A Control System for an Open-Source Rocket Engine

McGill Global Challenges Award Internship

The Open Space Makers Federation

Mohamed Amine Elfelsoufi

Mentors:

Luca Litterio

Ruben Di Battista

August 2021

Table of Contents

Acknowledgements

I would like to thank McGill University for taking interest in my project and for making this internship possible. I would also like to express my gratitude to the Open Space Makers Federation for hosting my internship.

I wish to extend my gratitude to Team Eve who consistently provided their support along this challenging journey.

Special thanks to:

***Mr. Ruben Di Battista**, for pushing me to explore new horizons and surpass my limits and for being always there when I needed help.*

***Mr. Luca Litterio**, for providing guidance and advice and for designing the layout of the control system.*

***Mr. Francesco Ieverse**, for helping me develop my critical thinking and for providing key advice for the development of the control system.*

***Mr. Ricardo Albi,** for providing guidelines and assistance and for helping me troubleshooting and finding solutions.*

Introduction

The goal of this internship is to design a complete control system for Eve: an open-source liquid rocket engine. The control system consists of the following components:

1. A data acquisition system (DAS). The micro-controller should collect temperature, pressure and force measurements from the engine and its tanks and transmit it wirelessly over Wi-Fi to a remote station. Optionally, the control system should also include a dashboard that would run on the remote computer to receive and store the data generated.

2. A control system. The control system was required to control the valves and the igniter and to be able to abort the engine in case an anomaly was detected.

Initially, it was planned that the control system would consist of a single microcontroller ESP32. It was also agreed that the control system will be written in C++.

The data acquisition system was planned to include a temperature sensor, a pressure sensor and a load sensor. The data would be transmitted on MQTT with the intermediary of a regular Wi-fi router.

For the design process, I acquired the necessary equipment to recreate the data acquisition system designed by Mr. Luca Litterio. This consisted of the sensors, their respective amplifier cards as well as the power sources required to power them. To keep the initial design simple, I employed a ready-to-use ESP32 development board to fulfill the role of a microcontroller.

For the testing part, I acquired the equipment to test every sensor individually. To test the control system, I made an (almost) faithful reproduction of the oxidant feedline of the rocket engine. This consisted of a ball valve and a solenoid valve, as well as the pipelines required to create a fluid circuit and an air compressor.

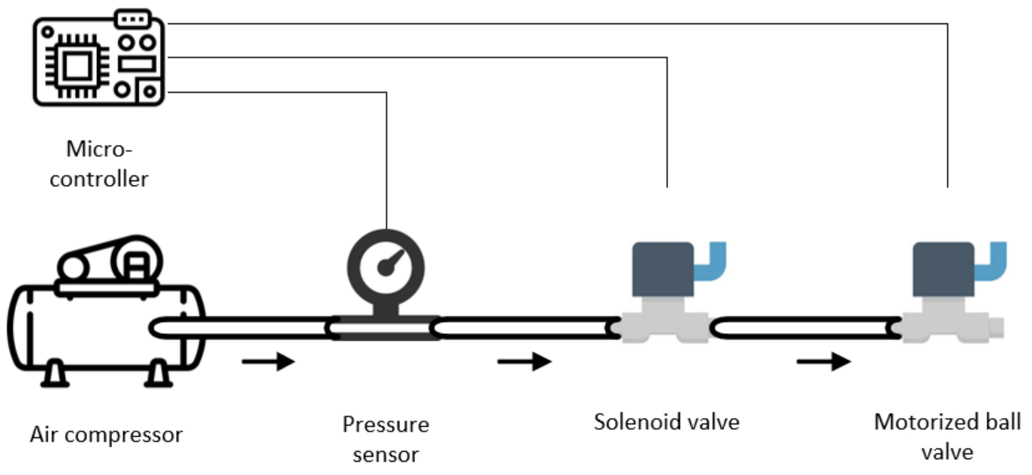

Figure 1. The air circuit used to simulate the engine’s oxidant feedline.

The Hardware Configuration of the Data Acquisition and Control System

The limitations of ESP32's wireless communication capabilities constrained us to use two microcontrollers. This allows us to free more computing power for independent tasks. At the same time, this allows us to reduce the complexity of the system's electrical wiring thus simplifying the assembly and the testing of the engine. The two microcontrollers that constitute the control system have distinct roles which are discussed more in detail below.

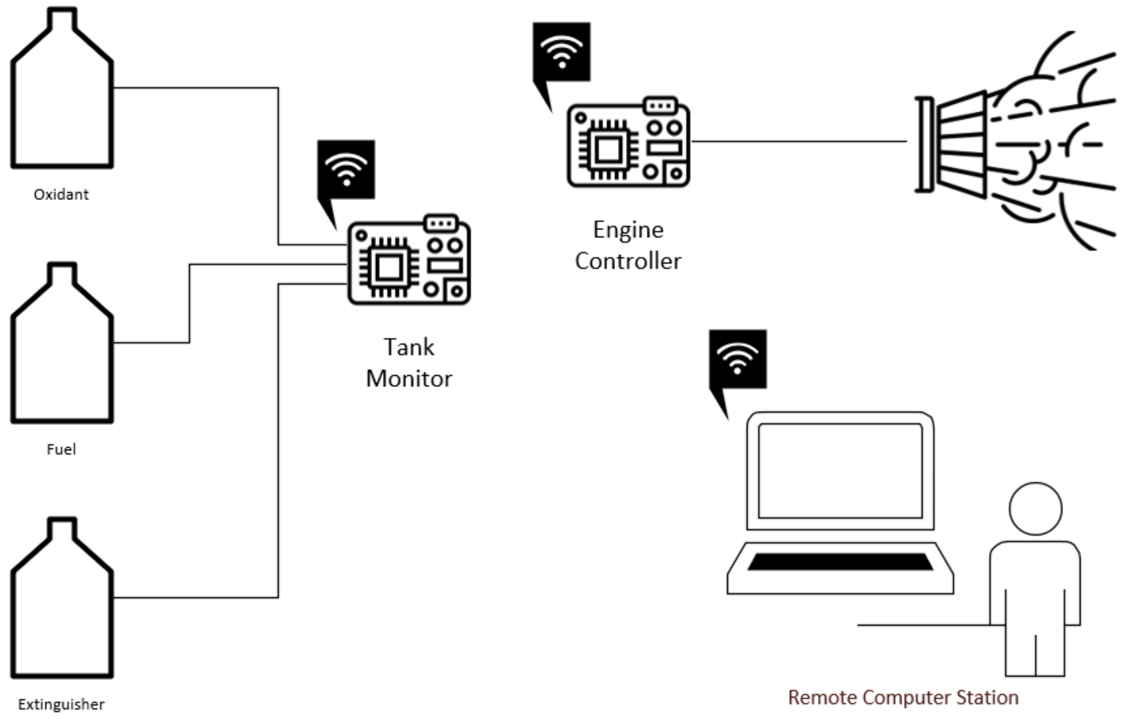

Figure 2. The data path of the rocket engine.

The Engine Controller

The engine controller is an ESP32 microcontroller that acts as the main controller of the rocket system. It entirely assumes the responsibility of controlling and monitoring the engine. This microcontroller also contains the list of the modes of operation—a series of predetermined tasks that the engine follows before and during combustion.

Data Acquisition

As one of the elements that constitute the data acquisition system, the Engine Controller monitors continuously the temperature, pressure, and force output by the engine. This information is transmitted wirelessly via MQTT to the computer station. By having the engine controller directly sample data from the engine, we can make sure that it performs its most important task in a timely fashion. Namely, controlling the combustion.

Control

The main purpose of the engine controller is to control the operation of the engine. The engine controller is connected to the valves and can therefore control the flow of fluids to the engine. As a result, the engine controller is the only microcontroller that can truly abort or terminate the system. The engine controller also receives messages from the tank monitor and from the computer station via MQTT to decide on the course of action to take.

The Tank Monitor

The tank monitor is a data acquisition microcontroller whose role is to collect data from the tanks, process it and assist the engine controller in its decision making.

Data Acquisition

The tank monitor is planned to have three pressure and three load sensors so that it can monitor each of the oxidant, fuel and extinguisher tanks. The tank monitor is planned to include simple algorithms to process and interpret the data collected. The results are then broadcasted via MQTT to assist the engine controller during pre-combustion tests or to abort the system if the tanks show abnormal conditions.

MQTT Wireless Communication

Wireless communication is used for both data acquisition and control. It transports the data collected from the microcontroller to the computer station and it is the intermediary by which the engine controller receives instructions and crucial information for decision making.

Although wired communication is faster and less susceptible to interference, wireless communication was chosen because it makes hot-fire tests safer. Furthermore, wireless communication is the only viable solution in the medium to long term for a rocket engine.

MQTT local network

A local network is established using a router. The router creates a Wi-Fi network and acts as the intermediary between the computer station and the ESP32 microcontrollers.

Using an MQTT broker, the computer station can communicate on the local MQTT network. The broker I used during this internship is Mosquitto.

MQTT with ESP32

The ESP32 microcontroller used for our design has an internal antenna, therefore no additional equipment is required for wireless communication. For the software, the ESP-IDF development framework provides an MQTT library that allows ESP32 to send and receive messages on MQTT. ESP-IDF MQTT library presents several characteristics:

- *It can recover from most communication errors.* After thorough tests, it seems that the MQTT software can recover from most error events (router power off, broker disconnected). For example, if the router is turned back on, the software is able to reconnect and resume operationAs an additional safety measure, I added to the control software awareness of the current state of connectivity. However, the library cannot recover from certain errors (like socket selection timeout error). This forced me to use a large timeout of 5 seconds. Because of the design of the MQTT library, this means that ESP32 cannot detect an MQTT disconnection before 5 seconds.
- *It allows for a transmission rate of 600 to 800 messages per second*. This limits to three the number of pressure sensors that an ESP32 can operate at the maximum sampling rate. This is the reason why an auxiliary ESP32 Tank Monitor was included in the design.
- *It is a ready-to-use library, which saves development time*. It can also be very easily modified and improved.

I have also tested this MQTT library for long-duration operations. I connected three sensors to my ESP32 and configured them to transmit at their maximum speed. The system operated for over an hour without aborting. For this reason, although ESP32 MQTT error recovery can be improved, in its current state, it is currently a befitting solution.

MQTT Data Organization

To facilitate data storage and message interpretation on the ESP32 side, I have implemented, on top of the MQTT layer, a standard format for messages that are transmitted on the local MQTT network.

In the MQTT protocol, each message published contains a topic and a message content (the message itself). The format I proposed is as follows. The content of every MQTT message contains only crude data. For example, if ESP32 transmits a message containing the current pressure of the engine, the message content will only include the numerical values, e.g: "9.05". I chose to include the relevant information to identify the message in the *topic*. The structure of the message topic is shown in Figure 3.

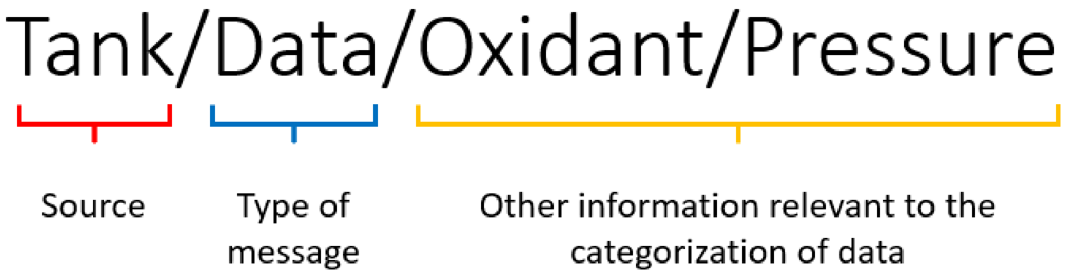
Tank/Data/Oxidant/Pressure
Source
Type of message
Other information relevant to the categorization of data

Figure 3. The structure of MQTT message topics.

For our design, message topics comply with the following structure. The first word identifies the source of the message. In the example illustrated in Figure 3., the sender of this message is the tank monitor. The second word in the topic identifies the type of data being transmitted. This can be "Data", "Error" or "Log". The rest of the topic contains additional information about the data sent. Essentially, the topic is a header that helps the receiving device to easily organize and store the data. Moreover, this structure facilitates message interpretation. For example, if a message has a topic including the word "Data", the receiver will know it needs to save it. On the other hand, if the message topic contains the word "Error", the receiver will know that it needs to interpret the message data, as the message might be, for example, an abort request.

The Data Acquisition System

The role of the data acquisition system is to collect information about the state of the rocket system and to help us characterize the combustion process.

The Distribution of the Data Acquisition Task

Because of the limitations of MQTT and to save as much computing power as possible for the control tasks, it was decided during the internship that the burden of data acquisition will be split between two microcontrollers. The main microcontroller collects the data from the engine. The secondary microcontroller collects the data from the tanks. The distinct role of these two microcontrollers was discussed details in a previous section.

The Data Acquisition Process

The data acquisition system consists of three main components. Namely, the temperature, load and pressure sensors that collect data. What follows is a summary of the characteristics of each sensor as well as a brief description of how they communicate with ESP32.

Although optional, I have also decided to create a simple dashboard application that runs on a personal computer (playing the role of the computer station). The dashboard is a crucial component of the data acquisition system because it allows the human operator to see the data generated by the engine system. It also the data generated during operation and saves it to a file so that it can be reviewed later.

The temperature sensor

The temperature sensor employed in this system is a thermocouple of type K because of its resistance to high temperatures (up to 800 Celsius).

To interface it with ESP32, we use a MAX31856 thermocouple amplifier. MAX31856 has a maximum sampling rate of 12 samples per second and communicates with the microcontroller using SPI. ESP-IDF development framework provides SPI drivers which makes communication with MAX31856 easier. For the design of our engine, we use an Adafruit MAX31856 breakout which includes an independent temperature sensor mounted on board. Therefore, we have two temperature sensors that allows us to know the temperature of both the engine and the control system at any time. To test the temperature sensor, I used ice and boiling water.

The pressure sensor

To measure the pressure in the combustion chamber, a high-temperature pressure sensor is required. For this reason, we use an HDP501 pressure transmitter. With this sensor, we can sample pressure in the combustion chamber at a maximum rate of 160 samples per second. To read data from the pressure sensor, we use the ESP32 ADC channel which can read the voltage output by the pressure sensor. Using a simple formula, we can transform the voltage reading into a pressure value.

To test the pressure sensor, I used the air compressor and fit the pressure sensor between the compressor and a closed valve. Since the HDP501 sensor I received outputs current instead of voltage, I couldn't test this specific sensor. Instead, I tested another pressure transmitter which ouputs voltage values, just like the voltage model of HDP501 normally would. Compatibility with a current outputting HDP501 is yet to be verified as it requires a chip that can convert a current signal to an analog voltage signal (ESP32 cannot read current signals by itself). An INA219 was a suggested solution but it is not yet tested.

The load sensor

The load sensor allows us to determine, among other things, the force output by the rocket engine. The load sensor signal is amplified using an HX711 which allows for a selectable sampling rate of 10 or 80 samples per second. To communicate with ESP32, HX711 requires a custom clock signal input that was simple to code and test. The load sensor was tested using weights with known values.

The dashboard

The dashboard is a simple python script that runs on the computer station to receive data sent by the microcontrollers and save it to a CSV file. To do so, it subscribes to the Mosquitto broker as a client. Whenever the mosquito broker receives a message, it is passed on to the dashboard. The dashboard applies a time stamp on the new message and saves its content to a file located on the desktop.

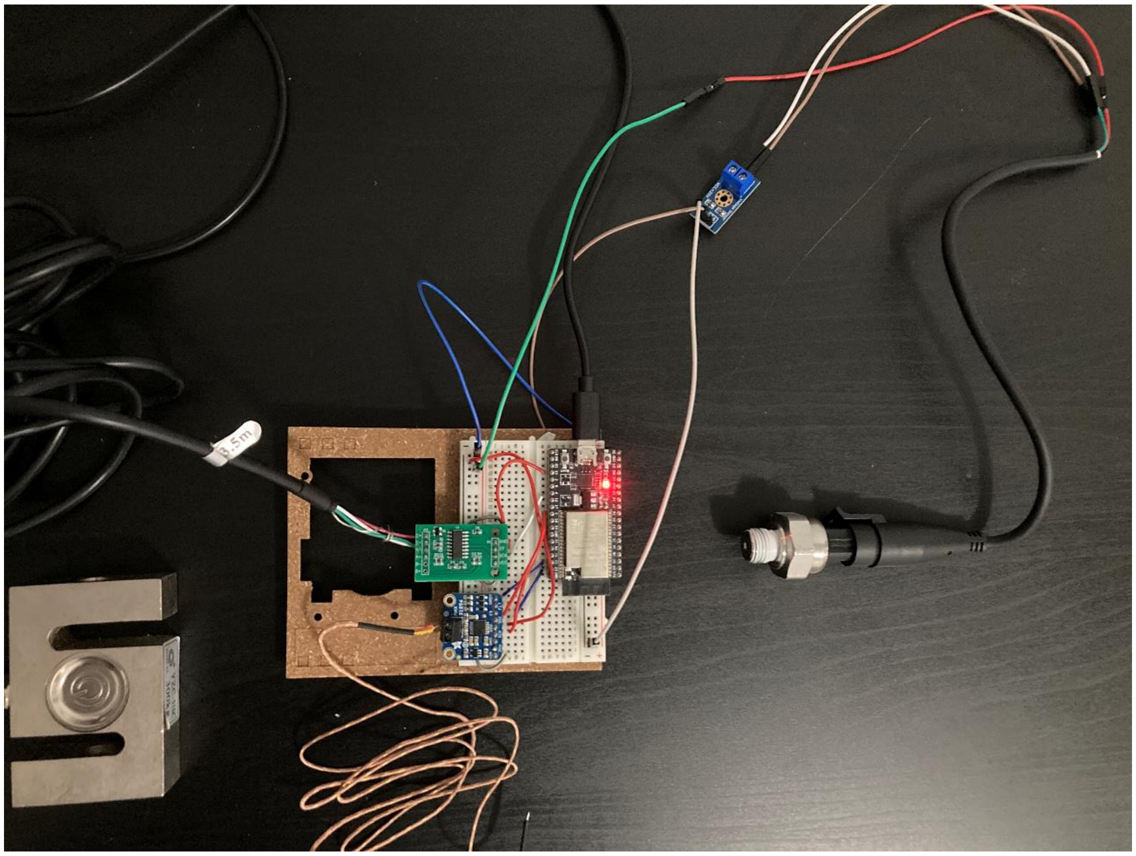
Figure 4. The three sensors which constitute the data acquisition system.

The Control System

The control system is composed of three different parts:

1. The operation manager. This is an algorithm that runs every mode of operation, one at a time, and determines the course of action to take depending on the current situation. The operation manager is the heart of the control and data acquisition system. It has access to all control, data acquisition and wireless communication functions and uses them to operate the engine.
2. Flow control. By controlling the valves, the control system can control the amount of fluid that can flow into the combustion chamber.
3. The abort system. This component is part of the previous two. Its role is to check for abort conditions and abort the system if needed.

The Operation Manager

The operation manager operates the engine by following a pre-determined list of instructions. These instructions are grouped into steps named “modes of operation”. The following is a summary of each mode of operation.

Modes of Operation

1. *Start setup*

   This function starts the wireless communication and the data acquisition system. If a wireless connection cannot be established, the system is immediately aborted. Otherwise, the function proceeds to start Safe Mode. Safe Mode is a special mode of operation. It is a function that runs continuously, non-stop, and it hosts the Abort System which will be discussed more in detail below. If the *Start Setup* mode was executed correctly, the operation manager proceeds to the next mode of operation: Functional Check Pre-filling.

2. *Functional check pre-filling*

   The purpose of this function is to verify that the system is working correctly before filling the tanks. The function starts by initializing the Flow Control System. This initializes communication with the ball valve and starts a task that continuously sends the current position of the valve on wifi. Afterwards, the function proceeds to verify that all the valves can open or close correctly. Using the feedback signal of the ball valve, the function conducts a quick test to verify that the ball valve can be controlled accurately. If this cannot be done successfully the algorithm will send an error message and repeat the ball valve test until it succeeds. Currently, there is no pre-filling solenoid valve test. If the *pre-filling functional check* is done correctly, the operation manager proceeds to the next mode.

3. *Functional check post-filling*

(This function is currently unimplemented). The post-filling functional check will verify that the valves can open and close correctly by detecting a pressure drop in the tanks. A very simple algorithm in the Tank Monitor microcontroller will be written to send a message when it detects a pressure drop.

4. *Tank Check*

This function is to be implemented.

5. *Feedline chill down*

This function is to be implemented.

6. *Combustion*

(This function is currently unimplemented). This will consist of a simple series of instructions. The Engine Controller will enter combustion mode. Then the igniter will be triggered, and the system will remain in combustion mode for a determined amount of time. When this function returns, the operation is completed, and the system will abort automatically.

Flow control

The engine controller can decide how much of each fluid can flow into the engine by controlling the corresponding valves. Currently, our control system can control two types of valves: 5 wires ball valves and solenoid valves.

Ball Valve

The ball valve I used during this internship is a five-wires current-controlled motorized ball valve that has a PWM feedback signal. To control the valve, a signal in the range of 4 to 20 mA is sent to the valve. At 4 mA the valve is completely closed. When this value is exceeded, the valve partially opens. At 20 mA the valve is completely open. Because of that, it is possible to control the flow of each fluid. However, since ESP32 is not powerful enough, it cannot output a current strong enough to open the valve more than 60%. Many solutions were proposed to solve this problem, but they have yet to be tested.

The ball valve outputs a PWM feedback signal which indicates the position of the valve. This way it is possible to determine how much fluid can flow through the valve. To read the PWM signal, it is necessary to know its frequency. Because different ball valves can have different frequencies, the flow control system has a function that can detect the frequency of a PWM signal. However, I have later discovered that it is (easier and) more reliable to determine the PWM frequency experimentally using a logic analyzer or an oscilloscope.

Solenoid Valve

Solenoid valves are used as emergency shutdown actuators. The very rapid execution time of the solenoid valve allows us to have a very quick response in case an abort is needed.

To control a solenoid valve, we use a relay module. By interfacing with the relay module, we can turn on or off the solenoid valve, effectively opening or closing it. The software for this is very simple.

By coupling a normally closed solenoid valve with a relay module, we can make sure that if the external DC power source or the microcontroller loses power, the solenoid valve will immediately close. This makes the system much more secure. Inversely, for the extinguisher tank, we can potentially have the opposite configuration so that the valve will open in case power is lost.

Abort System

As mentioned previously, the abort system is a mode of operation which runs continuously. It is also known as the Safe Mode. This function is the only one that can directly abort the system. Its main function is to monitor abort conditions. If Wi-Fi or MQTT connection is lost and cannot be recovered, or if an abort has been requested, Safe Mode will immediately terminate all tasks and stop the system. Although it isn’t implemented yet, the abort procedure will also include closing the fuel and oxidant valves and opening the extinguisher valve.

Conclusion

At the end of the internship, I have overall achieved my objectives. Although they are not perfect, the wireless communication and data acquisition parts of the control system are completed and functional. The control system is, for the most part, completed. We can control the valves and abort the system. The modes of operation, which are not finished, are, for the most part, extra features.

For the future, and as a member of the Eve control system team, I would like to fix all the minor issues mentioned in this report. I would also like to design a PCB for the microcontroller I am currently using. This way, the control system will be more robust and easier to assemble.

This summer internship at the Open Space Makers was a very enriching experience for me. It transformed me from a clueless student to a confident electrical engineer. Once again, I would like to express my gratitude to McGill University, the Open Space Makers and all the members of the team Eve who contributed to make this internship an invaluable experience.

Appendix