BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

SPECIALIZATION COMPUTER SCIENCE IN ROMANIAN

DIPLOMA THESIS

IMPLEMENTATION OF AN AUTOMATIC FLIGHT CONTROL SYSTEM FOR LOCKHEED MARTIN FLIGHT SIMULATORS

Supervisor
Assoc. Prof. Rareș Florin BOIAN, PhD

Author
Tudor-Adrian GÎNGA

2020

Universitatea Babeş-Bolyai Cluj-Napoca
Facultatea de Matematică şi Informatică

Abstract

A study published by The Boeing Company lists 1183 fatalities attributed to loss of control in-flight from 2009 to 2018 in civilian aviation. Although study in this domain has been extensively made regarding how to lessen the probability of such an event occurring, little effort has been put into combining a high-level programming language with the benefits of an automatic flight control system to be installed for this purpose in a real aircraft. The objective of this paper is to provide the necessary theoretical fundamentals and an implementation of such a system for use on a Lockheed Martin Prepar3D platform. The software should build on the foundation of the C++ programming language, using the Qt GUI programming tools and the Prepar3D SDK for communication with the platform.

Based on an engineering concept used in the system control theory and on an automatic flight control system in use on some civilian transport aircraft, this thesis provides an effective software solution to the stated issue, which proves itself under the entire operating regime of an aircraft in line operations.

Acknowledgements

I would like to express my most sincere appreciation to everyone that has supported me, not only during the development of this thesis, but also during my entire academic journey.

Foremost, I cannot be grateful enough of the support I received from my beloved parents [redacted] and [redacted] during these three years, and especially during the Coronavirus disease pandemic which hit Europe in early 2020. These are the two people that have always stood by my side, no matter whether I did right or wrong, good or bad. Words cannot describe the amount of gratitude I am ought to offer, but only the hope that I have not disappointed.

I would also like to extend my wholehearted gratitude to the many people who have contributed during these years to my formation as an instructor. I was profoundly marked to find that although almost nobody had a personal interest in supporting me, everyone I asked for something did their very best to help, even at times when other, more pregnant tasks were at their hand. The workplace I had, and I still have after three years of work has given me an indescribable amount of self-confidence and satisfaction of helping as many people as I could. In no particular order I mention:

- [redacted], Airbus A320 Check Captain, CRM Standardization Instructor
- [redacted], Airbus A320 Captain
- [redacted], Airbus A320 Captain
- [redacted], Airbus A320 Captain
- [redacted], Airbus A320 First Officer
- [redacted], EASA ATPL license student

Not least I would like to thank the entire teacher collective of the Faculty of Mathematics and Computer Science under the Babeș-Bolyai University in Cluj-Napoca for my formation as a programmer, and especially [redacted] and [redacted] for their assistance in the development of this thesis.

The author

Limitation of Liability

Under no circumstance shall the author or any of the co-authors, affiliates or partners be held liable for any direct, indirect, incidental or consequential damage to property, injury or loss of life arising out of or in connection with the use of this software or thesis, whether or not the damages were foreseeable and whether or not the author was aware of the possibility of such damages.

Table of Contents

Table of Figures

Introduction

In a variety of domains, the development and deployment of advanced automatic control systems has led to an increase in both the efficiency and safety of operating complex machinery. This trend has aligned itself to the inherent human need of making his life easier, by decreasing both the workload and the stress levels associated with the operation of such machines.[1]

A notable field of interest regarding this aspect is aviation. Although the first confirmed manned flight dates back to October 15th, 1783, it is not until December the 17th, 1903 that a telegram from Mr. Wright, Orville to his father would announce the success of four flights on the Kill Devil Hills in Kitty Hawk, North Carolina. Little did Mr. Wright realize at the time that his telegram would stand proof for the first powered, heavier-than-air machine to have achieved sustained, controlled flight with a pilot aboard.[2][3]

Why is it then, one may ask, that it took more than 120 years since the first manned flight to achieve sustained and controlled flight, given that research on the field of aerodynamics had started as early as 1726? It rapidly becomes apparent that a continuous, reliable means of controlling an aircraft is no less important than the lift producing surface itself for the purpose of achieving flight for substantial periods of time.[4]

This issue of lacking a reliable means of controlling the aircraft, although mitigated as research and studies were undertaken, had led to a new problem: growing demands of the air transport industry meant that the control surfaces were becoming increasingly more difficult to operate, let alone with the precision needed to operate the aircraft in entire safety, due to their increasing size and weight (especially under aerodynamic loads).[5]

Another aspect was the maneuverability of the aircraft outside the normal operating envelope. Even today, with all the advancements, low-altitude low-speed excursions from normal flight parameters frequently require aggressive corrective action from the flight crew, due to the very limited amount of energy the aircraft possesses in that particular flight phase.[6]

Loss of control in-flight (LOC-I) is a major fatality factor in air transport. The Boeing Company (hereinafter referred to as Boeing) has published a study which lists 1183 fatalities attributed to LOC-I from 2009 to 2018. The second occurrence category by number of fatalities in the same time frame is Controlled Flight Into Terrain (CFIT), accounting for 568 fatalities. It should thus be very beneficial to consider developing means of reducing the occurrence rate of LOC-I incidents for the sake of improving air transport safety.[7]

As any advancement in a particular field, a solution to all of these issues would not be something that would spring overnight, but rather slowly evolving through the years as air transport requirements change. By research, it was determined that the best results were achieved by employing an electronic system for the control logic, combined with a power-boosted mechanism for the actuation of flight controls. Fly-by-wire would become the term that denotes the replacement of the conventional means of controlling an aircraft (i.e. linking yokes and levers to flight control surfaces by cables and pulleys) with electrical wires, which would connect a controller available to the flight crew to a processing unit, and then, further to control surface actuators.[5]

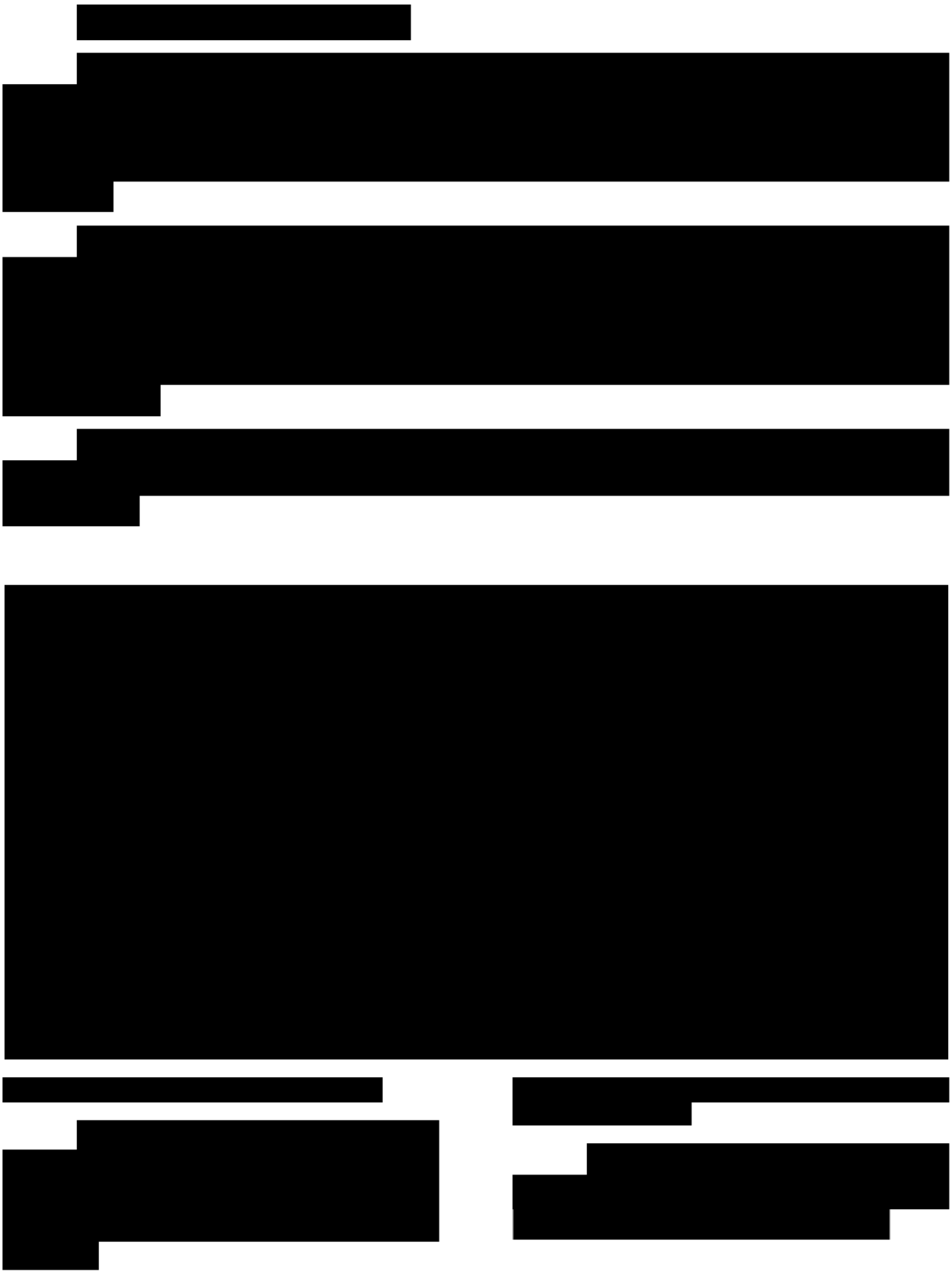
Besides relieving the flight crew of the aforementioned strength needed to operate the flight controls by use of actuators, the Fly-by-wire system offers yet another, even more significant feature: assisting the flight crew to maintain the operational envelope of the aircraft by use of augmentations and protections, enforced directly through the processing unit.[1]

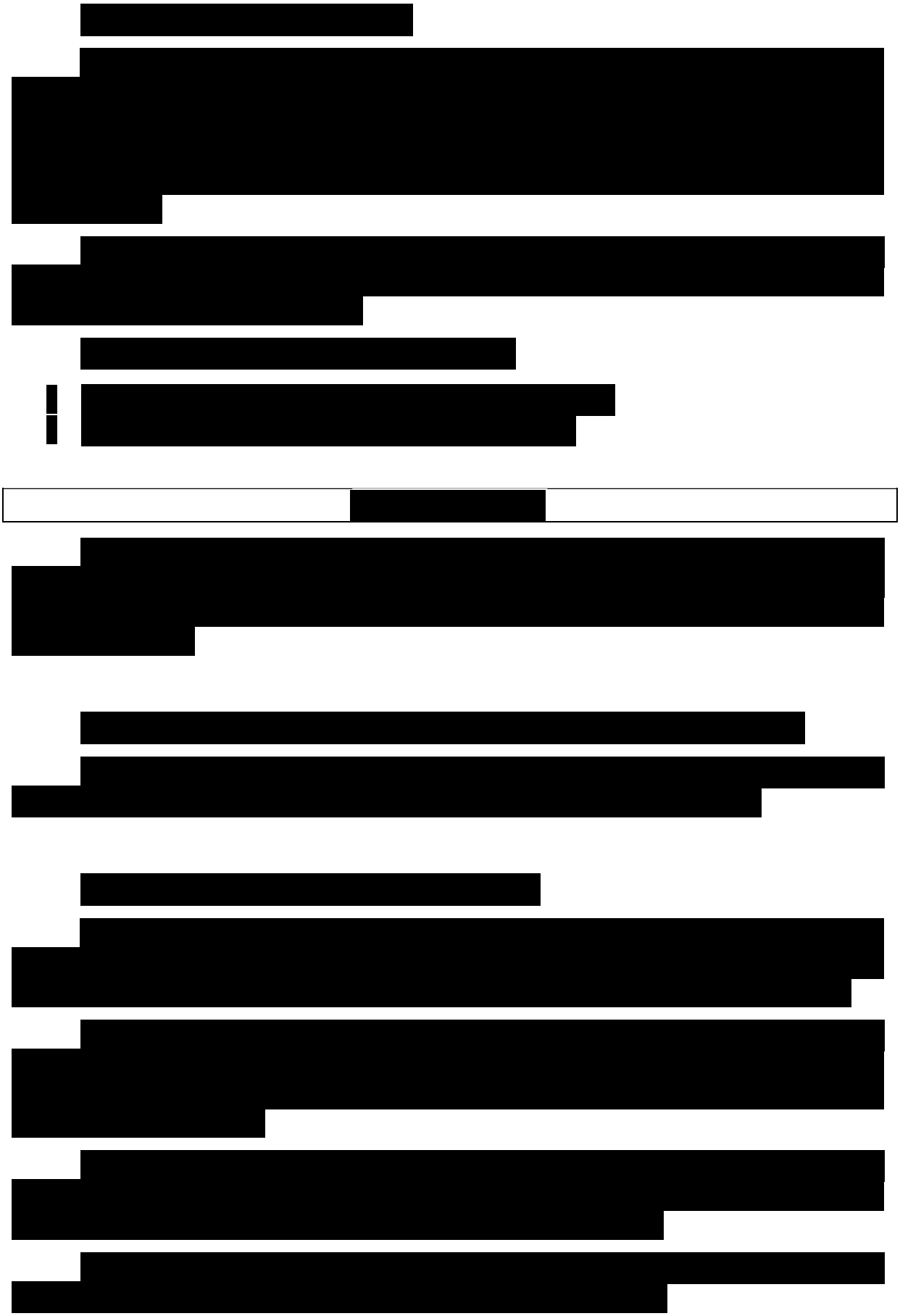
Although Fly-by-wire is being used extensively in modern commercial transport aircraft, little attention has been directed towards an open-source software implementation of such a flight control system using a modern programming language. This is partially due to the safety risk undertaken while altering the inner workings of such a critical system (see the Boeing 737 MAX crashes in the Java Sea, Indonesia and Bishoftu, Ethiopia).[8][9]

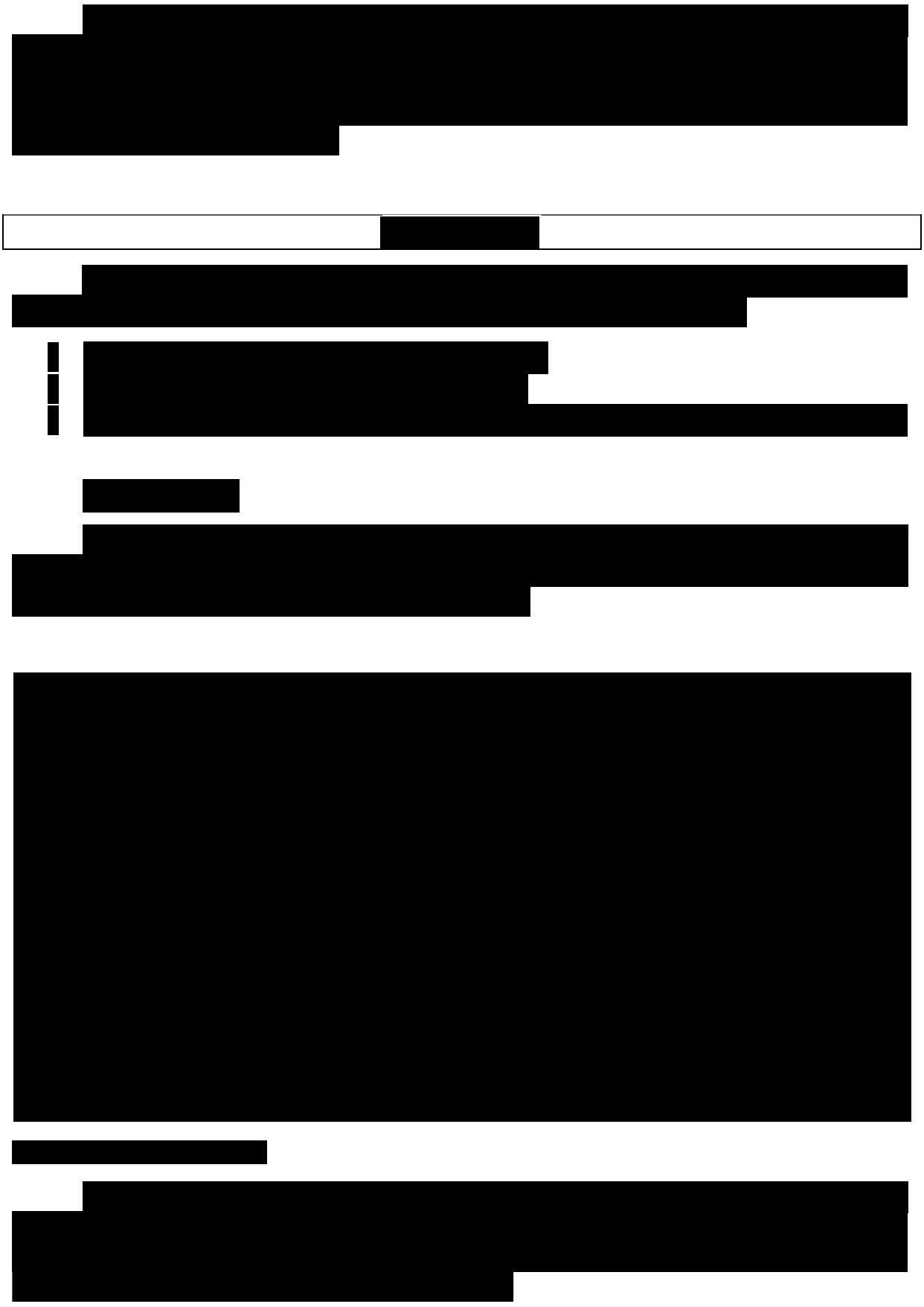
The purpose of this thesis is to describe the implementation of a Fly-by-Wire system using the Lockheed Martin Prepar3D v4 simulator and SDK. Additional, relevant aspects regarding existing technologies and theoretical aspects involved in the fields of physics, engineering and computer science will also be explored extensively.

Existing Technologies

Flight Control Laws in Airbus Aircraft







Comparison to the Author's Approach

Although the Airbus automatic flight control system has been the primary resource within the author's implementation, the purpose has been both to extend the functionalities and make them more effective, both in independent and simultaneous use. The implementation retains two flight control modes, as opposed to four on the Airbus, namely normal mode and direct mode.

Normal mode retains all five protections (with small improvements and/or modifications for each of them) available for use throughout the operational envelope. However, in the interest of safety, direct mode has also been implemented (disconnecting the automatic flight control system from the simulator) for use in case of a malfunction of the normal mode. A future extension of the software will contain a separate module which shifts automatically from normal to direct mode in case of fault, improper system response or unreliable data source (probably though use of AI).

The automatic flight module is also similar to the one implemented in Airbus aircraft, one notable difference being that the pitch and roll channels can be engaged separately, leaving, for example, the roll control at the pilot's discretion and the pitch control under the automatic flight control system's authority.

All the techniques used in modelling the software, as well as its mode of operation, architecture and flow of execution will be thoroughly examined in the following chapter.

Automatic Flight Control System

Theoretical Aspects

Flying Controls of an Aircraft

An aircraft can rotate along any one, or a combination of its three axes, all of which intersect into the aircraft's center of gravity and act at right angles to each other:[11]

- Movement along the lateral axis denotes a change in pitch[11]
- Movement along the longitudinal axis denotes a change in roll[11]
- Movement along the vertical axis denotes a change in yaw[11]

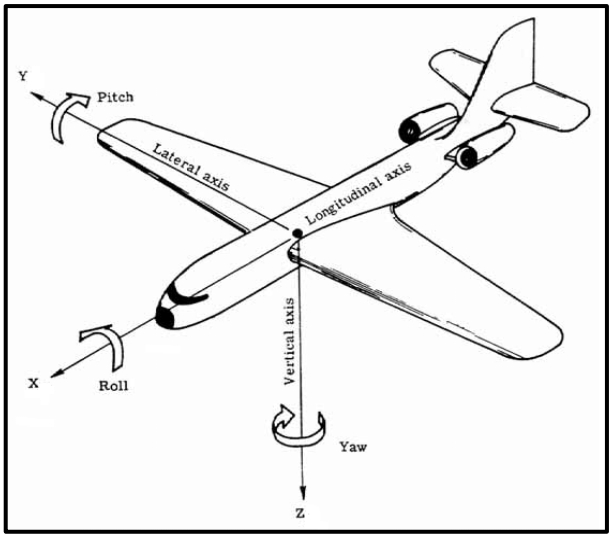


Figure 22: Aircraft rotation axes[12]

These movements are achieved by the primary flight control system, which (in its most rudimentary form) consists of moveable control surfaces linked by a series of cables and pulleys to controls in the cockpit. The primary flight controls are the elevator (for pitch control), ailerons (for roll control) and rudder (for direct yaw control and indirect roll control).[11]

PITCH CONTROL

The primary effect of the elevator (light blue) is to provide pitch control of the aircraft. Pushing the control column forward causes the elevator to move downwards, producing an upward resultant force which pitches the aircraft nose down. Pulling the yoke towards causes the elevator to move upwards, creating a downward force that pitches the aircraft up.[11]

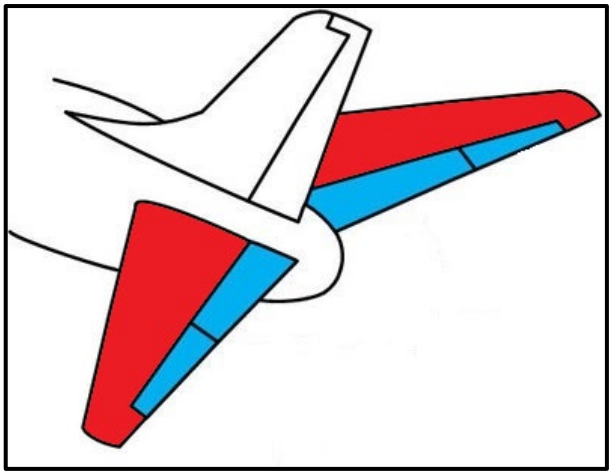


Figure 23: Stabilizator assembly[13]

The elevator is positioned on the aft section of the horizontal stabilizer (dark red), whose purpose is to alleviate the aerodynamic loads on the elevator (also by moving upwards and downwards). Ideally, at any given moment during flight (except when maneuvering) the pilot should exert as little force on the control column as possible in pitch. All the aerodynamic load should be taken by the trimmable horizontal stabilizer, which is actuated either by a control on the yoke, central console, or even automatically.[11]

ROLL CONTROL

The primary effect of the aileron (yellow) is to provide roll control of the aircraft. If the control column in moved either way the two ailerons will move in opposite directions, temporarily altering the shape (and aerodynamic characteristics) of the wing.[11]

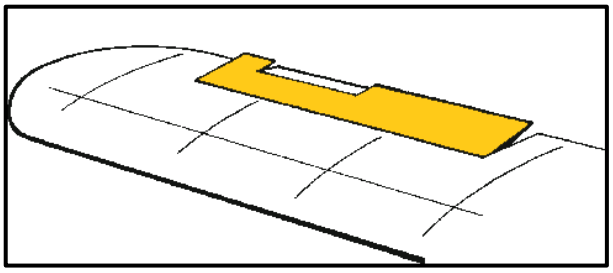


Figure 24: Aileron[14]

By principle, a downward movement of the aileron will cause an increase in angle of attack (and thus an increase in lift), while an upward movement of the aileron will act like an aerodynamic brake, reducing the angle of attack (and decreasing lift). The lift difference between the two wings produces the desired rolling moment.[11]

YAW CONTROL

The primary effect of the rudder is to provide yaw control of the aircraft. If the right rudder pedal is pushed, the rudder will displace toward starboard, and the aircraft will yaw right. The same principle applies when pushing the left rudder pedal, thus the aircraft will yaw left.[11]

The secondary effect of the rudder is to roll the aircraft in the direction of the yaw. This occurs because the outer wing travels faster through air than the inner wing, thus generating more lift.[11]

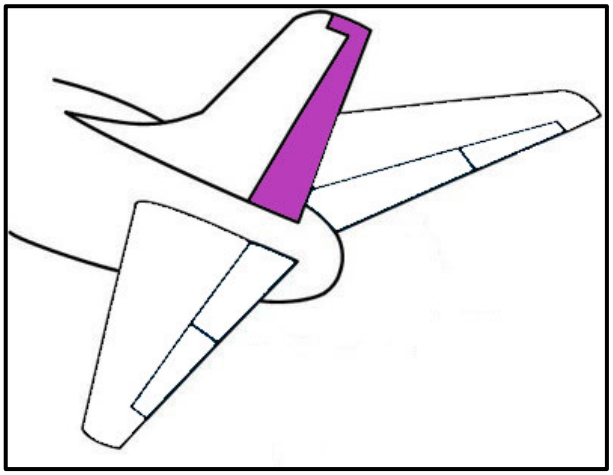


Figure 25: Rudder[13]

POWERED FLYING CONTROLS

Due to the size and heaviness of these surfaces (especially under aerodynamic loading), powered flying controls are being used in transport category aircraft. However, the same basic configuration resembling a primary flight control system is being used.[11]

The control surfaces are being controlled either mechanically or electronically, and hydraulically actuated (from the main hydraulic systems of the aircraft). Each hydraulic system may be pressurized (depending on aircraft system design) either by an engine-driven pump, an electrical pump, or in an emergency by a ram air turbine.[11]

No matter its design, any powered flight control system must comply with the following Joint Airworthiness Requirements:[11]

- Sense: the control surface must react in the direction indicated by the pilot.[11]
- Rigidity: the control system must withstand any aerodynamic loads without excessive or irreversible distortion.[11]
- Stability: the control surface must retain the position indicated by pilot regardless of external factors (vibration, aerodynamic loads etc.).[11]
- Sensitivity: the control surface must respond immediately to the pilot's input.[11]
- Safety: the control system must be guarded against any kind of interference from within the aircraft.[11]
- Fail-safe: the control system must be capable of continued operation in the event of hydraulic power failure.[11]

Lift and Stall in Fluid Mechanics

As an air mass moves around an airfoil, the difference of generated pressure between the inboard and outboard of the surface produces a force acting perpendicular to the relative airflow, known as lift. Of less importance is whether the airfoil moves through the air mass, or the airmass moves around the airfoil, as the following lift formula illustrates:[11]

$$L = \frac{1}{2}\rho V^2 S C_L \text{ where}$$

- L is lift[11]
- ρ is the density of the fluid (in this case, the air mass)[11]
- V is the relative velocity of the surface through the fluid[11]
- S is the lifting surface[11]
- C_L is the lift coefficient, dependent on the angle of attack[11]

TYPES OF AIRFLOW AROUND AN AIRFOIL

- Streamline flow: exists when succeeding air mass particles follow a steady, undisturbed path, thus flowing orderly along the surface of the object. At any given point along the flow of the fluid, all molecules have the same velocities and pressures as the preceding molecules.[11]
- Turbulent flow: if, however, there is a sudden change in the direction of the airflow, the molecules will no longer be able to follow the streamlined pattern and move unpredictably in regard to the preceding molecules.[11]
- Free stream airflow: exists far enough away from the lifting surface that is not disturbed by it.[11]

ANGLE OF ATTACK

The angle of attack is the angle formed by the incoming airflow vector (blue) and the wing chord line (imaginary line linking the leading edge to the trailing edge of the wing, red).[11]

Michael Paetzold, CC BY-SA 3.0

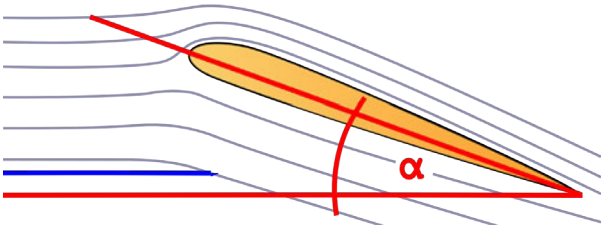


Figure 26: Angle of attack

If a symmetrical airfoil is placed in a steady stream airflow at zero angles of attack the airflow velocity will increase by the same amount, thus no differential pressure will exist, and the airfoil will not produce any lift. If, however, an airfoil is placed in an air stream at a positive angle of attack, the air will be accelerated more as it passes over the outboard surface (Venturi effect) and less over the inboard surface. This difference in speed generates a differential pressure which generates lift. Thus, it can be concluded that both the airfoil profile and the angle of attack have an influence on the lifting capability of the respective airfoil.[11]

In order to establish the relationship between angle of attack and the lift capability of an airfoil, a graph of lift coefficient against angle of attack can be plotted.[11]

It has been experimentally concluded that from 0 to around 10 to 12 degrees angle of attack (depending on the profile) the coefficient of lift is directly proportional to the angle of attack. Above this region, the increase in lift reduces, and the curve peaks at a certain value.[11]

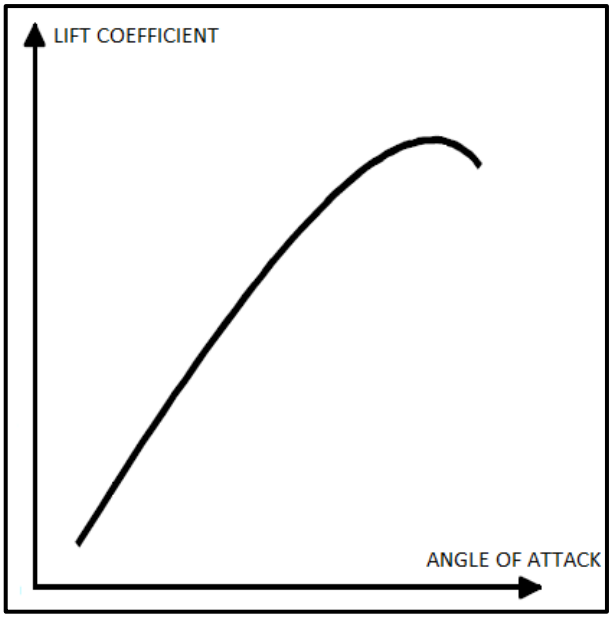


Figure 27: Lift coefficient vs. angle of attack

This point is called the critical angle of attack (x-axis), or the maximum lift coefficient (y-axis). As the name implies, two important aspects are worth noting here:[11]

- This is the angle of attack at which the wing develops the most amount of lift, given a certain fluid density, velocity of the surface through the fluid and lifting surface.[11]
- Any higher an angle of attack will determine a breakage of the smooth airflow over the lifting surface, destroying lift. Mathematically, the value of the lifting coefficent is undefined beyond the critical angle of attack. Experimentaly, it has been determined that the wing maintains (although greatly degraded) its lifting capabilites for a short period after exceeding the critical angle of attack.[11]

STALL

After reaching the critical angle of attack the smooth airflow over the outboard of the wing breaks, creating eddies (disturbed swirlings of fluid) which destroy the lifting capability. It is said that the airfoil is stalled when its upper surface is predominantly covered in separated airflow.[11]

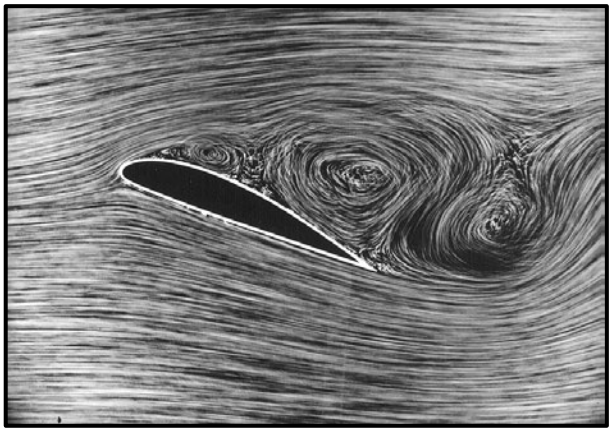


Figure 28: Stalled wing
DLR, CC-BY 3.0

Most stalls occur at low airspeeds and high angles of attack, causing turbulent flow to hit the empennage structure (horizontal and vertical stabilizers), creating surface trepidations (called buffet). This is known as pre-stall buffet, and occurs a few degrees (depending on the profile) before the actual stall, thus providing a good indication of it (it is usually felt through the control column and the rudder pedals).[11]

If the situation is not corrected rapidly, it may deterriorate up to the point at which instruments are extremely hard, even impossible to read (called deterrent buffet). From a recovery point of view, no difference shall be made between pre-stall buffet, deterrent buffet or the actual stall. Recovery consists in lowering the angle of attack below the critical value, usually by pushing the control column forward.[11]

Apart from these "natural" means of providing a stall warning to the flight crew, all aircraft are normally fitted with a device whose purpose is to signal the onset of a stall. It may come in the form of a visual warning, an aural warning, a tactile indication (stick shaker device) or a combination of these three.[11]

ACCELERATED STALL

An accelerated stall, by definition, is a stall that occurs at an overload factor of more than 1g. This may be the case while maneuvering the aircraft, such as when turning, or pulling out of a dive. In this case, the formula expressing the speed at which the aircraft will stall at a certain overload is as follows:[11]

$$V_{S\,acc} = V_{S\,1g} * \sqrt{\eta} \text{ where}$$

- $V_{S\,acc}$ is the accelerated stall speed[11]
- $V_{S\,1g}$ is the stall speed at no overload factor (1g)[11]
- η is the overload factor[11]

DEEP STALL

Conventional aircraft with straight wings and low-mounted stabilizers posess very good stall characteristics, assisting the pilot both in anticipating and recovering from the stall. Prior to the stall, turbulent airflow causes buffeting which alerts the flight crew, and even if the wing stalls completely, the forward center of gravity assists by lowering the nose of the aircraft, and thus the angle of attack (i.e. most often times the aircraft recovers by itself).[11]

This, however, is not the case when regarding aircraft with sweptback wings, a T-tail configuration of the stabilizer and rear-mounted engines, due to the following considerations:[11]

- The rear-mounted engines bring the center of gravity close to the center of pressure, thus making it harder for the crew to lower the nose of the aircraft for a forward-moving center of pressure (in an approaching stall condition).[11]
- The sweptback wing has a tendency to stall from the wingtips first, causing the center of pressure to rapidly move forward, thus reducing the available nose-down pitch moment.[11]
- The separated airflow does not pass over the stabilizer, resulting in no pre-stall buffet.[11]

And, most importantly:[11]

- The stabilizer becomes engulfed by turbulent air once the wing stalls entirely, thus elliminating all necessary authority to lower the angle of attack.[11]

Recovery from a deep stall condition is, theoretically, impossible. Advanced aircraft maneuvering techniques may be employed to attempt a recovery, such as removing power from an underwing mounted engine (creating a nose-down pitch effect), or moving the center of gravity forward (fuel transfer on capable aircraft, moving passengers or cargo). However, in the view of the author, the latter method is impracticable due to the amount of buffet present throughout the aircraft and the high stress levels associated with this upset.[11]

A more effective method of recovery from such an undesirable condition (which is also being mandated for the aircraft design mentioned earlier) is the fitment of a stick pusher device, whose purpose is to enforce a lowering of the angle of attack before it reaches the critical value, either by physically pushing the control column forward with a force that cannot easily be counteracted by the pilot (hence the name), or artificially, through the automatic flight control system.[11]

Operating Limitations of an Aircraft

Every airframe component is designed to withstand forces or stresses exerted on it while maneuvering the aircraft or under extreme weather conditions, by safely distributing them over the surface. It is thus natural for a certain airframe to posess a specific range of overloads ans speeds at which it can be safely operated. It is also worth noting that the aircraft, during its operational life, is constantly subject to various stresses, which may also occur due to cabin pressurization and depressurization, or vibrations.[11]

OVERLOAD LIMITATIONS

Each airframe has a very specific envelope within which it can be safely operated in accordance to Joint Airworthiness Requirements. This envelope is represented on a graph depicting velocity against overload factor (or a V-n diagram).[11]

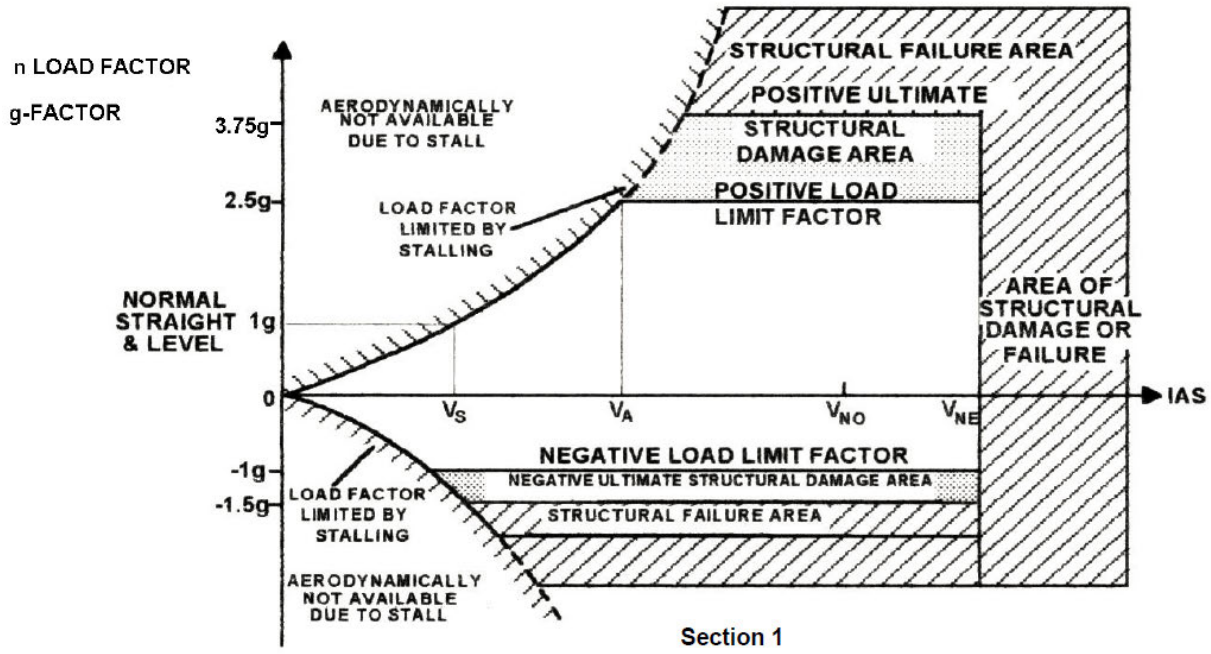


Figure 29: V-n diagram[11]

Although not illustrated in this graph, it must be noted that suddenly commanded full, or nearly full deflection against an aircraft moment may generate overloads that exceed the limit loads (and possibly the ultimate loads) and thus may result in structural failure. This statement is true even at speeds below V_A (maneuvering design speed).[11]

What this graph illustrates is the limit loads and design speeds, which if exceeded may result in structural failure. In transport category aircraft, these limits are around generally +2.5g and -1g. In view of safety a 50% safety margin toward ultimate limits is imposed, resulting in an ultimate load limit of +3.75g and -1.5g. However, it is not certain that by not overcoming the ultimate load structural damage will not occur if the limit load is crossed.[11]

SPEED LIMITATIONS

It is not necessary that an aircraft has the same V_{MAX} for the entire duration of the flight. Enumerated hereinafter are some examples that illustrate the further limitation of this speed:[11]

- V_{FE} is the maximum speed at which the aircraft may be flown with flaps extended. Although structural failure of the airframe will not occur at V_{FE}, the flap assembly may suffer permanent damage.[11]
- V_{LO} is the maximum speed at which the aircraft may be flown with the landing gear in transit. The problematic aspect here is that landing gear doors have limited resistance to high aerodynamic loads.[11]
- V_{LE} is the maximum speed at which the aircraft may be flown with the landing gear down.[11]
- V_{MO} or M_{MO} is the maximum speed/Mach number at which the aircraft may be flown with the landing gear up and flaps retracted. It is dependent on altitude.[11]

Control Theory

Many problems in current times are fortunate enough to have their solution be something that does not require much analysis, or put in other words, does not allow for much wandering before a solution is immediately apparent. An overly exaggerated example can be found below:[15]

$$\text{Find } x \text{ for which } x + 5 = 7.$$

Indeed, a solution for this problem is not hard to think of:

$$x = 7 - 5, \text{ thus } x = 2$$

Unfortunately, not all problems can be characterized as such, and thus the reader is asked to reflect for one minute upon the following problem:

$$\textit{A pot containing water at an unknown temperature is placed on the gas cooker.}$$
$$\textit{The stove is turned on.}$$
$$\textit{Indicate the moment the gas has to be turned off in order for the water to boil for the minimum possible amount of time.}$$

Control theory is applicable for everyday situations, such as in the examples above, as well as for controlling complex machinery, such as aircraft or nuclear powerplants. In fact, the field in which the theory is applied is of little importance since the concept is application independent. The argument for this matter is that if we have the capability of controlling a highly general system, we should be able to control any of its subassemblies with the same easiness.[15]

At this moment, the purpose of the control theory starts to reveal itself: to continuously operate complex systems (processes, machines etc.) in a manner which would lead to the desired state of the system in the minimum time possible, while also ensuring the stability of the system. For these purpose, certain aspects must be available beforehand:[15]

- A non-ambiguous final/future intermediary state of the system.[15]
- A subset of possible alterations to the current state of the system which can be made in order to move towards the desired state.[15]
- Some means of correctly choosing the best actions to be made that will efficiently lead to the desired state.[15]

The primary concept in the control theory is the control loop. It denotes a closed system in which a decision is continuously made considering information on the result of actions earlier undertaken. A control loop is error-driven, the error being defined as the absolute difference between the desired and actual output. The most often used metric for evaluating the performance of a control loop relates to the reduction of the rate of error.[15]

In its simplest form, an automatic control system is specified so that a certain system response should continuously and accurately chase a desired system response that is a requisite from the user. This response may come in many forms: it may be constant, it may by null, or may be equal to the last inputted value by the user. What is certain, however, is that this value must be tracked accurately regardless of external influences.[15]

The answer to this issue is designing a controller that issues commands physically connected to the process that has to be modeled. In order to model the controller as error-driven, the controller must further receive information on the difference between the expected and obtained output.[15]

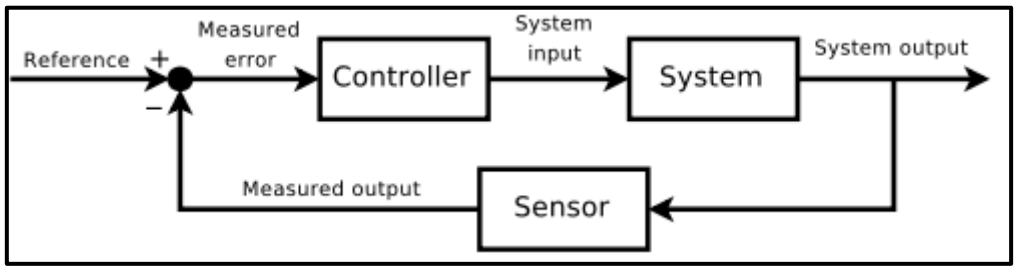


Figure 30: Generic PID Controller schematic

Orzetto, CC BY-SA 4.0

Experimentally obtained in these types of systems are excellent results, even despite using relatively simple controllers, or where the controlled process is not very well understood. In order to simplify, one could say that the controller has to take corrective action until the error is reduced to zero. Or, put in other words:[15]

When the water boils.

Proportional-Integrate-Derivative Controllers

By far one of the most common control algorithms is the PID controller. It, eventually by suffering minor variations, can control most feedback loops. In itself, the concept of feedback is extremely simple, and yet, it has a profound influence on technology. In essence, the concept can be summarized as follows:[16]

> *Increase the system output when the process variable is below the desired value and decrease the system output when the process variable is above the desired value.*[16]

This type of controller is said to possess *negative feedback*, since the process value moves in opposite direction with regard to the difference to the desired value. The reason why this type of controller is of interest is because it can achieve a desired system state despite system noise, disturbances, or divergence from ideal models.[16]

ON-OFF CONTROL

A simple on-off feedback mechanism can be mathematically described as follows:[16]

$$response = \begin{cases} -response_{max} \; if \; error > 0 \\ response_{max} \; if \; error < 0 \end{cases}$$

where $error$ is the difference between the desired variable value and the actual variable value.[16]

Notice that the mathematical model does not define the behavior of the system when the error is zero. This is since in real life operations very seldom does the system achieve the ideal state, and even if it does, it happens for a short period of time. Thus, it is common to introduce some modifications to the mathematical model, such as a dead zone, in order to illustrate the behavior of the system when closing in to zero error.[16]

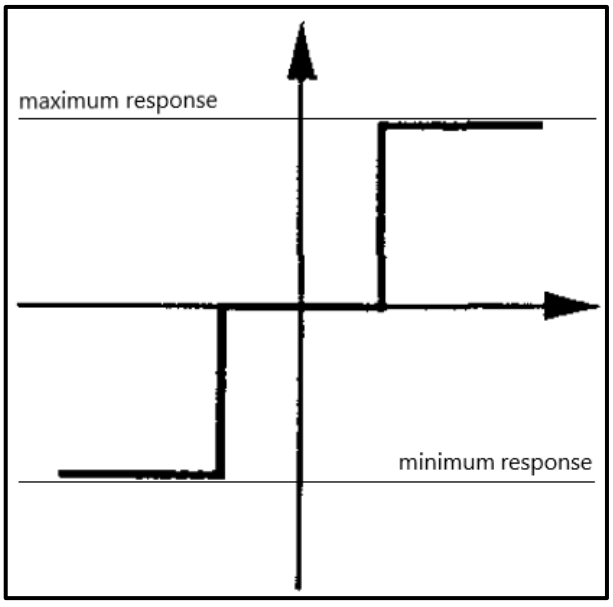


Figure 31: On-off feedback mechanism response[16]

An on-off controller often succeeds in maintaining the process variable close to the desired value but will also produce oscillations. This is because the controller reacts with the same “aggressiveness” to a large error, as well as to a small error.[16]

PROPORTIONAL CONTROL

The effect of overcorrection regarding an on-off controller can be avoided by deploying a proportional controller. It achieves this by limiting the amount of correction that is applied as the process variable closes in to the desired value.[16]

In order to mathematically describe the characteristics of a proportional controller, we must know beforehand the maximum response limits of the control variable. The linear range can be characterized either by providing the slope of the characteristic (controller gain K), or by providing the range on the x-axis where the characteristic is linear (proportional band P_b).[16]

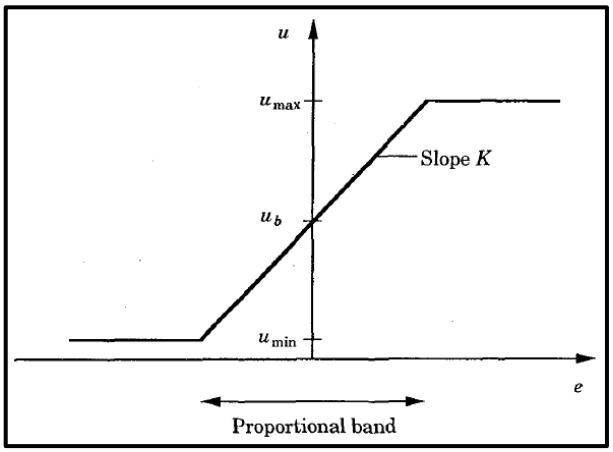


Figure 32: Proportional controller response[16]

One aspect that is to be noticed here is that a proportional controller acts like an on-off controller for large errors.[16]

PID CONTROL

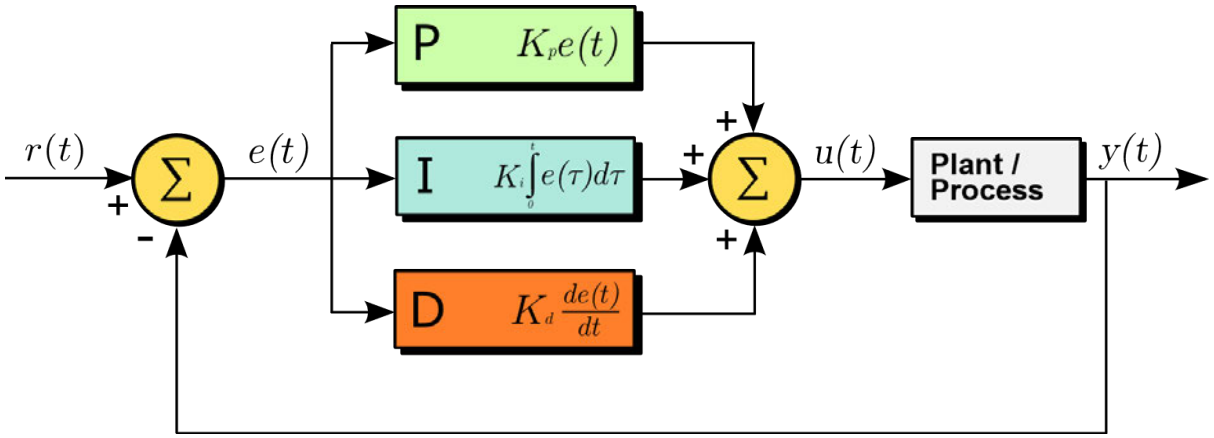
Given the susceptibility of the proportional controller to steady-state error, it is impracticable to use it in production. However, it has been experimentally found that a PID controller would solve this issue. It can be described as follows:[16]

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\,d\tau + T_d\frac{de(t)}{dt}\right), \text{ where}$$

- u is the system variable as a function of time t[16]
- K is the proportional gain[16]
- e is the control error as a function of time t[16]
- T_i is the integral step time[16]
- T_d is the derivative step time[16]

Figure 33: Mathematical PID Controller schematic

Arturo Urquizo, CC BY-SA 3.0



PROPORTIONAL TERM

If the PID controller would to be reduced solely to a proportional controller, then the law of Equation would reduce to:[16]

$$u(t) = K * e(t) + u_b$$

In other words, the control function is proportional to the control error. The variable u_b regulates the control output so that it is stationary at a certain value (depending on system specifics) when the error is zero.[16]

INTEGRAL TERM

The main purpose of the integral term can be summarized as to make sure that the process output matches with the target value when the system enters a steady state (process variables not changing over time). With a proportional controller only this would not be achievable since any deviation from the desired target value would produce a response which increases in amplitude, no matter how small the error is. Following is demonstration that by using an integral action, the steady-state error is zero.[16]

> *Allow us to assume by indirect proof that the system is in steady-state with a constant control input (u_0) and a constant error (e_0). By the Equation, the control signal is given by*[16]
>
> $$u_0 = K\left(e_0 + \frac{e_0}{T_i} * t\right)$$
>
> *Given that the constant error is not zero, a constant control input (u_0) would not be achievable since the error is a function of timestep (which is variable in time). It is thus proven that a proportional controller, when introduced with an integral term, will provide a steady-state error of zero.*[16]

The effect of the integral term can be described as automatically resetting the bias term u_b of its associated proportional controller. In fact, this was one of the earliest models of a proportional-integrate controller, or a proportional controller with “automatic reset” as it was also called.[16]

DERIVATIVE TERM

The purpose of the derivative term is to improve the stability of the closed-loop system. Due to the process dynamics, a change in the system variable will not be immediately apparent in the output, and so the control system will inherently expose a delay when correcting for an error. The control will thus be made proportional to the predicted process output, where the prediction is made by extrapolating the error by the tangent to the error curve. The basic structure of a proportional-derivate controller is:[16]

$$u(t) = K\left(e(t) + T_d \frac{de(t)}{dt}\right),$$

which by Taylor expansion becomes[16]

$$u(t) = K * e(t + T_d)$$

The control output is thus a function of the control error at time T_d ahead, where the estimate of T_d is obtained by linear extrapolation.[16]

CONCLUSION

The PID controller has three terms:[16]

- The proportional term P is proportional to the control error[16]
- The integral term I gives a proportional action to the time integral of the error[16]
- The derivative term D is proportional to the time derivate of the control error[16]

Finally, a pseudocode which implements a PID controller is given below:

```
OLD_ERROR := 0
INTEGRAL := 0

loop:
    ERROR := DESIRED_VALUE - ACTUAL_VALUE
    INTEGRAL := INTEGRAL + ERROR * dT
    DERIVATIVE := (ERROR - OLD_ERROR) / dT
    OUTPUT :=
        PROPORTIONAL_COEFFICIENT * ERROR +
        INTEGRAL_COEFFICIENT * INTEGRAL +
        DERIVATIVE_COEFFICIENT * DERIVATIVE
    OLD_ERROR := ERROR
    sleep(dT)
    goto(loop)
```

In today's networked world, it comes as granted that resources can be shared among multiple users. Thus, hardware and software infrastructures are designed to be able to process all user requests simultaneously. Even more, with the availability of multi-core processors, it is increasingly common to see mobile computing devices that can run multiple tasks concurrently. This is known as multitasking.[17]

In order to define multitasking, the concepts of process and thread must be explained. A process is the memory space of a program in execution. It may be divided into several independent workers, known as threads. Thus, a process is a reunion of a certain number of threads.[17]

The advantage of implementing a program to run in multiple threads becomes obvious when, for example, the task of implementing a GUI paired with a complex logic of a software comes at hand. It should be very beneficial to be able to maintain the GUI responsive by "delegating" resource-intensive computation/tasks to other threads.

The Qt framework offers, among others, a means of developing threaded application which maintains its responsiveness even under heavy resource loading, namely the *QThread* class. It manages the execution of one thread within the application.[18]

Each *QThread* offers the possibility of implementing a loop that runs continuously, until a certain condition is met. This is achieved by overriding the *run()* method. Even more important, it offers a way of connecting two running threads – method calls, for instance – outside the main execution loop. This is called the Signals and Slots mechanism.[18]

SIGNALS AND SLOTS

In Qt, the Signals and Slots mechanism is used to communicate between objects. It comes particularly in hand when operating with multiple threads that have to communicate with each other (for example, in GUI programming, an interface element should change when the system logic dictates it).[19]

Other toolkits achieve this by the use of callbacks (function pointers that can be used to notify other threads at appropriate times). However, this kind of system is unintuitive and is prone to errors from part of the programmer – for example, regarding the correctness of callback arguments.[19]

In Qt, the callback technique is replaced by the use of Signals and Slots. A signal is emitted when a particular event happens, while the slot is a function that is called in response to a signal. The type safety of the Signals and Slots mechanism is ensured: the signature of a signal must match its corresponding slot.[19]

Moreover, this mechanism has other advantages: the slots are regular member functions, so an object does not need to know whether its slots are connected to any signals and vice versa. Thus, any number of signals can be connected to a single slot, and any number of slots can be connected to a single signal.[19]

When a signal is emitted, the corresponding slot (if connected) is executed immediately and independent of any internal loop that the thread may be executing. Thus, it is possible to call slots that update the GUI while the system logic is still running.[19]

An example of Signals and Slots usage can be found below:

```
class A : public QObject
{
    Q_OBJECT

public:
    int value = 0;
    void SetValue(int NEW_VALUE);

signals:
    void ValueChanged(int NEW_VAL);
}

void A::SetValue(int NEW_VAL)
{
    value = NEW_VAL;
    emit ValueChanged(NEW_VAL);
}
```

```
class B : public QObject

{
    Q_OBJECT

public:
    int value = 0;

slots:
    void ValueChanged(int NEW_VAL);
}

void B::ValueChanged(int NEW_VAL)
{
    value = NEW_VALUE;
    std::cout << "B has changed valu
e! It is now " << value << '\n';
}
```

```
int main()
{
    A a; B b;
    QObject::connect(&a, &A::ValueChanged, &b, &B::ValueChanged);
    a.SetValue(10); // a.value == 10, b.value == 10
    // B has changed value! It is now 10
    return 0;
}
```

Lockheed Martin Prepar3D v4 SDK. SimConnect API

Prepar3D is a visual simulation platform used to create training scenarios in the fields of aviation, maritime and ground military. It is used for private pilot, commercial, organizations, academia and military training.[20]

The purpose of the Prepar3D product is to be used for a wide range of learning scenarios, such as procedures training, cockpit familiarization, flight planning, air traffic controller training and emergency response preparation.[20]

The Prepar3D SDK is used to develop new content for the simulator, such as aircraft, instruments, vehicles and other structures. It can also be used to create visuals, such as new scenery, terrain and special effects.[21]

The SimConnect API is designed to interface the programmer to Prepar3D. Some of the more notable achievements that can be obtained with SimConnect are implementing a complex gauge or instrument, replacing default Prepar3D events with proprietary logic, or even intercepting key events in order to provide a new response from the simulator.[21]

SimConnect works on a client-server design pattern to enable communication between a client written by the programmer and the Prepar3D simulator. Most often, the client first requests a connection to the simulator be made, then subscribes to a certain subset of events, variables etc. to be passed to it. Depending on the purpose of the client, the alteration of the desired parameters can begin, which will then be passed back to the simulator.[21]

DATA DEFINITIONS. SIMULATION VARIABLES. REQUESTS

A data definition is a structure used to hold data coming from the simulator. It may contain variables of different types, the only restriction being that the representation of any simulation variable inside the simulator must be compatible with its correspondent in the client. Following is an example of a data definition:

```
struct TYPE_FLIGHT_PARAMETERS
{
    double PLANE_PITCH_DEGREES;
    double PLANE_BANK_DEGREES;
    double PLANE_HEADING_DEGREES_TRUE;
    double INCIDENCE_ALPHA;
    double AIRSPEED_INDICATED;
    double VERTICAL_SPEED;
    double INDICATED_ALTITUDE;
    double G_FORCE;
} FLIGHT_PARAMETERS;
```

A simulation variable is a parameter contained within the simulator that has the purpose of holding the state of a certain parameter within the simulation. Some examples of such variables are:

- “HYDRAULIC SWITCH”, which may be ON/OFF (Boolean)[22]
- “AUTOPILOT HEADING LOCK DIR”, which is an integer[22]
- “AUTOPILOT MAX BANK” which is a real number expressed in radians[22]

The way in which a data definition member is linked to a simulation variable is by use of a SimConnect call. This call adds an offset and a type to a client data definition:[23]

```
HRESULT SimConnect_AddToClientDataDefinition(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_DATA_DEFINITION_ID DefineID,
    DWORD dwOffset,
    DWORD dwSizeOrType,
    float fEpsilon = 0,
    DWORD DatumID = SIMCONNECT_UNUSED
);
```

where

- `hSimConnect` is the handle to the SimConnect object[23]
- `DefineID` is the ID of the client-defined data definition[23]
- `dwOffset` is containing the offset into the client area, where the addition starts[23]
- `dwSizeOrType` is containing the size of the client data in bytes[23]
- `fEpsilon [optional]` is indicating the minimum absolute change in a specified parameter before it is sent to the client[23]
- `DatumID [optional]` is used to identify if the data is returned in tagged format[23]

It is, however, not enough for a data definition to be linked to a SimConnect Variable for the client to receive updated data on that subset of parameters. The link to the server is made by use of requests. A request is used to identify which data is received by the client:

```
HRESULT SimConnect_RequestDataOnSimObject(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    SIMCONNECT_DATA_DEFINITION_ID DefineID,
    SIMCONNECT_OBJECT_ID ObjectID,
    SIMCONNECT_PERIOD Period,
    SIMCONNECT_DATA_REQUEST_FLAG Flags = 0,
    DWORD origin = 0,
    DWORD interval = 0,
    DWORD limit = 0
);
```

where

- `hSimConnect` is the handle to the SimConnect object[23]
- `RequestID` is the ID of the client-defined request[23]
- `DefineID` is the ID of the client-defined data definition[23]
- `ObjectID` is the ID of the Prepar3D object the data should be about[23]
- `Period` specifies how often data is sent from the server and received by the client[23]
- `Flags [optional]` may indicate whether data is sent only when it changes, or continuously[23]
- `Origin [optional]` is the number of Periods that have to pass until data transmission begins[23]
- `Interval [optional]` is the number of Periods that have to pass between data transmissions[23]
- `Limit [optional]` is the number of times data should be sent before communication ends[23]

GROUPS

A group is a set of events that can be collected together in order to impose a priority of sending information from the simulator to clients. No two groups can be set at the same priority. If a group is to be assigned an already existing priority, then it will automatically be assigned the next lowest available priority.[24]

```
HRESULT SimConnect_SetNotificationGroupPriority(
    HANDLE hSimConnect,
    SIMCONNECT_NOTIFICATION_GROUP_ID GroupID,
    DWORD uPriority
);
```

where

- `hSimConnect` is the handle to the SimConnect object[24]
- `GroupID` is the ID of the client-defined group[24]
- `uPriority` is the priority of the group[24]

A SimConnect input can be, for example, a joystick movement from part of the player. In the same manner, an event can be viewed as a generic input, such as lowering the landing gear. All inputs and events can be mapped in the client to correspondent server structures.

```
HRESULT SimConnect_MapInputEventToClientEvent(
    HANDLE hSimConnect,
    SIMCONNECT_INPUT_GROUP_ID GroupID,
    const char* pszInputDefinition,
    SIMCONNECT_CLIENT_EVENT_ID DownEventID,
    DWORD DownValue = 0,
    SIMCONNECT_CLIENT_EVENT_ID UpEventID = (SIMCONNECT_CLIENT_EVENT_ID)
SIMCONNECT_UNUSED,
    DWORD UpValue = 0,
    BOOL bMaskable = FALSE
);
```

where

- `hSimConnect` is the handle to the SimConnect object[24]
- `GroupID` is the ID of the client-defined group that the input event is added to[24]
- `pszInputDefinition` pointer to a string containing definition of input events[24]
- `DownEventID` is the client-defined event that triggers when the input event occurs[24]
- `DownValue [optional]` specifies a return value for when the event occurs[24]
- `UpEventID` is the client-defined event that triggers when the input event occurs[24]
- `UpValue [optional]` specifies a return value for when the event occurs[24]
- `bMaskable` specifies that the client will mask the event[24]

Implementation

Architecture. Flow of Operation

Due to the complex architecture of the system, a direct inclusion of the UML class diagram is not feasible since no details will be distinguishable. Thus, the UML class diagram is being digitally attached to this file.

AFCS_UI.mdj

When the software starts, a thread corresponding to UI operation is created, which will in turn create a new thread corresponding to the AFCS logic. The reason behind this is the responsiveness of the UI. In other words, if the same thread were to be used for both the UI and the internal logic, the UI would have frozen at software launch due to the high computational resource demand of the background logic.

The controller logic is responsible for establishing the necessary connection, including continuous requests for flight parameters which may change in time from the simulator. Also, if the connection succeeds, the UI will be signaled to refresh accounting for the new connection status.

These requests can be seen as a "subscription" of the client (in this case, the controller), to the server's information (the simulator). Thus, with each simulation frame the simulator will send a "package" containing the subscribed parameters, updated with the values the simulator dictates.

From this point forward, the controller will run the internal logic for as long as the connection status is open. Within this internal logic, one of three actions are made at each step:

1. The client receives a `SIMCONNECT_RECV_ID_EVENT`, within which the event is either `JOYSTICK_EVENT_X_AXIS` or `JOYSTICK_EVENT_Y_AXIS`. This will determine the update of the associated position of the joystick axis within the client logic. It means that the client will have an updated position of all the necessary joystick axes with the frequency requested at the establishment of the connection – each simulation frame.

2. The client receives a `SIMCONNECT_RECV_ID_SIMOBJECT_DATA`, within which the request is `REQUEST_FLIGHT_PARAMETERS`. This will determine the update of all the required simulation parameters and a cycle of running the Autoflight and the protections logic, as well as a refresh of the UI with the new parameters. What this means is that:

    a. If the Autoflight system is active, it will send control surface outputs in order to meet the required parameter demand (according to the selection made by the user on the UI).
    b. All the protections will be "notified" by the arrival of new simulation parameters, and if within one protection the checked parameter exceeds the flight envelope limitations, then a control surface output will be sent to the simulator in order for the specified parameter to return within the envelope.

3. The client receives a `SIMCONNECT_RECV_ID_QUIT`, signaling that the connection with the simulator has ended.

In order to understand how the Autoflight and Protections modules function within the same logic in order to achieve a given goal, it must first be understood what they have in common, specifically, the `IAFCS_MODULE` and `IAFCS_MODULE_THREADED` interfaces. These interfaces are meant to provide a framework which offers a way for the main controller to run/stop the module, abstracting its logic.

The difference between the two interfaces is that, while the former runs the module on the same thread as the controller, the latter runs on a new thread. Thread safety is ensured by the design of the system, where no one but the controller writes data it receives from the simulator, while the rest of the threads are only reading it.

The working logic of every Autoflight and Protection module will be thoroughly investigated in the following chapter.

The autopilot function comprises automatic control of the elevator in pitch (in order to meet vertical speed/altitude demand), and ailerons in roll (in order to meet bank angle/heading demand). Likewise, the autothrottle function is designed to regulate engine thrust output in order to maintain a selected speed.

COMMAND PITCH

DESCRIPTION

The purpose of the autoflight command pitch function is to adjust the simulator vertical speed in order to meet a certain demand, necessary to attain and/or maintain a certain altitude.

This is achieved by computing a necessary vertical speed to achieve a selected altitude by running the altitude difference (error) through the PID controller. A delta is then made between the output and the actual vertical speed. This delta (required vertical speed adjustment) is fed into another PID controller, which will determine the required elevator position in order to meet the required demand. Lastly, the computed elevator position is sent to the simulator via the `SimConnect_SetDataOnSimObject` call.

In order to prevent the autopilot from performing abrupt maneuvers, uncomfortable for passengers, the elevator controller is restricted in operation to only half of the elevator's full travel range.

```
	/// Vertical speed controller output is adjusted according to the
difference between target altitude and actual altitude
	double ALTITUDE_ERROR = ALTITUDE_SELECT - AFCS_CTL-
>FLIGHT_PARAMETERS.INDICATED_ALTITUDE;
	double VERTICAL_SPEED_CONTROLLER_OUTPUT = VERTICAL_SPEED_CONTROLLER-
>Update(ALTITUDE_ERROR);

	/// Elevator position is adjusted according to the difference between
target vertical speed and actual vertical speed
	double VERTICAL_SPEED_ERROR = VERTICAL_SPEED_CONTROLLER_OUTPUT - AFCS_CT
L->FLIGHT_PARAMETERS.VERTICAL_SPEED;
	double ELEVATOR_POSITION = SURFACE_CONTROLLER-
>Update(VERTICAL_SPEED_ERROR);

	/// Send updated elevator position to the simulator
	AFCS_CTL->ELEVATOR_POSITION.ELEVATOR_POSITION = ELEVATOR_POSITION;
	SimConnect_SetDataOnSimObject(AFCS_CTL->hSimConnect, AFCS_CTL-
>DEFINITION_ELEVATOR_POSITION, SIMCONNECT_OBJECT_ID_USER, 0, 1, sizeof(AFCS_CT
L->ELEVATOR_POSITION), &AFCS_CTL->ELEVATOR_POSITION);
```

CONTROLS AND INDICATIONS

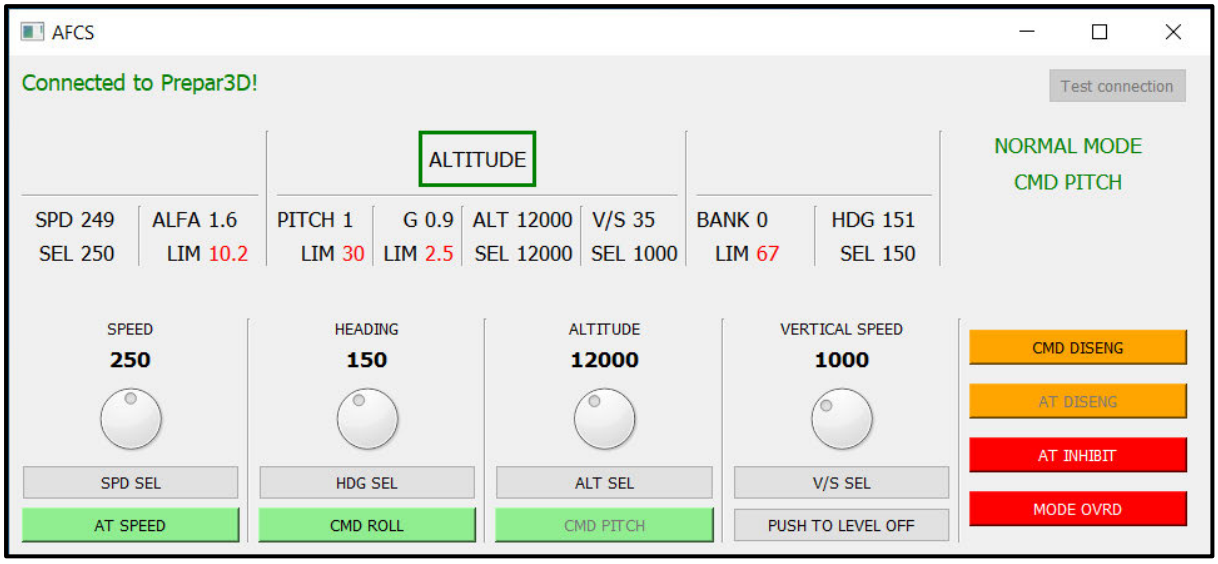


Figure 34: Autopilot command pitch

The autopilot command pitch is enabled by the following methodology:

1. The desired altitude is selected by rotating the altitude dial.
2. The "ALT SEL" button is pressed, which sends the altitude set by the knob to the controller. A check must be made to determine whether the altitude displayed on the flight mode annunciator is the desired one.
3. The desired vertical speed is selected by rotating the vertical speed dial. This is the maximum vertical speed that can be attained while attempting to intercept the selected altitude.
4. The "V/S SEL" button is pressed, which sends the vertical speed set by the knob to the controller. A check must be made to determine whether the vertical speed displayed on the flight mode annunciator is the desired one.
5. The "CMD PITCH" button can be pressed, which engages the autopilot command pitch. This can be confirmed by the "CMD PITCH" callout in the status column of the flight mode annunciator.
6. Any change in altitude and/or vertical speed can be carried out by using the same methodology, without the need to press "CMD PITCH" again.

The "PUSH TO LEVEL OFF" button, once pressed, will determine a selection of the current altitude into the controller, and a vertical speed of 500 feet per minute which can be used to attain this altitude.

In order to disconnect the autopilot command pitch, the button "CMD DISENG" has to be pressed. The autopilot disconnects in all channels, which can be confirmed by the disappearance of "CMD PITCH" and/or "CMD ROLL" in the flight mode annunciator status column.

COMMAND ROLL

DESCRIPTION

The purpose of autopilot command roll is to adjust the simulator bank angle in order to meet a certain demand, necessary to attain and/or maintain a certain heading.

This is achieved by computing a necessary bank angle to achieve a selected heading by running the heading difference (error) through the PID controller. A delta is then made between the output and the actual bank angle. This delta (required bank angle adjustment) is fed into another PID controller, which will determine the required aileron position in order to meet the required demand. Lastly, the computed aileron position is sent to the simulator via the `SimConnect_SetDataOnSimObject` call.

```
	/// Heading controller output is adjusted according to the difference
between target heading and actual heading
	double HEADING_ERROR = HEADING_SELECT - RadiansToDegrees(AFCS_CTL-
>FLIGHT_PARAMETERS.PLANE_HEADING_DEGREES_TRUE);
	double HEADING_CONTROLLER_OUTPUT = HEADING_CONTROLLER-
>Update(HEADING_ERROR);

	/// Aileron position is adjusted according to the difference between
target bank angle and actual bank angle
	double BANK_ANGLE_ERROR = HEADING_CONTROLLER_OUTPUT - -AFCS_CTL-
>FLIGHT_PARAMETERS.PLANE_BANK_DEGREES;
	double AILERON_POSITION = SURFACE_CONTROLLER->Update(BANK_ANGLE_ERROR);

	/// Send updated aileron position to the simulator
	AFCS_CTL->AILERON_POSITION.AILERON_POSITION = AILERON_POSITION;
	SimConnect_SetDataOnSimObject(AFCS_CTL->hSimConnect, AFCS_CTL-
>DEFINITION_AILERON_POSITION, SIMCONNECT_OBJECT_ID_USER, 0, 1, sizeof(AFCS_CTL
->AILERON_POSITION), &AFCS_CTL->AILERON_POSITION);
```

CONTROLS AND INDICATIONS

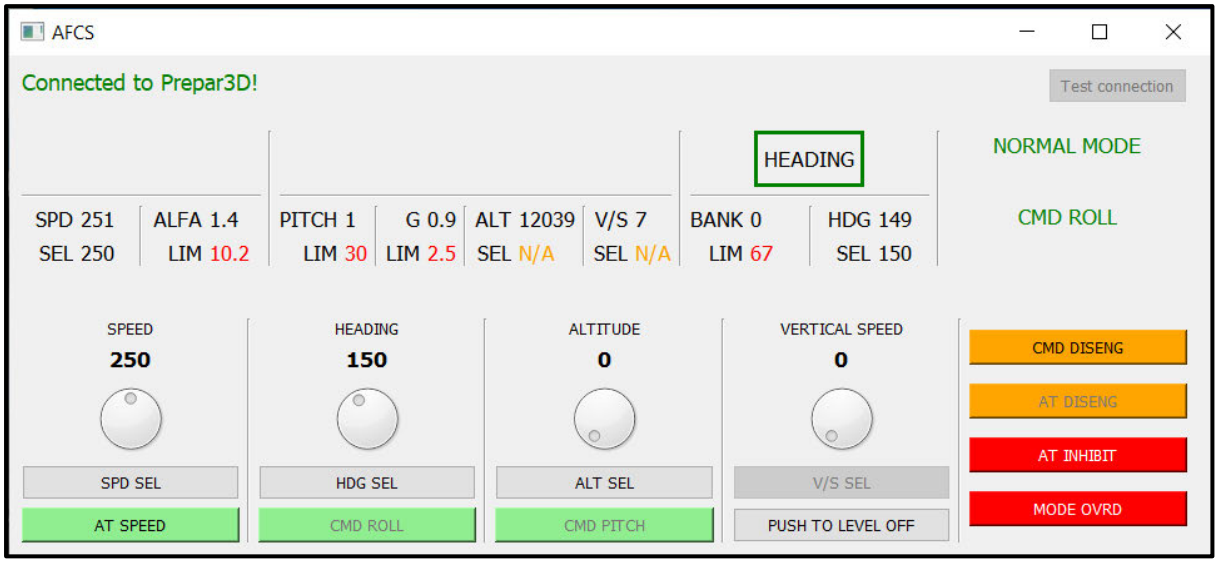


Figure 35: Autopilot command roll

The autopilot command roll is enabled by the following methodology:

1. The desired heading is selected by rotating the heading dial.
2. The "HDG SEL" button is pressed, which sends the heading set by the knob to the controller. A check must be made to determine whether the heading displayed on the flight mode annunciator is the desired one.
3. The "CMD ROLL" button can be pressed, which engages the autopilot command roll. This can be confirmed by the "CMD ROLL" callout in the status column of the flight mode annunciator.
4. Any change in heading can be carried out by using the same methodology, without the need to press "CMD ROLL" again.

In order to disconnect the autopilot command roll, the button "CMD DISENG" has to be pressed. The autopilot disconnects in all channels, which can be confirmed by the disappearance of "CMD PITCH" and/or "CMD ROLL" in the flight mode annunciator status column.

DESCRIPTION

The purpose of the autothrottle is to adjust engine thrust output in order to meet a certain engine N1 rating (first-stage compressor RPM, in percent), necessary to hold a selected speed.

The thrust requirement is being determined by calculating the difference between the required and the actual speed, and then feeding this difference into the thrust levers' PID controller. The computed thrust lever position is then being sent to the simulator via the `SimConnect_SetDataOnSimObject` call. In order to prevent the engine from operating outside normal regimes for flight, the thrust lever output is restricted between flight idle and maximum climb thrust.

```
	/// Throttle controller output is adjusted according to the difference
between target speed and actual speed
	double SPEED_ERROR = SPEED_SELECT - AFCS_CTL-
>FLIGHT_PARAMETERS.AIRSPEED_INDICATED;
	double THROTTLE_CONTROLLER_OUTPUT = SURFACE_CONTROLLER-
>Update(SPEED_ERROR);

	/// Send updated throttle levers position to the simulator
	AFCS_CTL-
>GENERAL_ENG_THROTTLE_LEVER_POSITION.GENERAL_ENG_THROTTLE_LEVER_POSITION_1 =
	  AFCS_CTL-
>GENERAL_ENG_THROTTLE_LEVER_POSITION.GENERAL_ENG_THROTTLE_LEVER_POSITION_2 =
	  AFCS_CTL-
>GENERAL_ENG_THROTTLE_LEVER_POSITION.GENERAL_ENG_THROTTLE_LEVER_POSITION_3 =
	  AFCS_CTL-
>GENERAL_ENG_THROTTLE_LEVER_POSITION.GENERAL_ENG_THROTTLE_LEVER_POSITION_4 =
	  THROTTLE_CONTROLLER_OUTPUT;
	SimConnect_SetDataOnSimObject(AFCS_CTL->hSimConnect, AFCS_CTL-
>DEFINITION_GENERAL_ENG_THROTTLE_LEVER_POSITION, SIMCONNECT_OBJECT_ID_USER, 0,
 1, sizeof(AFCS_CTL->GENERAL_ENG_THROTTLE_LEVER_POSITION), &AFCS_CTL-
>GENERAL_ENG_THROTTLE_LEVER_POSITION);
```

CONTROLS AND INDICATIONS

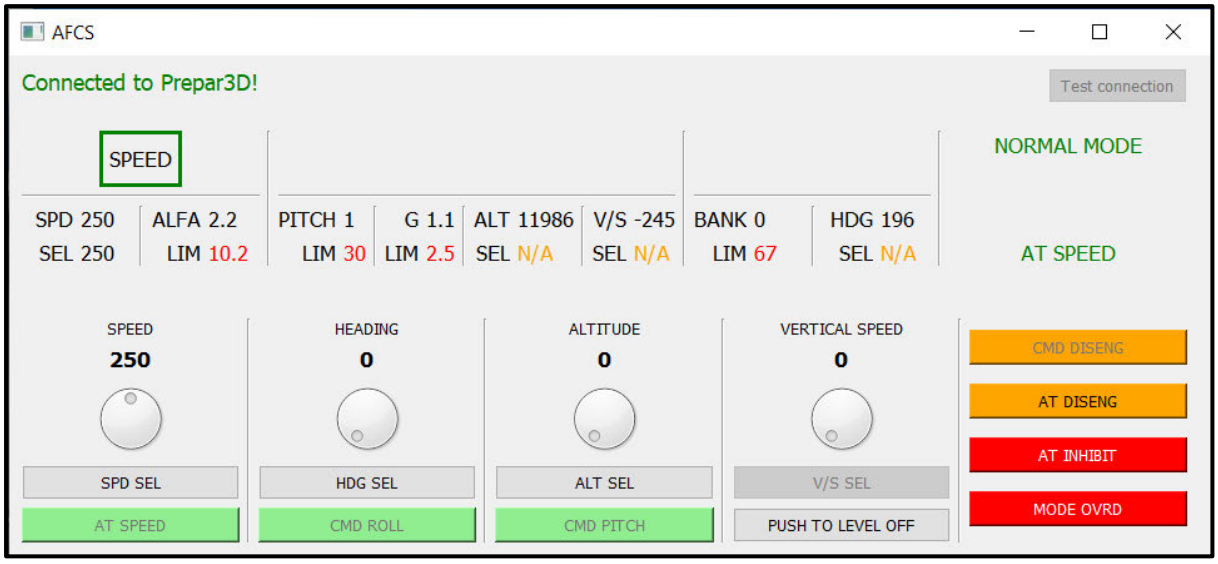


Figure 36: Autothrottle function

The autothrottle is enabled by the following methodology:

1. The desired speed is selected by rotating the speed dial.
2. The "SPD SEL" button is pressed, which sends the speed set by the knob to the controller. A check must be made to determine whether the speed displayed on the flight mode annunciator is the desired one.
3. The "AT SPEED" button can be pressed, which engages the autothrottle. This can be checked by the "AT SPEED" callout in the status column of the flight mode annunciator.
4. Any change in speed can be carried out by using the same methodology, without the need to press "AT SPEED" again.

In order to disconnect the autothrottle, the button "AT DISENG" has to be pressed. The autothrottle disconnects, which can be confirmed by the disappearance of "AT SPEED" in the flight mode annunciator status column.

In view of safety, if the autothrottle system malfunctions some type of measure has to be taken in order to ensure that it cannot be operated anymore until the end of the flight. Thus, by pressing the "AT INHIBIT" button, the autothrottle will cease operation for the remainder of the flight, regardless of pilot or AFCS engagement command. A red "AT INHIBIT" message will be displayed in the status column of the flight mode annunciator.

Operation in Manual Mode

The automatic flight control system normal mode provides flight envelope protection in both the pitch channel and roll channel. It is designed to aid the pilot in maintaining the flight parameters within the safe limits, by overriding his inputs when the parameters tend to exceed them. Normal mode provides protection through five modules, namely pitch protection, roll protection, alfa protection, overspeed protection and maneuver protection.

PITCH PROTECTION

DESCRIPTION

The AFCS pitch protection is designed to restrict the pitch authority of the aircraft within the following limits:

- A maximum of 30 degrees airplane nose up, and the pitch authority begins decreasing linearly when the pitch raises above 25 degrees airplane nose up.
- A maximum of 15 degrees airplane nose down, and the pitch authority begins decreasing linearly when the pitches raises above 10 degrees airplane nose down.

This is achieved by use of the following logic: if the pitch angle is above the limit threshold, use maximum elevator authority to arrest it, and bring the pitch angle back into the permissible envelope.

If the pitch angle is not above the permissible limit, but however above the maximum reduced authority threshold, the pitch authority toward the limit is restricted as follows:

$$P_{MA} = P_{MR} - P_{MR} * \frac{P_{ACT} - P_{MRAT}}{P_{MAX} - P_{MRAT}}$$

where

- P_{MA} is the maximum pitch authority at a certain simulation frame
- P_{MR} is the maximum requestable pitch rate demandable through elevator input
- P_{ACT} is the actual pitch of the aircraft at that simulation frame
- P_{MRAT} is the maximum reduced authority pitch threshold
- P_{MAX} is the maximum pitch which can be attained in normal mode.

Notice that

$$\lim_{P \to P_{MAX}} \left(P_{MR} - P_{MR} * \frac{P_{ACT} - P_{MRAT}}{P_{MAX} - P_{MRAT}} \right) = 0.$$

Thus, as the pitch increases toward the maximum allowable limit, the maximum attainable pitch rate decreases toward zero. The same is valid regardless if the pitch advances toward the maximum airplane nose up or airplane nose down limit.

Afterward, the maximum allowable pitch rate is sent to the elevator's PID controller in order to determine the maximum permissible elevator deflection. If the resulted elevator deflection is greater than the physical maximum elevator travel freedom, it is reduced to this value. This value is then sent to the simulator.

```
	double FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES = AFCS_CTL-
>FLIGHT_PARAMETERS.PLANE_PITCH_DEGREES;
	double PITCH_DESIRED_RATE = AFCS_CTL-
>JOYSTICK_INPUT.Y_AXIS * PITCH_MAXIMUM_REQUESTABLE_RATE;
	bool GOES_TOWARD_LIMIT = (Sign(FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES) !=
 Sign(PITCH_DESIRED_RATE));

	/// If the pitch angle is greater than the maximum pitch angle, use
maximum elevator available to reduce it
	if (FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES <= PITCH_MAXIMUM_POSITIVE_ANGL
E || FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES >= PITCH_MAXIMUM_NEGATIVE_ANGLE)
	{
		PITCH_DESIRED_RATE = PITCH_RECOVERY_RATE * Sign(FLIGHT_PARAMETERS
	_PLANE_PITCH_DEGREES);
	}
	else
		/// If the pitch angle is greater than the protection engagement
	threshold, linearly reduce the pitch autority to reach 0 at the maximum
	pitch angle
		if (FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES < PITCH_REDUCED_AUTHORI
	TY_POSITIVE_THRESHOLD || FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES > PITCH_R
	EDUCED_AUTHORITY_NEGATIVE_THRESHOLD && GOES_TOWARD_LIMIT == true)
		{
			double PITCH_MAXIMUM_RATE = 0;
			if (FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES < PITCH_REDUCED_A
		UTHORITY_POSITIVE_THRESHOLD)
			{
				PITCH_MAXIMUM_RATE = PITCH_MAXIMUM_REQUESTABLE_RATE +
			 (FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES - PITCH_REDUCED_AUT
			HORITY_POSITIVE_THRESHOLD) * (0 - PITCH_MAXIMUM_REQUESTABLE
			_RATE) / (PITCH_MAXIMUM_POSITIVE_ANGLE - PITCH_REDUCED_AUTH
			ORITY_POSITIVE_THRESHOLD);
			}
			if (FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES > PITCH_REDUCED_A
		UTHORITY_NEGATIVE_THRESHOLD)
			{
				PITCH_MAXIMUM_RATE = PITCH_MAXIMUM_REQUESTABLE_RATE +
			 (FLIGHT_PARAMETERS_PLANE_PITCH_DEGREES - PITCH_REDUCED_AUT
			HORITY_NEGATIVE_THRESHOLD) * (0 - PITCH_MAXIMUM_REQUESTABLE
			_RATE) / (PITCH_MAXIMUM_NEGATIVE_ANGLE - PITCH_REDUCED_AUTH
			ORITY_NEGATIVE_THRESHOLD);
			}
```

```
                if (abs(PITCH_DESIRED_RATE) > PITCH_MAXIMUM_RATE)
                {
                      PITCH_DESIRED_RATE = PITCH_MAXIMUM_RATE * Sign(PITCH_
                DESIRED_RATE);
                }
          }

      auto PITCH_CONTROLLER_OUTPUT = SURFACE_CONTROLLER-
>Update(PITCH_DESIRED_RATE - AFCS_CTL-
>FLIGHT_PARAMETERS.ROTATION_VELOCITY_BODY_Y);

      if (PITCH_CONTROLLER_OUTPUT > 1) PITCH_CONTROLLER_OUTPUT = 1;
      if (PITCH_CONTROLLER_OUTPUT < -1) PITCH_CONTROLLER_OUTPUT = -1;

      /// Send updated elevator position to the simulator
      AFCS_CTL->ELEVATOR_POSITION.ELEVATOR_POSITION = PITCH_CONTROLLER_OUTPUT;
      SimConnect_SetDataOnSimObject(AFCS_CTL->hSimConnect, AFCS_CTL-
>DEFINITION_ELEVATOR_POSITION, SIMCONNECT_OBJECT_ID_USER, 0, 1, sizeof(AFCS_CT
L->ELEVATOR_POSITION), &AFCS_CTL->ELEVATOR_POSITION);
```

CONTROLS AND INDICATIONS

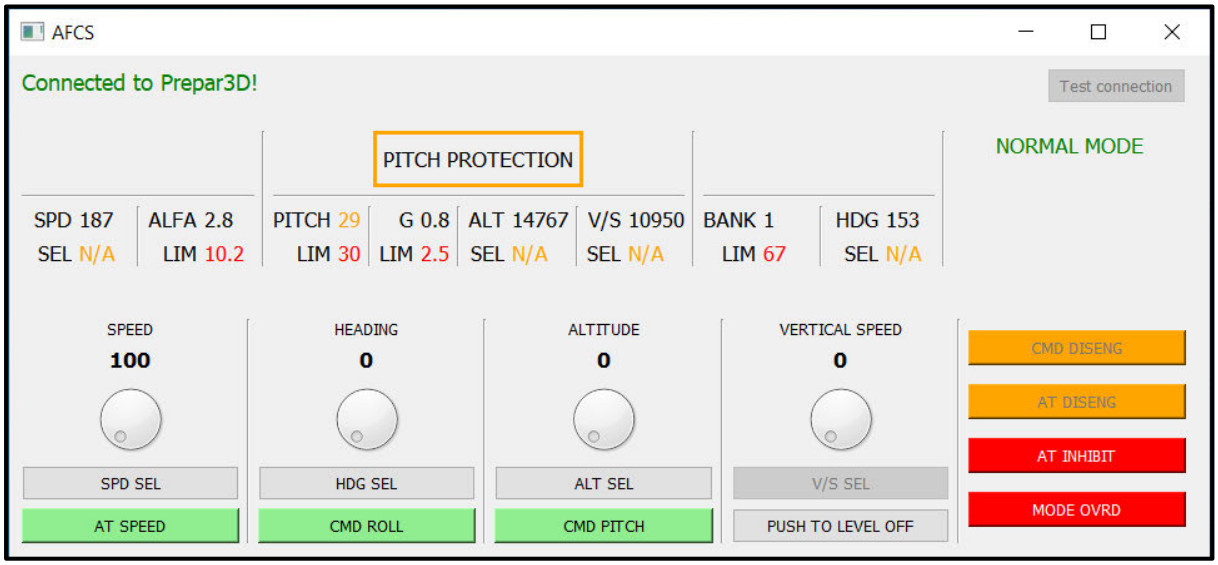


Figure 37: AFCS pitch protection

While the pitch protection is active, a “PITCH PROTECTION” message appears on the flight mode annunciator. All autopilot and autothrottle functions are temporarily inhibited.

ROLL PROTECTION

The AFCS roll protection is designed to restrict the roll authority of the aircraft within 67 degrees of bank angle, and the roll authority begins decreasing linearly when the bank angle reaches 33 degrees. Moreover, at any bank angle above 33 degrees, if the control column is released, the aircraft will return to and maintain 33 degrees of bank.

This is achieved by use of the following logic: if the bank angle is above the limit threshold, use maximum aileron authority to arrest it, and bring the bank angle back into the permissible envelope.

If the bank angle is not above the permissible limit, but however above the maximum reduced authority threshold, the roll authority toward the limit is restricted as follows:

$$R_{MA} = R_{MR} - R_{MR} * \frac{BA_{ACT} - BA_{MRAT}}{BA_{MAX} - BA_{MRAT}}$$

where

- R_{MA} is the maximum roll authority at a certain simulation frame
- R_{MR} is the maximum requestable roll rate demandable through aileron input
- BA_{ACT} is the actual bank angle of the aircraft at that simulation frame
- BA_{MRAT} is the maximum reduced authority bank angle threshold
- BA_{MAX} is the maximum bank angle which can be attained in normal mode.

Notice that

$$\lim_{BA \to BA_{MAX}} \left(R_{MR} - R_{MR} * \frac{BA_{ACT} - BA_{MRAT}}{BA_{MAX} - BA_{MRAT}}\right) = 0.$$

Thus, as the bank angle increases toward the maximum allowable limit, the maximum attainable roll rate decreases toward zero. Also, if the bank angle is above the reduced authority threshold and the control column is left to neutral, the AFCS will use a nominal recovery rate to bring the bank angle back to the reduced authority threshold limit.

Afterward, the maximum allowable roll rate is sent to the aileron's PID controller in order to determine the maximum permissible aileron deflection. If the resulted aileron deflection is greater than the physical maximum aileron travel freedom, it is reduced to this value. This value is then sent to the simulator.

```
      double ABS_FLIGHT_PARAMETERS_PLANE_BANK_DEGREES = abs(AFCS_CTL-
>FLIGHT_PARAMETERS.PLANE_BANK_DEGREES);
      double ROLL_DESIRED_RATE = AFCS_CTL-
>JOYSTICK_INPUT.X_AXIS * ROLL_MAXIMUM_REQUESTABLE_RATE;
      bool GOES_TOWARD_LIMIT = (Sign(AFCS_CTL-
>FLIGHT_PARAMETERS.PLANE_BANK_DEGREES) != Sign(AFCS_CTL-
>JOYSTICK_INPUT.X_AXIS));
```

```
	/// If the bank angle is greater than the maximum bank angle, use
maximum aileron available to reduce it
	if (ABS_FLIGHT_PARAMETERS_PLANE_BANK_DEGREES >= ROLL_MAXIMUM_ANGLE)
	{
		ROLL_DESIRED_RATE = ROLL_RECOVERY_RATE * Sign(AFCS_CTL-
	>FLIGHT_PARAMETERS.PLANE_BANK_DEGREES);
	}
	else
		/// If the bank angle is greater than the protection engagement
	threshold, linearly reduce the roll autority to reach 0 at the maximum
	bank angle
		if (ABS_FLIGHT_PARAMETERS_PLANE_BANK_DEGREES > ROLL_REDUCED_AUTHO
	RITY_THRESHOLD && AFCS_CTL-
	>JOYSTICK_INPUT.X_AXIS != 0 && GOES_TOWARD_LIMIT == true)
		{
			double ROLL_MAXIMUM_RATE = ROLL_MAXIMUM_REQUESTABLE_RATE +
			(ABS_FLIGHT_PARAMETERS_PLANE_BANK_DEGREES - ROLL_REDUCED_AUTHORIT
			Y_THRESHOLD) * (0 - ROLL_MAXIMUM_REQUESTABLE_RATE) / (ROLL_MAXIMU
			M_ANGLE - ROLL_REDUCED_AUTHORITY_THRESHOLD);

			if (abs(ROLL_DESIRED_RATE) > ROLL_MAXIMUM_RATE)
			{
				ROLL_DESIRED_RATE = ROLL_MAXIMUM_RATE * Sign(ROLL_DES
			IRED_RATE);
			}
		}
		else
			/// If the stick is released with the bank angle above the
			protection engagement threshold, recover to the nominal bank
			angle in order to exit the protection zone
			if (ABS_FLIGHT_PARAMETERS_PLANE_BANK_DEGREES > ROLL_REDUCED
			_AUTHORITY_THRESHOLD && AFCS_CTL->JOYSTICK_INPUT.X_AXIS == 0)
			{
				ROLL_DESIRED_RATE = ROLL_RECOVERY_RATE * Sign(AFCS_CT
			L->FLIGHT_PARAMETERS.PLANE_BANK_DEGREES);
			}

	auto ROLL_CONTROLLER_OUTPUT = SURFACE_CONTROLLER-
>Update(ROLL_DESIRED_RATE - AFCS_CTL-
>FLIGHT_PARAMETERS.ROTATION_VELOCITY_BODY_X);

	if (ROLL_CONTROLLER_OUTPUT > 1) ROLL_CONTROLLER_OUTPUT = 1;
	if (ROLL_CONTROLLER_OUTPUT < -1) ROLL_CONTROLLER_OUTPUT = -1;

	/// Send updated aileron position to the simulator
	AFCS_CTL->AILERON_POSITION.AILERON_POSITION = ROLL_CONTROLLER_OUTPUT;
```

```
SimConnect_SetDataOnSimObject(AFCS_CTL->hSimConnect, AFCS_CTL-
>DEFINITION_AILERON_POSITION, SIMCONNECT_OBJECT_ID_USER, 0, 1, sizeof(AFCS_CTL
->AILERON_POSITION), &AFCS_CTL->AILERON_POSITION);
```

CONTROLS AND INDICATIONS

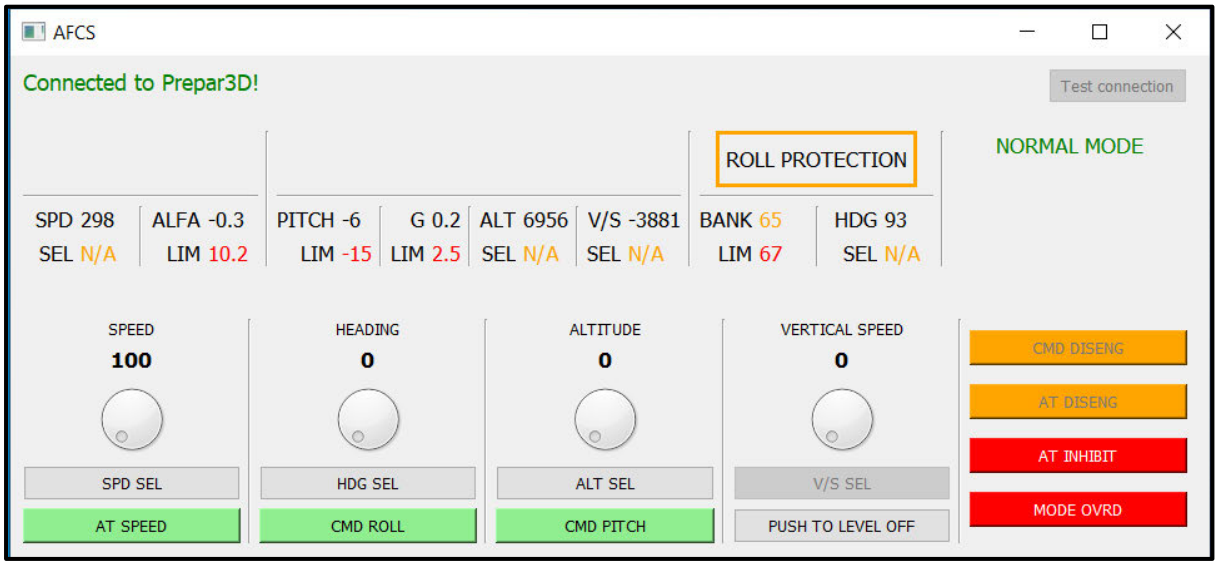


Figure 38: AFCS roll protection

While the roll protection is active, a "ROLL PROTECTION" message appears on the flight mode annunciator. All autopilot and autothrottle functions are temporarily inhibited.

ALFA PROTECTION

The AFCS alfa protection is designed to prevent an excursion of the aircraft into a stall condition when operating near maximum allowable angles of attack. The system is designed in such a manner so that no angle of attack above the stalling angle is ever achievable in normal mode for a prolonged period of time.

This is achieved by use of the following logic: if the angle of attack is above the limit threshold, use maximum elevator authority to arrest it, and bring the angle of attack back into the permissible envelope. The engines are also commanded maximum achievable thrust.

If the angle of attack is not above the permissible limit, but however above the maximum reduced authority threshold, the pitch authority toward the limit is restricted as follows:

$$P_{MA} = P_{MR} - P_{MR} * \frac{\alpha_{ACT} - \alpha_{MRAT}}{\alpha_{MAX} - \alpha_{MRAT}}$$

where

- P_{MA} is the maximum pitch authority at a certain simulation frame
- P_{MR} is the maximum requestable pitch rate demandable through elevator input
- α_{ACT} is the actual angle of attack of the aircraft at that simulation frame
- α_{MRAT} is the maximum reduced authority angle of attack threshold
- α_{MAX} is the maximum angle of attack which can be attained in normal mode.

Notice that

$$\lim_{\alpha \to \alpha_{MAX}} \left(P_{MR} - P_{MR} * \frac{\alpha_{ACT} - \alpha_{MRAT}}{\alpha_{MAX} - \alpha_{MRAT}} \right) = 0.$$

Thus, as the angle of attack increases toward the maximum allowable limit, the maximum attainable pitch rate decreases toward zero.

Afterward, the maximum allowable pitch rate is sent to the elevator's PID controller in order to determine the maximum permissible elevator deflection. If the resulted elevator deflection is greater than the physical maximum elevator travel freedom, it is reduced to this value. This value is then sent to the simulator.

```
	double FLIGHT_PARAMETERS_INCIDENCE_ALPHA = AFCS_CTL-
>FLIGHT_PARAMETERS.INCIDENCE_ALPHA;
	double ALFA_DESIRED_RATE = AFCS_CTL-
>JOYSTICK_INPUT.Y_AXIS * PITCH_MAXIMUM_REQUESTABLE_RATE;
	bool GOES_TOWARD_LIMIT = (Sign(FLIGHT_PARAMETERS_INCIDENCE_ALPHA) == Sig
n(ALFA_DESIRED_RATE));

	/// If the angle of attack is greater than the critical angle, use
maximum elevator available to reduce it
	if (FLIGHT_PARAMETERS_INCIDENCE_ALPHA >= ALFA_MAXIMUM_ANGLE)
	{
		ALFA_DESIRED_RATE = (-
	1) * PITCH_RECOVERY_RATE * Sign(FLIGHT_PARAMETERS_INCIDENCE_ALPHA);

		/// If the autothrust connection to the simulator is not inhibite
	d, set power to maximum thrust
		if (AFCS_CTL-
	>AFCS_ENGAGEMENT_STATUS.AUTOTHROTTLE_INHIBIT != true) SetTOGAThrust();
	}
	else
		/// If the angle of attack is greater than the protection
	engagement threshold, linearly reduce the pitch autority to reach 0 at
	the critical angle
```

```
		if (FLIGHT_PARAMETERS_INCIDENCE_ALPHA > ALFA_REDUCED_AUTHORITY_TH
	RESHOLD && GOES_TOWARD_LIMIT == true && AFCS_CTL-
	>JOYSTICK_INPUT.Y_AXIS != 0)
		{
			double ALFA_MAXIMUM_RATE = PITCH_MAXIMUM_REQUESTABLE_RATE +
			 (FLIGHT_PARAMETERS_INCIDENCE_ALPHA - ALFA_REDUCED_AUTHORITY_THRE
			SHOLD) * (0 - PITCH_MAXIMUM_REQUESTABLE_RATE) / (ALFA_MAXIMUM_ANG
			LE - ALFA_REDUCED_AUTHORITY_THRESHOLD);

			if (abs(ALFA_DESIRED_RATE) > ALFA_MAXIMUM_RATE)
				ALFA_DESIRED_RATE = ALFA_MAXIMUM_RATE * Sign(ALFA_DES
			IRED_RATE);

			/// If the autothrust connection to the simulator is not in
			hibited, set power to maximum thrust
			if (AFCS_CTL-
			>AFCS_ENGAGEMENT_STATUS.AUTOTHROTTLE_INHIBIT != true) SetTOGAThru
			st();
		}

	auto ALFA_CONTROLLER_OUTPUT = SURFACE_CONTROLLER-
>Update(ALFA_DESIRED_RATE - AFCS_CTL-
>FLIGHT_PARAMETERS.ROTATION_VELOCITY_BODY_Y);

	if (ALFA_CONTROLLER_OUTPUT > 1) ALFA_CONTROLLER_OUTPUT = 1;
	if (ALFA_CONTROLLER_OUTPUT < -1) ALFA_CONTROLLER_OUTPUT = -1;

	/// Send updated elevator position to the simulator
	AFCS_CTL->ELEVATOR_POSITION.ELEVATOR_POSITION = ALFA_CONTROLLER_OUTPUT;
	SimConnect_SetDataOnSimObject(AFCS_CTL->hSimConnect, AFCS_CTL-
>DEFINITION_ELEVATOR_POSITION, SIMCONNECT_OBJECT_ID_USER, 0, 1, sizeof(AFCS_CT
L->ELEVATOR_POSITION), &AFCS_CTL->ELEVATOR_POSITION);
```

CONTROLS AND INDICATIONS

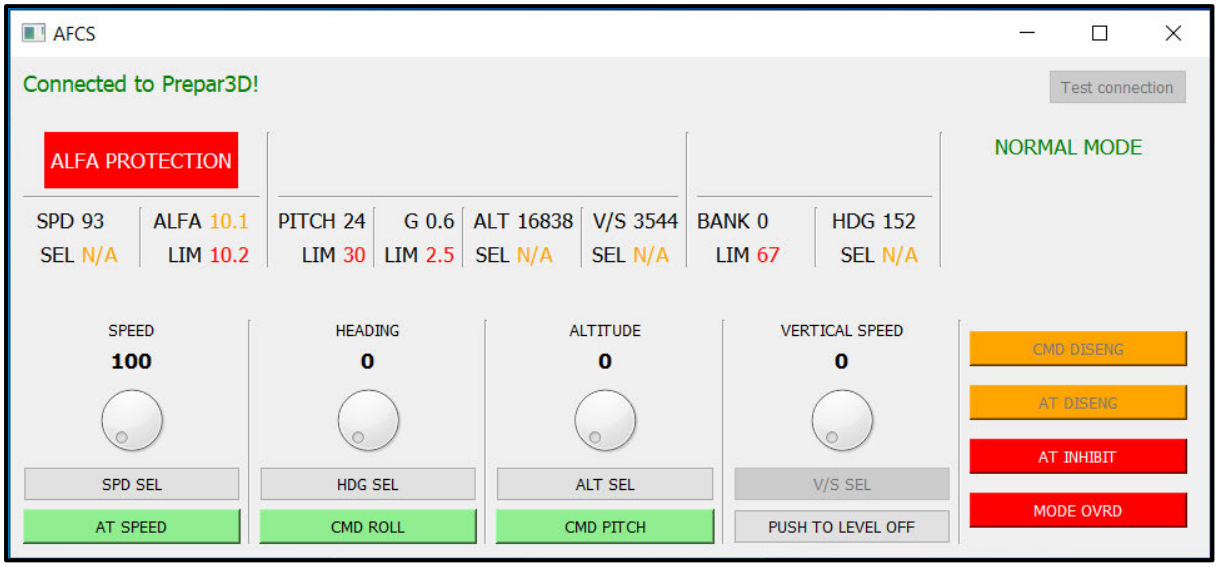


Figure 39: AFCS alfa protection

While the alfa protection is active, an "ALFA PROTECTION" message appears on the flight mode annunciator. All autopilot functions are temporarily inhibited. The behavior of the autothrottle is dictated by the alfa protection logic.

OVERSPEED PROTECTION

DESCRIPTION

The AFCS overspeed protection is designed to prevent an excursion of the aircraft into a high-speed regime where structural damage may occur. This is achieved by reducing thrust output to flight idle and commanding elevator pull up so that the speed recovers back to the safe envelope.

```
	double PITCH_DESIRED_RATE = 0;

	/// If the overspeed warning is active, use maximum elevator available
pull up
	if (AFCS_CTL->FLIGHT_PARAMETERS.OVERSPEED_WARNING == true)
	{
		PITCH_DESIRED_RATE = PITCH_RECOVERY_RATE;

		/// If the autothrust connection to the simulator is not inhibite
	d, set power to flight idle
		if (AFCS_CTL-
	>AFCS_ENGAGEMENT_STATUS.AUTOTHROTTLE_INHIBIT != true) SetIdleThrust();
	}
```

```
        auto PITCH_CONTROLLER_OUTPUT = SURFACE_CONTROLLER-
>Update(PITCH_DESIRED_RATE - AFCS_CTL-
>FLIGHT_PARAMETERS.ROTATION_VELOCITY_BODY_Y);

        if (PITCH_CONTROLLER_OUTPUT > 1) PITCH_CONTROLLER_OUTPUT = 1;
        if (PITCH_CONTROLLER_OUTPUT < -1) PITCH_CONTROLLER_OUTPUT = -1;

        /// Send updated elevator position to the simulator
        AFCS_CTL->ELEVATOR_POSITION.ELEVATOR_POSITION = PITCH_CONTROLLER_OUTPUT;
        SimConnect_SetDataOnSimObject(AFCS_CTL->hSimConnect, AFCS_CTL-
>DEFINITION_ELEVATOR_POSITION, SIMCONNECT_OBJECT_ID_USER, 0, 1, sizeof(AFCS_CT
L->ELEVATOR_POSITION), &AFCS_CTL->ELEVATOR_POSITION);
```

CONTROLS AND INDICATIONS

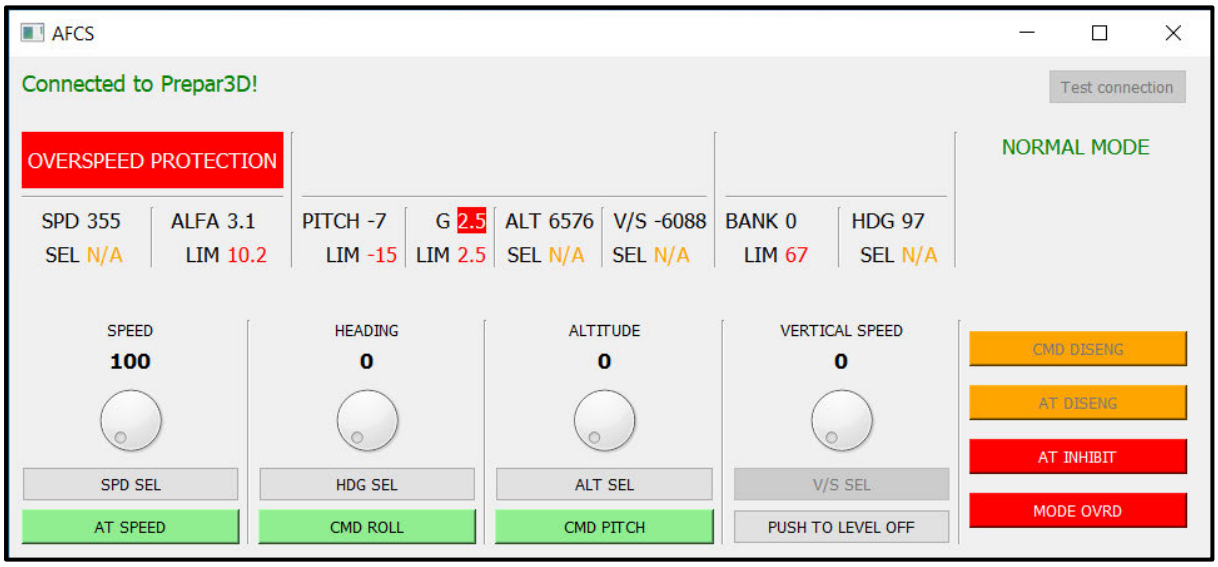


Figure 40: AFCS overspeed protection

While the overspeed protection is active, an “OVERSPEED PROTECTION” message appears on the flight mode annunciator. All autopilot functions are temporarily inhibited. The behavior of the autothrottle is dictated by the overspeed protection logic.

MANEUVER PROTECTION

DESCRIPTION

The AFCS maneuver protection is designed to prevent the aircraft from exceeding the limit load for extended periods of time. Due to the highly dynamic character of flight regarding overload values, it cannot be guaranteed that the limit load is not exceeded temporarily. However, because the ultimate load has a 50% margin from limit load, the ultimate loads are very little likely to be exceeded.

This is achieved by use of the following logic: if the overload value is above a predetermined margin to limit load, the pitch authority toward the limit is reduced as follows:

$$P_{MA} = P_{MR} - P_{MR} * \frac{\eta_{ACT} - \eta_{MRAT}}{\eta_{MAX} - \eta_{MRAT}}$$

where

- P_{MA} is the maximum pitch authority at a certain simulation frame
- P_{MR} is the maximum requestable pitch rate demandable through elevator input
- η_{ACT} is the actual aircraft overload value at that simulation frame
- η_{MRAT} is the maximum reduced authority overload threshold
- η_{MAX} is the limit load.

Notice that

$$\lim_{\eta \to \eta_{MAX}} \left(P_{MR} - P_{MR} * \frac{\eta_{ACT} - \eta_{MRAT}}{\eta_{MAX} - \eta_{MRAT}}\right) = 0.$$

Thus, as the overload value increases toward the limit load, the maximum attainable pitch rate decreases toward zero.

Afterward, the maximum allowable pitch rate is sent to the elevator's PID controller in order to determine the maximum permissible elevator deflection. If the resulted elevator deflection is greater than the physical maximum elevator travel freedom, it is reduced to this value. This value is then sent to the simulator.

```
	double FLIGHT_PARAMETERS_G_FORCE = AFCS_CTL->FLIGHT_PARAMETERS.G_FORCE;
	double PITCH_DESIRED_RATE = AFCS_CTL-
>JOYSTICK_INPUT.Y_AXIS * PITCH_MAXIMUM_REQUESTABLE_RATE;

	/// If the overload is greater than the protection engagement threshold,
 linearly reduce the pitch autority to reach 0 at the critical overload
	if (FLIGHT_PARAMETERS_G_FORCE < MANEUVER_REDUCED_AUTHORITY_MINIMUM_THRES
HOLD || FLIGHT_PARAMETERS_G_FORCE > MANEUVER_REDUCED_AUTHORITY_MAXIMUM_THRESHO
LD)
	{
		double PITCH_MAXIMUM_RATE = 0;
		if (FLIGHT_PARAMETERS_G_FORCE > MANEUVER_REDUCED_AUTHORITY_MAXIMU
	M_THRESHOLD)
		{
			PITCH_MAXIMUM_RATE = PITCH_MAXIMUM_REQUESTABLE_RATE + (FLIG
		HT_PARAMETERS_G_FORCE - MANEUVER_REDUCED_AUTHORITY_MAXIMUM_THRESH
		OLD) * (0 - PITCH_MAXIMUM_REQUESTABLE_RATE) / (MANEUVER_MAXIMUM_V
		ALUE - MANEUVER_REDUCED_AUTHORITY_MAXIMUM_THRESHOLD);
		}
```

```
            if (FLIGHT_PARAMETERS_G_FORCE < MANEUVER_REDUCED_AUTHORITY_MINIMU
      M_THRESHOLD)
            {
                  PITCH_MAXIMUM_RATE = PITCH_MAXIMUM_REQUESTABLE_RATE + (FLIG
            HT_PARAMETERS_G_FORCE - MANEUVER_REDUCED_AUTHORITY_MINIMUM_THRESH
            OLD) * (0 - PITCH_MAXIMUM_REQUESTABLE_RATE) / (MANEUVER_MINIMUM_V
            ALUE - MANEUVER_REDUCED_AUTHORITY_MINIMUM_THRESHOLD);
            }
            if (abs(PITCH_DESIRED_RATE) > PITCH_MAXIMUM_RATE)
            {
                  PITCH_DESIRED_RATE = PITCH_MAXIMUM_RATE * Sign(PITCH_DESIRE
            D_RATE);
            }
      }

      auto PITCH_CONTROLLER_OUTPUT = SURFACE_CONTROLLER-
>Update(PITCH_DESIRED_RATE - AFCS_CTL-
>FLIGHT_PARAMETERS.ROTATION_VELOCITY_BODY_Y);

      if (PITCH_CONTROLLER_OUTPUT > 1) PITCH_CONTROLLER_OUTPUT = 1;
      if (PITCH_CONTROLLER_OUTPUT < -1) PITCH_CONTROLLER_OUTPUT = -1;

      /// Send updated elevator position to the simulator
      AFCS_CTL->ELEVATOR_POSITION.ELEVATOR_POSITION = PITCH_CONTROLLER_OUTPUT;
      SimConnect_SetDataOnSimObject(AFCS_CTL->hSimConnect, AFCS_CTL-
>DEFINITION_ELEVATOR_POSITION, SIMCONNECT_OBJECT_ID_USER, 0, 1, sizeof(AFCS_CT
L->ELEVATOR_POSITION), &AFCS_CTL->ELEVATOR_POSITION);
```

CONTROLS AND INDICATIONS

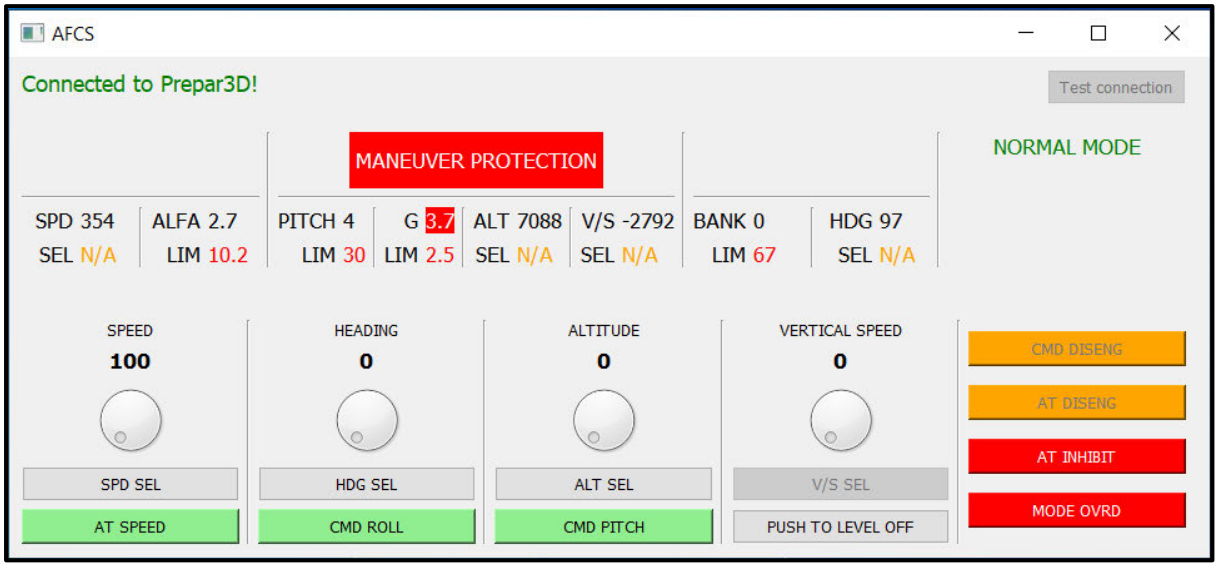


Figure 41: AFCS maneuver protection

While the maneuver protection is active, a "MANEUVER PROTECTION" message appears on the flight mode annunciator. All autopilot and autothrottle functions are temporarily inhibited.

DESCRIPTION

In view of safety, if the automatic flight control system malfunctions some type of measure has to be taken in order to ensure that it cannot be operated anymore until the end of the flight. Thus, by pressing the "MODE OVRD" button, the automatic flight control system active mode will revert to direct mode.

Flight control direct mode exhibits a direct control-to-surface relationship, reminiscent of the automatic flight control system being in the disconnected state. However, indications of parameters remain available in order to enhance the pilot's situational awareness.

CONTROLS AND INDICATIONS

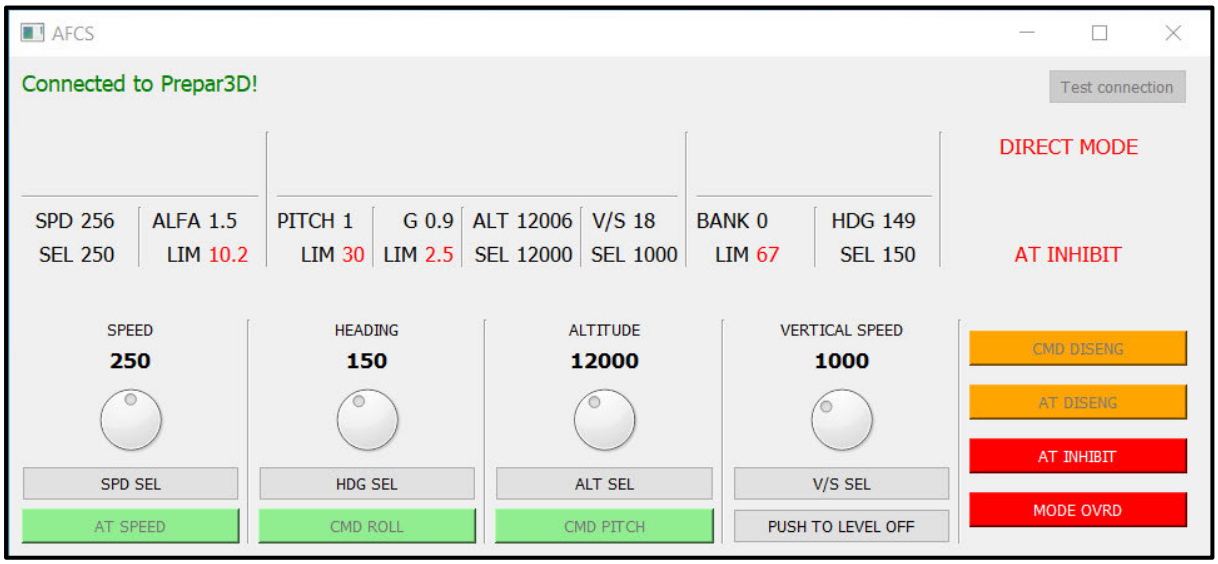


Figure 42: AFCS direct mode

A red "DIRECT MODE" message is displayed in the status column of the flight mode annunciator, indicating the active automatic flight control system has degraded to direct mode.

DESCRIPTION

Throughout the system's operation, its state can be visualized and altered via the user interface. Updates to the user interface are sent from the controller to the UI by use of the Signals and Slots system.

```
	SIMCONNECT_RECV_SIMOBJECT_DATA* pObjectData =
reinterpret_cast<SIMCONNECT_RECV_SIMOBJECT_DATA*>(pData);
	switch (pObjectData->dwRequestID)
	{
		case REQUEST_FLIGHT_PARAMETERS:
		{
			FLIGHT_PARAMETERS =
		*reinterpret_cast<TYPE_FLIGHT_PARAMETERS*>(&pObjectData->dwData);
			RunAutoflightLogic(RunProtectionsLogic());
			emit UpdateFlightParameters();
		}
		break;
	}
```

Updates are also sent, for example, when a protection engages, so that the pilot is always aware of the system state.

```
	/// Disengage Autoflight system if any protection is running
	if (OVERRIDE_CONDITION == true)
	{
		AFCS_ENGAGEMENT_STATUS.AUTOTHROTTLE_SPEED = false;
		AFCS_ENGAGEMENT_STATUS.COMMAND_ROLL = false;
		AFCS_ENGAGEMENT_STATUS.COMMAND_PITCH = false;
		emit AutoflightDisengaged();
	}
```

CONTROLS AND INDICATIONS

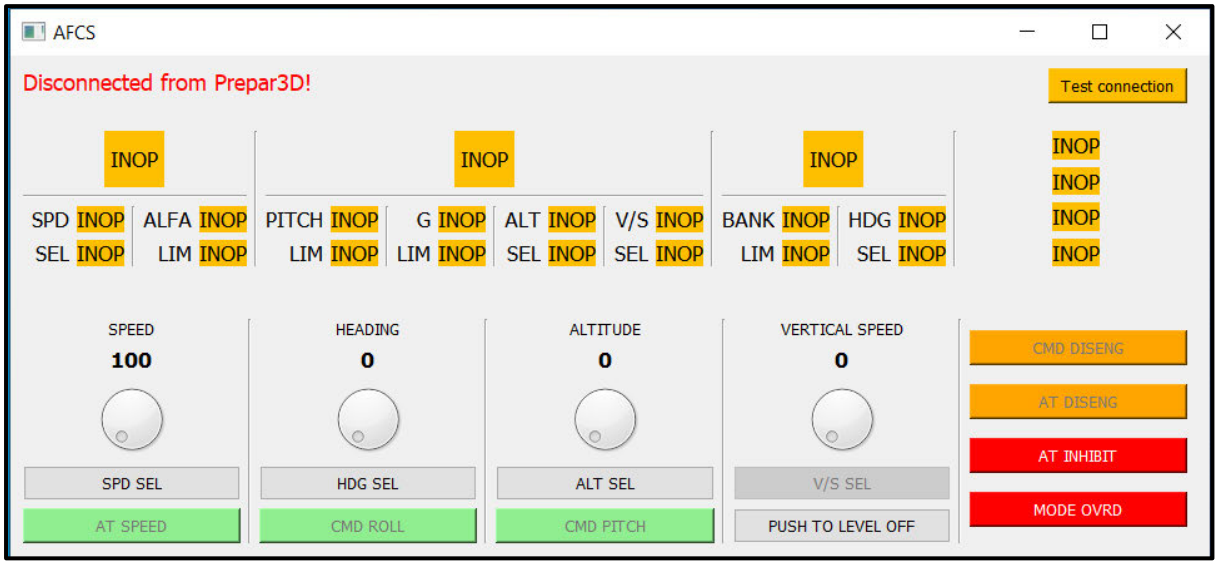


Figure 43: AFCS in disconnected state

In order to connect to the simulator, a test of the connection must be carried out. Should it be successful, the UI will refresh with data from the simulator.

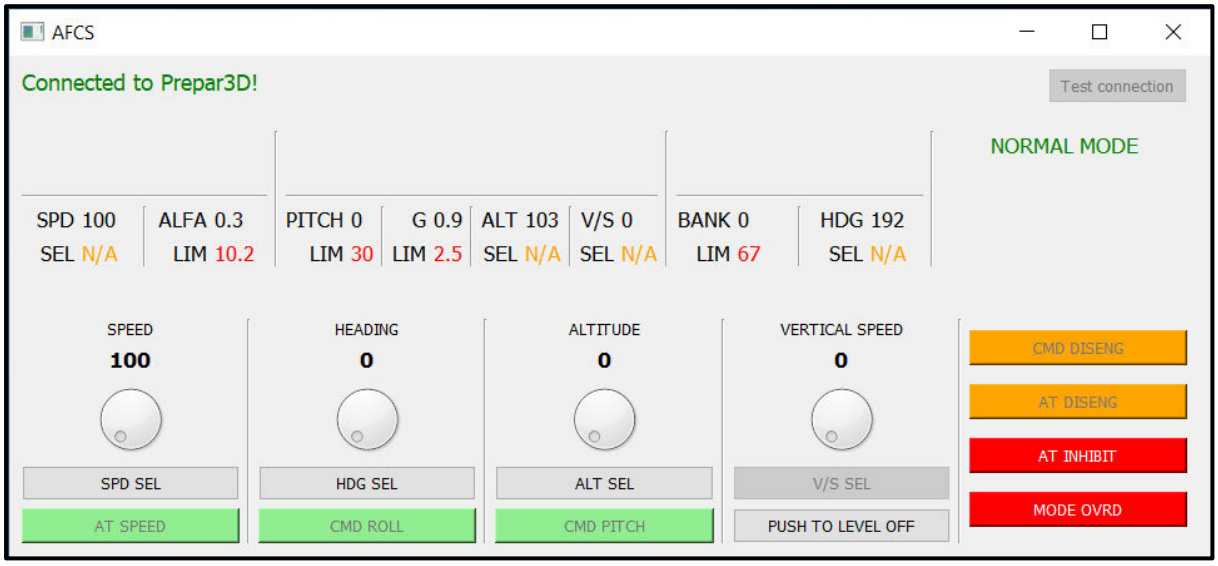


Figure 44: AFCS in connected state

HARDWARE

- Lenovo Y520-15IKBN laptop, Intel Core i7-7700HQ CPU (4 cores, 8 threads, 2.80 GHz base frequency, 3.80 GHz Max Turbo, 6 MB cache), 8 GB RAM DDR4, nVidia GeForce 1050Ti dedicated graphics card, Samsung 850 EVO SSD

  No limitations of the implemented software that are due to the hardware test platform can be identified.

- Joystick THRUSTMASTER T.16000M V.1

  Although the build of the joystick does not allow for precise control of the thrust levers, the yoke sensitivity is adequate for controlled flight. No limitations of the implemented software that are due to the used flight simulation hardware can be identified.

SOFTWARE

- Microsoft Windows 10 Home, version 1909 (OS Build 18363.778, 64-bit based OS)

  No limitations of the implemented software that are due to the operating system can be identified.

- Lockheed Martin Prepar3D v4.5.13.32097

  Although the automatic flight control system functionalities are not influenced by the deficiencies of the platform, Prepar3D has several shortcomings in comparison to its competitors.

  This is mainly due to Prepar3D considering an air vehicle as a single point in space (located in the center of pressure of the vehicle) where all aerodynamic forces are exerted. This makes the simulation unrealistic because in the real world hardly, if ever, can one find two forces symmetrically exerted between the port and the starboard of the ship.[25]

  Moreover, the Prepar3D platform does not simulate airflow in order to determine the behavior of the ship in the air. Rather, all data is extracted from generic tables which work for all aircraft.[25]

- Automatic flight control system

  The automatic flight control system is designed to make the best decision aerodynamically available in order to maintain the aircraft into the flight envelope or meet a certain demand.

  However, due to platform limitations or other unforeseen factors the aircraft may exceed the parameters of the safe flight envelope, moment at which the automatic flight control system's behavior is undefined. Should this happen, the pilot has the final responsibility to recover the aircraft from the upset situation, by disengaging the automatic flight control system using the direct mode functionality.

Scenario-based Demonstration

In order to demonstrate the capabilities of the automatic flight control system, an experimental test flight is carried out in order to test the response of the different modules under a test environment.

The scenario begins in-flight, a speed of 270 knots, an altitude of approximately 12000 feet and a heading of 195. In order to demonstrate the autothrottle module, the throttle levers are placed into an arbitrary position, a speed of 250 knots is selected into the speed panel, and AT SPEED is armed.

Immediately after activation, the throttle levers are automatically placed into the flight idle detent, while awaiting the speed to attain 250 knots. As the speed decays further, the throttle levers are smoothly advanced in order to maintain 250 knots. The speed shortly thereafter stabilizes in the area of 248 to 251 knots.

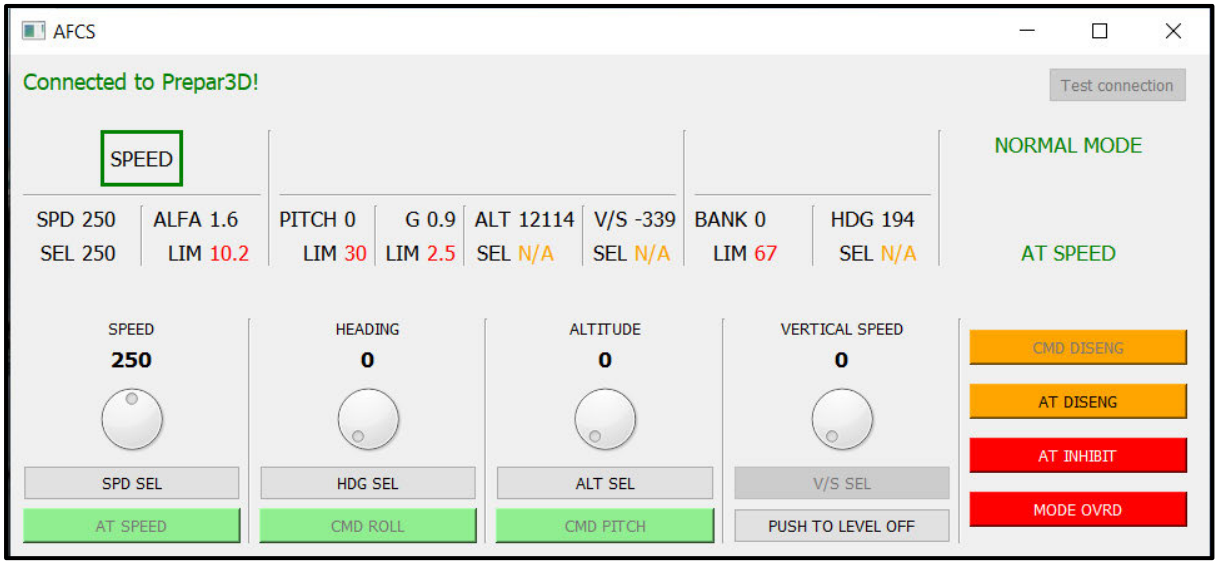


Figure 45: Autothrottle function maintaining a speed of 250 knots

Thereafter, in order to demonstrate the capability of the autopilot modules, an altitude of 15000 feet is selected, which is to be attained with a vertical speed of 1000 feet per minute. A heading of 090 is also selected into the mode control panel.

The behavior is extremely satisfactory: despite the ample flight path angle change, the autothrottle system manages to hold the speed very close to the target of 250 knots, with only a 3 knots difference which will shortly be corrected. Moreover, the aircraft immediately attains a 1000 feet per minute climb to the selected altitude of 15000 feet, and a left bank of 25 degrees in order to attain heading 090.

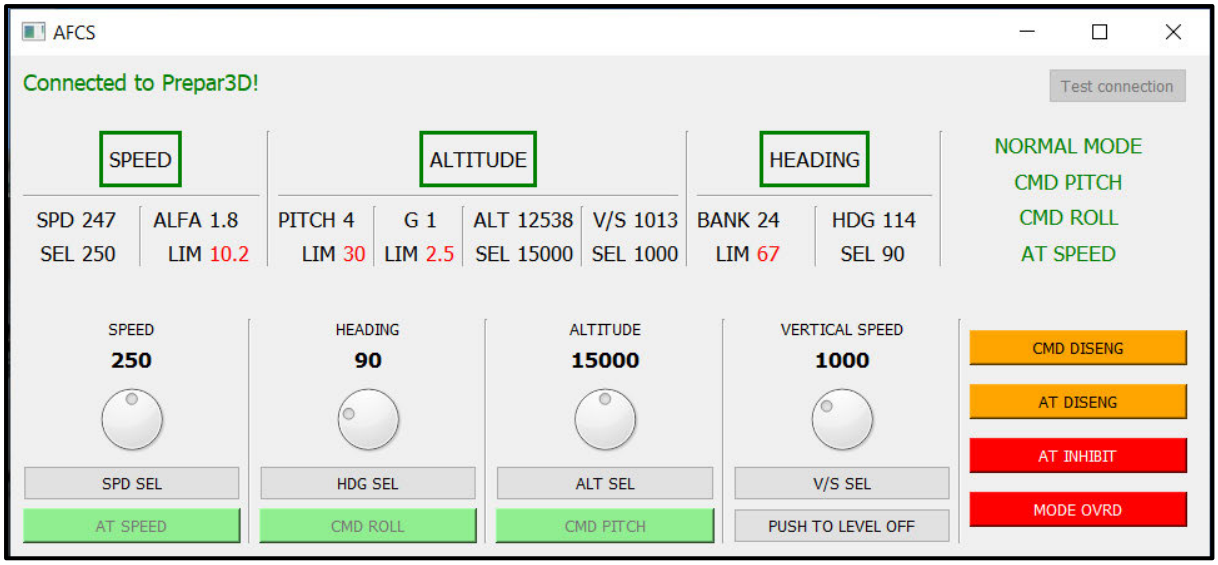


Figure 46: Indications in autopilot and autothrottle engaged state

In order to test the "PUSH TO LEVEL OFF" function, at an arbitrary moment the command is given into the mode control panel. Again, the response of the system is adequate: the current altitude is memorized, and a 500 feet per minute maximum rate is selected. Shortly thereafter, the aircraft will stabilize at the selected 14004 feet.

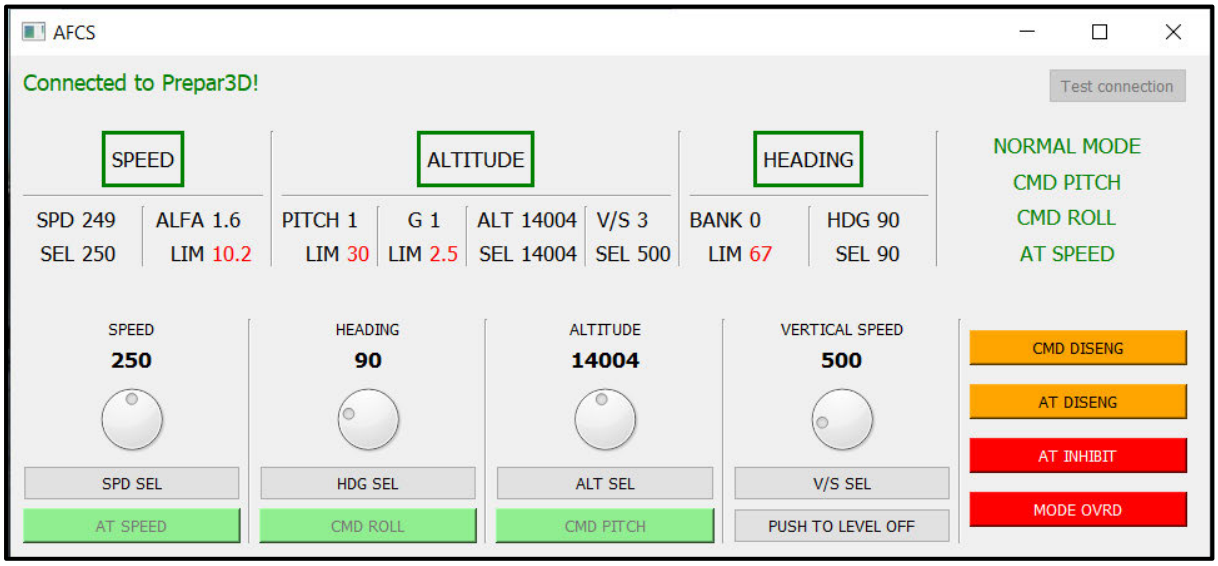


Figure 47: Push to level off function

An altitude of 15000 feet and a vertical speed of 1000 feet per minute are again selected in order to test the altitude capture mode. Although the speed drops by 5 knots, it is soon corrected and a vertical speed of 1000 feet per minute is attained immediately.

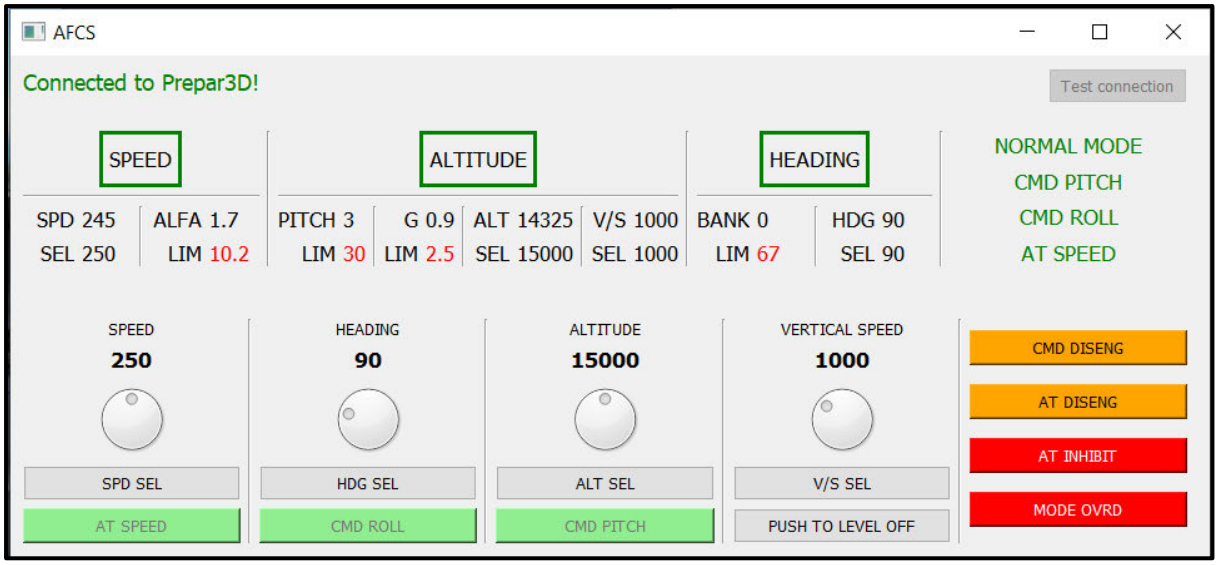


Figure 48: Climb to intercept 15000 feet

Arriving at 15000 feet, the system behaves as expected: thrust output is lowered in order to maintain 250 knots, and pitch is smoothly lowered, in order to attain and maintain 15000 feet. This value is attained with a precision of 1-2 feet.

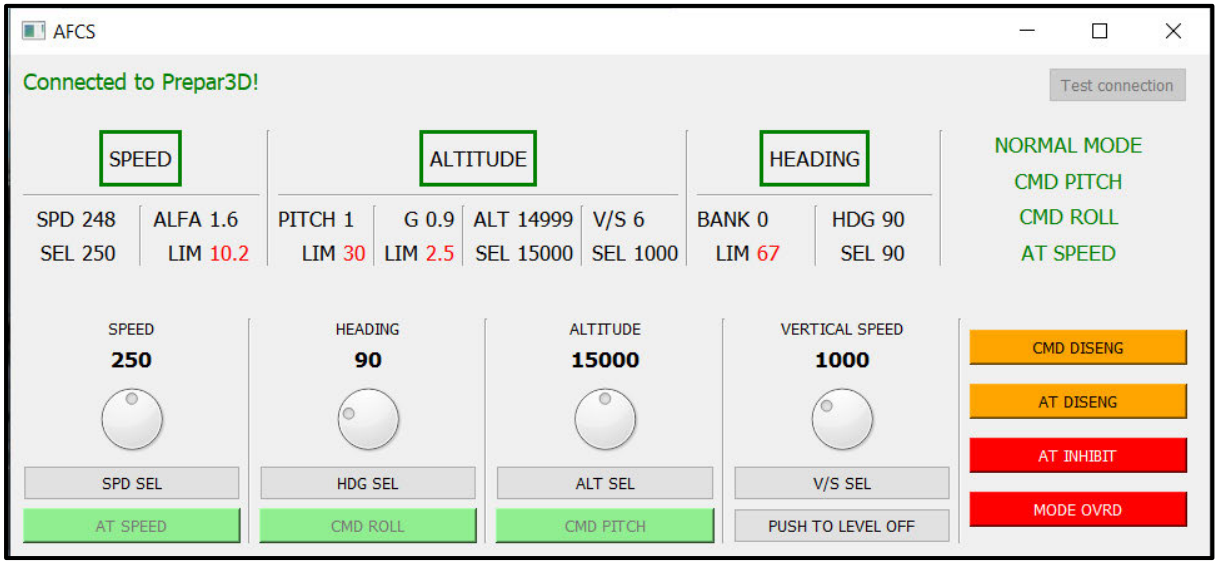


Figure 49: Altitude interception

In order to begin demonstrating the protection modules, a speed of 100 knots is selected in the mode control panel. Significant in this context is that a speed of 100 knots is insufficient in maintaining lift in a level flight condition at this thrust setting. Despite the rapid energy loss, the automatic flight control system succeeds in maintaining the altitude with the same precision.

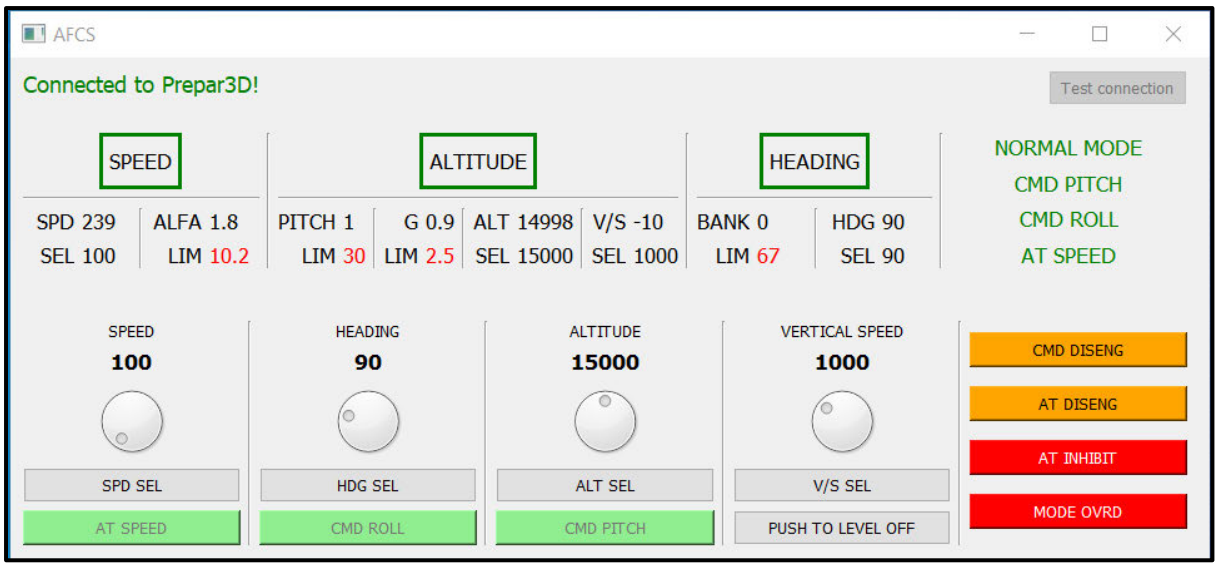


Figure 50: Speed is selected 100 knots

An angle of attack above the alfa protection engagement threshold is attained at a speed of 119 knots. At this point, all autoflight capability is temporarily and automatically inhibited. The engines are automatically commanded maximum available thrust in order to hamper the rapid decay of speed.

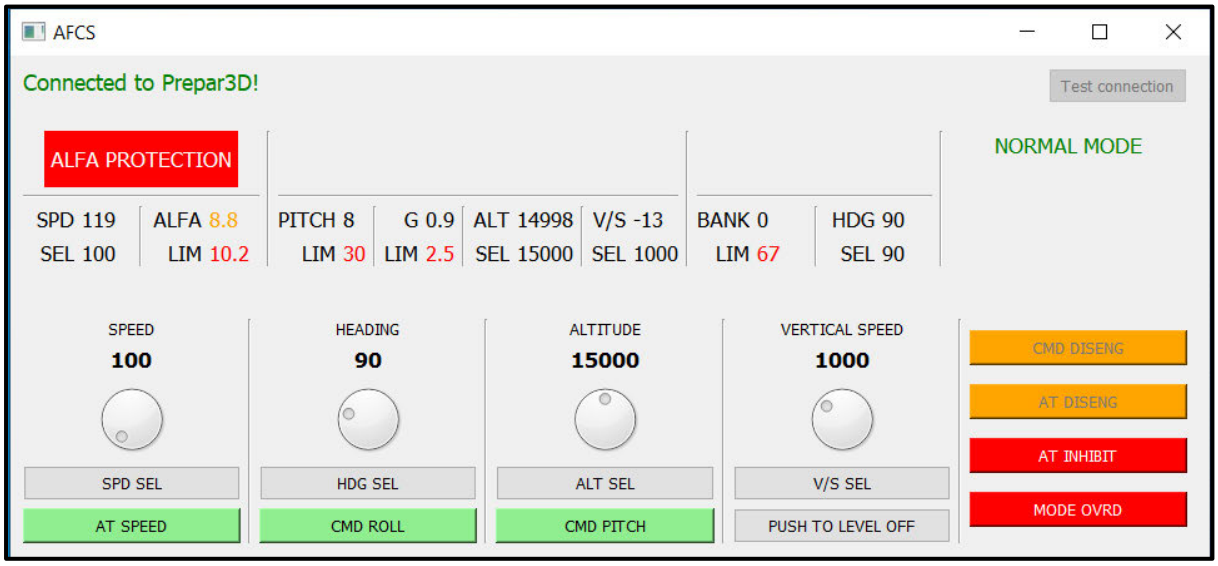


Figure 51: AFCS alfa protection activation

At this moment, if the control yoke is left to neutral, the angle of attack will naturally lower, while the engines at maximum power will encourage the speed to rise again toward the safe envelope. In order to prevent this, the control yoke is smoothly pulled toward the full aft position. The angle of attack will reach a value very close, but never in exceedance of the critical angle of attack. Coupled with the engines at maximum power will result in the aircraft climbing at the maximum attainable angle of attack.

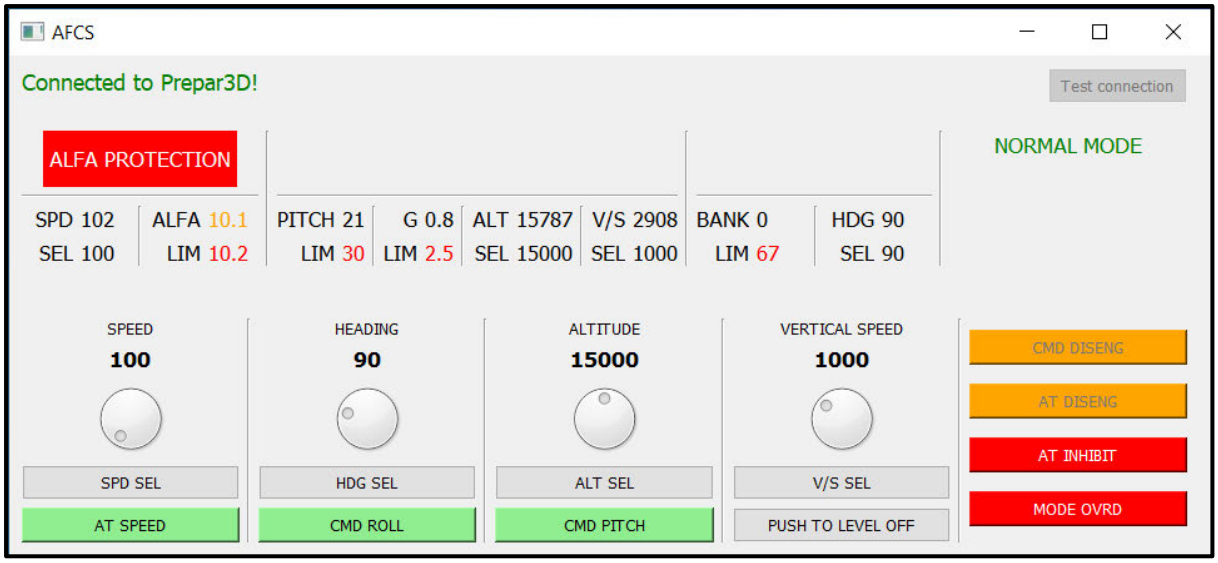


Figure 52: AFCS alfa protection is maintaining an angle of attack close to critical

In order to demonstrate the performance of the system under extreme circumstances, the autothrottle system is intentionally inhibited and the thrust output is intentionally reduced to flight idle. Moreover, the control yoke is violently pulled to the left mechanical stop. In a non-protected aircraft, such a maneuver would cause the aircraft to enter a spin and depending on the altitude a recovery would be impossible.

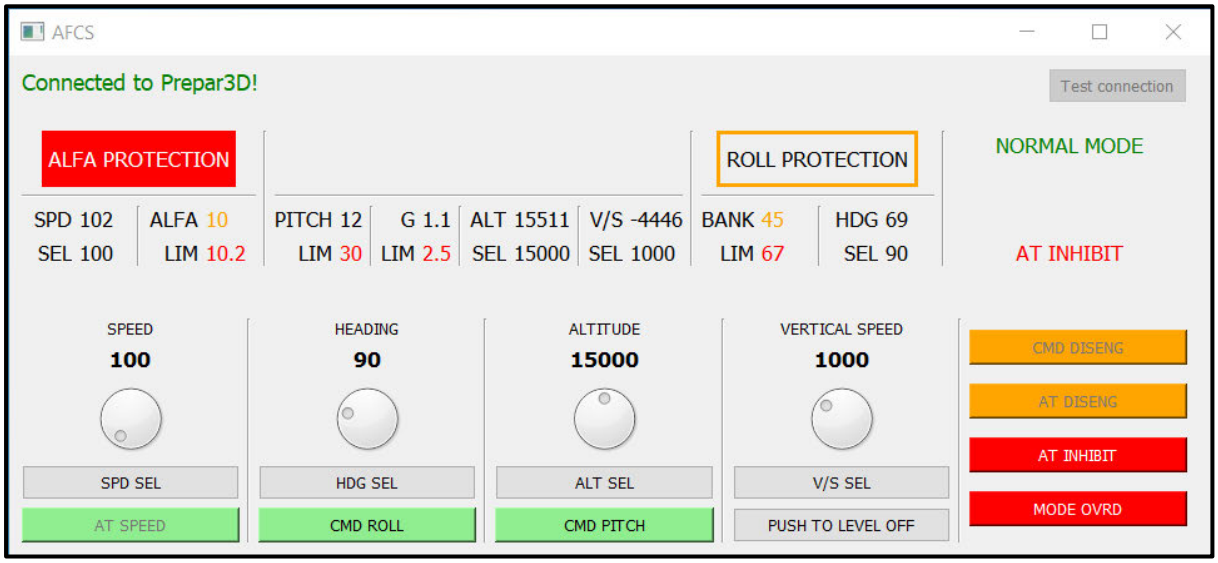


Figure 53: A violent left roll does not determine an exceedance of critical angles of attack

The behavior of the system is remarkable: a firm pitch-down moment is immediately introduced in order to prevent the aircraft from exceeding critical angles of attack. Moreover, the roll protection prevents the aircraft from exceeding bank angles at which lift would be impossible to be attained at this speed.

In order to demonstrate the rest of the protection modules, the automatic flight control system must be restarted. Maximum forward control yoke pressure is then applied. The system behaves as designed: it prevents the pitch from decreasing below 15 degrees airplane nose down even with full forward yoke pressure.

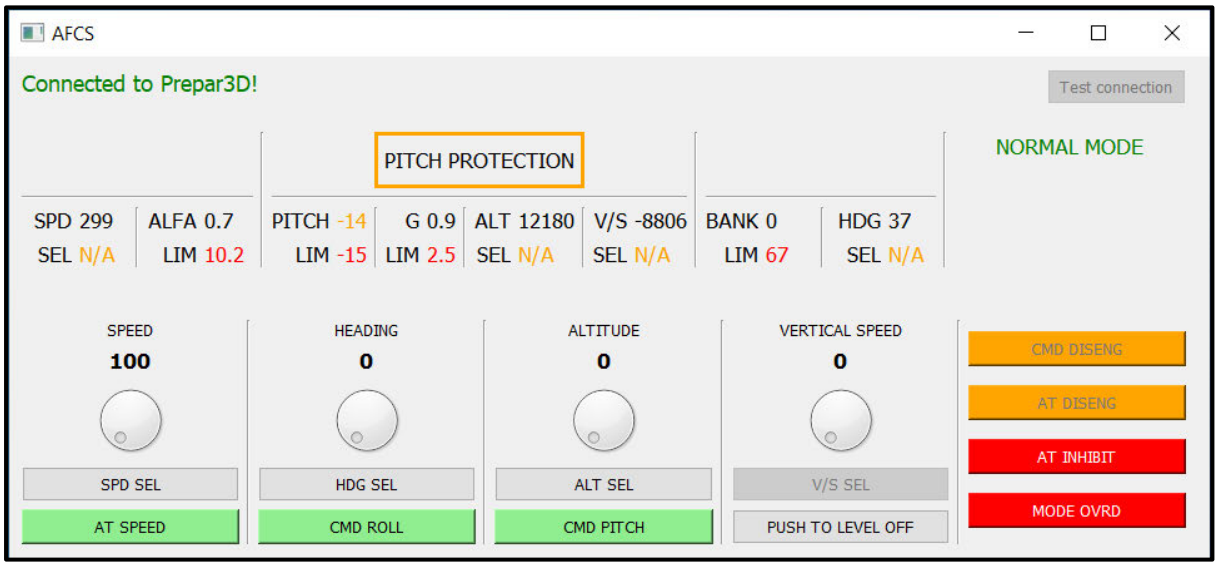


Figure 54: AFCS pitch protection activation

The speed will increase to the overspeed protection engagement threshold. At that moment, the autothrottle will command thrust output to flight idle and a pull-up in order to prevent the speed from further increasing to the never-exceed speed. Notice that, while the overload is above the admissible limit load, it is comfortably below the ultimate load.

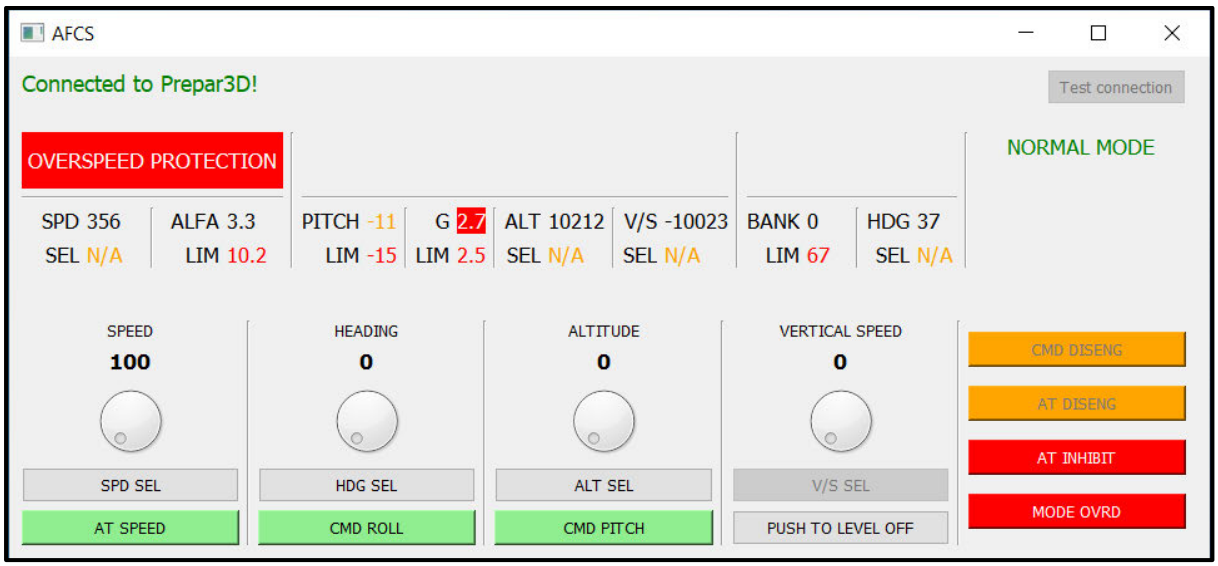


Figure 55: AFCS overspeed protection activation

In order to demonstrate maneuver protection, a dive is initiated from 12000 feet with the speed at approximately 280 knots. At an altitude of approximately 10000 feet and a speed of 310 knots an aggressive pull-up maneuver is initiated.

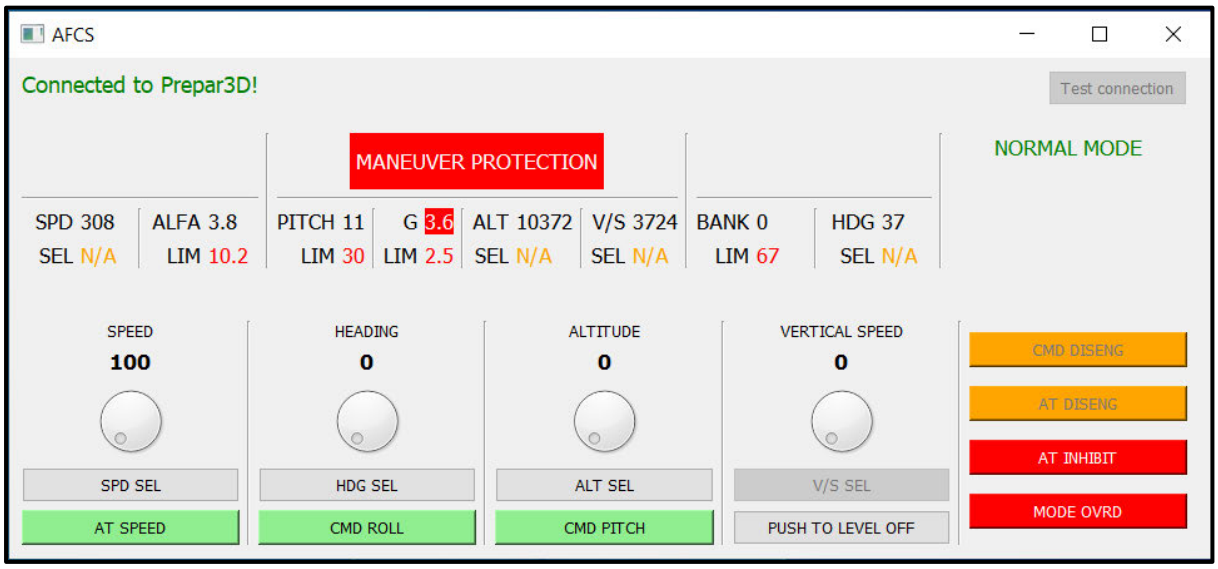


Figure 56: AFCS maneuver protection activation

Although temporarily the overload has almost reached the ultimate load, this condition will not last for more than a quarter of a second since the maneuver protection will unload the wings and bring the overload value back into the safe envelope.

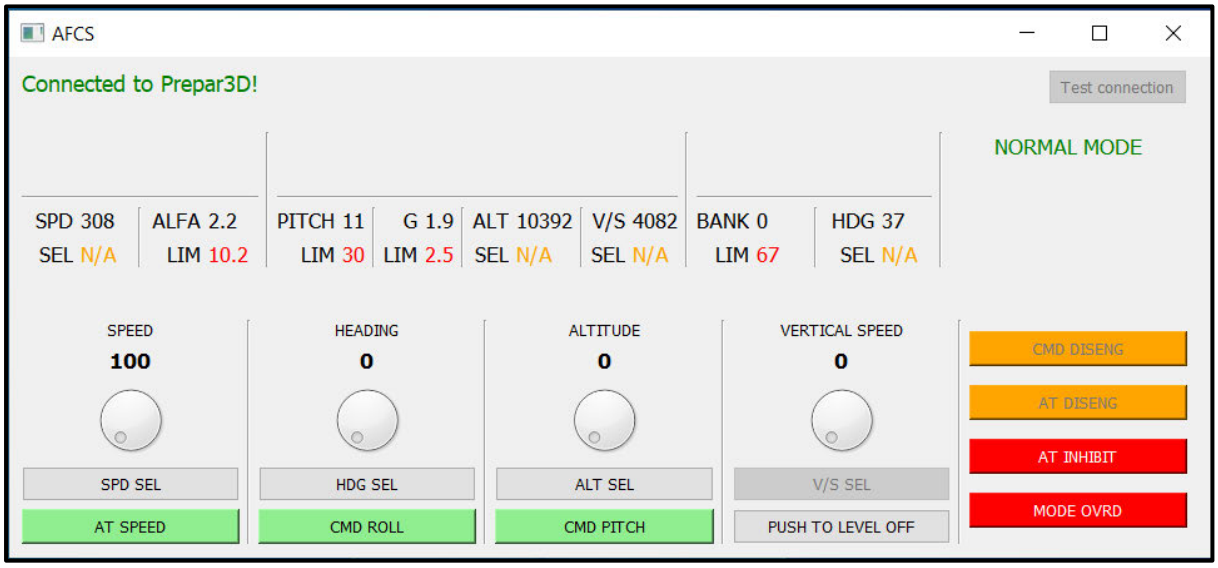


Figure 57: Overload decreases to values below limit load

Lastly, in order to demonstrate the reversion to direct mode, the pitch is brought to the pitch-up limit of 30 degrees. As the speed decays, the autothrottle is inhibited and the thrust levers are brought to the flight idle detent, in order to prevent application of maximum power. As the angle of attack reaches values close to critical angles of attack, sudden and full application of right rudder is performed. The effect is immediate, as the aircraft will flip over, entering a dive. Recovery must be manually made by the pilot by reverting to direct mode.

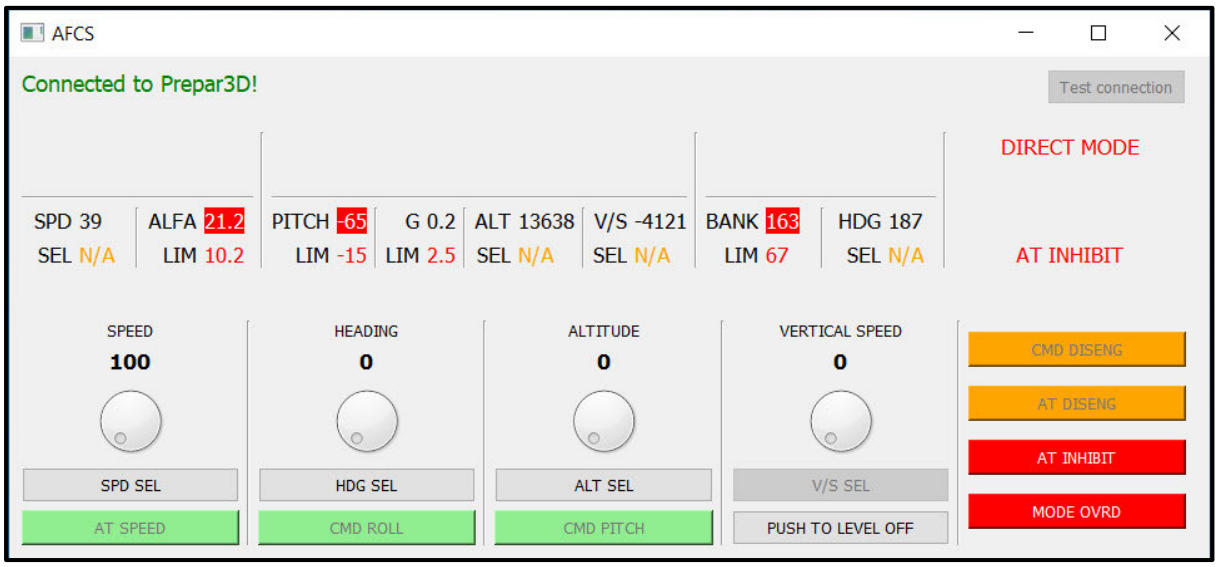


Figure 58: AFCS degradation to direct mode

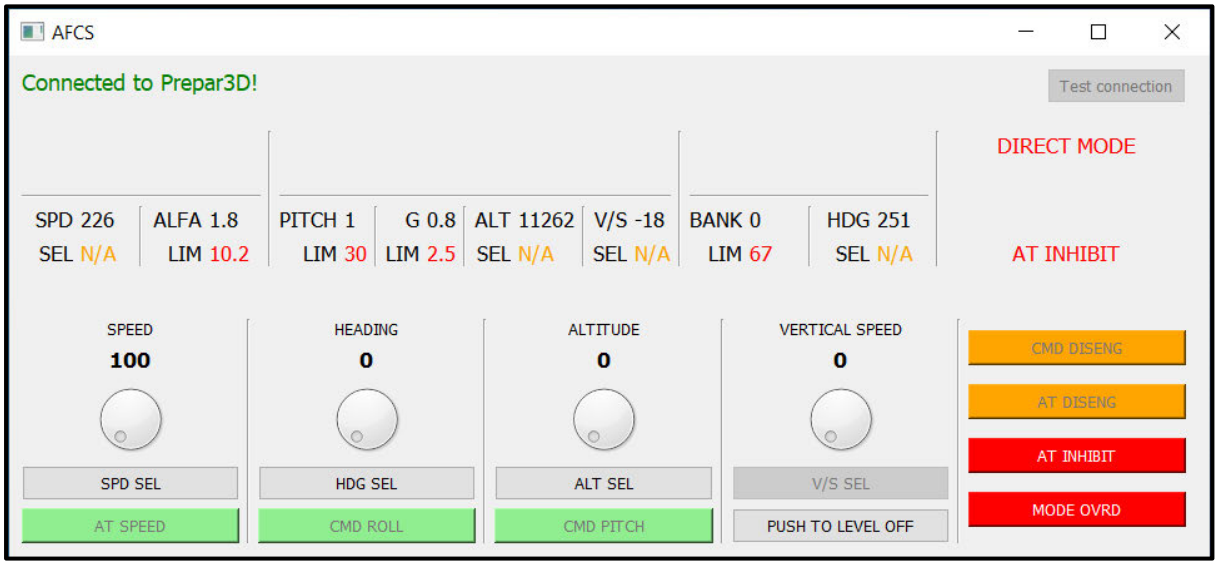


Figure 59: Upset recovery after direct mode degradation

Results and Discussion

The performance noted in the area of automatic flight is very good, as the speed is maintained with an average error of 1.5 knots, the altitude is maintained (with a proper stabilizer trim setting) with an average error of 1.5 feet. The vertical speed is also very precise, the average maximum error being 20 feet per minute.

Also to be mentioned is the notable performance under the condition of a rapidly changing parameter. It has been noted that despite a large energy decay, for example, or a sudden heading change (which causes the displacement of the lift vector, and thus a loss of altitude) the other parameters remain, after a short destabilization period (which is also small in amplitude) at the selected values.

The performance of the protection modules can be characterized as excellent, especially as flight close to the edge of the flight envelope requires extremely precise maneuvering if control is not to be lost. Likewise, during a high-speed descent, for example, if a terrain escape maneuver must be initiated, extreme care must be directed towards not exceeding the limit loads for a long period of time if the aircraft structure is not to be compromised. This can be especially difficult given the high stress levels associated with such a situation. The automatic flight control system assists the pilot in recovery from this undesired aircraft state, by providing him a means of performing an efficient maneuver by acting immediately, firmly and instinctively on the control yoke.

The most notable aspect is the maneuverability of the aircraft at elevated angles of attack. It must be noted that even in this flight condition gentle turns can be performed in order, for example, to avoid terrain. It is the author's opinion that flight this close to the edge of the flight envelope cannot be achieved without an extremely high risk of a loss-of-control in-flight of an unprotected aircraft.

Despite the excellent control characteristics through the automatic flight control system, an unforeseen situation which the system is not able to correct can happen at any time, and so the direct mode comes as an instrument which enhances the pilot's situational awareness in a highly stressful scenario.

Conclusions

This thesis aimed to describe the theoretical and practical aspects of implementing an automatic flight control system for the Lockheed Martin Prepar3D platform. Based on testing, it can be concluded that such a system has a great potential in both reducing workload during line operations and enhancing pilot performance while under abnormal conditions.

Although research has been directed in this area as aircraft manufacturers sought to digitalize aircraft control in view of efficiency, little attention has been paid to an accessible resource on this matter that can be further studied on or adapted for use in a real aircraft.

The most challenging aspect in developing such a system is always maintaining a view on redundancy and realizing that in no way can every possible flight scenario be thought of. Thus, the separate modules aim to enhance the effect of each other, or put in other words, prevent a situation from which another module may not be able to recover the aircraft from.

This being said, it is to be stressed that it is always the pilot who has final authority over the aircraft flight path, implying his responsibility of ensuring that the aircraft is operated within the flight envelope at all times. The automatic flight control system is designed to allow for an easy transition to fully manual control if any situation that imposes may arise.

Another challenging aspect was determining the ratio between parameter error and surface displacement output within the PID controller. Unfortunately, no available tool could allow for an objective determination of this ratio, and so all values were obtained experimentally. Further research may develop in the area of control theory in order to provide a means of determining this ratio objectively.

Further research may also develop in the area of aircraft controllability using thrust output only (for example, after a complete loss of the hydraulic system, or a total degradation of manual control capability). It is achieved by use of differential thrust for controlling the aircraft in yaw (and indirectly, in roll), and applying or removing power from an underwing-mounted engine for control in pitch (this is due to thrust vector effect).

In closing, it must be said that a successful safety culture within the cockpit fundaments itself onto rigorous training, onto which a layer of research in the field of enhancing safety by electronic means is placed. However, under no circumstance shall an electronic system replace a strong training culture within an organization, or else the purpose of the system can be considered as defeated.

References

[1] Sarter, Nadine, Woods, David: *Team Play with a Powerful and Independent Agent: Operational Experiences and Automation Surprises on the Airbus A-320*, Human factors, 39 (1998), 553-569.

[2] Ryan, Craig: *The Pre-Astronauts: Manned Ballooning on the Threshold of Space*, Naval Institute Press, 2003, ISBN 978-1591147480.

[3] Gray, Carroll: *The First Five Flights*, WW1 Aero – the Journal of the Early Aeroplane, 117 (2002), 26-39.

[4] Newton, Isaac: *Philosophiae Naturalis Principia Mathematica, Book II*, 1726.

[5] Sutherland, J. P.: *Fly-by-Wire Flight Control Systems*, Wright Patterson Air Force Base, Ohio, 1968.

[6] Gawron, Valerie, Jones, Patricia: *Airplane Upset Training Evaluation Report*, National Aeronautics and Space Administration, 2002.

[7] Pasztor, Beth: *Statistical Summary of Commercial Jet Airplane Accidents: Worldwide Operations | 1959 – 2018*, Boeing Commercial Airplanes, 50 (2019).

[8] Soerjanto, Tjahjono: *Final Aircraft Accident Investigation Report on the PT. Lion Metari Airlines Boeing 737-8 (MAX) PK-LQP in Tanjung Karawang, West Java, Republic of Indonesia, on the 29th of October, 2018,* 2019.

[9] Moges, Dagmawit: *Interim Investigation Report of accident 737-8 MAX ET-AVJ, ET-302*, 2020.

[redacted]

[11] WILJAM FLIGHT TRAINING: *Principles of Flight.*

[12] Pinheiro, Rafael, Colón, Diego: *An Application of Lurie Problem in Hopfield Neural Networks* (2017).

[13] Huang, Zhen, Tang, Chengcheng, Dinavahi, Venkata: *Unified Solver Based Real-Time Multi-Domain Simulation of Aircraft Electro-Mechanical-Actuator. IEEE Transactions on Energy Conversion* (2019).

[14] Bittar, Adriano, Oliveira, Neusa, Figueiredo, Helosman: *Hardware-In-the-Loop Simulation with X-Plane of Attitude Control of a SUAV Exploring Atmospheric Conditions. Journal of Intelligent & Robotic Systems* (2014).

[15] Leigh, J. R.: *Control Theory, 2nd Edition*, The Institution of Engineering and Technology, United Kingdom, 2008, ISBN 978-0-86341-339-1.

[16] Åström, K. J., Hägglund T.: *PID Controllers: Theory, Design, and Tuning, 2nd Edition*, Instrument Society of America, United States of America, 1995, ISBN 1-55617-516-7.

[17] Buyya, Rajkumar: *Java Multithreaded Programming*, http://www.buyya.com/java/Chapter14.pdf

[18] Lazar, Guillaume, Penea, Robin: *Multithreading with Qt*, https://hub.packtpub.com/multithreading-qt/

[19] The Qt Company: *Qt Documentation: Signals & Slots*, https://doc.qt.io/qt-5/signalsandslots.html

[20] Lockheed Martin Corporation: *Prepar3D Product Overview*, https://www.prepar3d.com/product-overview/

[21] Lockheed Martin Corporation: *Prepar3D SDK*, https://www.prepar3d.com/SDKv4/sdk/sdk_overview.html

[22] Lockheed Martin Corporation: *Prepar3D SDK Simulation Variables*, http://www.prepar3d.com/SDKv3/LearningCenter/utilities/variables/simulation_variables.html

[23] Lockheed Martin Corporation: *Prepar3D SDK SimConnect*, http://www.prepar3d.com/SDKv3/LearningCenter/utilities/simconnect/simconnect.html

[24] Lockheed Martin Corporation: *Prepar3D SDK General Functions*, https://www.prepar3d.com/SDKv4/sdk/simconnect_api/references/general_functions.html

[25] Asobo Studio SARL: *Microsoft Flight Simulator Feature Discovery Series Episode 3: Aerodynamics*, https://www.youtube.com/watch?v=Bw-opH4f8Qg