Design and Manufacture of a Fluidic Control Box for Soft Robotics Operation in Planetary Exploration

Toni Soler Arabí

ID 201781278

Supervisor: Ali Alazmani

Academic Year 2023-2024

SCHOOL OF MECHANICAL ENGINEERING

UNIVERSITY OF LEEDS

MECH3890 – Individual Engineering Project

PROJECT TITLE: Design and Manufacture of a Fluidic Control Box for Soft Robotics Operation in Planetary Exploration

PRESENTED BY	Toni Soler Arabi
SUPERVISED BY	Ali Alazmani

If the project is industrially linked, tick this box and provide details below ☐

COMPANY NAME AND ADDRESS:

STUDENT DECLARATION (from the "LU Declaration of Academic Integrity")

I am aware that the University defines plagiarism as presenting someone else's work, in whole or in part, as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance. I promise that in the attached submission I have not presented anyone else's work, in whole or in part, as my own and I have not colluded with others in the preparation of this work. Where I have taken advantage of the work of others, I have given full acknowledgement. I have not resubmitted my own work or part thereof without specific written permission to do so from the University staff concerned when any of this work has been or is being submitted for marks or credits even if in a different module or for a different qualification or completed prior to entry to the University. I have read and understood the University's published rules on plagiarism and also any more detailed rules specified at School or module level. I know that if I commit plagiarism I can be expelled from the University and that it is my responsibility to be aware of the University's regulations on plagiarism and their importance. I re-confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes. I confirm that I have declared all mitigating circumstances that may be relevant to the assessment of this piece of work and that I wish to have taken into account. I am aware of the University's policy on mitigation and the School's procedures for the submission of statements and evidence of mitigation. I am aware of the penalties imposed for the late submission of coursework.

Date 01/05/2024.

Signed 

Table of Content

Abstract

Unmanned space exploration has always been tied to traditional rigid robotics. However, the increasing tendency of soft robotics use introduces the possibility of integrating these robots in space missions, taking advantage of their flexibility and adaptability to unknown environments. This project describes the process of design and manufacture of a fluidic control box to be used for the operation of soft robots in planetary exploration. The main functionalities of the fluidic control box together with the process of creation of these operations will be analysed, highlighting its advantages and limitations, resulting in a final discussion of the feasibility of its use in these missions.

List of figures

List of tables

Nomenclature

PWM: Pulse Width Modulation.

V: Volts.

Hz: Herz.

DC: Direct current.

MOSFET: Metal-oxide-semiconductor field-effect transistor.

A: Amperes.

T: Period of the generated wave.

f: Frequency of the generated wave.

TCCRnA: Timer / Counter control register A of Timer n.

CSn2:0: Clock Select bits of Timer n.

WGMn3:0: Waveform generation mode bits of Timer n.

COMnx1:0: Compare output mode bits of Timer n.

T_{clk}: Period of the Arduino clock, time elapsed between clocks.

f_{clk}: Frequency of the Arduino clock.

TCNTn: Timer counter register for timer counter n, register that counts up or down.

ICRn: Input capture register of Timer n, number of clocks to reach the top.

OCRnx: Output compare register of pin n, number of clocks before the signal changes.

freq: frequency of the whole system for PWM in Arduino.

DutyC1: Duty cycle of valve 1 for PWM in Arduino.

DutyC2: Duty cycle of valve 2 for PWM in Arduino.

DutyC3: Duty cycle of valve 3 for PWM in Arduino.

DutyC4: Duty cycle of valve 4 for PWM in Arduino.

freq: frequency of the whole system for manual operation in Arduino.

DutyC1: Duty cycle of valve 1 for manual operation in Arduino.

DutyC2: Duty cycle of valve 2 for manual operation in Arduino.

DutyC3: Duty cycle of valve 3 for manual operation in Arduino.

DutyC4: Duty cycle of valve 4 for manual operation in Arduino.

Pset: Screen-set pressure.

P_{read}: Pressure read in binary.

P_{max}: Maximum pressure reading limit.

P_{min}: Atmospheric pressure.

t_i: Valve opening time stored in digit i of the sequence number.

t_s: Time set to check if the selected valve's opening time had elapsed.

Chapter 1. Introduction

1.1 Introduction

Exploration of celestial bodies such as planets and moons is one of the most interesting and fascinating scientific branches, as it helps to understand the cosmos and the history of life, possibly leading to the discovery of habitable environments, or even extraterrestrial life beyond the Earth. However, it comes with the disadvantage of the significant number of challenges derived from these missions. These challenges require innovative solutions to make possible the study of the different terrains, as well as the performance of demanding tasks in environments susceptible to changes.

Historically, traditional rigid robots have been used for these purposes, such is the case of the recent “Perseverance” rover [1] landing in Mars. While these robots have proved to be effective in controlled environments, they often struggle in unstable or unknown conditions due to their limited adaptability and low resilience to collisions [2].

In contrast, soft robotics is presented as a promising alternative for these missions, through the operation of robots constructed from malleable and deformable materials allowing safer interactions with both the environment and fragile objects. Soft robotics technology presents the advantages derived from flexibility and adaptability, together with greater resistance to wear and lower weight [3], making it particularly suitable for applications in planetary exploration.

Soft robotics presents an immense potential for revolutionizing planetary exploration through a wide range of applications in bio-inspired robots. Robotic crawlers, such as the Softworms, [4] presenting the capability of inching, rolling or even climbing steep inclines, are perfectly suitable for the operation in challenging terrains. Octopus-inspired manipulators such as the Octobot which has shown remarkable displays of dexterity and strength [5], offer versatile solutions for exploration tasks on planetary surfaces. Sample collection tasks would be covered by soft grippers [6] which enable gentle and precise handling of fragile objects. Even opening the frontiers of liquid environments exploration with the Soft Robotics Fish [7], soft robots are presented as a technology offering new possibilities for the study of extraterrestrial environments.

The effective management of these soft robots is completely dependent on the control of fluidic systems, which are responsible for their main operations. Being the fluid the medium of operation results in pressure control playing the most important role [8] in the performance of these robots.

These critical needs are intended to be met by the design and manufacture of a fluidic control box for the operation of soft robotics in planetary exploration missions. This system would serve as the main element in charge for controlling the movements and functions of soft robots. Through pressure control oriented to the operation of soft robotics, this project seeks to overcome the constraints of standard rigid robots and provide new opportunities for investigating and understanding celestial bodies beyond Earth.

1.2 Aim

The aim of the project is to design and manufacture a fluidic control box capable of managing the operation of soft robots in extraterrestrial planets through pressure modification.

1.3 Objectives

The different objectives that need to be completed, to ensure that the aim of the project is achieved, are listed here:

- Programming a microcontroller through Arduino to achieve the desired pressure control, offering the possibilities of manual or PWM (Pulse Width Modulation) operation.
- Configuration of a graphical screen which allows a user-friendly interface for the use of the device. The screen should display the necessary parameters for the operation of the control box.
- Storage of the relevant parameters for the operation of the control box in precisely controlled intervals of time.
- Design of a sequential pressure control for the movement of soft robots in planetary exploration.
- Performing several tests to ensure the correct functioning of the different components conforming the fluidic control box.
- Assembling of the components conforming the fluidic control box, considering organization and space available.

1.4 Report Layout

Firstly, in chapter 1 an introduction is given, where the motivation and significance of the project are described. Following with chapter 2, the design and manufacture process of the fluidic control box is explained. This includes an analysis of the different components and its organization inside the system. After that, chapter 3 describes the programming area of the fluidic control box. Chapter 4 explores the possibilities for the operation of soft robot in extraterrestrial planets using the fluidic control box. Finally, chapter 5 gives concluding comments on the project and describes future work that could be done.

Chapter 2: Design and manufacture

2.1 Introduction

As every complex system, the fluidic control box is formed by a diverse number of components. This chapter provides an analysis of these components, examining their individual performance, and the connections between them. Moreover, diagrams illustrating these connections, together with a representation of the system in real-life are displayed, providing a better understanding of the fluidic control box operation.

2.2 Methodology

Any system including electronic components requires a power supply. For the fluidic control box, an adapter was used to convert the alternating current given by the electrical network, typically operating at 230 V (voltage) and 50 Hz (frequency), into a direct current of 24 V. This 24 V DC supply was essential to power the MOSFET, which will be discussed in detail later. Figure 1 shows the adapter used in the project.

Figure 1: Plug-in AC/DC Adapter 24 V DC, 3 A output [9].

However, not all components within the system required the same electrical settings for operation. In fact, most of them were operated at 5 V DC. Therefore, an adjustable voltage regulator, as the one shown in Figure 2, was used to transform those 24 V DC into 5 V DC.

Figure 2: DFR0379 Adjustable DC to DC voltage regulator [10].

Given that the system consisted of various electronic components, temporary electrical connections between each component were established using an electronic breadboard, and male to male jumper wires (Figure 3), thus eliminating the need for soldering. The electronic breadboard facilitated connections by providing rails where components requiring identical electrical settings could be linked together using the jumper wires.

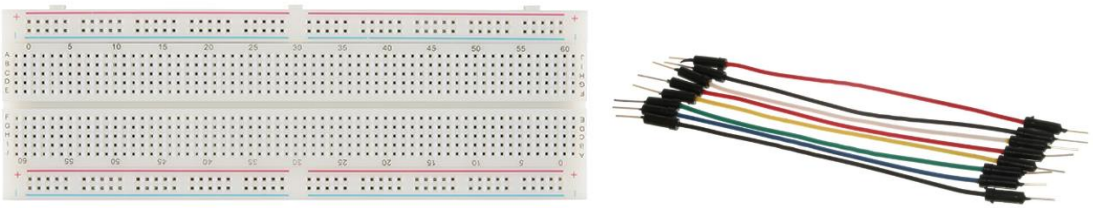

Figure 3: Electronic breadboard [11] and male to male jumper wires [12].

A touchscreen served as the primary interface for inputs in the system. Users can conveniently adjust different parameters of the fluidic control box operation by interacting with the screen. The screen operated as a slave to the microcontroller, meaning it could not autonomously update itself after an event like a button press. Instead, it communicated such events to the microcontroller which then gave the appropriate commands to update the screen accordingly.

This communication between screen and microcontroller was facilitated by the 4DSYSTEMS Arduino Adaptor, which was connected by simply aligning the pins identifier on the shield with those on the Arduino. The screen was powered via the white cable shown on Figure 4 left, which was connected to both the screen and the squared adaptor. Subsequently, the squared adaptor was linked to the Arduino adaptor via the cable shown in Figure 4 right. Therefore, as long as the screen remained connected to the Arduino and this last one was powered, the screen remained operational.

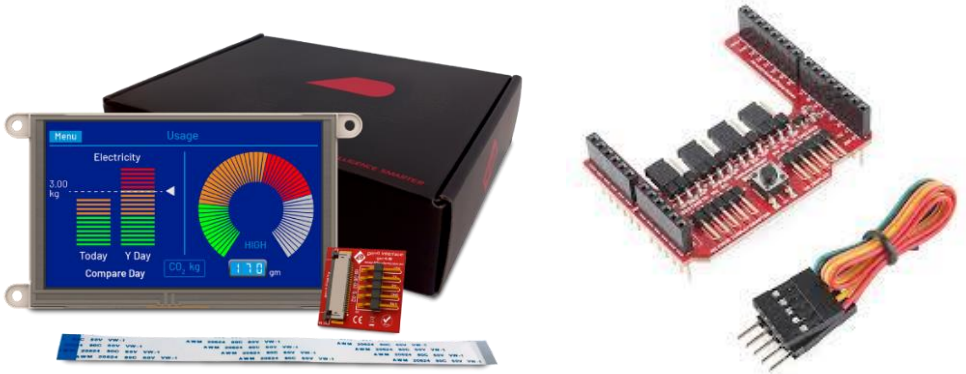

Figure 4: 4DSYSTEMS screen [13] (left) and Arduino adaptor [14] (right).

The screen displayed the main operational functions for the fluidic control box, as shown in Figure 5.

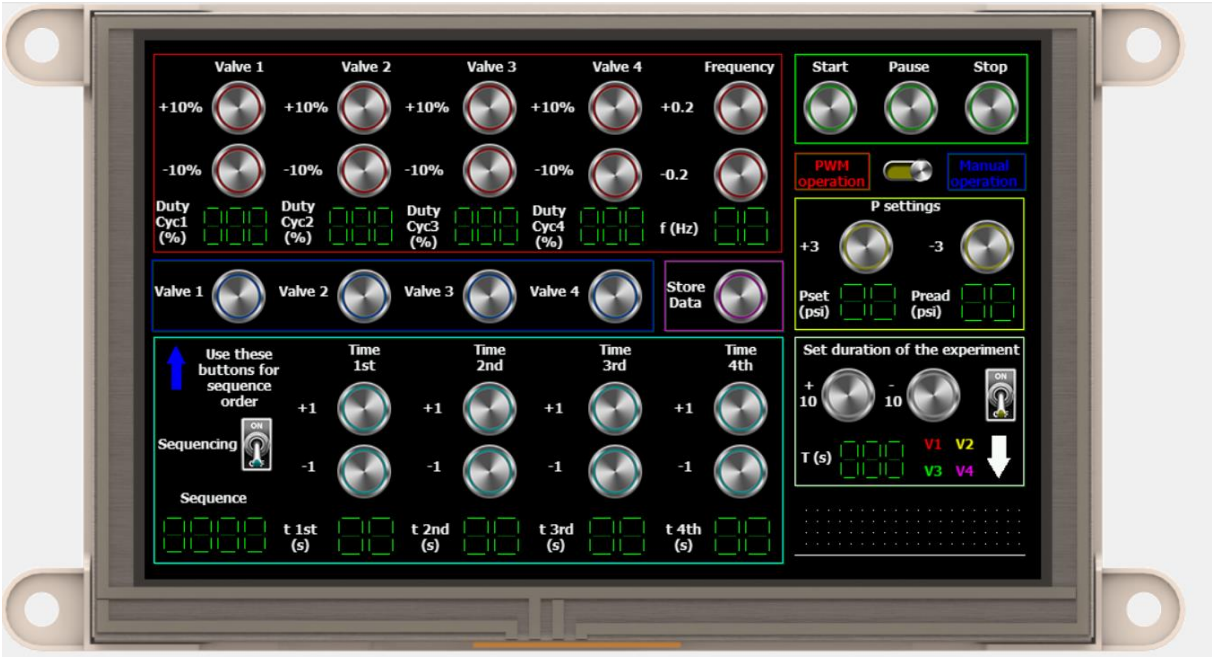

Figure 5: Main screen for the fluidic control box.

Starting with the upper section of the screen, options for manual or PWM (Pulse Width Modulation) operation were given. In manual mode, users can manually activate individual valves by pressing the corresponding button. PWM operation involved generating square-wave pulses with specified widths [15], which were repeated every period T. The width of these square waves was controlled by the duty cycle, corresponding to the percentage of time within the period T during which the signal was active. Users can modify both the duty cycle and the frequency f, directly affecting the period T, as $f = \frac{1}{T}$. Both parameters are shown in Figure 6.

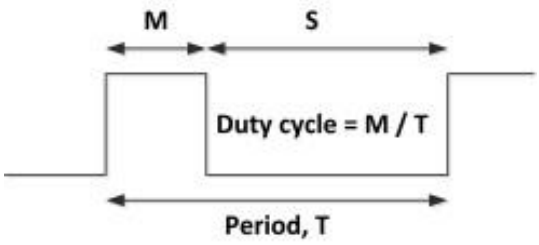

Figure 6: PWM main parameters [15].

Regarding the pressure settings, users can input the desired pressure to be set by the pressure regulator, which will be analysed later. Subsequently, the pressure is measured at the valve exits through a pressure transducer and displayed in the screen to indicate if it matches the user-defined settings.

The screen also offered the option for sequential valve opening, allowing users to input a 4-digit sequence number, and subsequently set the duration for each step in the sequence, corresponding to the opening time for each selected valve.

Additionally, a control panel including start, pause, and stop buttons was provided. The stop button reset the control box parameters to default values (0) and ended the experiment. Users can also set the experiment duration, displayed in the bottom right corner of the screen, which pauses the system once the specified time has elapsed.

Furthermore, a small display positioned below the experiment duration settings allows users to monitor the signals sent to each valve, facilitating verification of the system's performance.

Finally, a data storage option was made available. When selected, it functioned as follows: it created a new file in the Adafruit Data Logging Shield's SD (Figure 7) and recorded the values for the relevant parameters in the fluidic control box at regular intervals.

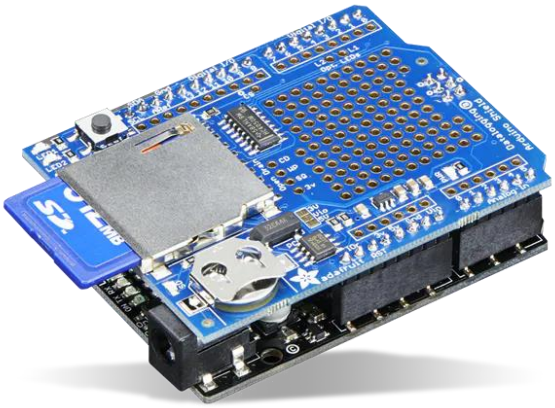

Figure 7: Adafruit assembled data logging shield for Arduino [16].

The key component of the fluidic control box is the microcontroller, in this case the Arduino Mega 2560 Rev 3 shown in Figure 8. Operating as a small computer, users program it to control the functions along the system [17], managing every signal sent or received between itself and the other components. The Arduino received inputs from the screen and executed the corresponding actions.

Pins 5 to 8, enabling the creation of PWM signals, were used for the communication between the microcontroller and the valves. Users adjusted signal parameters and Arduino generated those updated signals. Pin 11 controls the pressure regulator, while pins A0 to A3 served as analog inputs, monitoring signals transmitted to each valve by the Arduino. Pins 20 and 21 facilitated communications between the pressure transducer and the microcontroller.

In terms to the electrical settings, the Arduino operated on a 5 V DC input voltage, while producing an output voltage of the same characteristics.

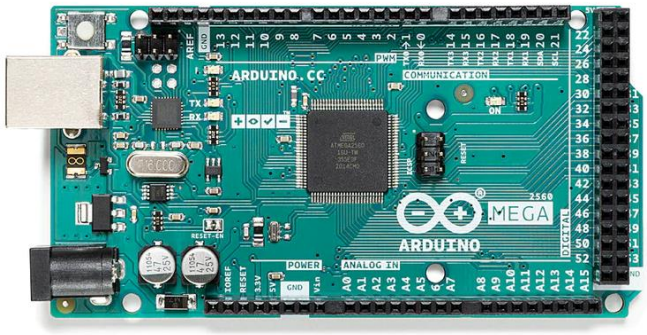

Figure 8: Arduino MEGA 2560 Rev 3 [18].

The fluidic control box also included a MOSFET, one of the most used transistors currently. MOSFETs are devices employed for signal amplification, control, or generation [19]. In the context of the fluidic control box, the MOSFET served to amplify signals received from pins 5 to 8 from the Arduino, which operated at a voltage of 5 V DC. Since the goal was to regulate the opening and closing of the valves, requiring a 24 V DC input, the MOSFET acted as an amplifier of the signals generated by the Arduino. Figure 9 shows the MOSFET used.

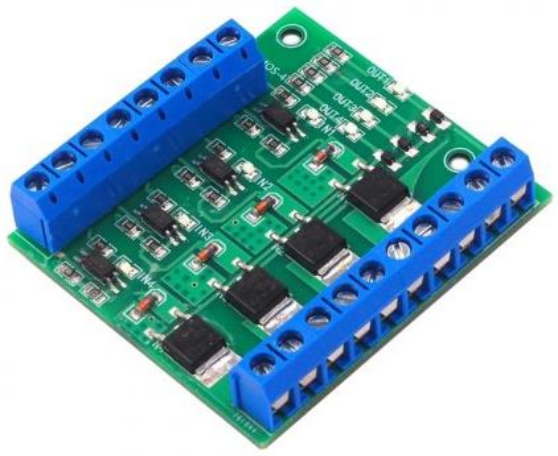

Figure 9: 4-Channel MOSFET [20].

In addition to the electronic components, the fluidic control box contained 4 pneumatic components: solenoid valves, pneumatic lines, a pressure regulator, and a pressure transducer.

The solenoid valves, mounted in a manifold, controlled the fluid flow direction by opening and closing accordingly. Operating at an electrical input of 24 V DC, the solenoid valves, illustrated in Figure 10, needed the use of a MOSFET between the microcontroller and the valves.

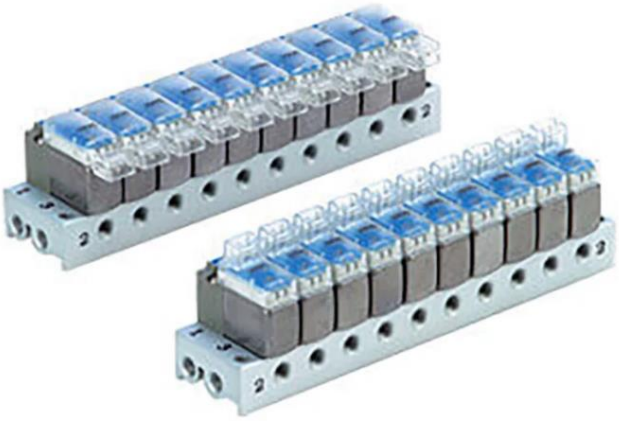

Figure 10: Solenoid valves mounted in a manifold [21].

The pressure regulator, displayed in Figure 11, adjusted the system pressure accordingly to the input voltage. Operating within a range of 0 to 5 V DC, the pressure regulator responded to signals sent from the microcontroller through Pin 11. These signals varied between the 0 to 5 V DC range based on the pressure settings inputted in the screen. These settings included a range of [0,15] psi. The compressed air was subsequently transported throughout the system using flexible tubes known as pneumatic lines [22].

Figure 11: ITV0010-2BS Pressure regulators [23] and pneumatic lines [22].

The final pneumatic component is the pressure transducer. Communication between the pressure transducer and the Arduino microcontroller was established via the SDA, SCL pins corresponding to pins 20 and 21 in the Arduino Mega 2560 Rev 3. The Arduino initiated the process by requesting bytes representing the pressure readings from the pressure transducer. Subsequently, it sent commands to display these readings in the screen. The pressure transducer used for the fluidic control box is illustrated in Figure 12.

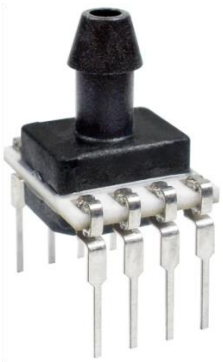

Figure 12: SSCDANN015PG2A3 Pressure transducer [24].

2.3 Results

After explaining the operational requirements of each component, an electronic scheme, illustrating the connections between all components is presented in Figure 13.

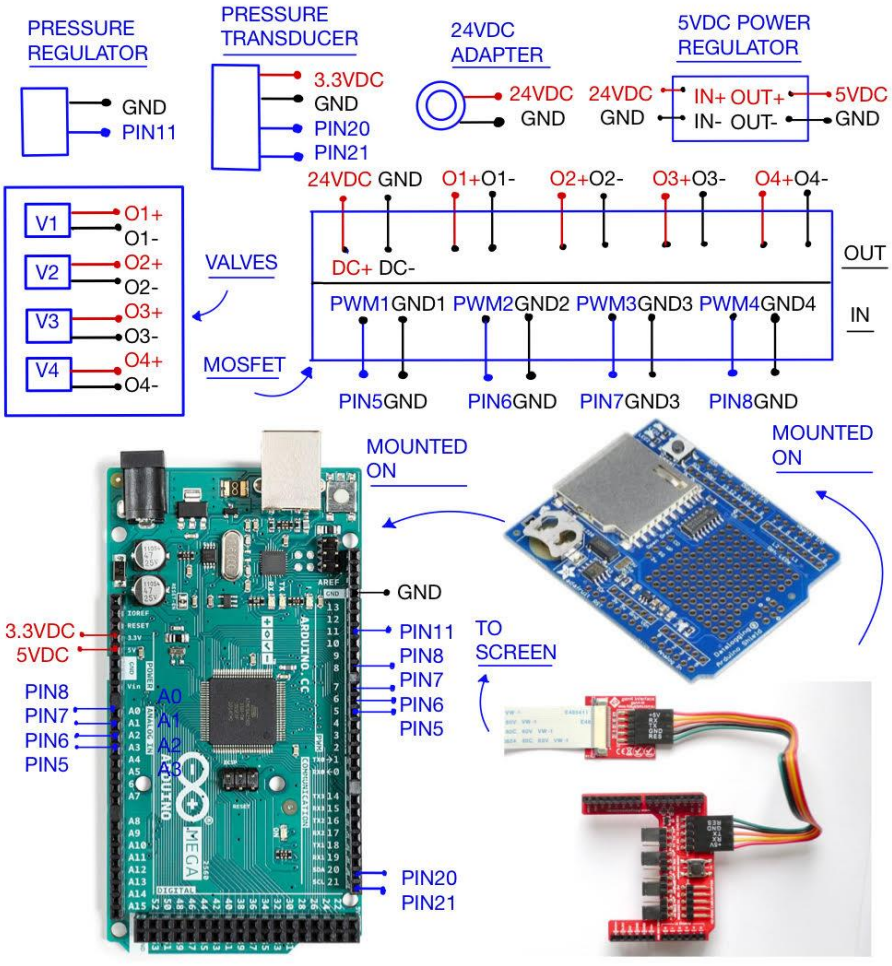

Figure 13: Electronic scheme for the fluidic control box.

A visual representation of the system is shown next in Figure 14. It is worth noting that the pressure regulator was not included in the picture since the lab presented controlled pressure supply facilities, however, the general design of the control box incorporates it.

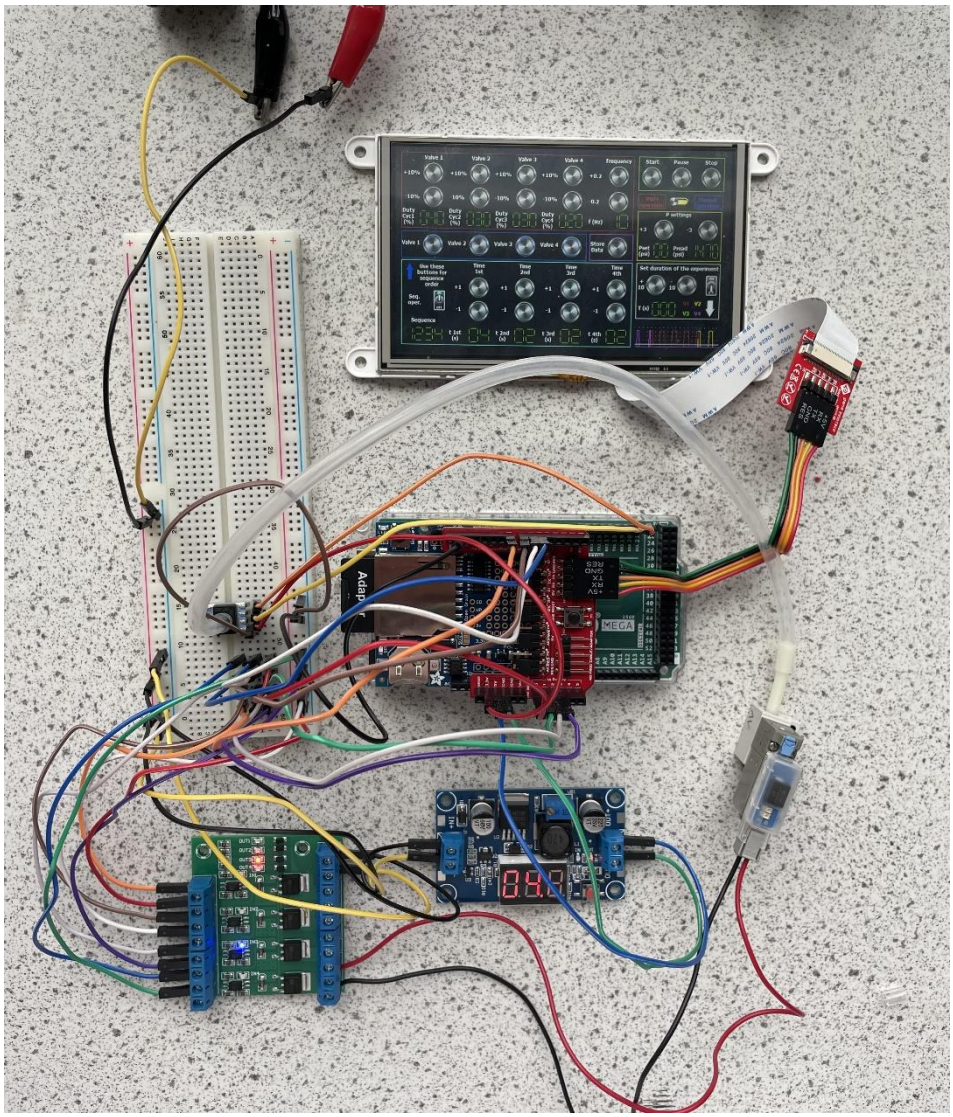

Figure 14: Fluidic control box system.

2.4 Discussion

The individual performance of each component has been analysed, providing a better understanding of their operation and contribution to the complete system of the fluidic control box.

Moreover, the electronic scheme provided an overview of the component connections within the system, approaching the design and layout of the fluidic control box.

Additionally, the physical implementation of these components was illustrated in the system display. Through this representation, the interaction of the components in real-world conditions was shown.

Together, these results provided an overview of the structural and operational aspects of the fluidic control box. Having understood the system's design and functionality the coding and software aspects of the fluidic control box are now to be analysed.

Chapter 3: Arduino programming

3.1 Introduction

In this chapter, the technical aspects of the Arduino programming, which is responsible for the fluidic control box operation, are explored. From managing fluid flow through PWM/Manual operation and communicating with the screen interface, to monitoring pressure sensors and sequencing controlled valve actuations, every aspect of the control box's functionalities is controlled by the Arduino code included in the microcontroller. The functions and algorithms enabling these operations are analysed, providing insights into how the Arduino programming facilitates pressure control and data acquisition.

3.2 Methodology

The touchable screen served as the main source of inputs for the system. Utilizing the 4D Workshop 4 IDE software [25] provided by the screen supplier, the screen's working environment was created through drag-and-drop operation, incorporating buttons, LED digits, oscilloscopes, etc.

To control these items, programming was performed in Arduino, with 4DSYSTEMS offering a dedicated library [26] for screen control. Communication between the screen and Arduino was facilitated through a function named `myGenieEventHandler()`, which was included into the Arduino loop to handle screen events effectively.

Identification of screen events was achieved using the `genie.EventIs(&Event, GENIE_REPORT_EVENT, object, index)` function, which detected events based on specified criteria such as object type and index number. Each item on the screen was assigned an object identification (e.g., button, scope, LED) and an index number ranging from 0 to j, where j represented the number of items of the same type.

While buttons served as inputs, other screen elements like scopes and LEDs required dynamic updates. This was achieved using the `genie.WriteObject( object, index, data)`, function from the library, allowing for real-time display updates based on the input data.

Having explained the communication commands between the screen and the Arduino, the focus now shifts to explaining the programming of the fluidic control box's main functionalities. The analysis starts with the manual and PWM operation of the valves, focusing on the process of signal generation for these operations.

The Arduino Mega 2560 Rev 3 included the possibility of PWM signal generation via pins 2 to 13, utilizing timers 0 to 5. These timers were used to measure time increments and executing tasks at specific time intervals.

Pins 5 to 8, associated to timers 3 and 4, were the ones chosen for PWM signal generation. The behaviour and operations of these timers was governed by their respective control registers denoted as TCCRnA, TCCRnB and TCCRnC, where “n” represents the identification for the timer.

Bit	7	6	5	4	3	2	1	0	
(0xA0)	COM4A1	COM4A0	COM4B1	COM4B0	COM4C1	COM4C0	WGM41	WGM40	TCCR4A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0xA1)	ICNC4	ICES4	–	WGM43	WGM42	CS42	CS41	CS40	TCCR4B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0xA2)	FOC4A	FOC4B	FOC4C	–	–	–	–	–	TCCR4C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Figure 15: TCCR4A, TCCR4B and TCCR4C control registers [27].

The timer’s operational behaviour was defined by the bits shown in Figure 15. For PWM signal generation, the clock select (CSn2:0), waveform generation mode (WGMn3:0), and compare output mode (COMnx1:0) bits were involved. Here “x” represented either A or B.

The clock select bits determined the prescaler value of the clock, affecting the time elapsed between clock pulses, $T_{clk} = \frac{1}{f_{clk}}$. When generating signals, if T_{clk} is too small, the system may exceed its maximum count limit. To solve this, f_{clk} was divided by a prescaler value, decreasing it, therefore increasing T_{clk}. For optimal operation of the control box, a prescaler value of 256 was chosen. The specific combination of Clock Select bits (CSn2:0) required to configure this prescaler is displayed in Table 1.

Table 1: Clock select bit configuration for 256 prescaler value [27].

CSn2	CSn1	CSn0	Description
1	0	0	f_{clk}/256 (From prescaler)

The waveform generation mode bits affected the counting sequence of the timer, crucial for signal generation. Among the available options for waveform generation, mode 8, PWM, phase and frequency correct, was selected due to its capability to modify both signal frequency and duty cycles, which was a requirement for the control box's functionality. Table 2 shows the required bit configuration to set mode 8.

Table 2: Bit configuration for waveform generation mode 8 [27].

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMnO (PWMnO)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM

In the phase and frequency correct mode, illustrated in Figure 16, the register TCNTn acted as a counter, incrementing with each clock pulse until reaching the maximum value (TOP), then counting backwards to 0.

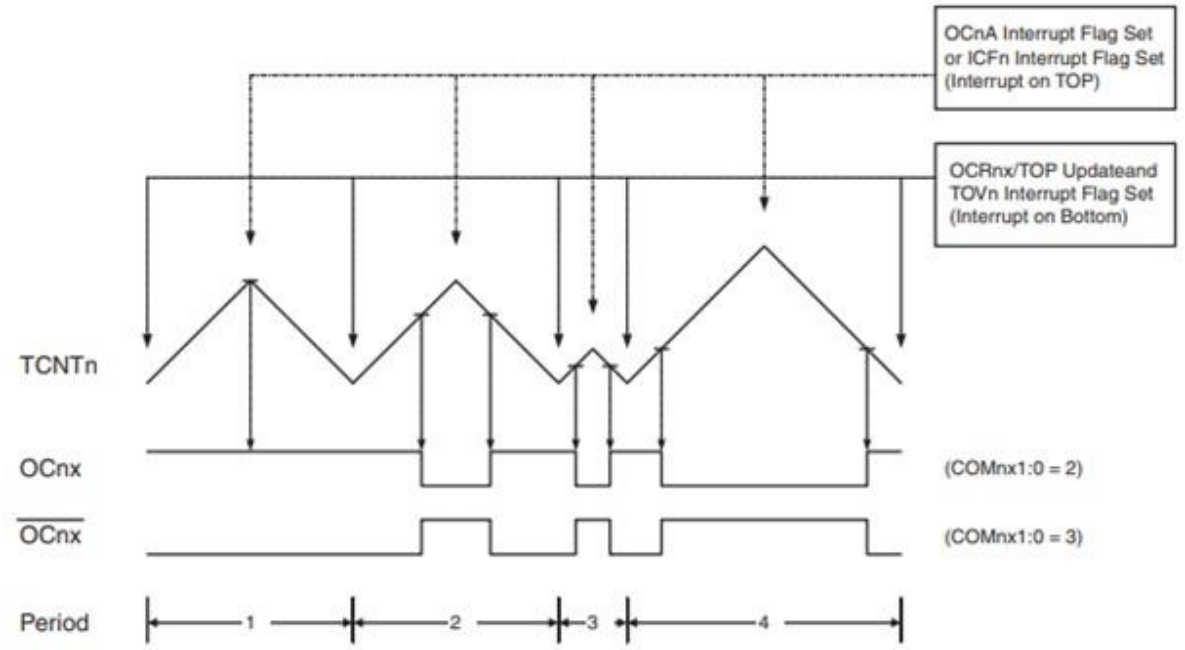

Figure 16: Phase and frequency correct PWM Mode, timing diagram [27].

In this mode, the maximum value (TOP) was determined by the ICRn, as shown in Table 2. By manipulating this register, the period T, and thus the frequency f of the wave were controlled. The period T was calculated using (3.1):

$$T = T_{clk} \cdot 2 \cdot \mathrm{ICRn} \cdot prescaler \tag{3.1}$$

Where T_{clk} is the time between clock pulses. To set the ICRn register for a desired frequency f, it was determined using (3.2):

$$f = \frac{1}{T} = \frac{1}{T_{clk} \cdot 2 \cdot ICRn \cdot prescaler} = \frac{f_{clk}}{2 \cdot ICRn \cdot prescaler} \rightarrow ICRn = \frac{f_{clk}}{2 \cdot f \cdot prescaler} \quad (3.2)$$

Another relevant parameter to be controlled was the duty cycle of the wave. The approach to its control depends on the user's choice of the compare output mode bits. Out of the available options, the non-inverting output mode was chosen as it facilitated duty cycle control. The bit configuration for the non-inverting output mode is illustrated in Table 3.

Table 3: Bit configuration for non-inverting compare output mode [27].

COMnA1 COMnB1 COMnC1	COMnAO COMnBO COMnCO	Description
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting Set OCnA/OCnB/OCnC on compare match when down counting

The OCRnx register controlled the duty cycle. In the non-inverting compare output mode, the signal remained on until TCNTn reached the OCRnx value during upcounting, then turned off. When TCNTn reached the TOP and started downcounting, the signal turned on again when reaching the OCRnx value. This behaviour is depicted in Figure 16 (OCnx).

Duty cycle represents the percentage of each period where the signal is on. It can be calculated using ICRn (the number of clocks to reach the top) and OCRnx (the number of clocks before the signal changes). To set the desired duty cycle using ICRn, OCRnx was configured using (3.3):

$$DutyCycle(\%) = \frac{\text{OCRnx}}{ICRn} \cdot 100 \rightarrow \text{OCRnx} = \frac{ICRn \cdot DutyCycle(\%)}{100} \quad (3.3)$$

An Arduino function, `PWMfunc(freq, DutyC1, DutyC2, DutyC3, DutyC4)`, was created to generate PWM signals as explained. It adjusted ICRn and OCRnx based on the desired system frequency and valve duty cycles.

A switch was incorporated in the screen (Figure 17) to toggle between PWM and manual operation modes. The operation of both modes is detailed in Figure 18's flowchart, with PWMfunc() facilitating the two of them. For manual operation, a 100% duty cycle fully opened the valve, while 0% closed it. Separate variables (freqm and DutyCxm, with "m" denoting manual) were employed for this operation. This approach facilitated switching between modes while retaining settings for both, allowing parameter modifications even when the other mode was selected.

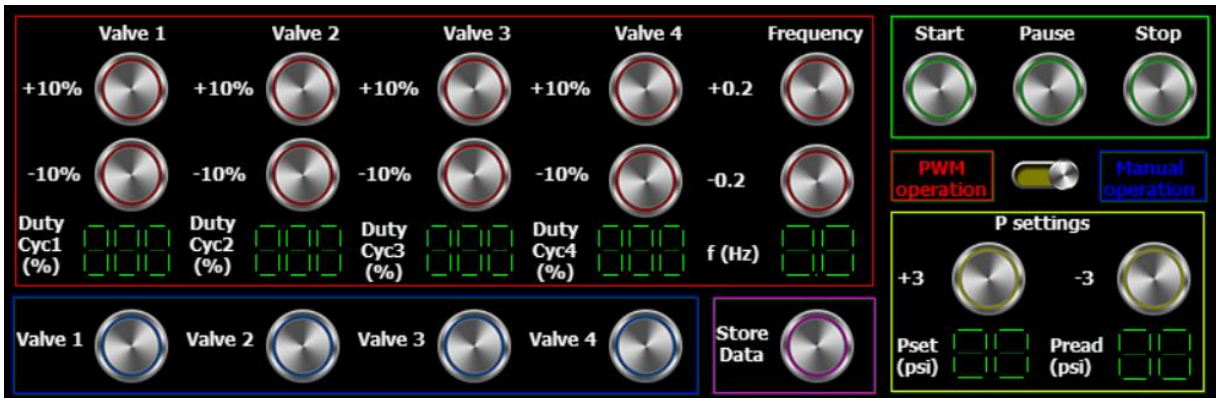

Figure 17: PWM/Manual operation, pressure settings and data storage screen interface.

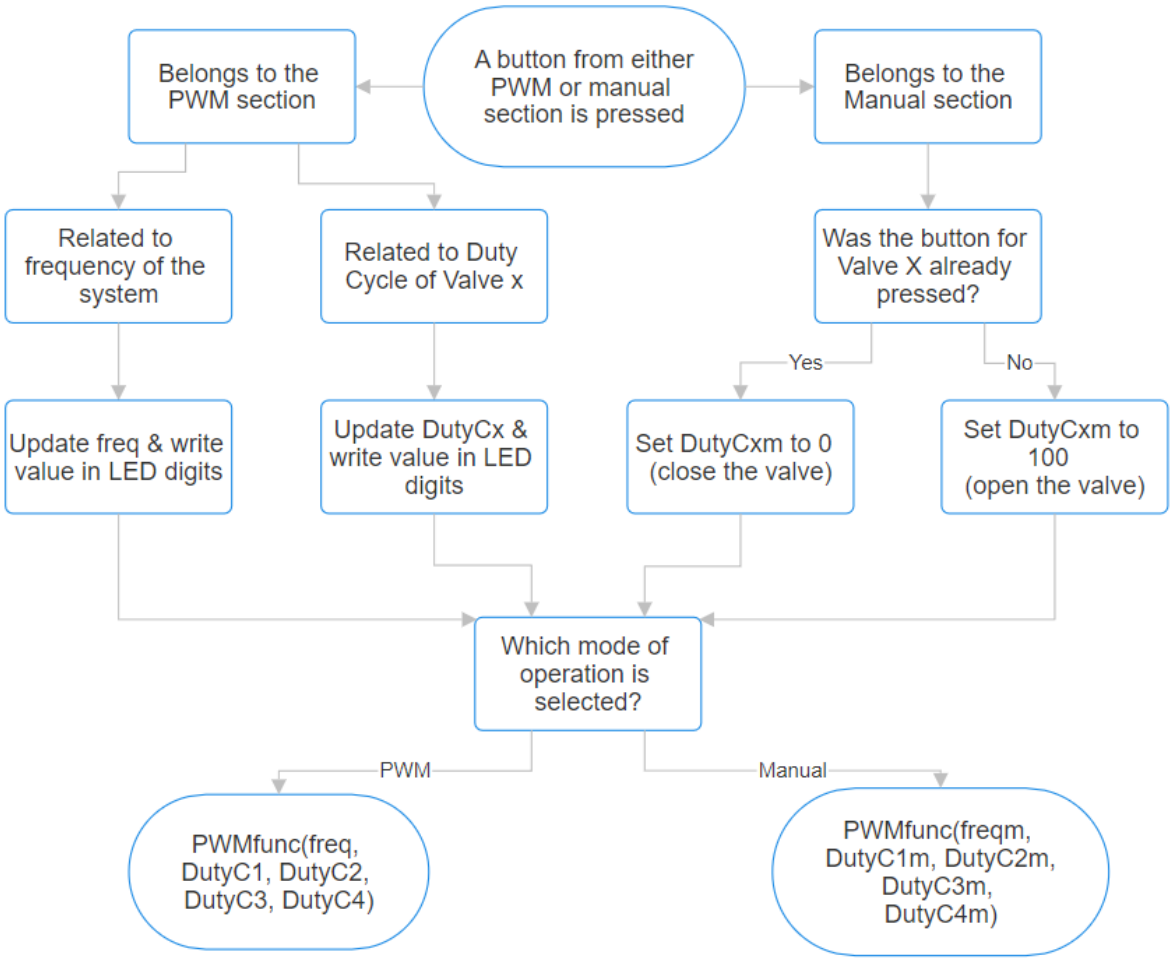

Figure 18: PWM/Manual operation flow chart.

Pressure control in the fluidic control box was managed via screen buttons (Figure 17), translating settings to the pressure regulator using `analogWrite(11, Pset)` [28], being Pset the screen-set pressure and 11 the output pin.

Pressure readings were facilitated through a pressure transducer using the I2C communication protocol, assisted by Arduino's Wire Library [29]. The communication line was identified using the Honeywell Pressure Sensors data sheet [30] and data was requested using

`Wire.requestFrom(0x28, 2)` [31]. Pressure value was obtained with `Wire.read()` [32], then scaled using Honeywell Pressure Sensors transfer function (3.4), and finally displayed on the screen.

$$P_{applied} = \left(\frac{P_{read}}{2^{14}} - 0.1\right) \cdot \frac{(P_{max} - P_{min})}{0.8} + P_{min} = \frac{\left(\frac{P_{read}}{2^{14}} - 0.1\right) \cdot (15 - 14.696)}{0.8} + 14.696 \quad (3.4)$$

Here P_{min} represents the atmospheric pressure, and P_{max} denotes the maximum pressure reading limit.

The fluidic control box also included a feature to store system parameters at specific time intervals. This process, illustrated in Figure 19's flowchart, was made possible through Arduino's SD library [33], and was controlled by the data storage button shown in Figure 17.

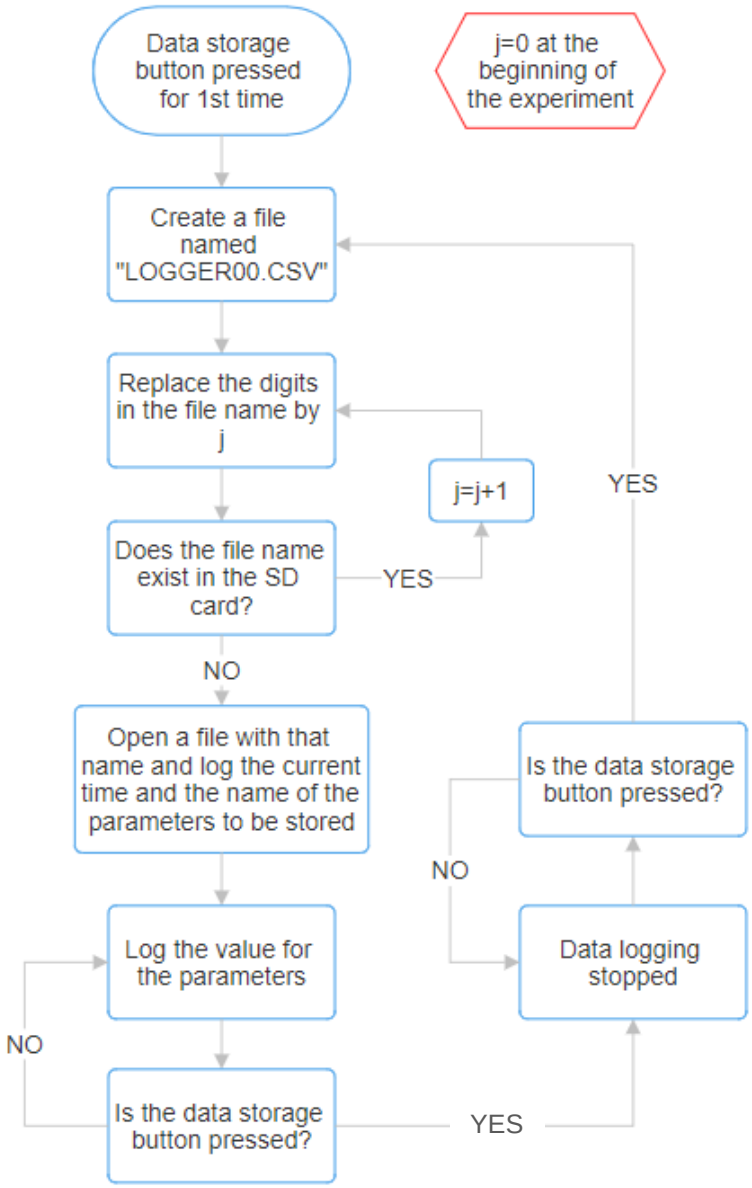

Figure 19: Data storage operation flow chart.

Another requirement for the fluidic control box was sequential valve opening, particularly useful for soft robot operation. A screen switch (Figure 20) activated this functionality, subsequently using the same buttons as manual operation to set the sequence This sequence was a 4-digit number, each digit representing a valve to open in order from left to right. Figure 21 illustrates the creation of this sequence number in a flow chart.

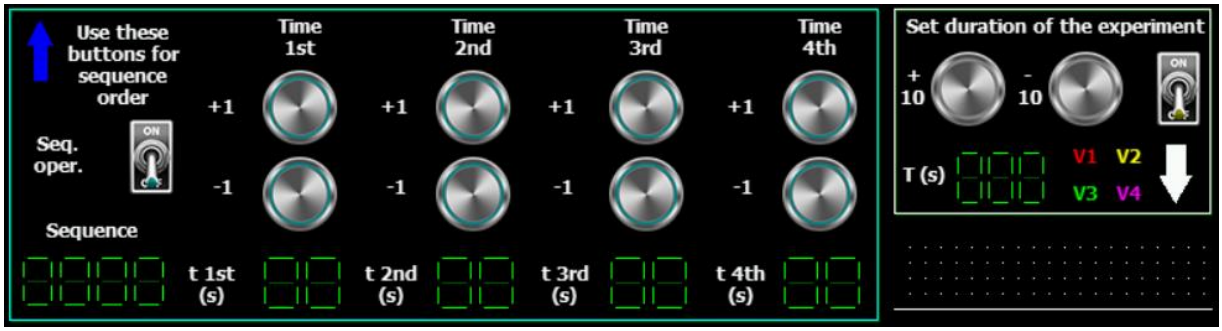

Figure 20: Sequential operation, duration of the experiment and scope screen interface.

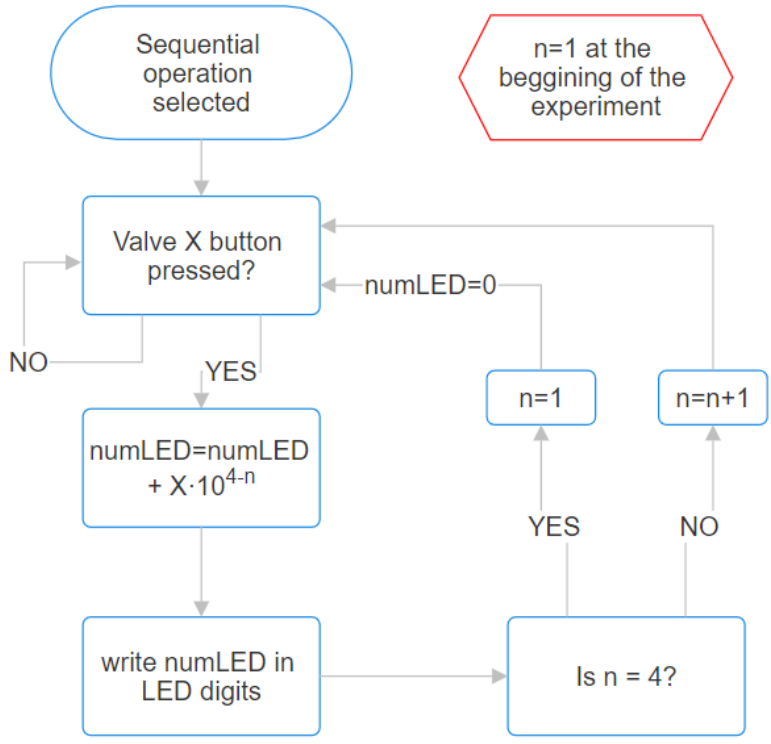

Figure 21: Flow chart for the creation of the sequence number.

It is important to emphasize that the sequence number was not restricted to incorporate all 4 valves and could include repeated valves. This number was displayed in the screen and served as a tool for programming sequential operation, as detailed in the following flow chart (Figure 22).

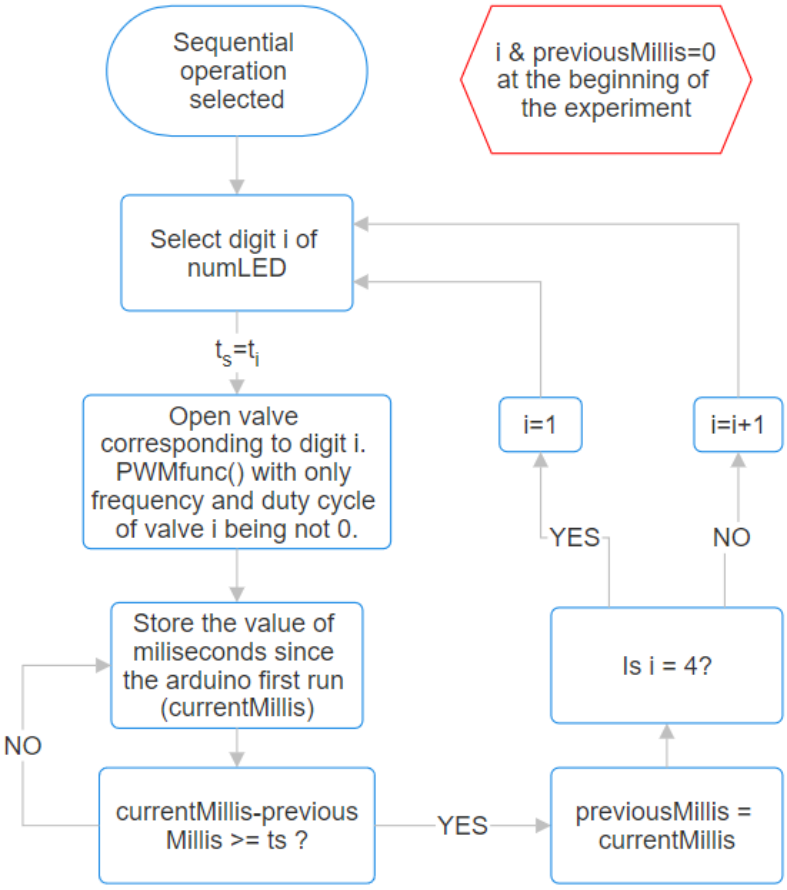

Figure 22: Sequential operation flow chart.

In the flow chart, t_i represents the valve opening time stored in digit i of the sequence number, set by the user using the screen (Figure 20). Each time a new digit was accessed, this time was reset as t_s. The loop in the lower-left corner of Figure 22 continuously checked if the selected valve's opening time had elapsed. It accomplished this by comparing the elapsed time since the experiment began to the time when the valve initially opened, ensuring that the difference exceeded t_s.

Additionally, the fluidic control box allows users to set the experiment duration using the screen buttons shown in Figure 20. Once set, the programming algorithm performed the exact same time comparison described above. However, in this case when the comparison condition was met, the system was paused.

Finally, a scope feature was incorporated into the screen, as displayed in Figure 20, enabling users to verify the characteristics of the generated signals. For this purpose, the output of Pins 5 to 8 (used for PWM) was connected to analog inputs A0-A2. The signal was processed using the `analogRead(Ax)` [34] command and scaled to match the scope's dimensions through `map(sensorValue,a,b,c,d)` [35] where a variable ranging between [a,b] was scaled to [c,d]. The full code that has been analysed here is included in the Appendix.

3.3 Results

After explaining the coding behind the main functionalities of the fluidic control box, the corresponding results are presented.

In Figure 23, the screen scope illustrates a possible manual operation, where the red, yellow, green, and pink signals corresponded to valves 1,2,3 and 4, respectively. The same valves are shown in Figure 24 together with the parameters used for the analogous PWM operation.

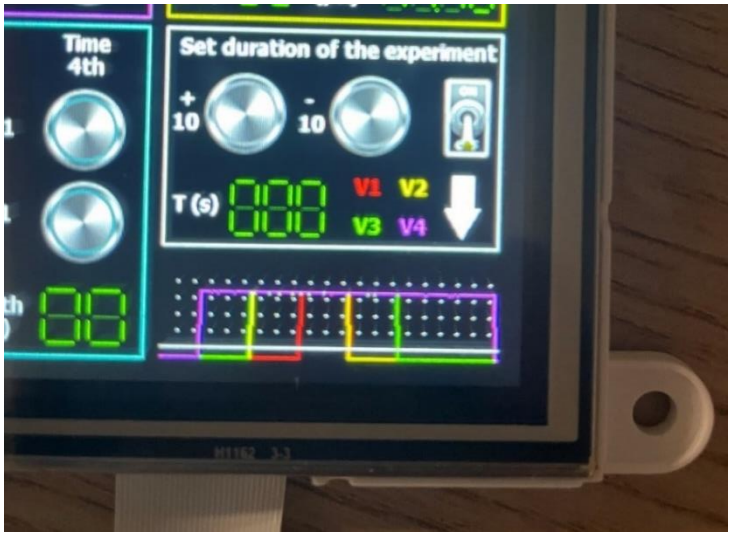

Figure 23: Manual operation output.

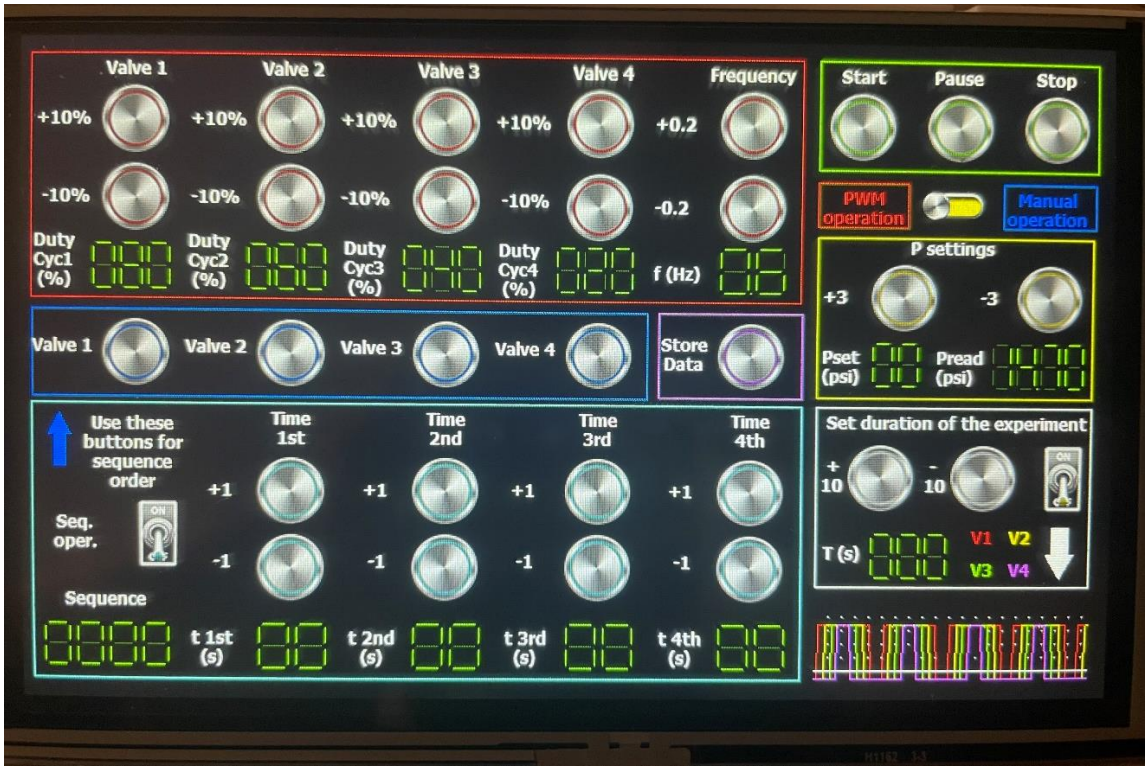

Figure 24: PWM operation parameters and output.

Subsequently, Table 4 displays the outcome of the data logging operation. It can be observed that every relevant parameter was logged precisely every 1 second. It's also worth mentioning that this data storage rate could be adjusted within the code. Additionally, the table provides a real-time update of each parameter as time progresses.

Table 4: Data storage operation output.

millis (ms)	P read (psi)	P set (psi)	PWM / Man.	DC 1 (%)	DC 2 (%)	DC 3 (%)	DC 4 (%)	f (Hz)	Seq.	Seq. num.	t1 (s)	T2 (s)	T3 (s)	T4 (s)	Dur.	T exp (s)
83171	14.90	15	PWM	60	80	40	50	100	On	2341	9	6	5	4	Off	-
84171	14.90	15	PWM	60	80	40	50	100	On	2341	9	6	5	4	Off	-
85171	14.90	15	PWM	60	80	40	50	100	On	2341	9	6	5	4	On	30
86171	14.90	15	PWM	50	80	40	50	100	On	2341	9	6	5	4	On	30
87171	14.90	15	PWM	30	80	40	50	100	On	2341	9	6	5	4	On	30
88171	14.90	15	PWM	30	70	40	50	100	On	2341	9	6	5	4	On	30

3.4 Discussion

The results demonstrated how effectively the fluidic control box performs in various operating modes. Through user adjusted non-repetitive valve opening, the manual operation mode was presented as an adaptable tool, essential for applications such as soft robotics.

In the PWM operation mode, the precise control over valve parameters such as frequency and duty cycle was presented as a characteristic making this mode of operation suitable for fluidic control systems where highly controlled performance is required.

The scope included on the screen enabled users to verify the characteristics of the generated signals, checking that they align with the desired parameters, providing validation and verification.

When it comes to pressure control, analysing Table 4, a slight difference between the pressure set and the one read could be observed, showing limitations in high precision. These challenges aree due to fluids having particular characteristics like compressibility and viscosity.

Nevertheless, the ability to coarsely control pressure through settings and readings was presented as one of the control box’s versatile characteristics. This versatility was further improved through the data logging operation, which provided data collection for analysis and offered the possibility to adjust the data storage rate.

Finally, it’s worth mentioning that the results for the sequential operation mode are detailed in the following chapter, where a movement sequence for a soft robot is replicated, providing deeper understanding of the fluidic control box’s capabilities.

Chapter 4: Soft robotics control in planetary exploration.

4.1 Introduction

An insight into the possible applications of the fluidic control box for soft robotics operation in planetary exploration is provided in this chapter. Based on the performance of an existing soft robot, it is explored how the control box allows the execution of the necessary movements for the investigation of extraterrestrial environments.

4.2 Methodology

A crawler robot with 5 different pneumatic channels [36] served as the soft robot to validate the fluidic control box's functionality. While the fluidic control box included a total of 4 pneumatic channels as outputs, expanding its capacity with an extra channel would simply implicate increasing in one unit the pneumatic components. Therefore, the explanation is provided considering an additional channel. The pneumatic channels were assigned numerical identifiers for clarification: being 1 and 2 the left and right front legs respectively, 3 and 4 the left and right hind legs, and 5 the core of the robot, corresponding to the auxiliary valve.

The main functionality of the robot was to navigate planetary terrain. To achieve this, a sequence of valve openings resulting in a crawling movement is shown in Figure 25. In the figures, pressurized pneumatic channels are represented while non-pressurized are shown in red. Replicating this movement sequence would simply consist of maintaining the 5th valve (central) open and creating a sequential number 4-1-3-2 with a time opening of approximately 0.6 seconds per valve.

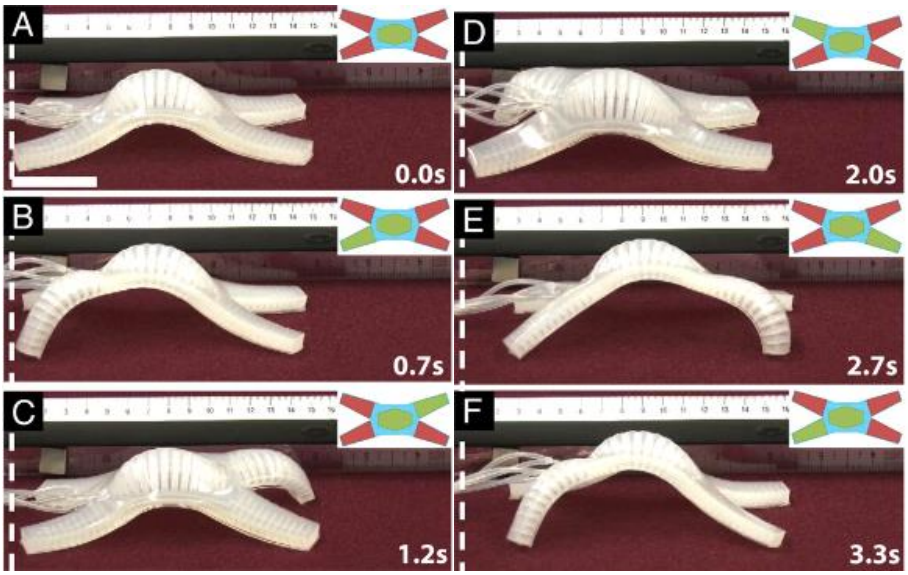

Figure 25: Crawling movement sequence [36].

Additionally, manual operation of the fluidic control box offered the possibility to overcome different obstacles being controlled by a robot operator. Figure 26 illustrates the movement of the robot underneath an obstacle, proving its superior adaptability to challenges compared to a traditional robot. For the considered operation, the operator would manually open the necessary valves at the specified times. For instance, at time point C valves 1, 2 and 5 would be opened, while at time point E valves 3 and 4 would be activated.

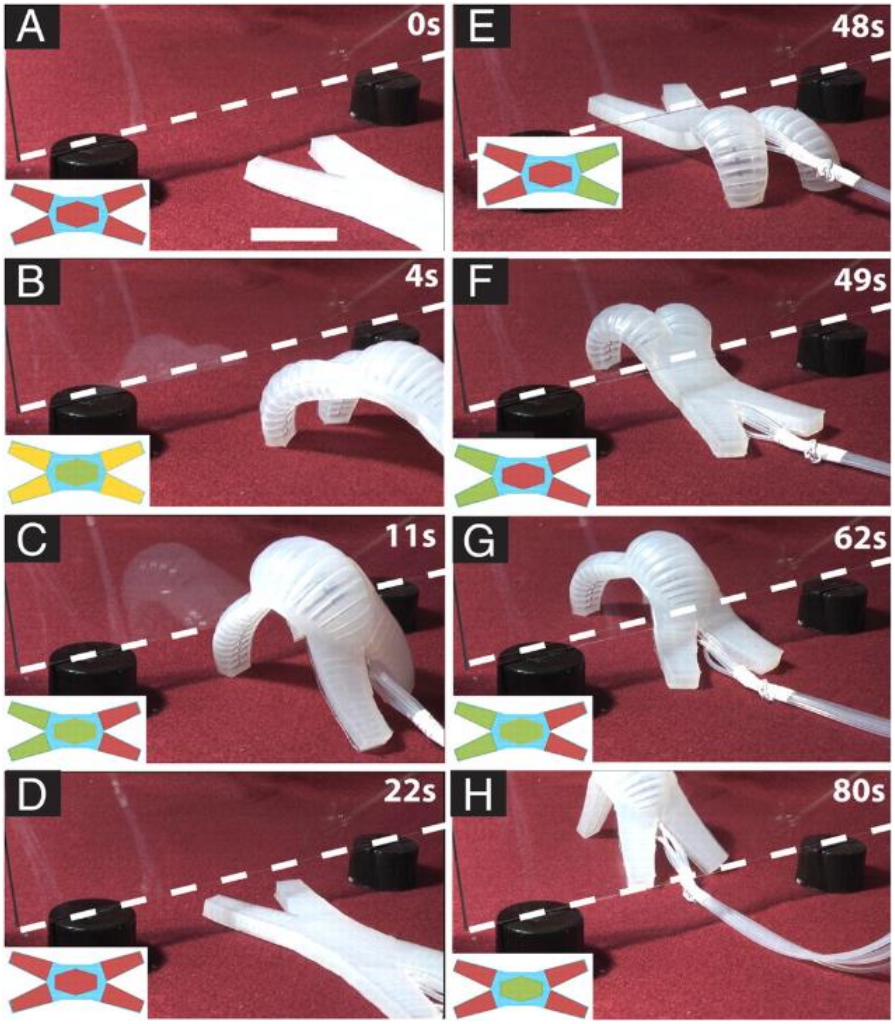

Figure 26: Obstacle avoidance movement sequence [36].

4.3 Results

The following Figure 27 will illustrate how the required signals to replicate the movement sequence shown in Figure 25 were generated by the fluidic control box. It is worth mentioning that for the fluidic control box, the time opening between valves was modified using 1 second increments for usability, and therefore the smaller operation value is 1 second, instead of the 0.6 second intervals used in Figure 25. However, this time increment could simply be adapted to include smaller values.

Figure 27: Sequential operation parameters and output.

4.4 Discussion

The fluidic control box is presented as a promising alternative for the control of soft robots in planetary exploration, demonstrating versatility and adaptability through the combination of both sequential and manual operation.

The successful sequence movement reproduction based on a previously existing soft robot operation showed the ability of the control box to generate motion requiring repetitive patterns, a critical aspect in planetary exploration.

Moreover, the manual operation has shown to provide the needed adaptability to overcome obstacles during exploration. The operators would control the robots in real-time through valve opening, guiding them through difficult environments, taking advantage of the flexibility offered by soft robots.

Furthermore, the fluidic control box proved great compatibility with existing soft robots, showing that the combination of both systems would simply require connecting the output pneumatic lines to the robot.

Chapter 5: Conclusion

5.1 Achievements

Throughout the project, different achievements related to the main objectives have been accomplished. Firstly, pressure control was obtained through the programming of the Arduino Mega 2560 Rev 3 microcontroller in Arduino, also offering the alternative for both manual and PWM operation.

Moreover, the configuration of the 4DSYSTEMS graphical screen has represented a key component to ensure user's accessibility and usability through the display of the different modes of operation and the creation of a user-friendly interface.

Additionally, the controlled storage of the relevant parameters at specified time intervals has been achieved through the data logger, offering the possibilities of data manipulation post operation.

Finally, a sequential pressure control system together with a possible controlled operation of a soft robot in space exploration have been provided, showing a promising advance of fluidic control technologies.

5.2 Discussion

The findings from chapters 2, 3, and 4 were closely related to the project's aim of designing and manufacturing a fluidic control box to manage soft robots' operation on planetary exploration through pressure modification.

In chapter 2 the analysis of the fluidic control box's design and manufacture provided an overview of its structural and operational aspects. Moreover, a better understanding of the contribution to the complete system of each component was gained after their individual explanation.

The analysis of the coding aspect in chapter 3 showed the effective performance of the fluidic control box in different operational modes. From manual or PWM operation, to pressure control or data storage, each mode offered advantages for the control of soft robotics and the collection of data, both being useful for extraterrestrial exploration. However, high precision control is still far from being reached, and is left as an aspect to be developed in the future by the fluidic control box and soft robotics.

Finally, chapter 4 revealed the adaptability of the fluidic control box. The successful demonstration of the sequencing operation together with the wide possibilities offered by manual operation mode, showed a versatile system suitable for extraterrestrial exploration.

5.3 Conclusions

The analysis of the fluidic control box throughout the report has provided an exploration of its abilities for soft robotics control on extraterrestrial environments. Its structure together with its different functionalities were investigated, highlighting its versatility and adaptability.

The fluidic control box represents a promising solution for planetary exploration. However, despite the characteristics making it suitable for these operations, high precision pressure control is still an aspect to be developed.

In conclusion, the analysis of the fluidic control box deepened the understanding of its advantages and limitations in extraterrestrial missions. Future research in the improvement of those limitations opens the door for the use of the fluidic control box in planetary exploration.

5.4 Future Work

Soft robotics has experienced a large increase in their use across different areas throughout the last years and it is forecasted that this expansion will be further incremented. However, these robots currently present different limitations that need to be considered.

One significant challenge is the fact that most current soft robots rely on a continuous external power supply, and therefore need to be connected to stationary sources, limiting their area of influence.

Additionally, the use of soft materials and low pressure, while presenting advantages for flexibility and adaptability, limits the construction of large, untethered robots where weight of components can lead to collapse [3].

Another aspect lacking is the precision of soft robots, which is essential for expanding their use in applications demanding highly precise control. Soft robotics involves dealing with fluids, which difficult the obtention of highly accurate results due to factors such as compressibility and thermal effects.

In summary, future work in soft robotics should focus on overcoming challenges related to power supply, sizing design, and high precision control. This approach would result in a diversification of their applicability in a wide range of areas.

References

[1] NASA (2019). *Mars 2020 Rover*. [online] Nasa.gov. Available at: https://mars.nasa.gov/mars2020.

[2] Ng, C.S.X. and Lum, G.Z. (2021). Untethered Soft Robots for Future Planetary Explorations? *Advanced Intelligent Systems*, p.2100106. doi: https://doi.org/10.1002/aisy.202100106.

[3] Whitesides, G.M. (2018). Soft Robotics. Angewandte Chemie International Edition, 57(16), pp.4258–4273. doi: https://doi.org/10.1002/anie.201800907.

[4] Umedachi, T., Vikas, V. and Trimmer, B.A. (2016). Softworms : the design and control of non-pneumatic, 3D-printed, deformable robots. Bioinspiration & Biomimetics, 11(2), p.025001. doi: https://doi.org/10.1088/1748-3190/11/2/025001.

[5] Wyss Institute. (2016). *The first autonomous, entirely soft robot*. [online] Available at: https://wyss.harvard.edu/news/the-first-autonomous-entirely-soft-robot/.

[6] Ilievski, F., Mazzeo, A.D., Shepherd, R.F., Chen, X. and Whitesides, G.M. (2011). Soft Robotics for Chemists. Angewandte Chemie International Edition, 50(8), pp.1890–1895. doi: https://doi.org/10.1002/anie.201006464.

[7] Marchese, A.D., Onal, C.D. and Rus, D. (2014). Autonomous Soft Robotic Fish Capable of Escape Maneuvers Using Fluidic Elastomer Actuators. *Soft Robotics*, 1(1), pp.75–87. doi: https://doi.org/10.1089/soro.2013.0009

[8] Xavier, M.S., Fleming, A.J. and Yong, Y.K. (2021). Design and Control of Pneumatic Systems for Soft Robotics: A Simulation Approach. *IEEE Robotics and Automation Letters*, 6(3), pp.5800–5807. doi: https://doi.org/10.1109/lra.2021.3086425.

[9] uk.rs-online.com. (n.d.). *RS PRO 72W Plug-In AC/DC Adapter 24V dc Output, 3A Output | RS*. [online] Available at: https://uk.rs-online.com/web/p/ac-dc-adapters/9048503.

[10] 20W Adjustable DC-DC Buck Converter. https://mou.sr/43RNOqI

[11] SparkFun (2020). *How to Use a Breadboard - learn.sparkfun.com*. [online] Sparkfun.com. Available at: https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all.

[12] Hemmings, M. (2018). *What is a Jumper Wire?* [online] blog.sparkfuneducation.com. Available at: https://blog.sparkfuneducation.com/what-is-jumper-wire.

[13] 4D Systems. (n.d.). *GEN4-ULCD-50DT*. [online] Available at: https://4dsystems.com.au/products/gen4-ulcd-50dt/.

[14] 4D Systems. (n.d.). *4D-ARDUINO-ADAPTOR-SHIELD-II*. [online] Available at: https://4dsystems.com.au/products/4d-arduino-adaptor-shield-ii/ .

[15] Ibrahim, D. (2014). *Pulse Width Modulation - an overview | ScienceDirect Topics*. [online] www.sciencedirect.com. Available at: https://www.sciencedirect.com/topics/engineering/pulse-width-modulation.

[16] Industries, A. (n.d.). *Adafruit Assembled Data Logging shield for Arduino*. [online] www.adafruit.com. Available at: https://www.adafruit.com/product/1141 .

[17] EIT | Engineering Institute of Technology. (n.d.). *Types and Applications of Microcontrollers*. [online] Available at: https://www.eit.edu.au/resources/types-and-applications-of-microcontrollers/#:~:text=Microcontroller%20is%20a%20compressed%20micro.

[18] Arduino.cc. (2024). Available at: https://docs.arduino.cc/hardware/mega-2560/#features.

[19] Riordan, M. (2019). transistor | Definition & Uses. In: *Encyclopædia Britannica*. [online] Available at: https://www.britannica.com/technology/transistor.

[20] www.martview.com. (n.d.). *4-Channel MOS FET PLC Amplifier Board Driver Module*. [online] Available at: https://www.martview.com/4-channel-mos-fet-plc-amplifier-board-driver-module.html .

[21] uk.rs-online.com. (n.d.). *VV100-S41-06-M5 | SMC V100 series 6 station Metric M5 Manifold | RS*. [online] Available at: https://uk.rs-online.com/web/p/manifold-bases-sub-bases-end-bases/1964442?gb=s.

[22] uk.rs-online.com. (n.d.). *Air Hose | Pneumatic Tubing | RS*. [online] Available at: https://uk.rs-online.com/web/c/pneumatics-hydraulics/pneumatic-connectors-fittings-hose/air-hoses.

[23] Automation Distribution. (n.d.). *SMC ITV0010-2BS Compact Electro-Pneumatic Regulator*. [online] Available at: https://automationdistribution.com/smc-itv0010-2bs/ [Accessed 10 Apr. 2024].

[24] es.rs-online.com. (n.d.). *Sensor de presión del calibrador, SSCDANN015PG2A3, DIP 8 pines 103kPa | RS*. [online] Available at: https://es.rs-online.com/web/p/sensores-de-presion-para-pcb/2119938 .

[25] 4D Systems. (n.d.). *Software*. [online] Available at: https://4dsystems.com.au/software/.

[26] GitHub. (2024). *4dsystems/ViSi-Genie-Arduino-Library*. [online] Available at: https://github.com/4dsystems/ViSi-Genie-Arduino-Library.

[27] Microchip Technology. ATmega640/1280/1281/2560/2561 data sheet. Available at: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[28] www.arduino.cc. (n.d.). *analogWrite() - Guía de Referencia de Arduino*. [online] Available at: https://www.arduino.cc/reference/es/language/functions/analog-io/analogwrite/

[29] Arduino (n.d.). *Wire - Arduino Reference*. [online] www.arduino.cc. Available at: https://www.arduino.cc/reference/en/language/functions/communication/wire/.

[30] Basic Board Mount Pressure Sensors. (n.d.). Available at: https://cdn.sparkfun.com/assets/5/9/8/4/f/ABPLLNN600MGAA3.pdf.

[31] www.arduino.cc. (n.d.). *requestFrom() - Arduino Reference*. [online] Available at: https://www.arduino.cc/reference/en/language/functions/communication/wire/requestfrom/

[32] www.arduino.cc. (n.d.). *read() - Arduino Reference*. [online] Available at: https://www.arduino.cc/reference/en/language/functions/communication/wire/read/

[33] www.arduino.cc. (n.d.). *SD - Arduino Reference*. [online] Available at: https://www.arduino.cc/reference/en/libraries/sd/.

[34] Arduino (2019). *Arduino Reference*. [online] Arduino.cc. Available at: https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/.

[35] Arduino (2019). *Arduino Reference*. [online] Arduino.cc. Available at: https://www.arduino.cc/reference/en/language/functions/math/map/.

[36] Shepherd, R.F., Ilievski, F., Choi, W., Morin, S.A., Stokes, A.A., Mazzeo, A.D., Chen, X., Wang, M. and Whitesides, G.M. (2011). Multigait soft robot. *Proceedings of the National Academy of Sciences*, 108(51), pp.20400–20403. doi: https://doi.org/10.1073/pnas.1116564108.

Appendix: Arduino complete code.

```
// Including the necessary libraries

#include <genieArduino.h> // Library for screen programming
#include <Wire.h>
#include "SD.h"
#include "RTClib.h"

Genie genie; // Creates a new instance named 'genie'

#define RESETLINE 4  // Change this if you are not using an Arduino Adaptor
Shield Version 2 (see code below)

//////////////////////////////////////////////////////////////////
Variables definition
//////////////////////////////////////////////////////////////////////////////
////////

int prescaler = 256; // Definition of the prescaler value used for Timers.
Change CS in TCCRxB if changed.

// PWM-Manual operation.

int man = 0; // Definition of the manual-pwm button. 0 for manual operation 1
for PWM.

float freq = 0; // Definition of the frequency of the whole system.
int DutyC1 = 0; // Definition of the duty cycle of valve 1 for PWM.
int DutyC2 = 0; // Definition of the duty cycle of valve 2 for PWM.
int DutyC3 = 0; // Definition of the duty cycle of valve 3 for PWM.
int DutyC4 = 0; // Definition of the duty cycle of valve 4 for PWM.

int valve1 = 0; // Definition of the valve 1 button. 0 for released. 1 for
pressed.
int valve2 = 0; // Definition of the valve 2 button. 0 for released. 1 for
pressed.
int valve3 = 0; // Definition of the valve 3 button. 0 for released. 1 for
pressed.
int valve4 = 0; // Definition of the valve 4 button. 0 for released. 1 for
pressed.
int DutyC1m = 0; // Definition of the duty cycle of valve 1 for the manual
operation. It will only be 0 or 100 meaning that the valve is either open or
closed.
int DutyC2m = 0; // Definition of the duty cycle of valve 2 for the manual
operation. It will only be 0 or 100 meaning that the valve is either open or
closed.
```

```
int DutyC3m = 0; // Definition of the duty cycle of valve 3 for the manual
operation. It will only be 0 or 100 meaning that the valve is either open or
closed.
int DutyC4m = 0; // Definition of the duty cycle of valve 4 for the manual
operation. It will only be 0 or 100 meaning that the valve is either open or
closed.
float freqm = 1; // Definition of the frequency of the whole system for the
manual operation. Its value won't matter since the Duty Cycles will be 0 or
100.

// Pressure settings.

int Pset = 0; // Definition of the pressure to be set.

// Start, pause, stop.

int pause = 0; // Definition of the pause button. 0 for released. 1 for
pressed.
int start = 0; // Definition of the start button. 0 for released. 1 for
pressed.

// Sequencing operation.

int sequencing = 0; // Definition of the sequencing button variable. 0 for
released. 1 for pressed.
int order = 1; // Definition of the variable used for the order of the
sequence's digits in the LED.
int seqnum = 0; // Definition of the current valve selected for the sequence
setting. It will change as the next valve is selected and will be added to the
LED digits.
int numLED = 0; // Definition of the complete number of the sequence which is
written in the LED.
int ts1 = 0; // Definition of the time between the 1st and 2nd valve opening
in the sequence.
int ts2 = 0; // Definition of the time between the 2nd and 3rd valve opening
in the sequence.
int ts3 = 0; // Definition of the time between the 3rd and 4th valve opening
in the sequence.
int ts4 = 0; // Definition of the time between the 4th and 1st valve opening
in the sequence.
int i = 0; // Definition of the variable used to know which valve is to be
opened at the considered time.
int ts = 0; // Definition of the time used for the sequential opening
programming, changing as it goes to the next valve in the sequence (ts1, ts2,
ts3...).
unsigned long previousMillis = 0; // Definition of the variable used for the
storage of miliseconds to check if it is time to open the next
// valve in the sequence.
```

```
// Duration of the experiment operation.

int duration = 0;  // Definition of the duration of the experiment button. 0
for released. 1 for pressed.
unsigned long startTime = 0; // Definition of the variable used for the
storage of miliseconds to check if it is time to stop the system.
int texp = 0; // Definition of the duration of the experiment. The system will
stop after the time is elapsed.
int n = 0; // Definition of the variable used to know if it is time to stop
the system.

// Pressure reading.

uint16_t P_bin = 0; // Definition of the variable used for reading the
pressure in 14 bits binary.
unsigned long previousMillisread = 0; // Definition of the variable used for
the storage of miliseconds to check if it is time to read the pressure.
float Pread = 0; // Definition of the variable used for reading the pressure.

// Scope display.
int traceValue1 = 0; // Definition of the mapped variable used for the 1st
signal in the scope.
int traceValue2 = 0; // Definition of the mapped variable used for the 2nd
signal in the scope.
int traceValue3 = 0; // Definition of the mapped variable used for the 3rd
signal in the scope.
int traceValue4 = 0; // Definition of the mapped variable used for the 4th
signal in the scope.
int sensorValue = 0; // Definition of the variable used for the reading of the
1st signal in the scope.
int sensorValue2 = 0; // Definition of the variable used for the reading of
the 2nd signal in the scope.
int sensorValue3 = 0; // Definition of the variable used for the reading of
the 3rd signal in the scope.
int sensorValue4 = 0; // Definition of the variable used for the reading of
the 4th signal in the scope.

// Data storage.
int datastor = 0; // Definition of the data storage button. 0 for released. 1
for pressed.
unsigned long previousMillisdatstor = 0; // Definition of the variable used
for the storage of miliseconds to check if it is time to store the data.
RTC_PCF8523 RTC; // define the Real Time Clock object
uint32_t syncTime = 0; // time of last sync()


// for the data logging shield, we use digital pin 10 for the SD cs line
const int chipSelect = 10;
```

```
   // the logging file
      File logfile;

void setup()
{
  Wire.begin();             // Join I2C bus (address is optional for
controller device)
  Serial.begin(9600); // Make sure it is the same baud rate for the screen in
ViSi Genie.
  Serial.println();

  genie.Begin(Serial); // Sets Serial0 to be used by the Genie instance
'genie'

  genie.AttachEventHandler(myGenieEventHandler); // Name of the function used
to Handle events.

  pinMode(RESETLINE, OUTPUT);  // Set D4 on Arduino to Output (4D Arduino
Adaptor V2 - Display Reset)
  digitalWrite(RESETLINE, 1);  // Reset the Display via D4
  delay(100);
  digitalWrite(RESETLINE, 0);  // unReset the Display via D4

  delay (7000); // Let the display start up after the reset for 3.5 s.

  // Definition of output pins for PWM
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);

  // Definition of output pin for Pressure Regulator.
  pinMode(11, OUTPUT);

 // Setting TCCR registers for Timer 3 and 4 to eliminate previous settings.
  TCCR3A = 0b00000000;
  TCCR3B = 0b00000000;
  TCCR4A = 0b00000000;
  TCCR4B = 0b00000000;

  // Setting WGM to PWM, Phase and Frequency correct. Clock Selector to
prescaler 256. Compare Output Mode to Non-Inverting.
  TCCR3A = _BV(COM3A1);
  TCCR3B = _BV(WGM33) | _BV(CS32);

  TCCR4A = _BV(COM4A1) | _BV(COM4B1) | _BV(COM4C1);
  TCCR4B = _BV(WGM43) | _BV(CS42);
}
```

```
void loop()
{

  genie.DoEvents(); // Receives responses from the display and then calls the
event handler function.

  if (start == 1) // If the system is operating.
  {
     if (datastor == 1) // If the data storage operation is selected.
    {
      datastorage(); // Calls the data storage function.
    }
    if (duration == 1) // If the duration of the experiment operation is
selected.
    {
      durationexp(texp); // Calls the duration function.
    }

    if (sequencing == 1) // If the sequencing operation is selected.
    {
      seqopen(numLED, ts1, ts2, ts3, ts4); // Calls the sequential opening
function.
    }

    else // If the sequencing operation is not selected.
    {
        if (man == 1) // If the PWM operation is selected.
        {
            // Generates the PWM signal for each of the valves with the
corresponding Duty Cycles and frequency.
            PWMfunc(freq, DutyC1, DutyC2, DutyC3, DutyC4);
        }
        else if (man == 0) // If the Manual operation is selected.
        {
            // Generates the full opening or closing the valves by Duty Cycles
of 0 or 100.
            PWMfunc(freqm, DutyC1m, DutyC2m, DutyC3m, DutyC4m);
        }
    }
    // Reads the pressure every 1s.
    pressurereading();
  }


  else if (start == 0) // If the system is not operating.
  {
    // No signal is sent.
    OCR3A = 0;
    OCR4A = 0;
```

```
    OCR4B = 0;
    OCR4C = 0;
  }


  sensorValue = analogRead(A0);
  traceValue1 = map(sensorValue,0,1023,0,100);
  genie.WriteObject(GENIE_OBJ_SCOPE, 0, traceValue1);

  sensorValue2 = analogRead(A1);
  traceValue2 = map(sensorValue2,0,1023,0,100);
  genie.WriteObject(GENIE_OBJ_SCOPE, 0, traceValue2);

  sensorValue3 = analogRead(A2);
  traceValue3 = map(sensorValue3,0,1023,0,100);
  genie.WriteObject(GENIE_OBJ_SCOPE, 0, traceValue3);

  sensorValue4 = analogRead(A3);
  traceValue4 = map(sensorValue4,0,1023,0,100);
  genie.WriteObject(GENIE_OBJ_SCOPE, 0, traceValue4);

  // Sets the desired pressure in the pressure regulator.
    analogWrite(11, Pset);
}

void error(char *str)
{
  Serial.print("error: ");
  Serial.println(str);

  while(1);
}

// Function to generate the PWM for each of the 4 valves considering frequency
and duty cycle for each of them.
void PWMfunc(float freq, int DutyC1, int DutyC2, int DutyC3,int DutyC4)
{
  // Sets the maximum value to be reached by the clock (top of the wave)
considering the desired frequency.
  ICR3 = F_CPU / (prescaler * freq * 2);
  ICR4 = F_CPU / (prescaler * freq * 2);

  // Sets the number of clocks to be reached before changing to LOW state,
considering the previously calculated
  // ICRx value and the desired duty cycle.
  OCR3A = (ICR4) * (DutyC1 * 0.01);
  OCR4A = (ICR4) * (DutyC2 * 0.01);
  OCR4B = (ICR4) * (DutyC3 * 0.01);
  OCR4C = (ICR4) * (DutyC4 * 0.01);
```

```
}

// Function to write the order in the sequence in the LED digits.
void seqLED(int seqNUM, int ord)
{
  if (order == 5) // Once the number of selected valves for the sequence is
bigger than the maximum.
  {
    numLED = 0; // Sets the LED number to 0 again.
    order = 1; // Sets the order of the sequence's digits to start again with
the first digit. This variable is external to the function.
    ord = 1; // Sets the order of the sequence's digits to start again with
the first digit. This variable is internal to the function and it is only set
internally
    // once the maximum number of digits has been reached, otherwise it just
takes the value of the external variable which is the input.
  }

  numLED = numLED + (seqNUM) * round(pow(10,(4 - ord))); // Sets the digits of
the sequence number by using powers of 10 starting by 10^3 for the first valve
of the sequence.
  // The previous value of numLED is added to keep the previous digits every
time a new digit of the sequence is set.

  // Write the selected sequence in LedDigits7.
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 7, numLED);
}

// Function to open the valves in the desired order. The inputs are the
sequencing order as a number of 4 digits and the time between the opening of
each valve.
void seqopen(int numLEDo, int ts1o, int ts2o, int ts3o, int ts4o)
{
  if (i == 1) // If it is the time to open the first valve.
  {
    ts = ts1o * 1000; // Sets the time to be elapsed to ts1. Multiplied by
1000 to convert to miliseconds.
  }

  else if (i == 2) // If it is the time to open the second valve.
  {
    ts = ts2o * 1000; // Sets the time to be elapsed to ts2. Multiplied by
1000 to convert to miliseconds.
  }

  else if (i == 3) // If it is the time to open the third valve.
  {
    ts = ts3o * 1000; // Sets the time to be elapsed to ts3. Multiplied by
1000 to convert to miliseconds.
```

```
  }

  else if (i == 4) // If it is the time to open the fourth valve.
  {
    ts = ts4o * 1000; // Sets the time to be elapsed to ts4. Multiplied by
1000 to convert to miliseconds.
  }

  else if (i == 5) // If the fourth valve has finished.
  {
    i = 1; // Start with the 1st valve again.
    ts = ts1o * 1000; // Sets the time to tbe elapsed to ts1.
  }


  unsigned long currentMillis = millis(); // Stores the current value of
miliseconds since the Arduino first ran.
  if (currentMillis - previousMillis >= ts) // Checks if the considered time
has been elapsed, by comparing the current value of millis() to the value of
miliseconds that
  // was stored when the opening of the valve started.
  {
    // Saves the value of the current miliseconds so that in the future this
value can be compared to the increasing millis() and check if the desired time
has been elapsed.
    previousMillis = currentMillis;

    i = i + 1; // Increases to start the opening of the next valve in the
sequence.
  }

  String strNum = String(numLEDo); // Converts the 4 digit number that stores
the sequence to a string.

  int valve = strNum.charAt(i - 1) - '0'; // Checks the current valve to be
opened by selecting the corresponding digit on the sequence number.

  if (valve == 1) // If the valve to be opened is the 1.
  {
    PWMfunc(freq, DutyC1, 0, 0, 0); // Generates the PWM signal for valve 1
and closes the other ones.
  }

  else if (valve == 2) // If the valve to be opened is the 2.
  {
    PWMfunc(freq, 0, DutyC2, 0, 0); // Generates the PWM signal for valve 2
and closes the other ones.
  }
```

```
  else if (valve == 3) // If the valve to be opened is the 3.
  {
    PWMfunc(freq, 0, 0, DutyC3, 0); // Generates the PWM signal for valve 3
and closes the other ones.
  }

  else if (valve == 4) // If the valve to be opened is the 4.
  {
    PWMfunc(freq, 0, 0, 0, DutyC4); // Generates the PWM signal for valve 4
and closes the other ones.
  }

}

// Function to stop the system after the time set for the duration of the
experiment has been elapsed.
void durationexp(int T)
{
  unsigned long currentMillisdur = millis(); // Stores the current value of
miliseconds since the Arduino first ran.

  if (currentMillisdur - startTime >= T * 1000) // Checks if the considered
time has been elapsed, by comparing the current value of millis() to the value
of miliseconds that
  // was stored when the duration of the experiment was set. Multiplied by
1000 to convert to miliseconds.
  {
    // Save the last time the sequence was updated
    startTime = currentMillisdur;

    switch (n) // Switches n to check if the conditions below are
accomplished.
    {
      case 0: // If n is 0 it will satisfy the if condition above because
startTime = 0 and the difference between the current milliseconds and
startTime will be bigger than T.
      {
        n = n + 1; // Therefore n is increased so that the next time the if
condition is satisfied the system will actually be stopped.
        break; // Leaves the switch loop.
      }

      case 1: // If n is 1 this will mean that it is not the first access to
the if condition and now actually the duration time has been elapsed.
      {
        start = 0; // Therefore the system is stopped.
        startTime = 0; // startTime is set to 0 to allow the next countdown.
        n = 0; // n is set to 0 to allow the next countdown.
        break; // Leaves the switch loop.
```

```
      }
    }

  }
}

// Function that reads the pressure every 1s through the pressure transducer.
void pressurereading(void)
{
  unsigned long currentMillisread = millis(); // Stores the current value of
miliseconds since the Arduino first ran.

  if (currentMillisread -  previousMillisread >= 1000) // Checks if the
considered time of 1s between pressure reading has been elapsed, by comparing
the current value of millis()
  // to the value of miliseconds that was stored when the duration of the
experiment was set. Multiplied by 1000 to convert to miliseconds.
  {
    // Save the last time the sequence was updated
    previousMillisread = currentMillisread;

    Wire.requestFrom(0x28, 2); // Request 2 bytes (16 bits) from the sensor

    if (Wire.available()) // Check if data is available
    {
      byte highByte = Wire.read(); // Read the high byte (most significant
bits)
      byte lowByte = Wire.read(); // Read the low byte (least significant
bits)

      // Combine the high and low bytes into a 14-bit number eliminating the
first two bits that correspond to the sensor state data.
      P_bin = ((highByte & 0x3F) << 8) | lowByte;

      // The transfer function for the Honeywell SSCDANN015PAAA5 pressure
transducer is used to calculate the pressure reading.
      // Transfer function given by Output (% of 2^14 counts) = 80%/(P_max -
P_min)*(P_read - P_min) + 10%.
      // Where P_max = 15 psi, P_min = P_atm = 14.696 psi and we want to
obtain P_applied. Rearranging and substituting:

      Pread = ((P_bin) / 16383.00 - 0.1) * (15 - 14.696) / 0.8 + 14.696;

      // Write the pressure value read in LED Digits 6.
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, round(Pread*100));
    }
  }
}
```

```
// Function to stop the system after the time set for the duration of the
experiment has been elapsed.
void datastorage(void)
{
  unsigned long currentMillisdatstor = millis(); // Stores the current value
of miliseconds since the Arduino first ran.

  if (currentMillisdatstor - previousMillisdatstor >= 1000) // Checks if the
considered time has been elapsed, by comparing the current value of millis()
to the value of miliseconds that
  // was stored when the last data storage was produced.
  {

    // log in the SD card the the miliseconds elapsed since last data storage.
    logfile.print(currentMillisdatstor); // miliseconds elapsed since last
data storage.

    // log in the SD card the current pressure read of the system.
    logfile.print(", ");
    logfile.print(Pread);

    // log in the SD card the current pressure set of the system.
    logfile.print(", ");
    logfile.print(Pset);

    // log in the SD card if the manual or PWM parameters.
    if (man == 0) // Manual operation set.
    {
    // log in the SD card that PWM operation is set.
    logfile.print(", ");
    logfile.print("Manual");

    // log in the SD card the Duty Cycles of each valve and frequency of the
system in manual operation.
    logfile.print(", ");
    logfile.print(DutyC1m);
    logfile.print(", ");
    logfile.print(DutyC2m);
    logfile.print(", ");
    logfile.print(DutyC3m);
    logfile.print(", ");
    logfile.print(DutyC4m);
    logfile.print(", ");
    logfile.print(freqm);
    }

    else if (man == 1) // PWM operation set.
    {
    // log in the SD card that PWM operation is set.
```

```
    logfile.print(", ");
    logfile.print("PWM");


    // log in the SD card the Duty Cycles of each valve and frequency of the
system in PWM operation.
    logfile.print(", ");
    logfile.print(DutyC1);
    logfile.print(", ");
    logfile.print(DutyC2);
    logfile.print(", ");
    logfile.print(DutyC3);
    logfile.print(", ");
    logfile.print(DutyC4);
    logfile.print(", ");
    logfile.print(freq);
    }

    // log in the SD card if the sequencing operation is set.
    if (sequencing == 0) // Sequencing operation not set.
    {
    // log in the SD card if sequencing operation is set.
    logfile.print(", ");
    logfile.print("Off");

    // log in the SD card the sequence number and the time of opening for each
valve.
    logfile.print(", ");
    logfile.print("-");
    logfile.print(", ");
    logfile.print("-");
    logfile.print(", ");
    logfile.print("-");
    logfile.print(", ");
    logfile.print("-");
    logfile.print(", ");
    logfile.print("-");
    logfile.print(", ");
    logfile.print("-");
    }

    else if (sequencing == 1) // sequencing operation set.
    {
    // log in the SD card if sequencing operation is set.
    logfile.print(", ");
    logfile.print("On");;


    /// log in the SD card the sequence number and the time of opening for
each valve.
    logfile.print(", ");
```

```
    logfile.print(numLED);
    logfile.print(", ");
    logfile.print(ts1);
    logfile.print(", ");
    logfile.print(ts2);
    logfile.print(", ");
    logfile.print(ts3);
    logfile.print(", ");
    logfile.print(ts4);
    }

    // log in the SD card if the duration of the experiment operation is set.
    if (duration == 0) // Duration of the experiment operation not set.
    {
    // log in the SD card if sequencing operation is set.
    logfile.print(", ");
    logfile.print("Off");

    // log in the SD card the sequence number and the time of opening for each
valve.
    logfile.print(", ");
    logfile.println("-");
    }

    else if (duration == 1) // Duration of the experiment operation set.
    {
    // log in the SD card if sequencing operation is set.
    logfile.print(", ");
    logfile.print("On");;


    /// log in the SD card the sequence number and the time of opening for
each valve.
    logfile.print(", ");
    logfile.println(texp);
    }

    // Save the last time the sequence was updated
    previousMillisdatstor = currentMillisdatstor;

    // Now we write data to disk! Don't sync too often - requires 2048 bytes
of I/O to SD card
    // which uses a bunch of power and takes time
    if ((millis() - syncTime) < 2000) return;
    syncTime = millis();

    logfile.flush();

  }
```

```
}
// Function that receives the events coming from the screen,i.e, the buttons
pressed and acts accordingly to the selected buttons.
void myGenieEventHandler(void)
{
  genieFrame Event;
  genie.DequeueEvent(&Event); // Remove the next queued event from the buffer,
and process it below.

  ////////////////////////////////////////////////// Start, pause, stop
buttons ////////////////////////////////////////////////////////////

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 10 (start button) and if the start button is released. If both are
satisfied then:
  if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 10)) &&
(start == 0))
  {
    start = 1; // Sets the variable start to 1. This represents that the
system is on.
    pause = 0; // Sets the pause button to released.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 11 (pause button) and if the system was operating.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 11))
&& (start == 1))
  {
    start = 0; // Sets the variable start to 0 to pause the system.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 12 (stop button) and if the system was operating.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 12))
&& (start == 1))
  {
    start = 0; // Sets the variable start to 0 to stop the system.

    DutyC1 = 0; // Clears Duty Cycle of Valve 1 setting it to 0 (initial
value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0, DutyC1);  // Write Duty Cycle
of Valve1 value to LedDigits0.

    DutyC2 = 0; // Clears Duty Cycle of Valve 2 setting it to 0 (initial
value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 1, DutyC2);  // Write Duty Cycle
of Valve2 value to LedDigits1.
```

```
    DutyC3 = 0; // Clears Duty Cycle of Valve 3 setting it to 0 (initial
value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 2, DutyC3);  // Write Duty Cycle
of Valve3 value to LedDigits2.

    DutyC4 = 0; // Clears Duty Cycle of Valve 4 setting it to 0 (initial
value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, DutyC4);  // Write Duty Cycle
of Valve4 value to LedDigits3.

    freq = 0; // Clears frequency of all valves setting it to 0 (initial
value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 4, freq);  // Write the frequency
of all valves value to LedDigits4.

    Pset = 0; // Clears pressure of the system setting it to 0 (initial
value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 5, Pset);  // Write the pressure
of the system value to LedDigits5.

    numLED = 0; // Clears the sequence number setting it to 0 (initial value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 7, numLED); // Write the selected
sequence in LedDigits7.

    ts1 = 0; // Clears time of opening for the 1st valve setting it to 0
(initial value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 8, ts1);  // Write time of opening
for the 1st selected valve to LedDigits8.

    ts2 = 0; // Clears time of opening for the 2nd valve setting it to 0
(initial value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 9, ts2);  // Write time of opening
for the 2nd selected valve to LedDigits9.

    ts3 = 0; // Clears time of opening for the 3rd valve setting it to 0
(initial value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 10, ts3);  // Write time of
opening for the 3rd selected valve to LedDigits10.

    ts4 = 0; // Clears time of opening for the 4th valve setting it to 0
(initial value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 11, ts4);  // Write time of
opening for the 4th selected valve to LedDigits11.

    order = 1; // Sets the variable used for the order of the sequence's
digits in the LED to 1 (initial value).

    i = 0; // Sets the variable used to know which valve is to be opened at
the considered time to 0 (initial value).
```

```
    ts = 0; // Sets the time used for the sequential opening programming,
changing as it goes to the next valve in the sequence (ts1, ts2, ts3...) to 0
(initial value).

    previousMillis = 0; // Sets the variable used for the storage of
miliseconds to check if it is time to open the next
    // valve in the sequence to 0 (initial value).

    startTime = 0; // Sets the variable used for the storage of miliseconds to
check if it is time to stop the system to 0 (initial value).

    texp = 0; // // Clears time of the duration of the experiment setting it
to 0 (initial value).
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 12, texp);  // Write time of the
experiment to LedDigits12.

    n = 0;

  }

  ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

  ////////////////////////////////////////////////////// Pressure set button
////////////////////////////////////////////////////////////////////////////

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 13 (pressure setting +) and prevents the pressure from going bigger
than 15 psi.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 13))
&& (Pset <= 12))
  {
    Pset = Pset + 3; // // Change the pressure of the system by increasing it
for 3 psi.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 5, Pset);  // Write the pressure
of the system value to LedDigits5.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 14 (pressure setting -) and prevents the pressure from going smaller
than 0 psi.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 14))
&& (Pset >= 3))
  {
    Pset = Pset - 3; // // Change the pressure of the system by decreasing it
for 3 psi.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 5, Pset);  // Write the pressure
of the system value to LedDigits5.
```

```
  }

  ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

  ////////////////////////////////////////////////// Data Storage button
////////////////////////////////////////////////////////////////////

  if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 20))
  {
    switch (datastor) // Switches man to check if the conditions below are
accomplished.
    {
      case 0: // If data storage was not ON.
      {
      datastor = 1; // Turn it ON.

      // initialize the SD card
      Serial.print("Initializing SD card...");
      // make sure that the default chip select pin is set to
      // output, even if you don't use it:
      pinMode(10, OUTPUT);

      // see if the card is present and can be initialized:
      if (!SD.begin(chipSelect)) {
        error("Card failed, or not present");
      }
      Serial.println("card initialized.");

      // create a new file
      char filename[] = "LOGGER00.CSV";
      // Changes the file name till it finds one that has not been used.
      for (uint8_t i = 0; i < 100; i++) {
        filename[6] = i/10 + '0'; // Sets the first digit of LOGGERnn
corresponding to the tens.
        filename[7] = i%10 + '0'; // Sets the second digit of LOGGERnn
corresponding to the units.
        if (! SD.exists(filename)) {
          // only open a new file if it doesn't exist
          logfile = SD.open(filename, FILE_WRITE);
          break;  // leave the loop!
        }
      }

      logfile.println("millis (ms),Pread (psi),Pset (psi),PWM/Manual,Duty
Cycle 1 (%),Duty Cycle 2 (%),Duty Cycle 3 (%), \
      Duty Cycle 4 (%),frequency (Hz),Sequencing,Sequence number,Time 1st
valve (s),Time 2nd valve (s),Time 3rd valve (s), \
```

```
      Time 4th valve (s),Duration of the experiment,texp (s) "); // Logs the
variables to be stored in columns in CSV format in the SD card.

      break; // Leaves the switch loop.
      }

      case 1: // If data storage was ON.
      {
      datastor = 0;  // Turn it OFF.
      break; // Leaves the switch loop.
      }
    }

  }

  ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

  ////////////////////////////////////////////////////// Sequencing button
////////////////////////////////////////////////////////////////////

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 21 (Sequencing).
  else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 21))
  {
    switch (sequencing) // Switches man to check if the conditions below are
accomplished.
    {
      case 0: // If sequencing option was not ON.
      sequencing = 1; // Turn it ON.
      break; // Leaves the switch loop.

      case 1: // If sequencing option was ON.
      sequencing = 0;  // Turn it OFF.
      break; // Leaves the switch loop.
    }
  }

  ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

  ////////////////////////////////////////////////////// Sequencing
Operation ////////////////////////////////////////////////////////////////

  if (sequencing == 1) // If the sequencing operation is selected.
  {
    // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 15 (Valve 1).
    if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 15))
```

```
    {
      seqnum = 1; // Sets the sequence digit to 1.
      seqLED(seqnum, order); // Calls the function to write the sequence order
in LED.
      order = order + 1; // Increases order variable to set the next valve in
the sequence.
    }

    // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 16 (Valve 2).
    else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON,
16))
    {
      seqnum = 2; // Sets the sequence digit to 2.
      seqLED(seqnum, order); // Calls the function to write the sequence order
in LED.
      order = order + 1; // Increases order variable to set the next valve in
the sequence.
    }
    // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 17 (Valve 3).
    else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON,
17))
    {
      seqnum = 3; // Sets the sequence digit to 3.
      seqLED(seqnum, order); // Calls the function to write the sequence order
in LED.
      order = order + 1; // Increases order variable to set the next valve in
the sequence.
    }
    // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 18 (Valve 4).
    else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON,
18))
    {
      seqnum = 4; // Sets the sequence digit to 4.
      seqLED(seqnum, order); // Calls the function to write the sequence order
in LED.
      order = order + 1; // Increases order variable to set the next valve in
the sequence.
    }
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 22 (ts1 setting +) and prevents the time of opening for the 1st
selected valve
  // from going bigger than 10.
  if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 22)) &&
(ts1 <=  9))
```

```
  {
    ts1 = ts1 + 1;                      // Change the time of opening for the
1st selected valve by increasing it 1 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 8, ts1);  // Write time of opening
for the 1st selected valve to LedDigits8.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 23 (ts1 setting -) and prevents the time of opening for the 1st
selected valve
  // from going smaller than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 23))
&& (ts1 >=  1))
  {
    ts1 = ts1 - 1;                       // Change the time of opening for the
1st selected valve by decreasing it 1 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 8, ts1);  // Write time of opening
for the 1st selected valve to LedDigits8.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 24 (ts2 setting +) and prevents the time of opening for the 2nd
selected valve
  // from going bigger than 10.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 24))
&& (ts2 <=  9))
  {
    ts2 = ts2 + 1;                      // Change the time of opening for the
2nd selected valve by increasing it 1 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 9, ts2);  // Write time of opening
for the 2nd selected valve to LedDigits9.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 25 (ts2 setting -) and prevents the time of opening for the 2nd
selected valve
  // from going smaller than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 25))
&& (ts2 >=  1))
  {
    ts2 = ts2 - 1;                       // Change the time of opening for the
2nd selected valve by decreasing it 1 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 9, ts2);  // Write time of opening
for the 2nd selected valve to LedDigits9.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 26 (ts3 setting +) and prevents the time of opening for the 3rd
selected valve
```

```
  // from going bigger than 10.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 26))
&& (ts3 <=  9))
  {
    ts3 = ts3 + 1;                      // Change the time of opening for the
3rd selected valve by increasing it 1 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 10, ts3);  // Write time of
opening for the 3rd selected valve to LedDigits10.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 27 (ts3 setting -) and prevents the time of opening for the 3rd
selected valve
  // from going smaller than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 27))
&& (ts3 >=  1))
  {
    ts3 = ts3 - 1;                       // Change the time of opening for the
3rd selected valve by decreasing it 1 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 10, ts3);  // Write time of
opening for the 3rd selected valve to LedDigits10.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 28 (ts4 setting +) and prevents the time of opening for the 4th
selected valve
  // from going bigger than 10.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 28))
&& (ts4 <=  9))
  {
    ts4 = ts4 + 1;                      // Change the time of opening for the
4th selected valve by increasing it 1 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 11, ts4);  // Write time of
opening for the 4th selected valve to LedDigits11.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 29 (ts4 setting -) and prevents the time of opening for the 4th
selected valve
  // from going smaller than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 29))
&& (ts4 >=  1))
  {
    ts4 = ts4 - 1;                       // Change the time of opening for the
4th selected valve by decreasing it 1 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 11, ts4);  // Write time of
opening for the 4th selected valve to LedDigits11.
  }
```

```
  ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////// Duration of the
experiment button ////////////////////////////////////////////////////////

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 31 (Duration).
  else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 31))
  {
    switch (duration) // Switches man to check if the conditions below are
accomplished.
    {
      case 0: // If duration operation was not ON.
      duration = 1; // Turn it ON.
      break; // Leaves the switch loop.

      case 1: // If duration operation was ON.
      duration = 0;  // Turn it OFF.
      break; // Leaves the switch loop.
    }
  }

  ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////


  ////////////////////////////////////////////////// Duration of the
experiment Operation ////////////////////////////////////////////////////////

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 30 (texp setting +) and prevents the time of the experiment
  // from going bigger than 100.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 30))
&& (texp <=  90))
  {
    texp = texp + 10;                       // Change the time of the
experiment by increasing it 10 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 12, texp);  // Write time of the
experiment to LedDigits12.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 32 (texp setting -) and prevents the time of the experiment
  // from going smaller than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 32))
&& (texp >=  10))
  {
```

```
    texp = texp - 10;                       // Change the time of the
experiment by decreasing it 10 second.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 12, texp);  // Write time of the
experiment to LedDigits12.
  }

  ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

  ////////////////////////////////////////////////// PWM / Manual Operation
button ////////////////////////////////////////////////////////////

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 19 (PWM/Manual).
  else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 19))
  {
    switch (man) // Switches man to check if the conditions below are
accomplished.
    {
      case 0: // If Manual operation was selected
      man = 1; // Change to PWM operation
      break; // Leaves the switch loop.

      case 1: // If PWM operation was selected
      man = 0;  // Change to manual operation
      break; // Leaves the switch loop.
    }
  }

  ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

  ////////////////////////////////////////////////// PWM Operation
////////////////////////////////////////////////////////////////////////////

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 0 (DutyCycle1 setting +) and prevents the Duty Cycle from going bigger
than 100.
  if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 0)) &&
(DutyC1 <=  90))
  {
    DutyC1 = DutyC1 + 10;                     // Change the Duty Cycle of
Valve 1 by increasing it a 10%.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0, DutyC1);  // Write Duty Cycle
of Valve1 value to LedDigits0.
  }
```

```
  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 1 (DutyCycle1 setting -) and prevents the Duty Cycle from going smaller
than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 1))
&& (DutyC1 >=  10))
  {
    DutyC1 = DutyC1 - 10;                       // Change the Duty Cycle of
Valve 1 by decreasing it a 10%.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0, DutyC1);  // Write Duty Cycle
of Valve1 value to LedDigits0.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 2 (DutyCycle2 setting +) and prevents the Duty Cycle from going bigger
than 100.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 2))
&& (DutyC2 <=  90))
  {
    DutyC2 = DutyC2 + 10;                       // Change the Duty Cycle of
Valve 2 by increasing it a 10%.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 1, DutyC2);  // Write Duty Cycle
of Valve2 value to LedDigits1.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 3 (DutyCycle2 setting -) and prevents the Duty Cycle from going smaller
than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 3))
&& (DutyC2 >=  10))
  {
    DutyC2 = DutyC2 - 10;                       // Change the Duty Cycle of
Valve 2 by decreasing it a 10%.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 1, DutyC2);  // Write Duty Cycle
of Valve2 value to LedDigits1.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 4 (DutyCycle3 setting +) and prevents the Duty Cycle from going bigger
than 100.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 4))
&& (DutyC3 <=  90))
  {
    DutyC3 = DutyC3 + 10;                       // Change the Duty Cycle of
Valve 3 by increasing it a 10%.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 2, DutyC3);  // Write Duty Cycle
of Valve3 value to LedDigits2.
  }
```

```
  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 5 (DutyCycle3 setting -) and prevents the Duty Cycle from going smaller
than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 5))
&& (DutyC3 >=  10))
  {
    DutyC3 = DutyC3 - 10;                       // Change the Duty Cycle of
Valve 3 by decreasing it a 10%.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 2, DutyC3);  // Write Duty Cycle
of Valve3 value to LedDigits2.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 6 (DutyCycle4 setting +) and prevents the Duty Cycle from going bigger
than 100.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 6))
&& (DutyC4 <=  90))
  {
    DutyC4 = DutyC4 + 10;                       // Change the Duty Cycle of
Valve 4 by increasing it a 10%.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, DutyC4);  // Write Duty Cycle
of Valve4 value to LedDigits3.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 7 (DutyCycle4 setting -) and prevents the Duty Cycle from going smaller
than 0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 7))
&& (DutyC4 >=  10))
  {
    DutyC4 = DutyC4 - 10;                       // Change the Duty Cycle of
Valve 4 by decreasing it a 10%.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, DutyC4);  // Write Duty Cycle
of Valve4 value to LedDigits3.
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 8 (freq setting +) and prevents the Duty Cycle from going bigger than 2
Hz.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 8))
&& (freq <=  1.8))
  {
    freq = freq + 0.2;                      // Change the frequency of all
valves by increasing it for 0.2 Hz.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 4, freq * 10);  // Write the
frequency of all valves value to LedDigits4.
  }
```

```
  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 9 (freq setting -) and prevents the Duty Cycle from going smaller than
0.
  else if ((genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 9))
&& (freq >=  0.2))
  {
    freq = freq - 0.2;                      // Change the frequency of all
valves by decreasing it for 0.2 Hz.
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 4, freq * 10);  // Write the
frequency of all valves value to LedDigits4.
  }

  ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

  ////////////////////////////////////////////////// Manual Operation
////////////////////////////////////////////////////////////////////////////

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 15 (Valve 1).
  if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 15))
  {
    switch (valve1) // Switches valve1 to check if the conditions below are
accomplished.
    {
      case 0: // If valve 1 was not open.
      DutyC1m = 100; // Opens Valve 1.
      valve1 = 1; // Sets the Valve 1 button to pressed.
      break; // Leaves the switch loop.

      case 1: // If valve 1 was open.
      DutyC1m = 0; // Closes Valve 1.
      valve1 = 0; // Sets the Valve 1 button to released.
      break; // Leaves the switch loop.
    }
  }

  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 16 (Valve 2).
  else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 16))
  {
    switch (valve2) // Switches valve2 to check if the conditions below are
accomplished.
    {
      case 0: // If valve 2 was not open.
      DutyC2m = 100; // Opens Valve 2.
      valve2 = 1; // Sets the Valve 2 button to pressed.
      break; // Leaves the switch loop.
```

```
      case 1: // If valve 1 was open.
      DutyC2m = 0; // Closes Valve 2.
      valve2 = 0; // Sets the Valve 2 button to released.
      break; // Leaves the switch loop.
    }
  }
  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 17 (Valve 3).
  else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 17))
  {
    switch (valve3) // Switches valve3 to check if the conditions below are
accomplished.
    {
      case 0: // If valve 3 was not open.
      DutyC3m = 100; // Opens Valve 3.
      valve3 = 1; // Sets the Valve 3 button to pressed.
      break; // Leaves the switch loop.

      case 1: // If valve 3 was open.
      DutyC3m = 0; // Closes Valve 3.
      valve3 = 0; // Sets the Valve 3 button to released.
      break; // Leaves the switch loop.
    }
  }
  // Check if the message stored in 'Event' is a GENIE_REPORT_EVENT from
Button 18 (Valve 4).
  else if (genie.EventIs(&Event, GENIE_REPORT_EVENT, GENIE_OBJ_4DBUTTON, 18))
  {
    switch (valve4) // Switches valve4 to check if the conditions below are
accomplished.
    {
      case 0: // If valve 4 was not open.
      DutyC4m = 100; // Opens Valve 4.
      valve4 = 1; // Sets the Valve 4 button to pressed.
      break; // Leaves the switch loop.

      case 1: // If valve 4 was open.
      DutyC4m = 0; // Closes Valve 4.
      valve4 = 0; // Sets the Valve 4 button to released.
      break; // Leaves the switch loop.
    }
  }
}
```