## **Software Design**

In the following section, an overview of the software modules that will be used is provided.

### **Purpose**

Software both on the on-board part of the experiment and the ground station is going to be used for different reasons.

* The **on-board system** is going to :
  + Monitor the heaters’ temperature (when to turn on and off).
  + Be responsible for the movement of the emulsification motor.
  + Turn on and off the I-VED technique.
  + Store the data from the I-VED technique in a logfile in the SD card.
  + Store the data from the pressure and thermal sensors in a logfile in the SD card.
  + Create a collective logfile where all of the data is going to be stored in the SD card.
  + Control the Camera and store the video data in the camera’s SD card.
  + Send a sample of the data of the sensors and the system events to the ground station.
* The **ground station** **system** is going to :
  + Receive the sample data from the on-board system using the rocket’s Downlink.
  + Include a GUI for the visualization of the data during the flight.
  + Log-Backup the sample of data that will be sent from the rocket.

### **Design**

#### **Process Overview**

As seen in the diagram below, the microcontroller will receive data from the two sensor sets (3 thermal + 3 pressure digital ones) which are going to be directly connected to it and according to them it will adjust the Heaters’ temperature while also storing their data in the SD card. Meanwhile, it will also be able to control the motion of the DC Motor using the Motor Controller and communicate with the Camera and Sound Card (for the I-VED technique) in order to turn them on and store their data in the SD Card. The Downlink will be used in order to send a sample of data to the ground station.

All of the components will be connected to the Raspberry communicating with it using the GPIO pins of it except for the camera that will use a USB cable.

**Diagram

Description automatically generated**

**Figure 4.24: Block diagram of the experiment components.**

#### **General and safety related concepts**

As shown in the 3.5.2 Risk assessment paragraph, the probable errors or safety issues that Software subteam has to be concerned over are :

* Microcontroller failure
* Connection Loss between the rocket and the ground station
* Sensor Software Failure

##### *4.8.2.2.1 Microcontroller Failure or Breakdown*

This error’s probability is extremely low, but if something like that happens the consequences are critical for the experiment. Without the microcontroller the data from the sensors, the control of the motor and the sound card are non-existent, and depending on the timing of the occurrence, it could be completely fatal for the experiment since the camera could never be signaled to start recording.

The ways to mitigate the probability of this happening are :

* Buying the product from a well-known and trusted vendor, in order to avoid badly made soldering or PCB printing, mistreatment of the product, and in order to have possible customer support in case something occurs.
* Testing it under extreme conditions and avoid mistreating or endangering it, so that we can be confident enough that the microcontroller won’t misbehave during the flight.

##### *4.8.2.2.2 Connection Loss*

This includes both total and temporary connection loss between the rocket and the ground station.

In case of total connection loss, no data will be sent to the ground station and thus the team will have no clue on the status of the experiment. There, also, will be no way to backup the data.

On the other hand, a case of a temporary connection loss equals a partial loss of data, which in this case will not be so severe, because the only reason the data is sent to the ground is to monitor the experiment and have a backup option for a sample of the data.

The probability of such a thing happening is quite high but manageable up to a point with Go-Back protocols and algorithms.

The actions that can be taken in order to avoid these problems are:

* Testing the code extensively in order to make sure that the problem is not created by itself.
* Testing different kinds of situations and try to come up with error handling scenarios for these.
* Testing with the rocket system mounted to see if some of the problems are created because of other connectivity issues besides the code itself or whether some problems are created because of different protocols.

##### *4.8.2.2.3 Sensor Software Failure*

Sensor software failure can be considered highly unlikely but its consequences can range between highly dangerous to the experiment to just a partial distortion of the final data. Therefore it creates an issue that should be addressed.

There are very few ways to mitigate the possibilities of such a thing happening, mainly because software vendors are taking care over these issues and publish new updates regularly, but these are some of the ways :

* Testing the code extensively for possible unknown mistakes or exceptions happening,
* Using well known libraries in order to avoid instabilities in the code,
* Update libraries frequently and test the code on the newest updates.

#### **Interfaces**

In terms of the interfaces:

* All of the experiments’ components will be wired-soldered together except for the camera whose connection will be defined later.
* The experiment will connect to the Downlink of the rocket using an RS-422 port. The data bandwidth usage is described below in the following tables.

**Table 4.10: Data to be sent through the Downlink.**

| **Data to be sent through the Downlink** | | |
| --- | --- | --- |
| Data | bits | Frequency |
| Thermal Sensor 1 | 16 | 3 |
| Thermal Sensor 2 | 16 | 3 |
| Thermal Sensor 3 | 16 | 3 |
| Pressure Sensor 1 | 16 | 3 |
| Pressure Sensor 2 | 16 | 3 |
| Pressure Sensor 3 | 16 | 3 |
| Motor Status | 8 | 1 |
| Heater Status | 1 | 1 |
| Sound Card Status | 1 | 1 |
| Camera Status | 8 | 1 |
| Error | 8 | 1 |
| Sum : | 122 | Max : 3 |

**Table 4.11: Example of data link protocol structure.**

| Sync1 | Sync2 | MsgID | MsgCNT | Data | CSM | CSM | CRC | CRC |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 8 | 8 | 8 | 122 | 8 | 8 | 8 | 8 |

**Table 4.12: Calculation result.**

| Baud Rate (bits/s): | 558 |
| --- | --- |

Which derived using the the following formula:

In kB/s that is roughly 0.07kB/s as a maximum Bandwidth.

#### **Data acquisition and storage**

The data collected are going to be split into 3 different categories:

* Video data, which are going to be stored separately inside the camera in order to avoid unnecessary connectivity issues or breakdowns in the connection,
  + The video file size is approximately going to reach 4-5 GBs, but further research is needed because the size depends on many factors, mainly on the technical details of the camera.
* Sound Card data, of the I-VED technique, which are going to be stored inside the the SD card of the Raspberry Pi,
  + The size of this file ranges between 164 and 52,626 kBytes because the frequency in which the Sound Card will be functioning hasn't been defined yet. But it definitely can be stored inside the SD card of the Raspberry Pi.
* Sensor and  Event data, which are going to be stored in the same folder as the Sound Card data but will not have any correlation with them.
  + In the image below we can deduct that the total storage size used up by the logfiles is 117kB.

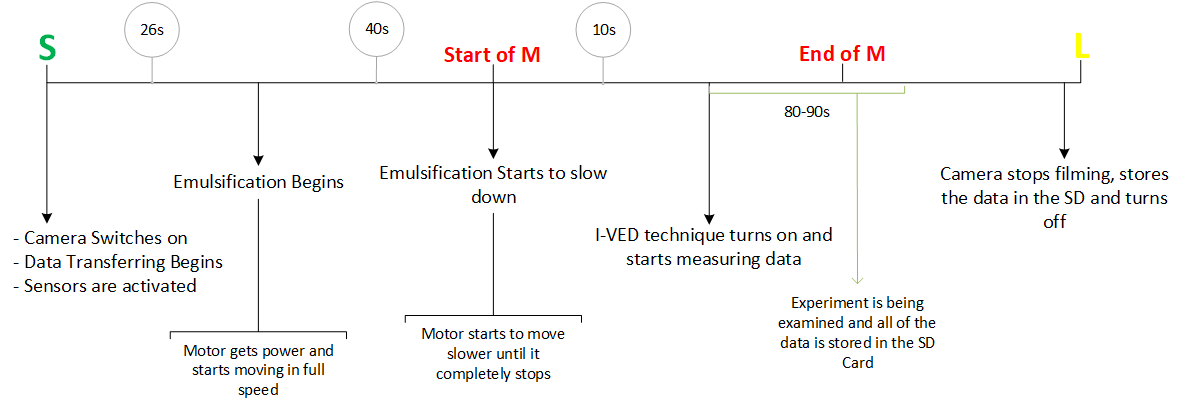
Εικόνα που περιέχει πίνακας

Περιγραφή που δημιουργήθηκε αυτόματα

**Figure 4.25: A simulation of the final data logged in the Raspberry Pi storage.**

Apart from this data, a backup logfile will be kept on the ground station from the sample of the data that will be sent from the experiment. Since the data sent is the CSV with the collective sensor and event data, the size of it is going to be about 57kB.

#### **Process Flow**

****

**Figure 4.26: Experiment timeline.**

#### **Modularization and pseudo code**

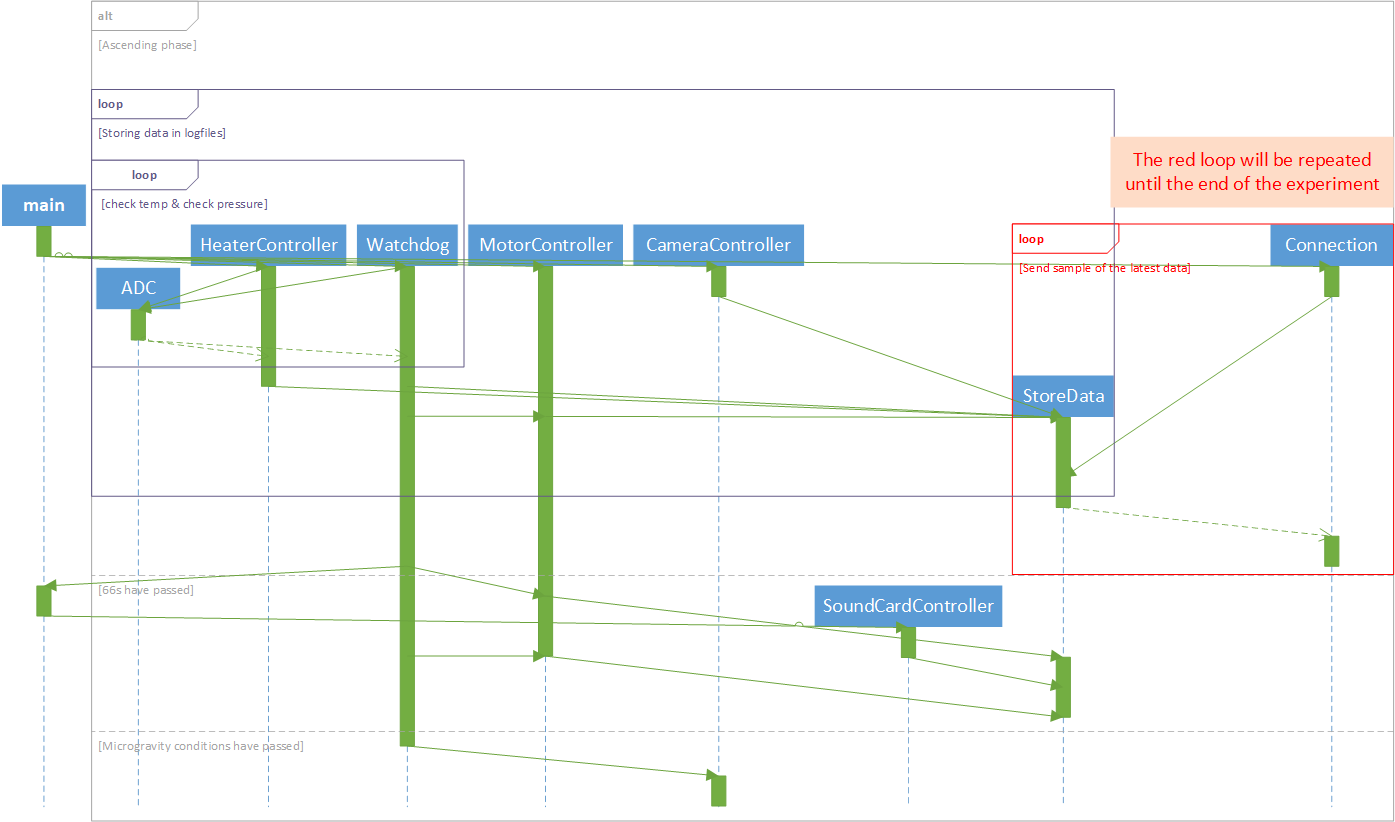
The code of the experiment used on the rocket is going to be separated into several different modules which can be seen in the diagram below. The `main.py` will initialize all the component Controllers, the Connection and the `Watchdog.py` in different threads. These threads will be able to draw and message data, when needed, to each other and they will all use the `StoreData.py` module in order to save their data in the SD card. `Connection.py` module will get a sample of this data from `StoreData.py` in order to send it to the ground station. ‘Watchdog.py` module is going to be really important in this setup as it is the one that will judge whether the temperature or the pressure in the rocket’s module is too high and stop the motor accordingly, it will also synchronize a timer with the rocket’s signals using the Raspberry’s IO pins.

As a backup solution, on the ground the sample of the data, sent from the rocket, will be saved by writing all the data on a logfile.

**Timeline

Description automatically generated**

**Figure 4.27: UML of the relations between different modules.**



**Figure 4.28: Sequence UML that explains the usage timeline of each module.**

The UML Sequence diagram above describes the usage of each class mentioned before, according to the Timeline represented in the 4.8.2.5paragraph

### **Implementation**

Regarding the implementation, the programming language Python will be used, since it’s easy to write, very rich and it is compatible with all the tasks the Software subteam is called to work on. It is one of the main languages that are used for Data Analysis, the main language that Raspberry Pi uses and it can create very rich GUIs when used correctly. Other lower-level languages like C or Java might also be used in order to implement the Data Link protocols as seen in paragraph 4.8.2.3but that remains to be defined after the STW.

To aid the development, Mu, Spyder, Visual Studio Code (setted up for Python) and PyCharm IDEs will be used each for different stages of the development as they offer different tools for debugging or projecting information.

The Libraries and Frameworks that are definitely going to be used are Anaconda, Django ~~or Flask (to be defined when the research on GUI will be done)~~, pandas, matplotlib, scipy and numpy. Some libraries, like the adafruit\_blinka, gpiozero, RPi.GPIO, spidev and smbus2, will have to be studied before use in order to choose which one is better or more safe to use.

In terms of Operating Systems, Raspbian is going to be used for the Raspberry, so the knowledge of Linux kernels and Debian Linux is vital, and everything is going to be developed in Microsoft Windows.

## **Ground Support Equipment**

In the lists below, the items needed in the ground station during the Launch Campaign can be seen. It is specified which of the items will be brought from the team and which are expected to be found in the ESRANGE Space Centre.

### **Subteam of Science**

* Containers for the emulsion’s liquids and surfactant. (DROPSTAR Team)
* Container for storing the emulsion after the flight. (DROPSTAR Team)
* Protective equipment: Gloves, masks, glasses. (DROPSTAR Team)
* Mixer for the dilution of the surfactant in the water. (DROPSTAR Team)
* Syringes for the placement of the liquids in the cell. (DROPSTAR Team)
* Heating plate for the degassing of the liquids. (ESRANGE Space Centre)
* Acetone and ultra-pure water for the cleansing of the cell and the piston before the flight. (DROPSTAR Team)
* Cloths, paper towels. (ESRANGE Space Centre)

### **Subteam of Software**

* ~~One computer to receive data from the REXUS service module for the visualization of the status and some of the data of the experiment. (DROPSTAR Team)~~

~~Python will be used to handle the data and project it in a Webform GUI using Django or Flask frameworks. As mentioned in paragraph 4.8.3, the development stage will be aided by IDEs like Visual Studio Code (setted up for Python) and PyCharm that will be used each for different stages of the development as they offer different tools for debugging or projecting information. In case a lower-level language is needed for the handling of the data protocol, C or Java will be used accordingly.~~

The ground station equipment, is split in two categories:

a. Hardware (computers, cables, etc.)

b. Software (IDE, browser, etc.)

As we settled to use an Ethernet Port (instead of a RS – 232 Port) to receive the downlinked data from the RXSM Module, the Ground Station is required to have an Ethernet Port. Additionally, a computer will be used for receiving, decoding and evaluating the data. The computer will be brought by the team members. As mentioned, the reception will happen through Ethernet, so an Ethernet cable needs to be provided as well. Last but not least, a power strip is needed (preferably with 4-5 sockets) and to be supplied.

The Ground Station software will help us visualize the data received, this will be implemented with a browser-based GUI (developed with the Django Python framework). The data will be shown through graphs and charts to make monitoring easy for the team members. The received data will be decoded using an interface developed in Python.

The programming language that will be mainly used is Python and the IDEs of choice are PyCharm and Visual Studio Code (setted up for Python). If there is a need for a lower-level language, C/C++ will be used accordingly.

### **Subteam of Mechanics**

* Wrenches, screwdrivers, and pliers that will be used in the vacuum degassing of the fluids. (ESRANGE Space Centre)
* General tools and spare parts. (ESRANGE Space Centre)
* Lubricants. (DROPSTAR Team)
* Multimeter for voltage and functionality control of electronics. (ESRANGE Space Centre)
* Aluminum tape, a blade, and silicone are needed for the experiment's final thermal insulation seal. (ESRANGE Space Centre)
* Protective equipment: Gloves, masks, glasses. (DROPSTAR Team)
* Cloths, paper towels. (ESRANGE Space Centre)

### **Subteam of Electronics**

* Multi-meter (ESRANGE Space Centre)
* Replacement parts (DROPSTAR Team)
* Extra cables (DROPSTAR Team)