2 on page 111). It looks for zero columns that point out the faulty actuators, calculates the matrix which eliminates the corresponding columns in the $\mathbf{B}$ matrix and returns it.

The rows and columns of $\mathbf{A}$ and $\mathbf{B}$ are then reduced according to the following equations:

$$\mathbf{A}' = \mathbf{T}_x^T \mathbf{A} \mathbf{T}_x \tag{32}$$
$$\mathbf{B}' = \mathbf{T}_x^T \mathbf{B} \mathbf{T}_u \tag{33}$$

The next step is to reduce the cost matrices $\mathbf{Q}_c$ and $\mathbf{R}_c$ accordingly:

$$\mathbf{Q}_c' = \mathbf{T}_x^T \mathbf{Q}_c \mathbf{T}_x \tag{34}$$
$$\mathbf{R}_c\prime = \mathbf{T}_u^T \mathbf{R}_c \mathbf{T}_u \tag{35}$$

This completes the transformation. Then, the *Matlab* function *lqr.m* is called to solve the reduced and well defined controller design problem. It calculates the reduced gain matrix ,$\mathbf{K}'$.

Finally, it is expanded up to the system dimensions, according to this equation:

$$\mathbf{K} = \left[\left(\mathbf{Z}_u^T \mathbf{Z}_u \right)^{-1} \mathbf{Z}_u^T \right]^T \mathbf{K}' \left[\left(\mathbf{Z}_x^T \mathbf{Z}_x \right)^{-1} \mathbf{Z}_x^T \right] \quad . \tag{36}$$

5.5 Reconfiguration Algorithm

The actuator fault case reconfiguration algorithm follows the following steps:

1. The actuator fault has to be detected and identified (this is assumed to be done by a FDI module).

2. The plant model is updated including unconnected inputs (calling *modeling.m,* see subsection B.2.1 on page 101).

3. The new model is linearized (done also by *modeling.m*).

4. The system matrices are reduced eliminating uncontrollable states and unconnected inputs (calling *ReduceX.m* and *ReduceU.m*, see subsections B.3.1 and B.3.2).

5. A new controller is design according to LQG techniques (calling *controller.m*, see subsection *B.2.2).*

6. The controller matrix is expanded up to the system dimensions.

7. The new control law is down-loaded into the real time system controlling the plant.

All steps of the process are done online, while the plant is in operation. After the completion of the algorithm, a reconfigured controller is in effect. The redesigned control loop stabilizes the plant despite the fault.

5.6 Simulations

The approach shown above is applied to the flight model. The following fault cases are considered.

Case 1: Actuator u_1 Failed. The first plot (figure 25) shows the system step response when the actuator u_1 fails but no reconfiguration is performed. By coincidence the system is still stable, but there is a significant overshoot. The second plot (figure 26) shows the behavior after the reconfiguration, and obviously the behavior is improved. The controller knows that the actuator u_1 is not working and does not use it. The system remains operational and the performance is almost as good as in the nominal case. There is very little overshoot (less than 1°) and a small steady state error (0.5°). It is nearly as fast as it was in the nominal case. The remaining actuators are working

slightly harder and longer to stand in for the faulty actuator, but they still remain within reasonable limits and the maximum rotor attack angle is below 20°.

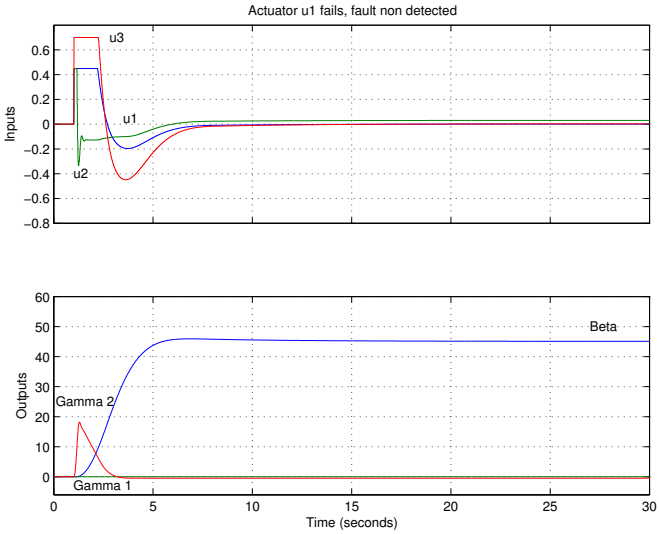


Figure 25: Actuator u_1 failed, nominal controller

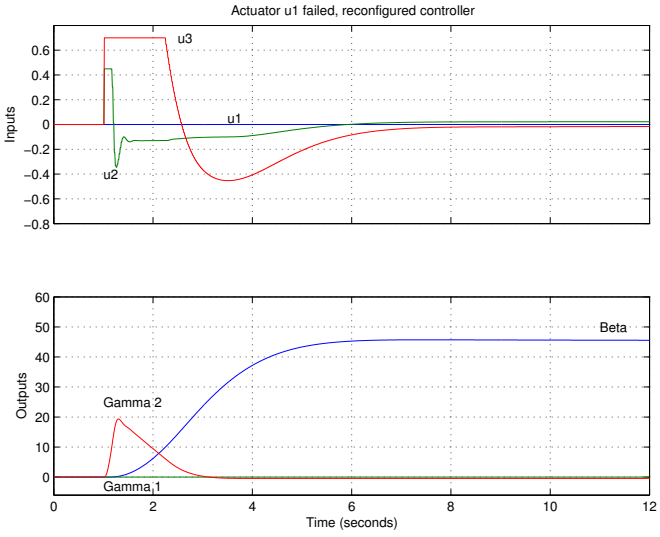


Figure 26: Actuator u_1 failed, reconfigured controller

Case 2: Actuator u_3 Failed. The system behavior before the reconfiguration is shown in the first graphic (figure 27) and afterwards in the second one (figure 28). It is obvious that the unreconfigured system shows heavy oscillations even in the simulation, which means that the experiment on the real system will probably fail.

The reconfigured control loop on the other hand shows a significantly improved performance. The main rotors have to be turned for a longer period of time (figure 27) than in nominal case to compensate for the loss of actuator u_3. The angle of attack reaches a maximum of 28°, which is still acceptable but close to the end of reasonable linear system behavior. This fault affects the system operation much more than the fault in the actuator u_1. It is possible to see this effects in the simulations because a detailed nonlinear model including limits and friction is used. The friction is the reason for the oscillation observed in β, before it reaches its final value just above the reference.

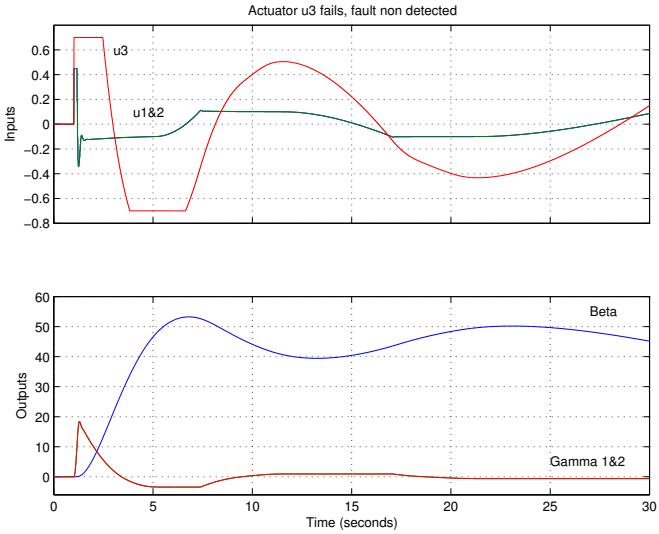


Figure 27: Actuator u_3 failed, nominal controller

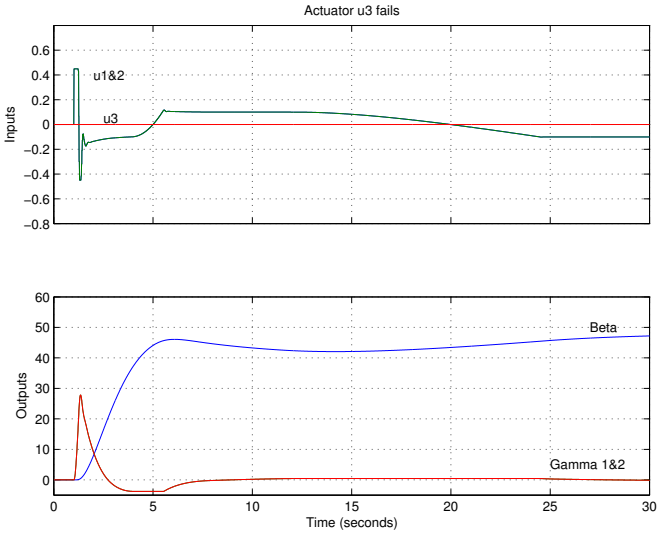


Figure 28: Actuator u_3 failed, reconfigured controller

5.7 Results

A reconfiguration algorithm has been developed. It update the system model and linearize it. Then, it calculates the new controller gain and down loads it into the Target PC. The functions which reduce the order of the system matrices and expand the gain matrix have been implemented. This results allow to reconfigure the plant after an actuator fault on line.

5.8 Conclusions

It has been shown how the redesign of the state feedback controller can be used to reconfigure the compensator in case of a fault in a system actuator.

It has been demonstrated that the LQG method used for the design of the nominal controller can also be applied to the fault cases. This allows to reconfigure the system without manual intervention once a fault is detected. The cost functions of the nominal controller design can be used for the fault cases. Rank deficiency problems that occur during the re-design process have been solved. Thus, the reconfigured system is stable and shows good to moderate performance depending on the fault case.

6 Reconfiguration After a Sensor Fault

6.1 Introduction

As in the actuator fault case, the main task after the loss of a sensor is to stabilize the system and to achieve reference tracking with the best possible performance. Thus, the system is kept operational despite a sensor fault. The faulty system must be still observable in order to reconfigure it. This implies that faults on α and β sensors are not considered.

Sensor faults are also modeled with amplifiers, this time before the outputs. The value is '1' when the sensor works properly and '0' if it is at fault.

6.2 Problem Description

When a sensor is at fault, the observer gets wrong output values from the plant. The error spreads through the controller into other parts of the system. System stability can be affected, therefore this fault can render the plant inoperational.

The faulty process model is identical to the nominal model with the exception of the process output matrix, $\mathbf{C}$; in this case, the fault causes a zero row:

$$\begin{aligned} \mathbf{A}_F &= \mathbf{A}_N = \mathbf{A} \\ \mathbf{B}_F &= \mathbf{B}_N = \mathbf{B} \\ \mathbf{C}_F &\neq \mathbf{C}_N \end{aligned}$$

6.3 Way of Solution

The sensor reconfiguration idea is to use a different observer for every fault case and switch between these observers. Therefore, a bank of dedicated observers is designed, one for every sensor fault case plus a standard observer for the nominal case which is used when all sensor works correctly. This approach is known as "dedicated observer scheme". When a fault is detected, the corresponding observer is used to supply the state observation. This approach assumes that in every fault case the plant is fully observable. In

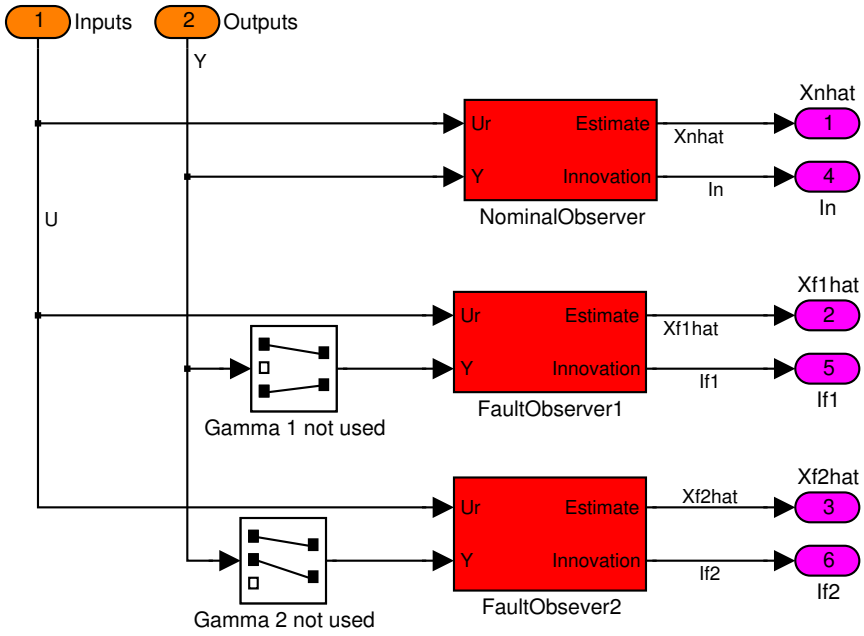


Figure 29: Observer bank

case of multiple sensor faults or if the β sensor is at fault, the system is no longer observable and therefore this case is not considered.

Each fault case observer uses all plant inputs and all outputs apart from the output at fault in the corresponding fault case. This way at least one observer is unaffected by the fault and can be used to supply the state observation necessary for the state feedback controller. Because of this observer produces a valid observation with and without the fault, its state is valid at any time. Thus, the plant can be controlled even after a sensor fault. This is implemented in the *Observer Bank* block (see figure 29) which is included in the reconfiguration model (see figure 30).

In order to design the Fault Case Observers, a fault is introduced in the $\mathbf{C}$ matrix by changing the corresponding '1' into '0'. Thus, the matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ are obtained.

In this case, there is no need to change the controller.

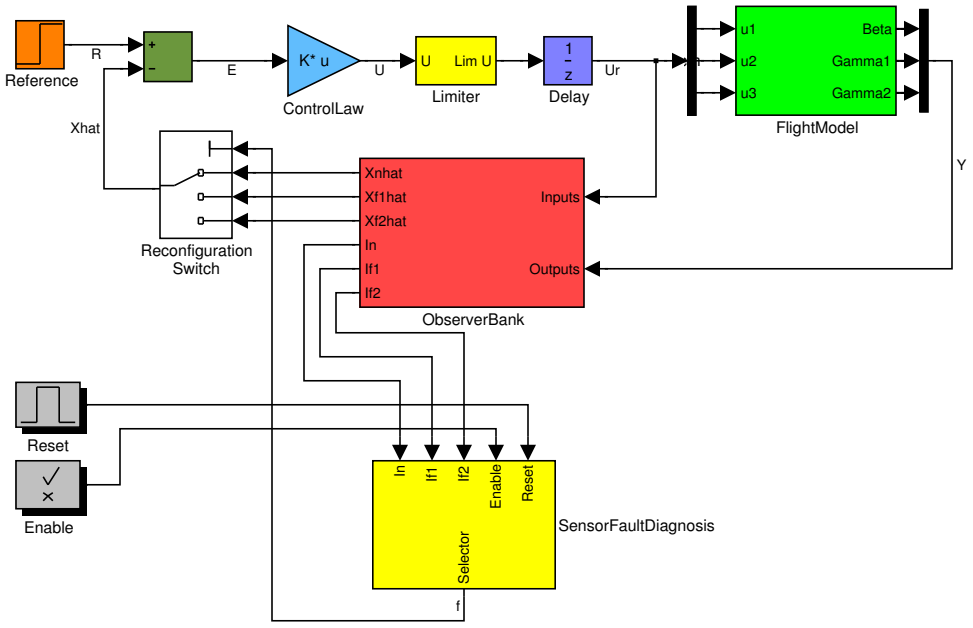


Figure 30: Reconfiguration model

6.4 Fault Detection

It seems reasonable that the bank of observer design used for the reconfiguration task can be used for fault detection, too. The fault detection relies on the innovation of each observer. The observer that does not use the faulty sensor should have the best state observation. Thus, a diagnosis logic will detect and isolate the fault and select the best observer (see figure 30). This logic has three main blocks, shown in the figure 31 and explain in detail in the following paragraphs.

Signal Processing. This block (figure 32) filters the innovation signals and calculates their absolute values. The low pass filters reduce the sensitive to noise or short disturbances. The cut frequency is selected to be 10 Hz, a value which ensure that the filter can follow the real signal but prevents that short signal disturbances affect the further FDI logic.

The filtered nominal innovation vector is then summed up, resulting the scalar deviation measure. The β measure is not take into account. Since the

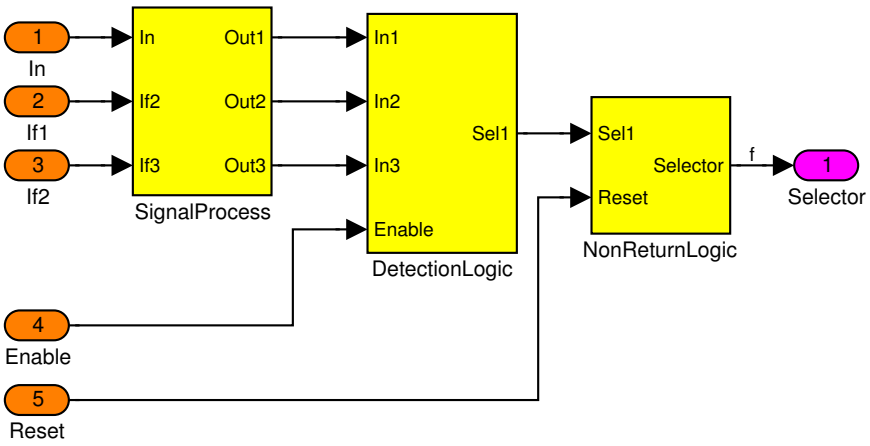


Figure 31: Diagnosis logic

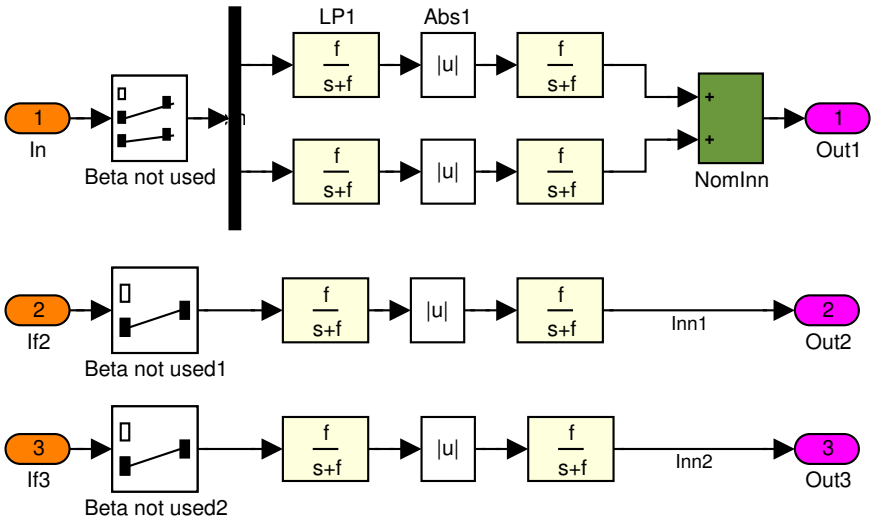


Figure 32: Signal process

considered sensor faults are restricted to both angle attack potentiometers, there is no reason to include the β measure. On the other hand, it could affect the selection logic.

Detection Logic. This logic detects and isolates the fault, providing the signal which switch into the correct observer (figure 33).

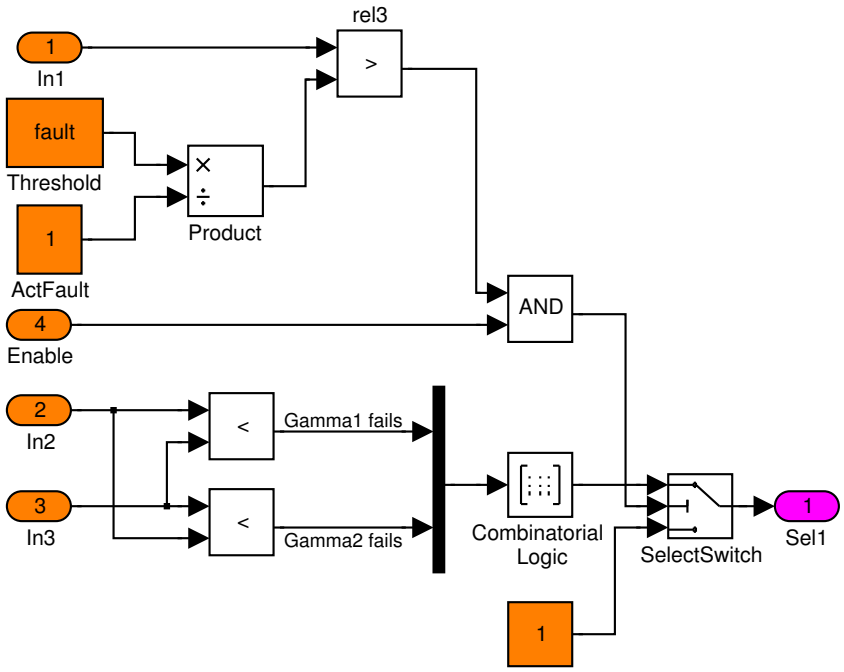


Figure 33: Detection logic

The measure from the nominal observer is compared to a threshold. This comparison is used to detect a fault in the process and to activate the identification logic. The threshold was found experimentally. Since in a nominal case the sum is around 0.25 and in a faulty case it is between 0.8 and 1.2, a threshold of 0.5 is selected.

The nominal innovation is the norm of the deviation vector. This vector has two components: the γ_1 and γ_2 deviations. When a servo-motor actuator fault happens, one of the components is zero. Therefore, the nominal innovation falls. In this case, a sensor fault could be non detected. Thus, a

Input1	Input2	Output	Comment
0	0	1	No fault, selects the Nominal Observer
0	1	3	γ_2 fails, selects the Fault Observer 2
1	0	2	γ_1 fails, selects the Fault Observer 1
1	1	1	γ_1 and γ_2 fail, selects the Nominal Observer

Table 2: Truth table

logic has been implemented to solve this problem. If one of the servo-motor inputs fails, the threshold is divided by two. This is done automatically once the fault is detected. The function *UpdateModel.m* changes the value of the block ActFault *(*see appendix B).

The others two innovations, coming from the two fault case observers, are compared with each other. The lower innovation indicates the better observer and therefore the most probably fault case. The truth table, used only if the first innovation is bigger than the threshold, is shown in the table 2.

If the nominal observer fails and so does one of the others, for instance, the fault observer 1, that means that γ_2 is wrong, so the logic selects the fault observer 2.

If there is more than one faulty sensor it selects the nominal observer because neither observer is adequate for this case.

Non Return Logic. Because the innovation may vary over time, the fault case is not changed any more after it has been identified once. This is implemented by a non-return logic (figure 34).

This logic has three switches. The main switch choose between the current output of the selection logic and the previous state of the non return logic. It starts selecting the selection output and changes when the input changes. This is done using a comparison between the output of the previous logic and '1'. When the input is different from '1', the *non-equal* block output changes into '1', the secondary switch changes an select an input which is always one, which ensure that the main switch will select from now on the previous output. Thus, once a fault is detected, the system remains using the faulty case actuator. Several memory blocks have be introduced in order to avoid algebraic loops that cannot be simulated in Simulink. Their initial values

are '0' except from the last one which is '1' to start selecting the nominal observer.

A reset input is included. Then, the secondary switch input which is usually '1' could be changed into '0' using the reset switch. Thus, the *Non Return Logic* is reseted.

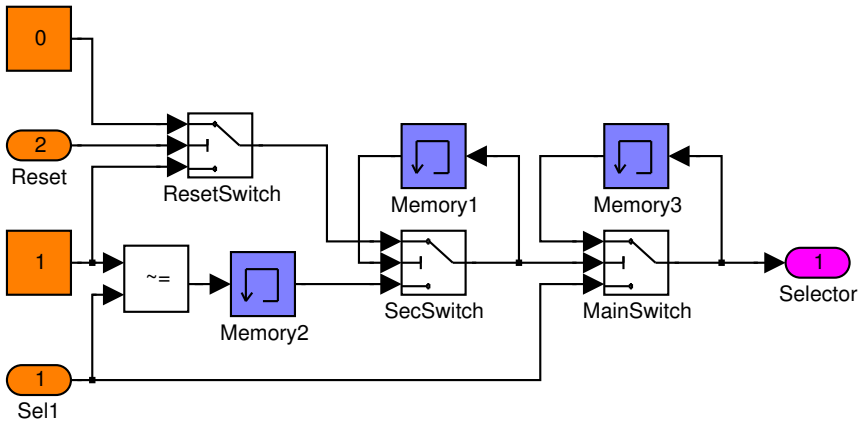


Figure 34: Non return logic

Other controls Two control have been included in the main model (see figure 30). The *enable* button displays a dialog where is possible to choose between enable the detection logic or not. If it is disable, the nominal observer is used in every case. The *reset* button resets the non-return logic by double clicking. Both are implemented as masked subsystems. Besides, a reset function is opened by clicking the reset button (see appendix B, subsection B.3.3on page 112).

6.5 Rank Deficiency

Like in the actuator fault reconfiguration, there is a problem—when calculating the feedback gain—in dealing with the singularities introduce by the fault occurs. It is again solved reducing the process model.

First, the output matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ have to be reduced to matrices of full rank, called $\mathbf{C}_1'$ or $\mathbf{C}_2'$ depending on the fault case. The reduced matrix is

calculated by multiplying $\mathbf{C}_1$ ($\mathbf{C}_2$) with a transformation matrix $\mathbf{T}_{y1}$ ($\mathbf{T}_{y2}$) which has as many rows as $\mathbf{C}$ and as many columns as remaining sensors. This matrix is calculated using the function *ReduceU.m*. Since this function was developed to eliminates zero columns, the $\mathbf{C}_1$ ($\mathbf{C}_2$) matrix is transposed before calling it. Then, the transformation is applied:

$$\mathbf{C}_1' = \mathbf{T}_{y1}^T \mathbf{C}_1 \quad (37)$$

$$\mathbf{C}_2' = \mathbf{T}_{y2}^T \mathbf{C}_2 \quad (38)$$

Thus, the elements corresponding to the faulty sensor have been eliminated. To solve the observer design problem it is also necessary to reduce the output cost matrix $\mathbf{R}_o$ accordingly:

$$\mathbf{R}_{o1}' = \mathbf{T}_{y1}^T \mathbf{R}_o \mathbf{T}_{y1} \quad (39)$$

$$\mathbf{R}_{o2}' = \mathbf{T}_{y2}^T \mathbf{R}_o \mathbf{T}_{y2} \quad (40)$$

Finally, the *lqr.m* function is called and observer design problem is solved. It returns the optimal gain matrix $\mathbf{L}_1$ ($\mathbf{L}_2$) for the observer.

6.6 Reconfiguration algorithm

The following algorithm has been used for the reconfiguration of sensors faults. Note that the first three steps are performed off-line at design time, while the last two steps are carried out online while the plant is operating.

1. A nominal observer is designed (by calling *observer.m,* see subsection B.2.3).
2. Fault case observers are design using a reduction of the nominal observer design problem (by calling *ReduceU.m*).
3. A sensor fault diagnosis logic is implemented (see figure 33).
4. A sensor fault is detected and identified by the diagnosis logic.

5. The corresponding fault case observer is selected to produce the state estimate used by the state feedback controller (see figure 30).

The result of this algorithm is a reconfigured control structure that stabilized the plant despite the detected sensor fault.

6.7 Simulations

Sensor γ_1 Failed. These simulations are shown in the figures 35 and 36. The first one shows the system behavior before the selection logic was implemented (only the nominal observer is used). In the second plot the selection logic switches to the correct fault case observer.

The first graphic shows that γ_1 reads zero during the whole experiment. This leads to heavy oscillations and it takes about 10 seconds to reach the reference values.

The performance is clearly improved in the second simulation. The symmetry of the process is broken, so that actuator u_1 and u_2 and the variables γ_1 and γ_2 are no longer identical. The observation is a bit problematic, leading to some oscillations and high actuator usage. However, the system behavior is still acceptable. The attack angle values do not exceed 20°. β has a significant overshoot of 10° and a steady error of 6°.

In the figure 37, it is shown how the fault detection unit changes the fault case value once the fault is detected. The differences between the nominal and the fault case observer are clear, they mainly differ in the faulty sensor variable and its derivative.

The innovation of each observer is shown in the figure 38. Thus, the innovation of the nominal observer is bigger than the threshold. Besides, the innovation of the Fault Observer 1 is lower than the one coming from the Fault Observer 2. Therefore, the Fault Observer 1 is selected.

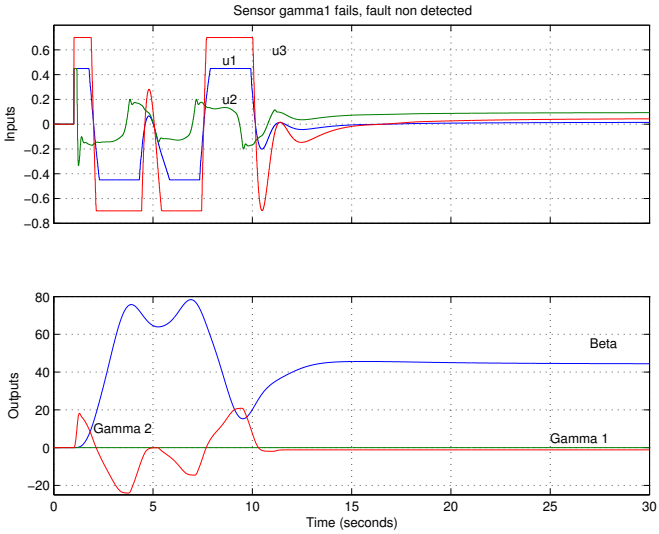


Figure 35: Sensor γ_1 failed

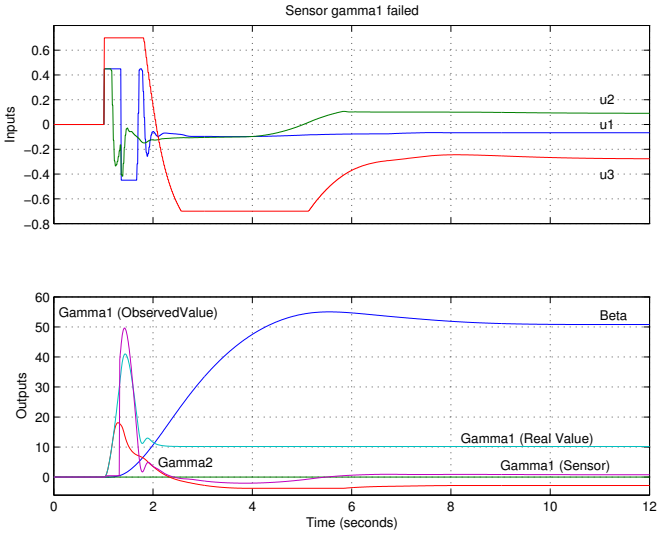


Figure 36: Sensor γ_1 failed, fault is detected

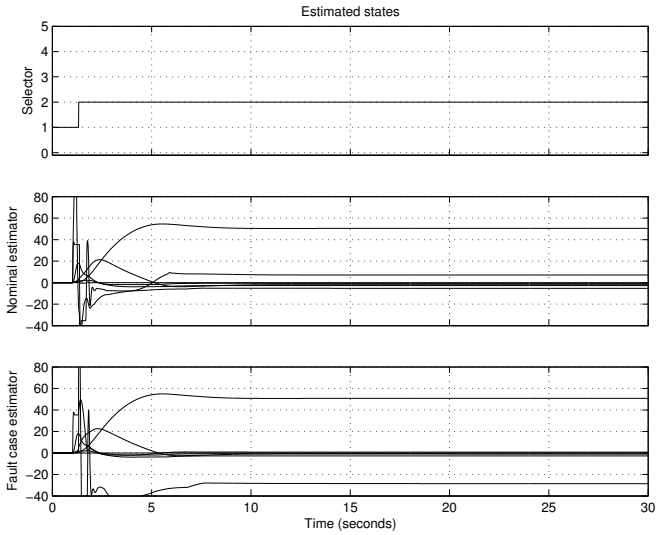


Figure 37: Sensor γ_1 failed, observed states

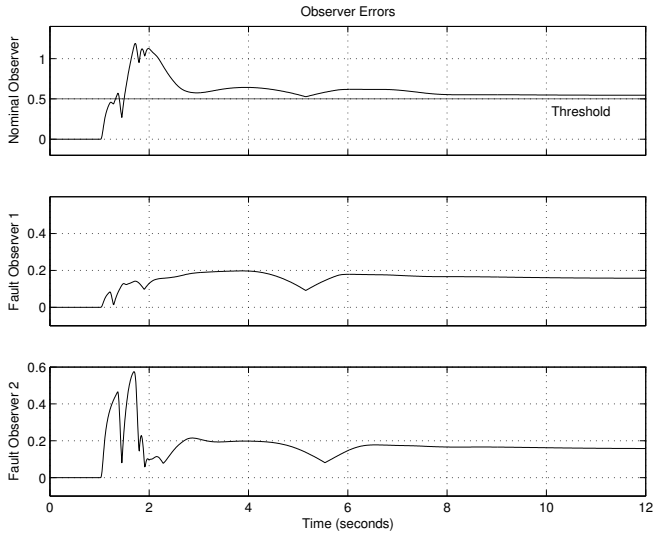


Figure 38: Sensor γ_1 failed, observer errors

6.8 Results

A bank of dedicated observers and a fault detection and isolation logic have been developed. The fault observers gain matrices have been obtained using the LQG design method, once the range deficiency problem was solved. The final gain matrix $\mathbf{L}_1$ — $\mathbf{L}_2$ is analogous — is shown in the equation (41). Obviously, this matrix don't have to be expanded since it is used in a observer with less inputs (but the same outputs). In this case, there is no need to down load the feedback matrices on line since the whole bank of observers is running at any time.

$$\mathbf{L}_1 = \begin{pmatrix} 0.0562 & 0 \\ -0.0001 & 0.8252 \\ 424.71 & 0.0383 \\ 0.0027 & 78.6751 \\ 0.0337 & 0 \\ 75.8540 & 0.5973 \\ 12.3575 & 0.0027 \end{pmatrix} \tag{41}$$

6.9 Conclusions

The use of an observer is a natural and successful way to reconfigure sensors faults. It has been shown how the bank of observers can be used to reconfigure the compensator in case of a fault in a system sensor.

On the other hand, the approach known as "dedicated observer scheme" can also be used for fault detection. This allows to reconfigure the system without manual intervention even in the fault detection process. The reconfigured system is stable and shows moderate performance.

However, the fault is only detected when the system is in motion since a stationary situation does not provides enough information.

Finally, it must be notice that these are simulation results. As it is shown in the $\mathbf{L}_1$ matrix, the dependency of γ_1 in the sensor β has been increased visibly. Hence, in a fault case, noise in β sensor may affect visibly the system behavior. This fact will be considered later increasing, in case it is necessary, β cost function.

7 Multiple faults

7.1 Introduction

Without a detailed theoretical foundation a few simulations with multiple and combined actuator and sensor fault were tried to explore the system reconfiguration possibilities in difficult cases. As it is shown in the figures 39 to 42, even multiple fault cases are reconfigured correctly.

7.2 Simulations

Case 1: Actuators u_1 and u_2 Failed In this case, both main rotors are not used. The reconfiguration of the remaining actuator allow to achieve a performance really closed to the nominal case (see figure 39)

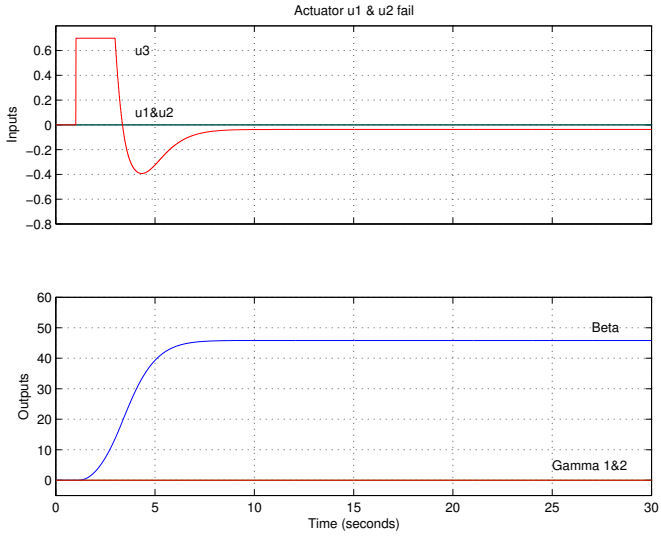


Figure 39: Actuator u_1 and u_2 failed

Case 2: Actuators u_1 and u_3 Failed In this case the actuator u_2 is over-used and causes that γ_2 reaches values quite high (40°). The system is much more sensitive to the loss of the actuator u_3. Using only the main

rotors implies higher changes in the system dynamic which cause a slight overshoot. Nevertheless, the performance is good enough and the plant reach the reference in about 12 seconds (see figure 40).

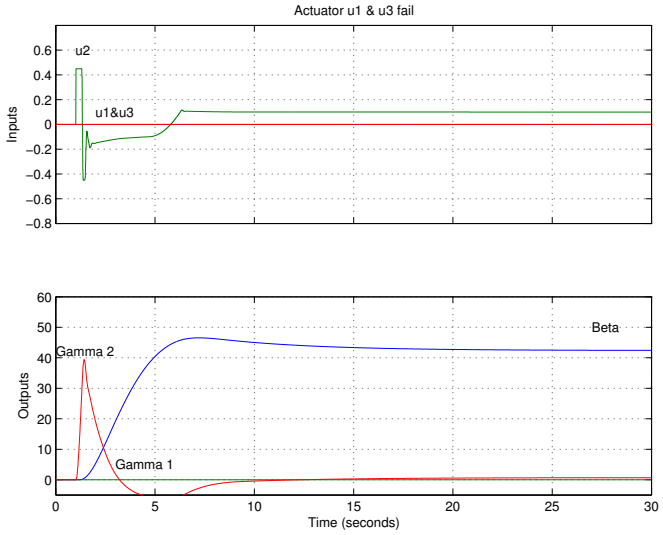


Figure 40: Actuator u_1 and u_3 failed

Case 3: Actuators u_1 and Sensor γ_2 Failed This case is more problematic since the actuator u_1 is not used and u_2 is used based on the observation of γ_2, which sensor has also failed. Although the system is stable, it shows moderate performances with and overshoot of $15°$ and a steady-state error of $5°$ (see figure 41).

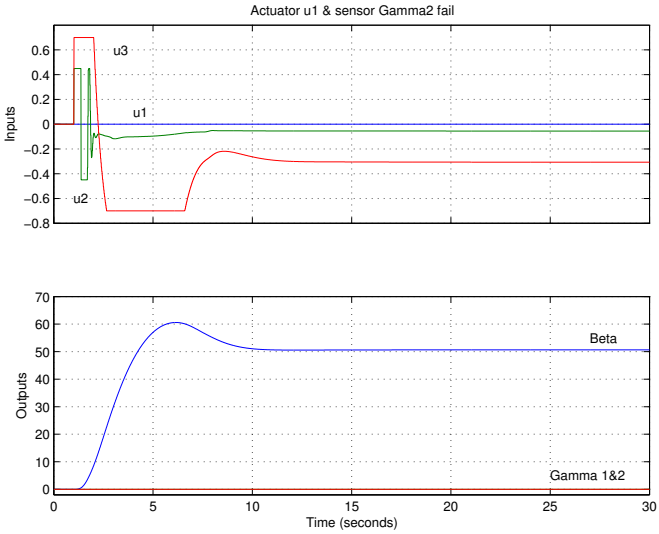


Figure 41: Actuator u_1 and sensor γ_2 failed

Case 4: Actuator u_3 and Sensor γ_2 Failed In this case the system shows the worst behavior. The system sensibility on the actuator u_3 and using the actuator u_2 when the sensor γ_2 has failed leads to slight oscillations. The systems needs 30 seconds to stabilize itself (see figure 42).

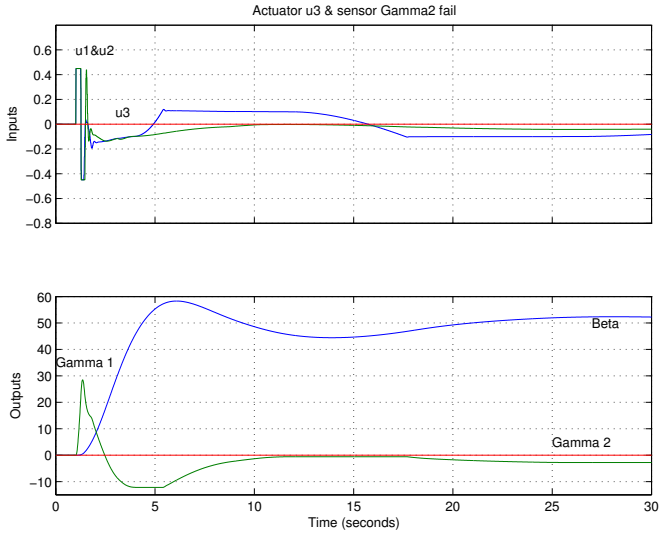


Figure 42: Actuator u_3 and sensor γ_2 failed

7.3 Simulation Environment

A *Matlab* script *flightmod.m* is set up as a front-end for reconfiguration experiments. It calls the scripts *modeling.m*, *controller.m* and *observer.m* in the right order and loads the se*lection logic* parameters (filter cut frequency and selector threshold). It also changes the simulation model (faulty amplifiers values) to the appropriate fault case. Then, it shows a menu that allows to simulate both actuator and sensor faults and to trigger the actuator fault reconfiguration (because actuator faults cannot be detected automatically at the moment). Each time an actuator amplifier value is modified, it is necessary to recall *modeling.m,* since the **B** matrix changes, and *controller.m* to recalculate the **K** matrix. The code is in the appendix B, subsection B.2.4 on page 107 and the menu is shown in the figure 43.

7.4 Conclusions

It has been demonstrated how the redesign of the state feedback controller can be used to reconfigure the compensator in case of a fault in a system

Figure 43: Simulation menu

actuator. The LQG method used for the design of the nominal controller can also be applied to the fault cases, using the same cost functions.

On the other hand, the use of dedicated observers is a natural and successful way to reconfigure sensors faults. The observers can also be used for fault detection at the same time.

This allows to reconfigure the system without manual intervention. The developed algorithm are efficient and automatic. The reconfigured system is stable and shows good to moderate performance depending on the fault case. Rank deficiency problems which occur during the redesign have been solved. The system changes are kept to minimum, reloading the controller gain in case of an actuator fault and selected the best observer in case of a sensor fault automatically.

The system can be reconfigured even in case of multiple or combined faults.

8 Plant Experiments

8.1 Introduction

Once the reconfigurable control loop have been simulated successfully, it must be tested on the plant. For this purpose, a new *Simulink* file has been developed. This model includes the reconfigurable control loop previously developed and the necessary interface to control and acquire data from the experiment (see figure 44). There are several changes in the control loop in order to improve the plant real behavior. Deviations between the simulated behavior and the real one appear due to unmodeled noise and non-linearities. Finally, an interface plant-operator has been developed to make the test process easier (see subsection 8.4).

The test is based on a step function which jumps from $\beta = 0°$ to $45°$. Thus, all sensor and actuators are concerned.

8.2 Blocks

8.2.1 Control Loop

The compensator and diagnosis logic blocks are used without any further changes (see section 4 on page 40 and subsection 6.4 on page 63 respectively). However, to avoid deviations from the desired behavior due to non linearities, some new blocks have been implemented. These blocks are commented in the next two paragraphs. On the other hand, the delay block has been removed since it was modeling a plant behavior.

Dead Zones This block simulates the actuators behavior. It is included before the observer bank so that its inputs are more accurate. This subsystem is shown in the figure 45.

Anti-Loose A potentiometer is used to read the main rotors attack angle. It works engaging a toothed wheel with another gear which is fixed to the rotor axis. There is a loose between the potentiometer teeth. When the gear

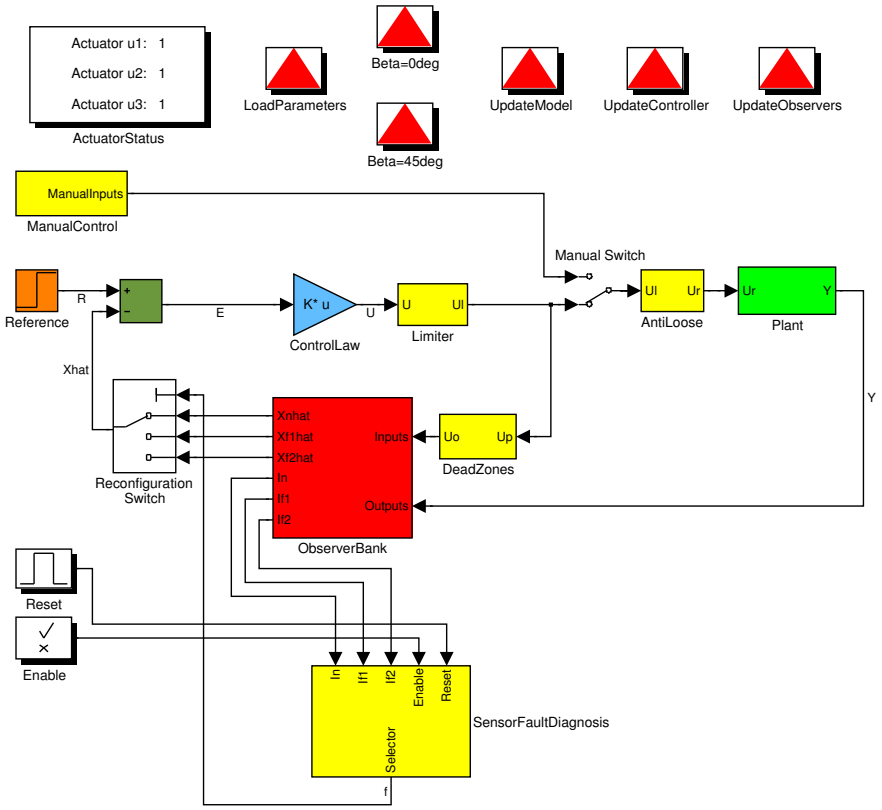


Figure 44: Plant

changes the movement direction, the sensor is reading the same position during a short time. This fact causes an oscillation around the desired position. The block shown in the figure 46 is implemented in order to avoid this effect. It has a anti-loose system for both servo-motor actuators. They are shown in detail in the figure 47. Both of them, divide the input signal into high frequency and low frequency paths. The cut frequency is set experimentally at 1 Hz. Then, a backlash block process both signals. It works as follows: the signal is kept slightly lower than the input; when the input change its sense, the processed signal is kept unchanged until the input pass through a dead-band width. Obviously, an steady state error is introduced. The low frequency path is implemented to minimize this fact. Its dead-band width is lower that the high frequency path one. Thus, when the system is around the final position, the oscillations become lower and the steady state error is

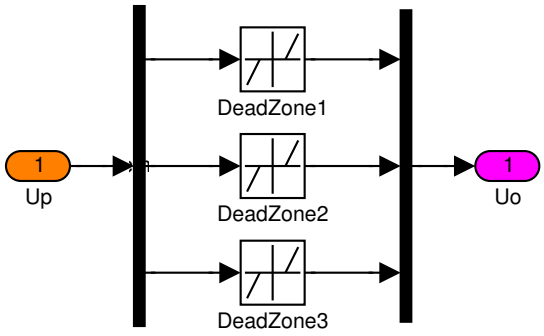


Figure 45: Dead zones

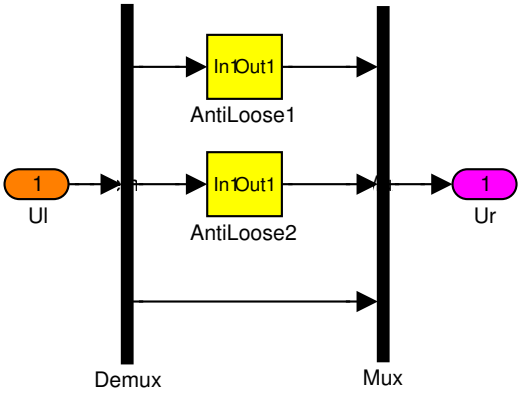


Figure 46: Anti-Loose block

reduced. The high frequency dead-band width is 0.2. Since both main rotors behave differently, so are the low frequency path dead-band width. They are set at 0.1 and 0.12 respectively.

8.2.2 Plant

The *plant* block shown in the figure 48 is the interface which connect the Target PC ports to the plant. The model is similar to the one developed to carry out the parameters identification (see subsection 2.4). It has been adapted to an automatic control removing manual inputs and slider gains.

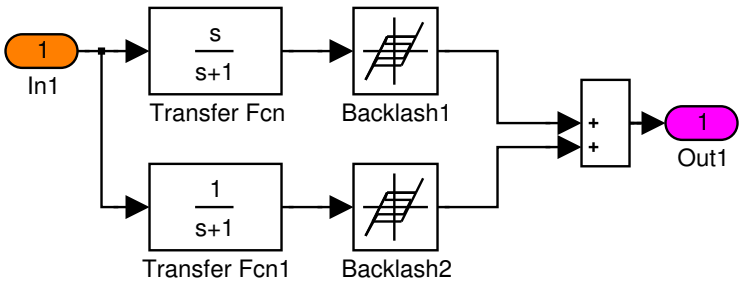


Figure 47: Anti-Loose system

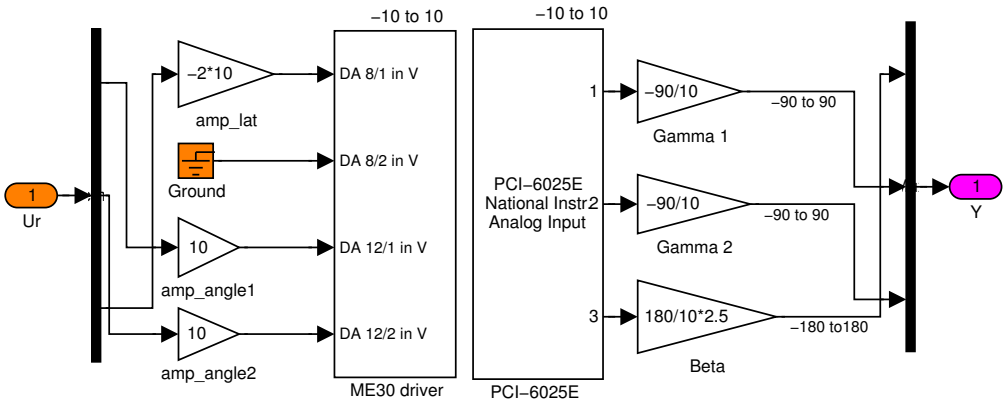


Figure 48: Plant interface

8.2.3 Manual control

There was introduce a block to move manually the actuators in order to reach the operating point before starting a test (see figure 49). This block works in a similar way as the interface we seen in the figure 7, on section 2. A manual switch (shown in the figure 44) choose between the manual control and the automatic one.

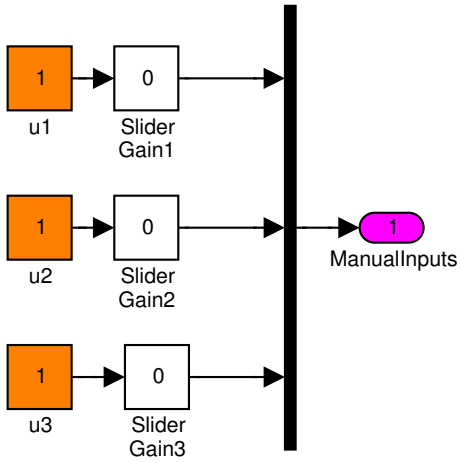


Figure 49: Manual control

8.3 Experiments

Nominal case The controlled plant is tested using as reference a function which jumps from $\beta = 0°$ to $45°$.

The first test was done without simulating any fault. The anti-loose block wasn't designed yet. The controller and observer parameters where the ones obtained after the simulations. Thus, it is possible to analyze the real behavior of the nominal control loop designed using *Simulink*. Due to noise and others non-modeled factors, the system behavior differs from the simulated one. Besides, heavy oscillations around the dead zone appear in the attack angle subsystems.

In order to improve the plant performance, several changes were introduced. These modifications are related in the following paragraphs:

- The matrix $\mathbf{K}$ and $\mathbf{L}$ were explored. The conclusions were that:

1. the dependency of $\dot{\gamma}_1$ and $\dot{\gamma}_2$ on actuators u_1 and u_2 respectively were a bit high (20.61). Thus, both subsystem are quite sensitive. The plant behavior could be greatly modified due to the non-linearities which are non considered during the controller design process. To solve this inconvenient, the cost function for these two inputs were increased 100 from to 300.

2. The dependency of $\hat{\gamma}_1$ and $\hat{\gamma}_2$ on sensors γ_1 and γ_2 respectively were very high (78.67). This means the trust on these sensors is quite important and the result behavior may depend on noise too much. In order to reduce the reliable of both sensors, the cost function for these two output were increased from 1 to 100.

3. The dependency of $\hat{\gamma}_1$ ($\hat{\gamma}_2$) on β in a fault case was too high (424.75). Therefore, the noise could affect the plant behavior considerably. Thus, the cost function for β output was increased from 1 to 10.

- Considering that the gain of F_l due to u_3 is quite low, to make the system faster the cost function of this actuator was decreased from 100 to 50.

- The real nominal innovation value is higher than the simulated one, mainly due to the system noise. To avoid false fault detections, the sensor fault detection threshold was increased up to 2.
- Finally, the anti-loose block, as is shown in subsection 8.2.1, was designed an its parameters calculated experimentally.

As a result of these changes, the swings have disappeared. The system performances are acceptable. The rising time is about 4 seconds, it has a very little overshoot of 1° and the steady state error is 5°. Both attack angles remain lower than 20° during the whole movement. The test is shown in the figure 50.

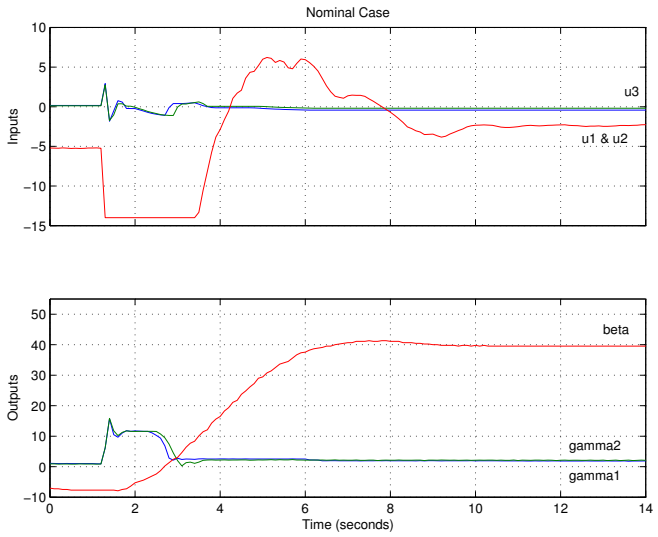


Figure 50: Non-faulty situation

Actuator fault cases As in the previous simulations, a fault in actuator u_1 (figure 51), and actuator u_3 (figure 52) were tested.

Case 1: Actuator u_1 Failed. The figure 51 shows the plant behavior after the fault has happened and detected. The system remains operational and the performance is almost as good as in the nominal case and only slightly slower. There is very little overshoot (less than 1°) and a steady state error (5°). This error occurs because a linear controller is used to control a non-linear system. This behavior was predicted during the simulations with the detailed model and is now magnified. The remaining actuators are working slightly harder and longer to stand in for the faulty actuator, but they still remain within reasonable limits and the maximum rotor attack angle is below 20°.

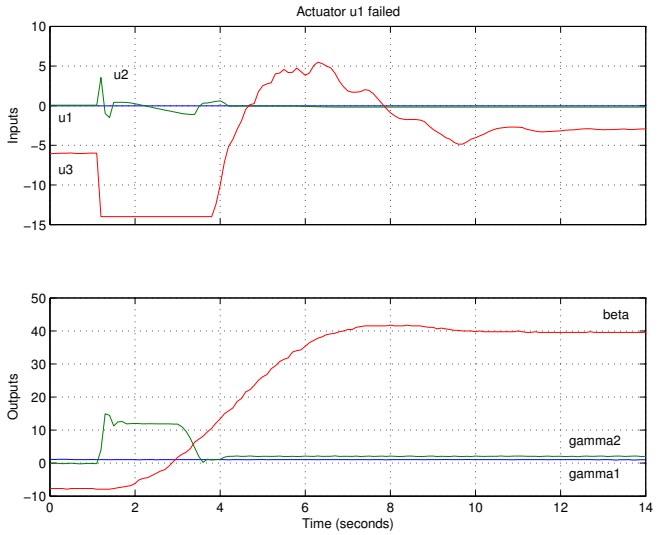


Figure 51: Actuator u_1 failed

Case 2: Actuator u_3 Failed. In this case, the system is strongly affected as it happened during the simulations. The main rotors have to be turned for a longer period of time (figure 52) than in nominal case to compensate for the loss of actuator u_3. The angle of attack reaches a maximum of 32°, which is close to the end of reasonable linear system behavior. In addiction, both

servo-motor actuators show an oscillating behavior which persists during a longer period of time. This is mainly caused by the looses between the gear teeth. Thus, when the plant depends exclusively on the main rotors, this behavior is amplified.

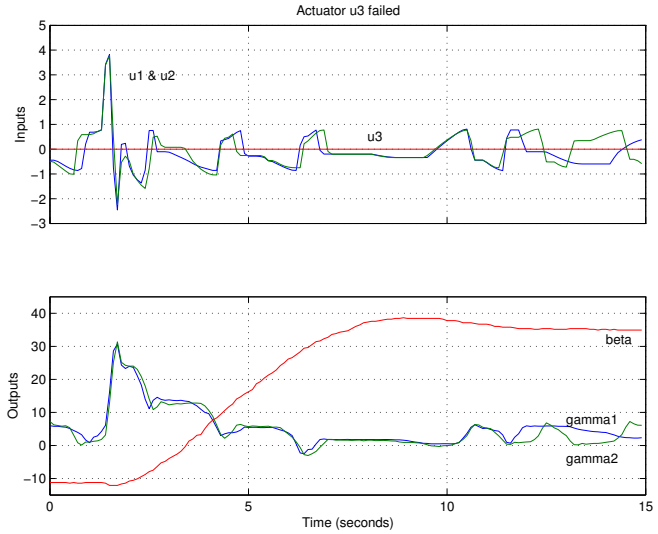


Figure 52: Actuator u_3 failed

Sensor fault case A fault in sensor γ_1 (figure 53) was tested. The system behavior after an γ_2 sensor fault is analog. Other possible faults, like the loss of sensor β or α are not tested since they render the system unobservable.

The fault is detected during the movement. Thus, there is a little 'confusion' while the transition is being done. This fact leads to some oscillations and high actuator usage. However, the system behavior is acceptable. The attack angle values do not exceed 20°. β has a little overshoot and steady state error, smaller that in the actuator fault cases.

The γ_1 signal, once the fault happened, derive to a final value around 10° (see the second plot).

As is shown in the third plot, the nominal observer innovation surpass the threshold. Both fault observer innovations are compared. Since the second

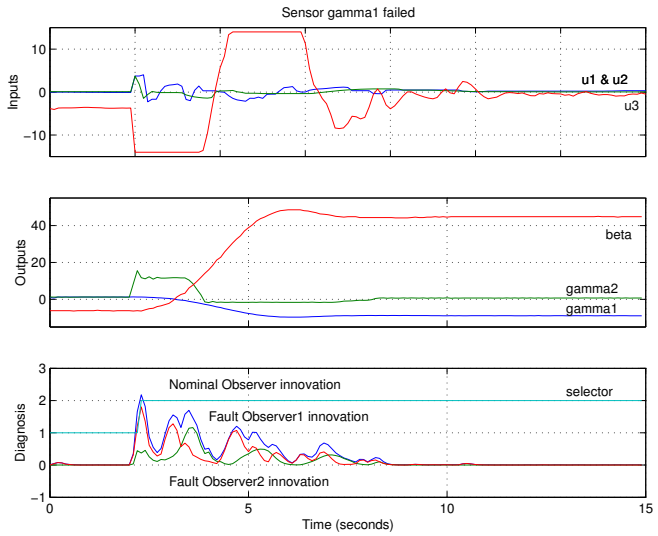


Figure 53: Sensor γ_1 failed

one is the lowest, the Fault Observer 2 is selected. The whole process is carry out at the very beginning of the movement.

Multiple actuator fault cases As in the simulations, faults on two actuators at the same time were tested exploring the system reconfiguration possibilities in difficult cases. As it is shown in the figures 54 and 55, even multiple fault cases are reconfigured correctly which means the system is stable and in operation. However, its performance is obviously reduced.

Case 1: Actuators u_1 and u_2 Failed The figure 54 shows the system behavior after faults on actuator u_1 and u_2. Thus, both main rotors are not use. The reconfiguration of the remaining actuator allow to achieve a performance closed to the nominal case, considering the steady state error and the overshoot, but obviously not as fast as it (see figure 54). The remaining actuator is working harder and longer.

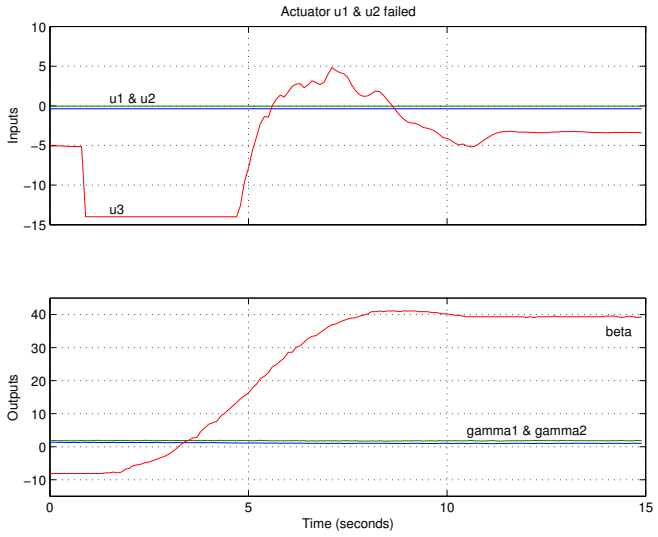


Figure 54: Actuators u_1 and u_2 failed

Case 2: Actuators u_1 and u_3 Failed In this case the actuator u_2 is overused. Therefore, γ_2 reaches values quite high (over 40°) during a short time. Nevertheless, the pitch angle, α, was fully controllable during the experiment. The system is much more sensitive to the loss of the actuator u_3. The teeth loose effect and the higher changes in the system dynamic which implies the

used of only a main rotor cause an important steady state error (more than 10°) and a slight overshoot. The experiment is shown in the figure 55.

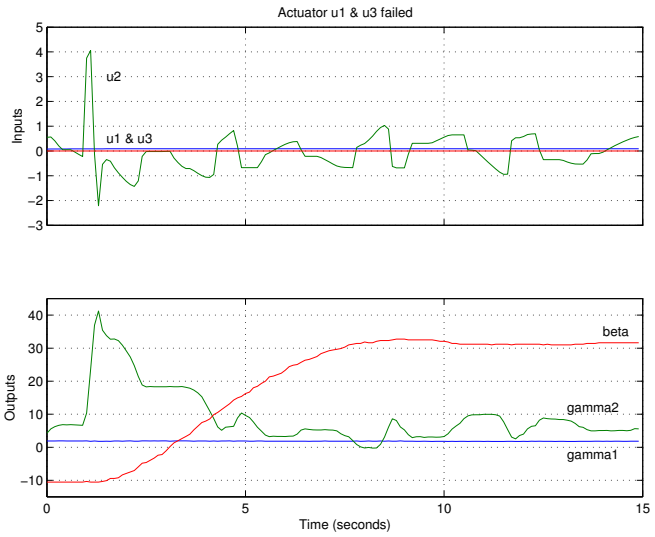


Figure 55: Actuators u_1 and u_3 failed

Combined actuator-sensor fault cases Simultaneous faults on an actuator and a sensor were tested. These are the most problematic cases since the remaining actuators are used based on the observation of the faulty sensor.

Case 1: Actuators u_2 and Sensor γ_1 Failed In this case, the actuator u_2 is not used and u_1 is used based on the observation of γ_2, which sensor has also failed. However, the system is stable, showing even better performances than in the simulation. It has a very slight overshoot and steady-state error, although the signal swings for a short time around the reference. The experiment is shown in the the figure 56.

The γ_1 signal, once the fault happened, derive to a final value around 10°. γ_2 is fixed at 0° since the associated actuator, u_2, is at fault (see the second plot). As is shown in the third plot, the threshold is reduced, due to one of the servo-motor actuators has failed. Thus, the fault is detected.

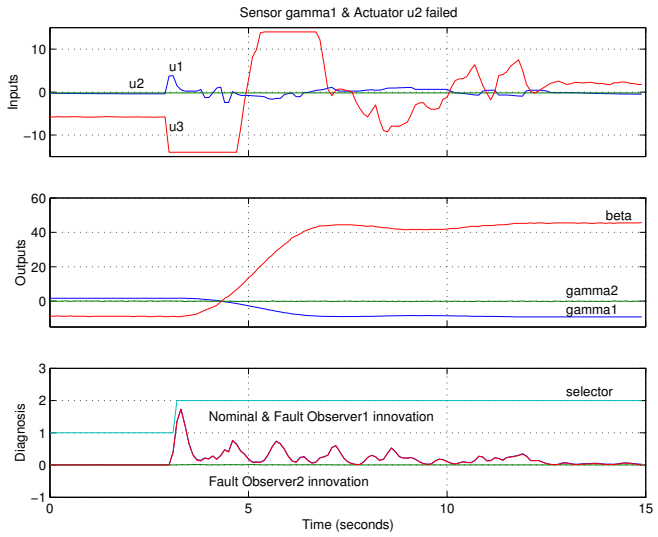


Figure 56: Sensor γ_1 and Actuator u_2 failed

Case 2: Actuator u_3 and Sensor γ_2 Failed In this case the system shows an oscillating behavior in both remaining actuators. This is caused due to the system sensibility on the actuator u_3 and the used of u_2 when γ_2 has failed. However this behavior do not appear in the system output, β, since the oscillation amplitude is not enough high to affect the whole system. Therefore, β reaches the reference with a reasonable low overshoot and steady state error in less than 4 seconds. The figure 57 shows the experiment.

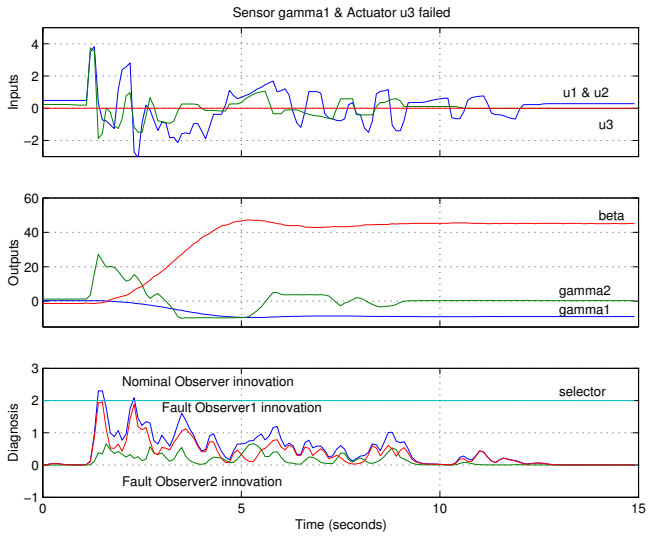


Figure 57: Sensor γ_1 and Actuator u_3 failed

8.4 Test environment

Several buttons and switches have been added to main Simulink file to make the whole testing process easier (see figure 44 on page 79).

The actuator and sensor faults can be caused by disconnecting the wire in question. The sensor fault diagnosis logic detects and isolate the fault. However, it is necessary to detect the actuator faults since the associated FDI module is not implemented. A masked subsystem allows, by double clicking, to do this task. Thus, once an actuator fault is caused, the require menu option must be selected. This information is used by other functions to reconfigure the control loop.

The *LoadParameter* button opens the system file model, loads all the system parameters and modifies the fault amplifier values and the detection threshold in case it is necessary. This process is done on line by the function *loadparameters.m* which is called once the button is double clicked.

The buttons *Beta=0deg* and *Beta=45deg* modifies on line the system reference.

The buttons *UpdateModel*, *UpdateController* and *UpdateObserver* remodel the system, recalculate the controller and the observer and down load the

new parameters into the target PC on line. They are also mask subsystem which call the related functions when the button in question is double clicked.

The function codes is place in the subsection B.4, on page 112.

8.5 Results

The control loop has been modified improving its behavior. There were implemented an anti-loose block and a dead zones block. There was also implemented a manual control for both rotor attack angle. A basic operator interface has been developed.

The cost functions have been adjusted to a real case where noise and non linearities are present. The equations 42, 43 and 44 show the final values of the controller and nominal and fault observer gain matrices.

$$\mathbf{K} = \begin{pmatrix} 14.31 & 0.224 & 0.059 & 0.001 & 0.278 & 0.024 & 0.018 \\ 0.234 & 14.31 & 0.001 & 0.059 & 0.278 & 0.024 & 0.018 \\ 1.669 & 1.669 & 0.011 & 0.011 & 2.110 & 0.180 & 0.128 \end{pmatrix} \quad (42)$$

$$\mathbf{L} = \begin{pmatrix} -0.001 & 0.055 & 0 \\ -0.001 & 0 & 0.055 \\ 0.146 & 20.29 & 0 \\ 0.146 & 0 & 20.29 \\ 0.016 & 0 & 0 \\ 4.522 & 0.589 & 0.589 \\ 3.023 & 0.015 & 0.015 \end{pmatrix} \quad (43)$$

$$\mathbf{L}_1 = \begin{pmatrix} 0.008 & 0 \\ 0 & 0.055 \\ 138.5 & 0.008 \\ 0.155 & 20.29 \\ 0.006 & 0 \\ 35.62 & 0.593 \\ 8.446 & 0.016 \end{pmatrix} \quad (44)$$

The fault detection threshold has been also modified due to the noise which makes the innovation values higher. Its final value is set at 2. In case of a servo-motor actuator fault, this value is reduced to 1.

Experiments for all possible fault cases have been carried out. Multiple and combined fault cases which implies two actuators or one actuator and one sensor have been also tested.

8.6 Conclusions

The tests show that the redesign of the state feedback controller and the use of a Dedicated Observer Scheme is a successful way to reconfigure the control loop. This approach keeps the system controllable and with acceptable performances even in multiple an combined fault cases. Thus, no predesigned controllers and no manual intervention is necessary — although, since fault detection was not a key focus of this project, an FDI module for the actuator fault cases has not been implemented –. The changes in the control loop are kept to minimum.

9 Summary and outlook

The first part of this project was the plant modeling and identification. Once a linear model was implemented, its parameters were identified during several experiments with the real system. A linear state-space model was derived and used to design the controller. In addition, nonlinear effects like saturation and friction have been identified to refine the simulation model.

The next step was to implement a nominal control structure and calculate a proportional feedback controller and an observer. The technique used to find the controller values was the well known LQG method. The controller was calculated by trying different weights and simulating the system behavior. A detailed nonlinear model has been used for the simulations. It includes the delay introduced by the digital/analog interface.

After this, an algorithm for the reconfiguration of this control structure was implemented. Several routines were programmed to deal with the singularities of the fault case model, because they break the implementation of the controller design methods.

A bank of observer was designed, each one dedicated to a fault case. A selection logic was build that switches to the correct observer depending on the actual fault case. Simulations were done in order to ensure that the observer and the fault detection logic work properly.

Once the design process was finished and the simulations of both nominal and reconfigured control loop showed acceptable performance, several experiments were carry out on the plant in order to verify experimentally the design. The control loop has been modified in order to accomplish the test successfully. A basic operator interface has been implemented.

In conclusion, the approach was shown to be successful. The system is kept in operation which reasonable performance for every fault case tested. The whole process of observer selection and controller redesign is carried out completely automatically. Rank deficiency problems which occur during the redesign have been solved. The changes in the control loop are kept to minimum. They are restricted to the part affected by the fault: a selection of the best observer or a recalculation of the control gain matrix using the same weights. Practical problems due to looses, noise and other non linearities have been solve successfully. Fault detection was not a key focus of this project, but a simple sensor fault detection was implemented.

The study on this subject can be extended to cover different control structures. This includes a reconfiguration approach where the control structure is changed even less than in this project. The aim would be to extend the control loop including a reconfiguration block. This block would work as an interface between the process and the controller. Thus, the behavior would be as closed as possible to the nominal one, allowing to keep the nominal controller in case of fault.

The control of the selected plant can be improve modeling the behavior of the main rotors and their controller system. Thus, it would be possible to control the whole system, reaching different pitch angles. In that case, the system could be controlled and reconfigured modifying the main rotor speed. However, neither the the main rotor actuators, u_1 or u_2, nor the pitch angle sensor, α, could fail since, in this case, the system wouldn't be controllable or observable respectively. The implementation of the FDI module for actuator fault cases would complete the control structure.

Finally, the plant hardware can be improve in different ways. The potentiometers used to measure the angles can be substituted by digital encoders. This change would solve many problems caused due to the moderate precision, the significant amount of noise, the strong nonlinearities at the end of the measurement range and the loose introduced by the current sensors. The main rotors wires can be replaced with brushes. Thus, the friction forces, which change from experiment to experiment depending on the wire positions, would be reduced. To avoid using different software interfaces and connection boards for both inputs and outputs, a panel with five outputs and four inputs is required.

A Symbol List

Symbol	Description	Symbol	Description
α	pitch angle	$\mathbf{u}$	system input vector
β	rotation angle	$\mathbf{x}$	system state vector
γ	generic rotor attack angle	$\hat{\mathbf{x}}$	system observed state vector
γ_1,γ_2	specific rotor attack angles	$\mathbf{y}$	system output vector
θ	generic angle	$\mathbf{A}$	generic state-state matrix
τ	generic time constant	$\mathbf{A}'$	reduced state-state matrix
τ_1, τ_2	attack angle subsystem time constant	$\mathbf{A}_F$	fault case state-state matrix
τ_3	lateral rotor time constant	$\mathbf{A}_N$	nominal case state-state matrix
ω	generic angular speed	$\mathbf{B}$	generic input-state matrix
a	system angular acceleration	$\mathbf{B}'$	reduced input-state matrix
d	generic dead zone	$\mathbf{B}_F$	fault case input-state matrix
F	generic force	$\mathbf{B}_N$	nominal input-state matrix
F_1, F_2	force applied by the main rotors	$\mathbf{C}$	generic state-output matrix
F_l	force applied by the lateral rotors	$\mathbf{C}_1, \mathbf{C}_2$	reduced state-output matrices
J	generic momentum constant	$\mathbf{C}_F$	fault case state-output matrix
J_m	main rotor momentum constant	$\mathbf{C}_N$	nominal state-output matrix
J_l	lateral rotor momentum constant	$\mathbf{D}$	generic input-output matrix
F_b	bearing friction	$\mathbf{K}$	controller feed back gain
F_s	speed proportional friction	$\mathbf{K}'$	reduced controller feed back gain
k	generic gain	$\mathbf{L}$	observer feed back gain
k_1,k_2	attack angle subsystem gain	$\mathbf{L}'$	reduced observer feed back gain
k_3	lateral rotor gain	$\mathbf{Q}_c$	controller state costs
t	time	$\mathbf{Q}_o$	observer states variance
u	generic actuator input	$\mathbf{R}_c$	controller input costs
u_1,u_2	servo motor inputs	$\mathbf{R}_o$	observer output variance
u_3	lateral rotor inputs	$\mathbf{T}_u$	input transformation matrix
y	generic system output	$\mathbf{T}_x$	state transformation matrix
v, w	normalized white noise	$\mathbf{T}_{y1}, \mathbf{T}_{y2}$	output transformation matrices

B Program Code

B.1 Parameters Identification

In order to identify the system parameters, there where created the functions *Datamatrix.m*. *Identify.m* and *Momentum.m*. The first one, form a matrix which contain the data from the experiments made with DSpace. These data are the instant time, the input voltage and the angle position. It represent also these data graphically. The second one, identify the parameters of a second order system such as the gain and the time constant working together *Datamatrix.m*. The last one, working also together with the first function, identify the constant J which contains the information about the system mass and the axis length.

B.1.1 Datamatrix.m

```
function [data] = datamatrix (filename)

%Form a matrix with the input voltage, the angle position and
%   speed at each time in order to identify the parameters of a
%   first-order system and represents graphically the data.
%Sintaxis: [data] = datamatrix (filename)
%Inputs: name of the .mat file generated by Trace.
%Output: the matrix.


load (filename);

time = trace_x;                    % time
timeInc = time (2) - time (1); % time between samples
teta = trace_y (1, :);           % angle
rTeta = round (teta*10) / 10;     % eliminates lower decimals
input = trace_y (2, :);   % input
omega = diff (rTeta) / timeInc; % instant speed

data = [time; input; rTeta; 0 omega];
                          % introduced zero to equiparate lenghts
```

```
h = figure;
subplot (2, 1, 1);
plot (time, teta, 'r'),
title (filename),
%set (h, 'XTickLabel',{});           % switch x axis off
ylabel ('Angle (degrees)')
grid

subplot (2, 1, 2)
plot (time, input, 'g'),
xlabel ('Time (seconds)')
ylabel ('Input')
grid
zoom
```

B.1.2 Identify.m

```
function [gain, tau, deadzone]= identify (data)

%Identify the parameters of a second-order system
% Sintaxis: [gain, tau, deadzone]= identify (data)
%Input: data is the array which contain the information of the
%   experiment.
%Outputs: gain, time constant and dead zone of the system.
%Functions called: none.


timeInc = data (1,2) - data (1, 1); % time between samples
time = length (data (1,:) ); % time of the experiment:
                                    %   (time - 1) * timeInc


% Searching when changes are (between -90° and 90°)

nTurns = 0;
for i = 2 : time - 1
   if (data (3, i) > 10 & data (3, i - 1) < 10)
```

```
                % 10° due to the sensor non linearities near -90°
      nTurns = nTurns + 1;
      turnTime (nTurns) = data (1, i);
end
end


% Find when the step is given

i = 1;
input (1) = data (2, 1); % value of the input

while ((data (2, i) == input (1)) & (i < time))
   i = i + 1;
end
stepTime = data (1, i);
input (2) = data (2, i);

k = 1;
while (turnTime (k) < stepTime)
k = k + 1;
end


% Identification of the gain

time11 = turnTime (k - 2);
time12 = turnTime (k-1) - (turnTime (k-1) - turnTime (k-2)) / 10;
                    % security margin due to the non-linearities
time21 = turnTime (k);
time22 = turnTime (k + 1) - (turnTime (k + 1) - turnTime (k))/10;

teta11 = data (3, round (time11 / timeInc) + 1);
teta12 = data (3, round (time12 / timeInc) + 1);
teta21 = data (3, round (time21 / timeInc) + 1);
teta22 = data (3, round (time22 / timeInc) + 1);

omega1 = (teta12 - teta11) / (time12 - time11); % average speed
```

```
omega2 = (teta22 - teta21) / (time22 - time21);

gain = (omega2 - omega1) / (input (2) - input (1));


% Identification of the time constant

time1 = turnTime (k - 1);
time2 = turnTime (k) - (turnTime (k) - turnTime (k - 1)) / 10;

teta1 = data (3, round (time1 / timeInc) + 1);
teta2 = data (3, round (time2 / timeInc) + 1);

teta10 = teta1 - omega1 * time1;   % teta = speed*time + teta0
teta20 = teta2 - omega2 * time2;

intTime = (teta20-teta10) / (omega1-omega2); % intersection point
tau = intTime - stepTime;

% Identification of the deadzone

omega0 = omega1 - gain * input(1); % omega = gain*input + omega0
deadzone = -omega0 / gain;  % limit input with no speed


% Results

gain,tau,deadzone, omega1, omega2
```

B.1.3 Momentum.m

```
%MOMENTUM
%
%Script that calculates the effects os the axis mass and length
%Functions called: none

timeInc = data (1,2)- data(1,1);
```

```
i = round (time1/timeInc);
time2 = data (1, i + 200);
beta1 = data (3, i);
beta2 = data (3, i + 200);

J = abs (Force * 180/pi * (time2-time1)^2/2 / (beta2 - beta1))
```

B.2 Control Loop Design

The following scripts are used to control the plant. *Modeling.m* loads the system parameters, locates the variables associated with each integrator and extract the linear space-state model. Afterwards, it reorders the space-state variables getting the order with use in the design and loads the non-linear parameters. *Controller.m* calculates the linear feed back controller using the desired weights for each input and state. It also deals with actuator faults working together with *ReduceU.m*, *ReduceX.m* and *Compress.m*. *Observer.m* calculates the observer using the design costs for each output and state. It calculates also the faulty case observer working together with *Compress.m*. The script *Flightmod.m* show a menu from where you can simulate every sensor and actuator fault and view the simulations. It uses the three previous scripts.

B.2.1 Modeling.m

```
%MODELLING
%
%Script which loads the value of the different
%   system parameters, locates the state-space variables,
%   reorder them and linerise the model.
% Functions called: linmod.m
% Model used: flightmodel.mdl


%  LINEAL PARAMETERS
```

```
% Rotors angle attack

k1 = 450; % gain
tau1 = 0.12; % time constant
sd1 = 0; ed1 = 0; % dead zone (0 for the linearisation)
sat1 = 1;               % saturation level (0 for the linearisation)

% Lateral rotors

k3 = 2.8;     % gain
tau3 = 0.2; % time constant
sd3 = 0; ed3 = 0; % dead zone (0 for the linearisation)
sat3 = 1;

% System

Jm = 6.1;  % mass and axis length effects, main rotor
Jl = 10.9; % lateral rotors.
F1 = 3.6; F2 = 3.6; % force aplied by the motors %control 7:1.3
eFriction = 0;          % static friction (0 for the linearisation)
dFriction = 0.2;        % dynamic friction


% IDENTIFICATION

[sizes, x0, xs] = flightmodel;
            % sizes: vector (# integr, , # outputs, # inputs...)
                % x0: state vector
            % xs: name of each state variable (integrator)
ns = sizes (1); % number od states
no = sizes (3); % number of outputs
ni = sizes (4); % numer of inputs

iu1 = 1; iu2 = 2; iu3 = 3;       % input position
                % u1 (gamma1), u2 (gamma2), u3 (lateral rotors)
jbeta = 1; jgamma1 = 2; jgamma2 = 3; % output position

% Locates the states in the vector
```

```
pos = 1:ns; % vector to calculate the position of each state
kgamma1speed = pos * strcmp (xs, ...
    'flightmodel/FlightModel/RotorAttackAngle1/RotorAttackAngle');
kgamma2speed = pos *strcmp (xs, ...
    'flightmodel/FlightModel/RotorAttackAngle2/RotorAttackAngle');
kgamma1 = pos * strcmp (xs, ...
    'flightmodel/FlightModel/RotorAttackAngle1/Integrator');
kgamma2 = pos * strcmp (xs, ...
    'flightmodel/FlightModel/RotorAttackAngle2/Integrator');
kforcel = pos * strcmp (xs, ...
    'flightmodel/FlightModel/LateralRotors/LateralRotors');
kbetaspeed = pos * strcmp (xs, ...
    'flightmodel/FlightModel/Inertia/Integrator1');
kbeta = pos * strcmp(xs, ...
    'flightmodel/FlightModel/Inertia/Integrator2');

% T transforms standard order of states into observer order

T = zeros(7,7);
T (kgamma1speed,1) = 1;
T (kgamma2speed,2) = 1;
T (kgamma1,3) = 1;
T (kgamma2,4) = 1;
T (kforcel,5) = 1;
T (kbetaspeed,6) = 1;
T (kbeta,7) = 1;


% LINEARISATION

% Operating point for Linearisation

beta0 = 0 * pi / 180;
gamma10 = 0 * pi / 180;
gamma20 = 0 * pi / 180;

% Linearisation
```

```
x0 = zeros (1, ns);
x0 (kbeta) = beta0;
x0 (kgamma1) = gamma10;
x0 (kgamma2) = gamma20;

[A,B,C,D] = linmod ('flightmodel', x0, []);
           % plant: X° = A*X + B*U, Y = C*X


% Change the order into the one we use

B = inv (T) * B;
C = C * T;
A = inv (T) * A * T;

kgamma1speed = 1;
kgamma2speed = 2;
kgamma1 = 3;
kgamma2 = 4;
kforcel = 5;
kbetaspeed = 6;
kbeta = 7;

% NON-LINEAR PARAMETERS

sd1 = -0.1; ed1 = 0.1; % dead zone for the angle attack motors.
sat1 = 0.45;     % saturation limit
sd3 = -0.01; ed3 = 0.01;    % dead zone for the lateral rotors.
sat3 = 0.7;          % saturation limit
eFriction = 0.4;            % static friction
```

B.2.2 Controller.m

```
% CONTROLLER
%
% Script that calculates a linear state feedback controller
```

```
% dealing with actuator faults.
% Functions called: ReduceU.m, ReduceX.m


% Eliminate singularities (non-affected states: row and column)

Zx = ReduceX (A, B);    % transformation matrix
F = Zx' * A * Zx;

% Eliminate faulty columns (unconnected inputs)

Zu = ReduceU (B);       % transformation matrix
G = Zx' * B * Zu;

% state costs

Qc = zeros (1, ns);

Qc (kgamma1) = 1; %1%2
Qc (kgamma2) = 1; %1%2
Qc (kgamma1speed) = 1;
Qc (kgamma2speed) = 1;
Qc (kforcel) = 1;
Qc (kbetaspeed) = 1; %1%10
Qc (kbeta) = 1;

Qc = diag (Qc);
Qc = Zx' * Qc * Zx;

% control costs: u1, u2, u3

R = zeros (1, ni);

R (iu1) = 100; %100%1000;
R (iu2) = 100; %100%1000;
R (iu3) = 100; %100

R = diag (R);
```

```
R = Zu'*R*Zu;

% optimal gain matrix: state feed-back law U = -K*X

K = lqr (F, G, Qc, R);
K = pinv (Zu)' * K * pinv (Zx);
```

B.2.3 Observer.m

```
%OBSERVER
%
%Script that calculates a standard observer and as faulty ones
%   as possible sensor faults using LQR method.
%Functions called: lqr.m.


% state costs

Qe = zeros(1, ns);

Qe(kgamma1speed) = 1;
Qe(kgamma2speed) = 1;
Qe(kgamma1) = 1;
Qe(kgamma2) = 1;
Qe(kforcel) = 1;
Qe(kbetaspeed) = 1;
Qe(kbeta) = 1;

Qe = diag (Qe);

% output costs: beta, gamma1 and gamma 2

S = zeros(1, no);

S (jbeta) = 1;    %1; %10   % first values for simulations
S (jgamma1) = 1; %1; %100   % second ones for real plant
S (jgamma2) = 1; %1; %100
```

```
Sn = diag (S);

% optimal gain matrix: prediction law (X^)° = AX^ + BU +L(Y-HX^)

L = (lqr (A', C', Qe, Sn))';


% Gamma 1 fault case observer

H1 = C;
H1 (jgamma1, kgamma1) = 0;

Zy1 = ReduceU (H1');    % transformation matrix
H1 = Zy1' * H1;         % eliminate the faulty row (faulty sensor)
Sf1 = Zy1' * Sn * Zy1;  % reduced output costs

Lf1 = (lqr (A', H1', Qe, Sf1))';  % optimal gain matrix


% Gamma 2 fault case observer

H2 = C;
H2 (jgamma2, kgamma2) = 0;

Zy2 = ReduceU (H2');    % transformation matrix
H2 = Zy2' * H2;         % eliminate the faulty row (faulty sensor)
Sf2 = Zy2' * Sn * Zy2;  % reduced output costs

Lf2 = (lqr (A', H2', Qe, Sf2))';  % optimal gain matrix
```

B.2.4 Flightmod.m

```
%FLIGHTMODEL
%
%Script that model the plant and calculates both the controller
%   and the estimator for nominal and faulty situation. It shows a menu
```

```
%   from were it is possible to simulate faults and see the simulations
%Scripts called: modelling, controller, estimator

flightfault

fAct1 = 1; % zero when loose an actuator
fAct2 = 1;
fAct3 = 1;

fSen1 = 1; % zero when loose a sensor
fSen2 = 1;
fSen3 = 1;

modelling
controller
estimator

f = 10;           % filter cut frequency
fault = 0.5;        % threshold for the estimator selection

done = 0 ;

while done == 0,

    tit = 'RECONFIGURATION CONTROL';
    o1 = 'Actuator u1 Fails                ';
    o2 = 'Actuator u2 Fails                ';
    o3 = 'Actuator u3 Fails                ';
    o4 = 'Sensor Gamma1 Fails              ';
    o5 = 'Sensor Gamma2 Fails              ';
    o6 = 'Actuator u1 fails, fault detected';
    o7 = 'Actuator u2 fails, fault detected';
    o8 = 'Actuator u3 fails, fault detected';
    o9 = 'Nominal behaviour                ';
    o10 = 'View simulations                ';
    o11 = 'Quit';
    choice = menu (tit, o1, o2, o3, o4, o5, o6, o7, o8, o9, o10, o11);
```

```
if choice == 0 | choice == 11                                   % quit
 done = 1 ;

elseif choice == 1
    fAct1 = 0;
    modelling;

elseif choice == 2
    fAct2 = 0;
    modelling;

elseif choice == 3
    fAct3 = 0;
    modelling;

elseif choice == 4
    fSen2 = 0;

elseif choice == 5
    fSen3 = 0;

elseif choice == 6
    fAct1 = 0;
    modelling;
    controller;

elseif choice == 7
    fAct2 = 0;
    modelling;
    controller;

elseif choice == 8
    fAct3 = 0;
    modelling;
    controller;

elseif choice == 9
    fAct1 = 1;
```

```
        fAct2 = 1;
        fAct3 = 1;
        fSen1 = 1;
        fSen2 = 1;
        fSen3 = 1;
        modelling
        controller

    elseif choice == 10

        disp('Write "return" to return to digraph editor') ;
        keyboard ;

    end

    %set_param ('plant/Control Law', 'Gain', 'K')

end
```

B.3 Reconfiguration After Fault Detection

There were implemented three functions to deal with the problems which appear when a fault happens. The first one, *ReduceX.m* calculates a transformation matrix to eliminate non controllable states from matrices **A** and **B** when an actuator fault occurs. On the other hand, *ReduceU.m* calculates the transformation matrix which eliminates zero columns from the matrix **B**. Finally, *ResetFunction.m,* working together with the Reset block, provides a impulse which reset the Diagnosis Logic.

B.3.1 ReduceX.m

```
function Zx = ReduceX (A, B)

%Calculates a transformation matrix to eliminate non controlable
%   states. Thus, Areduced = Zx' * A * Zx, Breduced = Zx * B
%Sintaxis: Zx = ReduceX (A, B)
%Inputs: the matrices to transform, A, B
```

```
%Outputs: the transformation matrix, Zx
%Funtion called: none


Sings = sum (ctrb (A,B)') == 0;
nsing = sum (Sings);            % number of singularities

[ns, ns] = size (A);            % number of states

k = 1;
Zx = zeros (ns, ns - nsing);    % tranformation matrix
for i = 1:ns
   if (Sings (i) == 0)          % where there is no fault
      Zx (i, k)= 1;             % keep the column
      k = k + 1;
   end
end
```

B.3.2 ReduceU.m

```
function Zu = ReduceU (B)

%Calculates a transformation matrix to eliminate zero columns
%   Thus, Breduced = B * Zu;
%Sintaxis: Zu = ReduceU (B)
%Inputs: the matrix to transform, B
%Outputs: the transformation matrix, Zu
%Funtions called: none


Faults = (sum (B) == 0);    % vector, 1 where a zero column is
nf = sum (Faults);          % number of faults
[ns, ni] = size (B);        % number of states and outputs

k = 1;
Zu = zeros (ni, ni - nf);   % tranformation matrix
for i = 1:ni
```

```
   if (Faults (i) == 0)     % where there is no fault
      Zu (i, k)= 1;         % keep the column
      k = k + 1;
   end
end
```

B.3.3 ResetFunction.m

```
function [] = ResetFunc ()

t0 = clock;
set_param ('plant/Reset/Reset', 'Value', '1')
while (etime(clock,t0) < 1)
    ;
end
clock;
set_param ('plant/Reset/Reset', 'Value', '0')
```

B.4 Test interface

The next functions are implemented to work on line. They modify the system parameters and down load the results into the Target PC while the system is in motion.

B.4.1 LoadParameters.m

```
nfault = 0;
flightmodel

status = get_param ('plant/ActuatorStatus','fAct1');
if (strcmp (status, 'on'))
    set_param ('flightmodel/FlightModel/Fault1','Gain', '1')
else
    set_param ('flightmodel/FlightModel/Fault1','Gain', '0')
    nfault = nfault + 1;
end
```

```
status = get_param ('plant/ActuatorStatus','fAct2');
if (strcmp (status, 'on'))
    set_param ('flightmodel/FlightModel/Fault2','Gain', '1')
else
    set_param ('flightmodel/FlightModel/Fault2','Gain', '0')
    nfault = nfault + 1;
end

status = get_param ('plant/ActuatorStatus','fAct3');
if (strcmp (status, 'on'))
    set_param ('flightmodel/FlightModel/Fault3','Gain', '1')
else
    set_param ('flightmodel/FlightModel/Fault3','Gain', '0')
end

Modelling
Controller
Observer
if (nfault == 0)
    set_param ('plant/SensorFaultDiagnosis/DetectionLogic/ActFault','Value', '1')
else
    set_param ('plant/SensorFaultDiagnosis/DetectionLogic/ActFault','Value', '2')
end
```

B.4.2 Reference call-backs

set_param ('Plant/Reference', 'After', '[0 0 0 0 0 0 0]')

set_param ('Plant/Reference', 'After', '[0 0 0 0 0 0 45]')

B.4.3 UpdateModel.m

```
nfault = 0;
status = get_param ('plant/ActuatorStatus','fAct1');
if (strcmp (status, 'on'))
    set_param ('flightmodel/FlightModel/Fault1','Gain', '1')
```

```
else
    set_param ('flightmodel/FlightModel/Fault1','Gain', '0')
    nfault = nfault + 1;
end

status = get_param ('plant/ActuatorStatus','fAct2');
if (strcmp (status, 'on'))
    set_param ('flightmodel/FlightModel/Fault2','Gain', '1')
else
    set_param ('flightmodel/FlightModel/Fault2','Gain', '0')
    nfault = nfault +1;
end

status = get_param ('plant/ActuatorStatus','fAct3');
if (strcmp (status, 'on'))
    set_param ('flightmodel/FlightModel/Fault3','Gain', '1')
else
    set_param ('flightmodel/FlightModel/Fault3','Gain', '0')
end

Modelling
if (nfault == 0)
    set_param ('plant/SensorFaultDiagnosis/DetectionLogic/ActFault','Value', '1')
else
    set_param ('plant/SensorFaultDiagnosis/DetectionLogic/ActFault','Value', '2')
end
```

B.4.4 Update controller call-backs

controller

set_param ('Plant/ControlLaw', 'Gain', 'K')

B.4.5 UpdateObserver.m

```
observer
set_param ('plant/ObserverBank/NominalObserver/L', 'Gain', 'L')
```

```
set_param ('plant/ObserverBank/FaultObserver1/L', 'Gain', 'Lf1')
set_param ('plant/ObserverBank/FaultObserver2/L', 'Gain', 'Lf2')
set_param ('plant/SensorFaultDiagnosis/DetectionLogic/Threshold','Value','fault')
```

C Remote Control

C.1 Introduction

To control the system manually, in order to make the experiments which identified the system parameters, there were used a real time card and the software provided by DSPACE. There are mainly two programs for the control and data acquisition. The first one, called *Cockpit*, is used to develop a control panel from where it is possible to modify the control orders and to read the data from the sensors. The second one, called *Trace,* is used to acquired data, represent them in a graphic window and stored them in *Matlab* format so that it is also possible to process them. It is also necessary to implement an interface which communicate the software with the real time system. This interface is programmed using Simulink, as a .mdl file. The blocks which represents the analog / digital and digital / analog converters which are in the real time system are selected from the software libraries provided with the real time system. The real time system ports are connected to the wires coming from the sensors and the wires going to the actuators. This .mdl file has to be compiled in order to down load the program into the real time system. It is built using an option situated in the *Tools* menu. The code is down loaded in the real time processor and run. When the *Simulink* interface is compiled, it is generated a .trc file which is necessary for the *Cockpit* and *Trace* programs in order to control and acquire data from the experiment.

C.2 Manual Control: *CockPit*

There are several controls stored in the program libraries for both input and output. It is also possible to select special controls in order to put up a background image or to write a text poster. There were selected one slider, one incremental input and one on / off buttons for each input, u_1, u_2, u_3, which control the attack angle of the rotor one and two and the voltage applied to the lateral rotors. The range is set from -1 to 1 and scaled later in the interface. There are also one display and one slider for each sensor: γ_1, γ_2 and β. The panel control is shown in the figure 58.

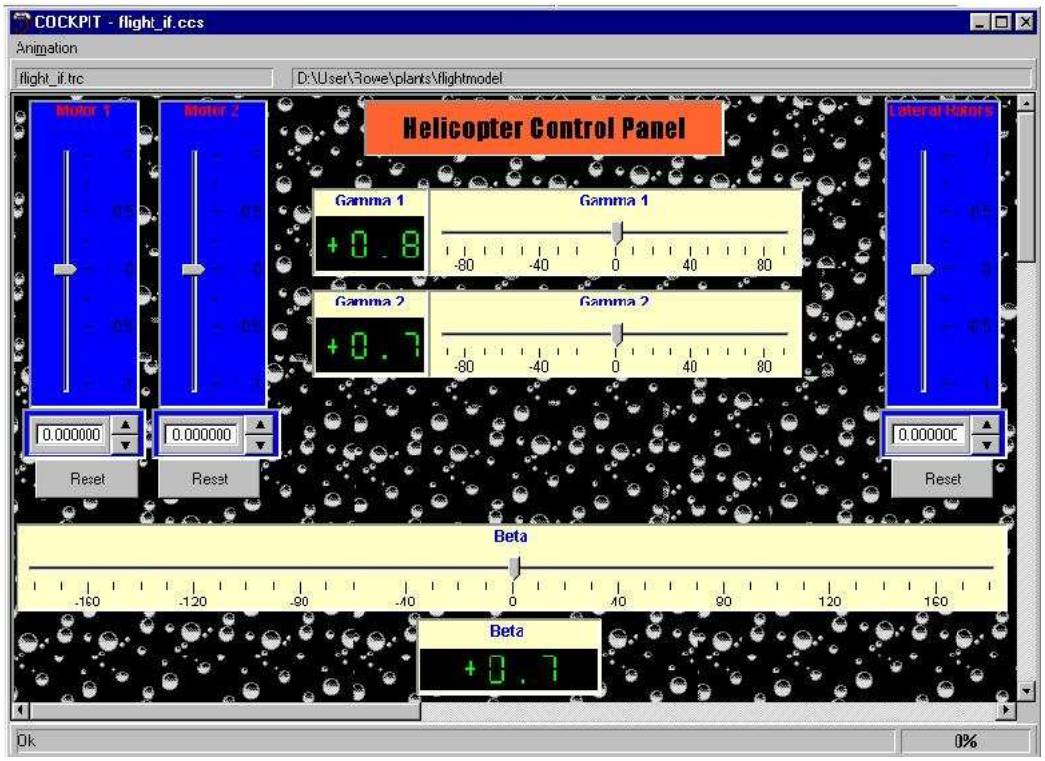


Figure 58: Control panel

C.3 Data Acquisition: *Trace*

As it is shown in the figure 59, there are two Trace windows. The upper one is the control panel for data acquisition where the experiment time length is chosen, and there is set when the experiment starts, the file in which you could store the information, the name of the experiment, the variables affected (*model root* has to be opened and the required variables selected in order to trace them), how and when the information will be shown and some other parameters. All the parameters selected can be stored in an experiment file. The lowest one is the graphic window where the signals are drawn. There are also some utilities in order to measure values, differences between signal, times, zooms and so on. The variables traced can be stored in a template file.

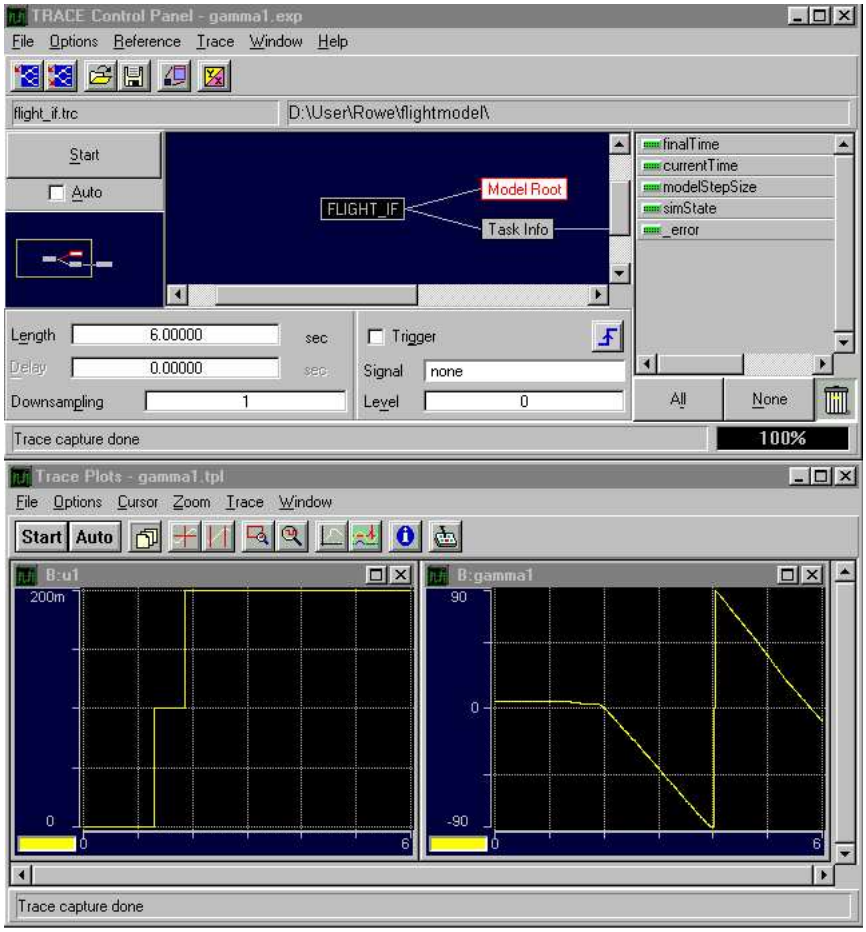


Figure 59: Trace

C.4 Interface: *Simulink*

The interface was implemented using *Simulink*. It was a similar design as the one in the figure 7 but, of course, the blocks which contain the port information were different since the real time system card was the one provided by DSpace. Therefore, there were only minor differences.

D Graphics

D.1 Rotor Attack Angle Identification

The figure 60 shows the system reaction to step functions from 0 to 0.1 and from 0.1 to 0.2. According to the plot, the dead zone upper limit is between 0.1 and 0.2. The plot shows a strong nonlinearity at the end of the measurement range. A zero degrees measure is read while changing from -90° to 90°.

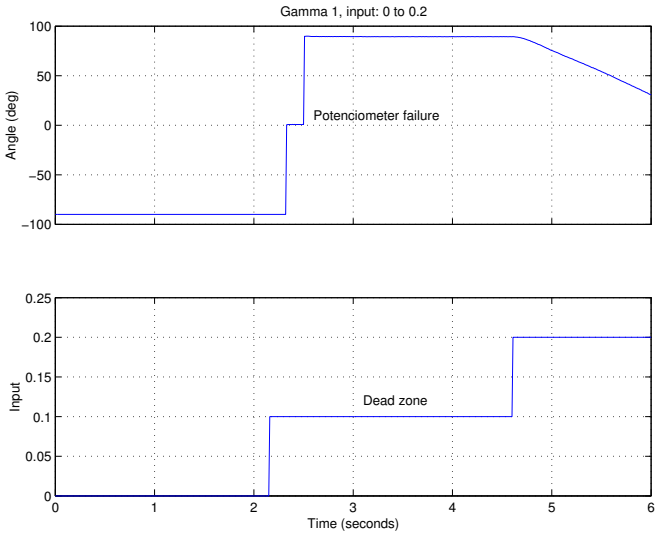


Figure 60: Rotors attack angle identification, experiment 1

The figure 61 shows the plant behavior when step functions from 0.4 to 0.5 and from 0.5 to 0.6 are given. According to the plot, the saturation level appears between 0.5 and 0.6.

The figure 62 shows the system behavior inside the dynamic range. Step functions from 0.2 to 0.3 and from 0.3 to 0.4 are used to show the reaction. Nonlinearities appears at the end of the measurement range. The sensor reads $-90°$ for several instance while the system is moving a measure of 0°is read before getting the initial range position of 90°.

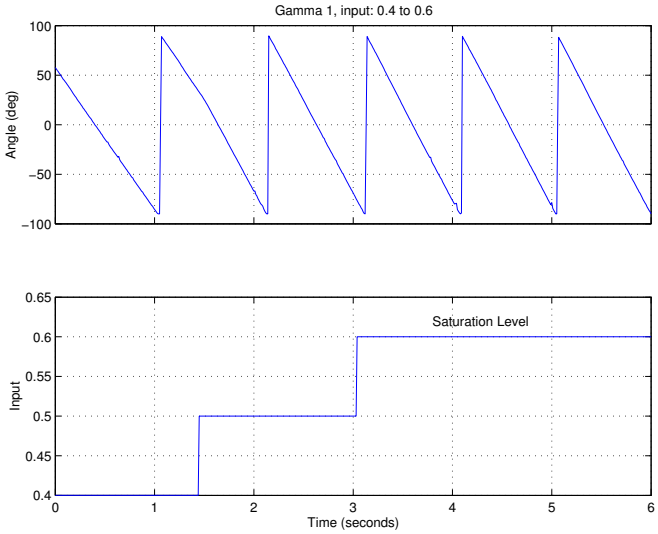


Figure 61: Rotors attack angle identification, experiment 2

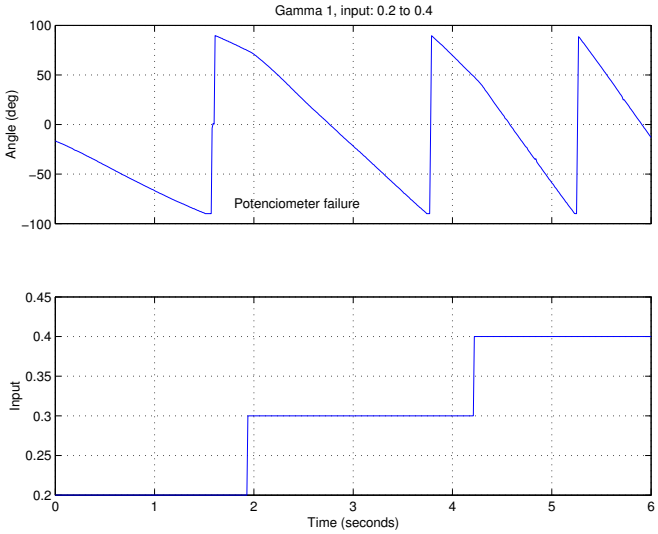


Figure 62: Rotors attack angle identification, experiment 3

The experiments shown in the figures 63, 64 and 65 are the ones used to identify the rotor attack angle subsystem parameters. The first plot shows the system behavior after an input jump from 0.2 to 0.3. The second one shows the transition from an input of 0.3 to another of 0.4. The last plot shows the reaction after a step function from an input of 0.4 to 0.5. The identified parameters differ slightly along the dynamic range. Mean values are taken as system parameters.

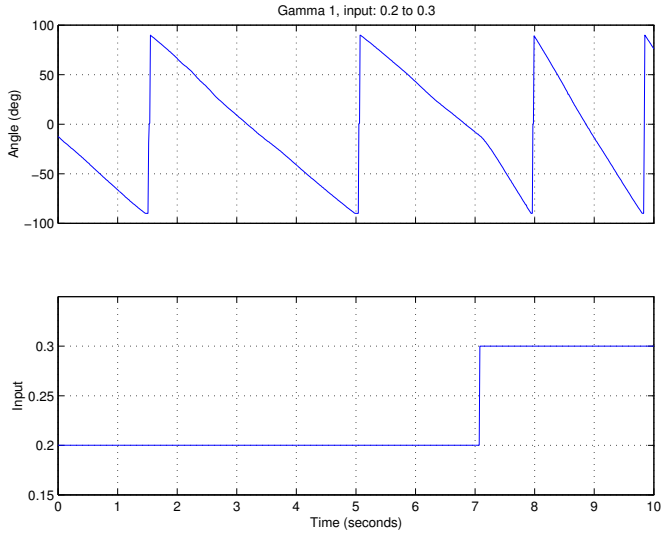


Figure 63: Rotors attack angle identification, experiment 4

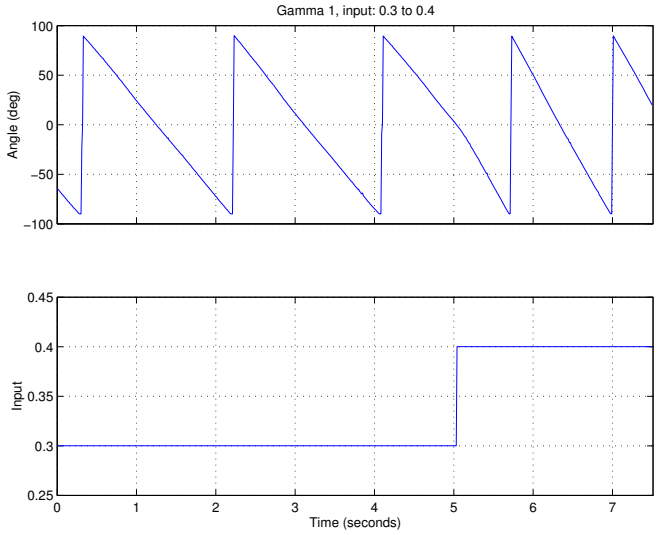


Figure 64: Rotors attack angle identification, experiment 5

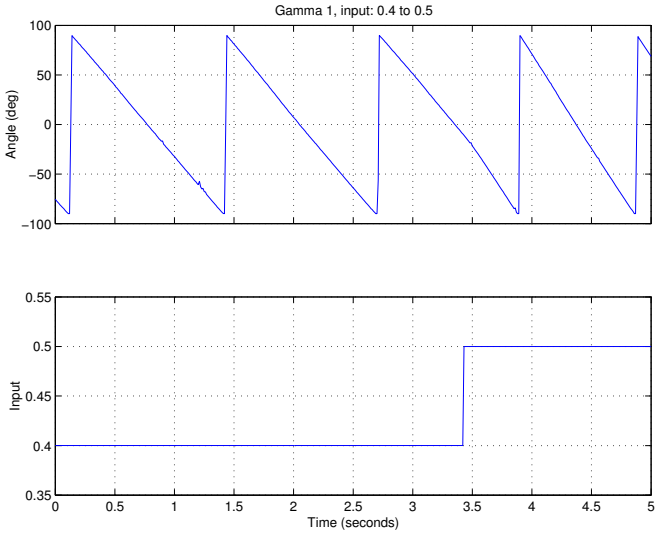


Figure 65: Rotors attack angle identification, experiment 6

D.2 Lateral Rotors Identification

The figure 66 shows the oscilloscope plot used to measure the lateral rotor current. A jump is given to the input in order to identify the time constant.

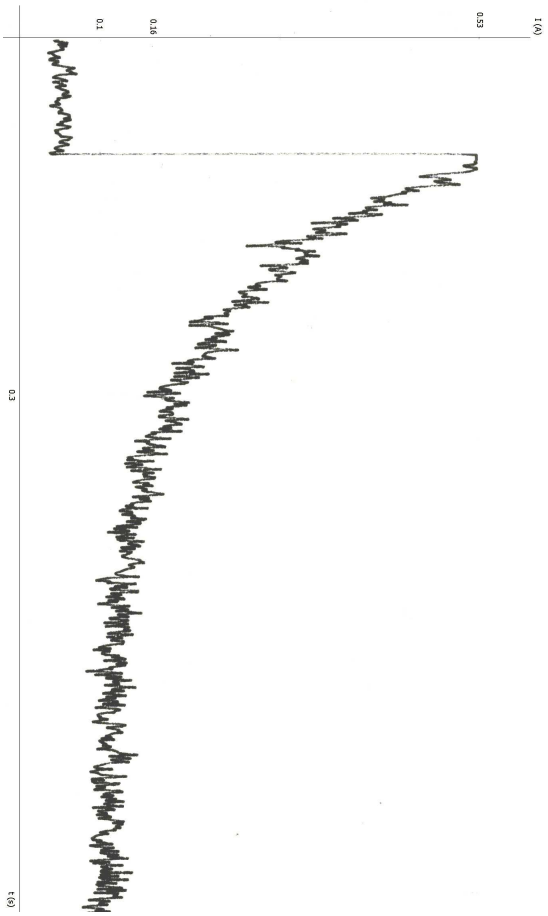


Figure 66: Identification of the lateral rotors time constant

D.3 Angular Position System Identification

The momentum constant relative to the lateral rotors is identified using the experiment shown in the figure 67.

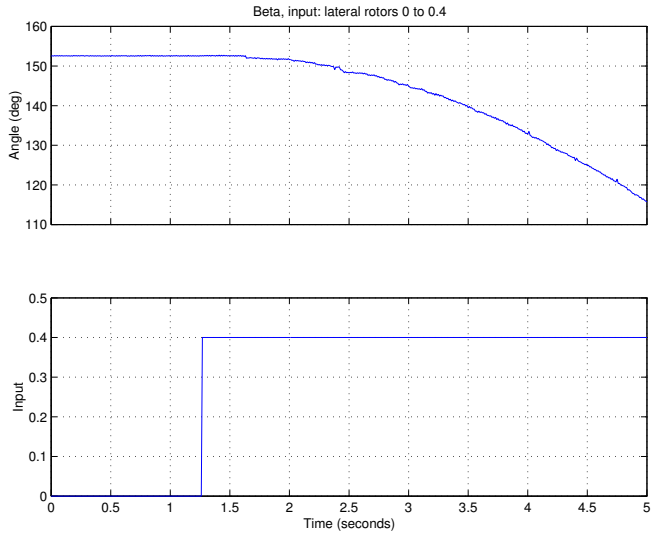


Figure 67: Identification of the momentum constant, experiment 1

The momentum constant relative to the main rotors is identified using the experiment shown in the figure 68.

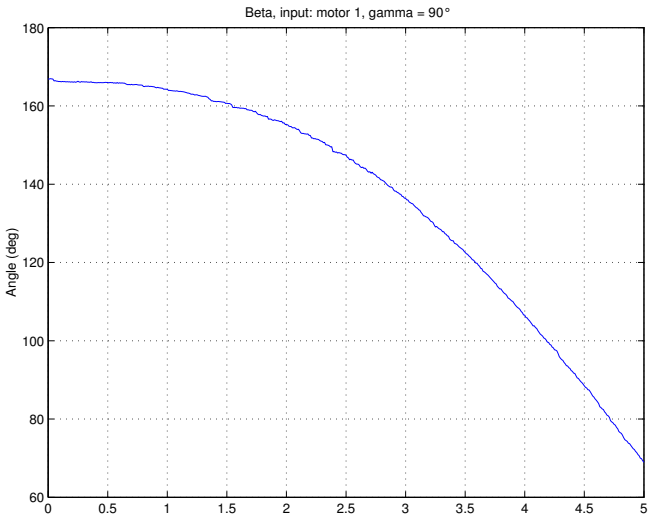


Figure 68: Identification of the momentum constant, experiment 2

E Tables

E.1 Datamatrix

The table 3 shows and example of the data acquired in the previous experiments and processed using the *datamatrix.m* function.

Time (s)	Input	Angle (°)	Instant speed ($°s^{-1}$)
0.00	0.20	-12.3	0
0.01	0.20	-12.7	-40
0.02	0.20	-13.4	-70
0.03	0.20	-13.9	-50
0.04	0.20	-14.6	-70
0.05	0.20	-15.0	-40
0.06	0.20	-15.7	-70
0.07	0.20	-16.2	-50
0.08	0.20	-16.7	-50
0.09	0.20	-17.3	-60
0.10	0.20	-17.9	-60
0.11	0.20	-18.3	-40
0.12	0.20	-18.9	-60
0.13	0.20	-19.3	-40
0.14	0.20	-19.8	-50
0.15	0.20	-20.1	-30
0.16	0.20	-20.8	-70
0.17	0.20	-21.2	-40
0.18	0.20	-21.9	-70
0.19	0.20	-22.3	-40

Table 3: Experimental Data

E.2 Lateral Rotors Identification

The data relative to the lateral rotor gain identification are related in the table 4. It shows the input given in each experiment, the forces applied by the rotors due to these inputs and the calculated gain. It also shows saturation levels, and dead zones due to the system friction and to the motor friction. The mean value of the gain is taken as a subsystem parameter. Extremes values are not taken in consideration.

Input	Force (N)	Comment	Gain
-0.70	+1.0	Saturated	
-0.60	+1.0		-1.4
-0.50	+ 0.9		-1.7
-0.40	+0.7		-1.8
-0.30	+0.4		-1.3
-0.20	0.0	System is not moving	
-0.10	0.0	System is not moving	
-0.01	0.0	Rotors are not moving	
+0.01	0.0	Rotors are not moving	
+0.10	0.0	System is not moving	
+0.20	-0.3		-1.5
+0.30	-0.9		-3.0
+0.40	-1.4		-3.5
+0.50	-1.7		-3.4
+0.60	-1.8		-3.0
+0.70	-1.9		-2.7
+0.80	-2.0		-2.5
+0.9	-2.0	Saturated	

Table 4: Identification of the l*ateral rotors* gain

References

[1] D. F. Franklin: *Feedback Control of Dynamic Systems*, Addison Wesley, 1994.

[2] J. Lunze, Th. Steffen: *Hybrid Reconfigurable Control*, in S.Engell, G. Frese, E. Schneider: Modeling, Analysis and design of Hybrid Systems. Spinger Verlag Berlin, 2002.

[3] F.L.Lewis: *Linear Quadratic Regulator (LQR) State Feedback Design,* lecture of the Advanced Controls, Sensors and MEMS Group, Automation and Robotics Research Institute, 1998.

[4] The MathWorks: *Using Simulink, Version 3, Modeling, Simulation, Implementation*, 1999.

[5] The MathWorks: *xPC Target User's Guide*, 2001.