Sally the Soft Sole

EPFL - ME410, *Max A. Aebi, Nina Bodenstab, Charles Froessel, Marine Moutarlier, Ambre Sassi*

***Abstract*—This document describes our project made during the fall 2023 following the ME410 class at EPFL. During this class, we designed a pneumatically driven sole to give foot massages. It uses a proportional control algorithm to regulate a user-chosen internal pressure.**
In this report we elaborate on the design choices. Furthermore, we discuss the engineering specifications and present which ones we fully met and which ones we only partially met.

I. Introduction

A. Motivation

Our aspiration is to develop an innovative massage sole that effectively alleviates the discomfort and pain commonly associated with prolonged walks, extended periods of standing, and physically demanding activities, such as hiking. Our focus lies in enhancing blood circulation and mitigating edema, with a specific target audience in mind – individuals engaged in professions involving prolonged periods of standing, such as the service industry. Our ultimate objective is to provide a solution that not only offers comfort and relief but also contributes to the overall well-being and improved quality of life for our users.

II. State of the art

At first glance, a product similar to the one we developed seems to exist, see [1]. Its name is InflaSole, and, as it's name indicates, it inflates at the bottom of one's foot. It is an adjustable comfort insole and uses a miniature air-pump and release valve allowing users to adjust the pressure manually. A remarkable quality of InflaSole is its capacity to absorb shock and reduce skeletal tension. This is why it is marketed to athletes, as well as heel spur sufferers.

Fig. 1: InflaSole advertisement poster

For only 200$, you can have your very own InflaSole. This large price is a major drawback. Another drawback is it's need for manual actuation. The entire mechanism is inflated using one pump, meaning that the inflated region cannot be controlled. Furthermore, because of the use of this singular miniature pump, there is a limit to the inflation pressure, as well as volume. Another major drawback, is that it is always "on". Once inflated, the pressure is always on. The major goal of the sole is comfort and weight support. Our goal, using automation, is to make a sole that is comfortable, but also capable of massaging. It is more flexible, and has far more improvement prospects.

Other from InflaSole, there are no remarkable state of the art products resembling ours. The main difference is that InflaSole is not automated, always on, and more geared towards comfort than towards "massage".

A. Overview

Here is an overview of the whole project:

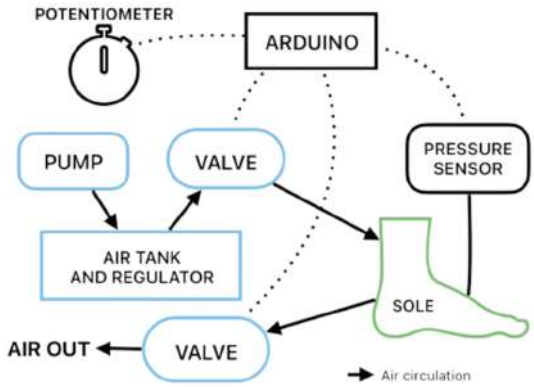


Fig. 2: Prototype Overview

Our system is mainly divided into three parts:

1) The sole, the design choices of the sole, and its implementation.
2) The pneumatic system: how the air is channeled from the pump to the sole.
3) The electronic system as well as the control implementation to follow the user-chosen desired pressure.

III. Precise Specifications

A. Sensors and actuators

1) Pressure sensor:

To implement a closed-loop control system, we require the knowledge of the current pressure inside the sole at any given point in time. A possible solution is the use of a pressure sensor. The model we use is the PG1005A5 by Honeywell. This sensor communicates with its master, the Arduino Uno presented in class, using I2C communication bus. The conversion from raw data to readable data is done using a pressure conversion function:

$$Pressure = \frac{(Out - Out_{min}) * (Pr_{max} - Pr_{min})}{Out_{max} - Out_{min}} + Pr_{min} \quad (1)$$

Where:

- Out : Output
- $Out_{min} = 1638$: Minimal Output
- $Out_{max} = 14745$: Maximal Output
- $Pr_{min} = 0$: Minimal Pressure
- $Pr_{max} = 100$: Maximal Pressure

The output is in psi, it must therefore be multiplied by 0.0689475729 to be converted into Bars.

2) Solenoids:
For pressure control, we use two solenoids, one between the pressure vessel and the sole, and a second one to vent air from the sole. The system is designed to use two VDW10AZ1D solenoid valves, but due to availability issues, we were only able to get one. The second solenoid used, is a 3-way solenoid which is also designed to work at $12[V]$, just like the valve we designed the system for. The valves are designed to work at a pressure differential of up to $9bar$, and work in a temperature range between $-10[°C]$ and $50[°C]$

3) Pump:
For our compressor, we use a D2028 Membrane Pump (Fig. 25, Appendix). It is fairly lightweight, supplies sufficient air flow to keep our pressure vessel pressurised, and only consumes 12[W] of power at nominal conditions.

IV. Sole design and implementations

A. Material choice

To implement the air patches, we used the material "TPU coated Nylon - 40 denier nylon". This thermoplastic polymer is airtight, robust, flexible and waterproof. Furthermore, it's heat-weldable and -sealable properties give us the advantage of flexible design and implementation. When heated to approximately $200[C]$, thermoplastic polyurethane (TPU) bonds itself. it also has great properties such as strong lamination, "low" lamination temperature, unstretchable, very good bending stiffness. We performed a computation of the bending stiffness, detailled in the following section.

B. Properties of the double-coated TPU layered fabric

The material used for the patches is a thermoplastic composite. We therefore tested it's bending stiffness in three different directions, depending on the direction of the fibers: parallel, orthogonal, and diagonal. To do this, we cut three samples, as shown in Fig. 3 below. We then applied the following equation:

$$\frac{B}{w} = \frac{{l^\star}^3 \cdot \rho \cdot h \cdot w \cdot g}{13.66 \cdot w} \tag{2}$$

where ${l^\star}^3$ is the length of the material at the point where it touches the surface again due to buckling (see Fig. 4), ρ is the material density, h is the material thickness, w is the sample width, and g is the force of gravity.

Fig. 3: Three strips in three different fiber directions

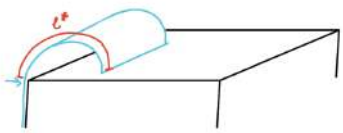


Fig. 4: Buckling length

The results for the three buckling lengths were almost identical : 55[mm], 56[mm], 57[mm] respectively. We therefore considered them all identical : $l^\star = 56[mm]$. The density was calculated by weighing several layers of the fabric and dividing by the volume, giving a density of $\rho = 177.347[\frac{kg}{m^3}]$. For the sample thickness, we use the measured thickness of $h = 0.204[mm]$. Doing a numerical application, we get:

$$\frac{B}{w} = \frac{(56 \cdot 10^{-3})^3 \cdot 177,347 \cdot 2.08 \cdot 10^{-4} \cdot 5 \cdot 10^{-2} \cdot 9.81}{13.66 \cdot 5 \cdot 10^{-2}} = 4.65 * 10^{-3}[\frac{kg*m^2}{s^2}] \tag{3}$$

We can then use the following equation to approximate the Young's modulus of our material. We do not have the poisson's ratio, but we can estimate it as being low, given the deformation of the material in the perpendicular directions of the loading is very low as the behaviour of TPU is strongly nonlinear and our material is a composit from TPU and Nylon. We therefore define a range, which will in turn allow us to define upper and lower bounds for the Young's Modulus. For the lower bound of our poisson's ratio, we take the poisson's ratio of Nylon $\nu_{nylon} = 0.39$, and for the upper bound the poisson's ratio of TPU at small deformation after a number of loading cycles, which is $\nu_{TPU} = 0.48$. We then use the following equation, relating the bending stiffness with poisson's ration and the Young Modulus:

$$B = \frac{E \cdot w \cdot h^3}{12 \cdot (1 - v^2)} \tag{4}$$

giving us a Young's modulus of

$$E = \frac{B}{w} \cdot \frac{12(1 - \nu^2)}{h^3}. \tag{5}$$

This gives us the range of Young's modulus we were looking for: $E_{min} = 4.77[GPa]$, and $E_{max} = 5.26[GPa]$.
As our material is very thin, we can consider it as a plate or shell structure where we can assume that all in-plane

deformation of the material will be dominated by out-of-plane deformation, i.e. bending. This assumption significantly simplifies simulation and analysis of the behaviour of the material.

C. Experimentation regarding the implementation of the sole

To weld the TPU coated fabric, we initially used a welding system. We created small patches of various shapes and inflated them in the goal of finding an ideal format (see Fig. 5 - 8).

Fig. 5: Single inflated patch using welding method

Once the possibility of inflating the patches with air was established, progression to bigger designs was possible. During this step, we replaced the welding machine with a laundry iron. Various sole designs were created on CATIA and cut into baking paper using a Computerized Numerical Control (CNC) laser-cutter. This approach allows for high precision and very quick fabrication. The cut-out baking paper sheets were then placed between two layers of TPU coated fabric, and the three-layered design was then ironed to bond the TPU (at $160[C]$). For the final design we used a hot press, that went up to $200[C]$, to unsure that the TPU was well sealed together to avoid any leakage or delamination.

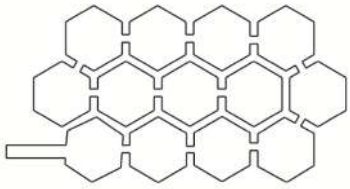


Fig. 6: Laser-cut shapes as honeycomb

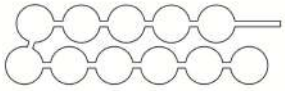


Fig. 7: Laser-cut shapes as circles

Fig. 8: Laser-cut shapes as triangle

The three designs tried were circles, triangles, rectangles and honeycombs. Firstly, the circles were not a good shape, because they were less resistant to delamination, meaning the patches area between the patches got unstuck leading to one large inflated volume, which could have been caused by insufficient welding temperature or pressure of the TPU. Furthermore, the area of inflated volume relative to the total area is fairly small. Then we tried triangles but decided to not go further with this implementation because of the big angle of bending from the design when inflated. We also noticed that with this design, the material would retract significantly more than with other designs. The sole was very bumpy, we were afraid that the massage feeling would not be optimum. The best fit seemed the honeycomb, the surface area touching the foot was important, the sole itself was not bending that much and it had a good resistance, no leakages, and the patches did not delaminate.

D. Final design of the sole

We decided to go with a general shape with different pattern sizes. The different sizes were implemented to better fit the foot. We also wanted to have more surface area on the sensitive parts of the foot. The idea was to put bigger patches on the most sensitive areas of the foot, to allow for a better contact surface and thus a heightened massage sensation. These areas are considered to be the inner part of your foot. The least sensitive ones are the sole and the heel. That is why in our final design, the heel and sole have smaller patches. We also decided to round up the edges of the honeycomb patches to avoid stress concentrations at these points, thus avoiding points of leakages or points were the honeycomb design could delaminate.

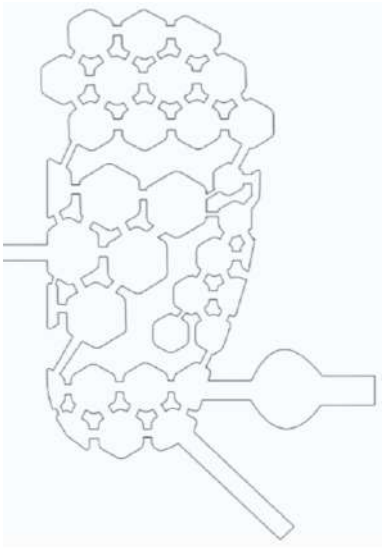


Fig. 9: Final 2D design of the sole

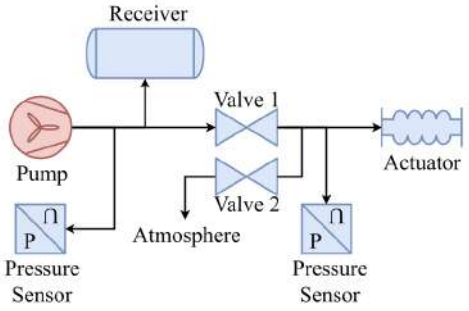
Fig. 10: Picture of the final sole

V. Pneumatic system

A. Inspiration

To implement our pneumatic system, we drew significant inspiration from the methods described in the paper [2].

In the latter, the configuration from Fig. 11 is described, which, apart of notation conventions, is almost identical to our setup.



(b) Pneumatic system with dual 2-way valves

Fig. 11: Pneumatic system with dual 2-way valves [2]

B. Pump

As specified in section V-B, the pump used in the design is a D2028 membrane pump. It supplies air at a nominal pressure of $2.2[bar]$ to our system. As the working pressure of our sole is below $1[bar]$ gauge pressure, this is largely sufficient for the application.

C. Pneumatic system

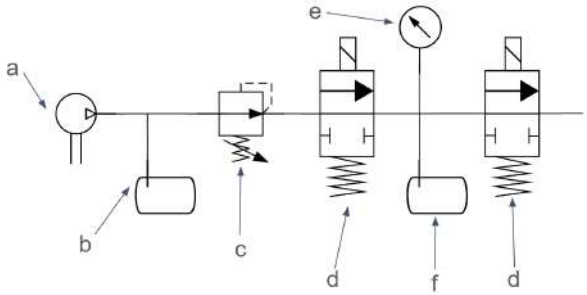


Fig. 12: Diagram representing the pressurised system. a: compressor, b: pressure vessel, c: pressure regulator, d: solenoid, e: pressure sensor, f: sole

The pneumatic system is shown in Fig.12. The pump supplies air compressed to $2.2[bar]$, which puts a pressure vessel under pressure. Said pressure vessel is a soda stream bottle, designed to withstand $8[bar]$ nominal pressure. Originally it was decided to be used to accomodate for the use of an electric bicycle pump which turned itself off once it reached a given pressure, but we decided to use it in our final system to be able to have a high peak airflow, and to furthermore be able to have a more constant entry pressure at the solenoids. Lastly, unexpected benefit of it is that it allows us to use the device with the compressor being turned off for a short time. We then downregulate the pressure to $1[bar]$ using a pressure regulator, with the goal of slowing the filling speed of the sole, thus improving the precision of the control, and generating a constant entry pressure for the feeding solenoid of the sole. Next in the system, we have a solenoid valve, responsible for increasing the pressure in the sole by PDM control as discussed in section VII-A. We use a pressure sensor to measure the pressure inside the sole in real time, we then use a second solenoid valve to release pressure from the sole. Finally, after the second valve, we reduce the exhaust speed by using a neck. This improves the control, and reduces the potential of overshoot. Further details on why this is needed are given in VII-A.

VI. Electrical Circuit

A. Circuit

The following lists the components our circuit includes: (all of them can be found on Fig. 13, except the push button)

- $12[V]$ battery
- Arduino Uno
- $12[V]$ 2/1 solenoid valve IN (two ports, one way)
- $12[V]$ 3/2 solenoid valve OUT (three ports, two ways)
- Pressure sensor
- Potentiometer
- Push button
- Two flyback diodes serving to protect against back electromotive force (EMF) generated when the solenoids deactivate and the ferromagnetic shaft returns in place pushed by the spring.
- Two N-MOSFETs are employed to drive the solenoids efficiently, operated on the $5[V]$ of the arduino's pins.
- Three $10[k\Omega]$ resistors are utilized for both the N-MOSFETs, the potentiometer and the purge push button.

Pin connections are established as follows:

- The pressure sensor communicates with the Arduino through an I2C communication bus. It is therefore connected to the SCL, and SDA pins.
- The potentiometer is connected via the analog pin A0.
- The solenoids are connected via digital pins to pins 3 (IN) and 4 (OUT).

The entire circuit can be seen further on in Fig 13.

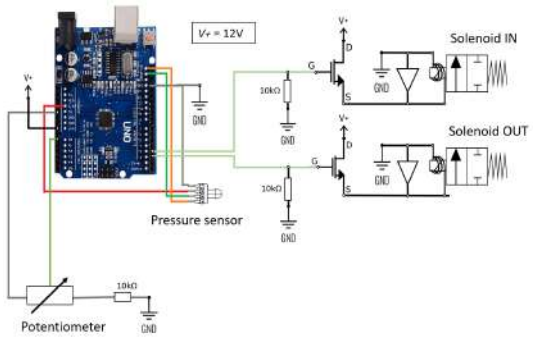


Fig. 13: Electrical circuit

B. Implementation

The prototyping of our circuit was tested on a breadboard. However, as we wanted our system to fit inside a compact box, we needed to make the circuit as compact as possible. To do so, the circuit was welded onto a solerable breaboard, as shown below (Fig.14). This allowed us to significantly reduce the size of our circuit. Furthermore it ensures electrical contact, since the Dupont wires did frequently disconnect from the breadboard.

Fig. 14: Compact Electrical Circuit

VII. Control theory

A. Inspiration and main idea for the control

The pressure inside our sole needs to react in real time to the real world disturbances, the weight applied by the user, the leakages, and the changes in desired pressure, given through the potentiometer.

To change the pressure inside the patches of our sole, we use two solenoids: one input AIR IN solenoid, and one output AIR OUT solenoid. The input solenoid is connected, with the intermediary of a pressure regulator and a bottle acting as an air reservoir, to the pump.

To implement our control, we drew significant inspiration from the methods described in the paper: Xavier, M., Fleming, A., & Yong, Y. (2021). Design and Control of Pneumatic Systems for Soft Robotics: A Simulation Approach [2]. The paper details the use of a low-frequency Pulse Width Modulation (PWM) signal to regulate the opening time of both valves.

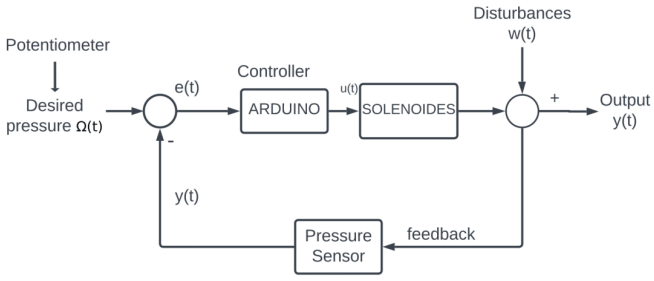


Fig. 15: Diagram of the control system

B. Our implementation

As mentioned before, our main objective is to maintain the pressure inside our inflatable sole within a small range of the desired pressure. For this, the system employs a pressure sensor to provide real-time feedback on the current pressure inside the sole, and an Arduino-based controller, responsible for managing the actuation of the two solenoids. The control algorithm implements a PWM on the opening time of the valves and is designed to adjust the pressure based on the error:

$$e(t) = \text{desired\_pressure}(t) - \text{measured\_pressure}(t) \quad (6)$$

When the error is positive, indicating that the current pressure is below the desired level, the system responds by increasing the pressure. In this goal, the algorithm opens the AIR IN solenoid by a calculated value t_{sol} and closes the AIR OUT solenoid.

On the other, if the error is negative, suggesting that the current pressure exceeds the desired level, the system works to decrease the pressure. It does so by closing the AIR IN solenoid and opening the AIR OUT solenoid by t_{sol}. In situations where the error is very close to zero and within an acceptable threshold, t_{sol} will follow a bang-bang control, as detailed in section VII-B3.

1) Pulse-Width Modulation (PWM):

As previously mentioned, we use low-frequency PWM to regulate the opening and closing duty cycles of the valves. This involves dividing our time into constant periods T. Each period T is further subdivided into n subperiods. In our implementation, we have 5 subperiods, each lasting $15[ms]$. Consequently, the total cycle length T is computed as $5 \times 15 = 75[ms]$. This is also the maximum reaction time of our system to a change in input.

At the onset of each cycle, the error is recalculated, and the duty cycles are computed utilizing a proportional controller. If the duty cycle exceeds zero, the corresponding solenoid is activated. Notably, only one of the two solenoids is opened during each cycle. This means that we only need a single control variable, denoted as t_{sol}. The latter is an integer that represents the subperiod number at which the necessary solenoid closes, in the event that it has been opened.

2) Controller:

As mentioned, the control variable t_{sol} is determined using proportional control, with its equation given by:

$$u(t) = K_p \cdot e(t) \quad (7)$$

In our specific implementation, we utilize two P-control versions based on the sign of the error:

$$t_{\text{sol}} = P_{\text{in}} \cdot e(t) \quad (8)$$

$$t_{\text{sol}} = P_{\text{out}} \cdot e(t) \quad (9)$$

The values of P_{in} and P_{out} are determined through a combination of trial and error and by referencing the Simulink simulation. The two P factors are different given the different behaviour of the two different valves.

Combining simulation and experimentation, we find that $P_{\text{in}} = 15[\text{bar}^{-1}]$ and $P_{\text{out}} = 20[\text{bar}^{-1}]$.

It is notable to mention that, initially, a 10-moving average on the pressure sensor's values was implemented in the pressure reading. This was done based on the method described in [2]. The goal of this approach was to increase the accuracy of our measurements by reducing the impact of potential outliers. In the end, it was observed that the pressure sensor's readings were accurate, thus making the moving average unnecessary.

Moreover, we opted to exclude the Integral (I) component from the potential PI-controller. This decision was made based on the understanding that the primary function of the integral factor is to remove any offset. Given that the user sets the desired pressure using a potentiometer relative to the pressure sensed at the time, the presence of an offset would go unnoticed by this user. Consequently, the integral factor is also unnecessary.

Finally, we also opted to not include a derivative factor. This decision was mainly based on the transient behaviour observed in the simulation (Appendix, Fig .29). Furthermore, an oscillatory behaviour on the pressure reading was observed [section X], this also prevented us from implement a derivative factor.

3) Bang-Bang control:

As previously detailed, the integral factor was neglected from our controller due to the relative and subjective nature of the offset in pressure in our specific application. This exclusion applies universally, with one exception: when targeting a desired pressure of zero. In this case, the user will notice the offset. But where does this offset come from? In our system, an observed "artificial" error threshold emerged, resulting from the discretization of the opening pulse width duty cycle of the valves. In fact, the use of integers to represent the latter results in a rounded value: this leads to a behaviour where no correction is applied on the pressure if the error does not exceed this artificial threshold. To fix this issue in the case of a 0-reference pressure, we implemented bang-bang control with a threshold set at 0.05 [bar].

In scenarios where the desired pressure is zero and the measured pressure exceeds the threshold (STATE ON), the AIR IN solenoid will close, and the AIR OUT solenoid will open, forcibly evacuating the pressure within our system. The inclusion of a threshold is also helps to prevent the solenoid from remaining constantly activated, which could potentially lead to overheating, overuse of energy and potential damage. The state conditions can be expressed as follows [2]:

$$\text{State ON:} \quad P_{desired}(t) < P_{measured}(t) - h \Rightarrow e(t) > h \tag{10}$$

$$\text{State OFF:} \quad P_{desired}(t) > P_{measured}(t) + h \Rightarrow e(t) < -h \tag{11}$$

The implementation of this control is done by assigning the $n = 5$ cycle-length value to our control variable t_{sol}, which will act upon our AIR OUT solenoid.

4) Pseudocode: Having detailed our method, here is the complete pseudocode of our control:

Algorithm 1 Pressure Control Algorithm

```
while true do
P_mes ← PRESSURE SENSOR VALUE
  if |counter| > (nb_cycle − 1) then counter ← 0
  end if
  if counter = 0 then
      sol_in ← OFF
      sol_out ← OFF
      DesiredPressure ← f(POTENTIOMETER)
      error ← desiredPressure − P_mes
       if error > 0 then
          tsol ← factor_P_in × error
          tsol ← constrain(tsol,0,Tsol_max)
              if tsol ≠ 0 then
                  sol_in ← ON
              end if
        else
          tsol ← factor_P_out × error
          tsol ← constrain(tsol,0,Tsol_max)
          if (desiredPressure = 0) and (P_mes > 0.05) then
              tsol ← nb_cycle
          end if
          if (desiredPressure = 0) and (P_mes < 0.05) then
              tsol ← 0
          end if
          if tsol ≠ 0 then
              sol_out ← ON
          end if
        end if
    end if
    if counter ≥ tsol then
      sol_in ← OFF
      sol_out ← OFF
    end if
counter ← counter + 1
delay(timedelay)
end while
```

The complete and detailed code can be seen in Fig.30

VIII. Outside box and final user friendly implementation

In order for our sole to be used by everyone, we decided to create a box that could be attached to a belt. As the box is a little heavy, this would allow the user to clip it onto a belt that they are already wearing, or even to their pants (which have to be fairly well fitted), and to continue their normal daily tasks. For this purpose, we designed a box using fusion, and fabricated it using Fused Deposition Modeling (FDM) additive manufacturing. The material used is PolyEthylene Terephthalate Tlycol (PETG). The box is designed to optimally fit all the electrical components and most of the pneumatic components. The lid can be screwed shut and unscrewed for maintenance purposes but the whole container is designed to

not be opened regularly. Everything that the user may need to access regularly can be accessed from the outside.

The box itself has a hole to mount the reservoir, another one to allow the mounting of the rechargeable battery, and a further one to plug and unplug the tube that will supply the sole with air, see Fig;16. When worn on a belt, the tube is long enough to reach the user's foot while they are standing upright.

Fig. 16: 3D sketch of the final box

During the manufacturing process, we redesigned the sole slightly by removing two of the access points for the air tubes. Initially we were planning to use three access points to a sole, i.e. one for "air in", one for "air out", and one for pressure sensing, as can be seen in Fig. 17. This decision was taken, as late during fabrication as it was noticed that the main source of leaks in the system was where we connected the pouch to the TPU tubing. It also reduces the amount of tubing needed, and allows for the sole to be plugged in and unplugged with just one single tube. On the other hand this resulted in minor oscillations in pressure sensing (seen in the section X), most likely due to inertial or venturi effects within some of the 3-way joints used in the system. Said oscillations are very high frequency relative to our control system and decay fast, which makes it so that the system does not react to them in any significant way.

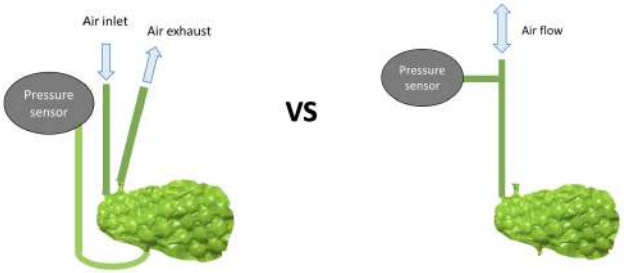


Fig. 17: Minors changes made to make the whole system more user friendly

On the outside of the box, we have:

- On and Off button. By switching this button, the user can simply turn on and off the pump and fill the reservoir and the sole with air.
- Battery box, where the user can just place the battery and the system would be supply in energy. Putting it outside allows the user to recharge the battery by removing it from its case without opening the box.
- Potentiometer to change the pressure. The user can simply twist the button to match its desired pressure
- Purge button. If the user decides that he wants to purge the reservoir, once can simply unplug the sole for the cable hole and press this button, releasing the air.

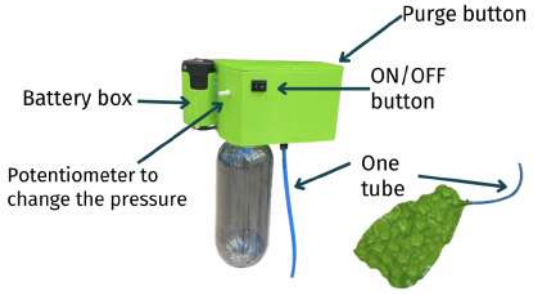


Fig. 18: Overall picture of our system

Finally, different sizes of soles can be placed into our foot massager. They can be easily customizable to fit anyone's foot, or even different parts of the body.

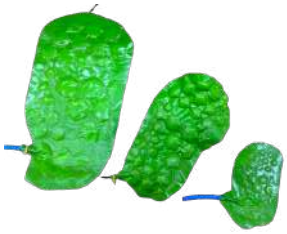


Fig. 19: Different sizes for the sole

IX. Detailed simulation of the final solution

Modeling the system in detail can bring us a better understanding of the behaviour of the system for various conditions. The main goal for us was to study the stability of our controller, and to guess the starting controller variables. These variables were then implemented and tuned in the real system. The simulation also allowed us to save some time, and, to some extent, test some experiments on the system while keeping the real one safe. Additionally, we wanted to estimate air loss in the system, and optimize our system to have a good control while saving the tank air. The simulation is made up of several sub-systems. The main ones are: (on Fig.20)

- The physical pneumatic system (Appendix, Fig. 26) IX-A1
- The control loop (Appendix, Fig. 27) IX-A3
- The pressure handling by the Arduino (Appendix, Fig. 28) IX-A2

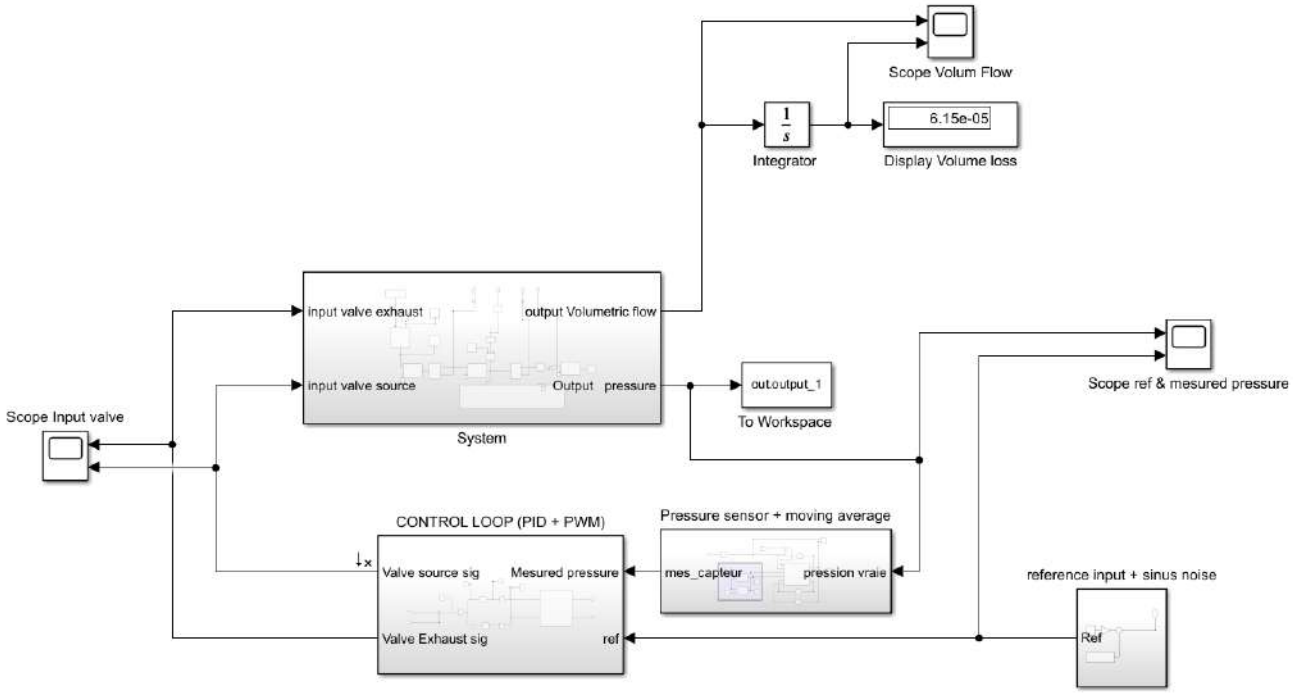


Fig. 20: Simulation: diagram of the whole system

A. Parameters of the simulation

Set of variables and their values:

- $Tpwm = 0.4[s]$; is the total length of a PWM (Pulse Width Modulation) cycle, corresponding to a 100% duty cycle.
- $Ttpwm = 0.015[s]$; is the minimum temporal resolution of the duty cycle. We estimated $15[ms]$ to be a sufficient time for an opening and closing cycle for the solenoids, by empiric testing.
- $Pressure\_tank = 100000[Pa]$; is the pressure of the reservoir before entering the system, output of regulator (modeled as being a constant pressure source).
- $Factor\_P\_In = 0.00002[Pa^{-1}]$; is the proportional factor for the source valve.
- $Factor\_P\_Out = 0.00002[Pa^{-1}]$; is the proportional factor for the exhaust valve.
- $threshold = 0[Pa]$; bandwidth of the deadzone for the error before entering the control loop, as explained in section VII.VII-B3 we set it to 0.

1) Physical system simulation: (Appendix, Fig. 26)
For the physical simulation of the pneumatic system, our work is based onM. Xavier and al's work [2]. As described in the paper, we decided to use the library Simscape Fluids to simulate the physical components: the sole (constant volume chamber), solenoid valves (on/off solenoids, with leakage fraction of 10^{-6}), the reservoir with its regulator (constant pressure input), tubes (as pipes), and necks (pipes with a small orifice). Two measuring elements are added: the pressure sensor and a volumetric flow rate sensor. Additional elements such as references (atmosphere and gas properties) and thermal insulators are added (we estimated thermal effects negligible in the real system compared to other simulated effects).

2) Pressure sensing modeling: (Appendix, Fig. 28)
The pressure sensor value is obtained in the physical system. We add a transportation delay and some random noise. To simulate the arduino, the value obtained is sampled each T_{tpwm}, and averaged on 10 values each T_{pwm}. This is the value given to the entrance of the controller part, at the beginning of each cycle.

3) Control loop simulation: (Appendix, Fig. 27)
At the start of each PWM cycle, the obtained pressure value is then subtracted to the target reference. The error is then multiplied by a proportional factor (different corrections are proceeded on the two solenoids, as their behaviour is very different). We then convert the correction into a time of opening for the valves. The resolution of this opening time is limited by T_{tpwm} as in the real system (see section VII, 3)).

B. Results, discussion and decision taking

The behaviour of the system has been studied for a large range of conditions (Appendix, Fig.29). The controller subsystem manages to reach the reference pressure with a proportional control. Due to the relatively small size of the air chamber ($0.2[l]$), and to the nature of the solenoid's binary behaviour, the pressure change in the chamber is very quick: the transient regime duration is smaller than a full cycle of the solenoids ($15[ms]$): it is unnecessary to try to do control on the transient regime.

The conclusion we took from the simulation are the following:

- The perception of the error is not very much impacted by a dead-zone. In reality, there is a physical limitation of the solenoid opening time: the minimum pulse width is set to be 0.015 seconds (We estimated this time to be sufficient for the solenoid pneumatic valve to make one open/close cycle). We learnt through the simulation that this minimum resolution actually limits the system for small variations: if the error is small enough, then the conversion in a PWM input has a 0% duty cycle. In other terms, for a small pressure offset (inferior to an order of magnitude of 0.2 [bar]), the system does not correct the pressure of the sole.
- A proportional control seems to be sufficient for correcting the pressure in the sole. As we want to optimize the air loss, tracking the system closely implies a larger loss of air (can go up to 10 times more) as we open and close the valves more frequently. This is especially true for high frequency perturbations of the reference pressure.
- Despite the previous point, we see that we have a non-zero offset on the negative step response (Appendix, Fig. 29.b). We still need to put either an integrator term in the controller, or to fix a threshold pressure under which we use a bang-bang control.

In conclusion, with the chosen parameters, the designed control seems to be robust for a vast array of application cases. (IX-A).

Nevertheless we have to keep in mind that the simulation is not a replica of reality, and each element of the simulation is based off of assumptions on the physical system. For example, we did not model the thermal exchange of the system with its environment, assuming this effect was negligible. The limitation of the simulation will be discussed below.

Note that for the implementation of the system, the variables in the code will have a factor of 10^5 as the simulation's pressure is given in pascals, and in our system we read the pressure sensor's values in bars.

X. Performance

A. Simulation vs Reality

To evaluate the performance of our system, we decided to test it with different typical responses: a step response (Fig.24), a negative step response (Fig.22), a stair reference response (Fig.23) and a complex signal. For the complex signal, we decided to implement a sinus function. The latter's reference follows an ascending ramp. For each response, we represented the reference pressure, as well as the measured pressure and the simulation results. Through these graphs, we can firstly conclude that the system is well represented by the simulation, thus for future developments we could use the same framework to anticipate the behaviour of the system. Secondly, we can observe some oscillations. These arose after we switched to the new architecture (17) explanations are given in section VIII.

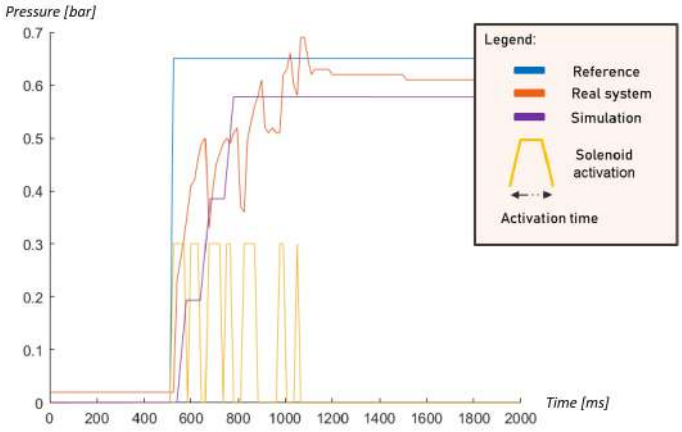


Fig. 21: Step response: a comparison between the system and the simulation

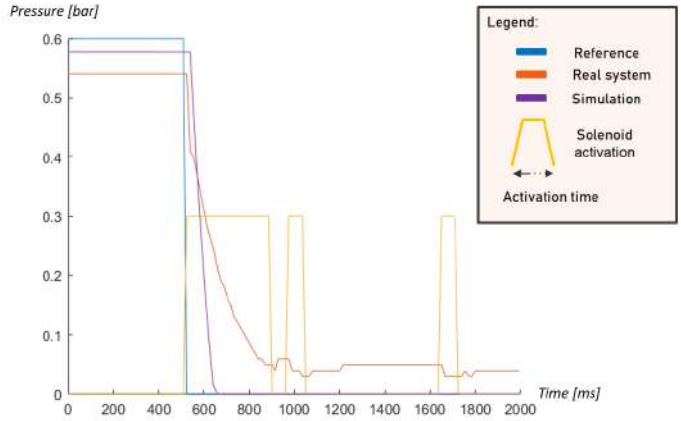


Fig. 22: Negative step response: a comparison between the system and the simulation

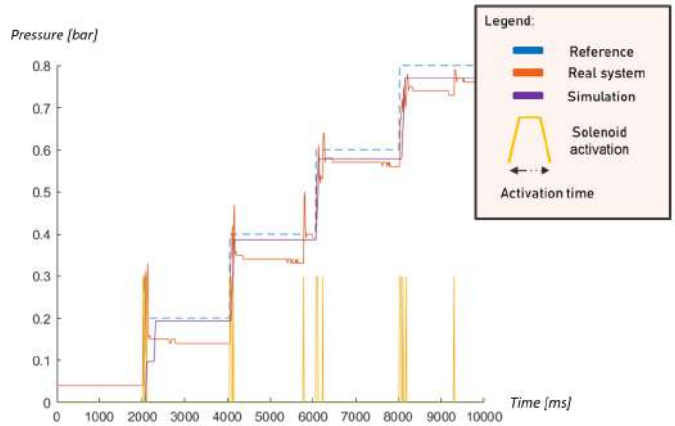


Fig. 23: Stair response: a comparison between the system and the simulation

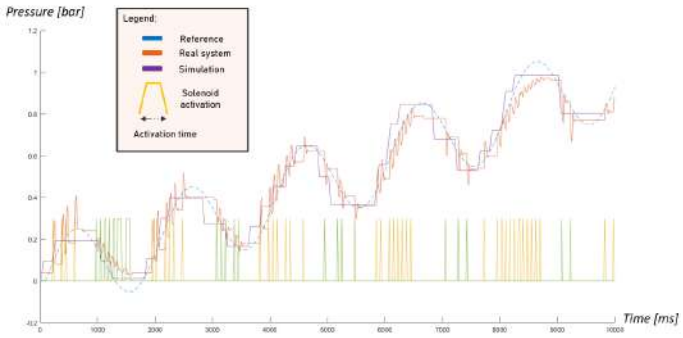


Fig. 24: Sinusoidal ramp response: a comparison between the system and the simulation

B. Further Metrics

To qualify the performance we want to quantify a number of factors. The idea is to define how we can meet the goal. The goal can be defined in a form of different points. We want to measure:

- If the sole can support a certain amount of weight
- If we feel enough contact surface
- If we have enough vertical deformation
- Finally we want to minimize the leakages
- If the sole inflates and deflates quickly

C. Supported weight and force

- Goal : m = 80kg (above average human weight)
  F = mg = 800 N
- Results : m = 75kg
  F = mg = 750 N
- Conclusion: This is a win! We are close to the expected measure - with one sole per foot it would lift the double.

D. Contact surface

- Goal : Big enough to feel it under the feet, but small enough to apply pressure points.
- Results : $A_{tot} = 5438[mm^2]$.
- Conclusion : Around a 1/4th of the foot's surface, the feeling is intense enough.

E. Vertical deformation

- Goal : Big enough to feel it under the feet
- Results : $Max = 14[mm]$
- Conclusion : A good result, as a 14 mm sole can easily fit in any type of shoes

F. Leakage

- Goal :Minimal
- Results : At constant Pmax, leakages after 3 seconds (solenoid in opens)
- Conclusion : Satisfying. To improve this we could localize where the leakages are.

G. Inflation / deflation reactivity

- Goal : settling time below 1 second
- Results : maximum settling time of $0,5[s]$ (Max reaction time: $75[ms]$) see Fig.24
- Conclusion : Very reactive, effective, and robust

XI. Future improvements

Our sole can be improved in numerous ways:

- Different sole sizes, as well as various sole patterns. These can easily be personalized to the specific needs and dimensions of the user.
- Different pouch designs for different parts of the body. The pouches could be tuned to specific applications and needs, such as medical conditions.
- Smaller and more compact overall system.
- Better battery for more autonomy.

It would also be pertinent to implement the existing control system in development at RRL to allow the user to draw a pressure cycle which the device automatically follow.

XII. Conclusion

In conclusion, we successfully designed and developed an closed-loop, wearable and actuated device. Our system is mainly comprised of a sole and box that contains the hardware. A potentiometer is employed to set the desired pressure. The use of a pressure sensor and a potentiometer give us real-time errors on the pressure. Our system then reacts using actuated solenoids, releasing and increasing pressure inside the sole, thus changing the felt pressure. Different designs and mechanisms were employed to optimally reach our goals.

We greatly appreciated the class, as it allowed us to broaden our knowledge through hands-on experience. This project especially enabled us to learn how to design a pneumatic system, control its pressure and fabricate a compliant pouch. The ressources at our disposal, the help and support of the assistants and teacher, as well as the regular attendance of class are all factors that allowed us to fully emerge ourselves into the project. Working as a team was important, and we learnt a lot about using our own strengths and weaknesses to complete each other.

XIII. Acknowledgement

We thank Prof. Jamie Paik for fighting for us to be able to complete this class with a budget that would enable us to build a well functioning device. We also thank all the assistants of the class ME-410, Alihan Bakir, Kevin Holdcroft, Hwayeong Jeong, Rohit Kadungamparambil John, Mustafa Mete, Alexander Schüssler, Ziqiao Wang and Fabio Zuliani. In particular, we would like to express our gratitude to Yuhao Jiang, who subbed Prof. Paik, and Serhat Demirtas who was working with us particularly closely. Furthermore we would like to thank the assistants from SPOT, who were always available to answer quick questions, in particular Simon Lütolf, Ivan Tomic, and Sébastien Martinerie. Lastly, we would like to thank Hugo Penichou, and the EPFL Carbon Team, who let us borrow their air compressor when there were issues with the delivery of the one we ordered ourselves.

REFERENCES

[1] *InflaSole*. © 2011 Air-Pump-Insoles, Inc. All Rights Reserved.. [Online]. Available: https://www.inflasole.com

[2] M. S. Xavier, A. J. Fleming and Y. K. Yong, "Design and Control of Pneumatic Systems for Soft Robotics: A Simulation Approach," in IEEE Robotics and Automation Letters, vol. 6, no. 3, pp. 5800-5807, July 2021, doi: 10.1109/LRA.2021.3086425.

XIV. APPENDIX

For better clarity we decided to put the big images in the appendix.

Fig. 25: Datasheet of the membrane pump used

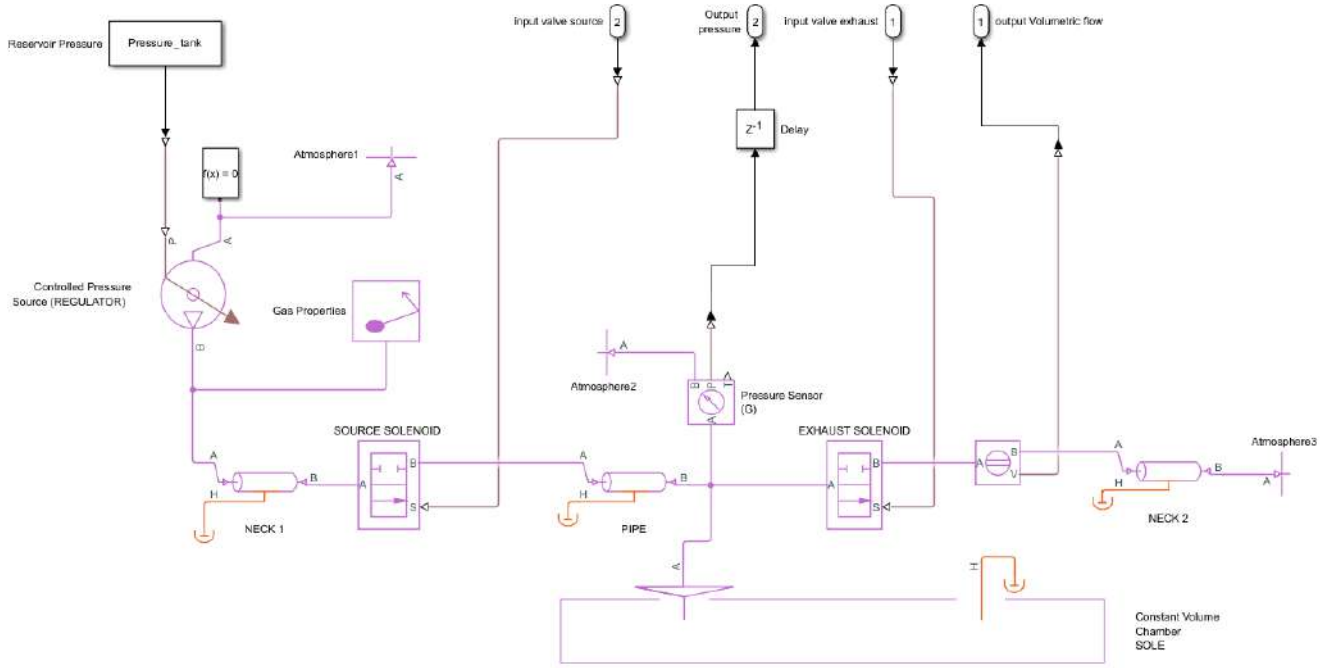


Fig. 26: Simulation: physical subsystem (Simscape)

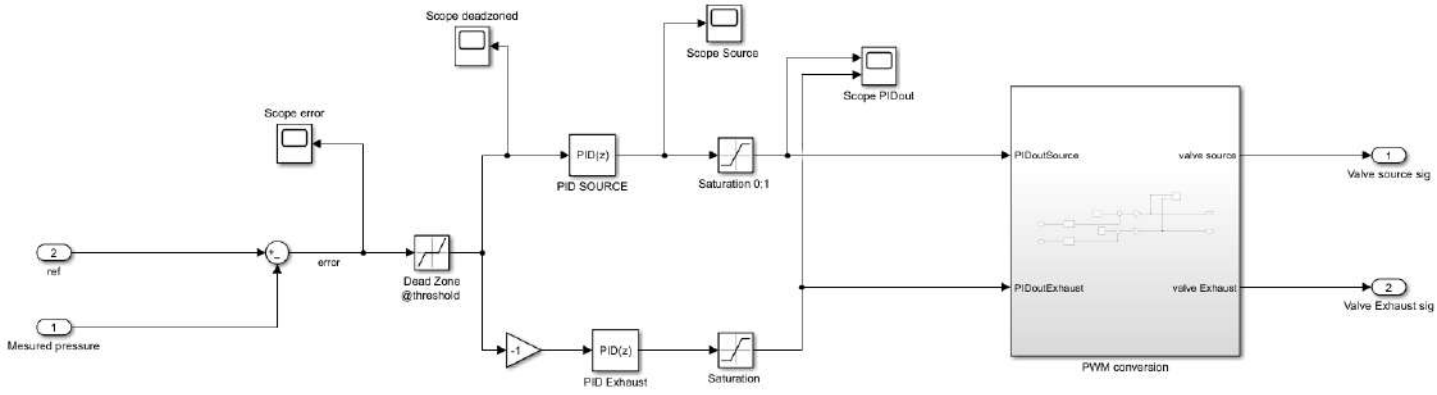


Fig. 27: Simulation: control subsystem (note that the proportional gain of the error is different for each solenoid)

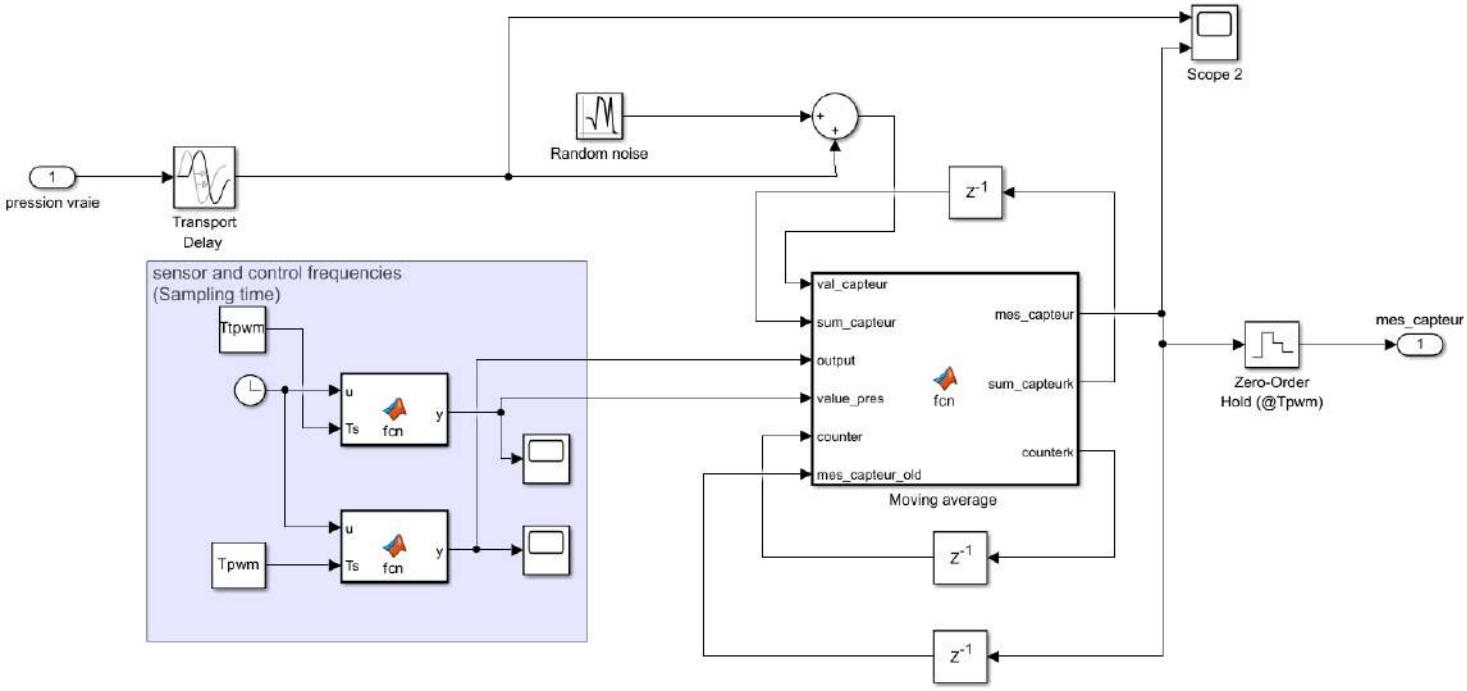


Fig. 28: Pressure sensor moving average

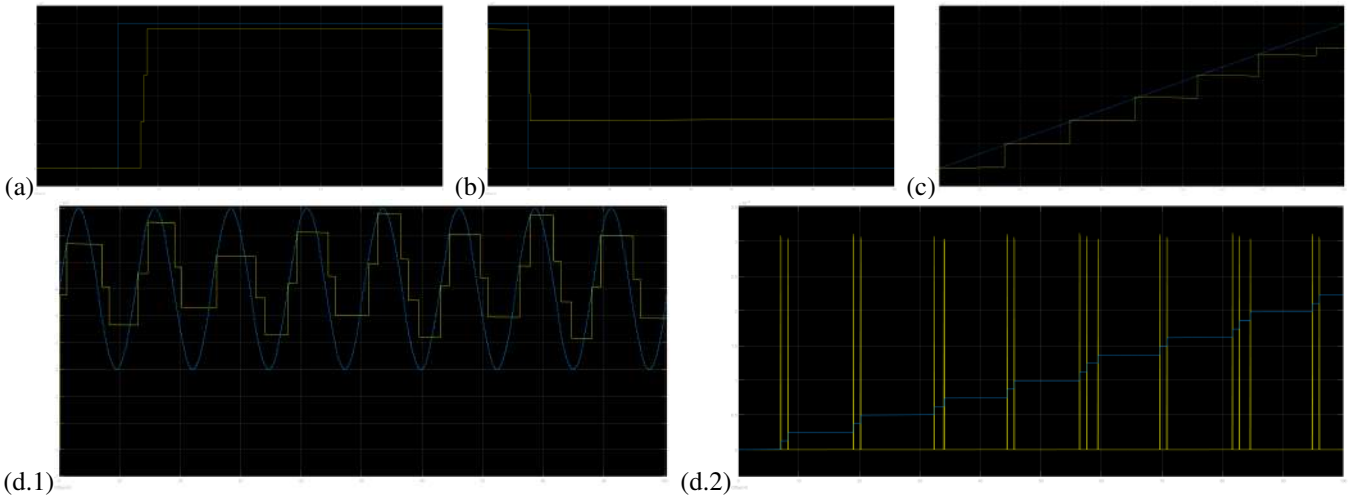


Fig. 29: (a) step response (b) step (c) ramp response (d.1) sinus reference (d.2) Volumetric flow loss for the profile (d.1)
For a,b,c,d.1:Blue=ref and yellow = measured pressure (by the arduino - but close to real
For d.2 : yellow = Volumetric air flow rate, blue = total air loss (by the arduino - but close to real))

```
#include <Wire.h>
const int PUSH = 2;const int SOLENOIDIN = 3;const int SOLENOIDOUT = 4;
const int POTENTIOMETERPIN = A0; ///////PIN CONSTANTS
float desiredPressure = 0;float scalingFactor = 0.00075;   //Potentiometre->desired pressure //0.0039 ////////////////////////////
float pressure_min = 0;////////////////////////////////////////////////////////////////Definir un pressure_max et Pressure_min
float pressure_max = 10;
int counter = 0; /////TIME MANAGEMENT FOR REGULAR INTERVALS
int timedelay = 15; ////Time Resolution
int Tsol_max = 4;
int state = 0;
float P_mes = 0;   //PID CONTROL VARIABLES
float error_mean = 0;       //error
float factor_P_in = 15;   //Solenoid IN P factors
float factor_P_out = 20; //Solenoid OUT P factor
int tsol = 0;/////variable duration opening valve, in OR out
const int nb_cycle = 5;
//plotting variables
bool sol_in = 0;
bool sol_out = 0;
void setup() {////////////////////////////SETUP////////////////////////////
  Serial.begin(115200);
  Wire.begin();//Pressure sensor communication setup
  pinMode(POTENTIOMETERPIN, INPUT);  //Potentiometer
  pinMode(PUSH
  , INPUT);
  pinMode(SOLENOIDIN, OUTPUT);   pinMode(SOLENOIDOUT, OUTPUT);//Solenoid
}
void loop() { ////////////////////////////////////////////////START OF GLOBAL LOOP/////////
  P_mes = readPressureValue();//}  //Array of measured error values
  if (abs(counter) > nb_cycle-1) { counter = 0;} ///reset cpt    MAX(cpt) * DELAY = T (period for 1 cycle)
  if (counter == 0) {////////////////////////////////error control loop (only at the begining of a cycle)
////////////////PURGE
if (digitalRead(PUSH)==LOW){
  while(digitalRead(PUSH)==LOW){ digitalWrite(SOLENOIDIN, HIGH);digitalWrite(SOLENOIDOUT, HIGH);sol_in = 1; sol_out = 1;}
  digitalWrite(SOLENOIDIN, LOW);digitalWrite(SOLENOIDOUT, LOW);sol_in = 0; sol_out = 0;
  }
    error_mean = 0;
    desiredPressure = scalingFactor*analogRead(POTENTIOMETERPIN); ///aquisition potentiomentre
    //desiredPressure = constrain(desiredPressure, pressure_min, pressure_max);
    error_mean = desiredPressure-P_mes;
      if (error_mean > 0) { // need to add pressure inside patches
        tsol = factor_P_in*error_mean;///////PID FUNCTION for in
        tsol = abs(tsol); tsol = constrain(tsol,0,Tsol_max);
        if ((desiredPressure <= 0) && (P_mes > 0.05)) {tsol = 0;}
        if ((desiredPressure <= 0) && (P_mes < 0.05)) {tsol = 0;}
        if (tsol != 0) {digitalWrite(SOLENOIDIN, HIGH); sol_in = 1;}
      } else {          // need to lower pressure if P_mean<0 (measurement is smaller than the reference -> increase the pressure )
        tsol = factor_P_out*error_mean;///////PID FUNCTION for out
        tsol = abs(tsol); tsol = constrain(tsol,0,Tsol_max);
        if ((desiredPressure <= 0) && (P_mes > 0.05)) {tsol = nb_cycle;}
        if ((desiredPressure <= 0) && (P_mes < 0.05)) {tsol = 0;}
        if (tsol != 0){digitalWrite(SOLENOIDOUT, HIGH); sol_out = 1;
        }
      }
  }
  if (counter >= tsol) {digitalWrite(SOLENOIDIN, LOW);digitalWrite(SOLENOIDOUT, LOW);sol_in = 0; sol_out = 0;}
//////////////////////////END OF error control loop//////////////////////////
  counter++;
  delay(timedelay); //time resolution of the duty cycle
}
///////////END OF GLOBAL LOOP/////////


//////////////////////////PRESSURE READING FUNCTION//////////////////////////
float readPressureValue() {
  Wire.beginTransmission(88);  // PG1005A5 address (I2C address of the target)
  Wire.requestFrom(88, 4); //the number of bytes you want to request from the device.
  byte s_DAT1 = Wire.read(); //reads the value
  byte s_DAT2 = Wire.read();
  int word = (s_DAT1 << 8) + s_DAT2; //combines s_DAT1 and s_DAT2 -> word = output
  //float pressure = float(word-1638)/1901.0096293; ////////////////////////////////////////////////////////////////////////////////
  float pressure = ((float(word-1638)*100)/(14745-1638))*0.0689475729; //1 psi = 0.0689476 bar

  return pressure;
}
```

Fig. 30: Code uploaded in the Arduino UNO (credit for the pressure reading function: Serhat Demirtas)