POLITECNICO DI TORINO

Master's degree course in Mechatronic Engineering

Master's Degree Thesis

Design, development and test of the control unit for a small cold-gas propulsion system.

Candidate:
Martina Sacco

Supervisors:
Prof. Sabrina Corpino
Eng. Fabrizio Stesina
Eng. Filippo Corradino

Academic Year 2019/2020

Summary

List of figures

List of tables

Abbreviations

ADC	Analog to Digital Converter
COTS	Component Off-the-Shelf
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DOF	Degree Of Freedom
FDIR	Fault Detection, Isolation and Recovery
GNC	Guidance, Navigation and Control
GND	Ground
GPIO	General Purpose Input Output
HW	Hardware
I2C	Inter Integrated Circuit
ISR	Interrupt Service Routine
LED	Light Emitting Diode
MIB	Minimum Impulse Bit
MOSFET	Metal-Oxide-Semiconductor Field-Effect
MSB	Most Significant Bit
PS	Propulsion System
RPi	Raspberry Pi
S\C	Spacecraft
SPI	Serial Peripheral Interface
SSH	Secure Shell
SW	Software
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver-Transmitter
uC	microController
uP	microProcessor
USB	Universal Serial Bus

Definitions

I2C protocol: it's a serial communication protocol where data is transferred bit by bit along a single wire (the SDA line). It's synchronous since the output of bits is synchronized to the sampling of bits by a clock signal (the SCL line) shared between the master and the slave.

Kapton: it a special polyimide film known for its thermal stability, its good chemical resistance and excellent mechanical properties used for insulation and high thermal resistance special in aerospace.

Microcontroller: it's a small computer on a single chip which contains one or more CPUs along with memory and programmable input/output peripherals.

Specific Impulse: it is a measure of how effectively a rocket uses propellant. It is the total impulse delivered per unit of propellant consumed.

Thread: it's the smallest sequence of programmed instructions that can be managed independently by a scheduler. Multithreads share resources such as memory.

UART protocol: it an asynchronous serial communication by means of two wires. The data flows from the Tx pin of the transmitting device to the Rx pin of the receiving device.

1 - Introduction

In recent years, thanks to the proposal of new missions that bring a human crew to the Moon and subsequently to Mars, the space sector has once again attracted the attention of the public. But next to the great and spectacular missions, there are smaller and less-known missions carried on with satellites and rovers. Among them there are a category of small satellites, called CubeSat.

A CubeSat is a standard for nanosatellites with the characteristic of being modular, where the base unit, 1U, is a cubic satellite with a 1dm^3 volume and a weight not more than 1.33 Kg, therefore it is possible to have 3U CubeSat with dimensions of 10x10x30 cm or 6U CubeSat 10x20x30cm and so on. The first CubeSat was developed in California in 1999, the goal was to use them for educational purposes, but thanks to their lower cost in terms of time and funds than traditional missions they are now widely used in the space industry. Today more than 1300 CubeSats have been launched (May 2020)[1] for missions ranging from Earth observation to providing internet service, NASA had also used two CubeSat for a mission in the orbit of Mars in 2018.

CubeSats are projected into a brilliant future, but the technology still needs improvements and the process of manufacturing, assembly, integration and verification shall become more efficient. In this framework, miniaturized propulsion systems deeply increase the range of mission concept achievable with multi-unit CubeSats (6U+) in terms of orbit change and raising, station keeping and orbit maintenance against the disturbances, formation flying, proximity operations and de-orbit, but an improvement is required from the technological point of view in order to increase the technology readiness level of some enabling technology for the future missions.
These enabling technologies are high data rate communication systems, high accuracy attitude and orbit determination and control devices, thermal control systems. Moreover, new logical and physical architectures, and the capability to re-plan the operations during the mission become fundamental for innovative applications and unprecedented missions where CubeSats are main characters. One of the most promising technologies is the small propulsion system that open new scenarios for the modern CubeSats that would perform controlled orbit (and attitude) manoeuvres.

In this context, Tyvak International is a company born in 2015 specialized in spacecraft development, launch services and on-orbit operations, which in 2019 starts a roadmap for the development of propulsion module for CubeSat.
To characterize a propulsion system there are many specifications to examine:

- Propulsion technologies: electric or chemical;
- Thruster delta-V capability[2]: the velocity change needed to achieve the required new trajectory, is the measure of the energy needed to transfer from one orbit to another;
- Operative power: the power supply to which the thruster operates.
- Minimum Impulse Bit (MIB) [3]: the measure of the smallest control force or torque that can be commanded to the satellite using the thruster;
- Integration requirements: requirements linked to the process of bringing together the propulsion module and the spacecraft;
- Size and weight.

The challenge is to develop new miniaturized technologies for CubeSat, which, compared to a large satellite, has the disadvantage of having less volume, mass, and power.
This thesis fit Tyvak project and has the purpose of design, development, and test of a control unit for this propulsion module.

1.1 State of art

To explain the motivations that have prompted Tyvak International to start the development of its own propulsion system, it's reported the results of a recent market analysis[4]in the framework of CubeSat Proximity Operations performed by Tyvak to identify the potential COTS solutions for Proximity-Operations missions. It also justifies by the increment of the potential missions of in-orbit servicing that can be enabled by this type of operations. The SROC mission foresees a CubeSat released by the Spice Rider vehicle and able to perform a set of rendezvous and docking manoeuvres, that allows the CubeSat to re-entry in the cargo bay of Space Rider after long phases of fly-around this vehicle. The CubISSsat program studies the possibility of one or more CubeSats to fly in proximity of the ISS to inspect the external elements of the Space Station and monitor outside extra-vehicular activities. Other example of proximity operations is the mission C-POD where two 3U CubeSats shall perform a mutual manoeuvres of rendezvous and docking. All these missions have to be accomplished through new miniaturized propulsion systems.
Where for proximity operations[5] are intended any type of activities between two spacecraft in the relatively close area, where the distances between the two S\C can be in a range from some kilometres to a few meters or less, depending on the mission characteristics and specificities of performed operations.

The analysis was performed for a 6U and 12U CubeSat, but for this thesis purpose only the considerations about the 6U satellite are reported. The results of analysis also separated the EU market from the US market, to highlight any regional differences in the propulsion systems availability, and the summary is that:

- the EU market for nanosatellite propulsion is mainly focused on warm gas thrusters or cold gas thrusters with moderate thrust but low delta-V, and on many electric propulsion systems with very low thrust, very high delta-V and very high-power draw.
- The US market, on the other hand, seems to offer cold gas with moderate thrust and a wide range of delta-V, in theory suited for long-duration Proximity Operations

Therefore, Tyvak identifies a significative gap in the EU for cold gas thrusters suitable for Proximity Operations of extended duration, although the US market provides many Proxy-Ops suitable products.
Subsequently, to identify which kind of propulsion system is suitable for this application, the main features of each technology are analysed and reported in the Table 1, extract from 'Technology Dossier'[4].

Technology	Criticalities	Suitability
Cold Gas	None	Yes
Warm Gas Chemical (e.g., monopropellant)	Hot and reactive exhaust Most systems have 1 DOF thrust Typically long preheating time	Limited Need to waive some requirements
Hot Gas Chemical (e.g., bipropellant)	Hot and reactive exhaust Most systems have 1 DOF thrust High MIB	No
Resistojets/Arcjets	Hot and reactive exhaust Most systems have 1 DOF thrust Typically long preheating time	Limited Need to waive some requirements
Radio Frequency Plasma Thrusters	Excessive power/thrust ratio Hot and reactive exhaust Highly ionized exhaust Most systems have 1 DOF thrust Potential EMI issue	No
Hall Effect Thrusters		
Gridded Ion Thrusters		
Pulsed Plasma Thrusters		
FEEP Thrusters		

Table 1: Propulsion technology for CubeSat proximity operations

The analysis addresses the cold gas system as the only suitable solution for the application.
Due to this consideration Tyvak International has started the preliminary development of a cold gas thruster for 6U specially targeted at Proxy-Ops with safety-sensitive targets.
Due to non-disclosure agreement the preliminary specifications for this thruster module are not listed, the following section reports a high-level description of the module in order to give an idea of the system for which the software is designed.

1.2 Propulsion system description

The propulsion system that Tyvak International want to develop is a monopropellant thruster for a 6U CubeSat able to deliver a thrust in the range of millinewtons in 6 degrees of freedom, torque and thrust for each of the three principal axes, using a cold gas storable at low pressure. Due to a non-disclosure agreement, more technical specifications about thrust performances or mechanical and fluidic parts are not reported in this thesis, only some electrical requirements needed to explain design choices and requirements that will be extracted from functional analysis will be illustrated.

A cold gas thruster is a rocket engine that uses the expansion of a gas to generate thrust. Unlike traditional engines, there is no combustion, indeed often the gas used in this type of propulsion is inert. It has a series of advantages[6] respect the traditional rocket engine:

- the absence of combustion and hot rocket engine eliminates the need for heat dissipation systems;
- it requires very little electrical energy to operate:
- the simple design is less prone to failures and also makes it smaller than a traditional rocket engine.
- The system and its fuel are less expensive compared to regular rocket engines.

On the other side this kind of thruster has some disadvantages - for instance:

- it cannot produce the high thrust that combustive rocket engines can achieve.
- the specific impulse of a cold gas thruster is much lower when compared with that of a traditional rocket engine.
- the maximum thrust of a cold gas thruster depends upon the pressure in the storage tank. As fuel is used up, the pressure decreases and maximum thrust decreases.

This kind of thruster has a lower efficiency and thrust but it simpler in term of design and less expensive since it is composed only by a tank, some regulation valves, nozzles and a small hydraulic system therefore it is advantageous especially in short space missions and in CubeSats.

The design of this propulsion system is entrusted for the fluidic part (tank, pipes, and propellant) to T4i with the support of the University of Padua , while system engineering, mechanical, electronics and control are entrusted to Tyvak International.

The fluidic part[7] has a configuration such that the propellant can flow directly to the nozzle and then ejected outside in gaseous state.
The propellant, a non-flammable cold gas selected by T4i that it is not described in this thesis due to a non-disclousure agreement, is stored in the tank, designed to have inside part of propellant in liquid and part in gaseous phase (the fluid is bi-phasic at ambient temperature) at pressure less than 50 bar. To keep the propellant in ideal pressure and temperature for thrusting and as a further guarantee of maintaining the propellant in a gaseous state one heater is placed on the

tank and one along the line. While to avoid the passage of debris that could clog the nozzles a filter is placed on the tank outlet.
Along the fluidic line, before the pressure regulator, there are two normally-closed valves to prevent the propellant flowing down the line when the system is not firing. The same kind of valve is placed on each of eight branches before the nozzles.

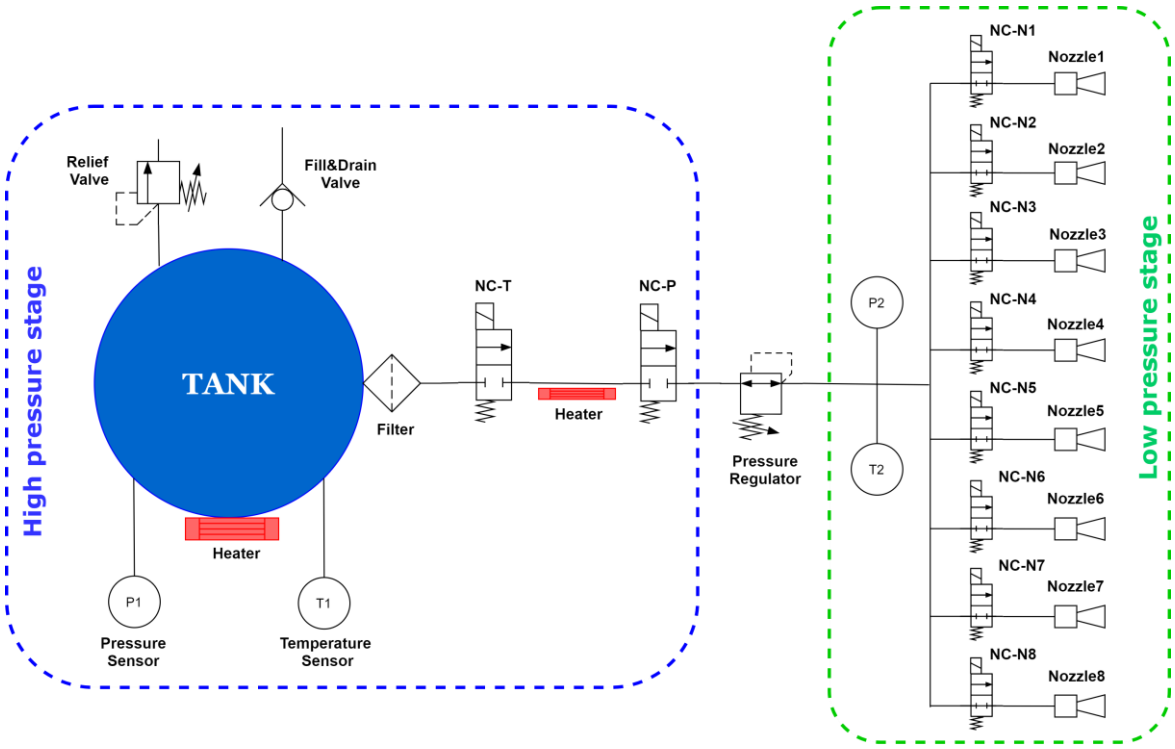


Figure 1: Fluidic system schematic

On the tank two valves are provided: one to fill and drain the propellant into the tank, one for pressure relief.
The thruster module also includes two temperature and pressure transducers: one to monitor the propellant status and one, after the pressure regulator, to measure parameters before the branching.

To generate thrust the temperature and pressure in the tank and along the pipes shall be in the range that maintain the propellant in gaseous phase, while to perform a manoeuvre, due to the nozzle configuration and orientation, at most four valves should be open at the same time in addition to the two valves along the fluidic line, used as inhibits in order to avoid unintentional propellant flow.

1.3 Requirements

Since the objective of the thesis is the development and validation of the software, only the requirements relating to the software are shown.

ID	*Title*	*Description*
PER-TEC-11	Control	Each nozzle shall be independently controllable.
PER-TEC-13	Interface - Data	The thruster module shall expose a RS-422 or RS-485 interface for command and data transfer to/from the flight computer
PER-TEC-15	Safety - Inhibits	At least 3 independent inhibits overall (i.e., at spacecraft level) shall control the opening of the flow control devices. 2+ such inhibits shall be monitored (at all times, even with thruster module off), 1+ shall be on the low-side, 1+ shall be on the high-side. No single inhibit failure shall open more than 1 flow control device.
PER-TEC-21	Technology	COTS components shall be employed wherever possible to do so.
PER-TEC-25	Valves Actuation Time	Flow control valves shall be controllable with a time resolution of 50 ms or better. 10 ms or better is a nice to have. Valves actuation can last (continuously) up to 600 s.
PER-TEC-26	Autonomy	The thruster module shall include an electronic control board providing autonomous functionality and accepting high-level software commands.
PER-TEC-27	Autonomy - Operation	The thruster module shall handle local power, control, and piloting of its own actuators and sensors.
PER-TEC-28	Autonomy - Housekeeping	The thruster module shall handle autonomously keep-alive and housekeeping operations (e.g., survival heaters, burn preparation, ...).
PER-TEC-29	Autonomy - Telemetry	The thruster module shall report on its own health status and telemetries (in response to flight computer polling).
PER-TEC-30	Autonomy - Fault handling	The thruster module shall offer local FDIR functionalities in case of non-nominal telemetry.
PER-TEC-31	Autonomy - Control	The thruster shall provide control algorithms for propulsive manoeuvres
PER-TEC-32	Control - Open Loop	The thruster module shall perform 6DOF impulsive burns in open loop (assigned thrust and torque vectors in body frame)
PER-TEC-33	Control - Closed Loop	The thruster module shall support 6DOF impulsive burns in semi-closed loop in synergy with the GNC controller on the flight computer (external command input: residual nozzles actuation times, feedback: actual valve actuation times and status)
PER-TEC-34	Control - Override	The thruster module shall offer override features for the single active elements (i.e., accept external commands for actuation or de-actuation of each single valve and heater)
PER-TEC-35	Safety - Timeout	The control signal for valves shall implement a hardware timeout of 1 second or less (i.e., the actuation signal needs to be actively refreshed somehow every second or less in order to continue firing)

Table 2: Starting avionic requirements for propulsion module

2 Project

The thesis aims to the development and test of the control unit to manage a cold-gas thruster presented in the Section 1.2.
The goal of this chapter is to present all the phases addressed for the development of this control unit in particular the software. Since there are still no detailed requirements and no hardware part designed, we start from a functional analysis to identify the main functions that the system must comply. Subsequently, a preliminary choice of components is made, based mainly on COTS components already used by Tyvak International in previous missions. At that point, having a clearer idea of the characteristics and requirements that the software shall comply, it is possible to divide the development of the code into various modules, develop one module at time and subsequently verify the overall performance of the system.

2.1 Functional analysis

The first step is to carry out a functional analysis, based on the preliminary design, to get software requirements. The main objective of the propulsion subsystem can be expressed by the function "To perform manoeuvre", from this function is possible to move down and divide it into five high-level functions:

- "*To handle the propellant*": that groups all functions linked to the fluidic part and the propellant management;
- "*To generate thrust*": that includes that operations necessary to correctly generate a thrust as keep propellant in ideal condition and direct the burst;
- "*To supply power to the system*";
- "*To guarantee safe operation*": all the functions used to guarantee safe conditions for the system
- "*To manage system operations*": all the functions linked to microcontroller and data signal processing.

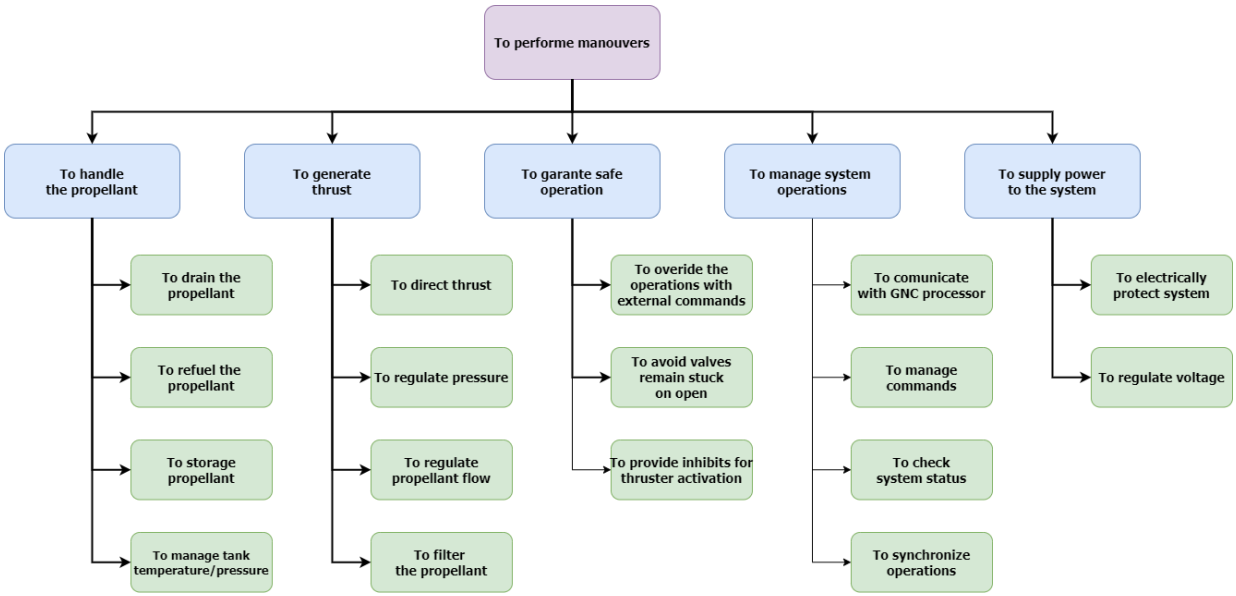


Figure 2: Propulsion module functional tree.

Each of high-level functions is developed in three levels of details, Figure 2 reports only the first two levels under the main function, the whole functional tree is reported into Appendix A1. Since the aim of the analysis is the identification of requirements related to microprocessor and its interfaces, for "To manage system operations" functions a fourth level of functions is developed Appendix A2.

Subsystems and components associated to each function have been specified in Product Tree Appendix A3, while to identify which type of interface exist between components, N2 matrix has been developed Appendix A4. Finally, functional analysis led to the definition of the functional architecture that shows the elements presented in the product tree and its interfaces in the form of a block diagram.

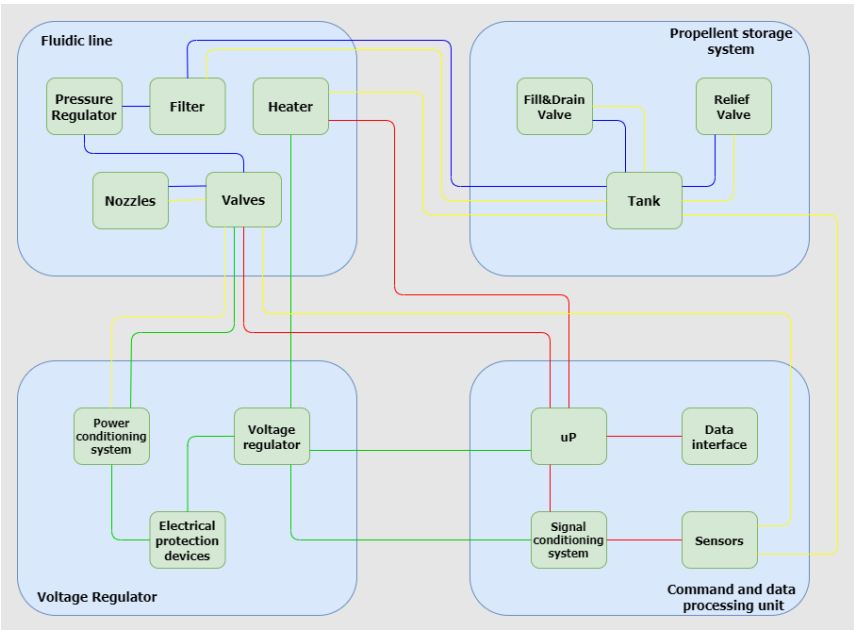


Figure 3: Propulsion system block scheme

2.2 Preliminary component selection

Thanks to functional analysis the main elements and interconnections of propulsion systems have been identified, but to start firmware development more details about the microcontroller are needed, therefore the next step is the choice of the microprocessor and a preliminary selection of the components that shall be interfaced with it.

The choice of these components is strictly related to the fact that these components have been widely used in other project and in-flight missions.

The choice for components of the fluidic line is made in the preliminary design by T4i, due to the non-disclosure agreement the selected components are not presented but we display the components type and their major characteristics:

- 8 solenoid valves that will need a conditioning circuit for correct piloting;
- 2 heaters considered as resistors that will also need a conditioning circuit;
- the pressure and temperature transducers that returns analog output so needs an ADC;
- Pressure regulator, mechanical component that decreases the pressure from 50 bar to 1,5 bar,
- Fill and drain valve, that is just a mechanical component,
- Relief valve, it is a mechanical component.

Therefore, the first identified components which interface the microcontroller are the solenoid valves, the heaters and pressure/temperature transducers; while pressure regulator, fill & drain and relief valves are just mechanical components, so they don't need any electrical control.
The pressure/temperature sensors return analog output, so they need ADCs, so MCP3421[8] is selected as a converter. This is, a 24-bit ADC with a single channel that communicates on a I2C bus, the choice is due to the previous use of these components in other missions by Tyvak International.

To control the board status, in the design are also considered some sensors of voltage and current, the selected components in this preliminary design is a component that include voltage and current measurement and communicate on I2C bus. In particular, for the code development five voltage/current sensors are considered: one for each voltage regulator, but the number can be easily changed at a later time.

Starting from the previous consideration and from the requirements that required the independent control of each actuator [PER-TEC-11] and the communication on RS422 [PER-TEC-13], a microcontroller with UART interface, I2C bus and a good number of GPIO is selected. The choice therefore fell on the MSP430-FR6898[9], a Texas-Instrument low-power controller with a good variety of interfaces and a good number of pins, already used in other missions, therefore already tested in flight.
Considering the requirement that requires at least two signals to drive the actuators, for each valves and heater, two pins are considered: an enable pin that will stay high for all the time the

actuator should be open, and a heart-beat pin that must send a pulse to a conditioning circuit for the whole time in which the enable signal remains active.

A more detailed schematic of the board is developed in order to have greater clarity of which are the main components and interfaces of the microcontroller [Figure 4].
Analysing the extracted schematic, the main identified interfaces of the microcontroller are:

- UART interfaces for the communication with the GNC,
- UART interface for the debug line for the development of the software,
- I2C bus for the data acquisition of all sensors (voltage, current, pressure and temperature),
- 2 GPIO for each valve and heater with a total of 24 pins to manage the actuators.

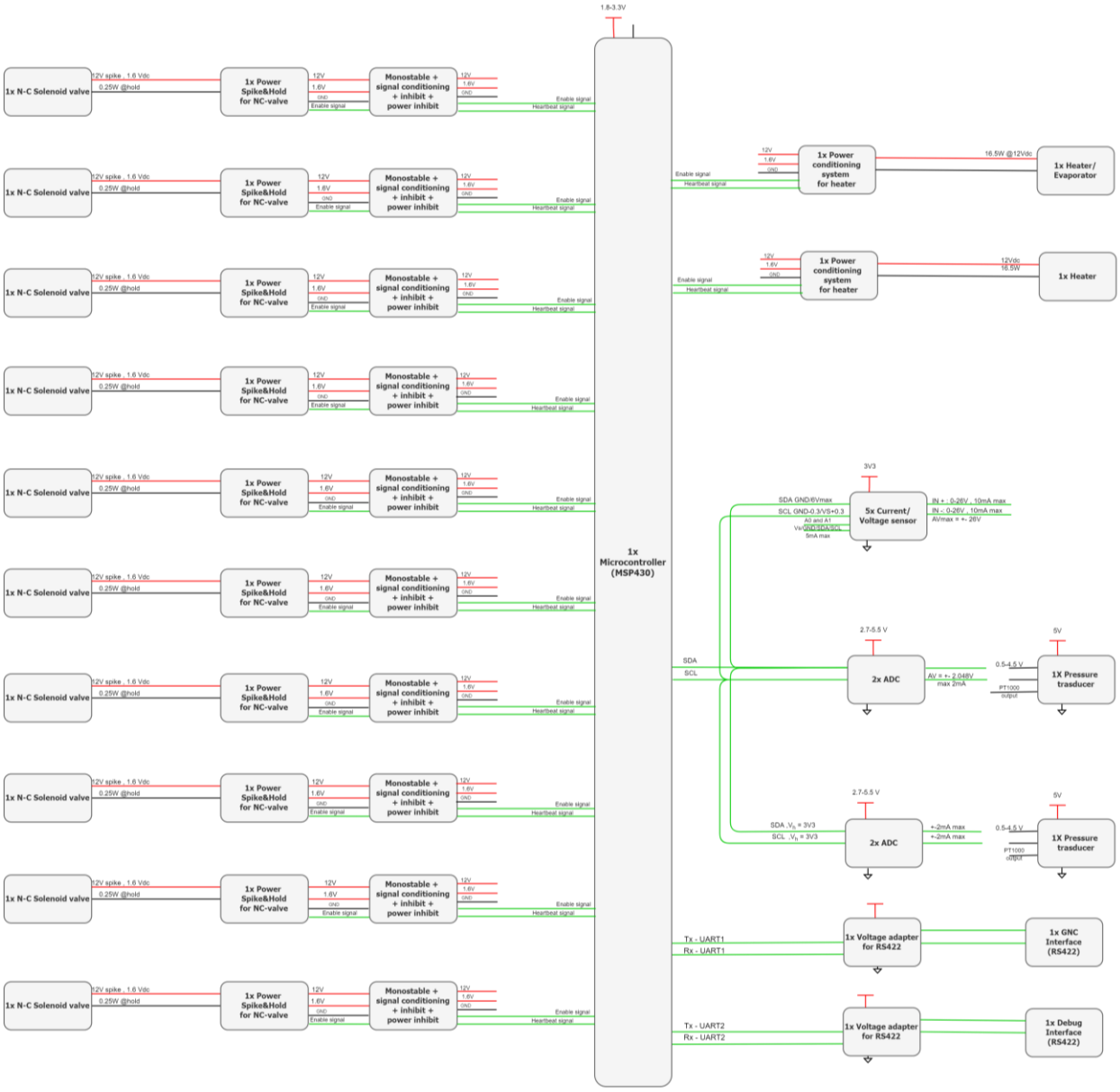


Figure 4: High-Level schematic of propulsion module electric board

From the functional analysis low level requirements are obtained for the software and hardware aspect of the control unit. The list of requirements is updated during all the development of the project, the complete list is reported in the Appendix B.

2.3 Development board

To develop and test software functionality during the writing of the code a development board is assembled. The MSP430-FR6898 microcontroller is used with its development board the MSP-EXP430FR6898 which allows to connect other devices to the microcontroller more easily. A Raspberry Pi 3 B+[10] is connected by means UART TX and RX pins to test serial communication and it runs a simple Python code in order to check the correct exchange of packets. While to develop the I2C driver and to test the data acquisition from sensors, a Tyvak International test board from another project is used. This test board has one temperature sensor and an ADC connected on the I2C bus. The temperature sensor is AD7415 sensor[11], a common temperature sensor that return directly a 10-bit converted data on I2C; while the ADC is MCP3421, a 24-bit single channel converter whose analog input is connected to the output of sun-sensors, a component that return a voltage proportional to the intensity of the light that hits its photodetector. Since the photodetector is covered to not damage it, the sun sensor output is fixed to full scale value (2.048 V), therefore also the digital value read from the ADC is fixed, but it's used anyway since it is useful to write the driver for the ADC, to check the I2C functionality with more than one element connected on the bus and to take into account the ADC acquisition time in the main loop.
In order to correctly work the I2C bus needs also two pull-up resistors connected between SDA and SCL pins and 3.3V voltage.
To have a quick visual feedback of the GPIO performances, one low-voltage LED with its respective resistance is connected on each pin that should manage actuators. To easily distinguish them three different colours are used: green for valves enable pins, yellow for valves heart-beat pins, red both for enable and heartbeat of heaters. LEDs with different colour have different voltage drop, so resistances are selected in order to let flow about 2-3 mA for each LED.

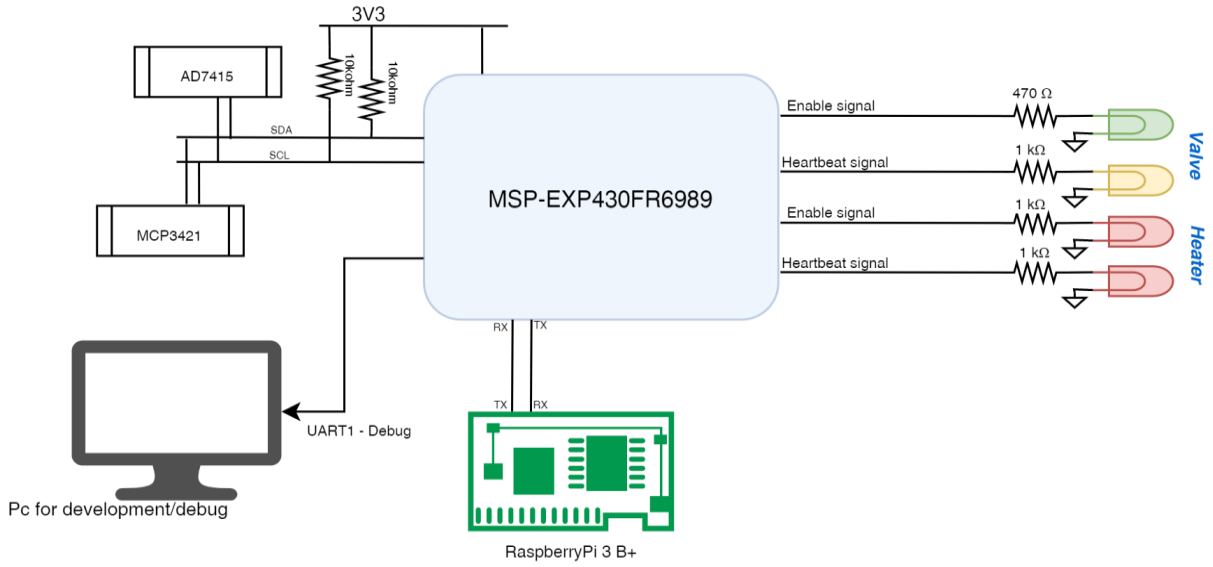


Figure 5: Simplify configuration of development board

2.4 Bare metal VS operating system

The last step before starting to write code, it is to decide which abstraction level to use to program the microcontroller. In programming, abstraction is the amount of complexity by which a system is viewed or programmed. The higher the level, the less detail. The lower the level, the more detail.
The choice falls on bare metal level or on operating system.

An operating system[12] is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of program. It performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices. The operating system allows to have multi-thread and priority among the various functions. In complex projects it helps a lot with the management of multiple processes simultaneously while the complexity of inserting an operating system isn't worth it in the case of applications that are not too complex and where multiple processes are not required.

Bare metal is a low-level method of programming and it is specific to the hardware used. It allows to directly control the hardware without device drivers from another operating system. It does not allow multi-threading in one core and the pre-emption comes from an interrupt, there is no scheduler. But its direct control of hardware makes it about 10 times faster than an operating system.
Therefore, since the software for propulsion module does not require high level of complexity, it is preferred to use the bare metal programming.

2.5 Code development

At this point, we have a clear idea of the system requirements, software main functions and microcontroller interfaces, therefore it is possible to divide the code development in modules and design, develop and test one a time. After all modules are developed and individually tested, each unit is integrated and the whole code is tested also with the help of an external 'tester'. The test campaign is described in the following chapter.

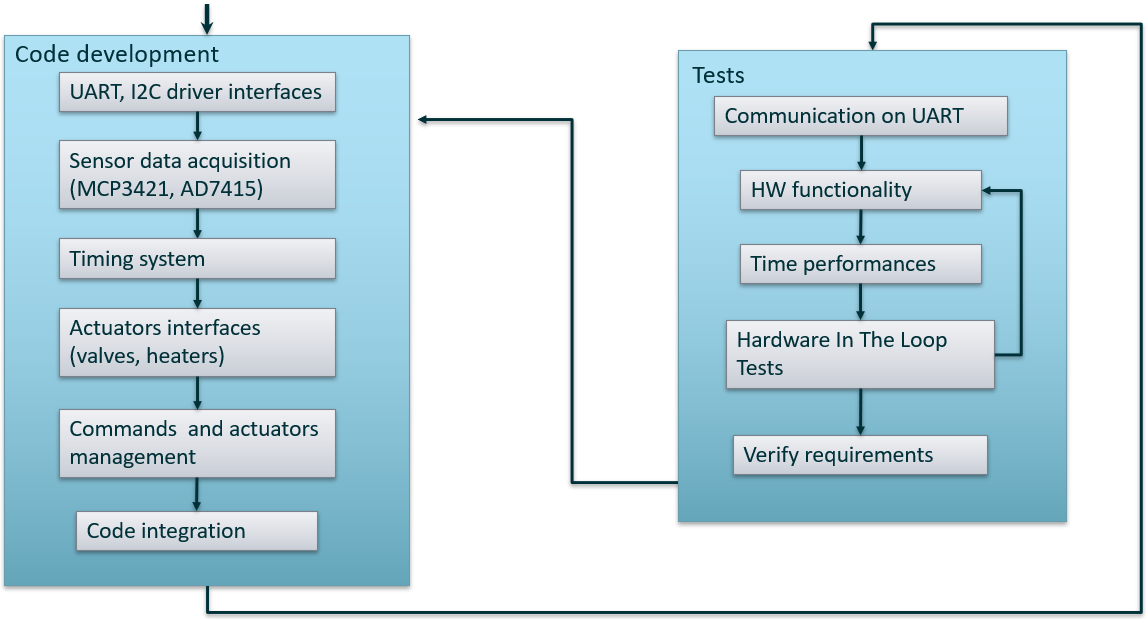


Figure 6: Work logic

2.5.1 UART driver

The preliminary activities on the software development is the setting of all interfaces since they are the basis for the other functions. The first communication interface that is designed is the UART (Universal Asynchronous Receiver-Transmitter), which is a serial asynchronous communication protocol. MSP430 has two UART lines and one of them can be used also for debugging, therefore they are designed as:

- UART0 for the communication with the GNC,
- UART1 as debug line.

At this time, we avoid detailing how each UART driver registers are set to keep from being longwinded since for our purpose we just have to know how the serial communication is configured. Both UART0 and UART1 are configured with these parameters:

- 115200 baud rates,
- 8-bit data
- MSB-first data,
- 1 start bit,
- no parity bit.

To speed up the main loop of the software and to not take up the uC in the communication, an interrupt for the reception is enabled while the transmission is executed with the use of DMA.
Since messages coming from GNC are asynchronous, to read them an interrupt is configured, with its interrupt software routine (ISR) that puts each received bytes in a buffer. Once the buffer size reaches the packet size (that it's fixed size, we present it later) a flag is set and the microcontroller loop, that read the status of that flag once a loop, gets the buffer and processes it.

Meanwhile, the transmission on UART0 is performed by the DMA (Direct Memory Access). It's a mechanism that allows certain peripherals to access memory independent of the CPU, this allows to release the CPU from data transfer.
These two upgrades are not implemented to the debug line since they add enough complexity to the system and the debug line is used only during code development, it will be removed after the final release of the software so it will not affect the final performance of the propulsion system.

UART1 Test

List of items

- Computer with MobaXterm;
- MSPEXP-430FR6989 with its USB-miniUSB cable.

Procedure

- Connect computer and MSPEXP-430 by means of USB cable;
- Open MobaXterm on PC click on 'Session' and launch new session configured on 'Serial connection' on serial port 'COM5 (MSP Debug Interface)' with speed 115200 bps.
- MobaXterm terminal shows a continuous sequence of sample sentences.

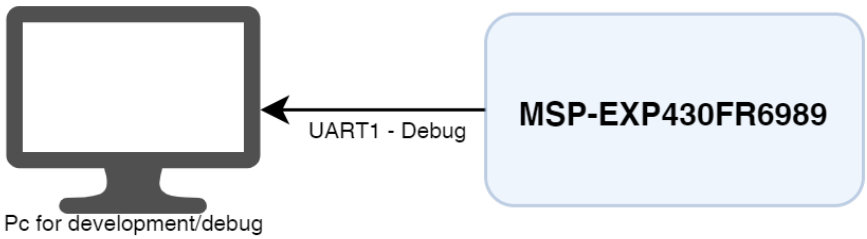


Figure 7: UART1 - Debug line test configuration

Results

UART0, debug line, correctly transmits string of bytes to the PC.

UART0 Test

List of items

- Computer with MobaXterm;
- MSPEXP-430FR6989 with its USB-miniUSB cable;
- Raspberry Pi 3 B+ with its power supply;
- 2 x jumper cable.

Procedure

- Connect computer and MSPEXP-430 by means of USB cable to power the board;
- Open MobaXterm on PC and launch new session configured on 'SSH' with Remote host "192.168.8.41". On the terminal that opens insert the password.
- To launch the application that reads on serial line, type: *python Rasp_serial*
- On the terminal it is possible to see a sequence of numbers that Raspberry Pi sends to MSP430 on UART and then returns increased by 1.

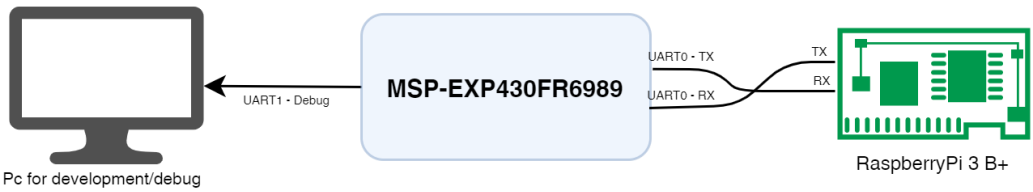


Figure 8: UART0 Test configuration

Results

UART0 correctly receives string of bytes by Raspberry Pi, increases by 1 the sequence of numbers and resent it to RPi that correctly receives the packet.

2.5.2 I2C driver and sensors acquisition

The next step is the development of I2C (Inter-Integrated Circuit) driver and to tests that it's needed at least a slave, therefore a temperature sensor (AD7415[11]) is connected to the bus and also its driver is developed.

I2C is a synchronous, multi-slave serial communication bus, so it allows to connect more than one device on the same lines; at this point only one sensor is connected [Figure 9], subsequently other sensors are added on the line.

Both transmission and reception are implemented with interrupt in order to be able to easily and quickly manage the I2C protocol, the driver on MSP430 is implemented as:

- Master mode,
- 100kHz speed,
- 7-bit device address,
- 8-bit data,
- 1 stop bit,
- 1 start bit.

To acquire data from temperature sensor, four steps shall be performed:

- Initialize the sensor: send to the sensor address a 2-byte data, where first byte is the address of configuration register and the second represent the desired

configuration. In this case the sensor is configured with 10-bit data, one shot conversion and MSB-first. This step is performed once at the power on of the sensor.

- Request value: MSP430 sends to sensors address a byte with the register address where the converted data is stored.
- Read value: the sensor answers to the previously request with the 3 bytes: the first is its address, on the last two there's the 10-bit acquired data.
- Convert the raw data to temperature: apply the following function to the 10-bit received data[11]:

$$Temperature\ [°C] = \frac{digital_{value}}{4}$$

I2C Test

List of items

- Computer with MobaXterm;
- MSPEXP-430FR6989 with its USB-miniUSB cable;
- AD7415 sensor;
- 2x 10kΩ resistors;
- jumper cables.

Procedure

- Connect AD7415 to MSPEXP-430FR6989 as shown in Figure 9;
- Configure the MSP430 debug line as described in section 2.6.1;
- On the terminal it is possible to see a sequence of temperatures read from the sensor.

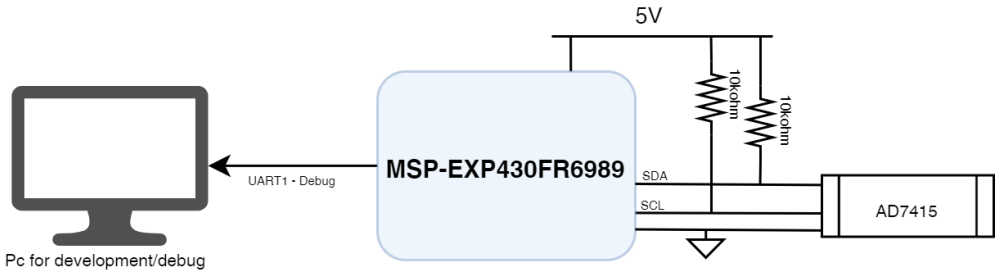


Figure 9: I2C Test configuration

Results

Debug terminal shows the temperatures correctly read from sensors; the values are around 25°C. It is also tested that holding in your hand the sensor to increase the temperature, the values increase up to 35°C.

2.5.3 System timing

One of the most critical aspect of this project is the timing accuracy, since to correctly perform a manoeuvre and not incurring in undesired movements, it is fundamental to open and close the valves with high accuracy and precision. To keep time and easily manage all variable links to time, a timer is implemented.
Timer is a counter that increases its value each hit of the clock, we intend to use it to trigger an interrupt every 1 ms, since the clock on MSP430 is configured at the frequency of 8MHz, the timer is set to count until 8000 before it triggers its ISR.
The timer ISR gets all variables that keeps time (for example, the residual opening times of the valves or timeouts) and decrease by 1 each of them, practically if we set an opening time for a valve in milliseconds, this time decreases every millisecond in the interrupt, regardless of main loop duration.

Timer Test

List of items

- Computer;
- MSPEXP-430FR6989 with its USB-miniUSB cable;
- Oscilloscope with a probe.

Procedure

- Connect MSPEXP-430FR6989 to PC with USB cable just to power the board;
- Connect the probe to pin 4.7 of the MSP430 that is set in order to toggle whenever the timer interrupt triggers;
- The oscilloscope screen shows a square waveform.

Figure 10: Timer Test configuration

Results

The oscilloscope shows a square waveform with a period of 2 ms, since the pin toggles every 1 ms, so to perform a whole period of as square waveform it needs 2 toggles.

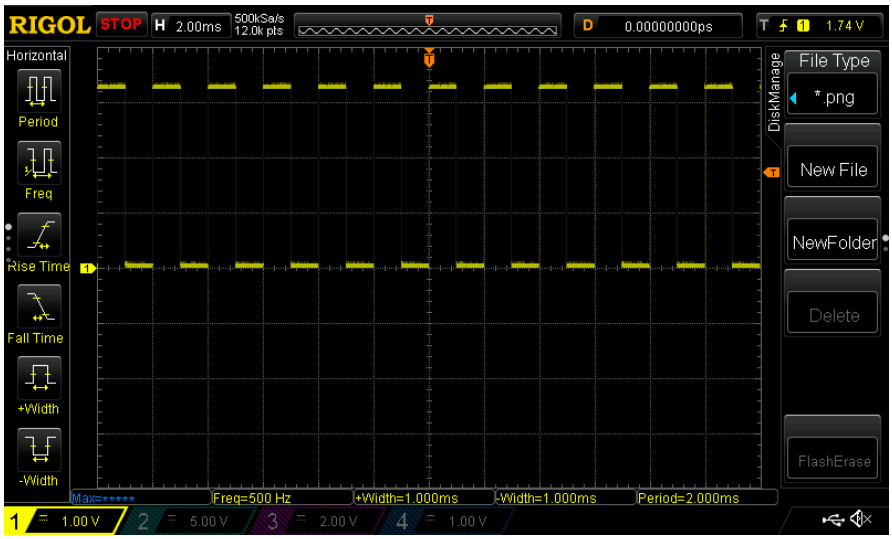


Figure 11: Performance of MSP430 pin that toggle every timer interrupt

2.5.4 Actuators Management

The last interfaces of the microcontroller that are left to develop are the GPIOs used to control the valves and heaters. All actuators are managed with the same method and each of them needs two signals to be switched on:

- an enable signal that stays high to activate the actuator;
- a heart-beat signal with a period of 100 ms;

both signals shall remain active during all the time that the actuator should remain ON, otherwise to turn OFF the actuator, they stay low.
Therefore, there are 24 GPIO pins of MSP430 used to manage the actuator:

- 10 enable signals for the valves;
- 10 heartbeat signals for the valves;
- 2 enable signals for the heaters;
- 2 heartbeat signals for the heaters.

To switch on/off valves and heaters, '*CheckValves*' function and '*CheckHeaters*' function are developed, the logic is the same for both the functions, so we present only the valves control meanwhile the heaters ones is equal.

To each valve is associated with a 32-bit variable that represents its opening time in milliseconds, if this variable is different from 0 it means that the correspondent valve shall be open. Once the opening time is set to a value different from 0, it begins to decrease by 1 every millisecond thanks to the timer interrupt. Therefore, to control the valves behaviour, the '*CheckValves*' function checks every loop for each valve the respective opening time value:

- If opening time is equal to 0 the enable signal and heart-beat signal are set to stay low;

- If opening time is greater than 0 the enable signal is set to stay high, while another timing variable is activated and used to count 50 ms to toggle the heart-beat signal.

The main feature of '*CheckValves*' is that the function is not blocking, this means that microcontroller can continue its operations and every loop checks if it should change the status of enable and heart-beat signals, for this reason it's also important that the loop duration is as fast as it can be.
The opening time variables will be set from GNC command, but, right now, we set them with a simple assignment.

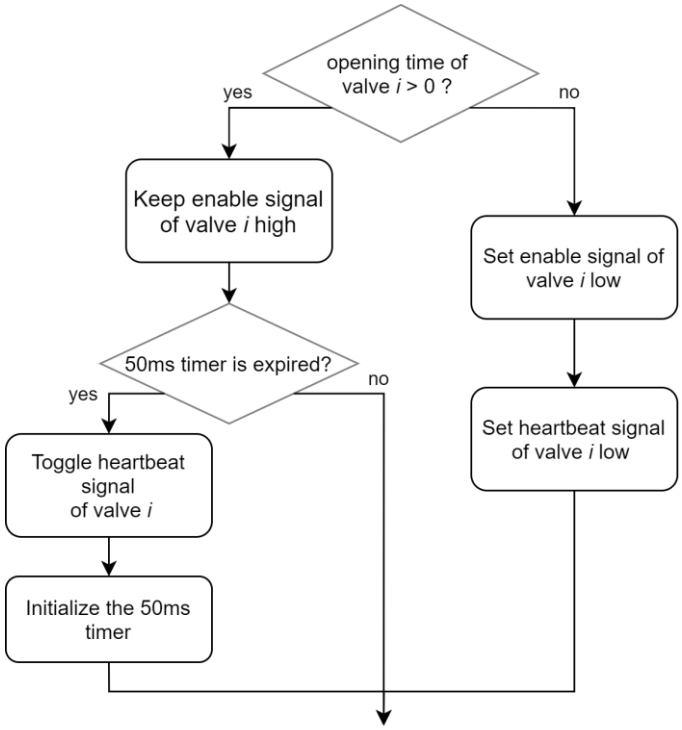


Figure 12: Flow chart of 'CheckValves' function, the algorithm is repeated for all valves

Actuators management Test

List of items

- MSPEXP-430FR6989 with its USB-miniUSB cable;
- Oscilloscope with a probe.
- LEDs, resistors, and jumpers.

Procedure

- Connect to each actuator GPIOs a resistor in series with a LED, to easily recognize the pins different colour of LED are assigned to each of them, therefore also a different resistor value

Actuator	***Signals***	***LED colour***	***Resistor value***
Valve	Enable	green	470 Ω

	Heartbeat	Yellow	1 kΩ
Heater	Enable	Red	1 kΩ
	Heartbeat	Red	1 kΩ

Table 3: Actuators pins: configuration of LED and resistors

- Connect two probes to:
    - Enable signal of first valve,
    - Heart-beat signal of first valve;
- Connect MSPEXP-430FR6989 to PC with USB cable just to power supply the board;
- See on the oscilloscope the enable signal stays high and the heartbeat that toggles every 50 ms, meanwhile the green LEDs are switch on and yellow LEDs quickly blinks.

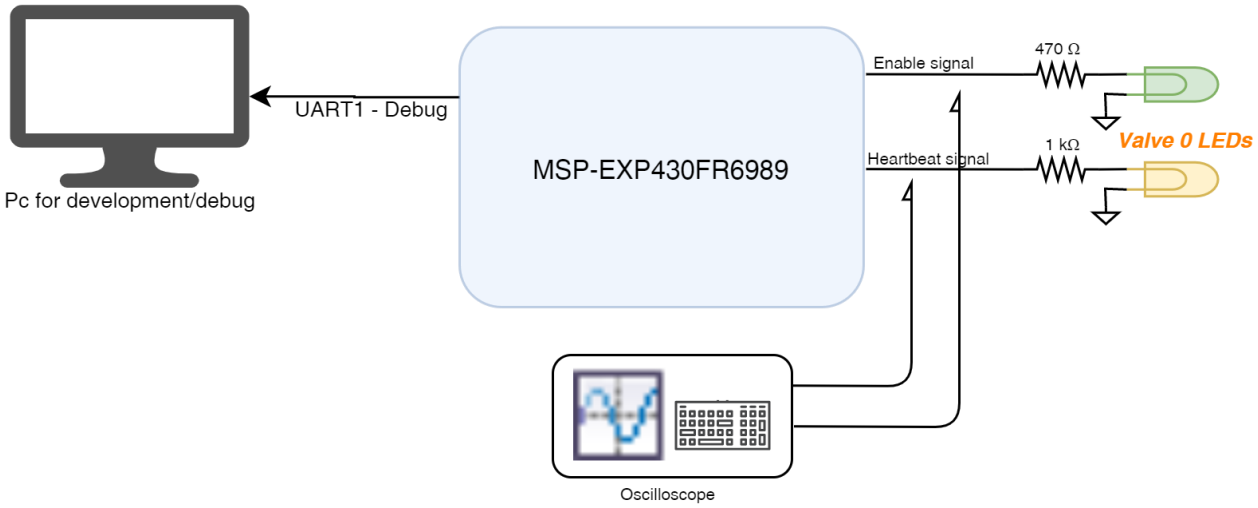


Figure 13: Actuators test configuration for valve 0

Results

The green LED stays on for a certain time, then switches off, the correspondent yellow LED quickly toggles. The oscilloscope proves that the heart-beat period is 100 ms with an accuracy of 2 ms and the enable signals is equal to the time set with an accuracy of 2 ms.

The test is repeated moving the probe on different GPIOs in order to analyse the behaviour of all signals and the delay between the opening time of first valve and the last.

With an opening time set to 450 ms, it's possible to see how the enable signal has a precision of ± 2 ms, and also the heart-beat signal has a semi-period of 50ms ± 1 ms.

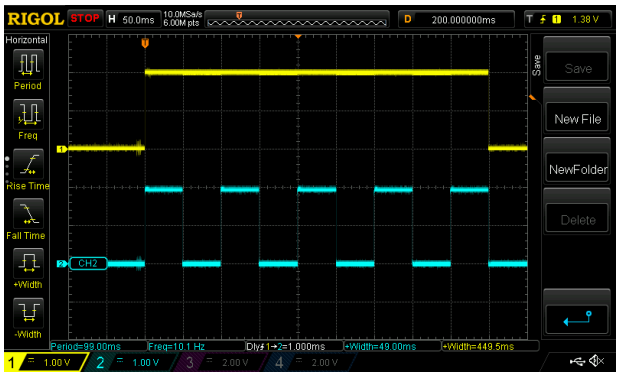
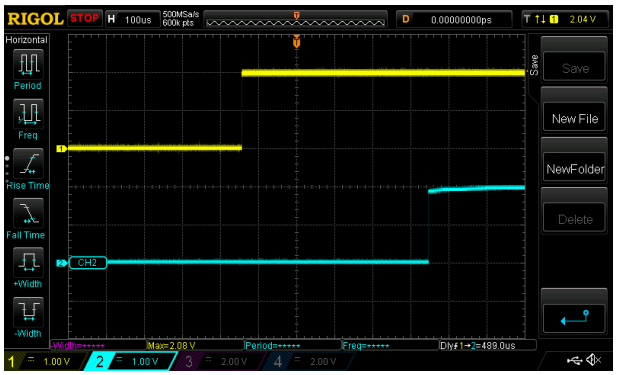


Figure 14: Oscilloscope screen, yellow track is the enable signal of valve 0, blue track is the heart-beat signal of valve 0.



The figure 7 shows the delay in the opening of the valves, the tests is repeated for all valves, but, predictably, the major delay is found between the valve 0 and valve 9 and it's less than 0.5 ms.

Figure 15: Oscilloscope screen that shows the delay in the opening of the valves, yellow track is the enable signal of valve 0, blue track is the enable signal of valve 9.

While figure 8 shows the delay in the closing of the valves, in this case the major delay is found between valve 0 and valve 9 and it is less the 2 ms.

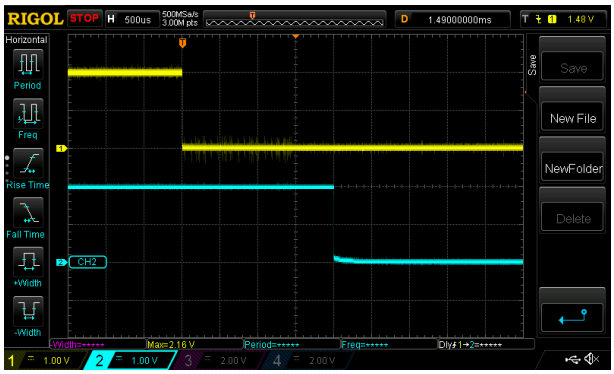


Figure 16: Oscilloscope screen that shows the delay in the closing of the valves, yellow track is the enable signal of valve 0, blue track is the enable signal of valve 9.

The tests are repeated on all valves with different opening times, but the oscilloscope has a limit in the maximum time interval that is able to capture, therefore, at this point there is no data about precision for opening times larger than 1s.

2.5.5 Temperature control algorithm

To guarantee the desired propulsive performances the propulsion module and, in particular, the propellant should reach the optimal conditions in terms of temperature and pressure. For an optimal thrust the propellant should be in gaseous state and has no fluidic parts. To have propellant only in gaseous phase, temperatures and pressures shall be in a specific range. The system does not allow to directly control the pressures, but it's possible to indirectly controlled them acting on the temperatures by means of heaters.
Consequently, a temperature control is essential to perform manoeuvres with optimal performances. The propulsion module has two heaters: one on the tank and the other along the pipes line; each of them has its own temperature control and it is independently controlled.

The implemented temperature control has three different thresholds, since there are not yet requirements about temperatures range, the thresholds are temporarily set as :

- Minimum threshold $27℃$
- Medium threshold $31℃$
- Maximum threshold $35℃$

Then two operative mode are introduced.

- *Pre_firing mode* : fire does not allow
- *Firing* : fire allows

The control algorithm is implemented in order to:

- not allow thrust and keep the heaters on if the temperatures are lower than the minimum threshold.
- Allow thrust and keep the heaters on if the temperatures are between the minimum and the medium threshold.
- Allow thrust and switch off the heaters, when the temperatures reach the maximum threshold.
- Switch on the heaters again if the temperatures go down beyond the medium threshold.
- Block all manoeuvres if the temperatures go under the minimum threshold. (e.g., in case of heaters failure).

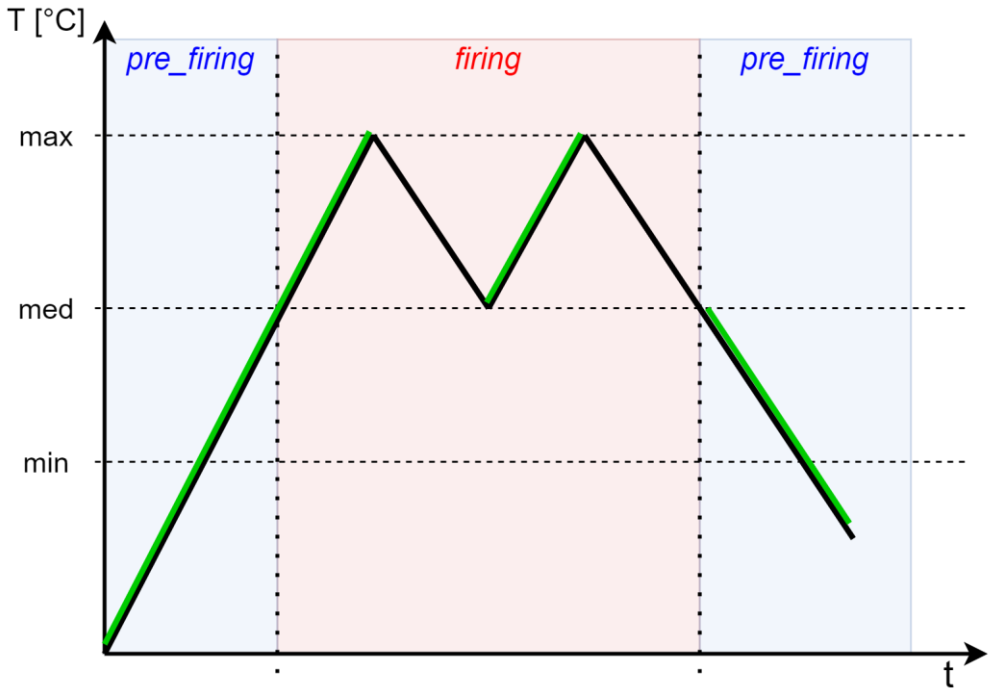


Figure 17 : Graphically representation of temperature control : black line is the temperature behaviour; green line represents when heaters are switched on

2.5.6 GNC communication

All commands and manoeuvres to be performed are decided by GNC processor, to the MSP430 is entrusted only with the management of the propulsion system in terms of acquiring telemetry, checking the system status and managing the actuators in case of a thrust command. Therefore, a communication with GNC is developed to provide a list of commands that allows the management of propulsion system by the flight computer.

The communication between GNC processor and thruster microcontroller is developed in order to be synchronous: the MSP430 sends a message to GNC only if the GNC has previously sent commands or requested some data. The communication uses a packet with a fixed length even if the data field changes depending on the command. The packet is structured as shown in Table 4, where the data field is defined as 'union'.

The union data type is used to store, at different instants, objects of different sizes and types that have in common the role within the program. Memory is allocated for the largest of the variables, since they can never be used simultaneously (the choice of one automatically excludes the others), sharing the same memory space. Therefore, even if the data field in the packet changes depending on the command, the size of that field remains equal to the largest possible data that can be inserted there, in this case is equal to 69 bytes due to the data structure of *Telemetry Answer*.

GNC_packet			
Variable type	***Number of bytes***	***Name***	***Comment***
uint8_t	*1*	src	message sender
uint8_t	*1*	dest	message receiver
uint8_t	*1*	msg_id	message identification number
uint8_t	*1*	cmd	command identifier
union GNC_data	*69*	data	data field, constant size independent from cmd
uint8_t	*1*	crc	crc-8

Table 4: GNC_packet structure

Since this communication is on a serial line, that by protocol definition does not use an address, PS microcontroller and the GNC processor do not have an assigned address by default, therefore an identification number is assigned to each of them:

- 51 to the thruster microcontroller;
- 61 to the GNC processor.

These values are used in the *src* and *dest* field, for example if a packet is sent from MSP430 to the GNC the *src* is set to 51 and the *dest* is set to 61, while if the sender is the GNC the *src* is 61 and the *dest* is 51.

Since it is a synchronous communication where the microcontroller responds only if it is interrogated by the microprocessor, when the MSP430 answers to the GNC it uses as msg_id the same msg_id belonging to the command to which it's responding to.

The *cmd* field is a number that identifies the command, the term 'command' when it's used to refer to the field of the packet, is used to identify indifferently a request, an answer or an effective command.
The list of commands and requests that the GNC can send to the propulsion system is reported in Table 5, while the possible answers of the microcontroller are reported in Table 6.

The CRC, Cyclic Redundancy Check, is an error-detecting code commonly used in digital networks to detect accidental changes to raw data during transmission. The main purpose of the CRC is to allow the detection of errors able to compromise the integrity of the data, thus acting as a validity check associated with a specific data structure.

Name	***Id number***	***Data***	***Comment***
Shut down command	10	No data needed	Command to switch off all actuators (valves and heaters)
Reboot command	15	No data needed	Command to do BOR of the system (Brownout reset)
Pre-Firing command	17	No data needed	Switch system to *pre_firing* operative mode
Change-Threshold	18	New temperature thresholds	Change temperature threshold for heater management
Timing all valves command	21	Opening time of each valves	Command to open/close independently all valves. Set for each valves the opening time. The opening time shall be up to 600 s. If one of the opening times is overrange, the command is not valid and deleted. New valid command overrides the past command.
Timing all heaters command	23	Turn-on time of each heaters	Command to switch on/off independently all heaters for a certain amount of time
Telemetry request	80	No data needed	GNC requires telemetry from propulsion system
Valves opening times request	81	No data needed	GNC requires residual opening times of valves
Heaters opening times request	82	No data needed	GNC requires residual opening times of heaters
Check firing mode	83	No data needed	GNC checks if the propulsion system is in firing mode

Table 5: List of commands that GNC can send to propulsion system

Name	*Id number*	*Data*	*Comment*
Answer ACK	41	No data needed	Sent by propulsion system when the received command is executed
Answer NACK	42	No data needed	Sent by propulsion system when there is some issue, and the last command is not executed
Answer telemetry	43	PS status and telemetries	Telemetry message from propulsion system to GNC
Answer valves opening times	44	Residual opening times	Propulsion system sends to GNC residual opening times of valves
Answer heaters opening times	45	Residual turn-on times	Propulsion system sends to GNC residual opening times of heaters

Table 6: List of answers that propulsion system cans send to GNC

It's possible to see how some commands do not need any parameters in the *data* field, so an example is shown in Table 6 : the command can be sent from GNC to propulsion system therefore the *src* is 61, *dest* is 51, *msg_id* is randomly chosen as 1, *cmd* is the id number corresponding to '*Shut Down command*' , the *data* field is empty, but it allocates however 69 bytes, crc is computed by crc8 function.

Shut Down command					
src	*dest*	*msg_id*	*cmd*	*data*	*crc*
61	51	*1*	10	-	*85*

Table 7: Example of Shut Down command

While there are some commands that need some parameters, so specific *data* structures are defined for each of them. An example is the *Timing_all_valves* command that switches on the valves: it needs as parameter the opening times for each valve. Since the time is expressed in milliseconds and, from requirements, the maximum continuously opening time is 600000 ms, the minimum size on which it's possible to represent these variables is 4 bytes variable; the *data* field is structured:

Field	*Variable type*	*Variable name*
data	uint32_t	opening time of valve 0
	uint32_t	opening time of valve 1
	uint32_t	opening time of valve 2
	uint32_t	opening time of valve 3
	uint32_t	opening time of valve 4
	uint32_t	opening time of valve 5
	uint32_t	opening time of valve 6
	uint32_t	opening time of valve 7
	uint32_t	opening time of valve 8
	uint32_t	opening time of valve 9

Table 8: Structure of data field of Timing_all_valves command

The specific structure and possible PS answers of each command are reported in Appendix C.

2.5.7 Main function - µC Operative Mode

At this point, that all single modules are developed, the next step is the development of the main function that should manage the propulsion system and all its functionality.
To carry out a manoeuvre, the two steps to accomplish are: one to bring the PS to an optimal condition to generate the thrust, then to generate the thrust by opening the valves. To easily manage these operations the PS algorithm is implemented with three different operating modes:

- ***keep_alive*** : PS is in this mode when no manoeuvre should be performed, the microcontroller communicates with the GNC, acquires measurements from one different sensor per loop, checks the status of the valves and heaters, which shall be off since it is not possible to turn them on while system is in this mode;
- ***pre_firing*** : system passed in this mode if an explicit command of change status or heaters switch-on command arrived; in this state the heaters can be switched on to bring the system into firing conditions in terms of temperature and pressure. In addition to the *keep_alive* loop, there is a temperature control algorithm that individually switches on the heaters if the tank or pipes temperatures is out of range for firing. Once both tank and pipes reach the necessary conditions the system automatically moves to the firing operating mode;
- ***firing*** : in this status it is possible to open the valves and perform manoeuvres. The system continuously checks its status and control the heaters in order to maintain the optimal temperatures for firing. If the conditions are no longer met, for example for too low temperatures and heater failure, any operation on the valves is interrupted and the system returns to pre_firing mode. It is possible to return to keep_alive mode from all other states by sending a command that shuts down all the actuators, valves and heaters.

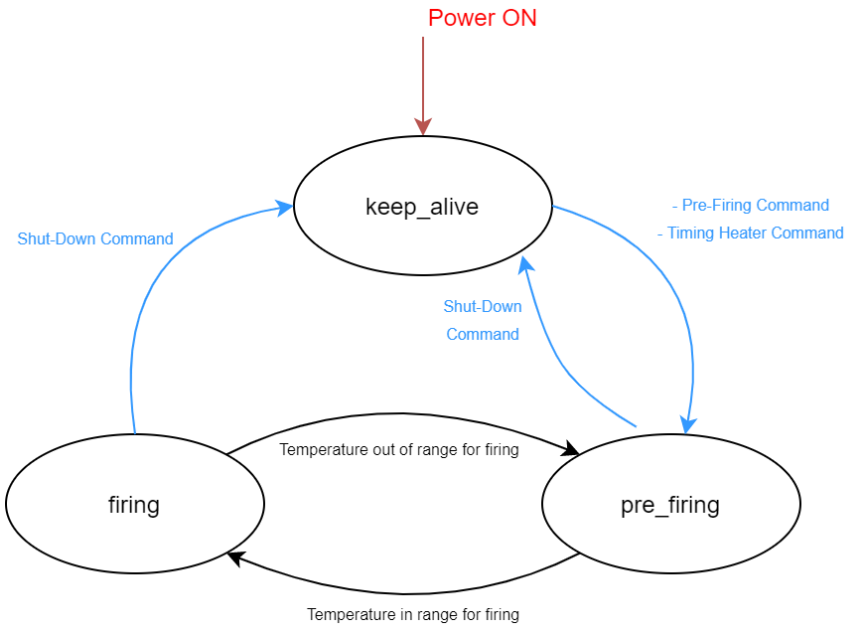


Figure 18: µC operative mode

As it's said before, one of the main critical aspect of this project is the need to have a fast loop in order to control the actuators status with the best possible time resolution, so the goal is to have a loop as short as it can be. With this in mind, considering sixteen sensors (three pressure sensors, three temperature sensors, five voltage sensors, five current sensors), the main function is implemented in order to acquire only one sensor measurement per loop, meanwhile the main loop lasts less than 5 ms and, consequently however in a time interval less than 80 ms all sensors are acquired.

Since the most expensive loops occur when a command should be processed and executed, another trick is used to reduce the loop time. It is based on the algorithm which makes the propulsion system send the answer to GNC in the next loop with respect to the loop in which it receives and process the command. Before the implementation of the use of DMA on UART communication, this method allowed to reduce the loop time by about 7 milliseconds compared to a loop that lasts 5 ms if there is no communication on UART. With the use of DMA, the transmission to GNC is not yet entrusted to the processor therefore it does not affect the loop time if not for a few lines of code.

A more detailed algorithm for each operative mode is represented into Figure 19.

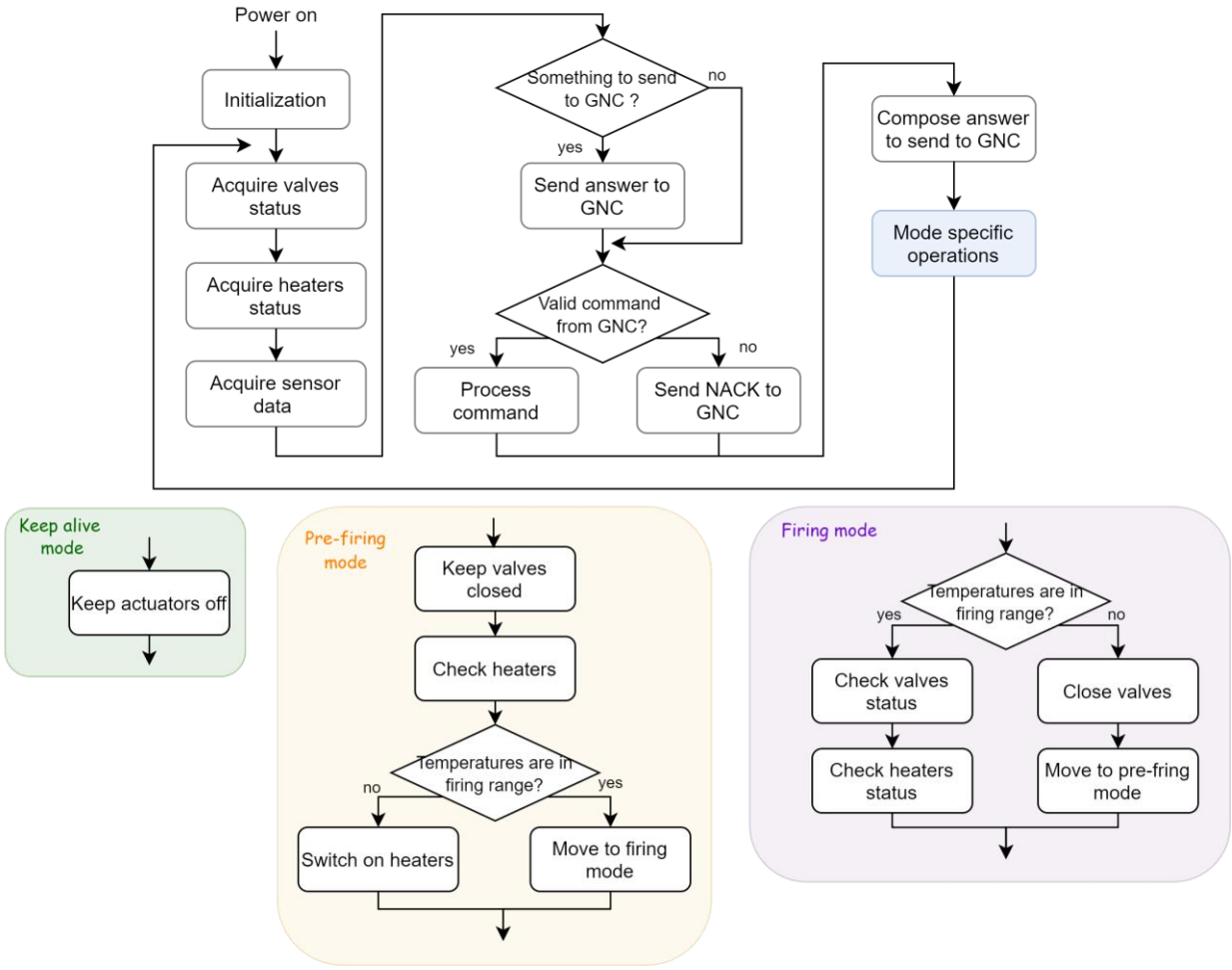


Figure 19: Flow Chart of propulsion system in the three different operative mode

3 Software Test

At this point, all software has been developed and assessed with the help of the oscilloscope and the debug line, the next step is to test the time performance of the software in the management of the actuators. The oscilloscope allows to have an excellent precision, in terms of time, on the control of the pins, but it can verify the behaviour up to 4 pins at the same time, while the inspection of the LEDs of the development board allows to control the general functioning of all the pins together but does not allow precise time control. So, to test all the pins with a good precision in terms of time, the system has been inserted into a sort of software in the loop test where the object under test is the propulsion system software and the Raspberry Pi acts as a tester, which communicates with the MSP430 and reads the status of all its pins. In this way it's possible to automatically check the behaviour of all pins at the same time and have a good time accuracy, moreover it's possible to perform tests over time in order to verify on a large number of tests that no errors occur.

The software tests performed and analysed in this chapter have the aims to verify the management of the commands, to test the functionality and time performances of the software functions that manage valves and heaters, and the correctness of the telemetry data.
Therefore, six different tests are performed, the goals of each of them are the following:

- Test 1: aims to check the operation of all actuators pins;
- Test 2a-b: aims to test the time performance of uC in actuators management;
- Test 3a-b: aims to test time performance when more than one actuator is switched-on at the same time;
- Test 4: aims to test Shut Down and Reboot commands;
- Test 5: aims to test Telemetry Request, Valves Time Request and Heaters Time Request;
- Test 6: aims to test the Change Threshold commands.

3.1 Set up

3.1.1 Hardware configuration

To assemble these bench tests, it's necessary to adequately set up the Raspberry Pi, which has two main tasks: send commands to propulsion system and read the status of the actuators pins. For the first task the RPi needs the serial communication that is already implemented and tested both from the point of view of hardware and software during the development of UART0. Meanwhile for the second task, each MSP430 pin that should be monitored is connected to a RPi's pin with a jumper cable and a specific algorithm is developed for the acquisition of all 24 pins status as fast as possible. All the software developed for the RPi is described in the Section 3.1.1, while the hardware configuration keeps the development board used for the software development in Section 2.4 and adds a cable to connect each LED to a RPi GPIOs [Figure 20]. MSP430 has a pin for each signal that is used to control an actuator already connected to a LED; each of these pins, thanks to a breadboard, is also connected to a Raspberry Pi GPIO.

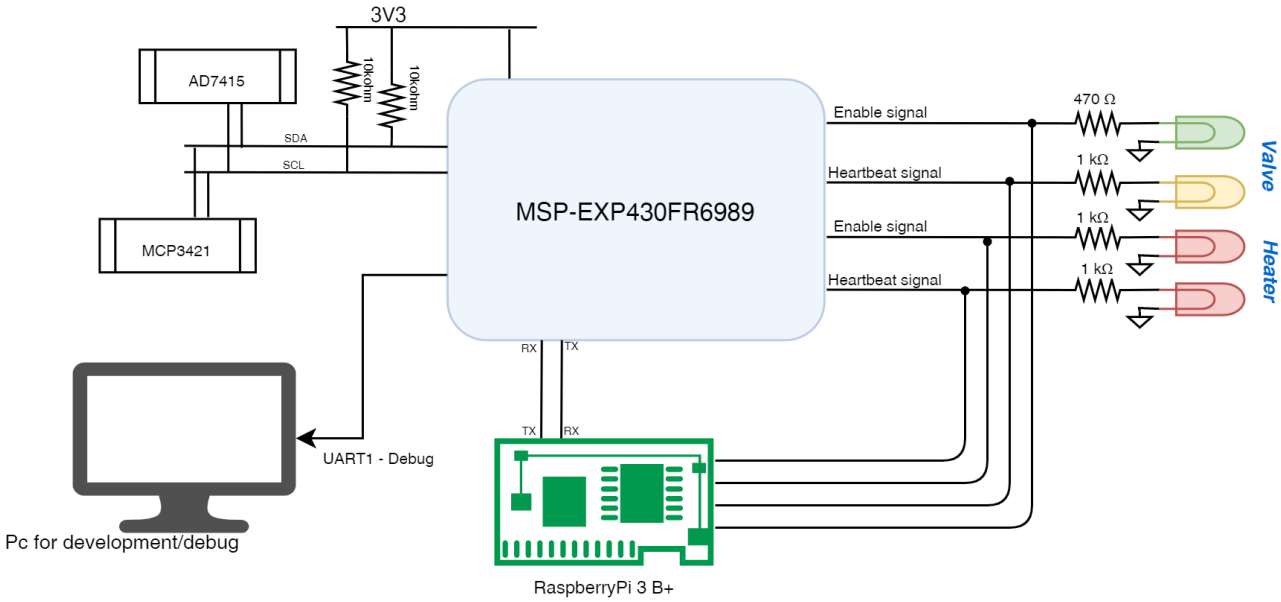


Figure 20: Simplify block scheme of SW tests configuration

3.1.2 Raspberry Pi software

The six tests have different objectives but each of them uses the same logic: send a command, wait for the PS answer and check the status of the actuators [Figure 21], therefore a common function for all tests is developed and then a specific algorithm is developed for each test. The individually flow chart of each tests are analysed in the following sections.

The Raspberry Pi software is developed in C++ language and as simple as possible, its aim is to act as the GNC simulator and to test the propulsion system software, so its three main tasks are:

1. Send commands to thruster on UART;

2. Read microcontroller GPIOs status;
3. Read microcontroller answer on UART.

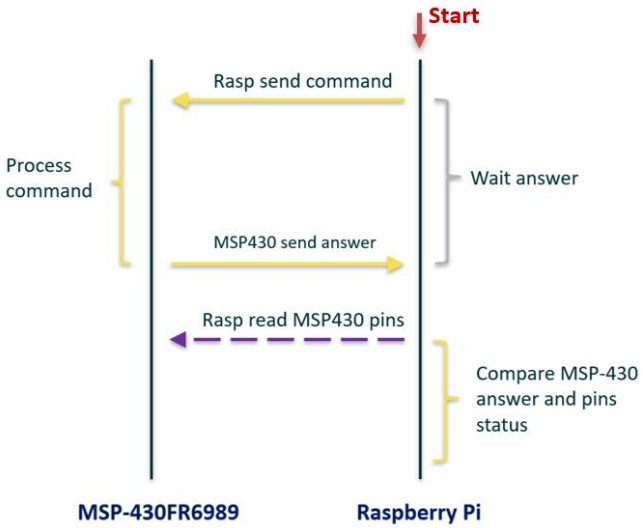


Figure 21: Software tests timeline logic

Therefore, the software is divided into 3 threads:

- The first thread manages the serial transmission to the propulsion system, so it offers high-level functions to other threads to compose packets and send them. The thread has also a function that allows to send pre-set commands, four different commands are implemented:
  - 1 – Shut Down command
  - 2 – Timing All Valves command with times of 2000 ms for each valve
  - 3 – Timing All Heaters command with times of 10000 ms for each heater
  - 4 – Pre-Firing command

  These commands are used to individually test the operation of a single command or to stop the execution of a test. For example, if before starting the tests, one wants to check that the settings of the Raspberry Pi terminal (which will be explained later) are correctly set, one can send one of the four commands and see if the MSP430 receives it correctly. To send one of these commands one can just type the respective number on the Raspberry Pi terminal and press Enter.
- The second thread manages the data reception: if some data arrived on the serial line a buffer is filled, and once it reaches the packet length, it's checked for the validity of the crc, the source, the destination; it then makes the buffer available to other threads by means of high-level functions;
- The third thread is the main thread and manages the execution of the tests: it includes the algorithms of each of the six different tests and allows to select which test should be performed. It's also the thread that is tasked to read the actuators status; the algorithm used to read the actuator status is described in Section 3.1.1.

3.1.3 'Read_GPIOs_status" function

One of the most critical functions that is developed for the GNC simulator is the 'Read_GPIOs_status' function. The criticality is the need to take the status of 24 GPIOs at the same time. If a "Timing_all_valves" command is sent, we want to read and check the status of all valves - not only four. In addition, we want to check that the heartbeat toggles every 50ms if the enable signal is read high.
To do that a state machine for each valve is implemented, all state machines move between their state independently by others:

- *Start :* initialize the timeout and move to *wait_for_high* state.
- *Wait_for_high :* read the enable signal and wait until it's equal to high, then if it also reads that heartbeat has changed its status respect to *start* status, moves to *wait_for_low*.
- *Wait_for_low :* read the enable signal and, until it does not return to low, thanks to some timers, checks that heartbeat toggles every 50 ms (with a certain tolerance). When the enable signal is low, checks that also heartbeat is returned to 0 and moves to *finish* status.
- *Finish* .

Thanks to an external while cycle the algorithm remains in loop on this state machines until all machines moves to *finish*. Also, in order to not block the system in case of errors or just in the probable event in which not all valves are open, a series of different timeouts are implemented.

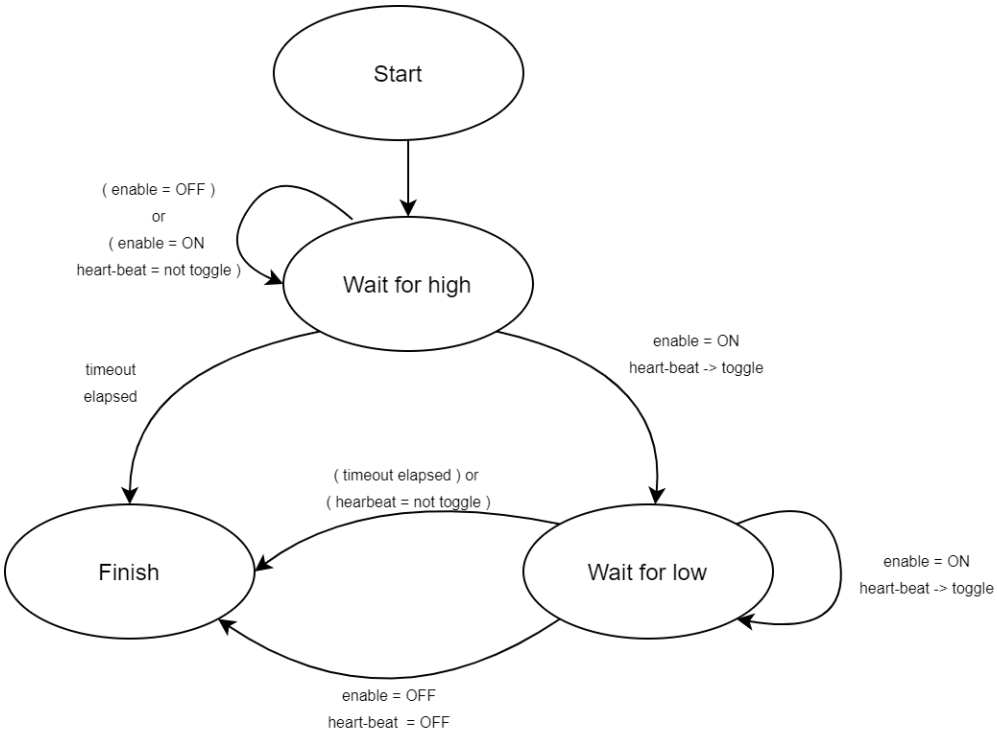


Figure 22: State machine of control logic

The function is thus implemented and tested on *"Timing_all_valves"* command with different opening times. It's noticed that, despite the propulsion system having a good accuracy (± 2 ms as

demonstrated in section 2.6.4 with the oscilloscope), the GNC simulator has a maximum accuracy of 50 ms, this means that the Raspberry Pi is not able to read potential GPIOs status changes that last less than 50 ms. Furthermore, during the tests, it is also shown how the inaccuracy increases with the increase of actuators opening times. While the 50 ms tolerance is due to the Raspberry Pi process, the increasing tolerance linked to opening times, is attributable to the MSP430 whose timer loses 5us every 1ms.

Therefore, all tests done with this algorithm have a tolerance of 50 ms + 0.5% of set time.
It is considered acceptable because on short opening times, the efficiency and precision of the microcontroller have been demonstrated with the oscilloscope, while very long valve opening times will be linked to "escape" operations from the target or from the orbit, therefore they won't be precision maneuvers, but removal maneuvers that subsequently require small rearrangement maneuvers.
Therefore, all the tests carried out will have a tolerance set as 50ms ± 0.5% of the set value, so all the values read that fall within that range of values will be considered acceptable for the correct execution of the test.

3.1.4 Preliminary Procedure

Items:

- MSP-EXP430FR6989 with its USB-miniUSB cable;
- Development board used in section 2.6.4;
- Raspberry Pi 3 model B+ with its power supply cable;
- PC with MobaXterm;
- Jumper cables.

Procedure:

- Connect the MSPEXP-430FR6989 to the breadboard with the LEDs and resistors used in section 2.6.4.
- Connect MSPEXP-430FR6989 to the PC with its USB cable.
- Open MobaXterm, click on "New Session", then select "Serial". In the field "Serial port" choose "COM 5 (MSP Debug Interface)" in the drop-down menu select 115200 bps as "Speed", finally click on the OK button at the bottom. A terminal is open, it shows the debug line of the MSP430, in fact it prints all message that microcontroller sends on the UART-A0; it's also possible to write on that terminal to send messages on UART-A0 line to the MSP430, but the microcontroller is not programmed to receive data on that bus, so if something is written on the terminal the microcontroller blocks and requires reboot.
- Connect the Raspberry Pi as described in Section 3.1 .
- Connect each actuators pin to a RPi's GPIO.
- Switch on the Raspberry Pi.

- On MobaXterm open a second terminal clicking on "+" near the first terminal name, select "SSH" and fill in as follows:
    - Remote host : 192.168.8.41
    - Specify username : pi
    - Port : 22
- A new terminal is opened, and a password is required, so insert "raspberry".
- On the RPi terminal type: '*cd martina/GNC/*'
- Subsequently, each test has its specific configuration.

Since, at this point, there aren't real actuators connected to the board and there is only one temperature sensor, therefore the temperature is not actively increased with the heaters. So to perform the software test before each test the temperature threshold is changed with the '*Change Threshold*' command in order to have the firing range at the ambient temperature, otherwise the system will never move from *pre_firing* mode.
The temperature thresholds are set as:

- Max temperature = 40°C
- Med temperature = 20°C
- Min temperature = 10°C

3.2 Test 1

Goal :

verify the correct operations of the functions that manage valves and heaters and to prove the correct wiring between RPi and MSP430.

The tests switch on one actuator at time and expect that the propulsion system sends a positive answer and activates both enable and heartbeat signal only for the selected actuator while keeping other pins off.

Procedure :

- Configure tests as described in section 3.2
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test1*

Algorithm:

1- *'Shut Down command'* : in order to report system to *keep_alive* mode and switch off all actuators in the event that someone is on;

2- '*Pre-Firing command'* : to move the system to *pre_firing* mode;

3- *Check Firing Mode command'* : before sending the command to switch on the valves, the algorithm waits for the thruster to have the right temperature and to be in *firing* mode; if the answer is *NACK* the tests waits 1 second then resends the

command. Since, at this point, the temperature thresholds are set in order to always allow the valves opening, this command should never receive a NACK.

4- '*Timing all valves command'* with 2000 ms as opening times for the first valves and 0 ms for others.

5- Reads the twenty pins controlling the valves with the 'Read_GPIOs_status' described in section 3.1

6- Wait for the MSP430 answer and check the validity of the received packet, while the third thread read pins status, the second thread has read eventually the microcontroller answer.

7- If the answer is ACK check that all times read in step 5 are 0 except for the time of valve 1, which will be equal to 2000 ms ± tolerance, then the algorithm returns PASS.
If the answer is NACK and all times read in step 5 are 0 (± tolerance) the algorithm returns a WARNING message, while if the one read time is different than 0 it returns an ERROR message, in both case the test returns FAIL.
If the answer is different from ACK or NACK, the algorithm returns FAIL.

8- Return to step 3 but testing the next valves. Repeat until all ten valves are tested

9- *'Shut Down command'* ;

10- '*Timing all heaters command'* with 10000 ms as opening times for the first heaters and 0 ms for the other.

11- Reads the four pins of the heaters with the 'Read_GPIOs_status' described in section 3.1

12- Wait for the MSP430 response;

13- If the answer is ACK check that the times read in step 12 are equal to 10000 ms ± tolerance for the first heater and 0 ms for the second, then the algorithm returns PASS.
If the answer is not ACK or the read time is not consistent with the set time, the algorithm returns FAIL.

Pass/fail criteria:

the test is passed if for all partial tests only the selected actuators are switched on, while the other remain off. An actuator is considered on if the RPi reads the enable signal as high and verifies a 50ms-toggle of its heartbeat signal.

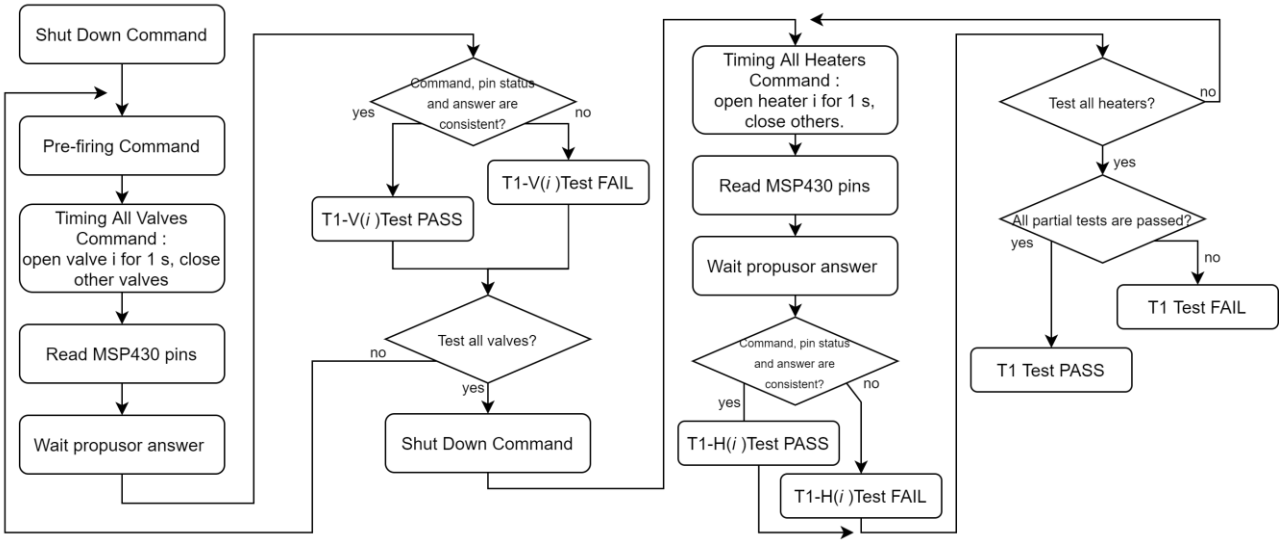


Figure 23: Flow chart of software test 1

Test Results:

All commands are correctly executed, and the read opening times are coherent with the set opening times. The report produced by the Raspberry Pi is attached in Appendix C

3.3 Test 2

Goal :

verify the time performance of microcontroller in the management of actuators pins and analyse the time performance setting different opening times.

Comment :

The tests switch on one actuator at time with a certain time and expect that the propulsion system switches on only the selected actuators pins for the set time while keeping other pins off. The test is repeated with different opening times in order to analyse if there are some differences in the performance with the increasing of time. Therefore, the values with which the tests are repeated are:

0 ms 100 ms 1000 ms 600000 ms

Where 600s is the maximum opening time from the requirements.
Since the test is repeated for all the actuators with different time intervals, it takes a significant time to be performed, so it is divided into two parts:

- Test 2a : on valves;
- Test 2b : on heaters.

3.3.1 Test 2a

Procedure :

- Configure tests as described in section 3.2
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test2a*

Algorithm:

1- *'Shut Down command'* : in order to report system to *keep_alive* mode and switch off all actuators in the event that someone is on;

2- '*Pre-Firing command'* : to move the system to *pre_firing* mode;

3- '*Check Firing Mode command'* : before sending the command to switch on the valves, the algorithm waits for the thruster to have the right temperature and to be in *firing* mode; if the answer is *NACK* the tests waits 1 second then resends the command. Since, at this point, the temperature thresholds are set in order to always allow the valves opening, this command should never receive a NACK.

4- '*Timing all valves command'* with 10 ms as opening times for the first valves and 0 ms for others.

5- Reads the twenty pins controlling the valves with the 'Read_GPIOs_status' described in section 3.1

6- Wait for the MSP430 answer and check the validity of the received packet: while the third thread read pins status, the second thread has read eventually the microcontroller answer.

7- If the answer is ACK check that all times read in step 5 are 0 except for the time of valve 1, which will be equal to 10 ms ± tolerance, then the algorithm returns PASS. If the answer is NACK and all times read in step 5 are 0 (± tolerance) the algorithm returns a WARNING message, while if the one read time is different than 0 it returns an ERROR message, in both case the test returns FAIL. If the answer is different from ACK or NACK, the algorithm returns FAIL.

8- Return to step 3 but testing the next valves. Repeat until all ten valves are tested

9- Return to step 3 but changing the set time. Repeat until all values listed before are tested.

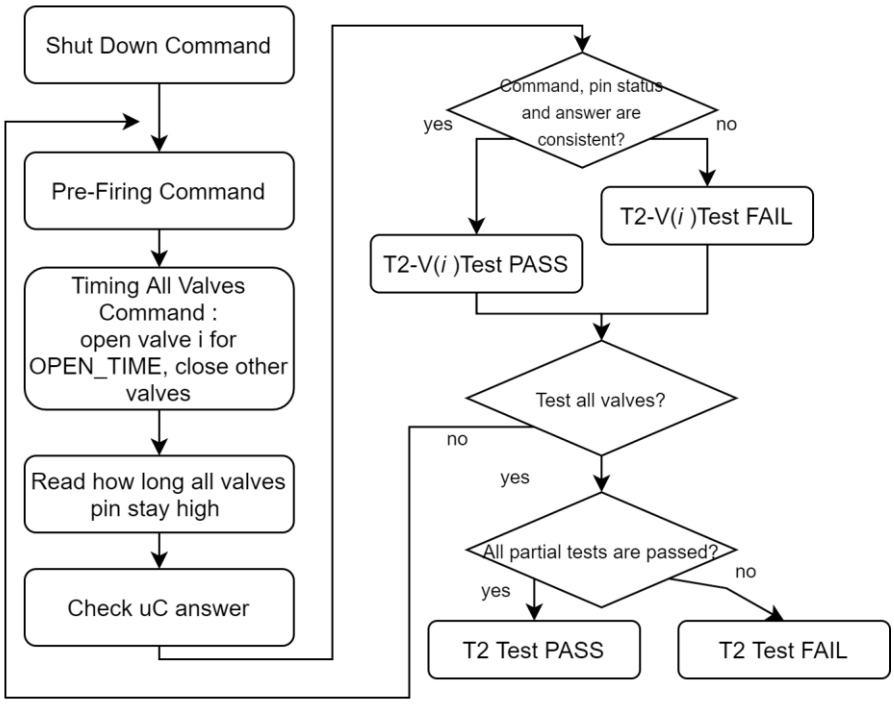


Figure 24: Flow chart of software test 2

Pass/fail criteria :

The test is passed if all partial tests switch on only the selected valves with an opening time in a time range of : set time ± tolerance, where the tolerance is equal to 50 ms ± 0.5% of the set time.

Test Results:

For all partial tests the read time values are less than the set opening times, but they are in the tolerance range, therefore the software time performances are acceptable, and the test is passed. The report produced by the RPi is attached in Appendix D .

3.3.2 Test 2b

Procedure :

- Configure tests as described in section 3.2
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test2b*

Algorithm:

1- *'Shut Down command'* : in order to report system to *keep_alive* mode and switch off all actuators in the event that someone is on;

2- '*Timing all heaters command'* with 10 ms as opening times for the first heater and 0 ms for the second.

3- Reads the pins controlling the heaters with the 'Read_GPIOs_status' described in section 3.1

4- Wait for the MSP430 answer and check the validity of the received packet.

5- If the answer is ACK check that all times read in step 3 are 0 except for the time of heater 1 that will be equal to 10 ms ± tolerance; then the algorithm returns PASS. If the answer is different from ACK, the algorithm returns FAIL.

6- Return to step 3 but testing the second heater.

7- Return to step 3 but changing the set time. Repeat until all switching-on times listed before are tested.

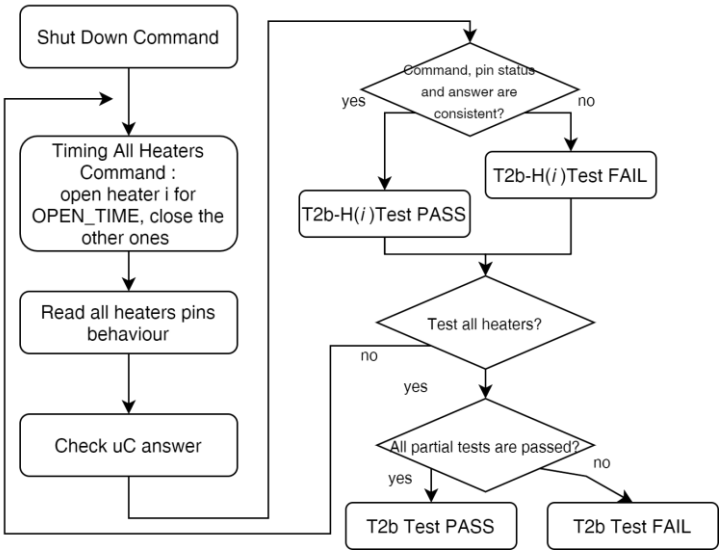


Figure 25: Test 2b flow chart

Pass/fail criteria :

The test is passed if all partial tests switch on only the selected heaters with an opening time in a time range of : set time ± tolerance, where the tolerance is equal to 50 ms + 0.5% of the set time, otherwise the test returns FAIL.

Test Results:

For both partial tests the read time values are less than the set switch-on times, but they are in the tolerance range, therefore the software time performances for the managing of heaters are acceptable and the test is passed. The report produced by the RPi is attached in Appendix D.

3.4 Test 3

Goal:

verify that the time performance of the microcontroller in the management of actuators pins does not change if more than one valve is open at the same time.

Comment:

The tests switch on all actuators at the same time for a certain time and expect that the propulsion system switches on all pins for the set time. The test is repeated with different opening times in order to analyse if there are some differences in the performance with the increasing of time.

The test is therefore performed with a range of opening time between 0 and 600000 ms, the maximum opening time from the requirements [PER-TEC-25], the intermediate values are taken according to a logarithmic scale, therefore the values with which the tests are repeated are:

0 ms	10 ms	20 ms	50 ms	100 ms	200 ms
500 ms	1000 ms	2000 ms	5000 ms	10000 ms	20000 ms
50000 ms	100000 ms	200000 ms	500000 ms	600000 ms	

Since the test is repeated for all the opening time listed before, it takes a significant time to be performed, so it is divided into two parts:

- Test 3a : for test valves;
- Test 3b : for test heaters.

3.4.1 Test 3a

Procedure :

- Configure tests as described in section 3.2
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test3a*

Algorithm:

1- *'Shut Down command'* : in order to report system to *keep_alive* mode and switch off all actuators in the event that someone is on;

2- '*Pre-Firing command'* : to move the system to *pre_firing* mode;

3- '*Check Firing Mode command'* : before sending the command to switch on the valves, the algorithm waits for the thruster to have the right temperature and to be in *firing* mode; if the answer is *NACK* the test waits 1 second then resends the command. Since, at this point, the temperature thresholds are set in order to always allow the valves opening, this command should never receive a NACK.

4- '*Timing all valves command'* with 10 ms as opening times for all valves.

5- Reads the twenty pins controlling the valves with the 'Read_GPIOs_status' described in section 3.1

6- Wait for the MSP430 answer and check the validity of the received packet.

7- If the answer is ACK and all times read in step 5 are equal to 10 ms ± tolerance, then the algorithm returns PASS.

If the answer is NACK and all times read in step 5 are 0 (± tolerance) the algorithm returns a WARNING message, while if the one read time is different than 0 it returns an ERROR message, in both case the test returns FAIL. If the answer is different from ACK or NACK, the algorithm returns FAIL.

8- Return to step 3 and repeat with different opening time. Repeat until all opening times listed before are tested.

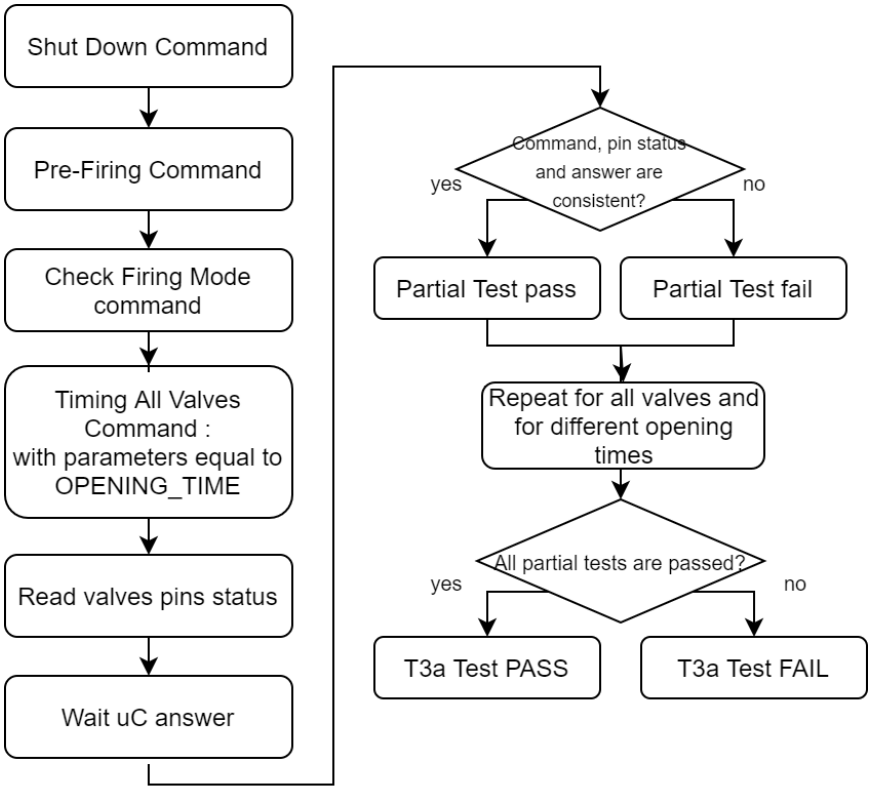


Figure 26: Flow chart of software test 3a

Pass/fail criteria :

The test is passed if all partial tests with different opening time switch on all valves with an opening time in a time range of : set time ± tolerance, where the tolerance is equal to 50 ms + 0.5% of the set time, otherwise the test is failed.

Test Results:

For all partial tests the read time values are less than the set switch-on times but they are in the tolerance range, therefore the software for the managing of valves does not change its time performance if more than one valve are opened at the same time. Therefore, the test is passed. The following table shows the read value for the whole test while the report produced by the RPi is attached in Appendix D.

Set value [ms]	***Read value***									
	Valve 0	***Valve 1***	***Valve 2***	***Valve 3***	***Valve 4***	***Valve 5***	***Valve 6***	***Valve 7***	***Valve 8***	***Valve 9***
0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

20	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
100	54	54	54	54	54	54	54	54	54	54
200	162	162	162	162	162	162	161	161	161	161
500	450	450	450	450	451	451	451	451	451	451
1000	942	944	944	944	944	944	944	944	944	944
2000	1957	1957	1957	1957	1957	1956	1956	1956	1956	1957
5000	4940	4940	4940	4940	4940	4940	4940	4940	4941	4941
10000	9918	9918	9918	9918	9918	9918	9918	9918	9918	9918
20000	19878	19878	19878	19878	19879	19879	19879	19879	19879	19879
50000	49778	49778	49778	49778	49778	49778	49778	49778	49778	49778
100000	99624	99605	99605	99605	99605	99605	99605	99605	99605	99604
200000	199283	199283	199283	199283	199283	199283	199283	199283	199283	199283
500000	498243	498243	498243	498243	498243	498243	498243	498243	498243	498243
600000	597897	597897	597897	597897	597897	597897	597897	597897	597897	597897

Table 9: Test 3a - Opening times read from Raspberry Pi

3.4.2Test 3b

Procedure :

- Configure tests as described in section 3.2
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test3b*

Algorithm:

1- *'Shut Down command'* : in order to report system to *keep_alive* mode and switch off all actuators in the event that someone is on;

2- '*Timing all heaters command'* with 10 ms as opening times for all heaters.

3- Reads the heaters pins with the 'Read_GPIOs_status' described in section 3.1

4- Wait for the MSP430 answer and check the validity of the received packet.

5- If the answer is ACK and all times read in step 5 are equal to 10 ms ± tolerance, the algorithm returns PASS. If the answer is different from ACK, the algorithm returns FAIL.

6- Return to step 2 and repeat with different opening time. Repeat until all opening times listed before are tested.

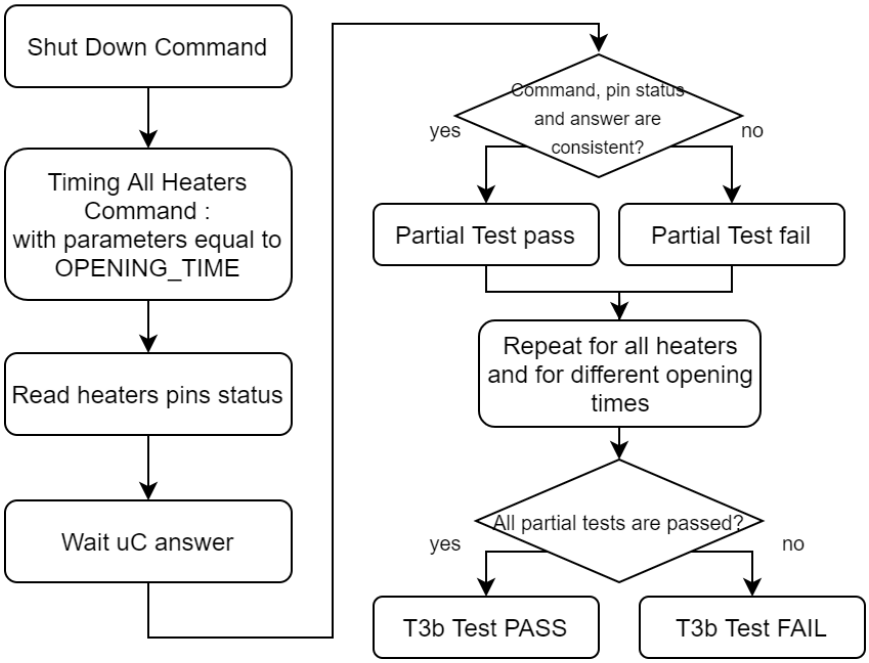


Figure 27: Flow chart of software test 3b

Pass/fail criteria :

The test is passed if all partial tests with different opening time switch on all heaters with a switch on time in a time range of : set time ± tolerance, where the tolerance is equal to 50 ms + 0.5% of the set time, otherwise the test is failed.

Test Results:

For both partial tests the read time values are less than the set switch-on times, but they are in the tolerance range, therefore the software time performances are acceptable, and the test is passed. The following table shows the read value for the whole test while the report produced by the RPi is attached in Appendix D.

	Set value [ms]								
	0	***10***	***20***	***50***	***100***	***200***	***500***	***1000***	***2000***
Heater 0	0	0	0	0	53	141	447	944	1933
Heater 1	0	0	0	0	53	141	447	944	1933

	Set value [ms]							
	5000	***10000***	***20000***	***50000***	***100000***	***200000***	***500000***	***600000***
Heater 0	4939	9918	19881	49775	99611	199266	498265	597918
Heater 1	4939	9918	19881	49775	99611	199266	498265	597918

Table 10: Test 3b - Heaters switch-on times read from Raspberry Pi

3.5 Test 4

Goal :

Test Shut Down and Reboot command.

Comment:

The test switches on valves and heaters, verifies that these pins are ON, then sends the '*Shut Down*' command and verifies that all actuators turn OFF, the same procedure is applied to test the '*Reboot'* command.
The microcontroller can supply a maximum of 48 mA on its pins, repeating the test several times it has been noticed that if there are 24 LEDs on at the same time, on the serial communication with the Raspberry errors occur, e.g., by losing one byte in the communication. Therefore, for this test it is decided to switch on the 2 heaters and 6 valves as a manoeuvre requires switching on the 2 valves along the fluidic line and at most 4 valves which regulate the nozzles.

Procedure :

- Configure tests as described in section 3.2
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test4*

Algorithm:

1. '*Pre-Firing command'* : to move the system to *pre_firing* mode and switch on valves;
2. '*Check Firing Mode command*' : waits until the thruster reaches the right temperature and is in *firing* mode, therefore if the answer is *NACK* the test waits 1 second then resends the command.
3. '*Timing all heaters command*' with 10000 ms as opening times for all heaters.
4. '*Timing all valves command'* with 2000 ms as opening times for the first six valves.
5. Read all the pins controlling the actuators, valves and heaters, with the 'Read_GPIOs_status' described in section 3.1 and check that both heaters and the first six valves are ON, while the last four are OFF.
6. *'Shut Down command'* .
7. Read all the pins controlling the actuators with the 'Read_GPIOs_status'.
8. Repeat steps 1 to 5.
9. '*Reboot command*'.
10. Reads the valves and heaters pins with the 'Read_GPIOs_status'.

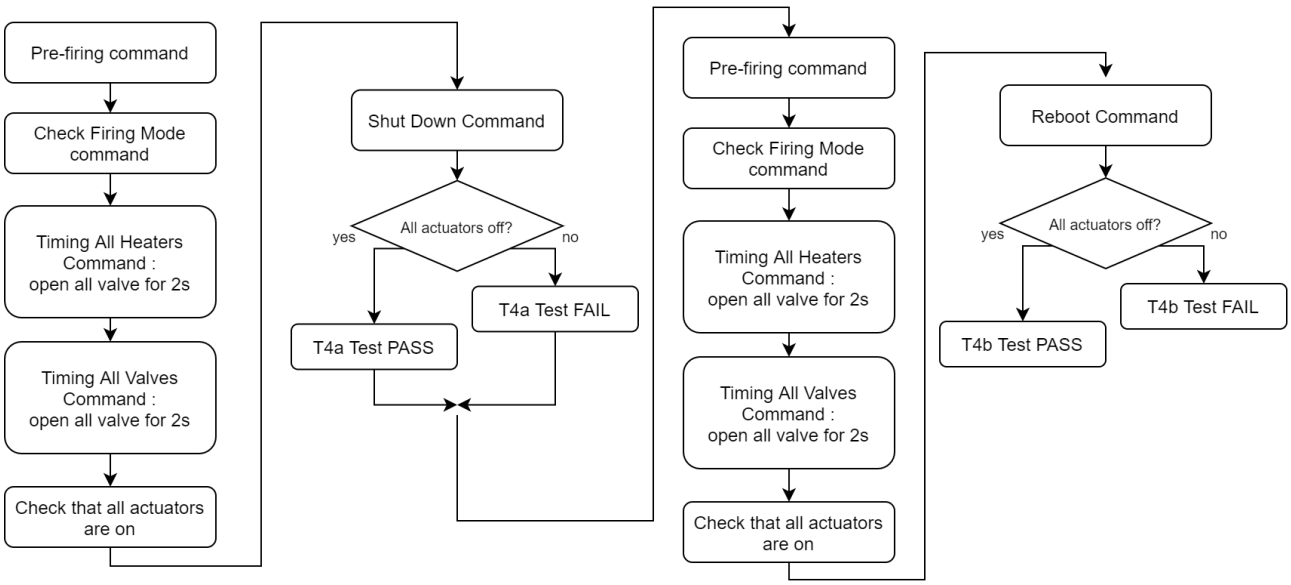


Figure 28: Flow chart of software test 4

Pass/fail criteria :

The test is passed if after the Shut Down command and, then after Reboot command, the status of all pins is OFF, if one or more pins are ON the test returns FAIL.

Test Results:

All actuators are off both after Shut Down command and Reboot command, therefore the test is passed. The report produced by the RPi is attached in Appendix D.

3.6 Test 5

Goal :

Test the command to request some data to propulsion system, in particular the tested commands are:

- "Valves timing request";
- "Heaters timing request";
- "Telemetry request".

Procedure :

- Configure tests as described in section 3.2
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test5*

Algorithm:

1. *'Shut Down command'* : in order to report system to *keep_alive* mode and switch off all actuators in the event that someone is on;

2. *'Valves timing request*' that should return 0 ms for all valves, since the PS is in *keep_alive*;
3. '*Pre-Firing command'* : to move the system to *pre_firing* mode;
4. '*Check Firing Mode command'* : before sending the command to switch on the valves, the algorithm waits for the thruster to have the right temperature and to be in *firing* mode; if the answer is *NACK* the test waits 1 second then resends the command;
5. '*Timing all valves command'* with 5000 ms as opening times for the first valve, 6000 ms for the second, 7000 ms for the third and so on up to 14000 ms for the lasts;
6. *'Valves timing request*';
7. Wait for the MSP430 answer;
8. '*Request telemetry';*
9. Wait for the MSP430 answer that should be the *'Answer_telemetry'*;
10. *'Shut Down command'* ;
11. *'Heaters timing request*' that should return 0 ms for all heaters, since the PS is in *keep_alive*;
12. '*Timing all heaters command'* with 10000 ms as opening times for the first heater and 20000 ms for the other.
13. *'Heaters timing request*';
14. Wait for MSP430 answer.

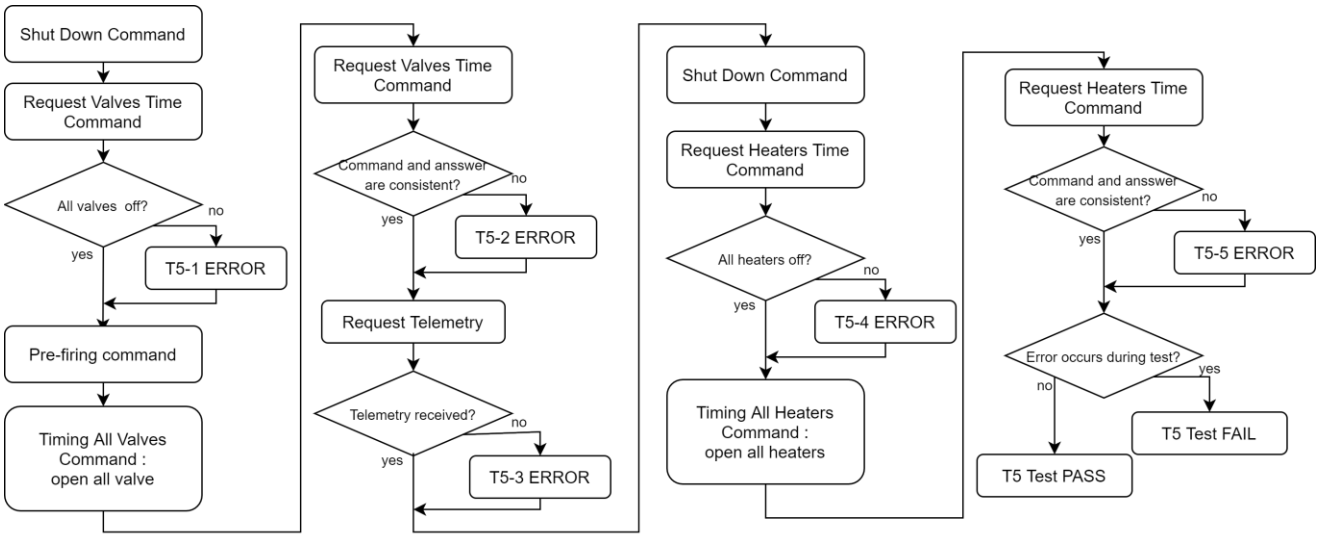


Figure 29: Flow chart of software test 5

Pass/fail criteria :

If the Raspberry Pi receives the telemetries and the telemetry data is coherent with the system status in terms of actuators status and temperature values, then the test is passed.

Test Results:

The RPi received telemetries and the data is consistent with expected values, considering that, since current and pressure sensors are not connected to the microcontroller, their values are set to default value of 40 bar and 10 mA, while the voltage values are read on a sun sensor that returns a full-scale value of 2,048 V. The report produced by the RPi is attach at the Appendix D.

3.7 Test 6

Goal :

test the 'Change threshold command', which has the purpose of changing the thresholds for temperature control.[Section 2.5.5]

Procedure :

- Configure tests as described in section 3.2
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test6*

Algorithm:

1. Send to the propulsion module the *'Change thresholds'* command with parameter:

	Max_threshold [°C]	***Med_threshold [°C]***	***Min_threshold [°C]***
Tank sensor	100	70	30
Pipes sensor	100	70	30

Table 11: New temperature thresholds set for the Test 6

2. Wait for the MSP430 answer
3. If the answer is ACK the algorithm returns PASS and the MSP430 debug lines prints the new temperature threshold, otherwise it returns an ERROR message, and the test outcome is FAIL.

Pass\fail criteria:

The propulsion system changes its temperature thresholds and prints on the debug line the new thresholds values.

Test Results:

The propulsion system answers with ACK; these mean that the operation is executed with success and the new thresholds are set. The MSP430 debug line prints the new

temperature thresholds that corresponds to the parameters sent from GNC: 100°C, 70°C and 30°C.

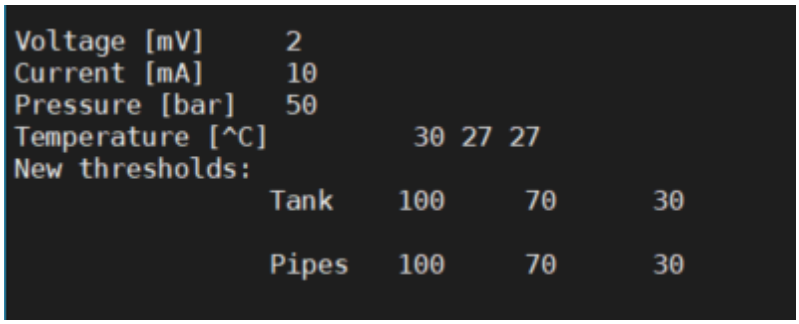


Figure 30: Test 6 - MSP430 debug line shows the new temperature thresholds

4 Test with Heaters

In the previous chapter the software is tested in terms of executions of commands and the linked functions that manages actuators, while the goal of this chapter is to test the temperature control logic [Section 2.5.5]. Since the propulsion system is designed in order to thrust only if the system is in the optimal temperature range, this control algorithm is fundamental in order to bring and hold the system in the right temperature range.
To test that algorithm some hardware was required to enable us to actually change and measure the temperature. Since the objective is just to test the correct functioning of the algorithm, i.e., the correct switching between operative modes, it is not necessary to use the final components that will be used for the propulsion module, therefore they are selected among the components available in the Tyvak laboratory. To change the temperature, we identified two simple heaters[13] which have a 5W power consumption, while to measure the temperature we selected two thermistors[14] which have a tolerance of ± 0.2°C, very acceptable for this purpose.

4.1 Set up

4.1.1 Hardware configuration

To perform this test the LEDs connected to the heaters enable signals are removed and replaced by two real heaters. Since they required more current than the MSP430 can provide, they are controlled by MOSFETs. The MOSFETs are implemented in open-drain configuration: the heater is connected between the 16V provided by an external power supply and the MOS drain, the gate is driven by the MSP430 pins, in particular it's connected to the heater enable signal, while the source is connected to ground. In this way, the transistor acts as a switch: when the enable signal is high the switch is closed and the heater is connected to GND allowing the current to flow, otherwise the switch is open, and the heater is off.
To measure the change in temperature two thermistors are added to the development board, one for each heater. A thermistor is a resistance that changes is value in function of its temperature, therefore, to have a measure of the temperature it's needed to read the voltage drop on it. To do that we mounted a voltage divider, and an ADC which has in input the voltage drop on the thermistor and gives in output the digital value on the I2C bus.

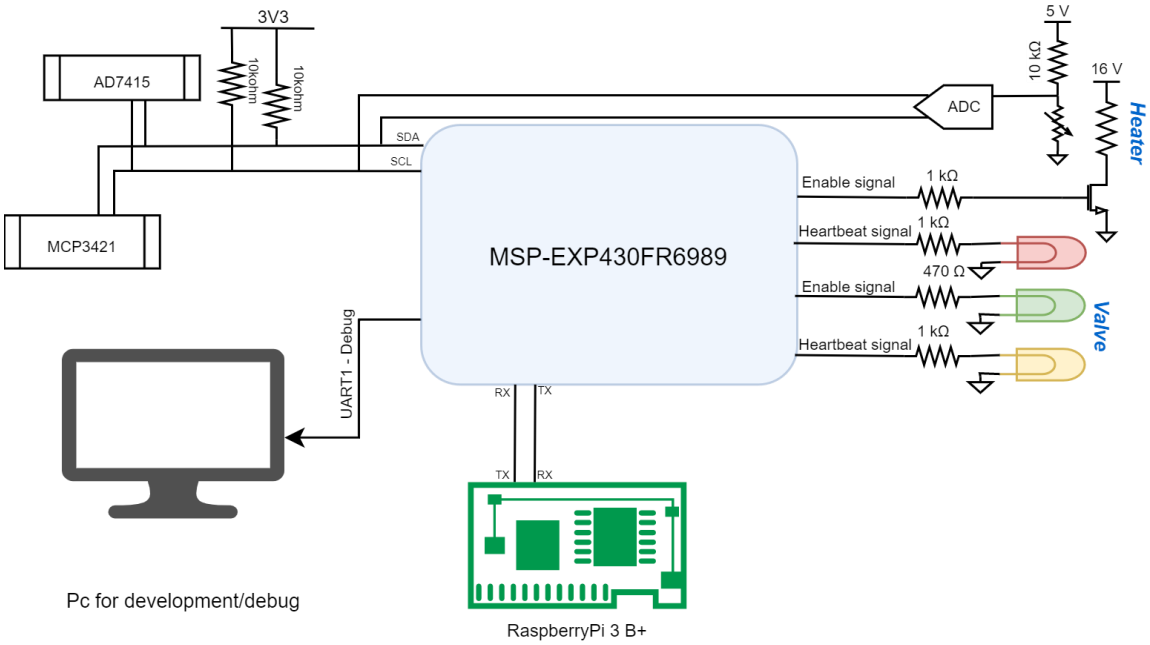


Figure 31: Hardware configuration for temperature control test

To improve the thermal inertia each heater and its thermistor is attached with Kapton to a metal plate.

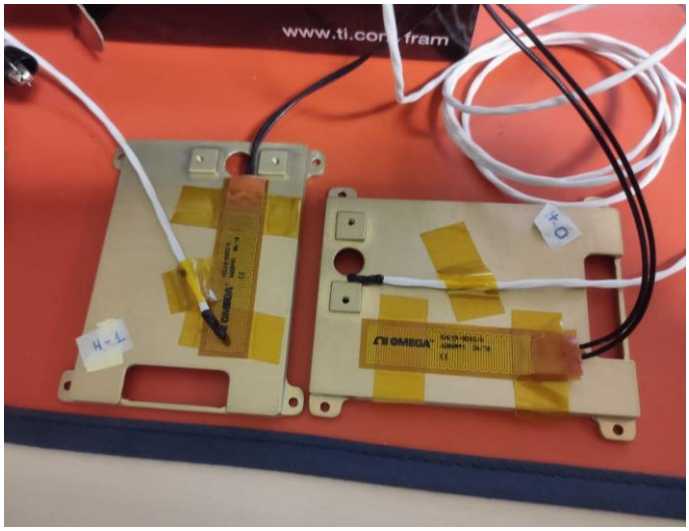


Figure 32: Photo of heaters and thermistors configuration for temperature control test

4.1.2 Software configuration

From the software point of view, it is necessary to add the thermistor temperature reading to the software of the propulsion module; while on the Raspberry Pi, with the same logic used in the software tests [Chapter 3], an algorithm has been implemented to evaluate that the temperature control system is able to maintain the temperature in a certain range.

Propulsion system software

Until now on the MSP430 we required three different measurements of the temperature, but they were carried out by the only temperature sensor present on the board, now by adding two thermistors, each of the three measurements is made by a different sensor. By design and requirements, the three different temperatures are linked to the tank temperature, pipes temperature and one extra sensor for possible future use / expansion. Since the heaters are one near the tank and the other is on the pipes line, the two thermistors are associated to the tank and pipes temperatures, while the extra measurement is associated to the temperature sensor AD7415.

The ADC returns the voltage drop on the thermistor in milli volts, so to obtain the temperature value we used the following formula [14]:

$$T\,[°C] = \frac{1}{\mathrm{A} + \mathrm{B} * \log\left(\frac{V * R}{V_{dc} - V}\right) + C * \left(\log\left(\frac{V * R}{V_{dc} - V}\right)\right)^3} - 273.15$$

$V = ADC\ voltage\ measure\ [mV]$

$V_{dc} = 5.4V\ measured\ power\ supply\ voltage\ of\ divider$

$R = 10k\Omega\,, voltage\ divider\ resistance$

Where A, B and C are parameters which come from the thermistor model:

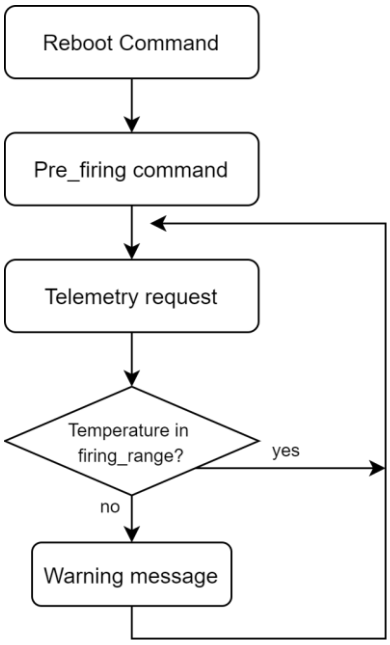
$$A = 1.285 * 10^{-3}\ [K^{-1}]$$

$$B = 2.362 * 10^{-4}\ [K^{-1}]$$

$$C = 9.285 * 10^{-8}\ [K^{-1}]$$

Raspberry Pi software

As for the software tests performed in Chapter 3, the Raspberry Pi is used as tester, it has the aims to send command to the propulsion module and evaluate its state. In this case, it's not necessary to read the MSP430 pins, the algorithm is structured as shown in the Figure 33.



1. Reboot the system in order to move the system to keep_alive mode, reset potential changes in the temperature thresholds and shut down all actuators.

2. Send command to move system to *pre_firing* mode

3. Periodically require the telemetry in order to evaluate the temperature measurement and propulsion module operative mode.

Figure 33: Raspberry Pi flow chart for temperature control tests

The temperature thresholds on the propulsion system are set to :

- Minimum temperature : 27°C
- Medium temperature: 31°C
- Maximum temperature: 35°C

Therefore, the system can be in firing range if the temperatures are between 31°C and 35°C. Considering the tolerance in the measurement, the Raspberry Pi are set in order to consider acceptable if the temperatures stay in a range of 30.5°C and 35.5°C.

4.2 Test execution

Items:

- MSP-EXP430FR6989 with its USB-miniUSB cable;
- Development board configured as described in section 4.1.1;
- Bench power supply;
- Raspberry Pi 3 model B+ with its power supply cable;
- PC with MobaXterm;

Procedure:

- Connect the MSPEXP-430FR6989 to the breadboard with the LEDs and resistors used in section 3.1.1.

- Connect MSPEXP-430FR6989 to the PC with its USB cable.
- Open MobaXterm, click on “New Session”, then select “Serial”. In the field “Serial port” choose “COM 5 (MSP Debug Interface)” in the drop-down menu, then select 115200 bps as “Speed”, finally click on the OK button at the bottom. A terminal is opened, it shows the debug line of the MSP430, in fact it prints all messages that the microcontroller sends on the UART-A0; it’s also possible to write on that terminal to send messages on the UART-A0 line to the MSP430, but the microcontroller is not programmed to receive data on that bus, so if something is written on the terminal the microcontroller blocks and requires reboot.
- Connect the Raspberry Pi as described in Section 3.1.1.
- Connect each actuators pin to a RPi’s GPIOs.
- Switch on the Raspberry Pi.
- On MobaXterm open a second terminal clicking on “+” near the first terminal name, select “SSH” and compile in the following way:
    - Remote host : 192.168.8.41
    - Specify username : pi
    - Port : 22
- A new terminal is open, and a password is required, so insert “raspberry”.
- On the RPi terminal type: ‘*cd martina/GNC/*’
- On Raspberry Pi terminal write and launch the command: *./gncsimulator_test7*

Results:

The propulsion module moves to *pre_firing* mode and since the temperatures read from thermistors are not in the firing range (29°C and 27°C), the heaters are switched on; meanwhile the Raspberry Pi highlights that the propulsion system has not the correct temperature for thrusting. After a while, the PS reaches the firing range and the RPi stops warning about the temperature out of range. Both heaters still stay switched on until their own thermistor measures 35°C, so each heater is individually driven in function of its thermistor measure. When the heaters are switched off, the temperature starts to decrease, if it reaches the medium threshold of 31°C the heater is switched on again in order to keep the temperature between medium and maximum thresholds.

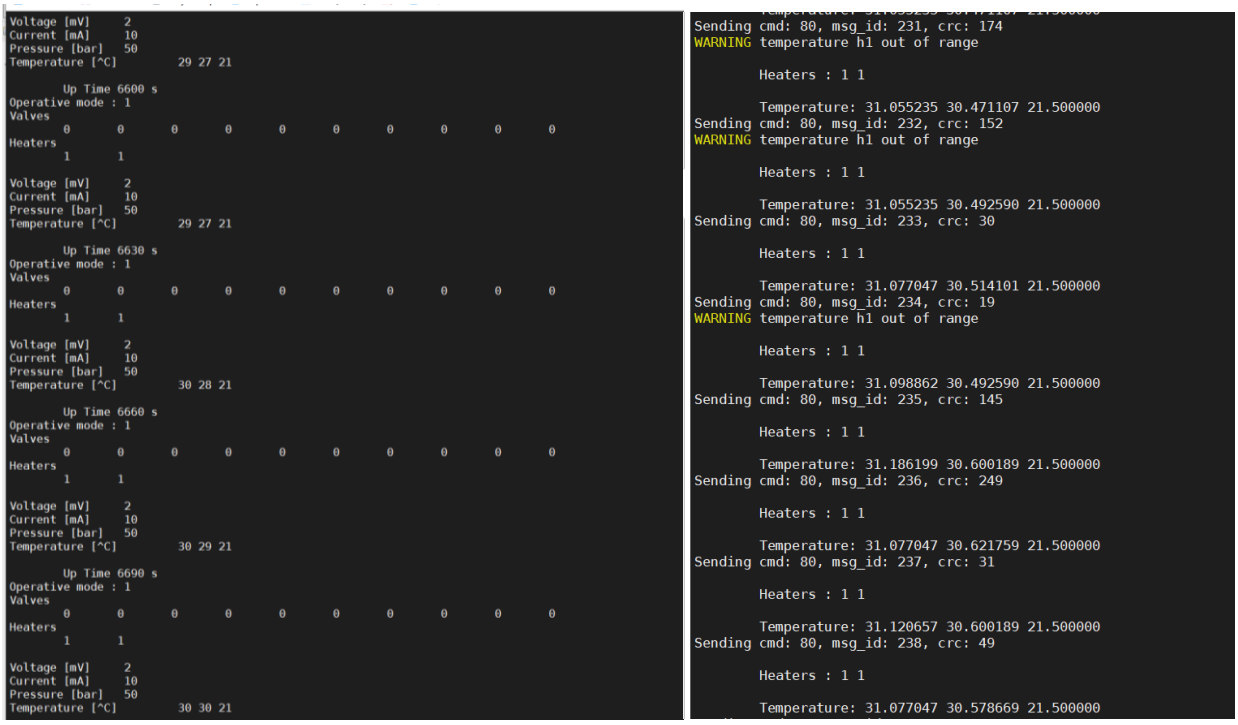


Figure 34: Propulsion microcontroller debug line (left), Raspberry Pi debug line (right) during temperature control test

5 MATLAB simulation

The last step is the validation of the software by means of the simulation of the manoeuvres. The objective of this chapter is to describe how the simulation has been implemented on MATLAB, highlighting the performances in the implementing manoeuvres. In these simulation, only elementary manoeuvres are tested: translation and rotation around one of the main axes of the satellite.
The model does not take care about non ideality introduced by hardware, but it pays attention only to the software performances.
In a satellite the position and orientation are calculated by the GNC system thanks to sensors, in this case they will be calculated analysing only the effective opening times of valves.
To implement these tests, in addition to the MSP430 controller to be tested, MATLAB to calculate the position and orientation of the satellite, a Raspberry Pi acting as a link between MATLAB and the propulsion module is required. RPi has the aims of managing the serial communication with MSP430 and reading the status of pins controlling the valves.

5.1 Set up

In these new tests the MSP430 is still connected to the RPi by means of serial line to receive the commands, and the RPi is still connected to enable signal of each valve in order to read their status.
While, since the RPi UART line is already employed to communicate with MSP430, to establish a communication line between the MATLAB computer and the RPi, the Ethernet socket[15] is used. Ethernet protocol requires a client and a server, the first one sends a request to the server, which then answers to that request. In this application the RPi is configured as client and the MATLAB code as server. The Ethernet socket is configured in the following way:

Raspberry Pi:

- Network Role : client
- IP : 192.168.15.21
- Port : 8080
- Byte Order : little-endian

MATLAB :

- Network Role : server
- IP : 192.168.15.20
- Port : 8080
- Byte Order : little-endian

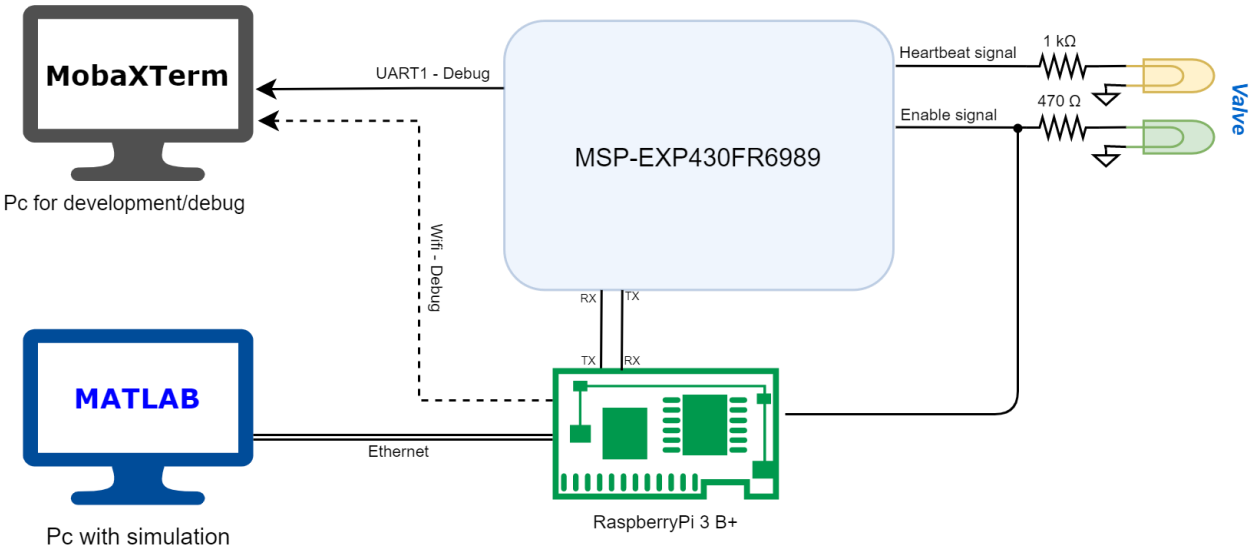


Figure 35: Configuration for MATLAB simulation

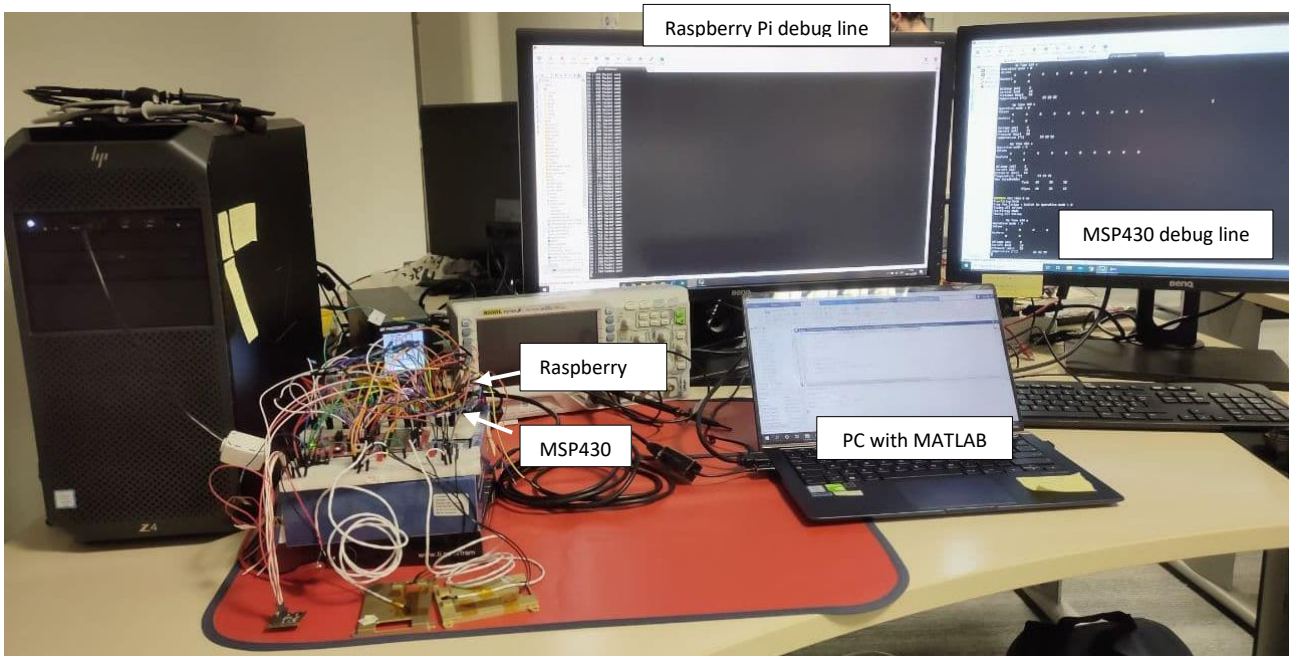


Figure 36: Bench test for MATLAB simulation

5.1.1 Raspberry Pi configuration

The Raspberry software is divided in three thread:

1. First thread is the main thread and has the task of managing the reception of manoeuvres to simulate from MATLAB, computing which valves shall be opened, and sending commands to propulsion module.
2. Second thread is responsible for the continuous reading of the valves pins and reporting that to MATLAB in real time via Ethernet. The sample of valves pins can be done until once every 1.5ms but, it is set every 5ms since MATLAB code takes around 5 ms to compute a loop.
3. Third thread is used to manage the serial communication with the propulsion system, it allows to send commands or requests to propulsion module and asynchronously receive the answer while the RPi is employed in other tasks.

The algorithm is able to receive command on Ethernet, re-send it on UART to propulsion module, and report to MATLAB the valves status in real time, this allows to monitor the status of valves and correct some manoeuvres in case of non-ideal behaviour.

The messages received by MATLAB are divided into two fields: the first is a number which selects the manoeuvre to be performed, for example if *manoeuvre* is 1 the wanted manoeuvre is a translation along X axis with negative direction [see Table12]. For these simulations only the twelve elementary manoeuvres are implemented. The selection of which valves shall be opened are done by RPi software and are reported in the Table 12.
The second field is the opening time of valves, since we want to implement only elementary manoeuvre the opening time shall be equal for all selected valves.

Data type	***Field***	***Comment***
uint32_t	Manoeuvre	Select which manoeuvre can be performed
uint32_t	Opening time	For how many times the manoeuvre should be performed

Table 12: Structure of packet exchange between MATLAB and Raspberry Pi

Manoeuvre	***Value***	***V0***	***V1***	***V2***	***V3***	***V4***	***V5***	***V6***	***V7***	***V8***	***V9***
Positive translation along X	0	on	off	off	off	off	on	on	on	on	off
Negative translation along X	1	on	on	on	on	on	off	off	off	off	on
Positive translation along Y	2	on	off	off	on	on	off	off	on	on	on
Negative translation along Y	3	on	on	on	off	off	on	on	off	off	on
Positive translation along Z	4	on	off	on	off	on	off	on	off	on	on

Negative translation along Z	5	on	on	off	on	off	on	off	on	off	on
Positive rotation around X	6	on	off	on	on	off	off	on	on	off	on
Negative rotation around X	7	on	on	off	off	on	on	off	off	on	on
Positive rotation around Y	8	on	on	off	on	off	off	on	off	on	on
Negative rotation around Y	9	on	off	on	off	on	on	off	on	off	on
Positive rotation around Z	10	on	off	off	on	on	on	on	off	off	on
Negative rotation around Z	11	on	on	on	off	off	off	off	on	on	on

Table 13: Valves to be opened in function of manoeuvre that should be performed. For the valves position see Section 5.1.2

5.1.2 MATLAB software

MATLAB algorithm aims to validate the correct performance of the manoeuvres work by the MSP430 software. As it's explained before, it has implemented an Ethernet socket which works as server in order to send messages with the manoeuvres to be performed, and periodically receives the status of pin controlling valves to compute the displacement and rotation obtained after the manoeuvres.

The main task of MATLAB software is computing the actual position and attitude of the satellite during and after the actuation of the manoeuvres. To pass from the status of the valves obtained from RPi to the position and attitude is necessary to introduce a mathematical model.
The input of this model is the valves vector where each element is 1 if the valves is open, 0 if it's closed. The output of the model is the attitude expressed by means of quaternion q and the position r referred to an inertial reference frame:

$$u = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \end{bmatrix} \qquad y = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ r_x \\ r_y \\ r_z \end{bmatrix}$$

To compute the output vector y, the analysis is divided in three steps:

1. <u>*Nozzle position model*</u> : computes the total force F and total torque that are generated if one or more valves are opened;

2. *Dynamic model* : starting from applied force and torque computes the linear velocity v and angular velocity ω;
3. *Kinematic model* : from v and ω computes the position r and the attitude q.

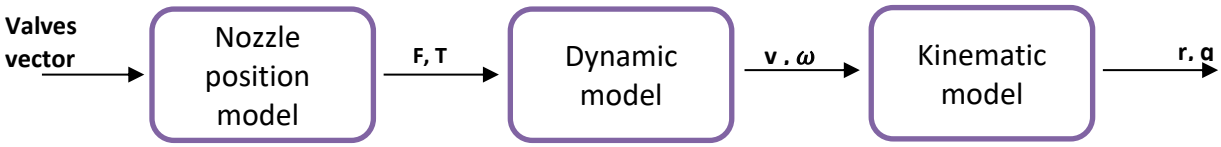


Figure 37: Mathematical models for MATLAB simulation

Since the simulation is used to validate the MSP430 software and there is not still a real satellite to be considered, in all three models an ideal 6U CubeSat with the following dimensions and sizes is considered:

- 6U CubeSat dimension along X axis : $d_x = 0.226\ m$
- 6U CubeSat dimension along Y-axis : $d_y = 0.1\ m$
- 6U CubeSat dimension along Z axis : $d_z = 0.366\ m$
- 6U CubeSat mass : $m = 12\ kg$

The propulsion system has in total 10 valves, but two are placed along the pipes line to avoid unwanted thrust, therefore only eight valves are directly connected to a nozzle.
The nozzles are placed as shown in the following figures:

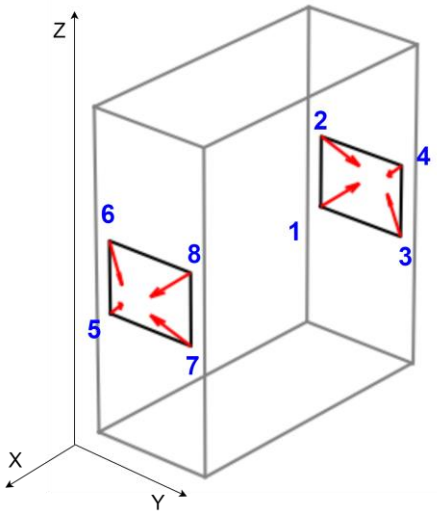


Figure 38: Nozzles direction and valves position relative to the 6U CubeSat

The dimensions are:

- Propulsion module dimension along X axis : $s_x = 0.226\ m$
- Propulsion module dimension along Y axis : $s_y = 0.075\ m$
- Propulsion module dimension along Z axis : $s_z = 0.075\ m$
- Absolute force generated from one nozzle : $f = 7.5\ mN$

- Angle between the nozzle and X axis : $\alpha = \frac{\pi}{6}$
- Angle between the nozzle and Z axis : $\beta = \frac{\pi}{4}$

The force component of each nozzle is computed as:

$$f_x = f * \cos(\beta) * \cos(\alpha)$$
$$f_y = f * \cos(\beta) * \sin(\alpha)$$
$$f_z = f * \sin(\beta)$$

Nozzle position model

This model takes as input the valves vector from RPi and gives as output the total force and torque that act on the spacecraft.

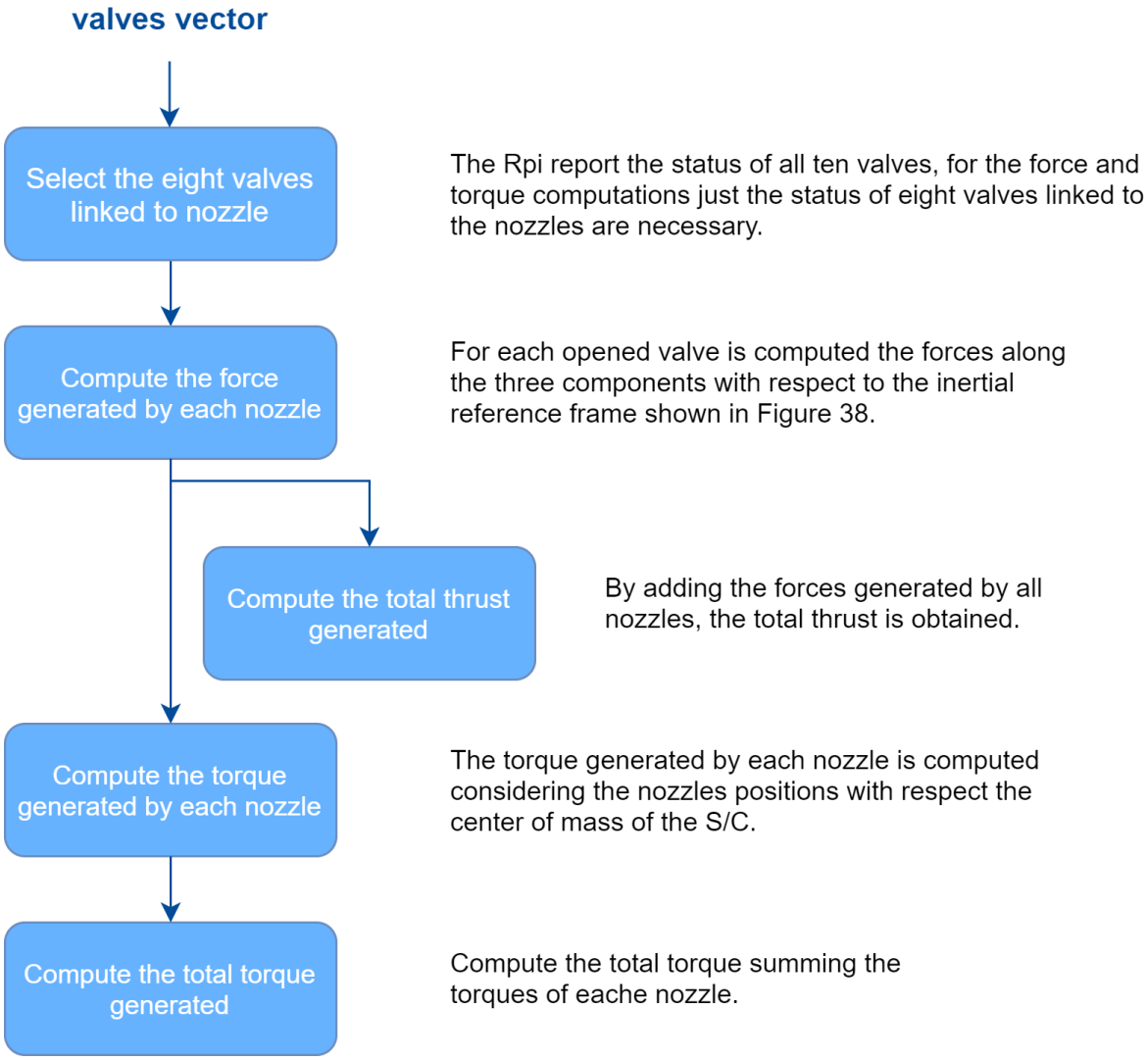


Figure 39: Flow chart of the Nozzle position model

Dynamic model

The dynamic model takes in input the thrust F and torque T computed by means of the *Nozzle position model* and computes the angular velocity ω and linear velocity v by the integration of the following equations[16]:

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix} * \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$$

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} 0 & \sigma_1 * \omega_z & 0 \\ \sigma_2 * \omega_z & 0 & 0 \\ \sigma_3 * \omega_y & 0 & 0 \end{bmatrix} * \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \begin{bmatrix} T_x/J_x \\ T_y/J_y \\ T_z/J_z \end{bmatrix}$$

Where :

$$\sigma_1 = \frac{J_2 - J_3}{J_1} \qquad \sigma_2 = \frac{J_3 - J_1}{J_2} \qquad \sigma_3 = \frac{J_1 - J_2}{J_3}$$

And J_x, J_y and J_z are the components of diagonal inertial matrix J and are computed as:

$$Jx = \frac{1}{12} \cdot m \cdot \left({d_y}^2 + {d_z}^2 \right)$$

$$Jy = \frac{1}{12} \cdot m \cdot \left({d_x}^2 + {d_z}^2 \right)$$

$$Jz = \frac{1}{12} \cdot m \cdot \left({d_x}^2 + {d_y}^2 \right)$$

Kinematic model

The kinematic model has as input the components of angular velocity ω and linear velocity v and obtains the attitude expressed by quaternion q and position $r$with respect the inertial reference frame, by integrating the following equations[17]:

$$\begin{bmatrix} \dot{q_o} \\ \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \end{bmatrix} = \frac{1}{2} * \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_o & -q_3 & q_2 \\ q_3 & q_o & -q_1 \\ -q_2 & q_1 & q_o \end{bmatrix} * \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

$$\begin{bmatrix} \dot{r_x} \\ \dot{r_y} \\ \dot{r_z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Dynamic-Kinematic equations

Putting together the dynamic and kinematic model the system can be described by the state equations:

$$\dot{x} = A(x) \cdot x + B \cdot u$$

Where

- x is the state vector
- u is the input vector

The state vector is a vector of size $13x1$:

$$x = \begin{bmatrix} q \\ r \\ \omega \\ v \end{bmatrix}$$

$q \in R^{4x1}$
$r \in R^{3x1}$
$\omega \in R^{3x1}$
$v \in R^{3x1}$

While input vector is $6x1$ vector:

$$u = \begin{bmatrix} T_x \\ T_y \\ T_z \\ F_x \\ F_y \\ F_z \end{bmatrix}$$

Since the linear quantities and angular quantities are reciprocally independent to easily explain and to reduce the matrices dimensions the state equations are divided in:

- Angular quantities which consider torques, angular velocities and quaternions;
- Linear quantities which regard translation forces, linear velocities and position changes.

Obviously, since these analyses are made to identify non-ideal behaviour of the manoeuvres, the MATLAB simulation always implements both cases in order to highlight if a thrust manoeuvre generates unwanted torque, or if a torque manoeuvre generates unwanted translation force.

Therefore, the state equations become:

Angular quantities

$$x = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \qquad u = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & -q_1 & -q_2 & -q_3 \\ 0 & 0 & 0 & q_0 & -q_3 & q_2 \\ 0 & 0 & 0 & q_3 & q_0 & -q_1 \\ 0 & 0 & 0 & -q_2 & q_1 & q_0 \\ 0 & 0 & 0 & 0 & \sigma_1 * \omega_z & 0 \\ 0 & 0 & 0 & \sigma_2 * \omega_z & 0 & 0 \\ 0 & 0 & 0 & \sigma_3 * \omega_y & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/J_x & 0 & 0 \\ 0 & 1/J_y & 0 \\ 0 & 0 & 1/J_z \end{bmatrix}$$

Linear quantities

$$x = \begin{bmatrix} r_x \\ r_y \\ r_z \\ v_x \\ v_y \\ v_z \end{bmatrix} \qquad u = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix}$$

5.2 Test : Translation manoeuvre along X axis

At this point that all model and software are implemented and described, it's possible to start tests. The first test is a manoeuvre of translation along X axis.

Procedure:

- At the begin of MATLAB software, it's possible to set the parameters of the spacecraft and the manoeuvres. In this case, the spacecraft is just a parallelepiped with the sizes specify in the previous section. The opening time is

set to 200s and the manoeuvre is equal to 1, which compared with Table 12 is a '*Positive translation manoeuvre along X axis*'

```
open_time = 200000; %ms
manouver = 1;       %refered to Table 12
```

Figure 40: MATLAB software to set a 'Positive translation manoeuvre along X axis' with 200s as opening time

- On RPi debug line, type *'.GNC/gncsimulator_sil'* to run the software for the simulation;
- Press *'Run'* on MATLAB, verify that the Ethernet connection is established and see the graphics plotting over time.

```
Wait for connection...
Connection Done
Manouver sent : 1 with 200000 s
Valves vector received..
```

Figure 41: MATLAB command window when a manoeuvre of translation along X axis is simulated

Results:

The MATLAB graphics show as the manoeuvre is like an ideal manoeuvre:

- no torques, no angular velocities, no rotations are generated;
- the only thrust generated is along X axis and it is equal to 18,4 mN;
- the component v_x increases linearly, while the displacement has a parabolic trend;
- after around 200s all valves are closed, and the force F_x goes to 0, the linear velocity remains constant at 2,46 m/s and the displacement trend becomes linear.

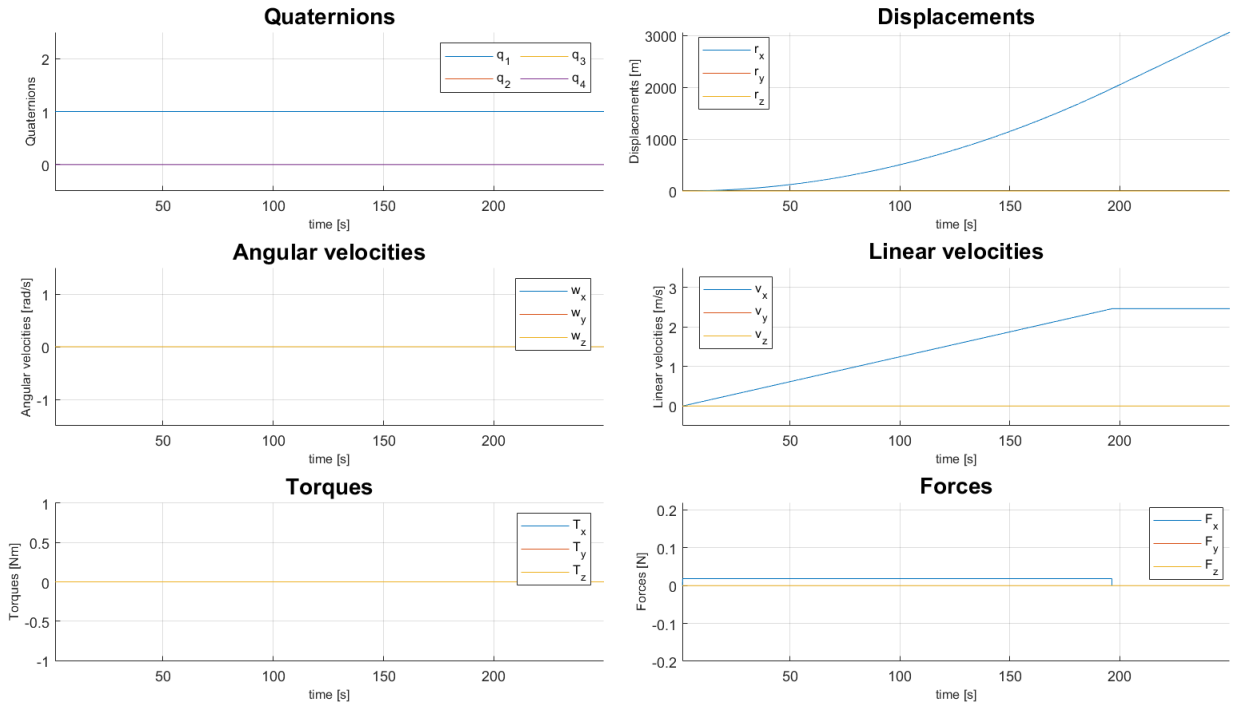


Figure 42: Linear and angular quantities trend due to a manoeuvre of translation along X axis

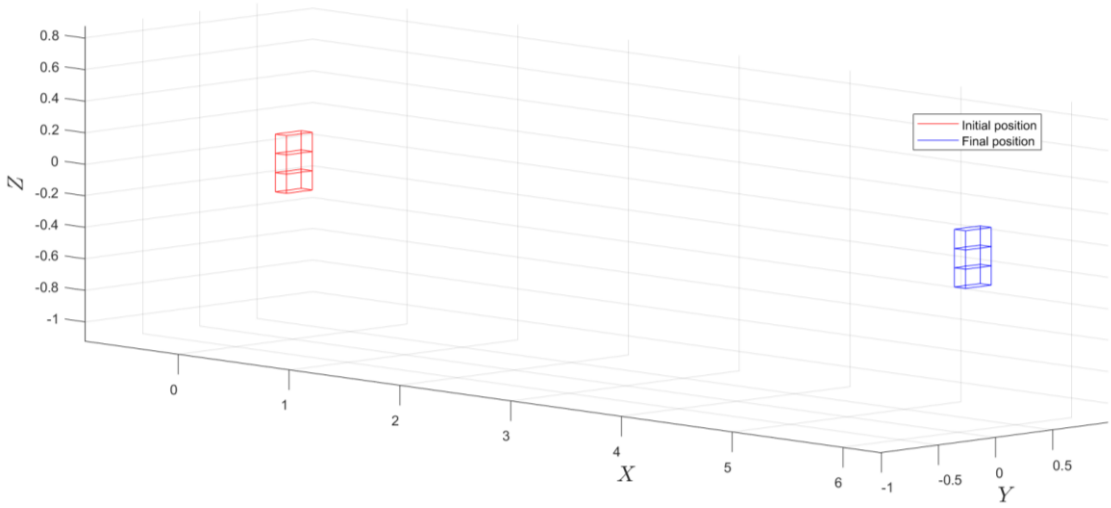


Figure 43: The representation of the spacecraft position after the manoeuvre of translation along X (the displacement is rescaled by a factor of 500, otherwise the image would not be visible)

5.3 Test : Translation manoeuvre along Y axis

Procedure:

- At the begin of MATLAB software set opening time to 200s and the manoeuvre variable equals to '3', which corresponds to the '*Positive translation manoeuvre along Y axis'* (see Table 12)

```
open_time = 200000; %ms
manouver = 3;       %refered to Table 12
```

Figure 44: MATLAB software to set a 'Positive translation manoeuvre along Y axis' with 200s as opening time

- On RPi debug line, type *'.GNC/gncsimulator_sil'* to run the software for the simulation;
- Press *'Run'* on MATLAB, verify that the Ethernet connection is established and see the graphics that are plotted over time.

```
Wait for connection...
Connection Done
Manouver sent : 3 with 200000 s
Valves vector received..
```

Figure 45: MATLAB command window when send a manouver of 'Positive translation along Y axis'

Results:

After the ethernet connection, MATLAB starts to plot the progress of linear quantities:

- the force has only the y component different from 0, in particular F_y is equal to 10.6 mN;
- the component v_y increases linearly until the force turns to 0, then the v_y remain stable to 1.42m/s;
- the r_y displacement has a parabolic trend when the velocity increases linearly, after 200s its trend becomes linear;
- the angular quantities remain constant during manoeuvring.

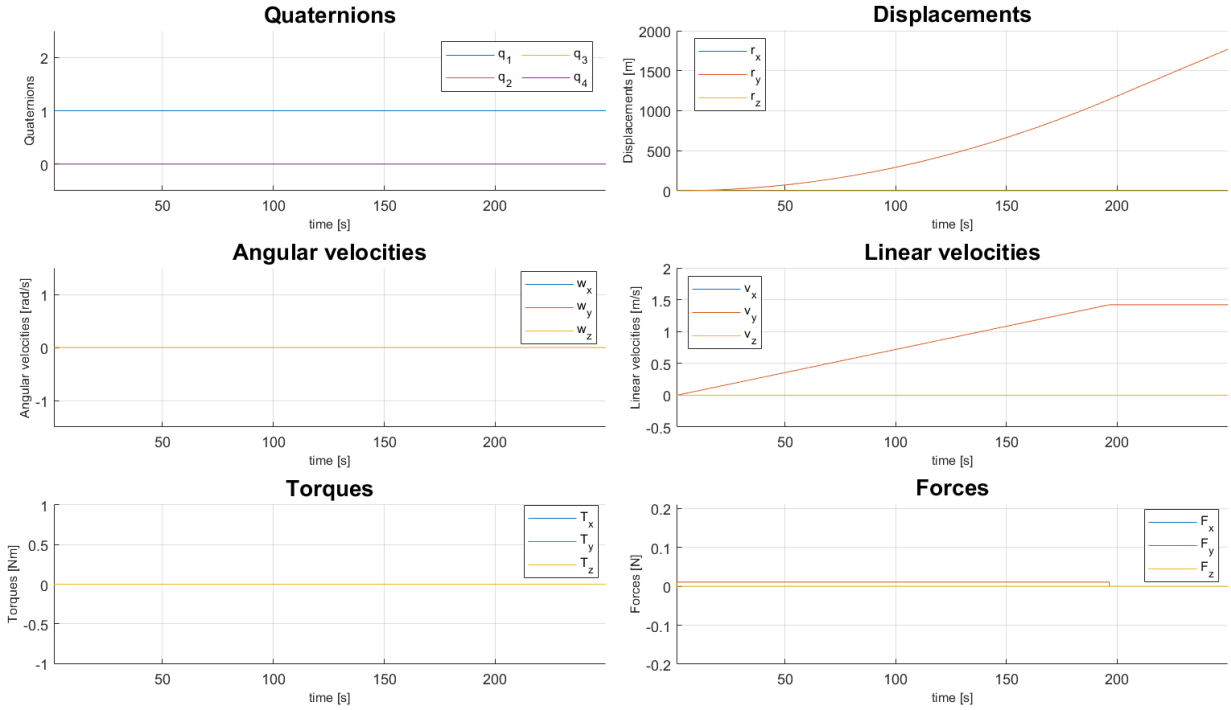


Figure 46: Linear and angular quantities trend due to a manoeuvre of translation along Y axis

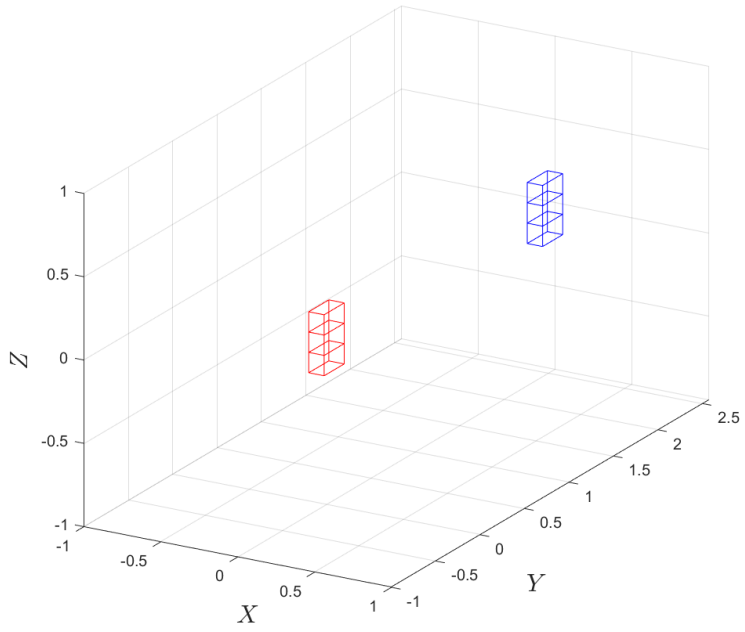


Figure 47: The representation of the spacecraft position after the manoeuvre of translation along Y (the displacement is rescaled by a factor of 100, otherwise the image would not have been visible)

These results show that the manoeuvre is well performed by the software, no disturbances or non-ideality behaviour are introduced by it.

5.4 Test : Translation manoeuvre along Z axis

Procedure:

- In the MATLAB code set open time variable to 200s and the manoeuvre equal to 5, which compared with Table 12 is a '*Positive translation manoeuvre along Z axis'*

```
open_time = 200000; %ms
manouver = 5;       %refered to Table 12
```

Figure 48: MATLAB software to set a 'Positive translation manoeuvre along Z axis' with 200s as opening time

- On RPi debug line, type *'.GNC/gncsimulator_sil'* to run the software for the simulation;
- Press *'Run'* on MATLAB, verify that the Ethernet connection is established and see the graphics plotting over time.

```
Wait for connection...
Connection Done
Manouver sent : 5 with 200000 s
Valves vector received..
```

Figure 49: MATLAB command window when send a manouver of 'Positive translation along Z axis'

Results:

MATLAB plots show that:

- the angular quantities remain constant during the manoeuvre;
- the z component of force is equal to 21.2 mN, while other components are 0;
- the component v_z increases linearly, then it stabilizes at the value of 2.84 m/s;
- the r_z displacement has a parabolic trend when the velocity increases linearly, then the trend becomes linear.

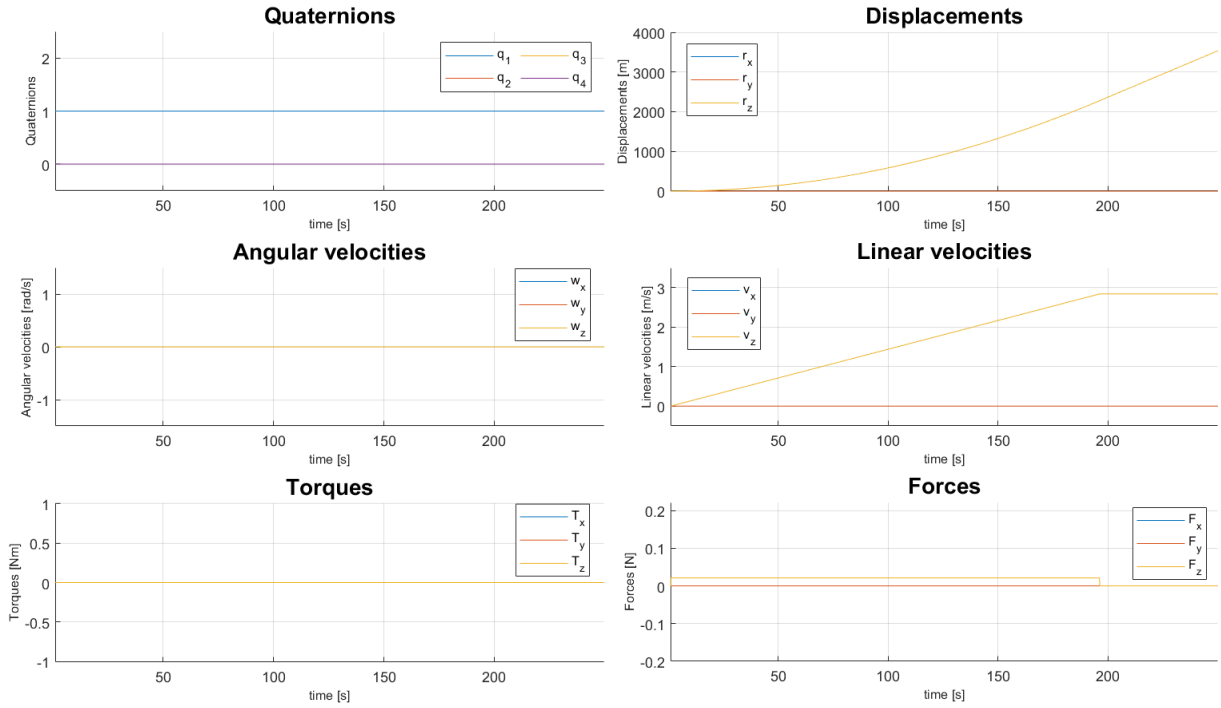


Figure 50: Linear and angular quantities trend due to a manoeuvre of translation along Z axis

This graphics show how the manoeuvre is well performed, no disturbances or no-ideality behaviour are introduced by the MSP430 software.

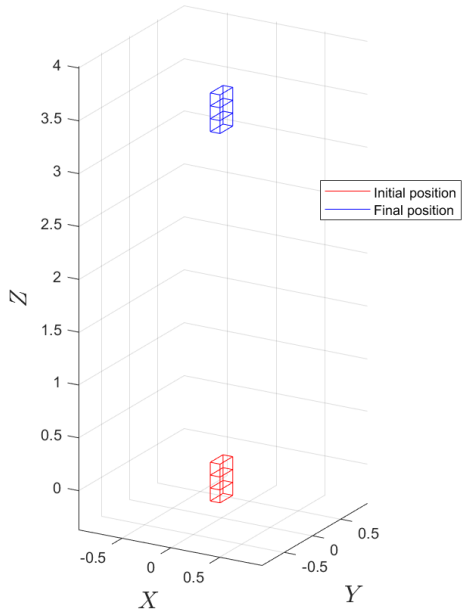


Figure 51: The representation of the spacecraft position after the manoeuvre of translation along Z (the displacement is rescaled by a factor of 1000, otherwise the image would not be visible)

5.5 Test : 90° rotation around Z axis

In the previous tests it's possible to see how even if a manoeuvre is finished, the satellite continues to move. This happens because the manoeuvre generates a velocity, and no other forces are opposed to that.
Therefore, if we want to perform a rotation of a certain angle two manoeuvres are necessary: one to generate the angular velocity for starting the rotation, and the other to stop the rotation. The torques generate by the second manoeuvres shall have some magnitude of the first torque, but opposite sign. In addition, to compensate the angular velocity generate by the first command, the second manoeuvre shall have the same opening time.

To compute the required opening time to perform a manoeuvre of 90°, some preliminary tests are done. The tests are performed with a delay between the sending of first command and the second one of 100s. Thanks to these tests, it was possible to analyse how many times the satellite requires to reach a certain angle. Finally, the opening times to perform a rotation of 90° seem to be around 40s. In orbit operations, this computation is not necessary, the GNC system monitors and controls in real time the attitude.

Procedure:

- On MATLAB software to perform a sequence of two manoeuvres the open time variable is defined as a vector whom components have the same value of 40s. While the manoeuvre variable is a vector where the first component is 10 (‘*Positive rotation manoeuvre around Z axis’) and the second one is 11* (‘*Negative rotation manoeuvre around Z axis’)*

```
open_time = [40000, 40000]; %ms
manouver = [10, 11];          %refered to Table 12
pause_ms = 100000;          %time between the sending
                              of the two manouvres
```

Figure 52: MATLAB setting to performe a 90° rotation around Z axis.

- On RPi debug line, type *'.GNC/gncsimulator_sil'* to run the software for the simulation;
- Press *'Run'* on MATLAB: after the Ethernet connection, the first manoeuvre of positive rotation is sent, and the algorithm starts to analyse the status changes. After 100 seconds from the first command, the second manoeuvre is sent. Meanwhile the algorithm continues to receive the valves vector and compute the actual attitude of the spacecraft.

```
Wait for connection...
Connection Done
Manouver sent : 10 with 40000 s
Valves vector received..
```

Figure 53: MATLAB command window when the first manoeuvre of rotation is sent to propulsion module

Results:

MATLAB outcomes highlight that:

- the linear quantities remain constant during both the manoeuvres performances, this means that no thrust is generated;
- due to the first manoeuvre the torque T_z becomes 0.019 Nm; then it returned to 0 when the first manoeuvre is completed; during the second manoeuvre it reaches the value of -0.019 Nm; finally returns to 0 mN;
- the component w_z increases linear; when the manoeuvre is finished it has the value of 0,125 rad/s; when the second manoeuvre starts, to w_z decreases linearly and reaches the value of 0.003 rad/s ;
- the quaternion after 200s from the start of the simulation is equal to [0.7078, 0, 0, 0.7064], which corresponds to a rotation angle of 89,8855.

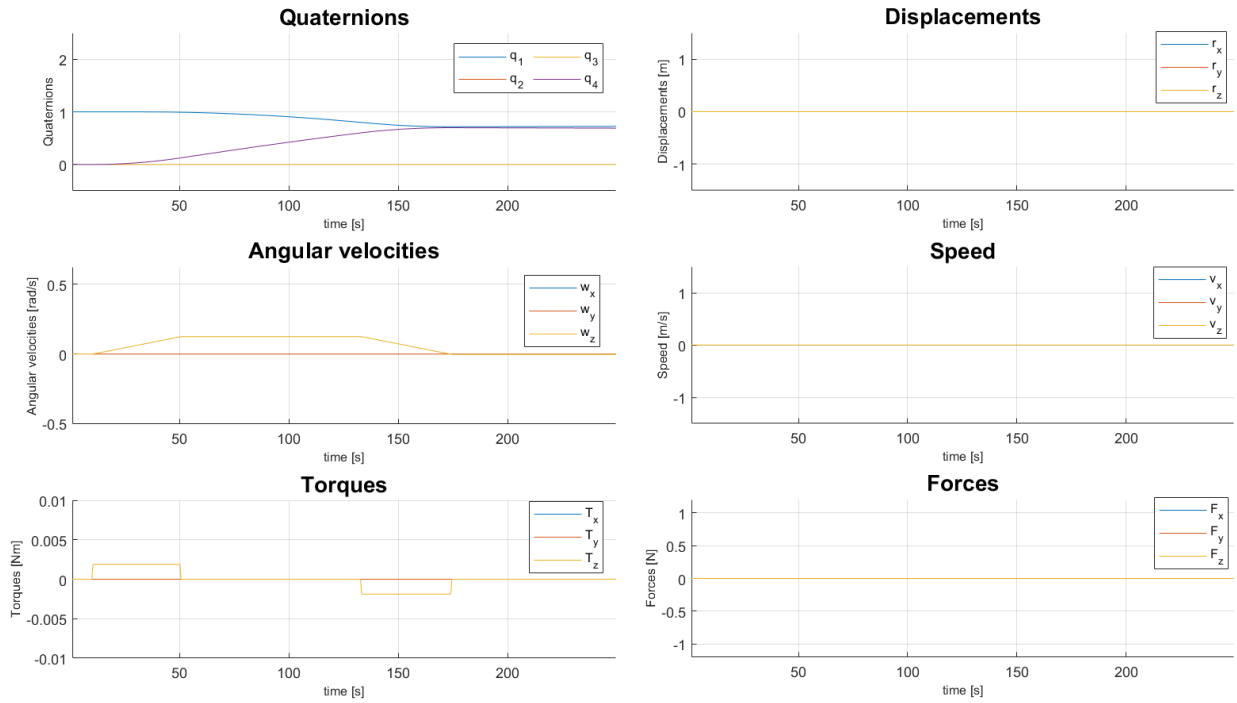


Figure 54: Linear and angular quantities behaviour of the rotation around Z

The results show as the manoeuvres are performed with good precision, at the end of the simulation the angular velocity is expected to return to 0, which does not happen. A small residual velocity of 0.003 rad / s remains, probably due to an inexact action of the MSP430 software or an inaccurate reading by the RPi. The rotation is also not perfect, at the end of the simulation it is an angle of 89.88 ° which will continue to decrease slowly due to the residual component of w_z. The positive notice is that no thrust or linear velocities are generated during the simulation, therefore, since when the satellite is in orbit, the attitude are monitored, controlled and piloted in real time thanks to GNC system, the results of this simulation is considered, however, positive from the point of view of performance of MSP430 software.

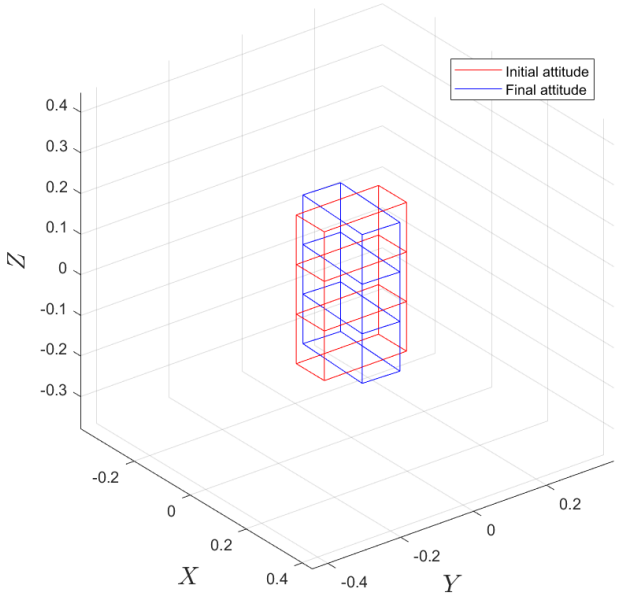


Figure 55: Rotation of 90° after the manoeuvre of rotation around Z axis

6 Conclusions

CubeSats are innovative platform that are changing the way to think about the space thanks to the paradigm “low cost and fast delivery” that get these spacecrafts and the related missions very affordable without losing in effectiveness of a space mission. CubeSats needs to improve their miniaturized technologies in order to support and/or compete with the bigger satellites. Among the enabling technologies in the CubeSats framework, the miniaturized propulsion represents a very important but challenging technology area.
This thesis moves within the development of a chemical miniaturized propulsion system that permits to perform a wide set of missions. Chemical propulsion system is very complex and requires a multidisciplinary approach, in which avionics covers an important role.
The thesis deals with the development of the control unit for the cold-gas propulsion system. After a careful analysis of the preliminary design of the system, a functional analysis and a selection of the main components are made. In this way it was possible to identify all the required functionalities and interfaces .
A development board was assembled to easily test each part of the software developed for the control unit.
To develop the code more easily, the software for the propulsion module is divided in units, each of them manages an interface or has a specific functionality. The debug and communication lines with the GNC were implemented; a message packet and a communication method with the GNC were defined; the DMA was configured to improve the communication performances; I2C driver was developed to read different sensors on the bus; to reach the temperature range for the optimal firing, a temperature control system was designed to actively control the temperatures; the algorithm to control the valves and heaters was designed.
When all parts were independently testes, each unit was integrated and tested. We have verified that the control unit loop lasts less than 2 ms, it widely respects one of the main requirements which requests to keep the loop as fast as possible. With the future implementation of other sensors or functionalities, the duration of loop will increase, but for the actual state of the system, we consider this a very good result.

Different kind of tests are implemented to test every aspect, interface and functionality of the developed control unit:

- Raspberry Pi is configured and interfaced with the development board in order to act as a GNC simulator. RPi requests a series of command to the MSP430 microcontroller and verifies that each of them is execution on the control unit.
- To test the temperature control algorithm is added the development board is improved with other hardware components: heaters that act as actuators of the control and sensors to get feedback on the temperatures trend;
- To validate the accuracy of the management of the valves a MATLAB simulation is developed. MATLAB implements the 6U CubeSat model to verify the theorical attitude and position of the satellite after a manoeuvre. The simulation performs the elementary manoeuvres (translation or rotation around one of the fundamental axes). Raspberry Pi is used to interface MATLAB pc with the control unit: it converts manoeuvres coming from MATLAB into commands for the propulsion module; and returns to MATLAB the reading status of the valves, in order to able the simulation to compute the position and attitude of the satellite in real-time.

After all of these tests have highlighted that each function and interface of the thruster module works, and the control unit has a rapid and accurate management of the valves.

The next steps will be the development of the hardware of the propulsion module: the design of the conditioning circuits for the valves and for the heaters; the implementation of voltage, current and pressure sensors; and the integration with the mechanical and fluidic parts. The MATLAB simulation can be improved with the introduction the models of the missing parts of the propulsion module, e.g. the valves and other actuators. These models should include non-linearities, delays, loss of charge, that should be compensated through a controller validated always in the MATLAB simulation. This controller have to monitor and control the manoeuvres in real time allowing to avoid and compensate the inaccuracies that we noticed during the tests.

7 References

[1] https://www.focus.it/scienza/spazio/che-cosa-sono-i-cubesat

[2] https://www.encyclopedia.com/science/news-wires-white-papers-and-books/what-delta-v

[3] https://artes.esa.int/projects/low-thrust-low-impulse-bit-reaction-control-thruster#:~:text=The%20minimum%20impulse%20bit%20(MIB,the%20satellite%20using%20the%20thruster.&text=Low%20thrust%20RCTs%20have%20been%20validated%20in%20the%20past.

[4] DIMEAS/ASSET/2019/091/ESA-SROC/TN/D4.2-TAD/I1R0 - SROC – Technology Assessment Dossier

[5] Paulina Swiatek, Luisa Innocenti , Andrew Wolahan, Fulvio Capogna, Antonio Caiazzo, and Christophe Vakaet - Design Principles for Sustainable Close Proximity Operations.

[6] Nguyen, Hugo; Köhler, Johan; Stenmark, Lars - The merits of cold gas micropropulsion in state of the Art space missions

[7] T4i - Propulsion system preliminary design

[8] https://ww1.microchip.com/downloads/en/DeviceDoc/22003e.pdf

[9] MSP430FR698x (1Mixed-Signal Microcontrollers datasheet (Rev. D)

[10] Raspberry Pi 3 Compute Module 3+ Datasheet - Ltd. 2019

[11] https://www.analog.com/media/en/technical-documentation/data-sheets/AD7414_7415.pdf

[12] https://www.tutorialspoint.com/operating_system/os_overview.htm

[13] http://www.farnell.com/datasheets/2849302.pdf

[14] https://users.obs.carnegiescience.edu/crane/pfs/man/Electronics/Omega-44000-thermistors.pdf

[15] Allen, Brandley – EtherNet/IP Socket Interface

[16] Novara C. – Non Linear Control And Aerospace Applications – Attitude Dynamics

[17] Novara C. – Non Linear Control And Aerospace Applications – Attitude Kinematics

8 Appendix A – Functional analysis

8.1 Functional analysis – Functional Tree, detail of "To manage system operations"

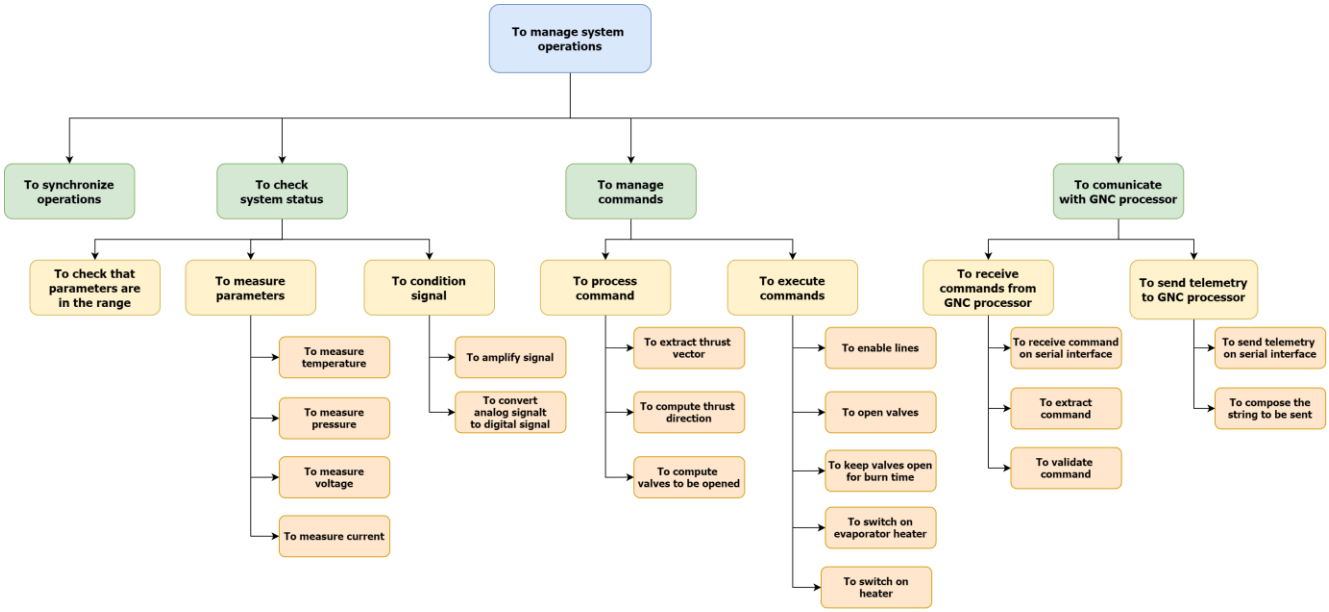


Figure 56: Functional tree: detail of 'To manage system operations' function.

8.2 Functional analysis - Product tree

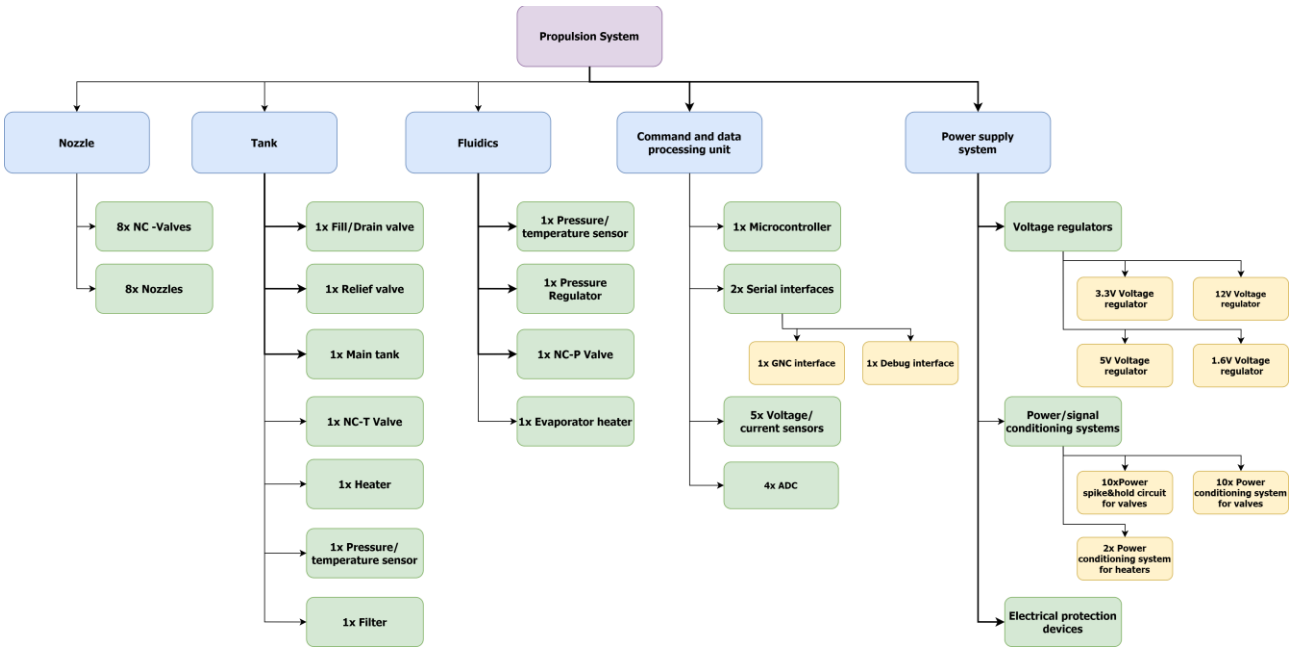


Figure 57 : Propulsion system product tree.

8.3 Functional analysis - N2 matrix

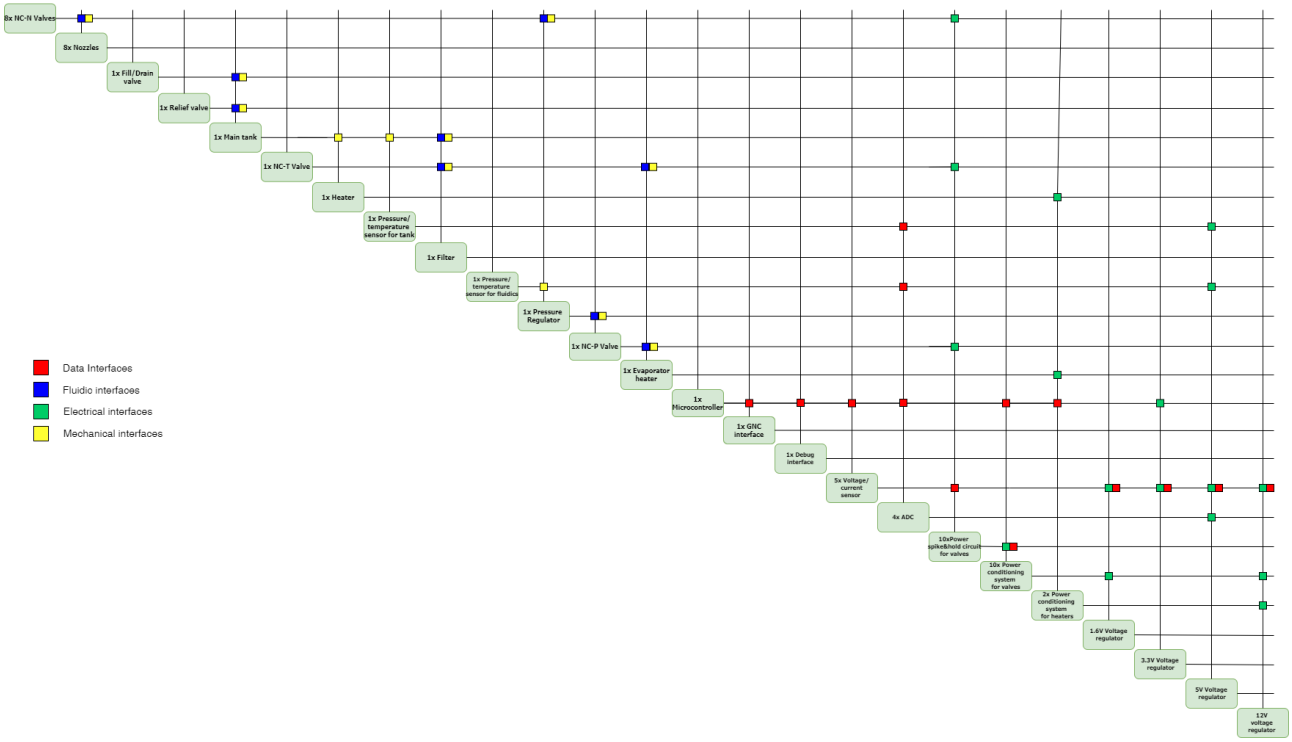


Figure 58 : Propulsion system N2 matrix

9 Appendix B – Requirements

ID	Description
-	The thruster module µC shall implement RS-422 protocol.
-	RS-422 parameters shall be: - 115200 baud rate, - no parity bit, - 8-bit data, - 1 start bit, - MSB-first data.
-	The thruster module shall provide at least 2 different signals in order to open each valve.
-	The valves status shall be monitored at least every 50 ms.
-	The control algorithm shall be able to change valves status at least every 50 ms.
-	The heart*beat valve* signal shall be activated continuously up to 600s.
-	The *heartbeat valve signal* shall have 100 ms period.
-	The *valves enable* signal shall be activated continuously up to 600s.
-	The thruster module µC shall be able to active heaters power line.
-	The thruster module µC shall provide enable signal in order to open valves
-	The thruster module µC shall provide heartbeat signal in order to keep valves open.
-	The thruster module shall keep valves closed if it not received burst command.
-	The thruster module shall acquire telemetries every TBD ms.
-	The thruster module shall monitor health status every TBD ms.
-	The thruster module µC shall acquire temperature measurement every TBD ms.
-	The thruster module µC shall acquire pressure measurement every TBD ms.
-	The thruster module µC shall acquire voltage measurement every TBD ms.
-	The thruster module µC shall acquire current measurement every TBD ms.
-	The thruster module shall acquire data from sensors on I2C bus.
-	The thruster module µC shall implement I2C protocol with: - 100kHz clock - 7-bit device address - 8-byte data - 1 stop bit - 1 start bit
-	The thruster module µC shall be set as master in I2C communication.
-	The thruster module shall check valve status every TBD ms.
-	The thruster module shall check heater status every TBD ms.
-	The thruster module shall switch on heaters in case of temperature lower than TBD °C.
-	The thruster module shall switch off heaters in case of temperature higher than TBD °C.

-	The thruster module µC shall configure ADCs with: - continuous conversion mode, - 12-bit conversion, - x1 PGA.
-	The current/voltage sensors shall be configured with: - TBD
-	The raw voltage data acquired from ADCs shall be converted into proportional pressure value.
-	The raw voltage data acquired from ADCs shall be converted into proportional temperature value.
-	The raw voltage data acquired from current sensors shall be converted into proportional current value.
-	The thruster module µC shall validate received commands from GNC controller.
-	The thruster module shall answer to GNC controller requests.
-	The thruster module shall communicate with GNC controller on RS-422.
-	The data/command packet shall have: - source, - destination, - message identifier, - command identifier, - data (more information about commands or telemetry), - crc8.
-	Telemetry packet shall include: -warning for voltage out of range; -warning for current out of range; -warning for AD7415 (temperature sensor) error; -warning for MCP34321 (ADC for voltage measurement) error; - heater command status ; - warning for tank temperature out of range; - warning for pipes temperature out of range; - warning for tank pressure out of range; - warning for pipes pressure out of range; - valves status, - heaters status, - temperature measurements, - pressure measurements, - voltage measurements, - current measurements.
-	The thruster module shall report on keep-alive message to GNC controller in case of non-nominal telemetry.
-	The thruster module shall check that pressure is in TBD range.
-	The thruster module shall check that temperature is in TBD range.
-	The thruster module shall check that voltage is in TBD range.
-	The thruster module shall check that current is in TBD range.
-	The thruster module shall receive thrust vector from GNC controller.
-	The thruster module shall receive torque vector from GNC controller.

-	The thruster module µC shall compute operation to be performed from thrust vector.
-	The thruster module µC shall compute operation to be performed from torque vector.
-	The thruster module shall provide to GNC controller the valve status.
-	The thruster module µC shall implement an algorithm to individual control each active element
-	External commands shall have higher priority than internal commands.
-	The control signal for valves shall refresh hardware timeout at least once per second.
-	Heart-beat signal shall be provided in order to keep valves open.
-	The GNC controller shall be able to override the current thruster module activities and shut down all the valves
-	The maximum latency between the GNC controller thrust command is received and the valves are opened shall be TBD ms
-	The maximum error on thrust duration shall be TBD ms.
-	The maximum latency between the first valve opening and the last valve shall be TBD ms.
-	The maximum latency between the first valve closing and the last shall be TBD ms.
-	The thruster module µC shall provide enable signal in order to switch on heaters
-	The thruster module µC shall provide heartbeat signal in order to keep heaters switched on.
-	The *heartbeat heater* signal shall be activated continuously up to TBDs.
-	The *heaters enable* signal shall be activated continuously up to TBD s.
-	The *heartbeat heater signal* shall have 100 ms period.
-	Period of heartbeat signal shall be less than 300 ms (nice to have 100 ms)

Table 14: Software requirements

10 Appendix C – Commands structures

10.1 Shut Down command

Feature: shut down all actuators and moved the propulsion system to *keep_alive* mode.

Structure:

src	*uint8_t*	61
dest	*uint8_t*	51
msg_id	*uint8_t*	id
cmd	*uint8_t*	10
data	-	No data needed
crc	*uint8_t*	crc

Table 15: Structure of 'Shut Down command'

PS answer:

- ACK
- NACK

10.2 Reboot command

Feature: reboot the propulsion system.

Structure:

src	*uint8_t*	61
dest	*uint8_t*	51
msg_id	*uint8_t*	id
cmd	*uint8_t*	15
data	-	No data needed
crc	*uint8_t*	crc

Table 16: Structure of 'Reboot command'

PS answer:

- No answer since the system is rebooting

10.3 Pre-Firing command

Feature: moves the propulsion system to *pre_firing* mode in order to prepare it for a possible thrust.

Structure:

src	*uint8_t*	61
dest	*uint8_t*	51
msg_id	*uint8_t*	id
cmd	*uint8_t*	17
data	-	No data needed
crc	*uint8_t*	crc

Table 17: Structure of 'Pre-firing command'

PS answer:

- ACK
- NACK

10.4 Change thresholds command

Feature: change the temperature thresholds used for the temperature control [Section 2.5.5]

Structure:

src	*uint8_t*	61
dest	*uint8_t*	51
msg_id	*uint8_t*	id
cmd	*uint8_t*	18
data	*float*	max threshold for tank sensor
	float	med threshold for tank sensor
	float	min threshold for tank sensor
	float	max threshold for pipes sensor
	float	med threshold for pipes sensor
	float	min threshold for pipes sensor
crc	*uint8_t*	crc

Table 18: Structure of 'Change thresholds command'

PS answer:

- ACK
- NACK

10.5 Firing mode check

Feature: ask to the propulsion module if it is in *firing* mode and able to thrust.

Structure:

src	*uint8_t*	61
dest	*uint8_t*	51
msg_id	*uint8_t*	id
cmd	*uint8_t*	83
data	-	No data needed
crc	*uint8_t*	crc

Table 19: Structure of 'Firing mode check'

PS answer:

- ACK
- NACK

10.6 Timing all valves command

Feature: send from GNC to propulsion system the command of thrust in terms of valves opening times.

Structure:

src	uint8_t	61
dest	uint8_t	51
msg_id	uint8_t	id
cmd	uint8_t	21
data	uint32_t	Valve 0 opening time
	uint32_t	Valve 1 opening time
	uint32_t	Valve 2 opening time
	uint32_t	Valve 3 opening time
	uint32_t	Valve 4 opening time
	uint32_t	Valve 5 opening time
	uint32_t	Valve 6 opening time
	uint32_t	Valve 7 opening time
	uint32_t	Valve 8 opening time
	uint32_t	Valve 9 opening time
crc	uin8_t	crc 8

Table 20: Structure of 'Timing all valves command'

PS answer:

- ACK
- NACK

10.7 Timing all heaters

Feature: send from GNC to propulsion system the command of switch on heaters in terms of switch-on times.

Structure:

src	uint8_t	61
dest	uint8_t	51
msg_id	uint8_t	id
cmd	uint8_t	23
data	uint32_t	Heater 0 switch-on time
	uint32_t	Heater 1 switch-on time
crc	uin8_t	crc 8

Table 21: Structure of 'Timing all heater command'

PS answer:

- ACK
- NACK

10.8 Telemetry request

Feature: GNC asks to PS the telemetry

Structure:

src	*uint8_t*	61
dest	*uint8_t*	51
msg_id	*uint8_t*	id
cmd	*uint8_t*	80
data	-	No data needed
crc	*uint8_t*	85

Table 22: Structure of 'Telemetry request'

PS answer:

- Answer telemetry
- NACK

10.9 Valves opening times request

Feature: ask to the propulsion module the valves residual opening times.

Structure:

src	*uint8_t*	61
dest	*uint8_t*	51
msg_id	*uint8_t*	id
cmd	*uint8_t*	81
data	-	No data needed
crc	*uint8_t*	crc

Table 23: Structure of 'Valves opening times request'

PS answer:

- Valves residual opening times

10.10 Heaters opening times request

Feature: ask to the propulsion module the heaters residual switch-on times.

Structure:

src	*uint8_t*	61
dest	*uint8_t*	51
msg_id	*uint8_t*	id
cmd	*uint8_t*	82
data	-	No data needed
crc	*uint8_t*	crc

Table 24: Structure of 'Heaters opening times request'

PS answer:

- Heaters residual opening times

10.11 ACK Answer

Feature: affirmative answer sent from PS to GNC.

Structure:

src	*uint8_t*	51
dest	*uint8_t*	61
msg_id	*uint8_t*	id
cmd	*uint8_t*	41
data	-	No data needed
crc	*uint8_t*	crc

Table 25: Structure of 'ACK Answer'

10.12 NACK Answer

Feature: negative answer sent from PS to GNC.

Structure:

src	*uint8_t*	51
dest	*uint8_t*	61
msg_id	*uint8_t*	id
cmd	*uint8_t*	42
data	-	No data needed
crc	*uint8_t*	crc

Table 26: Structure of 'NACK answer'

10.13 Telemetry answer

Feature: send to GNC the telemetry

Structure: the highlighted fields are more specified in Table 24, 25,26,27

src	uint8_t	51
dest	uint8_t	61
msg_id	uint8_t	Id
cmd	uint8_t	43
Data	uint8_t	**Status Field**
	uint8_t	**Operative mode**
	uint16_t	**Valves status**
	uint8_t	**Heaters status**
	float	Temperature sensor value [0]
	float	Temperature sensor value [1]
	float	Temperature sensor value [2]
	float	Pressure sensor value [0]
	float	Pressure sensor value [1]
	float	Pressure sensor value [2]
	float	Voltage sensor value [0]
	float	Voltage sensor value [1]
	float	Voltage sensor value [2]
	float	Voltage sensor value [3]
	float	Voltage sensor value [4]
	float	Current sensor value [0]
	float	Current sensor value [1]
	float	Current sensor value [2]
	float	Current sensor value [3]
	float	Current sensor value [4]

crc	uin8_t	crc 8

Table 27: Structure of 'Telemetry answer'

Status Field		
uint8_t	*uint8_t : 1*	*= 1 if voltages are out of range; else 0*
	uint8_t : 1	*= 1 if currents are out of range; else 0*
	uint8_t : 1	*= 1 if temperature is out of range; else 0*
	uint8_t : 1	*= 1 if temperature is out of range; else 0*
	uint8_t : 1	*= 1 if there is AD7415 failure; else 0*
	uint8_t : 1	*= 1 if there is MCP3421 failure; else 0*
	uint8_t : 1	*= 1 if temperature is in burst range; else 0*

Table 28: Structure of 'Status field' of 'Telemetry answer'

Value	Operative Mode
0	*keep alive mode*
1	*pre-firing mode*
2	*Firing mode*

Table 29: Values that Operative mode field can take

Valves Status		
uint16_t	*uint16_t : 1*	*= 1 if valve 0 is open; else 0*
	uint16_t : 1	*= 1 if valve 1 is open; else 0*
	uint16_t : 1	*= 1 if valve 2 is open; else 0*
	uint16_t : 1	*= 1 if valve 3 is open; else 0*
	uint16_t : 1	*= 1 if valve 4 is open; else 0*
	uint16_t : 1	*= 1 if valve 5 is open; else 0*
	uint16_t : 1	*= 1 if valve 6 is open; else 0*
	uint16_t : 1	*= 1 if valve 7 is open; else 0*
	uint16_t : 1	*= 1 if valve 8 is open; else 0*
	uint16_t : 1	*= 1 if valve 9 is open; else 0*

Table 30: Structure of 'Valves Status' field of 'Telemetry request'

Heaters Status		
uint8_t	*uint8_t : 1*	*= 1 if tank heater is switch on; else 0*
	uint8_t : 1	*= 1 if pipes heater is switch on; else 0*

Table 31: Structure of 'Heaters Status' field of 'Telemetry request'

10.1 Valves opening times answer

Feature: send to GNC the valves residual opening times.

Structure:

src	uint8_t	51
dest	uint8_t	61
msg_id	uint8_t	id
cmd	uint8_t	44
Data	uint32_t	Valve 0 residual opening time
	uint32_t	Valve 1 residual opening time
	uint32_t	Valve 2 residual opening time
	uint32_t	Valve 3 residual opening time
	uint32_t	Valve 4 residual opening time
	uint32_t	Valve 5 residual opening time
	uint32_t	Valve 6 residual opening time
	uint32_t	Valve 7 residual opening time
	uint32_t	Valve 8 residual opening time
	uint32_t	Valve 9 residual opening time
crc	uin8_t	crc 8

Table 32: Structure of 'Valves opening times answer'

10.2 Heaters opening times answer

Feature: send to GNC the heaters residual switch on times.

Structure:

src	uint8_t	51
dest	uint8_t	61
msg_id	uint8_t	id
cmd	uint8_t	45
Data	uint32_t	Heater 0 residual switch-on time
	uint32_t	Heater 1 residual switch-on time
crc	uin8_t	crc 8

Table 33: Structure of *'Heaters opening time answer'*

11 Appendix D – Tests reports

11.1 Test 1

```
Date: 2020-10-26   10:29

 TEST 1

 Sending command : TIMING ALL VALVES
       Set time : 2000 0 0 0 0 0 0 0 0 0
      Read time : 1955 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 0 2000 0 0 0 0 0 0 0 0
      Read time : 0 1952 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 0 0 2000 0 0 0 0 0 0 0
      Read time : 0 0 1955 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 2000 0 0 0 0 0 0
      Read time : 0 0 0 1950 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 0 2000 0 0 0 0 0
      Read time : 0 0 0 0 1947 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 0 0 2000 0 0 0 0
      Read time : 0 0 0 0 0 1951 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 0 0 0 2000 0 0 0
      Read time : 0 0 0 0 0 0 1957 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 0 0 0 0 2000 0 0
      Read time : 0 0 0 0 0 0 0 1955 0 0

 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 0 0 0 0 0 2000 0
      Read time : 0 0 0 0 0 0 0 0 1957 0

 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 0 0 0 0 0 0 2000
      Read time : 0 0 0 0 0 0 0 0 0 1943

 Sending command : TIMING ALL HEATERS
       Set time : 2000 0
      Read time : 1952 0
```

```
 Sending command : TIMING ALL HEATERS
      Set time : 0 2000
     Read time : 0 1951
```

Results: Test 1 PASS

11.2 Test 2a

```
Date: 2020-10-26   11:19
```

TEST 2a

```
 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 0
     Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
```

```
      Set time : 100 0 0 0 0 0 0 0 0 0
     Read time : 54 0 0 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 100 0 0 0 0 0 0 0 0
     Read time : 0 53 0 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 100 0 0 0 0 0 0 0
     Read time : 0 0 54 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 100 0 0 0 0 0 0
     Read time : 0 0 0 53 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 100 0 0 0 0 0
     Read time : 0 0 0 0 53 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 100 0 0 0 0
     Read time : 0 0 0 0 0 54 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 100 0 0 0
     Read time : 0 0 0 0 0 0 54 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 100 0 0
     Read time : 0 0 0 0 0 0 0 53 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 100 0
     Read time : 0 0 0 0 0 0 0 0 53 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 100
     Read time : 0 0 0 0 0 0 0 0 0 54

Sending command : TIMING ALL VALVES
      Set time : 1000 0 0 0 0 0 0 0 0 0
     Read time : 943 0 0 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 1000 0 0 0 0 0 0 0 0
     Read time : 0 943 0 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 1000 0 0 0 0 0 0 0
     Read time : 0 0 943 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 1000 0 0 0 0 0 0
     Read time : 0 0 0 942 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 1000 0 0 0 0 0
```

```
     Read time : 0 0 0 0 943 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 1000 0 0 0 0
     Read time : 0 0 0 0 0 942 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 1000 0 0 0
     Read time : 0 0 0 0 0 0 942 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 1000 0 0
     Read time : 0 0 0 0 0 0 0 943 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 1000 0
     Read time : 0 0 0 0 0 0 0 0 943 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 0 1000
     Read time : 0 0 0 0 0 0 0 0 0 956

Sending command : TIMING ALL VALVES
      Set time : 600000 0 0 0 0 0 0 0 0 0
     Read time : 597848 0 0 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 600000 0 0 0 0 0 0 0 0
     Read time : 0 597856 0 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 600000 0 0 0 0 0 0 0
     Read time : 0 0 597844 0 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 600000 0 0 0 0 0 0
     Read time : 0 0 0 597868 0 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 600000 0 0 0 0 0
     Read time : 0 0 0 0 597868 0 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 600000 0 0 0 0
     Read time : 0 0 0 0 0 597870 0 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 600000 0 0 0
     Read time : 0 0 0 0 0 0 597863 0 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 600000 0 0
     Read time : 0 0 0 0 0 0 0 597841 0 0

Sending command : TIMING ALL VALVES
      Set time : 0 0 0 0 0 0 0 0 600000 0
     Read time : 0 0 0 0 0 0 0 0 597850 0
```

```
 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 0 0 0 0 0 0 600000
      Read time : 0 0 0 0 0 0 0 0 0 597870

Results: Test 2a PASS
```

11.3 Test 2b

```
Date: 2020-10-26   14:24

 TEST 2b

 Sending command : TIMING ALL HEATERS
       Set time : 0 0
      Read time : 0 0

 Sending command : TIMING ALL HEATERS
       Set time : 0 0
      Read time : 0 0

 Sending command : TIMING ALL HEATERS
       Set time : 100 0
      Read time : 54 0

 Sending command : TIMING ALL HEATERS
       Set time : 0 100
      Read time : 0 54

 Sending command : TIMING ALL HEATERS
       Set time : 1000 0
      Read time : 943 0

 Sending command : TIMING ALL HEATERS
       Set time : 0 1000
      Read time : 0 936

 Sending command : TIMING ALL HEATERS
       Set time : 600000 0
      Read time : 597922 0

 Sending command : TIMING ALL HEATERS
       Set time : 0 600000
      Read time : 0 597910

Results: Test 2b PASS
```

11.4 Test 3a

```
Date: 2020-11-6   11:31

TEST 3a
```

```
 Sending command : TIMING ALL VALVES
       Set time : 0 0 0 0 0 0 0 0 0 0
      Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 10 10 10 10 10 10 10 10 10 10
      Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 20 20 20 20 20 20 20 20 20 20
      Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 50 50 50 50 50 50 50 50 50 50
      Read time : 0 0 0 0 0 0 0 0 0 0

 Sending command : TIMING ALL VALVES
       Set time : 100 100 100 100 100 100 100 100 100 100
      Read time : 54 54 54 54 54 54 54 53 53 53

 Sending command : TIMING ALL VALVES
       Set time : 200 200 200 200 200 200 200 200 200 200
      Read time : 161 161 162 162 162 161 161 161 161 161

 Sending command : TIMING ALL VALVES
       Set time : 500 500 500 500 500 500 500 500 500 500
      Read time : 450 450 450 450 450 450 450 450 450 450

 Sending command : TIMING ALL VALVES
       Set time : 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
      Read time : 942 942 942 942 942 942 942 942 942 942

 Sending command : TIMING ALL VALVES
       Set time : 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000
      Read time : 1942 1942 1942 1942 1942 1942 1942 1942 1942 1942

 Sending command : TIMING ALL VALVES
       Set time : 5000 5000 5000 5000 5000 5000 5000 5000 5000 5000
      Read time : 4932 4932 4932 4932 4932 4932 4932 4932 4932 4932

 Sending command : TIMING ALL VALVES
  Set time : 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000
 Read time : 9925 9925 9925 9925 9925 9925 9925 9925 9925 9925

 Sending command : TIMING ALL VALVES
 Set time : 20000 20000 20000 20000 20000 20000 20000 20000 20000 20000
Read time : 19890 19890 19890 19890 19890 19890 19890 19890 19890 19890

 Sending command : TIMING ALL VALVES
   Set time : 50000 50000 50000 50000 50000 50000 50000 50000 50000 50000
  Read time : 49786 49786 49786 49786 49785 49785 49785 49785 49785 49785

 Sending command : TIMING ALL VALVES
    Set time : 100000 100000 100000 100000 100000 100000 100000 100000
           100000 100000
```

```
    Read time : 99594 99594 99594 99594 99594 99594 99594 99594 99595
          99595

 Sending command : TIMING ALL VALVES
     Set time : 200000 200000 200000 200000 200000 200000 200000 200000
          200000 200000
     Read time : 199227 199227 199227 199227 199227 199228 199228 199228
          199228 199228

 Sending command : TIMING ALL VALVES
     Set time : 500000 500000 500000 500000 500000 500000 500000 500000
           500000 500000
     Read time : 498128 498128 498128 498128 498128 498128 498128 498128
           498129 498129

 Sending command : TIMING ALL VALVES
     Set time : 600000 600000 600000 600000 600000 600000 600000 600000
           600000 600000

     Read time : 597748 597748 597748 597748 597748 597748 597748 597748
           597748 597748

Results: Test 3a PASS
```

11.5 Test 3b

```
Date: 2020-11-6   11:2

 TEST 3b

 Sending command : TIMING ALL HEATERS
      Set time : 0 0
     Read time : 0 0

 Sending command : TIMING ALL HEATERS
      Set time : 10 10
     Read time : 0 0

 Sending command : TIMING ALL HEATERS
      Set time : 20 20
     Read time : 0 0

 Sending command : TIMING ALL HEATERS
      Set time : 50 50
     Read time : 0 0

 Sending command : TIMING ALL HEATERS
      Set time : 100 100
     Read time : 53 53

 Sending command : TIMING ALL HEATERS
      Set time : 200 200
     Read time : 141 141
```

```
 Sending command : TIMING ALL HEATERS
       Set time : 500 500
      Read time : 448 448

 Sending command : TIMING ALL HEATERS
       Set time : 1000 1000
      Read time : 945 945

 Sending command : TIMING ALL HEATERS
       Set time : 2000 2000
      Read time : 1933 1933

 Sending command : TIMING ALL HEATERS
       Set time : 5000 5000
      Read time : 4924 4925

 Sending command : TIMING ALL HEATERS
       Set time : 10000 10000
      Read time : 9915 9915

 Sending command : TIMING ALL HEATERS
       Set time : 20000 20000
      Read time : 19888 19888

 Sending command : TIMING ALL HEATERS
       Set time : 50000 50000
      Read time : 49782 49782

 Sending command : TIMING ALL HEATERS
       Set time : 100000 100000
      Read time : 99603 99603

 Sending command : TIMING ALL HEATERS
       Set time : 200000 200000
      Read time : 199243 199243

 Sending command : TIMING ALL HEATERS
       Set time : 500000 500000
      Read time : 498201 498202

 Sending command : TIMING ALL HEATERS
       Set time : 600000 600000
      Read time : 597821 597821

Results: Test 3b PASS
```

11.6 Test 4

```
Date: 2020-10-26   10:56

 TEST 4

 Sending command : TIMING ALL HEATERS
       Set time : 20000 20000
```

```
      Read status : 1 1

 Sending command : TIMING ALL VALVES
       Set time : 10000 10000 10000 10000 10000 10000 0 0 0 0
      Read status : 1 1 1 1 1 1 0 0 0 0


Send Shut Down command
      Read status : 0 0 0 0 0 0 0 0 0 0
      Read status : 0 0

 Sending command : TIMING ALL HEATERS
       Set time : 20000 20000
      Read status : 1 1

 Sending command : TIMING ALL VALVES
       Set time : 10000 10000 10000 10000 10000 10000 0 0 0 0
      Read status : 1 1 1 1 1 1 0 0 0 0


Send Reboot command
      Read status : 0 0 0 0 0 0 0 0 0 0
      Read status : 0 0

Results: Test 4 PASS
```

11.7 Test 5

```
Date: 2020-11-6   10:49

 TEST 5

 Sending command : TIMING ALL VALVES
       Set time : 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000
  Received time : 4757 5757 6757 7757 8757 9757 10757 11757 12757 13757

Telemetry :
      Valves status    :     1 1 1 1 1 1 1 1 1 1

      Heaters status   :     0 0

      Temperature      : 32.5649 33.5736 25

      Pressure         : 50 50 50

      Voltage          : 2.047 2.047 2.047 2.047 2.047

      Current          : 10 10 10 10 10

 Sending command : TIMING ALL HEATERS
       Set time  : 10000 20000
      Received time : 9746 19746

Results: Test 5 PASS
```