INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Degree Course in Electrical Engineering

Graduate Thesis in Electrical Engineering

AIRPLANE AUTOPILOT DEVELOPMENT

Tutors:

Prof. Ing. ***José Carlos BARROS OLIVEIRA***

Student:

Andrea SACCOMANNO

ACADEMIC YEAR 2005-2006

Aos meus pais, Angelo e Mariarosaria,

Ao Professor Josè Carlos Barros Oliveira,

A Claudette, a minha namorada.

Obrigado,

Andrea

INDEX

INTRODUCTION

The aim of this thesis work is to develop an autopilot system capable to control the motion axes of a General Aviation airplane.

Generally, an autopilot system carries out all the control functions that are boring to the pilots like manually maintain the route, heading or altitude, for this reason autopilot are used; is only necessary to set the target value of the parameter that pilot wants to maintain and check the flight parameter indicators to verify that everything is going well.

In the most modern airplanes, the flight control surfaces (ailerons, elevators and rudder) are moved by hydraulic actuators feeds by two independent hydraulic lines. The hydraulic power regulation is realized by proportional servo valves directly actuated, supplies with electrical engine.

The autopilot that we want to realize will has three fundamental functionalities, provided by three different controllers. The provided functionalities are:

- heading control and maintenance;
- altitude control and maintenance;
- route control and maintenance.

We want to design a system capable to control the lateral and the longitudinal airplane motion without implement a throttle controller. To realize this system we have divided the work in three phases described in following lines.

The first phase consists in to gain knowledge on the science that is at bases of the problems of the flight control and the automatic control system; chapters 1 and 2 concerns this phase, in chapter 1 we'll study in depth the problem of the flight control surfaces, we'll analyse the various kinds of flight control systems that are implemented

in the most common aircrafts; then we'll describe the on-board navigation and positioning systems currently used in the airplane.

In the chapter 2 will be treated the bases of automatic control theory and will be studied in depth a particular class of regulators: the PID controller.

The second phases represent the design of the autopilot system and this will be developed in the chapter 3, in this chapter will be explained and discussed the design methods that we have used to develop the controllers.

The third and last phase consist in the controllers test and these will be carried out using *Microsoft® Flight Simulator 2004*, each controller will be tested in a variable number of flight conditions, this number of simulations depends by the typology of controller that we want to test.

The aim that we have prefixed for this work is to realize an autopilot system with low develop costs and to obtain comparable performances in terms of accuracy, rapidity and stability of the autopilots that *Microsoft® Flight Simulator 2004* implements in his airplanes.

CHAPTER 1

FLIGHT CONTROL SURFACES, FLIGHT CONTROL SYSTEMS, AIRCRAFT NAVIGATION AND POSITIONING SYSTEMS

1.1 Introduction

Flight control has advanced considerably throughout the years. In this chapter we would to explain the basic concepts relative to the flight control systems.

It contains the basic principles of aircraft motion (*translational* and *rotational* motion) and explains which are the aircraft primary controls and how it's set of controls operate on the primary control surfaces called elevators, ailerons and rudder. In this chapter there is a short explication on difference between fighters and airliners. Then, it presents an illustration of the possible flight control system configurations.

Inside this chapter, will be showed the aircraft navigation systems useful for obtain the exact position of aircraft relative to the Earth (altitude, longitude, latitude) and all the flight information like air speed, vertical speed, angle of incidence, bank angle and heading.

The most accurate system is GPS and it will be the main reference instrument of this thesis work because it's integrated on our test aircraft.

Last, will be treats the operating concepts of modern autopilot systems.

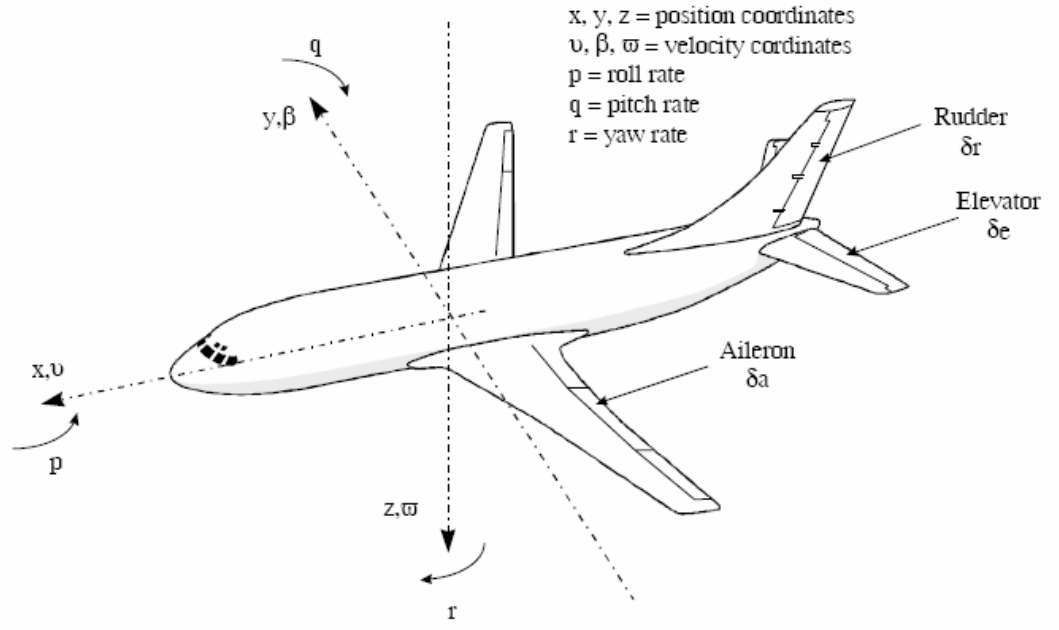
1.2 Principles of flight control

All aircraft are governed by the same basic principles of flight control, whether the vehicle is the most sophisticated high-performance fighter or the simplest model aircraft.

The motion of an aircraft is defined to translational motion and rotational motion around a fixed set of pre-determined axes. Translational motion is that by which a vehicle travels from one point to another in space.

For an orthodox aircraft there is only one direction in which translational motion occurs, that is the direction in which the aircraft is flying which is also the direction in

which it is pointing. The rotational motion relates to the motion of the aircraft around three defined axes, *pitch*, *roll* and *yaw*.



1.1: Definition of aircraft coordinates

Figure 1.1 shows the direction of aircraft velocity vector in relation to the pitch, roll and yaw axes. For most of the flight an aircraft will be flying straight and level and velocity vector will be parallel with the surface of earth and proceeding upon a heading that the pilot has chosen.

If the pilot wishes to climb the flight control system is required to rotate the aircraft around the pitch axis in a nose-up sense to achieve a climb angle. Upon reaching the new desired altitude the aircraft will be rotated in a nose-down sense until the aircraft is once again flying straight and level.

In most fixed wing aircraft, if the pilot wishes to alter the aircraft heading then he will need to execute a turn to align the aircraft with the new heading. During a turn the aircraft wings are rotated around the roll axis until a certain angle of bank is attained.

In a properly balanced turn the angle of roll (often called the *bank angle*) is maintained. This change in heading is actually a rotation around the yaw axis. The difference between the climb (or descent) and the turn is that the climb only involves rotation around one axis whereas the turn involves simultaneous coordination of two axes.

At certain times during flight the pilot may in fact be rotating the aircraft around three axes, for example during a climbing or descending turning manoeuvre.

The aircraft flight control system enables the pilot to exercise control over the aircraft during all portion of flight. The system provides control surfaces which allow the aircraft to manoeuvre in pitch, roll and yaw. The system has also to be designed so that it provides stable control for all parts of the aircraft flight envelope: this requires a through understanding of the aerodynamics and dynamics motion of the aircraft.

As will be seen, additional control surfaces are required during approach and landing phases of flight. The flight control system has to give the pilot considerable physical assistance to overcome the enormous aerodynamic forces on the flight control surfaces. This in turn leads to the need to provide the aircraft controls with artificial *feel* so that he does not inadvertently overstress the aircraft. These *feel* systems need to provide the pilot with progressive and well harmonized control which make the aircraft safe and pleasant to handle. A typical term which is commonly used today to describe this requirement is *carefree handling*. Many aircraft embody automatic flight control systems to ease the burden of flying the aircraft and to reduce pilot workload.[1]

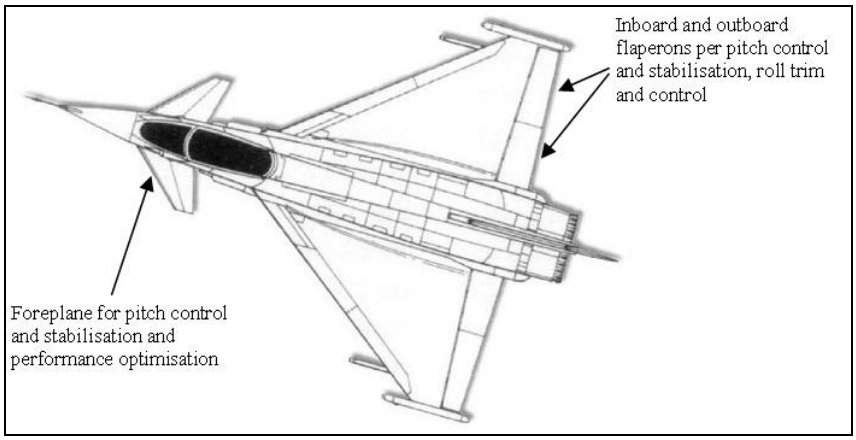
1.3 Flight control surfaces

The requirements for flight control surfaces vary greatly between one aircraft and another, depending upon the role, range and agility needs of the vehicle. These varying requirements may best be summarized by giving examples of two differing types of aircraft: an agile fighter aircraft and a typical modern airliner.

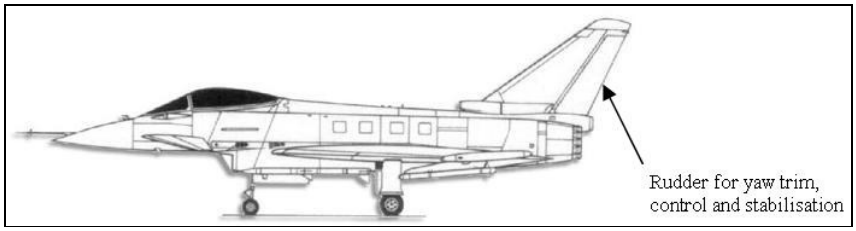
1.3.1 Fighter aircraft

In Figures 1.2 and 1.3 are showed the flight control surfaces of an Euro Fighter 2000 "*Typhoon*" (EF2000) that represents the state-of-the-art fighter aircraft as defined by European manufactures at the beginning of the 1990s.

The EF2000 is developed by the four nation consortium comprising Alenia Aeronautica (Italy), British Aerospace (United Kingdom), CASA (Spain), EADS (Germany).



1.2: Flight control surfaces – EF2000 (BAe, Alenia Aeronautica, EADS, CASA)



1.3: Flight control surfaces – EF2000 (BAe, Alenia Aeronautica, EADS, CASA)

Primary flight control in pitch, roll and yaw is provided by the control surfaces described below.

Pitch control is provided by the moving canard surfaces, or *foreplanes*, as they are sometimes called, located either side of the cockpit. These surfaces provide the very powerful pitch control authority required by an agile high performance fighter aircraft.

The position af the canard in relation to the wings renders the aircraft unstable. Without the benefit of an active computer driven control system the aircraft would be uncontrollable and would crash in a matter of seconds.

While this may appear to be a fairly drastic implementation, the benefit in terms of improved manoeuvrability enjoyed by the pilot outweighs the engineering required to provide the computer controller or “active” flight control system.

Roll control is provided by the differential motion of the foreplanes. In order to roll to the right the left foreplane leading edge is raised relative to the airflow generating grater lift then before.

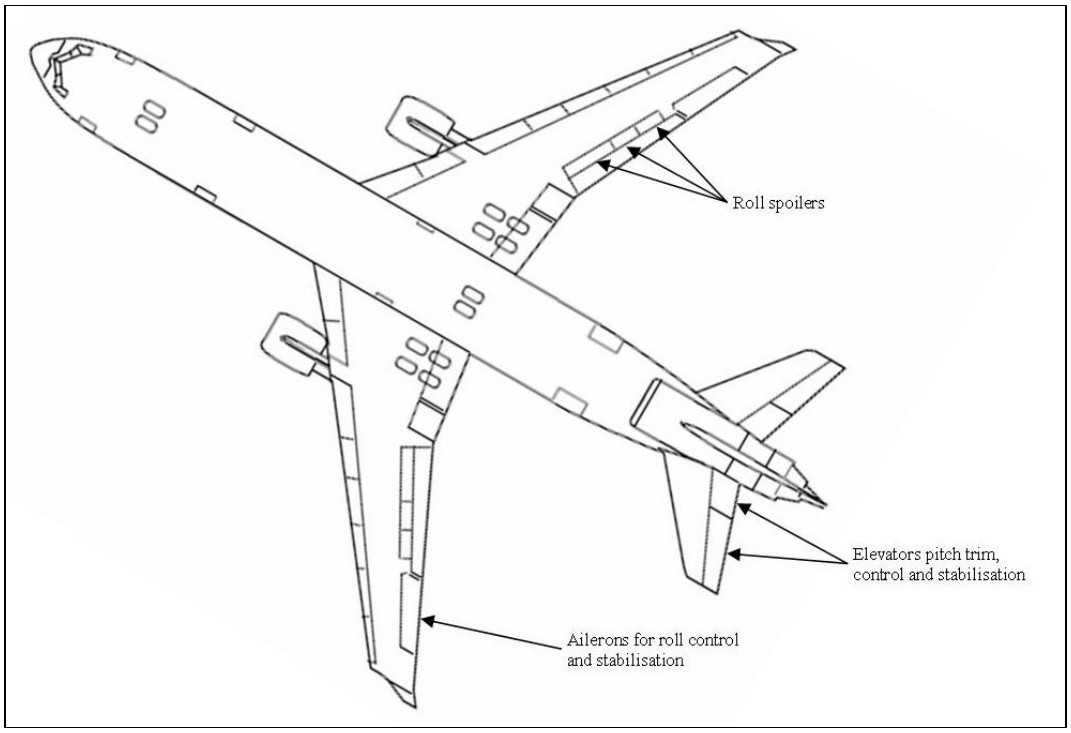
Conversely, the right foreplane moves downwards by a corresponding amount relative to the airflow thereby reducing the lift generated. The resulting differential forces cause the aircraft to roll rapidly to the right. To some extent roll control is also provided by the differential action of the wing trailing edge flaperons (sometimes called *elevons*).

However most of the roll control is provided by the foreplanes.

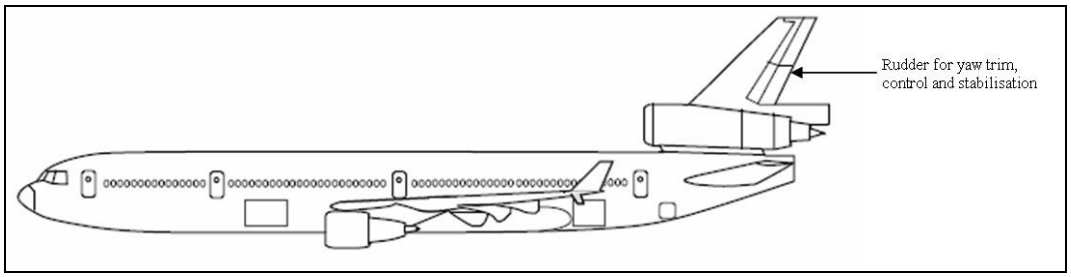
Yaw control is provided by the single rudder section. For high performance fighter aircraft yaw control is generally less important than for conventional aircraft due to the high levels of excess power. There are nevertheless certain parts of the flight envelope where the control yaw (or *side-slip*) is vital to prevent roll-yaw divergence.

1.3.2 Commercial aircraft

An example of flight control surfaces of a typical commercial airliner is shown in Fig. 1.4 and Fig. 1.5. Although the example is for the McDonnell Douglas MD-11 it holds good for similar airliners produced by Boeing or Airbus Industries.



1.4: Flight control surfaces – MD-11 (McDonnell Douglas)



1.5: Flight control surfaces – MD-11 (McDonnell Douglas)

The controls used by this type of aircraft are described below:

Pitch control is exercised by four elevators located on the trailing edge of the tailplane (or *horizontal stabilizer*). Each elevator section is independently powered by a dedicated flight control actuator, powered in turn by one of several aircraft hydraulic power systems. This arrangement is dedicated by the high integrity requirements placed upon flight control systems. The entire tailplane section itself is powered by two or more actuators in order to trim the aircraft in pitch. In a dire emergency this facility

could be used to control the aircraft, but the rates of movement and associates authority are insufficient for normal control purposes.

Roll control is provided by two aileron sections located on the outboard third of the trailing edge of each wing. Each aileron section is powered by a dedicated actuator powered in turn from one of the aircraft hydraulic systems. At low airspeed the roll control provided by the ailerons is augmented by differential use of the wing spoilers mounted upon the upper surface of the wing. During a right turn the spoilers on the inside wing of the turn, that is the right wing, will be extended. This reduces the lift of the right wing causing it to drop, hence enhancing the desired roll demand.

Yaw control is provided by three independent rudder section located on the trailing edge of the fin (or *vertical stabilizer*). These sections are powered in a similar fashion to the elevators and ailerons. On a civil airliner these controls are associated with the aircraft *yaw dampers*. These damp out unpleasant *dutch roll* oscillation which can occur during flight and which can be extremely uncomfortable for the passenger, particularly those seated at the rear of the aircraft.

1.4 Flight control systems

The pilot's manual input to the flight control is made by moving the cockpit control column (*cloche*) or rudder pedals in accordance with the universal convention:

- Pitch control is exercised by moving the cloche fore and aft; pushing the cloche forward causes the aircraft to pitch down, and pulling the cloche aft results in a pitch up.
- Roll control is achieved by moving the cloche from side to side; pushing the stick to the right drops the right wing and vice versa.
- Yaw is controlled by the rudder pedals; pushing the left pedal will yaw the aircraft to the left while pushing the right pedal will have the reverse effect.

There are two main methods of connecting the pilot's controls to the rest of the flight control system. These are *push-pull control rod systems* and *cable and pulley systems* but these aren't part of the goal that this work wants to achieve.

Flight control systems are available in a large variety of configurations in the following paragraphs.

1.4.1 Mechanical systems

Mechanical flight control systems are the most basic designs. They were used in early aircraft and currently in small airplanes where the aerodynamic forces are not excessive. The flight control systems uses a collection of mechanical parts such as rods, cables, pulleys and sometimes chains to transmit the forces of the cockpit controls to the control surfaces. The aircraft showed in figure 1.6, that is the Cessna C172SP *Skyhawk*, used to develop the controller, is a typical example.

1.6: Cessna C172SP *Skyhawk* – photo courtesy by Cessna

Since an increase in control surface area in bigger and faster aircraft leads to a large increase in the forces needed to move them, complicated mechanical arrangements are used to extract maximum mechanical advantage in order to make the forces required bearable to the pilots. This arrangement is found on bigger or higher performance propeller aircrafts.

Some mechanical flight control systems use servo tabs that provide aerodynamic assistance to reduce complexity. Servo tabs are small surfaces hinged to the control surfaces. The mechanisms move these tabs and aerodynamic forces in turn move the control surfaces reducing the amount of mechanical forces needed. This arrangement was used in early piston-engine transport aircraft and in early jet transports such as the mostly mechanical Boeing 707.

1.4.2 Hydromechanical systems

The complexity and weight of a mechanical flight control systems increases considerably with size and performance of the airplane. Hydraulic power overcomes these limitations. With hydraulic flight control systems aircraft size and performance are limited by economics rather than a pilot's strength.

A hydraulic flight control systems has 2 parts:

- The mechanical circuit;
- The hydraulic circuit.

The mechanical circuit links the cockpit controls with the hydraulic circuits. Like the mechanical flight control systems, it is made of rods, cables, pulleys, and sometimes chains.

The hydraulic circuit has hydraulic pumps, pipes, valves and actuators. The actuators are powered by the hydraulic pressure generated by the pumps in the hydraulic circuit. The actuators convert hydraulic pressure into control surface movements. The servo valves control the movement of the actuators.

The pilot's movement of a control causes the mechanical circuit to open the matching servo valves in the hydraulic circuit. The hydraulic circuit powers the actuators which then move the control surfaces. This arrangement is found in older jet transports and high performance aircraft.

In mechanical flight control systems, the aerodynamic forces on the control surfaces are transmitted through the mechanisms and can be felt by the pilot. This gives tactile feedback of airspeed and aids flight safety.

Hydromechanical flight control systems lack this "feel". The aerodynamic forces are only felt by the actuators. Artificial feel devices are fitted to the mechanical circuit of the hydromechanical flight control systems to simulate this "feel". They increase resistance with airspeed and vice-versa. The pilots feel as if they are flying an aircraft with a mechanical flight control systems.

1.4.3 Fly-by-wire systems

Mechanical and hydraulic flight control systems are heavy and require careful routing of flight control cables through the airplane using systems of pulley and cranks.

Both systems often require redundant backup, which further increases weight. Furthermore, both have limited ability to compensate for changing aerodynamic conditions.

By using computers and electrical linkages, designers can save weight and improve reliability. Electronic *fly-by-wire* (FBW) systems can respond more flexibly to changing aerodynamic conditions, by tailoring flight control surface movements so that airplane response to control inputs is consistent for all flight conditions.

Electronic systems require less maintenance, whereas mechanical and hydraulic system require lubrication, tension adjustments, leak checks, fluid changes, etc. Furthermore putting circuitry between pilot and aircraft can enhance safety; for example the control system can prevent a stall, or can stop the pilot from overstressing the airframe.

A fly-by-wire system literally replaces physical control of the aircraft with an electrical interface.

The pilot's commands are converted to electronic signals, and flight control computers determine how best to move the actuators at each control surface to provide the desired response. Those actuators initially are usually hydraulic, but electric actuators have been investigated.

The main concern with fly-by-wire systems is reliability. While traditional mechanical or hydraulic control systems usually fail gradually, the loss of all flight control computers will immediately render the airplane uncontrollable. For this reason, most fly-by-wire systems incorporated redundant computers and some kind of mechanical or hydraulic backup.

This may seem to negate some advantages of fly-by-wire, but the redundant systems can be simpler, lighter, and offer only limited capability since they are for emergency use only.[2]

1.5 Aircraft navigation and positioning systems

In flight, an aircraft and its operating crew form a *man-machine* system loop which, depending on the size and type of aircraft, may be fairly simple or very complex. The function of the crew within the loop is that controller, and the extent of the control function is governed by the simplicity or otherwise of the aircraft as an integrated whole. For example, in manually flying an aircraft, and manually initiating adjustments to essential systems, the controller’s function is said to be a fully active one. If, on the other hand, the flight of an aircraft and system’s adjustment are automatic in operation, then the controller’s function becomes one of monitoring, with the possibility of revering to the active function in the event of failure of systems.

Instruments, of course, play an extremely vital role in the control loop as they are the means of communicating data between systems and controller.

It is one in which the display of pitch and roll attitudes and heading of an aircraft are integrated with such radio navigation systems as *Automatic Direction Finding* (ADF), *VHF Omni directional Range* (VOR), and *Instrumental Landing System* (ILS) so as to perform a total directive command function.

It also provides for the transmission of attitude and navigational data to an *Automatic Flight Control System* (AFCS) so that in combination it can operate as an effective flight guidance system.[3]

1.5.1 Global Positioning System (GPS)

The GPS is a satellite-based navigation system made up of a network of 24 satellites placed into orbit by the U.S. Department of Defense. GPS was originally intended for military applications, but in the 1980s, the government made the system available for civilian use. GPS works in any weather conditions, anywhere in the world, 24 hours a day.

GPS satellites circle the earth twice a day in a very precise orbit and transmit signal information to earth. GPS receivers take this information and use triangulation to calculate the user's exact location. Essentially, the GPS receiver compares the time a signal was transmitted by a satellite with the time it was received. The time difference tells the GPS receiver how far away the satellite is. Now, with distance measurements from a few more satellites, the receiver can determine the user's position and display it on the unit's electronic map.

A GPS receiver must be locked on to the signal of at least three satellites to calculate a 2D position (latitude and longitude) and track movement. With four or more satellites in view, the receiver can determine the user's 3D position (latitude, longitude and altitude). Once the user's position has been determined, the GPS unit can calculate other

information, such as speed, bearing, track, trip distance, distance to destination, sunrise and sunset time and more.

1.5.2 Autopilot systems

An autopilot is a mechanical, electrical, or hydraulic system used to guide a vehicle without assistance from a human being. Most people understand an autopilot to refer specifically to aircraft, but autopilots for boats and ships are called by the same name and serve the same purpose.

In the early days of transport aircraft, aircraft required the continuous attention of a pilot in order to fly in a safe manner. This created very high demands on crew attention and high fatigue. The autopilot is designed to perform some of the tasks of the pilot.

The first aircraft autopilot was developed by Sperry Corporation. It connected a gyroscopic attitude indicator and magnetic compass to hydraulically operated rudder, elevator, and ailerons. It permitted the aircraft to fly straight and level on a compass course without a pilot's attention, thus covering more than 80% of the pilot's total workload on a typical flight. This straight-and-level autopilot is still the most common, least expensive and most trusted type of autopilot. It also has the lowest pilot error, because it has the simplest controls.

Modern autopilots generally divide a flight into taxi, take-off, ascent, level, descent, approach, landing, and taxi phases. Autopilots exist that automate all of these flight phases except the taxiing. Landing on runway and controlling the aircraft on rollout i.e keeping it on the centre of the runway is cat 3b landing, used on the majority of major runways today. Landing, rollout and taxi control to stand is Cat. IIIC. This is not usually used to date but may be used in the future and some incorporate automated collision-avoidance, The most popular collision avoidance for aircraft is called TCAS

(Traffic and collision avoidance system). An autopilot is often an integral component of a *Flight Management System* (FMS).

Modern autopilots use computer software to control the aircraft. The software reads the aircraft's current position, and controls a flight control system to guide the aircraft. In such a system, besides classic flight controls, many autopilots incorporate thrust control capabilities that can control throttles to optimize the air-speed, and move fuel to different tanks to balance the aircraft in an optimal attitude in the air.

Although autopilots handle new or dangerous situations inflexibly, they generally fly an aircraft with a lower fuel-consumption than all but a few of the best pilots.

The autopilot reads its position and the aircraft's attitude from an *Inertial Navigation System* (INS). INS accumulates errors over time. These errors are corrected by using satellite navigation systems and altimeters. They also may incorporate error reduction systems such as the carousel system that rotates once a minute so that any errors are dissipated in different directions and have an overall null effect. Error in gyroscopes is known as drift. This is due to physical properties within the system be it mechanical or laser guided that corrupt positional data. The disagreements between the two are resolved with digital signal processing, most often a six-dimensional Kalman filter. The six dimensions are usually roll, pitch, yaw, altitude, latitude and longitude. Aircraft may fly routes that have a required performance factor therefore the amount of error or actual performance factor must be monitored in order to fly those particular routes. The longer the flight the more error accumulates within the system. Radio aids such as DME, DME updates and GPS may be used to correct the aircraft position. Inertial Reference Units (IRU), i.e. gyroscopes, are the basis of aircraft on board position determining, as GPS and other radio update systems depend on a third party to supply information. IRU's are completely self-contained and use gravity and earth rotation to determine their initial position (earth rate). They then measure acceleration to calculate

where they are in relation to where they were to start with. From acceleration one can get speed and from speed one can get distance. As long as one knows the direction (from accelerometers) the IRU's can determine where they are (software dependent).

The hardware of a typical autopilot is a set of five 80386 CPUs, each on its own printed circuit board. The 80386 is an inexpensive, well-tested design that can implement a true virtual computer. New versions are being implemented that are radiation-resistant, and hardened for aerospace use. The very old computer design is intentionally favored, because it is inexpensive, and its reliability and software behavior are well-characterized.

The custom operating system provides a virtual machine for each process. This means that the autopilot software never controls the computer's electronics directly. Instead it acts on a software simulation of the electronics. Most invalid software operations on the electronics occur during gross failures. They tend to be obviously incorrect, detected and discarded. In operation, the process is stopped, and restarted from a fresh copy of the software. In testing, such extreme failures are logged by the virtualization, and the engineers use them to correct the software.

Usually, one of the processes on each computer is a low priority process that continually tests the computer.

Generally, every process of the autopilot runs more than two copies, distributed across different computers. The system then votes on the results of those processes. For triple autoland, this is called *camout*, and uses median values of autopilot commands versus mechanical centre and feel mechanism positioning as a possible computation. Extreme values are discarded before they can be used to control the aircraft.

Some autopilots also use design diversity. In this safety feature, critical software processes will not only run on separate computers, but each computer will run software created by different engineering teams. It is unlikely that different engineering teams

will make the same mistakes. As the software becomes more expensive and complex, design diversity is becoming less common because fewer engineering companies can afford it.[4]

CHAPTER 2

PROPORTIONAL-INTEGRAL-DERIVATIVE CONTROLLERS:

BASIC CONCEPTS , DESIGN AND TUNING

2.1 Introduction

In the autopilot systems all the flight information given by inertial instruments and by GPS are processed by a microprocessor that executes continuously control routines for to check the efficiency of all the systems which aircraft implements. Inside this main controller, there are a lot of sub-controller, each one of this realizes a specific control. To obtain a better controllability of the aircraft, is used the *feedback*; it allows to minimize the error of each flight variables and keep stable every plant that we want to control. A good feedback controller used in aeronautics for to obtain a good autopilot system in the *Proportional-Integral-Derivative Controllers* (PID). In this chapter there are explained the basic concepts of this kind of controllers.

The open-loop (*feedforward*) and closed-loop (*feedback*) control systems there are showed putting particular attention to the feedback control system because it is the basic structure of a PID control system.

The most commons system configurations able to achieve an appropriated control system and the fundamental principles of design are treated in this chapter.

The final paragraphs are dedicated to the PID design and tuning methods.

2.2 Control Systems

The basic ingredients of a control system can be described by:

- objectives of control;
- control system components;
- results.

The basic relationship between these three components is illustrated in Fig 2.1. in more technical terms, the objectives can be identified with *inputs*, or *actuating signal*, *u*, and the results are also called *outputs*, or the *controlled variables*, *y*.

Figure 2.1: Control system basic components

In general, the objective of the control system is to control the outputs in some prescribed manner by the input through the elements of the control system.[5]

2.2.1 Open-Loop (*Feedforward*) Control Systems

The elements of an open-loop control system can usually be divided into two parts: the controller and the controlled process, as shown in Fig. 2.2.

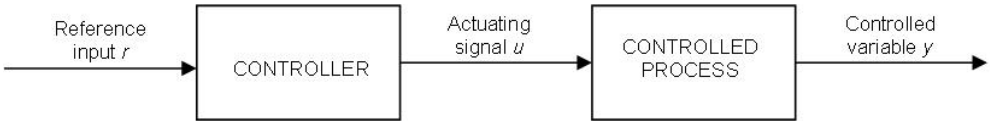


Figure 2.2: Elements of an open-loop control system

An input signal or command *r* is applied to the controller, whose output acts as the actuating signal *u*; the actuating signal the controls the controlled process so that the controller can be an amplifier, mechanical linkage, filter, or other control element, depending on the nature of the system. In more sophisticated cases, the controller can be a computer such as a microprocessor. Because of the simplicity and ecomomy of open-loop control systems, we find this type of system in many non-critical applications.

2.2.2 Closed-Loop (*Feedback*) Control Systems

What is missing in the open-loop control system for more accurate and more adaptive control is a link or *feedback* from the output to the input of the system. To obtain more

accurate control, the controlled signal y should be fed back and compared with the reference input, and an actuating signal proportional to the difference of the input and the output must be sent through the system to correct the error. A system with one or more feedback paths such as that just described is called a *closed-loop system* and is shown in Fig. 2.3.

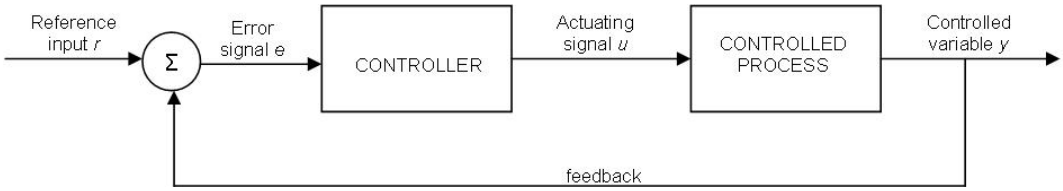


Figure 2.3: Elements of an closed-loop control system

2.2.3 System Transfer Function

The transfer function is a dynamic system representation that relates the Laplace Transforms of input and output variables. For this reason the transfer function is defined by the followings item.

Consider a system with *n* state variables, *m* input variables and *p* output variables:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.1}$$

$$y(t) = Cx(t) + Du(t) \tag{2.2}$$

Then the relatives Laplace transforms of *u(t)*, *x(t)* and *y(t)* are indicated with *U(s)*, *X(s)* and *Y(s)*. Transforming the equations (2.1) and (2.2) and applying its properties we obtain:

$$sX(s) - x(0) = AX(s) + BU(s) \tag{2.3}$$

$$Y(s) = CX(s) + DU(s) \tag{2.4}$$

Solving this new system we obtain:

$$X(s) = (sI - A)^{-1} BU(s) + (sI - A)^{-1} x(0) \tag{2.5}$$

$$Y(s) = (C(sI - A)^{-1} B + D)U(s) + C(sI - A)^{-1} x(0) \tag{2.6}$$

These equations (2.5) and (2.6) give the Laplace Transforms of state and input variations. We can observe the initial state component $(sI - A)^{-1} x(0)$ and the component with input effect $(sI - A)^{-1} BU(s)$ relatives to the state and the corresponding components of the output. We have obtained a $p \times m$ matrix showed below:

$$G(s) = C(sI - A)^{-1} B + D \tag{2.7}$$

The relation showed in (2.7) is called *Transfer Function* and multiplying this with the input *u* Laplace transform we can obtain the output *y* Laplace transform. Then the relations writes in (2.1) and (2.2) could be represented by this input-output relation:

$$Y(s) = G(s)U(s) \tag{2.8}$$

For these reasons we can write the system transfer function in the following fashion.

$$G(s) = \frac{Y(s)}{U(s)} \tag{2.9}$$

2.3 *Proportional-Integral-Derivative* (PID) Controller

The PID controller is the most common form of feedback. It was an essential element of early governors and it became the standard tool when process control emerged in the 1940s. In process control today, more than 95% of the control loops are of PID type, most loops are actually PI control.

PID controllers are today found in all areas where control is used. The controllers come in many different forms. There are standalone systems in boxes for one or a few loops, which are manufactured by the hundred thousands yearly. PID control is an important ingredient of a distributed control system. The controllers are also embedded in many Special purpose control systems. PID control is often combined with logic, sequential functions, selectors, and simple function blocks to build the complicated automation systems used for energy production, transportation, and manufacturing.

Many sophisticated control strategies, such as model predictive control, are also organized hierarchically. PID control is used at the lowest level; the multivariable controller gives the setpoints to the controllers at the lower level. The PID controller can thus be said to be the *bread and butter* of control engineering. It is an important component in every control engineer's tool box.

PID controllers have survived many changes in technology, from mechanics and pneumatics to microprocessors via electronic tubes, transistors, integrated circuits. The microprocessor has had a dramatic influence on the PID controller. Practically all PID controllers made today are based on microprocessors. This has given opportunities to provide additional features like automatic tuning, gain scheduling, and continuous adaptation.[6]

2.3.1 Basic modes of a PID algorithm

As the name suggests, the PID algorithm consists of three basic modes, the Proportional mode, the Integral and the Derivative modes. When utilizing this algorithm it is necessary to decide which modes are to be used (P, I or D) and then specify the parameters (or settings) for each mode used. Generally, three basic algorithms are used P, PI or PID.

2.3.2 Proportional (P) algorithm

The mathematical representation of this algorithm is:

$$\frac{mv(s)}{e(s)} = k_c \tag{2.10}$$

$$mv(t) = mv_{ss} + k_c e(t) \tag{2.11}$$

The proportional mode adjusts the output signal in direct proportion to the controller input (which is the error signal, *e*). The adjustable parameter to be specified is the controller gain, *K*.

The larger *K* the more the controller output will change for a given error. For instance, with a gain of 1 an error of 10% of scale will change the controller output by 10% of scale. Many instrument manufacturers use Proportional Band (PB) instead of *K*. The time domain expression also indicates that the controller requires calibration around the steady-state operating point. This is indicated by the constant term mv_{ss}. This represents the steady-state' signal for the *mv* and is used to ensure that at zero error the current value is at setpoint. In the Laplace domain this term disappears, because of the 'deviation variable' representation. A proportional controller reduces error but does not eliminate it (unless the process has naturally integrating properties), i.e. an offset between the actual and desired value will normally exist.

2.3.3 Proportional Integral (PI) algorithm

The mathematical representation is,

$$\frac{mv(s)}{e(s)} = k_c\left[1 + \frac{1}{T_i s}\right] \tag{2.12}$$

$$mv(t) = mv_{ss} + k_c\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau\right] \tag{2.13}$$

The additional integral mode (often referred to as reset) corrects for any offset (*error*) that may occur between the desired value (*setpoint*) and the process output automatically over time. The adjustable parameter to be specified is the integral time T_i of the controller. Reset is often used to describe the integral mode. Reset is the time it takes for the integral action to produce the same change in *mv* as the P modes initial change.

Figure 2.4 shows the output that would be obtained from a PI controller given a step change in error. The output immediately steps due to the P mode. The magnitude of the step up is $K \cdot e$. The integral mode then causes the *mv* to 'ramp'. Over the period 0 – T_i the *mv* again increases by $K \cdot e$.

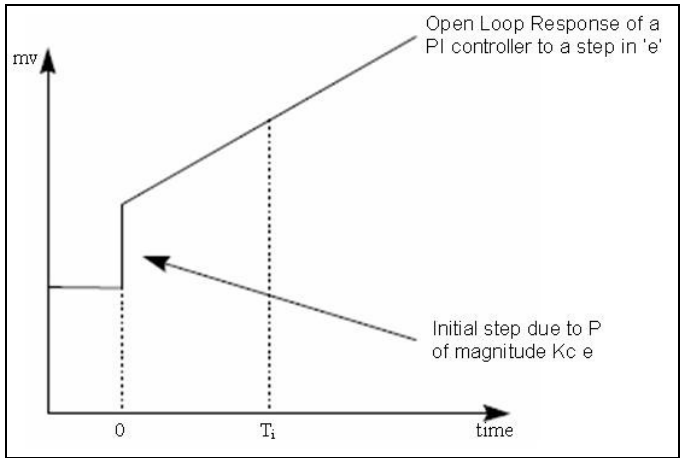


Figure 2.4: PI algorithm graph

2.3.4 Proportional Integral Derivative (PID) algorithm

The mathematical representations are:

$$\frac{mv(s)}{e(s)} = K\left[1 + \frac{1}{T_i s} + T_d s\right] \tag{2.14}$$

$$mv(t) = mv_{ss} + K\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d \frac{de(t)}{dt}\right] \tag{2.15}$$

Derivative action (also called rate or pre-act) *anticipates* where the process is heading by looking at the time rate of change of the controlled variable (its derivative). T_d is the rate time and this characterizes the derivative action (with units of minutes). In theory derivative action should always improve dynamic response and it does in many loops. Figure 2.5 shows a basic PID scheme. In others, however, the problem of noisy signals makes the use of derivative action undesirable (differentiating noisy signals can translate into excessive *mv* movement). Derivative action depends on the slope of the error, unlike P and I. If the error is constant derivative action has no effect.

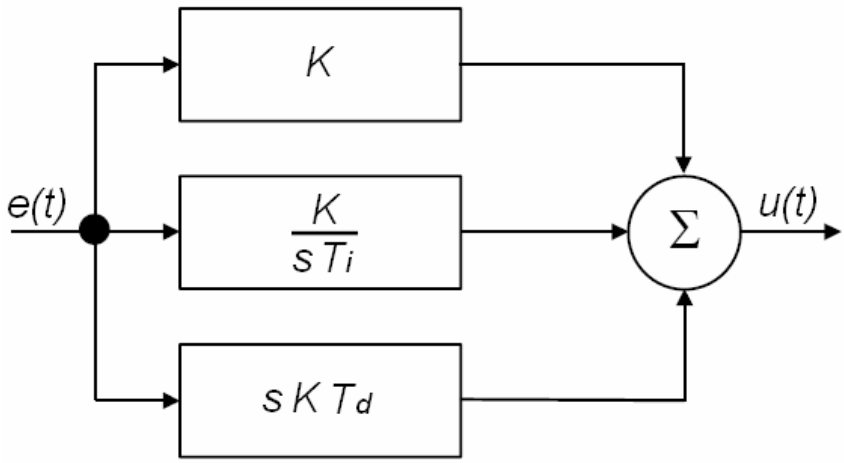


Figure 2.5: PID controller schema

2.3.5 Effects of Proportional, Integral and Derivative action

Proportional control is illustrated in Figure 2.6. The controller is given by (2.15) with T_i=∞ and T_d=0. The figure shows that there is always a steady-state error in

proportional control. The error will decrease with increasing gain, but the tendency towards oscillation will also increase.

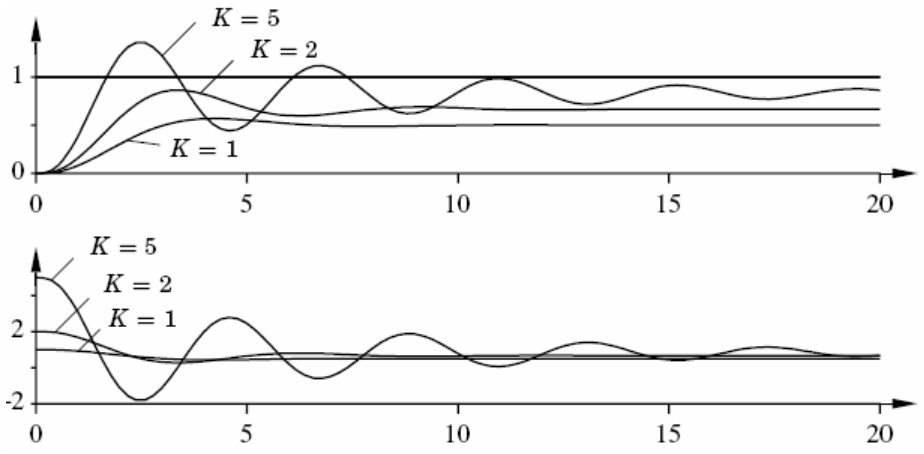


Figure 2.6: Simulation of a closed-loop system with proportional control.

Figure 2.7 illustrates the effects of adding integral. It follows from (2.15) that the strength of integral action increases with decreasing integral time T_i. The figure shows that the steady state error disappears when integral action is used. The tendency for oscillation also increases with decreasing T_i. The properties of derivative action are illustrated in Figure 2.8.

Figure 2.8 illustrates the effects of adding derivative action. The parameters K and T_i are chosen so that the closed-loop system is oscillatory.

Damping increases with increasing derivative time, but decreases again when derivative time becomes too large. Recall that derivative action can be interpreted as providing prediction by linear extrapolation over the time T_d.

Using this interpretation it is easy to understand that derivative action does not help if the prediction time T_d is too large. In Figure 2.7 the period of oscillation is about 6s for the system without derivative action.

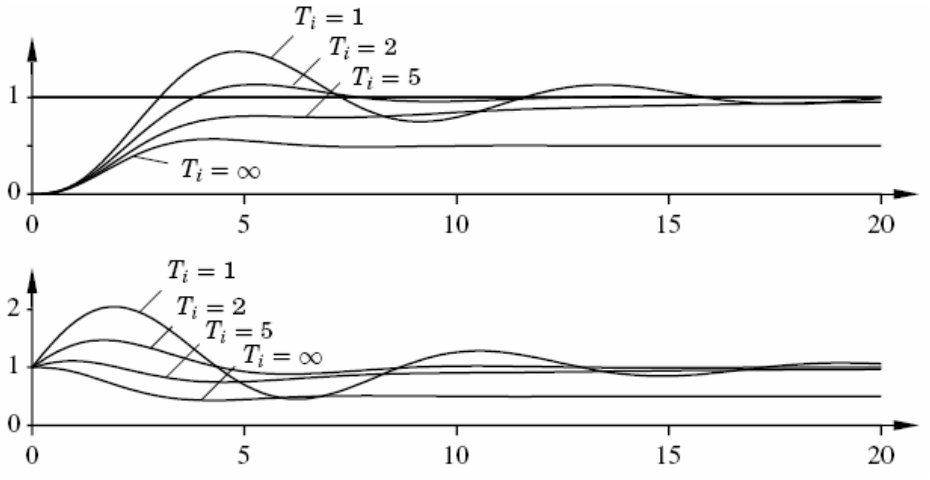


Figure 2.7: Simulation of a closed-loop system with proportional and integral control.

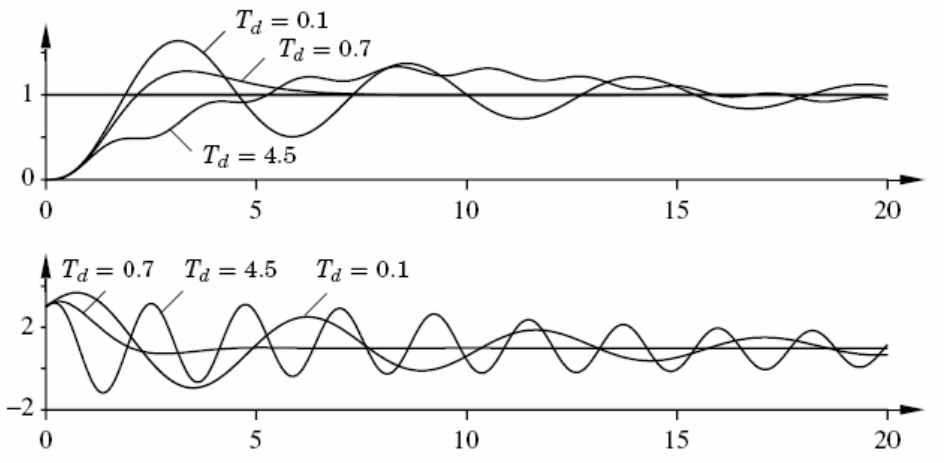


Figure 2.8: Simulation of a closet-loop system with proportional, integral and derivative control.

Derivative actions cease to be effective when T_d is larger than a 1 s (one sixth of the period). Also notice that the period of oscillation increases when derivative time is increased. There is much more to PID than is revealed by Equation 2.15.

A faithful implementation of the equation will actually not result in a good controller.

To obtain a good PID controller it is also necessary to consider:

- *Windup*;
- *Tuning*.

In the case of the PID controller these issues emerged organically as the technology developed but they are actually important in the implementation of all controllers. Many of these questions are closely related to fundamental properties of feedback.

2.3.5.1 Windup

Although many aspects of a control system can be understood based on linear theory, some nonlinear effects must be accounted for in practically all controllers. Windup is a phenomenon, which is caused by the interaction of integral action and saturations. All actuators have limitations: a motor has limited speed; a valve cannot be more than fully opened or fully closed, etc. For a control system with a wide range of operating conditions, it may happen that the control variable reaches the actuator limits.

When this happens, the feedback loop is broken and the system runs as an open loop because the actuator will remain at its limit independently of the process output. If a controller with integrating action is used, the error will continue to be integrated. This means that the integral term may become very large or, colloquially, it "winds up". It is then required that the error has opposite sign for a long period before things return to normal. The consequence is that any controller with integral action may give large transients when the actuator saturates.

The windup phenomenon is illustrated in Figure 2.9, which shows control of an integrating process with a PI controller. The initial setpoint change is so large that the actuator saturates at the high limit. The integral term increases initially because the error is positive; it reaches its largest value at time $t = 10$ when the error goes through zero. The output remains saturated at this point because of the large value of the integral term. It does not leave the saturation limit until the error has been negative for a sufficiently long time to let the integral part come down to a small level. Notice that the control signal bounces between its limits several times.

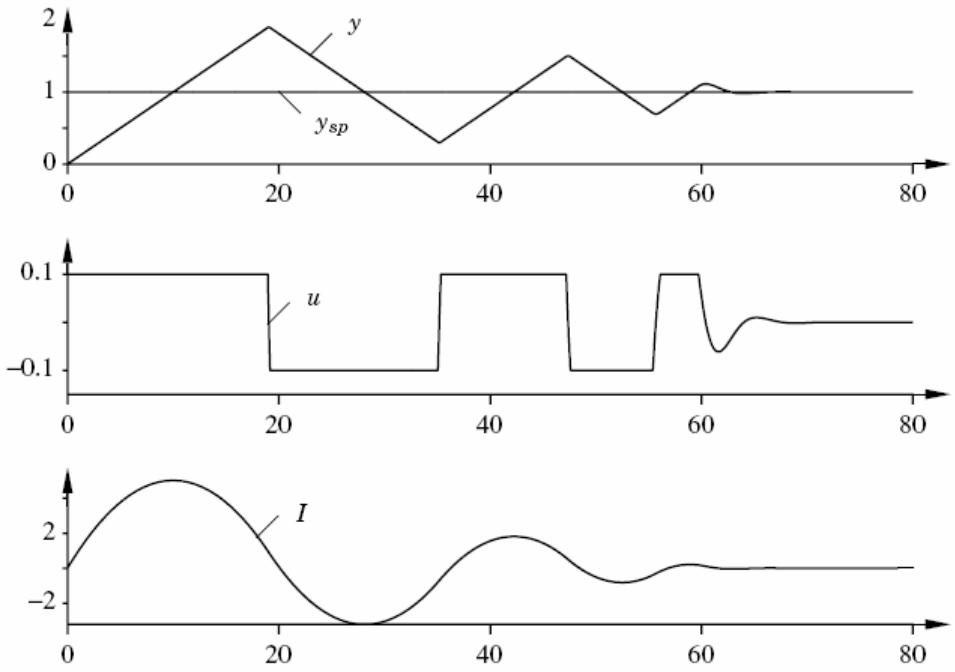


Figure 2.9: Illustration of integrator windup

The net effect is a large overshoot and a damped oscillation where the control signal flips from one extreme to the other as in relay oscillation. The output finally comes so close to the setpoint that the actuator does not saturate. The system then behaves linearly and settles.

The example show an integrator windup which is generated by a change in the reference value. Windup may also be caused by large disturbances or equipment malfunctions. It can also occur in many other situations. The phenomenon of windup was well known to manufacturers of analog controllers who invented several tricks to avoid it. They were described under labels like preloading, batch unit, etc.

Although the problem was well understood, there were often restrictions caused by the analog technology. The ideas were often kept as trade secrets and not much spoken about. The problem of windup was rediscovered when controllers were implemented

digitally and several methods to avoid windup were presented in the literature. In the following section we describe some of the methods used to avoid windup.

2.3.5.2 Setpoint limitation

One attempt to avoid integrator windup is to introduce limiters on the setpoint variations so that the controller output never reaches the actuator limits. This frequently leads to conservative bounds and poor performance. Furthermore, it does not avoid windup caused by disturbances.

2.3.5.3 Incremental Algorithms

In the early phases of feedback control, integral action was integrated with the actuator by having a motor drive the valve directly. In this case windup is handled automatically because integration stops when the valve stops. When controllers were implemented by analog techniques, and later with computers, many manufacturers used a configuration that was an analog of the old mechanical design. This led to the so called velocity algorithms. A velocity algorithm first computes the rate of change of the control signal which is then fed to an integrator. In some cases this integrator is a motor directly connected to the actuator. In other cases the integrator is implemented internally in the controller. With this approach it is easy to avoid windup by inhibiting integration whenever the output saturates. This method is equivalent to back-calculation, which is described below. If the actuator output is not measured, a model that computes the saturated output can be used. It is also easy to limit the rate of change of the control signal.

2.3.5.4 Back-calculation and Tracking

Back-calculation works as follows: When the output saturates, the integral term in the controller is recomputed so that its new value gives an output at the saturation limit. It is advantageous not to reset the integrator instantaneously but dynamically with a time constant T_t. Figure 2.10 shows a block diagram of a PID controller with anti-windup based on back-calculation.

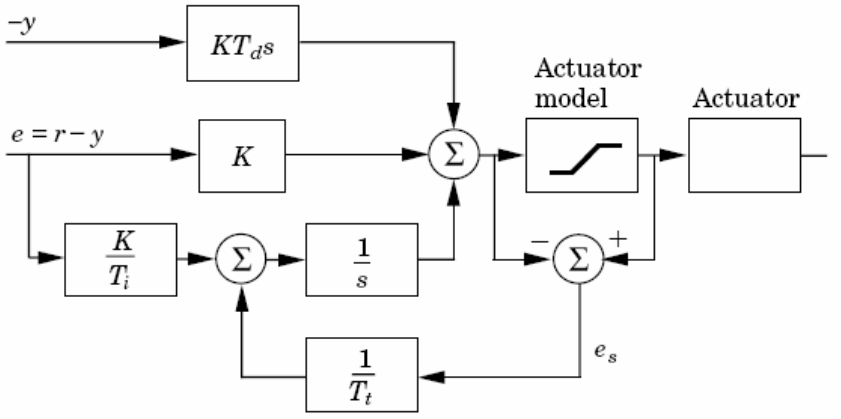


Figure 2.10: Controller with anti-windup

The system has an extra feedback path that is generated by measuring the actual actuator output and forming an error signal (e_s) as the difference between the output of the controller (v) and the actuator output (u). Signal e_s is fed to the input of the integrator through gain 1/ T_t. The signal is zero when there is no saturation.

Thus, it will not have any effect on the normal operation when the actuator does not saturate. When the actuator saturates, the signal *es* is different from zero.

The normal feedback path around the process is broken because the process input remains constant. There is, however, a feedback path around the integrator.

Because of this, the integrator output is driven towards a value such that the integrator input becomes zero.

The integrator input is:

$$\frac{1}{T_t} e_s + \frac{K}{T_i} s \tag{2.16}$$

where e is the control error. Hence,

$$e_s = -\frac{KT_t}{T_i} e \tag{2.17}$$

in steady state. Since $e_s = u - v$, it follows that

$$v = u_{\lim} + \frac{KT_t}{T_i} e \tag{2.18}$$

where u_{lim} is the saturating value of the control variable. This means that the signal v settles on a value slightly out side the saturation limit and the control signal can react as soon as the error changes time. This prevents the integrator from winding up. The rate at which the controller output is reset is governed by the feedback gain, $1/T_t$, where T_t can be interpreted as the time constant, which determines how quickly the integral is reset. We call this the tracking time constant. It frequently happens that the actuator output cannot be measured. The anti-windup scheme just described can be used by incorporating a mathematical model of the saturating actuator, as is illustrated in Figure 2.10. Figure 2.11 shows what happens when a controller with anti-windup is applied to the system simulated in Figure 2.9. Notice that the output of the integrator is quickly reset to a value such that the controller output is at the saturation limit, and the integral has a negative value during the initial phase when the actuator is saturated. This behavior is drastically different from that in Figure 2.9, where the integral has a positive value during the initial transient.

Also notice the drastic improvement in performance compared to the ordinary PI controller used in Figure 2.9.

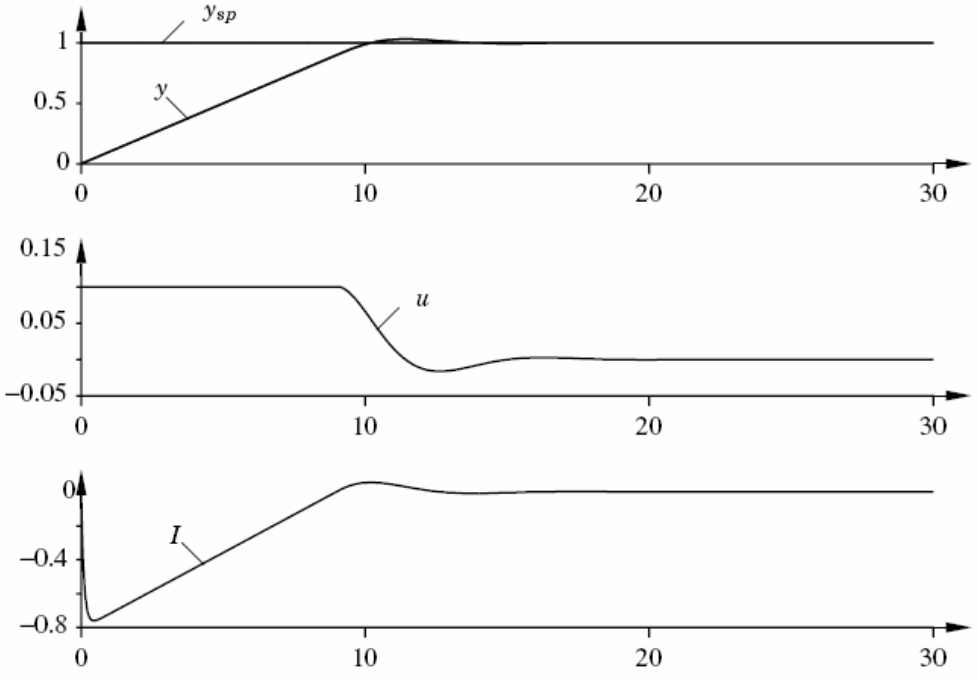


Figure 2.11: Controller with anti-windup applied to the system of Figure 2.10

The effect of changing the values of the tracking time constant is illustrated in Figure 2.12. From this figure, it may thus seem advantageous to always choose a very small value of the time constant because the integrator is then reset quickly. However, some care must be exercised when introducing anti-windup in systems with derivative action.

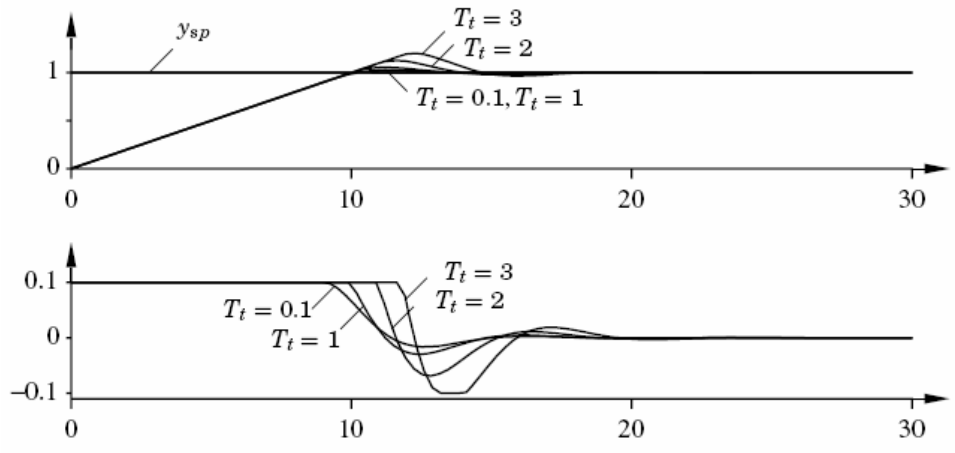


Figure 2.12: The step response of the system in Figure 2.10. for different values of T_t.

If the time constant is chosen too small, spurious errors can cause saturation of the output, which accidentally resets the integrator. The tracking time constant T_t should be larger than T_d and smaller than T_i.

A rule of thumb is suggested by equation (2.19).

$$T_t = \sqrt{T_i T_d} \tag{2.19}$$

2.3.5.5 Controllers with a Tracking Mode

A controller with back-calculation can be interpreted as having two modes: the normal *control mode*, when it operates like an ordinary controller, and a *tracking mode*, when the controller is tracking so that it matches given inputs and outputs. Since a controller with tracking can operate in two modes, we may expect that it is necessary to have a logical signal for mode switching. However, this is not necessary, because tracking is automatically inhibited when the tracking signal w is equal to the controller output. This can be used with great advantage when building up complex systems with selectors and cascade control.

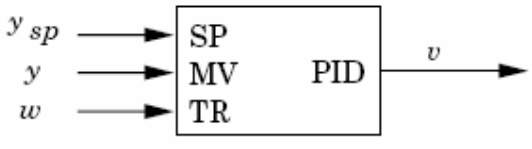


Figure 2.13: simplified representation of PID module with tracking signal (TR).

Figure 2.13 shows a PID module with a tracking signal. The module has three inputs: the setpoint, the measured output, and a tracking signal. The new input TR is called a tracking signal because the controller output will follow this signal. Notice that tracking is inhibited when $w = v$. Using the module the system shown in Figure 2.10 can be presented as shown in Figure 2.14.

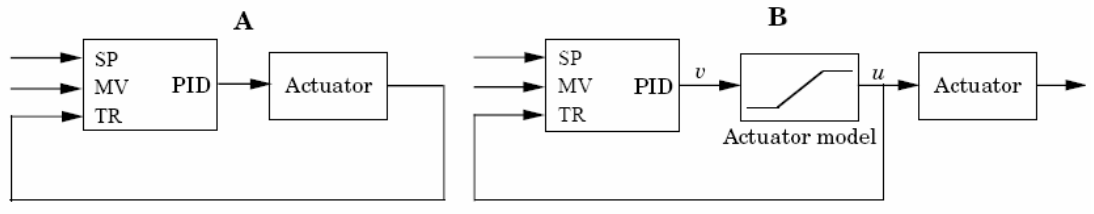


Figure 2.14: Representation of the controllers with anti-windup in Figure 2.10 using the basic control module with tracking shown in Figure 2.13.

2.3.6 PID Tuning

Controller tuning methods provide the controller parameters in the form of formulae or algorithms. They ensure that the obtained control system would be stable and would meet given objectives. These methods require certain knowledge about the controlled process. This knowledge, which depends on the applied method, usually translates into a transfer function.

The objectives which should be achieved by the application of the control system are associated with the following control system features:

- Regulating performance;
- Tracking performance;
- Robustness;
- Noise attenuation.

Often, the desired objectives put contradictory demands on the values of the controller parameters, so that various trade-offs have to be made. The objectives can be stated in many ways such as through:

- Specifications within the time domain;
- Specifications within the frequency domain;
- Robustness specifications;
- Other specifications.

The specifications within the time domain give some values related to the shape of control system signals in the time domain.

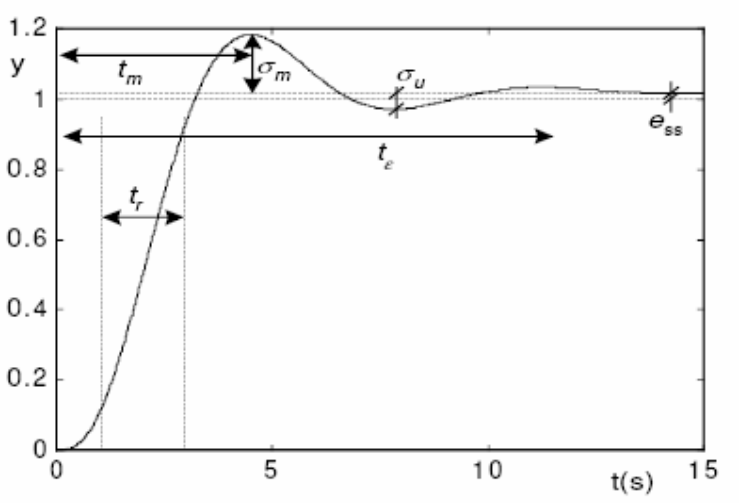


Figure 2.75: Specifications in Time domain

Figure 2.15 shows a typical output signal *y(t)*, a response to set-point change. Specification values within the time domain are marked on it: overshoot σ_m, undershoot σ_u, rise time t_r, time of first maximum t_m, settling time t_ε, and steady-state error e_{ee}.

Similar specifications within the time domain are used to describe characteristics of the control system response to load disturbance: peak perturbation σ_{dm} and disturbance settling time $t_{d\varepsilon}$.

Some controller tuning methods include recommendations of a suitable controller structure and its parameters. Tuning methods for the fixed structure of PID controllers are presented here. P, PI, and PD controllers are considered as special cases of PID controller. The tuning methods for PID controllers can be grouped according to their nature and usage, as follows:

- Heuristic methods evolved from practical experience in PID controller tuning;
- Frequency methods employ frequency characteristics of the controlled process to tune PID controller parameters;

- Analytical methods calculate PID controller parameters from analytical or algebraic relations that define control system by direct calculation;
- Loop-shaping methods seek to shape the open-loop transfer function of the control system into a desirable form;
- Optimization methods obtain PID controller parameters from different optimization algorithms;
- Methods in which PID controller represents a restriction of possible controller structure (e.g. PID controller tuning in the framework of Internal Model Control);
- Methods for tuning a PID controller which functions as a part of an advanced control strategy (e.g. usage of PI controller in dead-time compensating controllers).

The above groups do not sharply distinguish and some methods may belong to more than one group. An important criterion in the evaluation of the presented tuning methods is the suitability of a particular method for the on-line usage.

This especially refers to the possibility to use a particular method for automatic tuning. They are distinguished in open-loop and closed-loop tuning methods. These methods are very useful in case that the plant transfer function isn't known.

The most common closed-loop tuning methods are those developed by Ziegler-Nichols and Cohen-Coon. The methods are based on characterization of process dynamics by a few parameters and simple equations for the controller parameters. They consider a general plant of the form:

$$G_0(s) = \frac{A_0 e^{-s\tau}}{1 + \Delta T s} \tag{2.20}$$

It is surprising that the methods are so widely referenced because they give moderately good tuning only in restricted situations.[7]

2.3.6.1 Ziegler-Nichols Methods

Ziegler and Nichols have introduced a useful methodology for controller tuning. It consists of a simple experiment with a controlled process and extracts some of its features. Once the experiment is completed, the method provides tables by which it is possible to calculate the controller parameters. The tuning tables were developed through numerous experiments which involved different processes. The goal of the design was to find a controller which gives the quarter amplitude damping (*QAD*) ratio of the control systems in response to load disturbance.

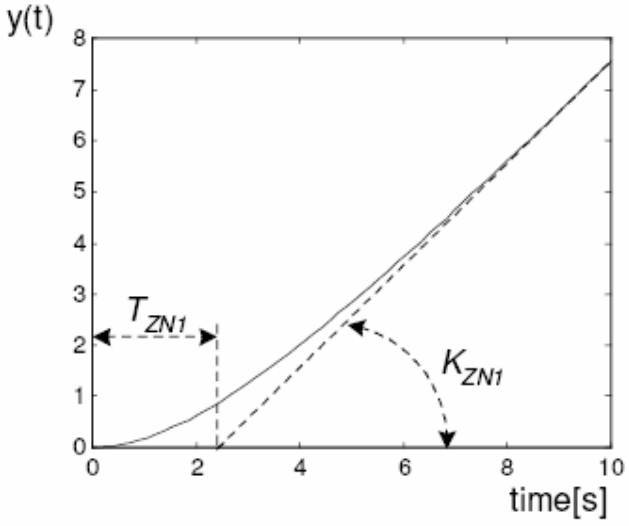


Figure 2.16: Experiment with process reaction curve

This design specification arises from empirical observations and has been used traditionally, but gives too oscillatory control systems. Ziegler and Nichols considered P, PI, and PID controllers in their work.

The first experiment is considerable in open-loop and consists of measuring apparent dead-time T_{ZN1} and the maximum slope of the response on the process reaction curve (response to step set-point change) K_{ZN1}. The measurements are shown in Figure 2.16, and relations for obtaining controller parameters in Table 2.1.

Controller	K_P	T_I	T_D
P	$1/(T_{ZN1}\ K_{ZN1})$	-	-
PI	$0.9/(T_{ZN1}\ K_{ZN1})$	$3\ T_{ZN1}$	-
PID	$1.2/(T_{ZN1}\ K_{ZN1})$	$2\ T_{ZN1}$	$T_{ZN1}/2$

Table 2.1: Ziegler-Nichols controller tuning rules – Reaction curve

In the second experiment, closed-loop, the process is controlled with a proportional controller.

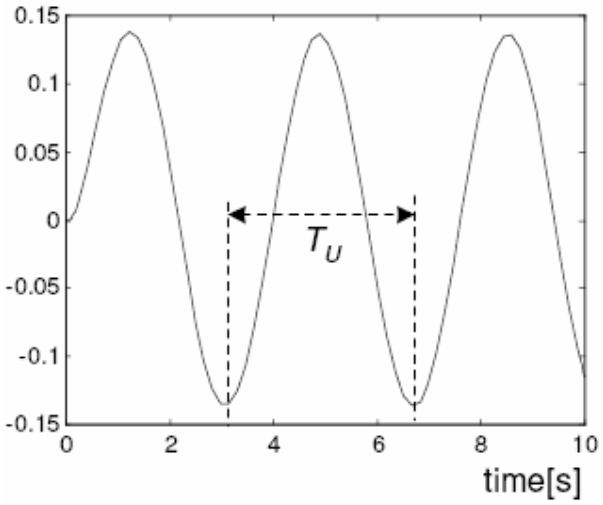


Figure 2.17: Experiment with the process on the stability limit

The gain of the controller is gradually increased until the control system reaches stable oscillations on the stability limit. The value of the controller gain K_U is called the ultimate gain, and the oscillation period T_U is called the ultimate period.

The two values serve as the basis for the calculation of the controller parameters (see Table 2.2). Figure 2.17 shows a typical process output *y(t)* during such an experiment.

Controller	K_P	T_I	T_D
P	0.5 K_U	-	-
PI	0.45 K_U	0.85 T_U	-
PID	0.6 K_U	0.5 T_U	0.125 T_U

Table 2.2: Ziegler-Nichols controller tuning rules – oscillation criterion

2.3.6.2 Cohen-Coon Method

This method determines the adjustable parameters for a desired degree of stability. It is based on a general plant transfer function that is showed in (2.20). In the equation appears a gain A_0, a delay τ and a time constant ΔT.

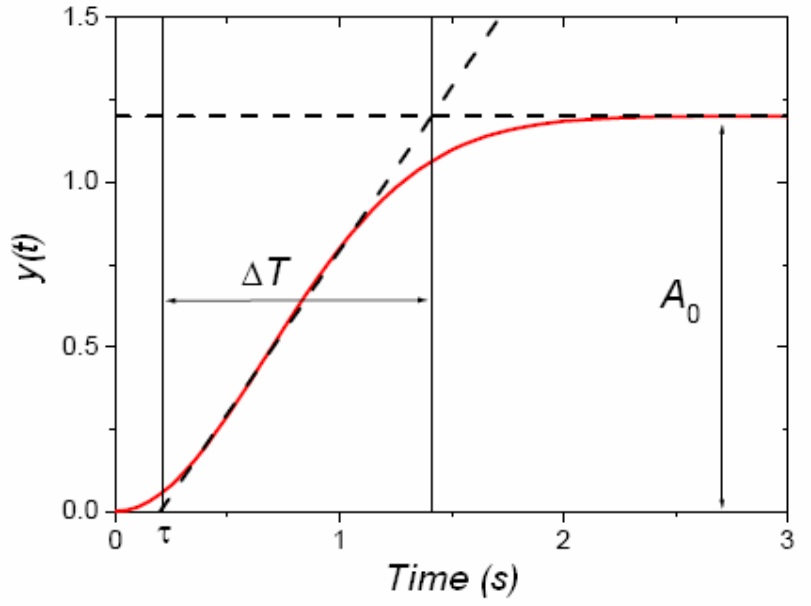


Figure 2.18: Cohen-Coon – reaction curve

This parameters can be obtained by experimental tests in open-loop configuration, is requested to apply a Heaviside step input at the plant and, to measure the time response

of the system. They are determined by an experimental data fit obtained by approximate the measure of the plant output transient at the tangent straight line in the flex point how shows Figure 2.18 where:

The Cohen-Coon method has small gain margin and phase margin when the process dead-time is short. This problem decreases when the dead-time of the process increases. This is why the (CC) tuning design is often used with a process that presents a large dead-time. Cohen-Coon recommended the following settings, showed in Table 2.3.

	k	$1/T_i$	T_D
P	$\frac{1+R/3}{A_0 R}$		
PI	$\frac{0.9+R/12}{A_0 R}$	$\tau \cdot \frac{30+3R}{9+20R}$	
PID	$\frac{1.33+R/4}{A_0 R}$	$\tau \cdot \frac{32+6R}{13+8R}$	$\tau \cdot \frac{4}{11+2R}$

Table 2.3: Cohen-Coon controller tuning rules

Where $R = \tau/\Delta T$.

CHAPTER 3

SPECIFICATION, DESIGN AND IMPLEMENTATION OF AN AUTOPILOT SYSTEM BASED ON PID CONTROLLERS

3.1 Introduction

The aim of this chapter is to make an analysis of the control problem, getting an appropriated control system able to realize an efficient flight control in the most common flight conditions.

First, problem specifications and physical limitations will be illustrated; then, we proceed to design the controllers by a mathematical approach based on control system theory.

Last, this chapter gives the mathematical formulations of the controller and then, in the next chapter will be possible to implement these in C++ algorithms.

3.2 Autopilot specifications

Before begin to explain the autopilot system design, it is necessary to explain what we had done. First, the autopilot must be able to control the following flight situation:

- Heading control and maintenance;
- Altitude control and maintenance;
- GPS navigation control and maintenance;

Every functionality of this autopilot needs one or more controller for actuate the flight control surfaces. In the following sections there is an explication of everyone of this controllers that we have implemented inside the main control system.

3.2.1 *Heading* controller specifications

In *Chapter 1*, the flight control surfaces problem was explicate with some practical examples. We know that for obtain an efficient yaw axis control we have to actuate the ailerons or the rudder. Usually rudder is actuated only for little route correction in

runway approaches and needs low air speed for to prevent surface damages; in normal flight condition, is required to actuate the ailerons for obtain a better lateral control.

The aim of this kind of controllers is to maintain an assigned route (or heading) and then, actuate ailerons for to correct some route perturbation due to an adverse weather conditions or in case of two or more engines, an asymmetrical power supply.

We want to realize a controller that reads the current heading from inertial navigation board systems or from GPS, in case that our aircraft is equipped with it, and compare it with the assigned heading. It mean that every sample rate the controller know the current error on yaw axis. To correct this error, the ailerons must be extended of a determined deflection for a determined time. The ailerons deflection determines a roll moment. The roll angle can't be arbitrary and needs a further control because an excessive roll angle can be produce stall and aircraft crashes down. Then an inner controller for actuate roll control is indispensable. Roll angle has upper and downer limitations; for Cessna C172SP we have considered an absolute maximum roll angle of ±25°. Then the first specifications for heading controller are max and min roll angle. The other specifications are the maximum and the minimum of ailerons deflection; in this case we have to consider the specific aircraft ailerons but we working with a virtual aircraft and then that limitation are imposed by flight simulator axis input.

In Microsoft® Flight Simulator every one of the available aircrafts have a fixed values for ailerons deflection input, that values are ±16383 but, for more safety and more flight stability, we consider this limit values: ±9200 (about 56% of max input). The considered aileron input isn't relative to a real aileron actuator but the proportional between input value given and its effect are the same of a common real surface control system. In the following table are showed all the specifications for the heading controller design.

	min	max
Roll angle (deg)	-25°	+25°
Aileron input*	-0.56	+0.56

Table 3.1: Range of roll angle and aileron input values – *: normalized, aileron input limit is ±16383

3.2.2 *Altitude* controller specifications

To achieve an altitude controller is very important to consider vertical speed and trim. Elevator is the surface that actuates this control; elevator acts on pitch axis producing a pitch moment that creates a pitch down or up of the aircraft.

To achieve a descent or a climb with a fixed rate, is necessary to control aircraft elevator trim. Trimming the aircraft we obtain a constant descent or climb with a fixed pitch angle. To obtain a desired pitch trim, engine power must be stable; for this reason in every simulation, we consider the same value of engine power fixed in 2100 rpm. This parameter must be adjusted manually because the autopilot doesn't implement an engine power control and then isn't able to keep it stable automatically.

The aim of altitude controller is to keep altitude inside a range of tolerance. Like heading controller, altitude controller presents physical limitation due to the same reasons explained above; these limitations are resumed in the Table 3.2.

	min	max
Vertical Speed (feet/sec)	-700	+500
Elevator incremental input	-200	+200

Table 3.2: Range of vertical speed and elevator input values

The best way to obtain a efficient altitude controller is to make a crosscheck of vertical speed (by trim) and *indicated airspeed* (KIAS) (by engine power). This controller presents some limitations offers by a lack of power control. This is a start point for future developments.

3.2.3 *GPS Navigation* controller specifications

It is based on the same scheme of heading controller; it uses a continuous feedback from GPS relative to the new heading that aircraft has to follows. A GPS flight plan is useful to obtain route maintenance, the controller actuates the same control surfaces used to achieve an aircraft lateral control and for this reason the specifications for this controller are those showed in Table 3.1. The only difference between this controller and in lateral controller the reference heading (*setpoint*) was been fixed while now the reference heading is calculated by GPS's microprocessor that return it.

For the reason above, this controller needs of a GPS navigation plan, available in *Microsoft® Flight Simulator 2004*.

3.3 Autopilot design

This paragraph consists in the explanation of design and implementation procedures carried out for developing the controllers by using the specifications described in the items written above. Simulation is the basic component of this work, all the algorithms realized are developed using experimental methods and practical testing. Before to design the controllers it was necessary to create an algorithm that provides to reads and prints in a file all the parameters that we want to use for monitoring and managing the aircraft virtual model. This algorithm is based on a *plug-in* named *FSUIPC* that allows data exchanging from *Microsoft® Flight Simulator* (FS) to the our programming language. The next sub-paragraphs are dedicated to Microsoft® Flight Simulator and

FSUIPC. This algorithm, named *Plotter*, contains a great number sub-programs that provides to read and write some variables from FS, i.e. for determine the current heading error we have to read this value from FS using *Plotter* configured for return this variable. For this reason, each controller needs of *Plotter.cpp*, the file that contains *Plotter* algorithm, and the related header file named *Control.h*. Controller structures are very similar; each one controller has a part able for data reading/writing followed by the control *core* that provides to determine the position error and the control surface input that needs its correction. The better solution for controller section of algorithm is to implement PID controllers; the reason is that the controlled variables are really vital for aircraft and requires a continuous monitoring. Luckily, the existence of PID controller permits to design a robust control system with a very simple mathematical structure.

3.3.1 *Microsoft® Flight Simulator* and the plug-in *FSUIPC*

As described above, FS and its plug-in FSUIPC are fundamental for the development of this autopilot. FS is a simulator characterized by a high realism level, it's used by commercial pilot student to improve their ability to fly. It is also used for researches and developments in aeronautical engineering. FS allows simulating several flight condition relatives to weather, weight and force solicitations. It allows simulating failure at instruments and engines. Every FS aircraft has equipped with GPS on-board receiver and it consents to improve the accuracy of the data read from inertial navigation instruments like as gyros.

The first problem that we have encountered is the data import and export from/to FS. Controller needs feedback, for this reason data extraction from FS is vital. To allows this we have used the plug-in FSUIPC made by *Pete Dowson*, it's a freeware available on the web. The plug-in package contains source code and headers file to achieve to

access at the FS variables. The most important functions for FS data I/O are contained in the file *IPCuser.c* and the related header file is *IPCuser.h*. Another header file is furnished in the package, it allows the use of IPCuser.c, it is *FSUIPC_User.h*.

In FSUIPC each FS variable has a univocal identifier (offset), a data typology and a size in bytes. The FSUIPC reading/writing functions are listed below:

```
BOOL FSUIPC_Open(DWORD dwFSReq, DWORD *pdwResult)
```
(3.1)

It is a preliminary function that permits the access to the variable; it is like a *key* that opens the FS *lock*. Every routine that provides for data extraction or introduction, can be begins with this called;

```
BOOL FSUIPC_Read(DWORD dwOffset, DWORD dwSize, void *pDest,
DWORD *pdwResult)
```
(3.2)

It provides to read variable data from FS, it needs of four arguments which are the variable offset, variable size (bytes), the destination filename and a indicative of processing status.

```
BOOL FSUIPC_Write(DWORD dwOffset, DWORD dwSize, void *pSrce,
DWORD *pdwResult)
```
(3.3)

It allows data writing in FS and its arguments are the same of `FSUIPC_Write()` exeption for the filename because here represents the source for the data introduction.

```
BOOL FSUIPC_Process(DWORD *pdwResult)
```
(3.4)

This function (3.4) actuates the processing of data after reading/writing FS. It is fundamental because without this, the I/O with FS it's impossible.

```
void FSUIPC_Close(void)
```
(3.5)

It represents the *key* above mentioned, it close the access to the FS data. For extract or insert again it's necessary to call the function `BOOL FSUIPC_Open()`. (3.6) and (3.7) shows how is possible read or write from/to FS by using FSUIPC's functions.

```
short ReadAilerons(void)
{
	short ailerons;
	DWORD pdwResult;
	FSUIPC_Open(SIM_FS2K4, &pdwResult);
	FSUIPC_Read(0x0BB6, 2, &ailerons, &pdwResult);
	FSUIPC_Process(&pdwResult);
	FSUIPC_Close();
	return ailerons;
}
```
(3.6)

```
void AileronsInput(short input)
{
	DWORD pdwResult;
	FSUIPC_Open(SIM_FS2K4, &pdwResult);
	FSUIPC_Write(0x0BB6, 2, &input, &pdwResult);
	FSUIPC_Process(&pdwResult);
	FSUIPC_Close();
}
```
(3.7)

These are the functions used to make the controller algorithms, every components of this autopilot system implements FSUIPC functions. Reading and writing carry out inside FS, opening and closing operations. Each one of these operations requires a

defined time that coincides with the FS screen refresh time, it's estimate in about 50*msec* (18Hz).

3.3.2 The function *Plotter*

This function allows to read some flight parameters and to control primary flight control surfaces like ailerons and elevators. It implements the functions listed in the next page:

```
void  AileronsInput(short input)
```
(3.8)

writes in FS the desired value of ailerons input;

```
float CTS_GPS(void)
```
(3.9)

read the required HDG correction to guide the aircraft on a GPS pre-assigned route;

```
void  ElevTrimInput(short input)
```
(3.10)

writes in FS the desired value of elevators trim input;

```
short ReadAilerons(void)
```
(3.11)

read the current ailerons deflection status;

```
int   ReadAltitude(void)
```
(3.12)

read the current aircraft absolute altitude;

```
short ReadElevator(void)
```
(3.13)

read the current elevators deflection status;

```
DWORD ReadKias(void)
```
(3.14)

read the current indicated airspeed (KIAS);

```
float ReadPitch(void)
```
(3.15)

reads the current aircraft pitch angle;

```
float ReadRoll(void)
```
(3.16)

reads the current aircraft roll angle;

```
short ReadTrim(void)
```
(3.17)

reads the current elevators trim deflection status;

```
int   ReadVS(void)
```
(3.18)

reads the current aircraft vertical speed;

```
float ReadHDG (void)
```
(3.19)

reads the current aircraft heading;

```
float ReadHDG_GPS(void)
```
(3.20)

reads from GPS, the current aircraft heading;

```
float RequiredHDG_GPS(void)
```
(3.21)

reads from GSP, the heading value required by current flight plan.

After given these definitions, it's possible to carry on controller design. The functions implemented inside Plotter.cpp are very useful to obtain an efficient class of controller

based on the data extracted from FS. Next step shows how this functions works inside the control loops, they support the main structures of controllers. The skeleton of each controller is composed by reading and writing functions and a structure that provide to calculate the entity of surface actuation.

3.3.3 *Heading* controller design

The problem of heading control and maintenance can be schematized like as showed in Figure 3.1: this figure shows the variables which this controller manages.

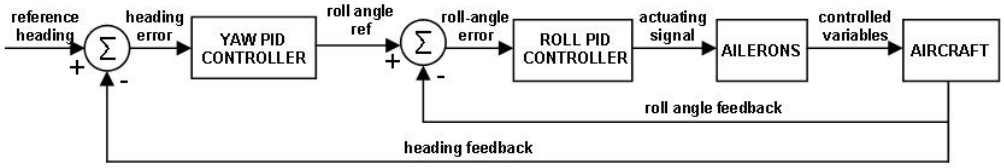


Figure 3.4:Scheme of feedback heading controller

We can see that in the figure above there are two control loops, an inner and an outer loop that provides to control directly roll angle by ailerons and indirectly heading by ailerons through roll angle. The choice of this kind of scheme is motivated by the necessity to obtain an efficient yaw axis control by ailerons actuation.

The main problem given by yaw control is that the heading adjustment directly by ailerons are produce a great unstably situation because it can't control the roll angle that is subject to physical limitations. Then we have considered the roll limitation showed in Table 3.1. Within this arrangement the risk is that the roll angle maybe excessive and then it can make loss of air lift on the wings.

Outer controller allows controlling yaw by roll because it calculates the required roll angle needed to correct the current yaw error. To regulate roll angle is required an inner control loop that receive as input signal, the roll reference above mentioned, and

provides movements the ailerons by an adjust value. This structure is very complex because implements two PID controllers that controls respectively roll and yaw by ailerons. The greatest difficulties of this configuration are offers by PIDs tuning and anti-windup. For that reasons the tuning of this controller is very complex because it presents two PIDs and then the setting must be done according these two devices. Dividing this scheme in two part is possible to make a single analysis of each PID.

First, the inner-loop, showed in Figure 3.2, include the reading/writing functions *ReadRoll* and *AileronsInput* showed in (3.16) and (3.8) respectively, these permits at controller to calculate the current roll error and to actuate the ailerons by the desired value.

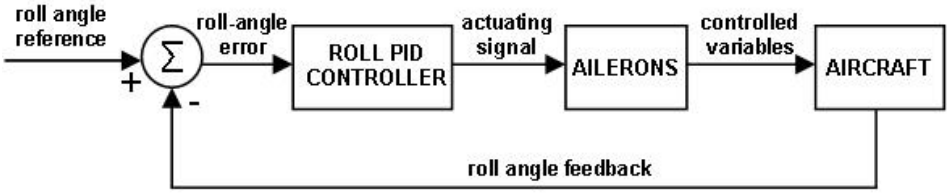


Figure 3.5: scheme of inner-loop – roll axis controller

It take as input the roll angle reference returned by outer-loop, reads by aircraft, though function *ReadRoll*, the current roll angle, determines the roll angle error and put it into the PID controller that provides to return the input to send to ailerons, ailerons actuation is provided by calling *AileronsInput* function. Roll PID controller is characterized by two parameters: K_r and T_r which are respectively the controller proportional gain and the integral time. This parameter are determinates with a tuning operation described after that. In this configuration of controller, is better to call it PI controller because the derivative component doesn't works. Now we proceed to analyse the outer control loop with the same approach used for the inner-loop. The control system scheme is that showed in Figure 3.3:

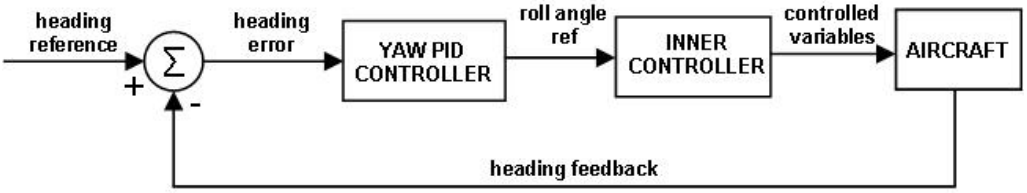


Figure 3.6: scheme of outer-loop – yaw axis controller

The reference signal can be given by GPS or can be established by pilot. There are then, two options. In case that we give as input the GPS navigation required heading, we have to implement, inside the controller, the function *RequiredHDG_GPS* (3.21) for read from GPS flight plan the heading which the aircraft has to follow. In the sum knot heading error is calculate considering the feedback signal represented by current heading that is read from FS by function *ReadHDG_GPS* (3.20). The best accuracy of GPS make that we use this instrument only for have aircraft positioning data. Another way is represented by giving a pre-assigned heading and t then the controller will works to keep it. In this case we have to implement the function *ReadHDG* (3.19) for data reading. The determined heading error is putted into the controller that returns the roll target signal needed to correct error. Sampling rate depends by the number of FS opening/closing operation that algorithm has to carry out. Normally, this rate fluctuate around 1*sec*. A robust controller needs anti-windup measures. To obtain a good controller is necessary to limit the integral term implicit of the PID structures. To solve this problem, is necessary to design a desaturator that limits the input value, comparing the current input with a upper and downer values derived from the problem physics. In this controller we need of two desaturator because we have two PID (or better PI) controller.

A desaturator block works in the fashion showed below, in the (3.22), and is represented by the scheme showed in Figure 3.4.

$$sat(u) = \begin{cases} u_{\min} \Rightarrow u \le u_{\min} \\ u \Rightarrow u_{\min} < u < u_{\max} \\ u_{\max} \Rightarrow u \ge u_{\max} \end{cases} \tag{3.22}$$

Where u_{min} and u_{max} represents the maximum and the minimum values that actuator allowed.

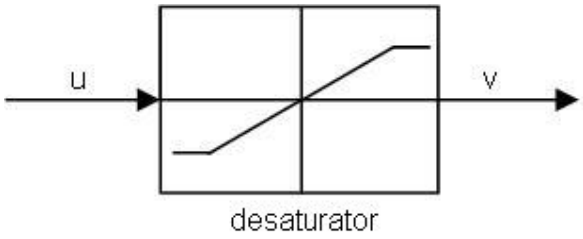


Figure 3.7: representation scheme of a desaturator function

Adding this further utility, the resultant system is that showed in Figure 3.5.

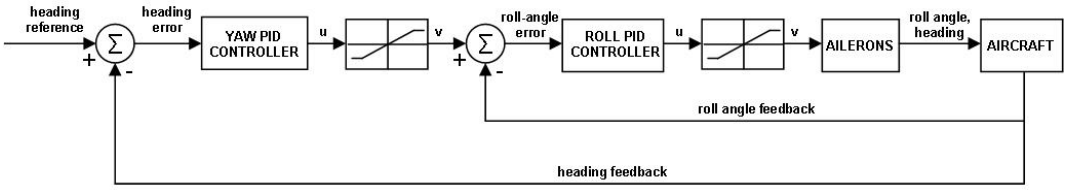


Figure 3.8: scheme of heading controller with desaturators

To implement a desaturator block, the algorithm needs of *if* structures that compares the input *u* and returns the actuation signal *v* without saturation effects. The solution adopted in this heading controller is shown in (3.23).

With these arrangements, controller design returns a robust and stable control system that provides every 1 second to adjust the ailerons for to achieve a heading control.

```
if (u_input > u_max)
      u_input = u_max;
if (u_input < u_min)
      u_input = u_min;
```
(3.23)

Now, the last problem to consider is the controller tuning. Each controller needs only of two constant K_p and T_i because we have considered a P-I controller configuration for both the control loops. Cohen-Coon method, treated in *Chapter 2*, allows PIDs tuning in a very simple fashion because doesn't needs of system transfer function that presents a elevated level of complexity. This method operates in open-loop configuration and with proportional part only, then without integrative effects, we have proceeded to set the roll controller before to set the yaw controller.

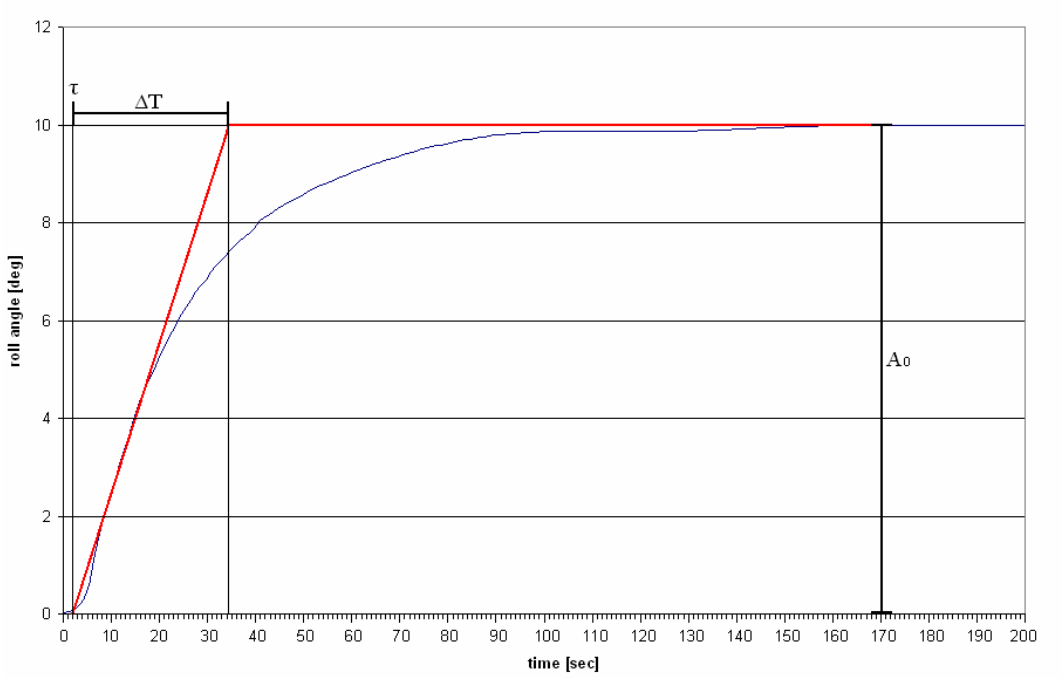


Figure 3.9: Cohen-Coon tuning method – Roll PID graph

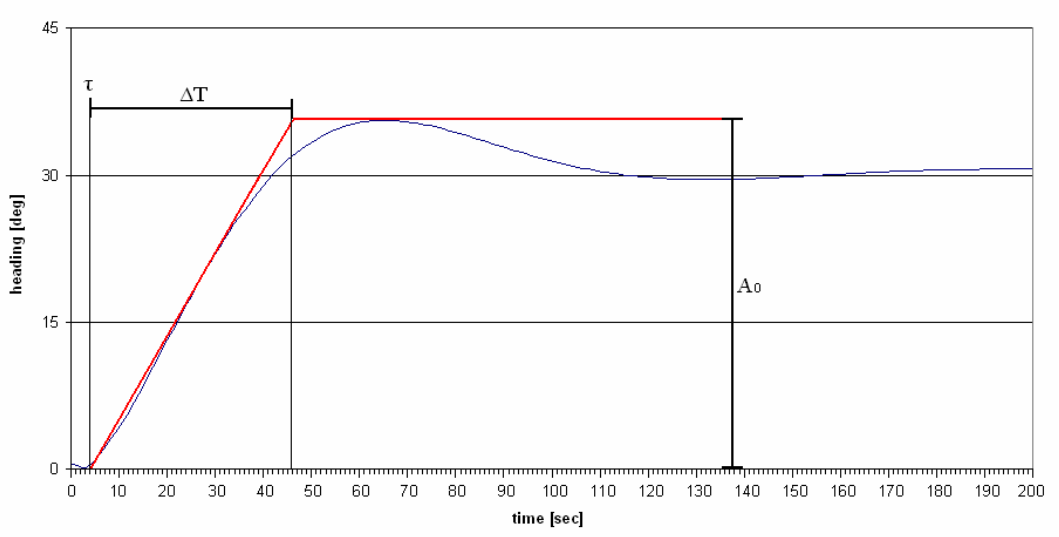


Figure 3.10: Cohen-Con tuning method – Yaw PID graph

Roll PID setting consists to put as input a specific value of roll that we want obtain, for simplicity we have chosen a roll value of 10 degrees to avoid input instability due to wind or turbulences. In Figure 3.6 is showed the graph relatives to the tuning procedure of roll controller.

To tune the yaw PID we have used an input of 30 degrees for the same reason explained above, the relate graph is showed in Figure 3.7. In Table 3.3 there are the values obtained by this graph.

	τ	ΔT	A_0
Inner loop (ROLL controller)	**2.0**	**34.0**	10.0
Outer loop (YAW controller)	4.0	46.0	36.0

Table 3.3: Cohen-Coon tuning setting table

These values are the starting points to obtain the PID parameters, using the Cohen-Coon tuning table showed in Table 2.3 we obtain the parameters listed in the table 3.4, their calculation is explained in the following steps.

For to calculate the proportional gain K_p we have used the following equation:

$$K_p = \frac{0.9 + R/12}{A_0 R} \tag{3.24}$$

Then for to calculate the integral time T_i we have used this other:

$$T_i = \tau \cdot \frac{30 + 3R}{9 + 20R} \tag{3.25}$$

Where $R = \tau / \Delta T$.

	K_p	T_i
Inner loop (ROLL controller)	**1.54***	**5.93**
Outer loop (YAW controller)	**0.29***	**11.27**

Table 3.4: PID controllers settings for heading PID controllers – *: non-normalized

Note that we have used a non-normalized input and then a normalization operation was been necessary to obtain the true PID values. This value was been developed by experimental tests in open-loop configuration, we have used the following values for the proportional gain of these two controllers:

	K_p
Inner loop (ROLL controller)	**500.00**
Outer loop (YAW controller)	**2.00**

Table 3.5: K_p values used in Cohen-Coon tuning procedures

Multiplying respectively the values showed in Table 3.5 by the values of Table 3.4 we obtain the wanted PID parameter, these parameters are showed in Table 3.6.

	K_p	T_i
Inner loop (ROLL controller)*	**770.00**	**5.93**
Outer loop (YAW controller)*	**0.58**	**11.27**

Table 3.6: Normalized PID parameters obtained by Cohen-Coon tuning

In the figures 3.8 and 3.9 are showed the performance of this controller in terms of roll angle and aileron actuations. Figure 3.8 shows that in this configuration, the maximum and the minimum of the roll angle is limited in the range [-25° ; +25°] like we have imposed. The same reasoning is applicable to Figure 3.9 because it presents a input value that don't exceeds the established range [-9216 ; +9216].

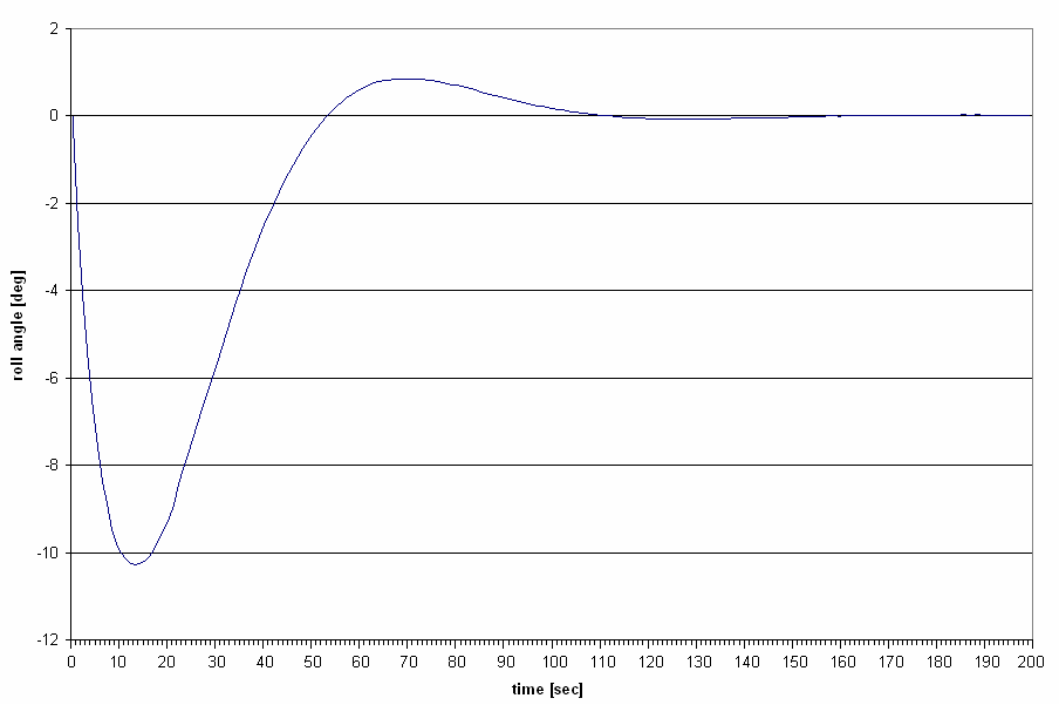


Figure 3.11: C-C tuning method graph – roll angle performance after an input of 30°

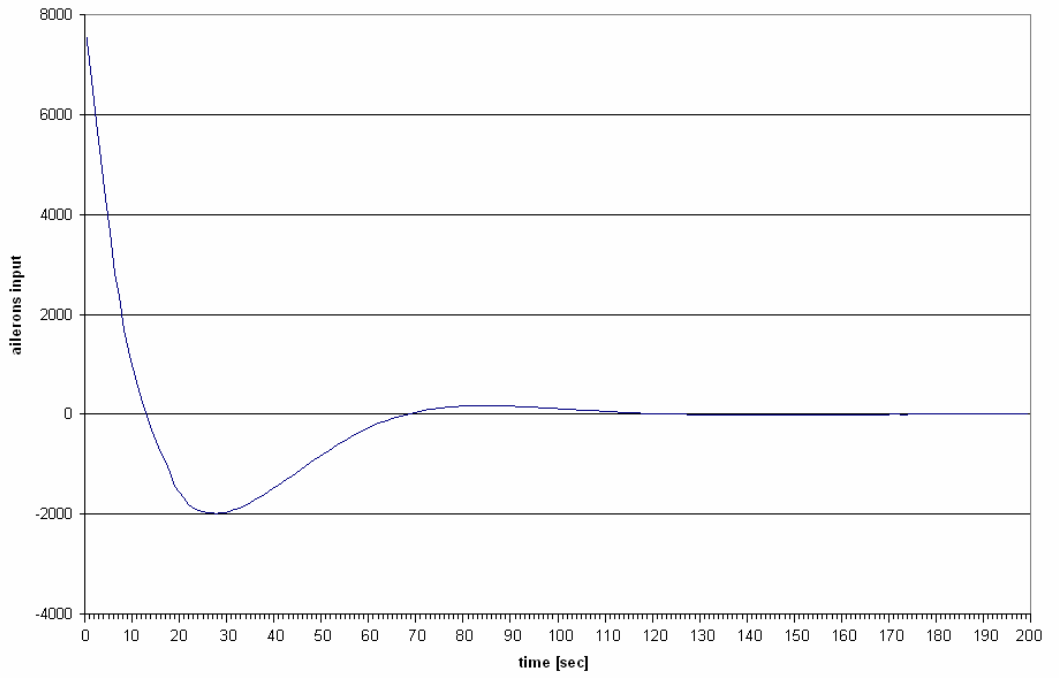


Figure 3.12: C-C tuning method graph – ailerons actuations performance after an input of 30°

3.3.4 *Altitude* controller design

This controller provides to change the aircraft altitude and to keep it stable. In Figure 3.9 is showed the control system schematization.

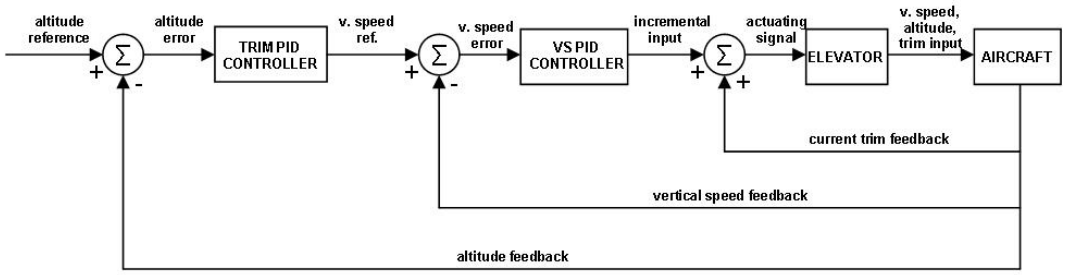


Figure 3.13: Scheme of feedback Altitude controller

This system presents two control loops and three feedback lines, two of these provides to feed the PID controller, the third instead, to calculate the exact amount of elevator

trim input by summing the previous trim value and the actual trim input calculated by PID controller.

The two control loops, an outer and an inner, provides to control indirectly aircraft altitude by elevator trim. Outer loop controls the vertical speed, it takes as input the altitude reference value and, after the processing, it returns the vertical speed target needed to lead the aircraft to the new altitude. Inner loop provides directly to the elevator trim control, it takes as input the vertical speed returned by outer loop, in this fashion we obtain a control system that regulates vertical speed for to climb or descent the aircraft. In case that we want keep an assigned altitude, the vertical speed target is equal to 0 feet/sec.

The choice of this configuration is imposed by the necessity of consider that vertical speed target needs of input limitation because the aircraft can't bear strong solicitations given by high speed. We must impose another limitation on the amount of adjusting signal (*trim*) to give to actuators. These are resumed in the Table 3.2.

In this way, the controller imposes the maximum and the minimum of vertical speed, preventing the risk of an excessive wing load due to an excessive vertical speed with consequents structural damages or loss of aircraft control.

To achieve a good altitude controller could be optimum to implement a engine power controller able to control the engine RPM. The aim of this work isn't to implement engine controller and it is limits only to surfaces control. Then it is necessary to control manually the engine RPM value for obtaining an optimal performance.

Dividing the system showed in Figure 3.9 we obtain two schemes relatives to the two control loops. In Figure 3.10 is showed a scheme relative to the inner control loop. This loop implements some FSUIPC functions seen above as *ReadVS* (3.18) that reads the current amount of aircraft vertical speed, *ReadTrim* (3.17) that read the current amount of elevators trim and *ElevTrimInput* (3.10) that provides to actuate the elevators.

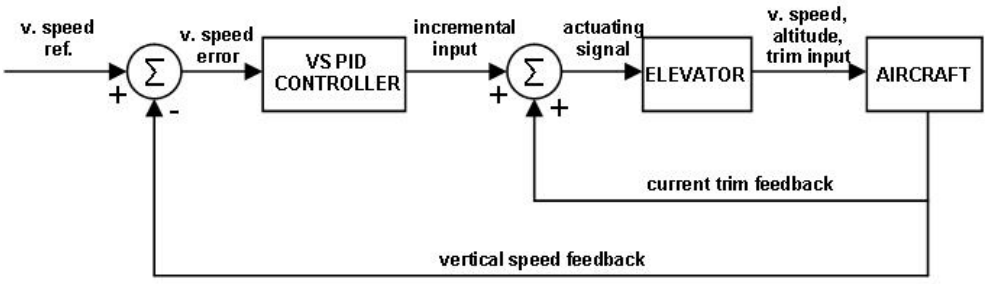


Figure 3.14: scheme of inner-loop – vertical speed controller

This controller determines the current vertical speed error resulted by difference between vertical speed reference and current vertical speed read by *ReadVS*, this error is sent to PID controller that provides to determine the amount of trim input to correct the error, this input isn't easy to handle, the effective trim input is determined by summing this input value with the previous value, given by a feedback line. This true input value is sent to elevators and in a reasonable time the aircraft finds and keeps the assigned altitude.

Outer loop works to calculate the vertical speed reference that must be sent to inner loop. Its scheme is illustrated in Figure 3.11.

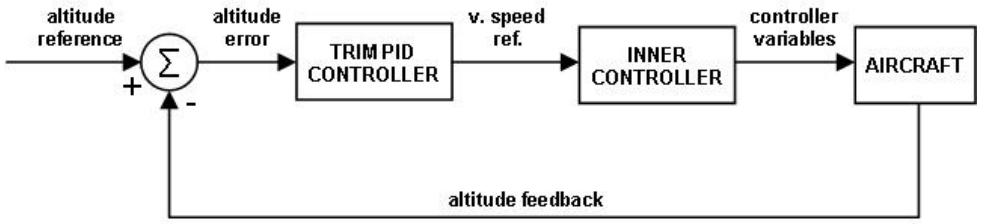


Figure 3.15: scheme of outer-loop – altitude controller

This control loop requires a function that reads the current altitude (*measured in feet*) named *ReadAltitude* (3.12). It takes as input the altitude reference given by pilot and calculates the difference between it and the current altitude of aircraft, then this error is sent to a PID controller that determines the vertical speed needed to correct the error.

Each input of these loops needs desaturation, then inserting the desaturator blocks the system became like that Figure 3.12 shown.

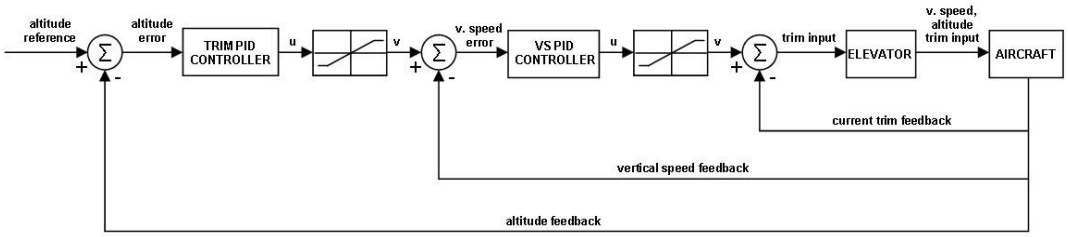


Figure 3.16: scheme of altitude controller with desaturators

The desaturators works in the fashion explained in the paragraph relative to heading controller design.

Like heading controller, this one needs to be tuned and to carry out this procedure we use the Cohen-Coon tuning method for each PID. The controller structure allows to consider a P-I controller configuration because we haven't implemented a derivative action, then the tuning problem consists in the evaluation of K_p and T_i.

The first step of this procedure is to tune the inner loop (vertical speed controller) to obtain better performance in the second tuning step. In open-loop configuration, we have sent to the controller a vertical speed input equal to 300 *feet/sec* and we have obtained the graph of Figure 3.14 that shows the vertical speed performance. The aircraft reacts with a gradual vertical speed change.

For the outer loop (altitude controller) we have sent an altitude input of 500 feet, in this case the elevator trim provides to carry out the current altitude to the reference value using the vertical speed controller that we have tuned before. The resultant graph is showed in Figure 3.15.

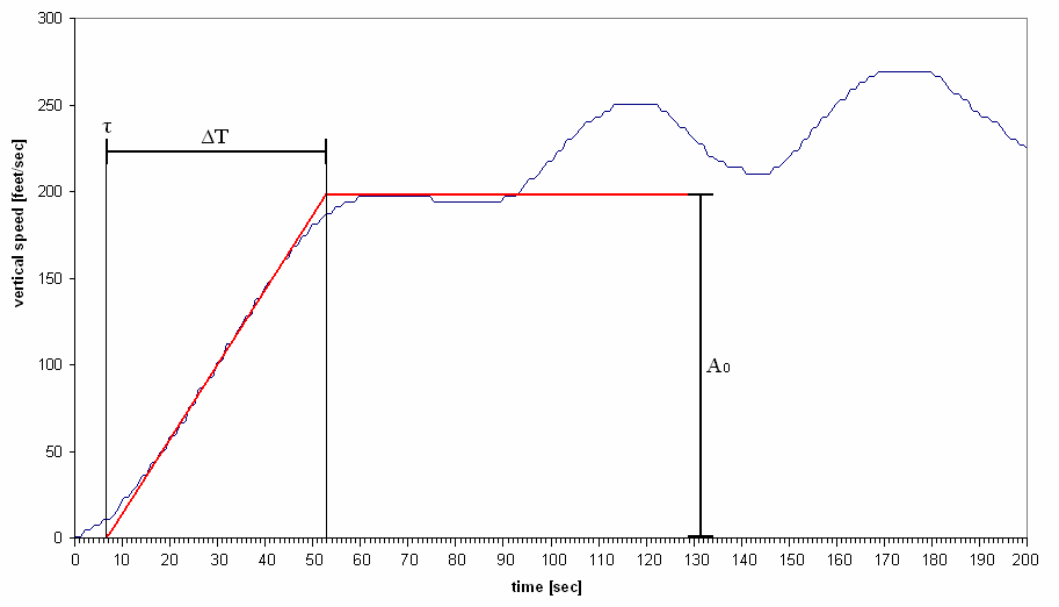


Figure 3.17: Cohen-Coon tuning method graph – vertical speed performance

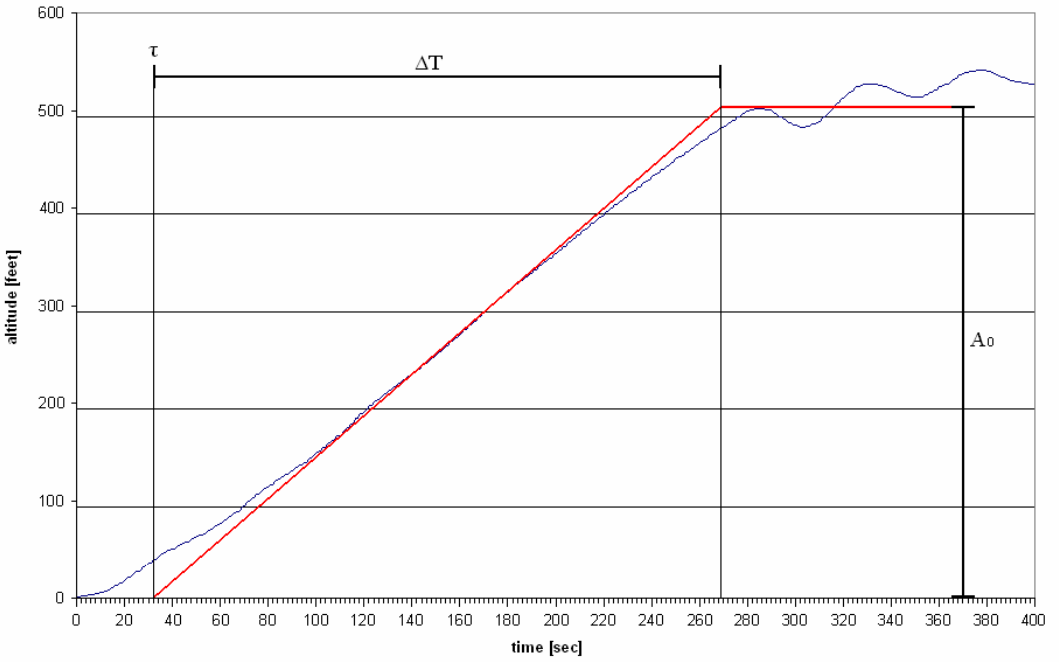


Figure 3.18: Cohen-Coon tuning method graph – altitude performance

The constants τ, ΔT and A_0, that are indicated in that graph are resumed in the setting table 3.7 showed in the next page.

	τ	ΔT	A_0
Inner loop (VS controller)	**6.50**	**53.00**	**197.00**
Outer loop (H controller)	**32.00**	**269.00**	**500.00**

Table 3.7: Cohen-Coon tuning setting table – Altitude controller

Using 3.24 and 3.25 to determine the PID constants we obtain the value listed in Table 3.8 showed below.

	K_p	T_i
Inner loop (VS controller)	**11.30***	**17.24**
Outer loop (H controller)	**7.65***	**85.37**

Table 3.8: PID controllers settings for altitude PID controllers – *: non-normalized

To calculate the normalized PID constants we have to multiply these values by the values showed in the Table 3.9 and to scale the resultants by the factor used as input in the simulation, these are:

- 300 for the vertical speed controller;
- 500 for the altitude controller.

	K_p
Inner loop (ROLL controller)	**0. 20**
Outer loop (YAW controller)	**2.00**

Table 3.9: K_p values used in Cohen-Coon tuning procedures

The constants of this controller are showed in Table 3.10.

	K_p	T_i
Inner loop (ROLL controller)	**2.26**	**17.24**
Outer loop (YAW controller)	**15.30**	**85.37**

Table 3.10: Normalized PID parameters obtained by Cohen-Coon tuning – Altitude controller

In the figures 3.16 and 3.17 are showed the performance of this controller in terms of vertical speed and elevator trim actuations. Figure 3.16 shows the vertical speed response of the inner loop when an altitude input of 500 feet is sent to the full system, the vertical speed reference given as input to inner loop is provided by outer loop. Note that the vertical speed doesn't exceeds the limits imposed by specifications.

Figure 3.17 shows the performance of elevator trim input produced by the same altitude input mentioned above for the inner loop. In this case the specification limits acts on the differential input and not on the maximum and the minimum total input that are limited by the nature of actuators.

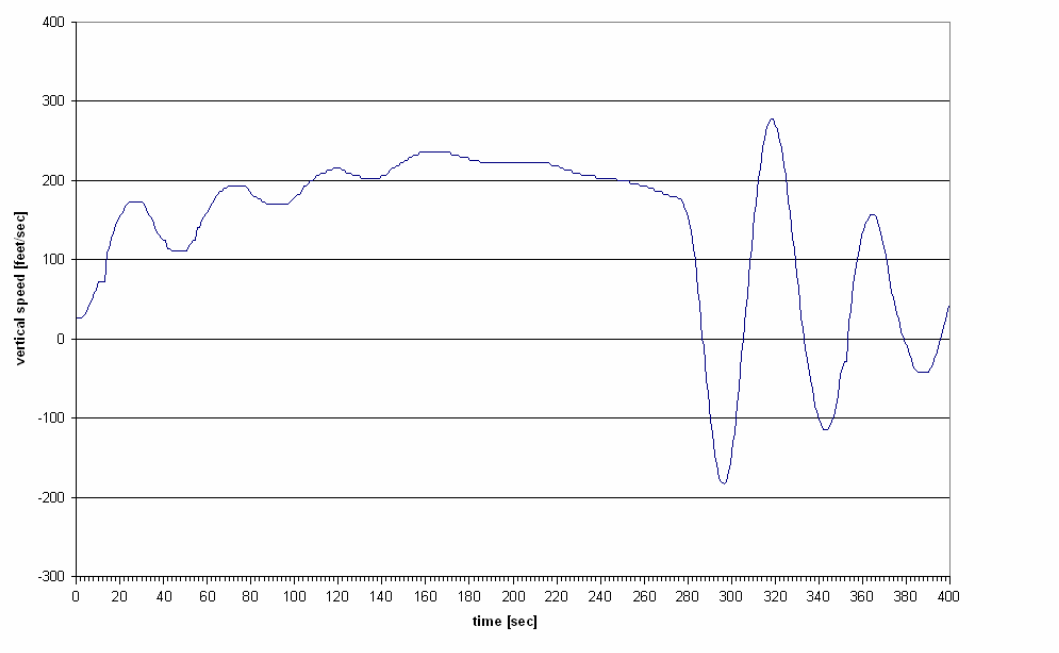


Figure 3.19: C-C tuning method graph – vertical speed performance after an input of 500 feet

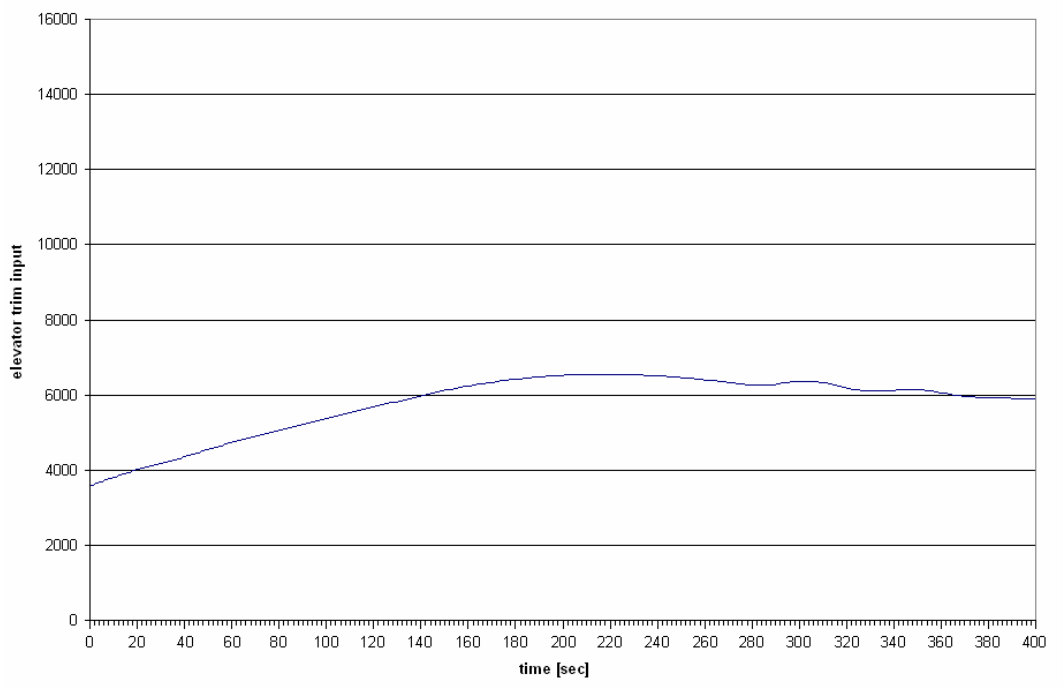


Figure 3.20: C-C tuning method graph – trim input performance after an input of 500 feet

3.3.5 *GPS navigation* controller design

This controller is based on the same concept of heading controller; its work is to follow a variable heading given by GPS. The imagine showed in Figure 3.10, represents an on-board GPS device implemented in the most common commercial aircrafts. In the centre of the display, we can see the schematization of the aircraft that is positioned in a point that represents its true GPS position relatively to the earth surface and the pink line represents the assigned GPS route. In the example is showed a route able to link *Francisco Sà da Carneiro Airport* (LPPR) and *Lisbon Airport* (LPPT).

This controller works to guide and keep the aircraft on the pink line showed in the figure above and carry out this operations, the GPS calculate automatically the value of heading error.

Then we need to read the GPS course to set (CTS) that represents the correction signal needed to guide the aircraft on-route.

Figure 3.21: GPS display – is visualized the route from (Porto) LPPR and Lisbon (LPPT)

The scheme of this controller is illustrated in Figure 3.11.

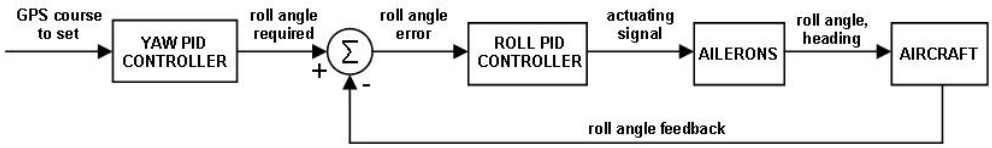


Figure 3.22: Scheme of feedback GPS navigation

All the considerations done about anti-windup, PID tuning for *heading* controller, are used to design this controller, how said, it is also an heading controller and the only difference is that its reference signal isn't fixed but change every instant because the airplane moves and changes its position.

The GPS course to set needs a continuous refresh and then the system has to read this value every loop tick. The function that reads the CTS value is called *CTS_GPS* (3.9). Like as *heading* controller, *GPS navigation* controller needs an anti-wind up measure,

is possible to implement saturators analogously. The Figure 3.12 shows the full system.

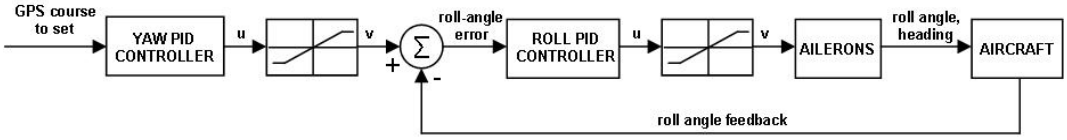


Figure 3.23: anti-windup scheme of GPS navigation controller

Notice that the PID controller constants are the same used in heading controller.

3.4 Controllers implementation

This paragraph is dedicated to controller's implementation, in some ways this is similar to the previous paragraph but the most important difference is which in this, numerical implementation only is showed instead of the design step illustrated in paragraph 3.3.

For to realize the controllers in *C++* algorithm, we have used *Microsoft® Visual C++* that will be explained in the next section.

The numerical realization of controllers requires C++ algorithms, each one of these are characterized by a variable number of function implemented inside it. This paragraph will show all the construction phases of the above mentioned algorithms.

3.4.1 The function *lateral*

This function implements the lateral controller, named *heading controller*. In this section we have to collect all the parts developed in the paragraph 3.3.3 and to make an assembling of these in a single function. The steps of this procedure are listed below.

The first step consist to define the number of C++ functions requires from problem specification, we know that in this controller there are two PID loops and then we have

to implement these in two C++ functions. We have chosen to implement a controller function that groups the two PID loops. The architecture of C++ algorithm needs a *main* function that provides to execute all the functions called in its body. The *main* function has to call the controller function only.

The two PI controllers was been implemented taking the design specification of paragraph 3.3.3 then, we are arrived to the implementation of the following functions.

```
float PID_yaw(float HDG_ref)
```
(3.26)

This statement (3.24) represents the outer loop of this controller, it take as argument a *float* named HDG_ref that represents the heading reference signal. It contains calls to *ReadHDG* (or *ReadHDG_GPS*) for extract the heading reference from FS and implements also, a structure useful to normalize the read heading because FS returns a heading value included in the range [-180° ; +180°]. This structure is written below:

```
if (HDG_err < 0)
{
      if (HDG_err < - 180)
            HDG_out = HDG_err + 360;
      else
            HDG_out = HDG_err;
}
if (HDG_err > 0)
{
      if (HDG_err > 180)
            HDG_out = - 360 + HDG_err;
      else
            HDG_out = HDG_err;
}
if ((int) HDG_err == 0)
{
      HDG_out = HDG_err;
}
```
(3.27)

In this way, we obtain a heading normalized in the range [0° ; +360°] like the most common navigation instruments of the real aircraft. The controller implementation is described by the statement (3.26), where *Kpy* is the proportional gain, *HDG_out* is the output of saturator and *Iy* is the integral term. The reason of the negative transformation is due to the FS control axes properties.

```
roll_input = (-1) * Kpy * (HDG_out + Iy);
```
(3.28)

The integral part *Iy* (like as a generic *I*) is calculated using the statement 3.29:

```
Iy = fc * (1/Ty) * integralY;
```
(3.29)

We can see a term named *integralY* that represents the current sum of error, when the loop starts to work, this value is equal to 0. Integral term requires desaturation because maybe affected by windup, to obtain an integral term without windup, we must implement the following lines:

```
if (Iy >= limY)
      Iy = limY;
if (Iy <= -limY)
      Iy = -limY;
```
(3.30)

Note that the integral terms of all PID of this autopilot, are calculated in this way then, is unnecessary to show this calculation for each PID illustrated in this chapter. Another structure implemented inside this function is an ailerons input desaturator, which is showed in the followings statement:

```
if (roll_input > roll_max)
      roll_input = roll_max;
```
(3.31)

```
if (roll_input < roll_min)
      roll_input = roll_min;
```

PID_yaw function returns a *float* that is the roll angle target.

```
void PID_roll(float roll_ref)
```
(3.32)

Function (3.27) is the inner loop of controller and it take as argument a *float* named roll_ref that represents the output of the outer loop or rather, the roll angle target. Subtracting the current roll angle, given by a call to *ReadRoll* function, to roll reference value, we obtain the current roll error. This value is used in the control statement for determine the ailerons input needed to correct this roll error. The controller statement is showed below:

```
ails_input = (-1) * Kpr * (roll_err + Ir);
```
(3.33)

Where *ails_input* is the input given to ailerons, *Kpr* is the controller proportional gain, *roll_err* is the error of roll angle and *Ir* is the controller integral term. Here too, there is a multiplication by *(-1)*, this term is due to the same reason explained above. The input furnished to ailerons must be filtered by a desaturator to eliminate actuators saturation. This functionality is actuated by the following statement:

```
if (ails_input > ails_max)
      ails_input = ails_max;
if (ails_input < ails_min)
      ails_input = ails_min;
```
(3.34)

The ailerons deflection are regulated by calling the function *AileronsInput* that take as input the above mentioned ailerons input value. *PID_roll* function doesn't return values. The function that groups these others two functions are:

```
void lateral(void)
```
(3.35)

That contains also, the *for* loop that provides to iterate the two PID functions. For to obtain a continuous heading control, we have chosen to implement an *infinite for* loop inserting the following code line:

```
for (;;)
```
(3.36)

Then we have achieved a perpetual routine that don't *break* in any case. In the *lateral*'s body, inside the infinite loop, there is a *PID_yaw* function call that provides the roll angle target, inside the *PID_roll* function though a variable demanded for this reason. Anyway, is possible adding inside this function body, a call to Plotter function, described above in paragraph 3.3.2 for storing and printing data.

3.4.2 The Function *longitudinal*

Altitude controller is implemented in this function. Its name is due to the typology of control that it provides. In this section are explained the implementation solutions used to develop this important autopilot functionality.

Like the case of altitude controller, we need of two functions that implements the two PIDs and of a function that provides to link these controllers. It is obvious that the linking function must be called inside the main function body.

Using the specifications imposed for the design of this controller, we are able to implement a C++ function for each one controller. Is better to begin to implement the outer loop because it returns the vertical speed reference used as input of the inner loop. The function that implements the outer loop is:

```
int PID_alt(int alt_ref)
```
(3.37)

It takes as input a *int* that represents the desired altitude value that we put into the controller. It contains a call to the function *ReadAltitude* (3.12) for determine the actual altitude error. This value is inserted into a statement that represents the PID controller line that is showed below:

```
vs_input = Kpa * (alt_err + Ia);
```
(3.38)

In this statement appears *Kpa* that represents the PID proportional gain, *Ia* that represents the integral term calculated in the way showed in the previous paragraph. This statement returns a value named *vs_input* that is the required vertical speed needed to correct the altitude error. This controller needs, like the previous controller, a structure that provides to eliminate the saturation effect of actuator input. This structure is the same seen above for the heading controller and, for this specific case, is the following:

```
if (vs_input > vs_max)
      vs_input = vs_max;
if (vs_input < vs_min)
      vs_input = vs_min;
```
(3.39)

After this desaturation, the function returns *vs_input*. This value is the input of the next function that is declared like this:

```
void PID_vs(int vs_ref)
```
(3.40)

Calling the function *ReadVS* (3.18) and subtracting this value to vertical speed reference, we obtain the vertical speed value, this is the error that must be put into the PID statement for obtain the trim input to give to elevators. The control line is:

```
trim_new = Kps * (vs_err + Is);
```
(3.41)

Like the previous function, *Kps* represents the proportional gain while *Is* represents the integral term. Elevator trim presents limitations due to the physic, these limitations, discussed in the design section, must be treated with a desaturator statement, showed below:

```
if (trim_new > trim_max)
      trim_new = trim_max;
if (trim_new < trim_min)
      trim_new = trim_min;
```
(3.42)

The returned trim input isn't an absolute value, is relative to the vertical speed error then it needs to be summed to the previous trim status. For this reason the controller needs of continuous trim input reading furnished by function *ReadTrim* (3.17). The total amount of trim is given by summing the actual trim value, read from FS, with the trim value, returned by PID. The relatives statement is the following:

```
trim_act = ReadTrim();
trim_input = trim_act + trim_new;
```
(3.43)

This last calculated value is the value that function gives to elevator as input, and is put as argument into function *ElevTrimInput* (3.10).

The linker function is declared in this way:

```
void longitudinal(int alt_ref)
```
(3.44)

It implements the *infinite for loop* that we have seen in the section above and, inside this loop, there made all the control operations calling the function *PID_vs* (3.35) to calculate the vertical speed target and the function *PID_trim* (3.38) to send to the elevators the trim input that it has provide to calculate. In the tail of this function, inside the routine, is possible to call the function Plotter if we need to realize graph or extract data from FS.

3.4.3 The Function *navigation*

This function implements the GPS navigation controller studied in the paragraph 3.3.5. This function implements two functions that we know *PID_yaw* and *PID_roll*; the function *navigation* calls these two functions to determine the required ailerons deflection useful for to guide the aircraft on-route and, when the aircraft is aligned, to keep it on that. In the design phase, we have seen that a continuous refresh of GPS data is fundamental to minimize the error margin and to obtain good performance by this controller. The refresh rate depends by the system sampling rate, because the GPS data reading is effected in-loop, a high sampling frequency insure an elevated number of correction and that a smaller ailerons input, it make the system more stable. To

implement this solution, it's necessary to have an infinite *for* loop. In the followings statements is showed a part of the body of the function *navigation*:

```
for(;;)
{
      float yaw_ref = CTS_GPS();
      float roll_input = PID_yaw(yaw_ref);
      PID_roll(roll_input);
}
```
(3.45)

Calling the function *navigation* inside a *main function*, we obtain an accurate and functionally navigation control. The functionalities given by this controller, may be used in approaches to runway, it furnishes only a lateral control of the airplane and requires a *glide slope* controller for became an *automatic landing* controller. The implementation of a glide slope controller may be another possible development of this project.

CHAPTER 4

CONTROLLER TESTING RESULTS

4.1 Introduction

In this chapter are treated the aspect relative to the controllers testing, each controller is been tested, for first separated and after combined, in various flight conditions. Every test is been executes using FS to simulate the flight conditions; the function *Plotter* was been used to extract the data relatives to the simulations from FS. In the next sub-paragraphs will be explained with more details the procedures used that includes the flight conditions considered, the variables observed and the data processing ways.

4.2 Experiment description

The procedures used to make these simulations presents few but important difference that depends by the kind of controller used and that are weather conditions, aircraft weight configuration. First, we have tested the controllers separately, to obtain data on their efficiency and accuracy and then was been possible the combined testing of both controllers.

The tests are developed with the following order:

- *lateral* controller;
- *longitudinal* controller;
- *navigation* controller;
- *lateral* and *longitudinal* controllers both;
- *navigation* and *longitudinal* controllers both.

To execute this kind of test is necessary to run *Microsoft® Flight Simulator* without start the controller loops, configure a flight choosing an aircraft, i.e. Cessna C172SP "*Skyhawk*", select a departure airport or a ATC flight plan, select data and hours of flight, chose the weather conditions and the weight configuration. After this, we can start the flight, is better to bring the aircraft to an assigned altitude, i.e. 3000 feet, keep it stable manually, follow a random heading and set the engine power to 2100 RPM to

obtain a constant cruise speed keeping the vertical speed to a value next to 0. Is possible to use the FS autopilot for to carry out these positioning procedures but is necessary to set manually the engine power. When the aircraft is straight and level is possible to start the tests. The following paragraphs describes the details of each test, explaining the configurations used and discussing the obtained data. The test results will be discussed in the end of each paragraph.

4.3 *lateral* controller testing

This section is dedicated to lateral controller tests carried out in four different flight conditions described in the following table:

	weight	*weather*	*input [deg]*	*GPS*
test #1	**MAX**	**wind 340/00**	**+90°**	**OFF**
test #2	**MIN**	**wind 340/00**	**+90°**	**OFF**
test #3	**MAX**	**wind 000/16 + turbulences**	**+90°**	**OFF**
test #4	**MIN**	**wind 000/16 + turbulences**	**+90°**	**OFF**

Table 4.1: *lateral* controller testing table

To test this controller, we have used a common heading input of 90 degrees, an aircraft weight included in the range [1610 lb – 2550 lb] and we have carried out the simulation with [340/00] and without [000/16] wind. For this controller we have chosen to carry out only a quartet of tests because its performances are the same in case of turn right and in case of turn left, the chosen of a turn right is casual and there aren't physical reasons. The parameters that we have monitored and plotted are: *heading*, *roll angle*, *pitch angle* and *aileron input*.

The following graph represents the data developed by Test #1:

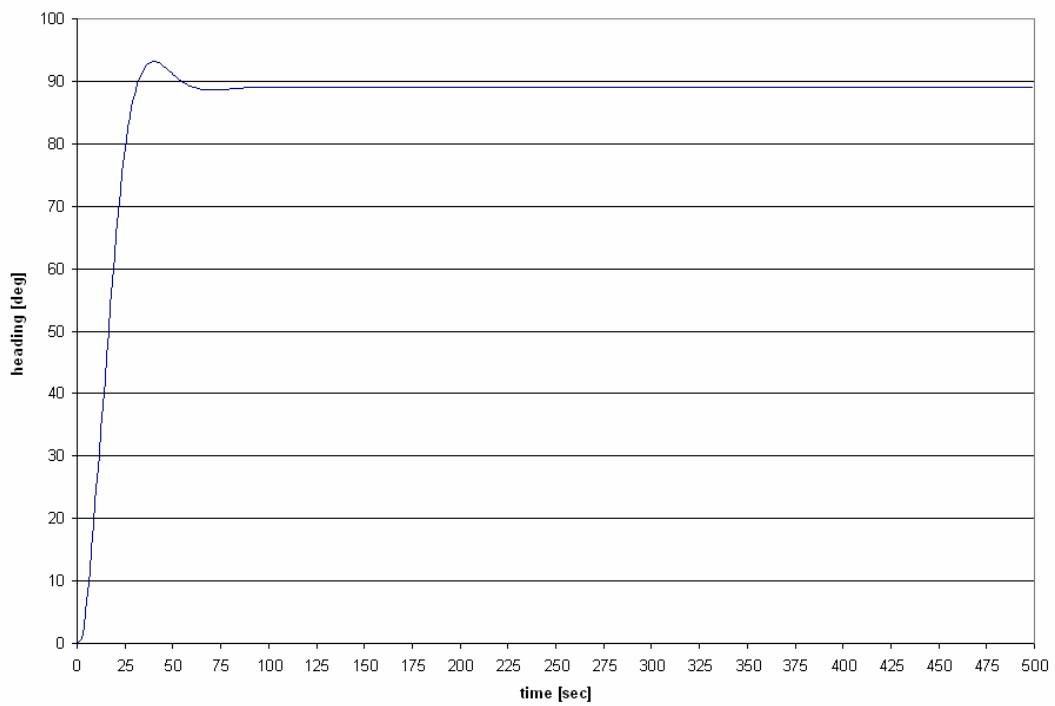


Figure 4.24: test #1 – heading response

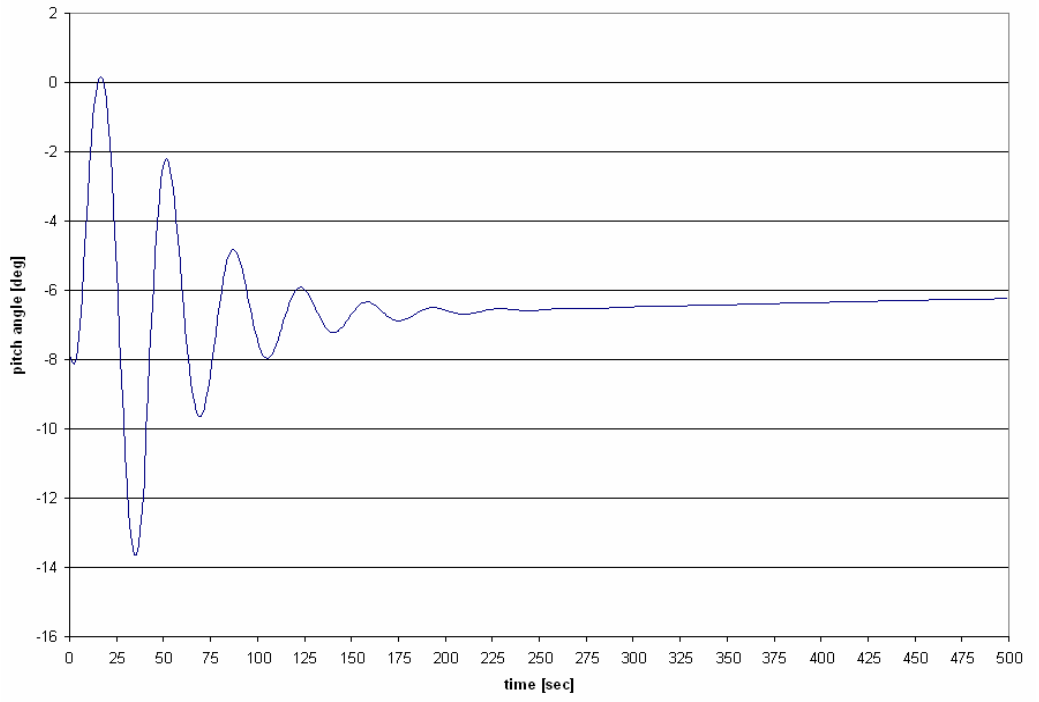


Figure 4.25: test #1 – pitch angle response

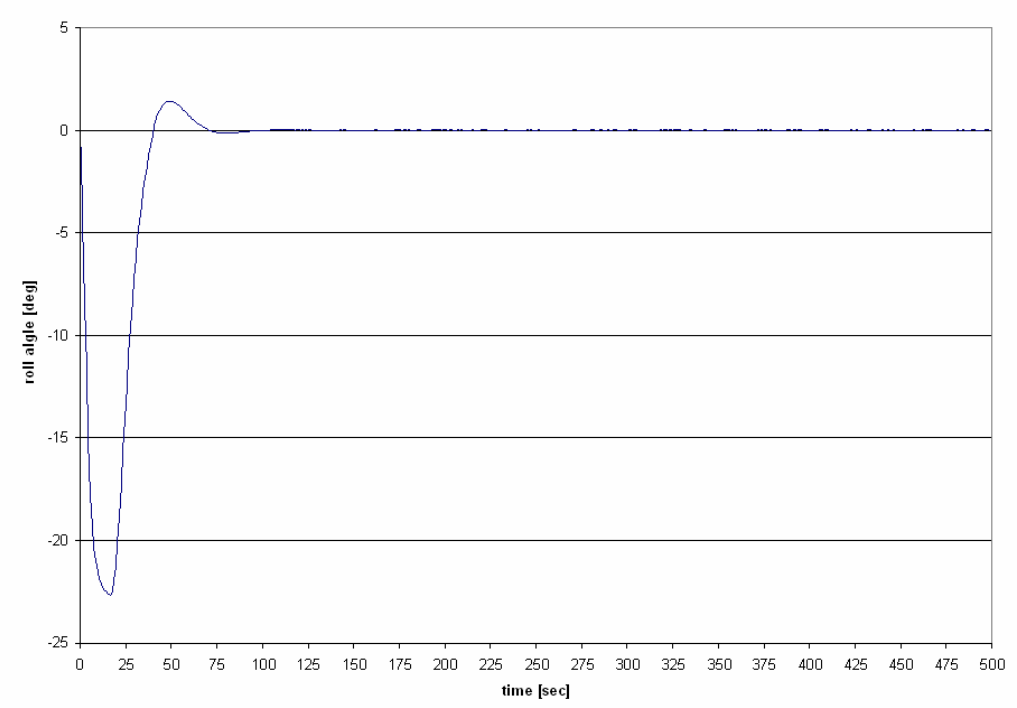


Figure 4.26: test #1 – roll angle response

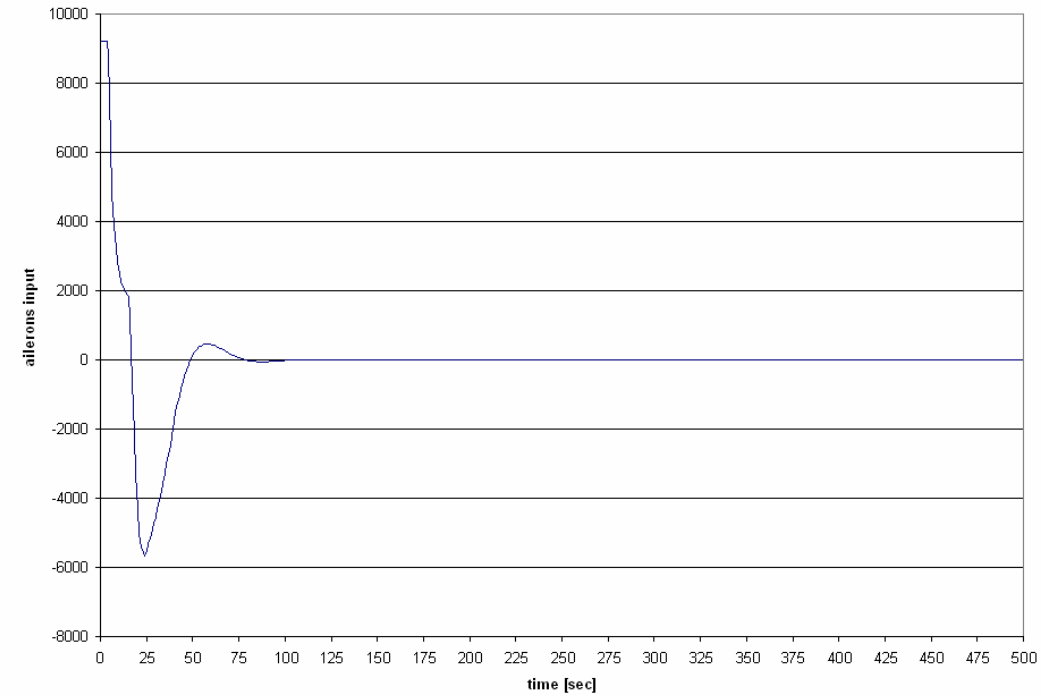


Figure 27.4: test #1 – aileron actuation

Test #2 graphs are showed in the following figures:

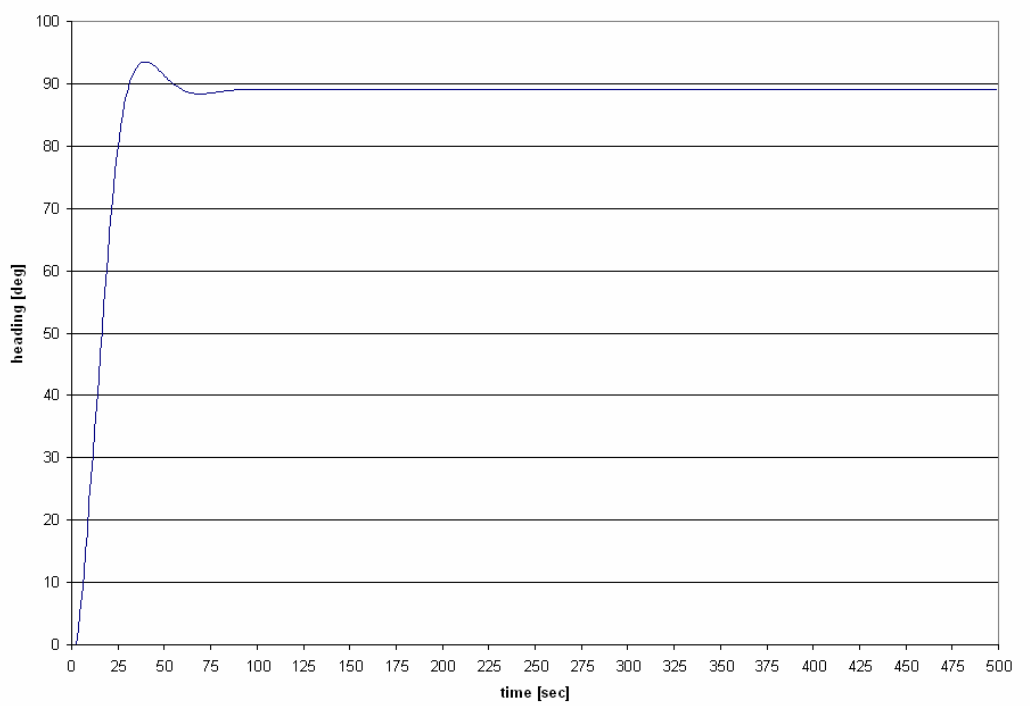


Figure 4.28: test #2 – heading response

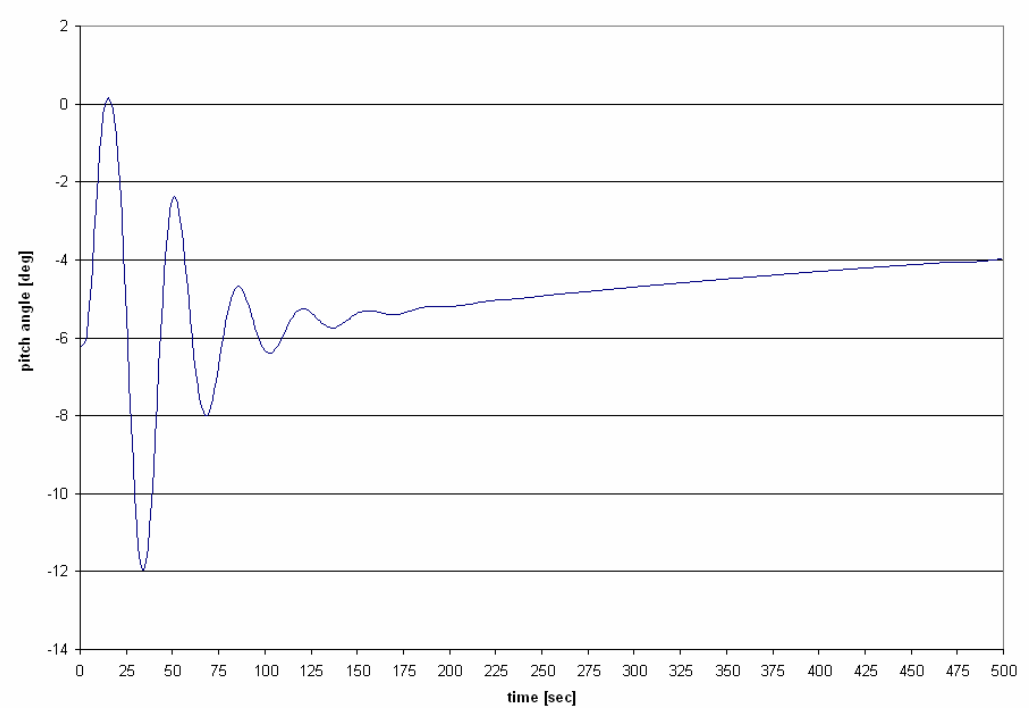


Figure 4. 29: test #2 – pitch angle response

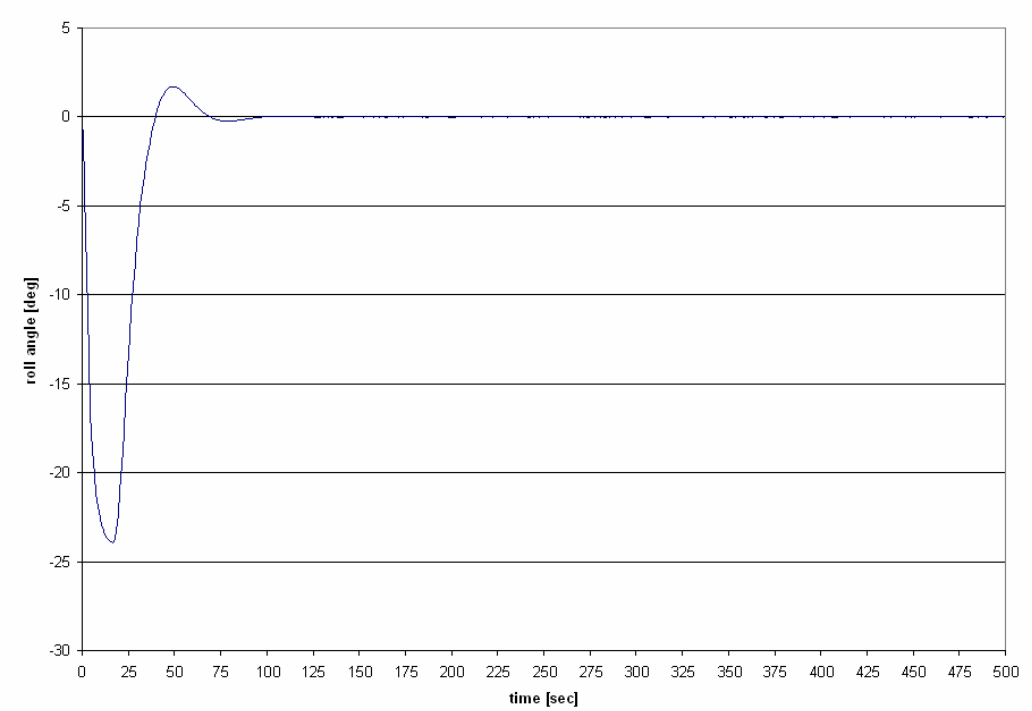


Figure 4.30: test #2 – roll angle response

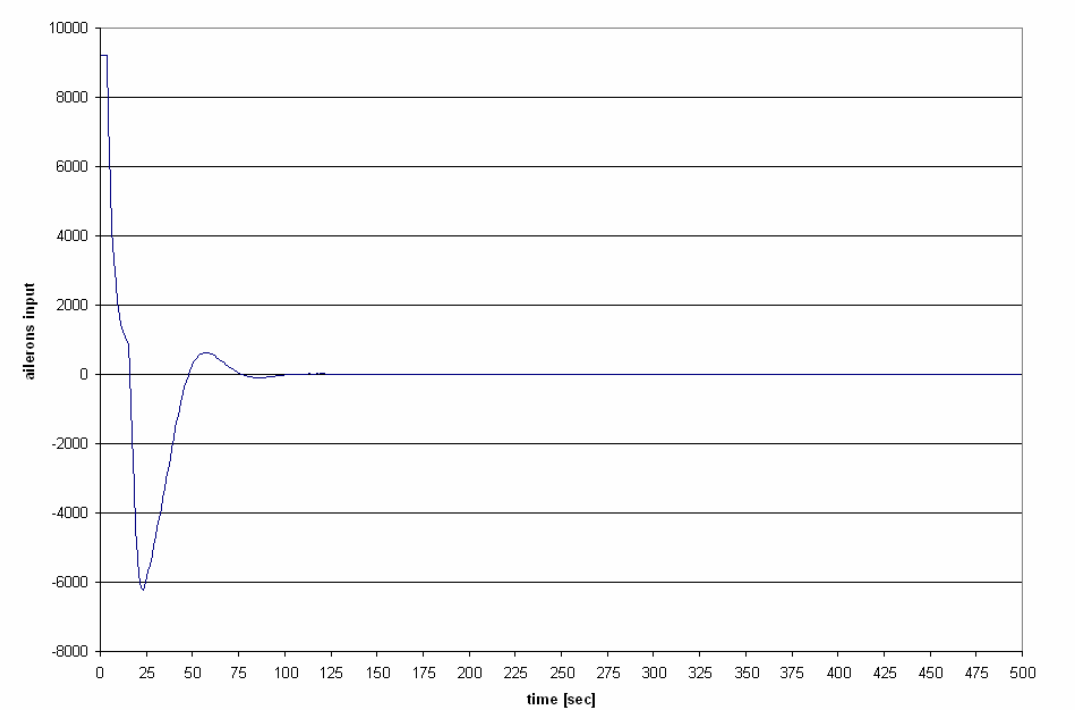


Figure 4.31: test #2 – aileron actuation

The following graphs are relatives to Test #3:

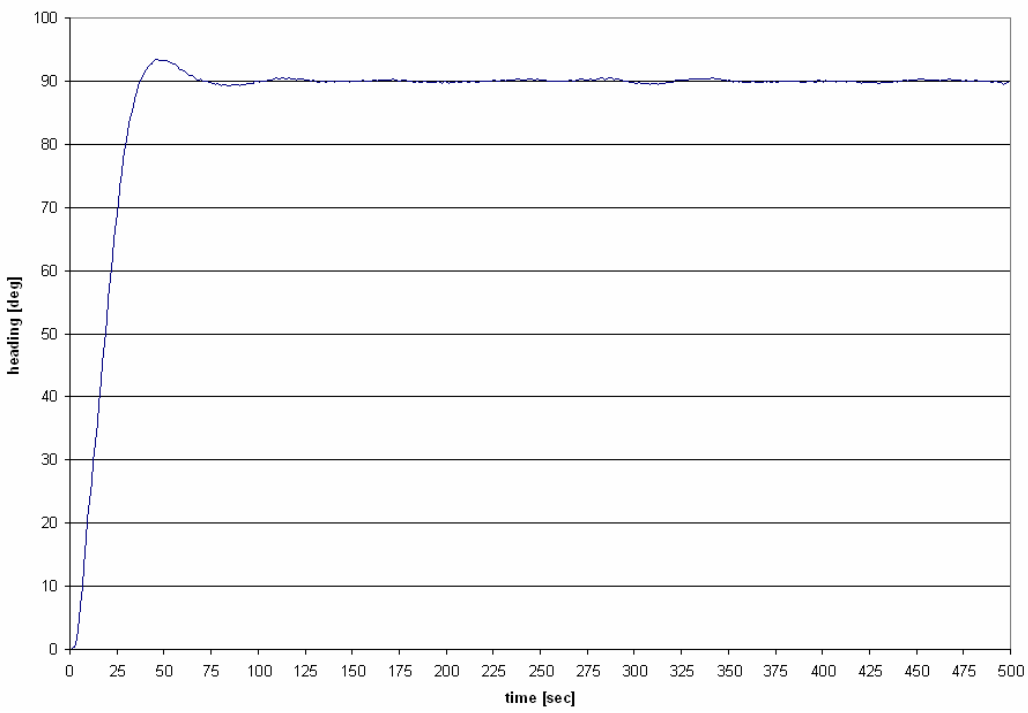


Figure 4.32: test #3 – heading response

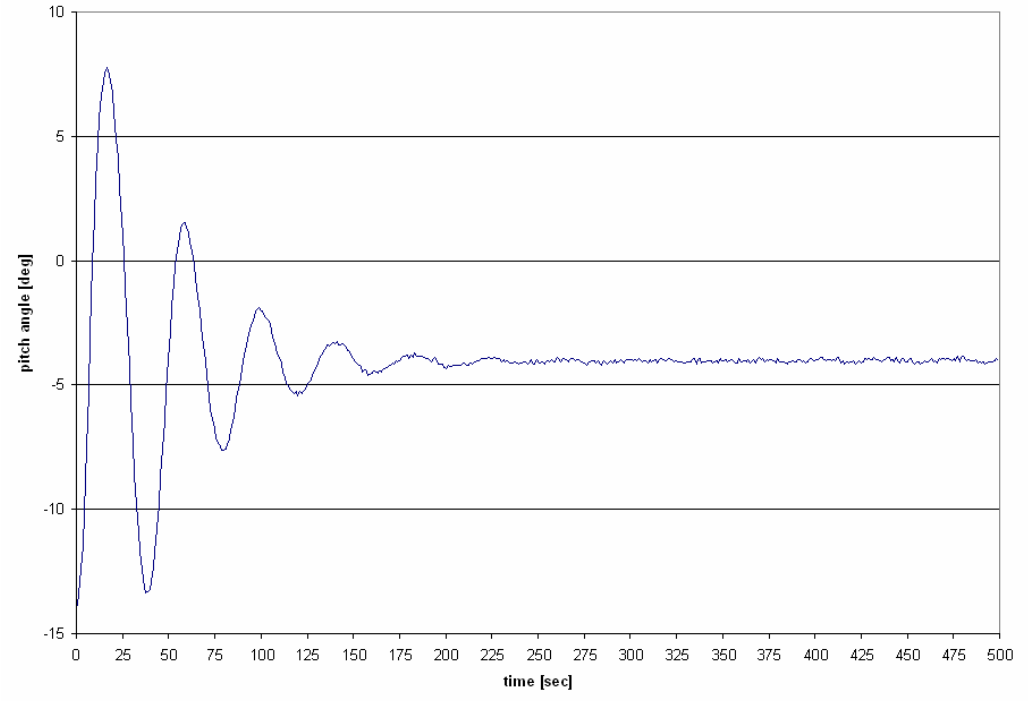


Figure 4.33: test #3 – pitch angle response

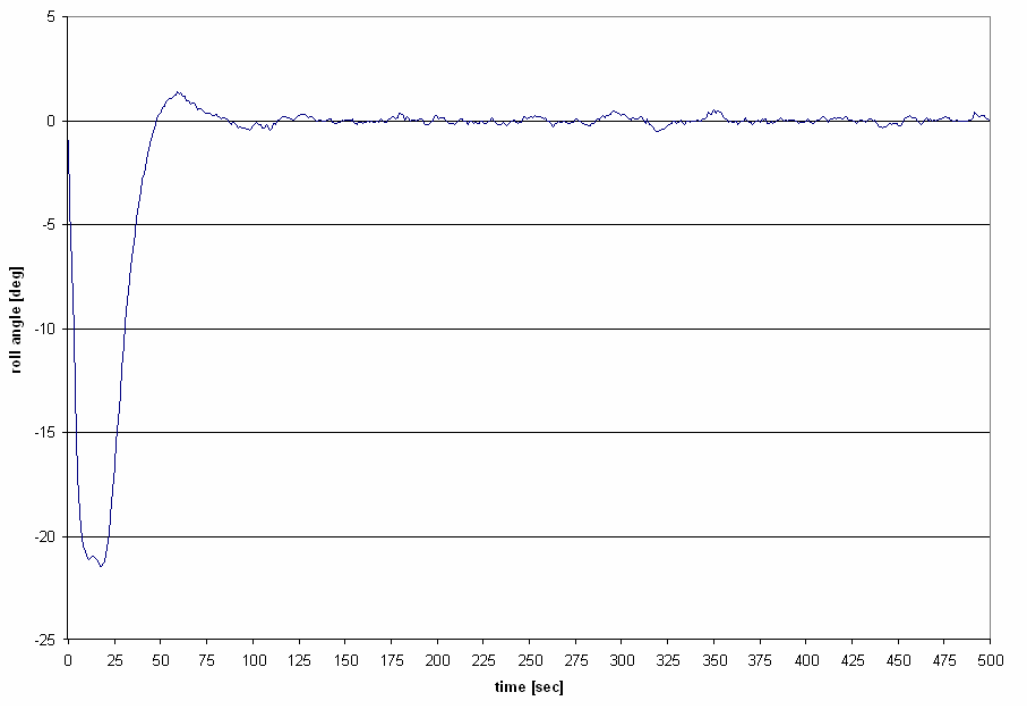


Figure 4.34: test #3 – roll angle response

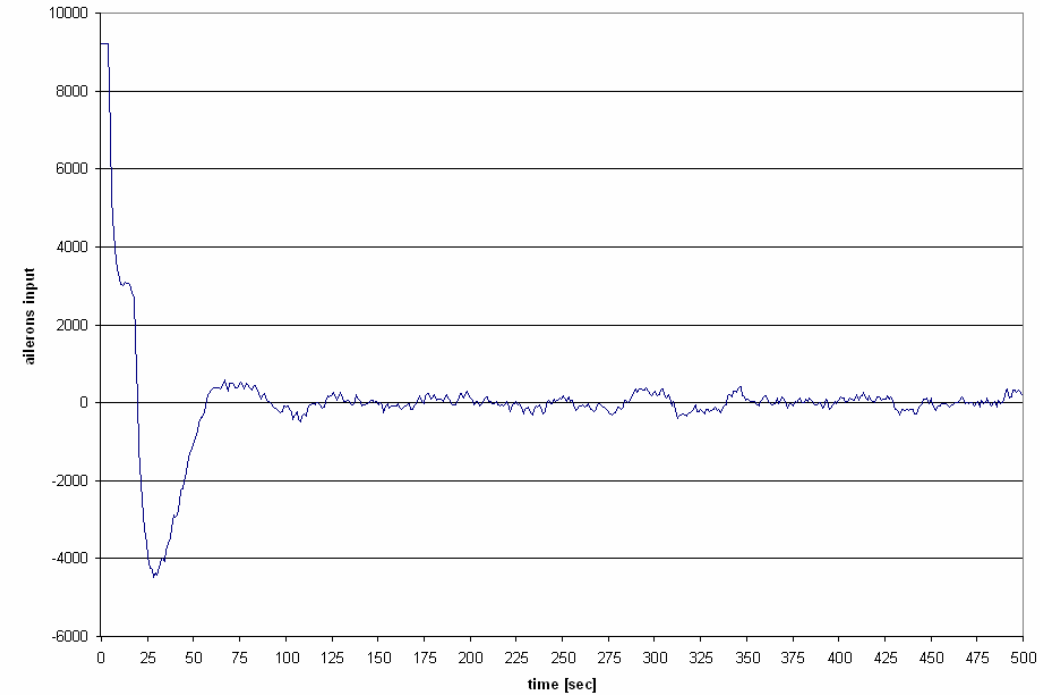


Figure 4.35: test #3 – aileron actuation

The following graphs are relatives to Test #4:

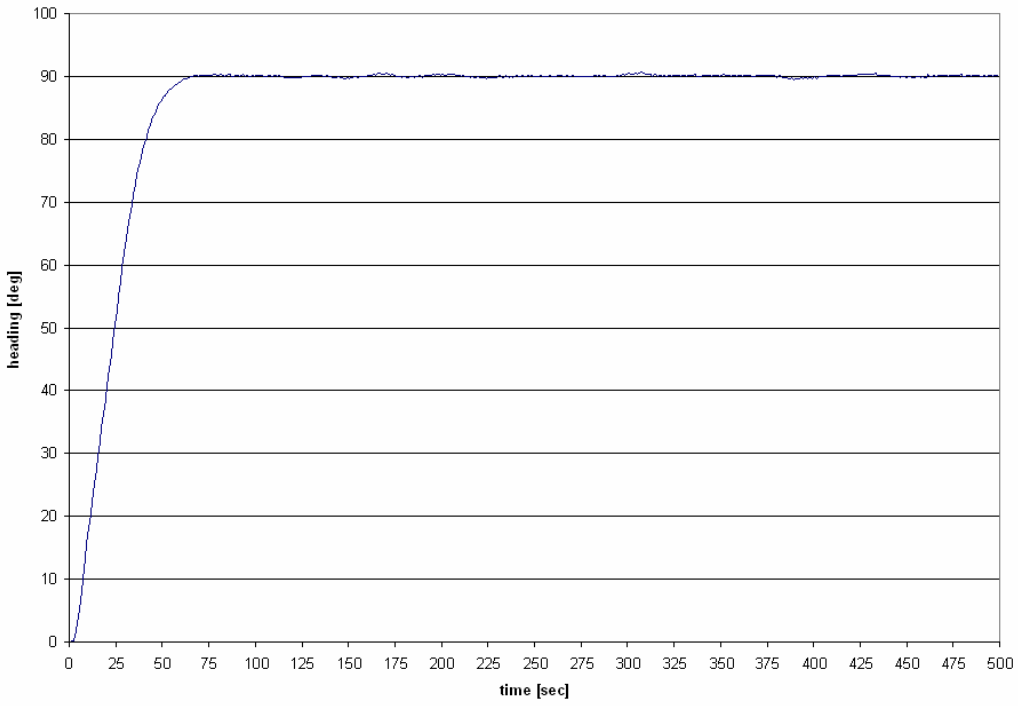


Figure 4.36: test #4 – heading response

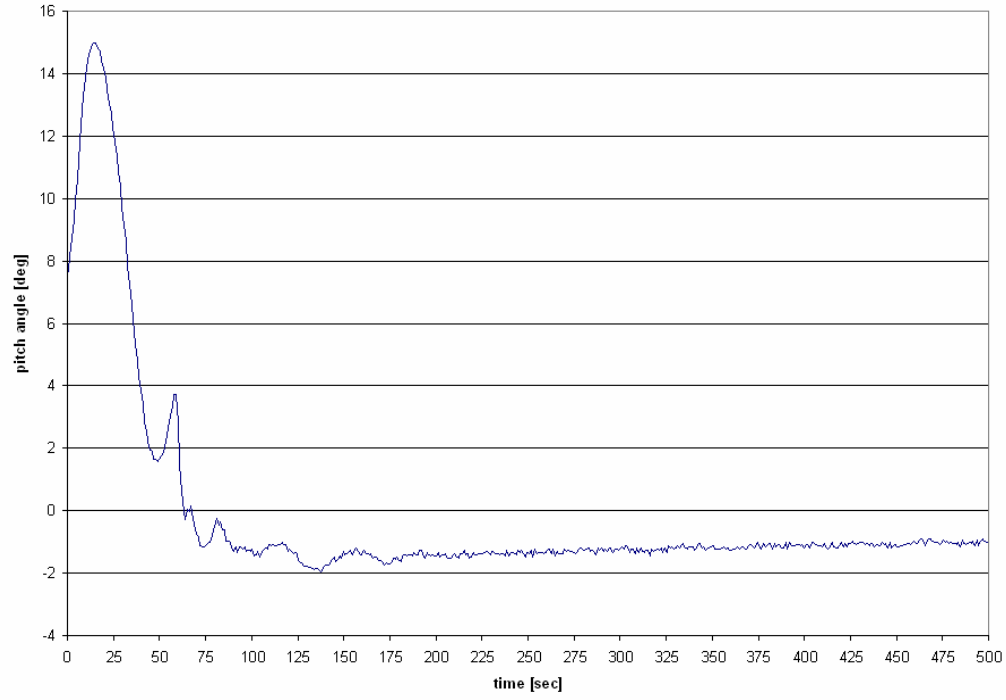


Figure 4.37: test #4 – pitch angle response

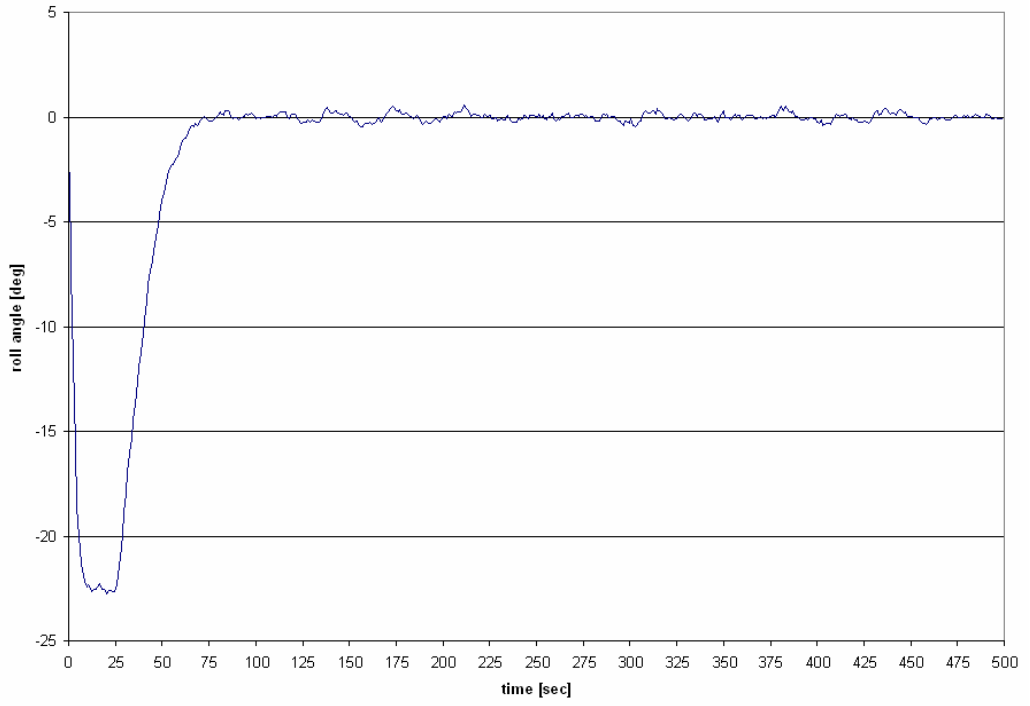


Figure 4.38: test #4 – roll angle response

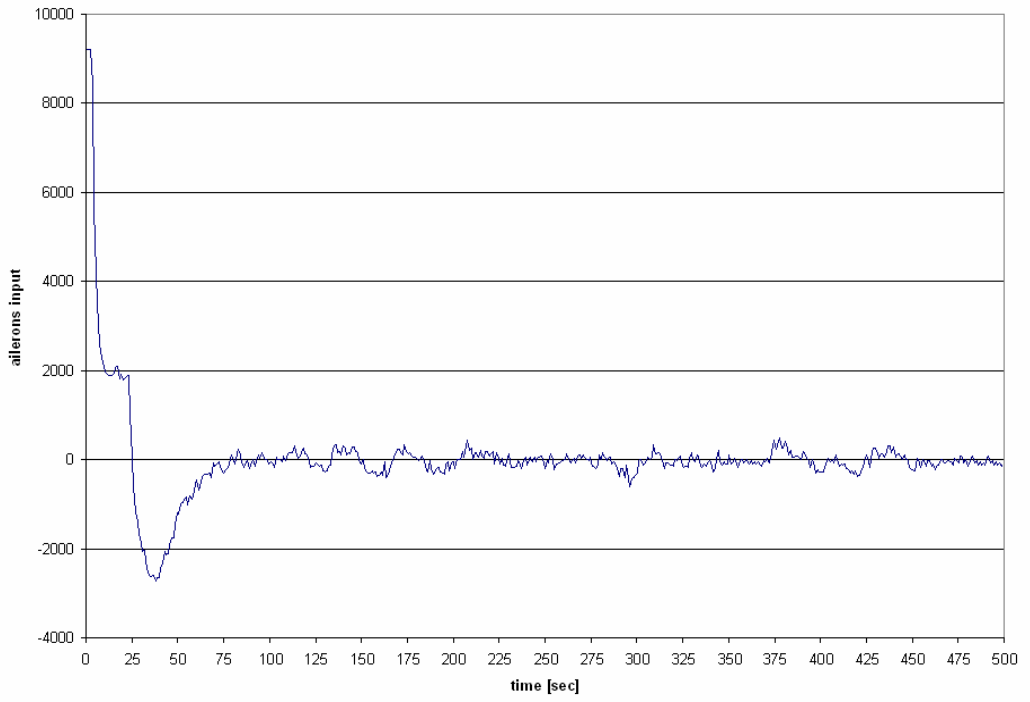


Figure 4.39: test #4 – aileron actuation

These graphs, developed after the tests, shows the performance of this controller in terms of stability and accuracy, we can see that in the cases of max and min load the response of the parameters that we have monitored isn't so closer and then the weight isn't a main parameter of influence on the controller performance. Is important to say that for the aircraft used in the simulations, a Cessna C172SP "*Skyhawk*", that is a ultra-light airplane, the weight variations are limited because these depends by the number of passenger (max 3) and the fuel quantity (max 318 lb).

In case of wind action, we can see that the ailerons must be used more frequently than the case of wind absence. The wind effects are more considerable in the graph relatives to pitch angle and roll angle while, in the heading graphs we can see that the continuous ailerons actuation permits to keep stable the assigned heading, we can see this comparing respectively Fig. 4.1 with Fig. 4.9 and Fig. 4.5 with Fig. 4.13.

On average, after 68" the controller carries out a turn of 90 degrees and keeps stable the heading value without relevant errors, in case of windless weather conditions, and with small heading variations induced by wind, in the other case, always limited in the range [89°30' – 90°25'].

When the controller works to correct heading, the roll angle has his absolute maximum value that is: 22°40' for Test #1, 23°57' for Test #2, 21°27' for Test #3 and 22°23' for Test #4. Then we can see that the limits imposed in design phase was been respected.

Pitch angle isn't important for this test, but we have preferred to plot this parameter to analyse the longitudinal variation caused by aircraft turns that corresponds to a loss of lift on the wings and then the aircraft pitch down when the absolute roll angle increases and pitch up when this value decreases. The aircraft, on average after 3'25", finds a pitch stability by himself.

Ailerons input used in these tests to correct the heading, keeps limited to the values imposed in design and is possible verify it in the relatives figures.

4.4 *longitudinal* controller testing

In table 4.2 are showed the setting used to carry out the test procedures, for this controller we have provide twelve different tests because this controller is able to produce a climb or a descent by an assigned value, and maintain the current altitude. The first quartet of tests is relative to a climb of 500 feet, the second quartet is relative to a descent of 500 feet and the last quartet of tests is relative to a situation of held altitude. Each test was been effectuated combining maximum and minimum load configurations [1920 lb – 2550 lb], and weather conditions variable from windless situation to wind and turbulences situation with a magnitude of 16 knots, the parameters plotted are: *altitude*, *pitch angle*, *vertical speed* and *elevator trim input*.

	weight	***weather***	***input [feet]***	***GPS***
test #1	**MAX**	**wind 340/00**	**+500**	**OFF**
test #2	**MIN**	**wind 340/00**	**+500**	**OFF**
test #3	**MAX**	**wind 000/16 + turbulences**	**+500**	**OFF**
test #4	**MIN**	**wind 000/16 + turbulences**	**+500**	**OFF**
test #5	**MAX**	**wind 340/00**	**-500**	**OFF**
test #6	**MIN**	**wind 340/00**	**-500**	**OFF**
test #7	**MAX**	**wind 000/16 + turbulences**	**-500**	**OFF**
test #8	**MIN**	**wind 000/16 + turbulences**	**-500**	**OFF**
test #9	**MAX**	**wind 340/00**	**altitude hold**	**OFF**
test #10	**MIN**	**wind 340/00**	**altitude hold**	**OFF**
test #11	**MAX**	**wind 000/16 + turbulences**	**altitude hold**	**OFF**
test #12	**MIN**	**wind 000/16 + turbulences**	**altitude hold**	**OFF**

Table 4.2: *longitudinal* controller testing table

The following figures are relative to the test #1:

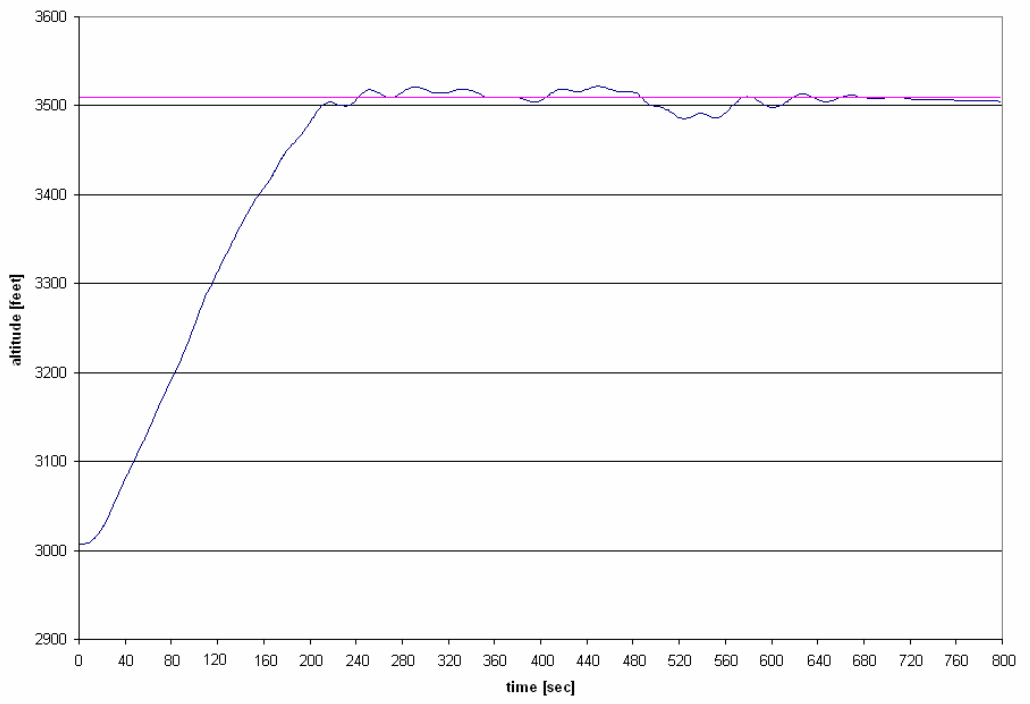


Figure 4.40: test #1 – altitude response

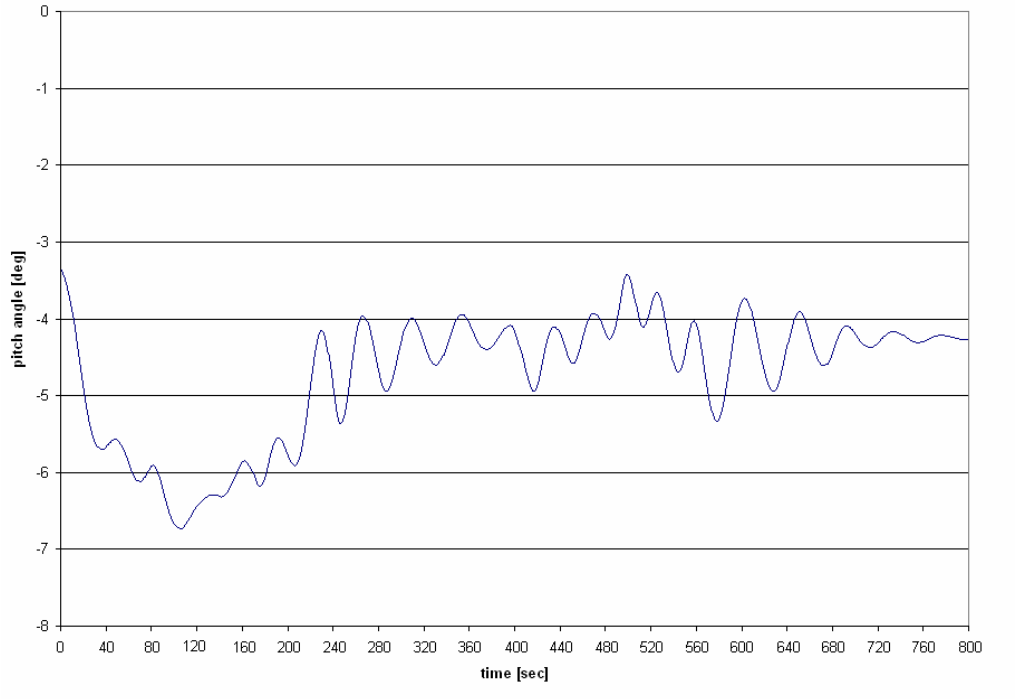


Figure 4.41: test #1 – pitch angle response

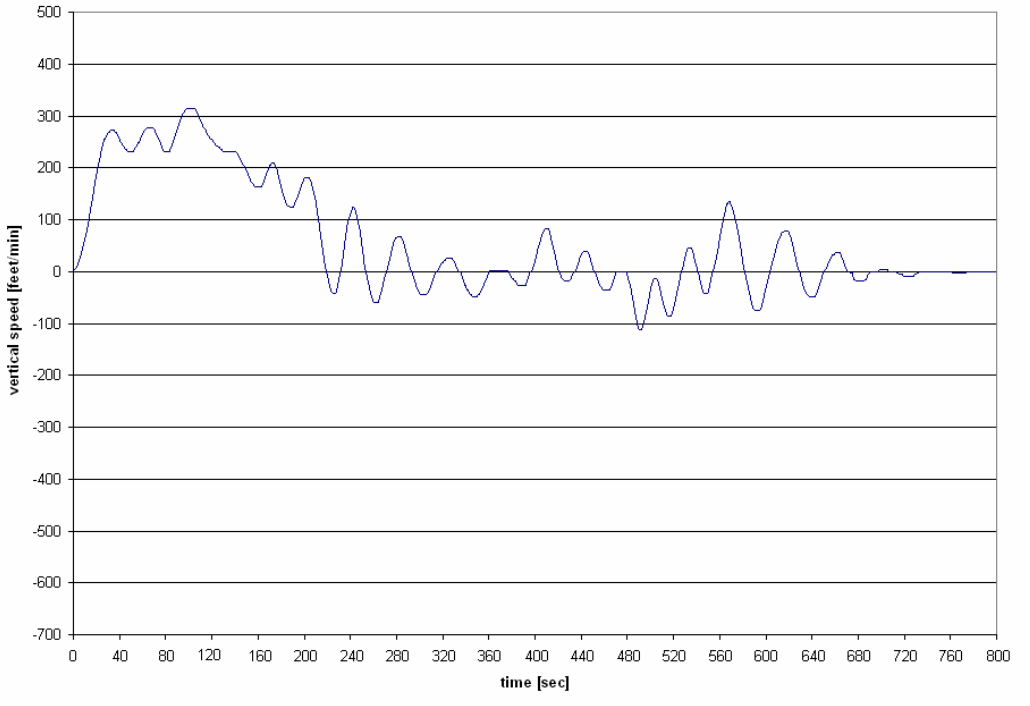


Figure 4.42: test #1 – vertical speed response

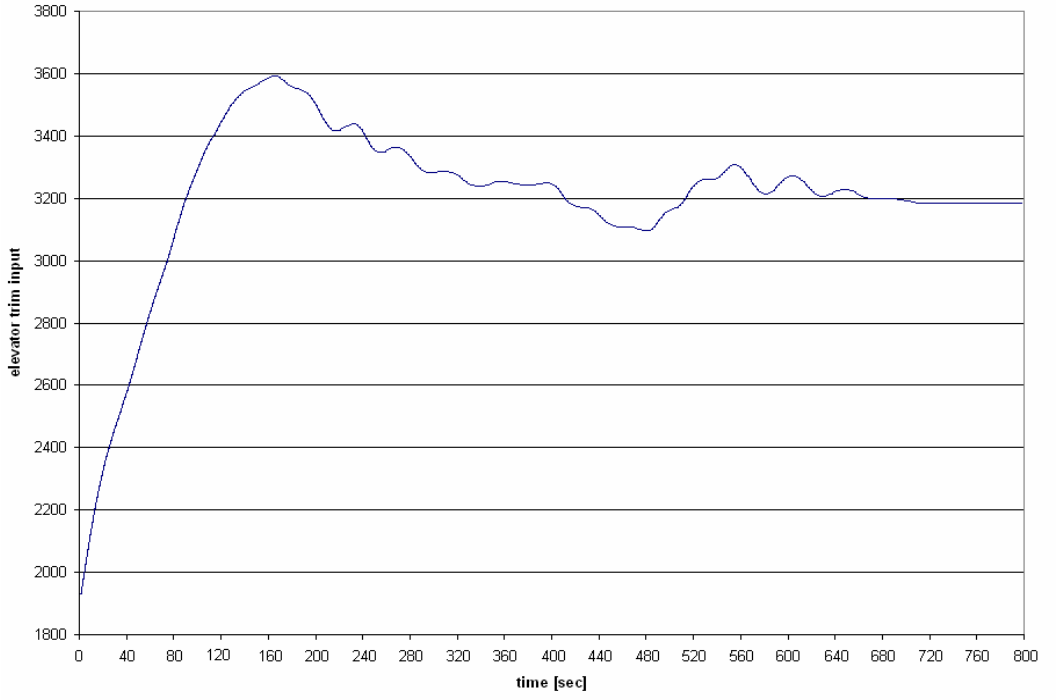


Figure 4.43: test #1 – elevator trim actuation

Graphs relatives to Test #2:

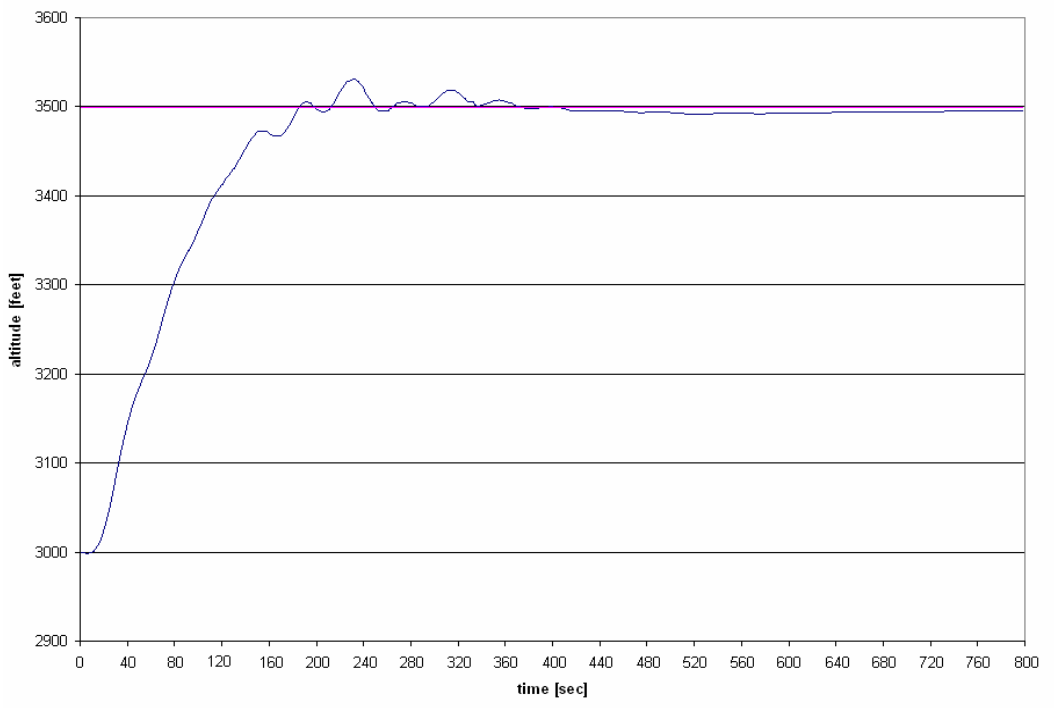


Figure 4.44: test #2 – altitude response

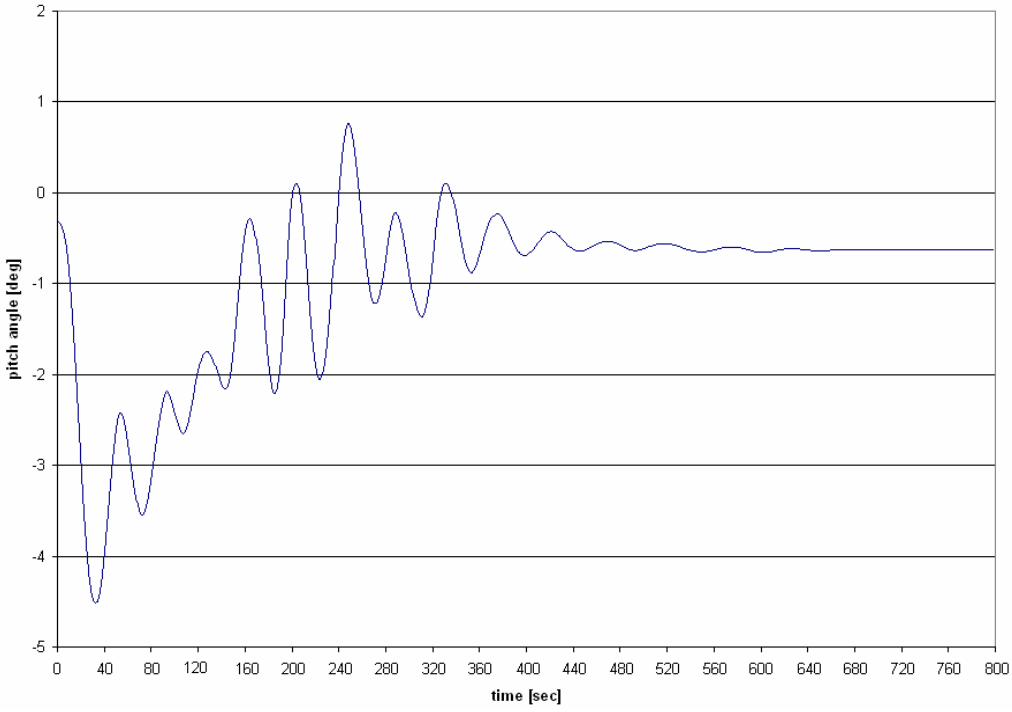


Figure 4.45: test #2 – pitch angle response

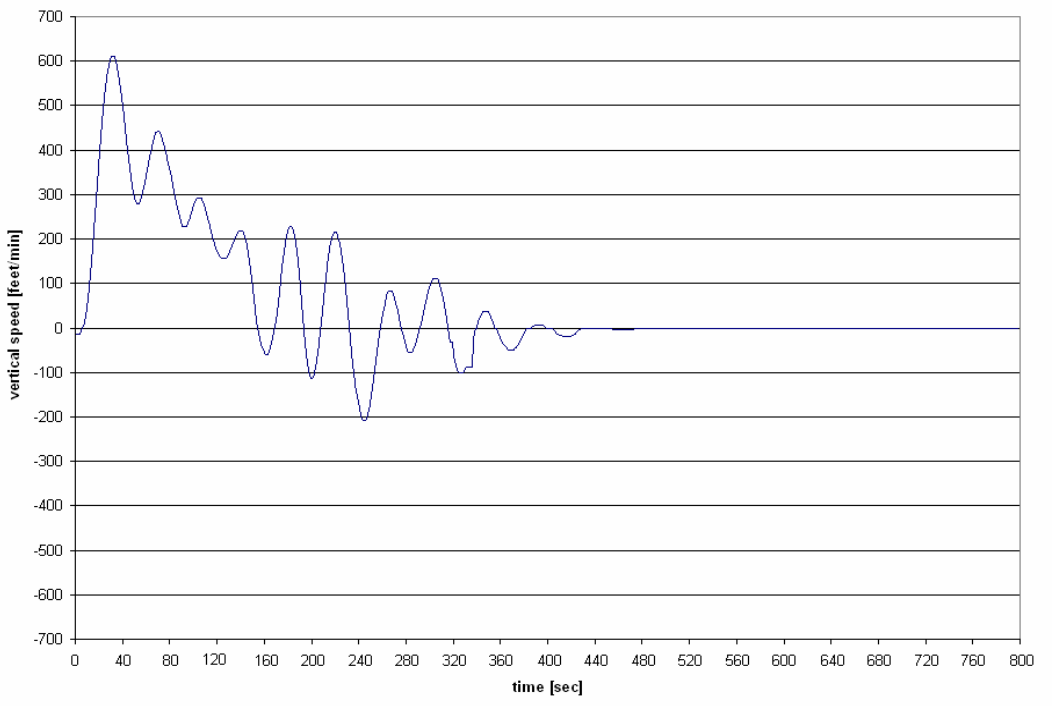


Figure 4.46: test #2 – vertical speed response

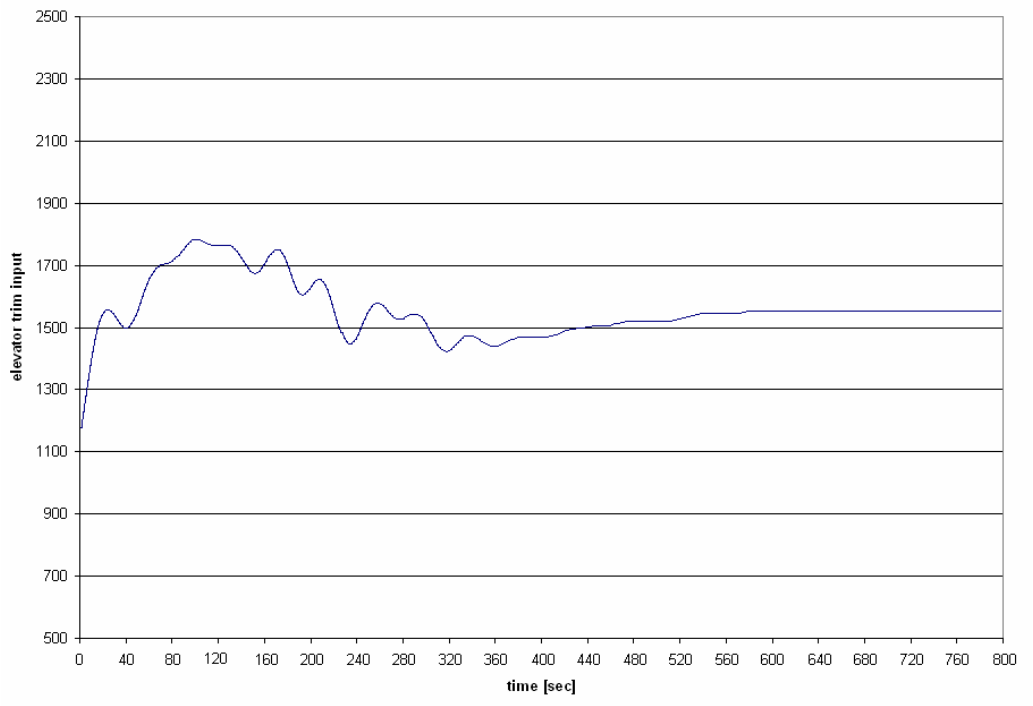


Figure 4.47: test #2 – elevator trim actuation

Graphs relatives to Test #3:

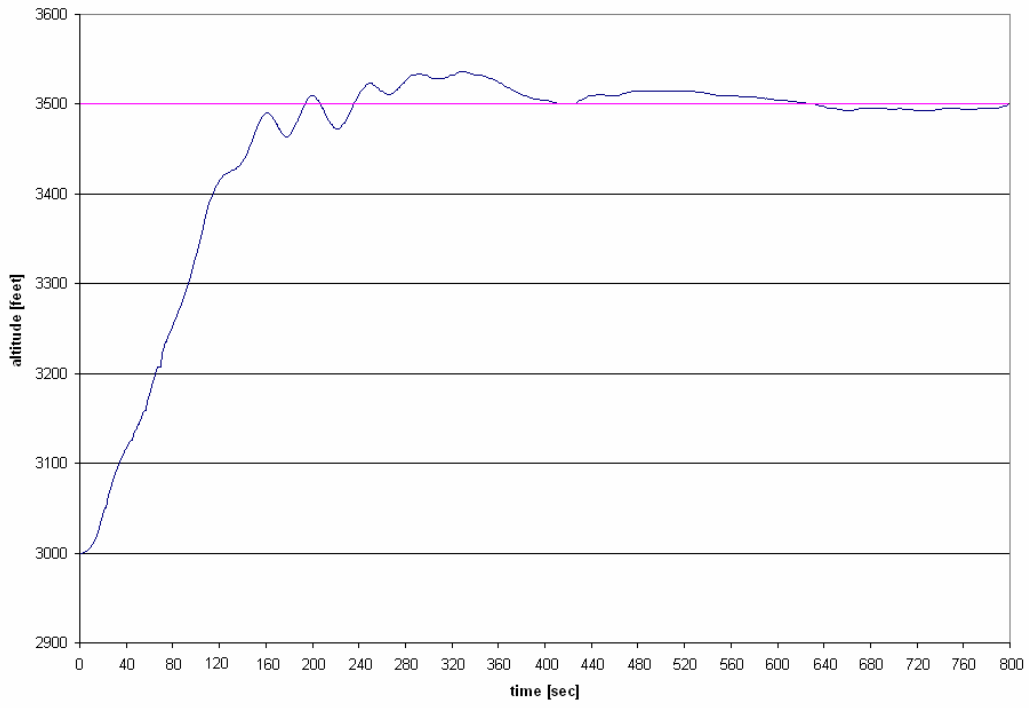


Figure 4.48: test #3 – altitude response

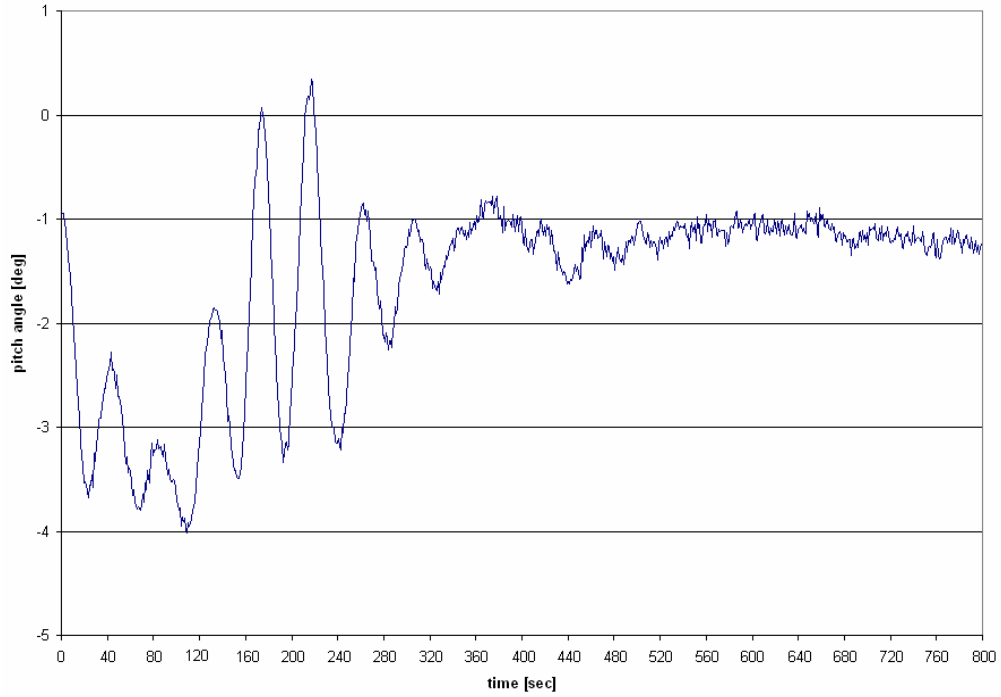


Figure 4.49: test #3 – pitch angle response

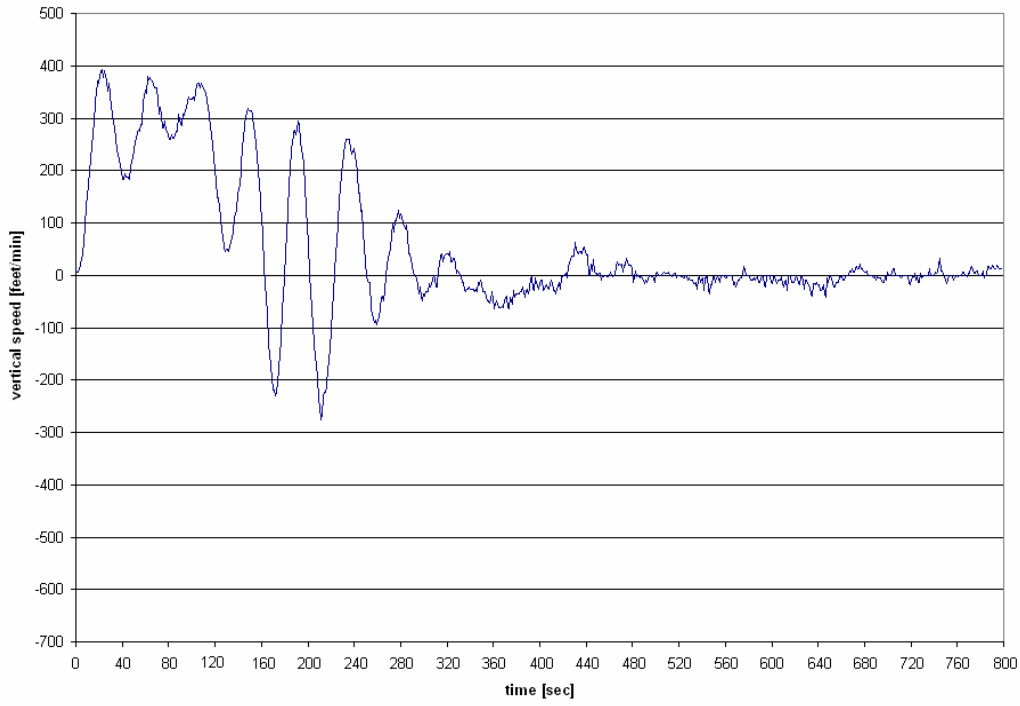


Figure 4.50: test #3 – vertical speed response

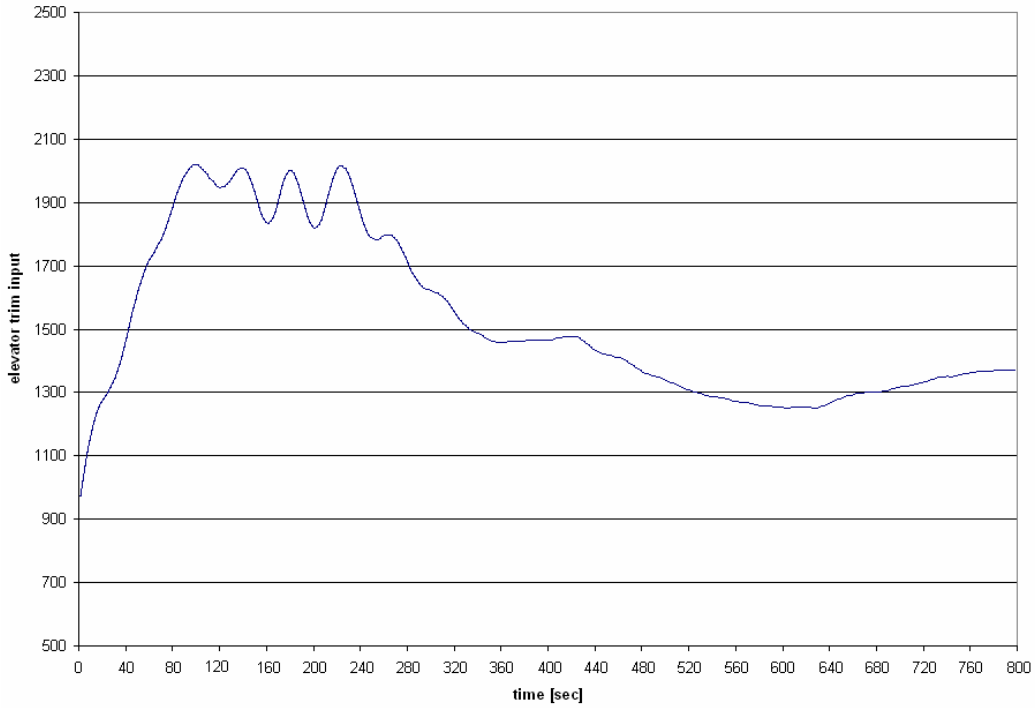


Figure 4.51: test #3 – elevator trim actuation

Graphs relatives to Test #4:

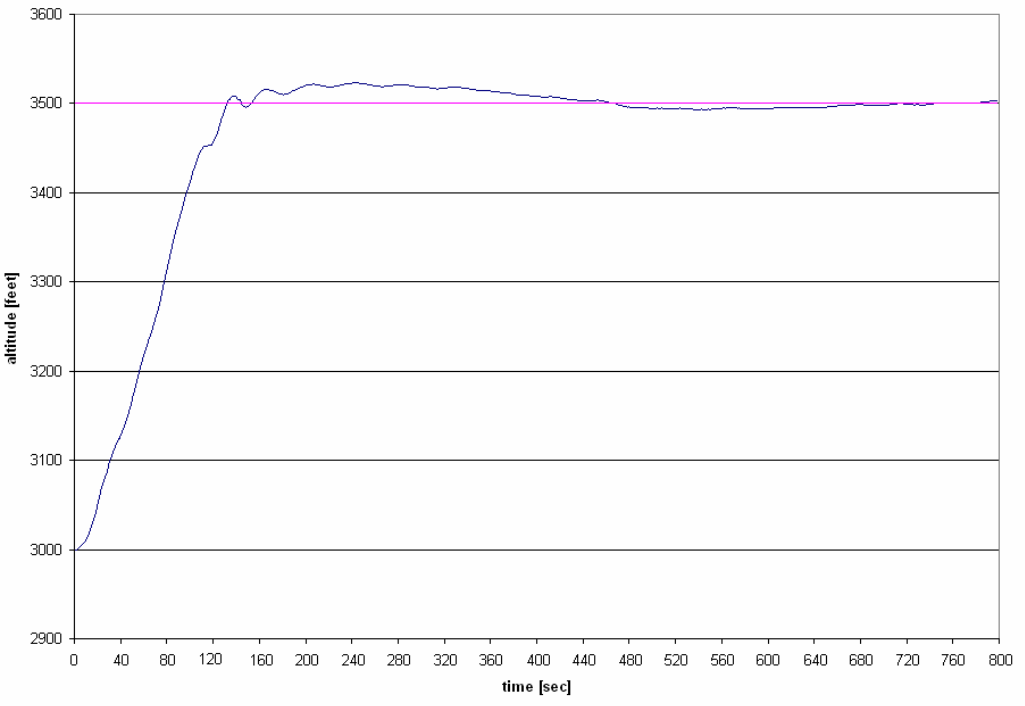


Figure 4.52: test #4 – altitude response

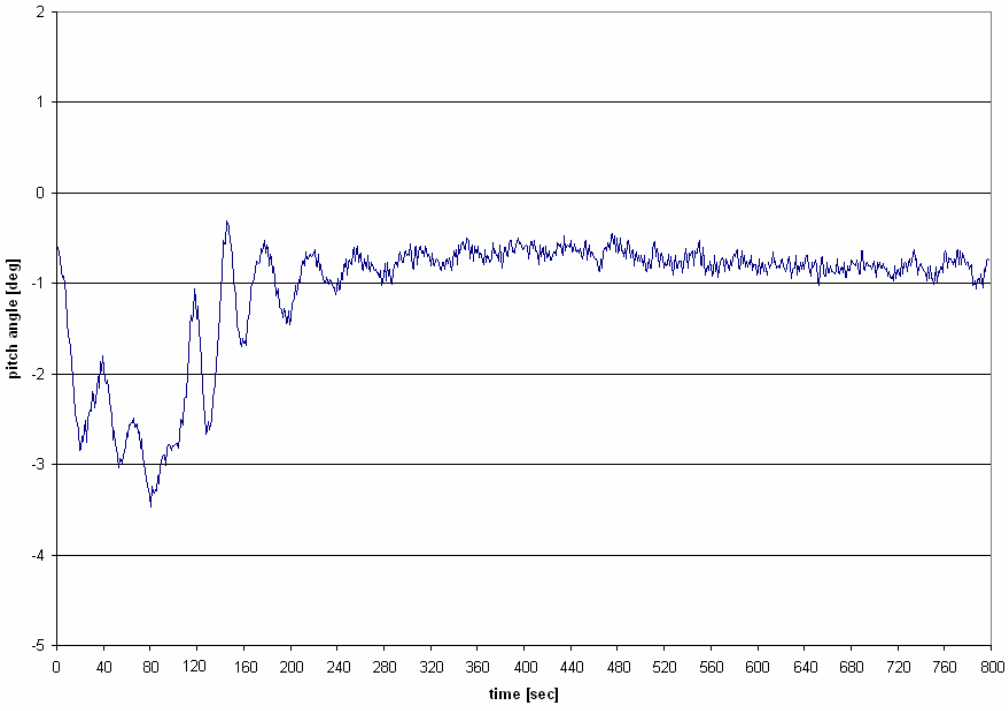


Figure 4.53: test #4 – pitch angle response

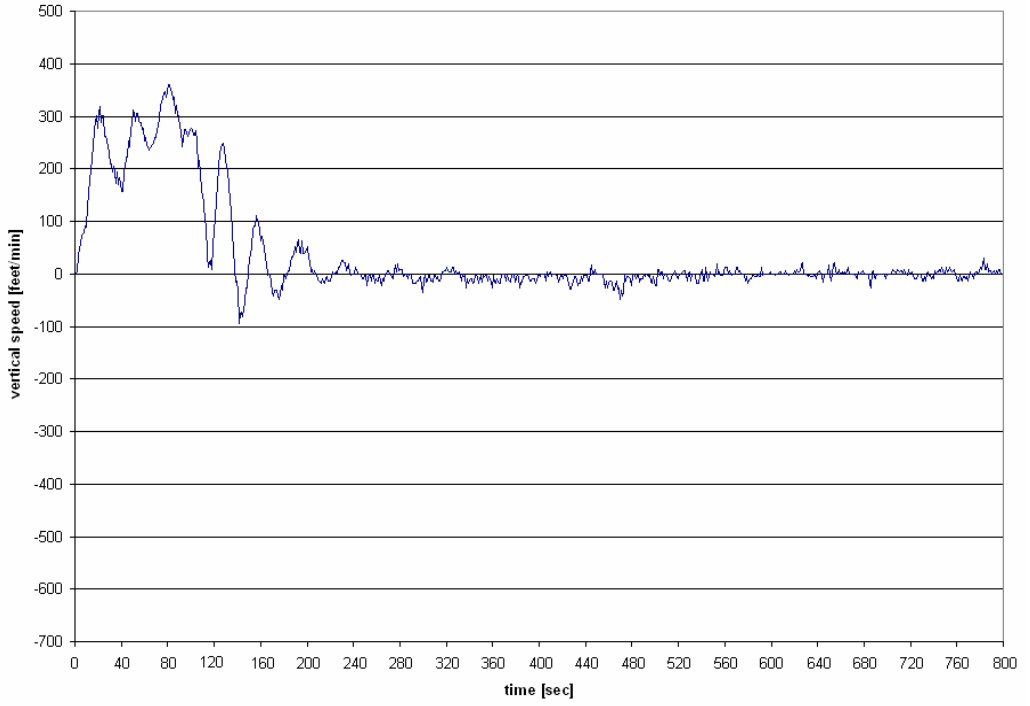


Figure 4.54: test #4 – vertical speed response

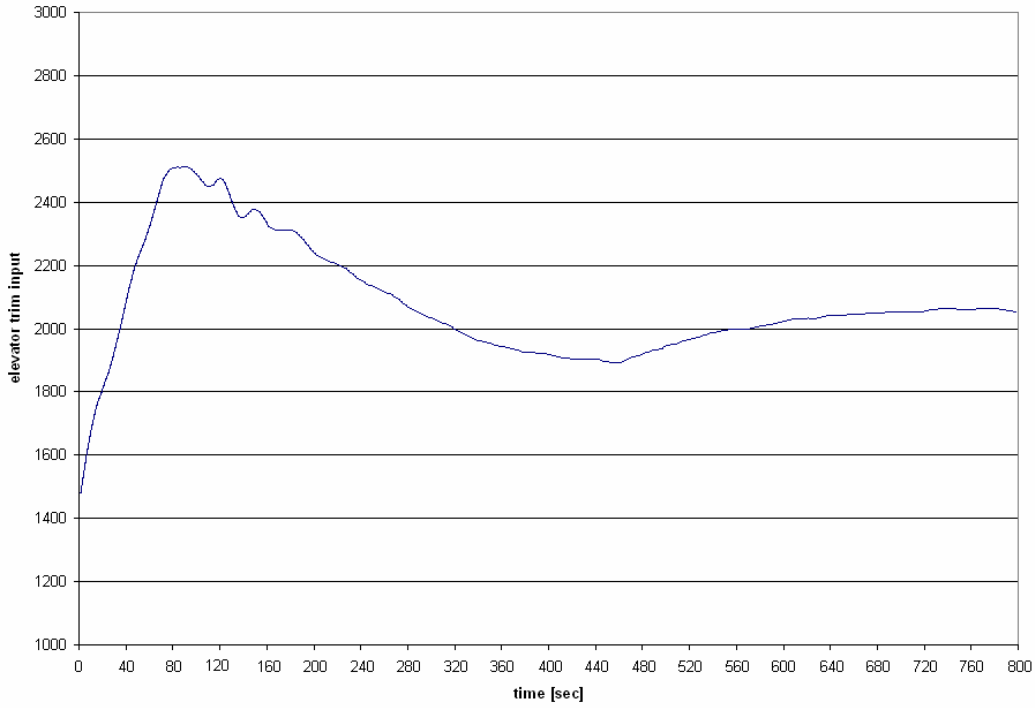


Figure 4.55: test #4 – elevator trim actuation

Graphs relatives to Test #5:

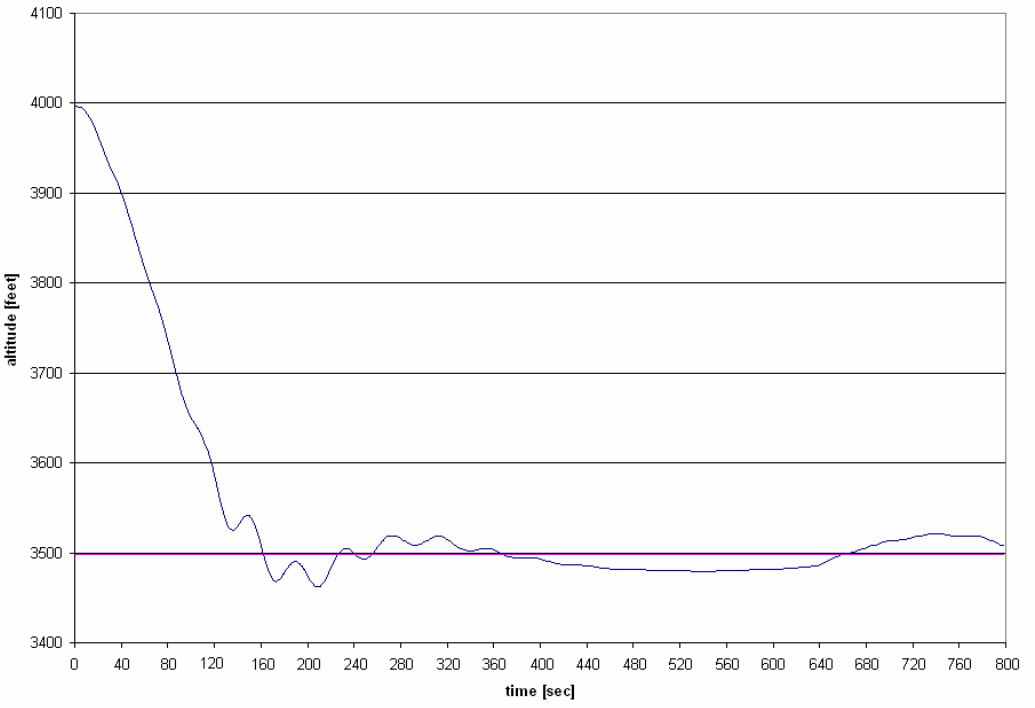


Figure 4.56: test #5 – altitude response

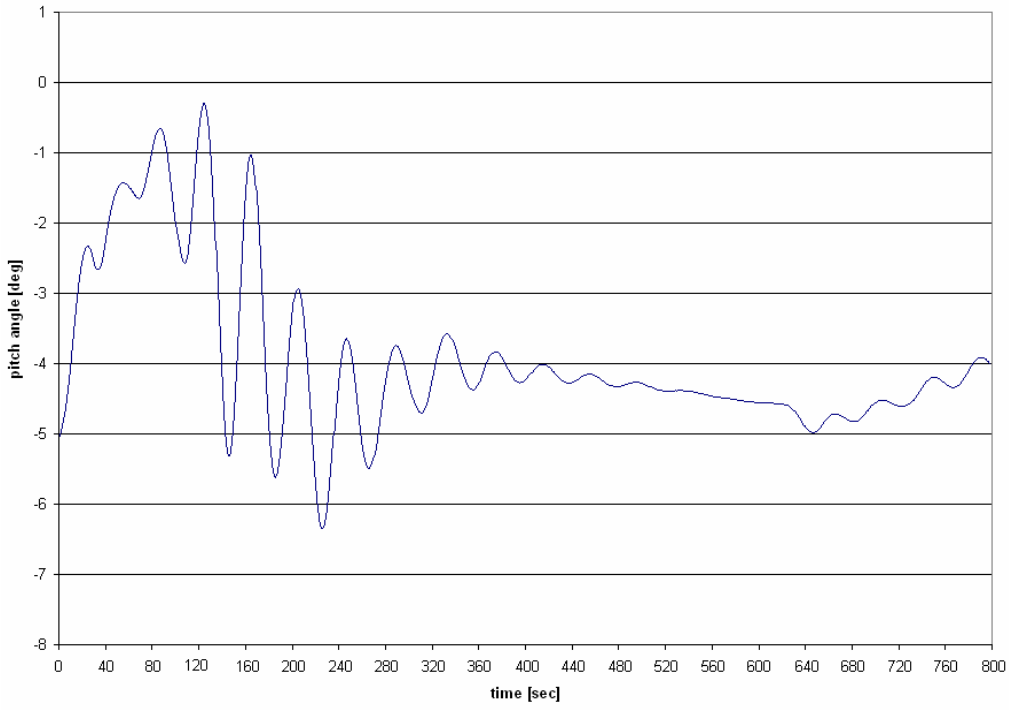


Figure 4.57: test #5 – pitch angle response

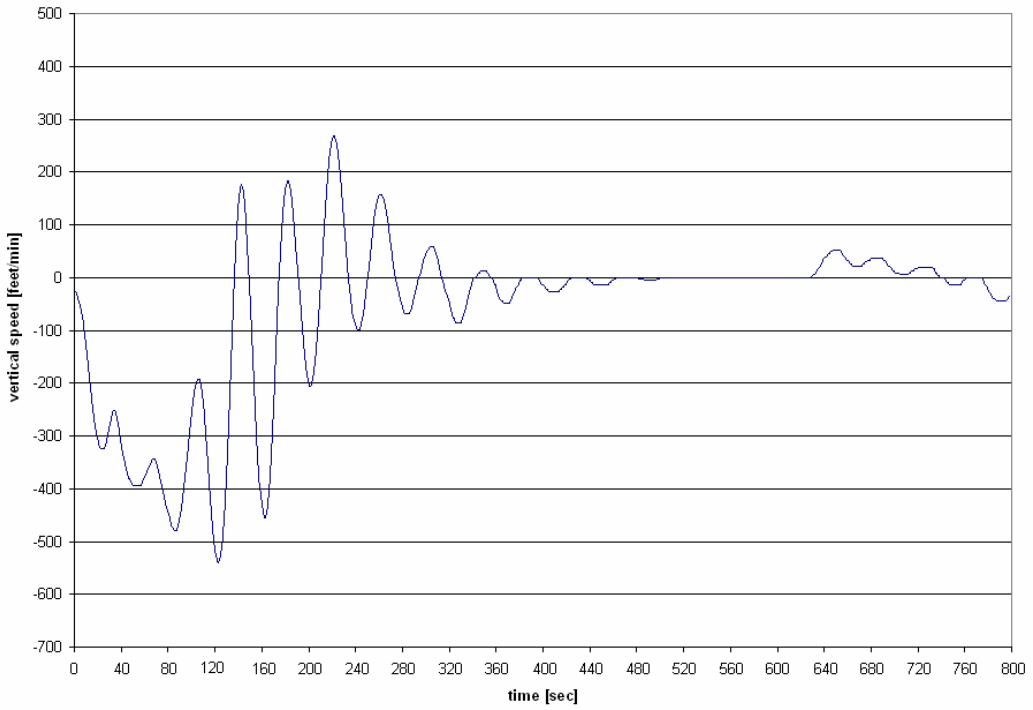


Figure 4.58: test #5 – vertical speed response

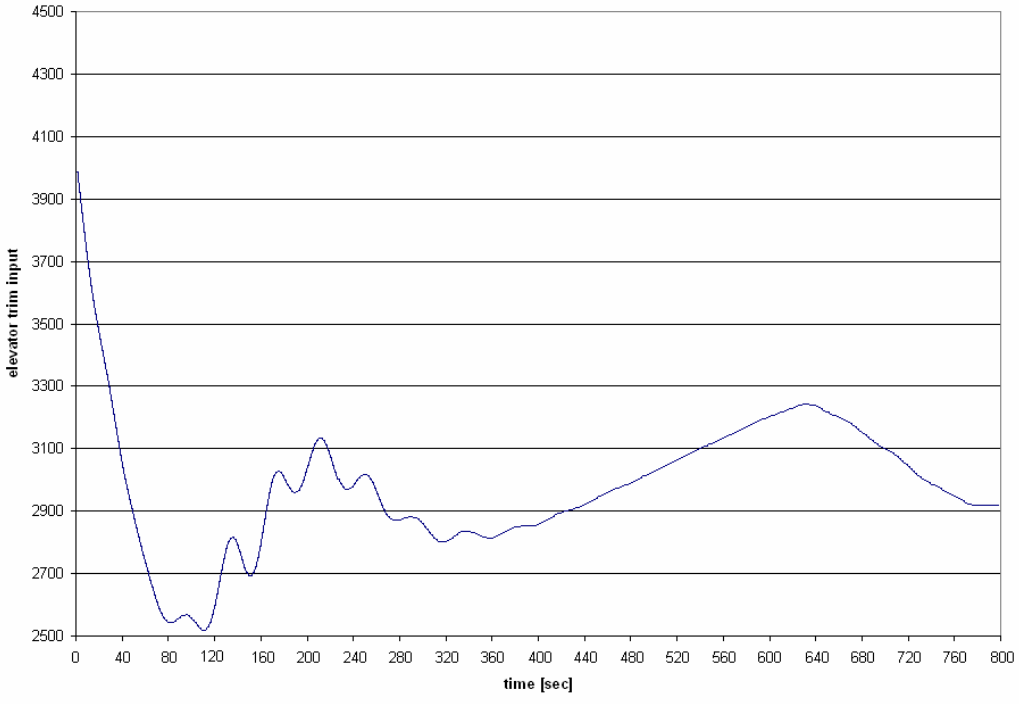


Figure 4.59: test #5 – elevator trim actuation

Graphs relatives to Test #6:

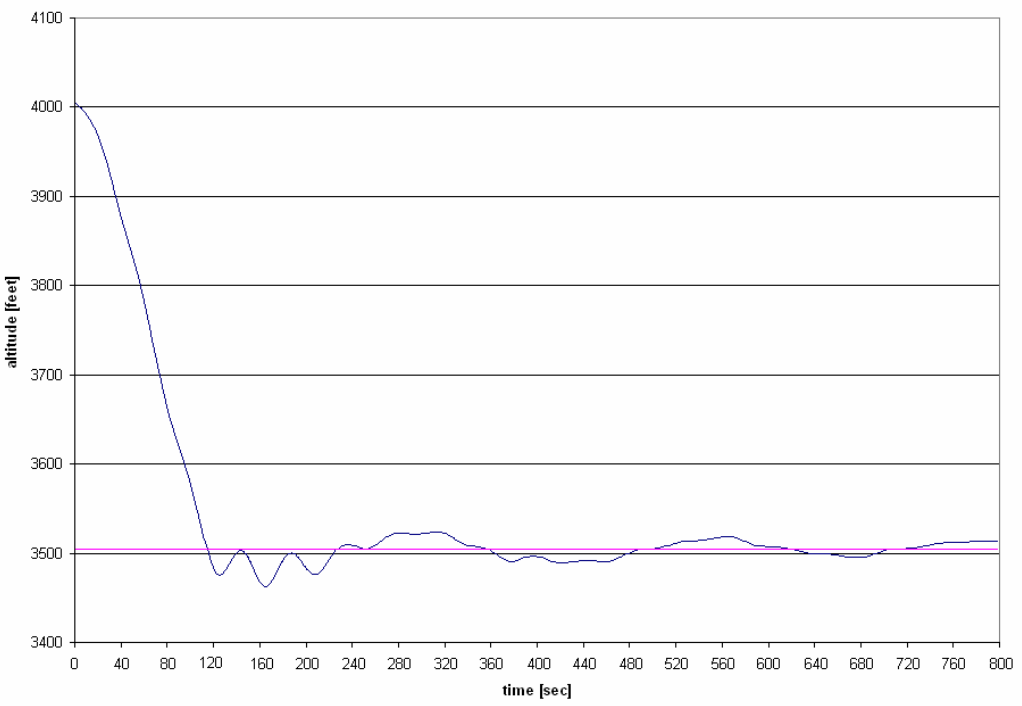


Figure 4.60: test #6 – altitude response

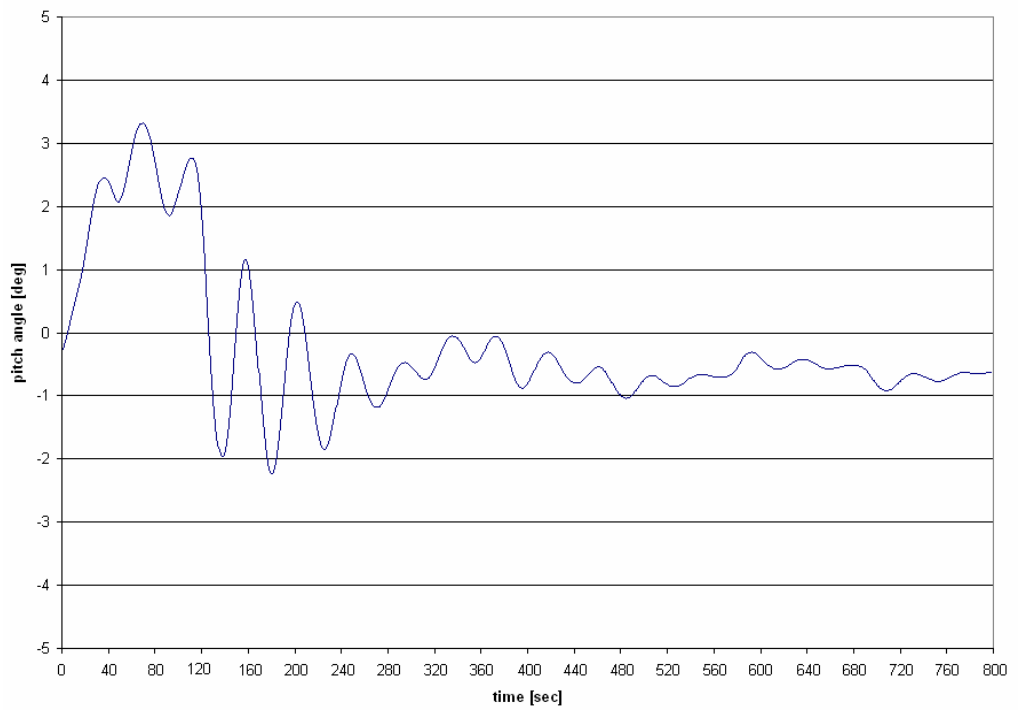


Figure 4.61: test #6 – pitch angle response

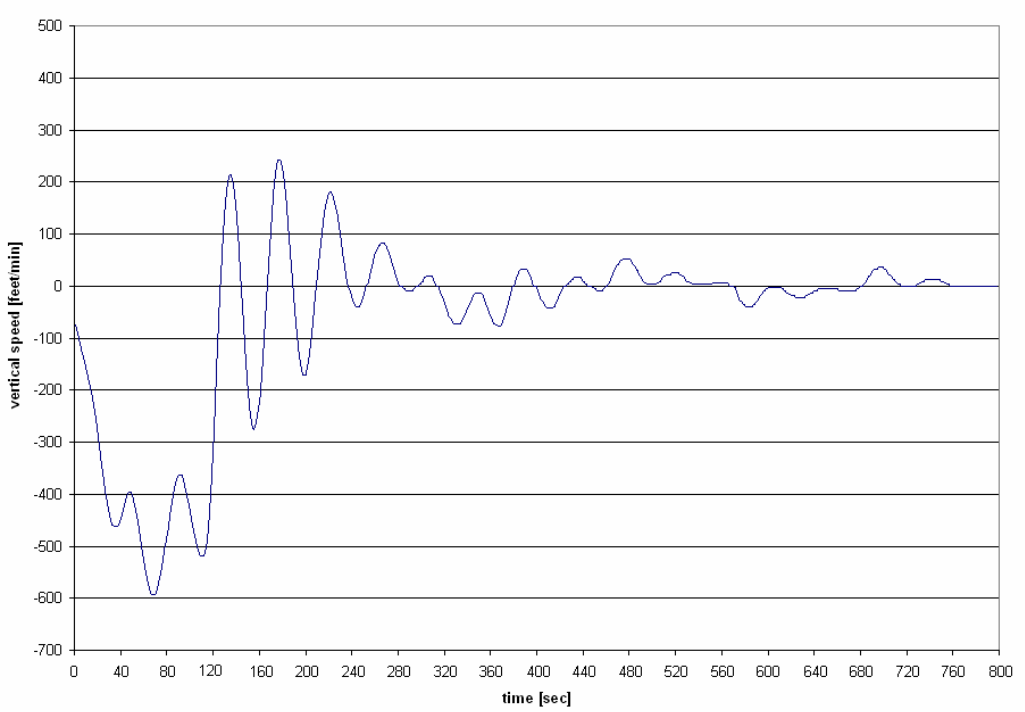


Figure 4.62: test #6 – vertical speed response

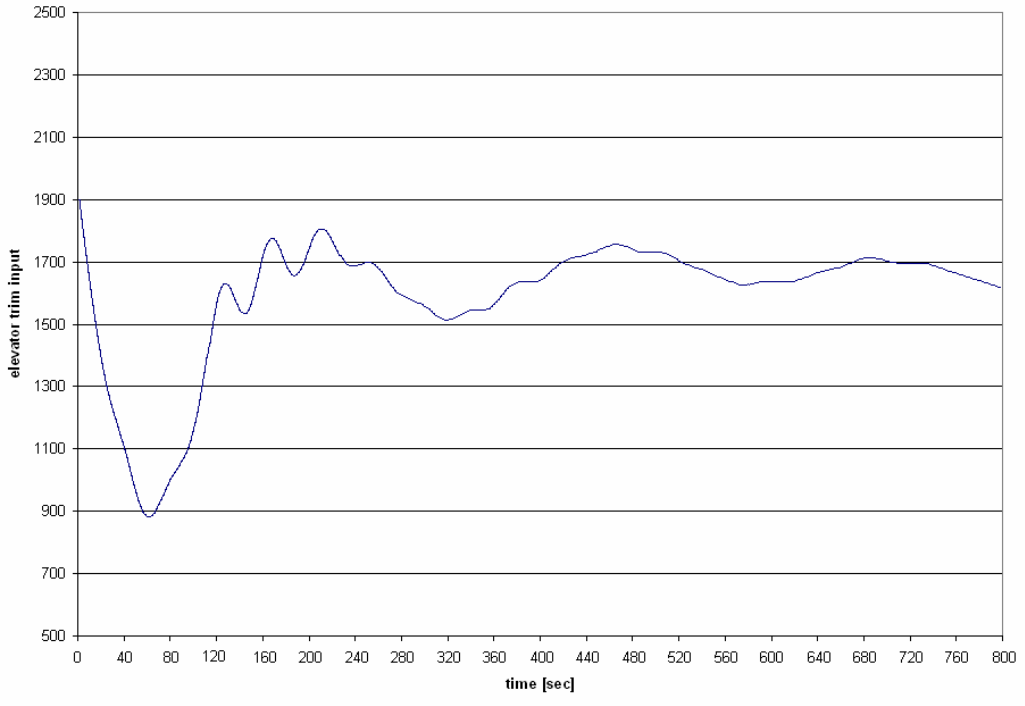


Figure 4.63: test #6 – elevator trim actuation

Graphs relatives to Test #7:

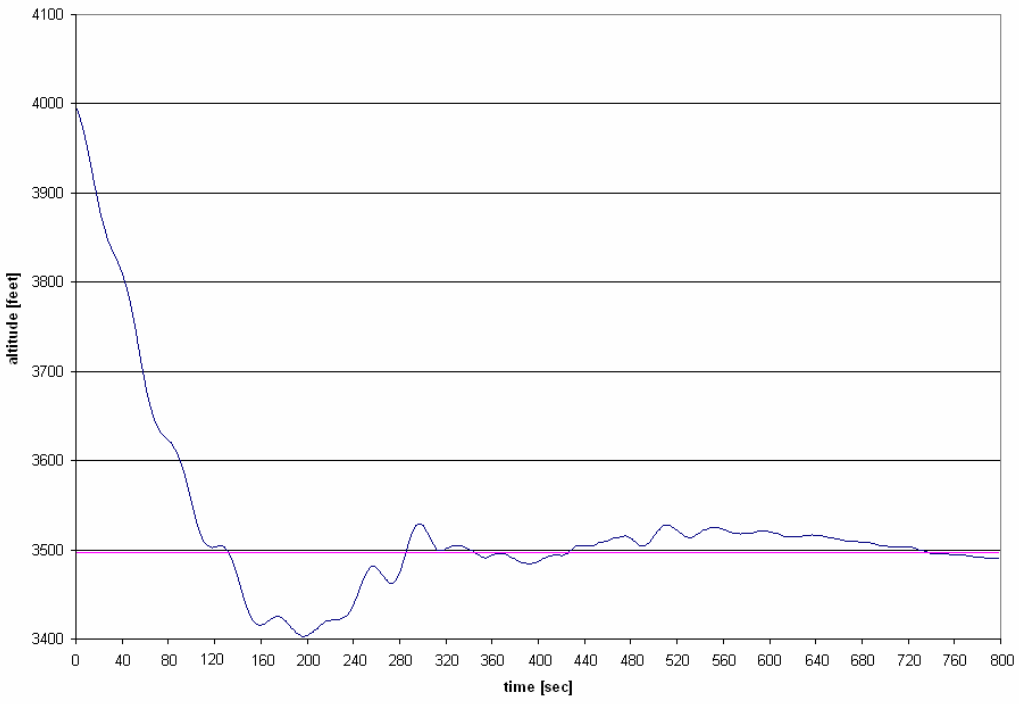


Figure 4.64: test #7 – altitude response

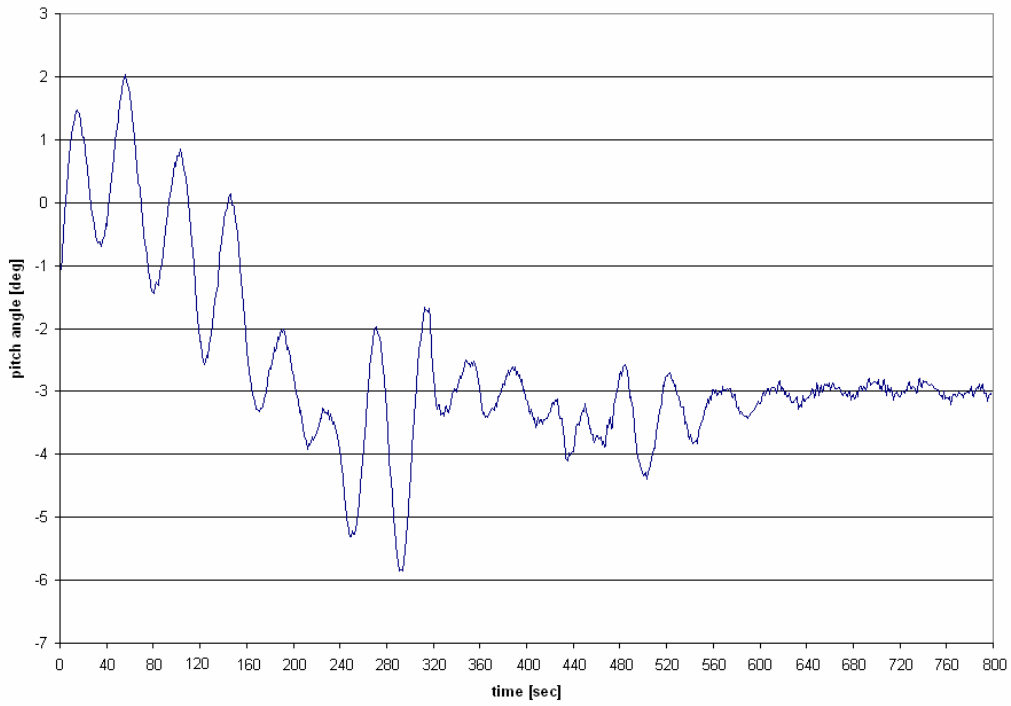


Figure 4.65: test #7 – pitch angle response

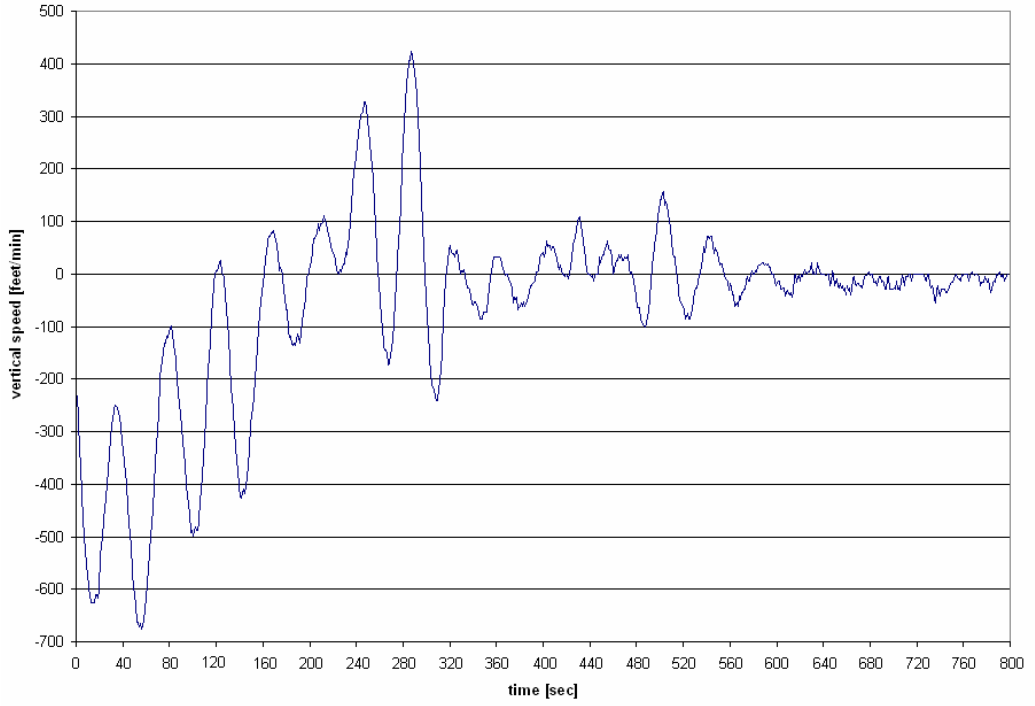


Figure 4.66: test #7 – vertical speed response

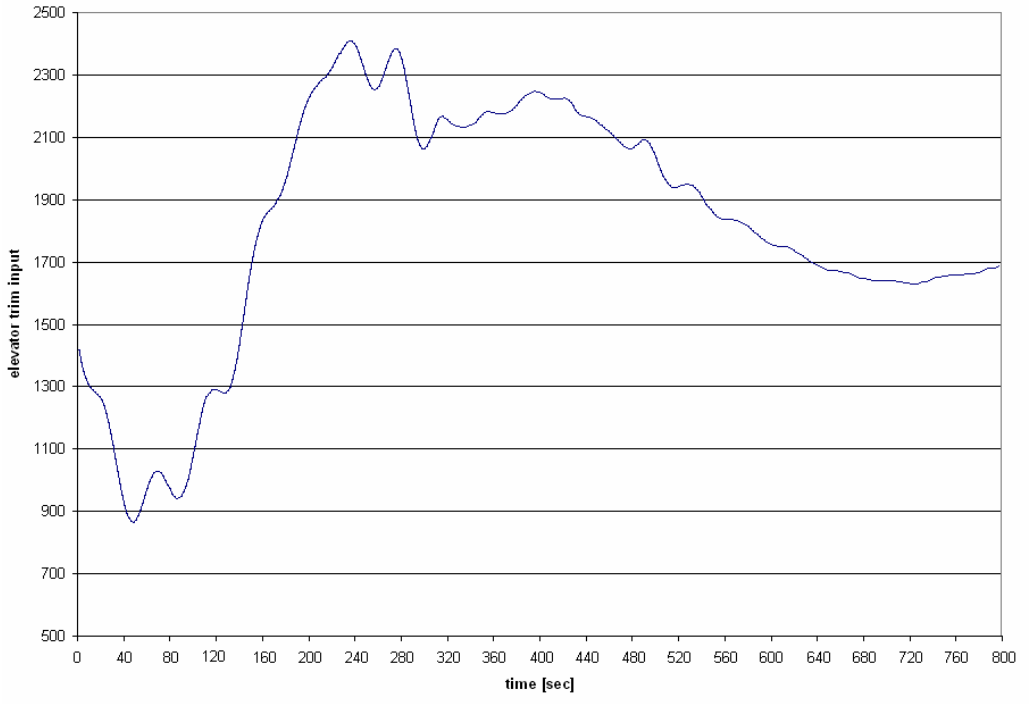


Figure 4.67: test #7 – elevator trim actuation

Graphs relatives to Test #8:

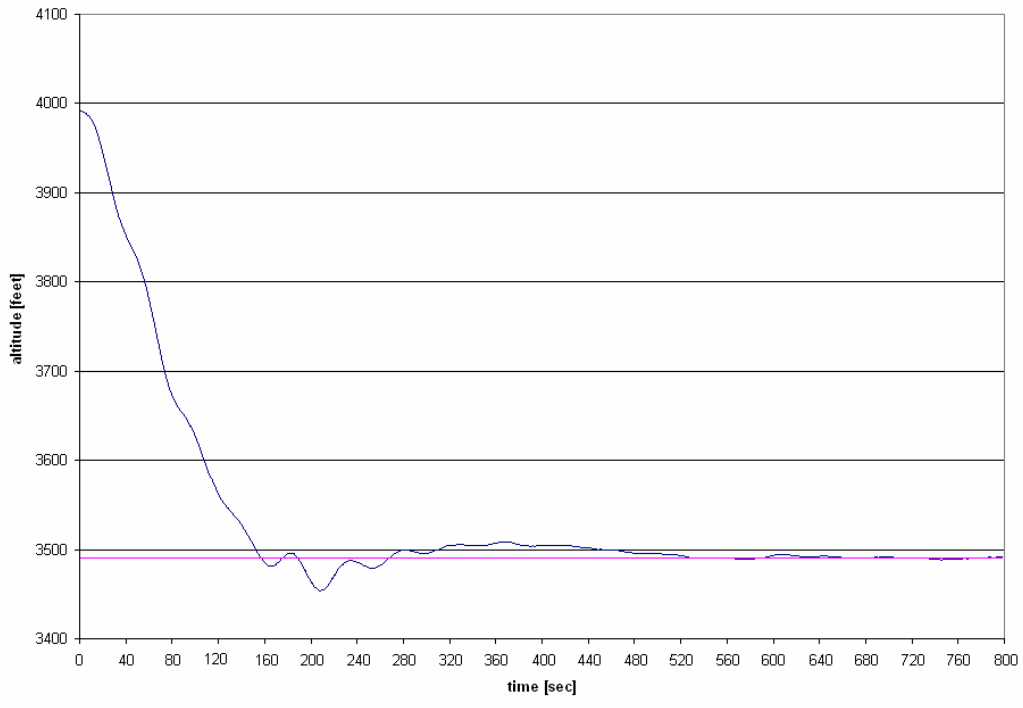


Figure 4.68: test #8 – altitude response

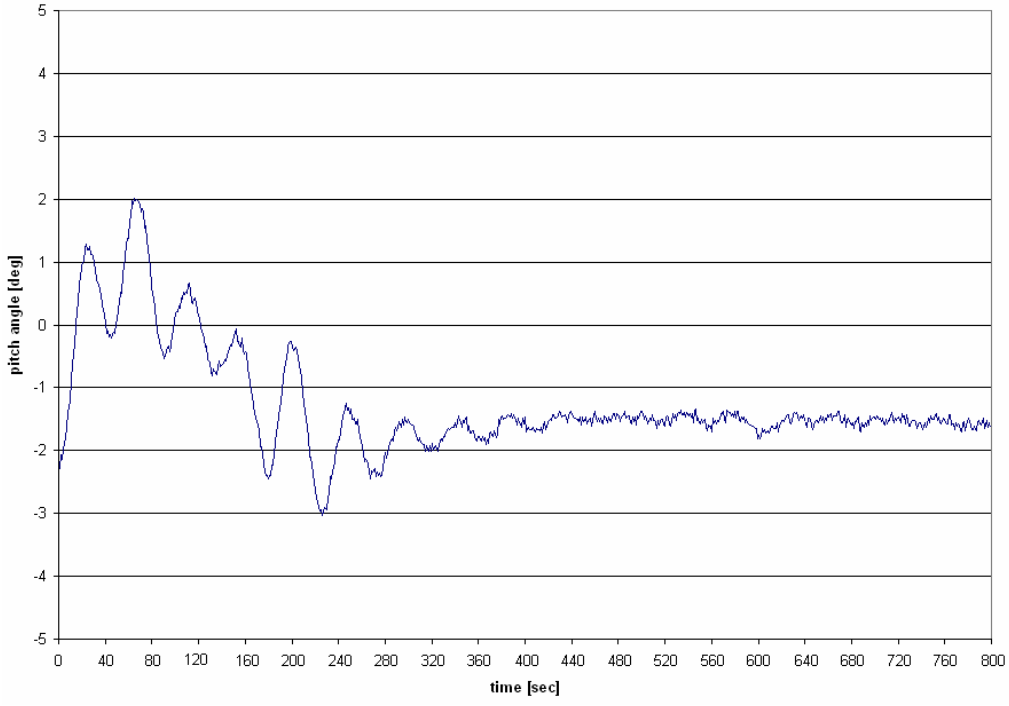


Figure 4.69: test #8 – pitch angle response

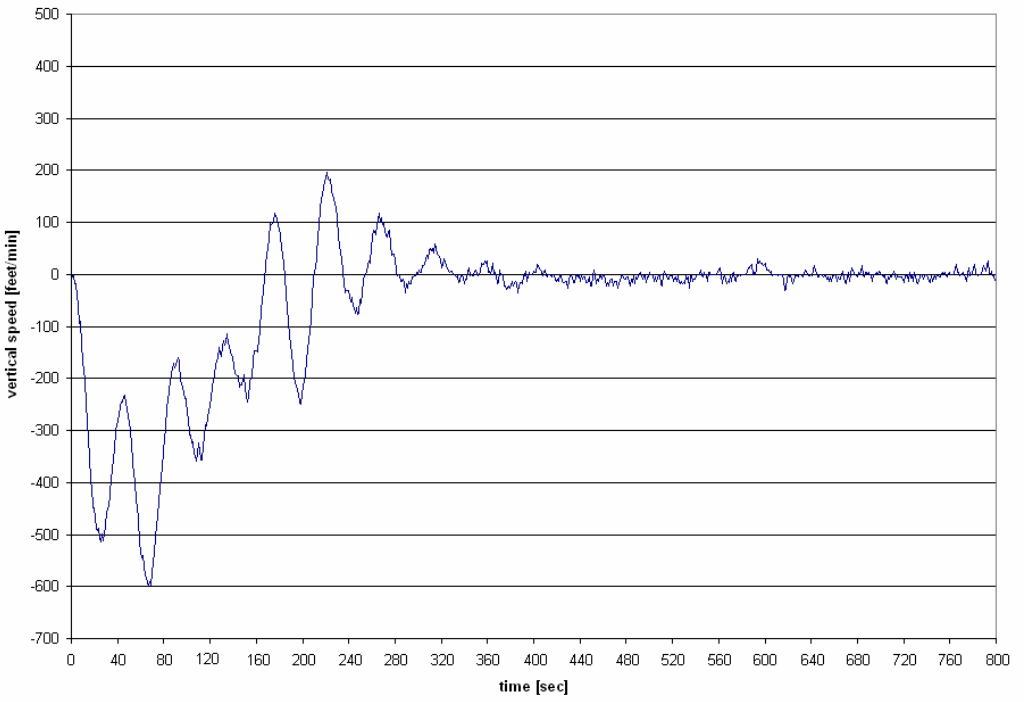


Figure 4.70: test #8 – vertical speed response

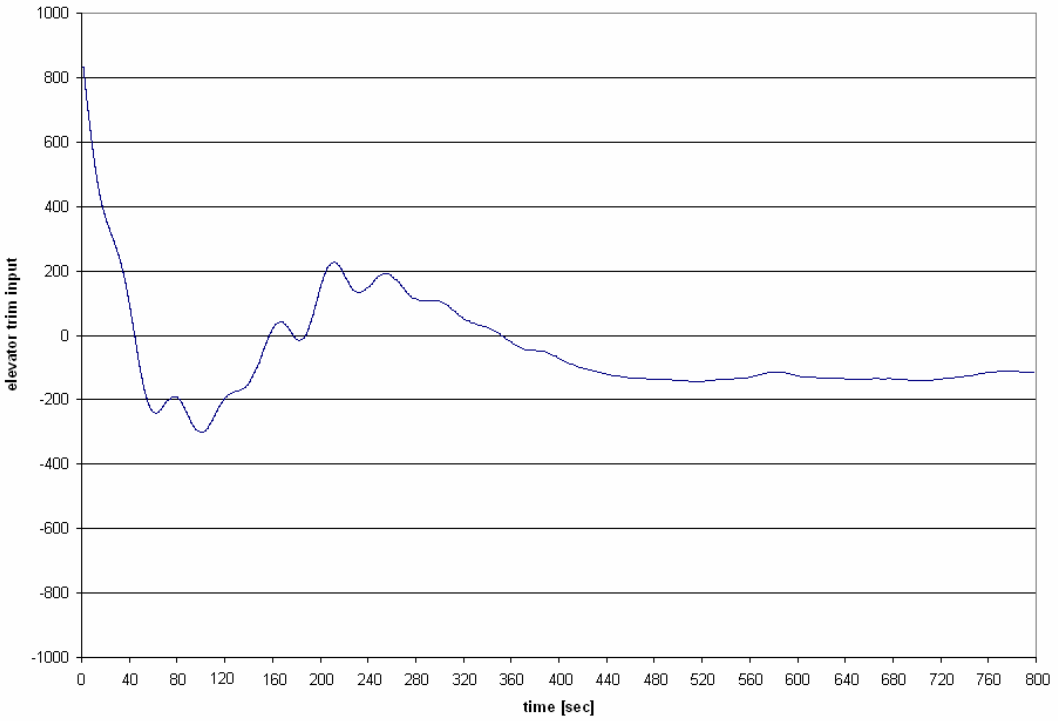


Figure 4.71: test #8 – elevator trim actuation

Graphs relatives to Test #9:

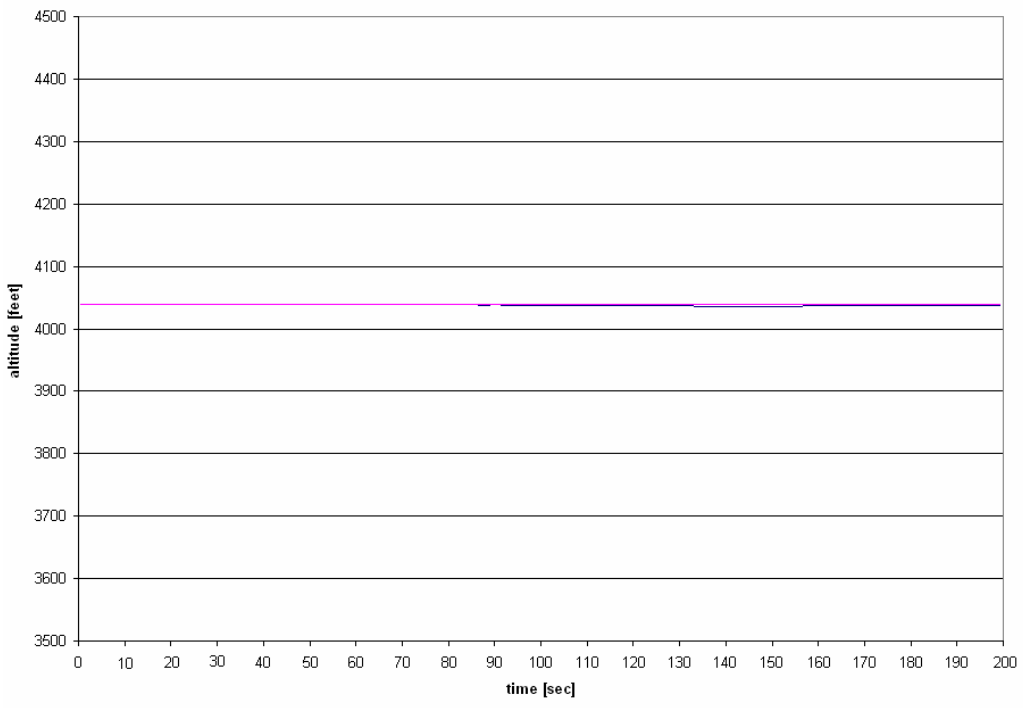


Figure 4.72: test #9 – altitude response

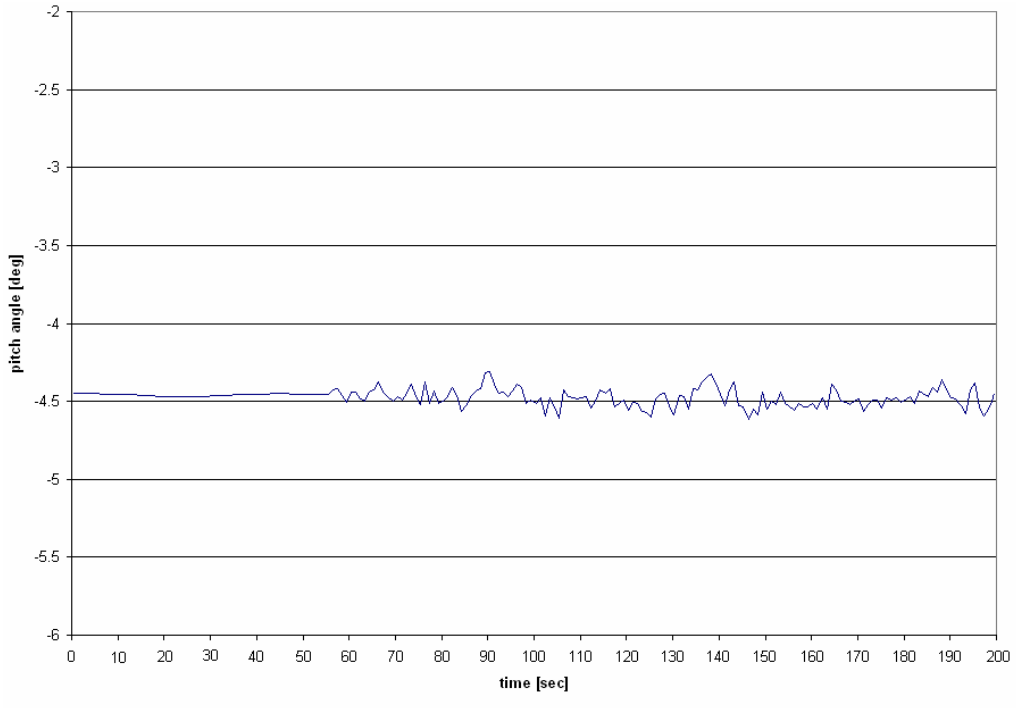


Figure 4.73: test #9 – pitch angle response

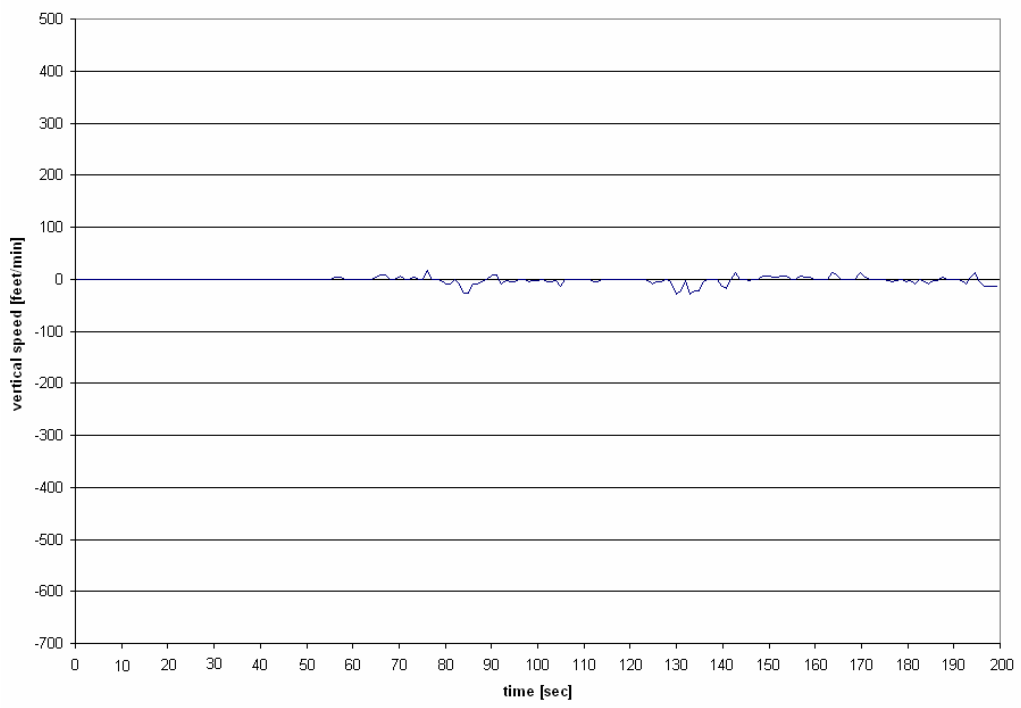


Figure 4.74: test #9 – vertical speed response

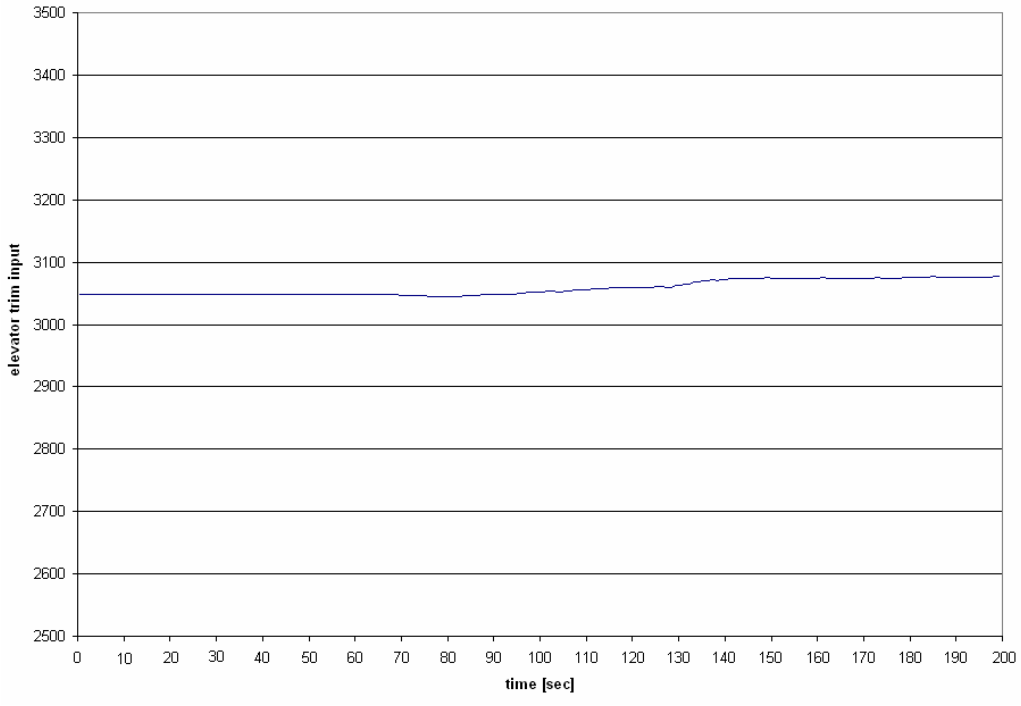


Figure 4.75: test #9 – elevator trim actuation

Graphs relatives to Test #10:

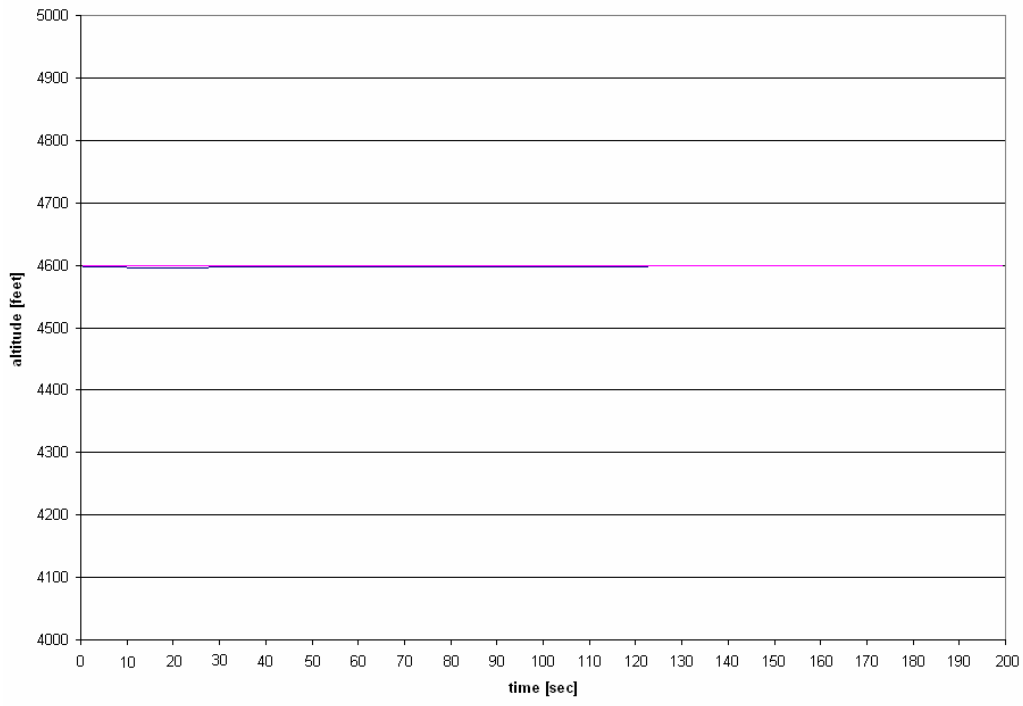


Figure 4.76: test #10 – altitude response

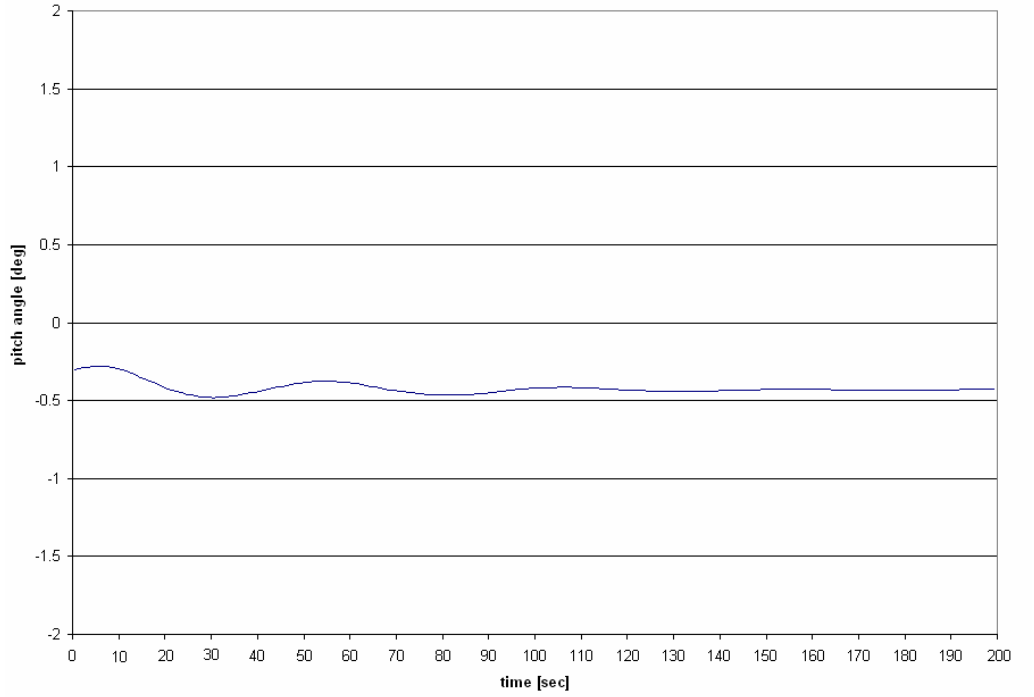


Figure 4.77: test #10 – pitch angle response

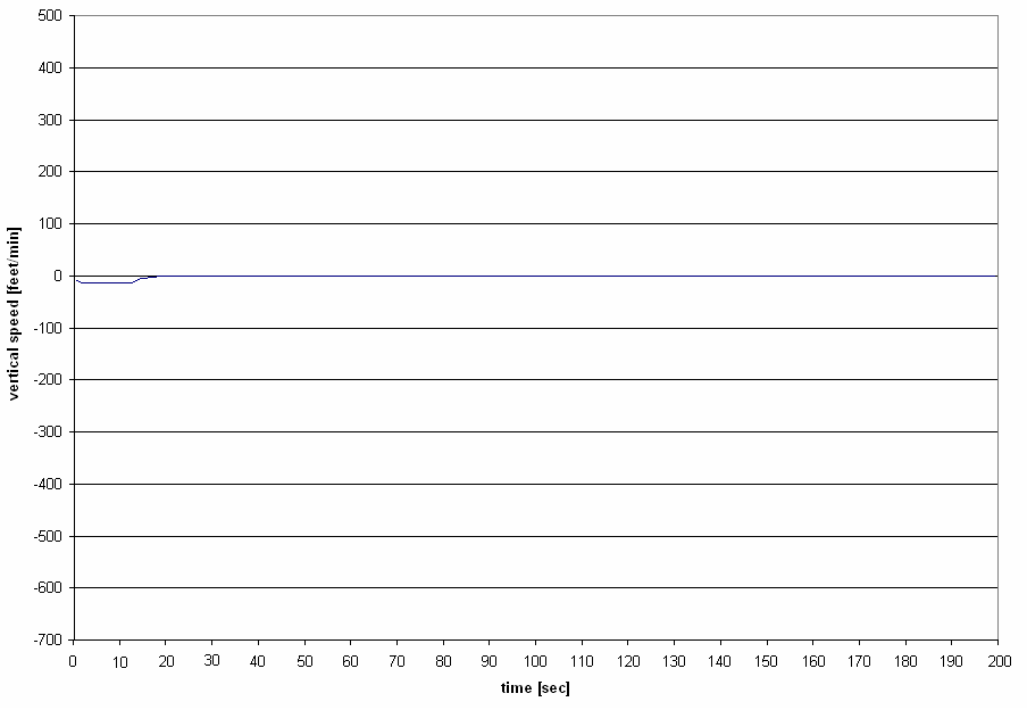


Figure 4.78: test #10 – vertical speed response

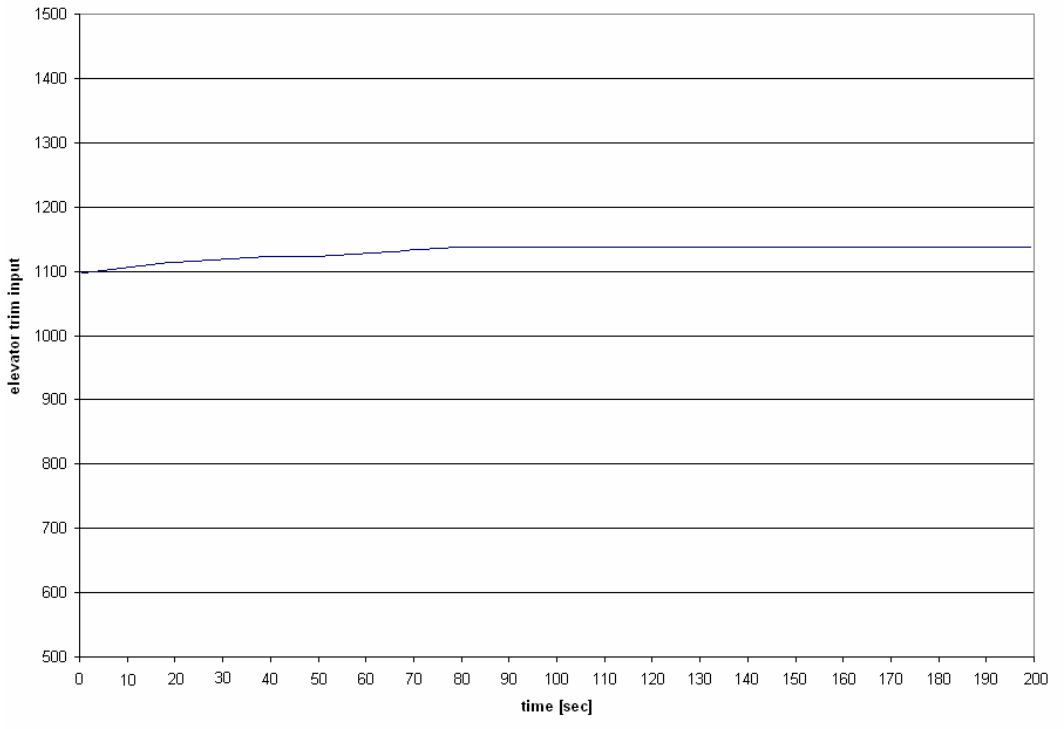


Figure 4.79: test #10 – elevator trim actuation

Graphs relatives to Test #11:

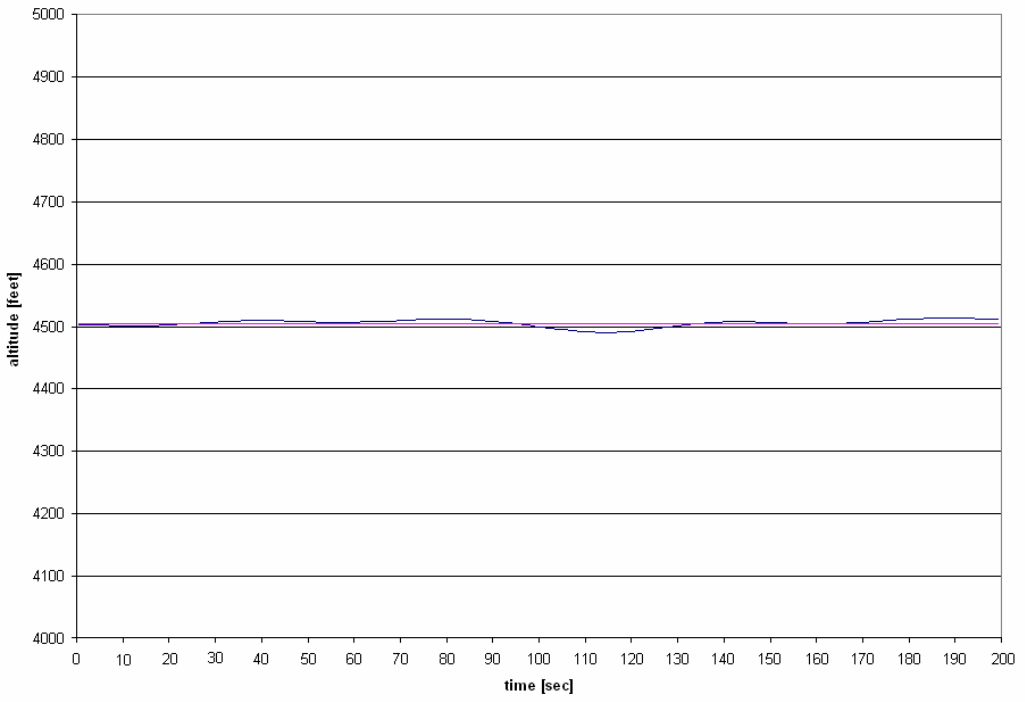


Figure 4.80: test #11 – altitude response

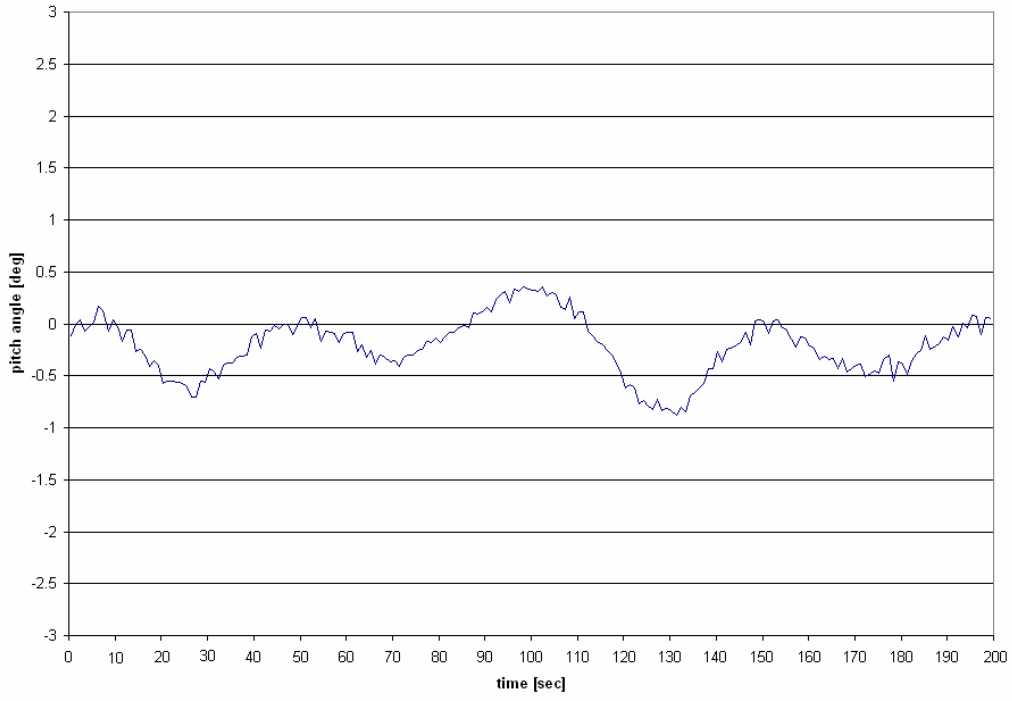


Figure 4.81: test #11 – pitch angle response

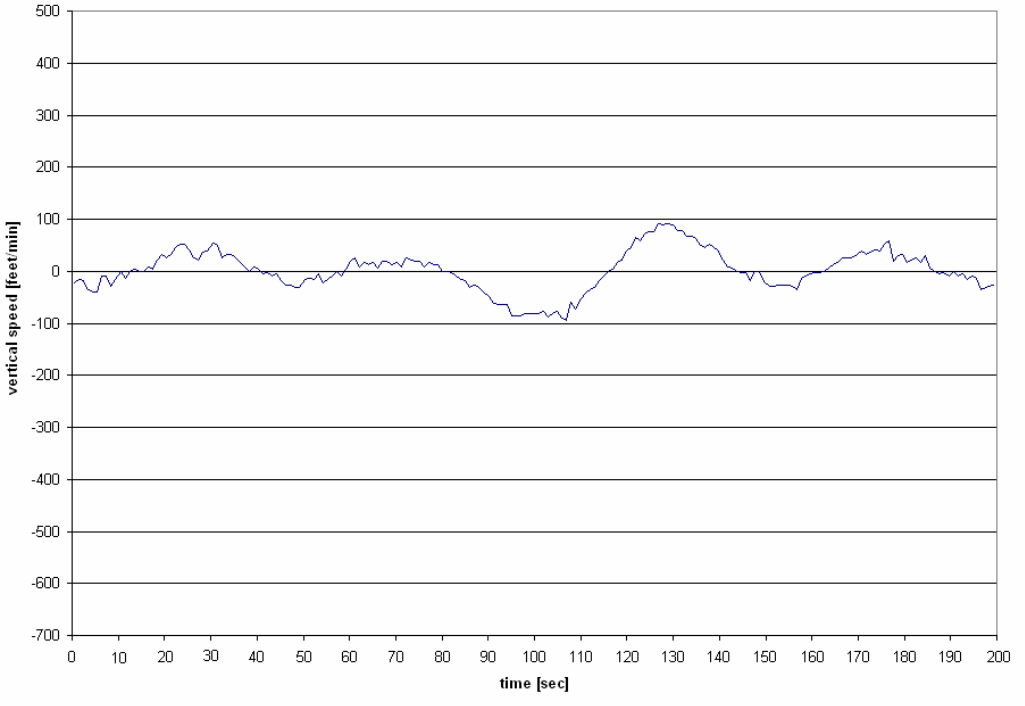


Figure 4.82: test #11 – vertical speed response

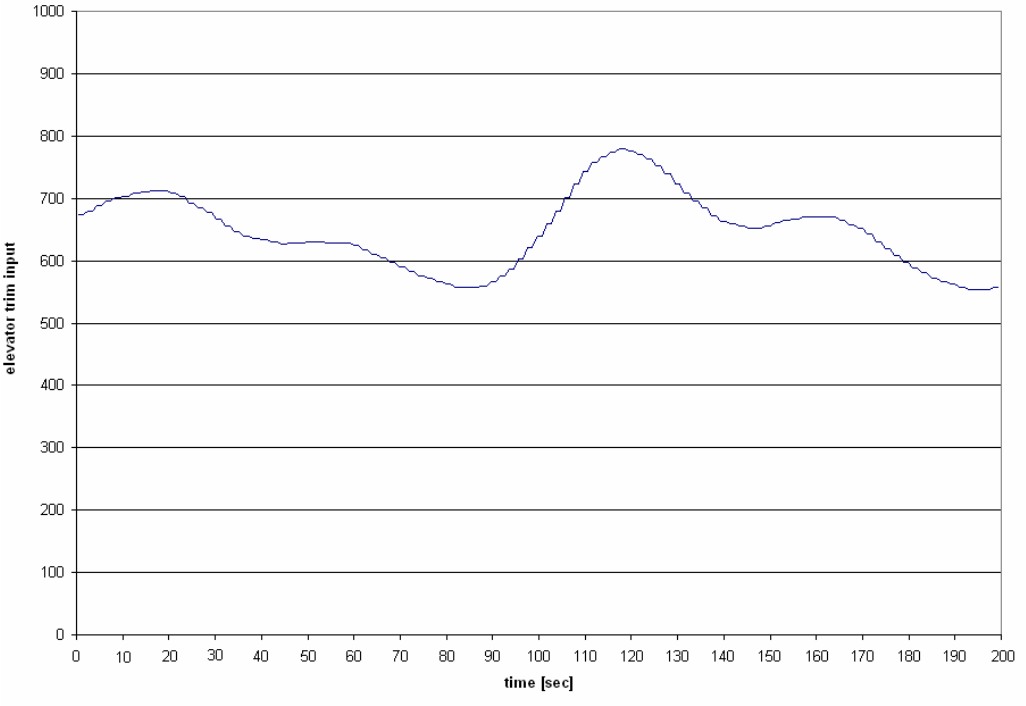


Figure 4.83: test #11 – elevator trim actuation

Graphs relatives to Test #12:

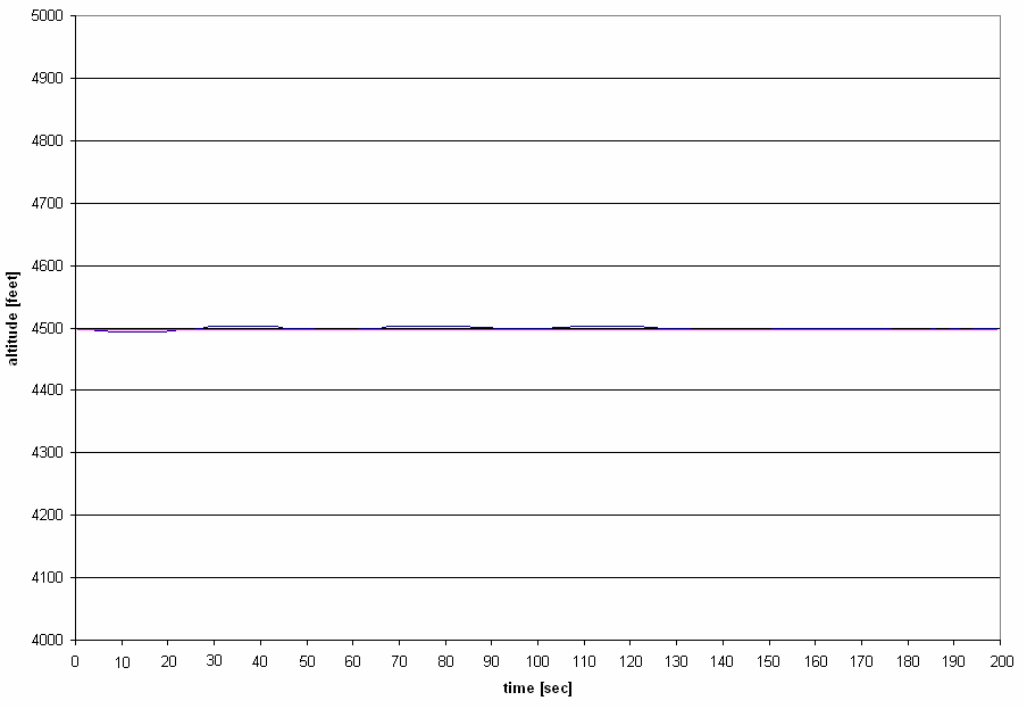


Figure 4.84: test #12 – altitude response

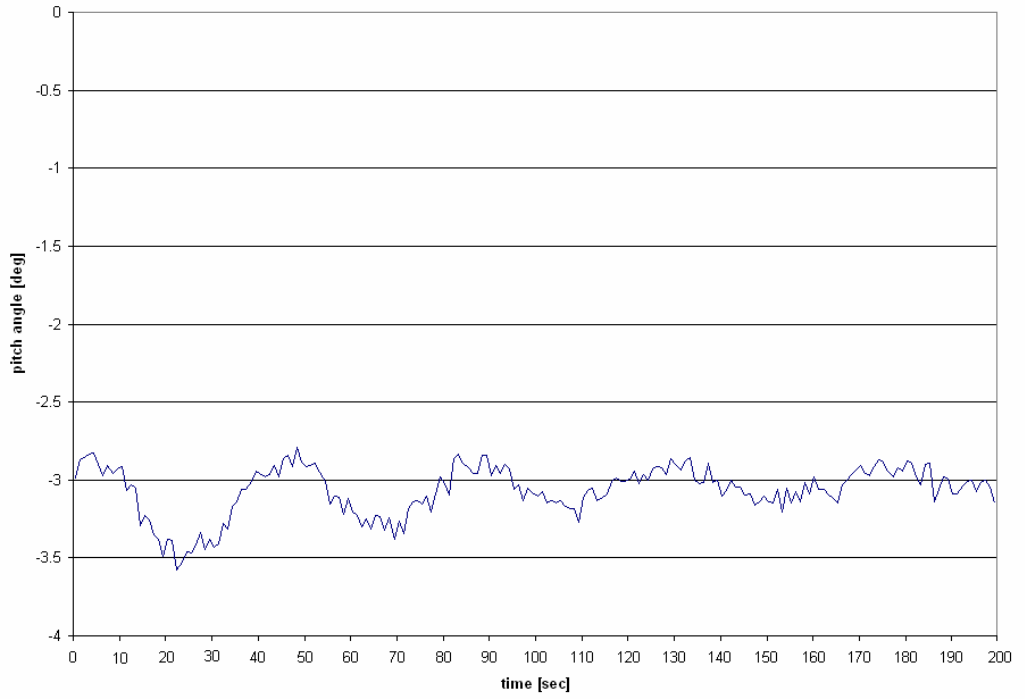


Figure 4.85: test #12 – pitch angle response

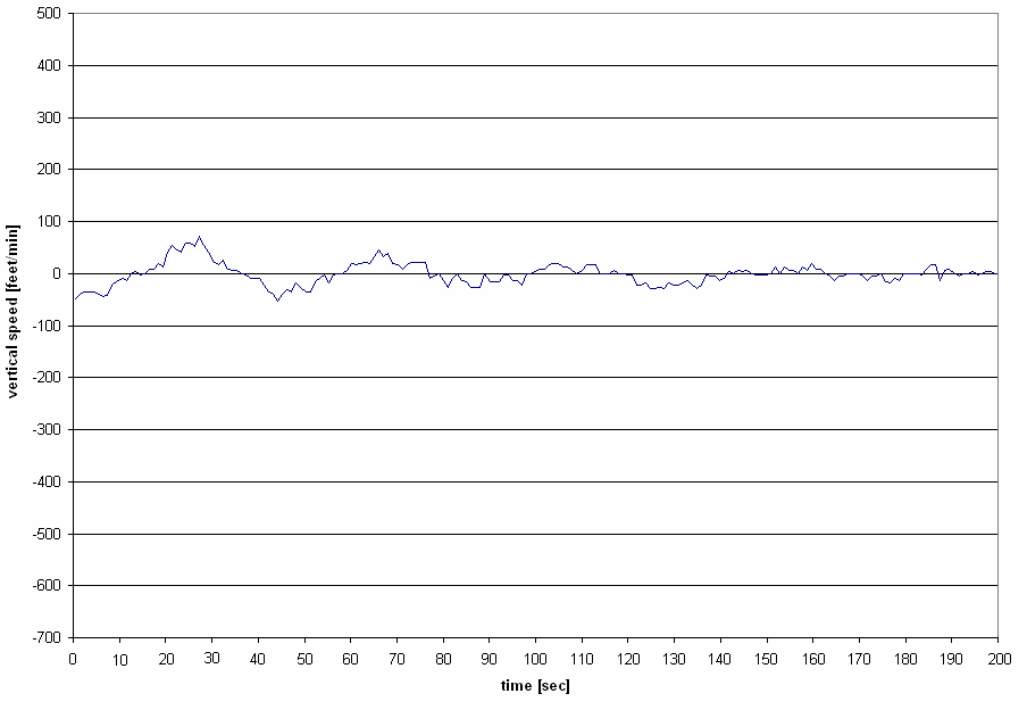


Figure 4.86: test #12 – vertical speed response

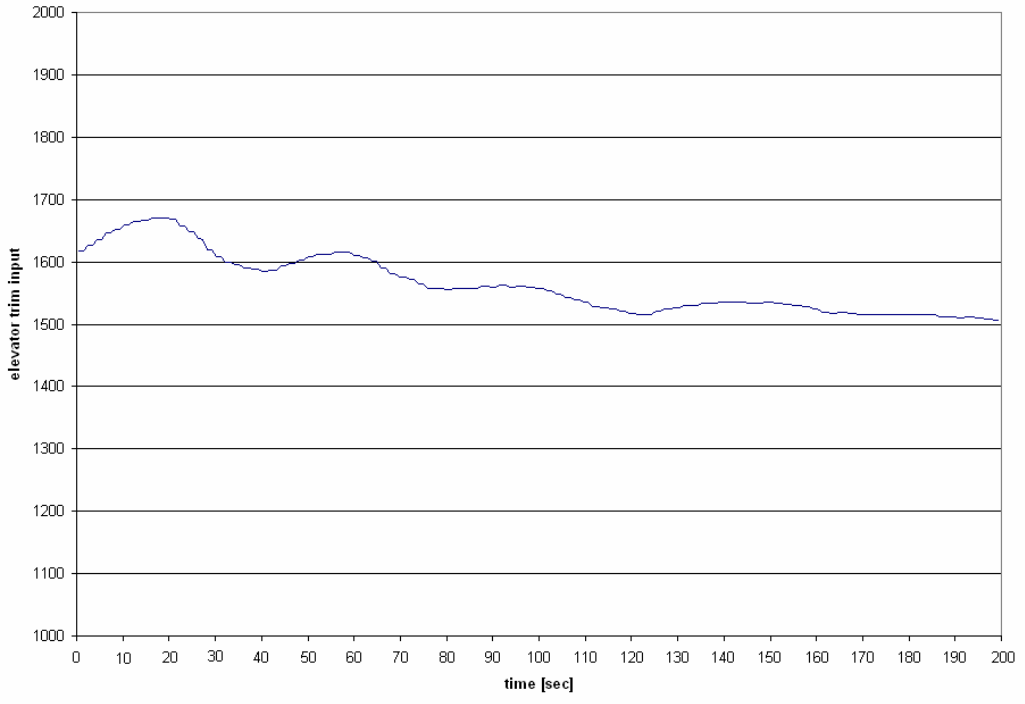


Figure 4.87: test #12 – elevator trim actuation

For this controller, the result analysis must be done separately for the three specific series of test. There are treated in order climb, descent and altitude maintenance.

For the climb of 500 feet, the controller achieves the required value on average after about 3'12", the absolute altitude fluctuations are included into the range [-27ft – 36ft], for this controller the incidence of weight is relevant because when aircraft load is minimum, the rise time is smaller than the case of maximum load. The relevance of weight is showed by pitch angle responses too because when aircraft load is minimum the fluctuations are minor and crew comfort is bigger. The maximum absolute pitch angle excursion varies by 3°18' measured in Test #1 to 5°13' measured in Test #2. Vertical speed variations are included in the range [425 ft/min – 816 ft/min], the absolute maximum value is stored in Test #2 (610 ft/min) and the minimum value in Test #4 (-95ft/min). Is useless to discuss on trim input because are relative values.

For the descent of 500 feet, the average value necessary to achieve the required altitude is 2'25", with the bigger time stored in Test #5 (2'47") and the smaller in Test #6 (1'57"). The descent involves a great number of pitch fluctuation because air speed increases when aircraft pitch down and then the lift increases causing an immediate pitch up, the maximum and the minimum pitch excursion is stored in Test #7 (7°52') and in Test #8 (5°02') respectively. About vertical speed, we can see which when the load is maximum (Test #5 and Test #7) the fluctuations are a lot and their magnitude are great. The absolute max and min value of vertical speed are stored in Test #7 and varies from -675 ft/min to 423 ft/min.

When we want maintain an assigned altitude everything going well because this controller works in a lovely fashion, we can see in the graphs relatives to altitude that the fluctuations are imperceptibles and the errors are unimportants, only in case of wind we have stored some little fluctuations but at global level this controller, for this aim is very accurate.

4.5 *navigation* controller testing

This controller works to carry on-route the aircraft and keep it on, testing of this controller was been developed like described in the following lines:

- configure a generic flight with an ATC flight plan;
- start the flight;
- keep aircraft straight and level on an heading closer than the GPS reference;
- run the controller.

Navigation controller is very similar to lateral controller in terms of heading following performance. For this reason we have used in each one test a normal weight configuration, with a total load of 2250 lb then, was been necessary only two tests carried out in two different weather conditions.

Aircraft configuration includes an ATC flight plan needed to allows to GPS to calculate the heading reference (*CTS*) that is sent to the controller, is possible to configure a ATC plan directly in-flight using the GPS panel available in the cockpit of our aircraft.

For the simulations used to develop tests and graph are used the following configurations relatives to weight, weather and type of input.

	weight	***weather***	***input [deg]***	***GPS***
test #1	**NORMAL**	**wind 340/00**	**CTS/GPS**	**ON**
test #2	**NORMAL**	**wind 090/16 + turbulences**	**CTS/GPS**	**ON**

Table 4.3: *navigation* controller testing table

The parameters observed and plotted in the graphs, for this controller testing, are in order: *heading* (current HDG and CTS), *pitch angle*, *roll angle* and *aileron input*. The following figures are relatives to aircraft position in Test #1: figure 4.65 represents the

GPS display at tests starting point and figure 4.66 shows the route track carried out by controller during the flight simulations.

Figure 4.88: test #1 – GPS display at simulation starting time

Figure 4.89: test #1 – track covered by aircraft during test simulation

Graphs relatives to Test #1:

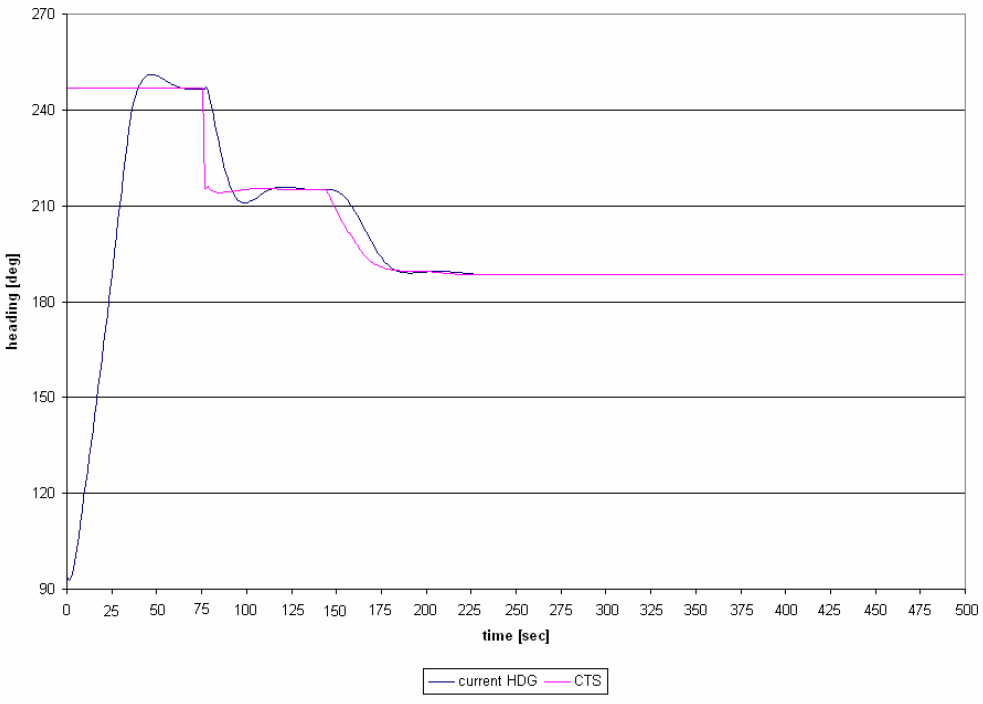


Figure 4.90: test #1 – heading response

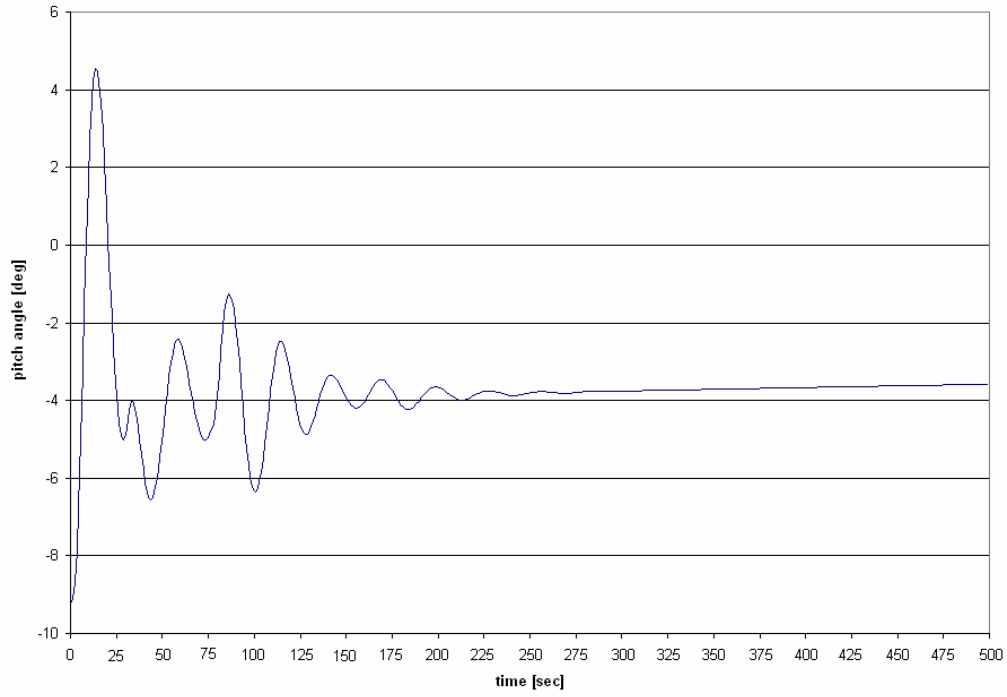


Figure 4.91: test #1 – pitch angle response

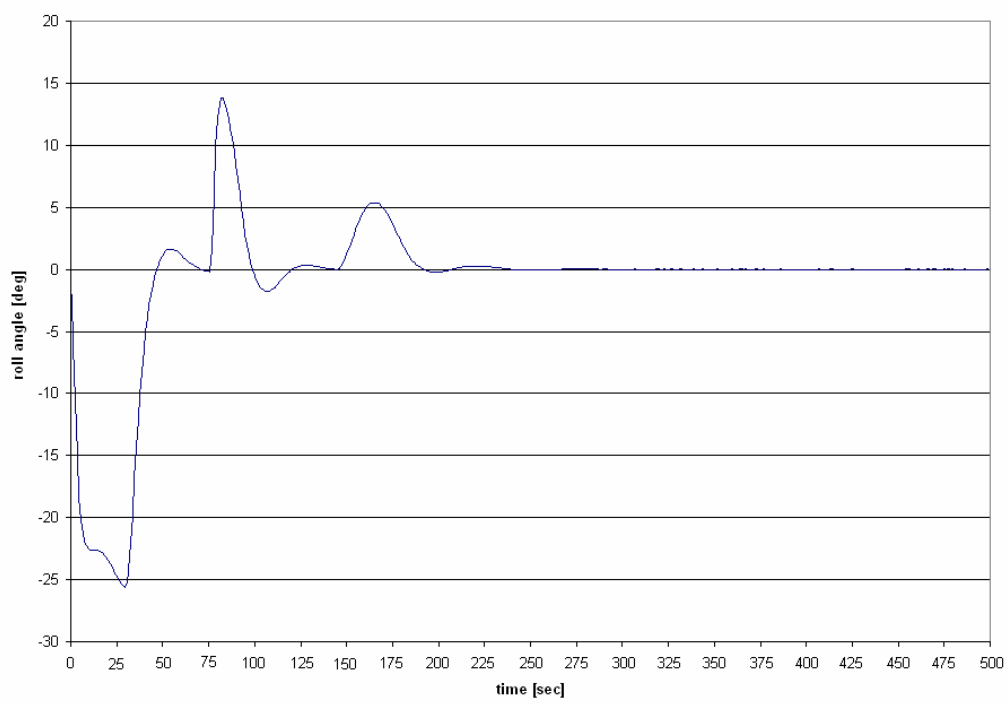


Figure 4.92: test #1 – roll angle response

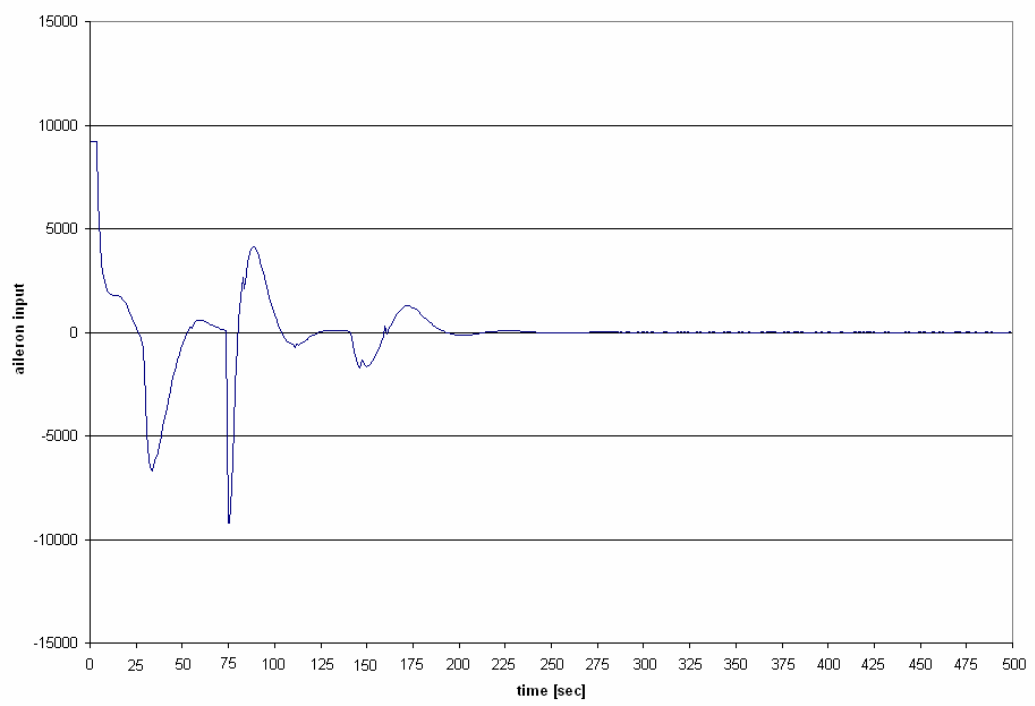


Figure 4.93: test #1 – aileron actuation

Figures 4.71 and 4.72, developed in Test #2, shows respectively the aircraft position at the starting point and the complete route track carried out during the simulation.

Figure 4.94: test #1 – GPS display at simulation starting time

Figure 4.95: test #2 – track covered by aircraft during test simulation

Graphs relatives to Test #2:

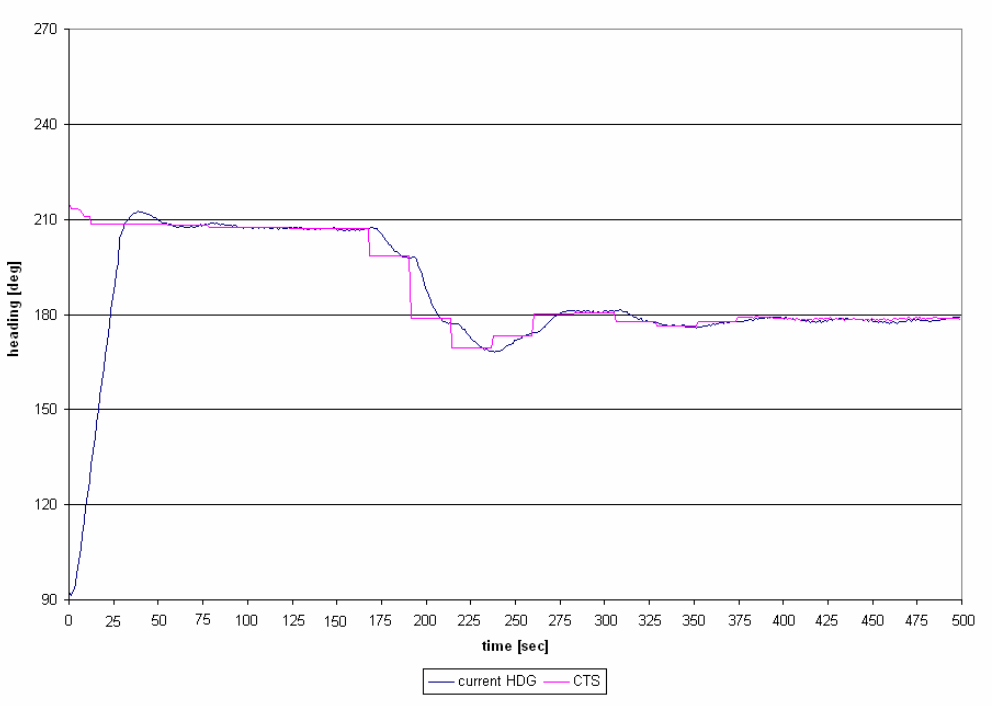


Figure 4.96: test #2 – heading response

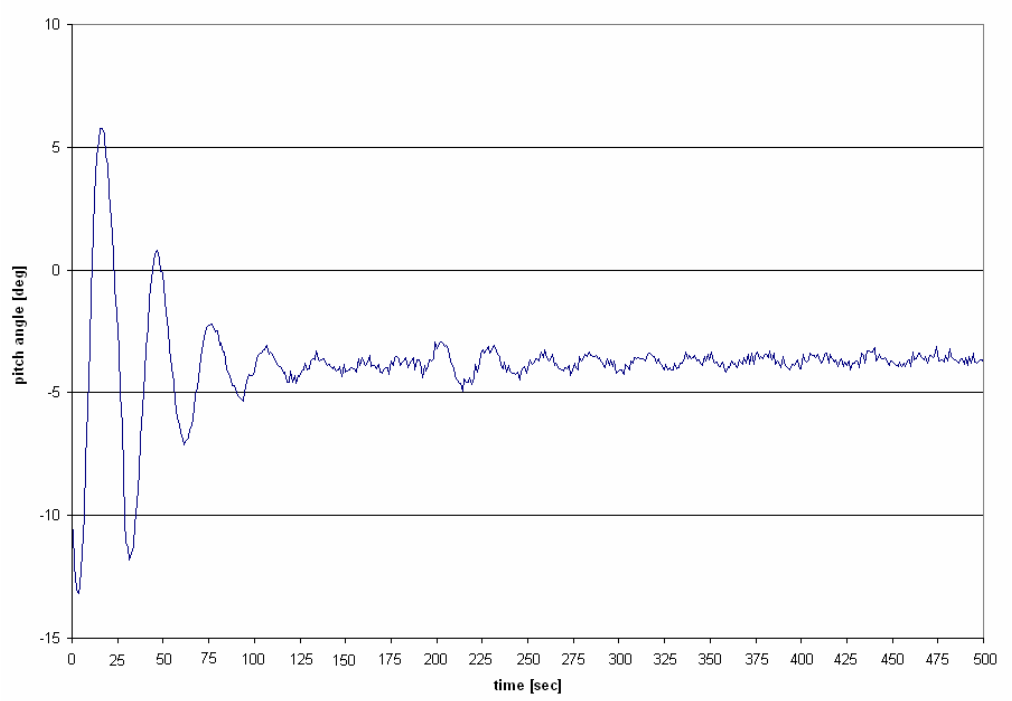


Figure 4.97: test #2 – pitch angle response

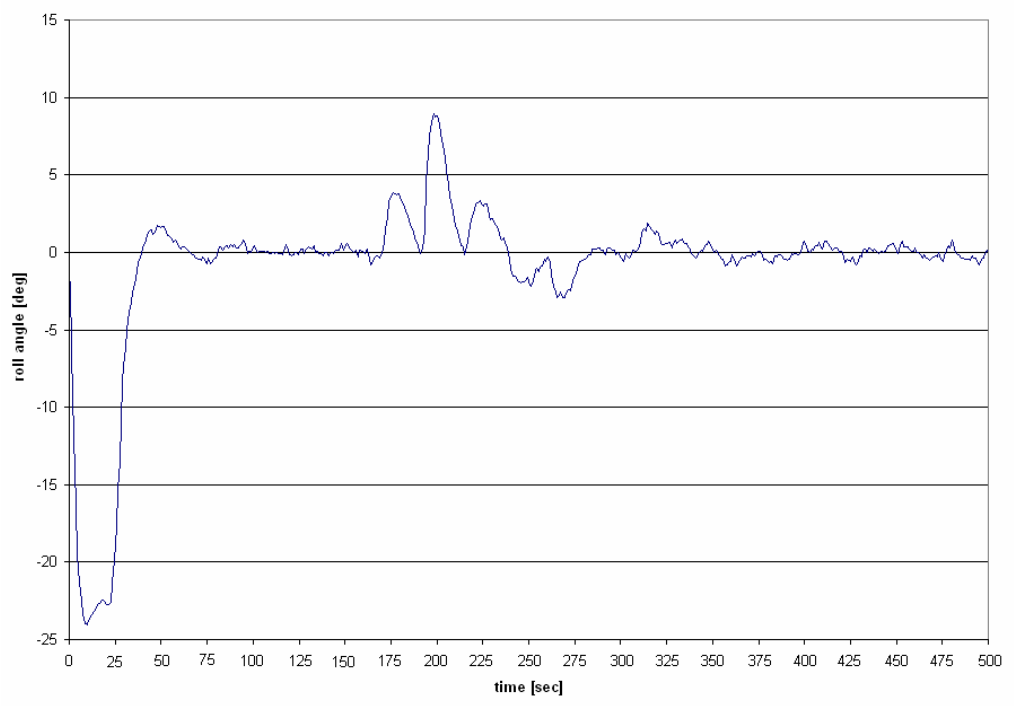


Figure 4.98: test #2 – roll angle response

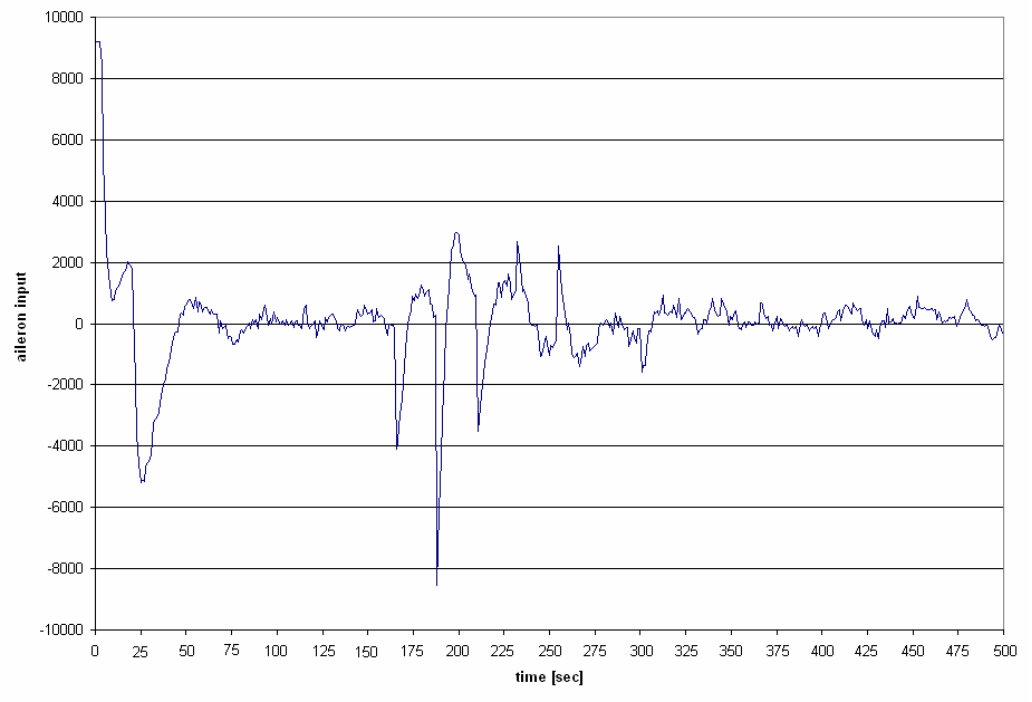


Figure 4.99: test #2 – aileron actuation

The important of GPS feedback for this controller testing is vital because we have used it to simulate a out-route situation to test the attitude of this controller to reach the GPS reference heading and to keep it stable.

This controller implements the same functionalities of *lateral* controller, for this testing we have performed two simulations in different weather condition, the most relevant parameter of weather is wind.

Is important to notice that in absence of wind, the airplane flight on-route than, in presence of wind, the airplane flight nearby the route line and presents little fluctuations of heading because of lateral wind disturbance, we can understand it comparing Figure 4.67 and Figure 4.73: in 4.67 when the aircraft looks the assigned route, flies straight without heading errors than, in Figure 4.73 we can see that because of wind, the CTS reference signal fluctuates nearby the route line and it determines little heading errors.

Figures 4.68 and 4.74 are relatives to pitch angle performance; like *lateral* controller, we can see that wind and turbulence produces instability but this controller isn't able to correct pitch fluctuations. Roll angle performances, showed in Figures 4.69 and 4.75, are interesting because we can see that the airplane rotates on the roll axis until current heading becomes the reference value and after this, in absence of wind the roll angle value keeps stable and next to 0 than, in presence of wind and turbulences, this value has fluctuations of little magnitude.

Wind effects are more evident if we see the figures relatives to ailerons actuation, in windless conditions the ailerons keeps still after the required heading value than in case of presence of adverse weather conditions ailerons moves frequently to correct the error induced by lateral component of wind, Figure 4.76 shows the frequents little corrections given by ailerons.

4.6 *lateral + longitudinal* controllers testing

This paragraph explain the preparation and the results of the test that combines the two functionalities of *lateral* and *longitudinal* controllers; the aim of this experiment is to verify if these controllers are compatible and if they can work in the same time to control aircraft heading and altitude in an efficient way. The details of these tests are showed in Table 4.4, we can see that the load configuration of the aircraft is "normal" or rather maximum fuel, one pilot and one passenger for a weight amount of about 2250 lb; weather variability is described in the 3° column, the first test is carried out windless while in the second the wind magnitude was 16 knots and there was turbulences in-altitude.

For each one test we have imposed to controller the altitude keeping (referred to the altitude value read at the starting time) combined with a turn right of 90 degrees.

	weight	***weather***	***input [feet,deg]***	***GPS***
test #1	**NORMAL**	**wind 340/00**	**altitude hold, +90°**	**OFF**
test #2	**NORMAL**	**wind 000/16 + turbulences**	**altitude hold, +90°**	**OFF**

Table 4.4: *lateral + longitudinal* controllers testing table

These tests produces a series of value that, after to be elaborated, are resumed in a set of graphs. In this case, function *Plotter* was been configured to read seven flight parameters because we have to monitoring the parameters of lateral and the parameters of longitudinal both.

The parameters monitored and plotted in the graphs are in order: heading, altitude, pitch angle, roll angle, vertical speed, aileron input and elevator trim input. After that we can see the graphs relatives to the tests carried out.

Graphs relatives to Test #1:

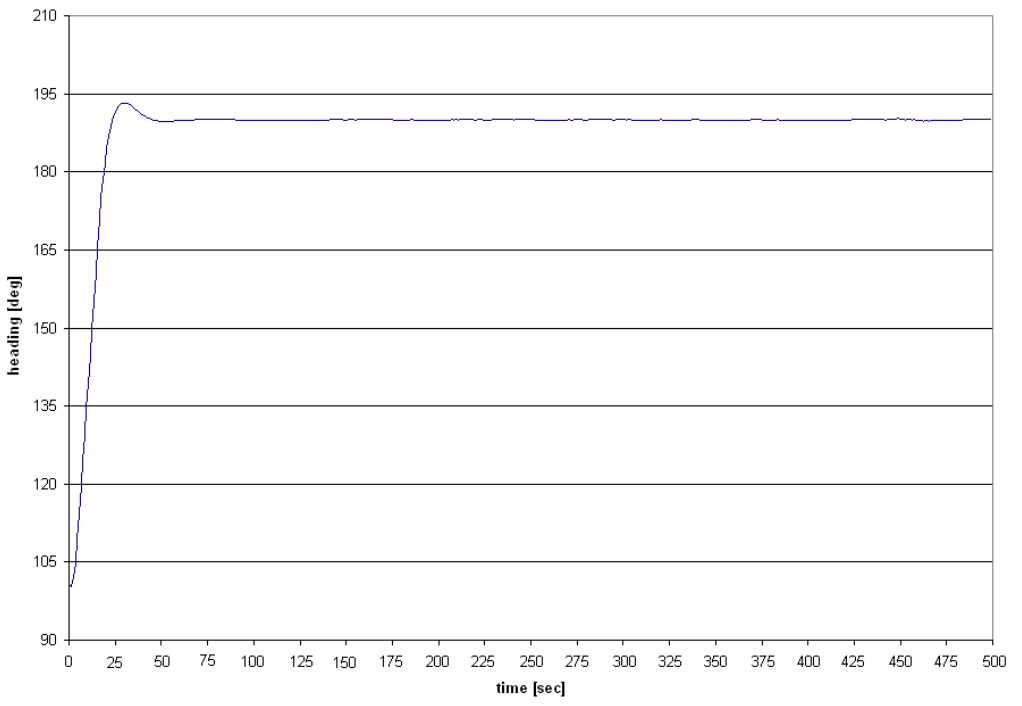


Figure 4.100: test #1 – heading response

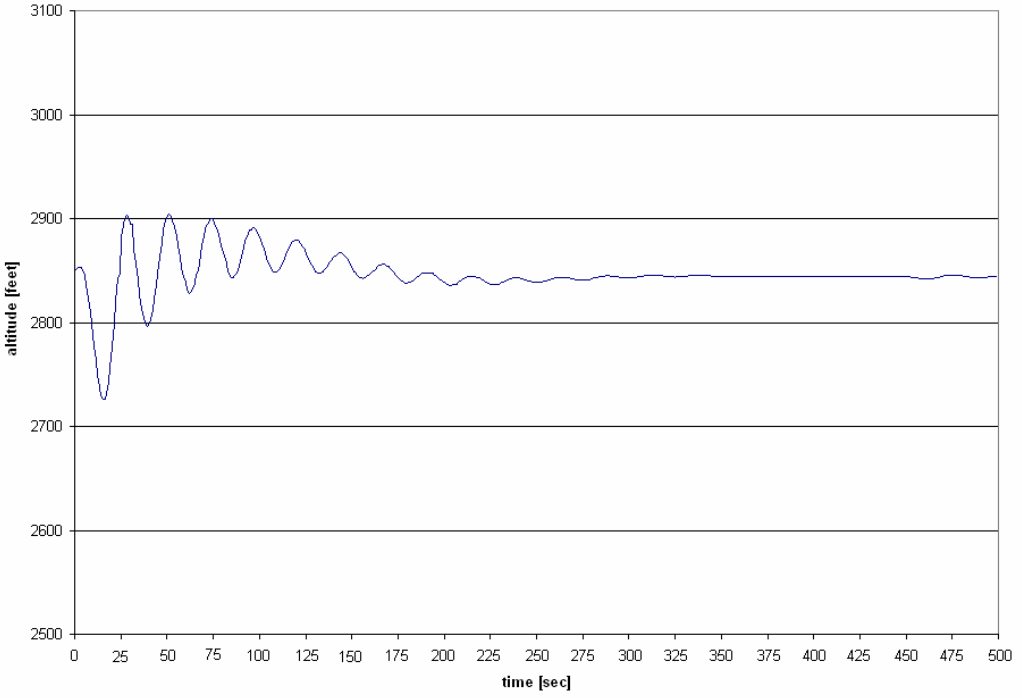


Figure 4.101: test #1 – altitude response

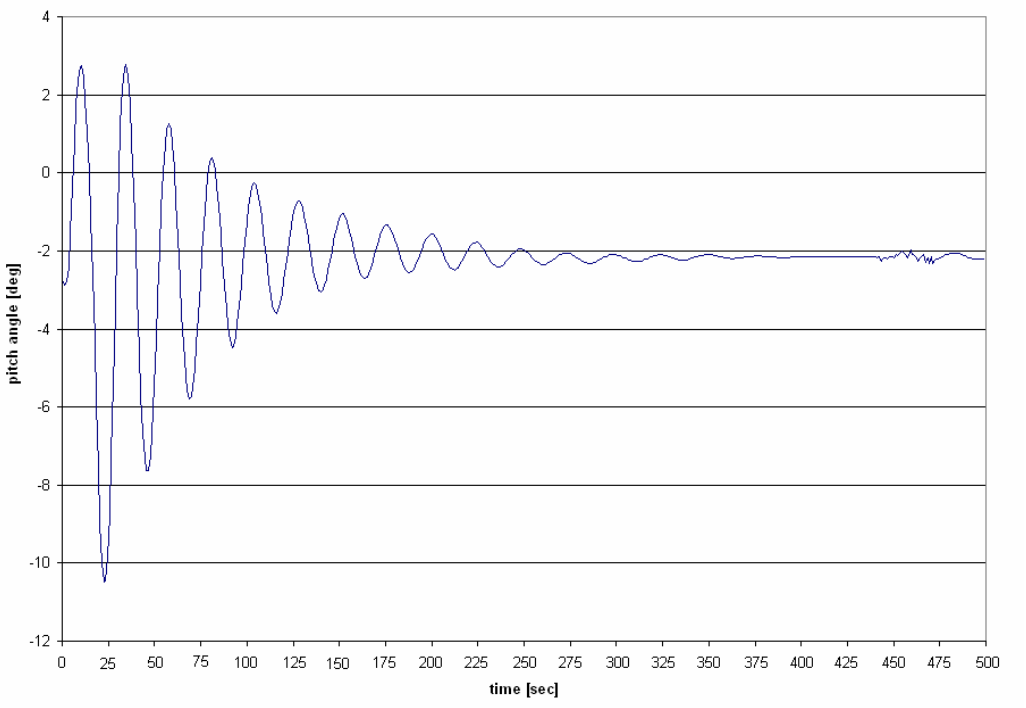


Figure 4.102: test #1 – pitch angle response

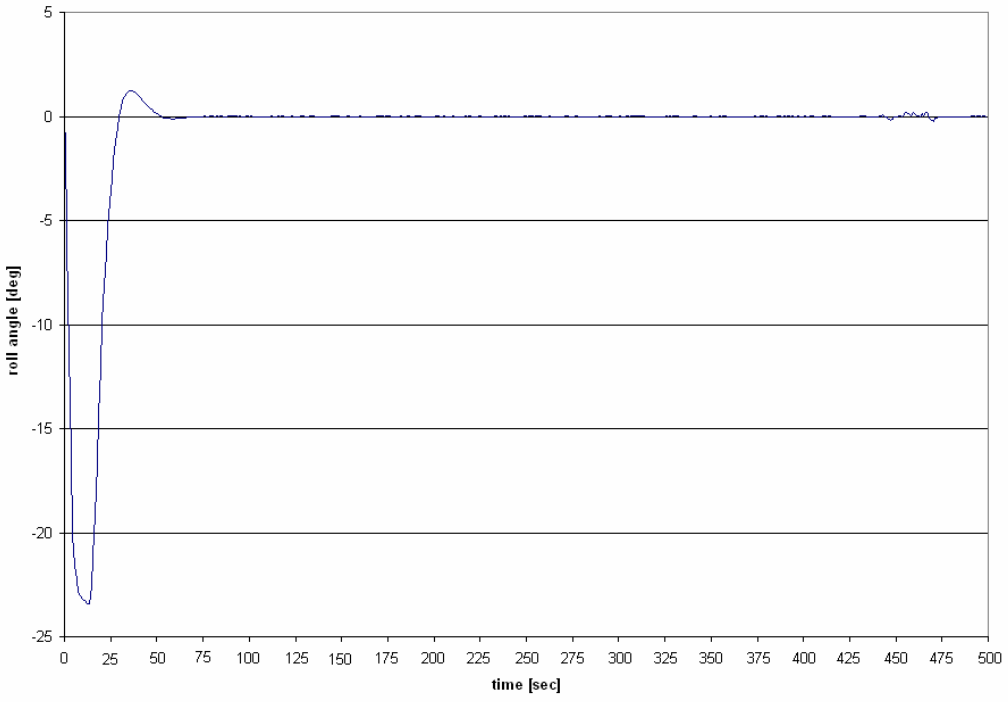


Figure 4.103: test #1 – roll angle response

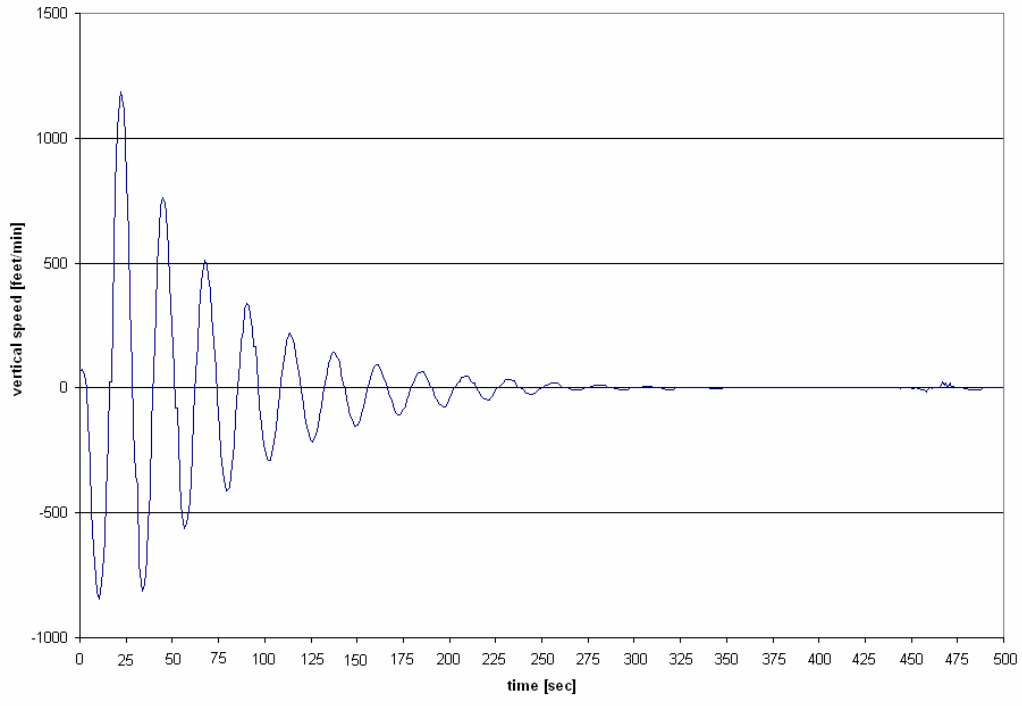


Figure4.104: test #1 – vertical speed response

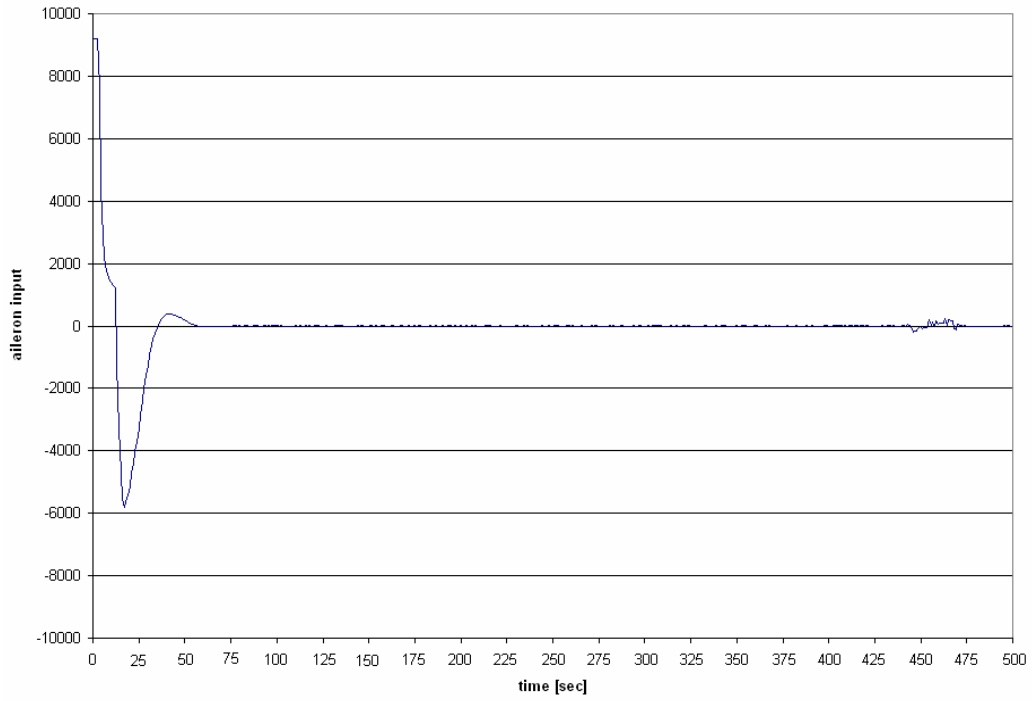


Figure 4.105: test #1 – aileron actuation

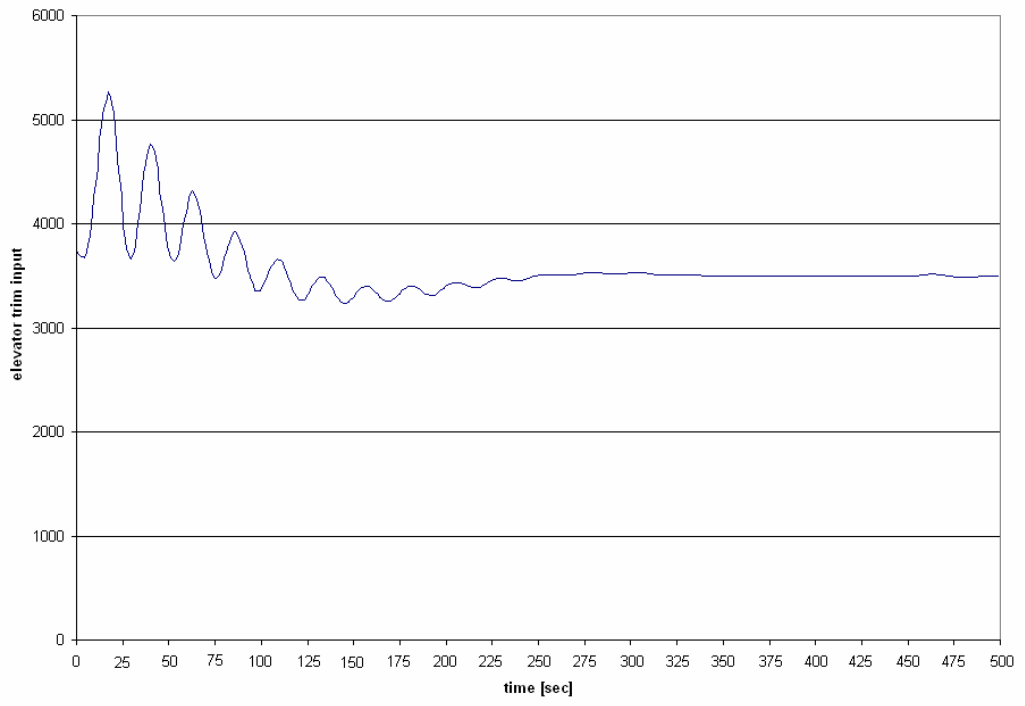


Figure 4.106: test #1 – elevator trim actuation

Graphs relatives to Test #2:

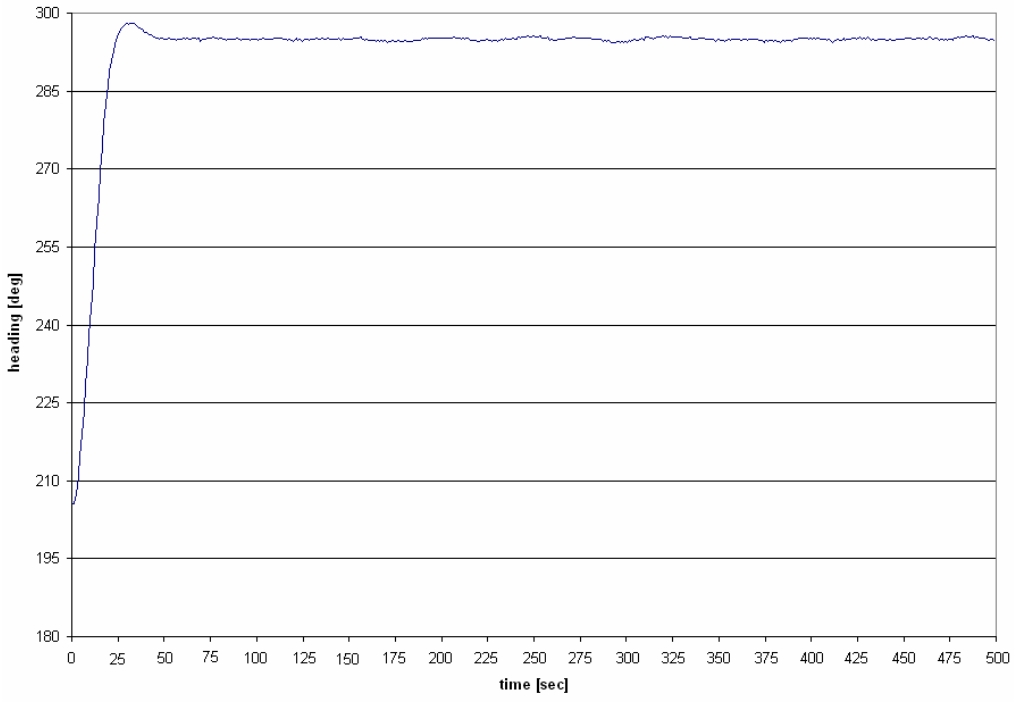


Figure 4.107: test #2 – heading response

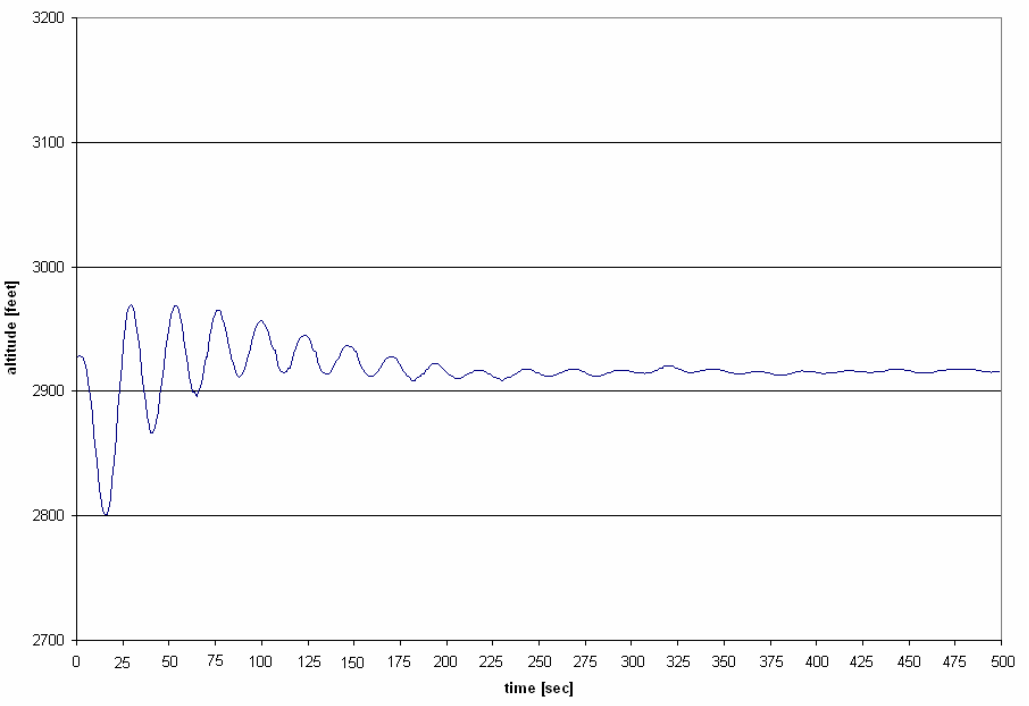


Figure 4.108: test #2 – altitude response

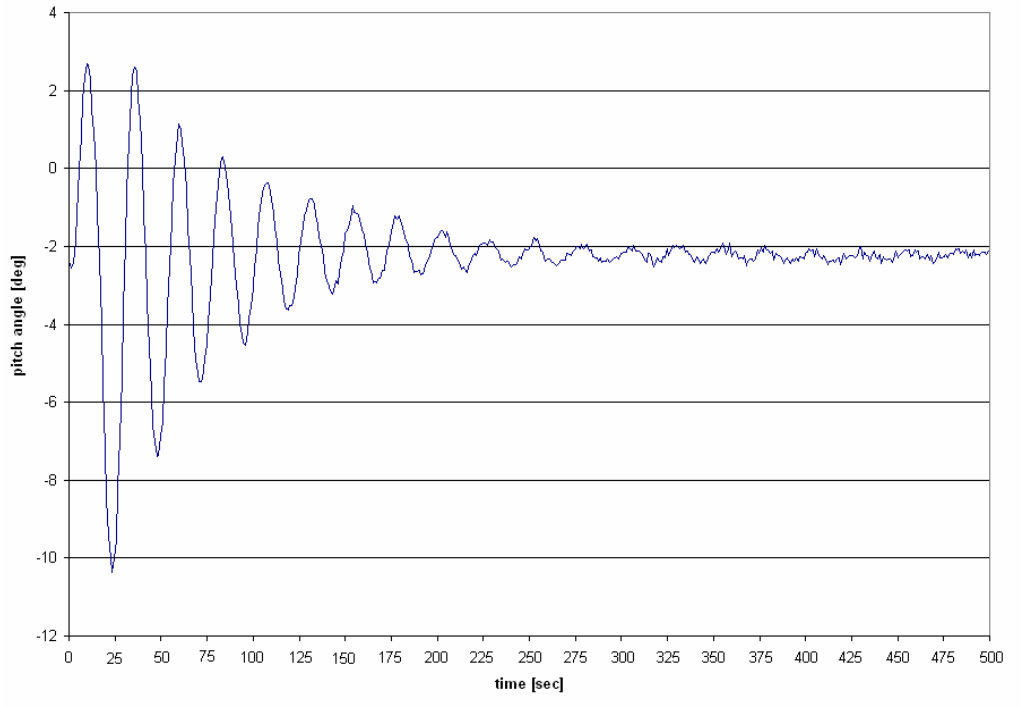


Figure 4.109: test #2 – pitch angle response

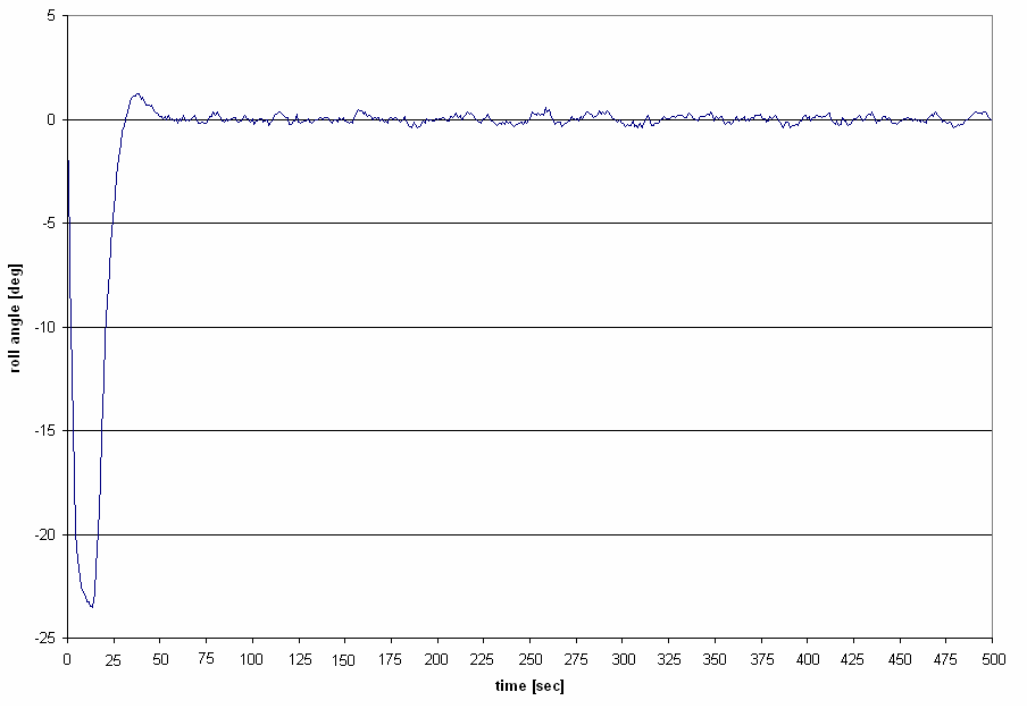


Figure 4.110: test #2 – roll angle response

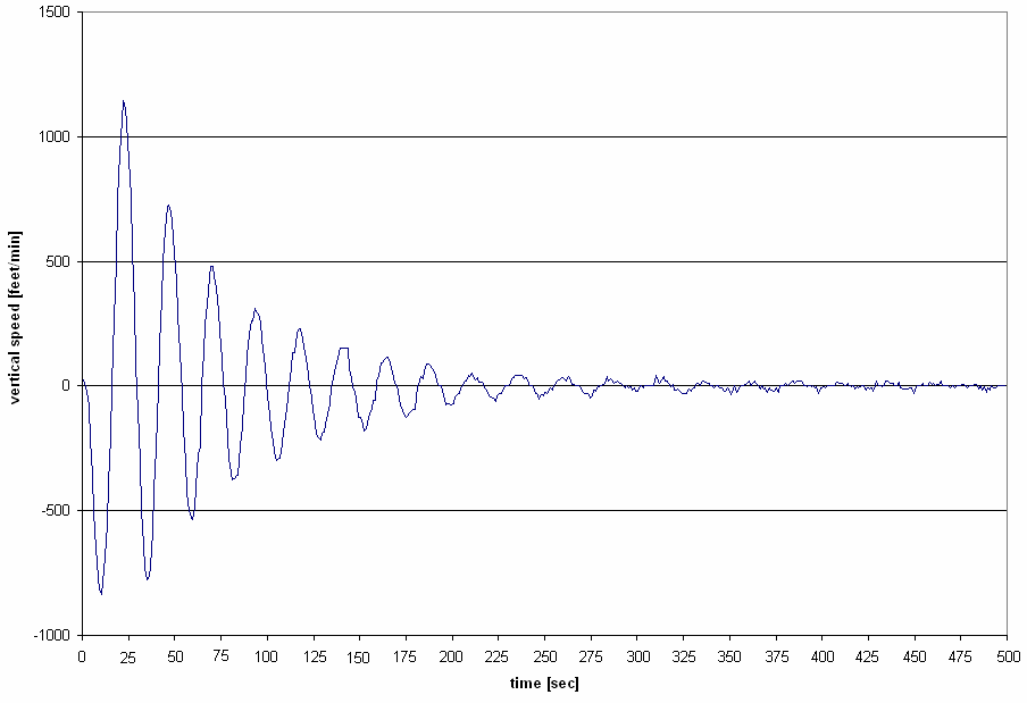


Figure 4.111: test #2 – vertical speed response

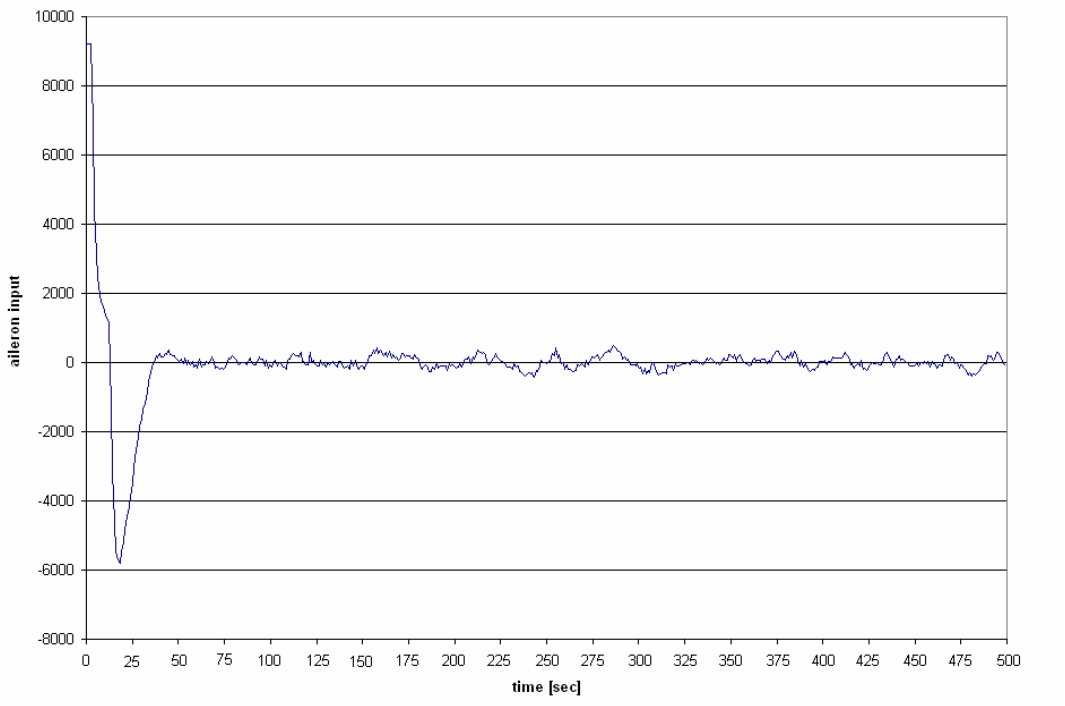


Figure 4.112: test #2 – aileron actuation

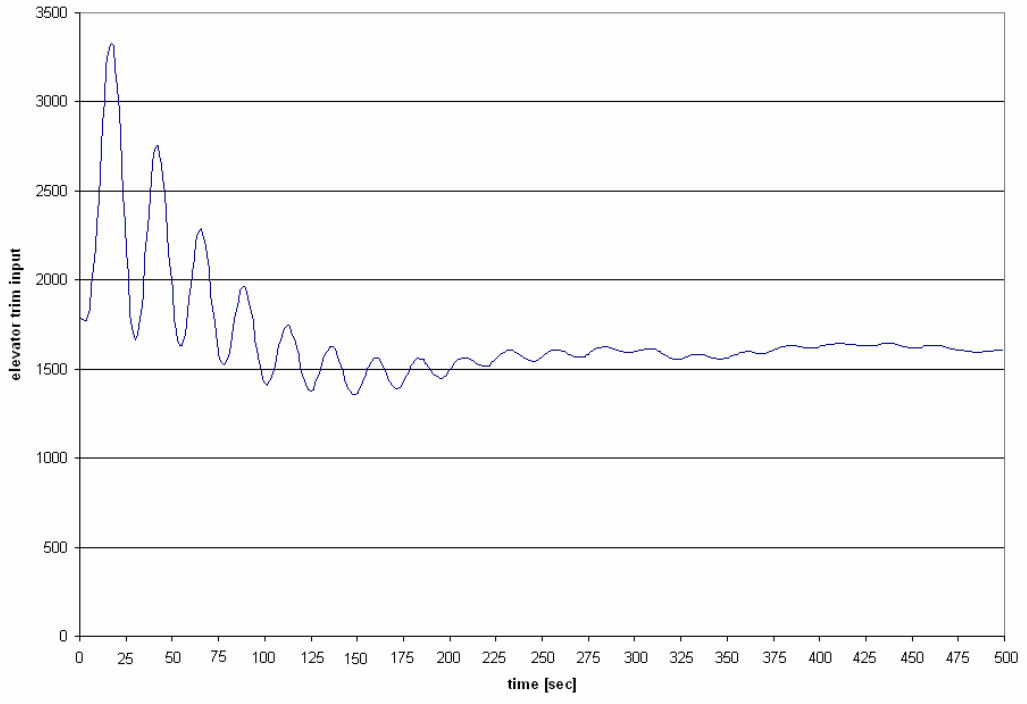


Figure 4.113: test #2 – elevator trim actuation

The two controllers used to carry out these tests works independently and at same time without produce yaw interferences, but producing pitch effects especially when aircraft turns left or right because, like we have explicated in the paragraphs above, a generic turn produces a loss of lift that involves a diminution of pitch angle, or in general, a turn down of aircraft muzzle.

Lateral controller combined with longitudinal controller, works in an independent fashion, it signify that the graphs produced, compared with the graphs produced in the lateral controller testing, doesn't return significant differences and then we are able to state that in this configuration, this controller doesn't feel the influence of longitudinal controller. For roll angle the talk is the same, i.e. there aren't difference between this test and that carried out only using lateral controller.

Notice that the global performance in terms of altitude keeping are better because the airplane flies straight and level after about 200", the heading stability is achieved before altitude stability because on average are sufficient about 50" to keep stable the aircraft direction.

The maximums altitude excursions are measured immediately after the controllers running and they are 177 ft in Test #1 and 169 ft in Test #2, in Test #1 the altitude value keeps limited in confidence range.

Vertical speed performances, showed in Figures 4.81 and 4.88, presents a great values immediately after the surfaces actuations, it fluctuates from -846 ft/min to 1184 ft/min in Test #1 and from -836 ft/min to 1145 ft/min in Test#2. The secondary lobes of these graphs present normal values included more or less in the range ±750 ft/min. By the 3rd order lobes, we can measure values included in the range [-700 ft/min – 500 ft/min] that represents the limit values imposed by design.

In these tests, wind produces the same effects measured in the previous testing procedures than is superfluous talking about it.

4.7 *navigation + longitudinal* controllers testing

Like the previous testing, this combines the functionalities of *navigation* controller (that allows to carry and keep an aircraft with a flight plan on the assigned route) and *longitudinal* controller that works, in this case, to keep the altitude measured at the starting time. The test configurations are listed in table 4.5, we have used a normal load configuration like that used in the previous testing, a wind variable from 0 knots (wind null) to 16 knots with turbulences, the aim of the testing is to verify the performance of altitude keeping allowed by lateral controller united to the performance of navigation controller; note that for this test we need of an ATC flight plan.

	weight	***weather***	***input [feet,deg]***	***GPS***
test #1	**NORMAL**	**wind 340/00**	**attitude hold, CTS/GPS**	**ON**
test #2	**NORMAL**	**wind 000/16 + turbulences**	**altitude hold, CTS/GPS**	**ON**

Table 4.5: *navigation + longitudinal* controllers testing table

The starting point used is the same for these two tests, the aircraft is positioned in a space point outside the assigned route like figure 4.91 shows, the altitude that we want keep must be the altitude that *longitudinal* controller reads when it starts his running. The route track covered by the aircraft are represented in figure 4.92, we can see that the aircraft follows the pink line that represents the route assigned in the flight plan.

The graphs derives by tests collects these parameters: heading (current HDG and CTS), altitude, pitch angle, roll angle, vertical speed, aileron input and elevator trim input, these graphs will be discussed in the end of this paragraph.

Figure 4.114: GPS display at starting points of Test #1 and Test #2

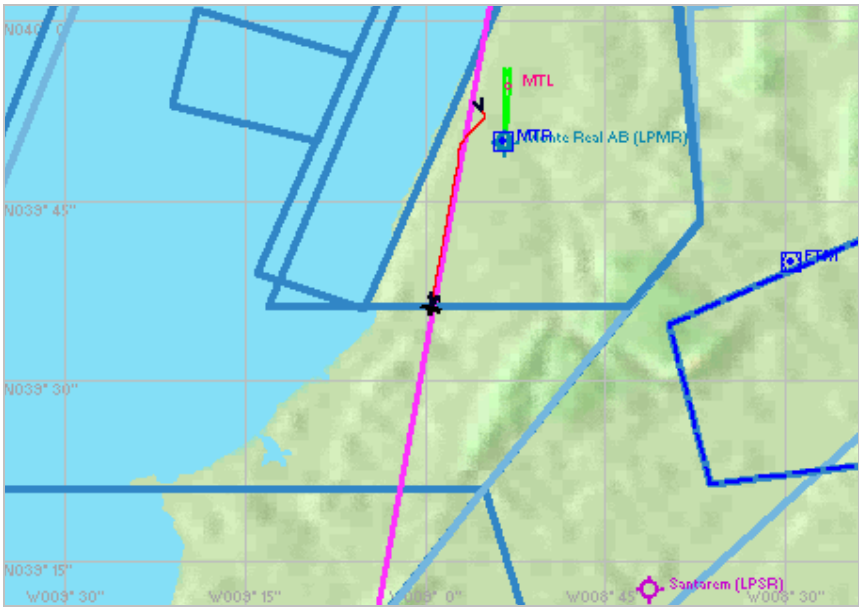


Figure 4.115: aircraft route track plotted in the end of tests

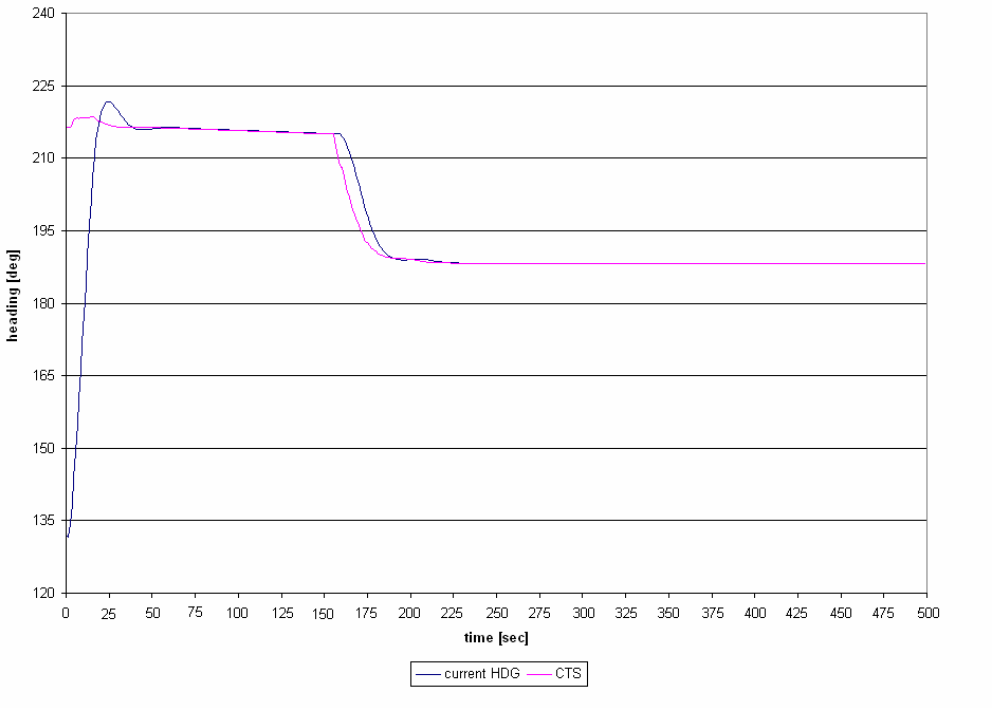


Figure 4.116: test #1 – heading response

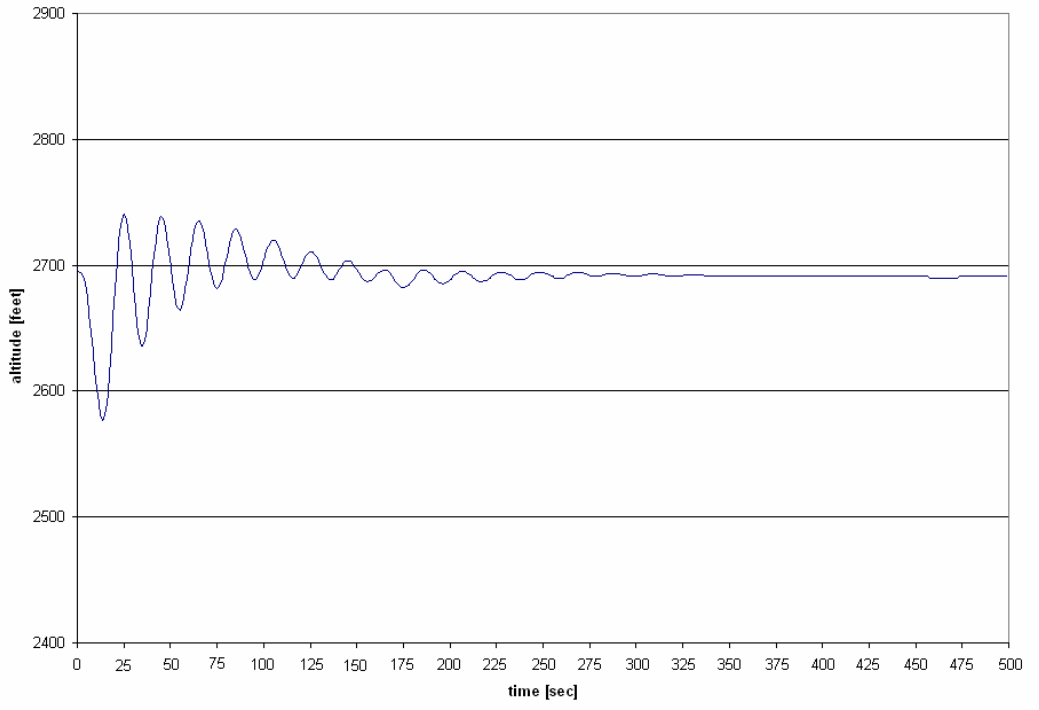


Figure 4.117: test #1 – altitude response

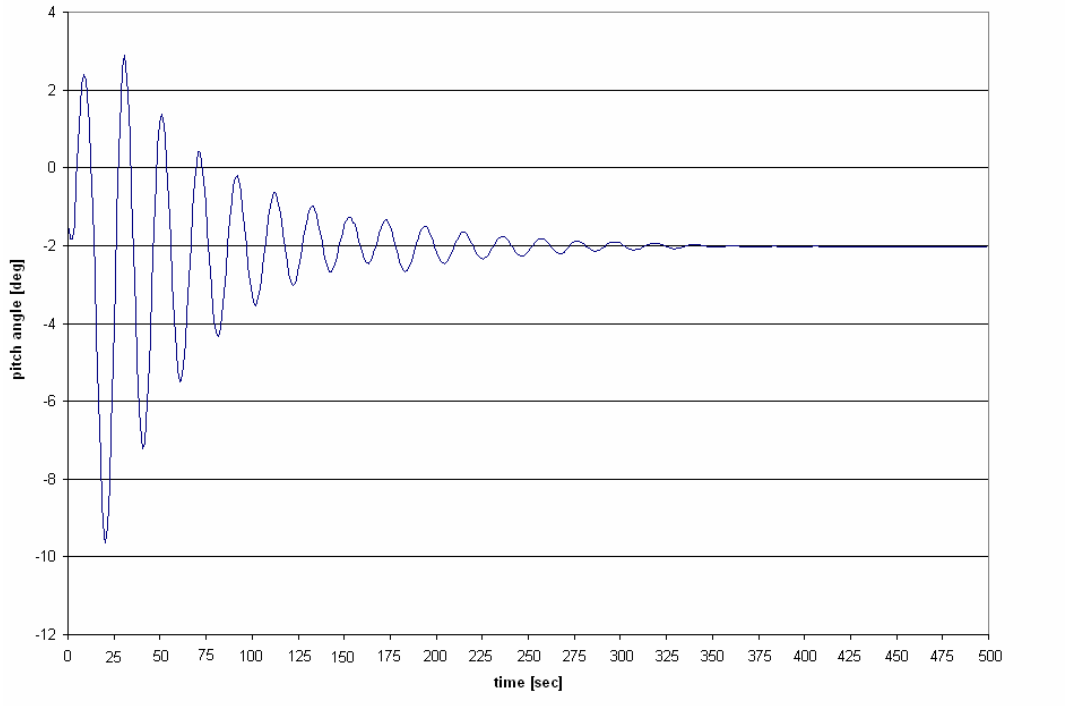


Figure 4.118: test #1 – pitch angle response

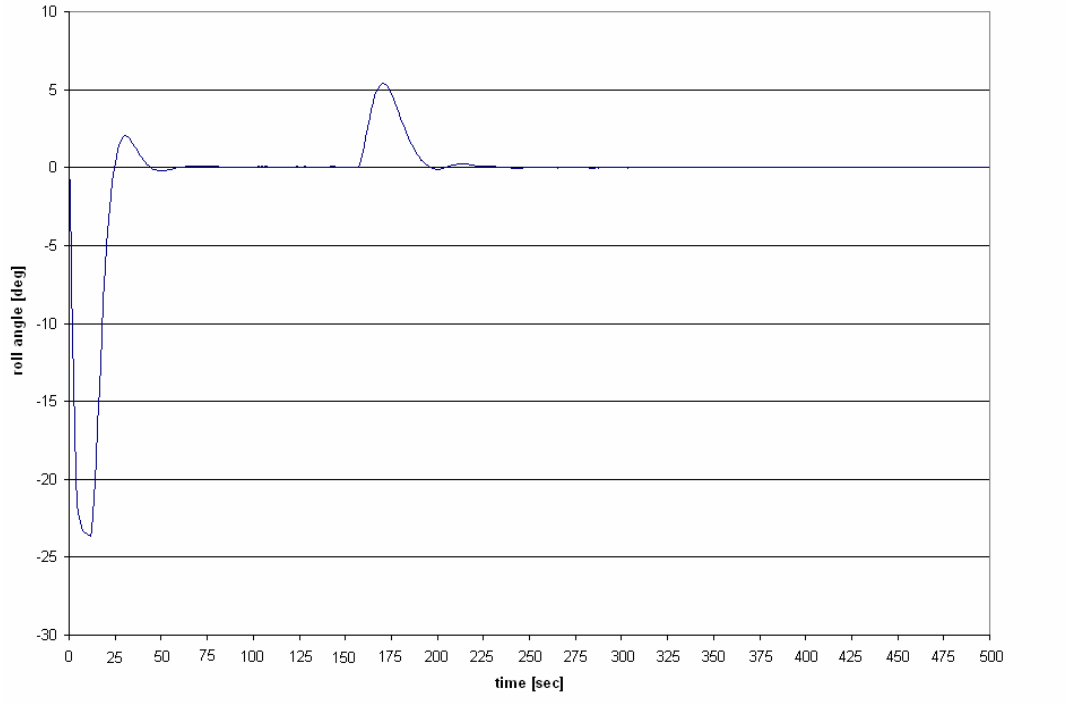


Figure 4.119: test #1 – roll angle response

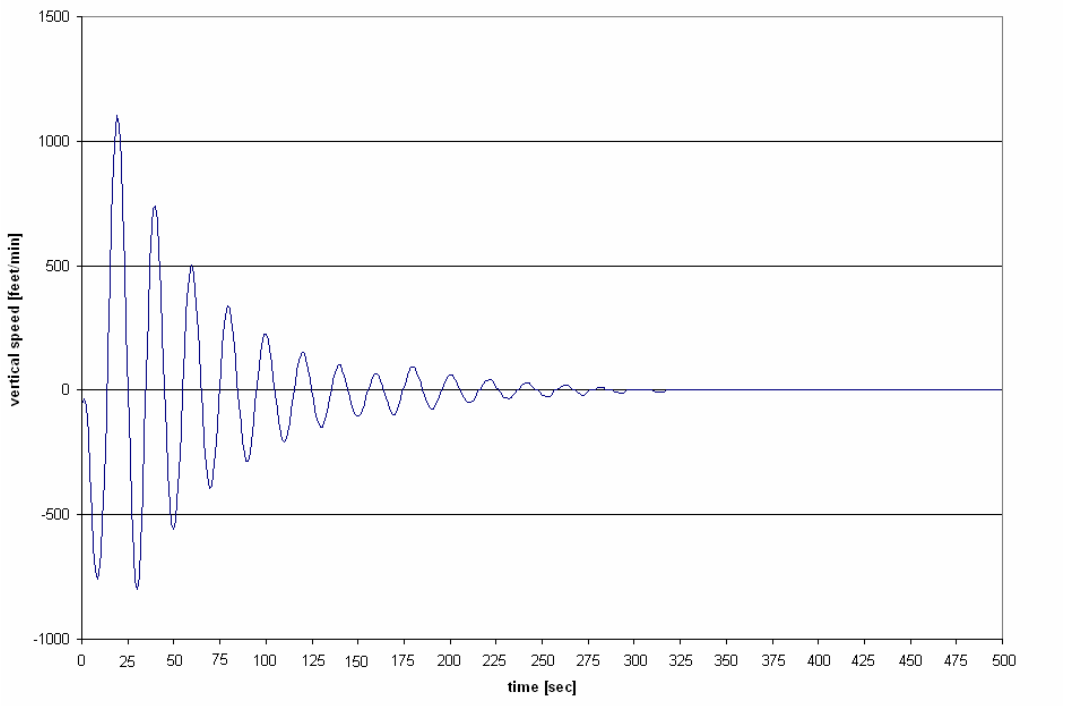


Figure 4.120: test #1 – vertical speed response

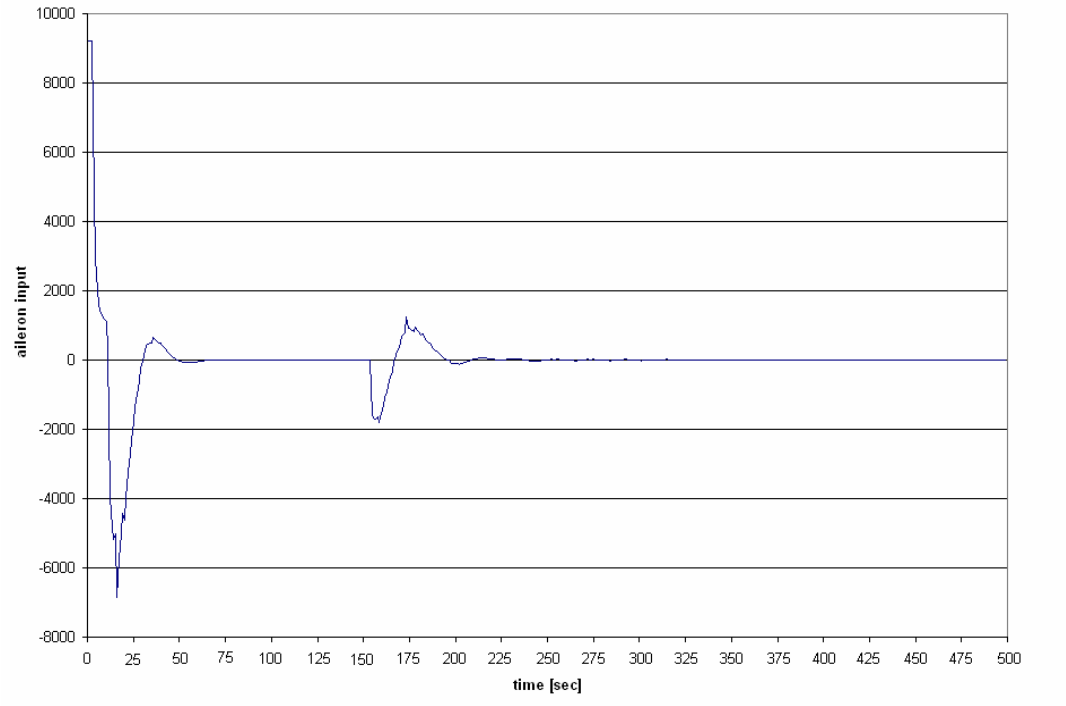


Figure 4.121: test #1 – aileron actuation

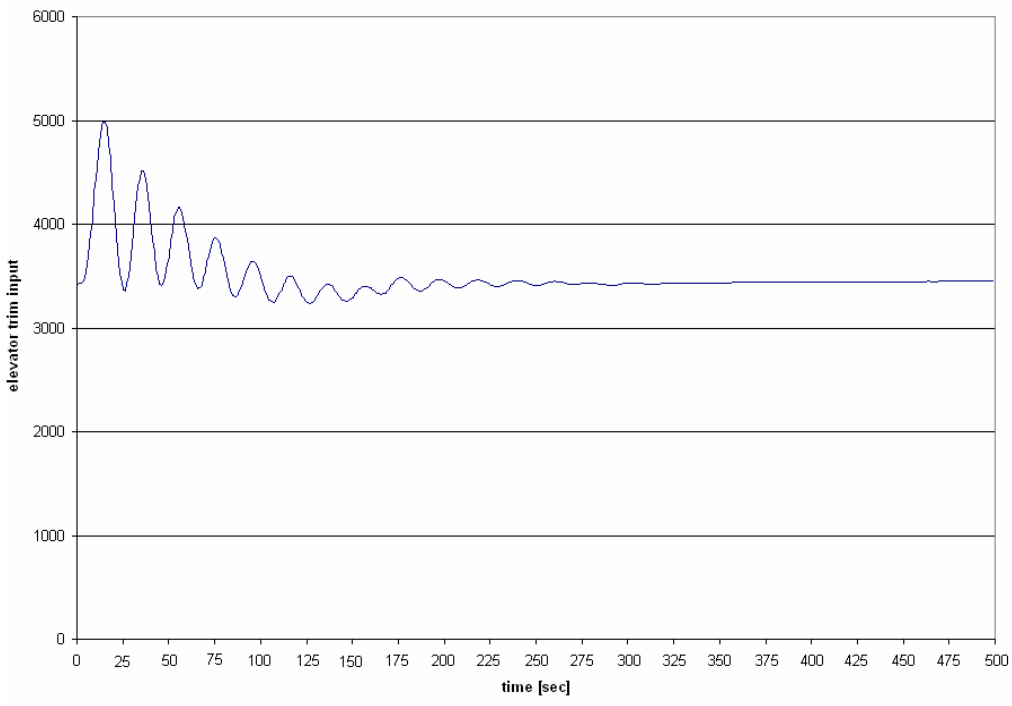


Figure 4.122: test #1 – elevator trim actuation

Graphs relatives to Test #2:

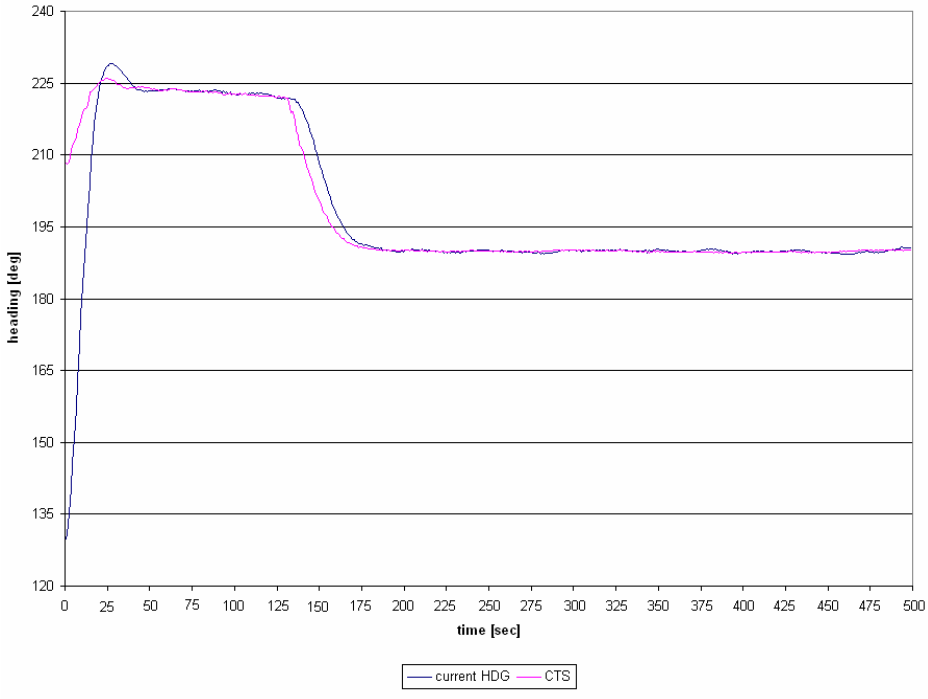


Figure 4.123: test #2 – heading response

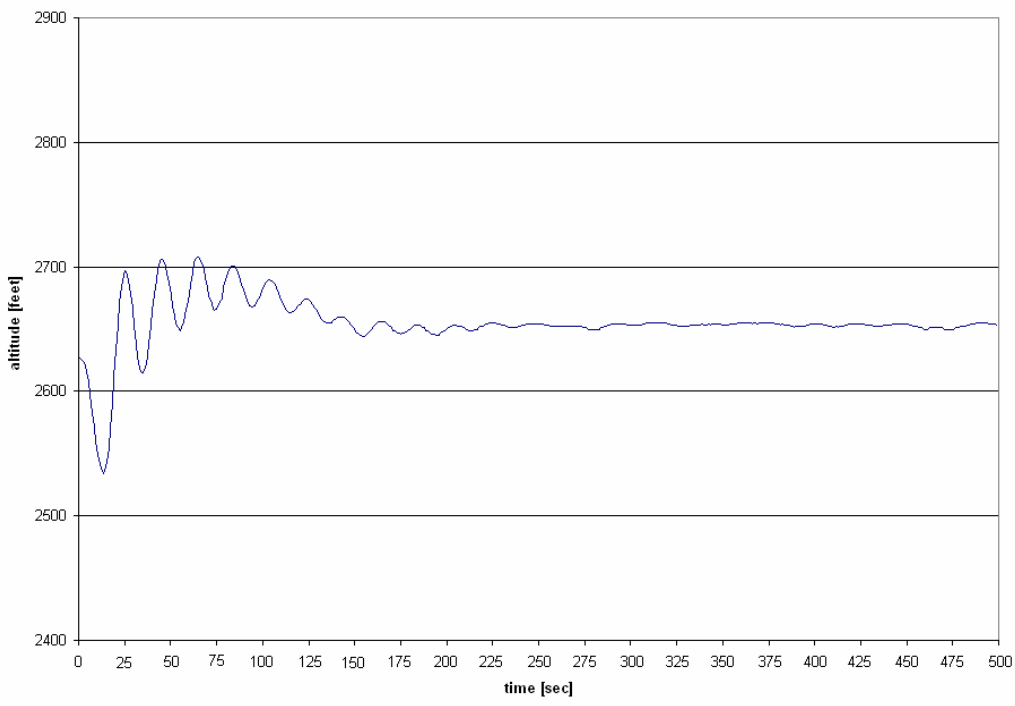


Figure 4.124: test #2 – altitude response

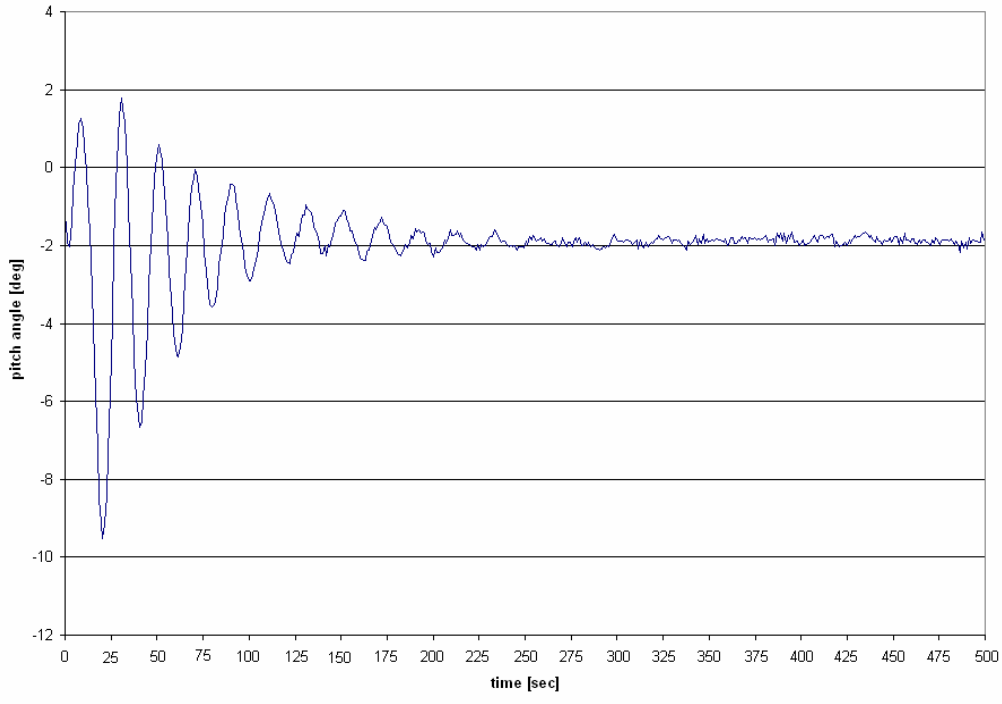


Figure 4.125: test #2 – pitch angle response

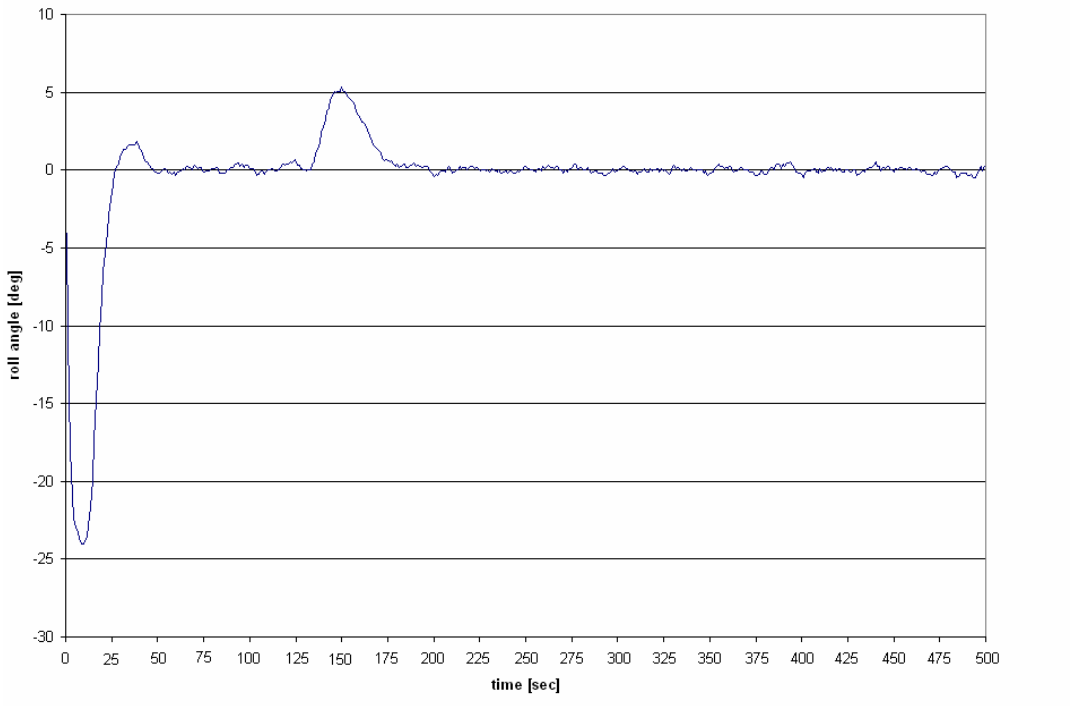


Figure 4.126: test #2 – roll angle response

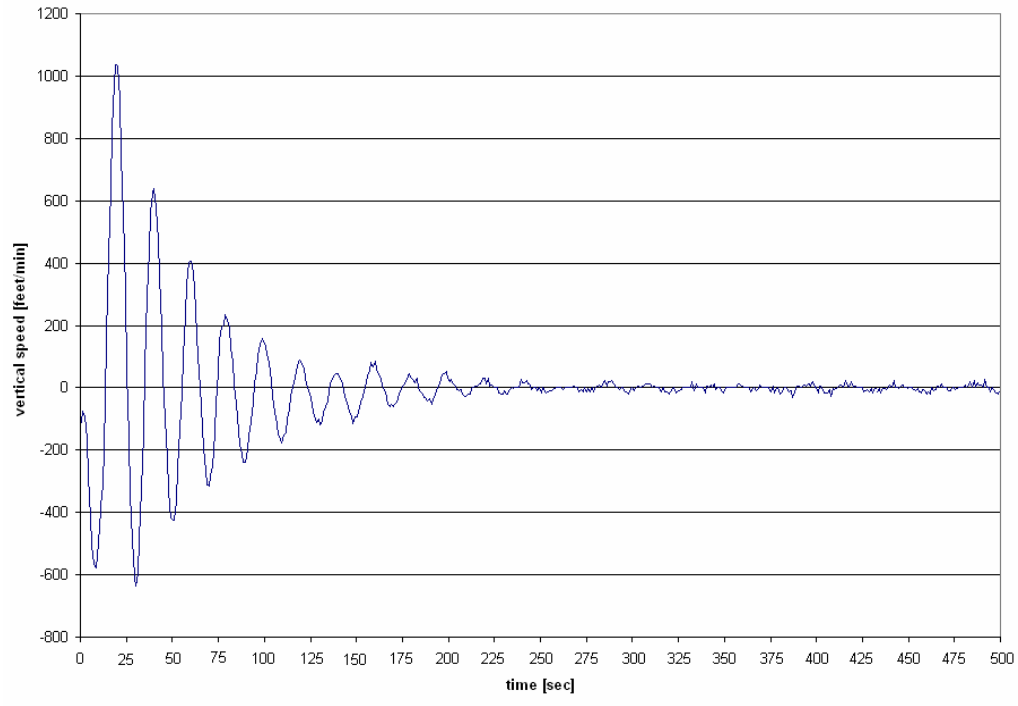


Figure 4.127: test #2 – vertical speed response

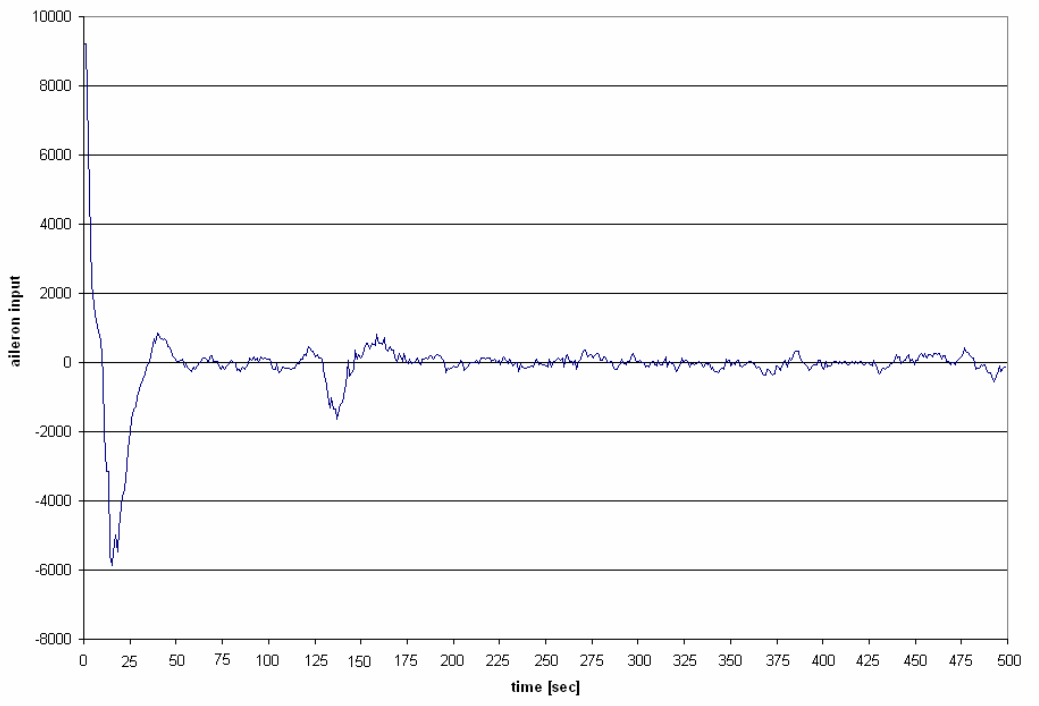


Figure 4.128: test #2 – aileron actuation

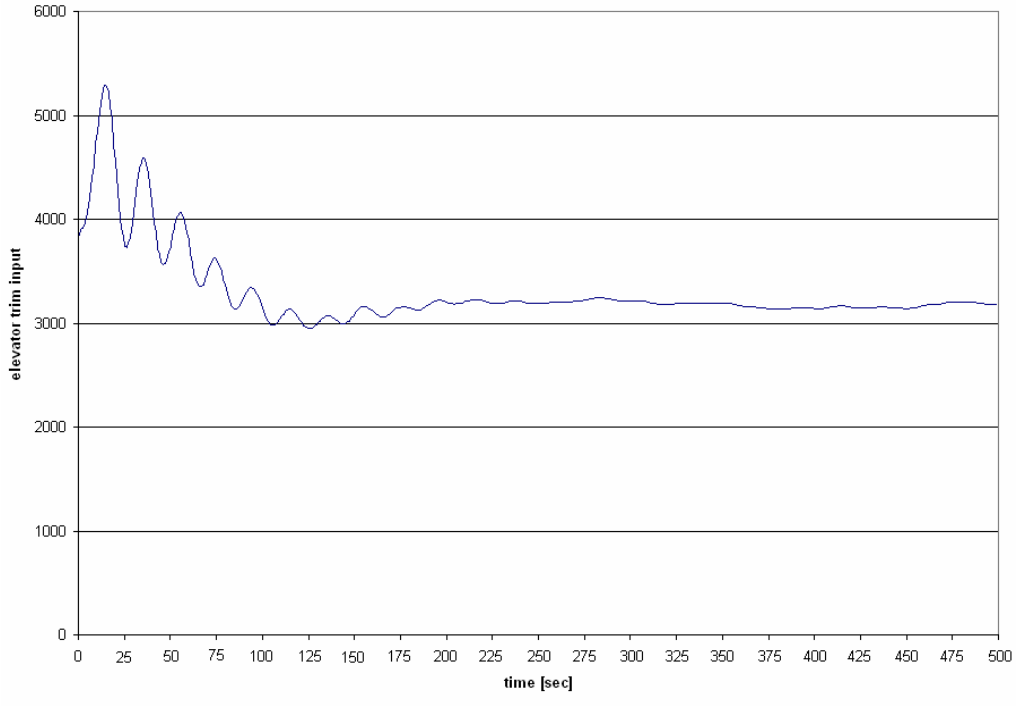


Figure 4.129: test #2 – elevator trim actuation

This verify has given results in coherency with lateral and longitudinal controllers testing because we have observed that wind produce some errors that controllers provides to correct. The tests are executed in the same flight condition (starting point, initial heading, initial altitude, initial air speed, initial vertical speed and initial flight surfaces status), only the wind constitutes the difference in terms of initial conditions.

Comparing figures 4.93 and 4.100 we can see that without wind the aircraft reaches the GPS reference signal and keeps it in a linear fashion while, in the other figure, the heading response is affected by perturbations due to wind. In Test #1, when aircraft is on-route, the heading error is next to 0 while in Test #2 this value has absolute fluctuation around the reference value of ±1° then, this controller works very good in adverse weather conditions also.

About roll angle responses, we can see that in Test #1, without wind, the airplane keeps itself balanced after has reached the desired value of heading and the roll angle error is 0 while adding wind, like done in Test #2, we have measured absolute errors included in the range [-0.20° – 0.30°], then the performances are considered good!

The property of altitude keeping are offered by longitudinal controller that, like showed in figures 4.94 and 4.101 relatives to Test #1 and Test #2 respectively, doesn't influenced a lot by the adverse weather condition because figure 4.101, developed by Test #2, doesn't present perturbations and the time needed to keep stable altitude is comparable to that of Test #1. The maximum altitude excursions measured in these tests are 163 ft in Test #1 and 174 f in Test #2.

Pitch angle performances of Test #1 and Test #2, like showed in figures 4.95 and 4.102 respectively, presents few differences only in terms of angle size, is possible to notice that in figure 4.95, after about 2'08" the pitch angle was stabilizing around a value equal to -2° and in figure 4.102 this value is the same but there are perturbation due to wind, in this case the aircraft reaches approximately this value in 2'18".

Vertical speed presents positive and negative peaks immediately after ailerons actuation like shows figures 4.97 and 4.104, their magnitudes are included in these ranges: [-800 ft/min – 1102 ft/min] in Test #1 and [-636 ft/min – 1033 ft/min]. After 46” and 47” respectively in Test #1 and Test #2, vertical speed keeps value included in the range imposed by design and after 1’27” and 1’25” respectively in Test #1 and Test #2 these values becomes sufficiently small.

The maximum peaks of aileron actuation needed by these tests are resumed in the ranges [-46% - 56%] and [-36% - 56%] in Test #1 and Test #2 respectively, these values are normalized to the maximum and the minimum aileron input values.

CONCLUSIONS

Aircraft autopilot development was described in this thesis work; this system was developed in the *Autonomous Systems Laboratory* (LSA) of the *Instituto Superior de Engenharia do Porto* (ISEP). The realization of this system was based in *Microsoft*® *Flight Simulator* simulation platform. Althought the airplane model depicted in flight simulator wasn't a completely correct one, namely there wasn't the possibility of side slip. The chosen platform has permitted to realize an efficient autopilot without a class-D flight simulation and with low development costs.

The first step has permitted to learn the aeronautic science and control theory basis needed to manage the project and to study in depth the problem of flight control surface and to research a technology that permits to automatically control the flight surfaces of a generic airplane. The choice of PID controllers has guaranteed an appropriated relation between inputs and controlled variables (output) and then good system stability. In the third chapter the autopilot system has take his shape, we have placed the specifications and then we have proceeded with the design in two steps: controllers design and implementation.

We have developed the design of the three controllers using a graphical approach based on the block scheme of the controllers that we have realized, then we have proceeded to design the "core" of each controller that is a PID regulator, this configuration needed *tuning* and *anti-windup* measures. We have used an experimental tuning method (Cohen-Coon) based on the reaction curve. There wasn't any information regarding the original controllers tuning. To solve the controller windup problem, we have used a saturator to limit integral components and to account for the physical actuators limitations.

Achieved the controllers we have proceeded to test these. The results proved that the developed achieves the desired position, in terms of heading, altitude and route maintenance. The collected data has evidenced that the three functionalities above mentioned, implemented in this autopilot system, had the expected results and that the lateral controller surpasses the one originally implemented by Microsoft.. The lateral controller as better accuracy, achieves the desired route or heading in a smallest time and performs better in roll axis .

The tests carried out on *longitudinal* controller has evidenced that this controller works like his analogous implemented in *Microsoft® Flight Simulator 2004* and that to obtain a good control of altitude or vertical speed is fundamental to implement a thrust controller.

APPENDIX A – *Cessna 172 "Skyhawk" data sheet*

A.1 History

In the late 1960s Cessna re-engined its already highly successful 172 four seater with the four cylinder Lycoming O-320. These O-320 powered models were the most successful to bear the 172 model number (and the Skyhawk name for the Deluxe option), as they were in production during GA's golden years, the 1970s.

Cessna re-engined the 172 with the Lycoming O-320-E as compared with the O-300 it had two less cylinders (and thus lower overhaul costs), a 200 hour greater TBO, improved fuel efficiency and more power. Even so, Cessna thought 172 production would be shortlived as the similarly powered but more modern 177 Cardinal was released at the same time. In spite of the Cardinal, the Lycoming powered 172 was a runaway success and easily outsold and outlived its intended replacement.

The first O-320 Skyhawk was the 172I introduced in 1968. The 1969 172K introduced a redesigned fin, reshaped rear windows and optional increased fuel capacity, while 1970's 172K sported conical camber wingtips and a wider track undercarriage. The 172L in production in the 1971/72 model years was the first to feature the enlarged dorsal fin fillet.

The 172M of 1973/76 gained a drooped wing leading edge for improved low speed handling. The 172M was also the first to introduce the optional `II' package of higher standard equipment. Also in 1976 Cessna stopped marketing the aircraft as the 172.

The 172N was powered by a 120kW (160hp) O-320-H designed to run on 100 octane fuel, but the engine proved troublesome and was replaced by the similarly rated O-320-D in the 172P of 1981. The P was the last basic 172 model, remaining in production until 1985.

Higher performance 172s include the R172 Hawk XP, powered by a 145kW (195hp) Continental IO-360 and the 135kW (180hp) Lycoming O-360 powered, retractable undercarriage 172RG Cutlass. The 172 was also produced under licence by Reims in France as the F172 and FR172.

A.2 Powerplants

172N - One 120kW (160hp) Lycoming O-320-H2AD flat four piston engine driving a two blade fixed pitch propeller. R172 Hawk XP - One 145kW (195hp) Continental IO-360-KB fuel injected flat six driving a two blade constant speed propeller.

A.3 Performance

172N - Max speed 232km/h (125kt), max cruising speed 226m/h (122kt). Initial rate of climb 770ft/min. Service ceiling 14,200ft. Max range with 45min reserves and standard fuel 1065km (575nm), with optional fuel 1390km (750nm).

R172 - Max speed 246km/h (133kt), max cruising speed 241km/h (130kt), long range cruising speed 177km/h (95kt). Initial rate of climb 870ft/min. Service ceiling 17,000ft. Max range with 45min reserves and standard fuel 1065km (575nm), with optional fuel 1510km (815nm).

A.4 Wheights

172N - Empty 649kg (1430lb), max takeoff 1043kg (2300lb).

R172 - Empty 710kg (1565lb), max takeoff 1157kg (2550lb).

A.5 Capacity

Typical seating for four in all models.

A.6 Dimensions

172N - Wing span 10.92m (35ft 10in), length 8.21m (26ft 11in), height 2.68 (8ft 10in).

Wing area 16.3m2 (175sq ft).

R172 - Span 10.92m (35ft 10in), length 8.28m (27ft 2in), height 2.68m (8ft 10in).

BIBLIOGRAFY

CHAPTER 1

[1] I. Morir, A. Seabridge, *Aircraft System*, Chapter 3, Longman Scientific & Technical, 1992

[2] M. Anthony, F. Mattos, *Advanced Flight Control Actuation Systems and their Interface with Digital Commands*, A-6 Symposium, 1985

[3] E. Pallett, *Aircraft Instruments and Integrated Systems*, Chapters 10, 12, 17, Longman Scientific & Technical, 1992

[4] D. Middleton, *Avionic Systems*, Chapter 5, Longman Scientific & Technical, 1989

CHAPTER 2

[5] P. Bolzern, R. Scattolini, N. Schiavoni, *Fondamenti di Controlli Automatici*, Chapters 1, 4, 14, 17, McGraw-Hill, 2004

[6] B. Kuo, *Automatic Control Systems*, Chapters 1, 6, 10, Prentice-Hall, 1995

[7] G. Magnani, *Tecnologie dei Sistemi di Controllo*, McGraw-Hill, 2000