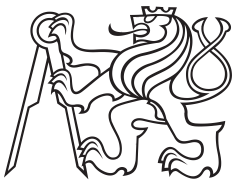
Bachelor Project



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Automatic fueling system for the CTU Space Research project

Matyáš Meisner

Supervisor: doc. Ing. Martin Hromčík, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Control Engineering
May 2024

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Meisner Matyáš** Personal ID number: **507592**

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:

Automatic fueling system for the CTU Space Research project

Bachelor’s thesis title in Czech:

Automatické tankování rakety CTU Space Research projektu

Guidelines:

This thesis aims to introduce a solution to the automatic efueling of CTU Space Research's rocket Illustria. The project arose ith the need for a more reliable solution for rocket refueling. Until now, Illustria has not been filled automatically, but the refueling process has been human-controlled. This method is relatively ineffective and imprecise, so automatic refueling is a logical step forward.
1) Explain the physical principles of flight and rocket propulsion. Familiarize yourself with types of rocket propellants and discuss > the advantages and disadvantages of their use (solid, liquid, hybrid).
2) Introduce the student rocket Illustria and GSE (Ground Support Equipment). Explain the communication method between the GSE server and the rocket.
3) Design a state machine for controlling the fueling process in Simulink and verify its functionality through simulation. Model pressure transition from a pressure tank to the rocket's pressure chamber in Simscape.
4) Verify the functionality of the solution on a real model. Convert the state machine into C++/C for it to be deployable on real hardware. Test the generation of C++/C code using static code generators in Matlab/Simulink.
5) Verify the state machine in C++ on the real system.

Bibliography / sources:

[1] Feedback Control of Dynamic Systems, 8th edition Published by Pearson (January 22, 2018), Gene F. FranklinDavid PowellAbbas F. Emami-Naeini
[2] Control of Spacecraft and Aircraft, Published by Princeton University Press (June 5, 1994), Arthur E. Bryson Jr.

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: **29.01.2024** 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

Acknowledgements

A sincere gratitude goes to the supervisor of this thesis doc. Ing. Martin Hromčík, Ph.D., who has always been very helpful and resourceful.

I also want to thank the CTU Space Research rocketry team members who made this project possible.

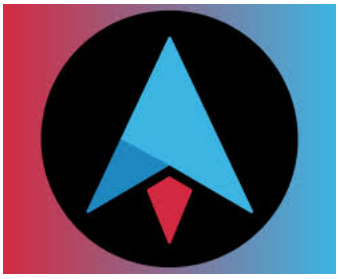


Figure 1: CTU Space Research's logo

Finally, special gratitude is extended to CTU FEL and to all who have taught me and encouraged me throughout the years, preparing me to address the vast majority of issues that arose during the project.

Declaration

I, Matyáš Meisner, hereby declare that I wrote this thesis individually and cited all used literature.

.....
Matyáš Meisner

Prague, 10. May 2024

Abstract

This thesis aims to depict the development of an automatic refueling system and a virtual twin for the Illustria rocket and its GSE (Ground Support Equipment), using MATLAB, Simulink, and Simscape. In conclusion, SIL (Software-in-the-loop) will be performed to validate the correctness of the refueling system. HIL (Hardware-in-the-loop) with the real system will not be included in this thesis for reasons discussed later, and neither will a C version of automatic refueling.

A whole chapter will be dedicated to Illustria's and GSE's structure to get a complete understanding. Since refueling is closely related to rocket propellants, rocket engine types will be introduced and compared. Furthermore, basic rocket flight principles aligned with Tsiolkovsky's equation will be explained.

Keywords: Rocket, Automatic refueling, GSE, SIL, CTU Space Research, Illustria, Control Engineering, System modeling, Rocket engines, Rocket flight principles, MATLAB, Simulink, Simscape, UDP, UnIO

Supervisor: doc. Ing. Martin Hromčík, Ph.D.
Resslova 307/9 Praha, E- 21

Abstrakt

Tato práce si klade za cíl představit vývoj automatického systému tankování a virtuálního dvojčete pro raketu Illustria a její GSE (Ground Support Equipment) s využitím programů MATLAB, Simulink a Simscape. Na závěr bude proveden SIL (Software-in-the-loop) k ověření správnosti tankovacího systému. HIL (Hardware-in-the-loop) s reálným systémem nebude v této práci zahrnut z důvodů, které budou diskutovány později. Podobně nebude zahrnuta ani verze systému automatického tankování v jazyce C.

Pro úplné porozumnění bude jedna celá kapitola věnována struktuře rakety Illustria a jejího GSE. Vzhledem k tomu, že tankování úzce souvisí s raketovými pohonnými látkami, budou představeny a porovnány typy raketových motorů. Dále budou vysvětleny základní principy letu raket a také Tsiolkovského rovnice.

Klíčová slova: Raketa, automatické tankování, GSE (Ground Support Equipment), SIL (Software-in-the-loop), CTU Space Research, Illustria, teorie řízení, modelování systémů, raketové motory, principy letu raket, MATLAB, Simulink, Simscape, UDP, UnIO

Contents

Figures

Tables

Chapter 1

Introduction and main thesis goals

1.1 Introduction

The aim of this thesis was to design the automatic refueling control system for CTU Space Research's Illustria, a modular SRAD (Student Researched and Developed) hybrid rocket. A dedicated chapter will be included to cover its construction. Until now, refueling of Illustria has been done manually. Although manual refueling has been sufficient, it lacked accuracy and required someone to control the whole process. These imperfections have motivated the development of an automatic refueling system.

Even though the ultimate goal was to develop a specific control design, this text will also include a theoretical part explaining the basic principles of rocket flight and propulsion because these topics are highly relevant to the subject.

To describe these principles, the three fundamental laws of dynamics, known as Newton's laws, will be used. Using the conservation of momentum rule, Tsiolkovsky's rocket equation, named after Russian scientist Konstantin Eduardovich Tsiolkovsky, will be derived. This equation is used to describe the motion of a rocket in a vacuum and allows for the calculation of the change in velocity resulting from the burning of fuel.

Afterwards, the advantages and disadvantages of different rocket engine types, including solid-fuel, liquid-propellant, and hybrid engines, will be discussed.

The theoretical section will conclude with a depiction of the structure of the Illustria rocket and its GSE (Ground Support Equipment). The parts required for automatic refueling will be described, and the GSE's communication protocol will be explained.

With an understanding of the theoretical principles, I will introduce the development of the refueling system. It is essential to correctly highlight that this project is taking place within the CTU Space Research rocketry team. This has both advantages and disadvantages stemming from the refueling system being built on top of systems designed by other team members. These systems will be briefly described but not explained in detail, even though the development of the refueling process relies on them. Drawbacks arise from the simultaneous development of all these systems. There are currently two main limitations. Firstly, the capacitive sensor for measuring the liquid level inside the oxidiser tank is currently unavailable. Secondly, the new version of GSE's UnIO (Universal Input Output) system will be designed soon. Therefore, the converted version of the refueling system in C would probably be unused due to its incompatibility with the new version of UnIO. Hence, this conversion will not be included in this thesis. I will adhere to its design in Simulink and MATLAB.

Illustria and its GSE form a rather complex system. There are two types of fluid (gaseous N2 and critical state N2O) with properties that the properties of water cannot approximate. Therefore, the identification techniques for hydraulic systems learned in the CTU FEL's MSD class (Modelling and Simulation of Dynamic Systems), which assume water in its liquid state, could lead to large deviations from reality. Furthermore, some SRAD valves do not have proper documentation, which also makes identification difficult.

As a result, deriving a state space model would have been complicated, and there would have been no certainty of obtaining a sufficiently accurate model. Thus, a different approach was taken. I designed a virtual twin of the Illustria and GSE in Simscape and verified it by comparing its outputs with those of the real system. The Simscape model was used to design the controllers and tune the whole refueling system. Using the virtual twin and the loopback IP address, I could also test and tune the communication between the Simulink file containing the refueling system and the one containing the Simscape model.

In conclusion, a few words should be said about the language used. I intentionally used the plural first person instead of the singular first person to maintain continuity while explaining theoretical principles. However, I switched to the singular first person to highlight the products of my work.

1.2 Main thesis goals

1. Develop a Simscape model of the rocket and its GSE emulating their physical principles.
2. Validate the Simscape model against real system data.
3. Enhance the Simscape model with a communication interface and create a virtual twin of the rocket and its GSE. Use the knowledge of blocks in Simulink that work with UDP packets.
4. Design a state machine in the Simulink Stateflow Chart block managing transitions between phases of the refueling procedure.
5. Create controllers for the inner rocket's N_2 tank pressurization and for pressure transfer between the inner rocket's tanks.
6. Perform SIL using the developed refueling system and the virtual twin.

Thesis results are discussed in the section [7.1].

Chapter 2

Rocket flight basic principles

A flight of a rocket is actually based on fairly simple physical principles. To explain why a rocket flies, we need to look at three fundamental laws of dynamics, called Newton's laws. In this chapter, we will apply these laws to a rocket flight problem and then, together with the conservation of momentum principle, derive Tsiolkovsky's equation. To illustrate, we will imagine a rocket as a conical object containing a chamber of pressurized gas. This gas is released through a small orifice and provides the thrust that pushes the rocket in the opposite direction. With this simplified rocket model, we can begin to explain the motion of the rocket using Newton's laws. Before we continue, we should note that this chapter was inspired by [22].

2.1 Newton's first law

Newton's first law is sometimes called the law of inertia. Walter Lewin, in his book *For the Love of Physics* [3], defines it as follows:

A body at rest perseveres in its state of rest, or of uniform motion in a right line unless it is compelled to change that state by forces impressed upon it.

(Lewin & Goldstein, 2011)

Applied to the motion of the rocket, the rocket remains stationary or in linear uniform motion until an external force appears that causes a change in the rocket's state of motion. So whenever we want to force the rocket to move or change its direction of motion, there must be an external force acting on it to do so. In our simplified rocket model, the thrust provided by the release of pressurized gas causes the rocket to change its state of motion. This concept will become clearer when we come to Newton's third law.

2.2 Newton's second law

To introduce Newton's second law, occasionally called the law of force, we will again use words of Walter Lewin and his aforementioned book [3].

The net force, F, on an object is the mass of the object, m, multiplied by the net acceleration, a, of the object.

(Lewin & Goldstein, 2011)

It is much more common to see it in its mathematical form.

$$F = ma. \tag{2.1}$$

We have already mentioned the word *thrust*, but we have not really explained it. As stated in [22], when fuel is burned in the rocket's combustion chamber, pressure is generated. The pressure that forces the gas through the main nozzle is called thrust. From a physical perspective, pressure is just a force acting on a surface per unit area, so Newton's second law applies.

Let us now examine equation (2.1). Suppose the rocket engine is capable of producing the constant thrust F, i.e., it is the same regardless of the amount of fuel in the rocket. By rearranging the terms in equation (2.1), we obtain:

$$\frac{F}{m} = a. \tag{2.2}$$

This equation implies that the lighter the rocket is, the greater the acceleration. Since fuel makes up the vast majority of the rocket's mass, the rocket's acceleration increases as the amount of fuel in the rocket decreases, even though the thrust remains constant. This is why rockets launch at relatively low speeds and then get faster and faster.

2.3 Newton's third law

Finally, we have come to the third and last of the three laws, which is sometimes referred to as the law of action and reaction. As always, we will begin by quoting Walter Lewin. [3]

To every action there is always an equal and opposite reaction.

(Lewin & Goldstein, 2011)

We know that the thrust forces the gas through the small orifice, which pushes the rocket in the opposite direction. Until now, we have taken this for granted rather than looking for a reason why this happens, but that ends here. If there is a force called thrust pushing the gas out of the rocket, there must also be a force acting in the opposite direction to satisfy Newton's third law. This is exactly the force that propels the rocket forward.

In summary, the basic principles of rocket flight can be explained by Newton's laws of motion. To change the state of motion of the rocket, there must be an external force causing the change, which tells us the first law of motion. The external propulsive force is the reaction force to the thrust that forces the gas out of the rocket, which is explained by Newton's third law. Finally, the lighter the rocket is, the greater the instantaneous acceleration, which is implied by Newton's second law.

2.4 Tsiolkovsky's rocket equation

With an understanding of Newton's laws of motion, we can slowly begin to derive Tsiolkovsky's equation. Before doing so, we should mention and explain a very important physical concept: the conservation of momentum. For introductory purposes, we will use a definition from the textbook [2] written by Michael Cohen.

If a system is subject to no external forces, the total momentum of the system remains constant in time.

(Cohen, 2012)

These types of systems, which are not subject to external forces, are usually referred to as isolated systems. According to [4], a rocket, together with its fuel, can be considered an isolated system when it is in space. This is because there are no external forces acting on it. Conservation of momentum, therefore, applies.

Now we have everything we need to derive Tsiolkovsky's rocket equation. Let's do it. We should note that the whole derivation is strongly inspired by [4], so the authors deserve considerable credit.

We will start by labelling certain quantities. Consider the rocket in the picture [2.1]. As we have already discussed, the rocket's mass and velocity change during its flight, which happens due to the burning of fuel. We should, therefore, determine the instantaneous mass m of the rocket, including its fuel and its instantaneous velocity v with respect to the Earth. The Earth acts as an inertial reference system. Fuel combustion produces a gas which leaves the rocket with the velocity u relative to the rocket. To obtain the velocity of the gas relative to the Earth, u must be subtracted from the rocket velocity v. Finally, we denote an infinitesimal mass of this gas by dm_g, which, multiplied by -1, is also equal to the infinitesimal change in rocket's mass.

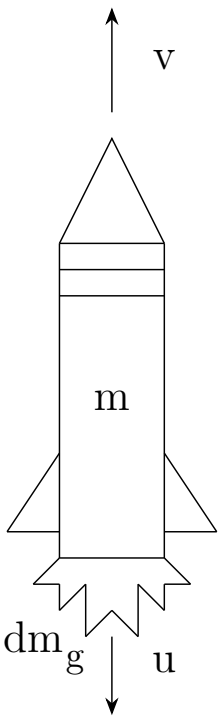


Figure 2.1: Tsiolkovsky's rocket equation - designation of quantities

The linear momentum of an object can be calculated as follows:

$$p = m \cdot v, \tag{2.3}$$

where m is the mass of the object and v is its velocity.

Applying this to the rocket shown in [2.1], we can write the following:

$$p_{\text{rocket}} = (m - dm_g) \cdot (v + dv), \tag{2.4}$$
$$p_{\text{gas}} = (v - u) \cdot dm_g, \tag{2.5}$$

where p_{rocket} and p_{gas} are the instantaneous momentum of the rocket and gas, respectively.

The total instantaneous momentum of our isolated system is then given by:

$$p_{\text{total}} = p_{\text{rocket}} + p_{\text{gas}} = (m - dm_g) \cdot (v + dv) + (v - u) \cdot dm_g, \tag{2.6}$$
$$p_{\text{total}} = mv + mdv - vdm_g - dvdm_g + vdm_g - udm_g, \tag{2.7}$$
$$p_{\text{total}} = mv + mdv - dvdm_g - udm_g. \tag{2.8}$$

Taking into account the principle of conservation of momentum, we obtain:

$$p_i = p_{\text{total}}, \tag{2.9}$$
$$mv = mv + mdv - dvdm_g - udm_g, \tag{2.10}$$
$$mdv = dvdm_g + udm_g, \tag{2.11}$$

where p_i designates the initial linear momentum of the rocket.

The right-hand side of equation (2.11) contains the product of two infinitesimally small differences, which will definitely be much smaller than the second term on the right-hand side. We can, therefore, leave it out and write:

$$mdv = udm_g. \tag{2.12}$$

Since dm (the change in rocket mass) is equal to $-dm_g$, we get:

$$mdv = -udm. \tag{2.13}$$

By reorganizing the terms in this equation, we obtain:

$$dv = -u\frac{dm}{m}. \tag{2.14}$$

The final step is to integrate both sides of this equation, each side from the initial value of the corresponding quantity to the final value.

$$\int_{v_i}^{v} dv = -u \int_{m_i}^{m} \frac{1}{m} dm, \tag{2.15}$$
$$v - v_i = -u \cdot (\ln m - \ln m_i), \tag{2.16}$$
$$\Delta v = u \cdot \ln \frac{m_i}{m}. \tag{2.17}$$

Equation (2.17) is known as Tsiolkovsky's rocket equation. This equation is used to describe the motion of a rocket in a vacuum and allows for the calculation of the change in velocity resulting from the burning of fuel.

Chapter 3

Rocket engine types and propellants

This chapter discusses the principles of how rocket engines work and introduces the types of propellants used. First, we will look at how the rocket engines work and what they do, and then we will look at the classification of the rocket engines in terms of the propellants used. We will also compare different types of rockets by mentioning their advantages and disadvantages.

3.1 Rocket engine principle

In the previous chapter, we explained the principles behind a rocket flight. We know that gases forced through a small orifice by a force (pressure), called thrust, propel the rocket in the opposite direction. This effect is described by Newton's third law of motion.

The mechanism that produces the so-called thrust is the rocket engine. Gases originate from the combustion process that takes place in the rocket engine's combustion chamber. According to [21], the propellants, comprising a fuel and an oxidiser, burn inside the combustion chamber and the emerging gases then expand through the main nozzle. As a result, the rocket is propelled forward.

3.2 Rocket engines categorized by propellants

There are many types of rocket engines, categorized by the phase of their propellant. In accordance with [21], the most commonly used are chemical rocket engines, which are based on exothermic redox reactions of their propellant. Let's take a closer look at these reactions. Jessica Wittman, in her book *Chemistry of Food and Cooking* [6], defines an exothermic reaction (process) as follows:

Matter undergoing chemical reactions and physical changes can release or absorb heat. A change that releases heat is called an exothermic process.

(Wittman, 2022)

Now that we know the meaning of the word exothermic let us examine the word redox. Redox is just an abbreviation for oxidation-reduction. For introductory purposes, we will use an explanation from the chemistry textbook [1].

Oxidation is defined as the loss of one or more electrons by an atom. Reduction is defined as the gain of one or more electrons by an atom. So oxidation and reduction always occur together; it is only mentally that we can separate them. Chemical reactions that involve the transfer of electrons are called oxidation-reduction (or redox) reactions.

(Ball & Key, 2019)

As we only wanted to give a brief overview of the reactions that take place in the combustion chamber, we will continue with engine categorization. Less commonly used rockets are thermal rockets, which, according to [21], use an inert fuel heated by electricity or a nuclear reactor. These rockets will not be discussed any further. We will now move to a deeper classification of the chemical rocket engines.

3.2.1 Solid-fuel engines

Rockets that contain solid-fuel engines typically have much simpler construction. This is due to the absence of systems pumping fuel components into the combustion chamber. The picture [3.1] shows a basic structure of a rocket with the solid-fuel engine.

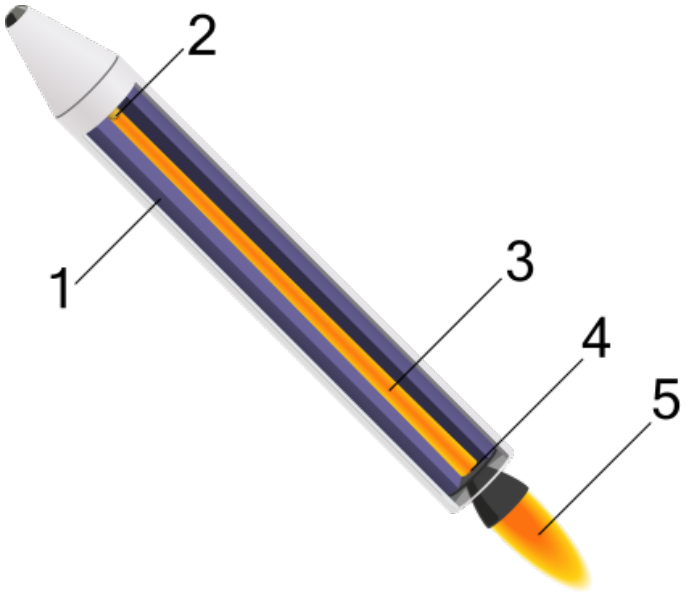


Figure 3.1: Solid-fuel rocket structure: 1) Solid-fuel oxidiser mixture, 2) Igniter initiates propellant combustion, 3) Central hole in propellant acts as the combustion chamber, 4) Exhaust nozzle expands and accelerates the gas jet to produce thrust., 5) Exhaust exit nozzle, source: [23]

In accordance with [24], solid-fuel engines use a solid-phase mixture of fuel and an oxidiser. After its ignition, produced gases flow through the central hole in the propellant, which acts as a combustion chamber. The structure of the solid-fuel rocket is indeed much simpler than that of a liquid-fuel rocket. Because of this, solid-fuel engines are commonly used in the military industry for missiles and as boosters for satellite launchers.

3.2.2 Liquid-fuel engines

Another type of rocket engine is the liquid-fuel engine. Rockets using these engines are structurally complex as they must contain tanks for both an oxidiser and a fuel. Furthermore, there must be pumping mechanisms that

transfer those propellant components into the combustion chamber. The basic structure of the liquid-fuel rocket is shown in the picture [3.2].

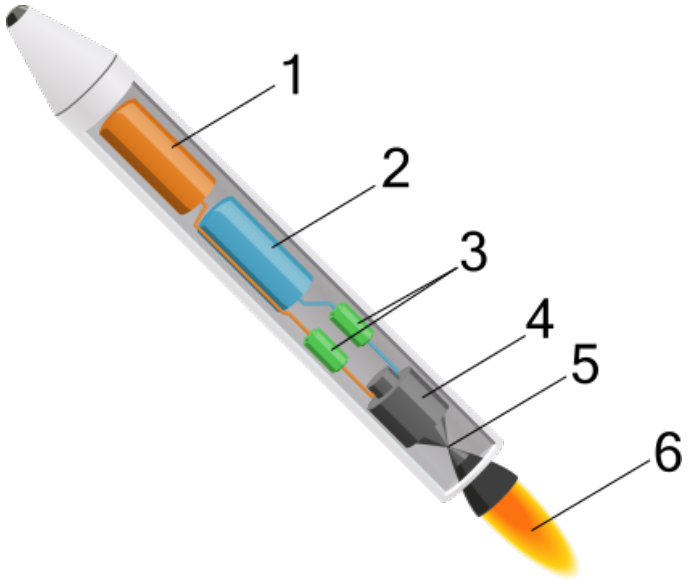


Figure 3.2: Liquid-fuel rocket structure: 1) Liquid-fuel tank, 2) Liquid oxidiser tank, 3) Pumps feed fuel and oxidiser under high pressure, 4) Combustion chamber mixes and burns the propellants, 5) Exhaust nozzle expands and accelerates the gas jet to produce thrust, 6) Exhaust exits nozzle, source: [14]

According to [15], gases are produced by mixing the liquid fuel and oxidiser inside the combustion chamber as they react. Because the fuel is not inert like in solid-fuel engines, the fuel and oxidiser must be kept separate until the ignition. Those rockets are usually used during space missions.

3.2.3 Hybrid rocket engines

Both previous engine types have their positives and negatives. Therefore, it is sometimes favorable to combine those two. By doing so, we get the hybrid rocket engines. They use a solid fuel without pre-injected oxidiser components. Corresponding to [13], the oxidiser is stored inside an individual tank and, during the combustion process, pumped into the combustion chamber. The structure of a hybrid rocket is less complicated than that of a liquid-fuel rocket but more complicated than that of a solid-fuel rocket. They usually substitute liquid-fuel rockets. The picture [3.3] underlines structural differences between types of rockets.

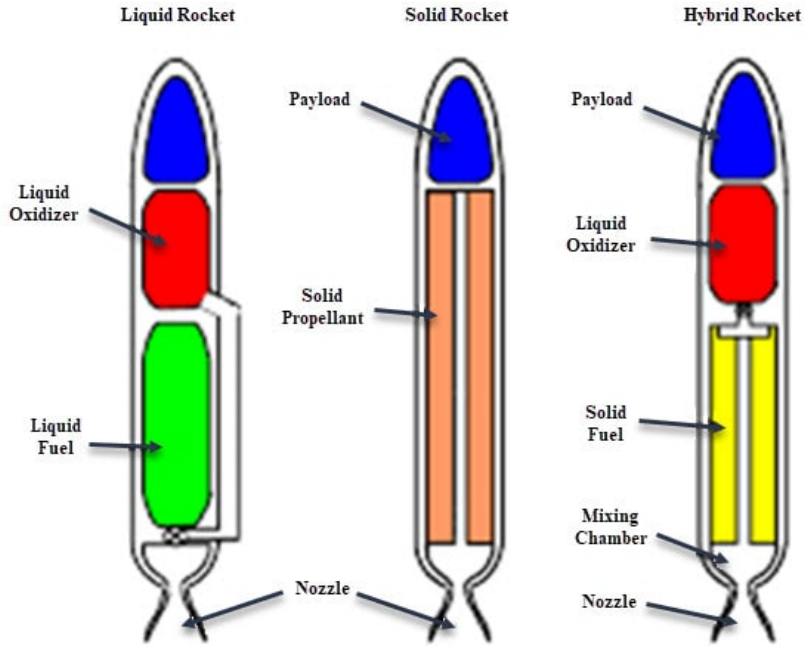


Figure 3.3: Structural differences between rocket engines, source: [12]

3.3 Rocket engine comparison

Until now, we have introduced various types of rocket engines, particularly solid-fuel, liquid-fuel, and hybrid. In this section, we will look at the drawbacks and benefits of those rocket engines. Before discussing those characteristics, we will briefly explain the specific impulse quantity as it characterizes the efficiency of a rocket motor. Let us start by definition from the Kerbal Space Program wiki [25].

The specific impulse (usually written as I_{sp}, or in-game as ISP) defines the efficiency of an engine. It is thrust per the rate of fuel consumption. Or, equivalently, it is a change in momentum per amount of fuel consumed.

(*Specific Impulse - Kerbal Space Program Wiki*, n.d.)

Mathematically written:

$$I_{sp} = \frac{F_T}{\dot{m}}, \tag{3.1}$$

where F_T is the thrust [N] and $\dot{m}$ is the fuel consumption [kg/s]. The greater the specific impulse is, the more efficient the engine is.

Knowing the specific impulse, we can start comparing all those rocket engine types. We will look at three aspects: constructional complexity, efficiency and safety. The comparison will be inspired by [11] and [13].

Considering their constructional complexity, solid-fuel rockets are the simplest type of rocket. This is because they do not require special systems to transfer the fuel or oxidiser into the combustion chamber. On the other hand, pumping propellant components into the combustion chamber must be precise for the rocket engine to work correctly. Furthermore, the fuel and oxidiser must be kept separated until they enter the combustion chamber. For that reason, constructing the liquid-fuel rocket engine is incredibly demanding, which is reflected in its price. Hybrid engines are generally more structurally complex than the solid-fuel engines but less complex than the liquid-fuel ones. Even though they involve systems for transferring the oxidiser into the combustion chamber, they do not need a special one for the fuel transfer.

We have already mentioned that the higher the specific impulse is, the greater the efficiency. According to [11], liquid-fuel engines can theoretically have higher specific impulses than liquid-fuel engines. That is because known solid oxidisers are usually more limited than liquid oxidisers. Therefore, the solid engines are generally the least efficient. However, increasing their specific impulse by adding reactive metal materials such as aluminum into a fuel grain (solid rocket fuel) is possible, resulting in greater efficiency. That is also an advantage a hybrid engine has over a liquid-fuel engine.

In this paragraph, we will discuss the safety of rocket engines as it should be addressed. We know that solid-fuel rockets are structurally much easier than hybrid and liquid-fuel ones as they must contain systems for pumping the propellant components into the combustion chamber. Although those systems can be expensive, they allow having more control over the combustion process. After the ignition of solid fuel grain, there is usually no way of stopping or controlling fuel burning. On the other hand, liquid-fuel and hybrid engines can control the amount of an oxidiser or fuel (in case of liquid-fuel engines) flowing into the combustion chamber. For that reason, solid-fuel engines are more prone to explosions. To be not so positive about liquid-fuel rockets, their combustion chamber's walls often suffer from being overheated, so they need to be cooled off. Usually, it is the fuel that simultaneously cools them. Solid-fuel or hybrid rocket combustion chamber walls are not typically exposed to enormous heat conditions as they are covered by solid-fuel grain. Moreover, the structural complexity of a hybrid or liquid-fuel rocket enables it to fail due to errors caused by system malfunctions.

Chapter 4

Illustria rocket and its GSE

This chapter aims to give an overview of Illustria and its GSE. Illustria and its systems are rather complicated, so our goal is to underline the rocket structure and describe the systems used for automatic refueling in more detail. We will also look at GSE's communication protocol, as it plays a significant role in our design. Before we start with the description, one important thing should be noted. Our primary source of information is the technical report [10] written for the EuRoC (European Rocketry Challenge) competition in 2023. Because Illustria is still in active development, there might be some minor deviations from reality. However, the overall structure should be still the same.

4.1 Illustria

In this section, we will look at Illustria's basic structure and its internal systems. Parts and systems used during the development of the refueling systems will be described more thoroughly.

The whole rocket is made out of 6 separate modules/sections: Engine section, Oxidiser tank section, Pressure section, Avionics module, Payload module, Recovery section + nosecone. Each module has its specific function and serves its particular purpose. When connected, they form a 3.5-meter-high rocket whose inner structure shows the picture [4.1].

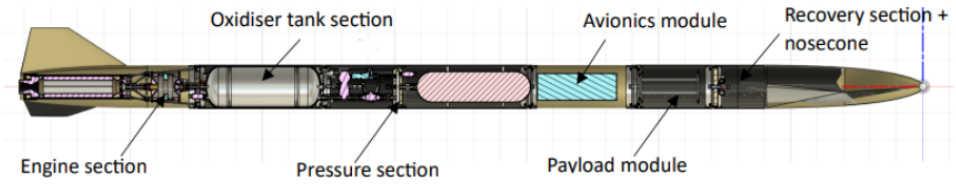


Figure 4.1: Illustria's inner structure, source: [10]

We will briefly introduce the purpose of each module and then look at the Oxidiser tank section and Pressure section in more detail.

1. **Engine section**

   It contains the hybrid engine and other equipment related to him. The engine uses ABS (Acrylonitrile butadiene styrene) as the solid grain and N_2O (Nitrous Oxide) as the oxidiser.

2. **Oxidiser tank section**

   This section houses the oxidiser tank. Liquid N_20 is stored inside, which will be refueled by the designed refueling system.

3. **Pressure section**

   It enables pressurization and depressurization of the oxidiser tank. For that purpose, it contains two valves. One is used for pressure transfer between the N_2 storage and the oxidiser tank (High servo); the other releases the gas out of the tank and depressurizes it (Low servo). There is also a hardware controller for safety reasons, which does not allow the oxidiser tank to be pressurized over a specified limit.

4. **Avionics module**

   This module contains the main flight computer and battery, which powers the electronics inside the rocket. The flight computer acts as the rocket's brain. It communicates with module management boards and the on-ground base. Module management boards appear inside some modules housing systems that need to be electronically controlled. In summary, the flight computer sends flight data to the on-ground base and allows for indirect (over module management boards) control of electronic systems inside the rocket.

5. **Payload module**

   The rocket has to be able to carry a payload according to EuRoC rules. Because of that, Illustria includes the Payload module.

6. **Recovery section + nosecone**

   For the purpose of rocket recovery, Illustria includes the Recovery section. It contains a parachute and a nosecone ejection system, the task of which is to eject the nosecone to release the parachute.

4.1.1 Oxidiser tank section

According to [10], the oxidiser tank is composed of two layers. The inner layer is made of aluminum, which prevents N_2O from coming into contact with the outer layer fabricated from CFRP (Carbon-fiber reinforced polymer). The oxidiser tank has a total volume of 7 liters and can withstand a burst pressure of 180 bars. However, the maximal operating pressure is just 60 bars. It is equipped with sensors measuring pressure (IFM PT5302) and temperature (IFM TA3145). The picture [4.2] shows the inner structure of the oxidiser tank.

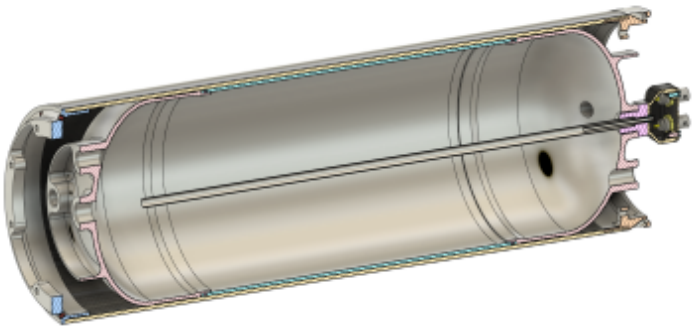


Figure 4.2: Oxidiser tank, source: [10]

4.1.2 Pressure section

It could be divided into two parts. The upper part consists of a CFRP N_2 tank. Its total volume is 3 liters, and the maximum expected operating pressure is 250 bars. The tank is equipped with a pressure sensor IFM PT5500. The lower part includes two digital servo valves; one enables pressure transfer between the N_2 tank and the oxidiser tank (High servo); the other is used to depressurize the oxidiser tank (Low servo). A hardware controller is also included because the oxidiser tank cannot withstand a long-term pressure level higher than 60 bars. The controller ensures that the transferred pressure from the N_2 tank will not overcome 40 bars.

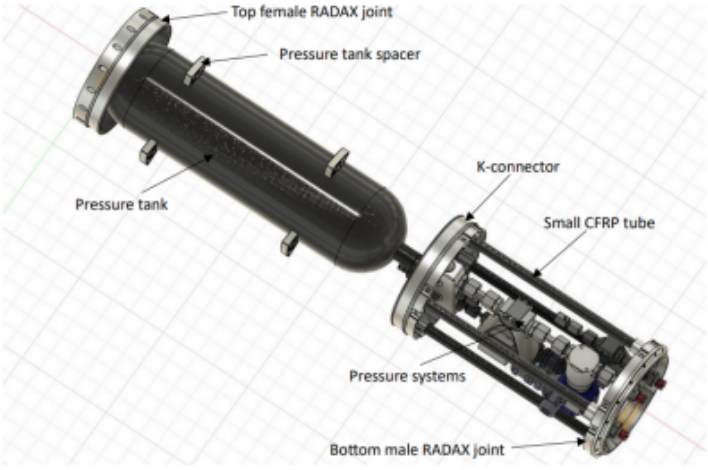


Figure 4.3: Pressure section, source: [10]

4.2 GSE

This section will first describe the GSE from its mechanical perspective, then introduce and depict the UnIO system architecture, and in the end, explain the UnIO's communication protocol. The UnIO system allows for electronic control of GSE's mechanics. The GSE is the primary system used for automatic refueling, so its description is essential for understanding the practical part.

4.2.1 GSE's mechanics

The mechanical part of the GSE comprises a series of pipes and ON/OFF ball valves (there is no command for partial opening of the valves, meaning that they can be set to either fully open or fully close) that allow controlling the flow rate of N_2 and oxidiser from outer tanks through the pipes into the rocket. It could be conceptually divided into two segments, the first controlling the flow rate of N_2 into its tank inside the rocket (further referred to as the upper line), the other doing the same thing with an oxidiser and the oxidiser tank (further referred to as the lower line). Each line includes one valve for filling a fluid and one for releasing it. Designations of the upper line and lower line come from their placement on the GSE [4.4]. Let us note that the presence of any pump has not been mentioned. That is because it is not needed. The fluids flow through the pipes due to pressure differences

between the tanks inside and outside the rocket. Before we go to another section, we should also say that the GSE contains two pressure sensors, IFM PT5500 (in the upper line) and IFM PT5302 (in the lower line), measuring pressure levels in the outer fluid tanks, containing the N_2 and oxidiser.

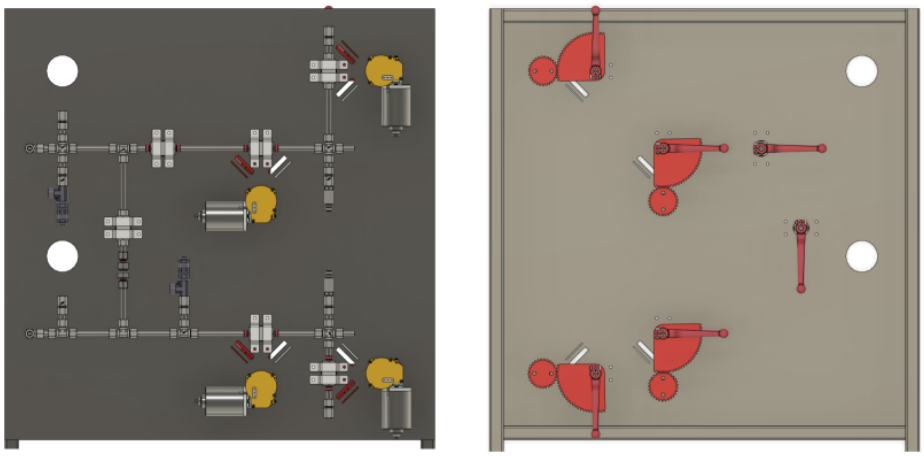


Figure 4.4: Back (left) and front (right) views of the GSE, source: [10]

4.2.2 UnIO system architecture

The UnIO system developed for electronic control of the GSE is a modular system consisting of the central motherboard and of six connectable modules (boards/cards) serving different purposes. Several ordinary buses (like SPI and GPIO) are included for communication purposes. The principle of its function is similar to that of the flight computer and module management boards. The UnIO motherboard communicates with a connected control device (like a base computer) over Ethernet or USB and indirectly controls GSE's valves or reads data from sensors via UnIO's modules. According to [10], currently available cards are CurrentLoop, IOcard, LoadCell, Capacitance, Thermistor, and Tensometer. CurrentLoop and IOcard are especially important to us as they are used to refuel the rocket, so we will describe them in greater detail. We will now examine their purposes individually. Because we have little to say about LoadCell, Capacitance, Thermistor, and Tensometer cards, as they are not used for automatic refueling, we will stick to the quoting of [10] to effectively explain their purposes.

1. CurrentLoop

   This card plays a crucial role in the development of the refueling system. It processes data from *standard 4-20 mA current output industrial*

sensors. Digitization is achieved by measuring voltage drop on a shunt resistor in line with the current loop flow. (CTU Space Research's members [10], 2023). For analog to digital conversion, 12-bit ADC is included, communicating with the primary measurement system via SPI. The card can simultaneously process data from 16 different sensors (like our pressure sensors) as it includes 16 independent channels.

2. IOcard

   The IO card handles valve control and informs us about current valve states. According to [10], it contains 12 GPIO with configurable voltage levels, four power outputs capable of driving up to 500 mA sustained or 1A in a pulse, four analog inputs, an I2C interface with configurable voltage levels for expansion, serial servo interface, 3 H-Bridge drivers capable of delivering up to 6A of power. Everything is processed by the microcontroller STM32L010. Communication between an IOcard and the UnIO motherboard is mediated via SPI.

3. LoadCell

   The LoadCell Module is used for measuring tensometric load cells configured as a Wheatstone bridge. It allows for the concurrent measurement of 4 channels, each with a dedicated ADC and a voltage regulator. (CTU Space Research's members [10], 2023)

4. Capacitance

   We constructed a cylindrical capacitance sensor for indirect measurement of the level of oxidiser in the liquid phase in its tank that is embedded into the tank. It measures the level via a change of capacity due to the difference in permittivity of liquid and gas N2O. (CTU Space Research's members [10], 2023). However, this sensor is not currently available.

5. Thermistor

   The thermistor module measures temperature data from 16 thermistors. It allows the utilization of 4-wire measurement to achieve the best accuracy. (CTU Space Research's members [10], 2023)

6. Tensometer

   Tensometric rosette measuring card We are using tensometric rosettes to measure mechanical stress in tanks under pressure during testing. (CTU Space Research's members [10], 2023)

The picture [4.5] shows the default configuration of GSE's UnIO system.

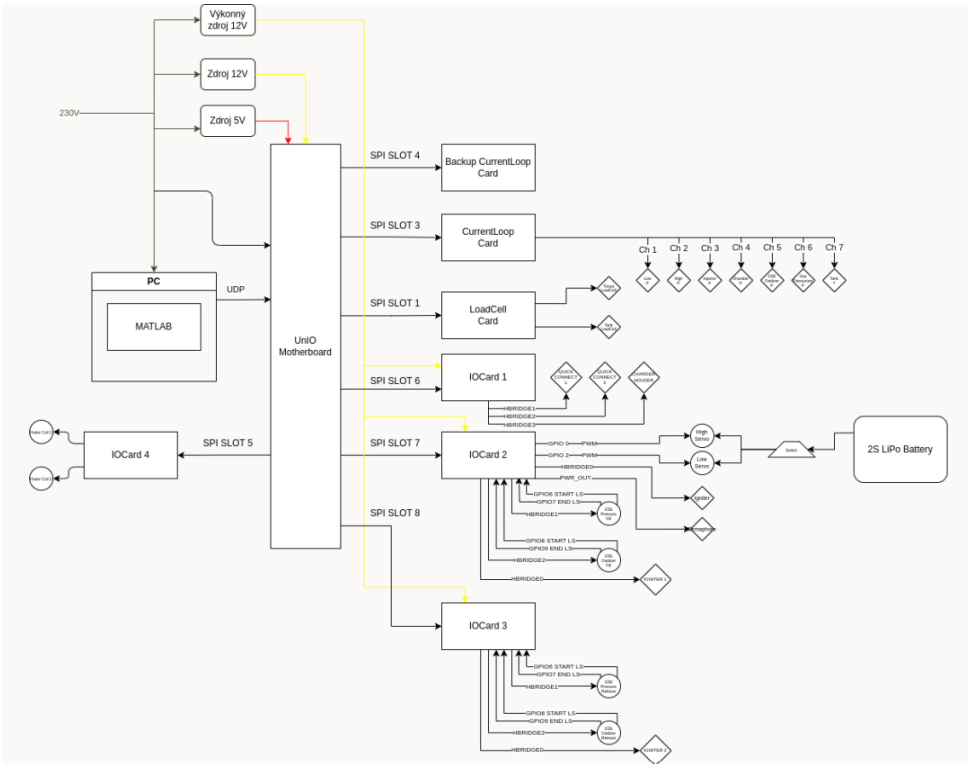


Figure 4.5: UnIO default configuration, schematics by Tomáš Ehrenberger

4.2.3 UnIO's communication protocol

Here, we will explain the format of messages (UDP packets) sent between the UnIO system and a control computer. As there is no need to use cards other than CurrentLoop and IOcard for automatic refueling, we will stick to messages corresponding to these cards. Messages differ with respect to the card type and the direction of packet flow, so we will depict them separately.

1. CurrentLoop
   a. Control computer → UnIO

      Because the CurrentLoop card is used to process sensor data, only one-way communication is needed. Therefore, messages are not sent from a control computer to the UnIO system.

b. UnIO → Control computer

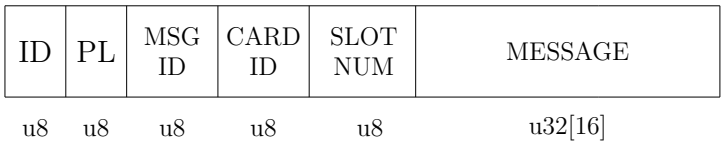


Figure 4.6: CurrenLoop message format, UnIO → Control computer

The data packet consists of 69 bytes (uint8 = u8 in the picture [4.6]); the first five are preamble. The first byte (ID) is firmly set to 0xa0 (in the hexadecimal number system) and, from our perspective, is not somewhat important. The second byte's value, 0x43, represents the packet length (PL, i.e., the number of bytes from the third byte to the end). The third byte is not again significant; it is firmly set to 0xc8. The fourth byte identifies the card type; for the CurrentLoop card, its value equals 0x02. The fifth byte's value indicates which slot on the UnIO motherboard the CurrentLoop card is connected to; according to 4.5, it is usually set to 0x02 (counted from zero). The message consists of 64 bytes, but they are conceptually divided into 16 groups consisting of 4 bytes (i.e., uint32[16] = u32[16] in the picture 4.5); each group corresponds with one channel containing data given by connected sensor. Primarily, pressure sensors measuring pressure in the inner oxidiser tank, in the inner N_2 tank, in the outer N_2 tank, and in the outer oxidiser tank are connected to channels 13, 11, 9, and 8 (in the decimal number system, counted from zero), respectively. We should note that this changed, so it does not match with the picture 4.5.

2. IOcard

a. Control computer → UnIO

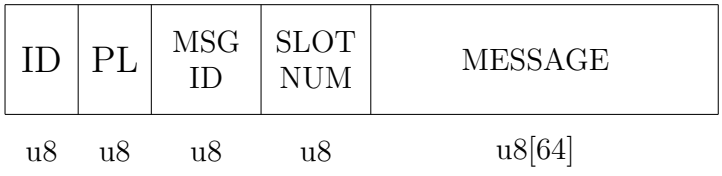


Figure 4.7: IOcard message format, Control computer → UnIO

Data packets flowing from a control computer to the UnIO system consist of 68 bytes. The first four bytes are preamble again. The first byte (ID) is firmly set to 0x96 (in the hexadecimal number system). The second byte's value, 0x42, represents the packet length (PL, i.e., the number of bytes from the third byte to the end). The third byte is firmly set to 0xc9. The fourth byte's value indicates which slot on the UnIO motherboard the IOcard is connected to. We use two IOcard cards for the refueling system: IOcard2 and

IOcard3. The IOcard2 is used to control valves for filling a fluid on GSE and to control servos inside the rocket. On the other hand, IOcard3 is used to control valves for fluid release. Let us designate valves that handle a fill of N_2 and oxidiser, H1 (H-bridge 1) and H2 (H-bridge 2), respectively. The same designation applies to release valves. For example, to control the valve filling N_2, we use the H1 valve connected to IOcard2. Analogically, we use the H2 connected to IOcard3 to control the valve that releases an oxidiser. In the default configuration, IOcard2 is plugged into slot six and IOcard3 into slot seven, so the fourth byte's values are equal to 0x06 and 0x07, respectively. The message comprises 64 bytes; the first includes command code (like open H1), and the others do not matter (can be set to zero). Relevant command codes are shown in the table [4.1].

Slot (dec)	Command code (hex)	Action
6	0x14	IOcard2 → H1 open
6	0x15	IOcard2 → H1 close
6	0x18	IOcard2 → H2 open
6	0x19	IOcard2 → H2 close
6	0x40	IOcard2 → High servo open
6	0x41	IOcard2 → High servo close
6	0x43	IOcard2 → Low servo open
6	0x44	IOcard2 → Low servo close
7	0x14	IOcard3 → H1 open
7	0x15	IOcard3 → H1 close
7	0x18	IOcard3 → H2 open
7	0x19	IOcard3 → H2 close

Table 4.1: IOcard command codes

b. UnIO → Control computer

ID	PL	MSG ID	CARD ID	SLOT NUM	MESSAGE
u8	u8	u8	u8	u8	u8[64]

Figure 4.8: IOcard message format, UnIO → Control computer

Given that the packet format is very similar to [4.6]. We will explain only packet parts that differ. The first difference is in the fourth byte, which is equal to 0x01. Next, the value of the fifth byte is equal to 0x06 in case of the IOcard2 and 0x07 in case of the IOcard3. However, what differs the most is the message part. It consists of 64 bytes, each byte serving its particular purpose. In our case, only some bytes are important, and those are shown in the table [4.2]. GSE's valves are SRAD ON/OFF valves, and their positions are

estimated by limit switches. It sometimes happens that these valves stuck. Therefore, fault state bytes are included.

Byte (dec, cnt. from zero)	Meaning	Possible values
2.	High servo pos. (deg)	0 - 90
3.	Low servo pos. (deg)	0 - 90
17.	H1 fault state	0-OK, 255-Fault
18.	H2 fault state	0-OK, 255-Fault
20.	H1 position	0-open, 127-Middle, 255-close
21.	H2 position	0-open, 127-Middle, 255-close

Table 4.2: IOcard state signalization

4.2.4 GSE's labeling summary

Before we go to the practical part, we should have a complete understanding of the GSE. Therefore, we will summarize all designations often used during the description of the refueling system.

1. The valve on the GSE's upper line used for N_2 filling
   → N_2 fill valve, IO2card H1, IO2card H-bridge 1
2. The valve on the GSE's lower line used for oxidiser filling
   → oxidiser fill valve, N_2O fill valve, IO2card H2, IO2card H-bridge 2
3. The valve on the GSE's upper line used for N_2 releasing
   → N_2 release valve, IO3card H1, IO3card H-bridge 1
4. The valve on the GSE's lower line used for oxidiser releasing
   → oxidiser release valve, N_2O release valve, IO3card H2, IO3card H-bridge 2
5. The valve enabling pressure transfer between the N_2 tank and the oxidiser tank → High servo, Pressure venting valve
6. The valve used for the oxidiser tank depressurization → Low servo, Pressure releasing valve
7. The UnIO card for sensor data processing
   → CurrentLoop, CurrentLoop card
8. The UnIO card used to control GSE's fill valves and servos inside the rocket as well as read their states
   → IO2, IO2 card

9. The UnIO card used to control GSE's release valves and servos inside the rocket as well as read their states
   → IO3, IO3 card

Chapter 5

Virtual twin design

In order to perform SIL (Software-in-the-loop), GSE's and rocket's virtual twin had to be designed. This chapter is devoted to its development. The virtual twin (further denoted as VT) enables testing the working principles of the refueling system before its application to the real model. Simulink and its module Simscape were used for that purpose. The goal was not to make an exact virtual twin of the real system but to reach sufficient accuracy to test designed control mechanisms. The core of the VT, emulating the physical principles, was designed in Simscape and will be further called the Simscape model. The whole .slx file comprising the Simscape model and corresponding communication interface is the VT itself. Because there was no fuel level sensor for the oxidiser, it was refueled manually. Therefore, there was no need to model GSE's lower line in Simscape. The developed Simscape model is focused solely on the GSE's upper line and the rocket's pressure section. Before diving into the development itself, I would like to point out that each important file is available in [5].

5.1 Simscape model

Simscape comprises many sub-libraries used for different physical domains. These sub-libraries contain blocks representing real systems such as pipes, constant volume chambers, reservoirs, etc. The pressurization of the rocket's pressure section tank involves N_2 transfer between two tanks of a constant volume. Given that N_2 is in its gaseous phase, using the Simscape gas sub-library was appropriate. Six particular blocks were used: Constant Volume

Chamber (G), Local Restriction (G), Pipe (G), Reservoir (G), Gas Properties (G), and Pressure & Temperature Sensor (G). Some blocks, such as Pipe (G) and Constant Volume Chamber (G), include thermal conserving ports that allow the modeling of energy flow through heat exchange. I used those ports to model heat exchange with the surrounding world and added blocks from the Simscape thermal sub-library: Convective Heat Transfer and Temperature Source.

1. Constant Volume Chamber (G)

   The Constant Volume Chamber (G) block models mass and energy storage in a gas network. The chamber contains a constant volume of gas. It can have between one and four inlets. The enclosure can exchange mass and energy with the connected gas network and exchange heat with the environment, allowing its internal pressure and temperature to evolve over time. The pressure and temperature evolve based on the compressibility and thermal capacity of the gas volume.

   (Constant Volume Chamber (G) - MATLAB [8], n.d.)

   I used it to model the tanks.

   Important parameters and their values in the Simscape model (others were set to default):

   a. Chamber volume
      - outer N_2 tank: 50 l
      - inner N_2 tank: 3 l
      - inner oxidiser tank: 7 l

   b. Number of ports - it can be either 1, 2, 3 or 4
      - outer N_2 tank: 1
      - inner N_2 tank: 2
      - inner oxidiser tank: 1

   c. Cross-sectional area at ports
      - outer N_2 tank: 38.48 $\rightarrow$ 7.48 mm^2
      - inner N_2 tank: A: 38.48 $\rightarrow$ 7.48 mm^2, B: 12.57 $\rightarrow$ 2 mm^2
      - inner oxidiser tank: 12.57 $\rightarrow$ 2 mm^2

   Default initial targets:

   a. Pressure of gas volume
      - outer N_2 tank: 300 bar (High priority)
      - inner N_2 tank: 1 bar (High priority)
      - inner oxidiser tank: 1 bar (High priority)

b. Temperature of gas volume
- outer N_2 tank: 20 °C (Low priority)
- inner N_2 tank: 20 °C (Low priority)
- inner oxidiser tank: 20 °C (Low priority)

2. Local Restriction (G)

The Local Restriction (G) block models the pressure drop due to a localized reduction in flow area, such as a valve or an orifice, in a gas network. Choking occurs when the restriction reaches the sonic condition. Ports A and B represent the restriction inlet and outlet. The input physical signal at port AR specifies the restriction area. Alternatively, you can specify a fixed restriction area as a block parameter. The block icon changes depending on the value of the Restriction type parameter.

(Local Restriction (G) - MATLAB [16], n.d.)

I used it to model the inner and outer valves.

Important parameters and their values in the Simscape model (others were set to default):

a. Restriction type - specifies whether the restriction is fixed or it changes over time
- this parameter was set to variable for each valve (there is an inlet enabling to set a restriction)

b. Minimum restriction area
- it was set to 1e-10 m^2 for each valve

c. Maximum restriction area
- N_2 fill valve: 38.49 → 7.49 mm^2
- N_2 release valve: 38.49 → 7.49 mm^2
- High servo: 12.58 → 2.01 mm^2
- Low servo: 12.58 → 2.01 mm^2

d. Cross-sectional area at ports
- N_2 fill valve: 38.48 → 7.48 mm^2
- N_2 release valve: 38.48 → 7.48 mm^2
- High servo: 12.57 → 2 mm^2
- Low servo: 12.57 → 2 mm^2

3. Pipe (G)

The Pipe (G) block models pipe flow dynamics in a gas network. The block accounts for viscous friction losses and convective heat transfer with

the pipe wall. The pipe contains a constant volume of gas. The pressure and temperature evolve based on the compressibility and thermal capacity of this gas volume. Choking occurs when the outlet reaches the sonic condition.

(Pipe (G) - MATLAB [18], n.d.)

I used it to model the pipes and the hoses between the GSE and the gas tanks.

Important parameters and their values in the Simscape model (others were set to default):

a. Pipe length
   - the whole GSE's upper line is about 1 meter long
   - the hoses between the GSE and the gas tanks are about 2 metres long

b. Cross-sectional area - specifies a pipe's internal cross-sectional area
   - all the GSE's pipes and the hoses have the same cross-sectional area of $38.48 \rightarrow 7.48$ mm^2
   - all the pipes inside the rocket have the same cross-sectional area of $12.57 \rightarrow 2$ mm^2

c. Hydraulic diameter - specifies a pipe's diameter
   - all the GSE's pipes and the hoses have the same diameter of $7 \rightarrow 3$ mm
   - all the pipes inside the rocket have the same diameter of $4 \rightarrow 1.6$ mm

4. Reservoir (G)

The Reservoir (G) block represents an infinite reservoir at fixed pressure and temperature. The reservoir and its inlet can be at atmospheric pressure or at a specified pressure. Port A, a gas-conserving port, represents the reservoir inlet.

(Reservoir (G) - MATLAB [20], n.d.)

I used it to model the surrounding world.

Important parameters and their values in the Simscape model (others were set to default):

a. Reservoir pressure
   - it was set to the atmospheric pressure level of 101325 Pa

b. Reservoir temperature
- it was set to 20 °C

c. Cross-sectional area at port
- behind N_2 release valve: 38.48 → 7.48 mm^2
- behind Low servo: 12.57 → 2 mm^2

5. Pressure & Temperature Sensor (G)

The Pressure & Temperature Sensor (G) block represents an ideal sensor that measures pressure and temperature in a gas network. There is no mass or energy flow through the sensor.

(Pressure & Temperature Sensor (G) - MATLAB [19], n.d.)

I used it to model the pressure sensors. The pressure is given in Pa.

It measures port A's pressure and temperature relative to port B's quantities.

6. Convective Heat Transfer

The Convective Heat Transfer block represents heat transfer by convection between two bodies by means of fluid motion. The Newton law of cooling describes the transfer, $Q = k \cdot A \cdot (T_A - T_B)$, where Q is the heat flow,k is the convection heat transfer coefficient, A is the surface area, T_A, and T_B are the temperatures of the two bodies. The heat transfer coefficient, k, can be either constant, which you specify by using the Heat transfer coefficient parameter, or variable, which you specify by using the physical signal at port K.

(Convective Heat Transfer - MATLAB [9], n.d.)

I used it to model a heat transfer between the surrounding air and the walls of the pipes and chambers.

Important parameters and their values in the Simscape model (others were set to default):

a. Heat transfer coefficient - specifies the heat transfer coeficient
- it was set to default 20 $W/(K \cdot m^2)$

b. Area - specifies the surface area
- it was set to the surface areas of the pipes, hoses, and chambers, computed as diameter times length (I considered them to be cylindrical, which is reasonable).

7. Temperature Source

The Temperature Source block represents an ideal source of thermal energy that is powerful enough to maintain a specified temperature at its outlet regardless of the heat flow consumed by the system. The source generates constant absolute temperature, defined by the Temperature parameter value.

(Temperature Source - MATLAB [26], n.d.)

I used it to model a reservoir with constant temperature (surrounding air).

Important parameters and their values in the Simscape model (others were set to default):

a. Temperature
   - it was set to default 20 °C

5.1.1 Gas properties (G)

In order to reach sufficient accuracy in the Simscape model, the physical properties of N_2 had to be set. Assuming N_2 is a real gas (not ideal), many properties define its behavior (density, specific entropy, specific enthalpy, specific heat at constant pressure, dynamic viscosity, thermal conductivity, isothermal bulk modulus, isobaric thermal expansion coefficient), and their computation could be quite challenging. That is because they differ relative to pressure and temperature (meaning their value differs for each combination of pressure and temperature). Luckily, Are Mjaavatten wrote two MATLAB functions estimating these properties: `nistdata()` [17] and `thermo()` [27]. These two functions enabled me to write a MATLAB file computing values of appropriate properties [1].

```
%Pressure vector setup
%----------------------

P_vec1 = 1:1:9;
P_vec2 = 10:10:290;
P_vec3 = 300:20:1000;
P_vec4 = 0.1:0.1:0.9;
P = [0.00001,P_vec4, P_vec1, P_vec2, P_vec3];
```

```
%Temperature vector setup
%-------------------------

Temp_vec = 150 : 10 : 2000;
T = Temp_vec;

%Using nistdata() function
%-------------------------

data1 = nistdata("N2", T, P);

%Getting appropriate properties
%------------------------------

                          %     Output
                          %     data   : Struct with the following fields:
                          %       Single values:
species = data1.species;  %         species : Chemical symbol (e.g. 'H2')
Tc = data1.Tc;            %         Tc      : Critical temperature (K)
Pc = data1.pc;            %         Pc      : Critical pressure (Pa)
Mw = data1.Mw;            %         Mw      : Molar mass (kg/kmol)
                          %       Arrays:
Rho = data1.Rho;          %         Rho     : Density (kmol/m3)
V = data1.V;              %         V       : Volume (m3/kmol)
U = data1.U;              %         U       : Internal energy (J/kmol)
H = data1.H;              %         H       : Enthalpy (J/kmol)
S = data1.S;              %         S       : Entropy (J/kmol/K)
Cv = data1.Cv;            %         Cv      : Heat capacity at constant volume (J/kmol/k)
Cp = data1.Cp;            %         Cp      : Heat capacity at constant pressure (J/kmol/k)
C = data1.C;              %         C       : Speed of sound (m/s)
JT = data1.JT;            %         JT      : Joule-Thompson coefficient (K/Pa)
mu = data1.mu;            %         mu      : Dynamic viscosity (Pa s)
k = data1.k;              %         k       : Thermal conductivity (W/m/K)

%Conversion to better units
%--------------------------

Rho = Rho*Mw;             % conversion to kg/m^3
Molar_V = V/1000;         % conversion to m^3/mol
Molar_U = U/1000;         % conversion to J/mol
H = H/(1000*Mw);          % conversion to kJ/kg
S = S/(1000*Mw);          % conversion to kJ/(kg*K)
Cp = Cp/(1000*Mw);        % conversion to kJ/(kg*K)
mu = mu*(1e6);            % conversion to uPa s
k = k*1000;               % conversion to mW/(mK)

%Getting bulk modulus and isobaric thermal expansion coefficient values
%----------------------------------------------------------------------

bulk_modulus = zeros(length(T), length(P));
isobaric_thermal_expansion_coeff = zeros(length(T), length(P));

th = thermo('N2');  % Initialise thermo object
for i = 1:length(T)
    for j = 1:length(P)
        th.Tpcalc(T(i),P(j)*(1e5)); %Calculate thermodynamic state at T, p
        bulk_modulus(i,j) = -th.v*th.p_v;
        isobaric_thermal_expansion_coeff(i,j) = -th.f_Tv/th.f_vv/th.v;
    end
end
```

Listing 1: Gas properties generation

5.1.2 Simscape model's structure and its validation

This subsection depicts the overall structure of the Simscape model and compares its outputs with the outputs of the real system. The overall structure shows the picture [5.1], the pictures [5.2a] and [5.2b] are used for validation of the GSE's upper line, and the pictures [5.3a] and [5.3b] for validation of pressure transfer between the inner rocket tanks.

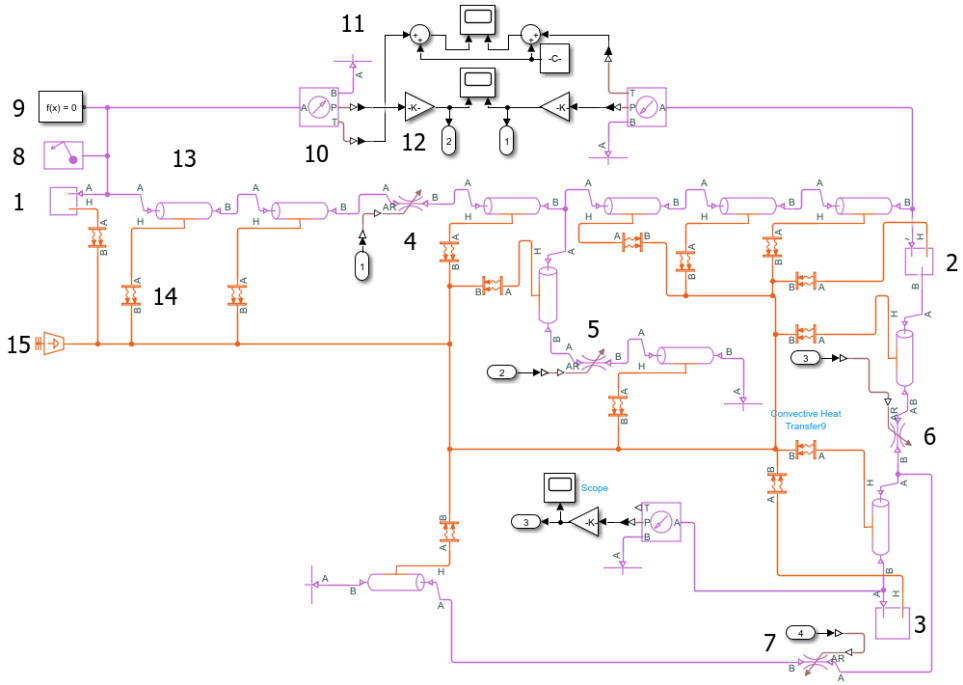
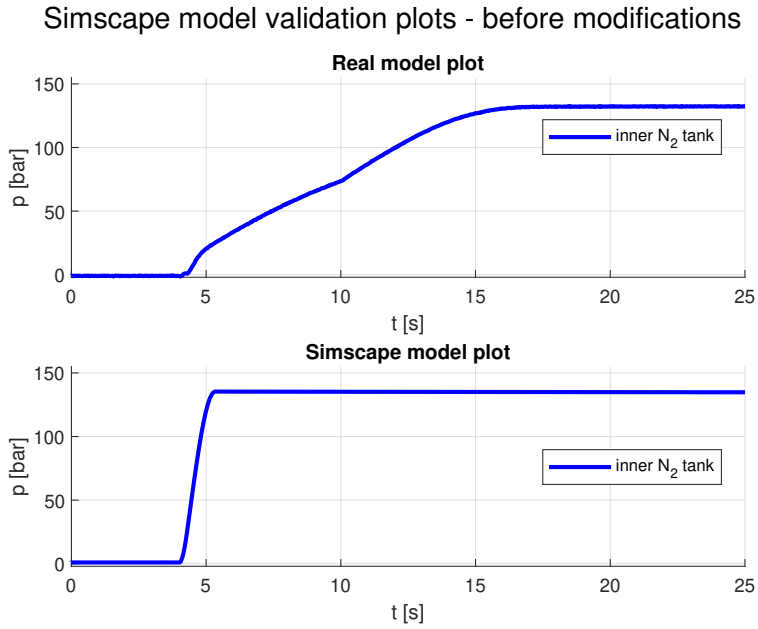
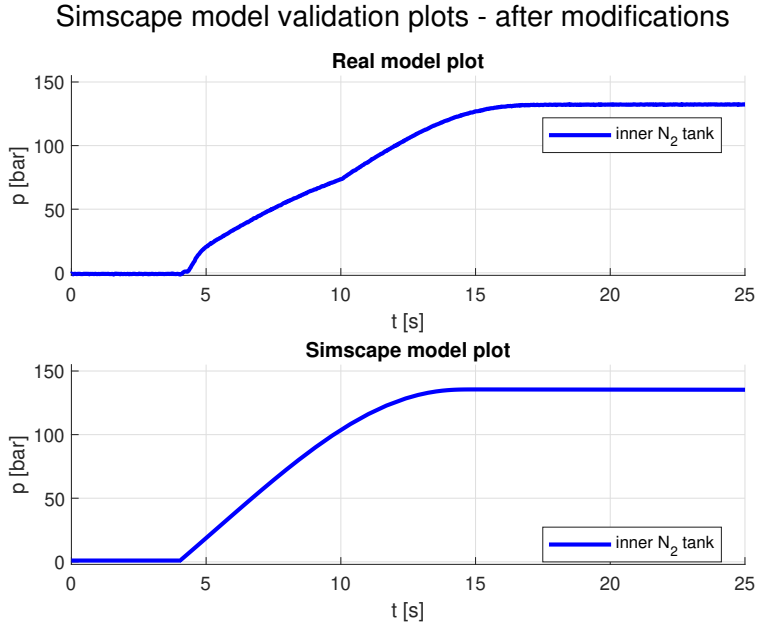


Figure 5.1: Structure of the Simscape model: 1) Outer N_2 tank [Constant Volume Chamber (G)], 2) Inner N_2 tank [Constant Volume Chamber (G)], 3) Inner oxidiser tank [Constant Volume Chamber (G)], 4) N_2 fill valve [Local restriction (G)], 5) N_2 release valve [Local restriction (G)], 6) High servo [Local restriction (G)], 7) Low servo [Local restriction (G)], 8) Gas Properties (G), 9) Solver Configuration, 10) Pressure & Temperature Sensor (G), 11) Surrounding world [Reservoir (G)], 12) Pressure conversion to bars, 13) Pipe (G), 14) Convective Heat Transfer, 15) Surrounding air temperature [Temperature Source]

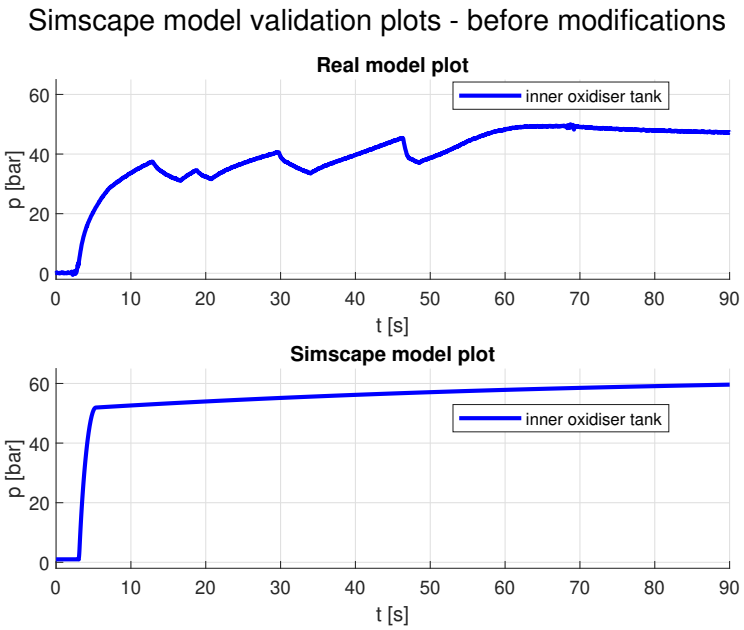


(a) : Cross-sectional area of the GSE's pipes = 38.48 mm^2

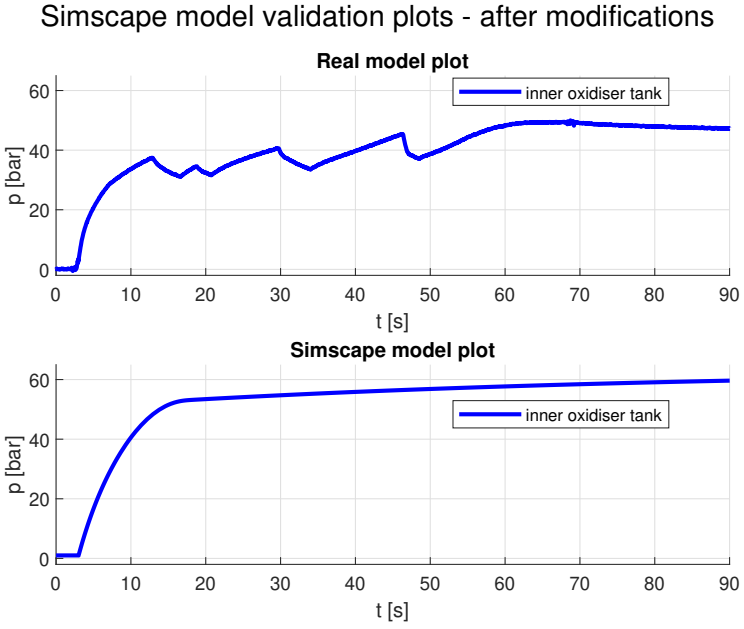


(b) : Cross-sectional area of the GSE's pipes = 7.48 mm^2

Figure 5.2: The GSE's **upper** line validation: initial pressure in the outer N_2 tank is 146 bars, initial pressure in the inner N_2 tank is 1 bar, N_2 release valve close, N_2 fill valve open at 4 seconds



(a) : Cross-sectional area of the rocket's pipes = 12.57 mm^2



(b) : Cross-sectional area of the rocket's pipes = 2 mm^2

Figure 5.3: Validation of the pressure transfer between the rocket's tanks: initial pressure in the inner N_2 tank is 216 bars, initial pressure in the inner oxidiser tank is 1 bar, Low servo close, High servo open at 3 seconds

By looking at [5.2a], the Simscape model was obviously inaccurate. Although there did not have to be a perfect fit between the Simscape model and the real one, it still should have been reasonably similar. There was one significant difference. While the time constant of the real system was approximately seven seconds, the Simscape model's time constant was less than one second. That difference was unacceptable; therefore, slight modifications needed to be made. I decided to modify the inner cross-sectional area of the GSE's pipes to 7.48 mm^2, and I got sufficient accuracy by doing so [5.2b]. Although the cross-sectional area of the GSE's pipes in the currently used Simscape model does not correspond with reality, having a similar time constant has higher priority.

Related to the pressure transfer between the rocket's tanks, similarly to [5.2], before I reduced the cross-sectional area of the pipes inside the rocket to 2 mm^2, the time constant of the Simscape model was significantly smaller than the one of the real model. After this reduction, both time constants resembled each other. Aside from this, there was one other more considerable difference. The response of the real model fluctuated a little bit, but the response of the Simscape model was utterly smooth. Computational errors could cause it, so I ignored it.

After those modifications, the Simscape model became similar to the real one. It is not entirely the same, but it is sufficient for testing the refueling system.

5.2 Valve dynamics approximation

The GSE's valves and the servos inside the rocket have their dynamics, meaning they do not reach their goal state instantaneously after the control signal arrives. The dynamics had to be included to make the VT more accurate. Given that the flight computer does not provide feedback about the state of the servos inside the rocket, I approximated their dynamics to be the same as the dynamics of the GSE's valves, assuming that all the GSE's valves are the same, and so are their dynamics (there could be some minor differences, but those are neglectable). After the control signal arrives, the valves change their state linearly at the same rate until they reach the goal state. Therefore, the rate limiter in Simulink was used to model their dynamics. The picture [5.4] shows the complete structure of the Simulink valve simulation model with information about block settings. The Rate Limiter ensures a gradual change of state, where the goal state is reached after 0.7 seconds. The Saturation block limits the signal to stay between 0 and 1. The output signal can then be scaled, but I will get to it later.

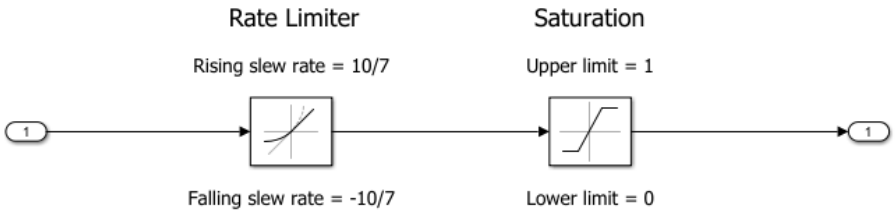


Figure 5.4: Simulink valve model

Figure [5.5] shows the validation of the valve model. The data obtained from the simulation are scaled by 255 to obtain values between 0 and 255 corresponding to the UnIO IO card messages [4.2].

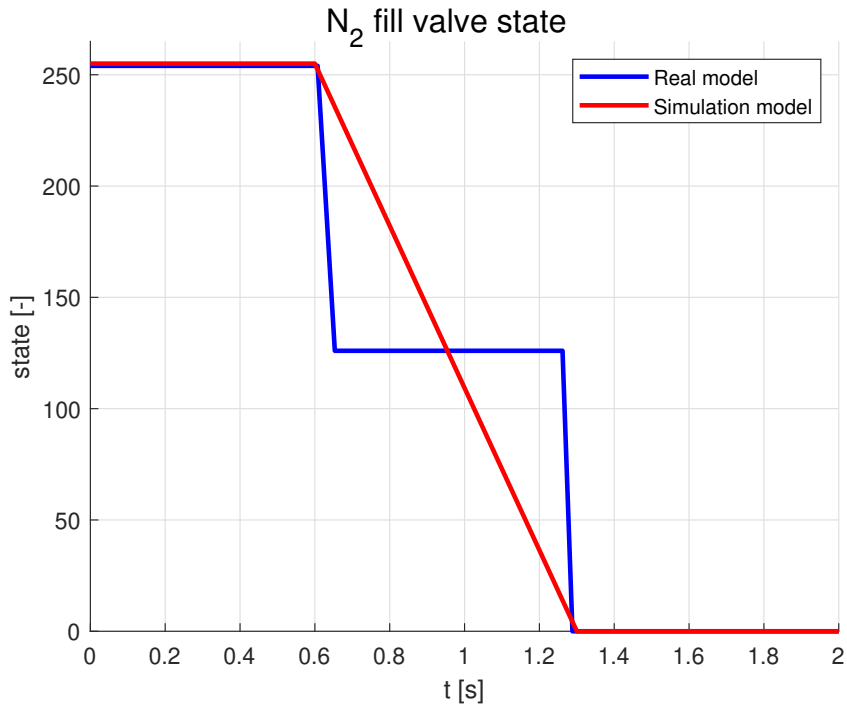


Figure 5.5: N_2 fill valve model validation - opening time 0.6 seconds

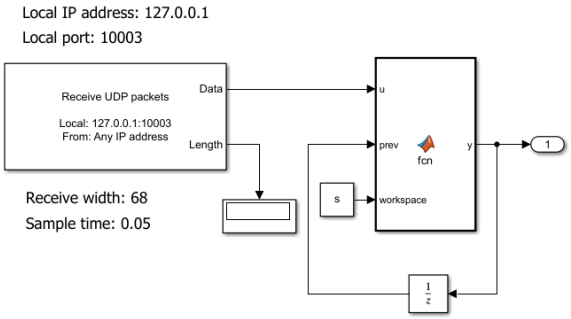
The simulation data do not exactly match the real model data, but this was intended. While the state of the valve model changes linearly, the real model plot is not linear at all. This is due to the discrete nature of the UnIO IO card messages. Limit switches indicate the position of a valve, so the UnIO sends 255 if the valve is closed, 0 if the valve is open, or 127 if none of the limit switches is on. Therefore, the valve simulation model better reflects the actual valve behavior. Furthermore, computational errors appeared during UnIO's data processing, causing 254 to be sent instead of 255, which is just a detail.

5.3 VT's communication interface

Similarly to the real model, the VT reacts to UDP messages sent from a control computer (or other program) and sends UDP packets back containing information about valve states.

5.3.1 IOcard's receiving side simulation

In the real model, two IO cards process control computer messages, each capable of processing 20 messages per second. The IOcard1 processes messages controlling the GSE's fill valves and servos inside the rocket, and the IOcard2 processes messages controlling the GSE's release valves. Two independent UDP receive blocks, each followed by MATLAB function block processing received UDP packets, simulate these two cards in the VT.




```
function y = fcn(u, prev, workspace)

y = zeros(1,4);
y = prev;
temp = u.';

if temp == workspace.N2O_fill_valve_open
    y(1) = 1;
elseif temp == workspace.N2O_fill_valve_close
    y(1) = 0;
end

if temp == workspace.Low_servo_open
    y(2) = 1;
elseif temp == workspace.Low_servo_close
    y(2) = 0;
end

if temp == workspace.High_servo_open
    y(3) = 1;
elseif temp == workspace.High_servo_close
    y(3) = 0;
end

if temp == workspace.N2_fill_valve_open
    y(4) = 1;
elseif temp == workspace.N2_fill_valve_close
    y(4) = 0;
end
```

Figure 5.6: IOcard2 simulation - receiving side

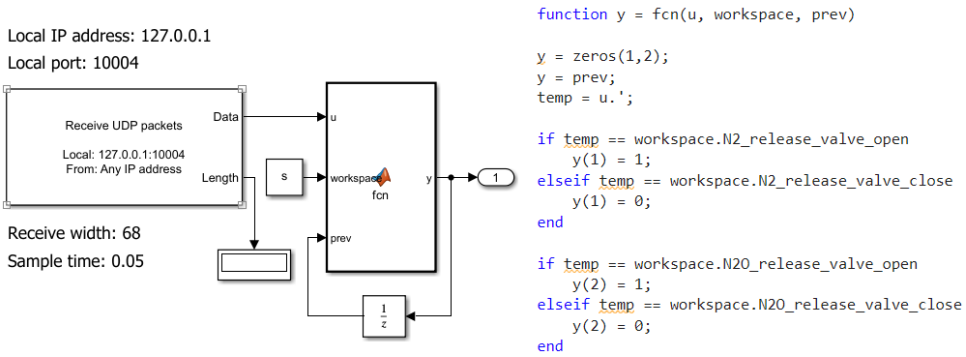


Figure 5.7: IOcard3 simulation - receiving side

The picture [5.6] shows the structure of the IOcard2's receiving side simulation model aligned with code in the MATLAB Function block. The picture [5.7] shows the same things but for the IOcard3. Both UDP blocks have the IP address parameter set to the loopback address 127.0.0.1 to achieve communication between two programs, such as two .slx files, on the same computer. The IOcard2's simulation model communicates on port 10003, and the IOcard3's simulation model communicates on port 10004. The MATLAB function block compares inputted data with predefined control messages [4.2.3] and outputs a vector containing valve goal states of the corresponding valves (1 - valve open, 0 - valve close).

5.3.2 IOcard's and CurrentLoop's sending side simulation

Three UDP Send blocks send data from sensors and information about current valve states in the VT. These blocks, aligned with one MATLAB Function block, simulate the sending side of the CurrentLoop, IOcard2, and IOcard3. The picture [5.8] shows the whole structure.

Similarly to [5.3.1], all three UDP blocks have the IP address parameter set to the loopback address 127.0.0.2 to achieve communication between two programs on the same computer, and they send data to port 10005. Because the CurrentLoop card is capable of sending 100 messages per second, the corresponding UDP block has the sample time set to 0.01. The IOcard2 and IOcard3 can send 20 messages per second; therefore, the corresponding UDP blocks have the sample time parameter set to 0.05.

```
function [Curr_loop, IO_2, IO_3]= fcn(u, workspace)

Curr_loop = uint8(zeros(1,69));
IO_2 = uint8(zeros(1,69));
IO_3 = uint8(zeros(1,69));

%Pressure sensor data conversion
press_sensor100bar_conv = @(x) (x - workspace.PT5302_OFFSET)/workspace.PT5302_GAIN;
press_sensor300bar_conv = @(x) (x - workspace.PT5500_OFFSET)/workspace.PT5500_GAIN;

%CurrentLoop message build
Curr_loop(1) = workspace.ID_RECEIVE;
Curr_loop(2) = workspace.PL_RECEIVE;
Curr_loop(3) = workspace.MSG_ID_RECEIVE;
Curr_loop(4) = workspace.CARDID_CURRENT_LOOP;
Curr_loop(5) = workspace.CARD_CURRENT_LOOP_SLOT;
Curr_loop_message = zeros(1,16);
Curr_loop_message(workspace.Low_p_ind) = press_sensor100bar_conv(u(3));
Curr_loop_message(workspace.High_p_ind) = press_sensor300bar_conv(u(1));
Curr_loop_message(workspace.GSE_Pressurizer_p_ind) = press_sensor300bar_conv(u(2));
Curr_loop_message(workspace.GSE_oxidiser_p_ind) = press_sensor100bar_conv(60);
Curr_loop_message(workspace.N2O_fuel_level_ind) = u(4);
Curr_loop(6:end) = typecast(uint32(Curr_loop_message), 'uint8');

...
```

Listing 2: The snippet of MATLAB function block's source code

The MATLAB function block builds messages in the predefined format [4.2.3]. It takes information about valve states and data from the Simscape model's sensors, forms messages, and distributes them to the corresponding UDP Send blocks. A snippet of the MATLAB function's source code is shown in [2]. This particular part of the source code builds CurrentLoop messages. Aside from that, two other similar parts build IOcard2's and IOcard3's messages. The whole code is available in [5]. Current loop sensors usually output values that must be scaled first to get a measured quantity in a required unit. The table [5.1] contains scaling constants for the pressure sensors used by the real model. An output value must be multiplied by the gain constant and then summed up with the offset constant to get the pressure in bars. Because the pressure values outputted from the Simscape model are already in bars, two anonymous functions (two types of pressure sensors) mediate back-calculation from bars to raw outputted value.

Sensor	Gain	Offset
IFM PT5302	0.05	-25
IFM PT5500	0.2	-100

Table 5.1: Scaling coefficients - pressure sensors

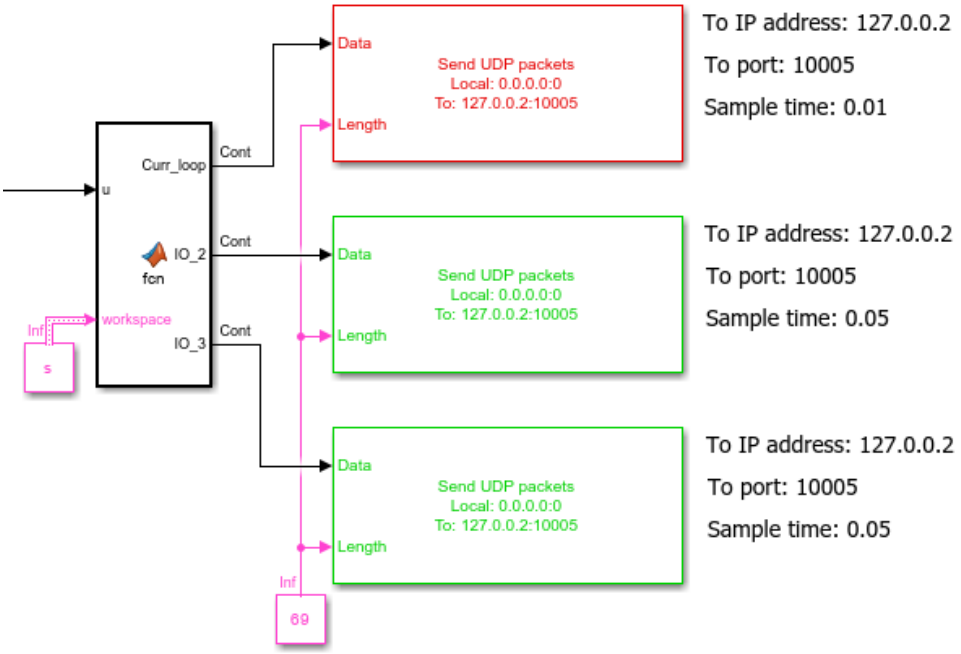


Figure 5.8: IOcard2's, IOcard3's, CurrentLoop's simulation - sending side

5.4 Overall structure of the VT

In this final section, the overall structure of the VT will be described. The picture [5.9] shows the whole VT.

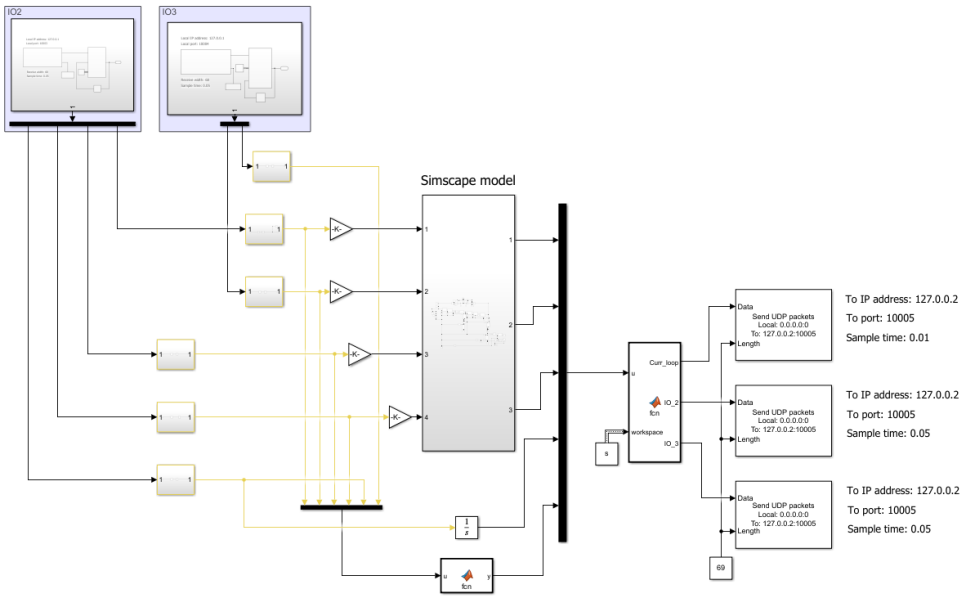


Figure 5.9: VT's scheme

The two blocks in the upper left corner are the subsystems simulating the receiving side of the IOcards. Between them and the Simscape model, six subsystems simulating valve dynamics are placed. Their outputs are then multiplied by $7.48 \cdot 10^{-6}$ or $2 \cdot 10^{-6}$ for the GSE's valves and the servos inside the rocket, respectively. That is because the Local Restriction (G) block takes the cross-sectional area (in m^2) of the restriction as its input. When a valve is fully open, I consider relating restriction having the same cross-sectional area as the corresponding pipe. Therefore, the valve dynamics subsystem's output is multiplied by the corresponding pipe's cross-sectional area.

The MATLAB Function block below the Simscape model subsystem transforms the continuous signal between 0 and 1 into discrete information about valve states (GSE: 0 - OPEN, 255 - CLOSE, 127 - MIDDLE, SERVO: 90 - OPEN, 0 - CLOSE, 45 - MIDDLE). A snippet of its source code shows [3].

```
function y = fcn(u)

y = zeros(1,6);

%N2 fill valve
if u(1) > 0.9
    y(1) = 0;
elseif u(1) < 0.1
    y(1) = 255;
else
    y(1) = 127;
end

%High servo
if u(3) > 0.9
    y(3) = 90;
elseif u(3) < 0.1
    y(3) = 0;
else
    y(3) = 45;
end

...
```

Listing 3: The snippet of valve states estimation

Chapter 6

Refueling system design

As was said in the introduction, the validated VT was used to develop the refueling system. This chapter will briefly uncover the refueling model structure, including communication interfaces, and emphasize the control Stateflow chart and designed P and simple controllers. The communication interfaces will not be described in detail as they work similarly to those of the VT.

6.1 OMA state machine using Stateflow

The abbreviation OMA stands for OFF/MAN/AUTO. One might have to control the refueling process manually (for example, during testing when automatic mode is not working correctly). Therefore, a simple OMA state machine was designed using the Stateflow Chart block. The Stateflow Chart block should be introduced briefly before describing the OMA state machine.

For state machine design, Simulink includes the sub-library called Stateflow, containing the Chart block. According to [7],

The Chart block is a graphical representation of a finite state machine based on a state transition diagram. In a Stateflow chart, states and transitions form the basic building blocks of a sequential logic system. States correspond to operating modes, and transitions represent pathways between states.

(Chart - MATLAB [7], n.d.)

The working principles of the Stateflow Chart block can be the most easily described on the OMA state machine itself [6.1]. There are three rectangles called OFF, MAN, and AUTO, called states. A state can be defined as a set of properties and instructions specifying the current behavior of the controlled system, and the current state is called active. The arrows between these rectangles are called transitions. Each arrow starts at an initial state and ends at a goal one. A transition has its corresponding condition, which must be fulfilled to change the active state from the initial one to the goal state. The transition condition is always enclosed in square parentheses in the Stateflow Chart block. There is also an option to add a transition action, enclosed in curly parentheses, which is performed during the transition. Besides transition actions, three types of state actions: entry, during, and exit, can be performed. Entry actions are executed just once when entering the state. Similarly, exit actions are performed just once when exiting the state. During actions are performed each time step while the state is active. Finally, any Stateflow Chart can have symbols defined. These symbols are of many types: inputs, outputs, local data, etc., and are listed in the symbols pane. When a symbol is of type input, its value is set from outside the Stateflow Chart block through an associated input port. Similarly, if a symbol is of type output, its value is outputed from the Stateflow Chart through an associated output port. Local data symbol's value can be set and read only from inside the Stateflow Chart block.

The OMA Stateflow Chart works as follows. The CNTRL input symbol controls the transitions between states. Its values 0, 1, and 2 trigger transitions to the OFF, MAN, and AUTO states, respectively. The OFF and MAN states are reachable from any other state, but to reach the AUTO state, the MAN state must be active. The corresponding output symbol is set to 1 by entering a state and the others to zero; these symbols govern the external system.

Two transition conditions may be fulfilled at the same time; therefore, if a state is initial for more than one transition, state priority has to be stated. A small number placed next to a transition indicates transition priority in the Stateflow Chart block; a smaller number means higher priority.

In the end, the `after()` function in square parentheses should be explained. This function schedules transitions between corresponding states after the specified time period has elapsed.

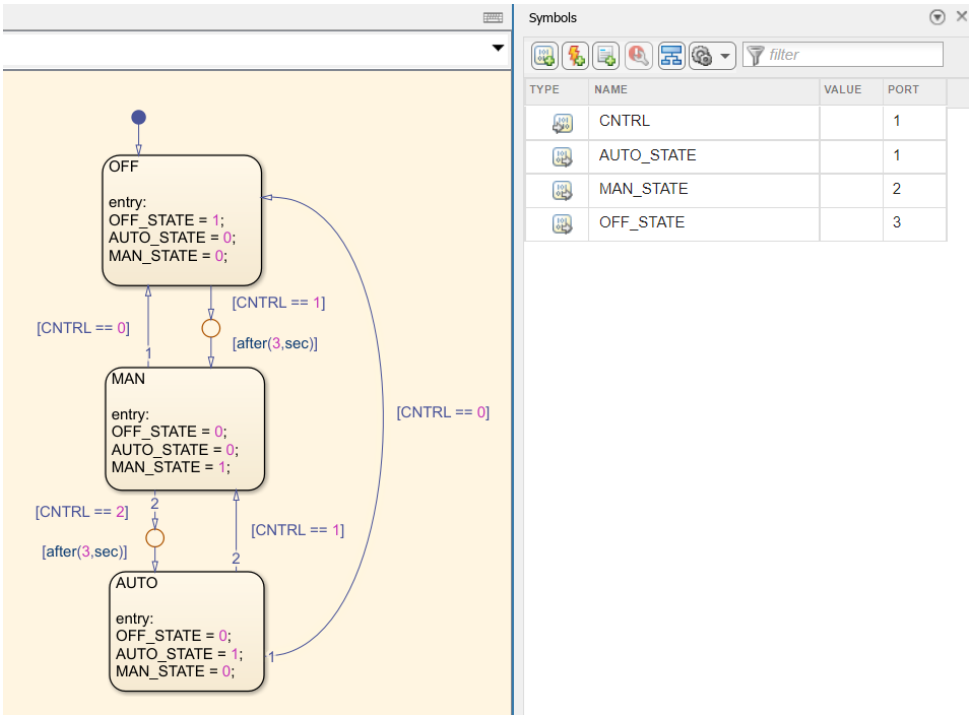


Figure 6.1: OMA - Stateflow Chart

6.2 Communication interfaces

This section will briefly examine the communication interfaces of the refueling system.

6.2.1 UDP packet reception

Similarly to [5.3.1], the receiving part is in its core build of two blocks: UDP Receive and MATLAB Function. The UDP Receive block receives raw packets, which are then decoded in the MATLAB Function block to get information about valve states and pressure values. The overall layout with the UDP Receive block parameters is displayed in the picture [6.2]. In order to beware of data stacking, the sample time parameter is set to 0.0025. This number comes from the CurrentLoop's and IOcard's sample times. The CurrentLoop sends one packet per 0.01 second, and the IOcard one per 0.05 second. I considered that all incoming packets must be processed before the next CurrenLoop packet. The critical situation happens when all three packets come at the same time. Therefore, the sample time must be at a maximum of 0.01/3 ($0.00\overline{3}$) seconds long.

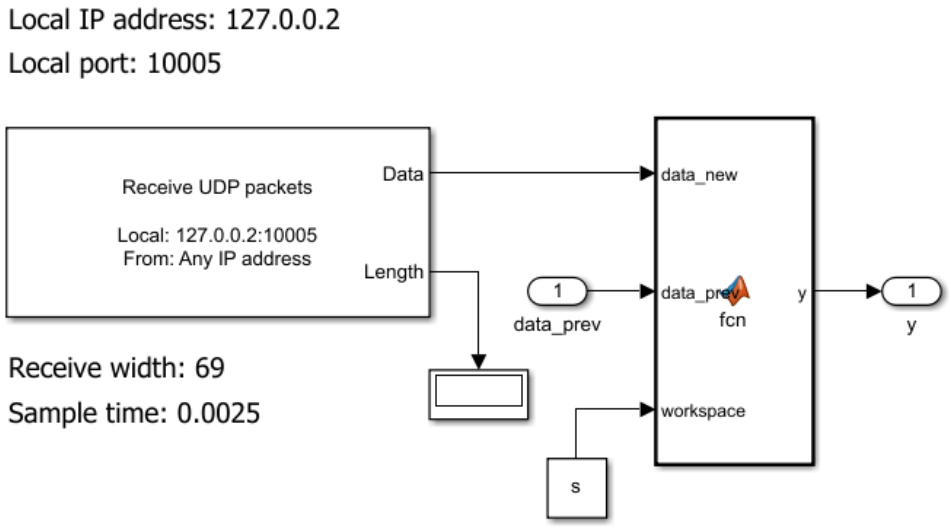


Figure 6.2: UDP packet reception - core subsystem

A snippet of the MATLAB Function block's source code is displayed in [4]. It is commented on for clarity. The purpose of the code is to extract information about valve states and pressure values from the inputted messages. However, a further explanation of `set_param( )` function usage would be appropriate. These functions serve for the H-bridge fault state indication. Whenever a fault state is detected, the `set_param()` sets the corresponding constant in the [6.3] (higher-level subsystem) to zero, triggering OFF mode (all valves close) until the H-bridge gets out of the fault state.

The higher-level subsystem provides further message processing. It takes extracted data (actual pressure values and valve states) and checks if everything is all right for the continuation of the refueling process. Because this thesis mainly focuses on the refueling process controllers, these security features will only be briefly described. The MATLAB Function block in the orange frame checks whether any H-bridge is in a fault state and alternatively takes an action. Furthermore, the two MATLAB Function blocks left to the orange frame monitor sudden pressure drops in the inner rocket tanks and trigger the OFF state if needed. Additionally, signals from the higher-level subsystem are bonded to control panel (will be depicted later) widgets to monitor all essential quantities simultaneously.

```
function y = fcn(data_new, data_prev, workspace)

%Enables the use of set_param function in a MATLAB Function block
coder.extrinsic('set_param')

%Anonymous functions for pressure values conversion
press_sensor100bar_conv = @(x) workspace.PT5302_GAIN*double(x) + workspace.PT5302_OFFSET;
press_sensor300bar_conv = @(x) workspace.PT5500_GAIN*double(x) + workspace.PT5500_OFFSET;

%Anonymous function for the output format
output_format = @(Fill_N2_state, ...
                  Release_N2_state, ...
                  Fill_N2O_state, ...
                  Release_N2O_state, ...
                  Servo_0_state, ...
                  Servo_1_state, ...
                  N2_tank_pressure, ...
                  N2_bomb_pressure, ...
                  N2O_tank_pressure, ...
                  N2O_bomb_pressure, ...
                  N2O_fuel_level_sensor) [Fill_N2_state, Release_N2_state, Fill_N2O_state, ...
                                          Release_N2O_state, Servo_0_state, Servo_1_state, ...
                                          N2_tank_pressure, N2_bomb_pressure, N2O_tank_pressure, ...
                                          N2O_bomb_pressure, N2O_fuel_level_sensor];

%Output predefinition
y = output_format(0,0,0,0,0,0,0,0,0,0,0);

%Input message splitting
data_n = typecast(data_new,'uint8');
id = data_n(1);
pl = data_n(2);
msg_id = data_n(3);
card_id = data_n(4);
slot = data_n(5);
message = data_n(6:end);

switch card_id
    case workspace.CARDID_CURRENT_LOOP
        %Division into 16 4-byte channels
        message_CUL = typecast(message, 'uint32');

        %Important pressure values extraction
        low_p = message_CUL(workspace.Low_p_ind);                           %N2O tank inside rocket
        high_p = message_CUL(workspace.High_p_ind);                         %N2 tank inside rocket
        GSE_oxidiser_p = message_CUL(workspace.GSE_oxidiser_p_ind);         %N2O tank outside rocket
        GSE_Pressurizer_p = message_CUL(workspace.GSE_Pressurizer_p_ind);   %N2 tank outside rocket
        N2O_fuel_level = double(message_CUL(workspace.N2O_fuel_level_ind));

        %Pressure values conversion
        low_p_bar = press_sensor100bar_conv(low_p);
        high_p_bar = press_sensor300bar_conv(high_p);
        GSE_oxidiser_bar = press_sensor300bar_conv(GSE_oxidiser_p);
        GSE_Pressurizer_bar = press_sensor300bar_conv(GSE_Pressurizer_p);

        y = output_format(data_prev(1), data_prev(2), data_prev(3), data_prev(4), ...
                          data_prev(5), data_prev(6), high_p_bar, GSE_Pressurizer_bar, ...
                          low_p_bar, GSE_oxidiser_bar, N2O_fuel_level);
    case workspace.CARDID_IO
        switch slot
            case workspace.CARD_IO1_SLOT
                ...

            case workspace.CARD_IO2_SLOT
                %Just to be sure
                message_CIO2 = typecast(message, 'uint8');

                %Fault state check
                if message_CIO2(workspace.H0_bridge_fault_state_ind) == workspace.H_bridge_fault_state_num
                    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant6', 'Value', '0')
                else
                    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant6', 'Value', '1')
                end

                if message_CIO2(workspace.H1_bridge_fault_state_ind) == workspace.H_bridge_fault_state_num
                    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant8', 'Value', '0')
                else
                    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant8', 'Value', '1')
                end

                if message_CIO2(workspace.H2_bridge_fault_state_ind) == workspace.H_bridge_fault_state_num
                    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant9', 'Value', '0')
                else
                    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant9', 'Value', '1')
                end
```

```
%Getting valve states
Fill_bridge_state_N2 = double(message_CIO2(workspace.H1_bridge_state_ind));
Fill_bridge_state_N2O = double(message_CIO2(workspace.H2_bridge_state_ind));
Servo_0_state = double(message_CIO2(workspace.Servo_0_state_ind));
Servo_1_state = double(message_CIO2(workspace.Servo_1_state_ind));

if Servo_0_state >= 10
    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant2', 'Value', '1')
else
    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant2', 'Value', '0')
end

if Servo_1_state >= 10
    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant3', 'Value', '1')
else
    set_param('StateFlow_varB/Subsystem6/Subsystem3/Constant3', 'Value', '0')
end

y = output_format(Fill_bridge_state_N2, data_prev(2), Fill_bridge_state_N2O, data_prev(4), ...
                  Servo_0_state, Servo_1_state, data_prev(7), data_prev(8), ...
                  data_prev(9), data_prev(10), data_prev(11));

...
```

Listing 4: The snippet of the MATLAB Function block - packet reception

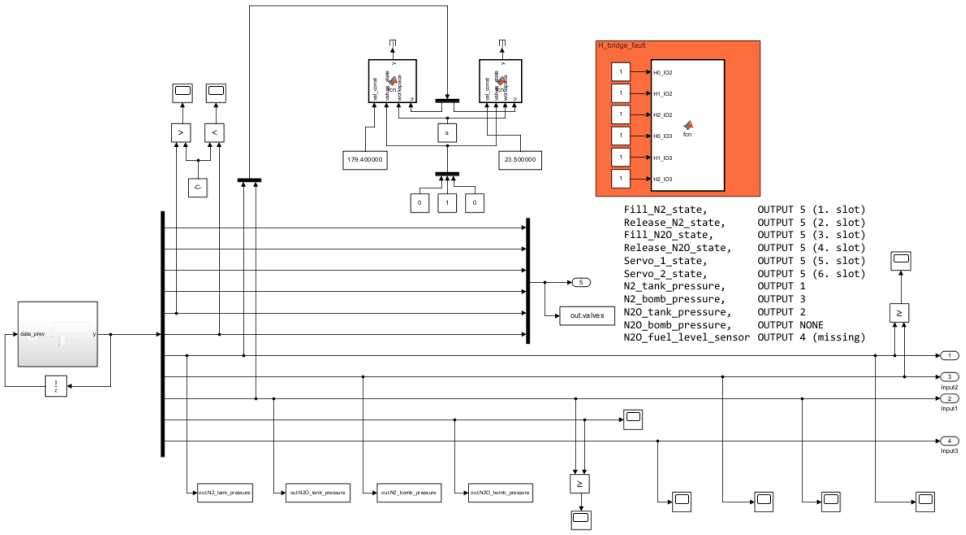


Figure 6.3: UDP packet reception - higher-level subsystem

6.2.2 UDP packet transmission

To give a complete description of the transmitting communication interface is fairly complicated. That is because it varies depending on the OMA mode and the phase of the refueling process (if the AUTO mode is active). For each situation, an individual subsystem performs packet transmission. However, the concept of all these subsystems is relatively the same. It allows sending packets controlling only valves needed in the present situation. For example, when the OFF OMA state is active, each valve must closed. Therefore, transmitting packets controlling each valve has to be enabled. However, when the AUTO OMA mode is active, and pressurization of the inner N_2 tank is taking place, only the N_2 fill valve and N_2 release valve need to be controlled. Because

of that, transmitting packets controlling any other valve is unnecessary. I will use these two situations to explain the concept as minor differences remain.

The picture [6.4] shows the structure of the transmission subsystem related to the OFF OMA state. It could be divided into two independent parts, each ending in an UDP Send block. The upper one serves for packet transmission of packets controlling valves connected to IOcard2. The lower one does the same thing, but for valves connected to IOcard3. Each part starts with multiple MATLAB Function blocks. These blocks output completely formed packets based on corresponding inputs. There is one MATLAB function block for each controlled valve (the one for the N_2 fill valve has its source code displayed in [5]). Output packets are then merged and sent through the delay block. The delay block prevents sending more packets per second than the UnIO's IOcard can process, preventing congestion. Its sample time is derived from the number of controlled valves. Therefore, it equals 0.2 (IOcard's sample time multiplied by the number of controlled valves: $0.05 \cdot 4$) in case of the upper part and 0.1 ($0.05 \cdot 2$) for the lower part. Another MATLAB Function block stands between the delay block and the corresponding UDP Send block. It takes merged packets and outputs them individually; one packet every 0.05 seconds. Its source code for the upper part is displayed in [6].

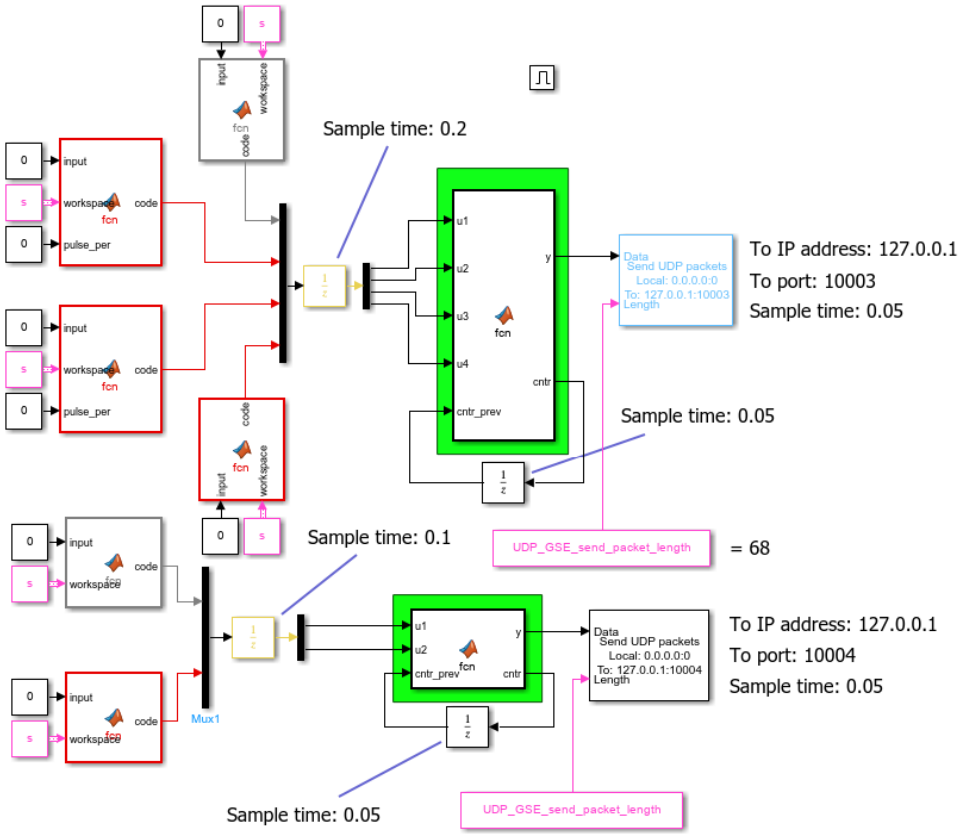


Figure 6.4: OMA-OFF transmission subsystem

```
function code = fcn(input, workspace)

if input == 1
    code = workspace.N2_fill_valve_open;
else
    code = workspace.N2_fill_valve_close;
end

end
```

Listing 5: The source code of the MATLAB Function block transmitting packets controlling the N_2 fill valve

```
function [y, cntr] = fcn(u1, u2, u3, u4, cntr_prev)

y = uint8(zeros(1,68));

switch cntr_prev
    case 1
        y = u1;
    case 2
        y = u2;
    case 3
        y = u3;
    case 4
        y = u4;
end

if cntr_prev < 4
    cntr = cntr_prev + 1;
else
    cntr = 1;
end

...
```

Listing 6: The source code of the upper part's green framed MATLAB Function block

During the pressurization of the inner N_2 tank, only two valves need to be controlled. These valves are connected to different IOcard cards, allowing packets controlling each valve to be sent every 0.05 seconds. The two MATLAB Function blocks work the same as the initial MATLAB Function blocks in [6.4], so the source code of the upper MATLAB Function block is displayed in [5].

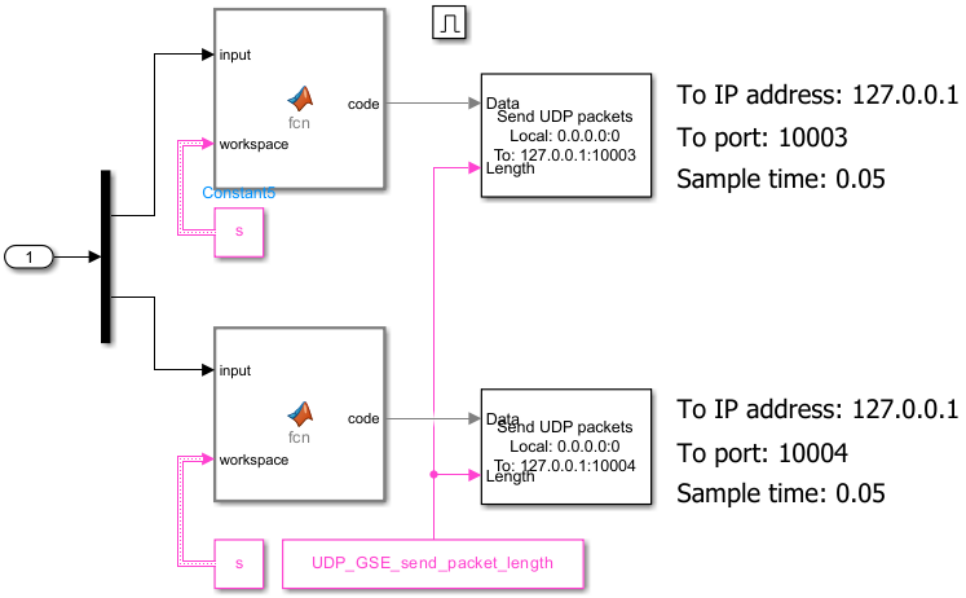


Figure 6.5: OMA-AUTO inner N_2 tank pressurization transmission subsystem

Which packet should be outputted from a MATLAB Function block (valve open or closed) depends on its input. In [6.4], the input was given by the value of the corresponding constant block. However, in [6.5], inputs come from outside into the whole subsystem. That is because they change frequently as they are products of a related controller.

6.3 Automatic refueling

Finally, this section describes the subsystem devoted to the automatic refueling procedure. It consists of a Stateflow Chart block that manages the switching between individual phases of the refueling procedure and particular controllers controlling each phase. There are seven fundamental stages of the refueling procedure:

1. The initial pressurization of the inner N_2 tank (Pressure_section_pressurization)
   - automatically pressurized to 150 bars

2. The pressure transfer between the inner rocket's tanks (Pressure_venting)
   - software controller was designed for the purpose of SIL
   - desired pressure in the inner oxidiser tank after this phase is 40 bars

3. The preparation of the inner oxidiser tank's refueling (oxidiser_fueling_pre_phase)
   - some valves are closed, and some are opened

4. The refueling of the inner oxidiser tank (oxidiser_fueling_pre_phase)
   - an oxidiser-level sensor is not currently available, so this phase is controlled manually

5. The completion of the inner oxidiser tank's refueling (oxidiser_fueling post_phase)
   - some valves are closed, and some are opened

6. The final pressurization of the inner N_2 tank (Pressure_section_post_pressurization1)
   - automatically pressurized to 180 bars

7. The terminal phase (END_phase)
   - each valve is closed

One individual state in the Stateflow Chart block is devoted to each phase, and its name is written in parentheses above. Because the whole Stateflow Chart is too large to display, each phase will be described separately, including corresponding controllers. However, the whole procedure is depicted by [6.6].

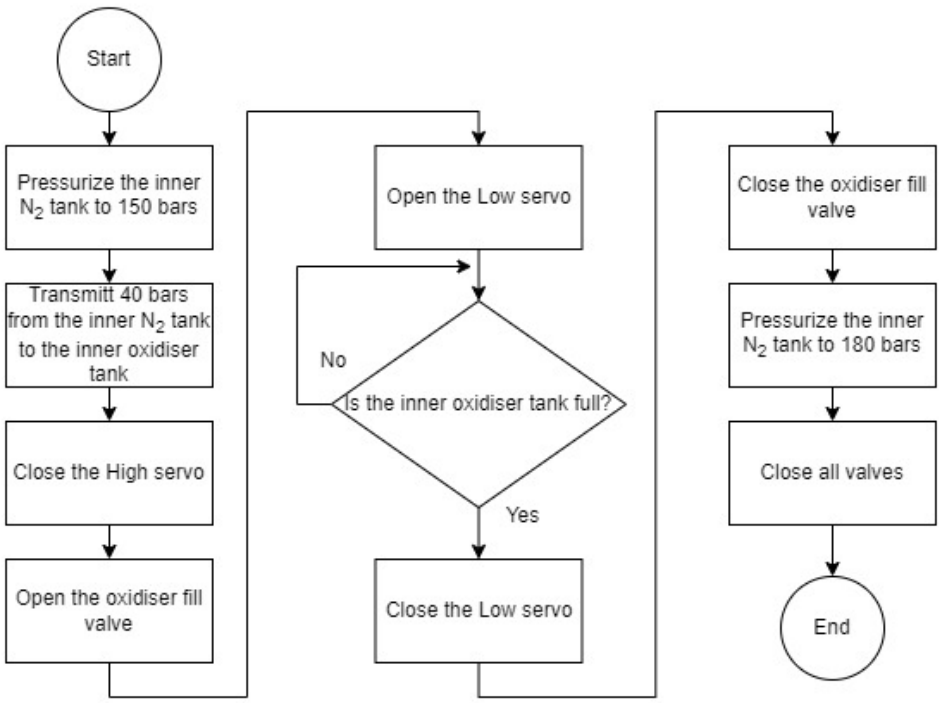


Figure 6.6: Refueling procedure

6.3.1 The initial pressurization of the inner N_2 tank

This phase involves pressurizing the inner N_2 tank to 150 bars, which is the initial phase of the refueling procedure. The corresponding state in the Statflow Chart block is shown in the picture [6.7]. Each symbol should be explained to get a complete understanding.

1. Pressure_section_pressurization (symbol type: output)
   - its value determines whether a controller for the inner N_2 tank's pressurization is switched on or off (1 - ON, 0 - OFF)

2. Pressure_section_cntr (symbol type: local data)
   - its value determines the number of critical periods during which the pressure level in the inner N_2 tank remains within an acceptable deviation from 150 bars

3. Pressure_section_pressure (symbol type: input)
   - it equals the actual pressure level in the inner N_2 tank

4. Pressure_section_pressure_des (symbol type: constant data)
   - desired pressure level after the pressurization = 150 bars

5. Acc_Press_Offset (symbol type: constant data)
   - the acceptable deviation from the desired pressure level ($\pm$ 5 bars)

6. Press_Transition_time_const (symbol type: constant data)
   - the critical period = 3 seconds

7. Num_dec_time_cycles (symbol type: constant data)
   - the number of critical periods = 5

8. State_transition_time_const (symbol type: constant data)
   - transition delay between main states = 5 seconds

In simple terms, after entering the state, the Pressure_section_pressurization symbol is set to 1, signalizing the start of the pressurization process (enabling the corresponding controller). The pressurization continues until the inner N_2 tank's pressure level remains within the acceptable deviation from 150 bars for 15 seconds (the critical period times the number of critical periods: $3 \cdot 5$). The pressurization process is then terminated by setting the Pressure_section_pressurization symbol to 0, and after 5 seconds, the next state is activated.

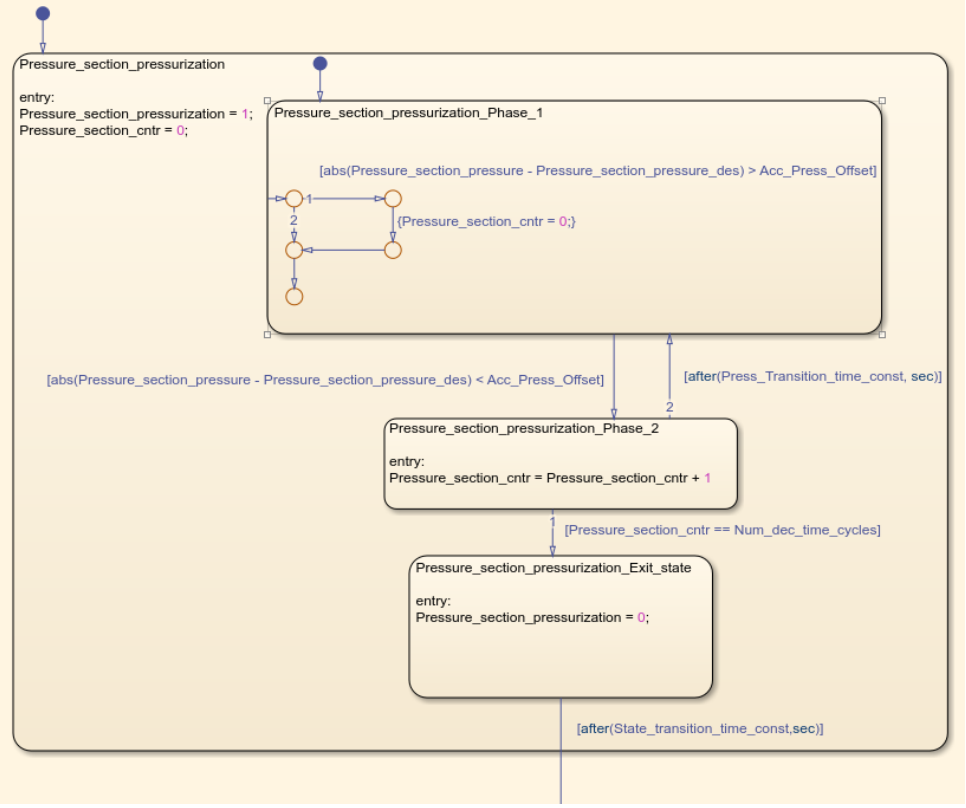


Figure 6.7: The initial pressurization of the inner N_2 tank - Stateflow state

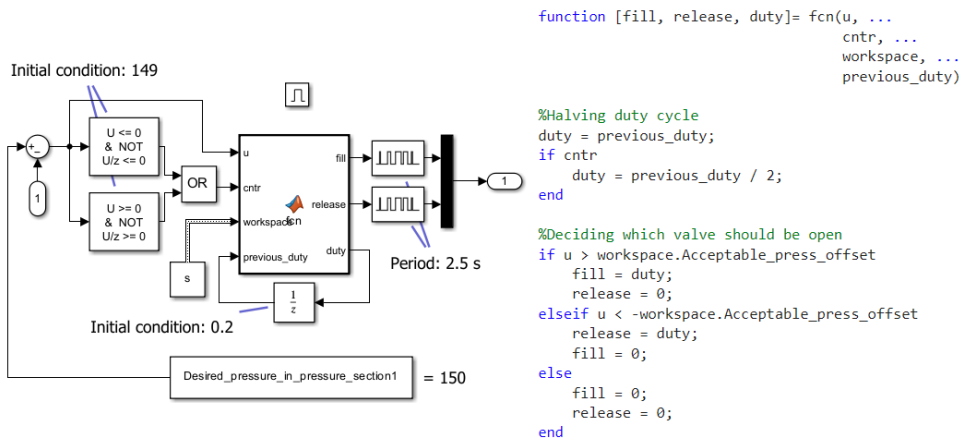


Figure 6.8: Simple controller for the inner N_2 tank's pressurization - structure

Two controllers were designed for the inner N_2 tank's pressurization. The first one [6.8] is pretty simple; it opens the N_2 fill valve or N_2 release valve based on an error (desired pressure value - actual pressure value). If the error is negative, the N_2 release valve is opened, and if positive, the N_2 fill valve is opened. The opening time is computed as the product of the current duty cycle and the PWM period. Whenever the error crosses zero, the duty cycle is halved,

increasing convergence to zero error. However, the absolute convergence to zero cannot be ensured because of the IOcard's sample rate (the minimal time interval between the opening and closing of a valve is $2 \cdot 0.05 = 0.1$ seconds). But the goal is just to maintain the pressure level within the acceptable deviation ($\pm$ 5 bars) from 150 bars for 15 seconds to transition to the next phase. The PWM's period is firmly set to 2.5 seconds, and the initial duty cycle is 0.2. Therefore, the opening time interval is 0.5 seconds until the first zero crossing. The picture [6.9] shows the validation plots, demonstrating proper functioning.

The second controller is the proportional controller [6.10], with the proportional gain of 0.008. Its output, the error term multiplied by the proportional gain, is inputted into the appropriate PWM block acting as its duty cycle. A PWM block's input (duty cycle) must be between 0 and 1, so every input lower than zero is interpreted as zero, and every input above one is interpreted as one. Therefore, when the PID block's output is negative, the N_2 release valve is opened (because of the gain block), and when positive, the N_2 fill valve is opened. The PID block's output is saturated between -1 and 1, and the period of both PWM blocks is set to 2 seconds. The picture [6.11] shows its validation plots.

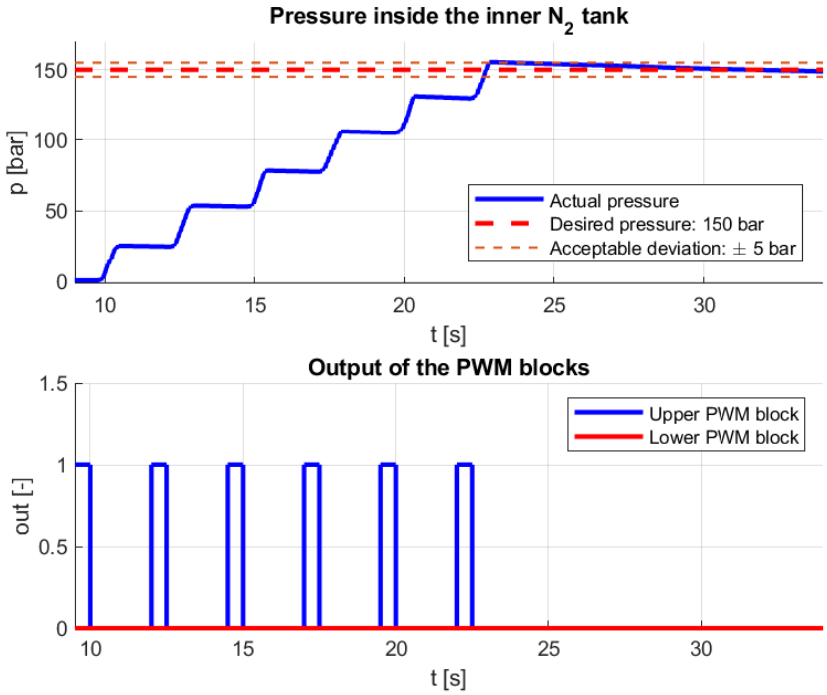


Figure 6.9: Validation of [6.8]. - initial pressurization

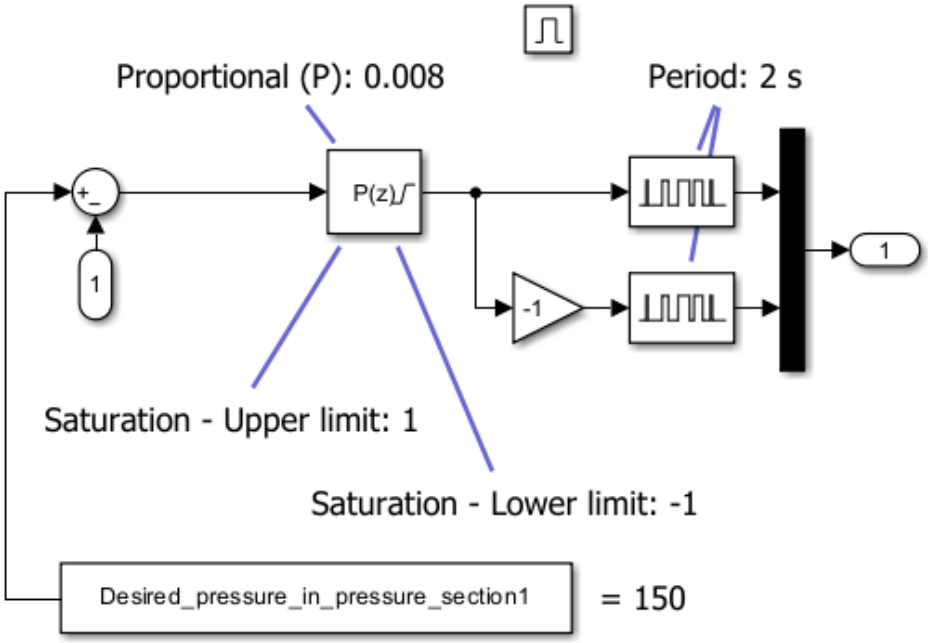


Figure 6.10: P-controller for the inner N_2 tank's pressurization

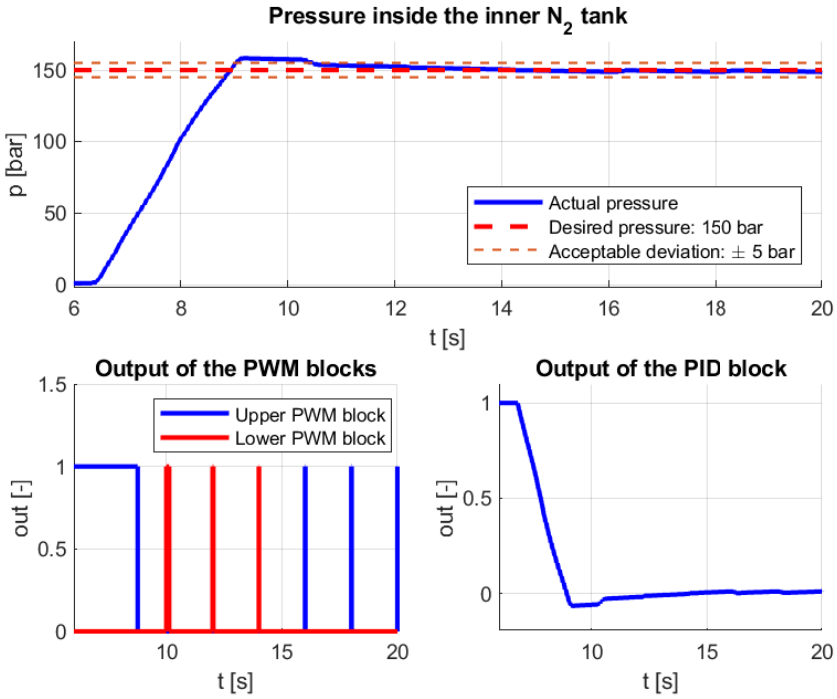


Figure 6.11: Validation of [6.10]. - initial pressurization

6.3.2 The pressure transfer between the inner rocket's tanks

Given that the pipeline between the inner rocket's tanks is equipped with the hardware pressure controller, disallowing the inner oxidiser tank to be pressurized over 40 bars, the designed software controller, pressurizing the inner oxidiser tank to 40 bars, was developed only for SIL. The corresponding Stateflow Chart state [6.12] is structurally analogous to that of the previous phase [6.7]. The present symbols are:

1. Pressure_venting (symbol type: output)
   - its value determines whether a controller for the pressure transfer between the inner rocket's tanks is switched on or off (1 - ON, 0 - OFF)
2. Pressure_venting_cntr (symbol type: local data)
   - its value determines the number of critical periods during which the pressure level in the inner oxidiser tank remains within an acceptable deviation from 40 bars
3. oxidiser_tank_pressure (symbol type: input)
   - it equals the actual pressure level in the inner oxidiser tank
4. oxidiser_tank_pressure_des (symbol type: constant data)
   - desired pressure level in the inner oxidiser tank after the pressure transfer = 40 bars
5. Acc_Press_Offset (symbol type: constant data)
   - the acceptable deviation from the desired pressure level ($\pm$ 5 bars)
6. Press_Transition_time_const (symbol type: constant data)
   - the critical period = 3 seconds
7. Num_dec_time_cycles (symbol type: constant data)
   - the number of critical periods = 5
8. State_transition_time_const (symbol type: constant data)
   - transition delay between main states = 5 seconds

Only one simple controller was designed to transfer pressure between the inner rocket's tanks because the software control is not needed apart from SIL. It is structurally the same as [6.8]; however, its parameters are slightly modified. Those modifications are visible in [6.13]. The picture [6.14] shows its validation plots.

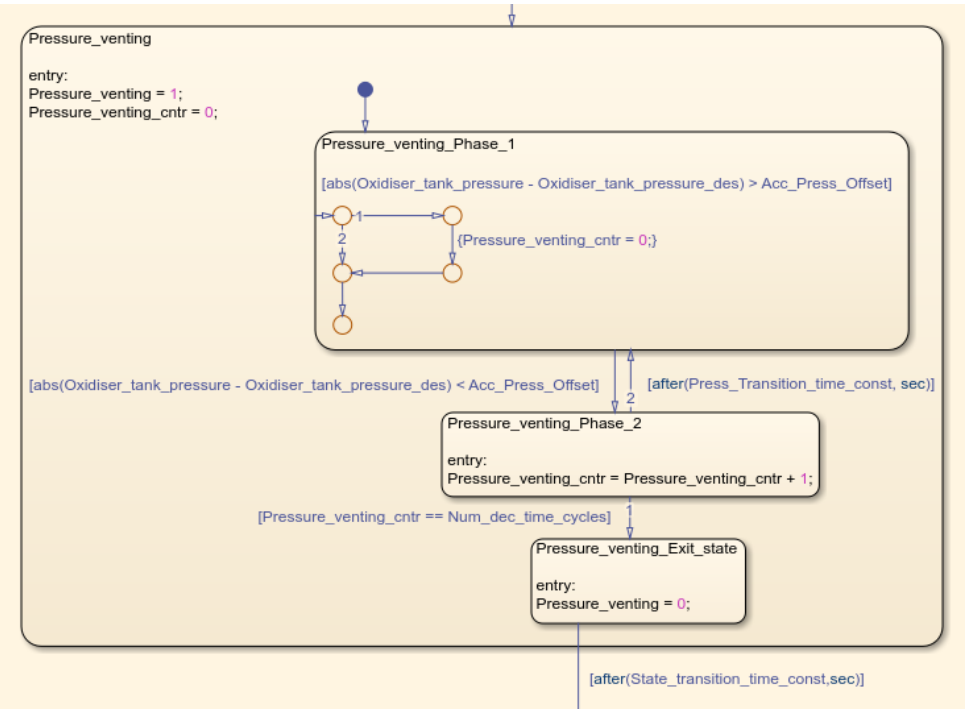
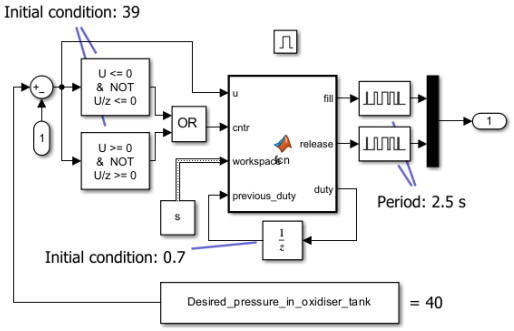


Figure 6.12: The press. trans. between the inner rocket's tanks - Stateflow state




```
function [fill, release, duty]= fcn(u, ...
                                    cntr, ...
                                    workspace, ...
                                    previous_duty)

%Halving duty cycle
duty = previous_duty;
if cntr
    duty = previous_duty / 2;
end

%Deciding which valve should be open
if u > workspace.Acceptable_press_offset
    fill = duty;
    release = 0;
elseif u < -workspace.Acceptable_press_offset
    release = duty;
    fill = 0;
else
    fill = 0;
    release = 0;
end
```

Figure 6.13: Simple controller for the pressure transfer - structure

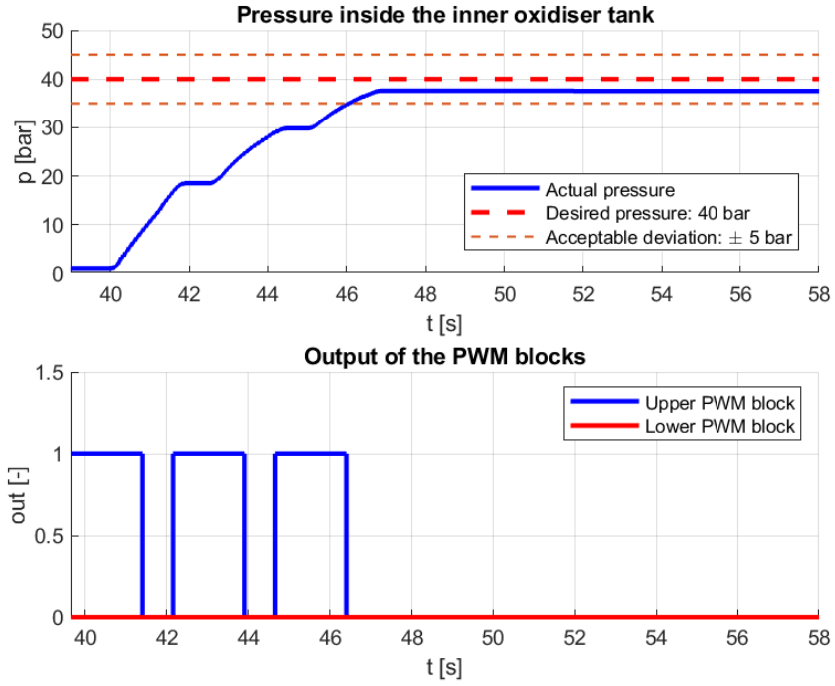


Figure 6.14: Validation of [6.13 - pressure transfer

6.3.3 The preparation of the inner oxidiser tank's refueling

Before the oxidiser refueling, the High servo, Low servo, and N_20 fill valve must be closed, opened, and opened, respectively. That is executed in this phase. After entering the corresponding Stateflow Chart state [6.15], signals triggering the transmission of packets controlling (closing and opening) the particular valves are sent. This state stays active until information is obtained that all those valves are fully closed or opened. The present symbols are:

1. oxidiser_fueling_pre_phase_cntrl (symbol type: output)
   - its value determines whether a subsystem transmitting the particular packets is enabled (1 - enabled, 0 - disabled)

2. Venting_valve (symbol type: output)
   - it equals the commanded state of the High servo (1 - ON, 0 - OFF)

3. Releasing_valve (symbol type: output)
   - it equals the commanded state of the Low servo (1 - ON, 0 - OFF)

4. oxidiser_fueling_valve (symbol type: output)

- it equals the commanded state of the N_2O fill valve (1 - ON, 0 - OFF)

5. High_servo_state (symbol type: input)
   - it equals the actual state of the High servo

6. Low_servo_state (symbol type: input)
   - it equals the actual state of the Low servo

7. N2O_Fill_valve_state (symbol type: input)
   - it equals the actual state of the N_2O fill valve

8. State_transition_time_const (symbol type: constant data)
   - transition delay between main states = 5 seconds

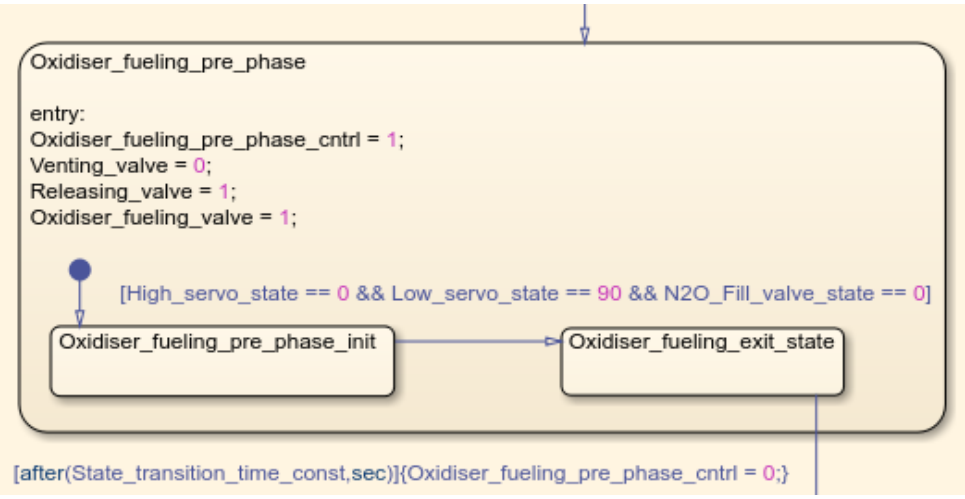


Figure 6.15: The prep. of the inner oxidiser tank's refueling - Stateflow state

6.3.4 The refueling of the inner oxidiser tank

Because the oxidiser level sensor in the inner oxidiser tank is missing, the corresponding Stateflow Chart state [6.16] enables manual refueling. There is a feature enabling pulse valve control, meaning that a valve is opened only for a specified time interval. The valve is opened manually by sending a trigger signal and automatically closed after. The present symbols are:

1. oxidiser_fueling_cntrl (symbol type: output)
   - its value determines whether a subsystem allowing for manual control of related valves is enabled (1 - enabled, 0 - disabled)

2. oxidiser_fueling_valve (symbol type: output)
   - it equals the commanded state of the N_2O fill valve (1 - ON, 0 - OFF)

3. oxidiser_fueling_valve_time_open (symbol type: input)
   - it equals the specified time interval

4. oxidiser_fueling_valve_cntrl_sig (symbol type: input)
   - it triggers valve opening (1 - OPEN, 0 - IGNORE)

5. oxidiser_fueling_exit (symbol type: constant data)
   - it equals to oxidiser_fueling_valve_time_open's value signalizing the end of oxidiser refueling

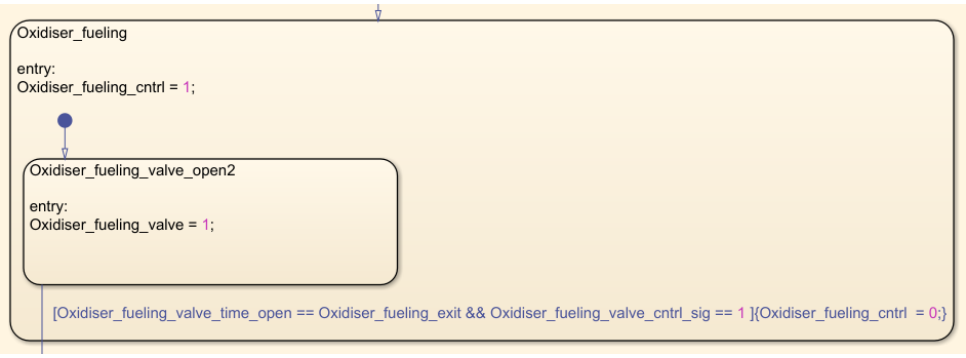


Figure 6.16: The refueling of the inner oxidiser tank - Stateflow state

6.3.5 The completion of the inner oxidiser tank's refueling

This phase resembles [6.3.3]. After refueling the oxidiser, the Low servo and N_20 fill valve must be closed. By entering the corresponding Stateflow Chart state [6.17], signals triggering the transmission of packets controlling the valves above are sent. This state stays active until information is obtained that both valves are fully closed. The present symbols are:

1. oxidiser_fueling_post_phase_cntrl (symbol type: output)
   - its value determines whether a subsystem transmitting the particular packets is enabled (1 - enabled, 0 - disabled)

2. Releasing_valve (symbol type: output)
   - it equals the commanded state of the Low servo (1 - ON, 0 - OFF)

3. oxidiser_fueling_valve (symbol type: output)
   - it equals the commanded state of the N_2O fill valve (1 - ON, 0 - OFF)

4. Low_servo_state (symbol type: input)
   - it equals the actual state of the Low servo

5. N2O_Fill_valve_state (symbol type: input)
   - it equals the actual state of the N_2O fill valve

6. State_transition_time_const (symbol type: constant data)
   - transition delay between main states = 5 seconds

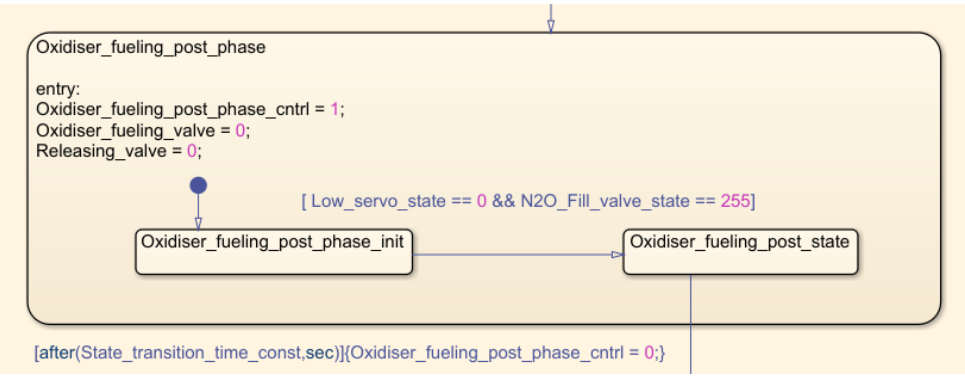


Figure 6.17: The compl. of the inner oxidiser tank's refueling - Stateflow state

6.3.6 The final pressurization of the inner N_2 tank

Before the terminal phase, the inner N_2 tank is supposed to be pressurized to 180 bars. There is little difference between the initial and final pressurization except for pressurizing to a higher pressure level. Therefore, the designed controllers [6.10] and [6.8] for the initial pressurization also hold for the final pressurization. The only things that differ are the reference pressure level and, in case of [6.8], the initial condition parameter in the zero-crossing detection blocks. The corresponding state in the Statflow Chart block is shown in the picture [6.18]. The present symbols are:

1. Pressure_section_post_pressurization (symbol type: output)
   - its value determines whether a controller for the inner N_2 tank's pressurization is switched on or off (1 - ON, 0 - OFF)

2. Pressure_section_post_cntr (symbol type: local data)
   - its value determines the number of critical periods during which the pressure level in the inner N_2 tank remains within an acceptable deviation from 180 bars

3. Pressure_section_pressure (symbol type: input)
   - it equals the actual pressure level in the inner N_2 tank

4. Pressure_section_post_pressure_des (symbol type: constant data)
   - desired pressure level after the pressurization = 180 bars

5. Acc_Press_Offset (symbol type: constant data)
   - the acceptable deviation from the desired pressure level (± 5 bars)

6. Press_Transition_time_const (symbol type: constant data)
   - the critical period = 3 seconds

7. Num_dec_time_cycles (symbol type: constant data)
   - the number of critical periods = 5

8. State_transition_time_const (symbol type: constant data)
   - transition delay between main states = 5 seconds

The pictures [6.19] and [6.20] show validation plots of [6.8] and [6.10] (for final pressurization), respectively.

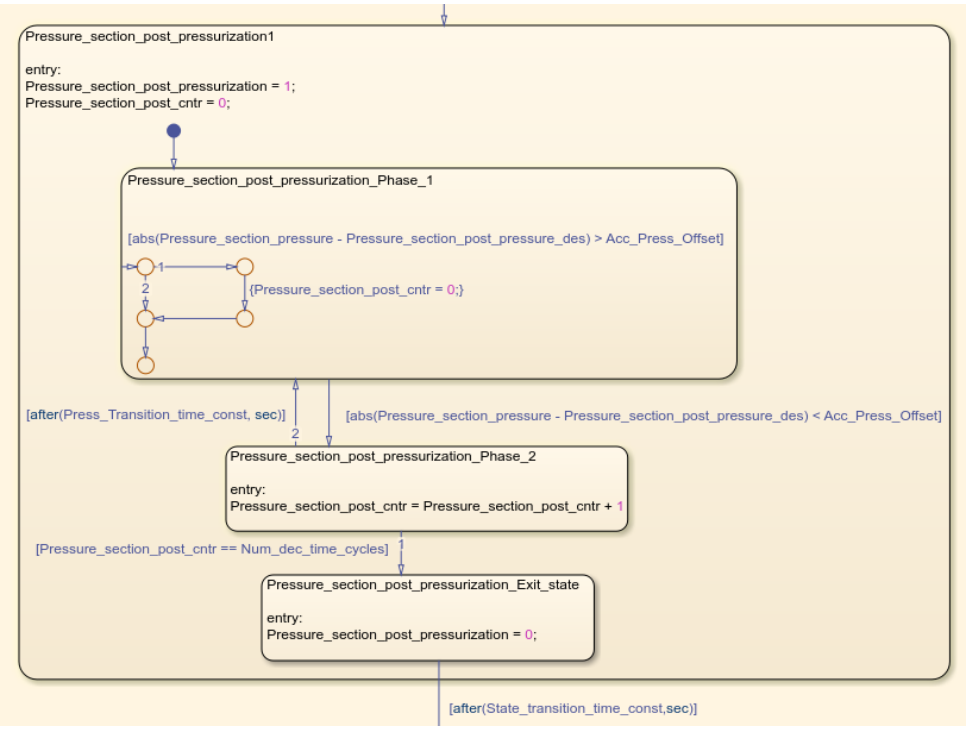


Figure 6.18: The final pressurization of the inner N_2 tank - Stateflow state

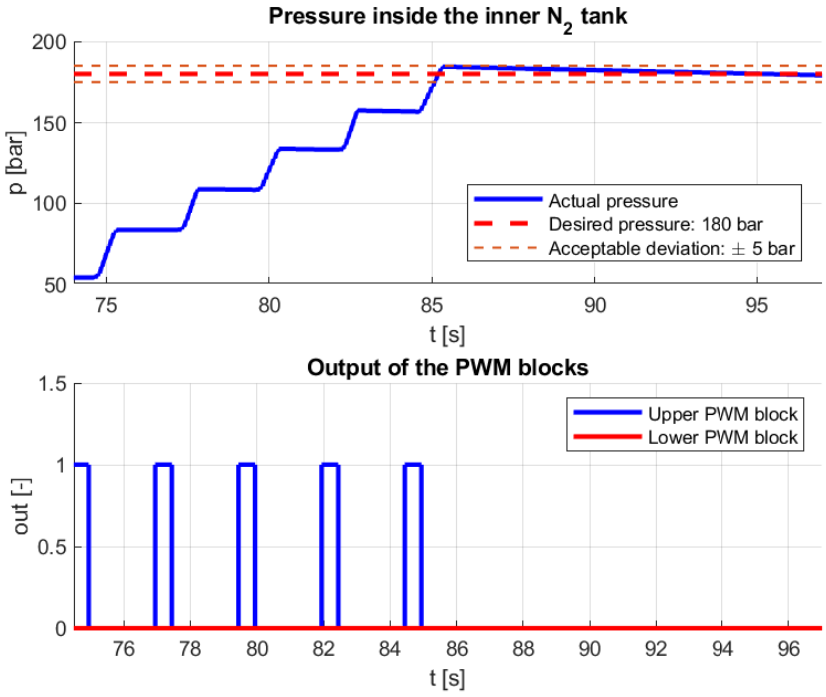


Figure 6.19: Validation of [6.8]. - final pressurization

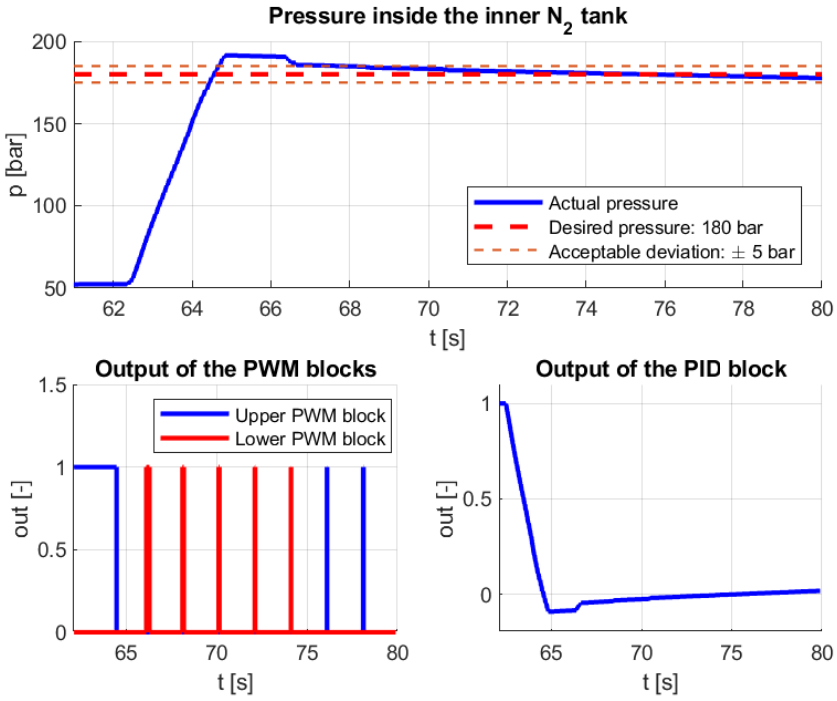


Figure 6.20: Validation of [6.10]. - final pressurization

6.3.7 The terminal phase

The terminal phase of the refueling procedure involves closing each valve, which was not forced to close before. The corresponding Stateflow Chart state is displayed in [6.21]. The present symbols are:

1. END_Phase_cntrl (symbol type: output)
   - its value determines whether a subsystem transmitting the packets closing the valves is enabled (1 - enabled, 0 - disabled)
2. Venting_valve (symbol type: output)
   - it equals the commanded state of the High servo (1 - ON, 0 - OFF)
3. Releasing_valve (symbol type: output)
   - it equals the commanded state of the Low servo (1 - ON, 0 - OFF)
4. oxidiser_fueling_valve (symbol type: output)
   - it equals the commanded state of the N_2O fill valve (1 - ON, 0 - OFF)

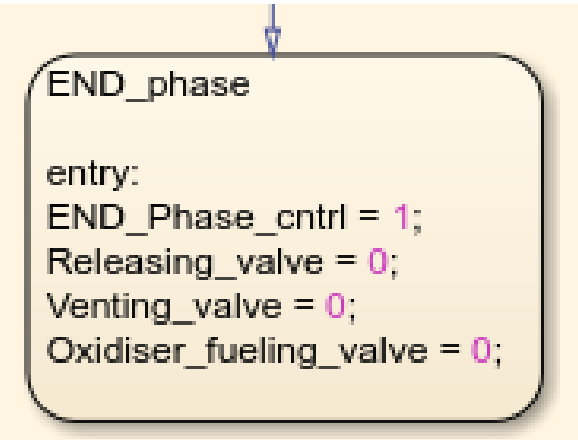


Figure 6.21: The terminal phase - Stateflow state

6.4 Control panel

A control panel was designed to control and supervise everything essential in one place. It enables changing the OMA modes, manual valve control, reading tank pressure levels, and seeing whether any fault was detected. Its structure is shown in [6.22].

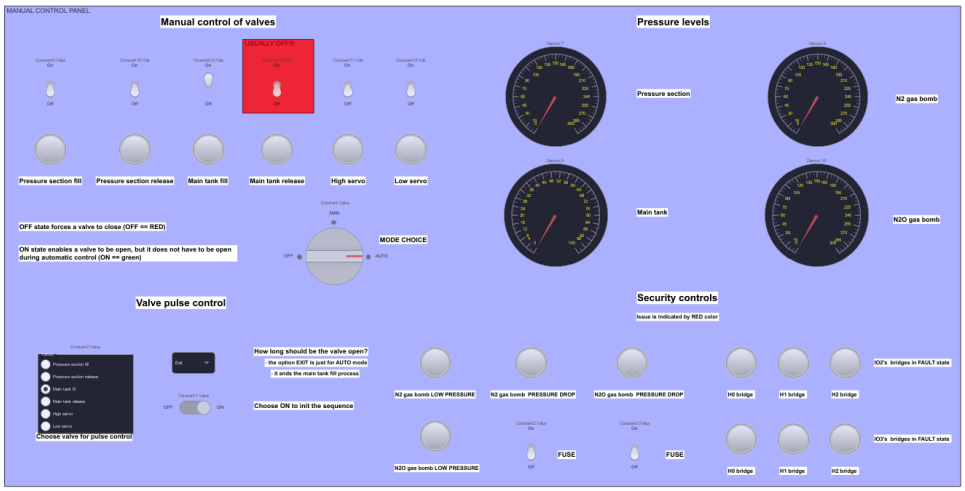


Figure 6.22: Control panel

Chapter 7

Thesis results and summary

7.1 Thesis results

1. Develop a Simscape model of the rocket and its GSE emulating their physical principles.
   - Discussed in the section [5.1].

2. Validate the Simscape model against real system data.
   - Validated in the subsection [5.1.2].

3. Enhance the Simscape model with a communication interface and create a virtual twin of the rocket and its GSE. Use the knowledge of blocks in Simulink that work with UDP packets.
   - Discussed in the section [5.3].

4. Design a state machine in the Simulink Stateflow Chart block managing transitions between phases of the refueling procedure.
   - Discussed in the section [6.3].

5. Create controllers for the inner rocket's N_2 tank pressurization and for pressure transfer between the inner rocket's tanks.
   - Discussed in the section [6.3].

6. Perform SIL using the developed refueling system and the virtual twin.
   - SIL results given in the section [7.2].

7.2 Summary

In conclusion, a few words about thesis results, achievements, and failures should be said. There were two cold-flow tests (a test testing systems inside the rocket and the GSE without ignition), but both were unsuccessful. The first failed because even though the IOcard can process 20 control messages per second, the power source is too weak, disallowing the UnIO to move with more valves simultaneously. That was something we did not expect. Since the new control app is currently being developed in Python, I decided to rewrite the designed refueling system from Simulink to Python. The Python version seemingly fixed the problems that occurred during the first cold-flow test (available in [5]). The second test was a complete failure, but the designed refueling system did not cause it. There were communication issues between the command center and the flight computer, so the servos inside the rocket were uncontrollable. Because there is currently no time for another cold-flow test as it involves a collaboration of many team members, the SIL must be sufficient for the purpose of this thesis.

Although the final solution for the automatic refueling system could not be tested on the real system, the SIL was successfully performed, which looks promising. I will build on that success in the future. Given that our team is currently designing a new rocket, including an oxidiser-level sensor, the actual version of the refueling system will be enhanced with automatic oxidiser refueling. So, its development will continue even after this thesis.

In the end, the pictures [7.1] and [7.2] show pressure profiles inside the inner rocket tanks obtained by performing SIL.

Every code this thesis presents is available on [5].

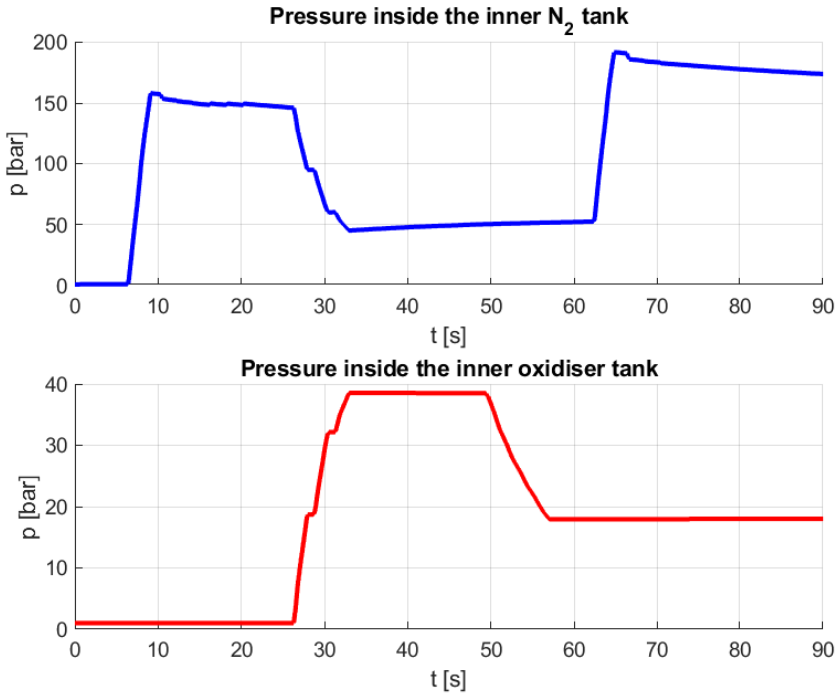


Figure 7.1: Pressure inside the inner rocket's tanks - SIL, using P-controller for the pressurization of the inner N_2 tank

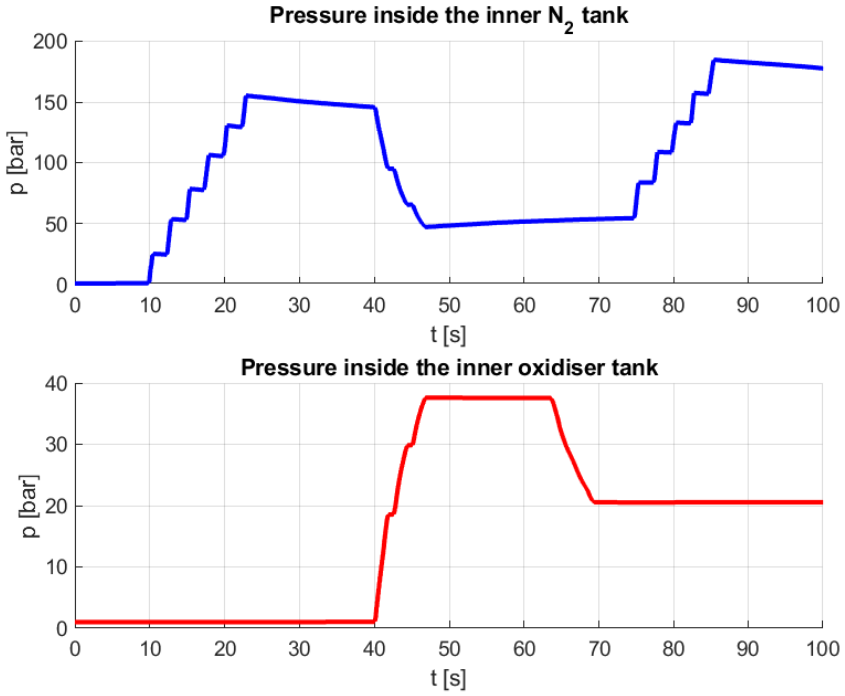


Figure 7.2: Pressure inside the inner rocket's tanks - SIL, using Simple controller for the pressurization of the inner N_2 tank

The refueling of the inner oxidiser tank was manually stopped in both cases when the pressure in the inner oxidiser tank reached approximately 20 bars.

Appendix A

Bibliography

[1] Ball, W. D., & Key, A. J. (2019, May). *Introductory Chemistry – 1st Canadian / NSCC Edition*. NSCC.

[2] Cohen, M. (2012). *Classical Mechanics: a Critical Introduction*. University of Pennsylvania.

[3] Lewin, W., & Goldstein, W. (2011). *For the love of physics*. Simon and Schuster.

[4] Ling, S. J., Sanny, J., & Moebs, W. (2016, September 29). *University Physics Volume 1*. OpenStax.

[5] Meisner, M. (2024, April). *GitHub - Matyas222/Bachelor-thesis-CTU-Refueling-system: This repository contains all essential files written during the development of the automatic refueling system for CTU Space Research's rocket Illustria*. GitHub. `https://github.com/Matyas222/Bachelor-thesis-CTU-Refueling-system`

[6] Wittman, J. (2022, March). *Chemistry of Food and Cooking*. MHCC Library Press.

[7] *Chart - MATLAB*. (n.d.). `https://www.mathworks.com/help/stateflow/ref/chart.html?searchHighlight=Chart&s_tid=srchtitle_support_results_2_Chart`

[8] *Constant Volume Chamber (G) - MATLAB*. (n.d.). `https://www.mathworks.com/help/simscape/ref/constantvolumechamberg.html`

[9] *Convective Heat Transfer - MATLAB*. (n.d.). `https://www.mathworks.com/help/simscape/ref/convectiveheattransfer.html`

[10] *Team 08 Technical Report to the 2023 EuRoC*. (2023). https://github.com/Matyas222/Bachelor-thesis-CTU-Refueling-system

[11] *Hybrid Rocket Propulsion*. (2021, February 11). https://aerospacenotes.com/propulsion-2/hybrid-rocket-propulsion/

[12] *Hybrid Rocket Propulsion 2*. (n.d.). aerospacenotes.com. https://aerospacenotes.com/propulsion-2/hybrid-rocket-propulsion/

[13] *Hybrid Rockets*. (n.d.). Penn State University. https://www.psu.edu/news/research/story/hybrid-rockets/

[14] *Liquid-propellant rocket*. (2024, April 26). Wikipedia. https://en.wikipedia.org/wiki/Liquid-propellant_rocket#/media/File:Liquid-Fuel_Rocket_Diagram.svg

[15] *Liquid Rocket Engine*. (n.d.). https://www.grc.nasa.gov/www/k-12/airplane/lrockth.html

[16] *Local Restriction (G) - MATLAB*. (n.d.). https://www.mathworks.com/help/simscape/ref/localrestrictiong.html

[17] *nistdata(species,T,p)*. (2020, April 6). Nistdata(Species,T,P) - File Exchange - MATLAB CentralFile Exchange - MATLAB Central. https://www.mathworks.com/matlabcentral/fileexchange/64578-nistdata-species-t-p

[18] *Pipe (G) - MATLAB*. (n.d.). https://www.mathworks.com/help/simscape/ref/pipeg.html

[19] *Pressure & Temperature Sensor (G) - MATLAB*. (n.d.). https://www.mathworks.com/help/simscape/ref/pressuretemperaturesensorg.html

[20] *Reservoir (G) - MATLAB*. (n.d.). https://www.mathworks.com/help/simscape/ref/reservoirg.html?searchHighlight=Reservoir%20%28G%29&s_tid=srchtitle_support_results_1_Reservoir%20%2528G%2529

[21] *Rocket engine*. (2024, March 28). Wikipedia. https://en.wikipedia.org/wiki/Rocket_engine

[22] *Rocket Principles*. (n.d.). https://web.mit.edu/16.00/www/aec/rocket.html

[23] *Solid-propellant rocket*. (2024, April 3). Wikipedia. https://en.wikipedia.org/wiki/Solid-propellant_rocket#/media/File:Solid-Fuel_Rocket_Diagram.svg

[24] *Solid Rocket Engine*. (n.d.). https://www.grc.nasa.gov/www/k-12/airplane/srockth.html

[25] *Specific impulse - Kerbal Space Program Wiki*. (n.d.). https://wiki.kerbalspaceprogram.com/wiki/Specific_impulse

[26] *Temperature Source - MATLAB*. (n.d.). https://www.mathworks.com/help/simscape/ref/temperaturesource.html

[27] *Thermodynamic models and tools. H2O, H2, CO2, air, and more*. (2023, May 12). Thermodynamic Models and Tools. H2O, H2, CO2, Air, and More - File Exchange - MATLAB CentralFile Exchange - MATLAB Central. https://se.mathworks.com/matlabcentral/fileexchange/73950-thermodynamic-models-and-tools-h2o-h2-co2-air-and-more