F3

Faculty of Electrical Engineering
Department of Control Engineering

Bachelor's Thesis

Modeling, simulation and control of the VANGUARD CTU student rocket

Active roll control of a rocket

Ondřej J. Kukla
Cybernetics and Robotics

Prague, May 2024
Supervisor: doc. Ing. Martin Hromčík, Ph.D.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kukla Ondřej** Personal ID number: **503237**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Modeling, simulation and control of the VANGUARD CVUT student rocket

Bachelor’s thesis title in Czech:

Modelováním simulace a řízení studentské rakety ČVUT Vanguard

Guidelines:

The goal of the thesis is to develop and validate control laws for roll-axis attitude stabilization and control of the vehicle.
1. Develop and analyze the simulation model of the rocket in MATLAB/Simulink.
2. Propose simple control laws for attenuation of the roll motion.
3. Validate the feedback control in high-fidelity simulation using Kerbal Space Program.
4. Design and realize suitable HIL demonstrator.
5. Discuss possibilities of implementation of the developed and validated control laws on the target platform / on-board computer.

Bibliography / sources:

[1] Peter Zipfel, Introduction to Tensor Flight Dynamics, Third edition, John Wiley & Sons, Inc. 2020.
[2] Roger W. Pratt Johnson Flight Control Systems: Practical Issues in Design and Implementation. Institution of Engineering and Technology, 2000 https://app.knovel.com/kn/resources/kpFCSPIDI2/toc
[3] B. L. Stevens, F. L. Lewise. N. Johnson Aircraft Control and simulation . Third edition, John Wiley & Sons, Inc. 2016 https://ebookcentral.proquest.com/lib/cvut/reader.action?docID=4039442

Name and workplace of bachelor’s thesis supervisor:

doc. Ing. Martin Hromčík, Ph.D. Department of Control Engineering FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **07.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until:
by the end of summer semester 2024/2025

doc. Ing. Martin Hromčík, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgement / Declaration

There are many people whom I would like to express my thanks and gratitude, too many to mention them all by name. First of all I would like to thank my supervisor, doc. Ing. Manrin Hromčík, Ph.D., for his help and guidance throughout this work. I would also like to thank Ing. Tomáš Čenský, Ph.D. for facilitating us in the university's wind tunnel and his help and guidence with the execution of the practical portion of this work. I also thank the CTU space research team and its members for allowing the creation of this thesis, as a part of an ongoing team project. I would then like to thank and express my deepest of gratitude to my beloved, my family and friends for supporting me throughout this work and the course of my studies. Finally, I would like to thank the Lord for His blessings, the strength He provides and the guidance He bestows.

I, Ondřej J. Kukla, as the author of this thesis, declare this thesis and the work presented in it are of my own and that I have properly quoted all the sources and material used and have cited it in the bibliography. I further acknowledge and confirm that this work has been developed as a larger group project and as such all the contents in this theses linked to the work of others are laid out as such.

In Prague, 23. May 2024

.....

Abstrakt / Abstract

Tato práce se zabývá chováním raket během letu se zaměřením na dynamiku rotace kolem svislé osy rakety. Pro účely práce byl využit tenzorový přístup k letové mechanice a následně použit pro konstrukci letových simulací. Práce využívá Kerbal Space Program jako testovací prostředí, pomocí kterého dále studuje dynamiku systému. V tomto prostředí je také navrhnuto a otestováno několik regulátorů pro řízení rotace kolem svislé osy. Je představena studentská raketa, identifikována její dynamika a navržen a na letovém počítači implementován regulátor pro kontrolu rotace kolem svislé osy během letu.

Klíčová slova: Raketa, Tenzorová letová dynamika, kinematika rakety, řízení rotace kolem svislé osy.

This thesis investigates the behaviour of rockets in flight, mainly focusing on roll dynamics of the rocket along its vertical axis. The work utilizes the tensor approach to study flight dynamics of rockets and applies this logic in constructing flight simulations. The work also utilizes Kerbal Space Program as a testing environment, transforming it into a software in the loop simulator, using it to further study dynamics of the system. Multiple controllers to control the roll of the rocket along its vertical axis will be designed and tested in this environment. A student rocket will be introduced, its dynamics identified and a controller will be designed based of the study we will carry out in the Kerbal Space Program environment and will be implemented on the rocket's flight computer to stabilize its roll during flight.

Keywords: Rocket, tensor flight dynamics, rocket kinematics, SITL simulator, roll control.

Contents /

Tables / Figures

Chapter 1
Introduction

Active control of rockets has a long history originating in 1930s and gaining pace during the second world war. During the war the ingenuity of German scientists and megalomania of the high command produced many interesting designs seemingly ahead of its time. Many people know about the V-2 rocket, a first long range ballistic missile in the world controlled by a rudimentary computer and steered using gyroscopes, thrust vectoring and aerodynamic control surfaces [1]. Much less people know about the surface to air and air to air missile programs Germany established trying to defend itself from the growing Allied bomber threat devastating its cities and industry. Germany first fielded unguided air to air missiles like the Orkan on Messerschmidt 262's [2] and surface to air missiles like the Taifun [3] in the later stages of the war in Europe. Both examples of missiles meant to be fired on mass into the Allied bomber formations, hoping to score a lucky hit. Still they greatly increased the efficiency of the German defense, the Orkan increasing the Me 262's combat effectiveness by the factor of 3.3 [4]. The German scientists then improved the concept and developed guided antiaircraft missiles like the Ruhrstahl guided air to air missile [5] and Rheintochter R I surface to air missile [6].

Although brilliant, the German inventions failed to turn the tide of war, but they signaled the way ahead in the development of anti air and anti projectile capabilities. The United States quickly followed suit after the war and started development of their own antiaircraft missiles, culminating in the development of air to air missiles such as the AIM-9 Sidewinder, AIM-7 Sparrow and AIM-54 Phoenix throughout the cold war, proving themselves in many conflicts. The Germans did set the direction of future development during the second world war, as today's militaries are very much dependant on plethora of missile technologies employed in operations over all domains. As such missile control and guidance plays a very important part not only in military technologies of today, but also in civilian use, enabling us to deliver astronauts and payload to the Earths orbit and beyond.

1.1 Thesis goals

The purpose of this thesis is to model and simulate flight dynamics of Vanguard student rocket and to create control laws for control of roll rate of the rocket during flight utilizing the rocket's aerodynamic control surfaces. The theses will encompass the following goals.

- Model flight dynamics of the Vanguard rocket and construct a flight simulation in SIMULINK. We will model both the translational dynamics to learn about the flight profile of the rocket and the rotational dynamics, specifically the roll dynamics of the rocket, so that we can identify the system for purpose of design of control laws.
- Transform Kerbal Space Program into a software in the loop (SITL) simulation environment which could be used to implement and test control laws and more complex

control programs. Establish if the environment is comparable to the real world and if it can be considered a high fidelity simulation.
- Test multiple types of control laws in the Kerbal Space Program simulation environment to establish which is best to be used on the real life rocket.
- Design and propose control laws for control and attenuation of the rolling motion of the Vanguard rocket.
- Implement the designed control laws on the rocket's flight computer and propose an experiment to validate the design of the system.

Chapter 2
Flight physics of a rocket

In order to understand the flight of a rocket or an airplane we must immerse ourselves in in flight dynamics which is a field of study focusing on describing forces acting on the craft and its responses to these forces, culminating in description of flight trajectories and dynamic responses of the system to input. As such we consider dividing the dynamics into two sections. Point-mass dynamics, which simplifies the craft only to its point of mass and applies all forces to this point, resulting in the trajectory of the craft being calculated. We refer to such trajectories as three degrees-of-freedom trajectories. The other section considers the craft to be a rigid body and takes into account both its translational and rotational degrees-of-freedom, we talk about six degree-of-freedom dynamics [7, pg. xvii].

In this chapter we will investigate both the point mass dynamics and the rigid body dynamics of rockets. With the point-mass dynamics we will put together equations of translational motion, which we will use in further chapters to simulate a three degree-of-freedom trajectory of a rocket. We will also utilize the rigid body dynamics to grasp the understanding about the rotational motions of such craft, mainly the roll motion along the crafts vertical axis. For these purposes we will utilize the tensor approach to flight dynamics.

2.1 Tensor approach to flight dynamics

Vector mechanics is commonly used to describe movement of objects in space and utilizes vectors to describe magnitude and displacement of forces and qualities of objects as acceleration and velocity. This method fundamentally relies on knowing the coordinate system and it's origin beforehand to be able to describe the surrounding phenomena. Indeed a reference is required for us to be able to determine displacement and magnitude of forces, velocities of objects etc. Commonly when approaching a problem within the confines of classical mechanics we start to develop the equations describing the object of interest and its surroundings with a coordinate system already in mind. This can then bring problems when we want to move into a different coordinate system or reference frame or we are required to account for relativity.

As the covariance principle states: *"The general laws of nature are to be expressed by equations which hold good for all systems of co-ordinates, that is, are co-variant with respect to any substitutions whatever (generally co-variant)."*, [8, pg. 153]. As we can learn from further publication [9], this means that natural laws are to be covariant, meaning invariant, to continuous and time dependent transformations. Meaning that natural laws and equations describing them should maintain their behaviour and form regardless of which coordinate system we found ourselves in and transformations between them. Sadly, although vector mechanics can be expanded to satisfy this principle, it does not satisfy it by it self. This is where tensor mechanics comes into play as it does satisfy the covariance principle from the get go, [7, pg. xviii].

Tensor mechanics utilizes tensors to describe laws of physics independent on coordinate systems. Both scalars and vectors are considered tensors of rank zero and rank one respectively. When the basic laws are established we then proceed with application of a reference frames onto the laws. Only in the last step we introduce our chosen coordinate system into the equations. While working with Newtonian dynamics and flight dynamics we are also able to easily switch between reference frames, the transformations between them, in the form of rank two tensors, are already wildly known. As we will apply a rather specific notation to construct our equations and movement laws affecting the objects of interest, in our case rockets, please refer to the list of symbols in the appendix. Our calculations and later simulations are based on this notation laid out by doctor Peter Zipfel in his book Introduction to Tensor Flight Dynamics, [7].

2.2 Rocket point-mass dynamics

To describe the motion of an unguided rocket or missile in flight we need to establish its equations of motion. As the rocket in our case never leaves the atmosphere nor travels larger distances and the time of flight is short, we can use a flat-earth representation as our reference frame. We call this frame the local level coordinates and denote it as 'L'. We also use a velocity reference frame 'V' to describe the movement of the rocket. This reference frame is linked with the velocity vector by its first axis, the velocity vector originates in the rocket's center of mass, as can be seen in picture 2.1. As the rocket is unguided we can expect that there is no active input influencing the pitch or yaw angles, therefor not influencing the flight path and so we can think of the rockets motion as of a three degree of freedom point-mass dynamics [7].

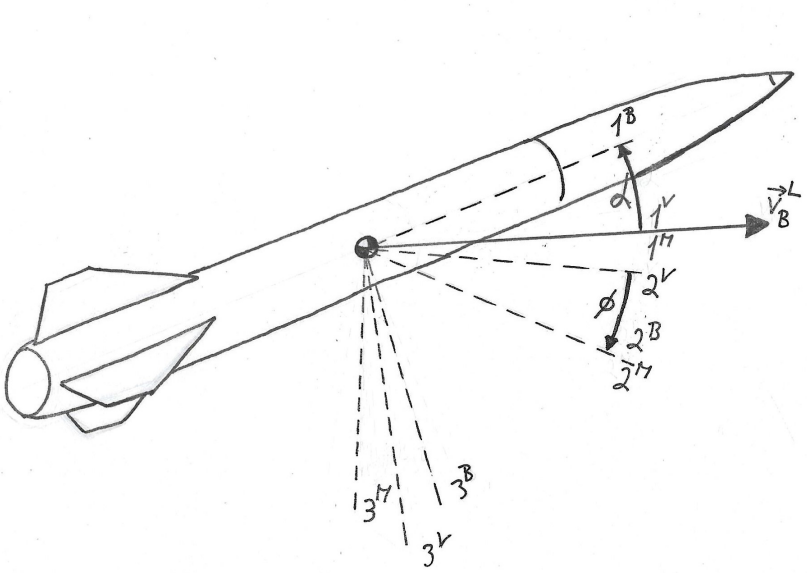

Figure 2.1. Reference frame connected with the rocket

To build the equations of motion of the rocket, which are for all intents and purposes a set of differential state equations of the system, we need to establish, which forces do affect the rocket. First of these forces are the propulsive forces. As the engine is directly linked to the rocket, it is best do describe these forces in the rocket's body frame 'B', which is directly tied to it. The second type of force effecting the rocket

is the aerodynamic force, being the culmination of all the aerodynamic affects acting on the object during its flight. The aerodynamic behaviour is greatly influenced by the angle of attack of the rocket α. Whence we model the aerodynamic forces in the maneuver plane 'M', which is shifted from the rockets body frame by the angle of attack around its second axis. The last of the forces affecting the flight of the rocket is gravity, which is best described in the 'L' reference frame as the force itself is tied to the Earth. To recapitulate the movement of the rocket is directed by the following term.

$$mD^V\mathbf{v}_R^L = \mathbf{f}_p + \mathbf{f}_a + \mathbf{f}_g \tag{1}$$

When we then proceed to apply the velocity frame to all elements we get:

$$m[D^V\mathbf{v}_R^L]^V = [T]^{VB}[\mathbf{f}_p]^B + [T]^{VM}[\mathbf{f}_a]^M + [T]^{VL}[\mathbf{f}_g]^L \tag{2}$$

Let us first resolve the left side of said equation, the rotational derivative must develop, [7, pg. 128].

$$m[D^V\mathbf{v}_R^L]^V = m[\frac{d\mathbf{v}_R^L}{dt}]^V + m[\Omega^{VL}]^V[\mathbf{v}_R^L]^V = m\begin{bmatrix}\dot{V}\\ \dot{\chi}V\cdot cos\gamma\\ -V\dot{\gamma}\end{bmatrix} \tag{3}$$

Where the $[\Omega^{VL}]^V$ is a skew-symmetric form of the vector $[\omega^{VL}]^V$ describing the angular rotation of frame L wrt. frame V. Now we can focus on the right hand side of the equation. Let us look once again on the relationship between reference frames B, M and L, as seen in pictures 2.1 and 2.2.

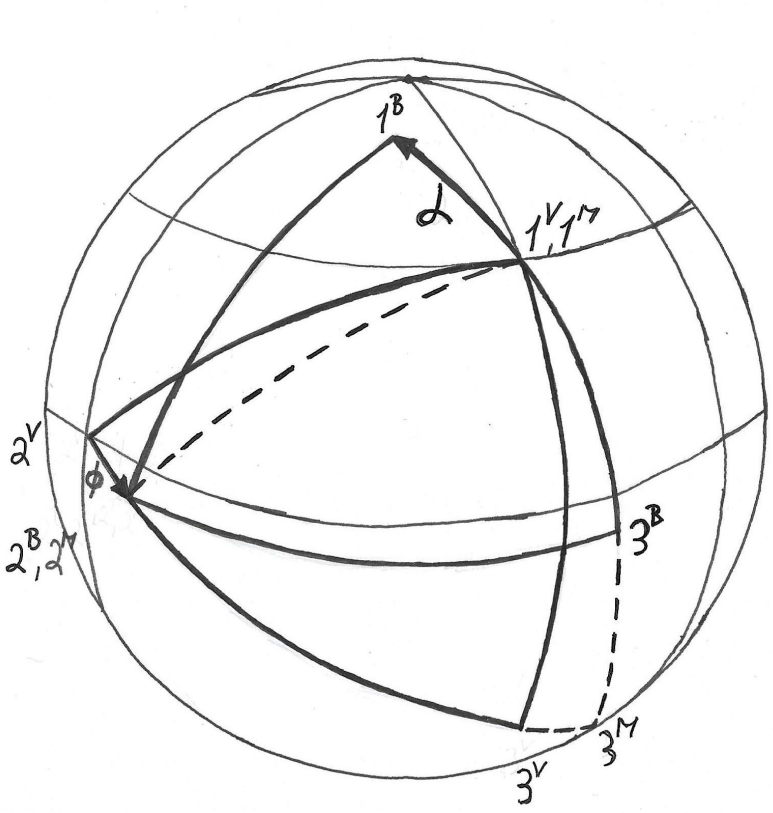

Figure 2.2. Relationship between reference frames B, M and V depicted by the orange peel method, drawn according to [7, pg. 120].

As the rocket does not bank and so its bank angle ϕ is equal to zero. We can also expect that the rocket, being unguided and designed to be marginally stable, will follow its direction of flight and so the angle of attack α will be close or equal to zero. Making this assumption we can simplify our problem by uniting the reference frames B, M and V, only focusing on the V frame. By plugging in the result from equation (3) to equation (2) and then applying this assumption enables us to unify the reference frame, dividing the whole equation by the rockets mass we arrive at our state equations describing the movement of the rocket (4). However, the work is not yet finished, we still need to finalize the right hand side of the equation.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}^V = \begin{bmatrix} \dot{V} \\ \dot{\chi} V \cdot cos\gamma \\ -V\dot{\gamma} \end{bmatrix} = \frac{1}{m}([\mathbf{f_a}]^V + [\mathbf{f_p}]^V) + [T]^{VL}[\mathbf{g}]^L \tag{4}$$

As seen in equation (4) the acceleration is greatly dependant on variable V which is equal to the norm from the velocity vector of the rocket originating in its center of mass.

$$V = |\mathbf{V}|$$

As can be also seen in the equation the two forces affecting the rocket are denoted in the velocity frame, but the gravitational acceleration is denoted in the local level coordinates. This is because it is easier to describe it in this frame, as can bee seen in the following expression.

$$[\mathbf{g}]^L = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

The gravitational acceleration in the L frame is simply a vector pointing strait down of a given magnitude. In our case the magnitude is $g = 9.8066\ [m/s^2]$. To work with the gravitational acceleration in the velocity frame we need to first transform it. The relationship between the V frame and the L frame can be seen in picture 2.3. As can be seen the relationship between the two reference frames is directed by the heading angle χ and path angle γ of the craft. From this we establish the transformation tensor T, from the local level coordinate frame L to the velocity frame V, resulting in matrix $[T]^{VL}$.

$$[T]^{VL} = \begin{bmatrix} cos\gamma \cdot cos\chi & cos\gamma \cdot sin\chi & -sin\gamma \\ -sin\chi & cos\chi & 0 \\ sin\gamma \cdot cos\chi & sin\gamma \cdot sin\chi & cos\gamma \end{bmatrix} \tag{5}$$

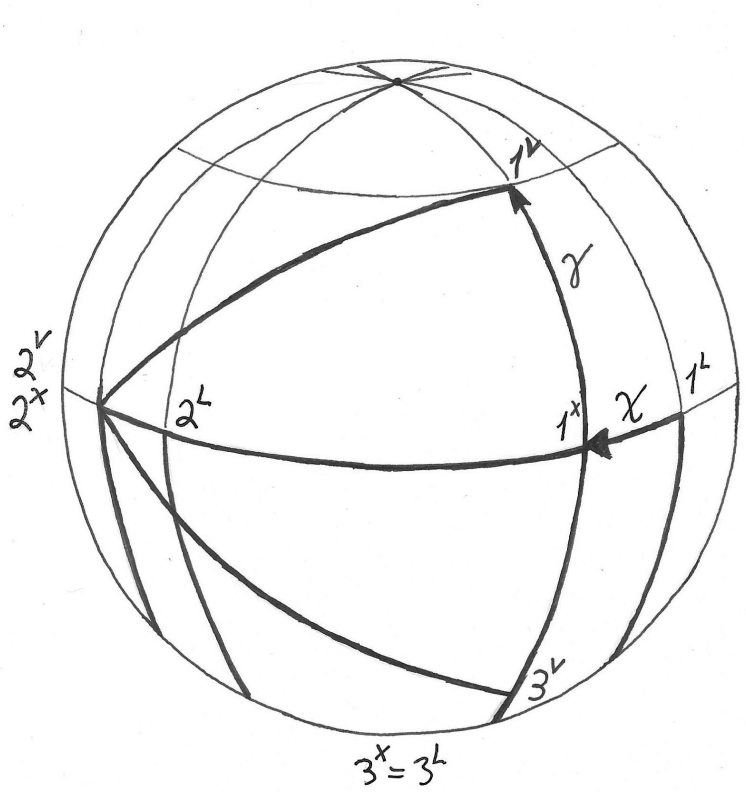

Figure 2.3. Relationship between reference frames V and L utilising an intermediate frame X, depicted by the orange peel method, drawn according to [7, pg. 113].

After transforming the vector of gravitational acceleration $[\mathbf{g}]^L$ to the velocity frame we get vector $[\mathbf{g}]^V$, which is the form we need, as equation (4) is wholly expressed in the velocity frame.

$$[\mathbf{g}]^V = g \begin{bmatrix} -sin\gamma \\ 0 \\ cos\gamma \end{bmatrix} \tag{6}$$

We can now focus on the other two, main forces beside gravity which affect the flight of the rocket, the aerodynamic and propulsion forces. It is easy to express these forces directly in the velocity frame. As we expect, as stated before, that the angle of attack is equal or close to zero, then the aerodynamic force $\mathbf{f_a}$ simplifies only to one component, which is directly tied to the velocity vector $\mathbf{V}$ and corresponds to the aerodynamic resistance in the direction of travel. Then the aerodynamic force effecting the rocket with respect to the velocity frame is given as follows.

$$[\mathbf{f_a}]^V = \bar{q}S \begin{bmatrix} -C_D(M) \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

Where $\bar{q}$ is the dynamic pressure, S is the total frontal area of the rocket and $C_D(M)$ is the frontal aerodynamic drag coefficient dependant on the Mach number.

$$\bar{q} = \frac{1}{2}\rho V^2$$

The propulsive force $\mathbf{f_p}$ is similarly tied to the direction of travel thanks to the zero angle of attack assumption. The force than has only one component in the direction of

travel of the rocket.

$$[\mathbf{f_p}]^V = \begin{bmatrix} I_{SP}\dot{m}_f g \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

The propulsive force with respect to the velocity frame $\mathbf{f_p}$, as described in expression (8), is dependant on the specific impulse of the rocket engine I_{SP} given in seconds, the mass flow rate $\dot{m}_f$ in kilograms per second and the magnitude of gravitational acceleration g. This expression gives the thrust given by the rocket engine and so can be simplified as seen in equation (9) which gives us the current thrust of the engine in Newtons.

$$T = I_{SP}\dot{m}_f g \tag{9}$$

We can now substitute expressions (6), (7) and (9) into equation (4) which gives is the following term.

$$\begin{bmatrix} \dot{V} \\ \dot{\chi}V \cdot cos\gamma \\ -V\dot{\gamma} \end{bmatrix} = \begin{bmatrix} -\frac{\bar{q}S}{m}C_D(M) + \frac{T}{m} - g \cdot sin\gamma \\ 0 \\ g \cdot cos\gamma \end{bmatrix}$$

As can be seen, only the acceleration in direction of travel and the angular velocity of the path angle do change, the heading angle being zero, that is the flight path of the rocket not being dependant on it. This then leads us to the final representation of the system as seen in equation (10).

$$\begin{bmatrix} \dot{V} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} -\frac{\bar{q}S}{m}C_D(M) + \frac{T}{m} - g \cdot sin\gamma \\ -\frac{g}{V} \cdot cos\gamma \end{bmatrix} \tag{10}$$

To get the position of the rocket in flight with respect to the local level coordinate frame we simply integrate term (11), putting the origin of our coordinate system at the base of the launch pad.

$$[\frac{ds_{RL}}{dt}]^L = \overline{[T]}^{VL}[V_R^L]^V \tag{11}$$

2.3 Rocket rigid body dynamics

Having established the laws guiding the translational motion of the rocket, thinking about the craft as of a point mass, we move on to the study of the craft as of a rigid body. As it is our focus in this work to study and develop control laws to stabilise the rolling motion of the rocket, we want to develop the structure and laws responsible for the roll rate of the rocket in flight. When studying rigid body mechanics of rockets most of the research is focusing on the dynamic response in the pitch and yaw movement of rockets in response to disturbances, as wind etc. Lot of this data also being established purely experimentally [10] or using CAD computer programs. As such, it was difficult for us to find out much information about rotational dynamics of rockets for construction our own simulations tailored to our rocket to later base our control laws on. Therefore we borrowed from the world of aircraft. Although not the same, the equations describing the movement of an airplane are not that different from equations guiding the movement of rockets. Equation (12) shows the whole set of differential equations directing the movement of an airplane in all three rotational axes.

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{\bar{q}Sb}{I_1}\left(C_{l_\beta}\frac{v}{u_r} + C_{l_p}\frac{b}{2u_r}p + C_{l_{\delta a}}\delta a + C_{l_{\delta r}}\delta r\right) \\ \frac{\bar{q}Sc}{I_2}\left(C_{m_\alpha}\frac{w}{u_r} + C_{m_q}\frac{c}{2u_r}q + C_{m_{\delta e}}\delta e\right) \\ \frac{\bar{q}Sb}{I_3}\left(C_{n_\beta}\frac{v}{u_r} + C_{n_r}\frac{b}{2u_r}r + C_{n_{\delta r}}\delta r + C_{n_{\delta a}}\delta a\right) \end{bmatrix} \tag{12}$$

As can be seen from this set of equations, the roll of the plane p, the pitch q and yaw r are cross-connected and influence each other. The most well known of these connections is the Duch roll dynamic where the plane starts to roll when the rudder of the vertical stabilizer gets deflected. As we do not actively deflect the rocket in the pitch nor the yaw axis and we assume that the influence of wind can be ignored, we can simplify the equations and terminate the cross-connections, leaving us with a simplified version of the roll dynamics differential equation (13).

$$\dot{p} = \frac{\bar{q}Sb}{I_1}\Big(\frac{b}{2u_r}C_{l_p}p + C_{l_{\delta a}}\delta a\Big) \tag{13}$$

The flight dynamics equations for rockets are often non-dimensionalized by the referential length of the wing, which can be seen under the label l_m on picture 2.4, instead of the wingspan b or elevator span c as it is often done with aircraft. We will denote the referential length of the wing as l, [7, pg. 173]. We also relabel the forward speed of the rocket, from u_r to V, to be consistent with other calculations we made.

$$\dot{p} = \frac{\bar{q}lS}{I_1}\Big(\frac{l}{2V}C_{lp}p + C_{l\delta a}\delta a\Big) \tag{14}$$

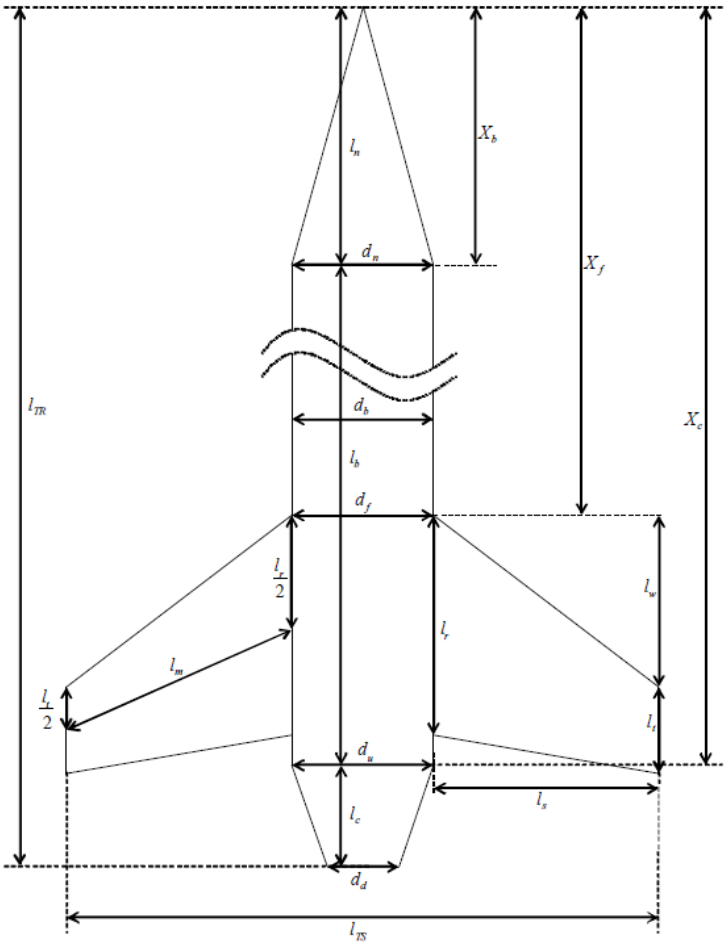

Figure 2.4. Schematic of a simple rocket showing the principal dimensions, taken from [11, pg. 8].

Chapter 3
Flight vehicle

CTU Space Research is a student team at the Czech Technical University in Prague, focusing on rocketry research and rocket development. The team takes part in regional as well as international rocketry contests, such as the European Rocketry challenge (EUROC), held in Portugal and organised by the Portuguese space agency. For this purpose the team developed the Illustria rocket, almost four meter high, thirty kilogram, high power rocket with target apogee of three kilometers. As EUROC is a constructors competition every part of the rocket must be developed by the team. No part can be developed externally. As such, the team prouds itself not only designing and constructing the airframe of the rocket, but also the propulsion system and the avionics. As it is unpractical to test flight smaller components, such as various avionics parts, on the Illustria platform, a smaller rocket was developed to enable the team to test-flight the components and fix any errors in design before proceeding with upgrading the Illustria platform. This testbed rocket is called Vanguard.

Since Vanguard is a team project this work is also closely tied to the work of other team members. Therefore the author would like to dedicate this chapter to their hard work and dedication while designing and building the Vanguard rocket. And while the author of this work had some input on the design of the hardware described in this chapter, it mostly encompasses the fruits of labor of fellow colleagues, who made the existence of this work itself possible.

The purpose of this chapter is to familiarize ourselves with the platform and its capabilities so that we gain an understanding of the system at hand and become able to create flight simulations and propose control laws in later parts of this work.

3.1 Vanguard advanced technology testbed platform

The Vanguard advanced technology testbed is a testing rocket mainly used to test new avionic components. Part of the reason for the birth of the Vanguard program is the long term push to develop and implement active control systems into the teams rockets. Something which has been deemed as too much of a risk to attempt directly on the Illustria rocket, while the concept remained untested. This way we can proceed with development of active control systems, such as active stabilisation of the rocket on a smaller and easier to fly platform, enabling us to test-flight the system multiple times while keeping the costs down. The rocket is made mainly of carbon fiber, this includes the main body, the stabilisation fins and the nosecone. It is powered by a solid rocket motor utilizing a stainless steel nozzle. The aim for it being maximum reusability of the whole system. This is again because each newly developed system will require multiple flights to prove it is working properly and effectively. One of the newly developed systems for the rocket is the Active Roll Control section, ARC for short. Picture of the rocket with the ARC section can be seen on figure 3.1.

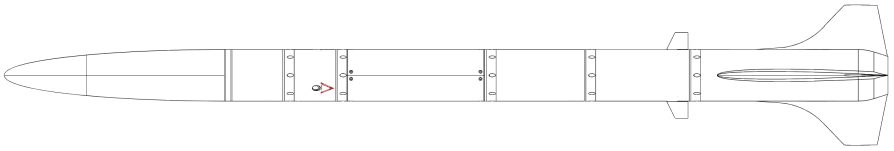

Figure 3.1. Schematic of the Vanguard rocket with included ARC section for roll control

As previously mentioned in the chapter 1, introduction, the aim of this work is to implement an active control system to the Vanguard rocket, controlling the roll rate of the rocket during flight. For this a Active Roll Control section was designed by the structures department, including the design of the control surfaces to minimise frontal drag and optimise action input. Our work is then focused on the design the controller which shall be implemented on the rocket's flight computer. To accomplish this we need to perform various tasks prior to designing the controller itself. One of the tasks being a development of a flight simulation to better understand the movement of the rocket and the speeds involved during flight. This data was then passed to the structures department, which then considered it during their own design process of the Active Roll Control section. The simulation data can be viewed in following section, section 4.1. To construct such simulation we first need to get the basic data about the rocket. It must be noted that the data written bellow and used as input data for our simulation, although processed by us, originate from the structures and propulsion departments of the CTU Space Research team, as they have provided them to us.

The Vanguard rocket is 1620 millimeters in length and has a outer diameter of 93 millimeters, its inner diameter is 90 millimeters. The frontal crossection of the rocket is eaqual to $6.36 \cdot 10^{-3}\ m^2$. The rockets surface is mainly comprised of carbon fiber, this includes a large part of the main body, the main stabilisers and the nosecone. The avionics section is covered by glass fiber to allow wireless communication and data transfer during the flight. As can be seen in figure 3.1, the nosecone is rounded, this is because the rocket is designed to reach only subsonic speeds around Mach 0.478. Mentioning speed, we were provided with the drag coefficient needed to calculate the aerodynamic drag acting on the rocket during flight, meaning we were provided both with direct simulation data from a fluid simulation environment and the polynomial representation resulting from said simulation. The dependence of the drag coefficient on the velocity of the rocket can be seen in figure 3.2.

Returning to figure 3.1 there can be also seen that the rocket is broken up into several parts by multiple rings, these are RADAX junctions. The RADAX junctions were developed within the CTU Space Research team and enable for a great level of modularity in the platform. For example enabling us to fly the rocket in multiple configurations and with fast swapping between these configurations. They also allow for easy transport of the rocket. The junctions on the vanguard rocket are 3D printed plastic. This leaves us with the total dry mass of the rocket at 5 kilograms. This weight does not include the mass of the propellant at liftoff, which is taken into account in the total wet mass of 6.04 kilograms. We were also provided with the expected propellant burning rate to calculate the change in mass during flight and with the thrust curve of the motor at sea level, this data can be seen in figures ?? and 4.2.

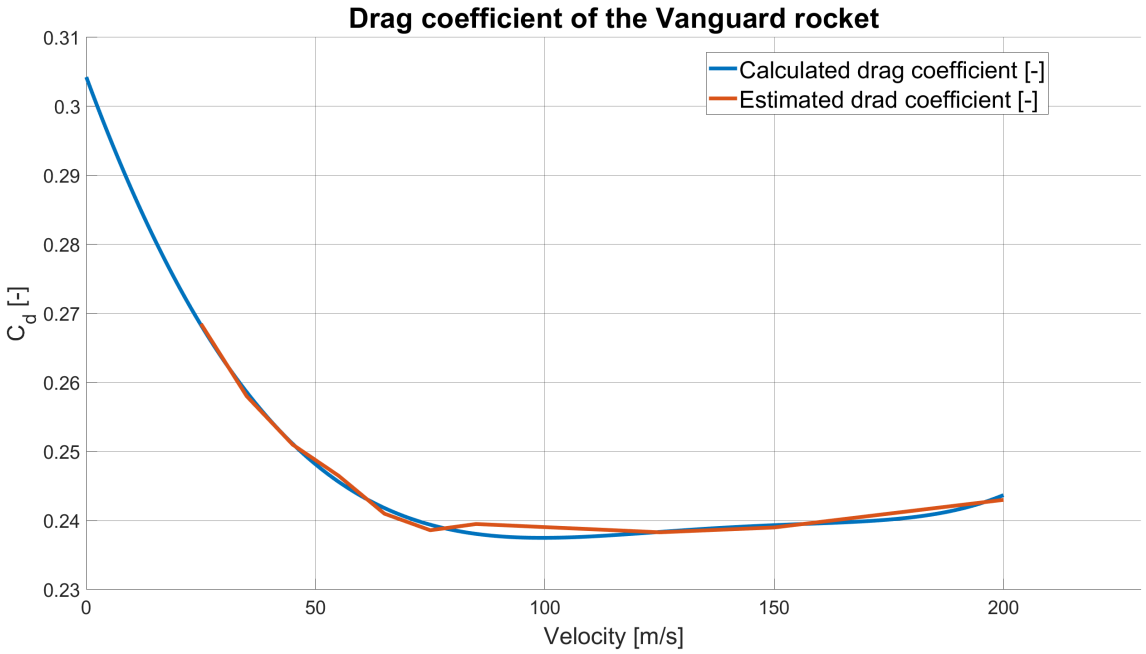

Figure 3.2. Drag coefficient of the Vanguard rocket estimated from fluid simulation and calculated using a polynomial approximation

3.2 Concept architecture and Design of the Active roll control section of the Vanguard rocket

As we have established one of the main motivations for the conception of the Vanguard program was to introduce and prove an active control system and to turn the rocket into a demonstrator of such a system. Many options were discussed, including thrust vectoring and deployable airbrakes for speed and altitude control. Upon the authors input it was decided to pursue a path of control via aerodynamic control surfaces. As a task of fully controlling a rocket in all three rotational axis is quite complex it was decided to first implement a simpler system designed to control just one of the rotational axis. The roll axis was then a strait forward choice as the change in roll of the rocket contributes the least out of the rotational axis to any changes in the rockets translational motion. Meaning that the rocket's flight path would be the least affected in case of any failures of the system. The author has taken the responsibility of system analysis and proposal of control laws leading to a design of a controller for control of the the roll rate of the rocket. Meanwhile, the task of designing and building the hardware enabling us to execute such control was put to the colleagues from the structures department. The diagram describing roll rate control logic can be seen in figure 3.3.

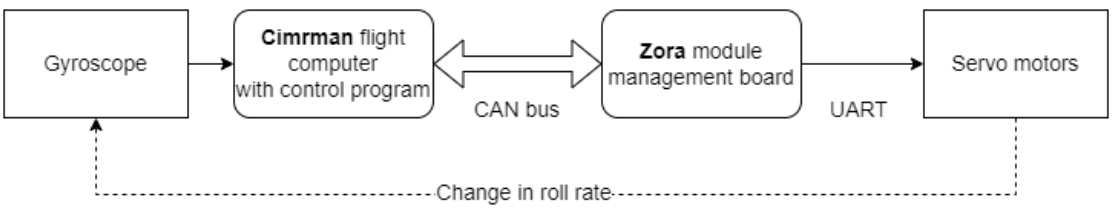

Figure 3.3. Control diagram with the feedback loop for control of roll rate of the rocket

The task laid out before the structures department was then to design a section of the rocket housing the servo motors, as well as the Zora module management board, for controlling the motors. It was also their task to design the control surfaces to be mounted on the servo motors. For the servo motors we have chosen Waveshare ST3215 serial bus servo motors. A 3D printed mount was created for mounting of the servo motors and the Zora board, this can be seen in figure 3.4. This initial design allows for attachment of the electronics to the section's top RADAX connector. Although it provides enough rigidity in the vertical plane it gives enough flux and allows for side movement of the assembly. For this reason branches must be created on the sides to hold the assembly against the housing. This version of the compartment was not yet available as of the submission of this work. The housing can be seen in figure 3.5.

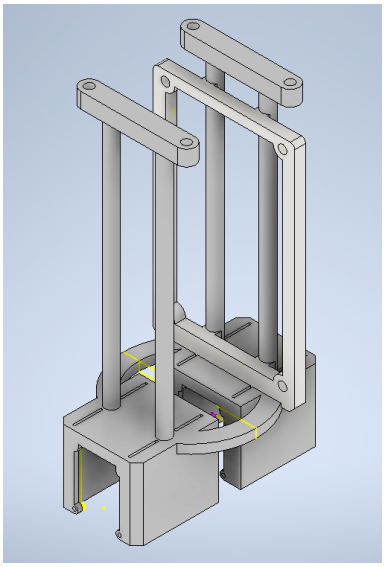

Figure 3.4. Mount for servo motors and module management board in the ARC section

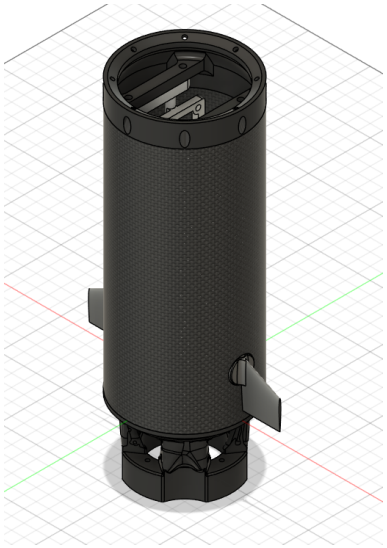

Figure 3.5. Outer housing for the ARC section with the control surfaces mounted

The housing itself consists of carbon fibre tube with an inner diameter of 93 milimeters. The housing has holes on the sides opposed from each other enabling mounting of control surfaces to the servo motors. A standard female RADAX connector is mounted on top and a modified male radax connector is mounted on the botom of the section to allow for mounting of the engine. As can be seen from figure 3.1, the ARC section is mounted quite low on the rocket. This is so, that the control surfaces are close to the center of mass of the rocket and don't have as much leverage as to flip the rocket in case of a malfunction such as failure of the servomotors, destruction of one of the control surfaces, control program bugs etc.

The final and the most important part of the design are the control surfaces. Two variants of control surface profiles were designed. A smaller primary version and a large backup profile to be used in case of an insufficient force being delivered from the surfaces. These two designes can be seen in figures 3.6 and 3.7.

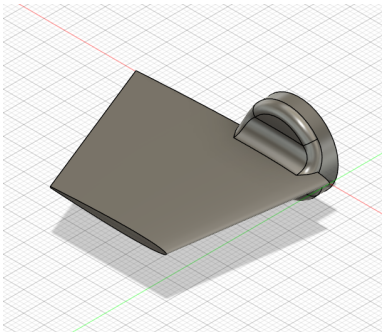

Figure 3.6. A primary design of the control surfaces for the ARC section

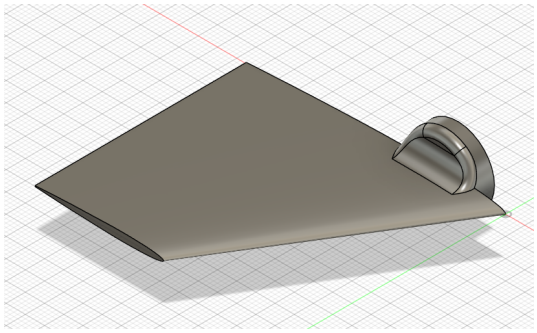

Figure 3.7. A secondary design of the control surfaces for the ARC section intended for maximalization of delivered force

The control surfaces were printed on a resin 3D printer. The overall surface finish was good but the brittleness of the resin resulted in chipping while drilling holes in the bases to allow for mounting to the servomotors.

Chapter 4
Flight simulation of the Vanguard rocket

In this work we have so far dealt with the physics guiding the motion of rockets and missiles throughout their flight and we have introduced the Vanguard platform. We have familiarized ourselves with basic characteristics of the Vanguard rocket and have taken a closer look at the active roll control section designed by fellow colleagues. Now it comes to us to construct various simulations to simulate flight characteristics of the rocket and get to understand the dynamics affecting its flight. We will study the translational motion of the rocket and then extend our view to rotational motion where we will focus on the roll dynamics of the rocket. It is possible for us to use this decoupled approach as we assume that the rocket will hold the angle of attack close to zero throughout its flight and that it will not perform any pitch or yaw motions. This assumption allows us to eliminate any dependence of translational motion on rotational motion of the rocket and simulate it independently.

4.1 Modeling translational flight dynamics of the rocket

We will construct a three degree of freedom, translational flight, simulation of the Vanguard rocket in SIMULINK. This simulation will be based mainly on the equations (10) and (11) from section 2.2. The state equations (10) represent the dynamics of the system and differential equations (11) represent the position of the rocket with respect to the launch site. This position is ultimately the output of our simulation together with other useful data as velocity, path angle, etc. As we are interested in the ascent stage of the flight, we will not model any chute or other recovery system into the simulation, letting the rocket fall ballistic. Let us start with the inputs to the simulation, with the input variables such as engine thrust, which changes during flight, and constants such as dry mass and wet mass of the rocket, which we mentioned in the previous chapter, defining the change in propellant mass during flight. As well as the influence of the , which greatly influences the flight as atmospheric density changes because of the increase in altitude as well as potential inluence of other reasons. The implementation in the simulation of such influences on the rocket during flight can be viewed in figure 4.1.

We first load the thrust curve of the motor at sealevel and propellant mass change into the simulation. We have been provided those data by the propulsion department in form of Microsoft Excel tables and have transformed them using MATLAB into timeseries type objects, which we then loaded into the simulation. As the thrust and propellant mass timeseries are of a finite length and are used in later calculations we need to externally terminate them or shift the source of the input upon the engines cutoff. For this purpose we included a unit square wave lasting from the moment of ignition to the engines burn out. The change of mass of the rocket during flight was approximated as a linear change in mass from the wet mass at time $t = 0\ s$ to the dry

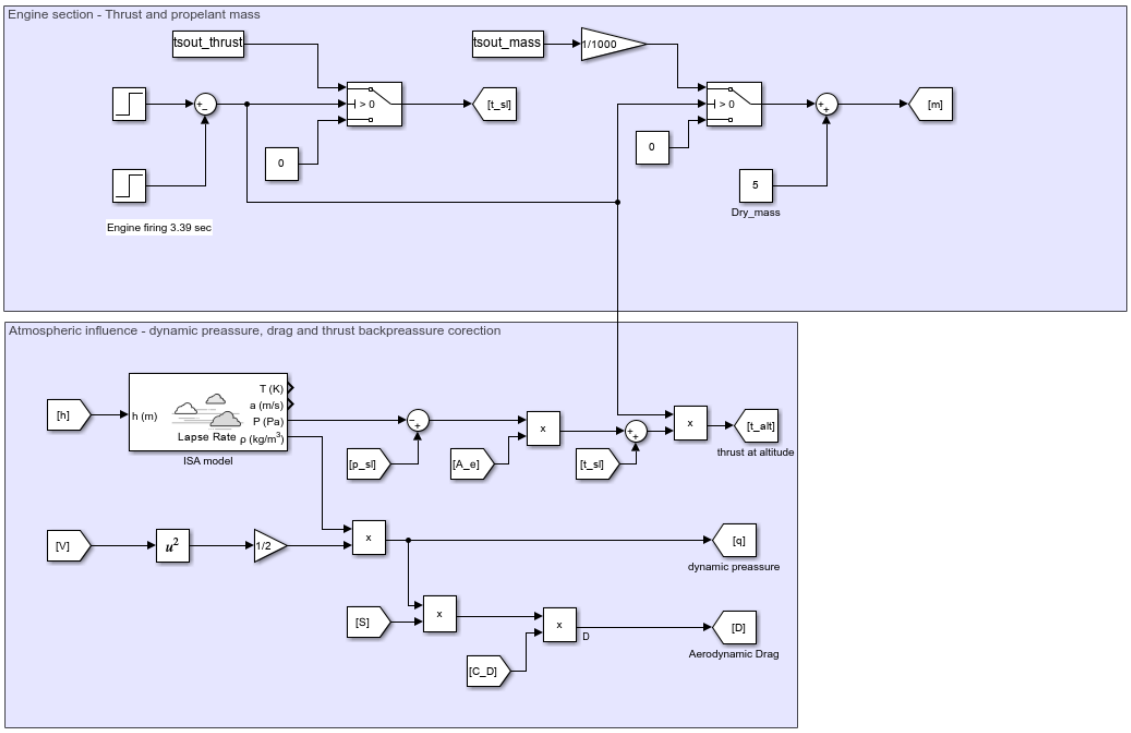

Figure 4.1. Implementation of the engine and atmospheric parts of the SIMULINK flight simulation of the Vanguard rocket

mass at the point of the burn out at time $t = 3.39\ s$. The thrust curve can be seen in figure 4.2.

Let us now handle the atmospheric influence on the flight. For this we use the International Standard Atmospheric model (ISA) utilizing its preprogrammed function block in the SIMULINK's aerospace block library. As can be seen in figure 4.1, we feed the function block with the rockets current altitude and in return get the atmospheric density and pressure at such altitude. We used this data to determine the following: i) We use the atmospheric density to calculate the dynamic pressure acting on the rocket and after multiplying it with the rocket's frontal area and the drag coefficient we get the aerodynamic drag force affecting the rocket in Newtons. ii) We utilize the atmospheric pressure to calculate the thrust backpressure correction. Which is a term defining the increase in thrust of the rocket engine with the increase in altitude and thus the decrease in the surrounding pressure. Decreasing the resistance the atmosphere gives to the exhaust fumes and thus increasing their exit speed, increasing the overall thrust of the engine with altitude. This increase in power is dictated by the term:

$$T_{alt} = T_{sl} + (P_{sl} - P_{alt}) \cdot A_e,$$

where the thrust at altitude is dependant of thrust at sea level, the difference in air-preassure and the area of the nozzle [7], [12]. This increase in thrust can be also seen in figure 4.2. Although the increase in thrust at the highest altitude where the engine still burns corresponds only to 1.5 % increase over the thrust at sea level, the increase might be more significant at higher altitudes.

As previously established the movement of the rocket is described by the state equations (10) dictating the dynamics of the system. The position of the rocket is then given, as follows in term (1). The downrange position of the rocket, which is more

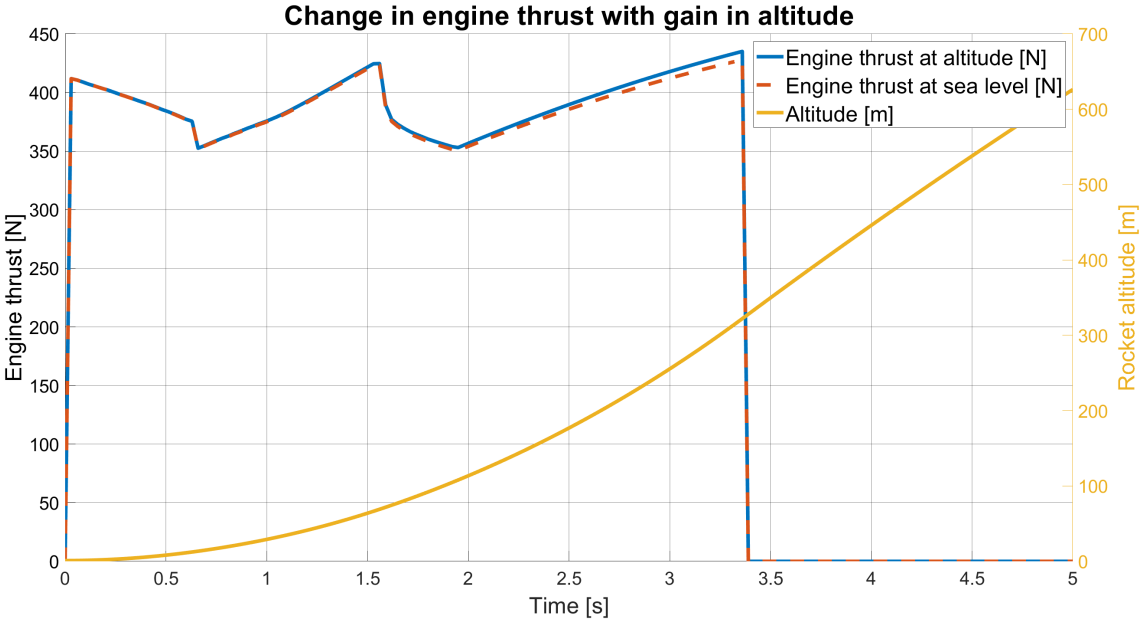

Figure 4.2. Thrust of the Kerberos solid engine at sea level and thrust gain with altitude

informative to us, can be calculated using Pythagorean theorem from the first two axis.

$$[\frac{ds_{RL}}{dt}]^L = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} V \cdot cos\gamma \cdot cos\chi \\ V \cdot cos\gamma \cdot sin\chi \\ V \cdot sin\gamma \end{bmatrix} \quad (1)$$

When we fed all the input data into the simulation we then got the following result, which can be seen in figure 4.3. As can be seen from the figure, the rocket reaches a peak altitude of around 1850 meters and travelers about 150 meters downrange during a roughly 40 second flight. The highest velocity reached by the rocket is equal to 198.43 m/s. The rocket was shot vertically with the path angle being set to 89.5 degrees at launch.

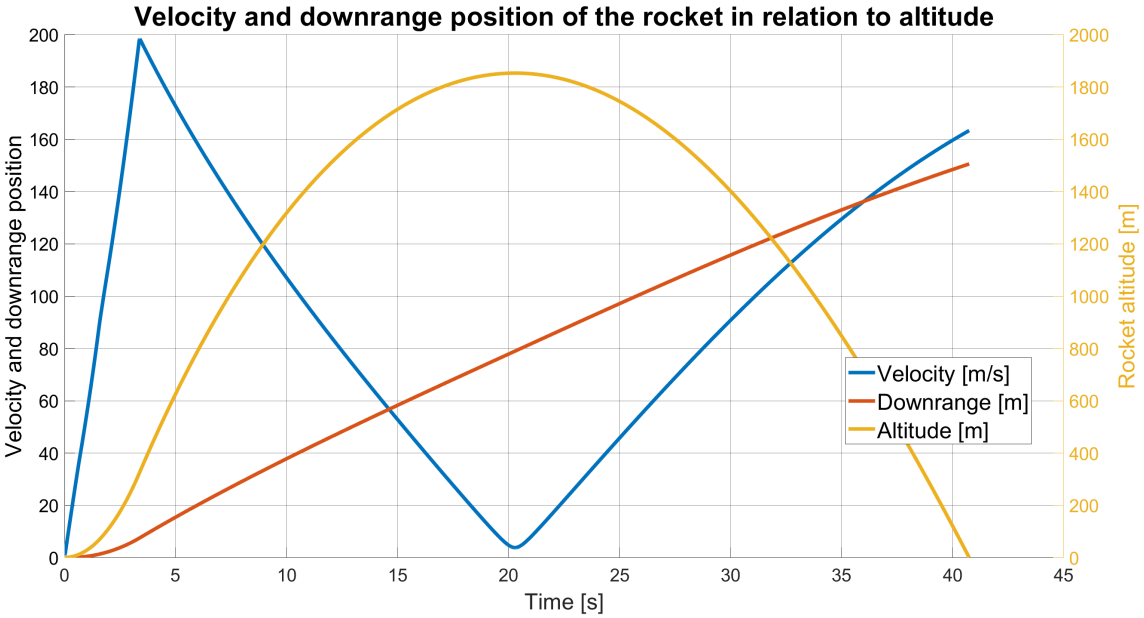

Figure 4.3. Results of the Vanguard translational flight simulation in SIMULINK

4.2 Modeling rotational flight dynamics of the rocket

Having constructed a transnational 3 DoF simulation of the rocket's flight, we will now construct a simulation concerning its rotation. We will construct the simulation in SIMULINK as a detached part and an add on to the translational simulation. Plucking data we need for the calculation, for example velocity and dynamic pressure from the translational simulation.

We will first consider the equation (14), from chapter 2. We will need make certain modifications to account for features of the Vanguard rocket. The equation counts on the roll control surfaces being the part of the main wing or our case stabilizer. Hence, it non-dimensionalizes the equation by the reference length and total area of the main stabilizers. As it is in our case, with the Vanguard rocket, the main stabilizers are static and only contribute to the dampening, the input of the system being simply the small control surfaces. Therefore, we will divide the non-dimensionalization part according to the belonging aerodynamic surface.

$$\dot{p} = \frac{\bar{q}}{I_1}\left(\frac{l^2 S}{2V} C_{lp} p + l_a S_a C_{l\delta a} \delta a\right) \tag{2}$$

We further need to establish the moment of inertia tensor as the rotational dynamics is greatly dependant on it. Thanks to the rocket being symmetric along its first body axis the tensor has non-zero elements only on its main diagonal.

$$\mathbf{I} = \begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{bmatrix}$$

As calculating the precise value of the inertia tensor is often strenuous we will only approximate its value by the inertia tensor for a homogeneous cylinder, basing our calculations on the mass and dimensions of the rocket, particularly its diameter r and height h.

$$\mathbf{I} = \begin{bmatrix} \frac{1}{2}mr^2 & 0 & 0 \\ 0 & \frac{1}{12}m(h^2+3r^2) & 0 \\ 0 & 0 & \frac{1}{12}m(h^2+3r^2) \end{bmatrix}$$

We are then only interested in the first element of the tensor, as it describes the inertia about the roll axis. The dimensions of the rocket remain constant during flight and we account for the change in mass by plugging in data from the translational simulation.

Now, the only two unknowns remaining in equation (2) are the aerodynamic derivatives C_{lp} and $C_{l\delta a}$. The roll dampening coefficient C_{lp} can be calculated from the dimensions of the main stabilizers, [13, pg. 954].

$$C_{lp} = \frac{(C_{l_\alpha} + C_{d0})C_r}{24S}(1+3\lambda) \tag{3}$$

The value of the lift curve slope C_{l_α} denotes how the lift coefficient changes per radian of change in the angle of attack. As we approximate the angle of attack itself, being zero, we set this coefficient to be also zero. The resistance coefficient C_{d0} we set to low value of $C_{d0} = 0.01$ per radian, [13, pg. 957]. The rest of the elements are given by the dimensions of the stabilizers. C_r denotes the root chord of the wing, in picture 2.4 this value can be found under the designation l_r. S denotes the total area of the stabilizers and λ is the ratio between root chord and tip chord. In case of the vanguard rocket those values are equal to $C_r = 0.3\ m$, $S = 0.6178\ m^2$ and $\lambda = \frac{1}{6}$.

Regarding the last of the aerodynamic derivatives, the aileron efficiency coefficient $C_{l\delta a}$, we know that the value of this coefficient should be lower than a value of 2π per radian, a value given by the thin airfoil theory for thin profiles [13, pg. 247], and should not be smaller than π, which is given by the Biot-Savart law. Although this law deals with electromagnetic fields by definition it has also corresponds with the world of aerodynamics as the speed of air in vortex has similar dynamics to that of a magnetic flux density described by the Biot-Savart law, [14, pg. 73, 122]. Although, we know of the fact, that this aerodynamic coefficient should be theoretically larger than π per radian, we will be running this simulation for multiple values of $C_{l\delta a}$ from 0 to 2π par radian, as well as for multiple values of the deflection angle of the control area δa from 0 to 10 degrees deflection.

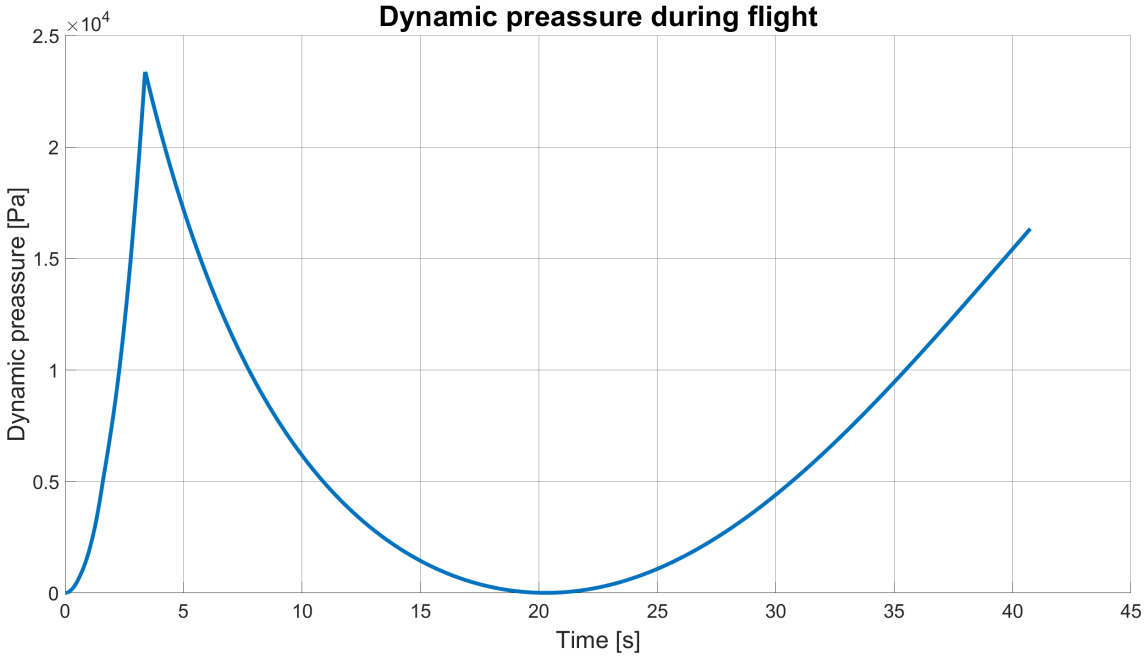

Figure 4.4. Dynamic pressure acting on the rocket during the simulated flight

After constructing the simulation add-on we can connect it to the translational simulation to get the data we need. Along with the speed of the rocket we also need the data about the dynamic pressure acting on the rocket during the flight. As can be seen from figure 4.4 the dynamic pressure peaks at the time of the burnout of the rocket engine at $\overline{q}_{max} = 23\ 372\ Pa$. The rotational flight simulation was run for five different values of the aileron efficiency coefficient and five different values of the deflection angle. The deflection angle is set for the whole duration of the flight. The results of the simulations can be seen in figures 4.5 thru 4.8.

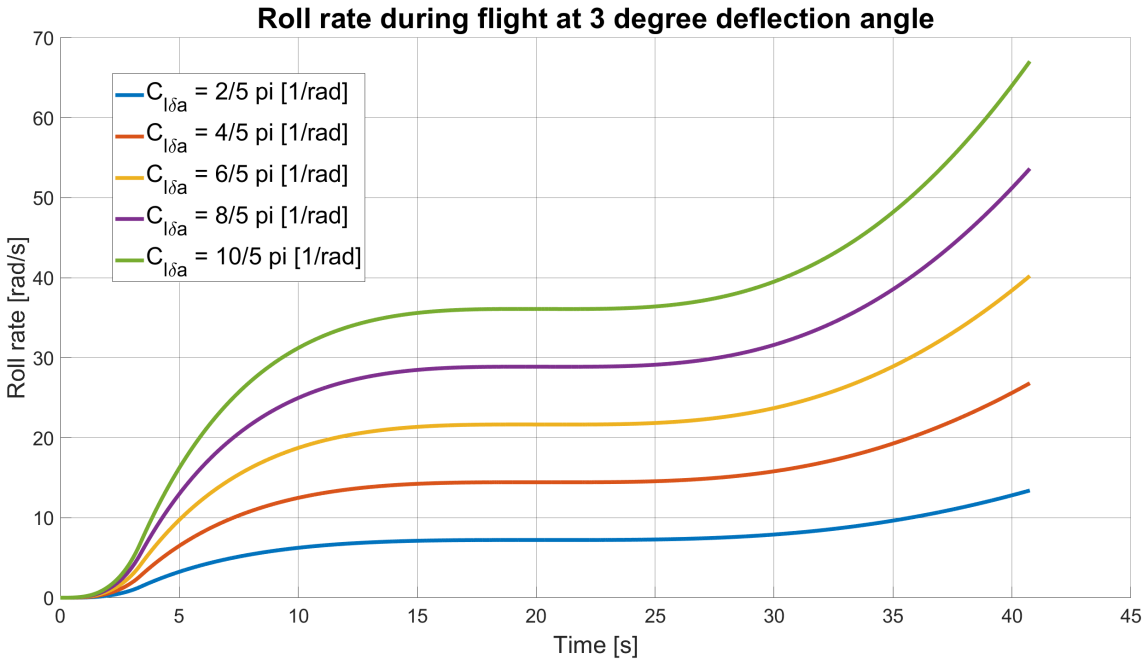

Figure 4.5. Results of the Vanguard rotational flight simulation in SIMULINK with 3 degree control surface deflection

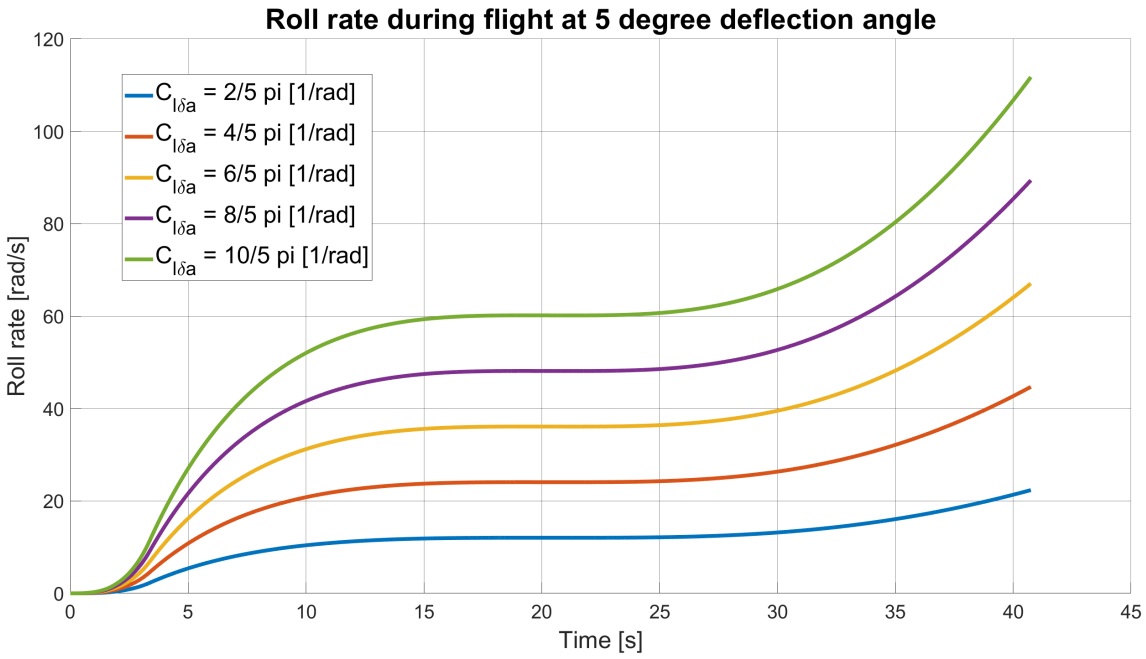

Figure 4.6. Results of the Vanguard rotational flight simulation in SIMULINK with 5 degree control surface deflection

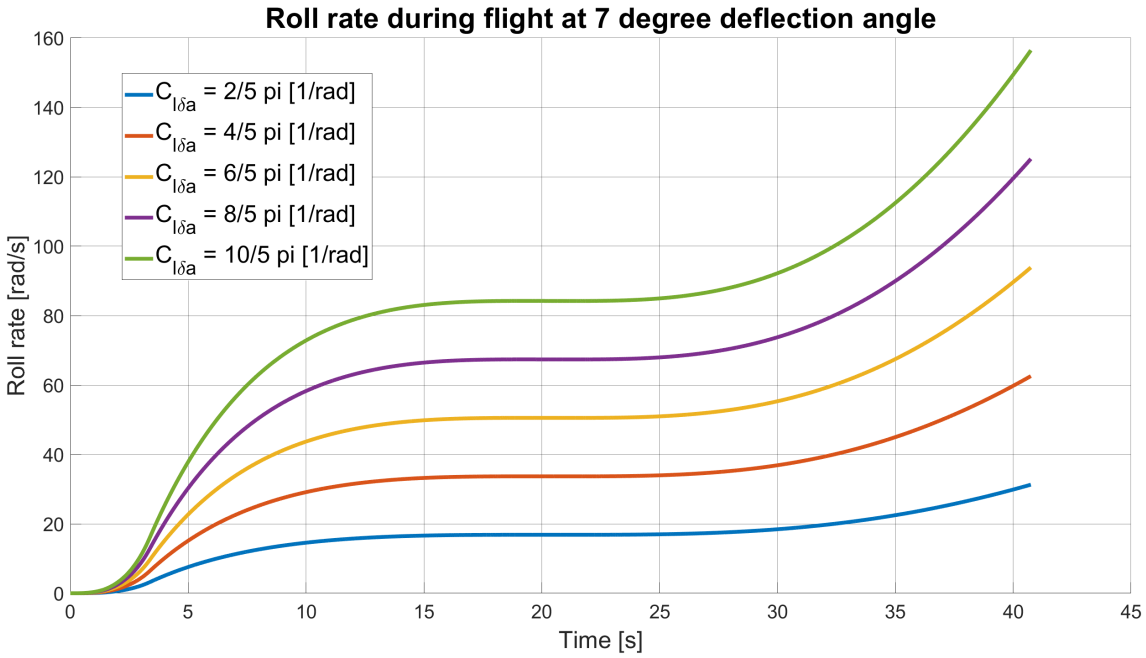

Figure 4.7. Results of the Vanguard rotational flight simulation in SIMULINK with 7 degree control surface deflection

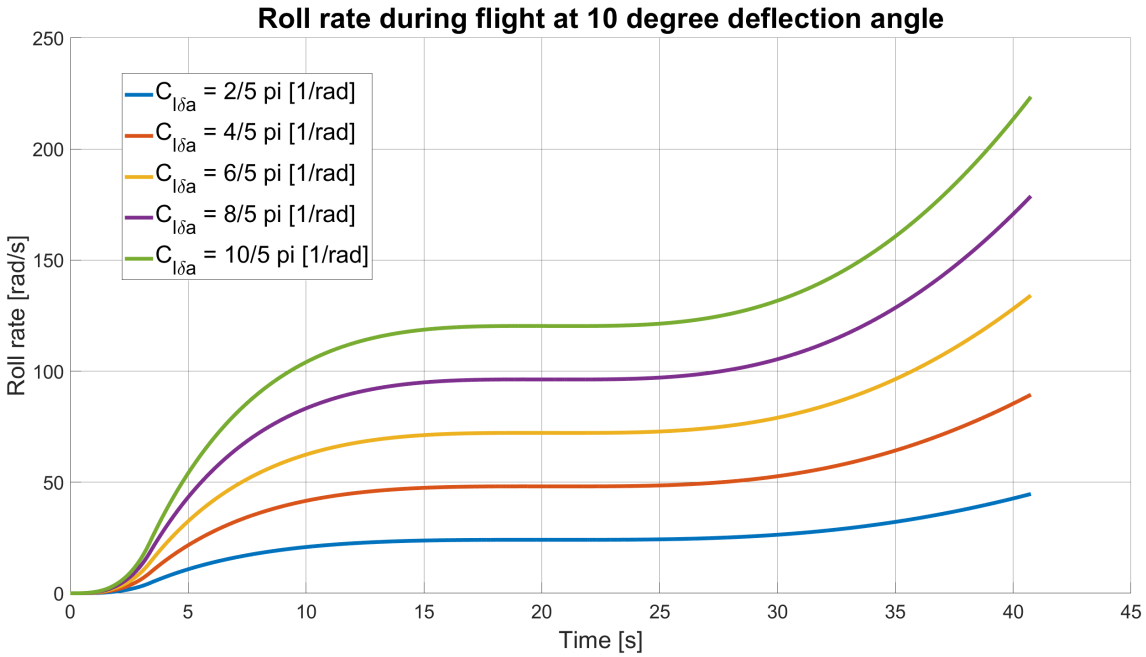

Figure 4.8. Results of the Vanguard rotational flight simulation in SIMULINK with 10 degree control surface deflection

As can be observed, the magnitude of the rotation changes linearly with the change of either the aileron efficiency coefficient or the deflection angle. Even though the equation of the rotational dynamics (2) points at a first order dynamic system, the behavior shortly after launch does not resemble it. The same is true after the rocket reaches its apogee roughly at the 20 second mark and stats falling increasing the rotation. As can be also seen from the simulations the rockets spin starts to increase sharply when the rocket starts falling. This might have several reasons. Firstly, the dynamic pressure increases steadily over the fall in contrast to the climb phase of the flight where it was

largest at the start with the motor burning and then lowered. Secondly, this might hint at the large amount of time required to reach the maximum roll rate when at high speeds and thus high values of dynamic pressure. Meaning, the rocket has by itself low roll dampening properties. A further study into the roll dynamics is advisable with the rockets velocity and dynamic pressure fixed over time.

4.3 Roll dynamics simulation in a fixed environment

As we have pointed out in the previous section, the data obtained from the flight simulation is insufficient for system identification. This is because the velocity dynamics which further influencing the dynamic pressure then both influence the dynamics of the roll rate during flight. This can be observer particularly at the start of the simulation in figures 4.5 thru 4.8. That is why we will now construct a simulation with fixed values of velocity and dynamic pressure. We start by implementing the roll differential equation (2) in MATLAB with the same constants which we used in the flight simulation. Then we choose multiple fixed points from the simulation depicting various parts of the climb stage of the flight and have exported the data. The chosen points can be seen in table 4.1.

Variable \ Simulation	1	2	3	4	5	6
Mass [kg]	5.669	5	5	5	5	5
Velocity [m/sec]	69.94	198.4	160	99.7	70	19.47
Dynamic pressure [Pa]	2983	23372	14517	5319	2561	194

Table 4.1. Chosen fixed points for the various simulation runs of the roll dynamics with a fixed environment

We then run the simulations at the chosen points in MATLAB using the ode45 nonstiff differential equation solver and have plotted the data as can be seen in figure 4.9.

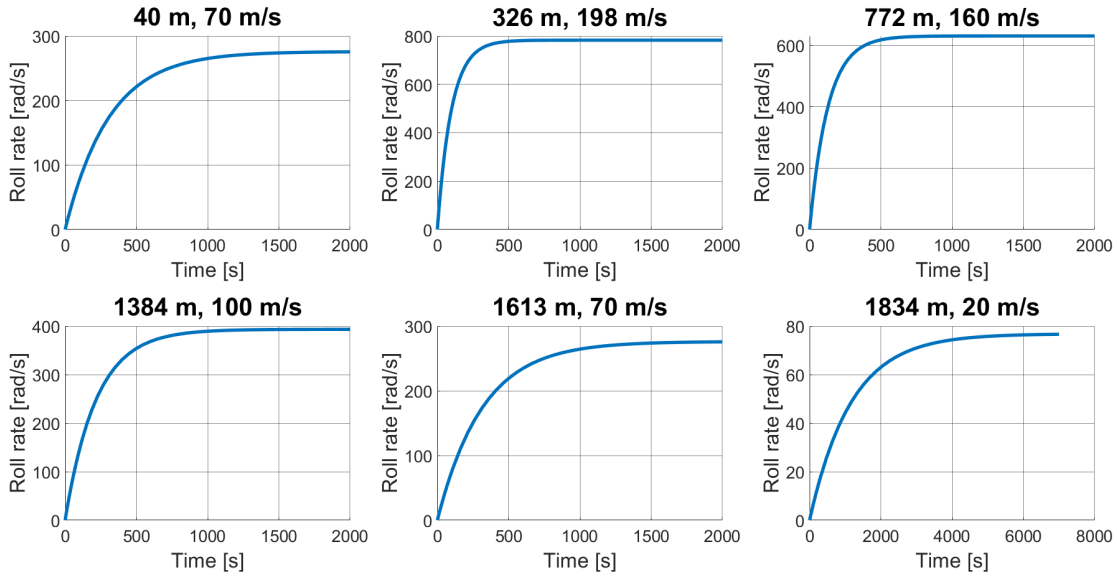

Figure 4.9. Simulation runs of the roll rate dynamic of the rocket in fixed points. The figgures correspond by number from left to right with the table 4.1

The simulations were done for the control surface efficientcy coefitient being equal to $C_{l\delta a} = 3/2\pi$ $[1/rad]$ and the deflection angle of the surfaces being set to 5 degrees deflection. We have also calculated with the change in weight of the rocket affecting a change in the moment of inertia tensor. To further justify the choice of the fixed point we have chosen mission critical points during the rocket flight. The first point occurring shortly after the takeoff of the rocket, the second being the point of engine cutoff which coincides with the point of peak dynamic pressure, the rocket experiences, and point six, which is close to the apogee. Points three thru five were chosen to fill out the coastal stage of the flight.

As can be seen from the simulations the rocket's roll rate is clearly a first order system. It can be also seen that the rocket has clearly low roll dampening properties so the dynamics take a long time to arrive at the steady state value. This said, the dynamics, which gain and time constant properties change depending on the altitude and velocity of the rocket, clearly change fast enough that the rocket can never reach the steady state value. From our translational simulation we know that the time of flight to apogee is equal to approximately 20 seconds. Having done these observations, we proceeded with identification of the six simulated first order systems. As can be seen in the following term (4), a first order system is given by its transfer function:

$$H(s) = \frac{A}{\tau \cdot s + 1} \tag{4}$$

Where A is the gain of the system end is equal to the steady state value divided by the input of the system in infinity.

$$A = \frac{y(\infty)}{u(\infty)}$$

τ is then the time constant of the system and is equal to time at which the system reaches the value of $0.63y(\infty)$. In other words τ is the tame the system takes to reach the 63% value of the value in its steady state. After identification of the system we have been left with values seen in table 4.2.

Variable \ Simulation	1	2	3	4	5	6
$A \cdot 10^3$ [-]	3.15	8.96	7.23	4.5	3.157	0.87
$\tau \cdot 10^3$ [sec]	0.305	0.097	0.127	0.216	0.314	1.15

Table 4.2. Gains and time constants of the six identified first order systems resulting from the six carried out simulation runs.

When we average these values we end up with a following first order system, as seen in transfer function (5) and step response 4.10, which is the system we will be focusing on and will design a controller for.

$$H(s) = \frac{811.5}{368.2 \cdot s + 1} \tag{5}$$

The author would also like to state that although a choice of six points may be sufficient for this kind of system identification procedure a larger amount of points would turn beneficial in coming up with an average system overall closer to the flight simulation. The best possible course of action would be to run a roll simulation with fixed values of velocity and dynamic pressure, as we have done, for every data-entry in the translational simulation, identifying the systems and then averaging all the results. Although this

kind of simulation would with certainty prove more precise, we have found the preparation of such simulation too time-consuming and in the end not worth the effort. As this kind of system identification and simulation already heavily relies on approximation and so such kind of improvement is, in the authors opinion, insignificant.

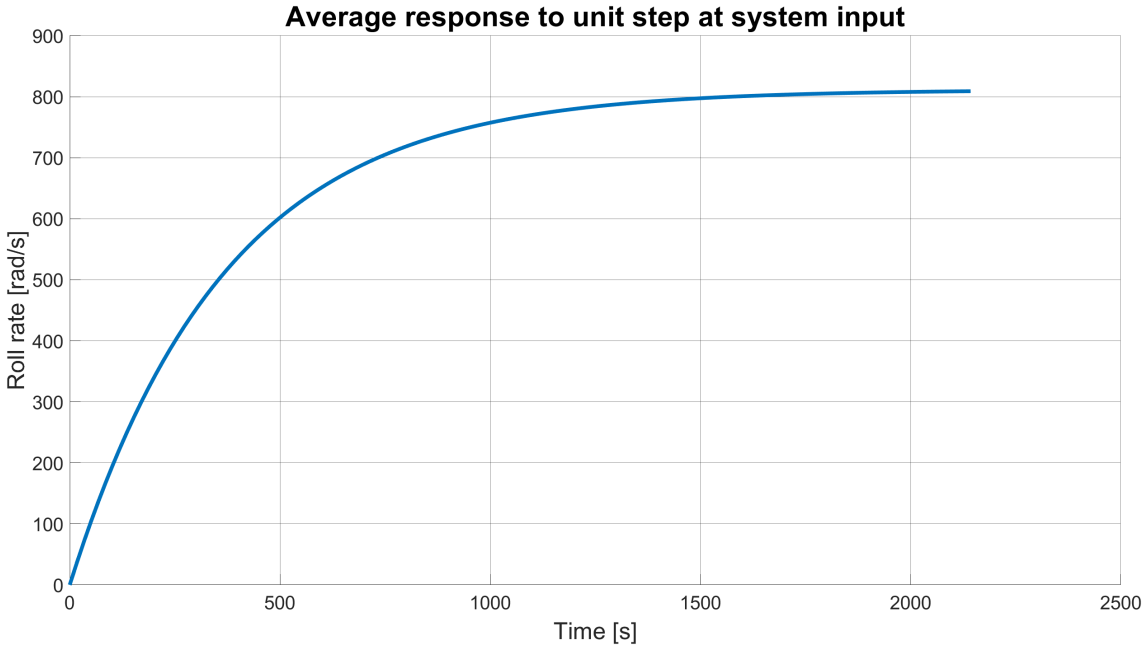

Figure 4.10. Step response of the identified first order system governing roll characteristics of the rocket with a unit step at the input equal to 5 degree deflection angle

Chapter 5
Using Kerbal Space Program as a flight test environment

Kerbal Space Program (KSP) is a space flight simulation game developed by studio Sqoad and published by Private Division. The game was first released in 2011 and focuses on designing and building aerospace vehicles such as rockets and spacecraft and launching them to explore the Kerbolar system, a star system resembling that of our own Solar system but on a smaller scale. The game is set around a highly developed physics engine simulating the behaviour of aerospace vehicles in atmospheres around planets and in space, simulating orbits of all planetary bodies and of vehicles orbiting them or traversing between them. The game has greatly evolved from its inception and first release. Prior the player could first only reach the orbit around Kerbin, the home planet, later a moon called Mun was added allowing the player to simulate his own moon mission. Since then the game added numerous planetary bodies and improvements, ending at a stage where one can truly simulate complex mission to Mars or further into the star system. The game allows the user to grasp the understanding of physics and orbital mechanics in a fun and engaging way. Indeed, since its inception the game has gained praise by institution as NASA and ESA.

In this chapter we will exploit the developed physics simulation behind KSP and will try to use it as a testing environment in a form of a software in the loop (SITL) simulation for testing of control laws to control the roll rate of the rocket during flight. And although the simulation is not strictly tailored to our needs. Such as the environment is different to ours in terms of atmosphere and gravity, the laws governing the basic dynamics and flight of the rocket should hold and as such provide us with good plug and play type environment for us to determine, which types of control might work the best for our needs. As part of this chapter we will first establish the background that will allow us to gather useful data from the simulation environment and also enable us give inputs into the simulation, allowing us to communicate both ways. Then we will design a simple rocket resembling that of our own rocket, Vanguard, and will carry out multiple simulation ending with identification of the system. Finally we will design and test multiple kinds of controllers in the environment to determine which kind seems to be the most promising for our real world application.

5.1 Setting up KSP as a testing environment

To use KSP as a simulation environment we must first get it into a form where we are able to extract raw data from the game and also be able to input control orders for the testing purposes. We will use the kRPC mod and server plugin [15] to transform KSP into a functioning SITL simulator.

The first step of the transformation is the installation the kRPC mod for the game from the source [16], then, as we will use the Python environment for the implementation of the interface, we have also installed the kRPC extension library using pip package manager.

```
pip install krpc
```

With these two steps done we can then proceed to establish the communication bridge between KSP and our Python program. To accomplish this we simply follow the instructions set in the documentation [15]. The communication then looks as follows.

```
conn = krpc.connect(
    name = 'KSP_matlab_com',
    address = '127.0.0.1',
    rpc_port = 5000,
    stream_port= 7000
)
```

After successfully establishing interface between KSP and Python, we have to decide if we want to set up a direct link between KSP and SIMULINK, using the Python program only as a communication interface between the two, or keep the direct communication between KSP and the Python program only. In the first case, we can establish further UDP communication between Python and SIMULINK, as can be seen in figure 5.1. This provides us with a direct connection between KSP and SIMULINK, where we can gather data, carry out data analysis and directly implement control laws to control the vehicle.

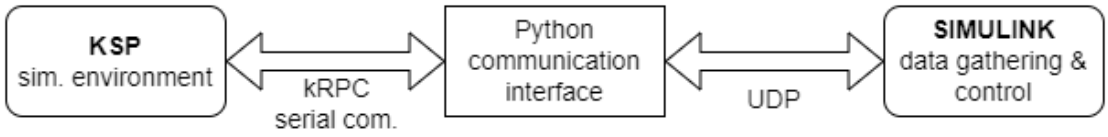

Figure 5.1. Communication diagram between KSP and MATLAB / SIMULINK a direct connection architecture

Another way to establish the architecture is to create an indirect communication between KSP and MATLAB and SIMULINK, as can be seen in figure 5.2. This is a preferable way of execution as the first option led to problems with frequent software upgrades. With the indirect architecture we use the Python program not simply as a communication interface but use it directly for data gathering. In such a case export this data in form of a timeseries object to be loaded in MATLAB or SIMULINK. There we can carry out data analysis and design control laws, which we can then implement in the Python program. In this way the program serves the same as would a program on a flight computer onboard our real life rocket. This similarity in implementation to a real life system is also a good reason for using the indirect architecture, which can be seen in figure 5.2.

Let us now outline the form of the Python program, counting on the use of the indirect connection architecture. To gather flight data from the simulation, we must first chose a reference frame to measure the data in. Multiple reference frames are provided in the kRPC library, as we are simply flying strait up and want to make measurements before we approach the apogee we ended up choosing the surface reference frame. In

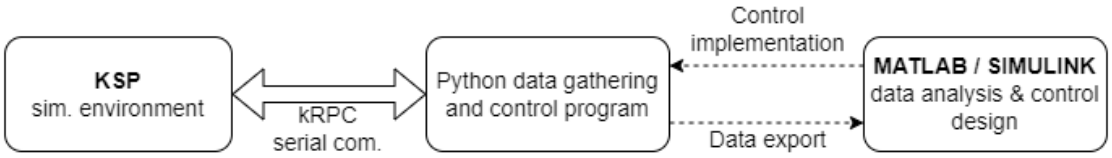

Figure 5.2. Communication diagram between KSP and MATLAB / SIMULINK an indirect connection architecture

this reference frame the x axis points vertically upwards, the y axis points north and the z axis points east.

```
vessel = conn.space_center.active_vessel
ref_frame = vessel.surface_reference_frame
```

We can also make the reference frame axies visible during the simulated flight using the following code.

```
def draw_reference_frame(reference):
    conn.drawing.add_direction_from_com((1, 0, 0), ref_frame).color =
(255, 0, 0) # x-axis is red
    conn.drawing.add_direction_from_com((0, 1, 0), ref_frame).color = (0,
255, 0) # y-axis is green
    conn.drawing.add_direction_from_com((0, 0, 1), ref_frame).color = (0,
0, 255) # z-axis is blue
```

With this set we can prepare a loop in the python code for gathering of the flight data. But first we need to define the flight variable which gives us access to certain types of data such as altitude and dynamic pressure.

```
flight = vessel.flight()
# --- flight loop ---
while(vessel.situation == vessel.situation.pre_launch or vessel.situation
== vessel.situation.flying) and (ascent):
# --- KSP draw ---
draw_reference_frame(ref_frame)

# --- KSP input ---

# --- KSP measure data ---

# --- loop sleep ---
time.sleep(0.01)
```

In this way we can input commands into the environment, gather data and store them in corresponding fields. As a single run of such loop is not time consuming we put a sleep command at the end of the loop so that we send commands and gather data approximately every 0.01 second. The ascent boolean value in the run condition of the loop allows us to measure only the scenting part of the flight until the rocket reaches the apogee. After that the loop terminates and the program proceeds with saving the data as a timeseries .mat file. The ascent condition is checked within the loop by calling of the following function:

```
def ascending_to_apogeum(altitude_now, alt_old):
    alt_delta = altitude_now - alt_old
    if alt_delta <= -0.1:
        print("Apogeum reached")
        return False
    else:
        return True
```

After the loop terminates the program proceeds with saving and exporting the data. For these purposes all data is gathered into a matrix of appropriate size. With the columns equal to the number of types of data measured, for example altitude, velocity, dynamic pressure etc., and the number of rows equal to the number of measurements taken. A time scale is then created, equal in length to the number of measurements and with a fixed step of 0.01 seconds. Then the time scale and data matrix is used to create a timeseries object which is then saved as a .mat file. A scipy.io library (sio) is needed to perform this operation.

```
time_series_data = {'time': time_values, 'data': data_values}
sio.savemat('timeseries_file_name.mat', time_series_data)
```

5.2 KSP rocket SITL simulation

Having modified the basic KSP environment so that we are capable to gather flight data from the simulation environment and also able to input commands into it, we will construct a simple rocket resembling that of our own Vanguard platform. Because we are using a stock version of KSP, aside from the kRPC mod, we only have access to the stock parts. As such, we cannot come up with a precise enough model and can only observe a similarity in the flight dynamics. Because of this reason the simulation is not precise enough to directly simulate our rocket. Still, we should be able to determine which type of control should work the best. With that said we have build a following rocket which can be seen in figures 5.3 and 5.4. The rocket is 8.2 maters high and weights 2.784 tons.

Figure 5.3. A side view of the rocket constructed within the KSP environment

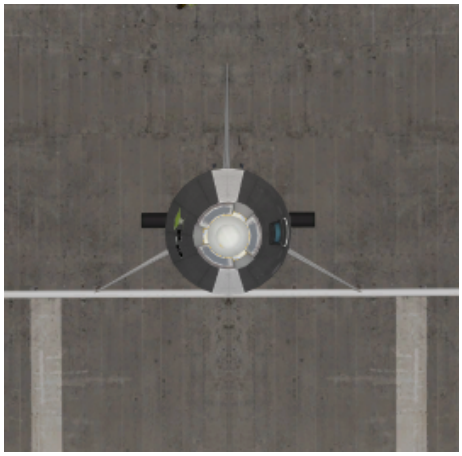

Figure 5.4. A top view of the rocket constructed within the KSP environment

The rocket consists of, from bottom up of a solid rocket motor, 3 vertical stabilizers, 2 movable control surfaces, a spacer compartment and a crew capsule on top. As the KSP simulation does not calculate any imperfections in the building process there will be no rotation generated from the misalignment of the main stabilisers. Instead we have created two rockets. One with perfectly straight stabilizers for purposes of system identification and a second one for testing of controllers further down the line.

For the first simulation we have launched the rocket without any input to the control surfaces and just measured the performance of the rocket. As the rocket at first flew at supersonic speeds we have, over multiple simulations, tweaked the performance of the engine until we arrived at subsonic speeds close to the performance of the Vanguard rocket. The result of such simulation in the KSP environment can be seen in figures 5.5 and 5.6.

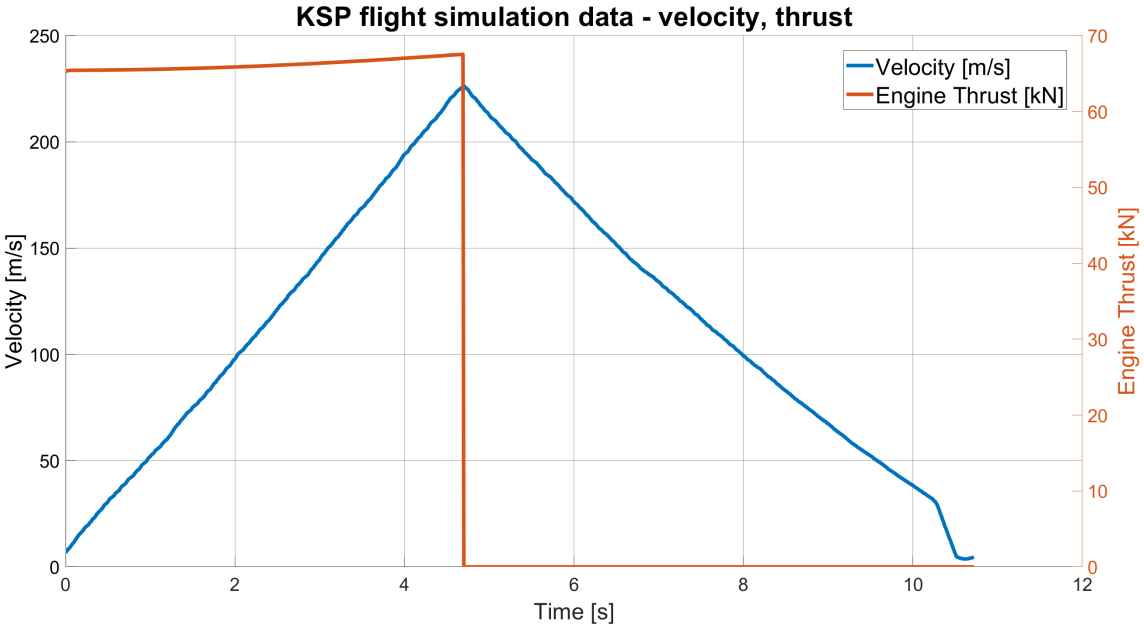

Figure 5.5. KSP simulation output of rocket's engine performance, the thrust of the rocket plotted over the ascending phase of the flight with speed plotted for reference.

As can be seen, the engine is much more powerful than the engine of the vanguard rocket, this corresponds to the precision problem of constructing rocket in KSP, where we can only get close to the concept of our real world rocket. Although this problem might be solvable by further modifying the game with further mod packages, including more rocket parts or designing our own parts and importing them into the game, this is out of the scope of this work. We can also take a look at figure 5.7, which displays the atmospheric density around the rocket throughout the flight. The value of the atmospheric density is calculated according to the altitude the vessel finds itself at, using the term:

$$\rho = 1.225 \cdot e^{-h/7000} \ [kg/m^3]. \tag{1}$$

Meaning that the air density at sea level in the simulation is the same as on Earth, using the standard atmospheric model, but drops more rapidly. The atmospheric density of Kerbin at 3700 meters is according to the term (1) equal to $\rho = 0.72\ kg/m^3$ compared to the atmospheric density of the Earths standard atmosphere which has roughly density of $\rho = 0.83\ [kg/m^3]$. This along with the smaller gravity of Kerbin, it being roughly one tenth size of the Earth, contributes to further deviation of the simulation environment

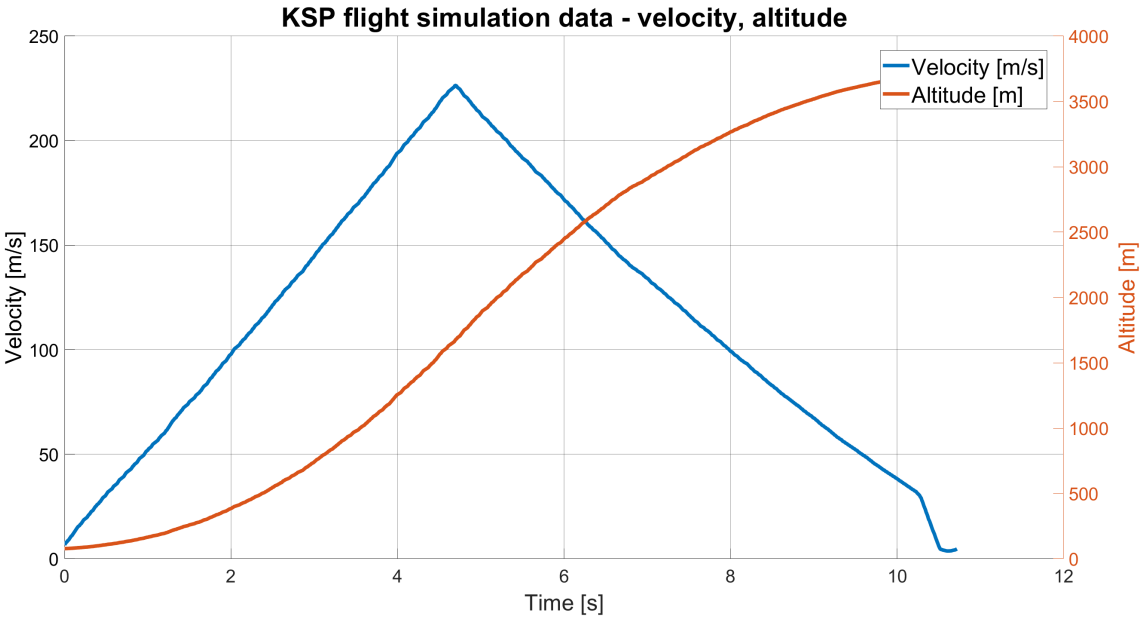

Figure 5.6. KSP simulation output of rocket's flight performance, the velocity given is calculated from partial velocities distributed along the three axes of the reference frame.

from the real world. This indiscrepancy could be mitigated by implementing the Real Solar System mod for KSP. Again, this would be over the scope of this work as it is our task to study the dynamics and control principals, which should hold for both the KSP environment as the real world. As such we are left with a non-calibrated high fidelity model.

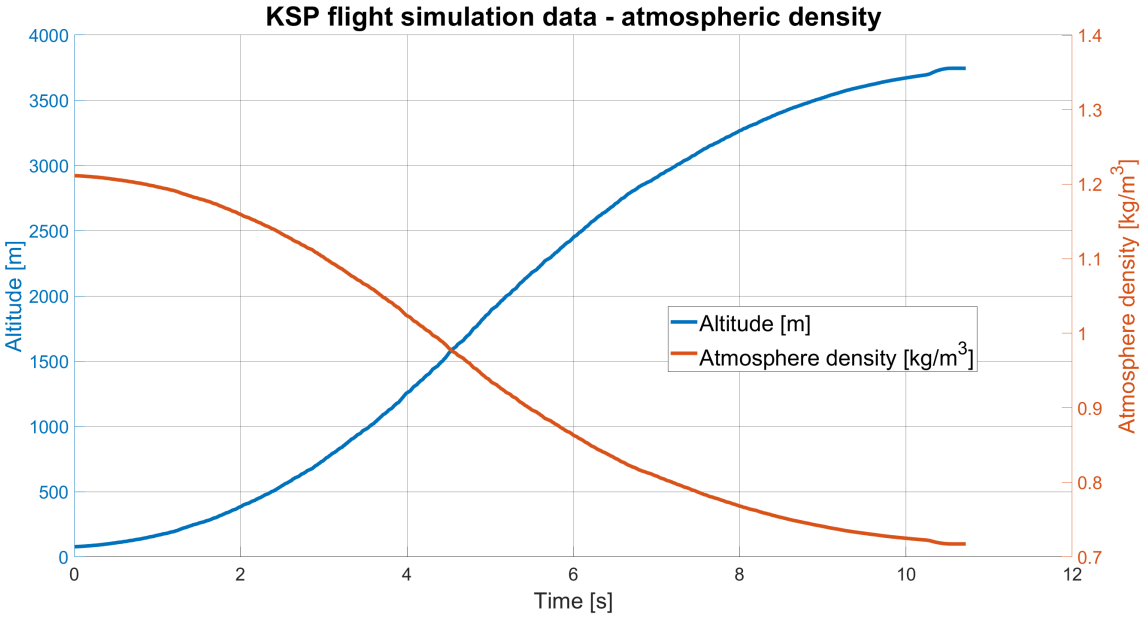

Figure 5.7. KSP simulation's environmental characteristics, the atmospheric density depending on altitude plotted over the duration of flight.

After familiarizing our selves with the simulation environment we have run a test flight in attempt to carry out system identification, which can be seen in figure 5.8. Here we have measured a similar dynamic to the ones from section 4.2, where we carried out

our own roll simulations. In contrast with those simulations we can see that here the maximum roll rate is much smaller and the rolling motion starts to slow down rapidly after the engine burns out. This points to two things. One, the rocket is much larger and heavier thus making the total value of its tensor of inertia larger, making it harder to spin. And second, the rocket has better dampening properties than the Vanguard rocket, as the roll rate quickly subsides. This is logical, given the large stabilizers the rocket is equipped with. Note that the measured roll rate in this graph is flipped for better orientation in the graph and the identification process. When we deflect the control surface by a positive angle, the rocket starts to spin counterclockwise, which is a negative direction for most sensors, such as gyroscopes (in the previous chapters we have denoted this as the positive direction to coincide with the right hand law for right-handed systems). This quality will be further reflected in the feedback loop architecture at a later section, 5.3. Regardless the design process of the controller is not affected by this and stays the same.

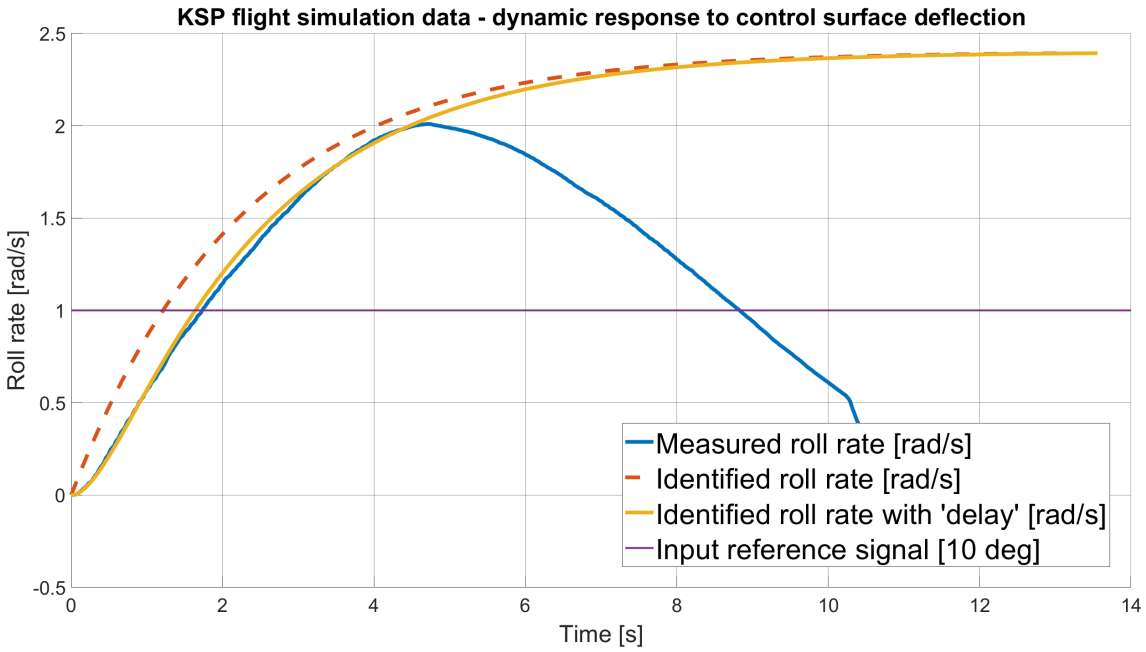

Figure 5.8. KSP simulation output of the roll rate of the rocket during flight with the deploy angle of the control surfaces being set to 10 degrees deflection.

From the measured roll rate of the rocket we can deduce that this is once again, similarly to section 4.2 a first order system with an onset delay. We have aproximated the steady state value of the dynamic response and calculate the time constant of the system, (2), this results in the dotted line seen in the figure 5.8. We can then introduce a time delay characteristic to better fit the system, (3), and confirm the correctness of our system identification.

$$H(s) = \frac{2.4}{2.25 \cdot s + 1} \tag{2}$$

$$G(s) = \frac{1}{0.4 \cdot s + 1} \cdot H(s) \tag{3}$$

5.3 Controller design and control loop logic testing in KSP

Having identified the system and understanding the relation between control surface deflection angle and roll rate response of the system governed by said dynamic we can approach the design of controllers to test, which kind of control is best suited for our needs in the real world application. As we previously showed the system response inverted, let us once again show the identified system, this time in its raw form. This can be seen in figure 5.9.

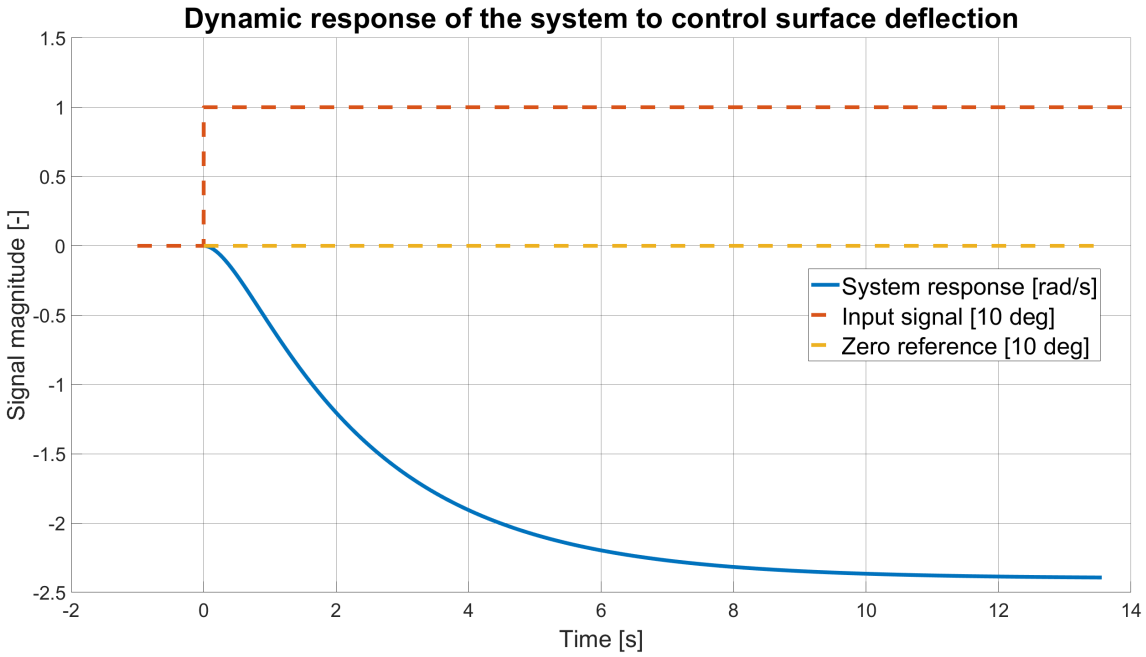

Figure 5.9. An identification of the system describing the relation between control surface deflection and rockets roll rate throughout its flight, the unit step is equal to 10 degrees deflection

As the transfer function of the system has in fact a negative gain we will need to introduce a multiplication by -1 on the output of the controller for the system to be stable. The proposed loop can be seen in figure 5.10

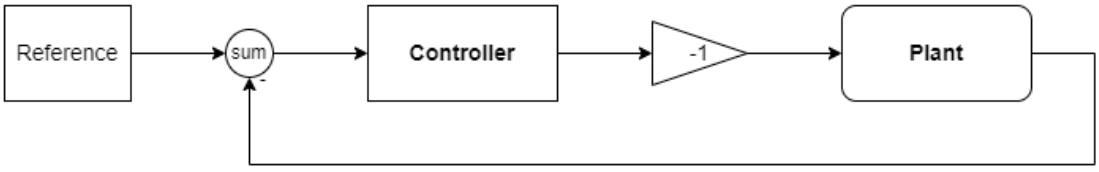

Figure 5.10. A proposed control loop with negative feedback loop and correction for the negative gain of the identified system by multiplying the controllers output by factor of -1

Before we proceed with designing controllers for control of the roll rate of the rocket we need to establish a control function for implementation of such controllers in our Python program. We have build a control function which can represent any of the following controllers depending on its input. The supported controllers are P controllers,

PI controllers, PD controllers and PID controllers. The following is the code of the Python function, it is called in the "KSP input" part of the flight loop.

```
def roll_pid(measurment, reference, kp, ki, kd, Tf):
    global integral, e_prev, time_prev

    my_time = time.time()
    # --- negative feedback loop ---
    e = reference - measurment

    # --- proporcional part ---
    proporcional = kp*e

    # --- integral part ---
    integral = integral + ki*e*(my_time - time_prev)

    # --- derivative part ---
    if Tf != 0:
        derivative = (kd/Tf)*(e - e_prev)/(my_time - time_prev)
    else:
        derivative = kd*(e - e_prev)/(my_time - time_prev)

    e_prev = e
    time_prev = my_time

    action_input = proporcional + integral + derivative
    return -action_input
```

Where "measurment" is the measured roll rate of the rocket, "reference" is the intended roll rate we want the controller to hold and "kp", "ki", "kd" and "Tf" are the constants of a full PID controller given by expression (4). Note that this type of controller also allows for the implementation of the first order filter in the derivative term, if it is used. The first order filter in the derivative form is used to filter signal noise and reduces the strength of the derivative term, limiting the clutter on the output of the controller, [17, pg. 376]. We can switch between the various controllers by setting the unwonted portions such as the integral portion or derivative portions to zero using their respective constants to achieve this.

$$PID(s) = K_p + K_i \cdot \frac{1}{s} + K_d \cdot \frac{s}{T_f \cdot s + 1} \tag{4}$$

Understanding the relation between the deflection of the control surfaces and the change in the rockets roll rate and having prepared the infrastructure needed for controller implementation and testing we can proceed with designing the controllers themselves. As we only want to experiment and figure out which type of controller is best suited for our needs we will not utilize any analytic methods, instead leaning on the SISO tool in MATLAB utilizing the root locus method to achieve roughly 5 second settling time and overshoot not larger than 20%.

We will start with a simple P controller, where we are just scaling the difference between the reference and the output of the system in the negative feedback loop. We have then implemented the controller into the Python program and tested it in the

KSP simulation on the variant of the rocket on which we carried out the identification of the system. This rocket has a slight offset introduced on its main stabilizers putting the rocket into a spin. The resulting reaction of the controller trying to correct and the resulting roll rate of the rocket can be viewed in figure 5.11.

$$C(s) = -5.53$$

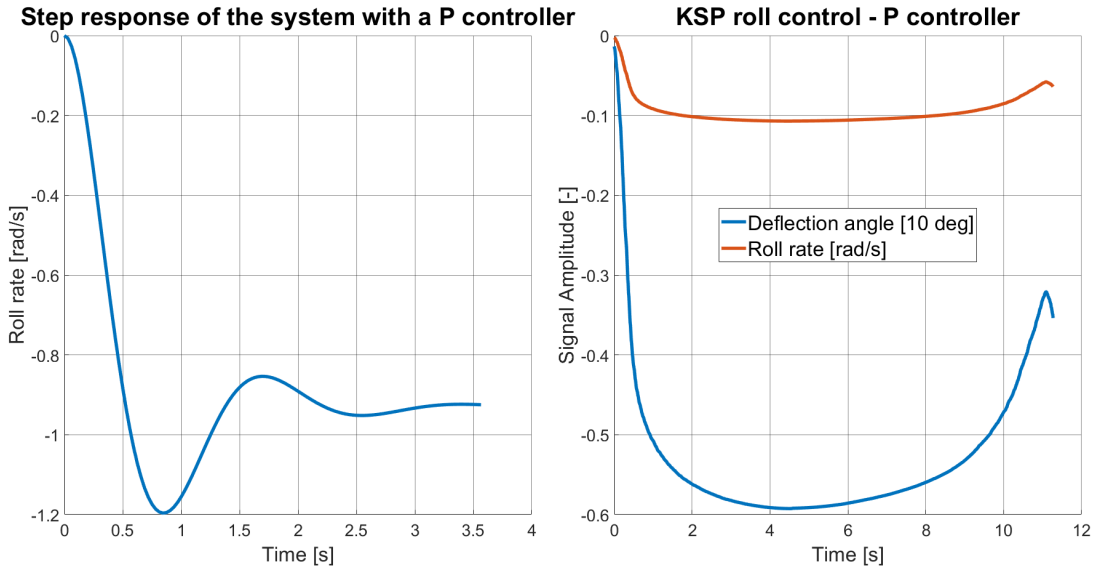

Figure 5.11. Step response of the system with the proposed P controller and a KSP simulation output of the controller correcting for the rotation created by stabilizer offset. The deflection angle is displayed with respect to the unit step of 10 degrees deflection.

We can observe that the controller does not achieve zero steady state error, as it lacks the properties of the integral term. In this sense it only stops the rocket from achieving any significant roll rate, but does not stop the rotation entirely. A similar behaviour can be expected from the PD controller, which can be seen in figure 5.12.

The PD controller is created with an introduction of real zero into the system and is in our case given by transfer function (5). Which results in the controller parameters of $K_p = 0.635$ and $K_d = 0.325$ in the controllers paralel form.

$$C(s) = -0.325 \cdot (s + 1.952) \tag{5}$$

We can observe that the system behaves, as we expected, in a way it does not reach zero steady state error, if anything it is underpowered. Furthermore, the controller introduces oscillations on input resulting in shaking of the control surfaces and the chaotic behaviour of the roll rate. This is a product of the D component of the controller reacting to the rapid changes in the dynamic of the system. With this we can state that the differential term of the controller introduces oscillations into the system and hence is not beneficial for our application. We will see if its chaotic behaviour will be somewhat mitigated with the introduction of the integral term in the PID controller. But lets first take a look at the influence of the integral part itself.

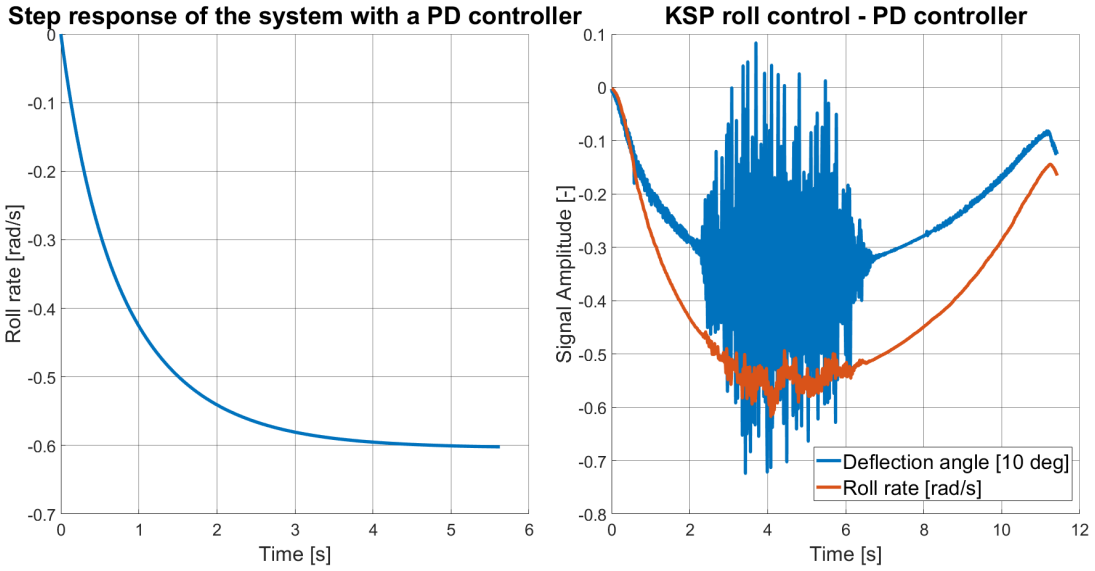

Figure 5.12. Step response of the system with the proposed PD controller and a KSP simulation output of the controller correcting for the rotation created by stabilizer offset. The deflection angle is displayed with respect to the unit step of 10 degrees deflection.

With the design of the PI controller we can expect a zero steady state error and smooth response to changes in contrast to the PD controller. We create the PI controller by introducing a real zero and an integrator into the system. This can be seen in term (6) and results in the controller parameters $K_p = 1.11$ and $K_i = 0.666$ in the controllers parallel form.

$$C(s) = 1.106 \cdot \frac{s + 0.602}{s} \tag{6}$$

The performance of the PI controller can be seen in figure 5.13.

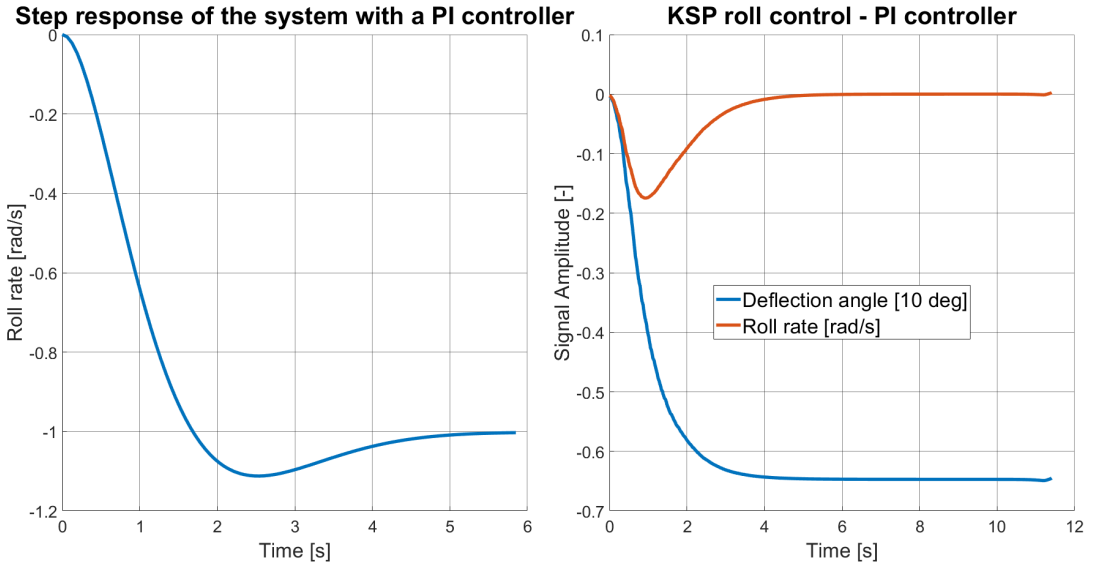

Figure 5.13. Step response of the system with the proposed PI controller and a KSP simulation output of the controller correcting for the rotation created by stabilizer offset. The deflection angle is displayed with respect to the unit step of 10 degrees deflection.

We can clearly observe that the controller has managed to fully suppress the rotation of the rocket. Furthermore this occurred without any oscillations we might expect with the PD controller. We will now investigate the behaviour of a PID controller to evaluate its performance and see the effect of the derivative term on the system.

The PID controller was one again designed using the root locus method with the introduction of a complex zero an integrator and a real pole. During the design we have made attention to the overshoot of the system and that it does not exceed the 20% mark. On the other hand, we have left the settling time lax. As a shorter settling time would have required larger overshoot and thus the controller would be more aggressive. A quality we do not strictly require and moreover might be harmful to the overall behaviour of the system. The design of the controller culminated in the form shown by term (7) and results in the controller parameters $K_p = 0.499$, $K_i = 0.6707$, $K_d = 0.338$ and $T_f = 0.778$ in the controllers parallel form. The parameters were as with the other controllers calculated from the transfer functions of the proposed controllers with the use of the MATLAB's pid() function. We can see that the T_f parameter is non zero and so this is a filtered version of the PID controller.

$$C(s) = 0.935 \cdot \frac{s^2 + 1.444 \cdot s + 0.9734}{s \cdot (s + 1.286)} \tag{7}$$

We can observe the behaviour of the system with the implemented controller in figure 5.14.

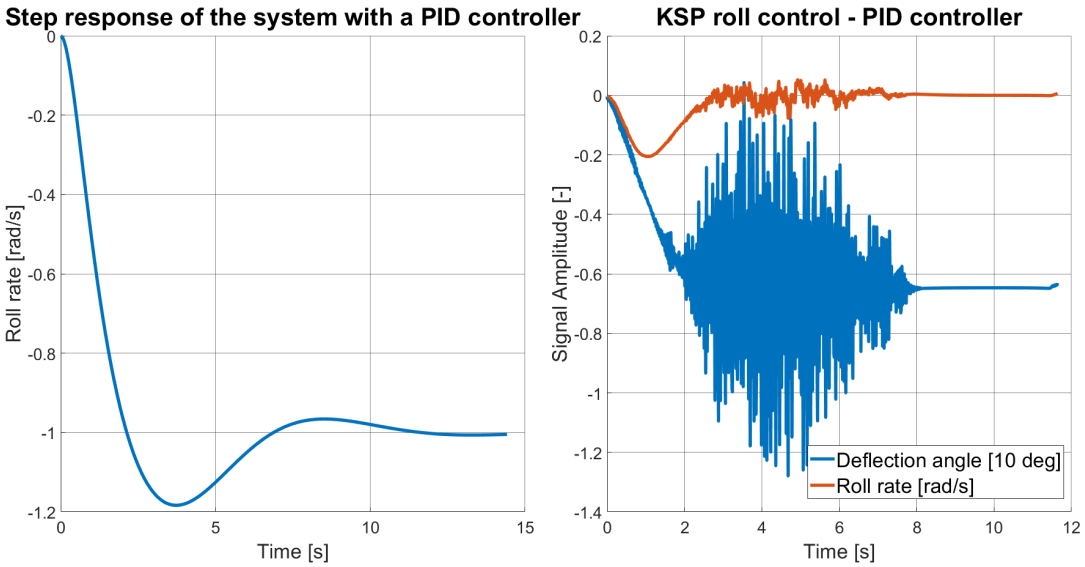

Figure 5.14. Step response of the system with the proposed PID controller and a KSP simulation output of the controller correcting for the rotation created by stabilizer offset. The deflection angle is displayed with respect to the unit step of 10 degrees deflection.

We can observe that the system behaves similarly to the system with the implemented PI controller, but does not come without the oscillations introduced because of the derivative term. We can see that some of the oscillations on system input are in fact larger than 10 degrees and so would be saturated by the system, as the maximum deflection angle is set to ± 10 *deg*.

After comparing all of the tested controllers we have decided to go with the PI controller for the implementation on the real world system, as it provides the wanted

properties such as zero error to reference in the steady state and goes without any unwanted oscillations on the input to nor output of the system. The oscillation of the control surfaces could cause problems and lead to unforeseen consequences, especially at high velocity, approaching speeds close to and above Mach 0.7. And although we will not achieve such speeds, only approaching Mach 0.58 according to our simulation of the flight of the Vanguard rocket, the influence of such oscillations could still have large impact on the flight performance of the rocket.

Chapter 6
Design of control laws for roll stabilisation

One of the goals of this work, and the most important, is to design a controller for control of the roll rate of the Vanguard rocket during flight. So far we have studied the physics behind the flight of the rocket and build simulations concerning the flight behaviour of the Vanguard rocket. We have also created a plug and play simulation environment with the use of the game Kerbal Space Program, where we studied the behavior of the vehicle with different kinds of controllers implemented to control its roll rate . It was thanks to this investigation in the KSP environment that we decided to use the PI controller for the control of our real world system. In this chapter we will design such controller and test it along with a few other controllers, using our SIMULINK flight simulation of the Vanguard rocket to see how they compare in performance in a simulation tailored to the real world system.

6.1 Analytical design of a PI controller

In chapter 4.3 we identified the first order system determining the relation between the input on the control surfaces in degrees of deflection and the roll rate of the rocket. We have modeled this on a "perfect" rocket with no roll induced by asymmetry of the main stabilizers or other sources. In other words the rocket in the simulation would not spin if we haven't have put a desired control surface deflection, as an input into the system. Now we will use the knowledge of the identified system to design a controller for stabilizing the rocket around the point of a set roll rate reference.

As we will be using a PI controller to govern the first order system we can use a more analytical approach then we used previously when designing the controllers to be tested in the KSP environment, 5.3. In the mentioned chapter, we utilized the root locus method, using it in an iterative design process until we reached the wanted parameters, mainly focusing on a short settling time. Now, we will use and analytical approach relying on an approximation of a first order system with a PI controller with a second order system. This is possible, as the first order system with a PI controller has the same characteristic polynomial in its closed loop tranfer function as a second order system, [17, pg. 212]. We will show this fact by comparing the two transfer functions. The term (1) shows a transfer function of a common second order system, where ω_n is the natural frequency of the system and ζ is its dampening.

$$N(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{1}$$

Now we will produce the closed loop transfer function of our system to compare it to the transfer function of the second order system. The closed loop transfer function is given by the term (2).

$$G(s) = \frac{C(s) \cdot H(s)}{1 + C(s) \cdot H(s)} \tag{2}$$

The functions $C(s)$ and $H(s)$ correspond to the controller and plant respectively. These are then given by terms (3) and (4). When we supplement them into the we get the form given by term (5).

$$C(s) = \frac{K_p s + K_i}{s} \tag{3}$$

$$H(s) = \frac{A}{\tau s + 1} \tag{4}$$

$$G(s) = \frac{A(K_i + K_p s)}{s(\tau s + 1)\left(\frac{A(K_i + K_p s)}{s(\tau s + 1)} + 1\right)} \tag{5}$$

This then simplifies to term (6), which we can directly compare to the transfer function of a second order system (1). As we can see, the characteristic polynomial of the two transfer functions is in the same form and so we can consider them equal to each other, thus making the approximation of the first order system with a PI controller by a second order system possible.

$$G(s) = \frac{A(K_i + K_p s)}{\tau s^2 + (AK_p + 1)s + AK_i} \tag{6}$$

To make the characteristic polynomials truly equal we divide the characteristic polynomial of the closed loop transfer function by τ. The final form of the polynomial is then as follows, ??.

$$s^2 + \frac{AK_p + 1}{\tau} s + \frac{AK_i}{\tau} = 0 \tag{7}$$

As the two polynomial are now equal we can formulate the terms for K_p and K_i, [17, pg. 212].

$$K_p = \frac{2\tau\zeta\omega_n - 1}{A}$$

$$K_i = \frac{\omega_n^2 \tau}{A}$$

Now, considering we know the terms defining the two control parameters we can observe that these are dependant on the dampening and natural frequency which now become variables on which we will base our design. These are then themselves dependent on settling time and overshoot properties as given by (8) and (9), [17, pg. 140, 186].

$$\zeta = \frac{-ln(\%OS/100)}{\sqrt{\pi^2 + ln^2(\%OS/100)}} \tag{8}$$

$$\omega_n \doteq \frac{4}{\zeta\tau} \tag{9}$$

We will now proceed with choosing the values for these two parameters to design the controller. While we have chosen multiple values to create multiple, different controllers, we only show one in detail, it being the one we later decided to implement onto the rocket. We will compare the performance of the remaining controllers in the next section, 6.2.

From the experiments we carried out in the KSP environment we know that we can go with a low settling time and want to limit overshoot, as we want to minimize any possible oscillations in the response of the system to the changes in reference. This said, the overshoot also allows the controller to be more aggressive an so the the system can be stabilized much quicker and the overall deviation from a set reference should

be smaller should any errors or destabilizing effects occur. With this in mind we have chosen a settling time of $\tau = 1.2\ s$ and maximum overshoot of $OS = 7\ \%$. With this input data the analytical solution resulted in the following PI controller, (10).

$$C(s) = \frac{2.89s + 1.877}{s} \tag{10}$$

When we take a look at the root locus of the closed loop system in figure 6.1 we can observe the zero at $-0.649 + 0j$ and two poles of the system brought over from the open loop at $0 + 0j$ and $-0.0033 + 0j$. The closed loop poles of the system could then be found at $-0.7162 + 0j$ and $-6.9505 + 0j$. As these are movable to allow for tweaking of the controller design we haven't plotted them directly into the root locus figure, but it is important to note that they move on the blue and orange parts of the root locus plot respectively.

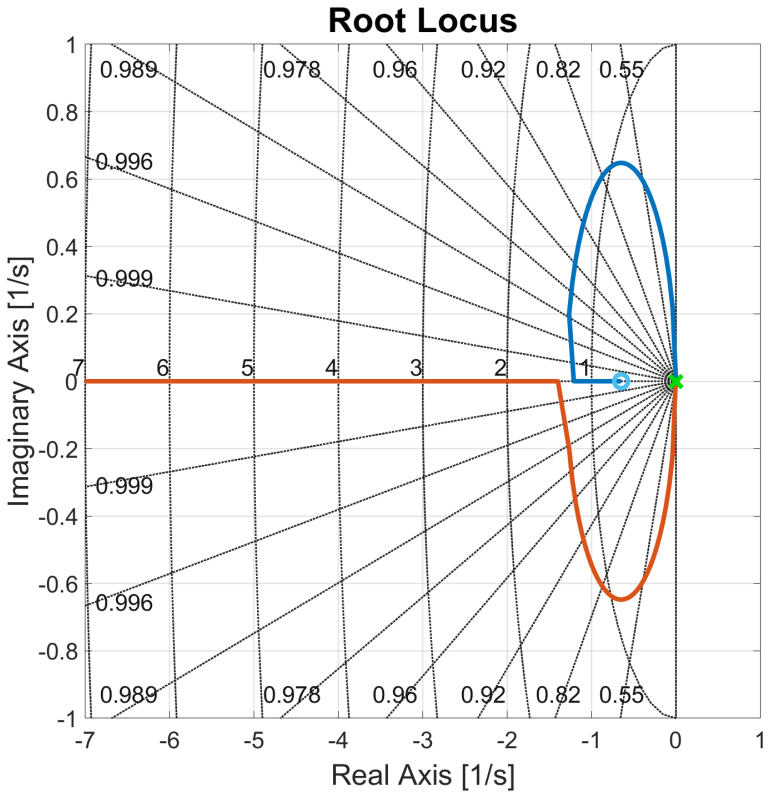

Figure 6.1. Root locus of the closed loop of the system

We can also look at the Bode diagrams of the systems open loop in figures 6.2 and 6.3. The gain margin of the system is infinite, indicating a great robustness in a case of an increase of the gain of the system. The phase margin is then 85.2 deg at a frequency of $f = 7.69\ rad/s$. This is well above the industry standard of minimum 30 deg and indicates that the system is robust to the occurrence of lag on system input. We can conclude, considering the values of the gain and phase margins, that the system is stable, if anything it might be too robust.

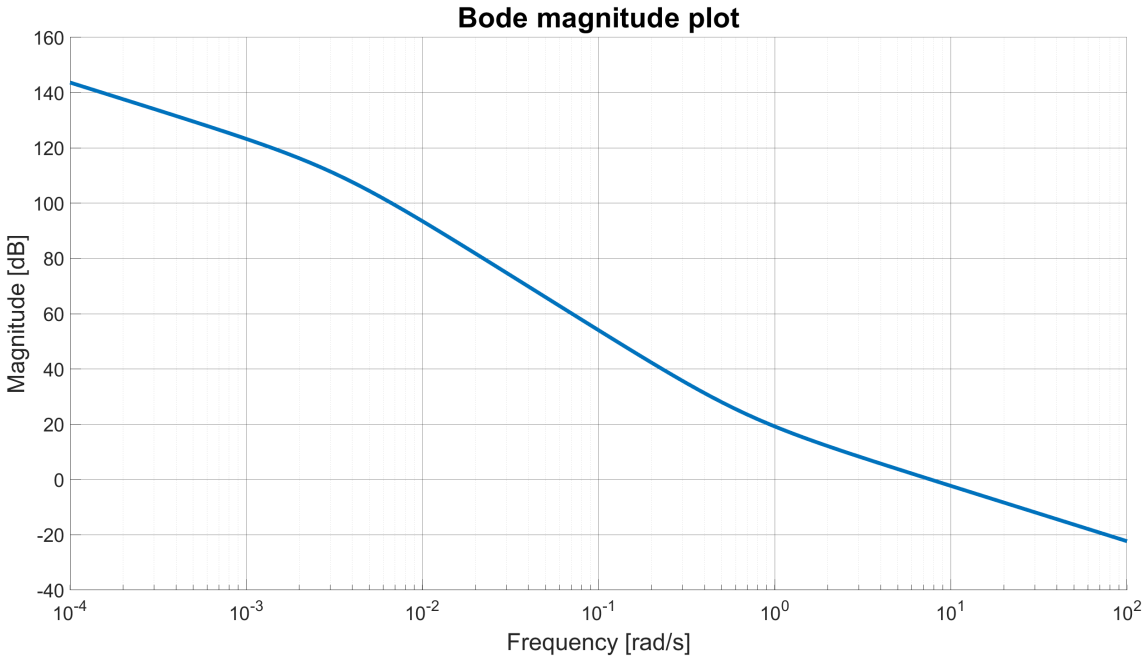

Figure 6.2. Bode magnitude plot of the system's open loop

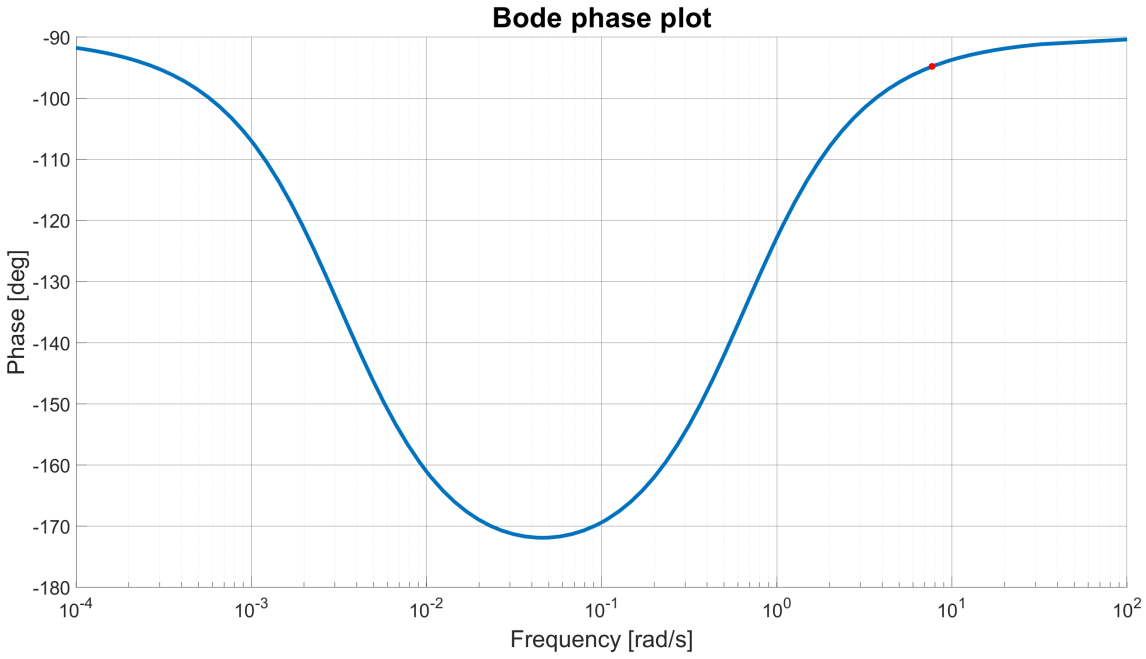

Figure 6.3. Bode phase plot of the system's open loop

Finally, we can take a look on the step response and simulated flight performance of the controller on the system, these can be seen in figure 6.4. We will talk about the setup of the flight simulation in the next section and will also compare other controller designs to this one. Overall, we were pleased with the simulated flight performance of the controller and so decided not to make any changes to its design. Although improvements could be done, such as reducing the exaggerated robustness of the controller, considering we had a larger time window for the controller design in this part of the project.

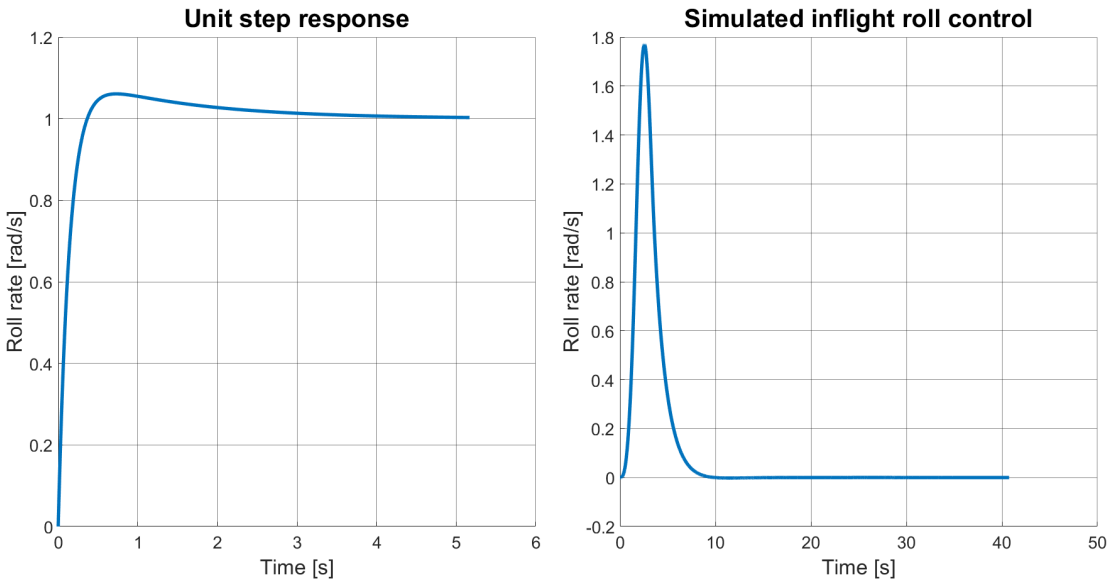

Figure 6.4. Step response and simulated flight performance of the PI controller chosen to be implemented on the Vanguard rocket

6.2 Controller testing within the Vanguard flight simulation

With the design of the controller finished we will now proceed with the construction of a simulation to test the various controllers we designed and in result will chose one of them. We will base our validation simulation on the roll simulation we have build in chapter 4.2 which we will modify to induce roll from the rockets imperfections and will implement the control loop.

We started with the modification of the simulation by implementing an error to induce roll rate throughout the flight. We do so by modifying the differential equation (2) which describes the roll acceleration of the rocket this can be seen in equation (11).

$$\dot{p} = \frac{\overline{q}}{I_1}\left(\frac{l^2 S}{2V} C_{lp} p + l_a S_a C_{l\delta a} \delta a + error\right) \tag{11}$$

The resultiong inflight roll rate induced by such error can be seen in figure 6.5. The value for the induced error was set at $error = 0.00001$. We can see that when uncontrolled the roll rate of the rocket gradually increases until it reaches its apogee, roughly at the 20 second mark. The maximum roll rate at this point reaches aproximately 60 rad/s or 9.55 rps. We then implemented the control loop to reduce this rotation, implementing a similar control scheme to the one seen in figure 5.10. The only difference being we will go without the multiplication by -1 at the output of the controller. This is because we have previously modeled the dynamic of the rockets roll rate not to be negative with relation to the control surface input as it is with KSP and in reality. This does not affect the simulation in any negative way, we just need to mind this when later implementing the chosen controller on the rockets flight computer, where the multiplication by -1 must appear for the control scheme to hold and the system not becoming unstable.

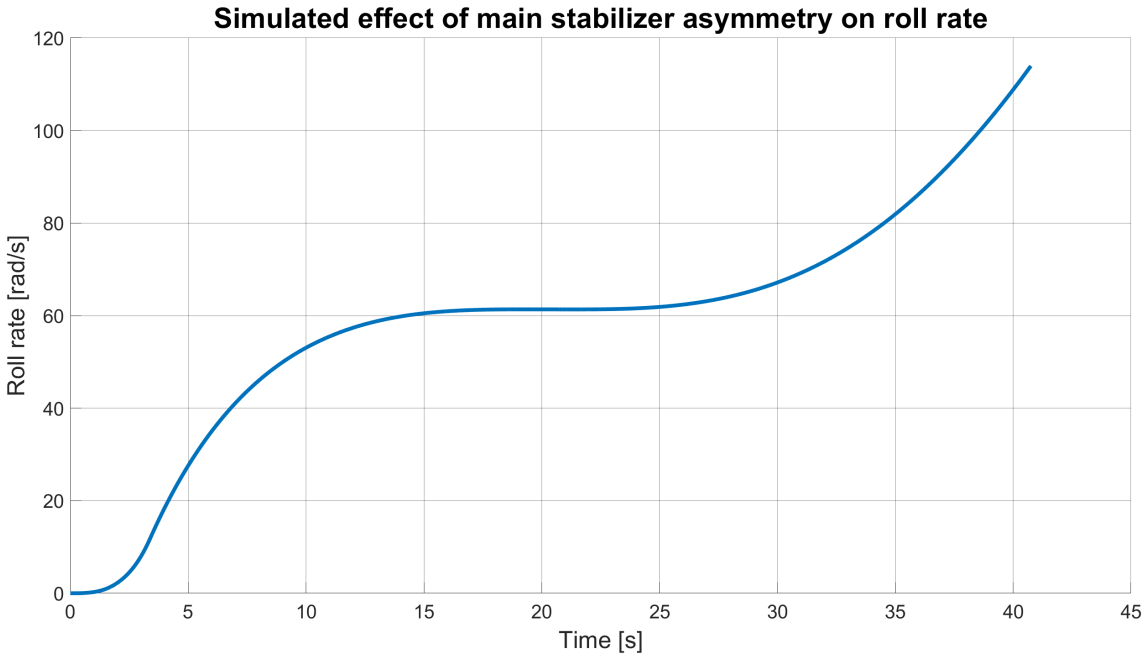

Figure 6.5. Simulated roll rate of the rocket in flight with an induced error from the misalignment of the main stabilizers

We have already tested the controller designed in the previous section 6.1, the results of which can be seen in figure 6.4. We will now show the simulation results of other controllers we have designed and tested in this way. First we will show the result of the PI controller designed for the testing in the KSP environment. The step response of this controller and its performance in the KSP environment can be again viewed in figure 5.13. The flight performance within the Vanguard simulation can be seen in figure 6.6.

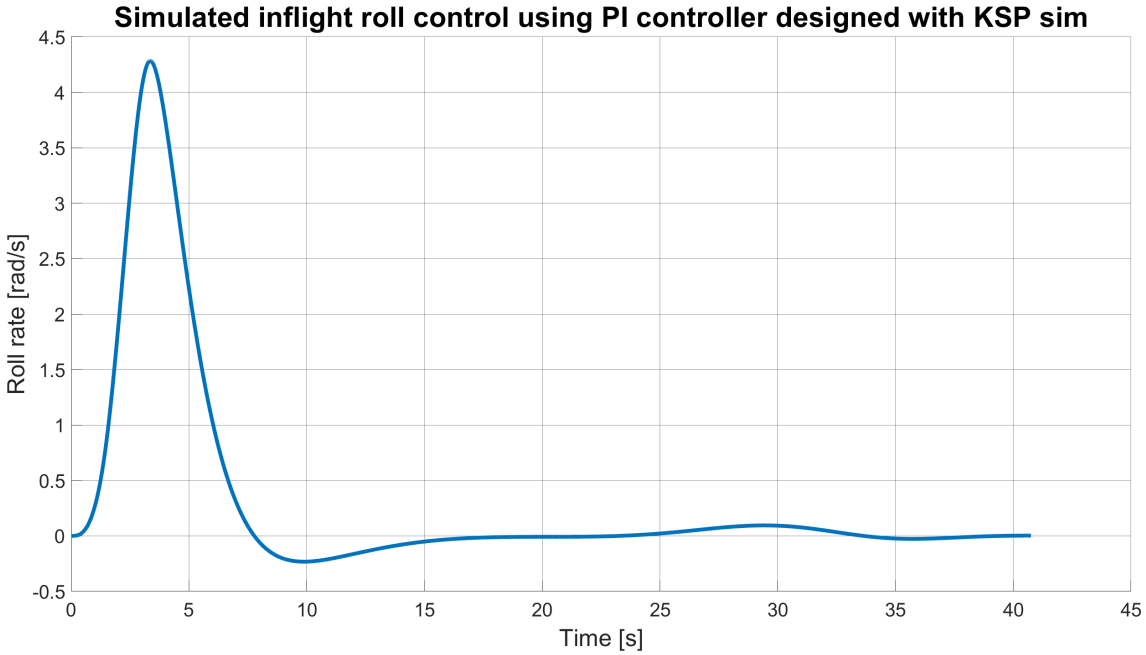

Figure 6.6. Simulated roll rate of the rocket in flight utilizing a PI controller originaly designed for the use in the KSP environment

There we can clearly see that the the controller is too weak and is not able to fully suppress the rotation of the rocket. Similar behaviour manifests with the following controller designed using the design process laid out in the previous section 6.1. This controller is defined by its transfer function (12) and its step response and flight performance can be seen in figure 6.7. The design restrictions chosen for the controller were a settling time of 2 seconds and overshoot of 5 %.

$$C(s) = \frac{1.733s + 0.4667}{s} \tag{12}$$

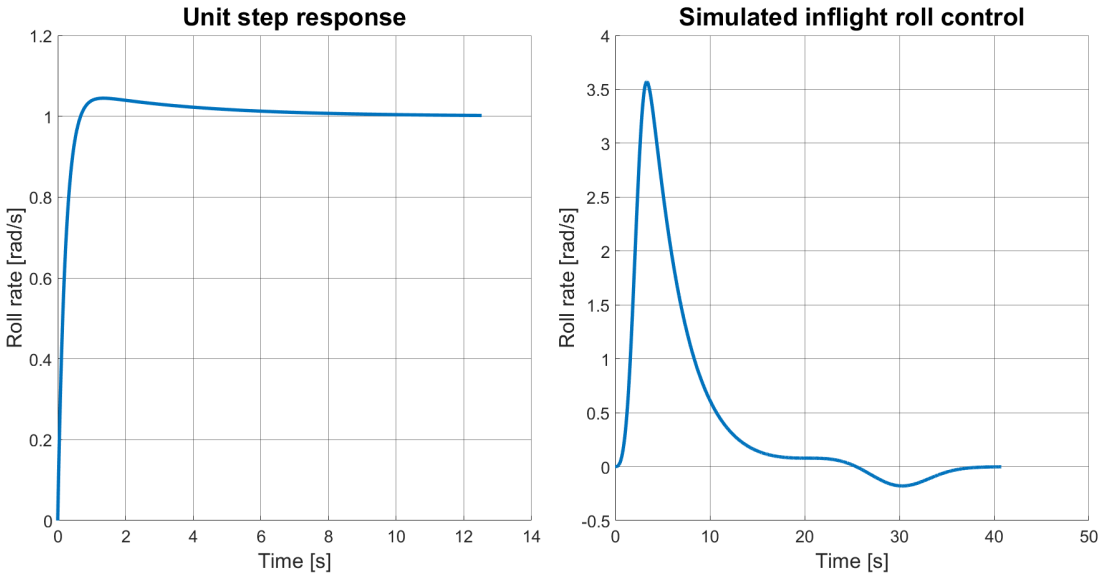

Figure 6.7. Step response and simulated flight performance of a underpowered PI controller, designed using the analytical design approach

This kind of design, although achieving smaller value of the maximal roll rate than the controller designed for KSP testing, is weaker in performance in sense of response speed and overall roll suppression.

A opposite problem can bee seen with the controller which step response and flight performance can be seen in figure 6.8. This controller is defined by transfer function (13) and was also designed using the analytical approach we have outlined with the parameters $\tau = 2.2\ s$ and $OS = 15\ \%$. This results in the integral parameter of the controller to be larger than the proportional parameter.

$$C(s) = \frac{1.576s + 1.714}{s} \tag{13}$$

The controller achieves a smaller maximum roll rate than the underpowered model and achieves to stabilize the system around the 10 second mark, same as the chosen PI controller (10). And while it tries to settle the roll rate quicker than the previous controllers the resulting stagger and the rotation in the other direction is unwanted in our design. This is why we ended up not going with such a design, instead opting for the previously described roll controller (10).

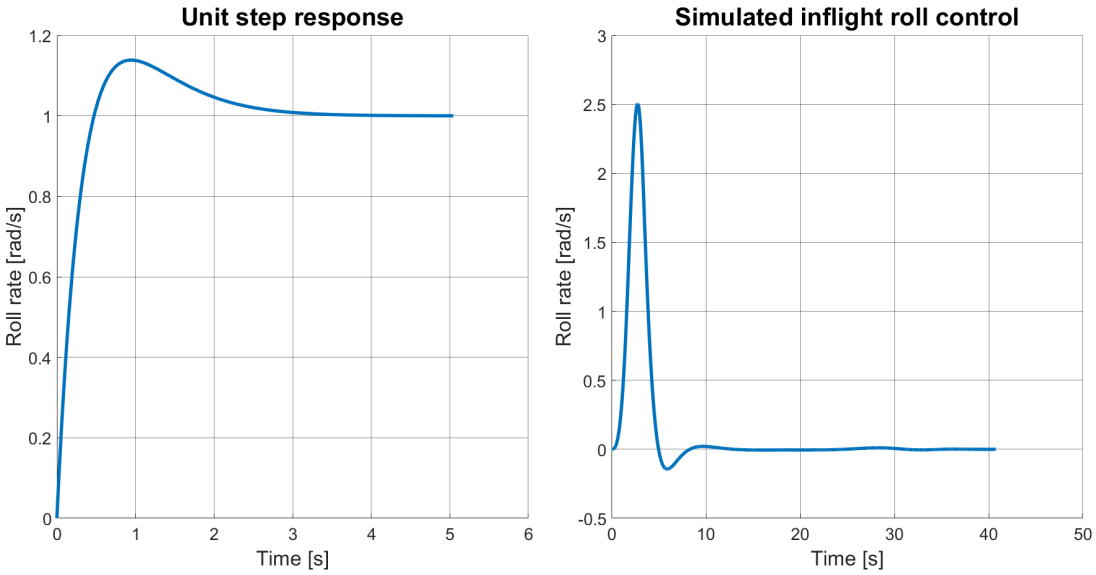

Figure 6.8. Step response and simulated flight performance of a PI controller with and overpowered integral parameter, designed using the analytical design approach

Chapter 7
Controller implementation and testing

In this work so far, we have addressed the modeling and simulation of the Vanguard rocket 4, we have studied the possible types of control to implement for the purposes of a roll rate control system onboard the rocket 5 and have designed the PI controller and tested it within a simulation 6. Now it is time to implement this controller into the rocket's flight computer and integrate our work with the hardware, about which we have written in chapter 3.2. We also want to propose an experiment to validate the controller and to test the roll control system as a whole out of a simulation and in the real world. This is important to catch any problems, which might occur with the system before its maiden flight.

This is also a good time to once again establish the team nature of this kind of a project. As we have previously written before in chapter 3, our work is a part of a larger project and so cannot go without the involvement of others. I would once again want to mention the dedication of the structures team, while designing and building the Vanguard rocket and the ARC section compartment in particular. I would now also like to acknowledge the assistance of fellow avionics team members whose assistance was pivotal in the process of implementation and integration of our system onto established team platforms, such as the Cimmerman flight computer. Large projects, such as this, take a lot of people with different types of expertise and just would not be possible to accomplish without their involvement.

7.1 Implementation of control laws on the Cimerman flight computer

The control program we have developed was originally planned to be implemented on the new Cimmerman PX4 flight computer but due to delays in its development we will for now implement the controller on an older Cimerman V2 mini flight computer. The Cimerman V2 mini flight computer is a legacy computer developed by the CTU Space Research team for the use on the team's rockets. The capabilities of the Cimerman flight computer include measurement of flight data as altitude, with the use of atmospheric pressure, acceleration, with the use of an onboard accelerometer, and rotation, with the use of an onboard gyroscope. The computer also includes a GPS module for the determination of the rocket's position and a LoRa module for communication with the ground during the flight.

The original concept for the archytecture of the system can be seen back in chapter 3, figure 3.3. With the implementation on the older platform we have simplify the architecture where the CAN bus is circumvented and the Cimerman and Zora modules are united into a single module where the roll control program and the servo control program run on the Cimerman V2 mini flight computer with an expander board facilitating the hardware support for the control of the servo motors.

The control program is implemented in C++ to conform to the rest of the program architecture of the Cimerman V2 mini flight computer. A library for controlling of the ST3215 servo motors was implemented and we can start on the implementation of our control program. We will now show a pseudocode of the implemented control function and describe the function of its different parts. The control function is based on the function we first showed in chapter 5 so it supports all of the types of the control we previously written about including the PD and PID controllers so that these can be quickly implemented in the case we would decide to do so in the future.

We use a data structure implementation to hold essential data for our calculations. This includes the time and error values from the previous calculation cycle, and a value for the integrator which are used in the calculating of our derivative and integrator terms. The structure also contains pointers to the servo object we are controlling and a gyroscope sensor we use to gather the measurements about the actual roll rate of the rocket. Lastly we include a roll_rate value, which we use to set a reference for the controller of the desired roll rate of the rocket.

```
typedef struct  {
   uint32_t last_time;
   float angle;
   float prev_error;

   float roll_rate;
   float integrator;
   st_servo_pid_reg_t* pid;

   UART_HandleTypeDef* servo;
   gyro_meas_t* gyro;
} st_servo_pid_data_t;
```

The control function holds similar structure to the one used in chapter 5. The function firstly calculates the error between the set reference and the current roll rate of the rocket, using the data form the gyroscope. It is important to note that we have designed the controller to work with the error value in radians per second. Because of that it is essential to make sure that the output of the gyroscope is also in radians per second. As with some models this might not hold true a correction term must be included for the transformation form degrees to radians.

$$rad = \frac{\pi}{180}\, deg$$

The function then continues with determining the time elapsed from the previous cycle and stores this value in the object ”dt”. It then immediately uses it to update the value of the integrator. As the time step is small, in order of milliseconds or smaller, the integration term changes into a sum of small increments. For this purpose we add error value multiplied by the time step to the current value of the integrator. We then proceed with calculating the proportional, integral and derivative terms, depending on the type of controller these values are either equal to zero or are non-zero values. We then sum the terms and calculate their negative value. This reflect on the fact that when the control surfaces deflect by a positive angle they force the rocket to spin counterclockwise, which is a negative direction for the gyroscope. The angle is then passed to the function which sets angle on the servo motors. At the end of the function we write the values of the current error and time into the data structure to be used in the next cycle.

```
bool control_pid(void* data){
    float error = data->roll_rate - data->gyro->ang_rate[0];
    float dt = time - data->last_time;
    data->integrator += error*dt;

    P = error*data->pid->kp;
    I = data->integrator*data->pid->ki;
    if (data->pid->tf != 0){
        D = (error - data->prev_error)*(data->pid->kd/data->pid->tf)/dt;
    }
    else{
        D = (error - data->prev_error)*data->pid->kd/dt;
    }

    data->angle = -(P+I+D)
    st_servo_set_angle(data->servo, ST_SERVO_ID_ALL, pid_data->angle);

    data->prev_error = error;
    data->last_time = time;
}
```

With the control program implemented on the flight computer we proceed with integration of the avionics into the rocket. This includes the assembly of the ARC section and the avionics compartments. As we are not using the Zora module management board, we only house the servomotors in the ARC section and connect them with the Cimerman flight computer located in the avionics section using a cable.

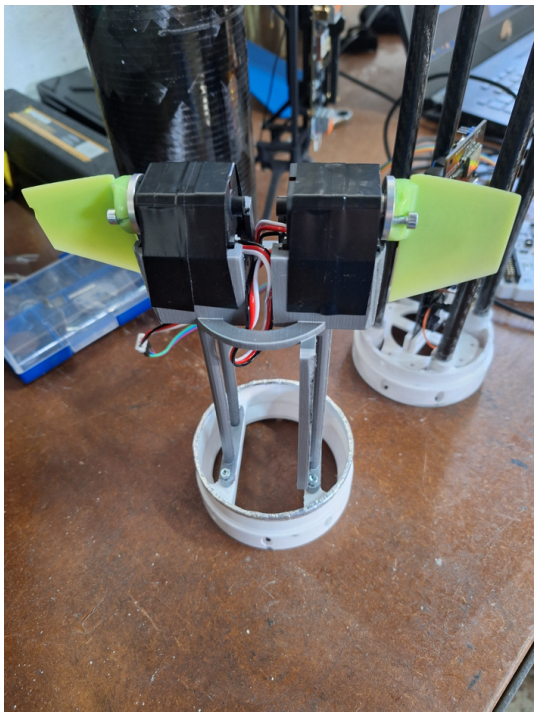

Figure 7.1. Assembly of the ARC section with the mounted servo motors and control surfaces. The avionic section of with the Cimerman flight computer can be seen in the background.

7.2 Proposal for a wind tunnel validation test for the implemented controller

The roll control system onboard the Vanguard rocket is a mission critical system with major influence on flight performance. With these kind of systems we might want to make sure they work before we proceed with launching the rocket. For these purposes we propose an experiment which might demonstrate to us the performance of the system during flight.

Wind tunnel testing is essential part of the design development of aircraft and missiles and has also taken root in the automotive industry. The primary purpose of such testing is to study the air currents flowing around the vehicle, tensometers or other kinds of force measuring sensors might then be used to measure the force generated from the vehicles body and aerodynamic surfaces. The knowledge of these forces can then be used to calculate aerodynamic coefficients such as the lift or drag coefficients and aerodynamic derivatives, such as the control surface effectiveness coefficient. We propose a wind tunnel experiment based around measurement of the roll dynamics of the rocket. The rocket would be mounted horizontally in the wind tunnel with a rotary encoder linked with the rockets body using a belt would be used to measure the rockets rotation.

During the experiment the rocket should start to spin, the spinning induced by the air passing around the rocket at high speeds. The controller would be then remotely activated and a decrease in roll rate should be observed visually and in the data measured.

For the purposes of such experiment the structures department has designed a shortened variant of the Vanguard rocket, build without a spacer compartment. And a special spinner was designed to facilitate the rotation of the rocket when in the wind tunnel and affixed to the construction. The construction of this component can be seen in figures 7.2 and 7.3.

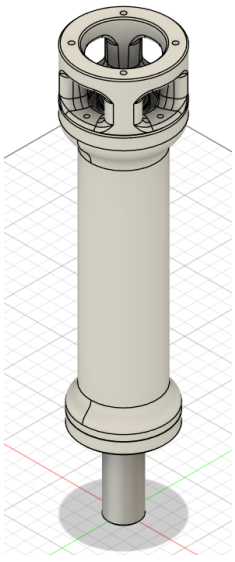

Figure 7.2. A roller assembly used to facilitate the rotation of the rocket in the wind tunnel.

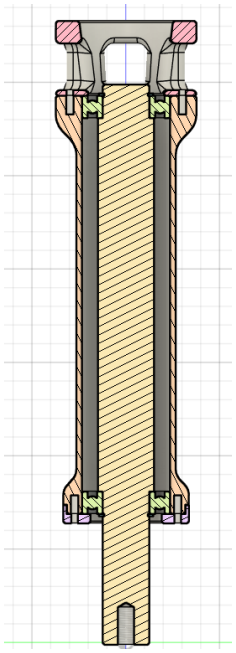

Figure 7.3. A split view of the roller assembly.

7.3 Execution of the wind tunnel validation test

With the proposal for the validation outlined we will now proceed with the preparation for its execution. As stated, we have build the shortened version of the rocket and installed the roller 7.2 into its engine compartment. The roller has the same type of mounting as the rocket's solid engine and so is mounted in its place. The rocket is then affixed to the supporting structure in the wind tunnel using the screw hole in the Rollers center rod.

Figure 7.4. Final assembly of the Vanguard rocket before wind tunnel testing. The ARC section was also shortened for the purposes of this test as it didn't need to house the Zora module management board and a the original section was found to have a defect.

The rocket was prepared in the tunnel for the first run of testing with the smaller set of control surfaces 3.6. This can be seen in figure 7.5, a rotary encoder may be seen connected at the beck of the rocket by a belt. The rocket was first subjected to the speed of the air stream of 20 m/s to check for the rigidity of the construction. At this speeds the rocket started to manifest its tendency to roll, but only oscillated by small increments of few degrees before returning to its previous position. We have gradually increased the speed in the increments of 10 m/s until we reached 60 m/s. The rocket started to shake visibly at around 30 m/s, the oscillations and the shaking increased with the air speed but the rocket refused to spin. Even with the control surfaces fully deflected to the maximum of 10 degrees deflection the rocket further refused to fully spin. With this result we have stopped the attempt.

We have discussed the possible problems that may be associated with the rocket not spinning and proceeded to take of the belt connecting the rocket to the rotary encoder. We have also swapped the smaller, primary control surfaces for their secondary larger variant 3.7. This setup can be seen in figure 7.6. The purpose of the change to the setup and of the second round of testing being to test if the rocket would start to spin at all when we reduced a possible source of resistance and would provide a larger force by the control surfaces. We have previously noticed that the rocket's has dropped by a few centimeters during the experiment. It was later measured that the angle of deflection

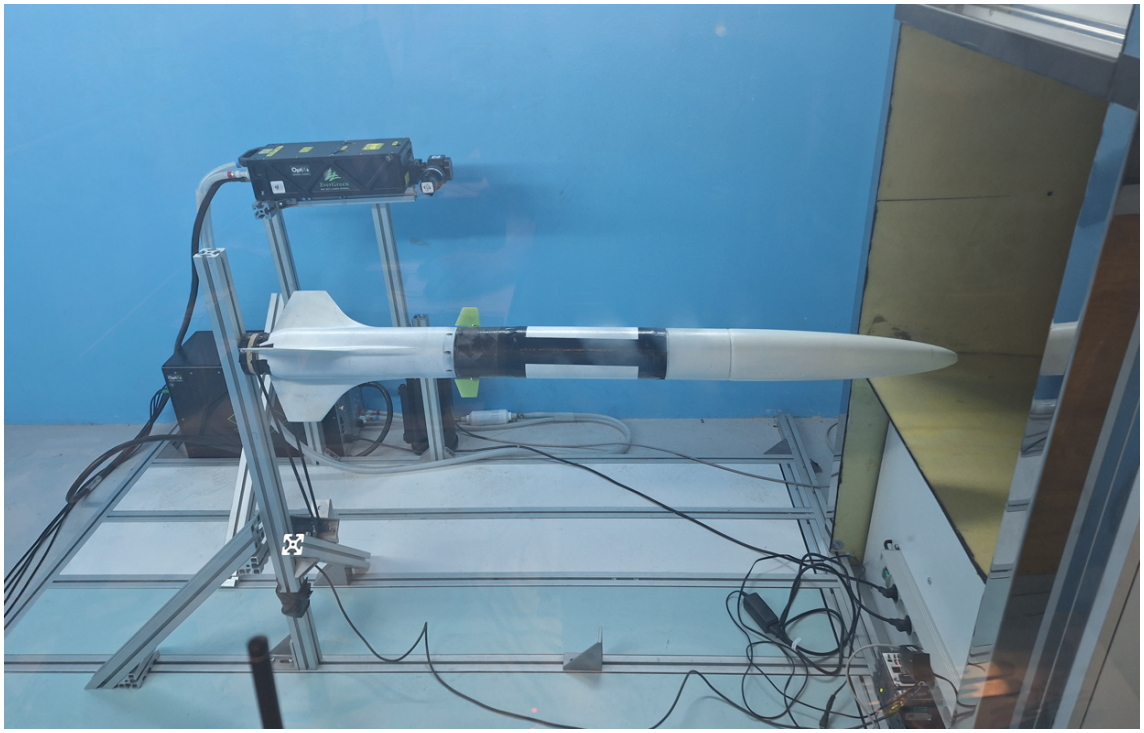

Figure 7.5. First round of the wind tunnel testing including the smaller primary design of control surfaces.

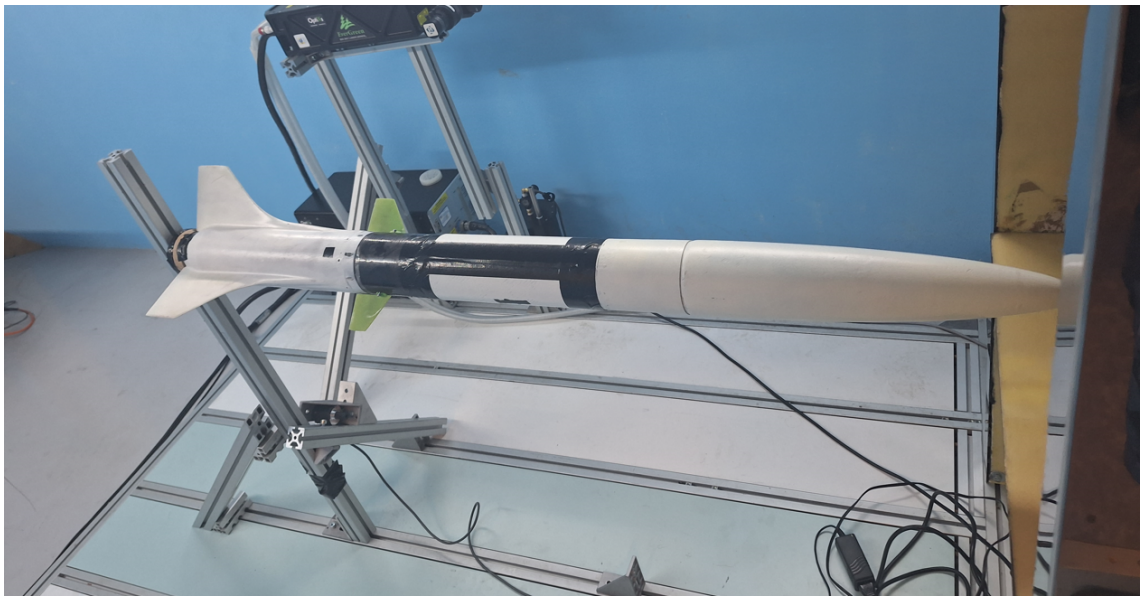

Figure 7.6. Second round of the wind tunnel testing including the larger secondary design of control surfaces.

from the horizontal plane equaled almost 1 degree. We at first thought that this drop resulted in the loading of the bearing, eventually overloading them and fixing them in place. The real reason would be revealed only later and we will return to it shortly.

The second round of testing largely resulted in the same way as the first one, the oscillations and shaking of the rocket have increased but the rocket still refused to spin. It was only after we fully deflected the control surfaces to one side and then fully to the

opposite when the rocket spun multiple times and then once again settled in place. It was clear that we have shortly overcome the forces holding the rocket in place and made clear that the fixation of the rocket could not result from the loading of the bearings alone as the forces resulting from the flow of air at speed of 70 m/s should be more powerful than any forces resulting from the weight of the rocket. It was only after later investigation of the footage from the first and second rounds of testing, as well as consultation with personnel from the institute of aircraft technology who facilitated us in the wind tunnel, when we discovered the true reason for the problems we have experienced with the rocket.

As the nose of the rocket drooped, the whole of the body slightly tilted which resulted in non zero angle of attack on the main stabilizers. The resulting lift being generated by the air stream held the rocket in place and prevented it from spinning. This came with the attempt of putting a flight version of the rocket, which does not bear horizontal loads well, into the wind tunnel. More rigid model would be needed to prevent this from occurring. Another option would be to place the rocket in a vertical wind tunnel, since the rocket is built to withstand vertical loads as opposed to horizontal loads [10]. Because vertical wind tunnels are not accessible by us, in the future a more rigid models would need to be build to carry out any testing in the horizontal wind tunnel or we would need to proceed with flight testing only.

Chapter 8
Results

At the start of this work, we set a series of goals aimed at advancing our understanding of flight characteristics of the Vanguard rocket and designing a control laws for the control of the roll rate of the rocket during flight using its aerodynamic control surfaces. In this chapter we will reflect on those objectives, assessing the extent to which the goals were achieved and the lessons learned during the course of the project. The following points will outline the results to each of the goals set at the beginning of this thesis:

- The first goal set was to model the flight dynamics of the Vanguard student rocket and construct a flight simulation based on our study of those dynamics. We have first studied translational and rotational flight dynamics in chapter 2, we then introduced the studied platform, the Vanguard rocket, and stated its basic characteristics in chapter 3. Here we have also introduced the active roll control (ARC) section, the conception and hardware portion of the system designed to control the roll rate of the rocket during flight. We have then used this knowledge in chapter 4 to construct first a translational simulation of the rockets flight in SIMULINK. Here we have simulated the influences of atmosphere on the rocket such as the change in drag with altitude as well as the increase of thrust of the engine with said altitude. The resulting simulation giving us understanding of the velocity and altitude of the rocket at various stages of the flight.

  We have then proceeded to build upon this simulation to simulate the roll of the rocket during flight, where we gained understanding on how the roll rate of the rocket might look light with different control area deflection angles and aerodynamic derivatives in place. As we were unable to clearly identify the system due to it being not completely a first order system and instead a culmination of multiple dynamics, we have lastly carried out a separated simulation in MATLAB using fixed values of velocity and dynamic pressure from multiple points in flight and ended up averaging the results to gain an estimation of the first order system behind the roll movement of the rocket with respect to the deflection angle of the control surfaces and have identified said system for purposes of further work.

- The second goal we did set was to transform Kerbal Space Program (KSP) into a usable simulation environment in which we would be able to test controllers and more advanced control programs. In chapter 5 we have laid out how can be KSP connected to python program or a program in different programming language like C# or C++ with the use of the kRPC mod. We have also proposed two kinds or architecture for connecting the KSP to MATLAB/SIMULINK with the use of such program bridge. We have then simulated multiple flights of a simple rocket and discovered that while KSP not exactly compare to real world data it still adheres to the dynamics of the real world systems. This is because the simulation differs in the environment compared to the real world and the options for building aerospace vehicles in the stock game are somewhat limited. Both of these problems are solvable

with further modding the base game. As such we work with a non-calibrated high fidelity model.

Overall we established that, while not particularly precise to the real world in terms of precise values, the dynamics of flight and functioning of systems do hold and the simulation environment is usable to for testing of controllers and more advanced control programs. Moreover, the plug and play nature of the environment allows for rapid development and testing of ideas, allowing us to see what might and might not work on the real world system so we can get an idea of what kind of system we want to implement on the real world vehicle.

- The third goal we set is closely tied to the second one in the sense we were to design multiple types of controllers for roll stabilisation of the rocket in flight and test them in the KSP simulation environment. We have formulated the logic behind the control loop based on the identified system in the later parts of chapter 5. We then written a basic controller structure in Python allowing for implementation of P, PD, PI and PID controllers. We then designed said controllers and tested them in the KSP environment. Base on the flight performance we have chosen the PI controller for the control of roll rate on our real world platform.

- The fourth goal set dealt with the design of the controller for the attenuation of the rolling motion on the real world platform, that being the Vanguard rocket. In chapter 6 we laid out an analytical approach to PI controller design and showed our best design. We then proceeded to compare it to other designs we developed using inflight roll rate simulation in SIMULINK we have developed as part of the first goal in chapter 4.

- The fifth and last of the goals set presented us with the task of implementing the choosen design from the controllers we developed, as described in chapter 6, on the Vanguard's flight computer and to propose an experiment allowing us to validate the functionality of the controller outside of a hot launch. We have done so in chapter 7, first implementing the control function onboard the Cimerman V2 mini flight computer (a team developed computer platform) in C++. We then proposed a wind tunnel experiment to test and validate the functionality of the controller. We then attempted to execute this experiment, but failed due to unforeseen reasons linked with the rigidity of the flight rocket. We have also talked about the possibilities of mitigating such issues in the future.

Chapter 9
Conclusion

Missile control and guidance is at the forefront of today's aerospace technology development, and enables us to create better defensive and offensive systems as well as allowing us to explore the universe around us. As the CTU Space Reseach rockets so far did not include any kind of active control, we were faced with the task of creating such system for the Vanguard technology testbed rocket. Multiple options of such systems were laid before us to chose from, such as thrust vector control, deployable aerodynamic airbrake system for control of the rockets speed and altitude and a control of the rocket with the use of aerodynamic control surfaces. We have chosen the latter and proceeded with the work to implement a control system for attenuation of the rocket rolling motion during flight.

We have studied the flight dynamics of the rocket's flight which we then simulated in SIMULINK and have identified the system behind the roll rate of the rocket during flight to be a first order system. This fact might not be clear at first sight from the raw data as the roll dynamic is influenced by the rockets velocity and changes in atmospheric density, both of these dynamics then transcribe into the dynamic of the rolling motion of the rocket. Because of this it is important to perform standalone simulation at various fixed points where the velocity of the vehicle and the conditions of the surrounding environment are firmly set. This left us with understanding of the dynamics at various stages of the rocket's flight and we are then able to either design multiple controllers for the various stages or can approximate the overall system by averaging all the standalone dynamics, which we have done.

A plug and play type SITL simulation was created for the purposes of control law testing, using the computer simulation game Kerbal Space Program. We have accessed the internal simulation data of the game allowing us to make measurements and analysis inside MATLAB and SIMULINK, which also allows us to create control programs directly influencing the flight of the vehicle in game. While not perfect, the simulation allows for a quick design loop where one can quickly test idea and concepts to see if they might function in the real world. A greater accuracy of the simulation can be achieved with further modification of the stock game either by already existing mods or by creating mods of our own.

We have tested multiple controller types in the Kerbal Space Program simulation environment we have created. A P controller proved insufficient for the control of the roll rate by a set reference. PD and PID controllers also proved to be problematic as they introduced rapid oscillations to the deflection angle of the control surfaces in certain parts of the flight. Because of these reasons a PI controller was selected to control the rockets roll rate as it did not suffer from the problems associated with the previously mentioned types of controllers.

With the understanding of the direction we ought to take we proceeded with the design of a PI controller for the real world system. We have used an analytical method of design utilizing the approximation of a first order system with a PI controller by a second order system. Terms linking the controller parameters with the natural frequency and

the dampening of the second order system were established, in turn linking them to the overshoot and settling time properties we desired from the controller. We then tested the controller in the flight simulation of the Vanguard rocket we created in SIMULINK.

With the controller design finished we proceeded with the implementation of the control program on the rocket's flight computer, a Cimerman V2 mini flight computer. We then proposed and carried out a wind tunnel experiment aimed at validating the roll control system outside of the simulation. Although the experiment failed due to our inherent inexperience of working with wind tunnels, we still learned valuable lessons we intend to use in the future development of the system.

For future work we envision to build upon the things we laid out in this thesis, extending the program of incorporating aerodynamic surface control systems into the team's rockets. We seek to make the system capable of fully controlling the rocket in all three rotational axis, enabling us to stabilise the rocket against wind and possibly control the target altitude and other parameters.

References

[1] Smithsonian institute, National Air and Space Musseum. *V-2 Missile*. [cit. 2024/04/22]. Available from https://airandspace.si.edu/collection-objects/missile-surface-surface-v-2-4/nasm_A19600342000.

[2] Smithsonian institute, National Air and Space Musseum. *Rocket, Air-to-Air, R4M Orkan (Hurricane)*. [cit. 2024/04/22]. Available from https://airandspace.si.edu/collection-objects/rocket-air-air-r4m-orkan-hurricane/nasm_A20190313000#:~:text=The%20German%20R4M%20unguided%20air,Messerschmitt%20Me%20262%20jet%20fighter..

[3] LandmarkScout. *Taifun – German Anti-Aircraft Rocket*. [cit. 2024/04/22]. Available from https://www.landmarkscout.com/taifun-german-anti-aircraft-rocket/.

[4] Ike Skelton Combined Arms Research Library. *Statistical Summary of Eight Air Force Operations, European Theater, 17 Aug 1942 - 8 May 1945*. [cit. 2024/04/22]. Available from https://cgsc.contentdm.oclc.org/digital/collection/p4013coll8/id/5407/rec/1.

[5] National Museuseum of the United States Air Force. *Ruhrstahl X-4 Air-to-Air Missile*. [cit. 2024/04/22]. Available from https://www.nationalmuseum.af.mil/Visit/Museum-Exhibits/Fact-Sheets/Display/Article/196222/ruhrstahl-x-4-air-to-air-missile/#:~:text=Allied%20bombing%20success%20in%20Germany,planes%20against%20B%2D17%20bombers..

[6] Smithsonian institute, National Air and Space Musseum. *Rheintochter R I Missile*. [cit. 2024/04/22]. Available from https://airandspace.si.edu/collection-objects/missile-surface-air-rheinmetall-borsig-rheintochter-r-i/nasm_A19710756000#:~:text=The%20Rheintochter%20(Rhine%20Maiden)%20R%20I,fuel%20rockets%20of%20the%20war..

[7] Peter H. Zipfel Ph.D. . *Introduction to Tensor Flight Dynamics*. Third ed. Modeling and Simulation Technologies, 2023. ISBN 9781793059390.

[8] Albert Einstein, trans. W. Perret, and G. B. Jeffery. The Foundation Of The General Theory Of Relativity. *Annalen der Physik*. 1916, trans. 1923. Available from https://einsteinpapers.press.princeton.edu/vol6-trans/158.

[9] Einstein, Albert, and trans. Robert W.Lawson. *Relativity, The Special and the General Theory*. Crown, 1961. Available from https://einsteinpapers.press.princeton.edu/vol6-trans/259.

[10] University of Canterbury Christchurch New Zealand , Department of Electrical and Computer Engineering and Department of Mechanical Engineering. Rocket Roll Dynamics and Disturbance - Minimal Modelling and System Identification. *IEEE*. 2010. Available from DOI 10.1109/ICARCV.2010.5707270. Available from https://ir.canterbury.ac.nz/server/api/core/bitstreams/56a03033-6e02-4958-a10a-fd7ba4e5c675/content.

[11] Box, Simon, Christopher M. Bishop, and Hugh Hunt. Estimating the dynamic and aerodynamic paramters of passively controlled high power rockets for flight simulaton. February, 2009. Available from `https://cambridgerocket.sourceforge.net/AerodynamicCoefficients.pdf`.

[12] Sesugh, Humphrey Iortyer Ph.D. and Nongo, and Kwaghger Aondona Ph.D. . Modelling of chamber pressure for rocket nozzle altitude compensation. *International Journal of Scientific & Engineering Research Volume 8, Issue 6*. June, 2017. ISSN 2229-5518. Available from `http://www.ijser.org`.

[13] Snorri Gudmundsson Ph.D. . *General Aviation Aircraft Design: Applied Methods and Procedures*. First ed. Elsevier, 2014. ISBN 978-0-12-397308-5.

[14] prof. ing. Václav Brož, CSc. *Aerodynamics of slow speeds*. CTU editorial center, 1990. ISBN 80-01-00198-9.

[15] *kRPC mod & plugin documentation*. [cit. 2024/05/12]. Available from `https://krpc.github.io/krpc/index.html`.

[16] *SPACEDOCK mod page, kRPC download*. [cit. 2024/05/12]. Available from `https://spacedock.info/mod/69/kRPC`.

[17] Gene F. Franklin , J. David Powell and Abbas Emami-Naeini . *Feedback Control of Dynamic Systems*. Sixth ed. Pearson, 2010. ISBN 978-0-13-601969-5.

Appendix A
Symbols and abbreviations

A.1 mathematical symbols and notation

s_{AB}	Displacement vector (generally denoted as 's') pointing to A wrt B.
v_A^B	Velocity vector of object A (frame A) wrt to object B (frame B).
$[a]^X$	Object described in reference frame X.
$[T]^{XY}$	Transformation matrix from frame Y to frame X.
$\overline{A}$	Transposition of tensor A.

A.2 Abbreviations

ARC	Active Roll Control section
MATLAB	A computational and programing language centered around matrix computation. Part of the MathWorks family of products.
SIMULINK	A MATLAB based graphical programming environment for modeling, simulation and analysis of dynamic systems.
KSP	Kerbal Space program, a computer game about space exploration build on a physics simulation.
SITL	Software in the loop
wrt	A short form of "with respect to".