NAZARBAYEV UNIVERSITY

School of Engineering and Digital Sciences

Bachelor of Engineering in
Mechanical and Aerospace Engineering

Interactive Airplane Controller Simulator with Different Control Algorithms

(Final Capstone Project Report)

by

Darkhan Zhangeldi, Bekzat Omirzak, and Telgara Beisenbayev

Lead Supervisor: Prof. Altay Zhakatayev

May, 2025

Abstract

The paper outlines the building process and the development of the airplane simulator that is based on the webpage and can be manipulated with the parameters that are chosen by the user. The description of the project is set by the use of JavaScript, CSS and HTML to make a site that is functional and can present a computational system model. The main focus of the project model lies in the demonstration of the dynamic behavior of the airplane and the change that it demonstrates based on the controller and its inputs. The JavaScript demonstrates the dynamic equations and changes perfectly in a visual graphs and animations with the real-time user interactions. The language choice was determined on the criteria as to be supported on the different devices, simple in understand and smooth in processing. Although, the MATLAB can be seen as an existing tool to educate the young minds, still not all educational systems has funding to educate the students using the licensed products. Despite the fact that there are simulations that can be available online, they lack the complex motion design and are focused on the interface and the different models of the aircraft rather than on the dynamic motion. The simulation that is described in the paper has been based on the three control methodologies of the modern control theory. A proportional-Integral-Derivative controller, Linear Quadratic Regulator and the Full State Feedback. PID can show how Kp, Ki and Kd gains affect the trajectory and stability of the airplane to reach the certain pitch angle. The tuning curve will present the variations of error within the set time as the user starts the simulation. As the FSF and LQR can be manipulated by the matrix parameters to further demonstrate the dynamic model, users obtain an important knowledge of the working principle of the control algorithms. The simulations are entertaining and with time the simulation's behavior becomes intuitive as the users repeat certain matrix combinations. Such method allows user to visually check the performance of any simulation that is controlled by any algorithms. Hence, the practice can be extended from airplane simulator to any automation processes in the industry, making the virtual training that is safe, accessible and entertaining.

Contents

Abstract i

Contents iii

List of Figures v

List of Tables vii

1 Introduction 1

1.1 Background 1

1.2 Literature Review of Controllers 2

2 Problem Statement 9

3 Methodology 13

3.1 MATLAB 13

3.2 **C++** 13

3.3 **JavaScript** 14

3.4 Roll and Yaw control 16

3.5 **Web-hosting** 18

4 Results 19

4.1 Pitch Controller Results 19

4.2 Roll and Yaw control 26

4.3 User Feedback 31

Bibliography 37

Appendices 39

A Source Code 47

List of Figures

1.1 Simple block design on the Full State Feedback Control 5

4.1 *LQR Model for Optimal K precomputed* 19
4.2 LQR with a manual setup and Ricatti solver – a Boeing 777-300ER case . 21
4.3 FSF – an Airbus A330-900 case 23
4.4 *PID Pitch Tuning Curve at Kp=0.5, Ki=0.1, Kd=0.2* 25
4.5 *The LQR Controller website page of the roll control* 28
4.6 *The FSF Controller website page of the yaw motion* 29
4.7 *The PID Controller website page of the roll motion* 30
4.8 *User ranking of the most effective controller* 31

A.1 *LQR Yaw Control Website Page* 52
A.2 *FSF Roll Control Website Page* 52
A.3 *Consent of participation* 53
A.4 *Device type* 53
A.5 *Familiarity with the aircraft control theory from 1 to 5* 53
A.6 *Rating the ease of navigation through the website* 54
A.7 *Rating the website loading* 54
A.8 *Rating the user interface of the website* 54
A.9 *User ranking of the website's functionality* 55

List of Tables

1.1 Determination of Parameters 3

4.1 The 3 controller test with the desired angle of 45° on the Airbus A330-200 29
4.2 The Lateral Derivatives of the general aviation airplane (Nelson, 1998) . 31

Chapter 1

Introduction

1.1 Background

Growth in the last decades, both in the fields of aerospace engineering and control systems, was nothing short of amazing, in fact brought by an ever-increasing request to enhance precision, safety, and efficiency in flying. At the same time, emerging web-based simulations have revitalized quite a few traditional education methods toward becoming interactive and engaging tools for students and professionals alike. In particular, web-based airplane control simulations open a thrilling avenue for understanding in depth the intricacies of flight dynamics and associated control algorithms. This interactive experience engenders a more meaningful and practical understanding of aircraft behavior. The importance of the controllers lies in the maintaining the stable path of flight using the control algorithms of the complex autopilot that alleviates any disturbances both internal and external. Among all control strategies, PID and LQR, and FSF controllers can be considered some of the fundamental methodologies in modern control theory. Each possesses different strengths and applications that make the algorithms invaluable tools for education simulations aimed at illustrating important principles and practical advantages for application. For instance, PID controllers are known for their simplicity and wide range of applications, while LQR controllers excel in their capability to optimize a certain cost function optimally. FSF control further elevates these systems by effectively predicting disturbances that may come, thus giving even better stability and responsiveness. Advanced simulations of aircraft dynamics have built a concrete platform for further understanding several control algorithms. Historically, aircraft simulation software has run mostly on desktop systems, often requiring special hardware and software. Tools such as MATLAB and Simulink are widely accepted both in the academic and industry sectors for their powerful capabilities in modeling and simulating flight dynamics. While these platforms indeed have strong simulation capabilities, they have often posed an accessibility challenge: installation and licenses, plus steep learning curves for some novices. Web integration dramatically changed this landscape: simulations became universally usable, and learning was truly democratized in all parts of the world. While most desktop solutions provide poor device independence, eliminate headaches with regard to installation, and reduce the collaborative learning effects by real-time interaction and cloud storage, web-based simulations take precedence. The development of an interactive, web-based airplane control simulator is where technology and education meet. This project focuses on

the simulation of an airplane's dynamics relative to pitch, roll, and yaw using control algorithms such as PID, LQR, and FSF control. The simulation initializes the modeling of the pendulum system in MATLAB-a simplified model airplane pitch motion that allowed for some initial establishment and robust testing of control algorithms under controlled conditions. The project then moved into JavaScript for its deployment on the web, using the versatility and compatibility of the language with modern web browsers. This progression underlines the critical need for intertwining theoretical modeling with its application in the real world, as the simulation needs to stay both precise and user-friendly. The importance of control algorithms in aviation and applications that have been made in education simulations are well supported by the literature. Among these, PID controllers are one of the most studied and widely applied control algorithms in many industries, including in aerospace applications. Nise, 2021. They propose a simple way to obtain the best system performance using adjustments of the proportional, integral, and derivative gains. Besides this, PID controllers can have some problems dealing with nonlinear systems and disturbances, for which the use of more advanced techniques is frequently necessary. LQR controllers overcome these by optimizing a cost function involving both system states and control efforts, hence very useful in applications that demand high precision and control efficiency, such as aircraft stabilization and navigation.

1.2 Literature Review of Controllers

1.2.1 Proportional Integral Derivative (PID)

The study conducted by Borase et al. (2020) gives a descriptive study on the PID tuning and the creation of control systems within the industries. Depending on the application, a PID controller can take two forms: Parallel type: Derivative and integral parameters are independent, along with the proportional parameter and the result of each parameter is added to the resulting gain as following:

$$u(t) = K_c \left(e(t) + \frac{1}{T_i} \int e(t)\, dt + T_d \frac{de(t)}{dt} \right) \tag{1.1}$$

proportional gain $K_c = K_p$, following $k_d = K_c T_d$ derivative gain. The T_d that is the represented time derivative. $k_i = K_c/T_i$ represents the gain of integral, while T_i is the integral time. Series type: This series of interactive equations emphasizes the characteristics of pneumatic and analog electronic circuits. Similar to a typical PID equation, altering the proportional gain K_c influences all control actions, while the derivative and integral constants also play a role in the proportional response. The systematic equation is:

$$e_1(t) = e(t) + T_d \frac{de(t)}{dt} \tag{1.2}$$

$$u(t) = K_c \left(e_1(t) + \frac{1}{T_i} \int e(t)\, dt \right) \tag{1.3}$$

Controller	K_p	T_i	T_D
P	1/a	-	-
PI	0.9/a	3.33 θ	-
PID	1.2/a	2 θ	0.5 θ

Table 1.1: Determination of Parameters

Adjusting T_d generally affects D and P actions, while changes to T_i influence P and actions of the component I. In contrast, altering K_c has an influence over all actions. Borase et al. (2020) describe the basic tuning methods for PID as the Trial and Error Method, the Ziegler-Nichols Step Response Method, the Ziegler-Nichols Frequency Response Method, the Relay Tuning Method, and the Cohen-Coon Method. PID formulas and tuning methods were investigated and researched by Weng Khuen Ho to evaluate the efficiency of Ziegler-Nichols and the method of Cohen-Coon. Overall the main idea behind all these methods lie within the the range that starts from 0.1 and goes up to 1 where the dead time constant stays. The summary of their key findings are that the Integral Absolute Error (IAE), Integral Time Absolute Error (ITAE) and Integral Square Error (ISE) calculations tend to be close to the gain margin of 1.5 any time when the load is applied. The margins of the phase increase from 30 degrees up to the 60 degrees as soon as dead time constant reaches the 1 from 0.1. Such tuning can be further optimized where the phase margin reaches 65 degrees while the margin of gain is close to 2. This can be done by setting the better set point. These formulas often rely on the PID controller zeros to counteract process poles. While classical tuning methods are straightforward to apply and computationally efficient, they rely on specific assumptions about the system and desired outcomes. These methods are particularly useful during the initial stages of tuning but may require additional adjustments to achieve optimal results due to the limitations of their assumptions.

Similarly, Bansal et al. (2012) wrote a review on PID Controller Tuning Techniques supporting that PID controllers are widely used within the industry systems, up to 90 percent. The study provides a deeper understanding of the algebraic computations and key formulas that influence the overall output of the system and the tuning. To optimize the tuning, to even set the tuning to starting point, the traditional methods make several assumptions regarding the output that put to be reached and the plant itself. Such techniques allow to visually assess certain parameters and start the controller by setting the initial guessed settings. Such iterations then result in the close values that are more practical than the first guess. Though, such methods often lead to the result that meant to be reached, but rather needs the tuning to go to the desired outcome. Therefore, the paper describes the

computational optimization techniques.

$$\text{IAE} = \int_0^t |e(\tau)|\, d\tau \tag{1.4}$$

$$\text{ISE} = \int_0^t [e(\tau)]^2\, d\tau \tag{1.5}$$

$$\text{ITAE} = \int_0^t \tau|e(\tau)|\, d\tau \tag{1.6}$$

$$\text{ITSE} = \int_0^t \tau[e(\tau)]^2\, d\tau \tag{1.7}$$

Despite the advantages of the PID controller, such as adjustments that are manipulated easily and the design simplicity, it is limited only to the attitude control. While the airplane requires a much more complex system for the nonlinearities it encounters. Therefore, the different control systems should be presented for an apparent comparison of such phenomena. Linear quadratic regulators can represent the implicit results of the aircraft trajectory by minimizing the suitable cost function. The trajectory of the airplane is not disturbed by external factors such as wind and freezing temperatures. Therefore, the complete model of the airplane can be analyzed by LQR (Zulu and John, 2014).

1.2.2 Full State Feedback

Ogata (2020) states that the principle of state variable feedback in controlling the dynamics of a system rests on using the state variables that at any moment carry within them all the information of its behavior. Feeding back the state variables or their estimates into the control input, an engineer can adjust the performance of the system for stability, precision, response time, and other purposes. This technique is rooted in state-space representation, which models a system using matrices that describe the relationships between its state, inputs, and outputs. In this context, feedback control is achieved by applying a control law that adjusts the input to the system based on the current state, often represented as

$$u(k) = -Kx(k) + v(k) \tag{1.8}$$

If there are some of these state variables that cannot be measured directly from the system, then a process called state estimation arises. Observers or estimators make use of measurable system outputs to reconstruct the unmeasured state variables, or estimates thereof, which may include techniques like Luenberger observer and Kalman filters. This permits feedback control to be used when part of the state is not accessible. What is important to note is that for any functioning and efficient control system, part of the state information should be used.

State variable feedback is very applicable, considering that an aircraft can be very complex and highly dynamic in its systems. One has states interacting with others such

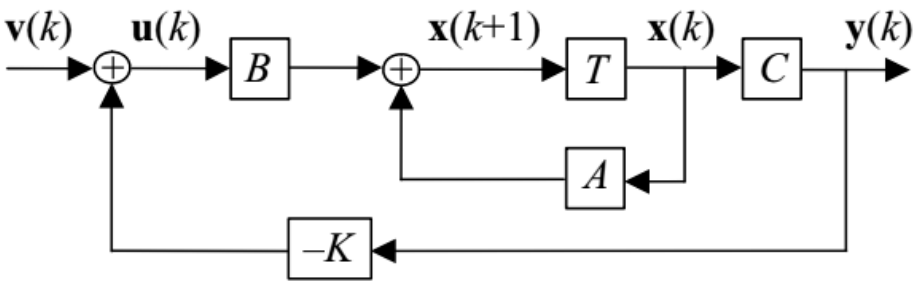

Figure 1.1: Simple block design on the Full State Feedback Control

as pitch, roll, yaw, velocity, and altitude to carry out control in an airplane. This would ensure that the simulator gives reasonably close real-world dynamics to be able to use the simulator for pilot training, control strategy testing, and the study of possible flight behaviors under different conditions.

Some of the major applications committed to an airplane simulator involve stability and control of the flight. Controlling a certain pitch angle during pitch-up, for example, state variable can be shown in use for the light takeoff simulation of autopilots or even for an aircraft performing dynamic maneuvers. Therefore, one is able upon incorporating state estimators to decide on how to work down real-world complexities of low sensor noise or unprepared variables. Realistic skills training will also be provided onto this simulator. This principle also forms a basis on which advanced techniques, such as PID, LQR, and adaptive control, can be studied in detail and compared, enlightening engineers and students on how to understand such techniques and refine their applications in aircraft. The mathematical foundation of state feedback is built on state-space equations, where the system's behavior is totally controlled by u vector and the resulting vector. The feedback is controlled by the following formula state: $u(k) = -Kx(k) + v(k)$. X vector is dictated by the matrix K, where that gains are manipulated and the input that is not internal, but rather external value of the v. Incorporating this feedback law into the system yields the equation of the state that is the controlled closed loop. $x(k+1) = (A - BK)x(k) + Bv(k)$. Matrix $A_{cl} = A - BK$ governs the system dynamics.

The design of K allows the engineer to shape the closed-loop dynamics by selecting the eigenvalues and eigenvectors, determining the system's stability and response characteristics. However, the practical challenge arises when some state variables cannot be directly measured. In such cases, state estimators, often based on output measurement, are designed to approximate the full state vector. These estimates are then used in place of the actual state vector for feedback control.

1.2.3 Linear Quadratic Regulator (LQR)

The Linear Quadratic Regulator is a precise and accurate method yet complex that is widely used in engineering systems to achieve a balance between system performance and control effort. It is particularly suited for linear dynamic systems, where it minimizes a quadratic cost function over time. The control strategy uses feedback from the system state to compute the optimal control input.

$$\dot{x} = Ax + Bu \tag{1.9}$$

A and B are matrices that are already known and set, resulting in the outcome of the process state.

$$u(t) = -Gx(t) \tag{1.10}$$

The purpose of such system is to determine the exact gain minimizing the function resulting in a specific performance criterion, V, known as a "cost function." Regulater optimizes the system instead of forcing the poles that are closed-looped to the desired outcome. This criterion is expressed as the integral of a quadratic form in the state and another quadratic form. In other words, a mathematical measure is used to define performance rather than directly specifying pole placement.

$$V = \int_t^\tau \left[x'(\tau)Q(\tau)x(\tau) + u'(\tau)Ru(\tau)\right] d\tau \tag{1.11}$$

The equation provides the symmetric matrices as it can be seen R and Q. Here, Q represents weighting matrix of the state and space, while R is weighting matrix of the input control. These matrices assign importance to specific components of the state and control effort, respectively.

The problem of minimizing V is unaffected by scaling it with a positive constant, such as $\frac{1}{2}$, which simplifies subsequent calculations. When the terminal time T is infinite, the focus shifts to steady-state behavior, making this scenario particularly relevant to long-term system performance. For finite T, the control interval decreases as it $T - t$ approaches zero, at which point the process concludes. Despite its mathematical formulation, the true design objective often cannot be captured fully by the quadratic performance criterion. In cases where the real objective is too complex or unsolvable, this approach offers a practical alternative. By simplifying the problem into one that can be solved, LQR provides a way to achieve near-optimal control while managing the limitations of mathematical representation (Fredland, 1986).

1.2.4 Existing virtual laboratories

Balamuralithara and Woods (2008) stated that the integration of web-based simulators into aviation education and research has seen significant growth, driven by advancements in communication technology and the increasing need for accessible and interactive learning

platforms. Such simulators are pivotal in providing students and researchers with a practical understanding of control systems without the need for expensive physical setups. This review explores the role of web-based simulators in understanding and experimenting with different airplane control systems, focusing on their benefits, methodologies, and applications.

Control systems in aviation, including pitch, roll, and yaw control, are critical for ensuring stable and precise aircraft operation. PID controllers with the traditional methods are common in applications within the several industries as they are not only effective, but also their simple use and the interpretation regulates the dynamical system. Modern advancements have introduced adaptive, robust, and model-based control systems, such as Linear Quadratic Regulator. Web-based simulators provide an accessible platform to study, compare, and analyze these systems without the complexities and costs of real-world test. The evolution of web technologies has enabled interactive and scalable simulations. Tools like JavaScript, WebGL, and frameworks such as Streamlit and Flask allow developers to create visually rich and highly responsive applications. Such platforms are not only accessible globally but also support real-time user interaction, critical for aviation control system experimentation (Balamuralithara & Woods, 2009). For instance, MATLAB/Simulink has been extensively used to model dynamic systems and can be integrated with web-based platforms for enhanced accessibility (MathWorks, 2023). Cost-Effectiveness compared to physical simulators, web-based systems significantly reduce costs while providing robust virtual experimentation capabilities. Scalability and accessibility of web-based simulators can be accessed globally, enabling learners from diverse locations to experiment with control systems in real time. Safety and Risk Mitigation of virtual environments eliminate risks associated with physical testing, particularly in high-stakes domains like aviation. Interactive learning By simulating dynamic airplane behaviors, these tools enhance conceptual understanding and practical skills through real-time adjustments and visual feedback.

While web-based simulators offer substantial benefits, challenges such as fidelity to real-world dynamics, latency in user interactions, and the need for advanced computational resources persist. These limitations, however, are being addressed with the integration of high-performance computing and AI-driven modeling approaches. The development and application of web-based airplane simulators for different control systems represent a significant step forward in aviation education and research. These platforms not only make learning and experimentation more accessible but also provide a safe and cost-effective alternative to physical setups. Continued advancements in web technologies and computational modeling are expected to further enhance the fidelity and functionality of these simulators, solidifying their role as indispensable tools in aviation and control systems education.

1.2.5 Educational benefits

Research has indicated that applying knowledge in realistic contexts is crucial for developing advanced skills. Studies of the development states that the students that were exposed to the virtual environment while learning can attain a strong understanding of the concept as the interact with the system. Such repetitive exposure suggests that the learners can sharpen their problem-solving skills and strengthen the foundation of their knowledge. Ideally, these practice opportunities should involve real-world problems relevant to a specific professional field. However, higher education programs often limit students' chances to address real-life problems directly. Moreover, application of such technologies solve the issue with the mental preparation of the learners. It is challenging to navigate through the real-world problem without guidance first time. Furthermore, practical knowledge or application of such knowledge can be limited as the situations with the critical state can be rare within the industries. Therefore, virtual preparation is an effective way of testing and educating the students in a safe and controlled manner.

As the first step towards the learning process the reduced-complexity practice approaches are implemented to engage learners in specific facets of professional practice, helping to clarify concepts and efficiently allocate learning resources. Such approximations can be achieved through simulations in higher education, enabling students to tackle authentic problems while cultivating an environment conducive to practicing and developing complex skills. Simulations are becoming increasingly prevalent in higher education settings. In STEM fields (science, technology, engineering, and mathematics), they enhance understanding of concepts and their interrelations while promoting inquiry, problem-solving, and decision-making. A substantial amount of research has focused on medical education, where simulations are employed to improve diagnostic skills and technical abilities in future doctors, nurses, and emergency responders. Simulation-based learning is also utilized in areas such as teacher education, engineering, and management.

This meta-analysis aims to explore higher education, particularly in disciplines that involve significant interpersonal interaction (physical, cognitive, social, etc.)—such as medicine, nursing, psychological counseling, management, teacher education, certain engineering fields, and economics. There is limited understanding of which learners benefit most from simulations, the effectiveness of various scenarios, and what additional instructional support enhances the effectiveness for learners with different backgrounds. There is a lack of synthesized findings on how different simulation features (like duration and technology) and instructional support (such as scaffolding) influence effectiveness for students (Ramos et al., 2020).

Chapter 2

Problem Statement

Accuracy, precision, control and comprehension of the systems are crucial in aerospace engineering. In the educational system, especially to the engineering students that are interested in control theory, equipping them with the understanding is essential component within the future specialists. maintaining a safe and stable flight is the main purpose of the controllers. However, modern control theory may be confusing for young scholars as the theoretical knowledge can be challenging to visualize the immediate outcome of any change in the parameters of the aircraft control framework. Matlab is the irreplaceable tool in education of the young engineers. High functional and powerful simulations allow the students to take a look at the Proportional-Integral-Derivative (PID), Linear-Quadratic Regulator (LQR), and Full-state Feedback (FSF) in a more creative ways. However, the high cost of such powerful tool often comes with the accessibility issues among the people with different financial backgrounds, especially in resource-poor educational systems, where the license for such products are not provided by the university. Besides, most of the students may struggle with the coding part as their programming background may not be sufficient to navigate through MATLAB. Such obstacles limits the use and integration of MATLAB to the community, particularly in the developing countries. Furthermore, existing simulations are fixed and cannot be changed in a more complex scale as the users are allowed only to manipulate only two to three parameters. Many simulators were built to show only the interface of the airplane or to introduce the different models of the airplane, while the fundamental understanding of the trajectory and the aircraft control is overlooked. Despite the fact that the user can understand the basic design and principle of the work, it cannot change any dynamic setting. It limits the knowledge to evaluate the performance of the aircraft under different control settings to positively assess it as a normal condition. Addressing these challenges will be important for the further development of aerospace engineering education. There is a great educational void in the present regime, wherein it is seen that no available platform may be accessible, user-friendly, and interactive to support the facilitation of learning by reducing this gap between theoretical learning and practical application. Regarding teaching and learning the dynamics of the control of an airplane, both in pitch, roll, and yaw, inadequacy and gaps are grossly observed. The development of such a platform can ensure increased access and quality in aerospace education. Web-based simulators promise the following: with the current improvements in the state of the art of web technologies set of standards, technologies, or tools that enable a unique experience in working with Internet information

such as JavaScript, and diverse frameworks like InfinityFree, it will be possible to devise interactive and real-time simulations, making them accessible from different devices in an unencumbered way, without the need to install specific software. That way, besides the economic and hardware difficulties, user-friendly interactions may be favored for a wide background of learners. The concept can democratize aerospace education including allowing students and professionals of various levels from different corners of the world hands-on use with advanced control systems concepts.

2.0.1 Project Aim

The project's objective is to create a web-based platform that have an airplane simulation based on the different control algorithms such as LQR, FSF and PID. The model of such control system demonstrate the behavior of the plane within the pitch and roll controls. The simulation demonstrates the real-time change in dynamic behavior based on the inputs of the users. Therefore, the dynamic motion of the aircraft can be grasped intuitively after some time by the user.

Development of a web-based interactive simulator that can demonstrate the dynamics of the airplane control system, model, and illustrate how the aircraft behaves in conditions of pitch, roll, and yaw control using different control strategies is the main goal of the project. The platform will be able to let users compare and contrast the performance of these methodologies under varying conditions of aircraft dynamics by incorporating algorithms such as PID, LQR, and FSF control. The comparison method will improve comprehension nt by enabling individuals to get first-hand exposure to how different control strategies respond to disturbances and input changes.

The key elements of the work involve developing a mathematical model of airplane dynamics. Aircraft behavior is predetermined through the interaction of complicated relationships between forces, such as aerodynamic lift and drag, thrust, gravity, and the moments of these, together with structural design and control inputs. The accuracy of modeling dynamics is basically important in developing a reliable simulator. The model will describe the equations of motion for airplane pitch, longitudinal axis; roll, lateral axis; and yaw, vertical axis, as it depicts the complicated relationships between control inputs and the aircraft's response.

The developed mathematical model will then be integrated with the control algorithms. PID control-one of the most common methodologies in engineering will provide the basis of comparison in this research. PID controllers are a family of controllers that control the input by minimizing error through a proportional, integral, and derivative term. Effective in a diverse array of applications, PID controllers may be challenged to keep the system steady in a rapidly changing environment, thus serving as an ideal foundation for exploring more advanced algorithms. The LQR algorithm, together with optimizing control actions based on a cost function, will finally be implemented. LQR is well adapted when applied to

some systems for which the end objective is to minimize anything from energy expenditure to substantial deviation from a desired target trajectory. Finally, FSF control, which views system behavior and applies compensation pre-emptively, is used when a fast response is wanted in applications.

2.0.2 Objectives

The Web-based airplane simulator is an interactive platform with the different parameters to control the ht trajectory of the plane such as the gains of PID, the cost function in the LQR that calculates through the iterations, or the full-state feedback. Different control algorithms lead to different results, and to comprehend these algorithms, the visual demonstration of each outcome plays a key role. As students navigate through the different results, behavioral memory can be adapted to memories of the effects of each control algorithm and its effects on the motion of the airplane. To make it accessible, the simulator is meant to be web-based platform. It can be accessed with any device that has an Internet connection. It alleviates the need to have a high functional hardware and no installations making it widely accessible to the global community with different background. As the simulations were built on JavaScript and HTML, they were cross-checked with MATLAB to test the accuracy of the results. As the development ends, the user feedback was collected and implemented to make it easier to navigate through the platform.

Chapter 3

Methodology

3.1 MATLAB

MATLAB is especially appreciated for its stable computations considering such fields as science, engineering, and numerical simulation. The in-built toolboxes like Simulink, Aerospace toolbox, and UAV toolbox make it ideal for aircraft model design and validation of control systems and flight dynamics (Garza & Morelli, 2003) (Horri & Pietraszko, 2022). MATLAB is considered powerful in many numeric computations and simulations of dynamic systems because it gives high accuracy. Its GUI and the vast number of tools for visualization make it easy to investigate nonlinear dynamics, stability, and control laws, as Lynch et al. (2019) pointed out. Nevertheless, MATLAB has some limitations, especially in cases where web-based simulation is needed, mainly due to its compatibility with the web. MATLAB, for instance, can only communicate with external systems using some protocols. For UDP, one will need some extra software and settings making the job more complicated with this coding language. Secondly, MATLAB is a commercial software and its license is expensive, especially for projects that seek to develop open-source web-based applications (Zhonghua & Haitao, 2021).

MATLAB is also not well suited for web-based applications because it does not natively support web technologies such as HTML5 or WebGL. While attempts have been made to interface MATLAB with flight simulators such as X Plane and FlightGear using co-simulation techniques, such configurations require significant computational power and are usually based on offline solutions (Horri & Pietraszko, 2022). These characteristics make MATLAB more appropriate for prototyping and desktop-based simulations rather than for lightweight, interactive web-based applications. For example, MATLAB is good at verifying control algorithms such as PID and LQR in a simulated flight environment, but its computational structure is rather different from the lightweight and client-side requirements of web applications. Therefore, its use in web-based aircraft simulations is still not feasible without major modifications and thus reiterates the need for a more web-friendly language such as JavaScript.

3.2 C++

C++ is another contender for the development of aircraft simulations. Because of its capacity to manage low-level hardware interactions and improve performance-critical

processes, it is used in real-time simulations and embedded systems (Zhonghua & Haitao, 2021). C++ has been applied in the Aerospace industry in developing flight dynamics and control systems, for instance, in AFPAS software where C++ is used for real-time flight simulation (Zhonghua & Haitao, 2021). However, similar to MATLAB, C++ raises numerous issues when it comes to web applications. When creating web-compatible simulations in C++ it is often necessary to use additional frameworks, for example, Emscripten for compiling C++ to WebAssembly, or external libraries, for example, OpenGL for displaying graphics. These tools bring in high learning curves and the time taken to develop is relatively higher than when using other tools, thus not suitable for rapid deployment (Herbst, 2018).

Also, there is no native support for Web technologies such as JavaScript or WebGL; even basic interactivity requires a lot of integrational efforts. Hence, C++ is well suited for standalone applications and back-end simulations but not for lightweight, browser-based systems because it requires server-side support for handling and visualization of real-time data. Moreover, when working with C++, the application interfaces with other libraries like Qt or GLFW to provide internet-simulating interfaces which enhance the difficulty of constructing smooth, user-friendly visualizations. While these dependencies demand a high level of technical skills in managing them, JavaScript is a plain language that compiles directly with web standards. Therefore, although C++ is still a valuable language for high-performance simulations, it is much less well suited to the development of web-based aircraft simulations since it does not have native web integration and the difficulties of creating interactive interfaces.

3.3 JavaScript

JavaScript as the core programming language to implement the web-based aircraft simulation system has been chosen deliberately and justified, based on the characteristic features of this language along with the nature of the application's extensive interface, and dynamic, interactive web application (Agyei & Agyei, 2021). JavaScript is now a standard for web development and simulation systems because it complies with modern web technologies, its capability to handle real-time interactions and its compatibility with back-end systems and user interfaces. For the aims of a simulation where models need to be controlled and visualized dynamically according to the users' irrational input regarding the flight, JavaScript turns out to be the most effective as well as a pragmatic solution rather than MATLAB or C++.

JavaScript's ability to run across platforms and being native support by web browsers makes it an unrivaled option for web-based simulation development. JavaScript is not compiled like languages such as C++, which requires utensils and libraries to interface with the Web technologies and also it does not require any plugin to run in a browser.

This capability not only makes the development process easier but also guarantees that the simulation is available on any device with a browser whether it is a desktop, a tablet, or a smartphone (Wagner, 2016). Research has shown that JavaScript is useful in physics and engineering courses because it is interactive and real-time helping users to learn more (Agyei & Agyei, 2021). In the context of aircraft simulation, these features mean dynamic feedback and control, which in other words allows the user to control parameters such as thrust, pitch, and yaw and observe their effects in real-time.

JavaScript's compatibility with website elements and structures like HTML5, CSS, and WebGL equally boosts the kind of graphics and interaction simulated applications can come up with. WebGL which allows for the usage of hardware acceleration in web browsers to render 3D graphics, has been successfully employed to develop rather complex and interactive physics models (Esquembre et al., 2019). Software like Easy JavaScript Simulations (EjsS) are available to help educators and developers build the simulation environment with little or no JavaScript programming knowledge (Clemente et al., 2017). These tools lower the technical hurdles associated with the development and deployment of simulations and make JavaScript a practical and feasible means of modeling the dynamic responses of aircraft systems to different control algorithms.

MATLAB has been widely used in numerical modeling and simulation, however, it is not very suitable for the latest web-based applications due to the unreliability of its web deployment. While MATLAB is highly computational, it is not very suitable for interactive, browser-based simulations unless these are designed with considerable workarounds (Schmid & Ali, 2000). Likewise, C++ has a better computational advantage but makes it difficult to build graphical user interfaces and their integration with web environments. While JavaScript enables the developers to implement the simulation logic, data processing, and visualization within the same language (Saenz et al., 2015), other languages let the developers design the entire solution of the Web-based systems, including the simulation and visualization parts. In addition, asynchronous features of JavaScript through the Node.js system allow for stable real-time data interaction and data processing, the main requirement for dynamic aircraft replicas.

The ability to define new algorithms by the user and the dynamic changing of the model also makes JavaScript useful for simulation purposes. Studies on scriptable simulation environments bring out how JavaScript can change variables, alter models, and bring about new sophisticated actions in real time (Divjak & Marolt, 2001). In aircraft simulations where the behavior of the system depends on control strategies like PID, MPC or LQR in real-time the event-driven architecture of JavaScript helps in integrating the user inputs and system outputs. Such a level of interactivity improves user experience and makes the system as close as possible to the actual physical systems.

The use of JavaScript in control engineering, education, and virtual reality also supports the use of JavaScript for simulation tasks. For instance, the analysis of medical training

simulations in practice has demonstrated how the implementation of JavaScript and 3D modeling technologies permits the construction of worlds of learning practice (Korocsec et al., 2005). Likewise, the application of JavaScript in business and engineering simulations shows its capability of dealing with dynamic system characteristics and interacting with the backend data systems (Pillutla, 2003). These applications offer strong proof for the use of JavaScript in the simulation of various scenarios without necessarily compromising the user interface or the system's capacity.

3.4 Roll and Yaw control

The airplane's roll and yaw simulations were achieved by the implementation of LQR, PID, and FSF controllers through the use of state-space equations in the HTML code syntax. The state-space equations for roll motion's lateral dynamics were as follows:

$$\dot{x} = Ax + Bu \quad x = \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} \quad u = \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix} \tag{3.1}$$

Where,

- $x^T = \begin{bmatrix} \beta & p & r & \phi \end{bmatrix}^T$ - the state vector equation;
- $u^T = \begin{bmatrix} \delta_a & \delta_r \end{bmatrix}^T$ - the control vector equation;
- δ_a, δ_r - the airplane wings' aileron and rudder deflections;
- β, ϕ - the airplane's sideslip and roll angles;
- p, r - the roll and yaw rates of the airplane (Nelson, 1998).

$$A = \begin{bmatrix} \frac{Y_\beta}{U_0} & \frac{Y_p}{U_0} & \frac{Y_r}{U_0} & \frac{gcos\theta_0}{U_0} \\ L_\beta & L_p & L_r & 0 \\ N_\beta & N_p & N_r & 0 \\ 0 & 1 & tan\theta_0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} \frac{Y_{\beta_r}}{U_0} & \frac{Y_{p_a}}{U_0} \\ L_{\beta_r} & L_{\beta_a} \\ N_{\beta_r} & N_{\beta_a} \\ 0 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.2}$$

3.4.1 LQR Controller Implementation

The main purpose of the LQR controller is to minimize the following cost function:

$$J = \int_0^\infty (x^T Q x + u^T R u)\, dt \tag{3.3}$$

Where:

- Q is a positive semi-definite matrix;
- R is a positive scalar term that penalizes the control input efforts.

The optimal control input is computed by the following equation:

$$u = -Kx$$

The gain matrix K that minimizes the cost function is computed as:

$$K = R^{-1} B^T P \tag{3.4}$$

The P in the equation above is the solution to the Algebraic Riccati Equation, defined as:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{3.5}$$

To create a website for the simulation of airplane control, we used JavaScript. This code simulates the roll control of an airplane using a Linear Quadratic Regulator (LQR).

Main points of the interface:

- **Dropdown menu** (controllerType) lets the user select between PID and LQR control methods, thereby, making it convenient for use.
- **Input fields** dedicated to indicate the desired roll angle and control parameters.
- Functions Breakdown:
- **LQR Control** (lqrControl):LQR control is set by manipulating the State-Cost Weighting Matirx (Q) that defines any variable within the cost function, while the Matrix (R) of the Control-Input limits the input of the disproportionally large coefficients.
- **State Update** (updateState): This function uses the state-space equations to update the roll angle and roll rate in order to simulate how the aircraft reacts to the control input and its changes.

3.5 Web-hosting

As the domain of the web-hosting platform was set as aircraft-cs.com, the hosting itself were decided to be through the InfinityFree provider. The STL certificate was installed to encrypt and secure the connection of the site. As the DNS of the provider went through the DNS caching and the nameservers were pointed to the InfinityFree, the hosting has been available to the users and the feedback data was collected through the google forms that was placed on the site as a link. Obtained data was grouped and then presented in the Results section.

Chapter 4

Results

4.1 Pitch Controller Results

4.1.1 LQR Controller

The LQR controller in this pitch simulation has two options. (1) Simple, which accepts the precomputed K values from MATLAB, and a manual, which offers a user the Q and R matrices. (2) Manual, which provides a user with $Q(3x3)$ and $R(1x1)$ matrix layouts to set the diagonal values for Q and R.
Since the LQR solves the complex Ricatti Algebraic Equation, the simple model was offered. Users can tune their own state-space model via MATLAB, that will be described further.

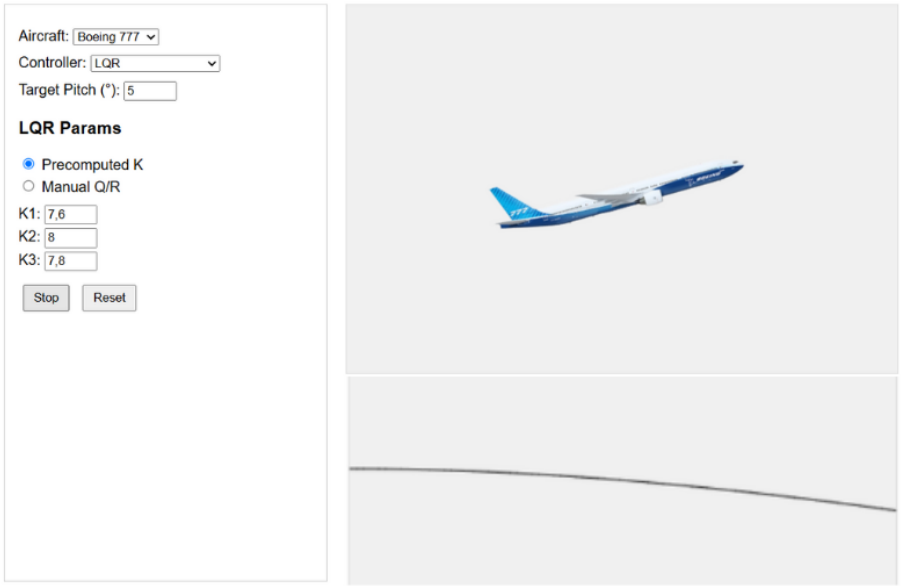

Figure 4.1: *LQR Model for Optimal K precomputed*

The `.js` conditional code implemented for the precomputed K (simple) model and advanced models are as follows:

```
    //        LQR Controller: pre-stored K  or  on-the-fly Riccati
class LQRController {
  constructor(ac) { this.ac = ac; }

  compute(refDeg, curDeg, x, dt) {
```

```
  const m = AIRCRAFT_DB[this.ac], A = m.A, B = m.B;
  let K;

  // option 1        manual Q/R     iterate Riccati
  if (lqrManMode.checked) {
    const Q = [
      [+q11.value, +q12.value, +q13.value],
      [+q21.value, +q22.value, +q23.value],
      [+q31.value, +q32.value, +q33.value]
    ];
    const R = [[+r11.value]];
    K = this.solveRiccati(A, B, Q, R, 50);   // returns [k1,k2,k3]
    [lqrK1, lqrK2, lqrK3].forEach((e,i)=>e.value = K[i].toFixed(3)); // show
  }

  // option 2        use current K-boxes (pre-entered or copied from above)
  else {
    K = [+lqrK1.value, +lqrK2.value, +lqrK3.value];
  }

  // state-feedback + Nbar reference
  const BK   = outer(B.map(v=>v[0]), K);
  const Acl  = matAdd(A, scalarMul(BK, -1));
  const inv  = invert3x3(Acl);
  const CNB  = matMulVec(inv, B.map(v=>v[0]))[2] || 1e-6;
  const Nbar = -1 / CNB;

  const  r    = refDeg * Math.PI / 180;
  const u    = -(K[0]*x[0] + K[1]*x[1] + K[2]*x[2]) + Nbar* r ;
  return clamp(u, -m.maxU, m.maxU);
}

//        continuous Riccati iteration (Newton-Raphson style)
solveRiccati(A, B, Q, R, iters) {
  let P = identity3x3();                      // initial guess
  const Bt = transpose(B), Rt = R[0][0];

  for (let k = 0; k < iters; k++) {
    const BtPB      = matMul(matMul(Bt, P), B)[0][0];
    const invTerm   = 1 / (Rt + BtPB);       // scalar
    const AP        = matMul(transpose(A), P);
    const Ktmp      = matMulScalar(matMul(Bt, P), invTerm)[0];
    const AK        = matAdd(A, scalarMul(outer(B.map(v=>v[0]), Ktmp), -1));
    const Pnext     = matAdd(Q, matMul(matMul(transpose(AK), P), AK));
    P = Pnext;
  }
  const BtP = matMul(Bt, P)[0];
```

```
    const inv = 1 / (R[0][0] + matMul(BtP, B)[0][0]);
    return BtP.map(p => inv * p);
  }
}
```

The option with manual LQR setup offers users the ability to insert the Q and R values themselves. Figure 4.2 below clearly illustrates the layout offered - a case with a tuned Boeing 777-300ER aircraft.

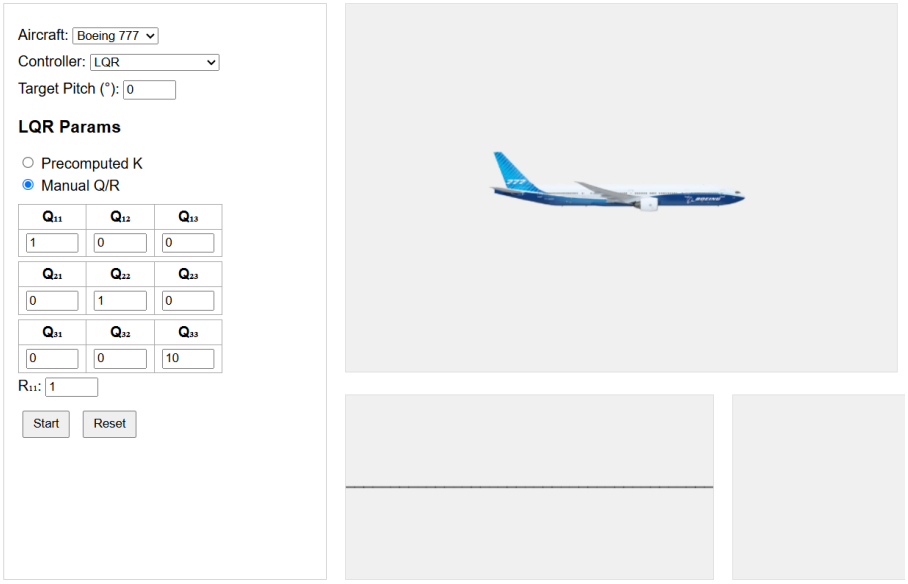

Figure 4.2: LQR with a manual setup and Ricatti solver – a Boeing 777-300ER case

The compute method first determines the feedback vector K. If the Manual Q/R option is active it calls solveRiccati, which starts from the identity and iterates the continuous-time Riccati update $P_{k+1} = Q + (A - BK_k)^T P_k (A - BK_k)$ roughly fifty times, then returns $K = \left(R + B^T P B\right)^{-1} B^T P$; otherwise it reads the gains already typed into `lqrK1-3`. With k fixed, the closed-loop matrix is formed as $A_{cl} = A - BK$ and the single-output pre-compensator is obtained from $N_{bar} = -\frac{1}{C A_d^{-1} B}$, where $C = [001]$.

The MATLAB implemented code below computes the optimal K gains for the state space models we defined.

```
    clear; clc;

%% 1) Define weighting matrices (tweak these to suit your app)
Q = diag([10, 50, 100]);   % state penalties [ u ,  w ,     ]
R = 1;                     % control penalty

%% 2) Cessna 172
A_C172 = [ -0.045,    0,  -32.2;
            0.010, -2.948,   0;
              0,      1,     0 ];
B_C172 = [  0.0656;
```

```
           -6.726;
             0      ];

[K_C172, S_C172, e_C172] = lqr(A_C172, B_C172, Q, R);

fprintf('C172 LQR gain K = [%.4f  %.4f  %.4f]\n', K_C172);

%% 3) Boeing 777
A_B777 = [ -0.0011,    0,  -32.2;
             0.0050, -2.200,   0;
               0,       1,     0 ];
B_B777 = [  0.0164;
           -13.123;
              0     ];

[K_B777, S_B777, e_B777] = lqr(A_B777, B_B777, Q, R);

fprintf('B777 LQR gain K = [%.4f  %.4f  %.4f]\n', K_B777);

%% 4) Airbus A330
A_A330 = [ -0.0015,    0,  -32.2;
             0.0100, -2.500,   0;
               0,       1,     0 ];
B_A330 = [  0.0263;
           -11.483;
              0     ];

[K_A330, S_A330, e_A330] = lqr(A_A330, B_A330, Q, R);

fprintf('A330 LQR gain K = [%.4f  %.4f  %.4f]\n', K_A330);

%% 5) Pack results for easy copy-paste into your web app
fprintf('\nCopy these K values into your UI placeholders:\n');
fprintf('C172: K = [%.4f, %.4f, %.4f]\n', K_C172);
fprintf('B777: K = [%.4f, %.4f, %.4f]\n', K_B777);
fprintf('A330: K = [%.4f, %.4f, %.4f]\n', K_A330);
```

The optimal gains computed are displayed in the command window below:

```
C172 LQR gain K = [3.1039  -7.4282  -40.8292]
B777 LQR gain K = [3.1605  -7.3232  -40.3009]
A330 LQR gain K = [3.1596  -7.3310  -40.4346]

Copy these K values into your UI placeholders:
C172: K = [3.1039, -7.4282, -40.8292]
B777: K = [3.1605, -7.3232, -40.3009]
A330: K = [3.1596, -7.3310, -40.4346]
```

These precomputed values can be inserted into the placeholders indicated on the left side of the UI (Figure 4.1).

4.1.2 FSF Controller

In the FSF-controlled pitch simulation, the user can input the poles and tune the aircraft at the desired target angle with the pole placement method. Figure 4.3 below is a visual result of the FSF pitch simulation layout.

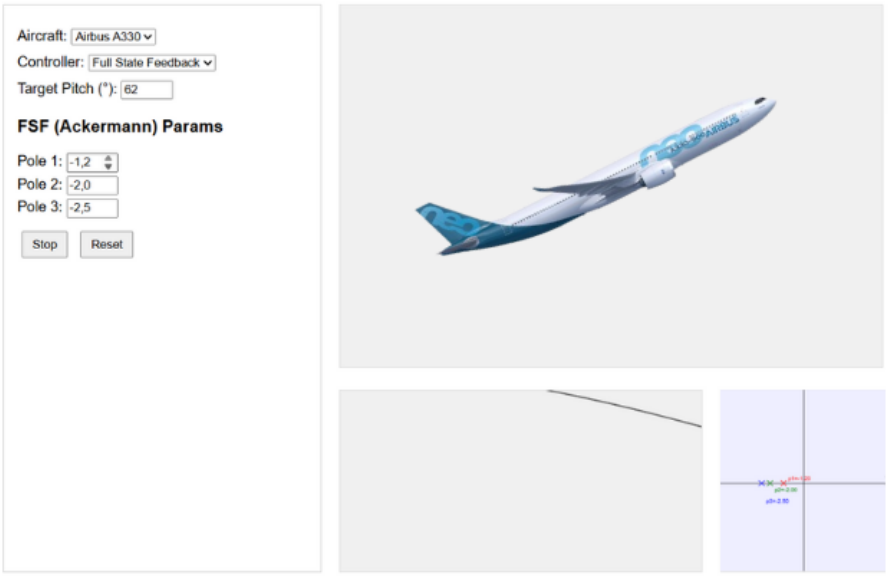

Figure 4.3: FSF – an Airbus A330-900 case

The FSF code snippet below explicitly shows the Ackerman method of pole placement applied in this system:

```
    //        Full-State Feedback (Ackermann pole placement)
class FSFController {
  constructor(ac) { this.ac = ac; this.updateK(); }

  updateK() {
    const mdl = AIRCRAFT_DB[this.ac];
    const A   = mdl.A,  B = mdl.B;

    // desired poles from UI
    const p1 = +pole1.value, p2 = +pole2.value, p3 = +pole3.value;
    const a2 = -(p1 + p2 + p3),
          a1 =   p1*p2 + p1*p3 + p2*p3,
          a0 = -(p1*p2*p3);

    //   (A) = A   +   a  A   +  a A  +  a I
    const A2  = matMul(A, A),
```

```
          A3  = matMul(A2, A);
    const Phi = matAdd( matAdd( matAdd(A3, scalarMul(A2, a2) ),
                                       scalarMul(A,  a1) ),
                                       identity3x3(a0) );

    // controllability matrix Mc = [B AB  A  B]
    const AB  = matMul(A,  B),
          A2B = matMul(A2, B);
    const Mc  = [
      [B[0][0], AB[0][0], A2B[0][0]],
      [B[1][0], AB[1][0], A2B[1][0]],
      [B[2][0], AB[2][0], A2B[2][0]]
    ];

    // K = [0 0 1] *  M c   *   (A)
    const Krow = matMul([ [0,0,1] ], invert3x3(Mc) )[0];
    this.K = matMul([Krow], Phi)[0];        // length-3 array
  }

  compute(refDeg, curDeg, x, dt) {
    // update K if any pole or aircraft changed
    if (this.ac !== aircraftSelect.value) { this.ac = aircraftSelect.value; this.updateK(); }
    if (this.lastP !== pole1.value+pole2.value+pole3.value) {
      this.lastP = pole1.value+pole2.value+pole3.value; this.updateK();
    }

    const  r  = refDeg * Math.PI/180;
    const e  = [ x[0], x[1], x[2] -  r  ];  // track pitch
    let u = 0;
    for (let i = 0; i < 3; i++) u += this.K[i] * e[i];
    return clamp(-u, -AIRCRAFT_DB[this.ac].maxU, AIRCRAFT_DB[this.ac].maxU);
  }
}
```

The Ackermann based full state feedback logic is held by FSFController. When the object is constructed, it calls `updateK()` which pulls the current aircraft matrices A and B, reads the desired poles from the inputs `pole1`, `pole2`, and `pole3`, builds the polynomial coefficients `a2 a1 a0`, forms the matrix polynomial $\Phi = A^3 + a_2A^2 + a_1A + a_0I$, and multiplies it by the last row of the controllability matrix inverse ($row = [[0, 0, 1]] \cdot \mathrm{Mc}^{-1}\big)$) to get the gain vector is a public method that when the aircraft or pole values have changed, refreshes K, forms the error vector $e = [x_1, x_2, x_3 - \theta\_ref]$ sums the weighted error $u = -\Sigma K[i]e[i]$ and returns the value limited by clamp to each aircraft's `maxU.` `updateK()` does the pole placement, and `compute()` applies the resulting full state feedback at every simulation step.

4.1.3 PID Controller

In this section, the results of a real-time aircraft pitch control simulation using a PID controller are discussed. The results are shown in the form of performance parameters such as pitch angle control and horizontal velocity control. Simulation of the aircraft's control of an aircraft's pitch angle to achieve a target value using a PID control approach. The users can control the simulation interface by changing PID gains (kp, ki, kd), target pitch angles, and thrust levels. Outputs are the pitch angle of the aircraft, the horizontal velocity, and a visualized tuning curve that shows the error of the system over time.

Key points of the interface include the following:

- **Interactive control panel**
- **Dynamic Visual Outputs**
- **PID tuning curve**

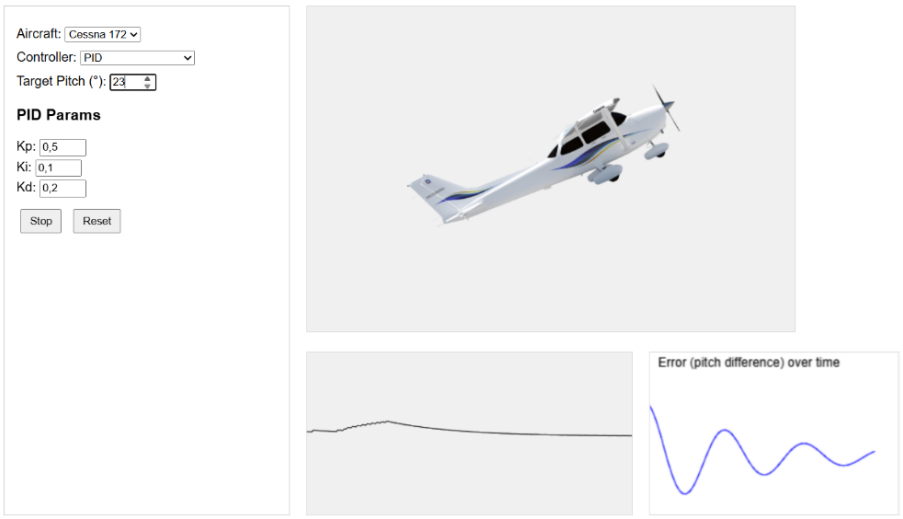

Figure 4.4: *PID Pitch Tuning Curve at Kp=0.5, Ki=0.1, Kd=0.2*

```
//        PID Controller (classic position form with anti-windup & derivative filter)
class PIDController {
  constructor() {
    this.I = 0;          // integral accumulator
    this.prevErr = 0;    // previous error (for derivative)
    this.D = 0;          // filtered derivative term
  }

  compute(refDeg, curDeg, _state, dt) {
    // --- error (degrees) ---
    const err = refDeg - curDeg;
```

```
    // --- gains (UI controls) ---
    const Kp = +pidKp.value;
    const Ki = +pidKi.value;
    const Kd = +pidKd.value;

    // --- integral term with simple clamp anti-windup ---
    const I_CLAMP = 10; // deg*s
    this.I = clamp(this.I + err * dt, -I_CLAMP, I_CLAMP);

    // --- derivative term with first-order low-pass filter ---
    const N = 10;
    const rawD = (err - this.prevErr) / dt;
    this.D = (N * dt / (1 + N * dt)) * rawD + (1 / (1 + N * dt)) * this.D;

    // save error for next step
    this.prevErr = err;
    return -(Kp * err + Ki * this.I + Kd * this.D);
  }
}
```

The new `PIDController` class is entirely contained within a single public method `compute()` that is called every simulation frame to compute the elevator command. It first calculates the pitch error `err = refDeg - curDeg` and retrieves the live-tunable gains `Kp`, `Ki`, `Kd` from the HTML input fields. `This.I` is then updated by `err*dt` and clamped to `±10 deg*s` to prevent wind up when the aircraft hits actuator limits. The raw slope `(err - this.prevErr)/dt` is passed through a first order low pass filter (aggressiveness N = 10) and stored in this.D to remove most of the high frequency noise without delaying the signal too much for the derivative term. The control law `u = -(Kp*err + Ki*this.I + Kd*this.D)` finally returns a negative sign so that a positive pitch demand produces the conventional negative elevator deflection; the main loop clamps u back to the aircraft's `maxU` limit and then sends it to the plant. In addition, the method also saves the current error in `this.prevErr` for the next derivative calculation, and no other state escapes the class, keeping the PID block fully self contained and ready for real time gain tweaking during simulation.

4.2 Roll and Yaw control

The airplane's roll and yaw simulations were achieved by the implementation of LQR, PID, and FSF controllers through the use of state-space equations in the HTML code syntax. The implementation of the state-space equations described in the Section 3.4 is shown in the following code snippet:

```
// Aircraft dynamics
let damping, effectiveness;
```

```
if (controlMode === 'roll') {
    damping = currentAircraft.L_p;  // Roll damping
    effectiveness = currentAircraft.L_delta_a;  // Aileron effectiveness
} else {
    damping = currentAircraft.N_r;  // Yaw damping
    effectiveness = currentAircraft.N_delta_r;  // Rudder effectiveness
}
// State derivatives
const x1_dot = x[1];  // angle_dot = rate
const x2_dot = damping * x[1] + effectiveness * u;  // rate_dot
// Integrate using Euler's method
x[0] += x1_dot * dt;
x[1] += x2_dot * dt;
```

Where, the $\dot{x_1}$ represents the first state derivative, which is the roll rate (x[1]). The $\dot{x_2}$ represents the second state derivative, which is the roll rate dynamics ($L_p \cdot x[1] + L_{\delta_a} \cdot u$).

4.2.1 LQR Controller Results

In the code below, the control input u is computed by the following formula, which brings the control effort needed to bring the system state $x = [\text{angle}, \text{rate}]$ to the desired state (desired angle), following the formulas given in the Section 3.4.1:

$$u = -K[0](x_0 - \text{desiredAngle}) - K[1]x_1$$

The code snippet that captures the implementation of the LQR Control is as follows:

```
// Solving LQR problem for roll or yaw control
function solveLQR() {
    // Getting aircraft dynamics parameters based on control mode
    let damping, effectiveness;
    if (controlMode === 'roll') {
        damping = currentAircraft.L_p;  // Roll damping coefficient
        effectiveness = currentAircraft.L_delta_a;  // Aileron effectiveness
    } else {
        damping = currentAircraft.N_r;  // Yaw damping coefficient
        effectiveness = currentAircraft.N_delta_r;  // Rudder effectiveness
    }
    // Getting weights from inputs
    const Q1 = parseFloat(document.getElementById('qAngle').value);
    const Q2 = parseFloat(document.getElementById('qRate').value);
    const R = parseFloat(document.getElementById('rControl').value);
    // Solving Algebraic Riccati Equation for this simple case
    const a = effectiveness * effectiveness / R;
    const b = -2 * damping;
    const c = Q1;
    const p11 = (-b + Math.sqrt(b*b + 4*a*c)) / (2*a);
```

```
    const p12 = (damping + Math.sqrt(damping*damping + a*p11)) / a;
    const p22 = Math.sqrt((Q2 + 2*p12*damping + a*p12*p12)/a);
    K = [
        (effectiveness/R) * p12,
        (effectiveness/R) * p22
    ];
    // Updating displayed gains
    document.getElementById('k1-value').textContent = K[0].toFixed(3);
    document.getElementById('k2-value').textContent = K[1].toFixed(3);
}
// Using LQR control input in simulation loop
if (controllerType === 'lqr') {
    u = -(K[0] * (x[0] - desiredAngle) + K[1] * (x[1] - 0));
}
```

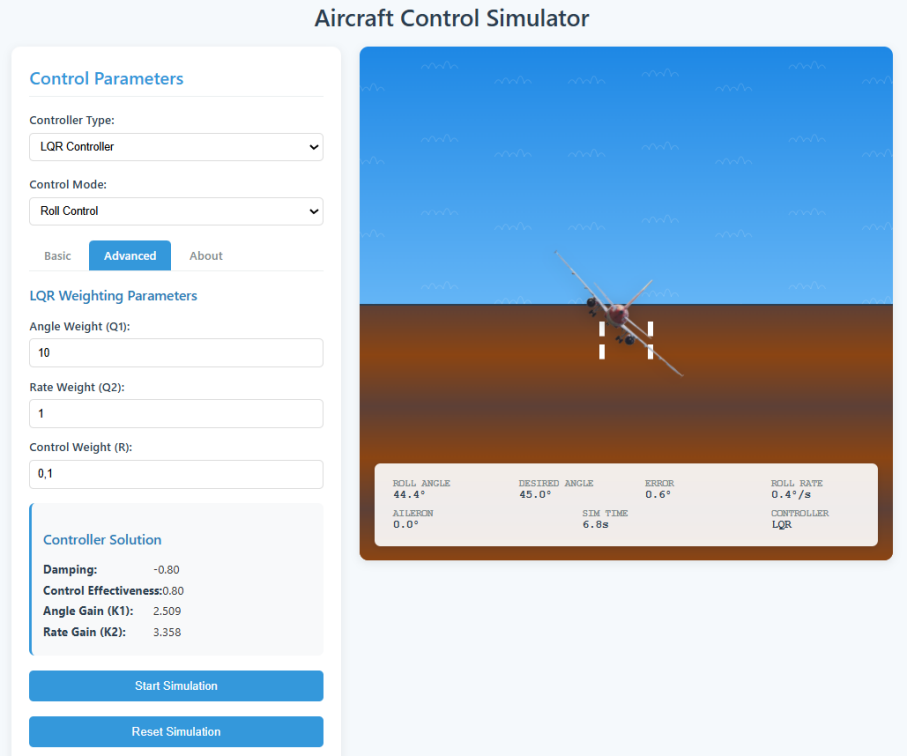

Figure 4.5: *The LQR Controller website page of the roll control*

The LQR Controller website page screenshot given above demonstrates the mix of a user-friendly interface and neat results output, shown in the lower bracket of the airplane canvas.

4.2.2 FSF Controller Results

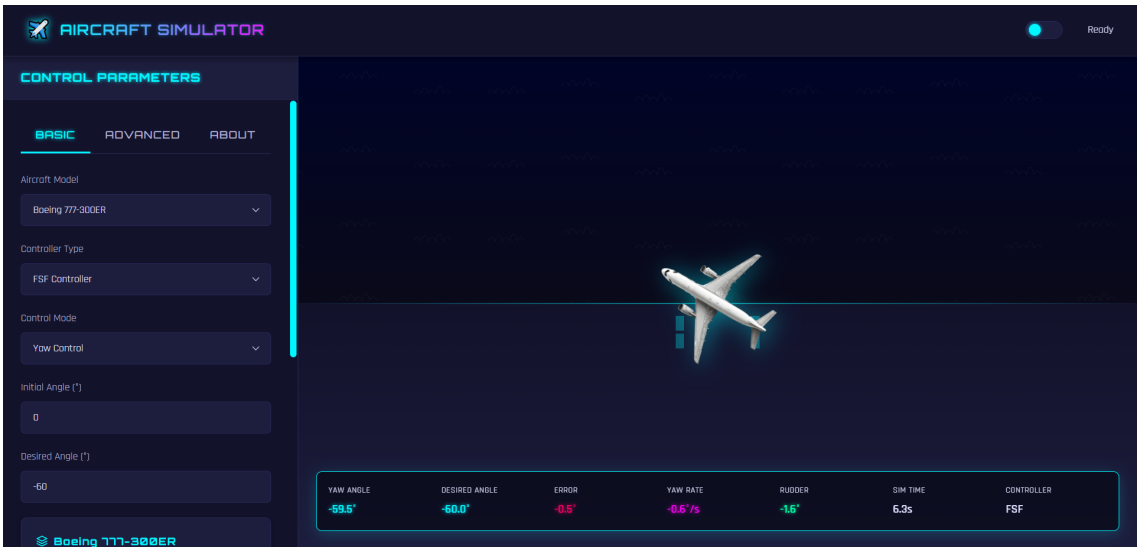

Figure 4.6: *The FSF Controller website page of the yaw motion*

The simulation result of a yaw motion by the FSF controller is shown below in the Figure 4.4. The yaw motion simulation of the Boeing 777-300ER outputted the response time of 6.3s, with a rudder value of -1.6° and a yaw rate of -0.6°/s.

Setting the control variables, and the Desired Angle = 45°, outputs the following results. The results given in the table below demonstrate the behavior of the Airbus A330-220 airplane controlled by 3 different types of controllers (LQR, FSF, PID).

	Roll Angle (°)	**Error (°)**	**Roll Rate (°/s)**	**Aileron Angle (°)**	**Simulation Time (s)**	**Angle Gain K_i**	**Rate Gain K_i**
Test #1. LQR	44.4	0.60	0.40	0.00	6.80	3.358	2.509
Test #2. FSF	44.4	0.60	0.50	0.10	6.10	5.00	6.00
Test #3. PID	45.6	-0.60	-0.10	0.10	25.6	1.00	0.50

Table 4.1: The 3 controller test with the desired angle of 45° on the Airbus A330-200

The simulation time differs across the controllers, due to the difference in their handling. The LQR controller minimizes the state and control errors, thereby converging faster than the PID controller. The PID controller converges at a slower rate, due to the gains not being tuned properly. Additionally, if the integral gain Ki accumulates too much over time, the plane's motion may overshoot. Whereas, the FSF controller operates through the use of system poles, meaning, the incorrect placement may slow the response time.

Additionally, in comparison with a white-themed user interface design shown in Figure 4.3, the dark-themed version, with a neon-based coloring, was added to increase user satisfaction with the simulation experience, as shown in Figure 4.4.

PID Controller Results

Figure 4.5 below demonstrates the success of a PID controller in simulating the Airbus A300-200 airplane's roll motion. The roll angle is close to the desired one, with a small error of -0.6°. This discrepancy is reasonable and explained by the small overshoot or damping effect of the controller parameters. The control effectiveness value of 0.80 indicates that the PID controller can almost completely control the time lag. However, the simulation needs further optimization to fully overcome the errors described above.

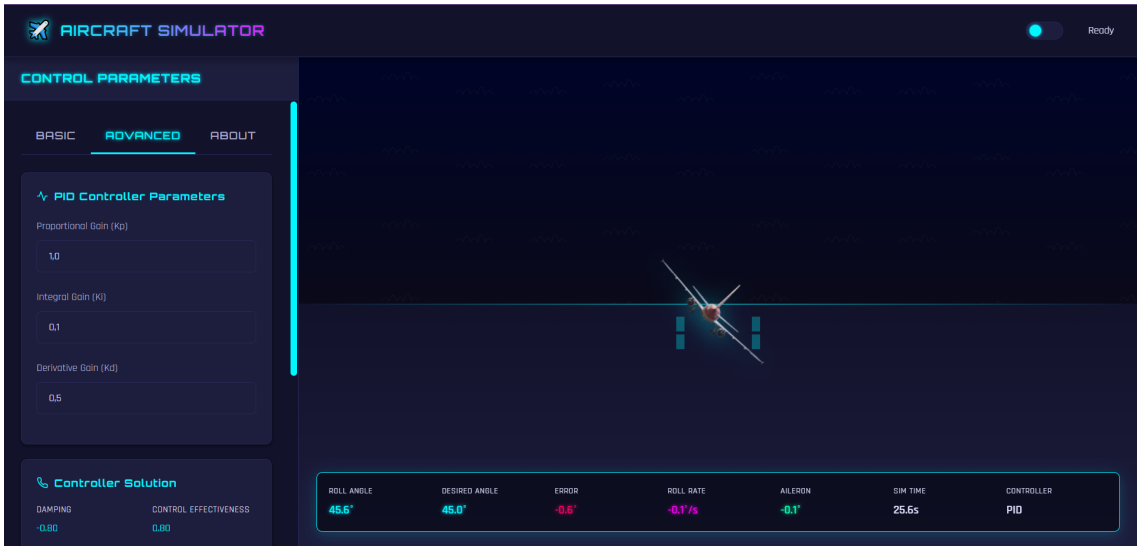

Figure 4.7: *The PID Controller website page of the roll motion*

Overall, the PID controller shows the ability to stabilize the aircraft's roll motion. Though the simulation values are fairly accurate, experimenting on different gain values and additional controllers, such as LQR and FSF, would considerably improve the control results.

4.2.3 Results Validation

The website simulation outputs the lateral derivative values of the airplane. Those values were compared with the experimental values given in the Flight Stability and Automatic Control book by Dr. Robert Nelson to validate the suitability and accuracy of the simulation results. The comparison of the simulated lateral derivatives and the experimental values in Table 4.2 above displays the error difference of up to 12%. This signifies that the simulation values are close enough for the limited computational scope of the HTML environment. The large discrepancies are explained by the presence of disturbances in the experimental environment.

Lateral Derivatives	Simulation Values	Experimental Values	Error difference (%)
Y_u	-0.254 (s^{-1})	-0.245 (s^{-1})	3.543
Y_β	-13.818 (m/s^2)	-13.935 (m/s^2)	5.863
Y_p	0	0	-
Y_r	0	0	-
N_u	0.00671 (ms^{-1})	0.00762 (ms^{-1})	11.942
N_β	4.55 (s^{-2})	4.49 (s^{-2})	1.319
N_p	-0.31 (s^{-1})	-0.35 (s^{-1})	11.429
N_r	-0.717 (s^{-1})	-0.76 (s^{-1})	5.658
L_u	-0.03110 (m/s)	-0.02773 (m/s)	10.84
L_β	-15.87 (s^{-1})	-16.02 (s^{-1})	5.301
L_p	-7.9913 (s^{-1})	-8.4 (s^{-1})	4.865
L_r	1.978 (s^{-1})	2.19 (s^{-1})	9.680

Table 4.2: The Lateral Derivatives of the general aviation airplane (Nelson, 1998)

4.3 User Feedback

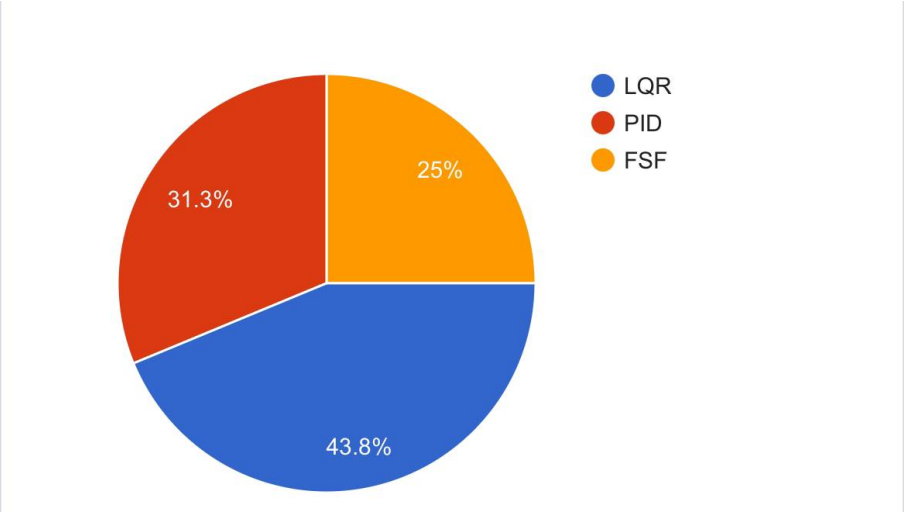

Figure 4.8: *User ranking of the most effective controller*

Google forms was presented in the website asking to rate the user the functionality, loading and the overall appeal. 32 users were asked to answer the questions. Majority of the users were not or less familiar with the aircraft control theory, however; the 43.8 percent of the users stated that the most effective controller was LQR. Over than 80 percent of the users were satisfied with the functionality of the website. Third of the users rated the overall design of the interface as 3, hence the design considerations were remade such as dark theme. Vast majority of the users stated that the interface is user friendly and intuitive, therefore the website's overall ranking is good. The results can be seen in Appendix C.

Conclusion

The work described here is based on how JavaScript forms the base of a web-based interactive aircraft simulation, focusing on the implementation of PID and LQR control methods. It takes advantage of JavaScript's compatibility with modern web technologies, including HTML5, CSS, and WebGL, and allows for the realization of a dynamic real-time interface accessible from multiple platforms. This interactive environment allows the user to tune key parameters, such as PID gains, target angles, and thrust, while immediately obtaining visual and numerical feedback on aircraft behavior. This level of responsiveness enriches the understanding and learning process and is, therefore, an extremely valuable tool for education and experimentation in control systems.

This allows the user to play with PID for pitch angle control, observing the performance of the system by tuning error curves and velocity feedback. Similarly, LQR will provide the dynamics of the roll angle and how to obtain an optimal control using a state-space model. Seamless switching between such controllers makes the system flexible for a wide range of users with different levels of prior knowledge in control theory.

JavaScript's event-driven architecture and ability to process asynchronously enforce its suitability for real-time simulation tasks. These features enable the creation of an interactive and scalable system that supports dynamic model updates and user-defined scenarios. In comparison with other more traditional tools, such as MATLAB and C++, JavaScript has far better integration with web platforms without any deployment barriers, thus enhancing user experience. This will make it a perfect choice for simulations where high interactivity and accessibility are needed.

The developed system further develops the cause of applying JavaScript to engineering. This system serves as a very strong, versatile, and user-friendly platform for learning aircraft control principles with broad areas of application from both educational and professional standpoints. An approach that has the potential of shifting ground in the classical simulation environment to web technologies opens up perspectives for new developments in aerospace education and interactive learning.

Bibliography

(1) Agyei, E. D., & Agyei, D. D. (2021). Enhancing Students' Learning of Physics Concepts with Simulation as an Instructional ICT Tool. European Journal of Interactive Multimedia and Education, 2(2), e02111. https://doi.org/10.30935/ejimed/11259

(2) Balamuralithara, B., & Woods, P. C. (2008). Virtual laboratories in engineering education: The simulation lab and remote lab. *Computer Applications in Engineering Education*, *17*(1), 108–118. https://doi.org/10.1002/cae.20186

(3) Bansal, Hari & Sharma, Rajamayyoor & Ponpathirkoottam, Shreeraman. (2012). PID Controller Tuning Techniques: A Review. 2. 168-176.

(4) Borase, R. P., Maghade, D. K., Sondkar, S. Y., & Pawar, S. N. (2020). A review of PID control, tuning methods and applications. *International Journal of Dynamics and Control*, *9*(2), 818–827. https://doi.org/10.1007/s40435-020-00665-4

(5) Clemente, F. J. G., Esquembre, F., & Wee, L. K. (2017). Deployment of physics simulation apps using Easy JavaScript Simulations. 2022 IEEE Global Engineering Education Conference (EDUCON). https://doi.org/10.1109/educon.2017.7942985

(6) Chernikova, O., Heitzmann, N., Stadler, M., Holzberger, D., Seidel, T., & Fischer, F. (2020). Simulation-Based Learning in Higher Education: A Meta-Analysis. *Review of Educational Research*, *90*(4), 499–541. https://doi.org/10.3102/0034654320933544

(7) Divjak, M., & Marolt, M. (2002). Flexibility of JavaScript controled simulations. EUROCON'2001. International Conference on Trends in Communications. Technical Program, Proceedings (Cat. No.01EX439). https://doi.org/10.1109/eurcon.2001.938149

(8) Esquembre, F., Clemente, F. J. G., Chicón, R., Wee, L., Kwang, L. T., & Tan, D. (2019). Easy Java/JavaScript Simulations as a tool for Learning Analytics. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1910.09156

(9) Friedland, B., & Stephen, W. (1987). *Control System Design: An Introduction to State-Space Methods*.

(10) Garza, F. R., & Morelli, E. A. (2003). A Collection of Nonlinear Aircraft Simulations in MATLAB. NASA Technical Reports.

(11) Herbst, S. (2018). Development of an Aircraft Design Environment Using an Object-Oriented Data Model in MATLAB. Thesis. https://mediatum.ub.tum.de/1431402

(12) Horri, N., & Pietraszko, M. (2022). A tutorial and review on flight control Co-Simulation using Matlab/Simulink and flight simulators. Automation, 3(3), 486–510. https://doi.org/10.3390/automation3030025

(13) Koročsec, D., Holobar, A., Divjak, M., & Zazula, D. (2005). Building interactive virtual environments for simulated training in medicine using VRML and Java/JavaScript. Computer Methods and Programs in Biomedicine, 80, S61–S70. https://doi.org/10.1016/s0169-2607(05)80007-0

(14) Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (2019). *Feedback control of dynamic systems* (8th ed.). Pearson.

(16) MathWorks. (2023). *MATLAB and Simulink for aerospace applications*. Retrieved from https://www.mathworks.com/

(17) Nise, N. S. (2021). *Control systems engineering* (8th ed.). Wiley.

(18) Ogata, K. (2020). *Modern control engineering* (6th ed.). Pearson.

(19) Ramos, J., Santos, D., & Martinez, E. (2020). Web-based simulation tools for engineering education: A case study on control systems. *Journal of Engineering Education*, 109(2), 145-162.

(20) Zulu, A., & John, S. (2014). A review of control algorithms for Autonomous quadrotors. *Open Journal of Applied Sciences*, *04*(14), 547–556.

(21) Nelson, R. (1998). Flight stability and automatic control. In Google Books. https://books.google.kz/books/about/Flight_Stability_and_Automatic_Control.html?

Appendices

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Aircraft Pitch Control Simulator</title>
  <style>
    body{font-family:Arial,Helvetica,sans-serif;margin:20px}
    .container{display:flex;gap:20px}
    .control-panel{width:320px;border:1px solid #ccc;padding:15px}
    .visualization{flex:1}
    canvas{background:#f0f0f0;border:1px solid #ddd}
    .param-group{margin:10px 0}
    .controller-section{display:none}
    .controller-section.active{display:block}
    .lqr-mode{margin:10px 0}
    table{border-collapse:collapse;margin:5px 0}
    th,td{border:1px solid #aaa;padding:4px 8px;text-align:center}
    label{display:block;margin:4px 0}
    button{margin:5px;padding:5px 10px}
    input[type="number"]{width:50px}
  </style>
</head>
<body>
<h1>Aircraft Pitch Control Simulator</h1>
<div class="container">
  <!-- ------------- control panel ------------- -->
  <div class="control-panel">
    <div class="param-group">
      <label>Aircraft:
        <select id="aircraft-select">
          <option value="C172">Cessna 172</option>
          <option value="B777">Boeing 777</option>
          <option value="A330">Airbus A330</option>
        </select>
      </label>
    </div>

    <div class="param-group">
      <label>Controller:
        <select id="controller-select">
          <option value="PID">PID</option>
          <option value="LQR">LQR</option>
          <option value="FSF">Full State Feedback</option>
        </select>
      </label>
    </div>
```

```
<div class="param-group">
  <label>Target Pitch (  ):
    <input type="number" id="target-pitch" value="0" step="1">
  </label>
</div>

<!-- ---- PID ---- -->
<div id="pid-params" class="controller-section active">
  <h3>PID Params</h3>
  <label>Kp: <input type="number" id="pid-kp" value="0.5" step="0.1"></label>
  <label>Ki: <input type="number" id="pid-ki" value="0.1" step="0.1"></label>
  <label>Kd: <input type="number" id="pid-kd" value="0.2" step="0.1"></label>
</div>

<!-- ---- LQR ---- -->
<div id="lqr-params" class="controller-section">
  <h3>LQR Params</h3>
  <div class="lqr-mode">
    <label><input type="radio" name="lqr-mode" id="lqr-pre-mode" value="pre" checked> Pr
    <label><input type="radio" name="lqr-mode" id="lqr-man-mode" value="man"> Manual&nbs
  </div>

  <div id="lqr-pre">
    <label>K1:<input type="number" id="lqr-k1" value="0.8" step="0.1"></label>
    <label>K2:<input type="number" id="lqr-k2" value="-0.3" step="0.1"></label>
    <label>K3:<input type="number" id="lqr-k3" value="5.2" step="0.1"></label>
  </div>

  <div id="lqr-manual" style="display:none">
    <table><tr><th>  Q    </th><th>  Q    </th><th>  Q    </th></tr>
      <tr><td><input id="q11" type="number" value="1"></td><td><input id="q12" type="numb
    </table>
    <table><tr><th>  Q    </th><th>  Q    </th><th>  Q    </th></tr>
      <tr><td><input id="q21" type="number" value="0"></td><td><input id="q22" type="numb
    </table>
    <table><tr><th>  Q    </th><th>  Q    </th><th>  Q    </th></tr>
      <tr><td><input id="q31" type="number" value="0"></td><td><input id="q32" type="numb
    </table>
    <label>  R    :<input id="r11" type="number" value="1"></label>
  </div>
</div>

<!-- ---- FSF ---- -->
<div id="fsf-params" class="controller-section">
  <h3>FSF (Ackermann) Params</h3>
  <label>Pole 1:<input type="number" id="pole1" value="-1.2" step="0.1"></label>
  <label>Pole 2:<input type="number" id="pole2" value="-2.0" step="0.1"></label>
```

```
      <label>Pole 3:<input type="number" id="pole3" value="-2.5" step="0.1"></label>
    </div>

    <div class="param-group">
      <button id="sim-toggle">Start</button>
      <button id="reset-btn">Reset</button>
    </div>
  </div>

  <!-- ------------ visualisation ------------- -->
  <div class="visualization">
    <canvas id="main-canvas"  width="600" height="400"></canvas>
    <div style="display:flex;gap:20px;margin-top:20px">
      <canvas id="error-canvas" width="400" height="200"></canvas>
      <canvas id="pole-canvas"  width="400" height="200"></canvas>
    </div>
  </div>
</div>

<!-- ====================== MAIN SCRIPT ====================== -->
<script>
/* ---------- tiny linear-algebra helpers ---------- */
function matMul(A,B){const r=A.length,c=B[0].length,m=A[0].length,C=Array(r).fill().map(()=>Array(
function matMulVec(A,v){return A.map(row=>row.reduce((s,x,i)=>s+x*v[i],0));}
function matAdd(A,B){return A.map((r,i)=>r.map((v,j)=>v+B[i][j]));}
function scalarMul(A,s){return A.map(r=>r.map(v=>v*s));}
function identity3x3(k=1){return[[k,0,0],[0,k,0],[0,0,k]];}
function invert3x3(M){const[a,b,c]=M[0],[d,e,f]=M[1],[g,h,i]=M[2],det=a*(e*i-f*h)-b*(d*i-f*g)+c*(d
function transpose(M){return M[0].map((_,j)=>M.map(r=>r[j]));}
function matMulScalar(M,s){return M.map(r=>r.map(v=>v*s));}
function outer(col,row){return col.map(v=>row.map(w=>v*w));}
function clamp(x,min,max){return Math.max(Math.min(x,max),min);}

/* ---------- aircraft models ---------- */
const AIRCRAFT_DB={
  C172:{A:[[-0.05,0,-32.2],[0.2,-2.1,0],[0,1,0]],B:[[0.0],[-12.0],[0]],img:'images/c172.png',lqrK:
  B777:{A:[[-0.03,0,-32.2],[-0.01,-0.9,0],[0,1,0]],B:[[0.05],[-4.5],[0]],img:'images/b777.png',lqr
  A330:{A:[[-0.04,0,-32.2],[-0.015,-1.0,0],[0,1,0]],B:[[0.06],[-5.0],[0]],img:'images/a330.png',lq
};

/* ---------------- PID controller ---------------- */
class PIDController{
  constructor(){this.I=0;this.prevErr=0;this.D=0;}
  compute(ref,cur,_x,dt){
    const e=ref-cur,Kp=+pidKp.value,Ki=+pidKi.value,Kd=+pidKd.value;
    this.I=clamp(this.I+e*dt,-10,10);
    const N=10,rawD=(e-this.prevErr)/dt;
```

```
    this.D=(N*dt/(1+N*dt))*rawD+(1/(1+N*dt))*this.D;
    this.prevErr=e;
    return -(Kp*e+Ki*this.I+Kd*this.D);
  }
}

/* ----------- LQR controller (pre / Riccati) ----------- */
class LQRController{
  constructor(ac){this.ac=ac;this.rbPre=lqrPreMode;this.rbMan=lqrManMode;
    [this.rbPre,this.rbMan].forEach(r=>r.onchange=_=>this.updateK());
    [...document.querySelectorAll('#lqr-params input')].forEach(i=>i.oninput=_=>this.updateK
    this.updateK();}
  solveRiccati(A,B,Q,R,iters=50){
    let P=identity3x3();const Bt=transpose(B),R0=R[0][0];
    for(let k=0;k<iters;k++){
      const BtPB=matMul(matMul(Bt,P),B)[0][0],inv=1/(R0+BtPB);
      const Ktmp=matMulScalar(matMul(Bt,P),inv)[0];
      const AK=matAdd(A,scalarMul(outer(B.map(v=>v[0]),Ktmp),-1));
      P=matAdd(Q,matMul(matMul(transpose(AK),P),AK));
    }
    const BtP=matMul(Bt,P)[0],inv=1/(R0+matMul(BtP,B)[0][0]);
    return BtP.map(p=>inv*p);
  }
  updateK(){
    const m=AIRCRAFT_DB[this.ac],A=m.A,B=m.B;
    if(this.rbMan.checked){
      const Q=[[+q11.value,+q12.value,+q13.value],[+q21.value,+q22.value,+q23.value],[+q31.va
            R=[[+r11.value]];
      this.K=this.solveRiccati(A,B,Q,R);
      [lqrK1,lqrK2,lqrK3].forEach((e,i)=>e.value=this.K[i].toFixed(3));
    }else this.K=[+lqrK1.value,+lqrK2.value,+lqrK3.value];

    const BK=outer(B.map(v=>v[0]),this.K),Acl=matAdd(A,scalarMul(BK,-1)),
          inv=invert3x3(Acl),CNB=matMulVec(inv,B.map(v=>v[0]))[2]||1e-6;
    this.Nbar=-1/CNB;
  }
  compute(ref,_cur,x,_dt){
    const  r =ref*Math.PI/180,ufb=this.K.reduce((s,k,i)=>s+k*x[i],0);
    return clamp(-ufb+this.Nbar* r ,-AIRCRAFT_DB[this.ac].maxU,AIRCRAFT_DB[this.ac].maxU);
  }
}

/* ---------- FSF controller (Ackermann) ---------- */
class FSFController{
  constructor(ac){this.ac=ac;this.updateK();}
  updateK(){
    const m=AIRCRAFT_DB[this.ac],A=m.A,B=m.B,p=[+pole1.value,+pole2.value,+pole3.value];
```

```
    const a2=-(p[0]+p[1]+p[2]),a1=(p[0]*p[1]+p[0]*p[2]+p[1]*p[2]),a0=-(p[0]*p[1]*p[2]);
    const A2=matMul(A,A),A3=matMul(A2,A),
          Phi=matAdd(matAdd(matAdd(A3,scalarMul(A2,a2)),scalarMul(A,a1)),identity3x3(a0));
    const AB=matMul(A,B),A2B=matMul(A2,B),
          Mc=[[B[0][0],AB[0][0],A2B[0][0]],[B[1][0],AB[1][0],A2B[1][0]],[B[2][0],AB[2][0],A2B[2][0
          row=matMul([[0,0,1]],invert3x3(Mc))[0];
    this.K=matMul([row],Phi)[0];
  }
  compute(ref,_cur,x,_dt){
    if(this.ac!==aircraftSelect.value){this.ac=aircraftSelect.value;this.updateK();}
    if(this.lastPoles!==pole1.value+pole2.value+pole3.value){this.lastPoles=pole1.value+pole2.valu
    const  r =ref*Math.PI/180,e=[x[0],x[1],x[2]- r ];
    const u=-this.K.reduce((s,k,i)=>s+k*e[i],0);
    return clamp(u,-AIRCRAFT_DB[this.ac].maxU,AIRCRAFT_DB[this.ac].maxU);
  }
}

/* ------------------ simulator core ------------------ */
class Simulator{
  constructor(){this.dt=0.01;this.ctx=mainCanvas.getContext('2d');this.eCtx=errorCanvas.getContext
  reset(){this.ac=aircraftSelect.value;this.state=[0,0,0];this.errLog=[];this.img=new Image();this
  setController(){const t=controllerSelect.value;if(t==='PID')this.ctrl=new PIDController();if(t==
  step(){const mdl=AIRCRAFT_DB[this.ac],ref=+targetPitch.value,cur=this.state[2]*180/Math.PI;let u
  draw(){
    this.ctx.clearRect(0,0,600,400);const w=this.img.width*0.3,h=this.img.height*0.3;this.ctx.save
    this.eCtx.clearRect(0,0,400,200);
    if(this.errLog.length>1){this.eCtx.beginPath();this.errLog.forEach((e,i)=>{const x=i/(this.err
    this.pCtx.clearRect(0,0,400,200);
    if(controllerSelect.value==='FSF'){[pole1,pole2,pole3].forEach(el=>{const v=+el.value,x=200+v*
  }
  loop(){if(!running)return;this.step();this.draw();requestAnimationFrame(_=>this.loop());}
}

/* ------------------ DOM wiring ------------------ */
const mainCanvas=document.getElementById('main-canvas'),
      errorCanvas=document.getElementById('error-canvas'),
      poleCanvas=document.getElementById('pole-canvas'),
      aircraftSelect=document.getElementById('aircraft-select'),
      controllerSelect=document.getElementById('controller-select'),
      targetPitch=document.getElementById('target-pitch'),
      pidKp=document.getElementById('pid-kp'),
      pidKi=document.getElementById('pid-ki'),
      pidKd=document.getElementById('pid-kd'),
      lqrK1=document.getElementById('lqr-k1'),
      lqrK2=document.getElementById('lqr-k2'),
      lqrK3=document.getElementById('lqr-k3'),
      lqrPreMode=document.getElementById('lqr-pre-mode'),
```

```
      lqrManMode=document.getElementById('lqr-man-mode'),
      q11=document.getElementById('q11'),q12=document.getElementById('q12'),q13=document.getE
      q21=document.getElementById('q21'),q22=document.getElementById('q22'),q23=document.getE
      q31=document.getElementById('q31'),q32=document.getElementById('q32'),q33=document.getE
      r11=document.getElementById('r11'),
      pole1=document.getElementById('pole1'),pole2=document.getElementById('pole2'),pole3=doc
      simToggle=document.getElementById('sim-toggle'),resetBtn=document.getElementById('reset
      pidSection=document.getElementById('pid-params'),
      lqrSection=document.getElementById('lqr-params'),
      fsfSection=document.getElementById('fsf-params');

let sim=new Simulator(),running=false;

controllerSelect.onchange=_=>{
  [pidSection,lqrSection,fsfSection].forEach(s=>s.classList.remove('active'));
  if(controllerSelect.value==='PID')pidSection.classList.add('active');
  if(controllerSelect.value==='LQR')lqrSection.classList.add('active');
  if(controllerSelect.value==='FSF')fsfSection.classList.add('active');
  sim.setController();
};
aircraftSelect.onchange=_=>sim.reset();
simToggle.onclick=_=>{
  running=!running;
  simToggle.textContent=running?'Stop':'Start';
  if(running)sim.loop();
};
</script>
</body>
</html>
```

Appendix A

Source Code

sec:source

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Airplane Roll Simulation with LQR and Full State Feedback Control</title>
  <style>
    canvas {
      background-color: #e0f7fa;
      display: block;
      margin: 0 auto;
      border: 1px solid #000;
    }
    .controls {
      text-align: center;
      margin-top: 10px;
    }
    .hidden {
      display: none;
    }
  </style>
</head>
<body>
  <canvas id="airplaneCanvas" width="600" height="400"></canvas>

  <div class="controls">
    <label for="controllerType">Controller Type: </label>
    <select id="controllerType" onchange="toggleControllerSettings()">
      <option value="lqr">LQR</option>
      <option value="fsf">Full State Feedback</option>
    </select>

    <label for="roll">Desired Roll Angle (deg): </label>
    <input type="number" id="roll" value="10" step="1">

    <!-- LQR Parameters -->
    <div id="lqrControls">
      <label for="kp">LQR State-Cost Weighting (Q): </label>
      <input type="number" id="kp" value="0.5" step="0.1">
```

```
      <label for="kd">LQR Control-Input Weighting: </label>
      <input type="number" id="kd" value="0.2" step="0.1">

      <label for="ki">LQR Cross-Term Weighting (N): </label>
      <input type="number" id="ki" value="0.1" step="0.1">
    </div>

    <!-- Full State Feedback Parameters -->
    <div id="fsfControls" class="hidden">
      <label for="k1">FSF Gain K1: </label>
      <input type="number" id="k1" value="50.0" step="1">

      <label for="k2">FSF Gain K2: </label>
      <input type="number" id="k2" value="20.0" step="1">
    </div>

    <button onclick="startSimulation()">Start Simulation</button>
  </div>

  <p id="angleDisplay">Current Roll Angle: 0  </p>
  <canvas id="graphCanvas" width="600" height="200"></canvas>

  <script>
    const airplaneCanvas = document.getElementById("airplaneCanvas");
    const ctx = airplaneCanvas.getContext("2d");

    const graphCanvas = document.getElementById("graphCanvas");
    const graphCtx = graphCanvas.getContext("2d");

    const airplaneImage = new Image();
    airplaneImage.src = 'airplane_behind.png';

    const dt = 0.02;
    const a21 = -0.5;
    const a22 = -0.8;
    const b1 = 1.0;

    const A = [[1, dt], [a21 * dt, 1 + a22 * dt]];
    const B = [0, b1 * dt];

    let controllerType, desiredRoll, kp, kd, ki, k1, k2;
    let state = [0, 0];
    let integralError = 0;
    let angleHistory = [];
    let timeElapsed = 0;

    function lqrControl(state, desiredRoll, kp, kd, ki) {
```

```
  const rollError = desiredRoll - state[0];
  const rollRateError = -state[1];

  integralError += rollError * dt;

  const controlOutput = kp * rollError + kd * rollRateError + ki * integralError;
  return controlOutput;
}

function fsfControl(state, k1, k2) {
  const rollError = desiredRoll - state[0];  // Roll angle error
  const rollRateError = -state[1];           // Roll rate error
  const controlOutput = k1 * rollError + k2 * rollRateError;
  return controlOutput;
}

function updateState(controlInput) {
  const newRollRate = A[1][0] * state[0] + A[1][1] * state[1] + B[1] * controlInput;
  const newRollAngle = A[0][0] * state[0] + A[0][1] * state[1];
  state[0] = newRollAngle;
  state[1] = newRollRate;
}

function drawAirplane() {
  ctx.clearRect(0, 0, airplaneCanvas.width, airplaneCanvas.height);

  const centerX = airplaneCanvas.width / 2;
  const centerY = airplaneCanvas.height / 2;
  const rollAngle = state[0];

  if (airplaneImage.complete) {
    ctx.save();
    ctx.translate(centerX, centerY);
    ctx.rotate(rollAngle);

    const imgWidth = airplaneImage.width / 2;
    const imgHeight = airplaneImage.height / 2;
    ctx.drawImage(airplaneImage, -imgWidth, -imgHeight);

    ctx.restore();
  }

  document.getElementById("angleDisplay").innerText = `Current Roll Angle: ${(rollAngle
}

function updateGraph() {
  angleHistory.push({ time: timeElapsed, angle: state[0] * 180 / Math.PI });
```

```
  if (angleHistory.length > 600) angleHistory.shift();

  graphCtx.clearRect(0, 0, graphCanvas.width, graphCanvas.height);

  graphCtx.beginPath();
  graphCtx.moveTo(50, graphCanvas.height / 2);
  graphCtx.lineTo(graphCanvas.width, graphCanvas.height / 2);
  graphCtx.moveTo(50, 0);
  graphCtx.lineTo(50, graphCanvas.height);
  graphCtx.strokeStyle = "#000";
  graphCtx.stroke();

  graphCtx.beginPath();
  for (let i = 1; i < angleHistory.length; i++) {
    const x = 50 + i;
    const y = graphCanvas.height / 2 - angleHistory[i].angle;
    if (i === 1) {
      graphCtx.moveTo(x, y);
    } else {
      graphCtx.lineTo(x, y);
    }
  }
  graphCtx.strokeStyle = "#00796b";
  graphCtx.stroke();

  timeElapsed += dt;
}

function simulate() {
  let controlInput;
  if (controllerType === "lqr") {
    controlInput = lqrControl(state, desiredRoll, kp, kd, ki);
  } else if (controllerType === "fsf") {
    controlInput = fsfControl(state, k1, k2);
  }

  updateState(controlInput);
  drawAirplane();
  updateGraph();
  requestAnimationFrame(simulate);
}

function startSimulation() {
  controllerType = document.getElementById("controllerType").value;
  desiredRoll = parseFloat(document.getElementById("roll").value) * (Math.PI / 180);

  if (controllerType === "lqr") {
```

```
        kp = parseFloat(document.getElementById("kp").value);
        kd = parseFloat(document.getElementById("kd").value);
        ki = parseFloat(document.getElementById("ki").value);
      } else if (controllerType === "fsf") {
        k1 = parseFloat(document.getElementById("k1").value);
        k2 = parseFloat(document.getElementById("k2").value);
      }

      state = [0, 0];
      integralError = 0;
      angleHistory = [];
      timeElapsed = 0;

      simulate();
    }

    function toggleControllerSettings() {
      const controllerType = document.getElementById("controllerType").value;
      document.getElementById("lqrControls").classList.toggle("hidden", controllerType !== "
      document.getElementById("fsfControls").classList.toggle("hidden", controllerType !== "
    }

    toggleControllerSettings();  // Initialize controller settings visibility
  </script>
</body>
</html>
```

Figures

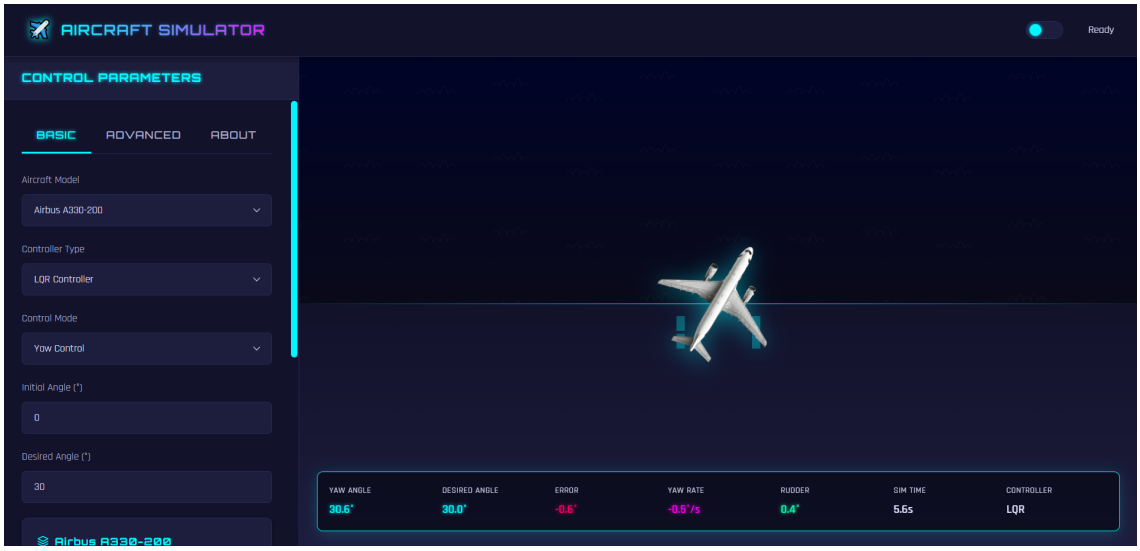

Figure A.1: *LQR Yaw Control Website Page*

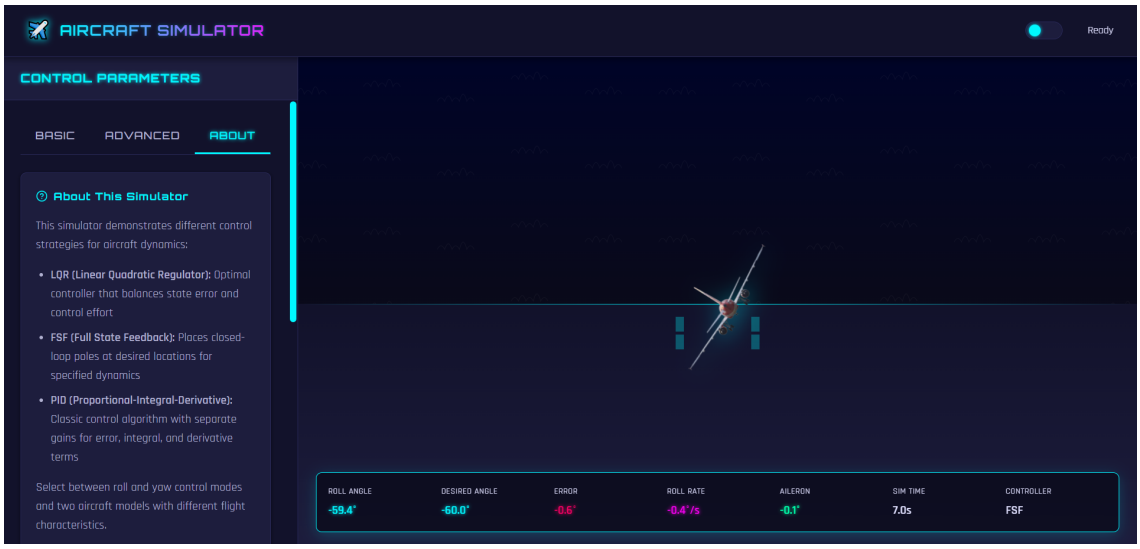

Figure A.2: *FSF Roll Control Website Page*

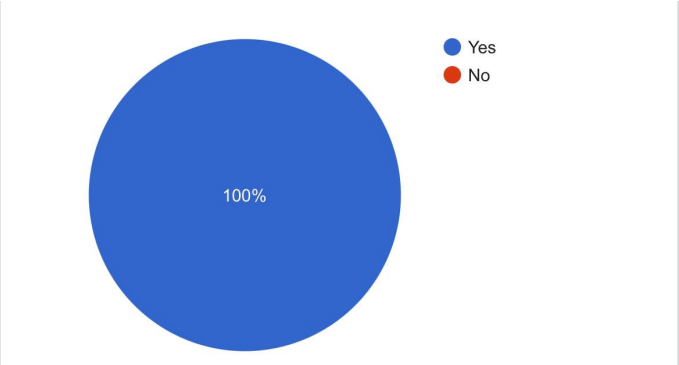

Figure A.3: *Consent of participation*

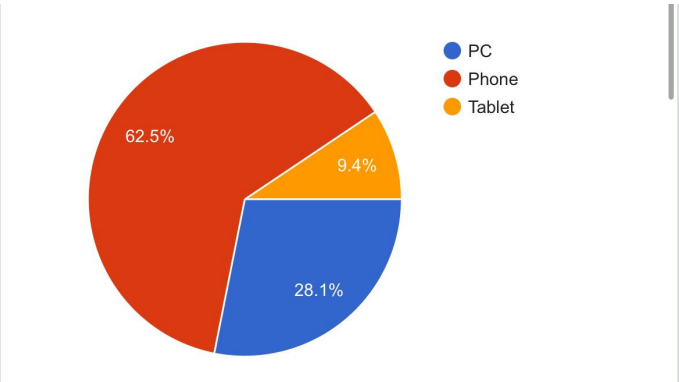

Figure A.4: *Device type*

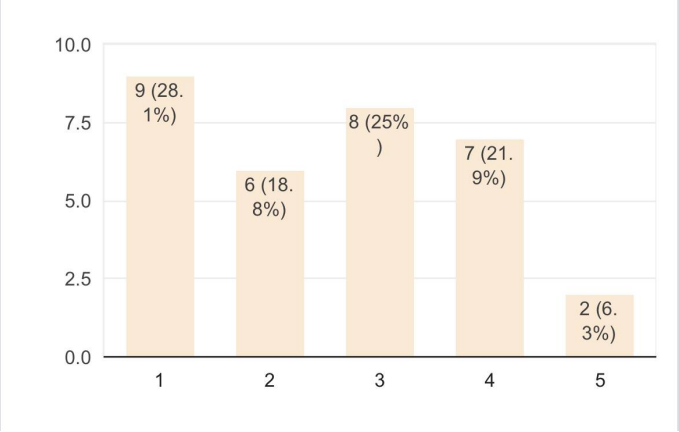

Figure A.5: *Familiarity with the aircraft control theory from 1 to 5*

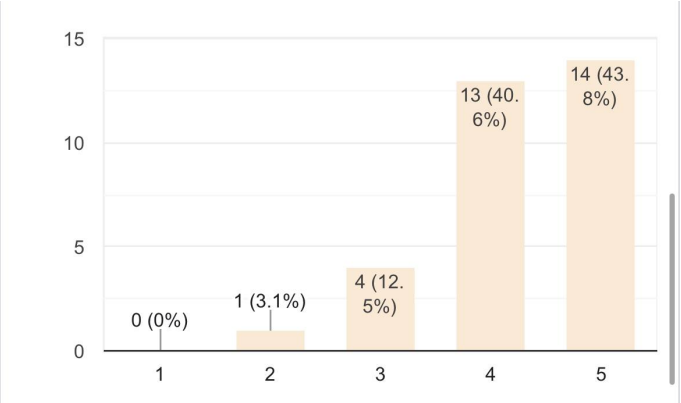

Figure A.6: *Rating the ease of navigation through the website*

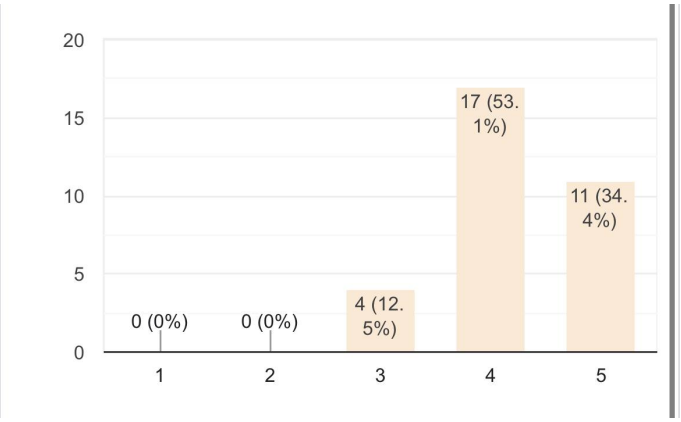

Figure A.7: *Rating the website loading*

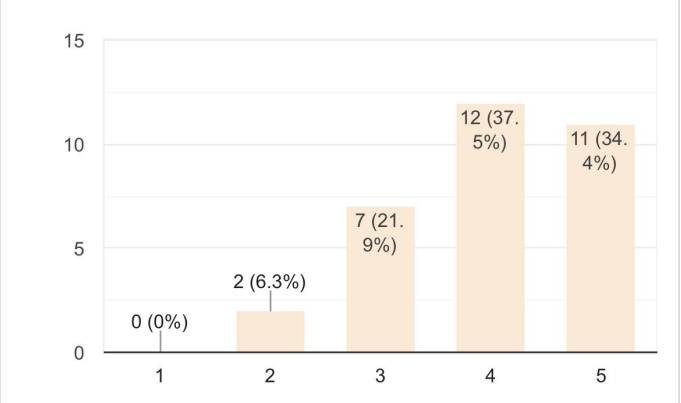

Figure A.8: *Rating the user interface of the website*

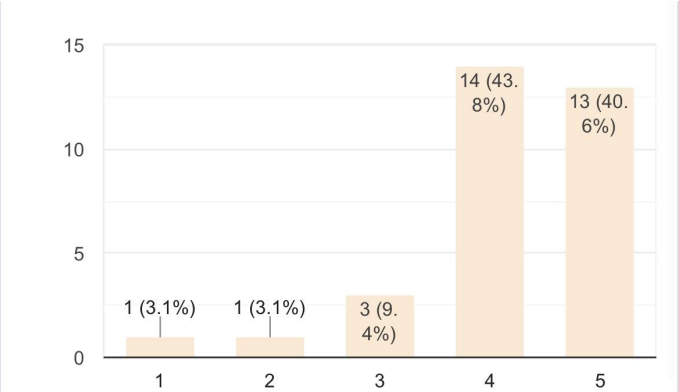

Figure A.9: *User ranking of the website's functionality*