POLITECNICO DI TORINO

Master's Degree in
Mechatronic Engineering
(Ingegneria Meccatronica)

Master's Degree Thesis

Implementing the control strategy of an industrial exoskeleton and testbench simulation

Advisor:
Prof. Luigi Mazza

Co-Advisors:
Prof. Gabriella Eula
Prof. Terenziano Raparelli
Dr. Giuseppe Pepe
Eng. Francesco Pietrafesa

Candidate:
Ahmed Hassan
Aboubakr Shaaban Ali

Academic Year 2020/2021

TABLE OF CONTENT

TABLE OF FIGURES

TABLE OF TABLES

Abstract

The ergonomics study is dominant in every workshop such that due to fatigue loads that the workers undertake for prolonged periods of work which can cause several human joint and bone diseases and malfunctions and these problems does not appear after short notice, however, after long periods, a few people experience illness caused by labor work. Exoskeletons are devices used to enhance the ergonomics studies and to ensure better experiences for labor workers along several other applications as well. This thesis study oriented to the design and development of an industrial-type pneumatically powered exoskeleton that is used in labor to assist workers during fatigue loads during lowering and lifting loads. Moreover, designing a testbench in lab to perform all the tests needed on the device. The exoskeleton relieves the wearer of 30% of his weight load during bending through his hip joint. First, the designed model of the exoskeleton was available on MATLAB, Simscape along with the dedicated control law to actuate the device at the correct torques and trajectories. In this thesis, other control strategies were tested and compared to reach the ultimate law. Afterwards, implementing the control law on a target hardware and developing the coding procedures was studied and implemented soldering a PCB to implement the control law on a proportional pressure regulator FESTO valve controlled by an arduinoUNO board. Finally, importing CAD models of the testbench into MATLAB, Simscape and performing simulations to emulate the human dynamics by the testing device.

1 *Introduction*

1.1 *Motivation*

In recent years, exoskeletons have widely spread throughout different fields. Most widely in the industrial manufacturing sector as it introduces new ways of enhancing work production as well as task effectiveness, precision and relieving very hard efforts from wearer's body. Exoskeletons make use of the human intelligence with the robustness and dexterity of mechanical systems. In our scope, the development of an industrial type exoskeleton is a necessity to keep pace with industrial developments and need in different fields. Thus, the main goal of this thesis is to continue the work of previous colleges on the studying, design and control optimization of our industrial type exoskeleton that factory workers could wear to hinder lower-back and hip pain.

1.2 *History*

As stated, before exoskeletons have different areas of application, but as a starting point the history of their introduction to life was studied in several fields. It started in the seventeenth century that the human body can be supported by a purely mechanical device that can be, for instance, steam powered. In 1738, the first exoskeleton was introduced by Jacques de Vaucanson, which was an android playing a flute. Then in the eighteenth century a steam powered legs were introduced (Fig. 1.1) (Kazerooni, 2008)[1]. Near the end of the 1800s, the scientist Nicolas Yang designed another shape for the exoskeleton using leaf springs that was used to assist Russian soldiers to run faster and jump to higher amplitudes (Fig. 1.2) [2].

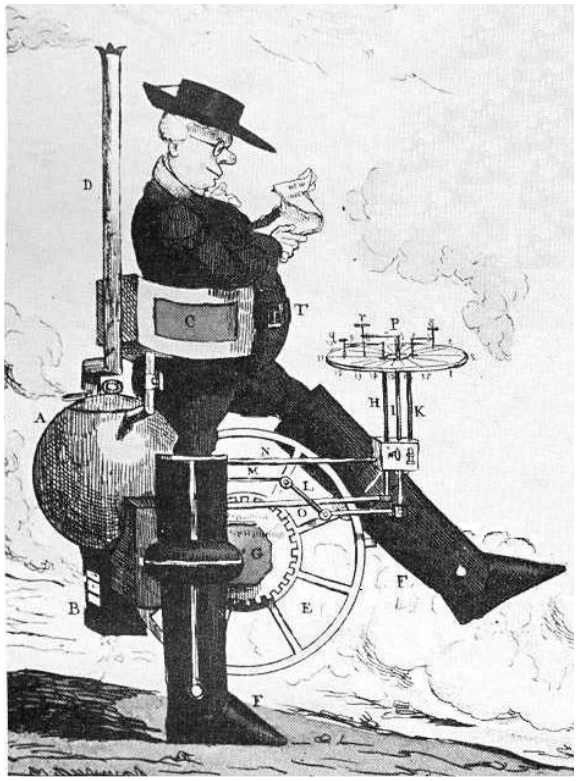


Figure 1.1: Steam powered legs.

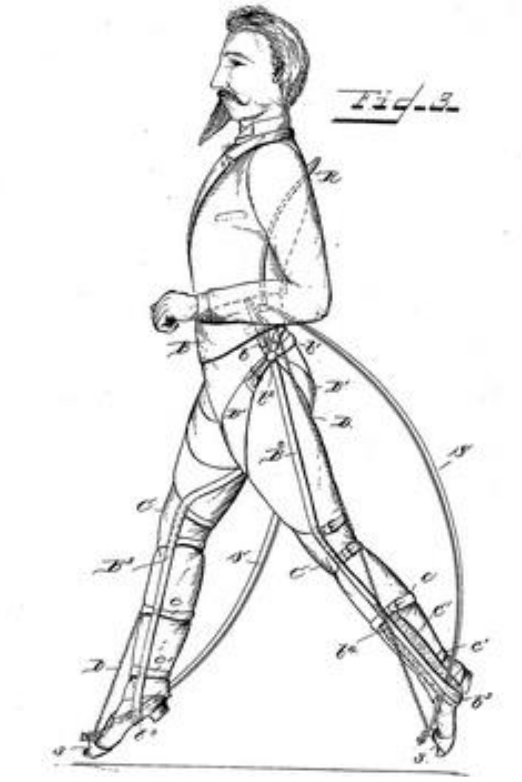


Figure 1.2: soldier's passive assistive legs. N. Yang [2]

In 1893, George Moor created a steam man (Fig. 1.3). The system was powered by a 0.5 hp gas fired boiler and reached a speed of 14 Km/h.

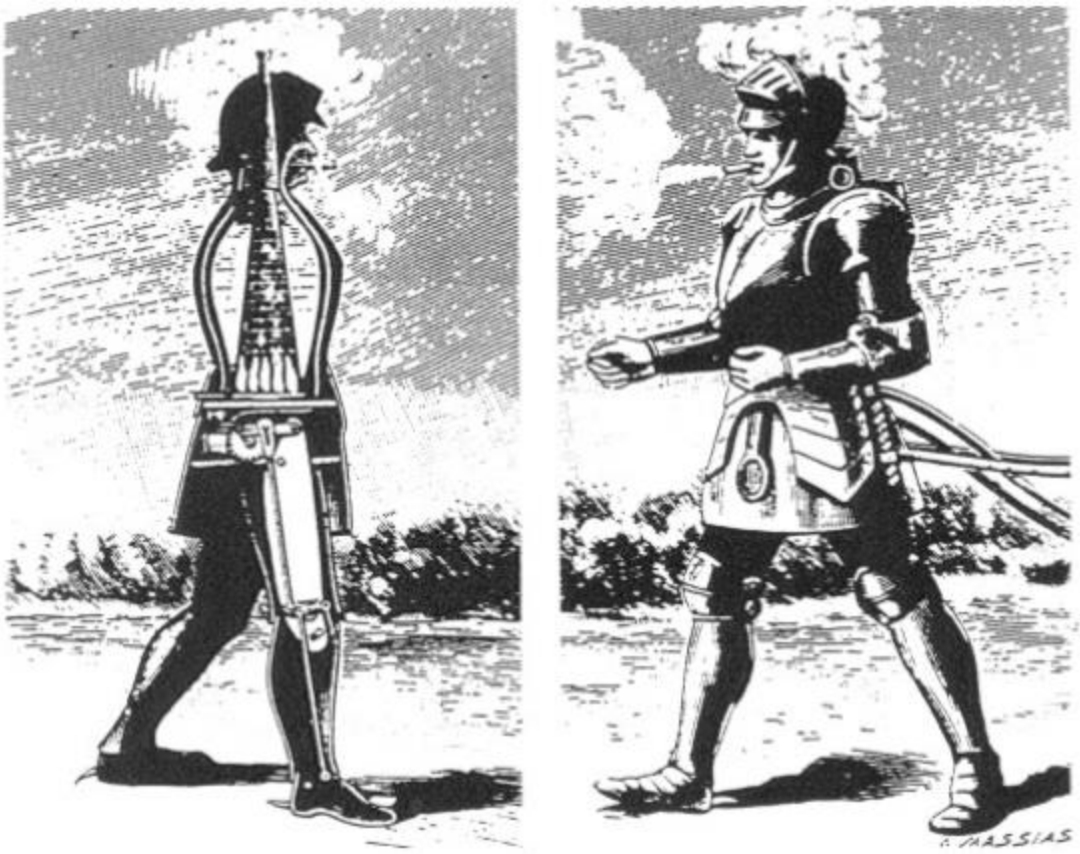
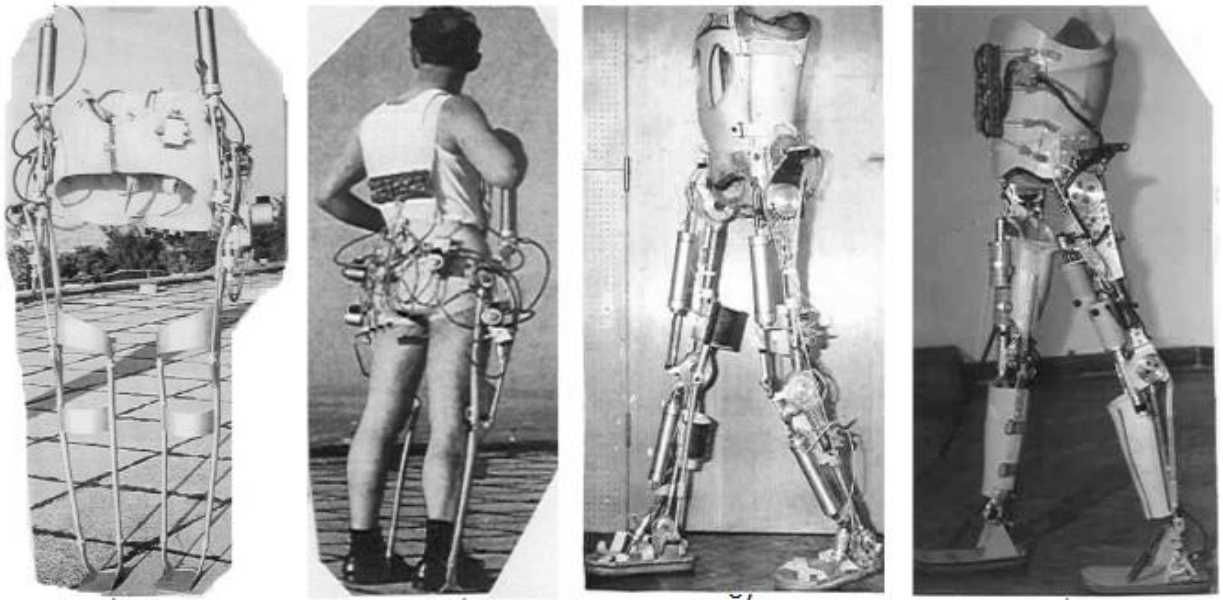


Figure 1.3: Steam powered man by Moore [3]

1.2.1 Humanoid Robots

Although all previous examples were studied but they were not implemented due to technological limitations then. Ideas kept on developing till the mid-1900s as the humanoid robots were introduced and implemented. One of the world's first active exoskeletons was implemented at the Mihailo Pupin Institute in 1969, under the guidance of Prof. Vukobratovic [4] [5] [6] [7]. It should be noted that legged locomotion systems were developed first. In the previous articles attempts to produce the first exoskeletons to assist handicapped persons were introduced. These exoskeletons were mainly pneumatically powered and some of them had

more complex realization using electric controls. The world's first active exoskeleton in (Fig.1.4a, b) was pneumatically powered and partly Kinematically programmed for producing andromorphic gait. Most successful version of an active exoskeleton for rehabilitation disabled persons, pneumatically powered and electronically programmed. It was realized and tested at Belgrade Orthopedic Clinic in 1972 (fig. 1.4c). Then in 1974, exoskeletons with electromechanical drives and electronically programmed were build and used to develop electro-mechanical drives for active orthotic devices. This was the first example known in the world of active exoskeleton that used electric motors as actuators (Fig1.4d). As such, it can be considered as a predecessor of the contemporary humanoid robots driven by electric motors.



a *b* *c* *d*

Figure 1.4: World's first walking active exoskeleton (a and b); active exoskeleton for rehabilitation of paraplegics and similar disabled persons (c); Active exoskeleton with electromechanical drives (d).

In the mid-20th century, NASA details the background of 'man–machine relationship in the report space exploration on a large scale. As in the 60s and 70s,

the field of Artificial Intelligence (AI) was advancing, scientist thought out its research. As the philosopher Hubert Dreyfus argued in his book 'What computers can't do? The limitation of artificial intelligence' [8], he presented four progressively weaker interpretations of the physical system hypotheses that he termed the biological, psychological, epistemological, and ontological assumptions of traditional AI. And so, during the end of the previous century many scientists and researchers acknowledged that the interference of human intelligence side by side with AI is a necessity and is more beneficial than spending research in another end. Two of these scientists: Lu and Chen [9] from Zhejiang University, introduced the concept of humachine [10, 11] which defines the man-machine system.

1.3 *Development of exoskeletons*

In the timeline of the development of humachines or exoskeletons and specifically in the period from 1960s till the beginning of this century, they could be classified into three stages. The first one is as an arm used in telerobotics and then followed by a tool for human upper limb or finger pose measurements and finally for rehabilitation usage for the disabled.

These three stages introduced slowly the force feedback systems and as the development of feedback control systems prevailed in 1990s, force feedback exoskeletons were widely employed. And with the human intelligence in the system sideways with force feedback, the exoskeleton systems improved very rapidly into several fields as power amplification and haptic devices. Presented below two main areas of development.

1.3.1 Exoskeletons in teleoperation

Due to the advances already mentioned the master slave robots can generate a realistic feeling as if a human is doing the task. In 1970s, several companies and universities developed master system for robotic teleoperation as GE and university of Washington [12], Ohio state university and Stanford [13, 14]. However, all these models were not portable. In this thesis we are dealing with the other type which is power augmentation of a wearable device for workers rather than a virtual master-slave arm.

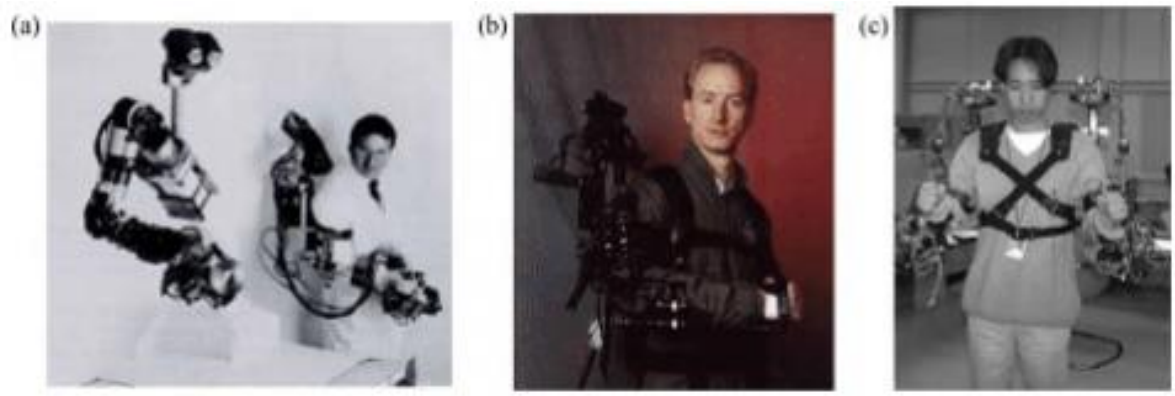


Figure 1.5: Exoskeleton-type master arm: (a) exoskeleton master of GE; (b) exoskeleton arm of Stanford [13]; (c) KIST exoskeleton arm with electric brakes

1.3.2 Exoskeletons for power augmentation

Many researchers and scientists all over the world are working to develop a new type of exoskeleton acting as power amplifier to help human workers accomplish tasks with heavy loads which exceed the human power and can be a cause of serious health problems. The robotics laboratory at UC Berkley designed a system called 'Human Powered Extender' to assist the workers on assembly lines[15] such that when the worker is operating a 100 kg load he only feels 5% of the load while the remaining 95 kg are supported by the exoskeleton. Thus, by combining the

human operator and the robotic body the difficulties of the gait task is simplified and solved in a manner that is more comfortable for the human operator so he can be more able to accomplish more tasks and work for longer shifts and longer distances with less effort exerted.

Figure 1.6: Walking assistive exoskeleton: (a) HAL-5 system [16]; (b) BLEEX lower limb exoskeleton [17]; (c) exoskeleton leg for human's walking power augmentation of Zhejiang University

1.4 *Biological concept*

The concept of our exoskeleton was extracted from nature as the term exoskeleton is a biological term which indicates an external covering of an animal's body to supply it with protection or covering such as a shell of a crab. It also provides the animal with w surface for muscle attachments, a water seal, and a sensory interface with the surroundings. And the table below (Table 1.1) illustrated the main resemblance between the biological and industrial concept of exoskeletons.

Function	The biological exoskeleton	The exoskeleton technology	Application
Support	Supporting the body of the invertebrates	Supporting physically disabled patient or walking assistance	Rehabilitation robotics or power amplifier
Enhancement	Enhancing the power of animals	Strengthening the human operator	Assistance equipments
Protection	Protecting the animal's body	Protecting the human operator	Automatic armour for soldier, rescue devices or safe manipulation for the radioactive materials in nuclear plant
Sensing and data fusion	Obtaining the information, acting as the sensorium	Interface of human operator and the environment and making data fusion with information obtained by the human operator	Telerobotics, VR

Table 1.1: The comparisons between the exoskeleton technology and that concept in biology.

1.5 *Human anatomy*

During the design procedure of exoskeletons, the human anatomy is studied and resembled on the exoskeleton to assist the movements of the body and the muscles. The human links as the thigh have several elements: passive as bones, ligaments, and discs as well as active as the muscles. And because humans vary on a large scale in their body formations designing exoskeletons to perfectly fit different wearers are not possible, which makes it even more hard for manufacturers to accept a design and refuse another. Unfit exoskeletons not only are not comfortable but also can have diverse effects on the wearer as can cause a joint to rotate around an axis of rotation that is not perfectly true causing serious injuries. For instance, the knee joint axis of rotation moves with each deflection of the joint (Fig. 1.7). Thus, several design guidelines related to sizing and fitting of exoskeletons to wearers were described in [15] such that if the device does not fit well the wearer is to make the adjustments. A device called the goniometer was designed to measure the joint axis of rotation locations due to its movement during rotation which highly depends in the measurement on the alignment of the device with the limb. However, it's known exactly the position of rotational axes

of several human joints. Although it may differ for a single human being between left and right limbs. Below are figures to discuss human joints. (Fig.1.8A)

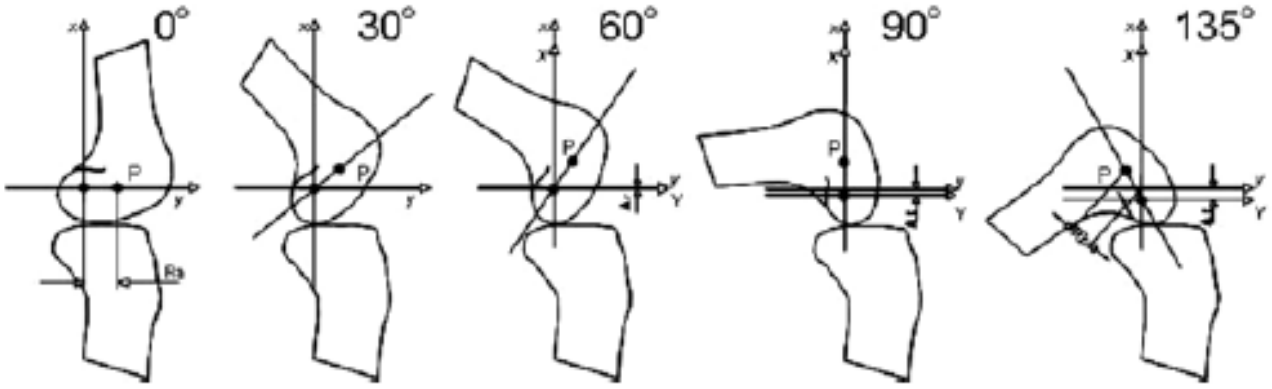


Figure 1.7: Compounded knee rotation and lift at various joint angles.

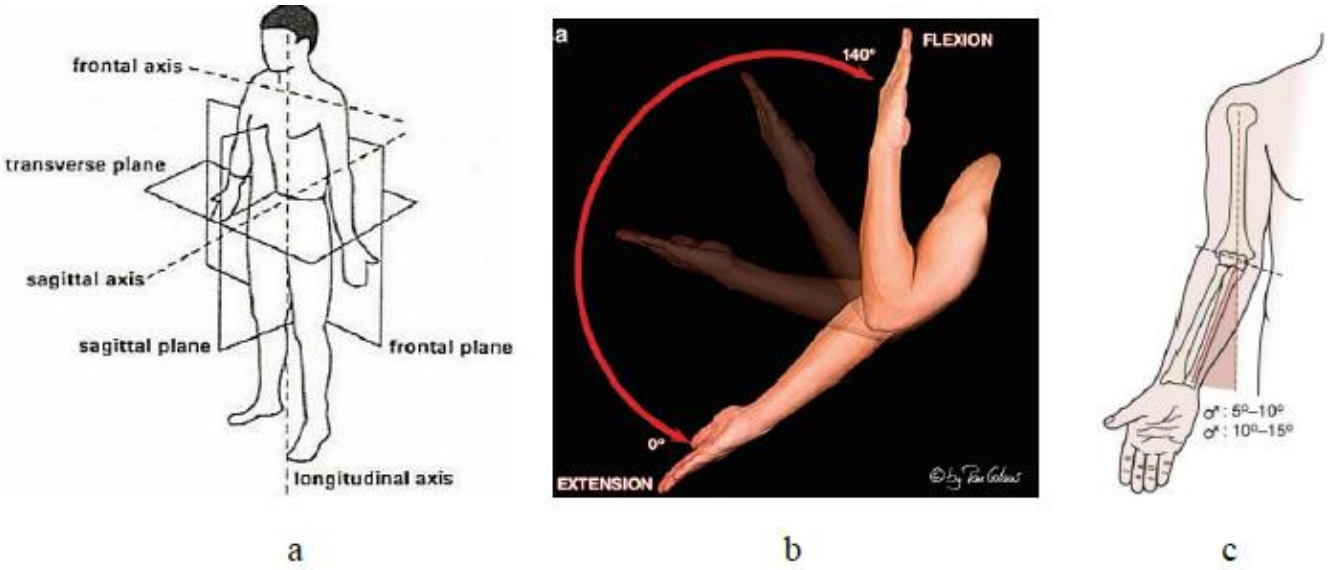


Figure 1.8A: (a) Human body reference planes. (b) Elbow extension through flexion angle. [19] (c) Side-to-side joint rotation between the humerus and the ulna during normal carrying angle.

1.6 *Standards and safety*

1.6.1 ISO 13482

In 2014 after several years of developments the international standardization organization (ISO) 13482 [16] made a publication to address the safety concerns

of robots including exoskeletons including the exoskeletons used for health care services (Fig.1.8B). On the other hand, those used for rehabilitation of the disabled where excluded from the publication. However, the safety standard does not include any normative requirements on the analysis and data collection of the

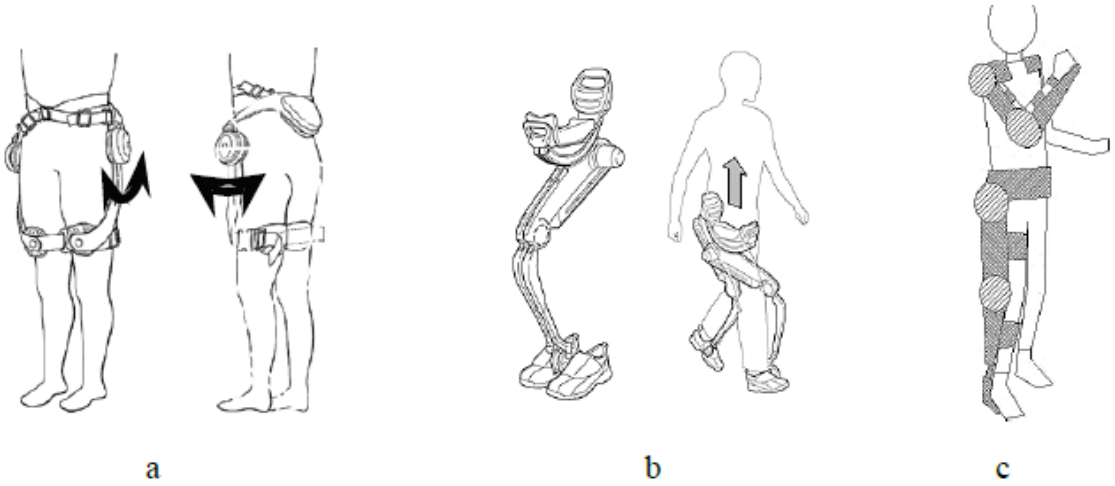


Figure 1.8B: Basic exoskeleton designs shown in ISO 13482 (a) Leg motion assistive device, (b) Body weight supportive device, (c) Exoskeleton wearable robot.

exoskeleton under validation to understand mainly the long-term effects of its usage. But these test data are needed by manufactures to replicate on the standards and tests in a formal manner. These tests and data collected have an impact in understanding the long-term problems and their consequences on the human and environment. Thus, exoskeletons have some safety considerations. For instance, what are the issues faced upon usage of low extremity exoskeletons on the wearer? To what extent can the device provide lift to augment the wearer so that after a long period of wearing and performing tasks, as crouching and standing up, wearing the exoskeleton, what are the signs of fatigue or maybe sore skin that the wearer encounters? Are there any signs of overdrive to the human joints that can cause fatigue of the joints and can have an impact on the long run? And if we have one or more of these problems or other ones how should they be

encountered and treated? Are there some problems neglected, or all complications must be accounted for and provide radical solutions? So, how should the exoskeleton be designed to be safe? Thus, manufacturers must be provided with formal test methods and standards to implement their designs and be able to have a comparison between safety and capability to perform specific tasks according to their design. Moreover, performance tests also provide results about specific tasks of the extent the device is improved and can perform its tasks better. For example, better trajectory, higher speeds, higher loads, more comfort to the wearer and better precision in positioning. Unfortunately, nowadays there are no such tests or standards to provide manufacturers with broad guidelines of how they should design and evaluate their products to provide the market with human safe robots. Additionally, there are no commonly agreed upon physiological measurements of the human user to provide baseline measurements of before and after long term use of exoskeletons.

1.6.2 NIST

Although the ISO has no data about testing of exoskeletons but The National Institute of Standards and Technology (NIST), Robotic Systems for Smart Manufacturing (RSSM) Program [17] provides test methods and measurement science for stationary and mobile robot arms as well as vehicles to support calibration, measurement, and advance understanding of current and target performance of emerging new capabilities for manufacturing applications. The program develops and deploys advances in measurement science that enhance U. S. innovation and industrial competitiveness by improving robotic system performance to achieve dynamic production for assembly-centric manufacturing. Moreover, NIST also works on developing the performance of remotely operated

emergency response robots by evaluating the standard tests and qualifications required to specify the system capabilities under Emergency Response Robot Project. [18]

One of the examples of RSSM program is the development of test methods for mobile arms that are used for assembly. Assembly arms must have very high precision and accuracy of some variables as speed and trajectory when performing a task as gear meshing or inserting a pin into a socket. In addition, when evaluating the test results, one of the important aspects to be measured is the repeatability so as when performing a task that needs high precision the arm or exoskeleton should be able to redo the task in the same precise variables. Thus, the lesson to be noted form studying measurement methods is applying them to exoskeletons where, positioning of loads or human knee or elbow joints should be performed in a repeatable way. Also, there are more tests to be studied and accounted for and published in the future that need to be developed by understanding exoskeleton performance metrics.

1.6.3 Exoskeleton performance metrics

Tests on industrial robot systems does not include human-in-the-loop testing procedures because robot functions are automatically actuated and are separated from human impacts. However, this perspective is changing when dealing with response robot systems as the main concept of its operation is combining the human intelligence with the robustness and power of mechanical systems such that a human operator is performing a task with no loads and the slave robot is mimicking the movements inside the workspace with all working conditions. From [19], we extract common metrics for task-oriented mobile robot human-robot interaction that may also apply to exoskeletons:

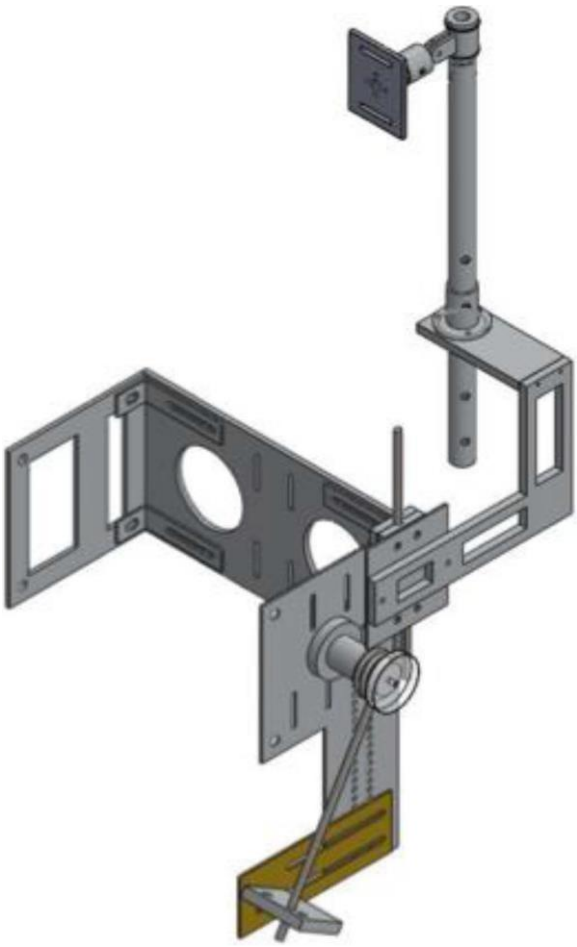
- Navigation: ability of the exoskeleton to support the wearer in gait and in lifting actions.
- Perception: ability of the exoskeleton to acquire body movements and act upon it.
- Management of tasks: ability of the exoskeleton to act upon emergency situations as facing an obstacle or sudden human movements.
- Manipulation: ability of the exoskeleton to move the supported human limb in the correct trajectory to perform the specified task.
- Duration: comparison between the task performance duration with and without wearing the exoskeleton.
- Speed: Comparison between the velocities of performance with and without wearing the exoskeleton.
- Acceleration/Deceleration: Comparison between the rate of change of the velocities with and without wearing the exoskeleton.
- Pose uncertainty includes accuracy and precision to perform a task in a correct manner, as well as repeatability to provide the ability of the devise to repeat the same task exactly each time.
- Control force: the force required to move one or all components of the exoskeleton weather or not they are under actuation.
- Ergonomics: the comfort sensed by the wearer during usage of the exoskeleton in complex tasks or for long periods of time.
- Ingress/Egress complexity: complexity of putting on or taking off the exoskeleton.
- Ease of use: easiness of learning how to use the exoskeleton and perform tasks faster using it.

- Other: such as cost and battery life.

1.7 *Industrial exoskeletons*

As mentioned before exoskeletons has several applications in several fields. As an example: medical, military, and industrial. In medical fields, there are two branches depending on usage either used by a patient or by a caregiver. Patients recovering form accidents or even the disabled own a huge segment in the manufacturing market of exoskeletons (Fig. 1.9a). However, caregivers who work for prolonged hours with the elderly and their job is about lifting another person to assist them, they mostly suffer from severe back pains. As it has been studied in [20] lower back pain occurs under compression of lumbar region to elevated forces reaching 3,400N which corresponds to a lifting weight of 34 Kgs according to [21] (Fig. 1.9b). Secondly, in the military field that is still expanding day by day, many researches were made to design exoskeletons to assist soldiers with their heavy gear and to perform abnormal tasks to human ability (Fig. 1.9c).

(a)

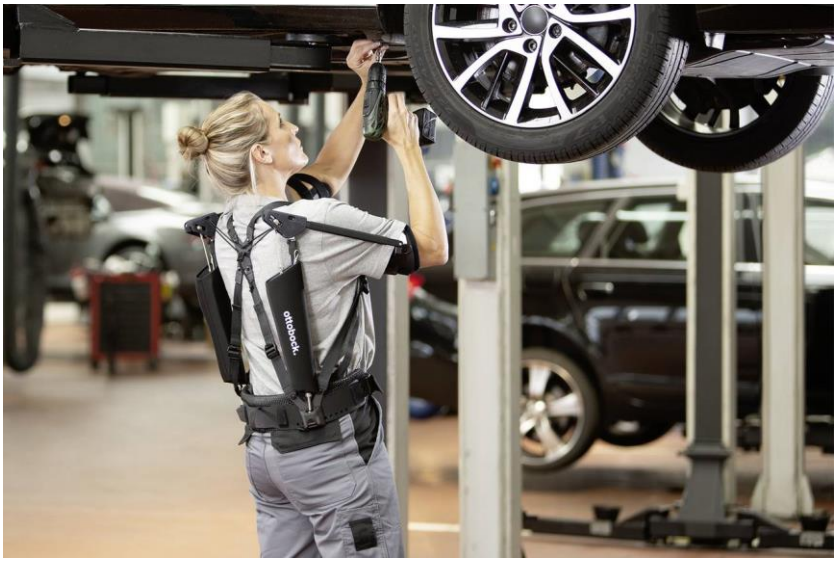


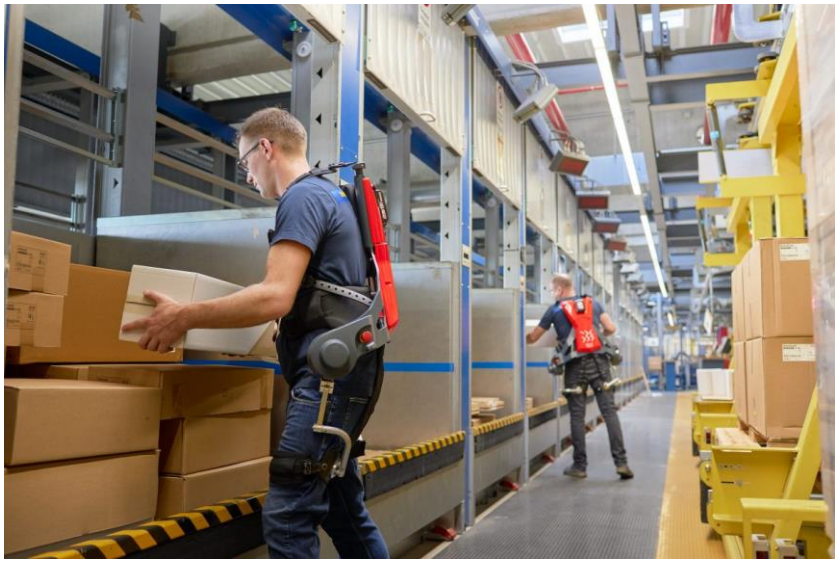
(b) (c)

Fig. 1.9: (a) Medical exoskeleton used by a patient. (b) Medical exoskeleton used by a caregiver designed by [21]. (c) Military exoskeleton.

In our study, we are designing a pneumatic powered active exoskeleton to be used in industrial applications such as workers in factories with heavy loads as automotive fields (Fig. 1.10a), inside warehouses (Fig. 1.10b) or rubble rescuers (Fig. 1.10c).

(a)





(b)

(c)

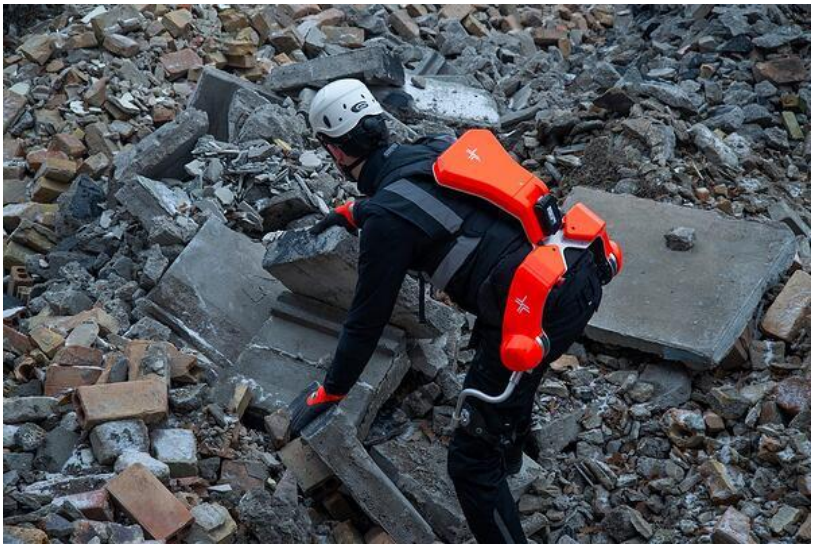


Figure 1.10: (a) Automotive exoskeleton. (b) Wearhouse exoskeleton worker (c) Rubble rescuer wearing an exoskeleton.

1.7.1 Types of exoskeletons

Due to Diversity in applications and design, to categorize exoskeletons we find several classifications.

Firstly, by body part:

- **Upper extremity exoskeletons:** These provide support to the upper body, including the arms, shoulders, and torso. Such as Ottobock Paexo (Fig. 1.10).

Figure 1.11: Ottobock Paexo Upper extremity exoskeleton.

- **Lower extremity exoskeletons:** These provide support to the legs, hips, and lower torso. Such as ReWalk Robotics Restore soft exoskeleton (Fig. 1.12).

Figure 1.12: ReWalk Robotics Restore soft exoskeleton type of exoskeleton.

- **Full body exoskeletons:** These combine the first two types thus, supports the upper and lower extremities concurrently which makes them the most powerful yet most complicated types of exoskeletons. Such as Guardian XO from Sarco Robotics (Fig. 1.13).

Figure 1.13: Guardian XO from Sarcos Robotics.

Secondly, By Formation:

- Rigid/Hard/Active: These exoskeletons use rigid links, joints and actuators and include an external source of power so that they provide very high-power ratings (Fig. 1.14).

Figure 1.14: Pneumatically powered active (rigid) type exoskeleton.

- Soft/Passive: Made of soft materials as fabrics and bands made of fabrics such that they provide support for the muscle during actuation by human. This makes the power provided by these types of exoskeletons very small compared to the first type (Fig. 1.15).

Figure 1.15: Passive type exoskeleton.

- Mixed: Many exoskeletons nowadays have both powered actuators on some joints and passive straps on others to make a functional suit that emulates several human joints (Fig.1.16).

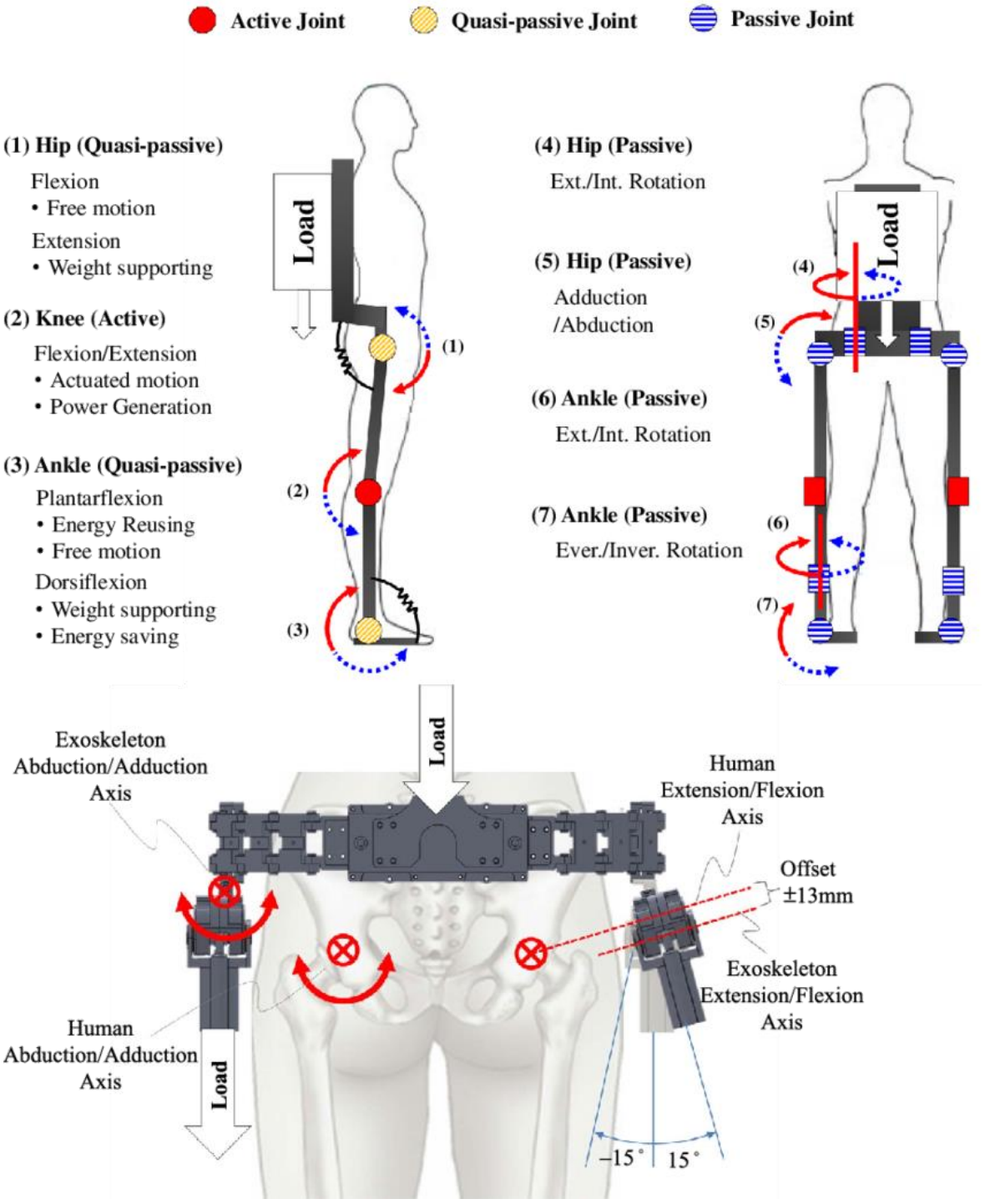


Figure 1.16: Mixed type exoskeleton between active and passive.

1.8 *Our Goal*

Previous master's student colleagues and I are working sequentially on developing a pneumatically powered active industrial exoskeleton. It shall be used to assist 30% of the weight being carried. In the previous studies, my colleagues made some design procedures (Fig. 1.17) and components selection and then others developed CAD drawings and simulation tools (Fig. 1.18) and finally some control schemes where presented by my predecessor who developed a complete model on MATLAB which comprises different control schemes using PID, LQR and MPC side by side with the CAD design on Simulink, Simscape thus, integrating the whole system on MATLAB.

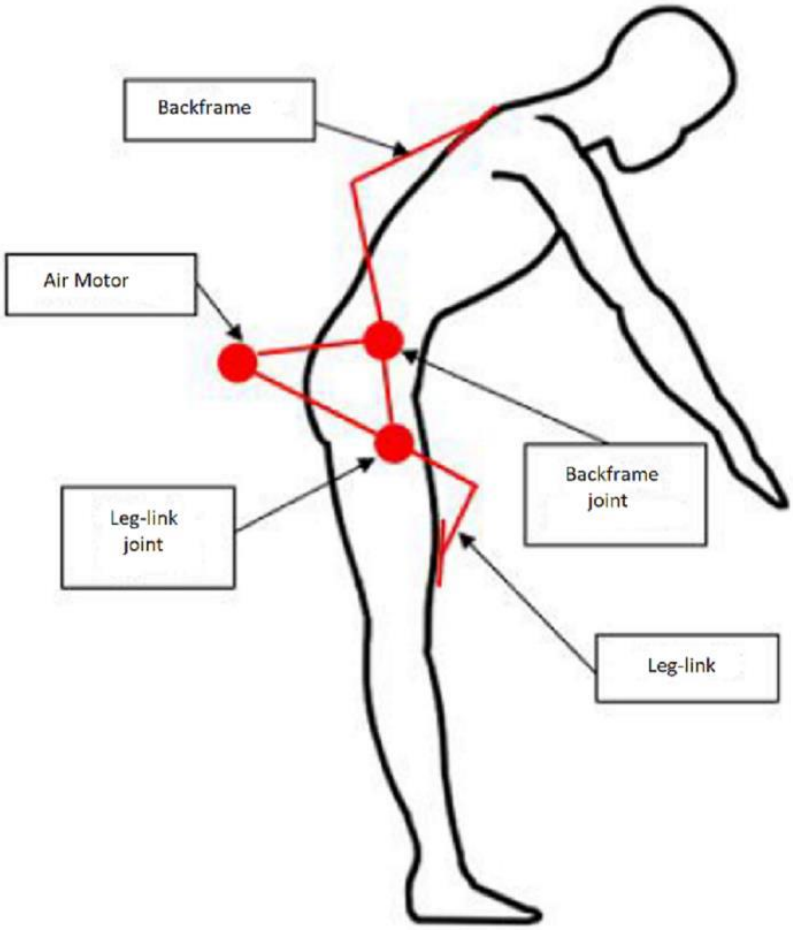


Figure 1.17: Design concept of the pneumatic exoskeleton.

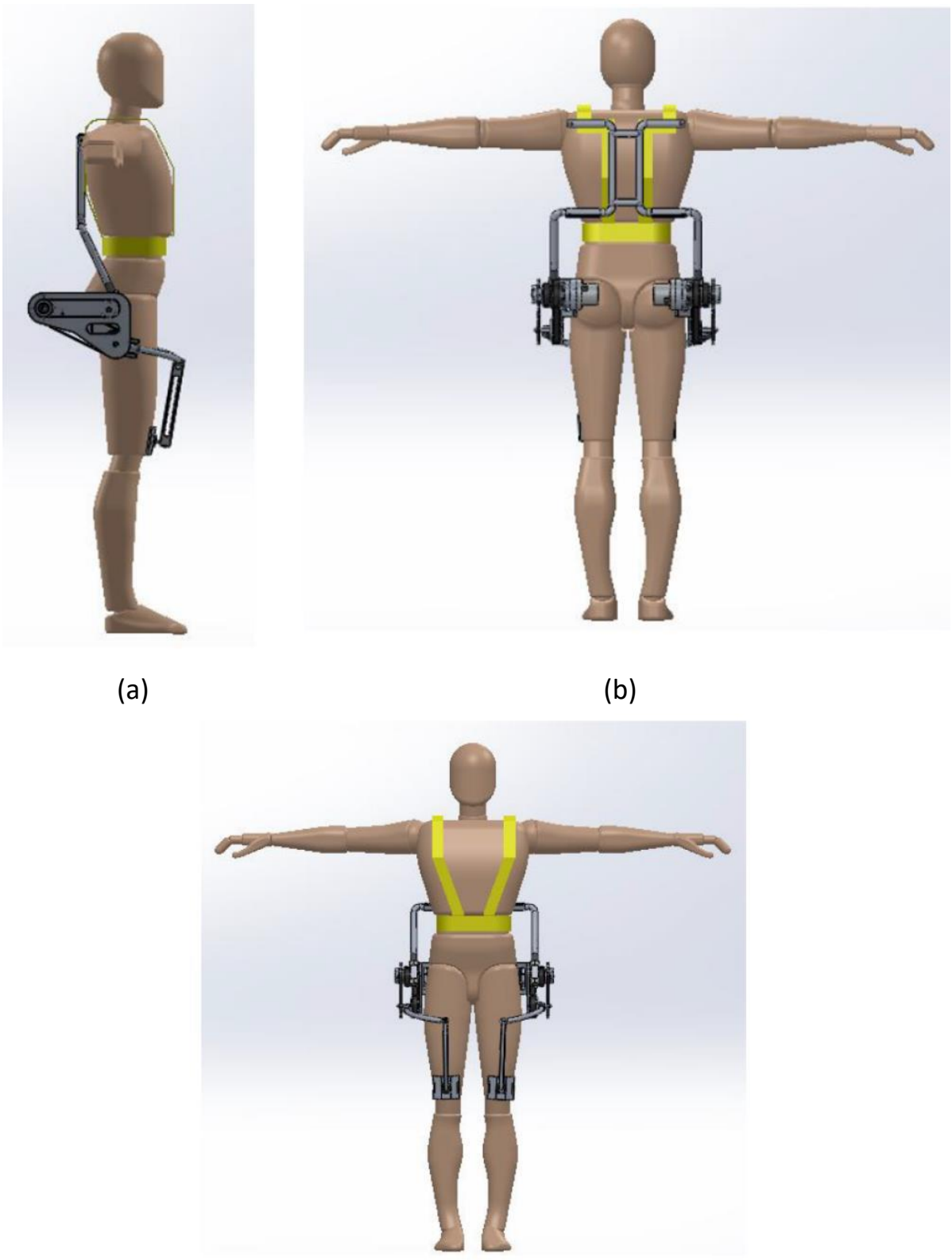


Figure 1.18: (a) Side-view, (b) Back-View, (c) Frontal view Complete CAD design.

2 Controlled model

Defining a simple mathematical model that is yet enough to describe our pneumatically powered exoskeleton was already developed and simulated to provide accurately 30% of the work done by the wearer, which is our main goal. Then comes the stage of developing a robust control strategy based of studied control theories and implemented on similar projects. At this stage, several control laws were implemented, simulated and tested to provide several outputs that are all highly efficient for our project. Finally, the best control law was chosen, verified and tested.

2.1 Cycle of operation

The lifting operation of a human can be modeled into several scenarios. Simply, we defined this operation as a cycle which consists of 4 phases and are repeated continuously. These phases are defined according to (table 2.1) and (fig. 2.19). With integrating the 4 phases the trajectory of the operation is present and the following step is modeling the different components of the exoskeleton system and defining the inputs, outputs and reference to the control law.

Phase	Time (s)	Law of motion
Bending	3	Cycloidal
Working	30	Costant
Lifting	3	Cycloidal
Resting	24	Costant

Table 2.2: 4 phases of the lifting operation.

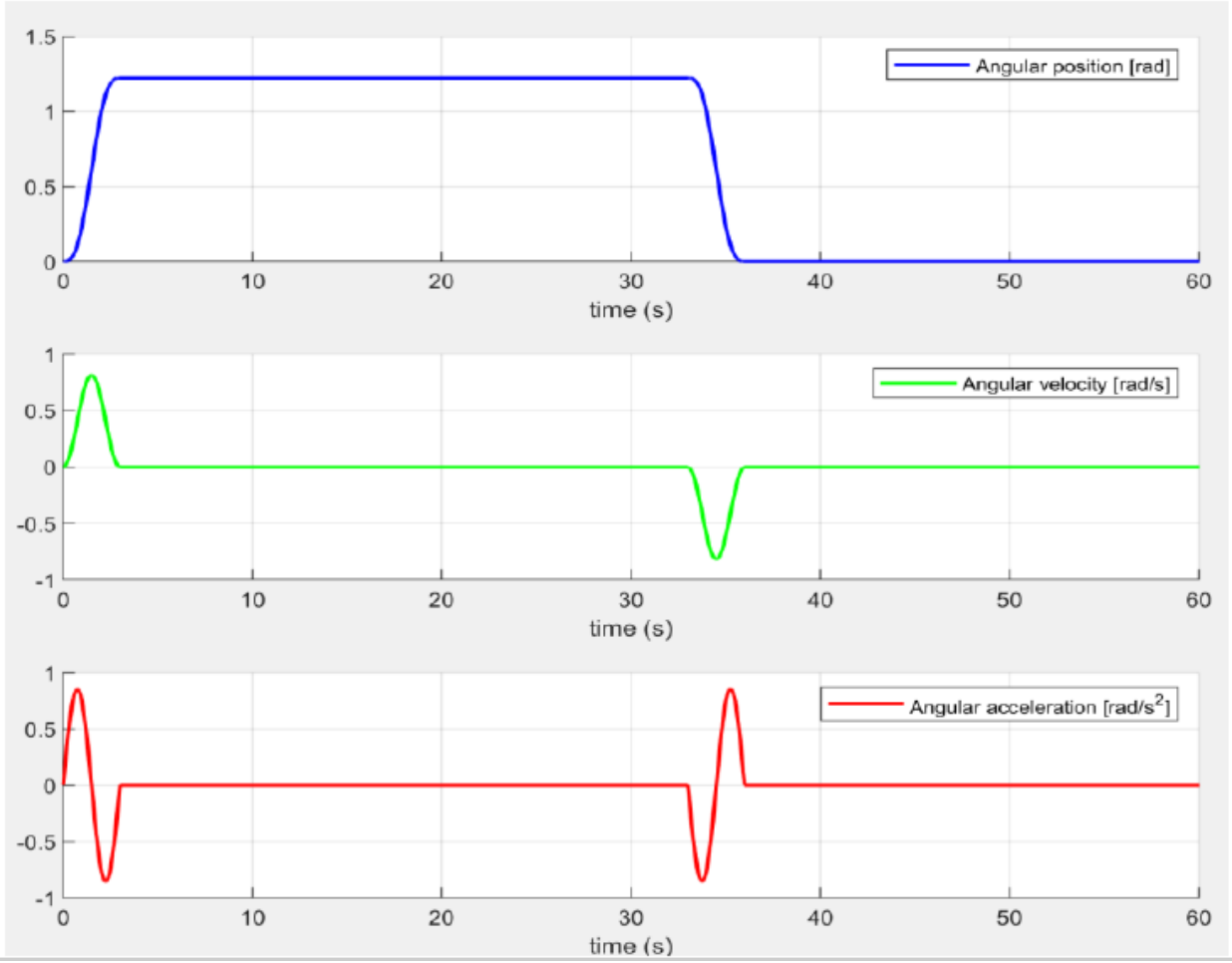


Figure 2.19: Input quantities trends.

2.2 Modeling

After studying the work cycle, modeling of different components step is evaluated such as:

1) Human body: The human model is described through simple equation that has as output C_{SET} which is the human torque (fig.2.20) .As described before the input to the control law is supposed to be the torque whose the value is defined which is 30% (our goal). Thus, the human model developed has as an output the human torque and

therefore the input to the controller of our plant is 30% of that torque. Yet defining the torque as the input to the control law.

2) Proportional valve: Its model defines the voltage-pressure characteristics of the valve through a simple graphical representation. (fig.2.21)
3) Air motor: The actuator is chosen pneumatic for several reasons. For instance: high power to weight ratio to account for the overall weight of the device as well as ability to work in dusty atmosphere. It is reported in detail the trend of the angular torque-speed characteristic (Fig. 2.22a) and the trend of the correction factor for torque only (Fig. 2.22b).

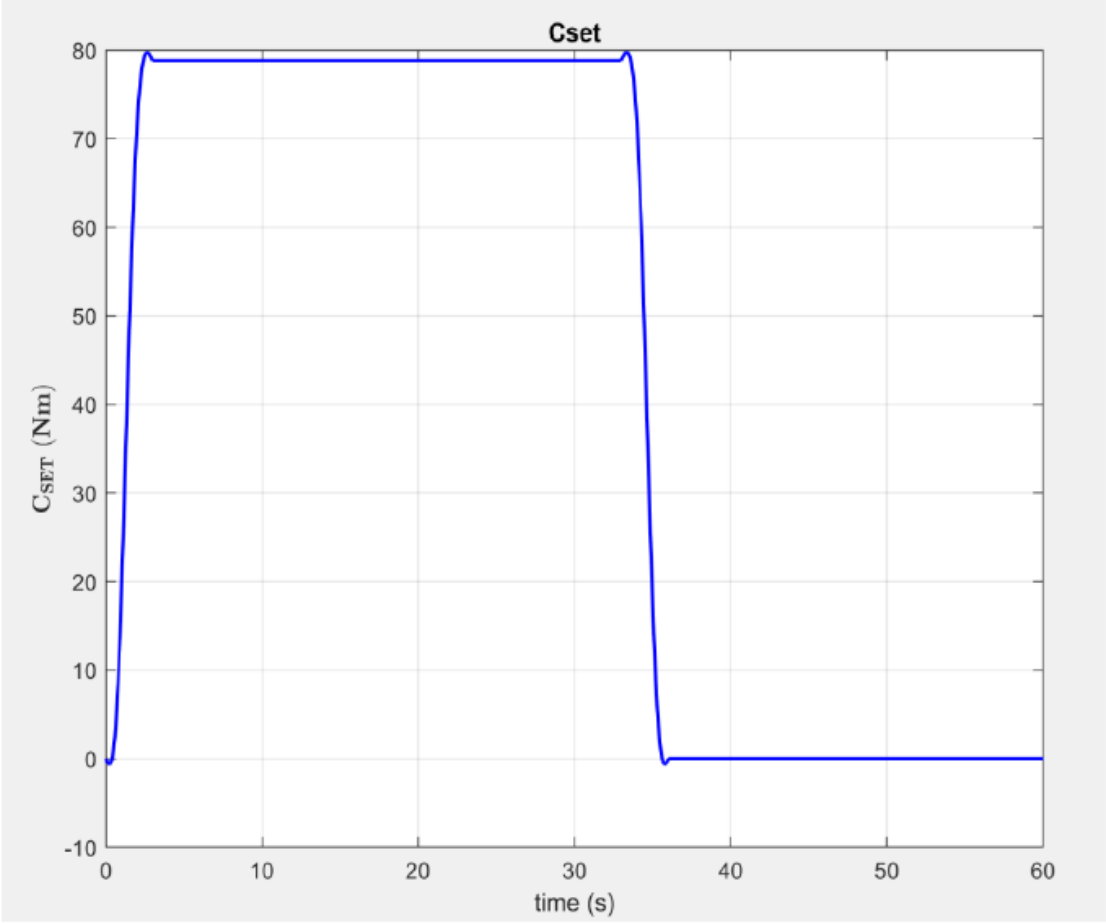


Figure 2.20: C_{SET} value of the torque that is 30% of the human toque.

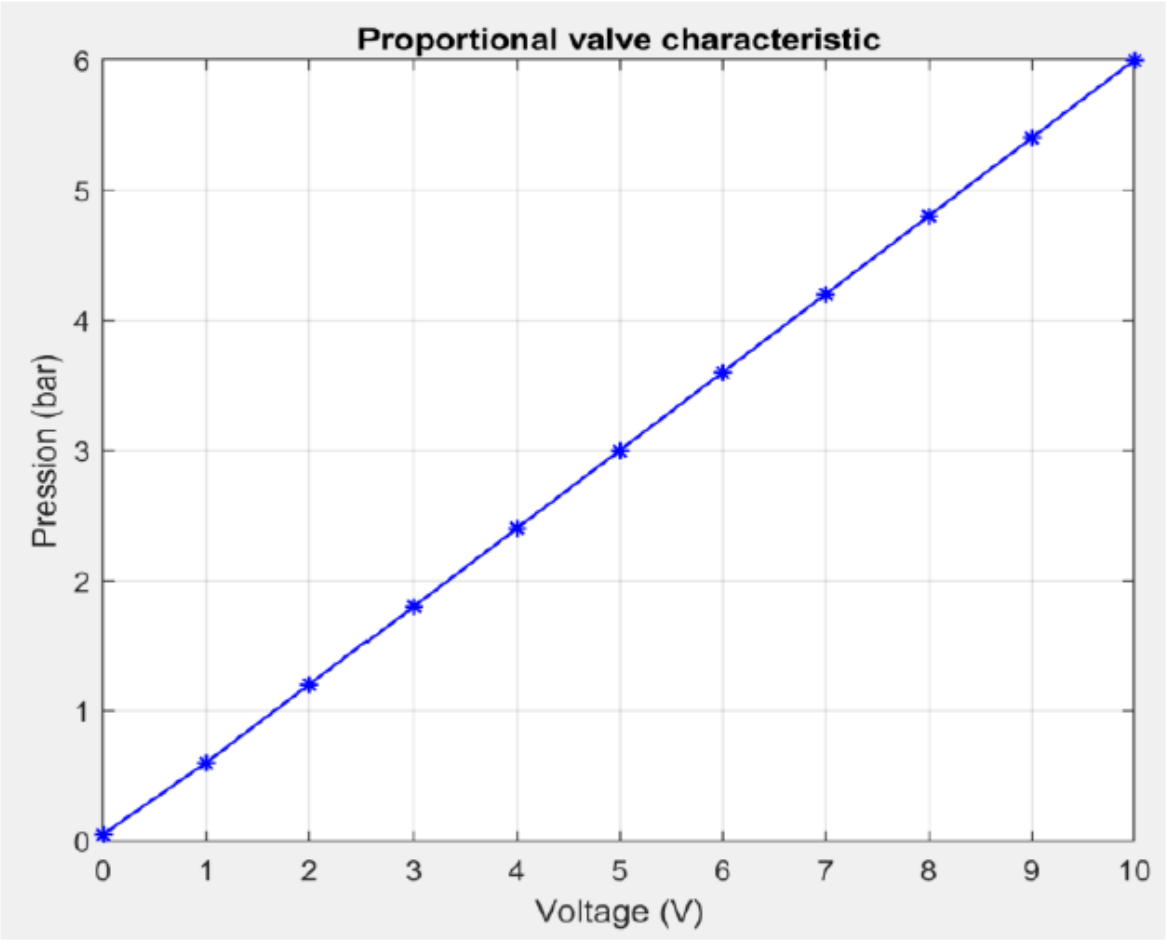


Figure 2.21: Voltage Pressure characteristics of the proportional valve.

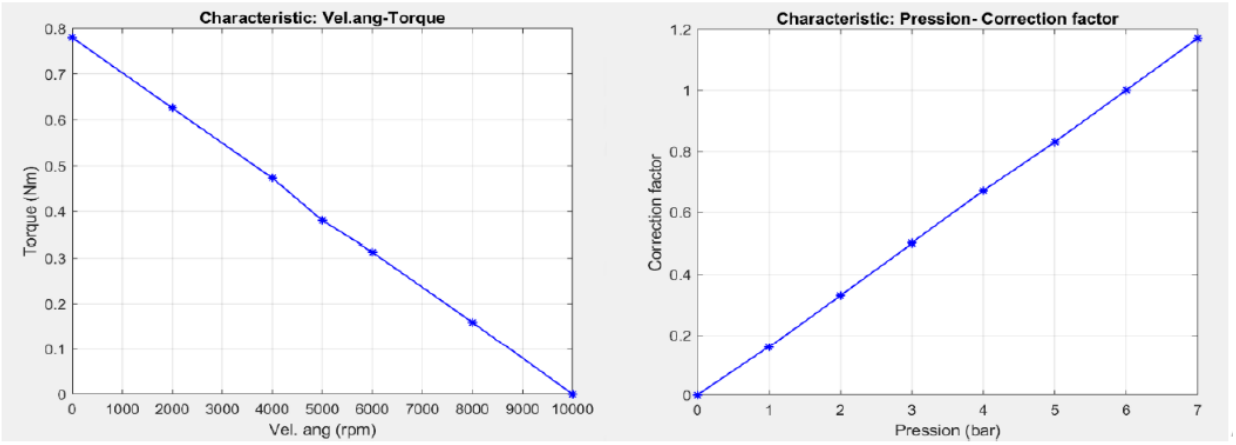


Figure 2.22: (a) Trend of the angular torque-speed characteristic. (b) Trend of the correction factor for torque.

After assembling all these models in a Simulink scheme including the controller, which is to be defined, we can simulate a primary evaluation of the entire plant. The whole scheme is found in (fig.2.23). Understanding that C_{SET} is the input to controller and the generated torque as output (C_{FB}) and comparing them to guarantee a zero-tracking error and a robust system (Fig.2.24).

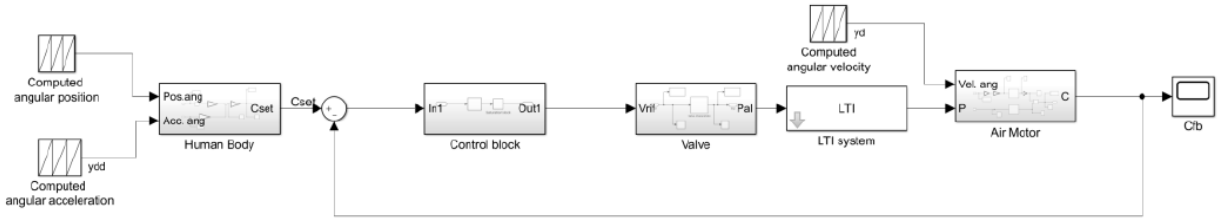


Figure 2.23: Primary Simulink Scheme.

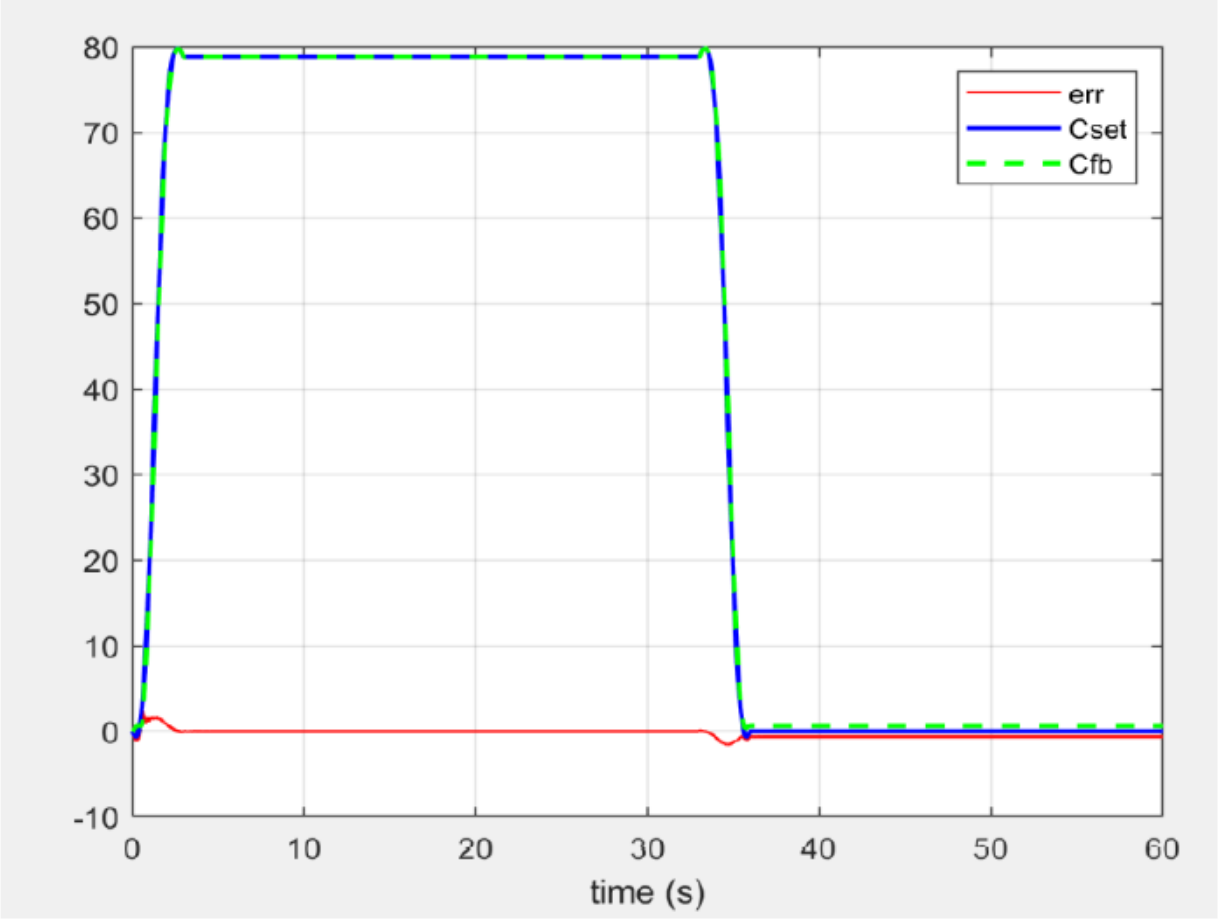


Figure 2.24: Comparison between C_{SET} and C_{FB}, and the error developed.

2.3 Control Strategies

Model-based control system is a typical exoskeleton control strategy. According to the model, the control strategy for the skeleton is classified into two types: the dynamic model and the muscle model-based control [22]. The dynamic exoskeleton model is implemented by modeling the human body as rigid links joined by joints. This model is built from combination of inertial, gravitational, Coriolis and centrifugal effects. The dynamic model can be obtained through two ways; the mathematical model and the system identification.

The Mathematical model, already described, is obtained by theoretical modelling based on physical characteristics of the system. The good practice of this control system is our model of the exoskeleton. It consists of two joints in the pelvic region with the same direction of motion. The flexion-extension at the hip is actuated while rotation. The model at hand relies on its dynamic model to aid the user's movement, with force/torque sensor to detect the interaction between the user and the exoskeleton. The control goal is to attain the system with high sensitivity, yet safe on the user. However, this sort of control aim demands the precise dynamic model. However, that high level of precision is not possible with our computing capabilities. So, simpler models were developed for the human body, the air motor and the rigid links. The second way to obtain the dynamic model is the system identification method. This method is used since it is difficult to attain a good dynamic model by using theoretical mathematic model. Thus, defining the following steps in our modeling procedures. A complete model of the exoskeleton

was developed on SolidWorks with all the design parameters including masses, weights, inertia, etc. (Fig.2.25).

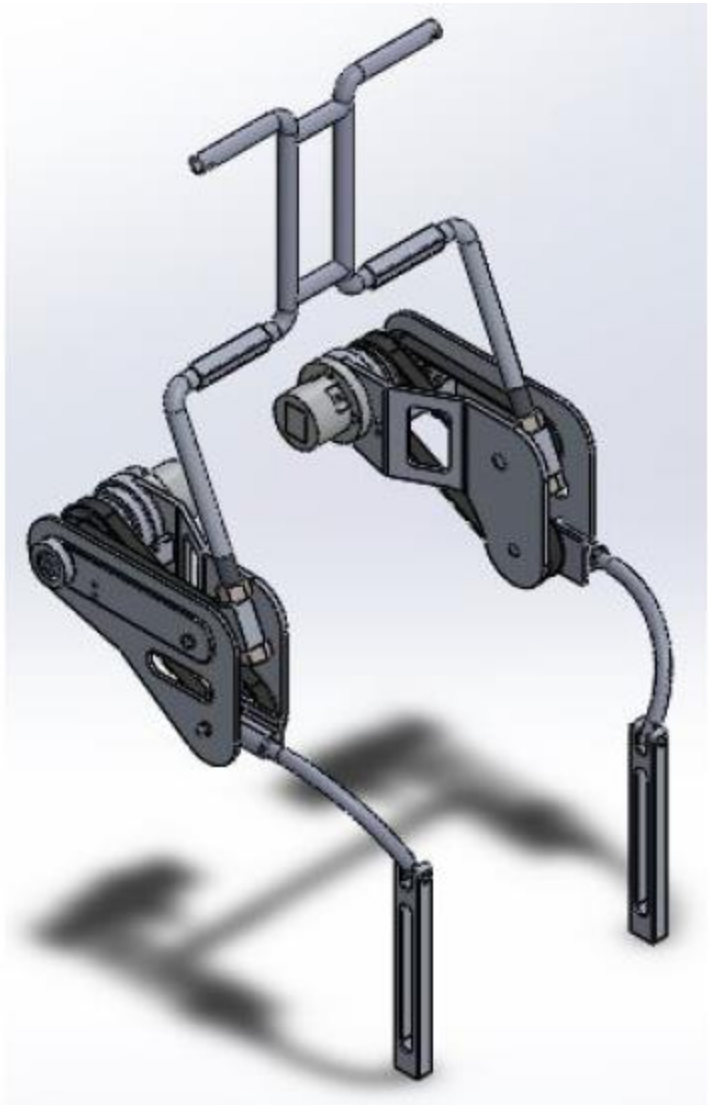


Figure 2.25: SolidWorks sketch.

Based on the physical parameters, the exoskeleton control system can be categorized into position/velocity/acceleration or torque/force. The position control scheme is commonly utilized to make sure the exoskeleton joints turn in a desired angle (for example is PID controller) Fig.2.26. The position controller is mostly implemented as low-level controller.

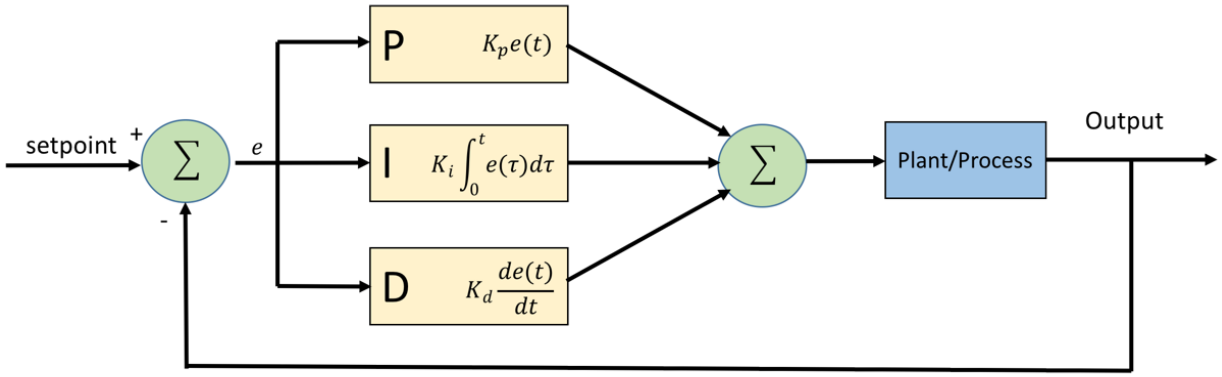


Figure 2.26: Schematic of a PID controller.

Exoskeletons existing nowadays have several implemented control strategies. After studying several control theorems as PID (Proportional, Integral, Derivative) control, Linear quadratic control based on linear quadratic regulator that the solver searches for the perfect poles and zeros according to some weighted parameters and finally, using predictive control in which a defined model is simulated and the solver computes a so called minimizer that predicts the upcoming values of certain variables and selects the value at the first time instant then neglects the rest. No exoskeleton has several control laws implemented together. However, each application has its own best control law which is chosen by simulation, validation and testing. Moreover, the cost of implementing a certain control law can be a great factor with respect to the complexity of the application.

2.3.1 PID Control Strategy

Starting with the PID control, the solver continuously computes an error value which is the difference between the reference value wanted and the output generated. Then, the controller continuously performs operations on that error value which can be just a proportionality with the error to manipulate the results to the desired output. Alternatively, the error can vary in an integral or derivative

manner with the output according to how much the following or preceding values of the output at a certain instant affects the numbers at any other instant. Below is the defining equation for any PID controller.

$$C(s) = K_p + \frac{K_i}{s} + K_d * s = K_p * (1 + \frac{1}{T_i * s} + T_d * s)$$

K_p, K_i, K_d are called proportional, integral and derivative gains respectively, while Ti and Td are the integral and derivative time constants that are defined as follows:

$$T_d = \frac{K_d}{K_p} \qquad T_i = \frac{K_p}{K_i}$$

After defining the control law, tuning the law parameters is the main basis that verifies the validity of the PID control strategy for our application. Using Simulink and thanks to the PID controller tool, all the previous values were defined obtaining very good results of the C_{SET}, C_{FB} and the error between them (Fig. 2.27, Fig.2.28).

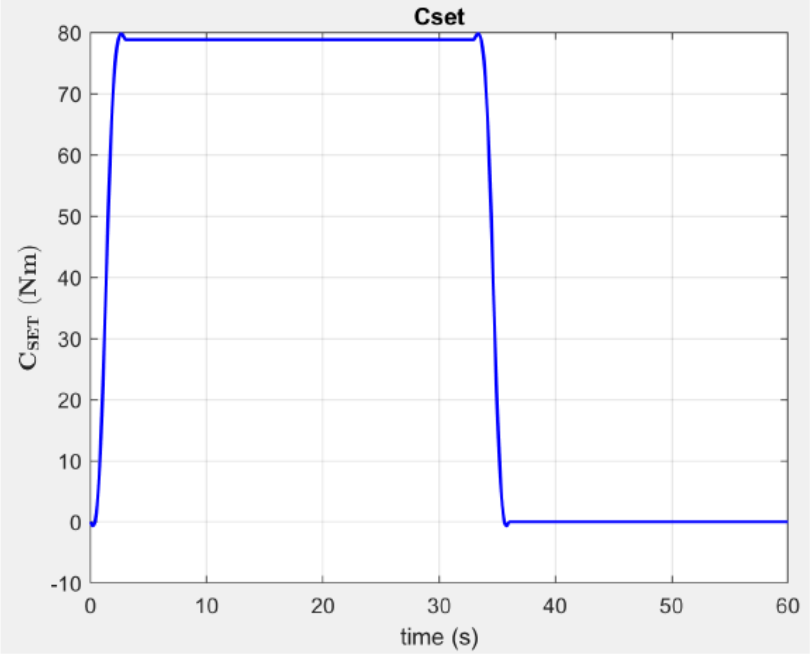


Figure 2.27: Reference Torque.

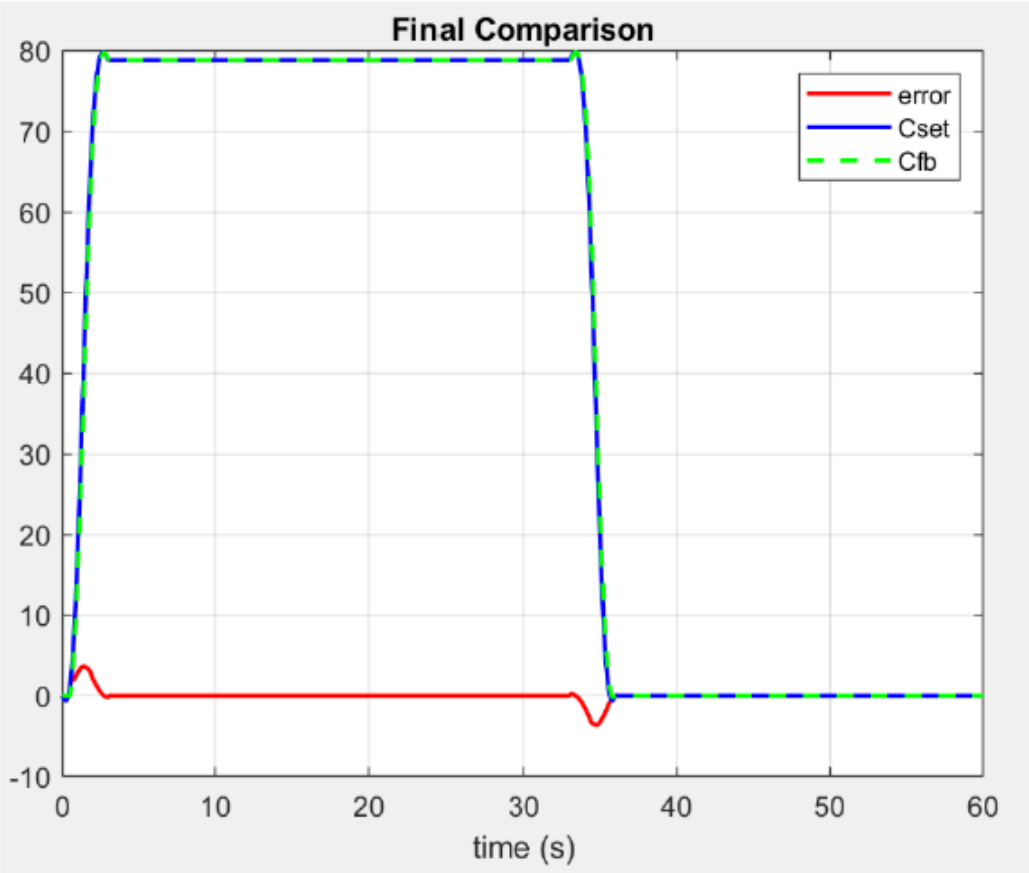


Figure 2.28: Reference, feedback and the error between them.

Analyzing the results, we obtained from PID control strategy, it is obvious that this control method is very enough for our exoskeleton application due to extremely low error which is in the transient period only. Very short time constant for our system validates the robustness of the design which leads to short settling and rise times. Moreover, the steady state tracking error is zero which concludes the validity of this control law.

Simulink results:

Maximum overshoot	**6.59%**
Rising Time	0.11 s
Settling Time	0.516 s

However, other control theories were studied as well. More complex theories yet poor outputs compared to PID control leads to the choice of PID control over the other theories.

2.4 Simscape multibody modeling

After studying simpler models for our plant and the human body simulated as well as selecting the control scheme and validating its efficiency using the simpler models on Simulink, more complex models for the system must be implemented and simulated before the execution of the exoskeleton and testing on a test bench. MATLAB offers a toolbox in Simulink that enables engineers to draw 3D models of different shapes, sizes, masses and inertia. Simscape toolbox is a very powerful tool to complete the design and simulation procedures in one place and integrate different systems together as mechanical, electrical and pneumatic.

In order to validate the modeling procedure, two stages were defined:

1. Simplified model using Simscape.
2. Imported model from SolidWorks.

2.4.1 Simplified model using Simscape

To obtain data that can be compared to validate the design of a final model we start by defining a much simpler model. Validation phase mainly tests the accuracy of the representation of the real system. It consists of assumptions that must return true after simulation the model at hand. These assumptions fall into two categories: structural assumptions and data assumptions.

Structural assumptions define how the model at hand works as well as its shape. For example: the famous model of a bull made by Picasso (Fig. 2.29) where the main concept of this example is to explain how he simplified the reality into several simple parts integrated together that show the structure and satisfy the pull shape and structure.

Figure 2.29: A bull and Picasso's "The Bull."

Similarly, a simplified model was implemented consisting of three parts only (Back frame, Leg link and Traction system) that resembles the reality of the designed exoskeleton found in Fig. 2.30.

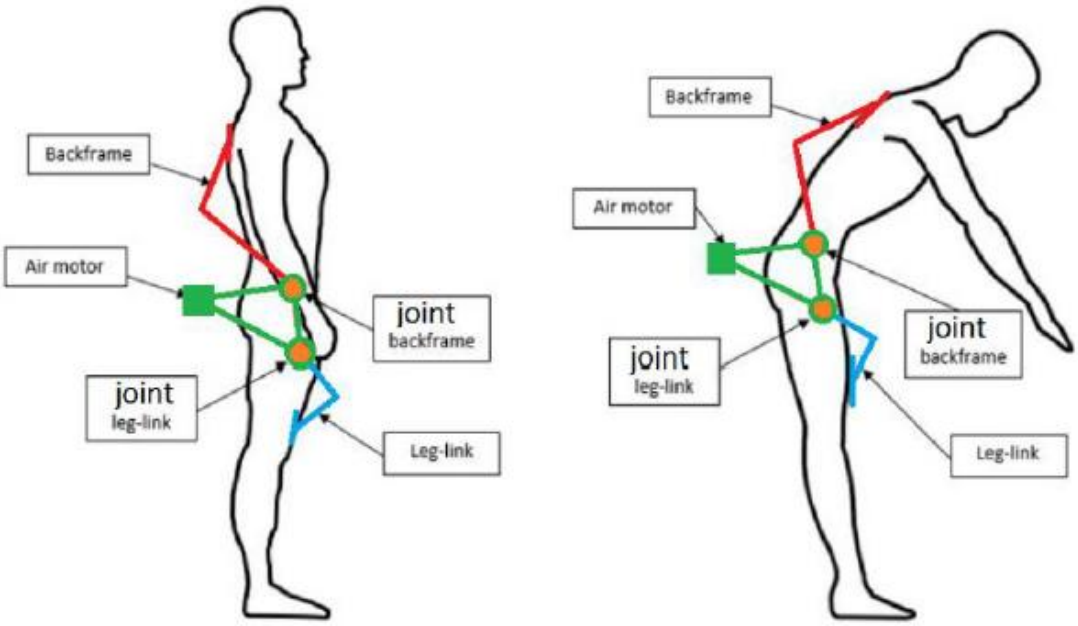


Figure 2.30: Simplified model.

After studying Simulink, Simscape, designing a model for the three components listed above becomes easy. They were integrated together using various subsystem blocks and revolute joints that implement the degrees of freedom of the robot. To model the air motor, at the driving joints, elasticity and damping in the joints are as well considered to satisfy the internal mechanics of the joints:

- Spring Stiffness = 1.13 Nm/deg

  This is the torque required to rotate the joint primitive by a unit angle.

- Damping Coefficient = 0.3 Nm/(deg/s)

  This is the torque required to maintain a constant joint primitive angular velocity between base and follower frames.

Finally, editing the joints blocks in the final Simscape simplified model (Fig. 2.31) allows us to add measurement ports where measuring the torque produced (Fig. 2.32) is possible to have the result of the simplified model in order to make a comparison with the actual model to validate our design.

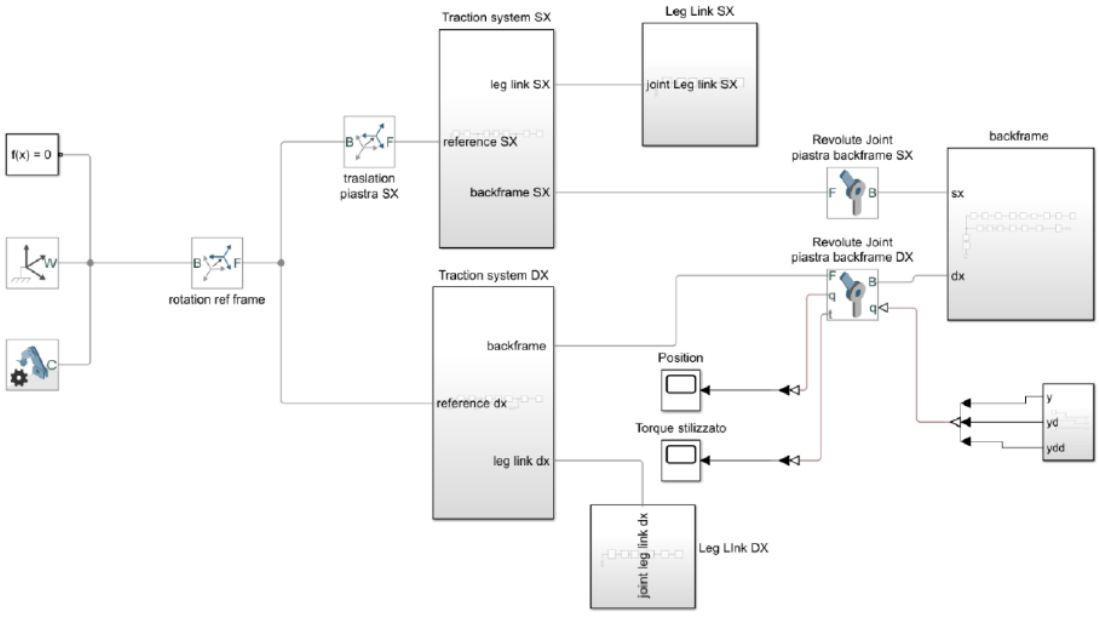


Figure 2.31: Final Simscape simplified model.

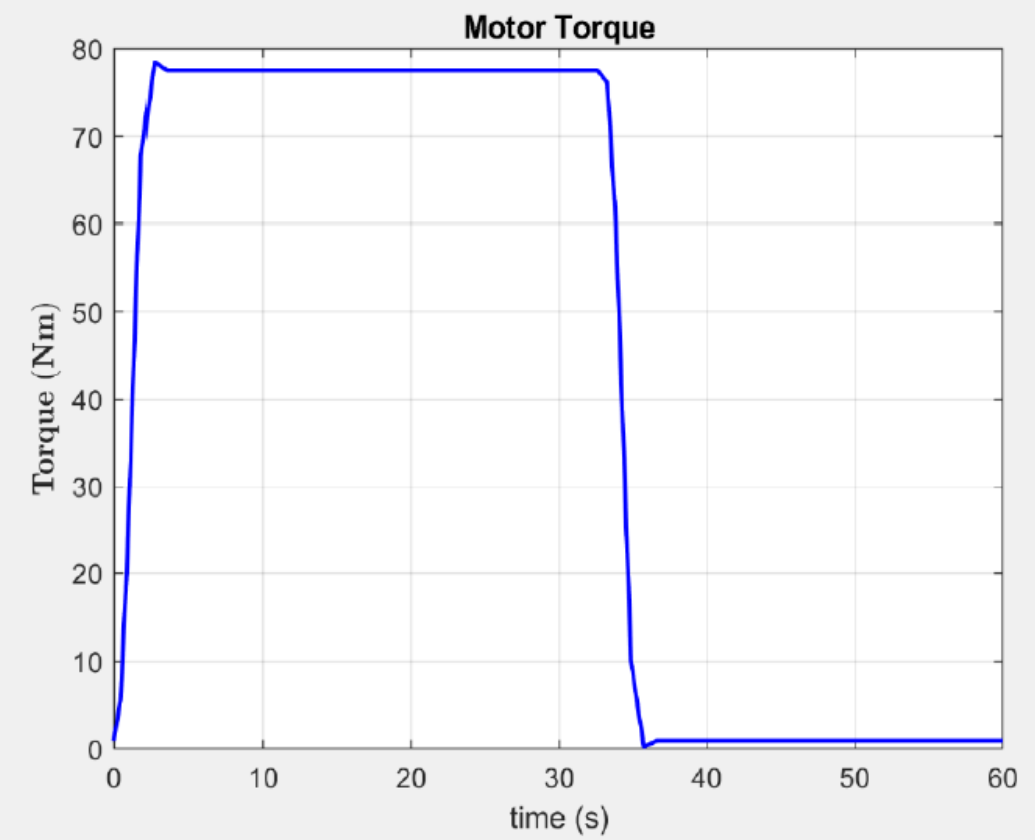


Figure 2.32: Torque produced at joints.

2.4.2 Simscape multibody imported model from Solidworks

As presented in the introduction chapter, a complete sketch of the exoskeleton is already implemented on Solidworks with different parts, pulleys, bearings and finally assembled. Using Simscape tool, it is possible to import Solidworks models into Simulink using STEP file generated from Solidworks. Thus, importing different parts separately and assembling them on Simulink with creating subsystems and using rigid transform and joints blocks. (Fig. 2.33)

Figure 2.33: Complete model of exoskeleton on Simscape after assembly.

By adding measurement port on the joint as done in the simplified model to measure the torques produced by the actuator in order to make the comparison between the torque generated by the simplified and the actual model, it was found that the torques are precisely the same (Fig. 2.34). To further validate the control scheme, a comparison, as well, is studied between the control torque and the torque generated through Simulink complete exoskeleton model. It returns some separation from the control model due to inertia consideration as well as mass, damping and stiffness study in the exoskeleton imported model from Solidworks. (Fig. 2.35)

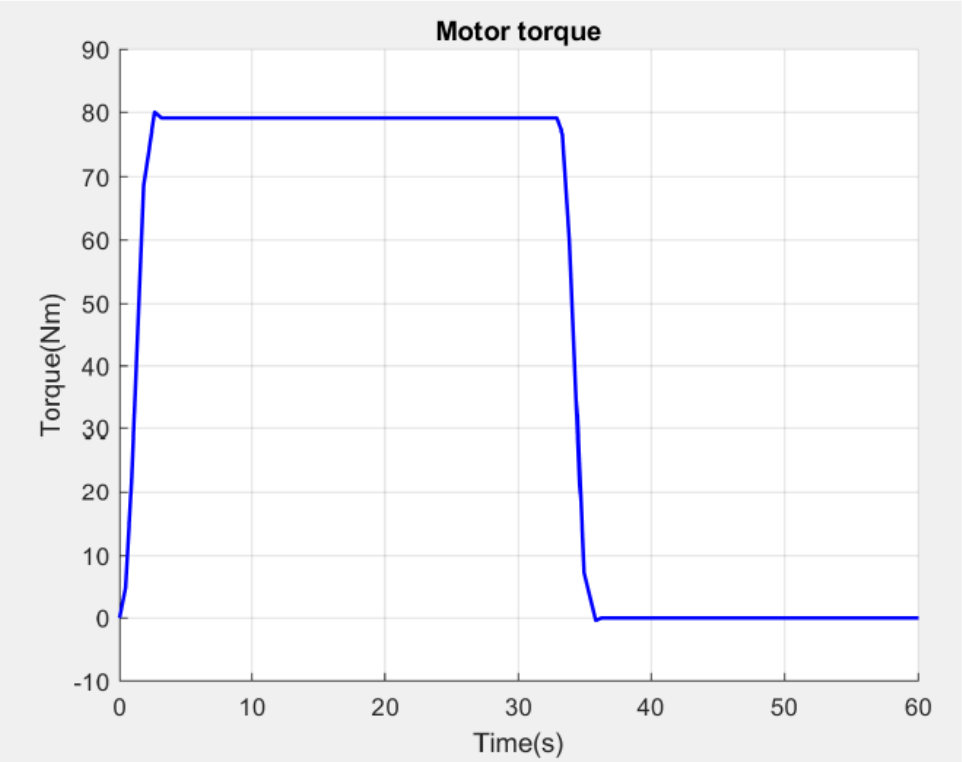


Figure 2.34: Torque produced by the imported Solidworks model.

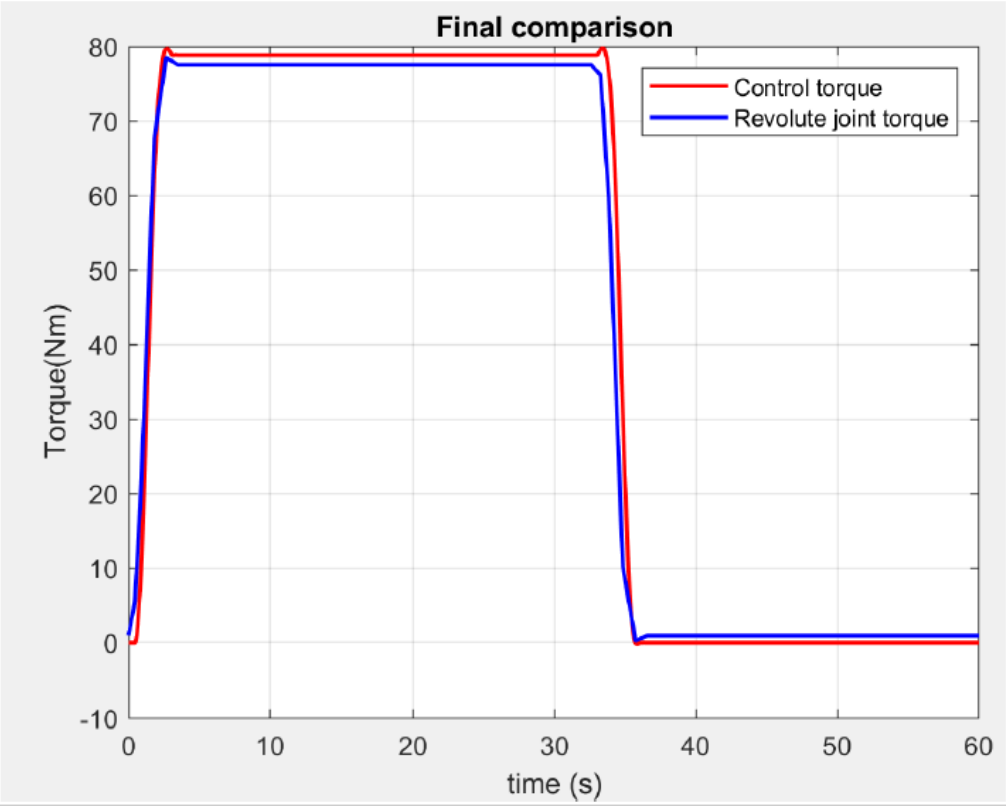


Figure 2.35: Comparison between control scheme torque and imported model torque.

2.4.3 Human body

Our exoskeleton is designed to assist 30% of the weight carried by the carrier, which is, according to standards, is equivalent to a torque of 80 Nm (as described before), while the human body can support up to 260 Nm. Under ISO-7250-2 [25], defining an average weight for a human body for an industrial worker and the weight that should be supported by him/her without medical issues, the values chosen for the exoskeleton to support weights is chosen. Thus, modeling of a human body in Solidworks and importing it as well into Simscape (Fig. 2.36) is a must to perform a complete simulation of the system and perform the tests needed in simulation (Model-in-the-loop). To perform the Model-in-the-loop procedure the human torque is defined by a formula (2.2) and then compared to the Simscape model results.

$$C_{musc} = mgL_G \sin\theta - (I + mL_G{}^2)\ddot{\theta} \qquad (2.2)$$

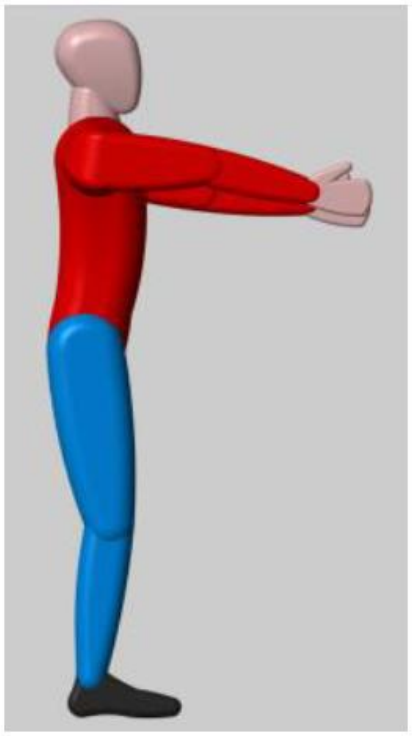


Figure 2.36: Simscape model of the human body.

The results shown (Fig. 2.37) illustrates that there are differences in the graphs due to varying design parameters in Simscape model. However, both graphs show that the human torque lies around 260 Nm during flexion. And due to the use of the exoskeleton the human body then would be supporting not 260 Nm but 180 Nm due to the exoskeleton effort of 180 Nm. (Fig. 2.38)

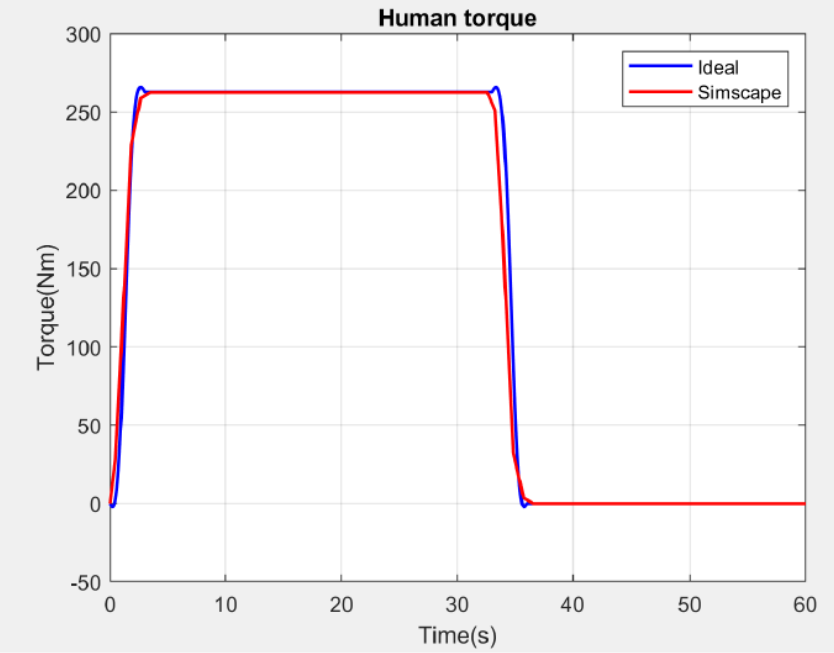


Figure: 2.37: Comparison between the torque generated by the formula and the one produced by the Simscape model.

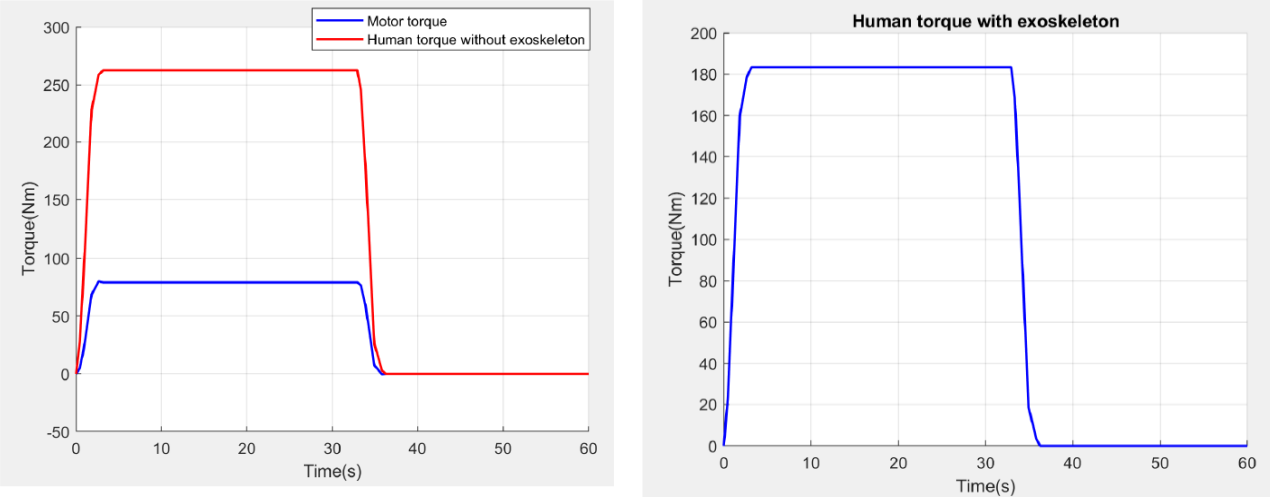


Figure 2.38: Torque produced by the air motor and the human with and without the exoskeleton.

3 Materials and methods

Defining all the hardware materials used in implementing the control law already defined and then to start the coding procedure, it is a must to define the cycle of operation of the device. Luckily, the exoskeleton operation is cyclic, and the cycles can be easily defined through simulation (Stateflow). Defining two cycles of operation depending on the bending angle that the wearer does. Each cycle has four states that the controller must navigate through to constantly be functional and assist the wearer whichever movement he/she makes. Following the Stateflow procedure, selection of several components becomes the next step. Air flow valves, angular position sensor and the micro-controller are the main components that should be selected.

3.1 Materials

3.1.1 Electronic board and development environment

Implementing the control system designed before is the main scope of this thesis study. To have a fully functional exoskeleton, software development of the control system is performed. As mentioned before, the control law used in simulation and tested compared to other control strategies is PID. Among the very wide range of development boards running on different environments, selection of an appropriate development hardware and its environment becomes a challenge. There are several manufacturers of microcontroller boards as: Texas Instruments, ARM, PIC, Arduino, AVR, National Instruments, (etc..). All these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. For instance, the National Instruments myRIO board (fig.3.39) is an expensive board that has lots of capabilities than AVR STK200-DRAGON (fig.3.40).

Figure 3.39: myRIO – 1900 board.

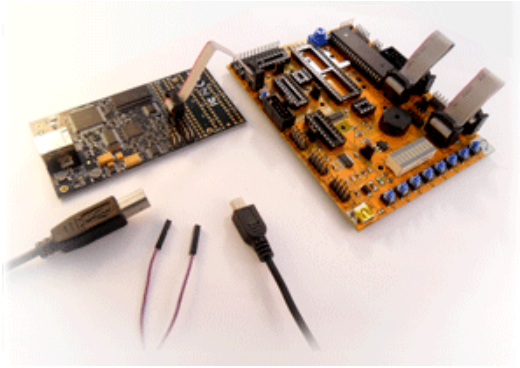


Figure 3.40: AVR STK200-DRAGON Kit.

Development boards capabilities vary very widely from one board to another and they are related to the price of the board. Following is some of these specifications:

- Real-Time capabilities.
- FPGA.
- WIFI and Bluetooth.
- Processing power.
- Storage size and type.
- Number and types of peripherals.
- Number of digital and analog pins included.
- Development environment used.
- Interactivity with the board to allow debugging easily.
- Ability to extend the capabilities of the board (ex: by using shields as in Arduino and Raspberry Pi).
- Ability to extend the capabilities of the software (ex: by making it an open source as in Arduino).
- Cross-platform capability.
- Finally, the cost.

So, the selection of the board or microcontroller is application oriented. And a variety of boards can be stuffiest for the robot to be functional. That is because, the application at hand does not need very high processing power or very large memory storage. The exoskeleton does not need to be connected to another device wirelessly as well thus, there is no use of WIFI or Bluetooth on the board. Simply, the robot has a position sensor integrated that acts as an input to the board with a simple output of a signal to act on a solenoid air flow valve to open it at the

appropriate position to control the air flow rate. To narrow the selection of development boards, myRIO and Arduino are selected and compared to choose the most appropriate with respect to specifications as well as cost.

Arduino is an Italian electronics company working on developments boards (fig.3.41), who defined Arduino as:

"Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards can read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing."

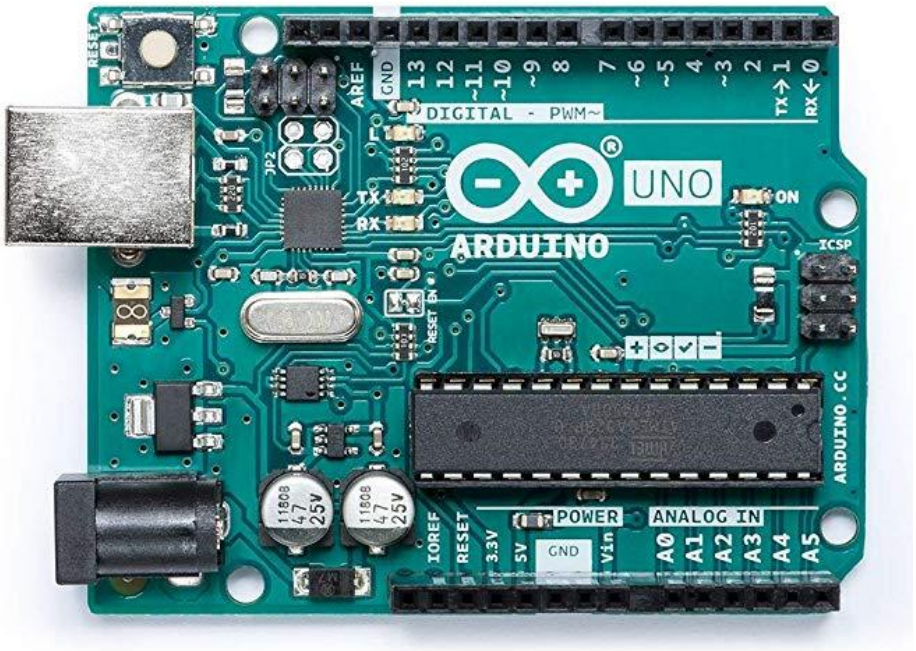


Figure 3.41: Arduino UNO (Appendix I), one of the most common Arduino boards.

The concept of the board was created in Ivrea interaction design institute. It was aimed as to encourage students to work in the field of electronics and programming. The concept prevailed which led to development of the simple concept into making several boards and shields that have a wide range of applications and advanced ones as IoT applications and embedded environments. The main peculiarities of Arduino boards are that they are open source platform where engineers and programmers can use others work and develop on it freely with no fees for the software. Secondly, the hardware is extendable using the shields or by designing their own modules as well as there is a breadboard version of the module. Moreover, it is flexible, offers a variety of digital and analog inputs, SPI and serial interface and digital and PWM outputs. It is inexpensive, around 30 euro per board and comes with free authoring software. However, myRIO board has advantages over Arduino, which cannot be called disadvantages of Arduino boards. Firstly, the myRIO runs a linux-based real-time operating system, this means you can run your LabVIEW code on it as well as it is having other utilities to allow you to configure it. myRIO has included FPGA so, the user can run extremely high-speed logic, data acquisition and filtering at 40 MHz clock rates. myRIO as well has WIFI included while in Arduino a shield must be used to provide WIFI. Moreover, it has more I/O digital and analog pins than Arduino. However, myRIO boards cost 200+ euro, which is relatively expensive.

To make an appropriate selection based on the exoskeleton robot at hand, it is simple to just follow the straight-forward quote found on the National Instruments forum website [26], that states:

"Do you write programs in C? Do you want them to run head less without needing to be connected to a PC? Do you want built in WIFI, USB Host, do you want an FPGA also written in C? Do you want to leverage the power of the Linux community? Do you have a need for vision application via a USB camera?"

Answering all these questions is simple and forward according to the application. the programing language used is C, and it will run both with and without being connected to the computer in different procedures. However, no WIFI capabilities are needed and there is no visual data needed in this application.

3.1.2 BNO055 9-axis absolute orientation BOSCH sensor

Implementing the desired control strategy there are two main concepts: firstly, controlling the torque output from the air motors and thus making a feedback signal with the acquired data from a torque transducer, secondly, instead of using a torque senor, it is possible to calculate the torque required from the air motors by measuring the angular position, velocity and acceleration of the exoskeleton at all times and feed this data into an algorithm which computes the required torque at each instant in time. Following the second conduct, research have been done to make an effective selection of a desired sensor to feed the three types of data needed. However, velocity and acceleration data can be computed from the position by taking the rate of change consecutively. This logic is correct in theory,

on the other hand, it is hard to be implemented in real time due to very small ripples in the data acquired by a position sensor will lead to very high ripples in velocity calculation and extremely un acceptable data in case of acceleration. There are several sensors in industry that can measure angles position with hall effect which depends mainly on the magnetic field of a magnet placed on the rotating shaft and the sensor facing this piece of magnet with a very small air gap up to a few millimeters. Some theses sensors are AS5600 Grove 12-bit Magnetic Rotary Position Sensor (fig.3.42), and several Penny+Giles sensors which are used in the industry of automotive applications to acquire the throttle valve position in internal combustion engines. (fig.3.43)

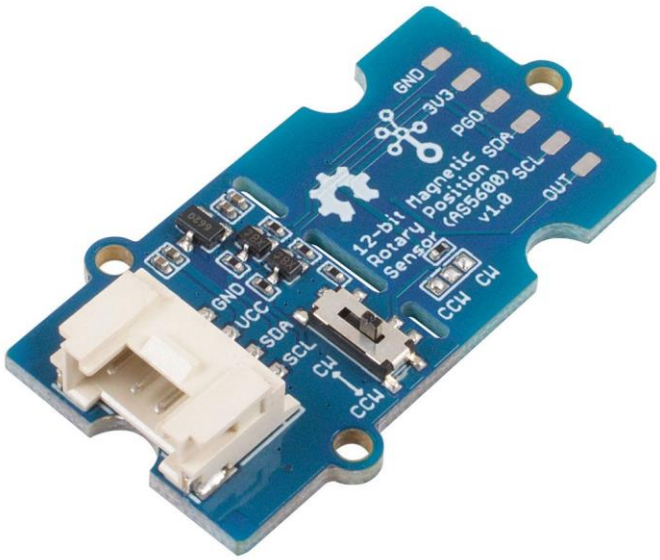
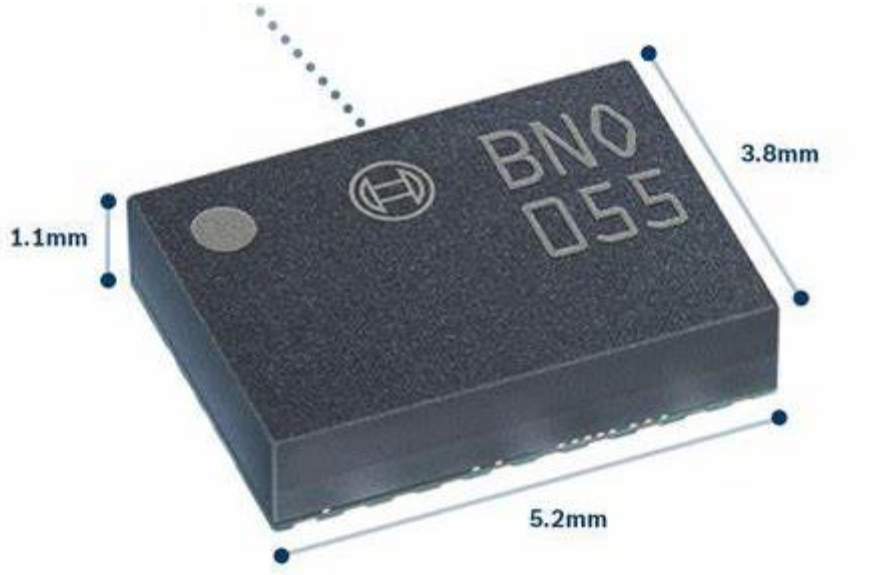


Figure 3.42: AS5600 Grove 12-bit Magnetic Rotary Position Sensor.

Figure: 3.43: NRH300DP NO-CONTACT, ROTARY POSITION SENSOR.

All the hall effect sensors have the same limitations in this thesis application: firstly, they all need to be mounted very close to a rotating shaft while there is no rotating shaft in the designed model, moreover, they only read position which is a bad conduct as explained before.

Accordingly, the sensor needed must feed angular position and velocity at least and have the acceleration part computed with no problem. In September 2013, the German company BOSCH designed an intelligent 9-axis absolute orientation sensor called



BNO055 (fig.3.44) which has integrated 3 different sensors: 3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer. The sensor has intelligent data fusion algorithms that uses all data inputs and feed as output quaternions, linear acceleration, rotation, gravity and robust heading. It is used in several applications as: automotive to feed the orientation and acceleration of a vehicle, robotics applications and even gaming joysticks to define heading of the controller in the gamer hands.

Figure 3.44: BOSCH bno055 9-axis intelligent absolute orientation sensor.

Moreover, to interface this sensor with Arduino, Adafruit designed a board that integrates this sensor along with a controller to feed data into Arduino using I^2C communication protocol. (fig.3.45)

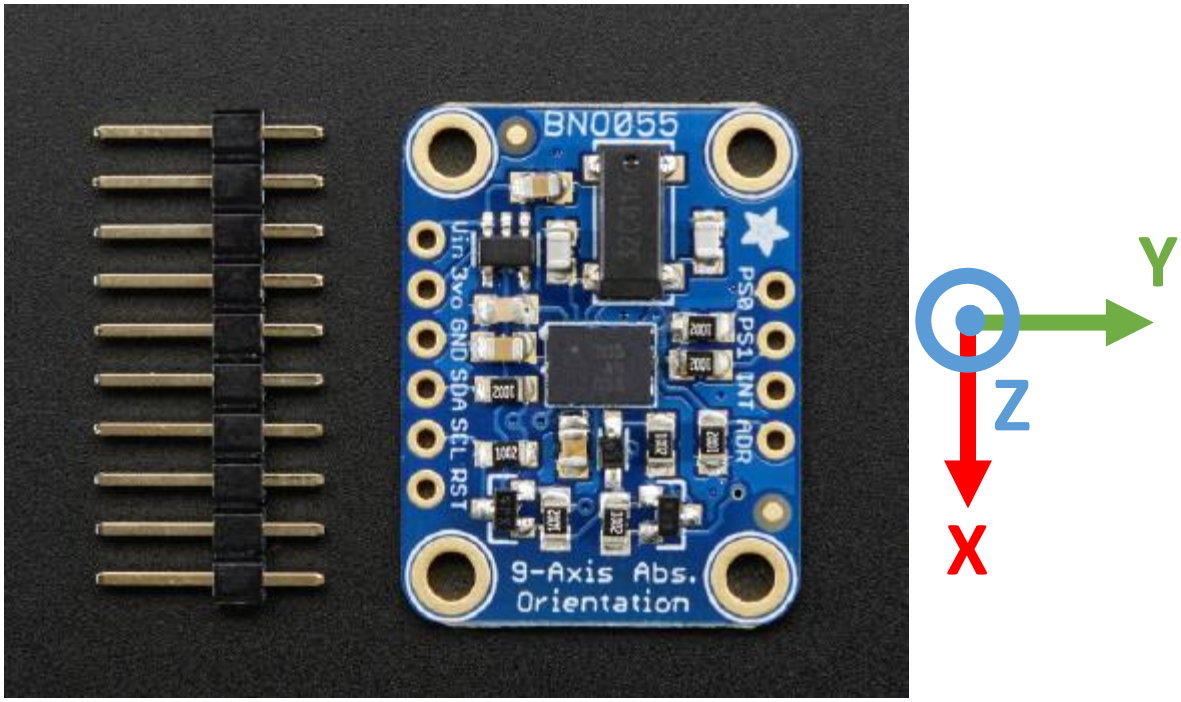


Figure 3.45: Adafruit BNO055 absolute orientation sensor (Appendix J).

According to the right-hand rule, the three coordinates are defined. The pinout of this board is defined as follows:

1) VIN: 3.3-5.0V power supply input.
2) 3VO: 3.3V output from the on-board linear voltage regulator, you can grab up to about 50mA as necessary.
3) GND: The common/GND pin for power and logic.
4) SCL - I2C clock pin, connect to your microcontrollers I2C clock line. This pin can be used with 3V or 5V logic (there's a 10K pullup on this pin).
5) SDA - I2C data pin, connect to your microcontrollers I2C data line. This pin can be used with 3V or 5V logic (there's a 10K pullup on this pin).
6) RST: Hardware reset pin. Set this pin low then high to cause a reset on the sensor. This pin is 5V safe.

7) INT: The HW interrupt output pin, which can be configured to generate an interrupt signal when certain events occur like movement detected by the accelerometer, etc. (not currently supported in the Adafruit library, but the chip and HW can generate this signal). The voltage level out is 3V.
8) ADR: Set this pin high to change the default I2C address for the BNO055 if you need to connect two ICs on the same I2C bus. The default address is 0x28. If this pin is connected to 3V, the address will be 0x29.
9) PS0 and PS1: These pins can be used to change the mode of the device (it can also do HID-I2C and UART) and also are provided in case Bosch provides a firmware update at some point for the ARM Cortex M0 MCU inside the sensor. They should normally be left unconnected.

The Adafruit company provides an Arduino library to interface the BNO055 sensor. Thus, using the sensor fusion algorithm to measure angular position, using the gyroscope to acquire angular velocity (in radians/s) and, finally, computing the angular acceleration by taking the time derivative of the gyroscope data in coding as the baud rate is set to 115200 bauds per second which implies that 115200 signal level changes are happening within a second. Finally, two parts of this sensor is used to measure the angles in the lumbar and hip joints respectively. Thus, using two sensors with different I^2C addresses which is supported by this board type by applying 3.3V on the ADR pin it changes the address from 0x28 to 0x29.

3.1.3 Pressure regulator

The Arduino and the whole control logic are intended to control the pneumatic pressure into the air motors to manipulate the torque generated. This means that the actuator at hand is a pressure regulator which can be operated with the output voltage range of the Arduino PWM outputs. However, the pressure regulator

available in the fluid automation lab is the Festo proportional pressure regulators VPPE with display of type VPPE-3-1-1/8-10-010-E1 (fig.3.46) which, according to the data sheet, is proportional pressure regulator with switching valve head, 3-way valve, with a maximum pressure up to 10 bar which can be decreased in settings, with an operating signal which is analog voltage in the range of 10 VDC, a voltage supply of 24 VDC and finally with a 7-segment LED display.

Figure 3.46: Festo proportional pressure regulators with display of type VPPE-3-1-1/8-10-010-E1 (Appendix K)

From the electric point of view, using a 24 VDC supply, available in the lab, the regulator is operated. However, the input control signal is needed to be 0-10 VDC which correspond to 0-10 bar of regulated pressure. However, the pneumatic supply to the exoskeleton is going to be only up to 6 bar (relative) and given the Arduino analog output as a 0-5 V PWM signal raises the urge to design a filter to provide a pure digital signal with no PWM components and an amplifier to raise the output of the Arduino from 0-5 V to 0-10 VDC.

3.1.4 Electronic board

Using LTspice simulation tool, a model of the intermediate stage between Arduino and the Festo regulator is simulated with components available in the lab (fig.3.47)

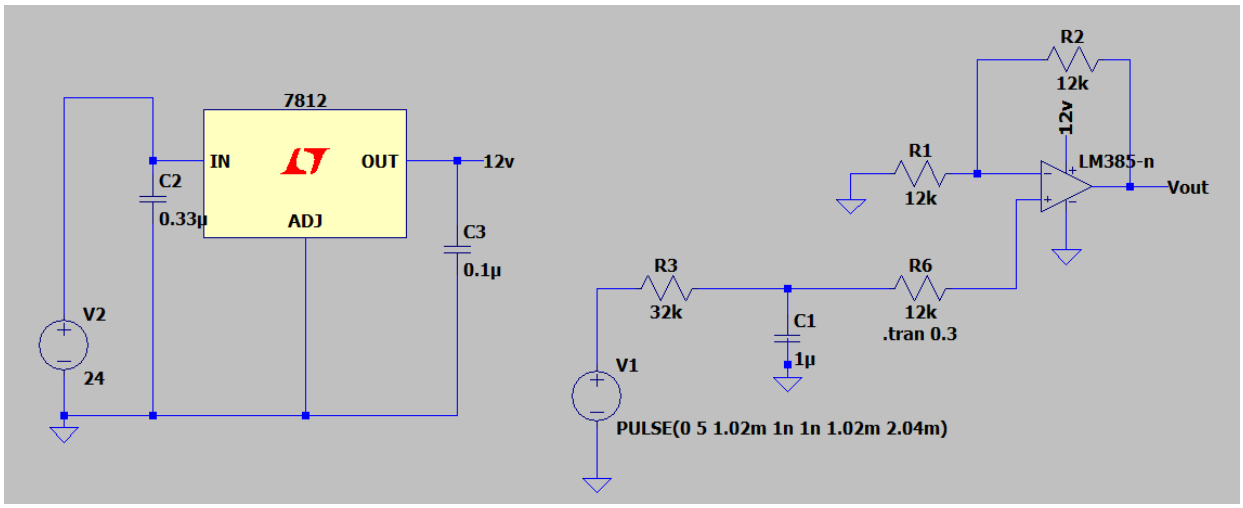


Figure 3.47: LTspice simulation of the electronic board.

Using the available lab equipment, a 24 VDC Instek GPS-2303 power supply (Fig.3.48) is used to operate the control circuit which is represented by V2 in figure 3.47. However, to operate the Arduino UNO board (pin Vin) and operational amplifier LM358-n (Appendix A) used (fig.3.49) a 12 VDC supply is required.

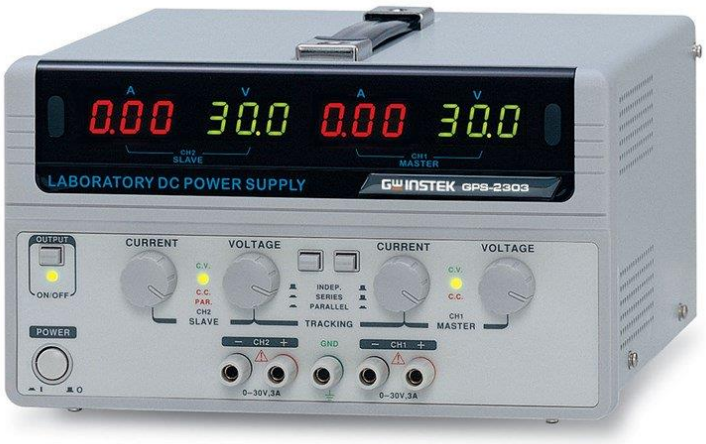


Figure 3.48: Instek GPS-2303 0-30 VDC power supply.

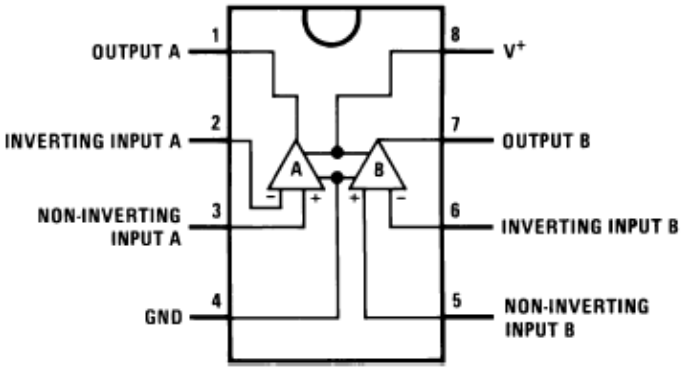


Figure: 3.49: LM358-n operational amplifier and the pinout data.

To step down the voltage a pressure regulator of type TS7812 (Appendix B) (fig.3.50) is used. According to the datasheet of the op-amp, it is recommended to use on the supply line 0.33 µF and on the output line 0.1 µF to ensure decoupling of supply. The LM358 series are operational amplifiers which can operate with only a single power supply voltage, have true-differential inputs, and remain in the linear mode with an input common-mode voltage of 0 VDC.

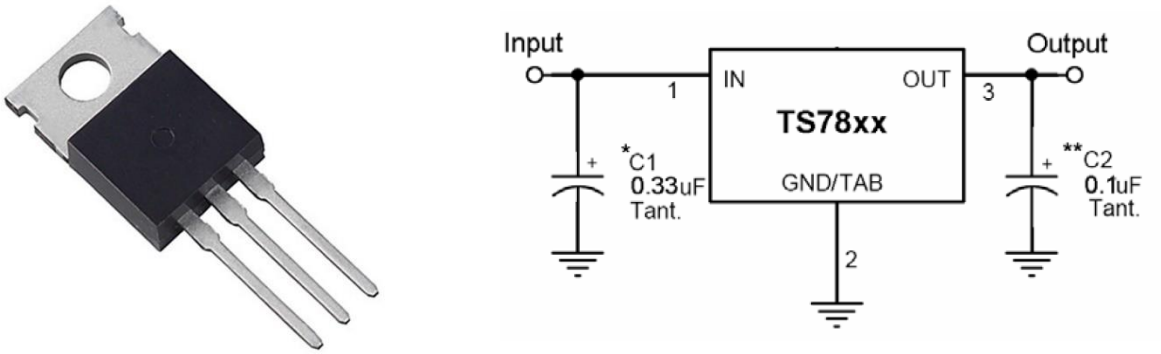


Figure 3.50: TS7812 Voltage regulator and pinout

The Arduino UNO output signal is a 0-5V PWM signal (0-255 in code) which is used to operate the Festo proportional valve, However the valve is operated with 0-10 VDC signal. So, the Arduino output PWM is fed into an RC filter to remove the ripples and then to a non-inverting amplifier with a ratio at 2 to step up the voltage into twice its value. This means that when applying a signal in Arduino code with a value of 255 the voltage output on the proportional valve is going to be 10 VDC. The designed RC filter should provide fast response with as low ripples as possible, however, some ripples will not damage the Festo valve as it was tested to PWM signals and It works perfectly. Studying the transient response of the capacitor the R and C values were chosen. The output is represented with a transfer function of:

$$G(s) = \frac{31.25}{s + 31.25}$$

The cutoff frequency is chosen to be:

$$f_C = 4.974\ Hz$$

Final max ripple voltage:

$$\Delta V_{PK-PK} = 79\ mV$$

Settling time (0-90% V_{in}):

$$T_s = 73\ ms$$

The step response of a 0-5 V PWM with 50% duty cycle is shown. (fig.3.51)

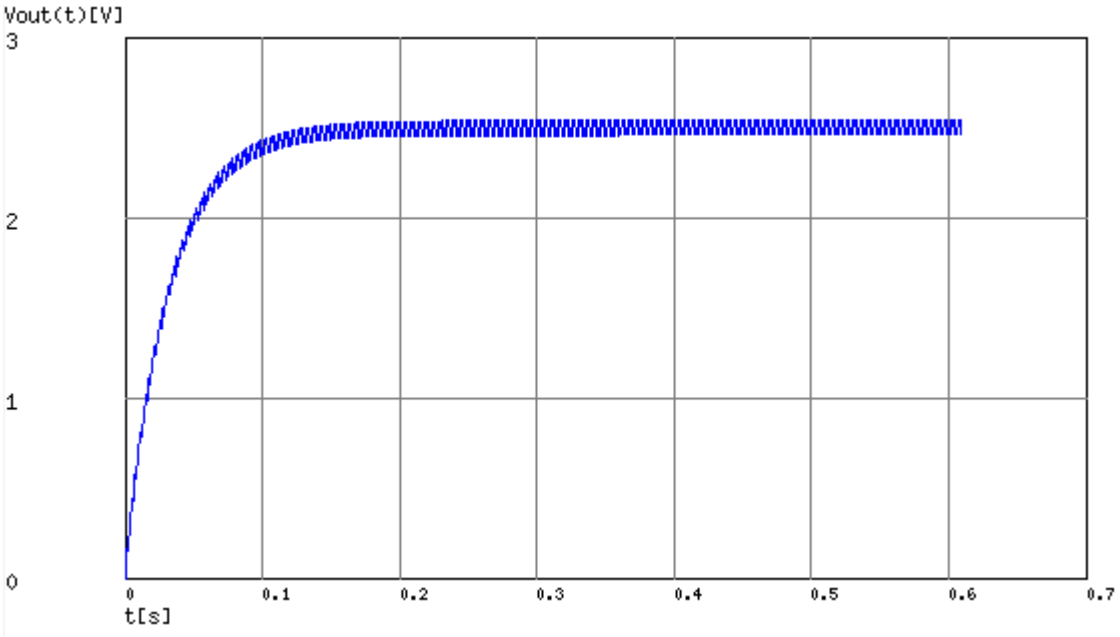


Figure 3.51: step response of 0-5 V PWM with 50% duty cycle.

Finally, the op-amp is operated with a single supply not a differential one as the application at hand is only DC and any negative signal is not needed and should be removed. The overall circuit shown in figure 3.49 is simulated with the following output. (fig.3.52)

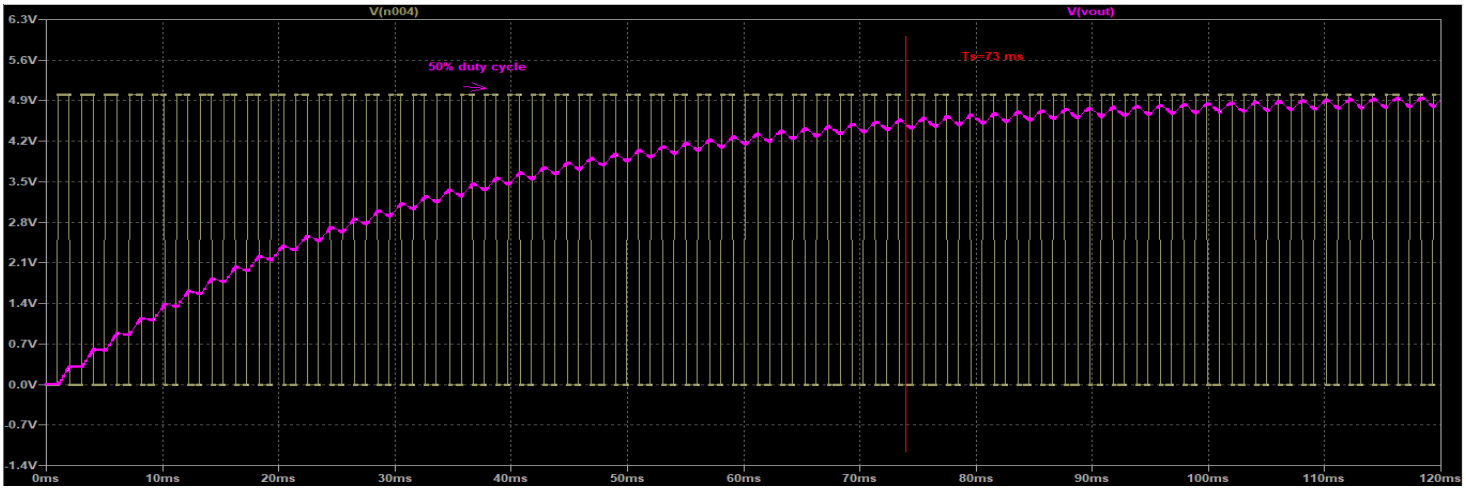


Figure: 3.52: comparing the PWM signal from Arduino with 50% duty cycle with the output from the designed electronic board.

Following the simulation, the circuit was implemented on a bread board and using electronic components found in the lab and connecting the Arduino board as input

and Festo valve as output while picking the signal to the valve on an oscilloscope. The Arduino input was set to be a stepping increase in the duty cycle to see all the outputs in real time. (fig.3.53) (fig.3.54)

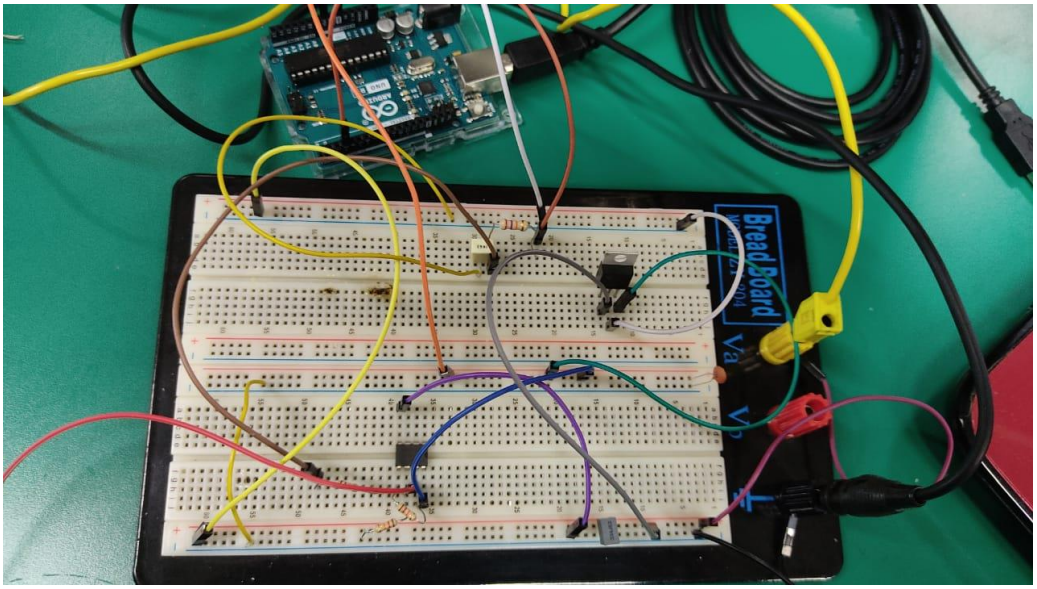


Figure 3.53: electronic circuit connected on bread board and Arduino integrated.

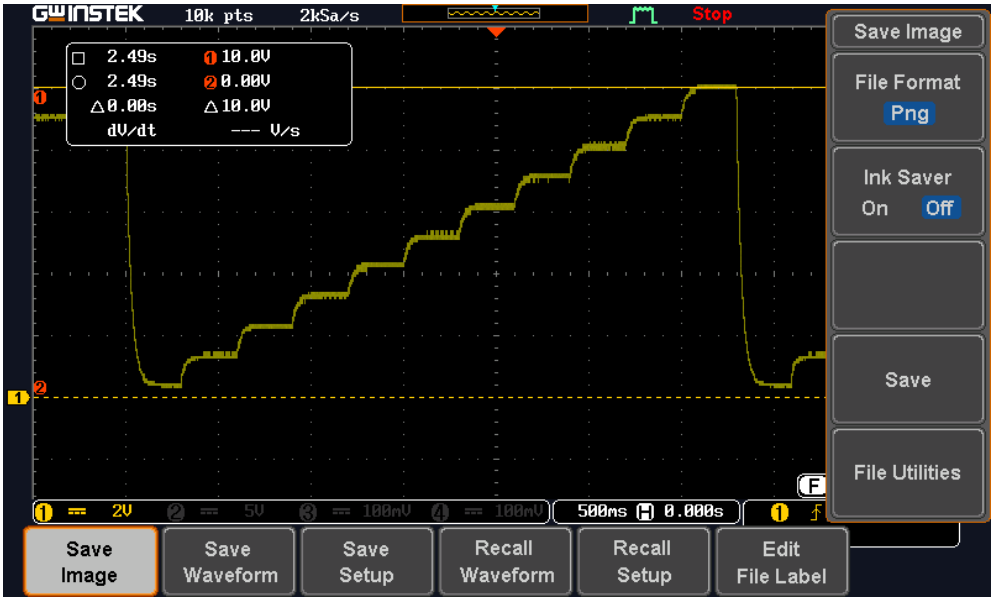


Figure 3.54: stepping duty cycle output from electronic board.

The oscilloscope shows an offset in the voltage which can be read in zero step (zero duty cycle), however, this offset is not from the board design but can be from the oscilloscope itself or from the valve as when zero voltage to the valve the output had the same offset of 3.1 bar. Taking table (Table 3.2) shows voltage inputs and

pressure outputs to be used to calibrate the Festo valve which, according to the datasheet, is linear.

V(Arduino)	V(Festo)	Bar	V(Arduino)	V(Festo)	Bar
0	0.31	0.32	**130**	5.41	5.36
5	0.51	0.51	**135**	5.61	5.57
10	0.7	0.685	**140**	5.8	5.74
15	0.9	0.905	**145**	6	5.93
20	1.09	1.1	**150**	6.19	6.06
25	1.29	1.295	**155**	6.39	6.06
30	1.485	1.49	**160**	6.59	6.06
35	1.68	1.7	**165**	6.78	6.06
40	1.88	1.85	**170**	6.87	6.06
45	2.07	2.065	**175**	7.07	6.06
50	2.27	2.25	**180**	7.26	6.06
55	2.47	2.47	**185**	7.45	6.06
60	2.66	2.625	**190**	7.64	6.06
65	2.86	2.82	**195**	7.84	6.06
70	3.06	3.025	**200**	8.03	6.06
75	3.25	3.245	**205**	8.22	6.06
80	3.45	3.42	**210**	8.41	6.06
85	3.645	3.62	**215**	8.6	6.06
90	3.84	3.82	**220**	8.8	6.06
95	4.04	4.03	**225**	8.99	6.06
100	4.23	4.195	**230**	9.19	6.06
105	4.43	4.39	**235**	9.38	6.06
110	4.63	4.58	**240**	9.58	6.06
115	4.82	4.8	**245**	9.79	6.06
120	5.02	4.97	**250**	9.97	6.06
125	5.21	5.165	**255**	10.17	6.06

Table 3.3: Arduino input data to the valve and pressure.

Analyzing the table data, the Arduino PWM output is nearly linear with the pressure from the valve except for the zero voltage. Moreover, the maximum pressure output on the valve is 6.06 bar as the supply was set to this maximum value and this is according to the application.

Using python linear regression tool, a perfect fit line can be computed (Appendix C) to map the signal from Arduino to the pressure required on the valve (fig.3.55).

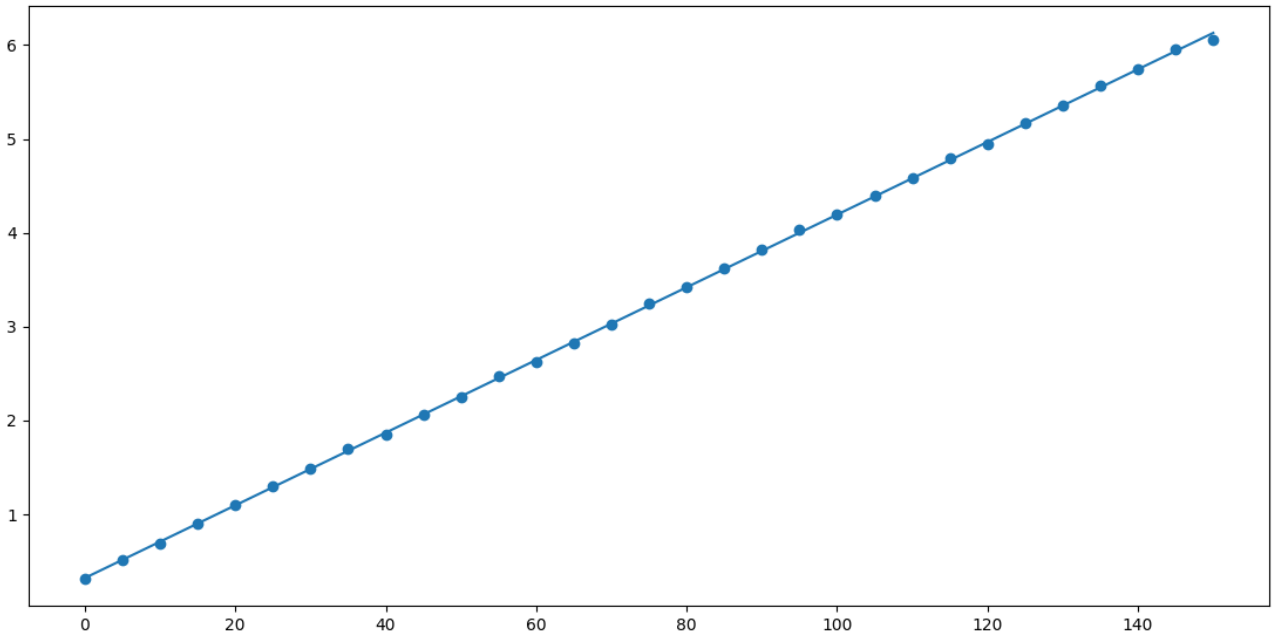


Figure 3.55: Python linear regression output (x-axis is Arduino code data; y-axis is valve Pressure).

Analyzing the line, it has the following line equation:

$$\boldsymbol{Pressure = 0.038715 * A_{code} + 0.322379}$$

Moreover, the regression correctness as acquired from the SciPy library in python, with 1 as highly correct and zero as incorrect, is 0.9999.

3.2 Methods

3.2.1 Stateflow

Stateflow has been chosen to manage the transition between the four states mentioned before which are: Resting, Bending, Working and Lifting phases. The design should accommodate for the slight bending angle due to the wearer movements as walking. It should also accommodate for reverse cycle of two phases as the wearer can bend and then not flex till the bending angle is zero degrees, instead, the exoskeleton can be in the bending phase followed by the lifting phase then bending again instead of resting. All these phases and the transition among them with valid conditions is illustrated through Stateflow.

In Stateflow we should define:

1. Input signals: as: Power switches and Angular position, velocity and acceleration.
2. Output signals: Displays to show the angular quantities computed.
3. Diagram that illustrates the states and transitions. (fig.3.56)

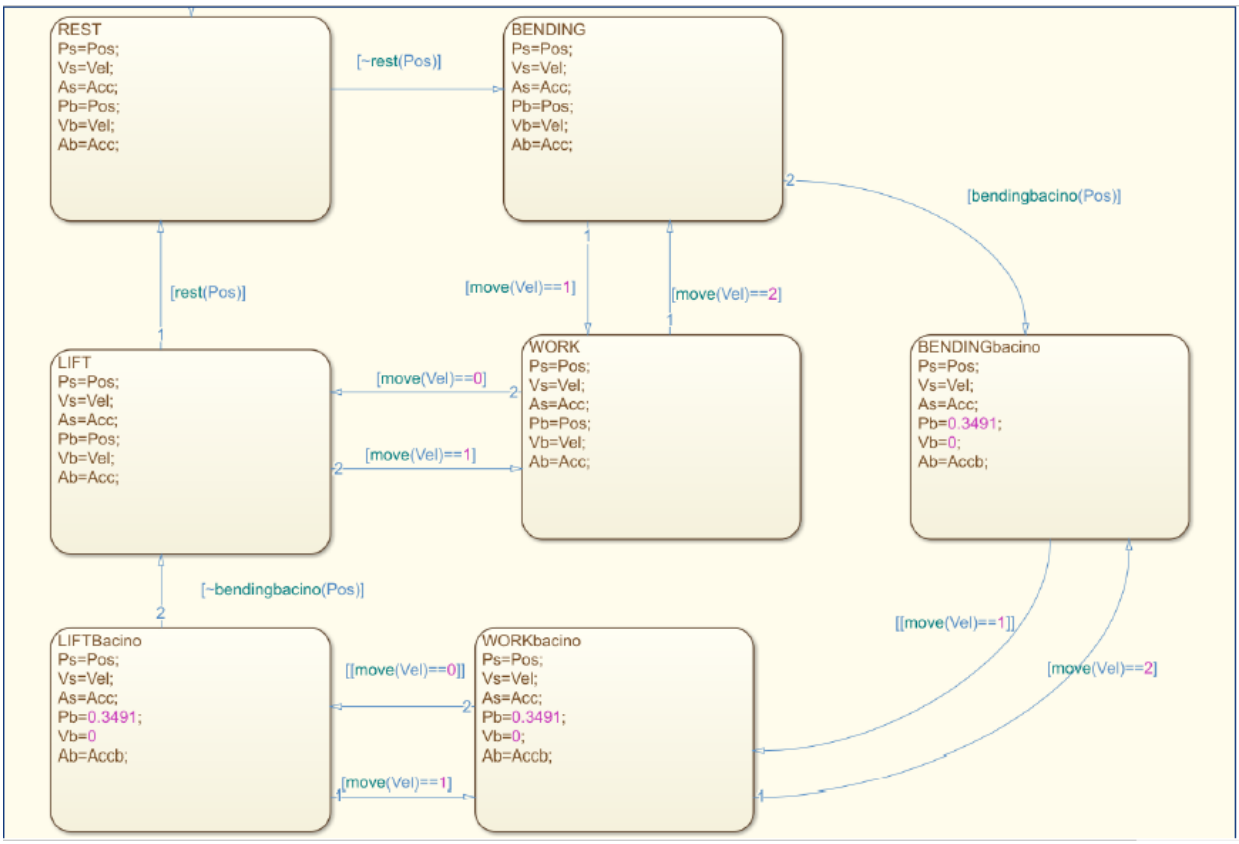


Figure 3.56: complete Stateflow scheme.

Defining the different phases as follows:

1. REST Phase: when the bacino angle (Pb) is less than 5° so the wearer is standing up right and the tolerance of 5° is to accommodate for simple movements and bending during walking or tilting with a very small angle.
2. BENDING Phase: Can be entered when the bending angle exceeds 5° but less than 20°. This is due to the existence of two separate angles in the pelvic region: P_b (bacino) and P_s (structure). Where P_b has a maximum of 20° while P_s is 70° with respect to P_b. this phase is followed by one of the two phases: either WORK phase or BENDINGbacino phase.
3. WORK Phase: this phase is entered either from the BENDING or LIFT phases. In case of BENDING, WORK phase is established only when the velocity of

the bacino joint is zero (this means that the wearer was bending and stopped in order to stand up again or even return by a short angle and thus needs to be assisted). While in case of LIFT, this phase is entered when the velocity of the bacino joint is zero as well, but the difference is that there is a flag that states that the cycle passes through the WORK phase before.

4. LIFT Phase: It is entered also by two conditions. Firstly, from WORK (cycle 1), when the velocity of the bacino joints is less than zero (i.e. the wearer is moving in the other direction and standing up). Finally, from LIFTbacino (cycle 2), when the bacino angle is between 5° and 20° as well as a flag to indicate that the cycle passed through the WORKbacino.
5. BENDINGbacino Phase: it can be entered either from BENDING or WORKbacino. The first occurs when the bacino angle is 20° while the bacino joint velocity is still greater than zero so the wearer is still bending and is going to bend the structure angle (P_s). while the second possibility is when the structure angle velocity is greater than zero and there is a flag that indicates the cycle passed through BENDINGbacino.
6. WORKbacino Phase: Can be entered either from BENDINGbacino or LIFTbacino. The first one when the structure joint velocity is zero as well as, a flag that indicated that the cycle entered the BENDINGbacino. While the second is when the structure joint angle is less than 70°, velocity is zero and, finally, a flag that indicates that the cycle entered the LIFTbacino.
7. LIFTbacino: Is entered if and only if the structure angle is equal to 70° and the velocity is less than zero.

There are wo cycles defined in the previous phases:

1. Cycle 1: Defined through phases 1 through 4 and finished by 1 again.

2. Cycle 2: Defined through the phases 1 -> 2 -> 5 -> 6 -> 7 -> 1.

Where the air motor should operate during these phases only: 3,4,6,7.

In the following diagram are the functions used in Stateflow to define the transition conditions between different phases (fig.3.57):

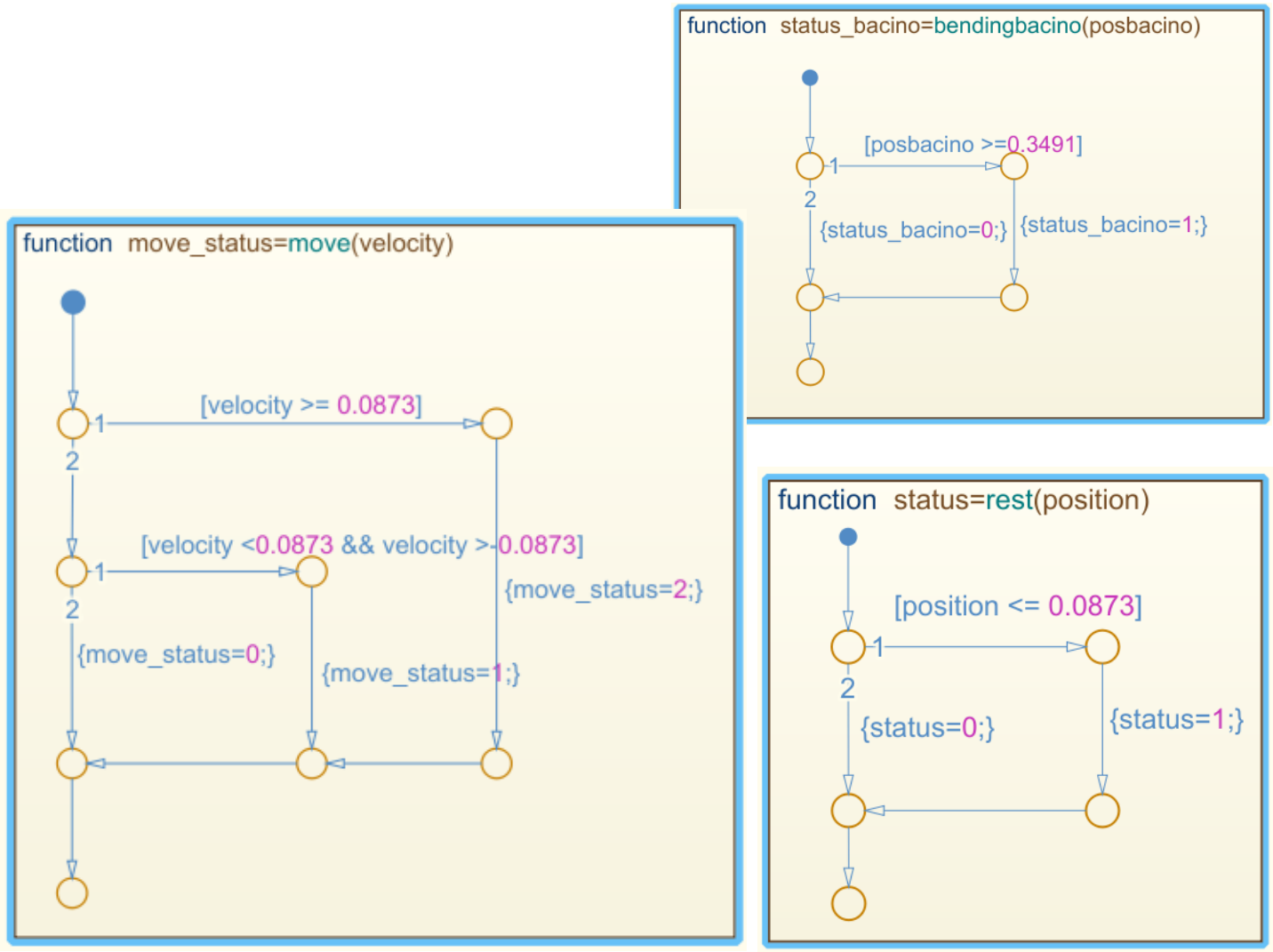


Figure 3.57: Functions used to define conditions to translate between phases.

3.2.2 Logical algorithm

This thesis mainly focuses on the implementation of the control strategy already defined in previous colleague's work. Previously, it was defined the states of operation of the exoskeleton through Stateflow. Arduino is coded using the Arduino IDE software where the programming language is C++. The concept of the Stateflow diagram is to simulate the model and to generate the logical representation of the coding procedure. Thus, using the Stateflow diagram the states were defined into logical data that can be code with C++ easily(fig.3.58). Maintaining the outputs of the logical circuit as only one output which is the signal to the Festo valve, the lift phase defines that this is the only state in which the valve is operating with a pressure corresponding to the bending angle of the user which is then transferred into torque on the air motor.

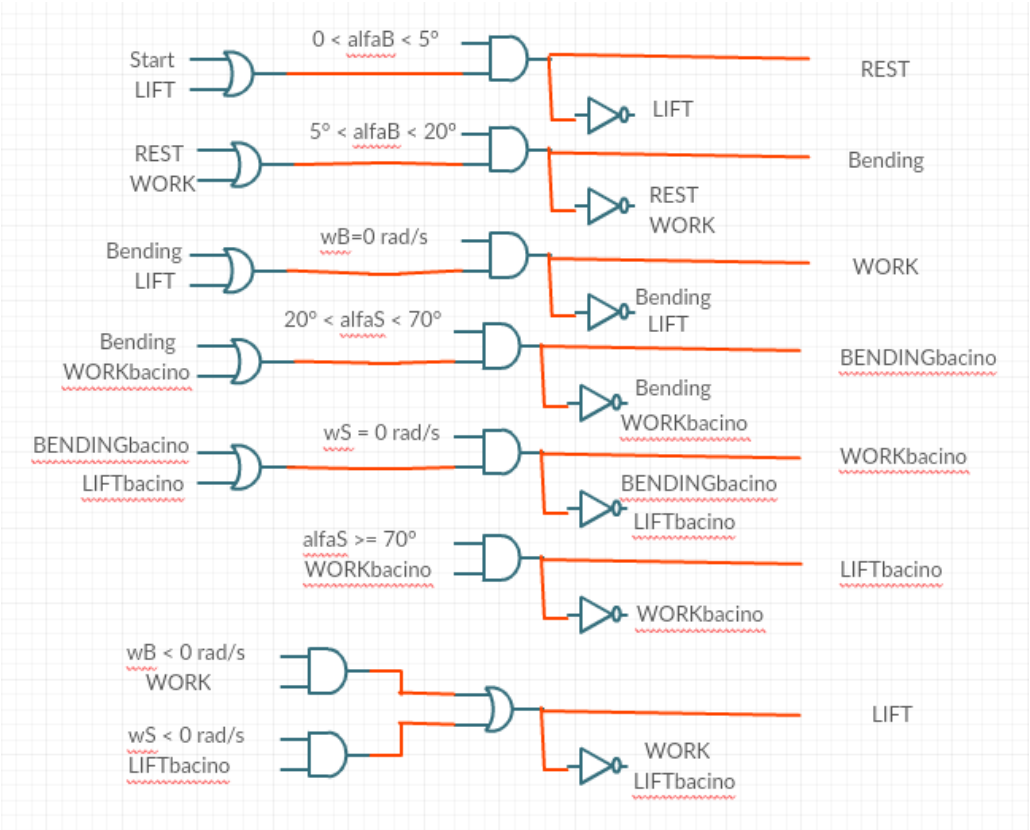


Figure 3.58: logical control representing the sates of operation.

However, the main difference between this logical representation of the states and the C++ code designed (Appendix D) is that the states are only stored in one string which unlatches when the state changes. Moreover, the angles from the two sensors must be read each cycle not with an interrupt that feeds the angle at constant time stamps and this is because of the very large mechanical response delay of the Festo valve which can be nearly 0.5 second. However, the execution of the code is at 115200 baud rate which will be useful not only in rapid execution of the control strategy but also to log data into simulation environments to monitor the operation of the device.

3.2.3 Measuring angular data from 9 DOF sensor

The underhand BNO055 sensor is a 9 DOF sensor as described before. It has integrated an accelerometer, gyroscope and magnetometer which sends acceleration, angular velocity, magnetic field strength data respectively. BOSCH company made an algorithm to fuse the data from these sensors together to provide other types of outputs as well. For instance, the fusion algorithm provides absolute orientation in space for the three axes depending on the sensor orientation in space. Moreover, they provide quaternion data as well as gravity vector on the three axes.

In the application at hand, only one axis of rotation is required as the exoskeleton is a 1 DOF structure at each joint with angular limits (0-20° at the lower joint and 20°-70° at the upper joint). This implies using the data fusion orientation data to provide the angular position of the sensor; however, this sensor is as intelligent as it is designed to recalculate the offsets from the vertical and horizontal plane each time the sensor is operated. This feature is not recommended to be used in this application as the user should be perfectly standing vertically each time the device

is operated. Moreover, this feature is automatically set in the sensor and cannot be switched off.

Instead, it's better to use the fusion gravity vector data to compute the orientation at a certain instant which is better than using the accelerometer data as the accelerometer data changes not only with gravitational acceleration but also with any acceleration component along the axis referred to.

Solving the trigonometric problem (fig.3.59)with the sensor mounted in the vertical position if the positive x-axis of the sensor is pointing in the positive gravity vector, the angular position can be precisely computed with only a single limitation that the measured angle is only between -90° up to 90° for if it exceeds 90° in a certain direction it goes to -90° not 91°.

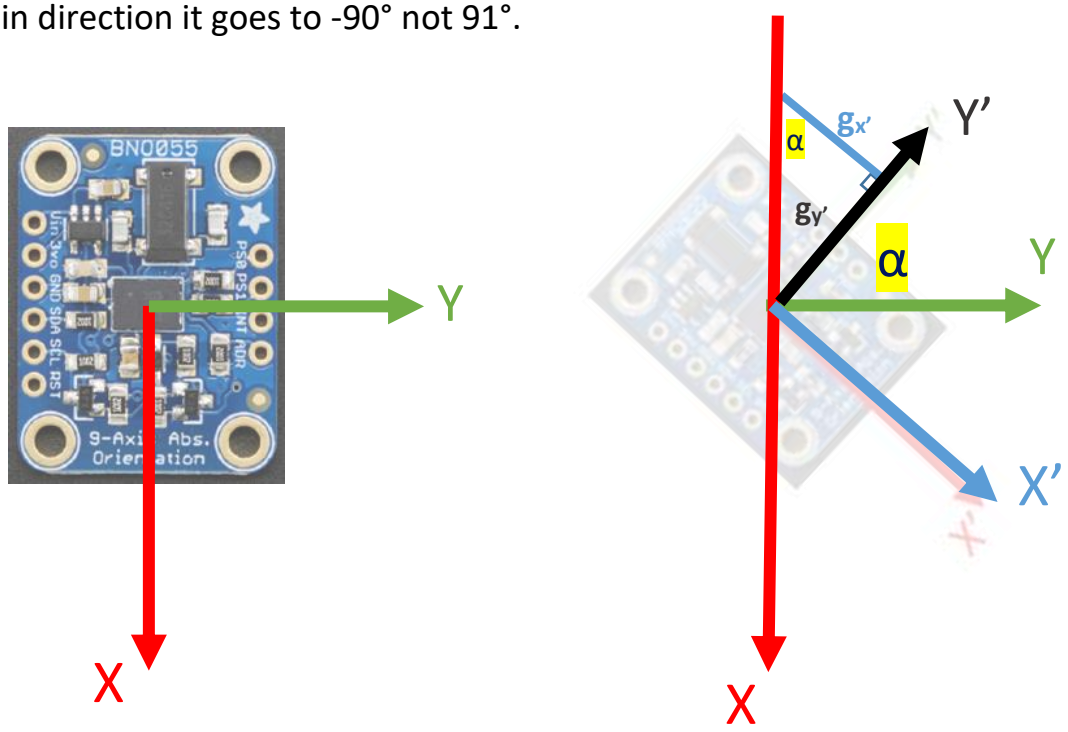


Figure 3.59: Counterclockwise graphical orientation of the sensor.

By taking the tangent of the angle α from the sketch, the heading can be computed:

$$\tan \boldsymbol{\alpha} = \frac{\boldsymbol{g_{y'}}}{\boldsymbol{g_{x'}}}$$

$$\boldsymbol{\alpha} = \tan^{-1} \frac{\boldsymbol{g_{y'}}}{\boldsymbol{g_{x'}}}$$

Secondly, the sensor is used to measure angular velocity (**ω)** via the gyroscope sensor raw data which indicates the velocity as positive when the sensor is rotates counterclockwise and negative clockwise. Finally, the angular acceleration ($\dot{\boldsymbol{\omega}}$) is computed as the time derivative of the angular velocity:

$$\dot{\boldsymbol{\omega}} = \frac{d\boldsymbol{\omega}}{dt} = \frac{\Delta\boldsymbol{\omega}}{\Delta t}$$

In coding it is computed as the difference between the data at a certain instant and the data at the previous cycle as the baud rate is very high (115200), so the time slot is negligible.

Using SerialPlot, a software designed to catch data from the serial bus and represent it as a plot with the desired data on the y-axis and the time as x-axis, it is possible to monitor the sensor outputs across time (Appendix E). Firstly, tilting the sensor from 0 to 45° (counterclockwise) (fig.3.60) and then doing the same movement but while measuring velocity (fig.3.61) and finally repeating again while measuring acceleration (fig.3.62). Secondly, tilting from 0 to -45 (clockwise) and measuring the same data again respectively. (fig.3.63) (fig.3.64) (fig.3.65)

The graphical representations show the validity of the data measured by the sensor as the angular position resembles reality and the velocity profile is

increasing till a certain value and then is constant for a short amount of time and then returns to zero as the movement is terminated while the computed angular acceleration contain very high ripples and the computed data are rejected. This acceleration behavior is unacceptable and is explained by the very minute difference in velocity will produce a very large signal in acceleration which leads to the urge of filtering these data.

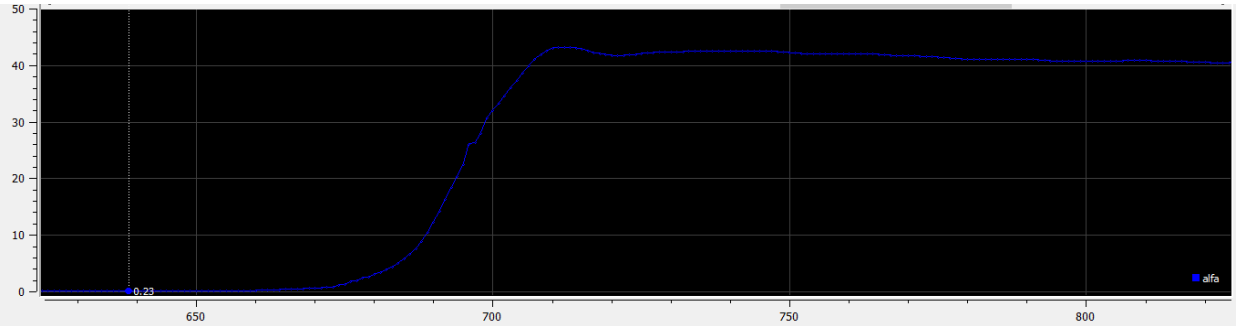


Figure 3.60: Measurement of the angular position (blue) in counterclockwise rotation (x-axis is time in ms, and y-axis is angle in degrees).

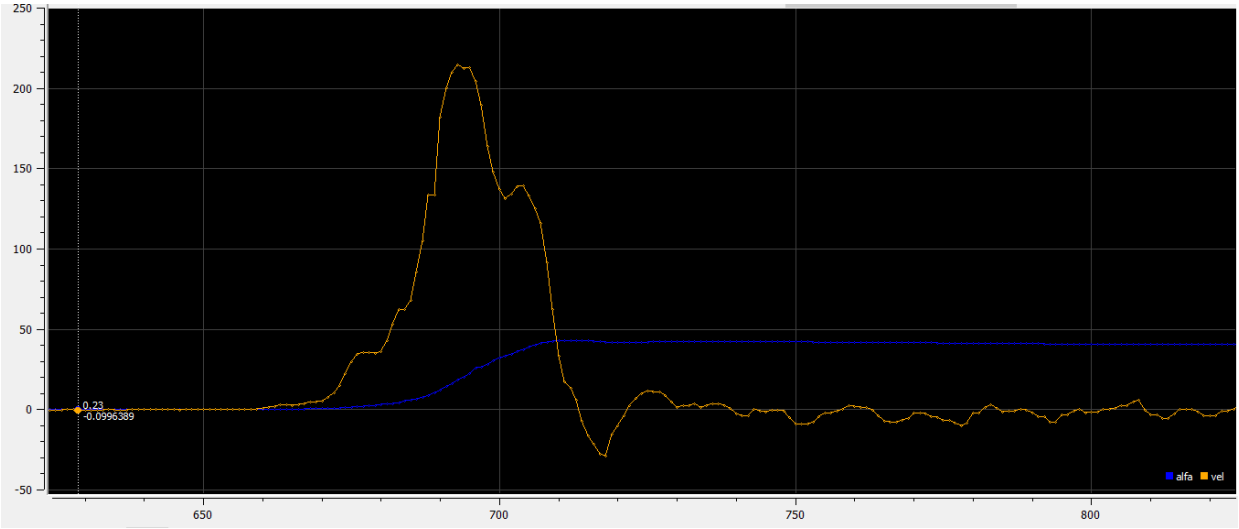


Figure 3.61: Measurement of the angular velocity (yellow) in counterclockwise rotation (x-axis is time in ms, and y-axis is angle rate in rad/s).

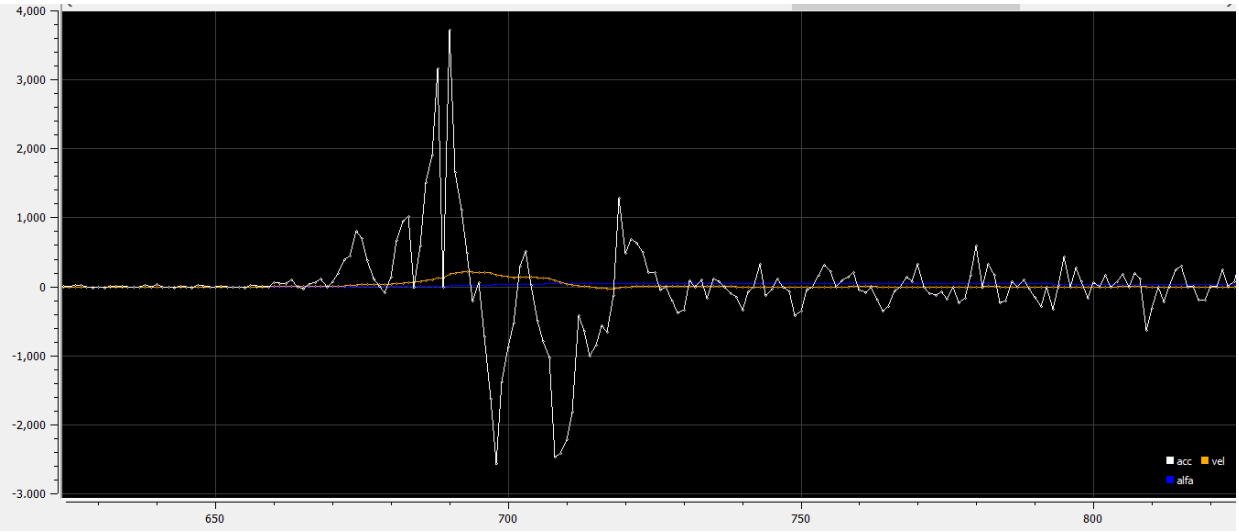


Figure 3.62: Measurement of the angular acceleration (white) in counterclockwise rotation (x-axis is time in ms, and y-axis is velocity rate in rad/s^2).

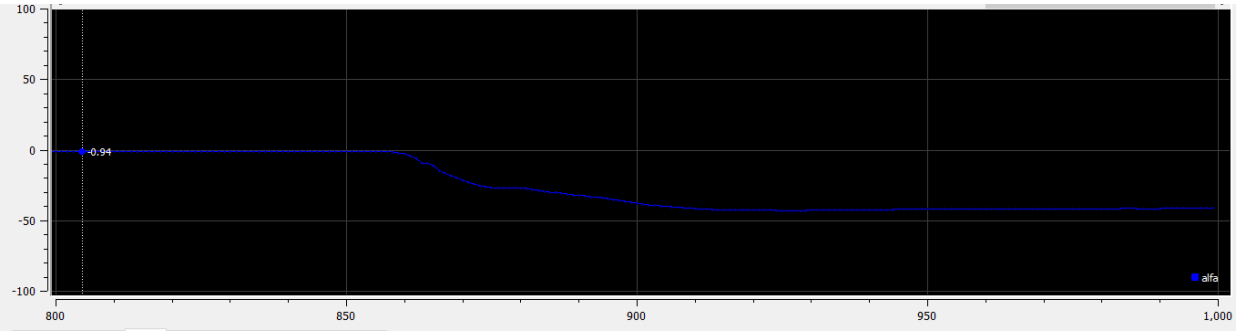


Figure 3.63: Measurement of the angular position (blue) in clockwise rotation (x-axis is time in ms, and y-axis is angle in degrees).

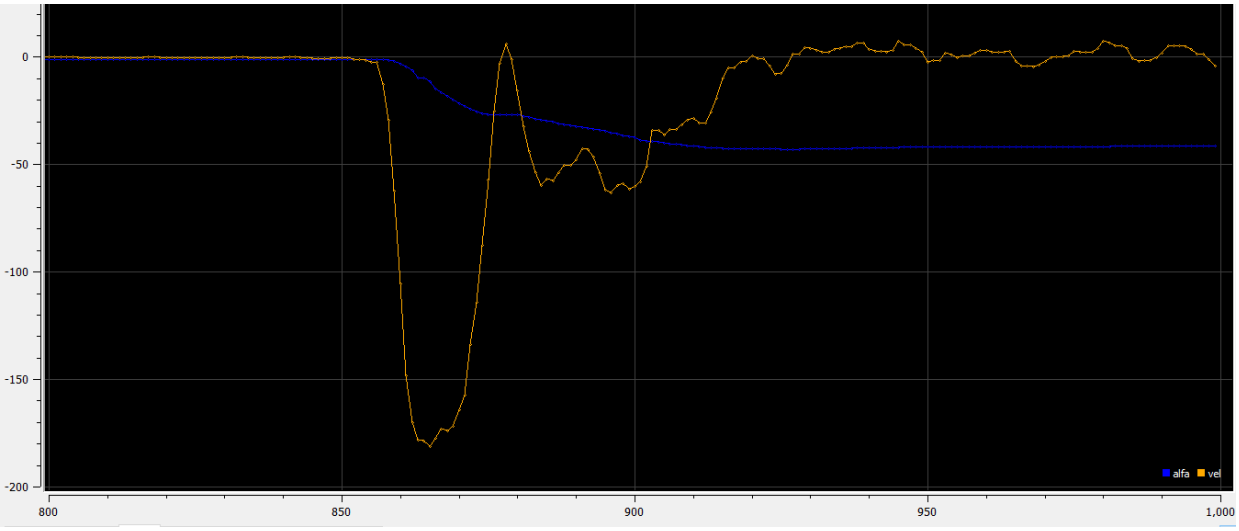


Figure 3.64: Measurement of the angular velocity (yellow) in clockwise rotation (x-axis is time in ms, and y-axis is angle rate in degrees).

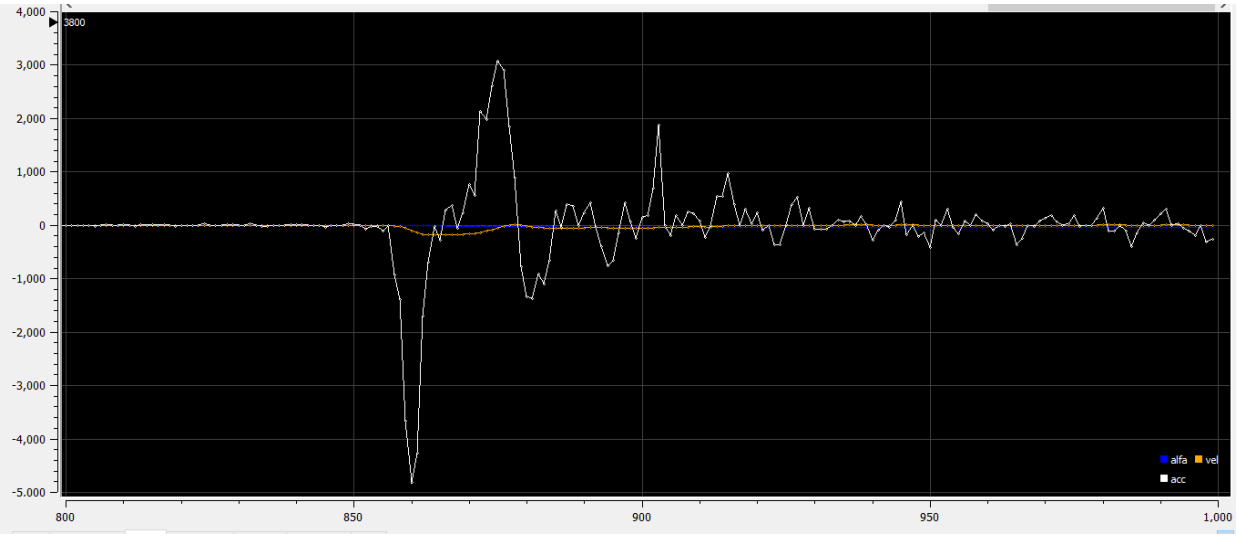


Figure 3.65: Measurement of the angular acceleration (white) in clockwise rotation (x-axis is time in ms, and y-axis is velocity rate in rad/s^2).

3.2.4 Acceleration data filtering

Very high ripples in the angular acceleration data acquired from the calculation of the rate of angular velocity will lead to incorrect calculation of the torque required by the air motors which is controlled by the varying pressure from the Festo proportional valve. There are two main concepts on filtering any type of data in code: first, by using a delay between every computation and the previous one or using a so called “low pass filter”.

The delay algorithm: is as obvious as it is called as instead of measuring data in high rate and computing these data in the same rate, the data are only recognized in a lower rate i.e. the data read from the sensor device is the same data and at the same rate but some of them are rejected depending on a specified delay that is set to better visualize the correct data needed. Moreover, this method leads to rejection of small ripples acquired from unneeded movements of the sensor in space as in the exoskeleton, a human is operating the devise while wearing it, so

the sensor is not fixed in space and the only motion present is not of the joint angular motion but also the normal human movements.

As per (Appendix F), the same tests done in the previous section is repeated while introducing the delay filtering method and then reading the angular position, velocity and acceleration in counterclockwise (fig.3.66), (fig.3.67), (fig.3.68) respectively and clockwise directions (fig.3.69), (fig.3.70), (fig.3.71) respectively. In this filter, data is read every 100 ms and the filter delay is 200 Ms.

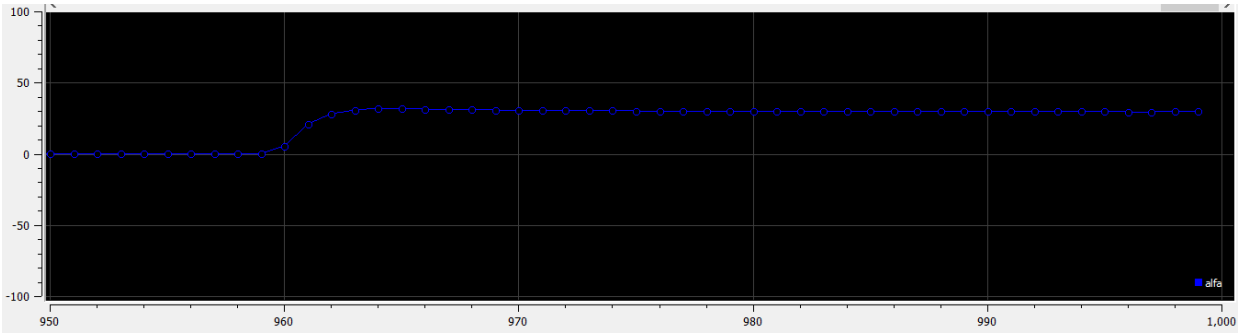


Figure 3.66: Measurement of the angular position (blue) in counterclockwise rotation (x-axis is time in ms, and y-axis is angle in degrees) with delay filter.

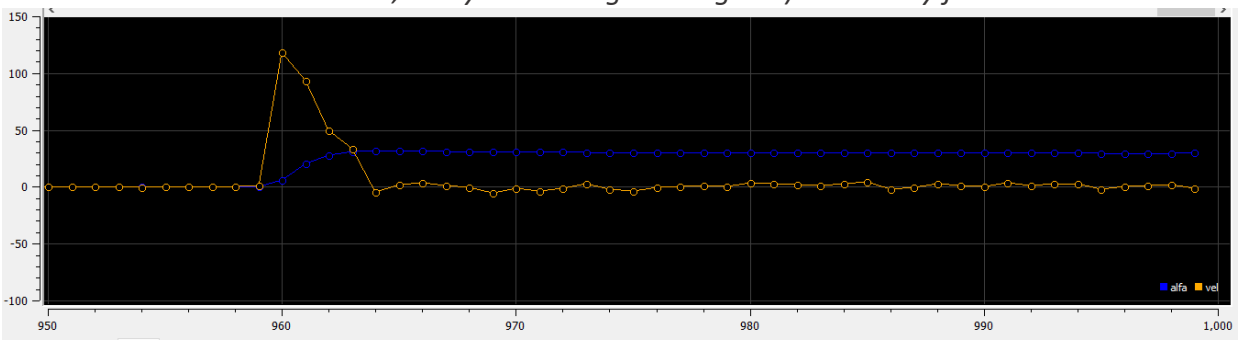


Figure 3.67: Measurement of the angular velocity (yellow) in counterclockwise rotation (x-axis is time in ms, and y-axis is angle rate in rad/s) with delay filter.

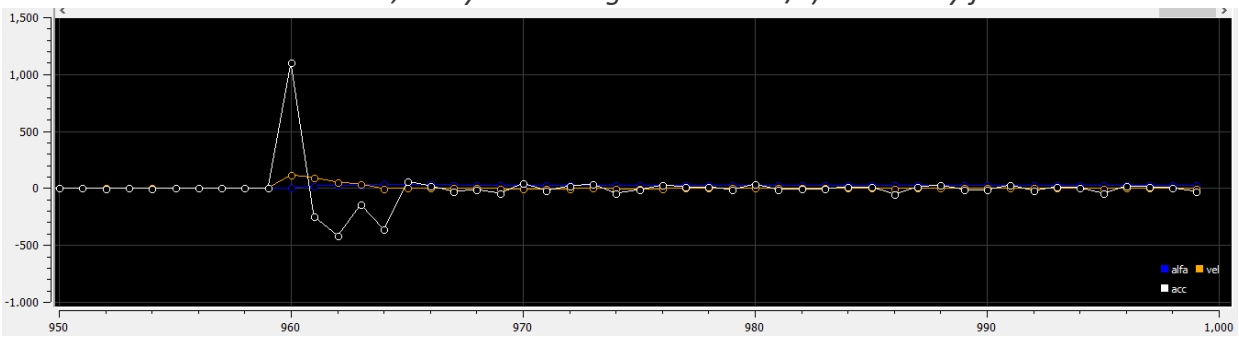


Figure 3.68: Measurement of the angular acceleration (white) in counterclockwise rotation (x-axis is time in ms, and y-axis is velocity rate in rad/s^2) with delay filter.

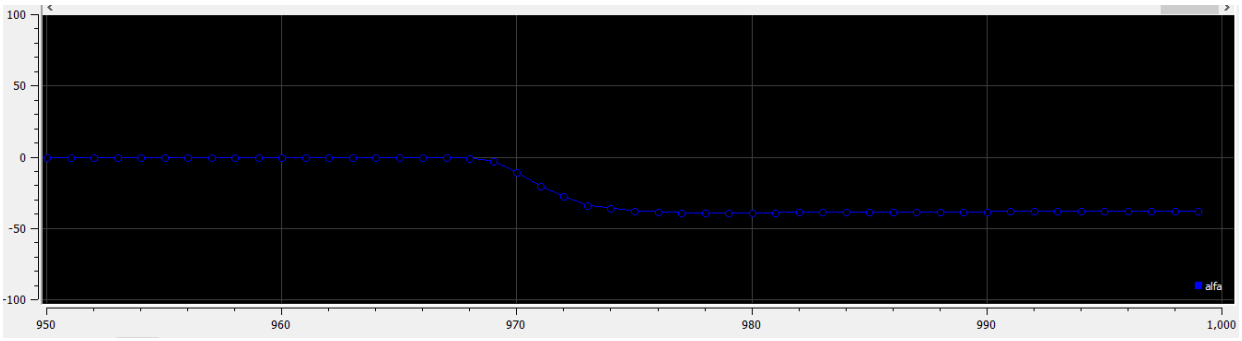


Figure 3.69: Measurement of the angular position (blue) in clockwise rotation (x-axis is time in ms, and y-axis is angle in degrees) with delay filter.

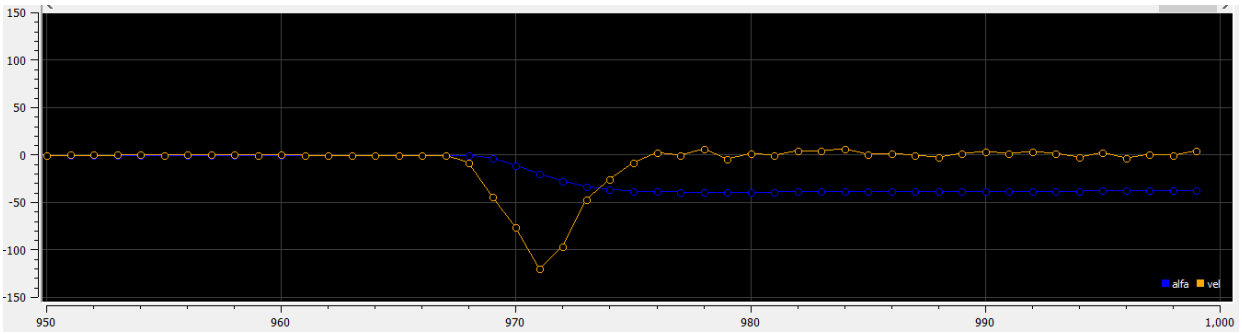


Figure 3.70: Measurement of the angular velocity (yellow) in clockwise rotation (x-axis is time in ms, and y-axis is angle rate in degrees) with delay filter.

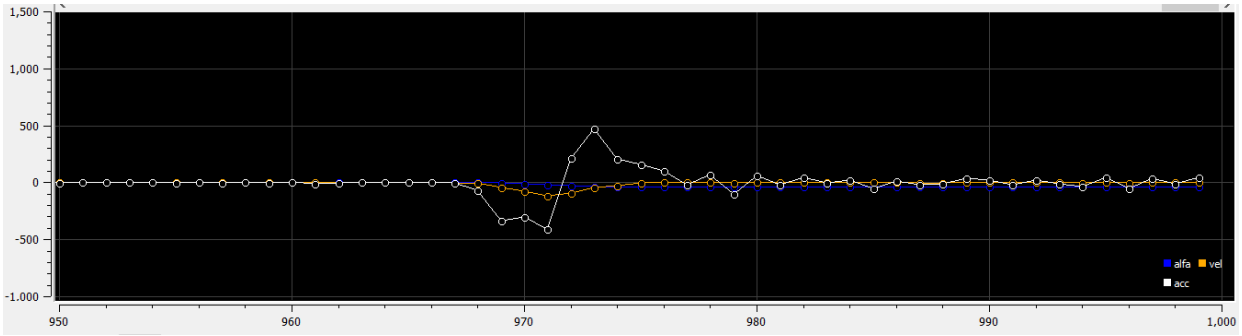


Figure 3.71: Measurement of the angular acceleration (white) in clockwise rotation (x-axis is time in ms, and y-axis is velocity rate in rad/s^2) with delay filter.

As shown in the acceleration profiles the spikes are more obvious and closer to reality, however this method has a drawback which is the very large delay time that the sensor high data rate is wasted. However, the proportional valve has a high mechanical response delay so this filter can be used in this application. Moreover, the delay times can be chosen with smaller values to increase the data

rate, but this depends on the real time, hardware in the loop (HIL) testing phase to show the best data profile acquired.

The weights algorithm is mainly defined by assigning weight for the data acquired at a certain instant and adding it to the weighted data at the previous instant. The weights should be defined as the previous instant measurand weight is more than 90% while the new instant measurand weight is less than 10%. This guarantees that any high change in the measured data will not be trusted, however, only a small percent of the new measurand is taken into consideration. This method is effective in the angular acceleration calculation as it results in more stable behavior which resembles the function of a low pass filter as it rejects very high ripples generated from minor movements of the sensor in space.

As per (Appendix G), the same tests done in the previous sections is repeated while introducing the weights filtering method and then reading the angular position, velocity and acceleration in counterclockwise (fig.3.72), (fig.3.73), (fig.3.74) respectively and clockwise directions (fig.3.75), (fig.3.76), (fig.3.77) respectively. In this filter, the code is executed without any delay and with 115200 as baud rate.

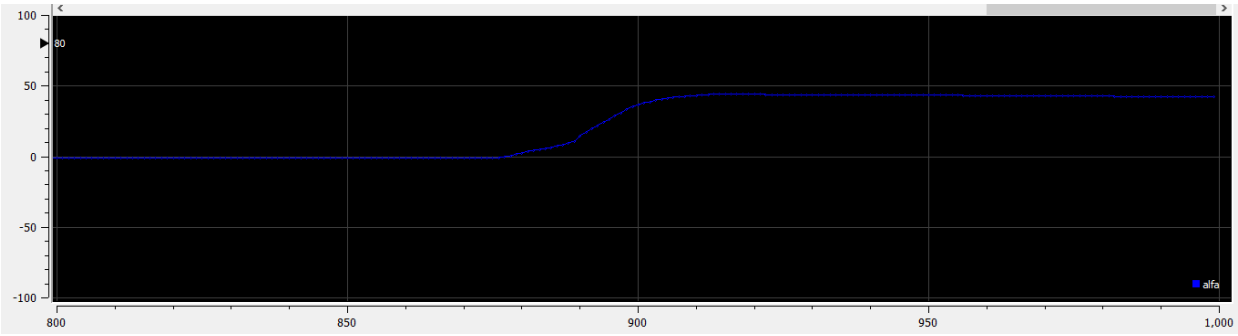


Figure 3.72: Measurement of the angular position (blue) in counterclockwise rotation (x-axis is time in ms, and y-axis is angle in degrees) with weights filter.

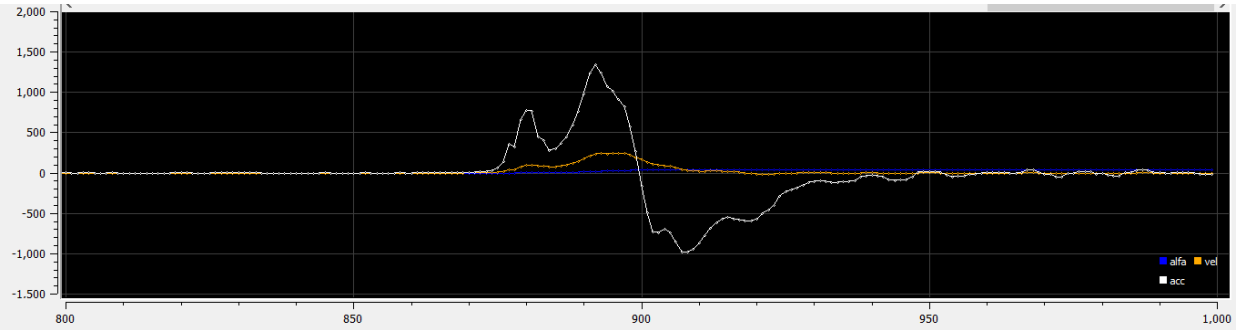


Figure 3.73: Measurement of the angular velocity (yellow) in counterclockwise rotation (x-axis is time in ms, and y-axis is angle rate in rad/s) with weights filter.

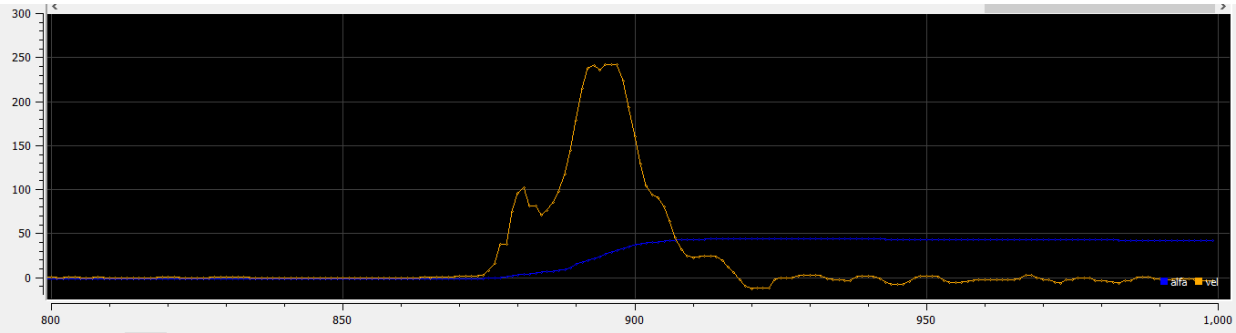


Figure 3.74: Measurement of the angular acceleration (white) in counterclockwise rotation (x-axis is time in ms, and y-axis is velocity rate in rad/s^2) with weights filter.

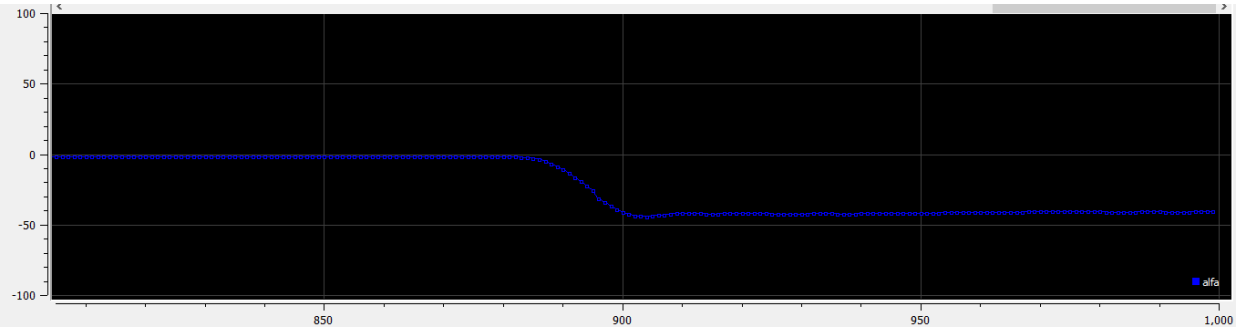


Figure 3.75: Measurement of the angular position (blue) in clockwise rotation (x-axis is time in ms, and y-axis is angle in degrees) with weights filter.

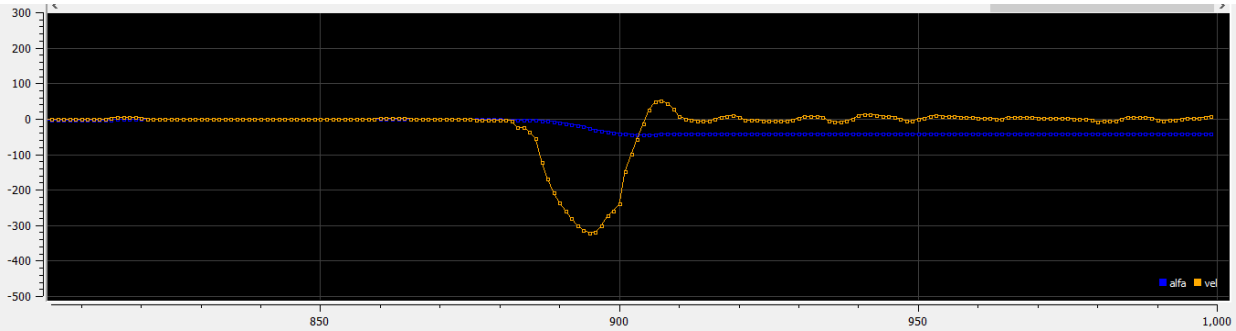


Figure 3.76: Measurement of the angular velocity (yellow) in clockwise rotation (x-axis is time in ms, and y-axis is angle rate in degrees) with weights filter.

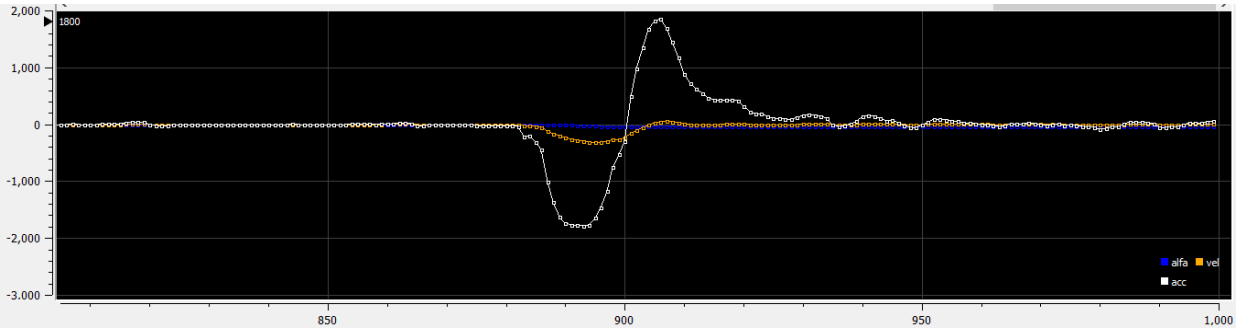


Figure 3.77: Measurement of the angular acceleration (white) in clockwise rotation (x-axis is time in ms, and y-axis is velocity rate in rad/s^2) with weights filter.

As seen from the acceleration figures, the angular acceleration data are more accepted than the data obtained without filtering, however, for this type of filtering the actual values of the acceleration can be not very close to reality at it requires a period of time for the measured value to settle at the new instant but again the very high baud rate validate accepting these values.

4 Testbench Simulation

Before the exoskeleton can be used by human, it is tested using a designed testbench. This device is to emulate the user whom the exoskeleton is mounted on and strapped to its links where the joints are aligned. This device is actuated by pneumatic cylinders to perform the trajectory of the human bending phases described before. Firstly, the device is designed on SolidWorks and then imported for testing on MATLAB. Using the Simscape library a complete model is designed to compare with the calculations of the forces of the air cylinders and the desired torques on the joints are validated as well.

4.1 Parts and assembly

The test bench mainly consists of three parts that are joined through revolute joints (fig. 4.78): Base (fig. 4.79A), Pelvic part (fig. 4.79B) and Backframe (fig. 4.79C). These three parts emulate the human better than with only one joint between the backframe and base and it services well the exoskeleton device as it is actuated by two rotary joints.

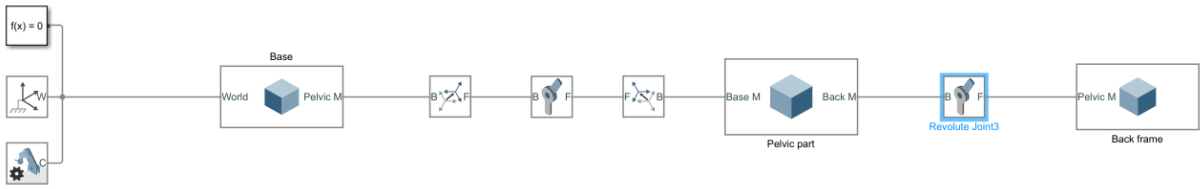


Figure 4.78: Testbench main parts.

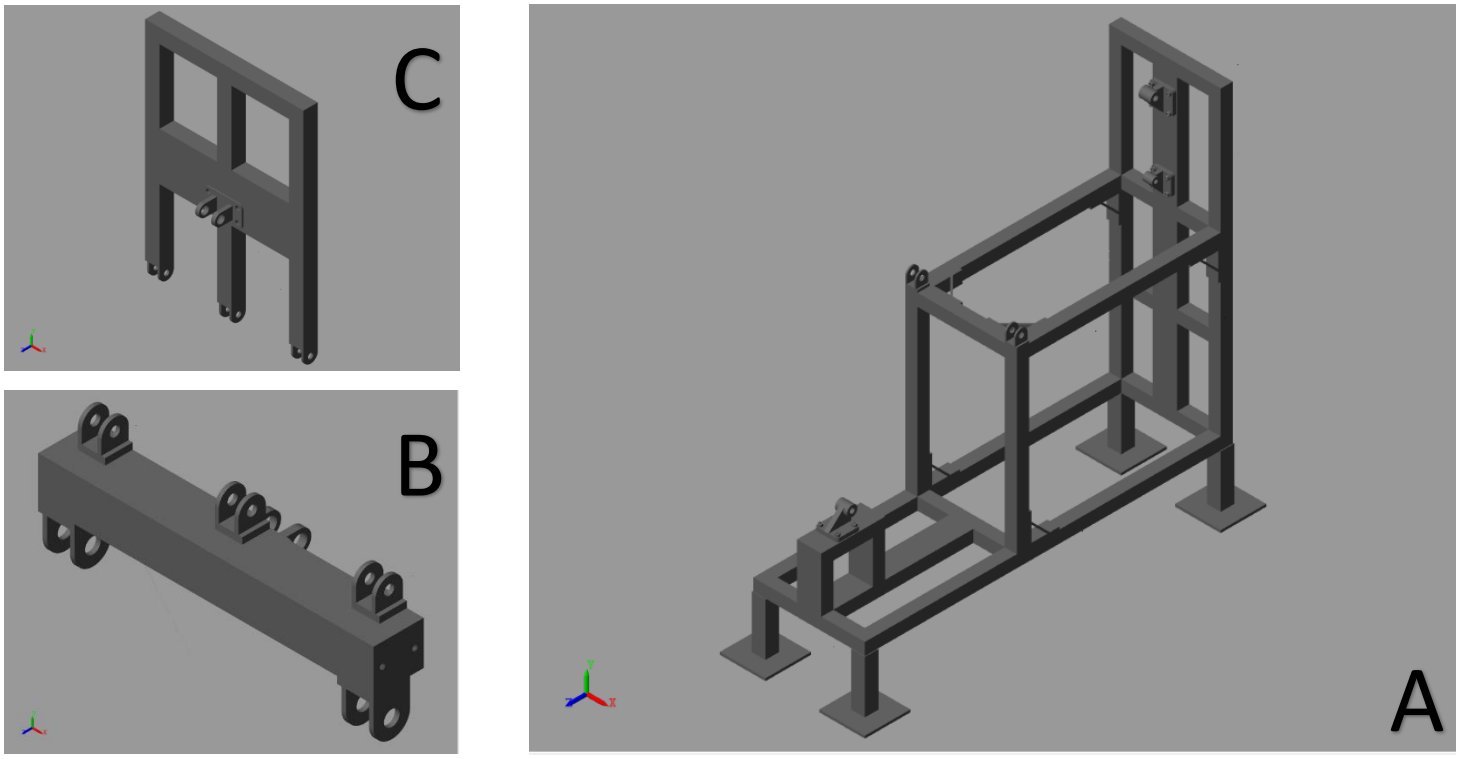


Figure: 4.79: Testbench parts, A is Base, B is Pelvic part, C is Backframe.

Moreover, the testbench is actuated by pneumatic cylinders which were imported from the manufacturer CAD files as well. Three pneumatic cylinders are used to actuate the model, two of which are the same (fig. 4.80) used to actuate the backframe from the front and back and the third is a smaller (fig. 4.81) one used to actuate the pelvic part.

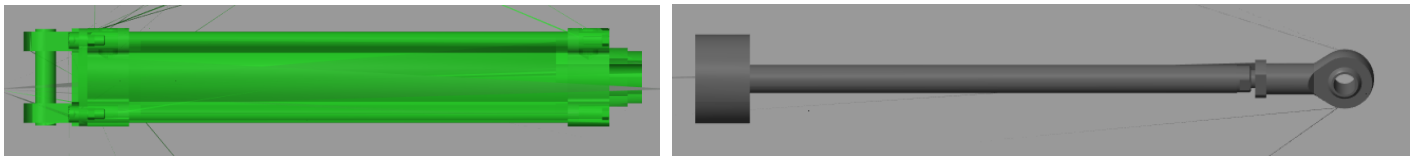


Figure 4.80: The bigger cylinder used with the backframe (2 cylinders used) (colors are only for visualization).

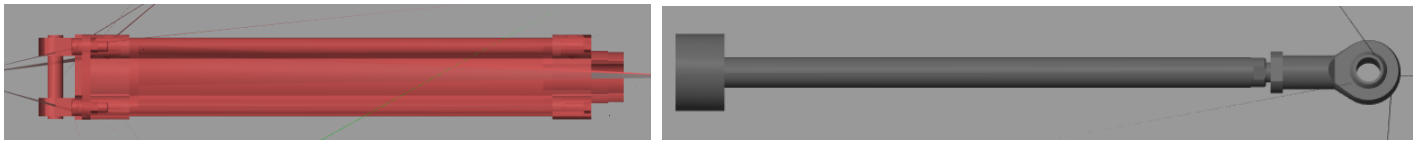


Figure 4.81: The smaller cylinder used with the pelvic part (1 cylinder used) (colors are only for visualization).

The complete model (fig. 4.82) is then assembled with the correct orientation of the connection frames which were defined on the parts geometry instead of using rigid transform blocks as the origin of the imported step files are defined not correctly, however, this does not affect the simulation.

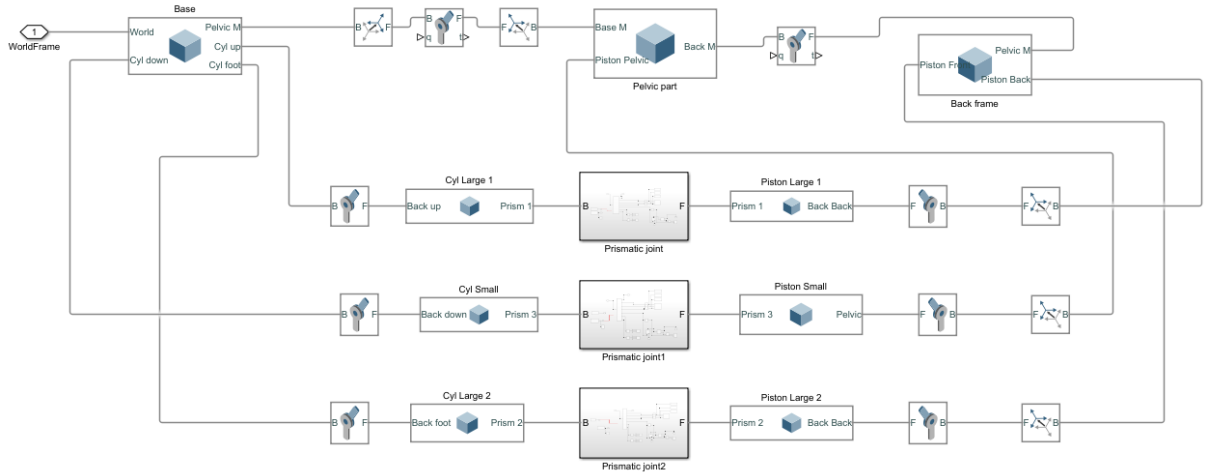


Figure 4.82: Complete testbench model.

As shown in the assembly (fig. 4.83), the joints of the base and pelvic part which are two joints with rotational motion, however in simulation only one joint is used on the midpoint along the centerline connecting the two joints and this is because during simulation the more the joints, the more complications occur during simulation as the simulating tool actuates with precise trajectories and minute data values always cause kinematic singularities which makes it impossible to simulate the model. This is also done on the joint between the pelvic part and backframe, where there are three rotational joints and only the one in the middle is actuated. Finally, all these rotational joints are aligned with the midplane of the testbench where the midplane of the cylinders in position coincide.

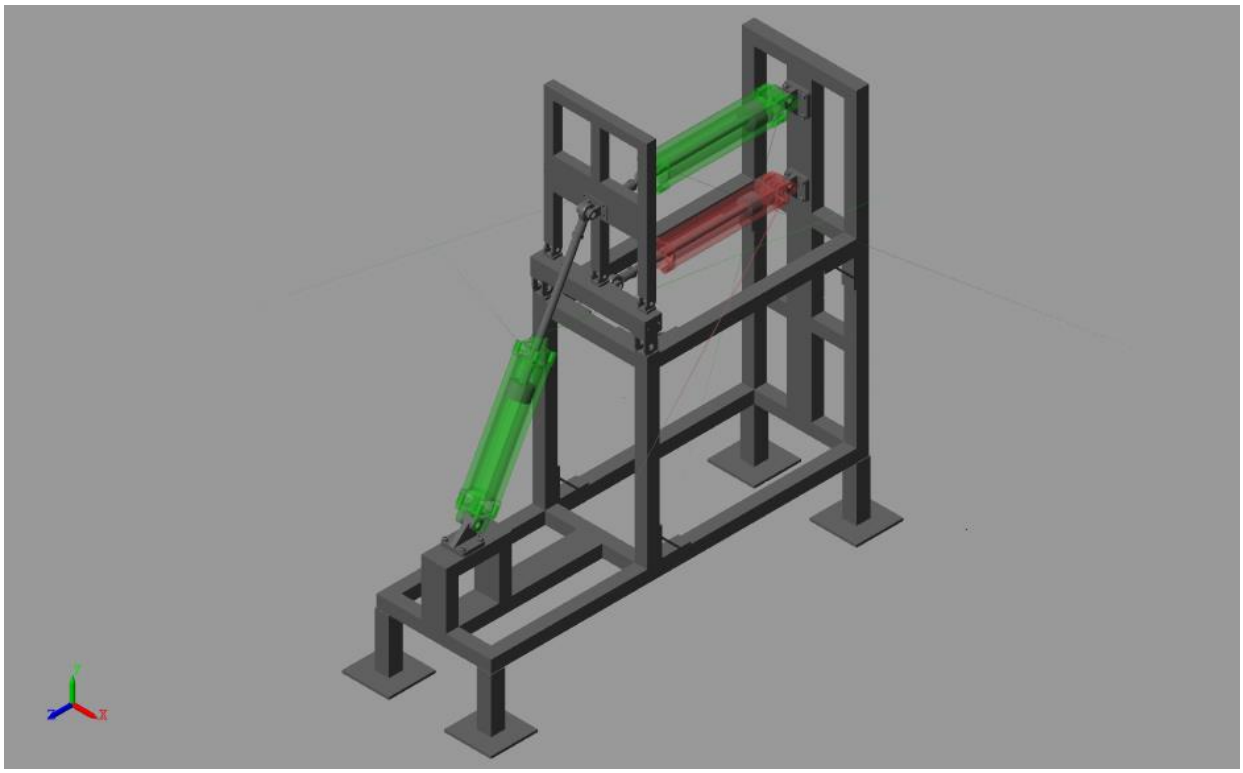


Figure 4.83: Complete assembly of the testbench.

4.2 Actuation and sensing

The function of the testbench is to emulate the human movement to test the exoskeleton on it. This movement is emulated by the three pneumatic cylinders only which are actuated using linear motion and forces. However, during studying the human movement, it was defined by a cycloidal law on the rotary joints of the hip and pelvis and the torques generated at them. Thus, to translate the cycloidal law from the revolution joints to the cylinders, the model was actuated at the rotary joints with the desired motion which is:

- 0-20° cycloidal law on the hip joint (lower joint) during (0 to 1s) and then fixed at 20° during (1s to 35s) followed by 20°-0 during (35s to 36s). Finally, steady at 0 till 60s. (fig. 4.84)
- 0 on the pelvic joint (upper joint) during (0 – 1s) followed by 0-50° cycloidal during (1s to 3s) then fixed at 50° during (3s to 33s) then 50°-0 cycloidal during (33s to 35s). Finally, steady at 0 till 60s. (fig. 4.85)

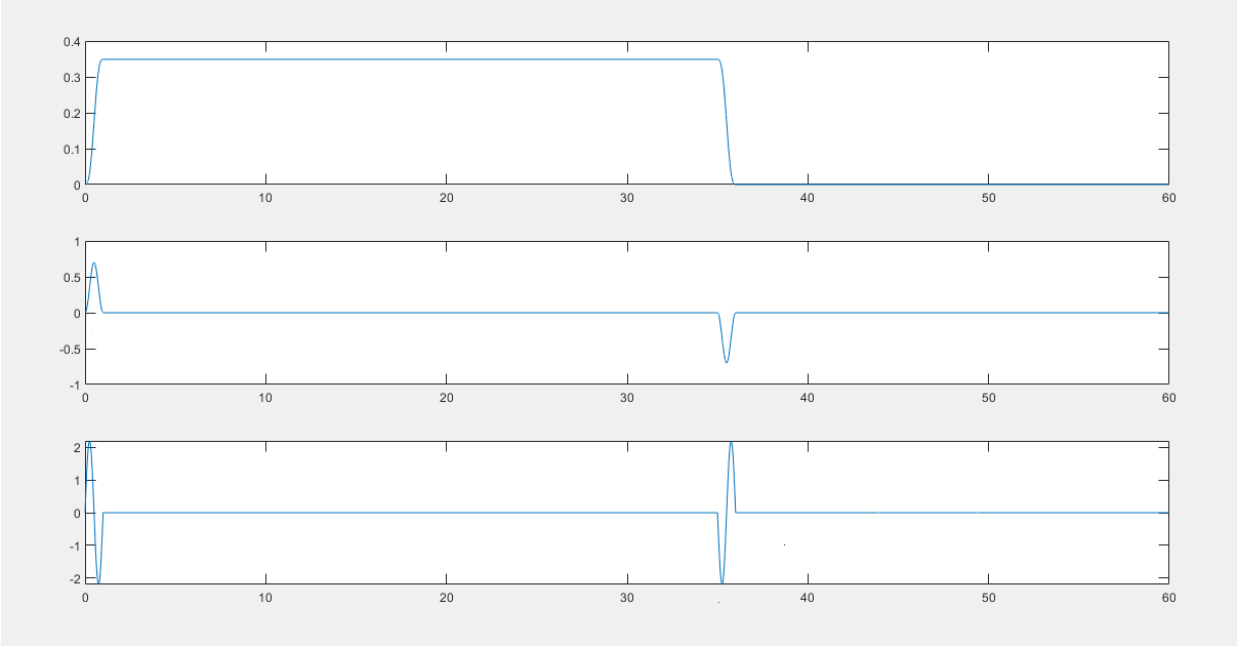


Figure 3.84: Kinematics of the hip joint. (x-axis is in s) (y-axis is in rad, rad/s and rad/s/s).

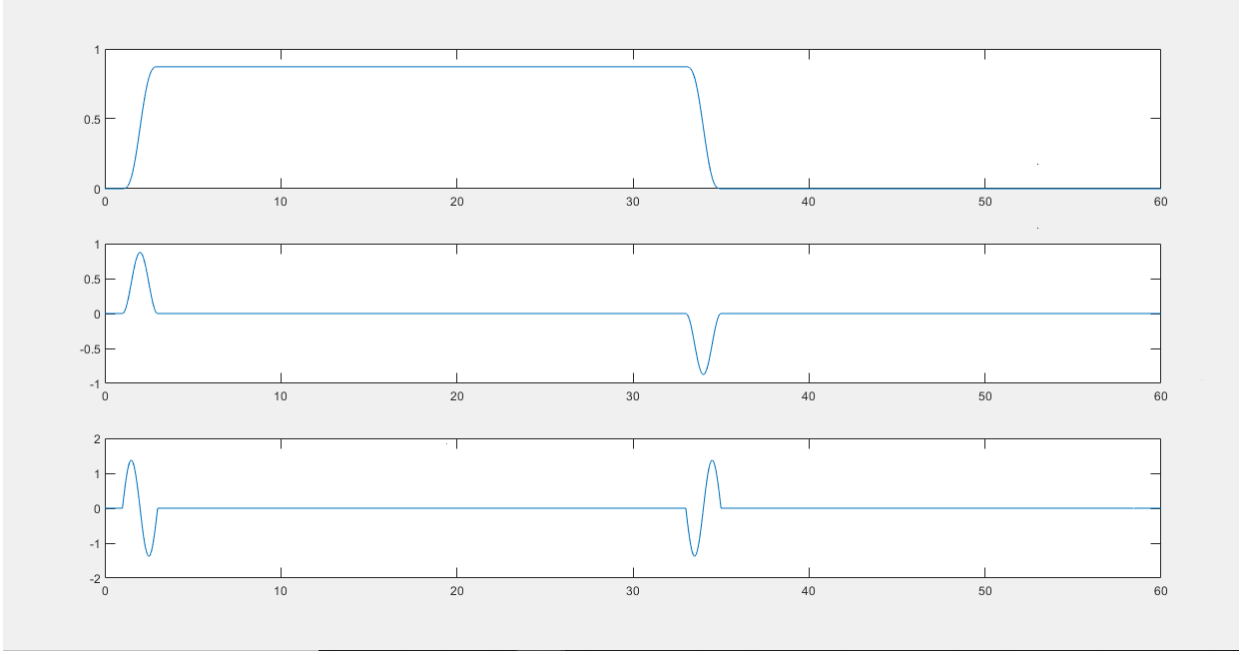


Figure 4.85: Kinematics of the pelvic joint. (x-axis is in s) (y-axis is in rad, rad/s and rad/s/s).

The cycloidal law is transformed to the pneumatic cylinders through actuating the previously described kinematics on the rotational joints of the assembly model (fig. 4.86) in the model and automatically computing the kinematics of the cylinders then. The result kinematics (fig. 4.87,4.88,4.89) on the three cylinders are always stored into the workspace to be used with the models used to analytically compute the forces required from the cylinders which is the main scope of this simulation.

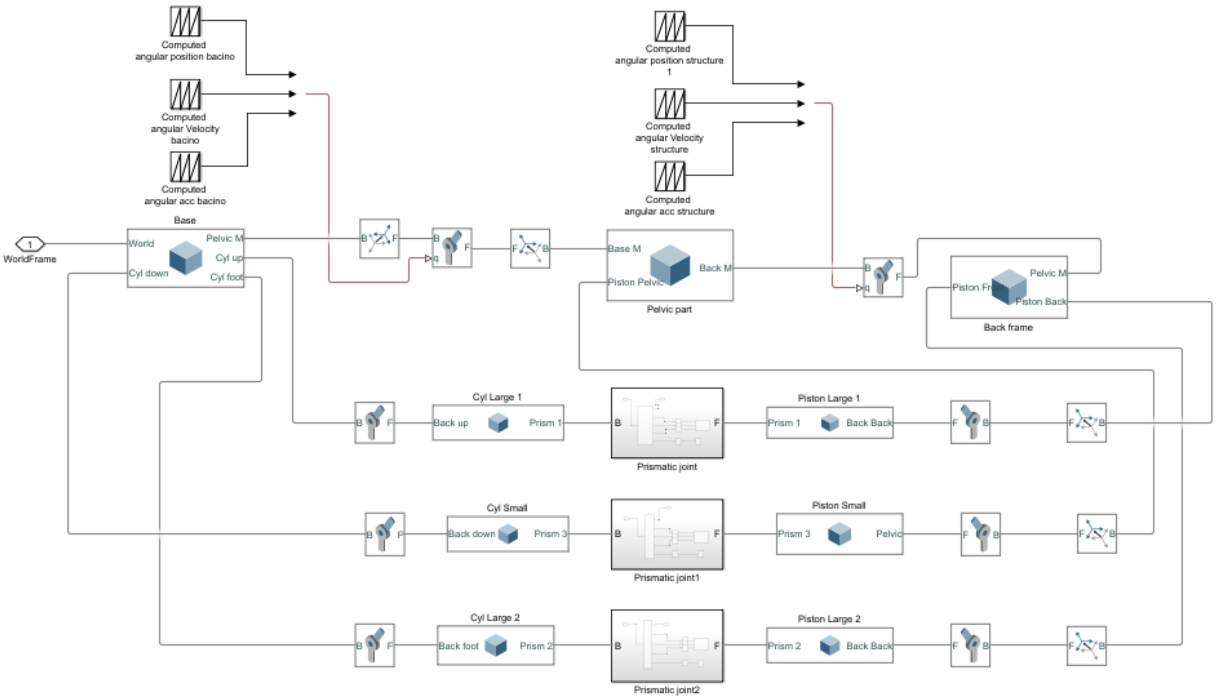


Figure 4.86: The assembly model used to transfer the cycloidal law to the cylinders.

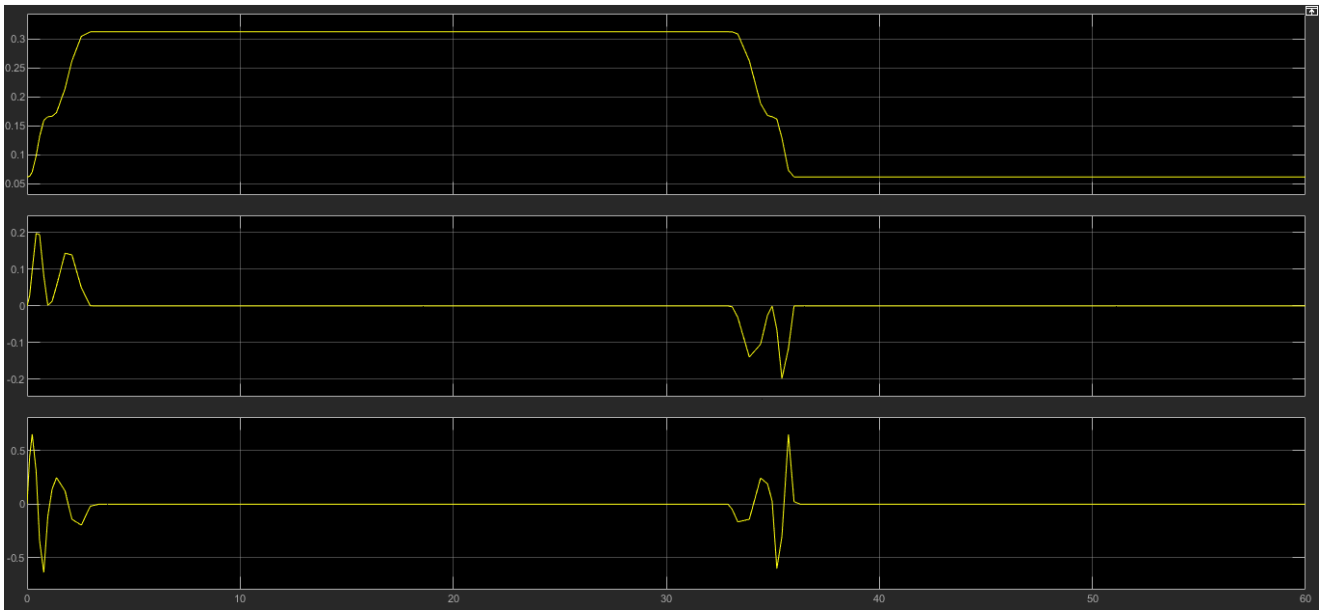


Figure 4.87: Kinematics of the large upper cylinder. (x- axis is in s) (y-axis is in m, m/s and m/s/s).

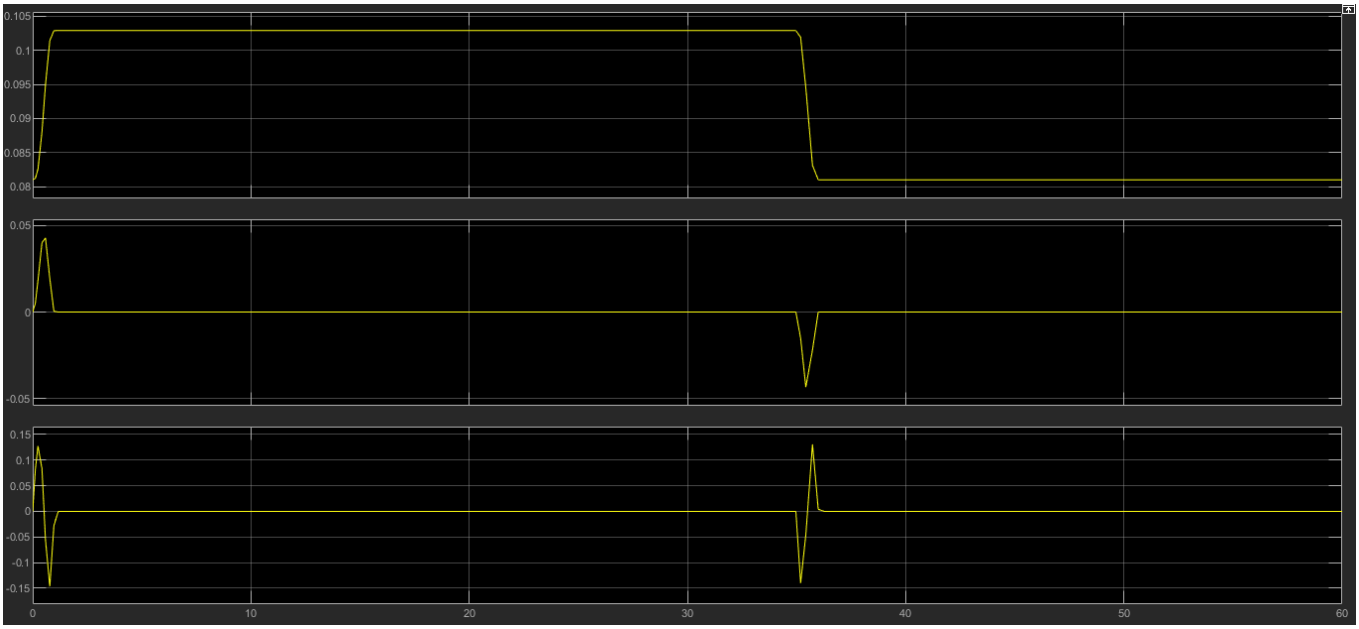


Figure 4.88: Kinematics of the small lower cylinder. (x- axis is in s) (y-axis is in m, m/s and m/s/s).

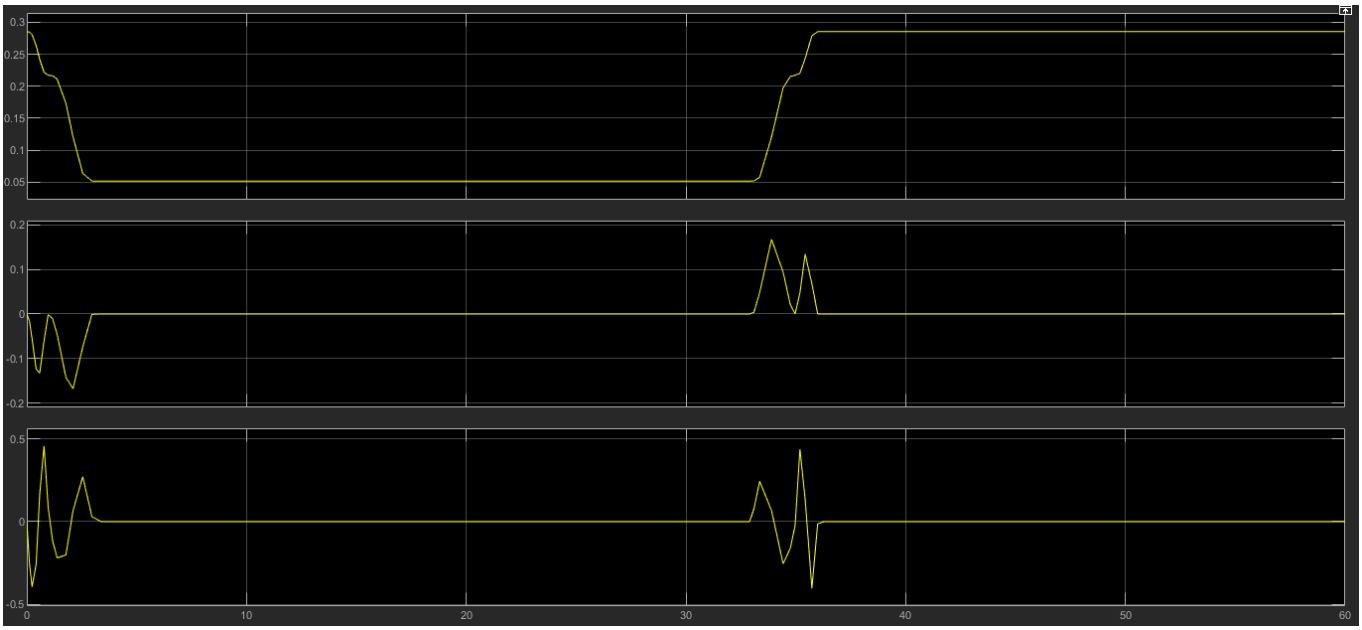


Figure 4.89: Kinematics of the large frontal cylinder. (x- axis is in s) (y-axis is in m, m/s and m/s/s).

4.3 Analytical calculation of forces

The testbench is actuated using three pneumatic cylinders which are connected to the pelvic part and the backframe which is the real kinematics of a human with a slight modification as during human bending the pelvis rotates around an axis through itself not through the hip joint. This is compensated as when the exoskeleton is mounted on the testbench, it is mounted on the pelvic part of the testbench such that the whole pelvis of the exoskeleton rotates.

Forces computations were made, and air cylinders were already selected at the time of simulation. The forces were computed by relaxing some of the movements of the testbench. Such that the cylinder connected to the pelvic part was selected to only provide the kinematics of this joint to correctly emulate the human motion but was not used as part of the dynamics calculations to provide the correct torques at specified joints.

As mentioned before, the exoskeleton provides 30% of the load generated by the human body during bending where the maximum value is at 70° bending angle. The torque then is calculated a simulated to be 266 N*m, which is the maximum torque provided by the exoskeleton and then, the maximum torque required by the testbench.

The main concept of choosing the mounting positions of the three cylinders were such that, for the two back cylinders, forces in the extension direction is required to produce the kinematics, however for the frontal cylinder, the main function is to produce 70% of the human torque while in working positions (WORKING, LIFT stateflow states). The frontal cylinder is only used to assist the exoskeleton such

that, when the exoskeleton produces 30% and the cylinder produces 70% the total torque reacts to the 100% torque produced by the two back cylinders. This means that to simulate the testbench forces through the complete cycle of 60s (through all the states), each state should be simulated alone and tested and then all the states are integrated together to produce the forces profiles through the 60s. This means that the cycle is broken into six states with six models to calculate the forces and twelve models to test them and finally a model to check the torques of the rotational joints when the testbench is actuated by the forces on the cylinders and the kinematics on the rotational joints

4.3.1 First state (Bending 0-20°)

These calculation of forces models all have the same concepts where the rotational joints are actuated by the required torques and the cylinders are actuated by the correct kinematics calculated before. Thus, the prismatic joints provide the forces required to produce the kinematic law on the prismatic joints in the opposite direction which is corrected by multiplying the analytically calculated forces by (-1).

The first state, with a period of 1s, is bending the hip joint 0-20° by cycloidal law while the pelvic joint is fixed at zero degrees. (fig. 4.90)

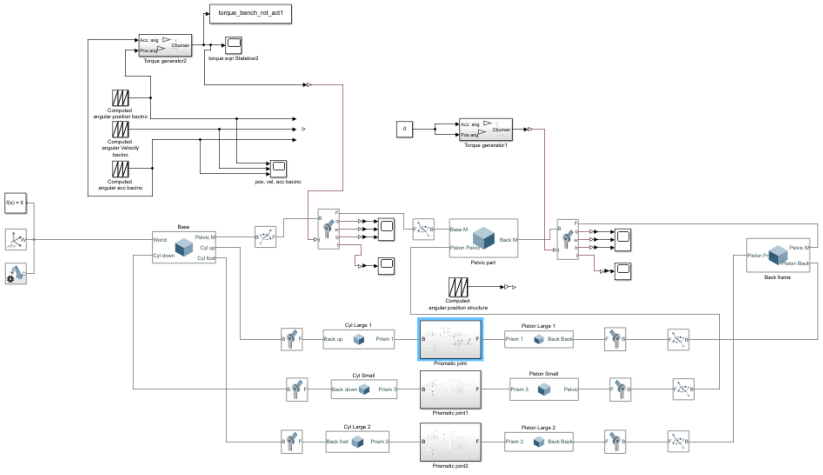


Figure 4.90: 0-20° cycloidal bending on hip joint.

During this state the back cylinders are the only ones actuated with forces (fig. 4.91) as the exoskeleton is passive (it only senses the bending position and it computes the torques by itself). Thus, the frontal cylinder is passive in this state.

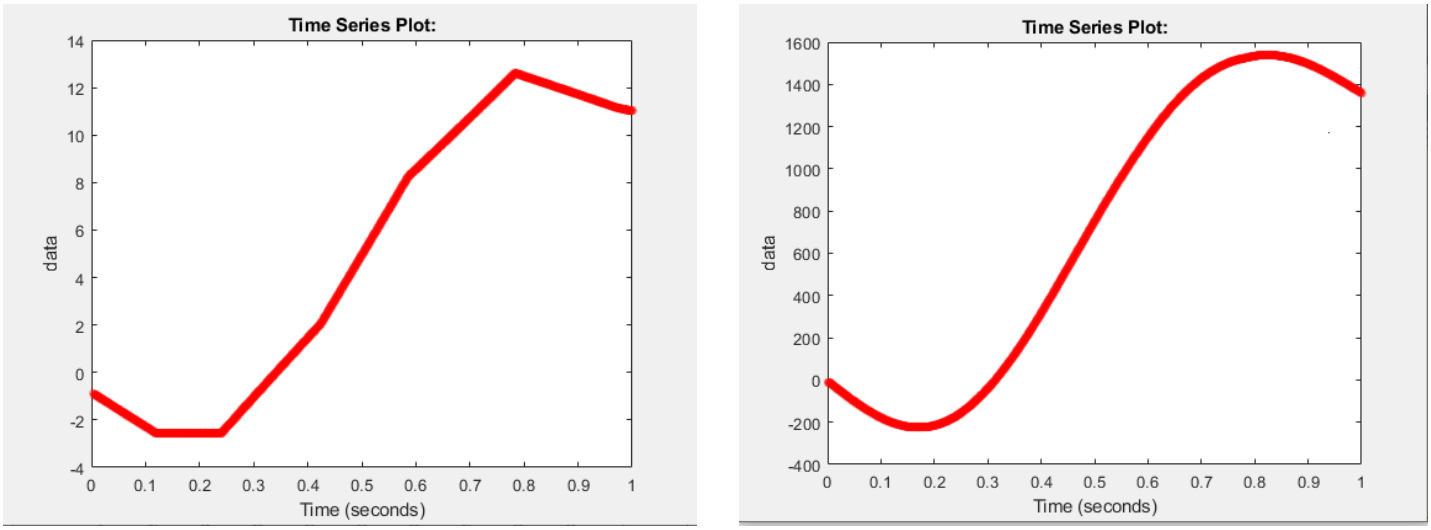


Figure 4.91: Left: upper cylinder force (N), Right: lower cylinder force (N).

4.3.2 Second state (Bending 0-50°)

As done in the previous state, this time the hip joint is fixed at 20° and the pelvic joint is actuated by the torque required to produce the cycloidal motion of (0-50°) in 2s (1s - 3s). this torque is the not computed as the torque of the bending of 50°, however it is computed as the rest of the torque of the 70° after subtracting the torque of the previous state and this is because the exoskeleton has only one air motor that is producing all the torque. The prismatic joints are actuated by the computed kinematics. (fig.4.92)

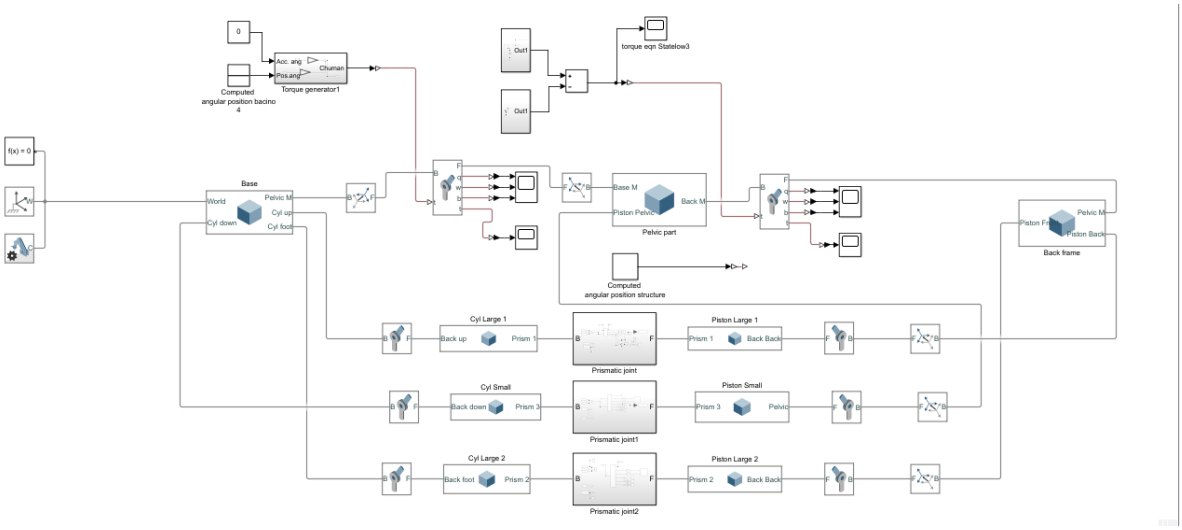


Figure 4.92: 0-50° cycloidal bending of the pelvic joint.

During this state, as well, the back cylinders are the only ones actuated with forces (fig. 4.93) as the exoskeleton is passive (it only senses the bending position and it computes the torques by itself). Thus, the frontal cylinder is passive in this state.

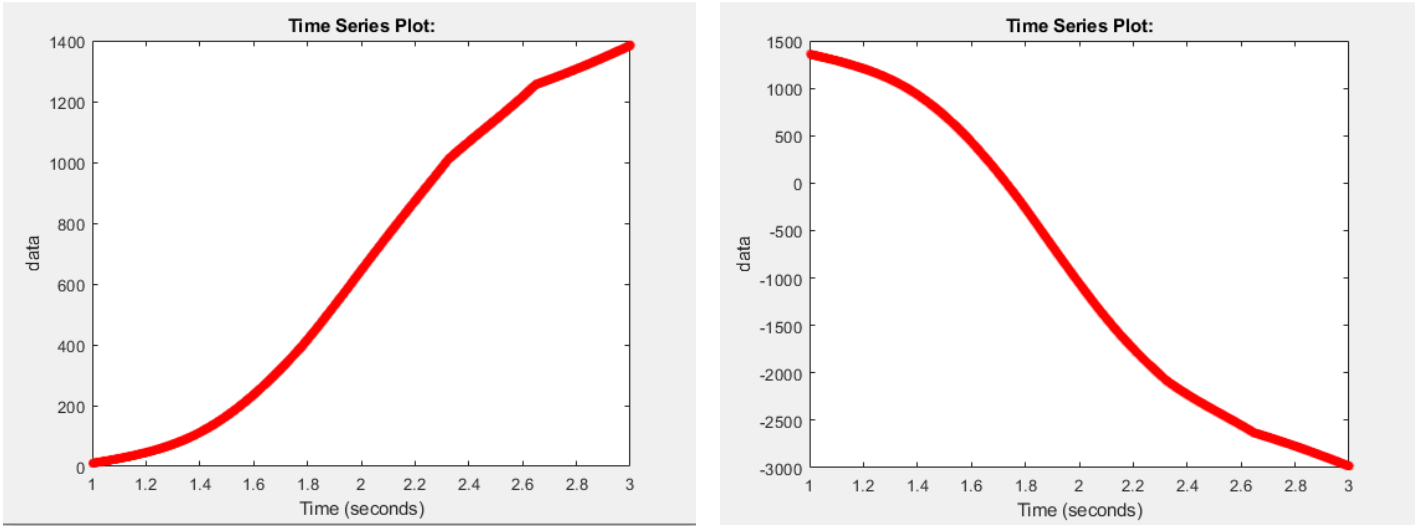


Figure 4.93: Left: upper cylinder force (N), Right: lower cylinder force (N).

4.3.3 Third state (steady bending)

During this state, the testbench model (fig. 4.94) is bending by the whole 70° and is producing a torque of 266 N*m where the exoskeleton is supporting 30% and the frontal cylinder should be supporting the rest 70%. However, the front cylinder can be passive in this state as well and the back cylinders only produce 30% of the load which will be supported by the exoskeleton and this is in order to decrease total error in calculations and to relax the actuated forces to prevent any failure. The hip joint is actuated by 30% of the torque produced at 20° and the pelvic joint is actuated by 30% of the rest of the torque produced at 70°. These torques are constants where the rates are always zeros for 30s (3s – 33s).

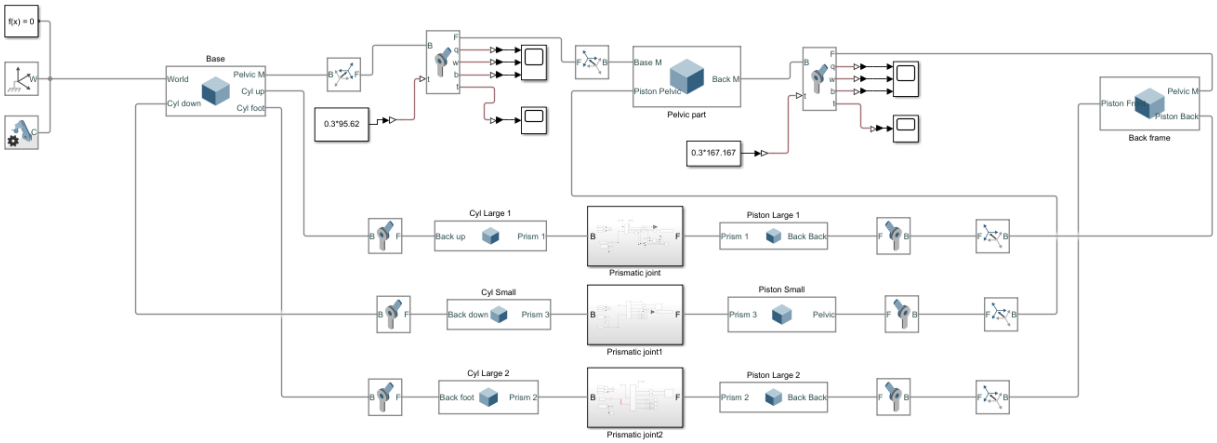


Figure 4.94: Steady bending of a total of 70°.

During this state, as well, the back cylinders are the only ones actuated with forces (fig. 4.95) as the exoskeleton is supporting these forces (it only senses the bending position and it computes the torques by itself). Thus, the frontal cylinder is passive in this state.

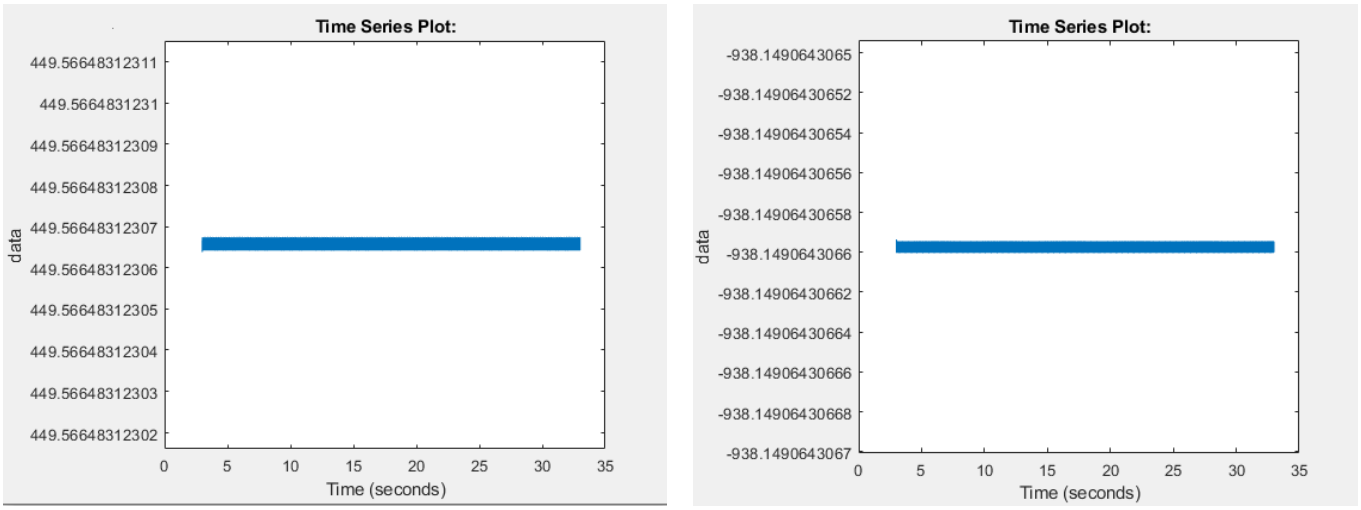


Figure 4.95: Left: upper cylinder force (N), Right: lower cylinder force (N).

4.3.4 Fourth state (Return to 20°)

This is the first state in which the front cylinder will be actuated to support 70% of the load of the emulated human such that along with the 30% torque produced by the exoskeleton, they react to the forces of the back cylinders.

From basic dynamics, to measure dynamics, one should actuate with kinematics and vice versa. And due to the fact that the model of the testbench is a closed loop model so, it is impossible to actuate the three cylinders with kinematics at the same time in the same simulation as this produces a kinematic singularity even if the kinematics were computed analytically. So, the model should be actuated with kinematics on the back cylinders and then use these forces in another model to actuate the frontal cylinder.

First of all, a model (fig. 4.96) is created to calculate the forces required from the back cylinders to produce 100% of the human emulated torque in the counterclockwise direction at the dedicated kinematics of (50°-0) on the pelvic joint and 20° on the hip joint for 2s (33s – 35s). The back cylinder forces (fig. 4.97) are then computed.

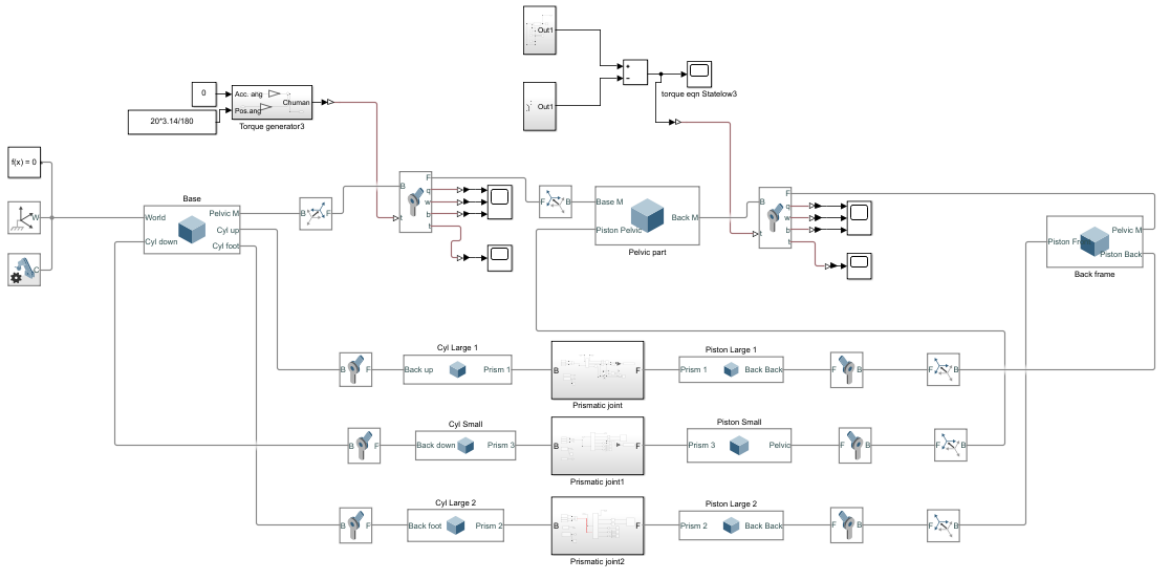


Figure 4.96: Return of the pelvic joint to zero degrees and 100% torque calculation.

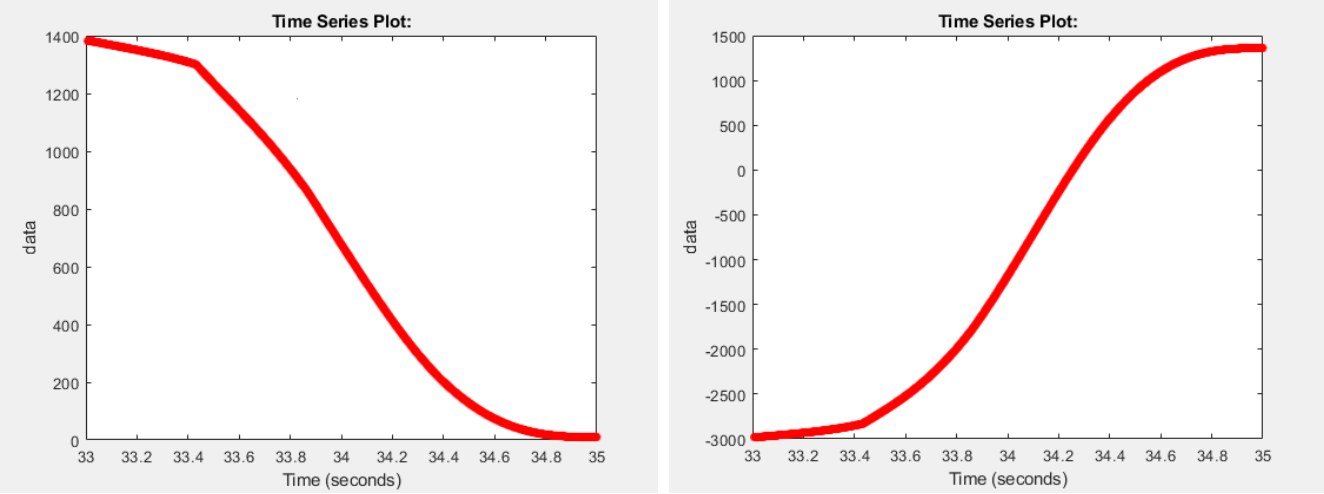


Figure 4.97: Left: upper cylinder force (N), Right: lower cylinder force (N).

Secondly, another model (fig. 4.98) is created and actuated by 70% of the human emulated torque in clockwise direction on the rotational joints and with the kinematics on the front cylinder and lower back cylinder together. This is because if only the front cylinder is actuated by kinematics, there will be no constraints to the motion of the hip joint as it is controlled only by the lower back cylinder. So, in this case the front cylinder and the lower back cylinder are used to actuate the 70% of the required torques in clockwise direction. Thus, the calculated force for

the lower back cylinder in figure (4.97 Right) is just a component of the total force required by this cylinder, while the total force (fig. 4.99) is the superposition of the two forces calculated in two models. The lower back cylinder force along with the front cylinder force (fig. 4.100) assist the exoskeleton to produce the 100% emulated human torque.

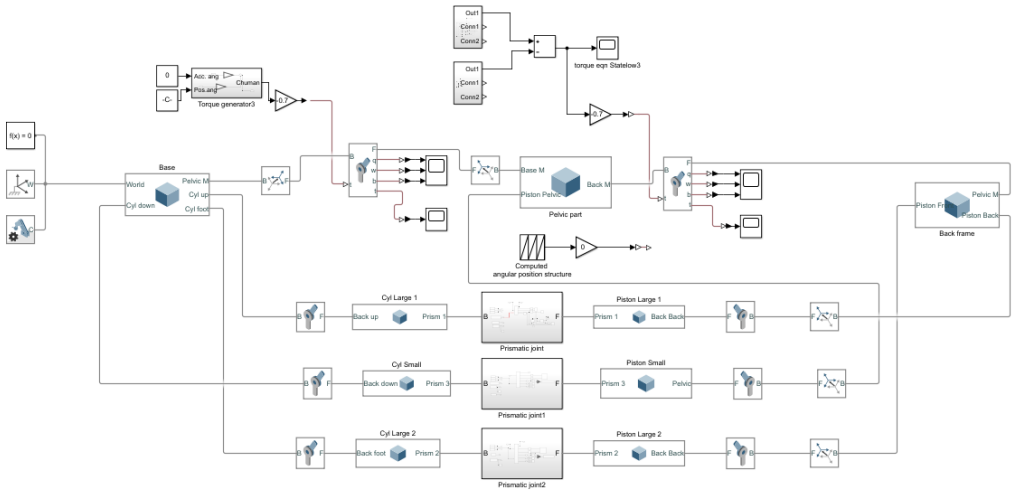


Figure 4.98: Return of the pelvic joint to zero degrees and 70% torque calculation.

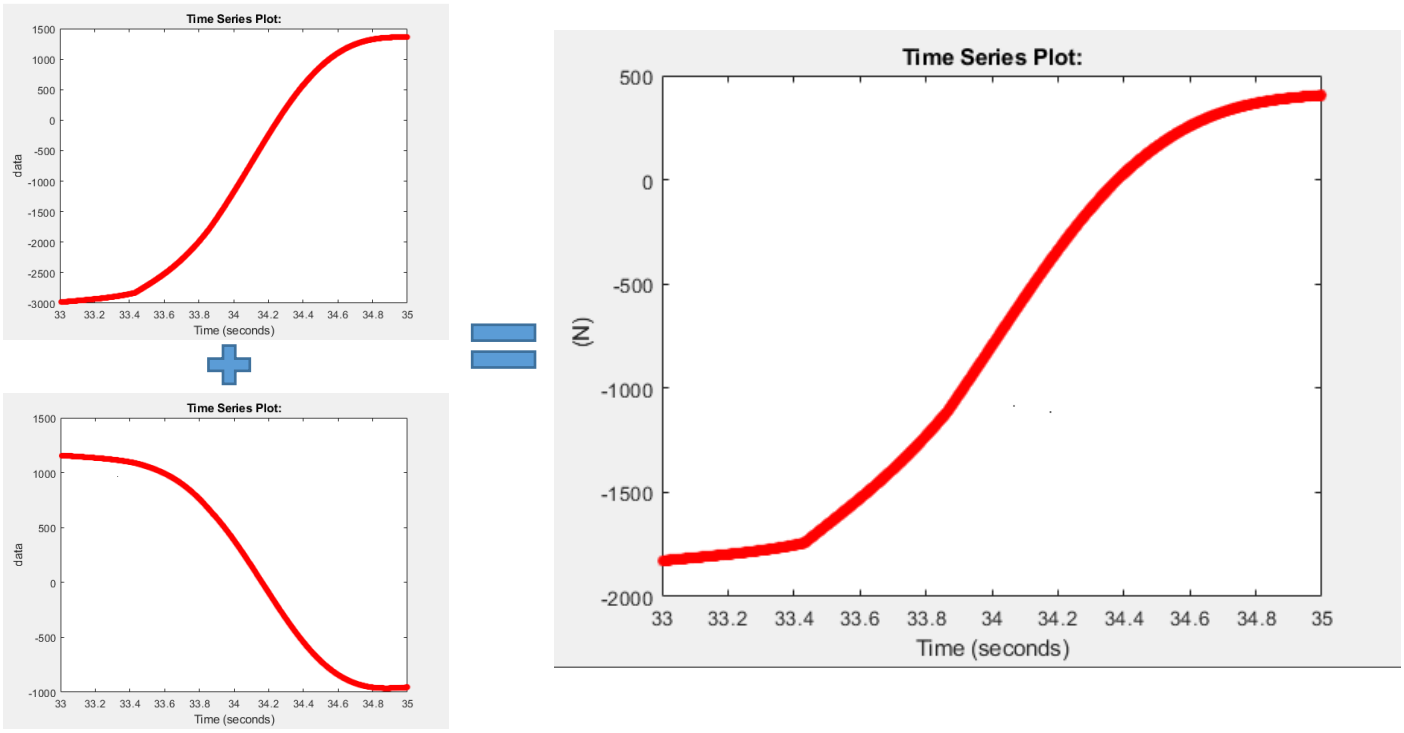


Figure 4.99: Summation of the lower back cylinder forces in fourth state and the final force required from this cylinder at this state (Forces in N).

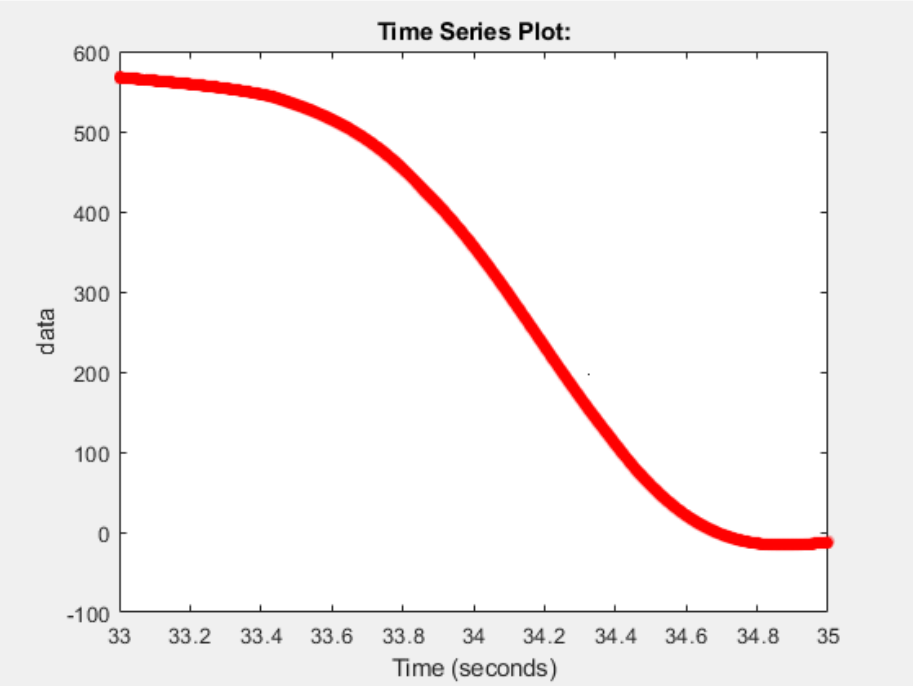


Figure 4.100: Front cylinder force (N).

4.3.5 Fifth state (Return to zero)

During this state, the testbench model (fig. 4.101) return the final 20° on the hip joint to reach the upright position. The back cylinder is used to generate the 100% human torque as the previous state and the front cylinder is used along the back lower cylinder to produce the reaction 70% which is added to the exoskeleton 30% of the torque.

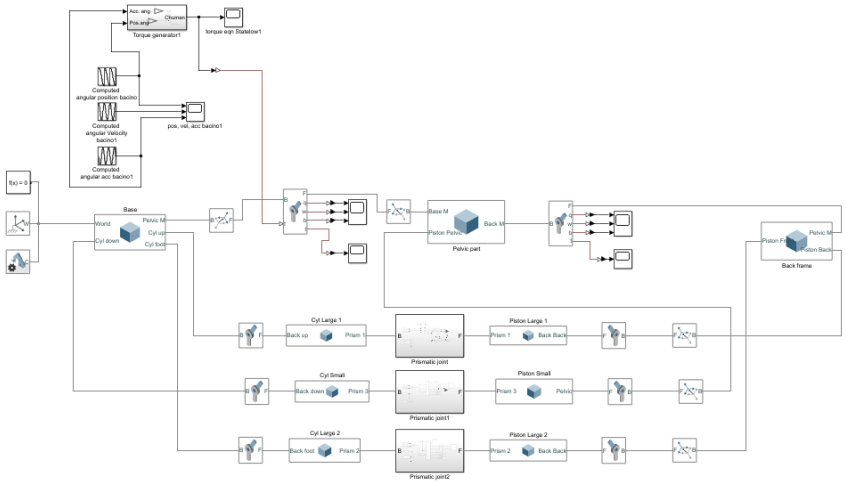


Figure 4.101: Return to upright position.

The same procedure of calculating the forces needed to produce 70% of the torque in the previous state is used in this state as well. Where the back cylinders produce forces (fig. 4.102) that produce 100% of the torque in counterclockwise direction. While another model (fig. 4.103) where, the lower back cylinder along with the frontal one are used to produce 70% in the clockwise direction. Finally, forces calculated on the two models in the lower back cylinder are also added together to provide the total force (fig. 4.104) on this cylinder. So, the front cylinder force (fig.4.105) is integrated with the lower back one to assist the exoskeleton 30% of the torque.

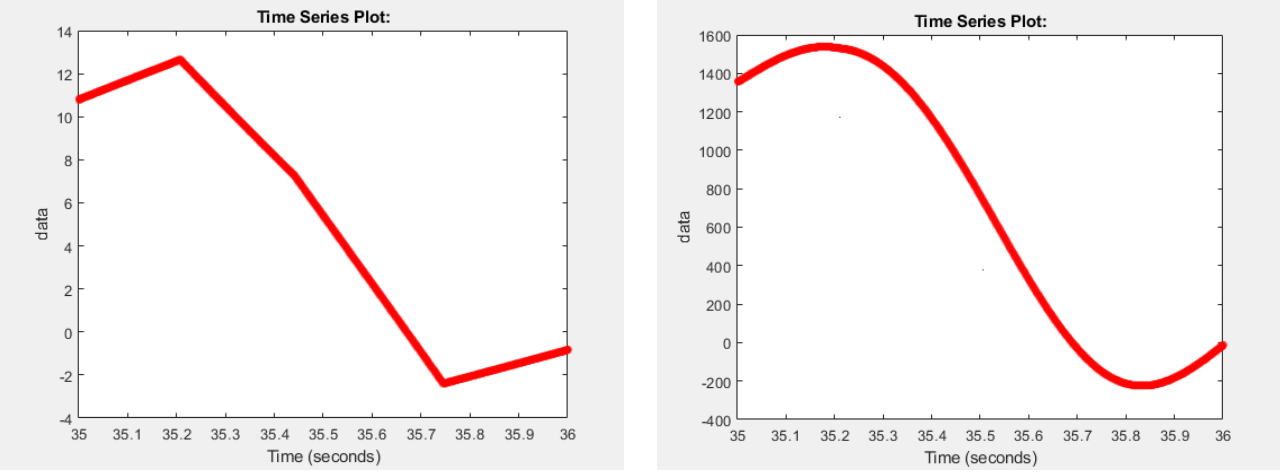


Figure 4.102: Left: upper cylinder force (N), Right: lower cylinder force (N).

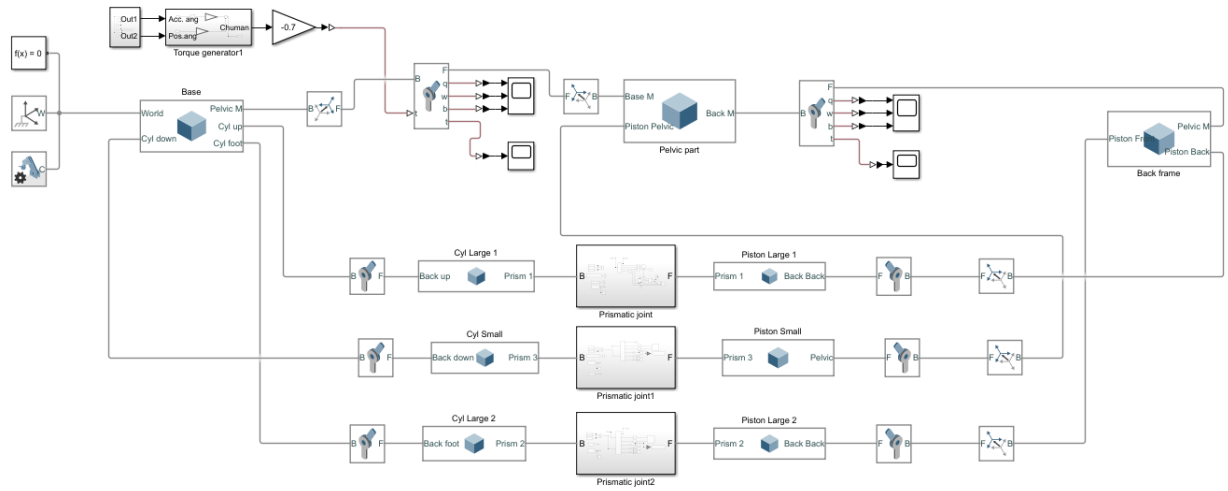


Figure: 4.103: Return to upright position and 70% torque calculation.

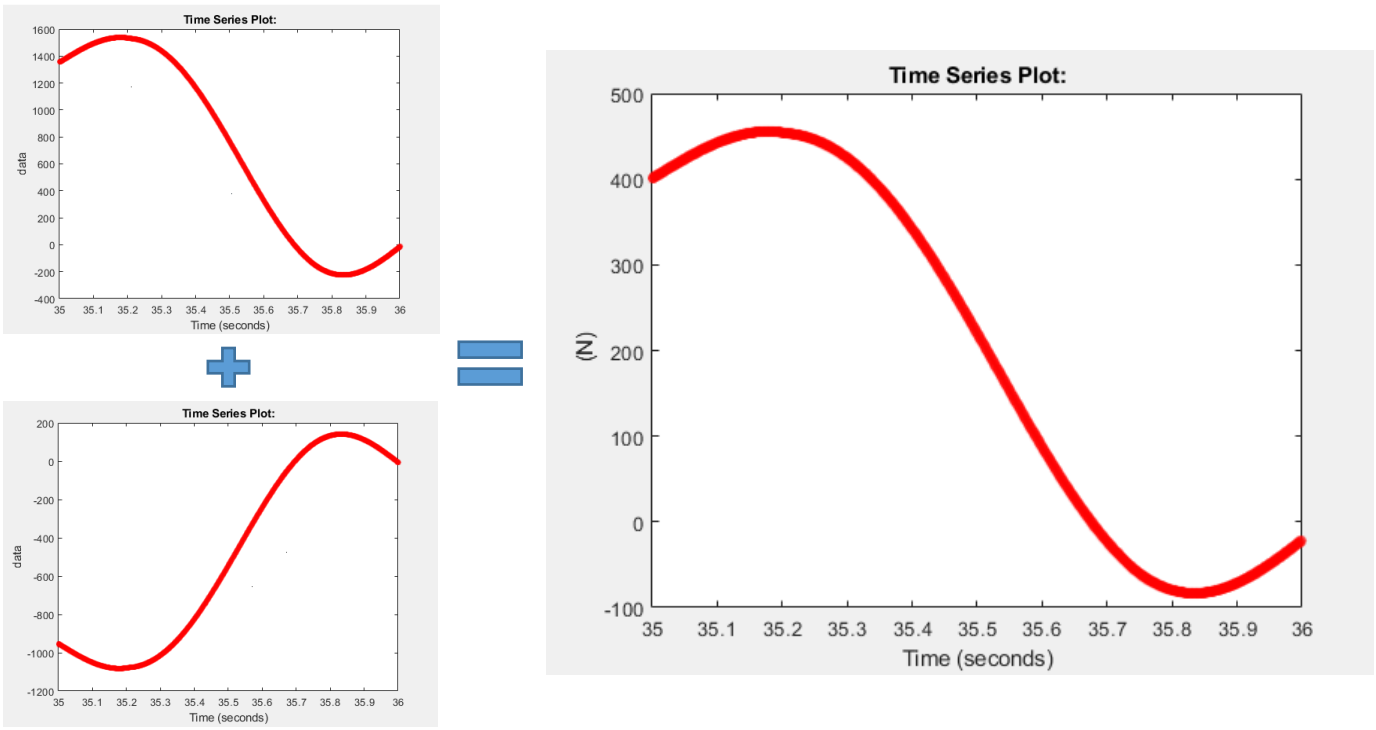


Figure 4.104: Summation of the lower back cylinder forces in fifth state and the final force required from this cylinder at this state (Forces in N).

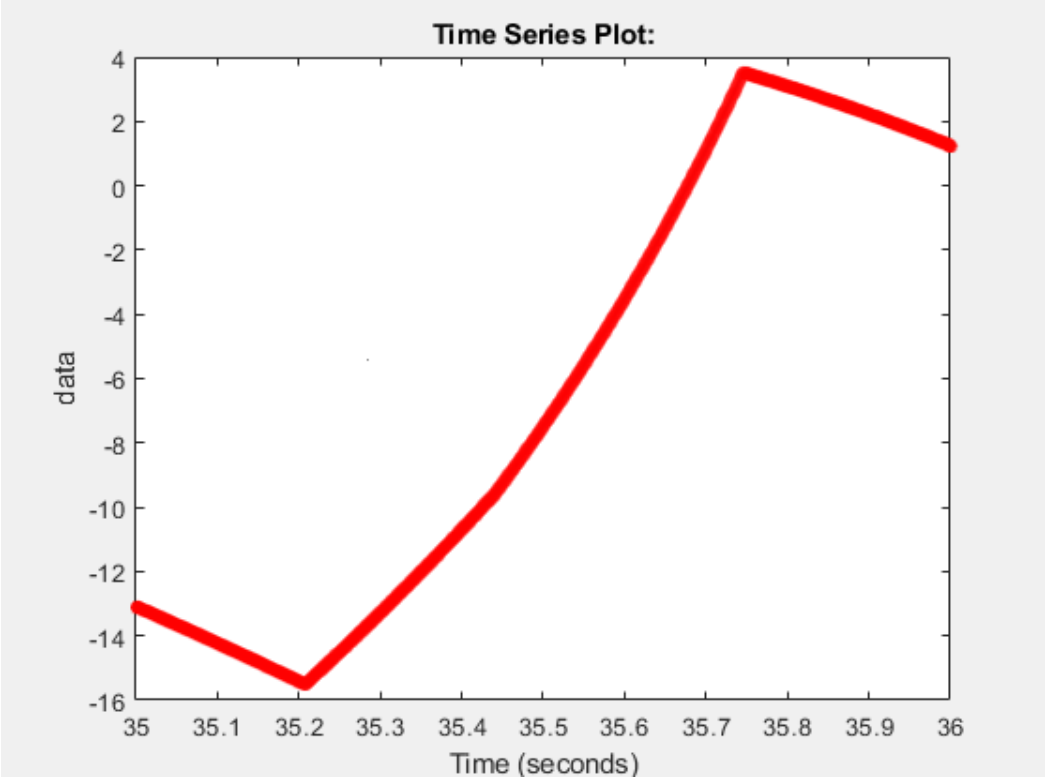


Figure 4.105: Front cylinder force (N).

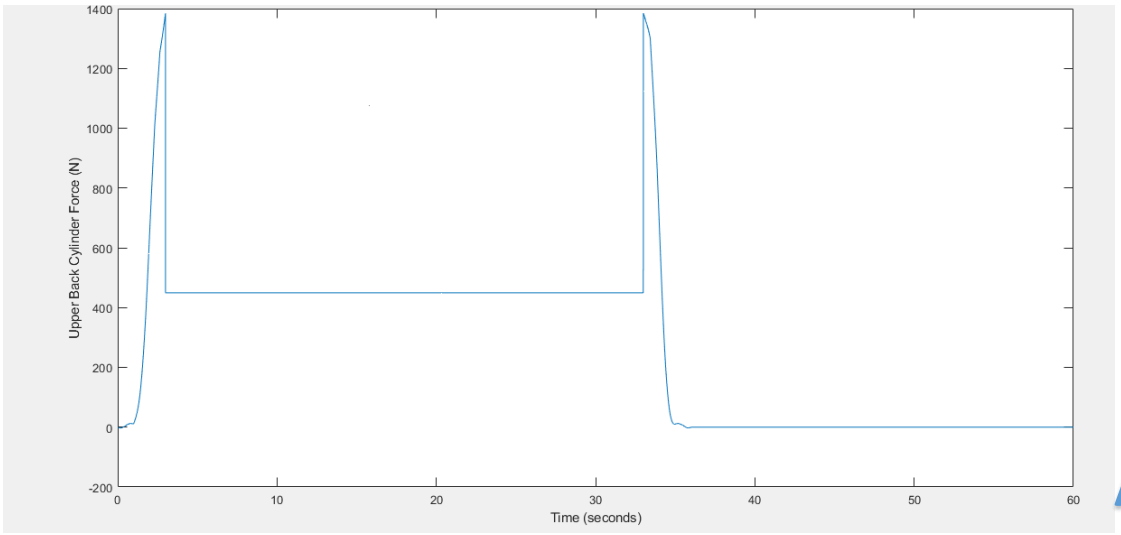
4.3.6 Sixth state (Resting at zero)

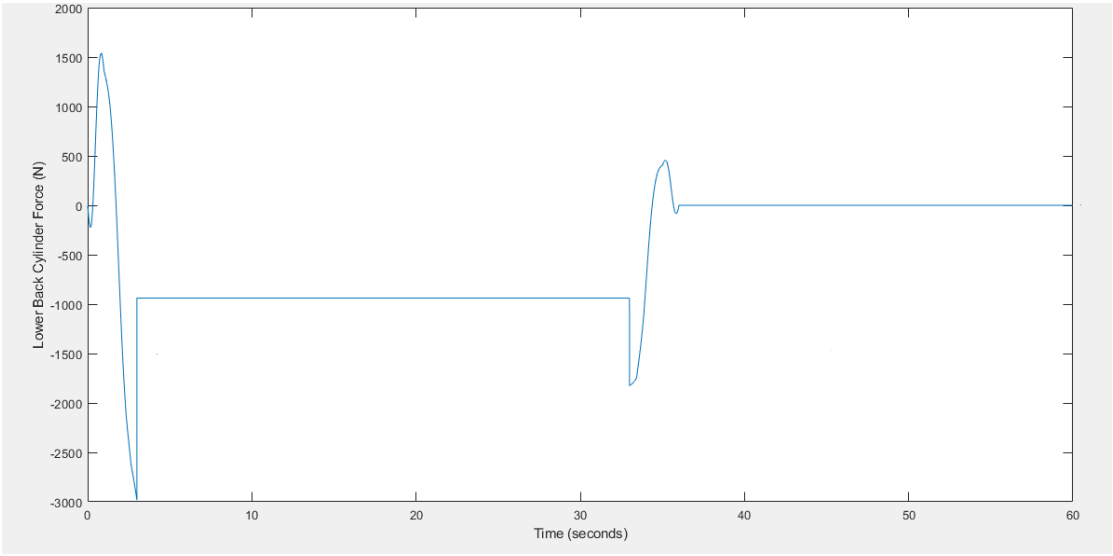
At this state, all the values are zero. The forces are zero, the torques are zero and the testbench is standing in the upright position and there is no motion due to friction between the joints of the testbench and the exoskeleton is not actuated. This state is added to simulate the total 60s that were defined in the simulation models of the exoskeleton with the human body. This state is 24s (36s – 60s).

Finally, all the previous force calculations were tested with other models of the testbench where the forces are applied as input and the kinematics are applied on the rotational joints at the correct times and the torque on each rotational joint is measured and compared to the initial requirements. In the first three states the torque calculations were simple as the frontal cylinder is always passive. However the fourth and fifth states, the testing was done by the same method and the output torques on the two rotational joints (hip and pelvic joints) were added together to check the total torque to be always 30% of the emulated human torque but in the clockwise direction which is perfectly correct as this is the torque produced by the exoskeleton when it is tested on the testbench.

4.3.7 Full simulation

Finally, after analytically calculating all the forces of all the three cylinders (fig. 4.106A, B, C). They are used to actuate the model and the rotational joints are actuated with kinematics to measure torques on both joints. (fig. 4.107A, B)





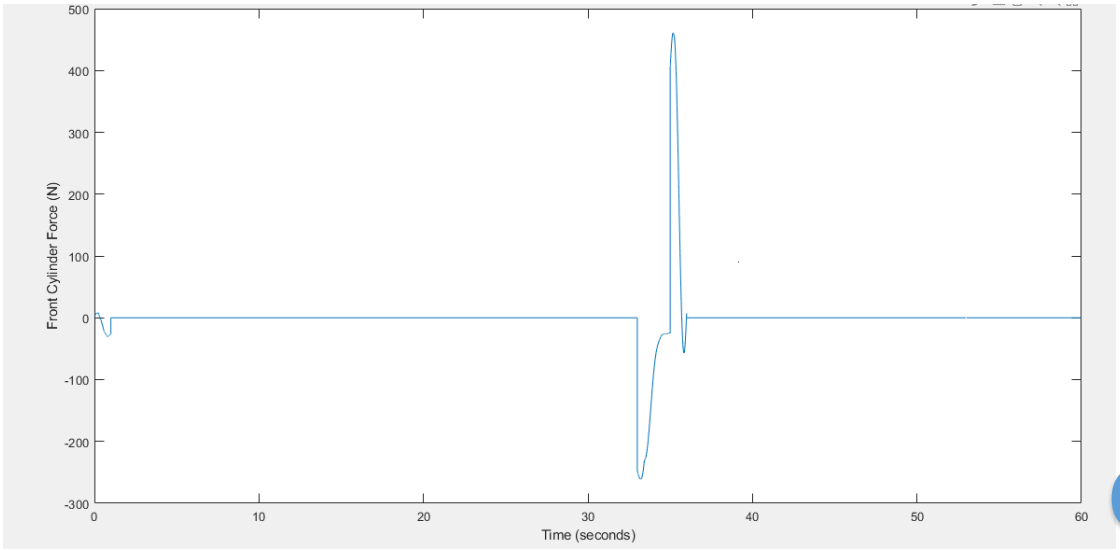


Figure 4.106: A) Upper back cylinder force. B) Lower back cylinder force. C) Front cylinder force.

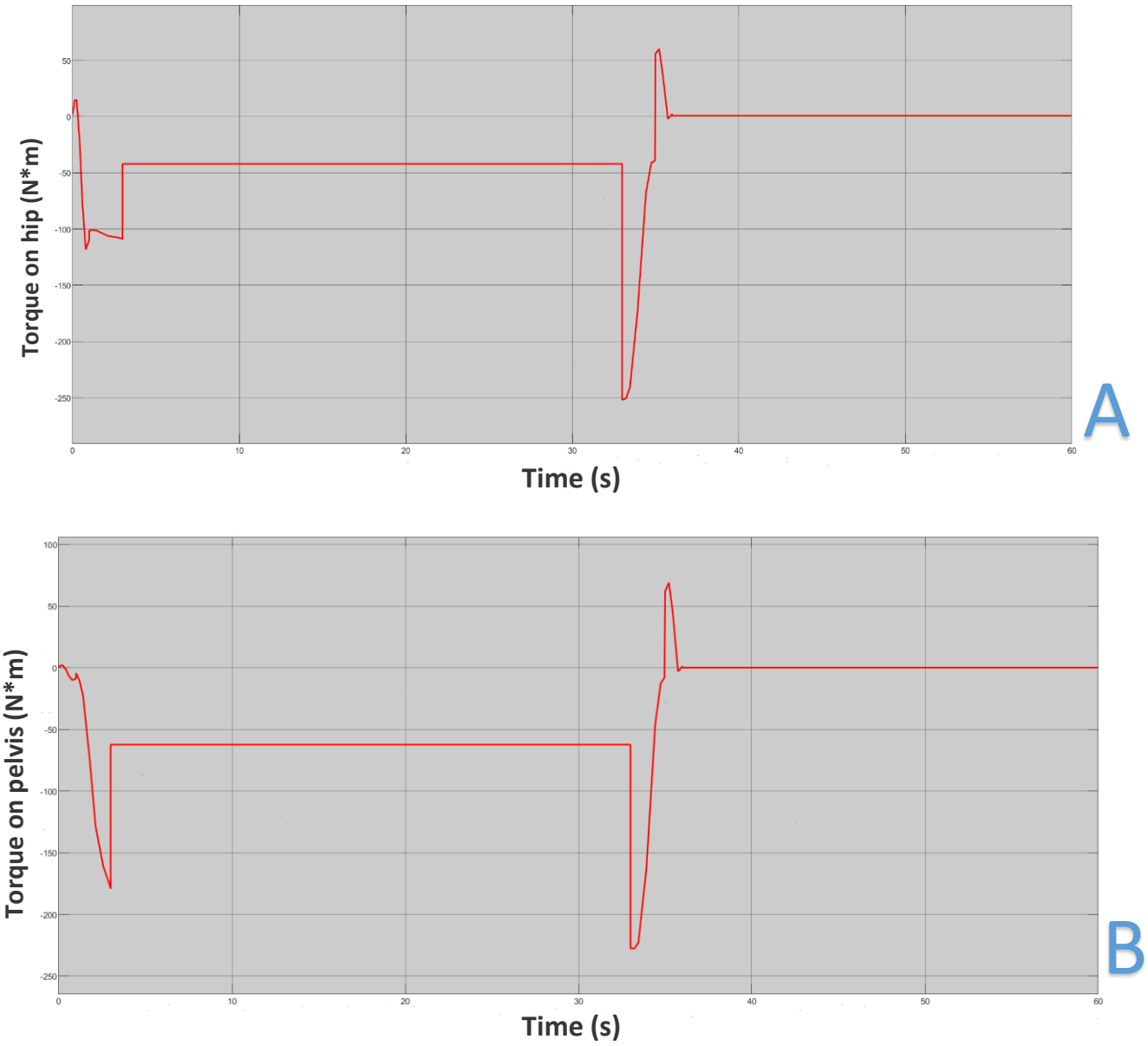


Figure 4.107: A) Torque on the hip joint. B) Torque on the pelvic joint.

5 Conclusion and future development

The design of exoskeletons is perfect mechatronic project as it encounters several fields as control, simulation, embedded software and electronics. In this thesis study, developments were added to a work in progress project of an industrial exoskeleton and designing a testbench to test the device kinematics and dynamics. The thesis starts with defining the control law to be applied and optimizing the selection of a control strategy by comparing between different methods and implementing this strategy in simulation through stateflow. Afterwards, stateflow was used to define the states of operation of the device and actuate it through these states in simulation and translating these states to logic operations to be implemented on the dedicated hardware. Secondly, the peripherals of the systems were defined as a 9-axis sensor which is used to measure the angular, position, velocity and acceleration on two rotational joints that emulate the hip and pelvic joints and calibrating this sensor from BOSCH to the exoskeleton system. Then, using python visual library (Appendix H), designing a simple model of the exoskeleton and visualizing the real-time motion of the sensor using data logging to the computer into python and MATLAB. Moving forward to designing an electronic board to accurately process the signal of the controller to actuate the proportional pressure regulator of FESTO to precisely actuate the control law and implementing these electronic devices on a PCB (Appendix L). Finally, designing and simulating the testbench on Simscape multibody tool and use the model to calculate and verify the forces that are to actuate the device into the correct

function. Last but not least, the exoskeleton and the testbench were assembled in the lab during this study.

For future developments, the control law should be actuated on the Arduino board and connected to exoskeleton device to do the required function. A load cell can be used to measure the relative force between the exoskeleton and the wearer which will contribute to the control law to perform the testing on the device. The testbench cylinder forces should be actuated using a controller on the cylinders and controlled through another pressure regulator valve or a flow control valve and a controller should be designed to actuate the forces. Finally, HIL testing should be implemented and perform more development to the device, if needed.

Appendices

Appendix A

TEXAS INSTRUMENTS

LM158-N, LM258-N, LM2904-N, LM358-N

SNOSBT3I – JANUARY 2000 – REVISED DECEMBER 2014

LMx58-N Low-Power, Dual-Operational Amplifiers

1 Features

- Available in 8-Bump DSBGA Chip-Sized Package, (See AN-1112, SNVA009)
- Internally Frequency Compensated for Unity Gain
- Large DC Voltage Gain: 100 dB
- Wide Bandwidth (Unity Gain): 1 MHz (Temperature Compensated)
- Wide Power Supply Range:
  - Single Supply: 3V to 32V
  - Or Dual Supplies: ±1.5V to ±16V
- Very Low Supply Current Drain (500 μA)—Essentially Independent of Supply Voltage
- Low Input Offset Voltage: 2 mV
- Input Common-Mode Voltage Range Includes Ground
- Differential Input Voltage Range Equal to the Power Supply Voltage
- Large Output Voltage Swing
- Unique Characteristics:
  - In the Linear Mode the Input Common-Mode Voltage Range Includes Ground and the Output Voltage Can Also Swing to Ground, even though Operated from Only a Single Power Supply Voltage.
  - The Unity Gain Cross Frequency is Temperature Compensated.
  - The Input Bias Current is also Temperature Compensated.
- Advantages:
  - Two Internally Compensated Op Amps
  - Eliminates Need for Dual Supplies
  - Allows Direct Sensing Near GND and V_{OUT} Also Goes to GND
  - Compatible with All Forms of Logic
  - Power Drain Suitable for Battery Operation

2 Applications

- Active Filters
- General Signal Conditioning and Amplification
- 4- to 20-mA Current Loop Transmitters

3 Description

The LM158 series consists of two independent, high gain, internally frequency compensated operational amplifiers which were designed specifically to operate from a single power supply over a wide range of voltages. Operation from split power supplies is also possible and the low power supply current drain is independent of the magnitude of the power supply voltage.

Application areas include transducer amplifiers, dc gain blocks and all the conventional op-amp circuits which now can be more easily implemented in single power supply systems. For example, the LM158 series can be directly operated off of the standard 3.3-V power supply voltage which is used in digital systems and will easily provide the required interface electronics without requiring the additional ±15V power supplies.

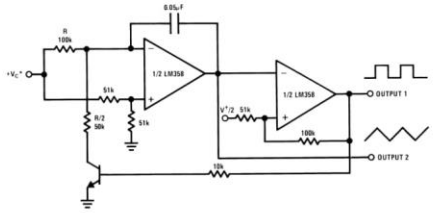
The LM358 and LM2904 are available in a chip sized package (8-Bump DSBGA) using TI's DSBGA package technology.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM158-N	TO-CAN (8)	9.08 mm x 9.09 mm
	CDIP (8)	10.16 mm x 6.502 mm
LM258-N	TO-CAN (8)	9.08 mm x 9.09 mm
LM2904-N	DSBGA (8)	1.31 mm x 1.31 mm
	SOIC (8)	4.90 mm x 3.91 mm
	PDIP (8)	9.81 mm x 6.35 mm
LM358-N	TO-CAN (8)	9.08 mm x 9.09 mm
	DSBGA (8)	1.31 mm x 1.31 mm
	SOIC (8)	4.90 mm x 3.91 mm
	PDIP (8)	9.81 mm x 6.35 mm

(1) For all available packages, see the orderable addendum at the end of the datasheet.

Voltage Controlled Oscillator (VCO)



Appendix B

TS7800 series

3-Terminal Fixed Positive Voltage Regulator

TO-220 ITO-220

Pin assignment:
1. Input
2. Ground
3. Output

(Heatsink surface connected to Pin 2)

Voltage Range 5V to 24V
Output Current up to 1A

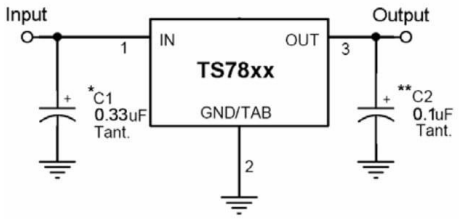
General Description

These voltage regulators are monolithic integrated circuits designed as fixed-voltage regulators for a wide variety of applications including local, on-card regulation. These regulators employ internal current limiting, thermal shutdown, and safe-area compensation. With adequate heatsink they can deliver output currents up to 1 ampere.
Although designed primarily as a fixed voltage regulator, these devices can be used with external components to obtain adjustable voltages and currents.
This series is offered in 3-pin TO-220, ITO-220 package.

Features

- ✧ Output current up to 1A
- ✧ No external components required
- ✧ Internal thermal overload protection
- ✧ Internal short-circuit current limiting
- ✧ Output transistor safe-area compensation
- ✧ Output voltage offered in 4% tolerance

Standard Application



A common ground is required between the input and the output voltages. The input voltage must remain typically 2.0V above the output voltage even during the low point on the Input ripple voltage.
XX = these two digits of the type number indicate voltage.
* = Cin is required if regulator is located an appreciable distance from power supply filter.
** = Co is not needed for stability; however, it does improve transient response.

Ordering Information

Part No.	Operating Temp. (Ambient)	Package
TS78xxCZ	-20 ~ +85°C	TO-220
TS78xxCI		ITO-220

Note: Where xx denotes voltage option.

Absolute Maximum Rating

Input Voltage		Vin *	35	V
Input Voltage		Vin **	40	V
Power Dissipation	TO-220	Without heatsink	2	W
	TO-220	Pt ***	15	
	ITO-220	Without heatsink	10	
Operating Junction Temperature Range		T_J	0 ~ +150	°C
Storage Temperature Range		T_{STG}	-65 ~ +150	°C

Note : * TS7805 to TS7818
** TS7824
*** Follow the derating curve

Appendix C

```
import matplotlib.pyplot as plt
from scipy import stats

x = [0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100,105,110,115,120,125,130,135,140,145,150]
y = [0.32,0.51,0.685,0.905,1.1,1.295,1.49,1.7,1.85,2.065,2.25,2.47,2.625,2.82,3.025,3.245,3.42,3.62,3.82,4.03,4.195,4.39,4.58,4.8,4.95,5.165,5.36,5.57,5.74,5.95,6.06]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
print(slope,intercept,r)
```

Appendix D

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>


uint16_t BNO055_SAMPLERATE_DELAY_MS = 10; //how often to read data from the
board
uint16_t PRINT_DELAY_MS = 500; // how often to print the data
uint16_t printCount = 0; //counter to avoid printing every 10MS sample

// Check I2C device address and correct line below (by default address is 0x29
or 0x28)
//                                   id, address
Adafruit_BNO055 bno1 = Adafruit_BNO055(55, 0x28);
Adafruit_BNO055 bno2 = Adafruit_BNO055(56, 0x29);

//input/output pins
int valve = 0;

// variables
String state = "start";
bool PowerON = true;
float alfaB = 0 , wB = 0 , alfaS = 0 , wS = 0 , accB = 0 , accS = 0;
float p_alfaB = 0 , p_wB = 0 , p_alfaS = 0 , p_wS = 0;
float millisOld;
//int nowTime = 0 , p_nowTime = 0;
float C , Setpoint;

//constants
const int m = 57;        // massa tronco (kg)
const float g = 9.81;          // accelerazione gravitazionale (m/s^2)
const float Lg = 0.5;           // posizione baricentro (m)
```

```
const float Inerzia = 7.7;            // inerzia sistema (kgm ^2) --> (I+mLg^2)
const float k = 0.3;                // effetto utile esoscheletro--> per
riduzione 30%
const int i_rid = 100;         // rapporto di riduzione harmonic drive

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  if(!bno1.begin() ||!bno2.begin())
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C
ADDR!");
    while(1);
  }

  delay(1000);

  Serial.println(state);
  analogWrite(valve,0);
  millisOld=millis();
  read_angles();
}

void loop() {
  // put your main code here, to run repeatedly:

  while ( ( 0 < alfaB <= 5 ) && ( 0 < alfaS <= 5 ) )
  {
    if (state == "LIFT........." || PowerON)
    {
      PowerON = false;
      state = "REST.........";
      analogWrite(valve,0);         //valve off
```

```
    Serial.println(state);
  }
  read_angles();
}

while ( ( 5 < alfaB <= 20 ) && ( 5 < alfaS <= 20 ) )
{
  if ( (wB > 0) && (state == "REST........." || state == "WORK.........") )
  {
    state = "BENDING......";
    Serial.println(state);
    analogWrite(valve,0);            //valve off
  }
  if ( (wB = 0) && (state == "BENDING......" || state == "LIFT.........") )
  {
    state = "WORK.........";
    Serial.println(state);
    analogWrite(valve,0);               //valve off
  }
  if ( (wB < 0) && (state == "WORK........." || state== "LIFT.........") )
  {
    state = "LIFT.........";
    Serial.println(state);
    airmotor( alfaB , accB );            //valve work
  }
  read_angles();
}

while ( ( 5 < alfaB ) && ( 20 < alfaS < 70 ) )
{
  if ( (wS > 0) && (state == "BENDING......" || state == "WORKbacino...") )
  {
    state = "BENDINGbacino";
    Serial.println(state);
    analogWrite(valve,0);               //valve off
```

```
    }
    if ( (wS = 0) && (state == "BENDINGbacino" || state == "LIFT.........") )
    {
      state = "WORKbacino...";
      Serial.println(state);
      analogWrite(valve,0);            //valve off
    }
    if ( (wS < 0) && (state == "WORKbacino..." || state== "LIFT.........") )
    {
      state = "LIFT.........";
      Serial.println(state);
      airmotor( alfaS , accS );             //valve work
    }
    read_angles();
  }

  while (alfaS >= 70)
  {
    state = "exceed70.....";
    Serial.println(state);
    if ( wS < 0 )
    {
      state = "LIFT.........";
      Serial.println(state);
      airmotor( alfaS , accS );              //valve work
    }
    read_angles();
  }
}

/*********************************************************************
/* Method: read_angles
/* In: sensor1(bacino), sensor2(structure) : data read from i2c
/* Out: position, velcity, acceleration of bacino and structure sensors
/* Description: compues the velcity and acceleration values
```

```
/**************************************************************************/
void read_angles()
{
  unsigned long tStart = micros();
  //read sensor 1
  /* Get a new sensor event */
  sensors_event_t angVelData1 , angVelData2;
  bno1.getEvent(&angVelData1, Adafruit_BNO055::VECTOR_GYROSCOPE);
  bno2.getEvent(&angVelData2, Adafruit_BNO055::VECTOR_GYROSCOPE);
  /*sensors_event_t event1;
  bno1.getEvent(&event1);
  alfaB = event1.orientation.x;
  /* Display the floating point data */

  float grav_X1 = bno1.getVector(Adafruit_BNO055::VECTOR_GRAVITY).x();
  float grav_Y1 = bno1.getVector(Adafruit_BNO055::VECTOR_GRAVITY).y();
  float grav_X2 = bno2.getVector(Adafruit_BNO055::VECTOR_GRAVITY).x();
  float grav_Y2 = bno2.getVector(Adafruit_BNO055::VECTOR_GRAVITY).y();
  alfaB = - atan2 (grav_Y1/9.81 , grav_X1/9.81) /2 /3.141592654 *360;
  alfaS = - atan2 (grav_Y2/9.81 , grav_X2/9.81) /2 /3.141592654 *360;

  wB = angVelData1.gyro.z;
  wS = angVelData2.gyro.z;

  float dt=(millis()-millisOld)/1000;
  millisOld = millis();


  accB = (wB - p_wB)/dt;
  accS = (wS - p_wS)/dt;

  p_wB=wB;
  p_wS=wS;

  if (printCount * BNO055_SAMPLERATE_DELAY_MS >= PRINT_DELAY_MS)
```

```
{
  //enough iterations have passed that we can print the latest data
Serial.print("alfaB: ");
Serial.println(alfaB);
Serial.print("wB: ");
Serial.println(wB);
Serial.print("accB: ");
Serial.println(accB);
Serial.print("alfaS: ");
Serial.println(alfaS);
Serial.print("wS: ");
Serial.println(wS);
Serial.print("accS: ");
Serial.println(accS);

/*
  Serial.print(alfaB);
  Serial.print(" , ");
  Serial.print(wB);
  Serial.print(" , ");
  Serial.print(accB);
  Serial.print(" , ");
  Serial.print(alfaS);
  Serial.print(" , ");
  Serial.print(wS);
  Serial.print(" , ");
  Serial.print(accS);
 */

 printCount = 0;
}
else {
  printCount = printCount + 1;
}
while ((micros() - tStart) < (BNO055_SAMPLERATE_DELAY_MS * 1000))
```

```
  {
    //poll until the next sample is ready
  }
  /*
  //read sensor 2

  sensors_event_t event2;
  bno2.getEvent(&event2);
  alfaS = event2.orientation.x;

  Serial.print("alfaS: ");
  Serial.print(event2.orientation.x, 4);

  nowTime = millis();
  float interval = (double)( p_nowTime - nowTime );        // Calculate the time
it takes to run a cycle
  p_nowTime = nowTime;

  if (( 0 < alfaB <= 20 ) && ( 0 < alfaS <= 20 ))
  {
    wB =  1000 * ( p_alfaB - alfaB ) / interval;
    accB =  1000 * ( p_wB - wB ) / interval;
    p_alfaB = alfaB;
    p_wB = wB;
    Serial.println("alfa_B:");
    Serial.print(alfaB);
    Serial.println("      w_B:");
    Serial.print(wB);
    Serial.println("      acc_B:");
    Serial.print(accB);
  }
  else if (( 20 < alfaB ) && ( 20 < alfaS ))
  {
    wS =  1000 * ( p_alfaS - alfaS ) / interval;
    accS =  1000 * ( p_wS - wS ) / interval;
```

```
    p_alfaS = alfaS;
    p_wS = wS;
    Serial.println("alfa_S:");
    Serial.print(alfaS);
    Serial.println("      w_S:");
    Serial.print(wS);
    Serial.println("      acc_S:");
    Serial.print(accS);
  }

  */
}

/*********************************************************************
/* Method: compute the output to valves
/* In: position and acceleration of angles
/* Out: control value on valves (analogWrite)
/* Description: caomputes the control value depending on position
/*********************************************************************/
void airmotor(float pos, float acc)
{
  //C = m * g * Lg * sin((pos * 3.14 / 180)) - Inerzia * acc;  //torque equation
  //Setpoint = 0.3 * C;
  //add ctrl law here
  //analogWrite(valve,255);         //analogWrite values from 0 to 255
}
```

Appendix E

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>

float dt = 0 , millisOld = 0;
float velold , tilt, vel , grav_X , grav_Y , gamma;

double xPos = 0, yPos = 0, headingVel = 0;
uint16_t BNO055_SAMPLERATE_DELAY_MS = 10; //how often to read data from the
board
uint16_t PRINT_DELAY_MS = 500; // how often to print the data
uint16_t printCount = 0; //counter to avoid printing every 10MS sample

//velocity = accel*dt (dt in seconds)
//position = 0.5*accel*dt^2
double ACCEL_VEL_TRANSITION =  (double)(BNO055_SAMPLERATE_DELAY_MS) / 1000.0;
double ACCEL_POS_TRANSITION = 0.5 * ACCEL_VEL_TRANSITION * ACCEL_VEL_TRANSITION;
double DEG_2_RAD = 0.01745329251; //trig functions require radians, BNO055
outputs degrees

// Check I2C device address and correct line below (by default address is 0x29
or 0x28)
//                                   id, address
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);

void setup(void)
{
  Serial.begin(115200);
 // if (!bno.begin())
  if (!bno.begin(Adafruit_BNO055::OPERATION_MODE_NDOF))
  {
    Serial.print("No BNO055 detected");
    while (1);
```

```
  }

  delay(1000);
    millisOld=millis();
}

void loop(void)
{
   uint8_t system, gyros, accel, mg = 0;
  bno.getCalibration(&system, &gyros, &accel, &mg);
  //
  unsigned long tStart = micros();
  sensors_event_t orientationData , linearAccelData , angVelData ,
accelerometer;
  bno.getEvent(&orientationData, Adafruit_BNO055::VECTOR_EULER);
    bno.getEvent(&angVelData, Adafruit_BNO055::VECTOR_GYROSCOPE);
        bno.getEvent(&accelerometer, Adafruit_BNO055::VECTOR_ACCELEROMETER);
  bno.getEvent(&linearAccelData, Adafruit_BNO055::VECTOR_LINEARACCEL);
  xPos = xPos + ACCEL_POS_TRANSITION * linearAccelData.acceleration.x;
  yPos = yPos + ACCEL_POS_TRANSITION * linearAccelData.acceleration.y;

  // velocity of sensor in the direction it's facing
  headingVel = ACCEL_VEL_TRANSITION * linearAccelData.acceleration.x /
cos(DEG_2_RAD * orientationData.orientation.x);

  //float vec = bno.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER).x();

    //enough iterations have passed that we can print the latest data
  /*  Serial.print("Heading: ");
    Serial.println(orientationData.orientation.x);
    Serial.print("Headingy: ");
    Serial.println(orientationData.orientation.y);
    Serial.print("Headingz: ");
    Serial.println(orientationData.orientation.z);
```

```
    Serial.print("Position: ");
    Serial.print(xPos);
    Serial.print(" , ");
    Serial.println(yPos);
    Serial.print("Speed: ");
    Serial.println(headingVel);

    Serial.print("acc ");
    Serial.print(linearAccelData.acceleration.x);
    Serial.print(" , ");
    Serial.print(linearAccelData.acceleration.y);
    Serial.print(" , ");
    Serial.println(linearAccelData.acceleration.z);

    Serial.print("acclerometer ");
    Serial.print(accelerometer.acceleration.x);
    Serial.print(" , ");
    Serial.print(accelerometer.acceleration.y);
    Serial.print(" , ");
    Serial.println(accelerometer.acceleration.z);
*/

  grav_X = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).x();
  grav_Y = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).y();
 // float grav_Z = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).z();

  tilt = -atan2(grav_Y/9.81,grav_X/9.81)/2/3.141592654*360;
  vel = angVelData.gyro.z;
  dt=(millis()-millisOld)/1000;
    millisOld=millis();

   //Serial.println("-----------------------------------");
   // Serial.print(" dt ");
   // Serial.print(millis());
```

```
   //  Serial.print(" , ");
  //  Serial.print(millisOld);
  //  Serial.print(" , ");
   // Serial.println(dt);

  gamma = (vel - velold)/dt;
  velold = vel;

//Serial.println("-----------------------------------");
   // Serial.print(" Temp ");
   // Serial.println(bno.getTemp());
  // Serial.print("gravity: ");
  //  Serial.print(grav_X);
  //  Serial.print(" , ");
  //  Serial.print(grav_Y);
  //  Serial.print(" , ");
  //  Serial.println(grav_Z);
  //Serial.print(" Angular position:  ");
  // Serial.print(" , ");

    Serial.print(system);
    Serial.print(" , ");
    Serial.print(gyros);
    Serial.print(" , ");
    Serial.print(accel);
    Serial.print(" , ");
    Serial.print(mg);
    Serial.print(" , ");
    Serial.print(tilt);

//    Serial.print("Angular velocity: ");
    Serial.print(" , ");
    Serial.print(angVelData.gyro.z);
        Serial.print(" , ");
```

```
//    Serial.print("Angular acc: ");
    Serial.println(gamma);
  //Serial.println(vec);
  // Serial.print(" , ");
  //  Serial.println(modeback);
  //  Serial.println("-------");
}

void printEvent(sensors_event_t* event) {
  Serial.println();
  Serial.print(event->type);
  double x = -1000000, y = -1000000 , z = -1000000; //dumb values, easy to spot
problem
  if (event->type == SENSOR_TYPE_ACCELEROMETER) {
    x = event->acceleration.x;
    y = event->acceleration.y;
    z = event->acceleration.z;
  }
  else if (event->type == SENSOR_TYPE_ORIENTATION) {
    x = event->orientation.x;
    y = event->orientation.y;
    z = event->orientation.z;
  }
  else if (event->type == SENSOR_TYPE_MAGNETIC_FIELD) {
    x = event->magnetic.x;
    y = event->magnetic.y;
    z = event->magnetic.z;
  }
  else if ((event->type == SENSOR_TYPE_GYROSCOPE) || (event->type ==
SENSOR_TYPE_ROTATION_VECTOR)) {
    x = event->gyro.x;
    y = event->gyro.y;
    z = event->gyro.z;
  }
```

```
  Serial.print(": x= ");
  Serial.print(x);
  Serial.print(" | y= ");
  Serial.print(y);
  Serial.print(" | z= ");
  Serial.println(z);
}
```

Appendix F

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>

int zz;
float dt = 0 , millisOld = 0;
float velold , tilt, vel , grav_X , grav_Y , gamma;

double xPos = 0, yPos = 0, headingVel = 0;
uint16_t BNO055_SAMPLERATE_DELAY_MS = 10; //how often to read data from the
board
uint16_t PRINT_DELAY_MS = 50; // how often to print the data
uint16_t printCount = 0; //counter to avoid printing every 10MS sample

//velocity = accel*dt (dt in seconds)
//position = 0.5*accel*dt^2
double ACCEL_VEL_TRANSITION =  (double)(BNO055_SAMPLERATE_DELAY_MS) / 1000.0;
double ACCEL_POS_TRANSITION = 0.5 * ACCEL_VEL_TRANSITION * ACCEL_VEL_TRANSITION;
double DEG_2_RAD = 0.01745329251; //trig functions require radians, BNO055
outputs degrees

// Check I2C device address and correct line below (by default address is 0x29
or 0x28)
//                                   id, address
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);
void setup(void)
{
  Serial.begin(115200);
  // if (!bno.begin())
  if (!bno.begin(Adafruit_BNO055::OPERATION_MODE_NDOF))
  {
    Serial.print("No BNO055 detected");
    while (1);
```

```
  }
  delay(1000);
  millisOld = millis();
}

void loop(void)
{
  uint8_t system, gyros, accel, mg = 0;
  bno.getCalibration(&system, &gyros, &accel, &mg);
  //
  unsigned long tStart = micros();
  sensors_event_t orientationData , linearAccelData , angVelData ,
accelerometer;
  bno.getEvent(&orientationData, Adafruit_BNO055::VECTOR_EULER);
  bno.getEvent(&angVelData, Adafruit_BNO055::VECTOR_GYROSCOPE);
  bno.getEvent(&accelerometer, Adafruit_BNO055::VECTOR_ACCELEROMETER);
  bno.getEvent(&linearAccelData, Adafruit_BNO055::VECTOR_LINEARACCEL);
  xPos = xPos + ACCEL_POS_TRANSITION * linearAccelData.acceleration.x;
  yPos = yPos + ACCEL_POS_TRANSITION * linearAccelData.acceleration.y;

  // velocity of sensor in the direction it's facing
  headingVel = ACCEL_VEL_TRANSITION * linearAccelData.acceleration.x /
cos(DEG_2_RAD * orientationData.orientation.x);

  //float vec = bno.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER).x();

  if (printCount * BNO055_SAMPLERATE_DELAY_MS >= PRINT_DELAY_MS) {
    //enough iterations have passed that we can print the latest data

    grav_X = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).x();
    grav_Y = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).y();
    // float grav_Z = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).z();

    tilt = -atan2(grav_Y / 9.81, grav_X / 9.81) / 2 / 3.141592654 * 360;
    vel = angVelData.gyro.z;
```

```
    dt = (millis() - millisOld) / 1000;
    millisOld = millis();

    gamma = (vel - velold) / dt;
    velold = vel;

    Serial.print(system);
    Serial.print(" , ");
    Serial.print(gyros);
    Serial.print(" , ");
    Serial.print(accel);
    Serial.print(" , ");
    Serial.print(mg);
    Serial.print(" , ");
    Serial.print(tilt);
    Serial.print(" , ");
    Serial.print(angVelData.gyro.z);
    Serial.print(" , ");
    Serial.print(gamma);
    Serial.print(" , ");
    Serial.print(millis());
    Serial.print(" , ");
    Serial.println(millis()-zz);
    zz=millis();
    printCount = 0;
  }
  else {
    printCount = printCount + 1;
  }

  while ((micros() - tStart) < (BNO055_SAMPLERATE_DELAY_MS * 1000))
  {
    //poll until the next sample is ready
  }
}
```

```
void printEvent(sensors_event_t* event) {
  Serial.println();
  Serial.print(event->type);
  double x = -1000000, y = -1000000 , z = -1000000; //dumb values, easy to spot
problem
  if (event->type == SENSOR_TYPE_ACCELEROMETER) {
    x = event->acceleration.x;
    y = event->acceleration.y;
    z = event->acceleration.z;
  }
  else if (event->type == SENSOR_TYPE_ORIENTATION) {
    x = event->orientation.x;
    y = event->orientation.y;
    z = event->orientation.z;
  }
  else if (event->type == SENSOR_TYPE_MAGNETIC_FIELD) {
    x = event->magnetic.x;
    y = event->magnetic.y;
    z = event->magnetic.z;
  }
  else if ((event->type == SENSOR_TYPE_GYROSCOPE) || (event->type ==
SENSOR_TYPE_ROTATION_VECTOR)) {
    x = event->gyro.x;
    y = event->gyro.y;
    z = event->gyro.z;
  }

  Serial.print(": x= ");
  Serial.print(x);
  Serial.print(" | y= ");
  Serial.print(y);
  Serial.print(" | z= ");
  Serial.println(z);
}
```

Appendix G

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>

float dt = 0 , millisOld = 0;
float velold , tilt, vel , grav_X , grav_Y , gammaM , gammaold , gammanew;

double xPos = 0, yPos = 0, headingVel = 0;
uint16_t BNO055_SAMPLERATE_DELAY_MS = 10; //how often to read data from the
board
uint16_t PRINT_DELAY_MS = 500; // how often to print the data
uint16_t printCount = 0; //counter to avoid printing every 10MS sample

//velocity = accel*dt (dt in seconds)
//position = 0.5*accel*dt^2
double ACCEL_VEL_TRANSITION =  (double)(BNO055_SAMPLERATE_DELAY_MS) / 1000.0;
double ACCEL_POS_TRANSITION = 0.5 * ACCEL_VEL_TRANSITION * ACCEL_VEL_TRANSITION;
double DEG_2_RAD = 0.01745329251; //trig functions require radians, BNO055
outputs degrees

// Check I2C device address and correct line below (by default address is 0x29
or 0x28)
//                                   id, address
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);

void setup(void)
{
  Serial.begin(115200);
 // if (!bno.begin())
  if (!bno.begin(Adafruit_BNO055::OPERATION_MODE_NDOF))
  {
    Serial.print("No BNO055 detected");
    while (1);
```

```
  }
  delay(1000);
    millisOld=millis();
}

void loop(void)
{
   uint8_t system, gyros, accel, mg = 0;
  bno.getCalibration(&system, &gyros, &accel, &mg);
  //
  unsigned long tStart = micros();
  sensors_event_t orientationData , linearAccelData , angVelData ,
accelerometer;
  bno.getEvent(&orientationData, Adafruit_BNO055::VECTOR_EULER);
  bno.getEvent(&angVelData, Adafruit_BNO055::VECTOR_GYROSCOPE);
  bno.getEvent(&accelerometer, Adafruit_BNO055::VECTOR_ACCELEROMETER);
  bno.getEvent(&linearAccelData, Adafruit_BNO055::VECTOR_LINEARACCEL);
  xPos = xPos + ACCEL_POS_TRANSITION * linearAccelData.acceleration.x;
  yPos = yPos + ACCEL_POS_TRANSITION * linearAccelData.acceleration.y;

  // velocity of sensor in the direction it's facing
  headingVel = ACCEL_VEL_TRANSITION * linearAccelData.acceleration.x /
cos(DEG_2_RAD * orientationData.orientation.x);

  grav_X = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).x();
  grav_Y = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).y();
 // float grav_Z = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY).z();

  tilt = -atan2(grav_Y/9.81,grav_X/9.81)/2/3.141592654*360;
  vel = angVelData.gyro.z;
  dt=(millis()-millisOld)/1000;
  millisOld=millis();

  gammaM = (vel - velold)/dt;
  velold = vel;
```

```
  gammanew = .9*gammaold + .1*gammaM ;
  gammaold = gammanew;

    Serial.print(system);
    Serial.print(" , ");
    Serial.print(gyros);
    Serial.print(" , ");
    Serial.print(accel);
    Serial.print(" , ");
    Serial.print(mg);
    Serial.print(" , ");
    Serial.print(tilt);
    Serial.print(" , ");
    Serial.print(angVelData.gyro.z);
    Serial.print(" , ");
    Serial.println(gammanew);
}

void printEvent(sensors_event_t* event) {
  Serial.println();
  Serial.print(event->type);
  double x = -1000000, y = -1000000 , z = -1000000; //dumb values, easy to spot
problem
  if (event->type == SENSOR_TYPE_ACCELEROMETER) {
    x = event->acceleration.x;
    y = event->acceleration.y;
    z = event->acceleration.z;
  }
  else if (event->type == SENSOR_TYPE_ORIENTATION) {
    x = event->orientation.x;
    y = event->orientation.y;
    z = event->orientation.z;
  }
  else if (event->type == SENSOR_TYPE_MAGNETIC_FIELD) {
```

```
    x = event->magnetic.x;
    y = event->magnetic.y;
    z = event->magnetic.z;
  }
  else if ((event->type == SENSOR_TYPE_GYROSCOPE) || (event->type ==
SENSOR_TYPE_ROTATION_VECTOR)) {
    x = event->gyro.x;
    y = event->gyro.y;
    z = event->gyro.z;
  }

  Serial.print(": x= ");
  Serial.print(x);
  Serial.print(" | y= ");
  Serial.print(y);
  Serial.print(" | z= ");
  Serial.println(z);
}
```

Appendix H

```
from vpython import*
from time import *
import serial

ad=serial.Serial('com3',115200)
sleep(1)

scene.range=20
toRad=2*pi/360
toDeg=1/toRad
scene.forward=vector(1,0,0)

scene.width=600
scene.height=600
pos_labl = label( pos=vector(0,15,15), xoffset=10 , yoffset=50, text="Tilt =
{:.3f}".format(1) , height=16 , border=4 )
vel_labl = label( pos=vector(0,13,15), xoffset=10 , yoffset=50, text="Vel =
{:.3f}".format(1) )
acc_labl = label( pos=vector(0,11,15), xoffset=10 , yoffset=50, text="Acc =
{:.3f}".format(1) )

pos_display = graph(xtitle="Time", ytitle="Position")
pos_curve = gcurve(color=color.red, label = "Position") # Plot with datapoints
connected.
vel_display = graph(xtitle="Time", ytitle="Velocity")
vel_curve = gcurve(color=color.green, label = "Velocity") # Plot with datapoints
connected.
acc_display = graph(xtitle="Time", ytitle="Acceleration")
acc_curve = gcurve(color=color.blue, label = "Acceleration") # Plot with
datapoints connected.

xarrow=arrow(lenght=2, shaftwidth=.1, color=color.red,axis=vector(1,0,0))
yarrow=arrow(lenght=2, shaftwidth=.1, color=color.green,axis=vector(0,1,0))
```

```
zarrow=arrow(lenght=4, shaftwidth=.1, color=color.blue,axis=vector(0,0,1))


sphere(radius=0.1)
circle = shapes.circle(radius=0.05)
lumbar1 = extrusion(pos=vector(-5,0,0), path = [vector(-5,0,0),vector(-
5,3,0),vector(-5,3,-4),vector(-5,0,0)], shape = circle , color=color.green)
lumbar2=lumbar1.clone(pos=vector(5,0,0))
sphere(radius=0.2,pos=lumbar1.pos)
lumbar=compound([lumbar1,lumbar2])

backframel1=cylinder(axis=vector(0,1,0),pos=vector(-5,1.5,2.05),
size=vector(10,0.1,0.1),color=color.red)
backframel2=cylinder(axis=vector(1,0,0),pos=vector(-5,10.5,2.05),
size=vector(5,0.1,0.1),color=color.red)
backframer1=backframel1.clone(pos=vector(5,1.5,2.05))
backframer2=cylinder(pos=vector(5,10.5,2.05),axis=vector(-1,0,0),
size=vector(5,0.1,0.1),color=color.red)
backframeup=compound([backframel1,backframel2,backframer1,backframer2])
backframedown=backframeup.clone(pos=vector(0, -3.5,
2.05),color=color.red,opacity=0)
backframedown.rotate(axis=vector(1,0,0), angle=pi)
backframe=compound([backframeup,backframedown])


leglinkl=cylinder(axis=vector(0,-1,0),pos=vector(-5,-1.5,2.05),
size=vector(5,0.1,0.1),color=color.red)
leglinkld=cylinder(axis=vector(0,1,0),pos=vector(-5,-1.5,2.05),
size=vector(5,0.1,0.1),color=color.red,opacity=0)
leglinkr=cylinder(axis=vector(0,-1,0),pos=vector(5,-1.5,2.05),
size=vector(5,0.1,0.1),color=color.red)
leglinkrd=cylinder(axis=vector(0,1,0),pos=vector(5,-1.5,2.05),
size=vector(5,0.1,0.1),color=color.red,opacity=0)
leglink=compound([leglinkl,leglinkr,leglinkld,leglinkrd])
```

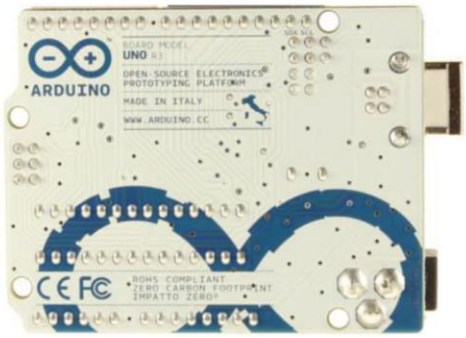
```
angle1old=0
angle2old=0
time = 0 # Time in simulation.
dt = 0.1 # Time step size.

while (time <= 1000):
    while (ad.inWaiting()==0):
        pass
    dataPacket=ad.readline()
    dataPacket=str(dataPacket, 'utf-8')
    splitPacket=dataPacket.split(',')
    systemCal1=float (splitPacket[0])
    gyroCal1=float (splitPacket[1])
    accelCal1=float (splitPacket[2])
    mgCal1=float (splitPacket[3])
    angle1=-float (splitPacket[4])
    speed1=float (splitPacket[5])
    acc1=float (splitPacket[6])
    #print("I am in endless loop")

#print("SCal1=",systemCal1,"GCal1=",gyroCal1,"ACal1=",accelCal1,"MCal1=",mgCal1,
"angle1=",angle1,"speedz1",speed1,"acc1=",acc1)

    backframe.rotate(axis=vector(1,0,0), angle=(angle1*pi/180)-
(angle1old*pi/180))
    #leglink.rotate(axis=vector(1,0,0), angle=pi/4)

    ##text
    pos_labl.text="Tilt = {:.3f}".format(angle1)
    vel_labl.text="Vel = {:.3f}".format(speed1)
    acc_labl.text="Acc = {:.3f}".format(acc1)
```

```
    '''
    pos_labl.text="Tilt = {:.3f}".format(angle2)
    vel_labl.text="Vel = {:.3f}".format(speed2)
    acc_labl.text="Acc = {:.3f}".format(acc2)
    '''



    ##plots
    pos_curve.plot(pos=(time, angle1))
    vel_curve.plot(pos=(time, speed1))
    acc_curve.plot(pos=(time, acc1))

    '''
    pos_curve.plot(pos=(time, gngle1))
    vel_curve.plot(pos=(time, speed1))
    acc_curve.plot(pos=(time, acc1))
    '''



    time = time + dt
    angle1old=angle1



'''
Here is documentation on the older VPython 6 (the "visual" module), which as of
January 2016 is still available but no longer supported. The main differences
are that vectors must now be represented as vector(x,y,z) or vec(x,y,z), not
as (x,y,z), the name "display" has been changed to "canvas", the name "gdisplay"
has been changed to "graph", and the curve and points objects have a new set of
methods. This Python program does an imperfect but useful job of converting old
programs to the new syntax.
'''
```

Appendix I

Arduino Uno

Arduino Uno R3 Front

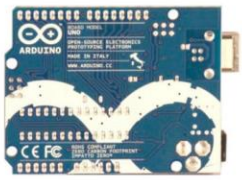


Arduino Uno R3 Back

Arduino Uno R2 Front

Arduino Uno SMD

Arduino Uno Front



Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.
The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.
Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.
Revision 3 of the board has the following new features:

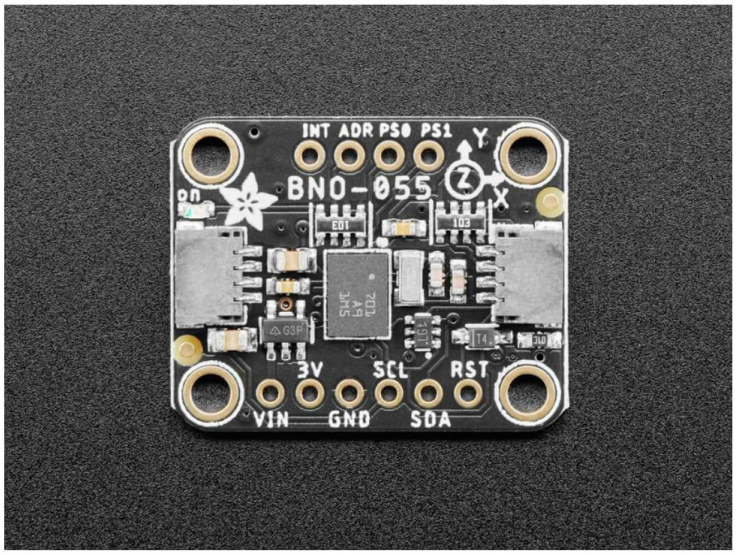
- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V

Appendix J



The new version of the board includes SparkFun qwiic (https://adafru.it/Fpw) compatible **STEMMA QT (https://adafru.it/Ft4)** connectors for the I2C bus so **you don't even need to solder!** Use a a plug-and-play STEMMA QT cable to get 9 DoF data ASAP.

Rather than spending weeks or months fiddling with algorithms of varying accuracy and complexity, you can have meaningful sensor data in minutes thanks to the BNO055 - a smart 9-DOF sensor that does the sensor fusion all on its own!

Data Output

The BNO055 can output the following sensor data:

- **Absolute Orientation** (Euler Vector, 100Hz)
  Three axis orientation data based on a 360° sphere
- **Absolute Orientation** (Quaterion, 100Hz)
  Four point quaternion output for more accurate data manipulation
- **Angular Velocity Vector** (100Hz)
  Three axis of 'rotation speed' in rad/s
- **Acceleration Vector** (100Hz)
  Three axis of acceleration (gravity + linear motion) in m/s^2
- **Magnetic Field Strength Vector** (20Hz)
  Three axis of magnetic field sensing in micro Tesla (uT)
- **Linear Acceleration Vector** (100Hz)
  Three axis of linear acceleration data (acceleration minus gravity) in m/s^2
- **Gravity Vector** (100Hz)
  Three axis of gravitational acceleration (minus any movement) in m/s^2
- **Temperature** (1Hz)
  Ambient temperature in degrees celsius

Related Resources

Appendix K

Proportional pressure regulators VPPE with display

FESTO

Technical data

- Flow rate
  310 ... 1250 l/min
- Voltage
  21.6 ... 26.4 V DC
- Pressure regulation range
  0.02 ... 2 bar
  0.06 ... 6 bar
  0.1 ... 10 bar

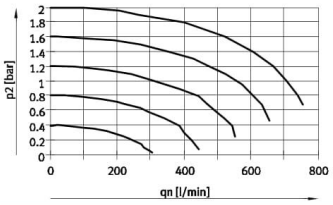
Variants
- Setpoint input as analogue voltage signal 0 ... 10 V
- Setpoint input as analogue current signal 4 ... 20 mA
- Assembly as in-line valve or on mounting rail
- 3-digit LED display

General technical data			
Pneumatic connection 1, 2, 3			G1/8
Design, valve function			Piloted diaphragm regulator, 3-way proportional pressure regulator
Sealing principle			Soft
Actuation type			Electric
Reset method			Mechanical spring
Type of control			Piloted via 2/2-way valves
Type of mounting			Via through-hole
Mounting position			Any, preferably vertical
Nominal size	Pressurisation	[mm]	5
	Exhaust	[mm]	2.5
Standard nominal flow rate		[l/min]	➔ Graphs
Product weight		[g]	390

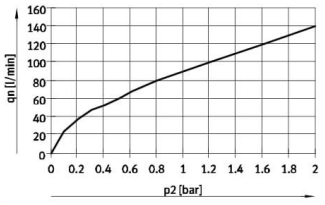
Flow rate qn as a function of output pressure p2

VPPE- ... -2- ...

qn 1 ⟶ 2

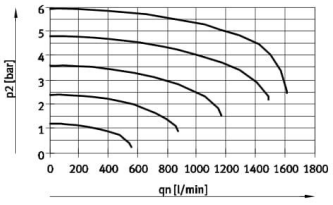


qn 2 ⟶ 3



VPPE- ... -6- ...

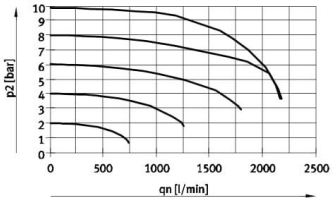
qn 1 ⟶ 2



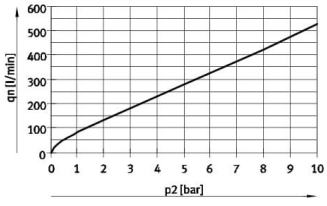
qn 2 ⟶ 3

VPPE- ... -10- ...

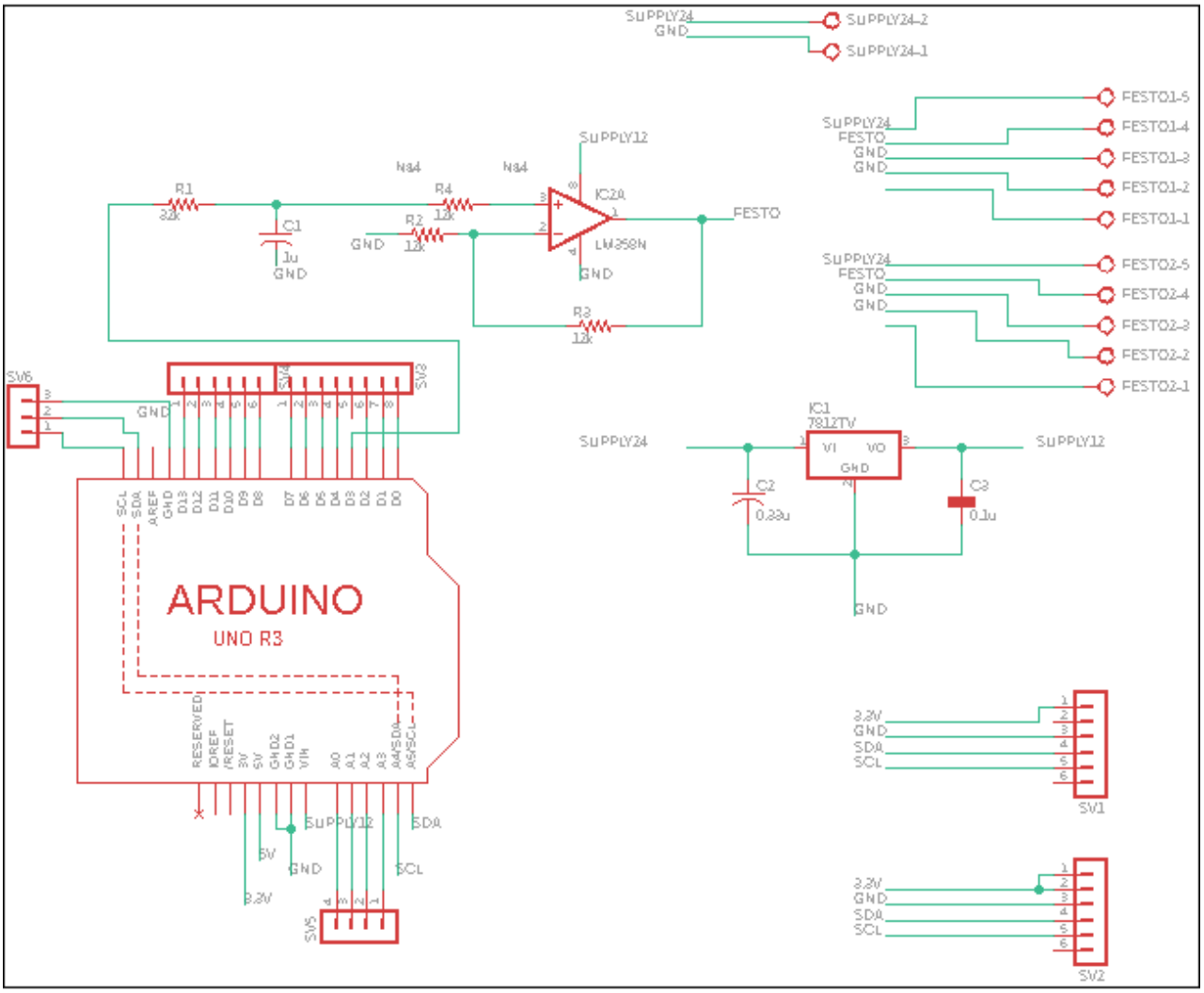
qn 1 ⟶ 2

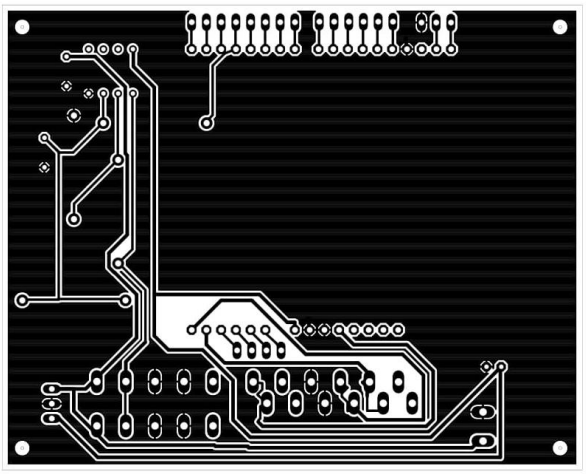


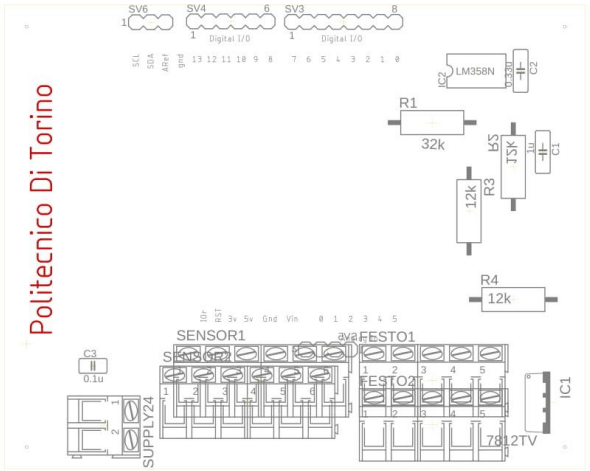
qn 2 ⟶ 3



Appendix L


SUPPLY24
GND
SUPPLY24_2
SUPPLY24_1
FESTO1-5
FESTO1-4
FESTO1-3
FESTO1-2
FESTO1-1
SUPPLY24
FESTO
GND
GND
SUPPLY12
R1
32k
C1
1u
R4
12k
R2
12k
IC2A
LM358N
R3
12k
FESTO
FESTO2-5
FESTO2-4
FESTO2-3
FESTO2-2
FESTO2-1
IC1
7812TV
VI
VO
C2
0.33u
C3
0.1u
SV6
SV4
SV3
SCL
SDA
AREF
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
ARDUINO
UNO R3
RESERVED
IOREF
RESET
3V
5V
GND2
GND1
VIN
A0
A1
A2
A3
A4/SDA
A5/SCL
SUPPLY12
SDA
5V
GND
SCL
3.3V
SV5
3.3V
GND
SDA
SCL
SV1
SV2




Politecnico Di Torino
SV6
SV4
SV3
LM358N
IC2
R1
32k
12k
R3
15K
R4
12k
C1
C2
SENSOR1
FESTO1
FESTO2
C3
0.1u
SUPPLY24
7812TV
IC1

References

1. H. Kazerooni, “Exoskeletons for human performance augmentation,”in Springer Handbook of Robotics. Springer, 2008, pp. 773–793.
2. M. K. Vukobratovic, “When were active exoskeletons actually born?”International Journal of Humanoid Robotics, vol. 4, pp. 459–486,2007.
3. Mark E. Rosheim, Robot Evolution – The Development of Anthrobotics, John Wiley & Sons, Inc, 1994. USA.
4. Vukobratovic M., Hristic D., Stojiljkovic Z., "Development of Active Anthropomorphic Exoskeletons", Medical and Biological Engineering, Vol. 12, No 1, 1974.
5. Vukobratovic M., Legged Locomotion Robots and Anthropomorphic Mechanisms (in English), research monograph, Mihailo Pupin Institute, Belgrade, 1975, also published in Japanese, Nikkan Shumun Ltd. Tokyo, 1975, in Russian "MIR", Moscow, 1976, in Chinese, Beijing, 1983.
6. Hristic D., Vukobratovic M., “Active Exoskeletons Future Rehabilitation Aids for Severely Handicapped Persons”, Orthopedie Technique, 12/1976, pp 221-224, Stuttgart, Germany.
7. Vukobratovic M., Borovac B., Surla D., Stokic D., Scientific Fundamentals of Robotics, Vol. 7, Biped Locomotion: Dynamics, Stability, Control and Application, Springer-Verlag 1989.
8. Vukobratovic M., Borovac B., Stokic D., Surdilovic D., Active Exoskeleton, Ch. 27: Humanoid Robots, pp 727-777, Mechanical Systems Design Handbook: Modeling, Measure and Control, CRC Press, 2001.
9. Dreyfus, H. L. What computers can’t do? the limitation of artificial intelligence, 1979 (Harper & Row, HarperCollins, New York).
10. Lu, Y. X. and Chen, Y. Foundation of the Humachine System (in Chinese). Chin. J. Mech. Eng., 1994, 30(6), 1–9.
11. Yang, C. J. Study on the theory of humachine intelligent system and its application. PhD Thesis, Zhejiang University, Hangzhou, 1997.
12. Rosen, J., Hannaford, B., and Burns, S. Neural control of an upper limb powered exoskeleton type system-grant report. In the First NSF Robotics and Computer Vision (RCV) Workshop, Las Vegas, Nevada, 26–27 October 2003, pp. 21–23.
13. 16 Turner, M. L., Findley, R. P., Griffin, W. B., Cutkosky, M.R., and Gomez, D. H. Development and testing of a telemanipulation system with arm and hand motion. In Proceedings of the ASME Dynamic Systems and Control Division, Orlando, FL, 2000, pp. 1057–1063.
14. 17 Koyamal, T., Yamano, I., Takemura, K., and Maeno, T. Multi-fingered exoskeleton haptic device using passive force feedback for dexterous

teleoperation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausannne, Switzerland, 2002, pp. 2905–2910.

15. Lockheed Martin, Fortis Exoskeleton User's Manual, Version 1.0, 2016.
16. Available from http://www.me.berkeley.edu/hel/hydextender.htm.
17. Available from http://sanlab.kz.tsukuba.ac.jp/HAL/indexE.html.
18. Zoss, A. B., Kazerooni, H., and Chu, A. Biomechanical design of the Berkeley lower extremity exoskeleton (BLEEX). IEEE/ASME Trans. Mechatronics, 2006, 11(2),128–138.
19. ISO 13482:2014 Robots and robotic devices -- Safety requirements for personal care robots, http://www.iso.org, 2014.
20. NIST Robotic Systems for Smart Manufacturing Program, https://www.nist.gov/programs-projects/robotic-systems-smart-manufacturing-program, accessed August 21, 2017.
21. NIST Emergency Response Robots Project, https://www.nist.gov/programs-projects/emergency-response-robots, accessed August 21, 2017.
22. Steinfeld A, Fong T, Kaber D, Lewis M, Scholtz J, Schultz A, Goodrich M, "Common metrics for human-robot interaction", Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-Robot Interaction, pp. 33-40, ACM, Mar 2, 2006.
23. M. Aragane, T. Noritsugu, M. Takaiwa and D. Sasaki, "Power assist wear for upper limb driven by sheet-like pneumatic rubber muscle", Proceeding of the 7th JFPS International Symposium on Fluid Power, Toyama, September 2008.
24. Kimdaejung Convention Center, Gwangju, Korea, "A mechanism design of waist power assist suit for a caregiver by using torsion springs" International Conference on Control, Automation and Systems (ICCAS 2013), Oct. 2013.
25. Lo, H.S. and S.Q. Xie, 2011. Exoskeleton robots for upper-limb rehabilitation.
26. https://forums.ni.com/t5/LabVIEW/myRIO-Vs-Arduino/td-p/3733362?profile.language=it.