CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF BIOMEDICAL ENGINEERING

Department of Biomedical Technology

Design of a control and actuator system of smart lower extremity brace

Návrh řídicího a pohonného systému chytré ortézy dolních končetin

Bachelor Thesis

Study program: Biomedical and Clinical Technology
Study branch: Biomedical Technician

Bachelor thesis supervisor: doc. Ing. Patrik Kutílek, MSc., Ph.D.

David Sebastian Martinez Lema

Kladno 2020

BACHELOR'S THESIS ASSIGNMENT

I. PERSONAL AND STUDY DETAILS

Student's name:	**Martinez Lema David Sebastian**	Personal ID number:	**473070**
Faculty:	**Faculty of Biomedical Engineering**		
Department:	**Department of Biomedical Technology**		
Study program:	**Biomedical and Clinical Technology**		
Branch of study:	**Biomedical Technician**		

II. BACHELOR'S THESIS DETAILS

Bachelor's thesis title in English:

Design of control and actuator system of smart lower extremity brace

Bachelor's thesis title in Czech:

Návrh řídicího a pohonného systému chytré ortézy dolních končetin

Guidelines:

Design and develop a prototype of control and actuator system of lower extremity brace. Propose and calculate the structure and components of smart brace control subsystem and actuators for rehabilitation based on clinical requirements and research on the current state of smart braces. Design control algorithm and wireless communication of brace with PC. The actuator and control subsystem should allow upright standing and slow walking. The control system and actuators will generate a support torque in case of weak muscles. Create an SW to control selected actuators and test actuator control. Evaluate the functionality of the created control system and actuators by testing real prototype. Compare prototype testing results with calculated assumptions.

Bibliography / sources:

[1] Ron Seymour, Prosthetics and Orthotics: Lower Limb and Spine, ed. 1st, LWW, 2002, ISBN 10: 0781728541
[2] Michelle M. Lusardi, Orthotics and Prosthetics in Rehabilitation, ed. 2nd, Butterworth-Heinemann, 2006, ISBN 10: 0750674792
[3] AK Agarwal , Essentials of Prosthetics and Orthotics , ed. 1st, Jaypee Brothers Medical Publishers (p) Ltd., 2013, ISBN 10: 9350904373

Name of bachelor's thesis supervisor:

doc. Ing. Patrik Kutílek, MSc., Ph.D.

Name of bachelor's thesis consultant:

doc. MUDr. Vojtěch Havlas, Ph.D.

Date of bachelor's thesis assignment: **17.02.2020**
Assignment valid until: **19.09.2021**

[illegible]
prof. Ing. Peter Kneppo, DrSc.
Head of department's signature

[illegible]
prof. MUDr. Ivan Dylevský, DrSc.
Dean's signature

III. ASSIGNMENT RECEIPT

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

20/02/2020	[illegible]
Date of assignment receipt	Student's signature

DECLARATION

I hereby declare that I have completed this thesis having the topic “Design of a control and actuator system of smart lower extremity brace” independently, and that I have attached an exhaustive list of citations of the employed sources.

I do not have a compelling reason against the use of the thesis within the meaning of Section 60 of the Act No.121 / 2000 Sb., on copyright, rights related to copyright and amending some laws (the Copyright Act).

In Kladno 2020 …..…………….…………………..

David Sebastian Martinez Lema

ACKNOWLEDGEMENTS

I would like to thank my supervisor doc. Ing. Patrik Kutílek, MSc., Ph.D. for his guidance and assistance provided during the realization of this project.

ABSTRACT

Title of the Thesis:

Design of a control and actuator system of smart lower extremity brace.

Exoskeletons can facilitate the rehabilitation of lower limbs as they provide additional structural support and strength. The design and implementation of a functional prototype of lower extremity brace actuation and control system, capable of wireless communication is this thesis's main aim. The design focus is to provide a supportive torque and increase the range of motion after complications that reduce muscular strength such as Arthrofibrosis. Its structure, components and control algorithms were proposed, calculated and tested. The prototype supports leg raises and gradual standing and slow walking. The main control modalities are based on an Artificial Neural Network and a Finite State Machine. The prototype's functionality was monitored by time-angle graphs. The final prototype shows the potential to treat joint impairments in an adaptive way.

Key words

Exoskeleton, range of motion, actuator system, control system, neural networks.

ABSTRAKT

Název práce:

Návrh řídicího a pohonného systému chytré ortézy dolních končetin.

Exoskeletony dokáží usnadnit rehabilitaci dolních končetin tím, že poskytují dodatečnou strukturální oporu a sílu. Hlavním záměrem této bakalářské práce je navržení a implementace funkčního prototypu ovládacího a pohonného systému dolní koncové ortézy, schopného bezdrátové komunikace. Navržené zaměření poskytuje podpůrný točivý moment, kterým se zvyšuje rozsah pohybu končetin při zdravotních komplikacích, snižující svalovou sílu, jako je například artrofibróza. Byla navržena, vypočtena a testována její struktura, komponenty a řídicí algoritmy. Prototyp podporuje zdvih nohou, postupné postavení se a pomalou chůzi. Hlavní způsoby ovládání jsou založeny na umělé neuronové síti a konečném stavovém automatu. Funkčnost prototypu byla monitorována pomocí grafů s časovým úhlem. Konečný prototyp představuje potenciál adaptivně léčit poruchy kloubů.

Klíčová slova:

Exoskeleton, rozsah pohybu, akční systém, řídicí systém, neuronové sítě.

Table of Contents

List of symbols and abbreviations

List of symbols

Symbol	Meaning
A	Amperes
a	Distance of contact
D	Distributed Load
d	Distance to center of mass
F	Force
fs	Pulse Frequency
g	Gravitational acceleration
GR	Gear Ratio
m	Mass
Pa	Probability of class with preference
PPR	Pulses Per Revolution
Px	Probability of estimated class
Rpm	Revolutions Per Minute
S	Stepper Motor Steps
t	Time
T	Torque
Th	Angle Threshold
Tl	Torque of load
V	Voltage
w	Scaling factor
α	Activation Angle
θ	Encoder Angle
φ	Desired Angular Position

List of abbreviations

Abbreviation	Meaning
ANN	Artificial Neural Network
CPM	Continuous Passive Motion
DOF	Degrees of Freedom
LSB	Least Significant Bit
MSB	Most Significant Bit
ROM	Range of Motion
TKA	Total Knee Arthroplasty
W/R	Read or Write

1 Introduction

The term exoskeleton is originally from the field of Biology. It is defined as the external skeleton of insects and other invertebrates which protects and supports their body. In other words, a biological exoskeleton would perform the same task as a mechanical or robotic exoskeleton. A series of biometric sensors track the position of the limbs, nerve impulses and other available information from the environment to produce a movement. The processing unit of exoskeletons then detects these signals, processes them and acts accordingly. It allows the exoskeleton to work in cooperation with its user as if it was part of the body. This process is similar to how the brain reads the nervous signals from the body and then adapts itself to the environment.

The demand for exoskeletons is constantly increasing as efforts to modernize and automate healthcare are made globally. Such devices can facilitate limb rehabilitation process since they provide additional strength and support to the patients. At the same time, they can reduce the therapist’s fatigue caused by the exhaustive nature of rehabilitation therapy. Exoskeletons are used in different types of therapies where joint impairments are treated, such as gait rehabilitation. These type of robotic devices are called lower limb exoskeletons. They are designed to work parallelly to the human lower limbs and mimic the human gait [1].

2 Overview of the current state of the art

As opposed to the normal human skeleton, which supports the body from the inside, an exoskeleton supports the body from the outside. Exoskeletons are usually designed to allow people with mobility disorders to walk or increase strength and endurance. Exoskeletons have several key components; the frame, usually made of lightweight materials, must be strong enough to support the weight of the body, as well as the weight of the exoskeleton and its components. Sensors capture information about how the user wants to move. The sensors can be manual, like a lever, or they can be electric and detect the physiological impulses generated by the body. The controller acts as the brain of the device, the controller is an on-board computer which takes the information captured by the sensors and controls the actuators. The computer coordinates the actuators in the exoskeleton and allows the exoskeleton and its user to stand, walk, climb or descend [2]. To do it in the bes possible way the actuators are one of the most important elements that should be chosen.

A stepper motor is a brushless DC motor that has the characteristic of rotating in both directions, moving with precise angular increments, holding a holding torque at zero speed, and controlled by digital circuits. The stepper motor is very useful because it can be precisely positioned without any feedback sensor, therefore it can be represented as an open circuit controller. The number and rate of pulses controls the position and speed of the motor shaft. Stepper motors are typically manufactured with 12, 24, 72, 144, 180, and 200 steps per revolution, resulting in shaft increments of 30 °, 15 °, 2.5 °, 2 °, and 1.8 °. As the stepper motor coils are activated in a particular order, a current is allowed to flow through them that magnetizes the stator causing electromagnetic poles that will cause motor drive. Special microstep circuits can be designed to allow more steps per revolution, often 10,000 steps per revolution or more.[3]

For instance a servo motor allows the user to control the position of the rotor. This makes it posible to move it to a certain amount of degrees and then remain fixed in that position. They are especially useful in the field of robotics. A servo motor is a type of motor that, by reversing its polarity, its direction of rotation changes, the axis moves by desired degrees. Its main advantage is the control of position and speed. It is made up of a mixed system of electromechanical and electronic parts. The motor and a set of gears are the electromechanical part, it rotates thanks to an applied current, and the electronic part is the control circuit. They are composed by a an electronic control circuit and a potentiometer internally.Servo motors have three cables while common motors only have two. Regarding its operation, it can be determined that it responds to the width of the modulated signal.[4]

Model-based Control systems for exoskeletons are commonly used nowadays. They part from a dynamic model which can be obtained by identifying the system involved in the movement and then replicating its behavior [5]. Another approach is to drive the exoskeleton by means of a Finite state machine. These algorithms are based in states and transitions. To reach a certain outcome the control system must undergo such transitions. They have been use in combination with a foot pressure sensor to detect if the system should provide a supportive movement[6]. In 2015 a team of researchers in Korea proposed gait phase classification method based on neural networks using sensor signals and foot force sensors to classify gait phases.[7] Exoskeletons used in gait rehabilitation are often designed to assist the limited movements caused by a condition. To achieve this, they have to be developed in such a way that they follow rehabilitation standards and implications such as the ones proposed by The International Classification of Functioning and Disability [8].

After a knee complication such as Arthrofibrosis or a TKA surgery the knee's ROM could be reduced from 130 degrees to as low as 80 degrees. Ideally an assistive device such as an exoskeleton or a continuous passive motion (CPM) machine should allow this range to increment progressively until it is back to the post-operative state [9]. There are reported cases of patients who are able to regain their original knee extension and flexion after using such devices and even return fully to do sports and physical activities [10]. A study conducted in 2006 showed that short-term usage of CPM causes a greater short-term ROM [11].

3 Aims

The main aim of this project is to design and realize a prototype of actuator system for a lower extremity brace developed by the company Prokyber s.r.o., Kladno, Czech Republic. For this, the control algorithms, structure and components of the smart brace control system and brace actuation have to be proposed, calculated and developed. The actuator and control system should allow upright standing and slow walking by generating a supportive torque in case of debilitated muscles as a result of conditions such as Arthrofibrosis and the post-operative stage of TKA. A testing module able to connect wirelessly to the testing software would be created to control the selected actuator and test actuator control. The aforementioned elements should comply with rehabilitation standards to address joint impairments. This also means that it should allow biomedical technicians or medical staff to test it in an easy manner, thus the software interface for actuator testing of the smart brace must be user friendly. The outcome of the bachelor thesis would be a functional prototype of lower extremity brace actuator system designed to increase the ROM following such complications. Finally the functionality of the created system and must be evaluated.

4 Methods

During the realization of this project an actuator and control system prototype for a smart knee brace, designed by the company Prokyber s.r.o., was developed. In order to fulfill the principal aims of the project, the desired movements where thoroughly analyzed so that the behavior of actuator system complies with clinical requirements for orthotic devices used in the human knee.
Based on this requirements, its structure, components and control algorithms were proposed, calculated and tested. This development sequence is described in the next chapters.

4.1 Proposal of structure and components

The movements analyzed in the following subchapters were used as a reference to select the components of the control system and design the control algorithms and the control modalities that drive the exoskeleton.

4.1.1 Leg rises evaluation

The American Academy of Orthopaedic Surgeons states that supported and unsupported knee bends while sitting are often recommended by orthopedic surgeons and physical therapists as part of rehabilitation exercises during the recovery stage of TKA. The supported knee bends involve the aid of the healthy leg and the unsupported knee bends are performed by the affected leg only, as shown in Figures 4.1 and 4.2 respectively. Both movements involve the extension of the affected knee as far as the patient withstands, then the current position is held for 5 to 10 seconds and finally the leg is moved to the original position [12].

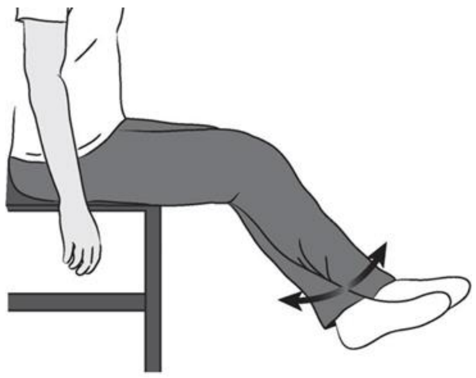


Figure 4.1: Supported knee bend while sitting (taken from [13])

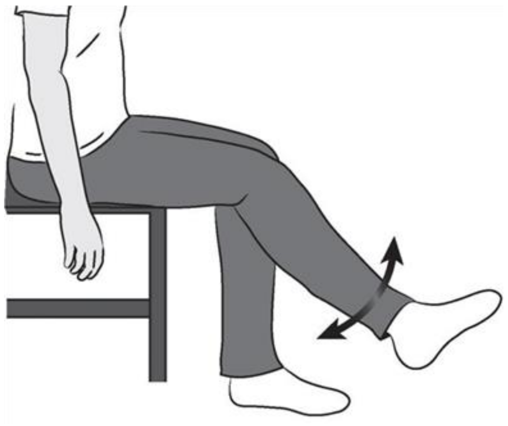


Figure 4.2: Unsupported knee bend while sitting (taken from [14])

Based on the movements described above the desired ROM was defined. Ideally, the initial position is when the leg is perpendicular to the ground and the foot is in contact with it, as described in Figure 4.2, though it can be different depending on the knee's degree of initial flexion. For this reason, the initial angular position was set to 0 degrees and the final angular position to 120 degrees to give the user an extended ROM. The control algorithms of the smart brace use this ROM (0° to 120°) to generate the main boundaries applied in the leg rise movement for actuator testing and also include the 5 seconds of pause in the extended position.

4.1.2 Standing up and walking evaluation

The process of standing up requires the action of the biomechanical pulley in the knee, as illustrated in Figure 4.3, which involves the femur, tibia, patella (including its ligament and tendon) and quadriceps [15]. The components of the actuator system were selected in such a way that they mimic the aforementioned structure and behavior.

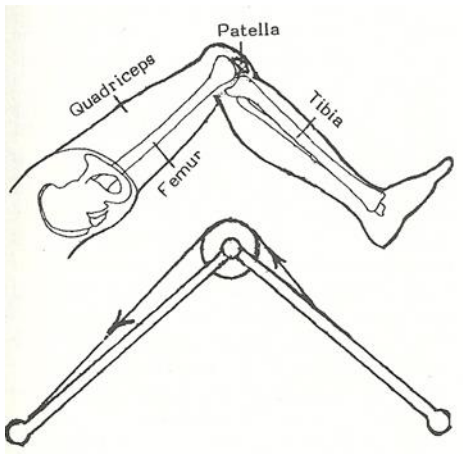


Figure 4.3: A biomechanical pulley of the human body compared to a mechanical pulley (taken from [16])

The tibial-on-femoral extension for standing up is driven by the flexion of the quadriceps in the same way as the tibial-on-femoral extension for leg rises (Figure 4.4).For this reason, the same ROM was used in the design of the control algorithms for standing up.

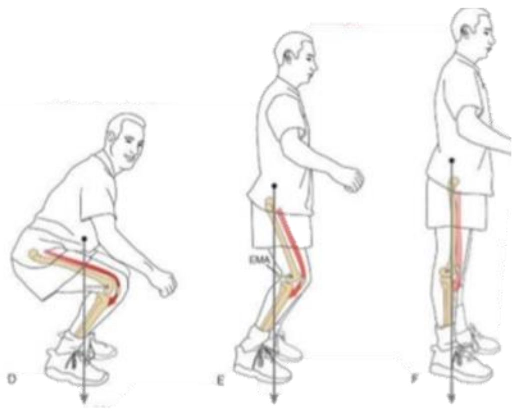


Figure 4.4: Tibial-on-femoral extension for standing up (taken from [17])

In the sagittal, transverse and coronal planes, the knee joint is displaced during gait. Nevertheless, most of the knee joint motion is done in the sagittal plane, including the knee joint's flexion and extension. The knee joint's flexion and extension is periodical and ranges from 0 to 70 degrees, although there is some variability in the exact amount of peak flexion that occurs. Such variations may be due to differences in walking speed, personal gait pattern of the subject and the terrain evenness. This process is described in the book Tidy's Physiotherapy [18].

To replicate the described gait pattern the control system uses an angle encoder and two interrupters; they are placed in such a way that only one is pressed at the time when the user’s leg is extending, through phase 5, or flexing, through phases 3 and 4, as shown in Figure 4.6.

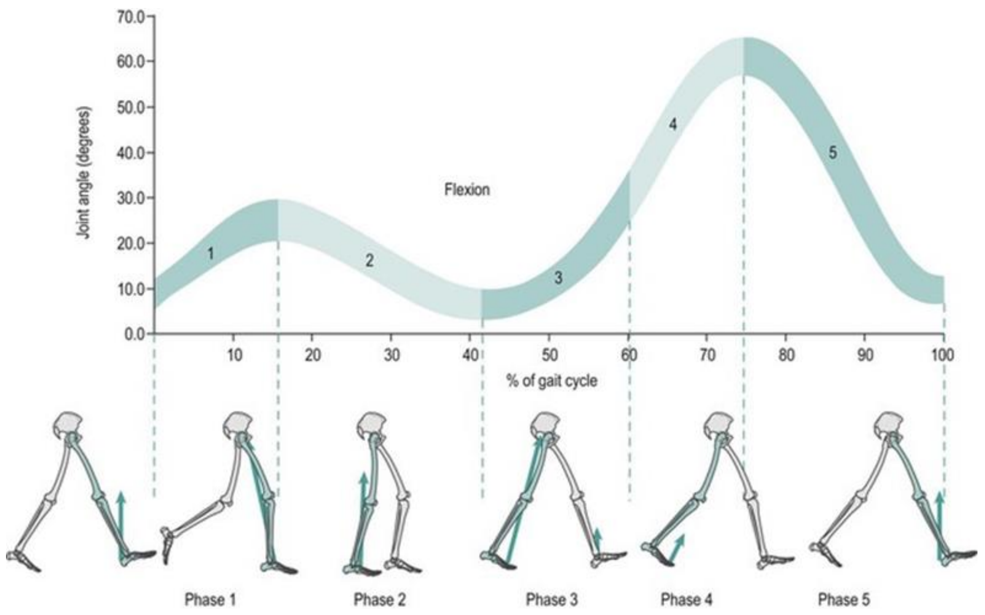


Figure 4.6: Knee angular movement through the gait cycle (taken from [19])

4.1.3 Calculation of forces based on movement analysis

The mechanical structure of the smart brace was designed by Prokyber s.r.o. to resemble the biomechanical pulley system of the human knee and is mainly made of aluminum. It has three links with two revolute joints. The three links of the exoskeleton mimic to the structure of the human leg, which is composed by the foot, shank and thig; the two revolute joints correspond to the ankle, which is merely used as a pivot and support point and the knee, which is actuated by mechanism composed by gears and pulleys as described on (Figure 4.7).

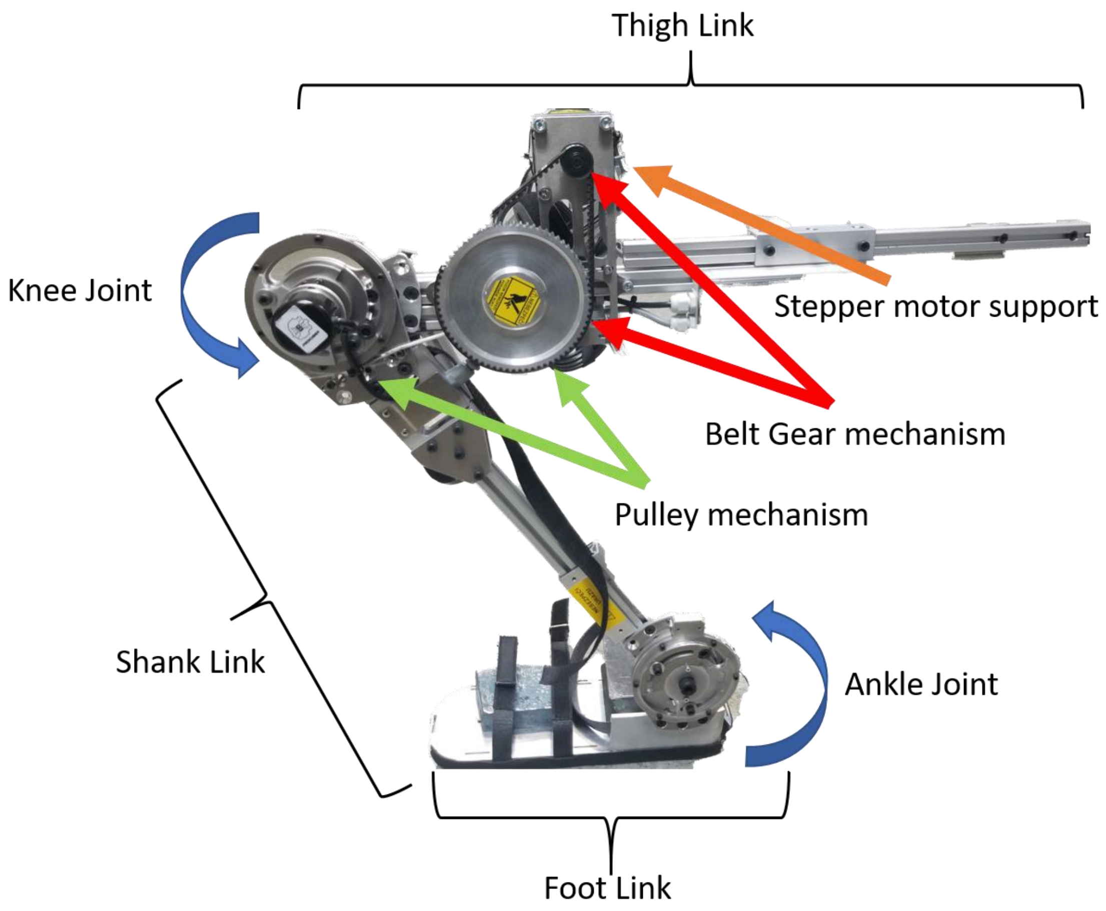


Figure 4.7: Two DOF smart brace structure.(Manufactured by [20])

In order to evaluate the structural support provided to the patient and select the actuator for the smart brace actuator system a structural and mechanical analysis was performed. It consisted in calculating the total reaction force and moment required on the upper segment of the exoskeleton, which is the one in charge of creating a structural support external to the knee joint, and the holding torque of the exoskeleton's actuator system which would ease the process of standing up and walking. The first step was identifying all the relevant components that exert a weight on the link. These main elements would be the gear fixed to the link by means of an axle and bearing, the motor support fixed by two screws, the chosen motor and the link itself. The masses and distances of the mentioned elements were measured and their forces were calculated. The weight of the motor was set to 2 kg as a maximum rating to keep the total load weight as low as possible. Since a human subject is intended to use the brace, the weight of the thig and upper body segment on an average human[21] of 62 kg were considered using anthropometric tables found in the book Human Body Dynamics [22].
To obtain the force of each element following equation was used:

$$\vec{F} = m \times g \tag{4.1}$$

Where $\vec{F}$ is the force,
m— mass,
g— gravitational acceleration constant.

To get the force vector of the distributed load this equation was used:

$$D = \frac{\vec{F}}{a} \tag{4.3}$$

Where D is the distributed load,
$\vec{F}$ — the applied force,
a— distance of contact.

To obtain the resultant vector of reaction force this equation was used:

$$\sum_{i=1}^{n} \vec{F_i} = 0 \tag{4.5}$$

Where n is the total number of forces in the system,
i— the index of each applied force,
$\vec{F}$— the applied force vector.
Additionally 0 represents that the system is in equilibrium .

To obtain the resultant reaction moment the following equation was used:

$$\sum_{i=1}^{n} \vec{F_i} \times d = 0 \tag{4.6}$$

Where n is the total number of forces on the system,
i— the index of each applied force,
$\vec{F}$— the applied force,
d— the distance from the joint to the center of mass of the element.
Additionally 0 represents that the system is in equilibrium .

To estimate the gear ratio of the actuation mechanism this equation was used:

$$GR = \frac{\Delta Ld}{\Delta Eff} \tag{4.7}$$

where *GR* is the gear ratio,
ΔLd — the driven angular displacement (load),
ΔEff— the driver angular displacement(effort).

To obtain the required torque for the actuator system the following equation was used:

$$T = \frac{T_L}{GR} \tag{4.8}$$

Where *T* is the final required torque,
T_L— torque of the load,
GR— Gear ratio of the system.

A free body diagram was created to illustrate the applied forces and the resultant force, that was calculated using equation 4.5 and the resultant moment, which was calculated with equation 4.6. A moment of 233.18 Nm and a resultant force of 467.85 N were obtained. This implies that the structure of the exoskeleton should be able to reduce the stress on the muscles and knee joint caused by lifting the upper body and thigh link of the exoskeleton, including all of its components. The gear ratio of the smart brace is 11 approximately. It was determined by setting an initial position marker in the knee joint (driven element), then the small gear (driver element) was rotated 360 degrees and finally the angular displacement Δθ in the knee joint was measured with a value of 33 degrees. For this estimation the formula 4.7 was used. Finally the required torque for the smart brace was approximated using the calculated moment applied on the joint by means of formula 4.8. This resultant torque of 21.2 Nm was used as a reference to select the actuator.

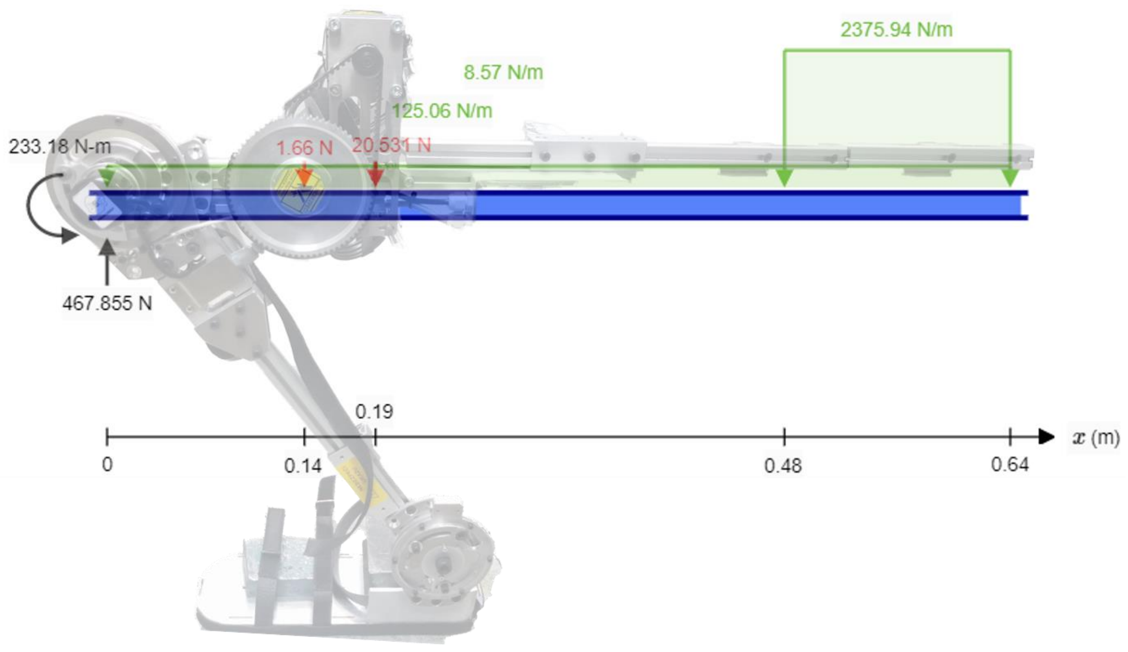


Figure 4.8: Free body diagram of forces applied on the thigh link.

4.1.4 Actuator System

During the overview of the current state of the art it was determined that the best motor for this application is a bipolar stepper motor due to the high accuracy, torque provided and the fact that its open loop. The stepper motor LDO-60STH86-3004A, manufactured by LDO Motors, was selected based on the structural analysis. The torque rating of this motor is 2.75 Nm at low *Rpm*, however if we multiply it by the gear ratio of the actuation mechanism the final torque of 30.25 Nm is obtained. This value is greater than the required torque of 21.1 Nm. Based on this rough theoretical approximation, the actuation system would be able to lift the applied load and generate a supportive torque. The stepper motor would be incorporated to the small gear of 14 teeth situated at the top of the stepper motor support. After this it would be connected to the main gear of 72 teeth using the already provided dented belt PowerGrip-HTD-50-5m. The main gear is fused with a small pulley of 25 mm of diameter facing the brace. This pulley is in turn linked to the knee joint pulley by means of a steel cable. The knee joint pulley has a diameter of 100 mm. Finally, the Rotary encoder BHK 16.05A.0500-I2-5, manufactured by BHK, was selected to detect the angular position of the leg. This encoder was chosen due to the high precision given by the number of pulses per revolution it has (500 pulses per revolution) and the fact that it operates with 5V, given that most microcontrollers operate on that range. The structure of the smart brace actuator system is described on the Figure 4.9.

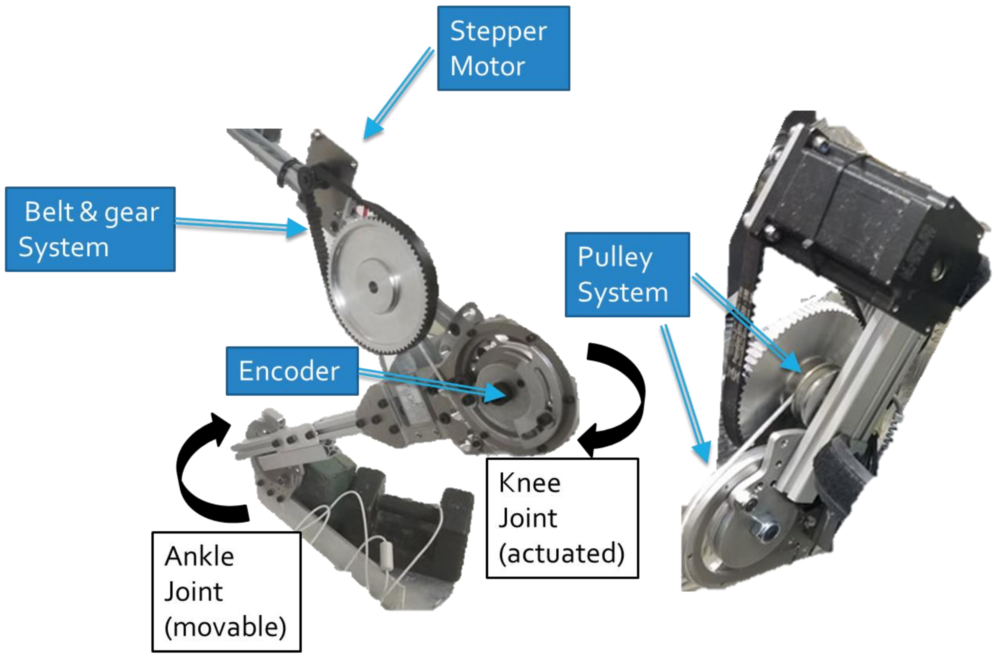


Figure 4.9: Two DOF smart brace actuator structure.

4.1.5 Control System

For the control system, the architecture in Figure 4.8 was proposed based on the movement, sensing and control requirements. The microcontroller board used in this prototype is the STM32 F446 which uses a custom firmware developed by the company Prokyber.s.r.o. specifically for the control of the smart brace. The embedded functions of the mentioned firmware operate with input and output values stored in its registers. The pins of the STM32 are assigned to these inputs and outputs as shown on table 4.. The system uses a computer running Ubuntu Linux 20.04LTS as the one in charge of the decision-making, it sends R/W requests and receives responses trough Modbus RTU from the STM32, acting in a master-slave relationship with it.

Table 4.1. List of pins of the STM32

Pin	Description
PB_8	Encoder A
PB_9	Encoder B
PB_10	Motor pulse counter(connect to pulse pin)
PC_7	Motor pulse
PC_6	Motor enable
PB_12	Motor direction
PA_10	Button 1
PB_1	Button 2

The control system uses the encoder's pulses, from its terminals A and B, as an input. Internally the signals from the encoder are transformed into a degree value and assigned to a register that can be read when needed. Some pins can be used as digital inputs, in this case they are used to get the binary state from the buttons so that their instance can be used for control. As an output the STM32 sends three different pulses to the stepper driver denominated Direction, Enable and Pulse used to control the stepper motor. These rectangular pulse signals are in the range of 0V to 5V. The stepper driver DM805-AI manufactured by Leadshine was selected based on the fact that it matches the current raiting of 3A required by the stepper motor and the diversity of control modalities it allows, which were useful in the stage of actuator prototyping, being the Pulse/Direction the one mainly used in this project.

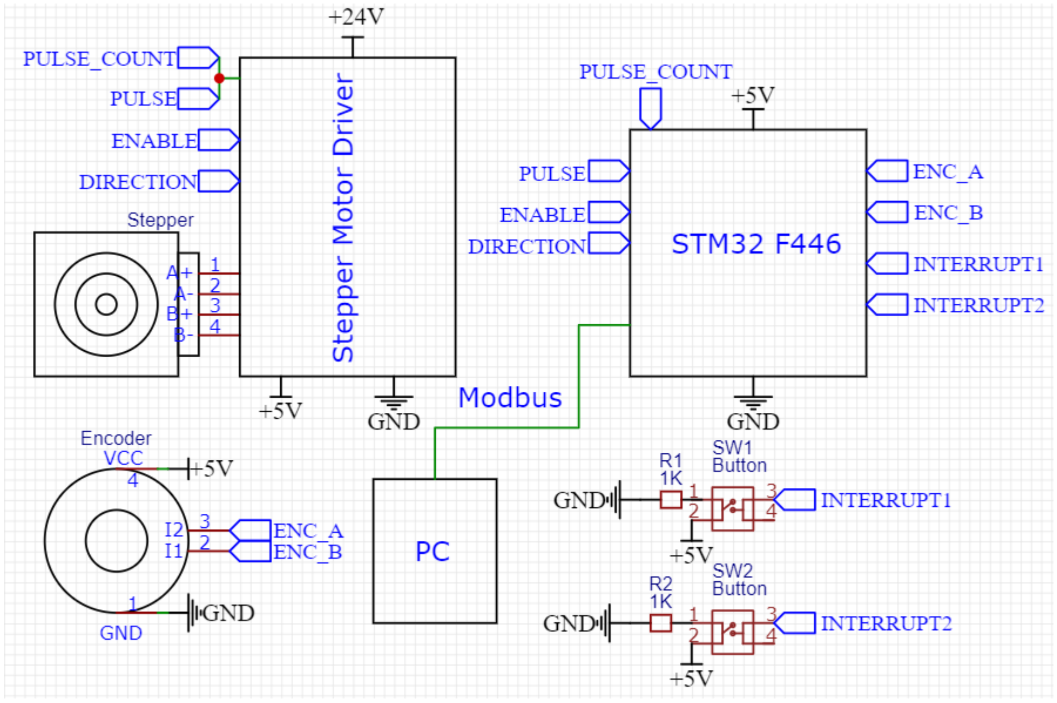


Figure 4.8: Architecture of the control system.

4.1.6 Actuator testing wireless module

In order to test the actuator control a testing module capable of wireless Bluetooth communication was developed. This testing module is designed for biomedical technicians and other medical staff to test individual actuators by plugging it to a selected actuator with a pin connector and control it by means of a software created for tablets or smartphones running Android Operating System; the software is fully described in the next subchapter.

The actuator testing module is controlled by an Arduino Nano microcontroller board which uses an Atmega328p microchip. This microcontroller board was selected given that it is compact, it allows fast prototyping and it is able to send and read data through serial communication using the transmit and receive (TX and RX) pins. The component chosen for wireless communication is the ZS-040 Bluetooth module due to its compatibility with Arduino microcontrollers and serial communication capabilities. To connect it with the serial pins of the Arduino a voltage divider was used because the Arduino has an input logic level of 3.3V. Oppositely the stepper driver requires the pulse and direction signals to be of 5V of amplitude, therefore a series of 2N2222A transistors were used as interrupters mediated by the 3.3 V signals produced by the pins D5 and D6 of the Arduino Nano. The testing module connection is represented on Figure 4.9.

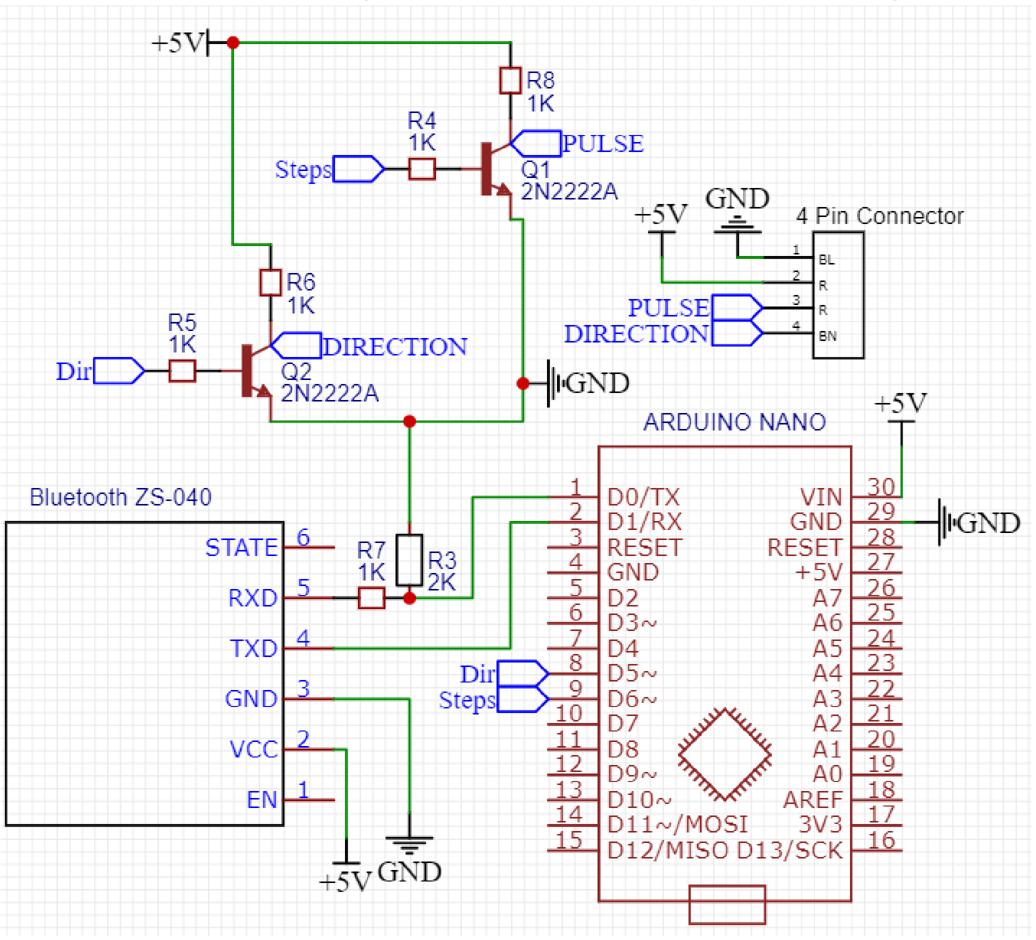


Figure 4.9. Bluetooth actuator testing module

4.2 Design of control algorithms

4.2.1 Main functions

The main embedded functions on the STM32 firmware are the function that transforms the encoder pulses into degrees and the function that generates the pulses to move the stepper motor to a specific angular position at a given *Rpm* as presented in Figure 4.10.

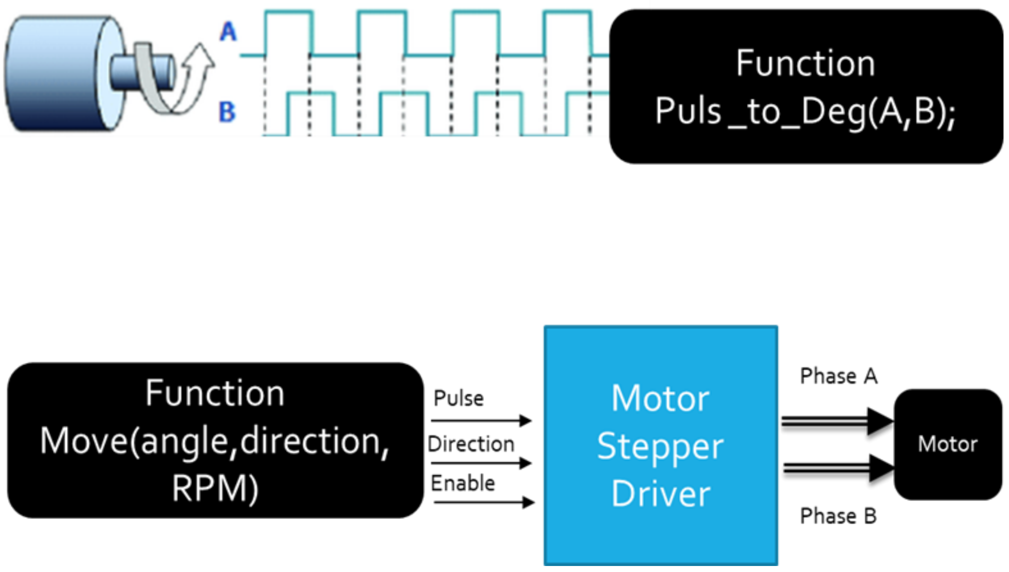


Figure 4.10: Main functions used in the actuator control (Taken and edited from [17])

To obtain the angle from the encoder the following equation was used:

$$\theta = \frac{P}{PPR} \times 360 \tag{4.9}$$

Where θ is the encoder angle,
P— the number of pulses,
PPR— the pulses per revolution of the encoder.

To measure the number of steps required to move the motor to a specific angular position this equation was used:

$$S = \frac{\varphi}{360} \times 200 \tag{4.10}$$

Where S is the stepper motor steps,
φ — the desired angular position,
Additionally 200 represents the steps per revolution of the used stepper motor.
For the wireless actuator testing module an algorithm based on equation 4.1 was used to move the actuator to a desired angular position.

To measure the revolutions per minute the following equation was used:

$$Rpm = \frac{1.8}{360} \times fs \times 60 \tag{4.11}$$

Where Rpm is the revolutions per minute of the stepper motor,
fs — the pulse frequency,
Additionally 1.8 is the angle per step of the stepper motor .

For accessing and using the main functions the control algorithms read and write values from the STM32 registers through Modbus RTU communication by using their respective address shown in Table 4.2. These functions use hexadecimal notation, therefore a function for transforming them into floating point values and vice versa was implemented, these functions are called “floatToMod” and “modToFloat”; they are used throughout the Python control algorithms.

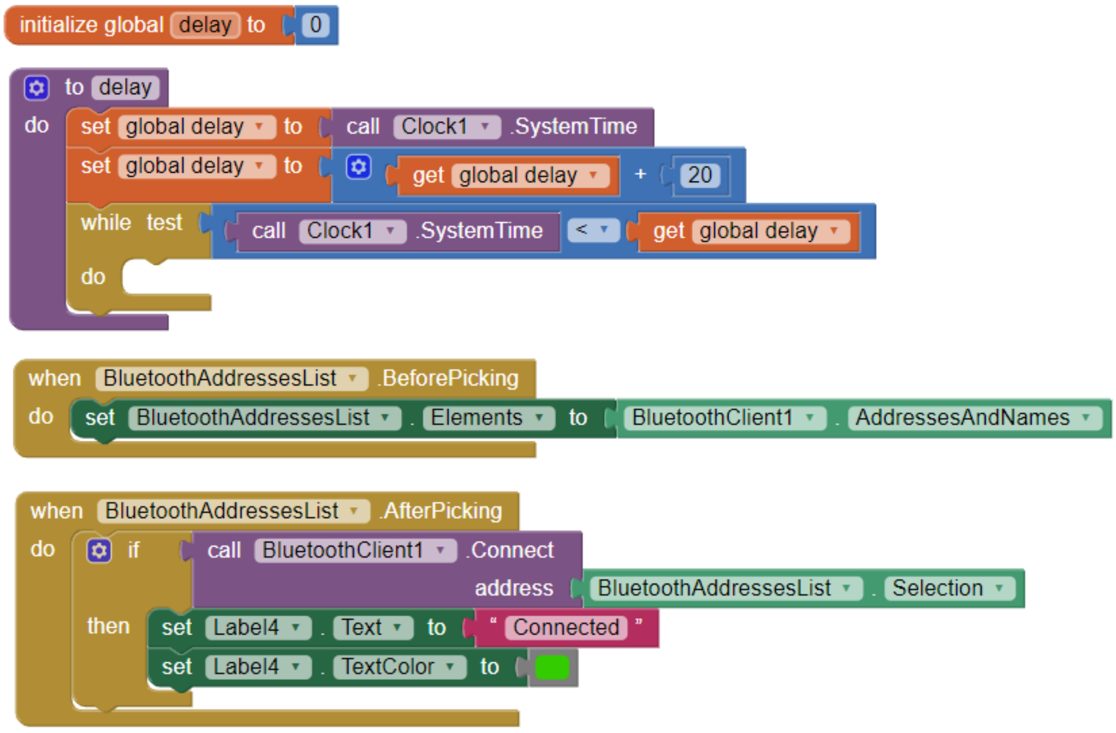
Table 4.2. List of registers in the STM32

Address	Register description	Type
40356	Enable register	W
40357	RPM(Revolutions per minute)	W
40358	Stepper angular position (MSB)	R/W
40359	Stepper angular position (LSB)	R/W
40360	Mode (1 to run continuously, 2 for angular movement)	W
40361	Run (0 to stop, 1 or anything else to run)	W
40350	Encoder angle	R
10001	Digital inputs	R

4.2.2 Actuator testing software and wireless communication

The actuator testing software was developed using the MIT App Inventor development environment for applications. It uses visual objects to create block diagrams as a programming paradigm. This software was programmed to allow the user to test a selected actuator by moving it to a desired angular position or by generating a leg rise cycle. These commands are sent from an Android device to the smart brace actuator through Bluetooth with the wireless testing module as a mediator.

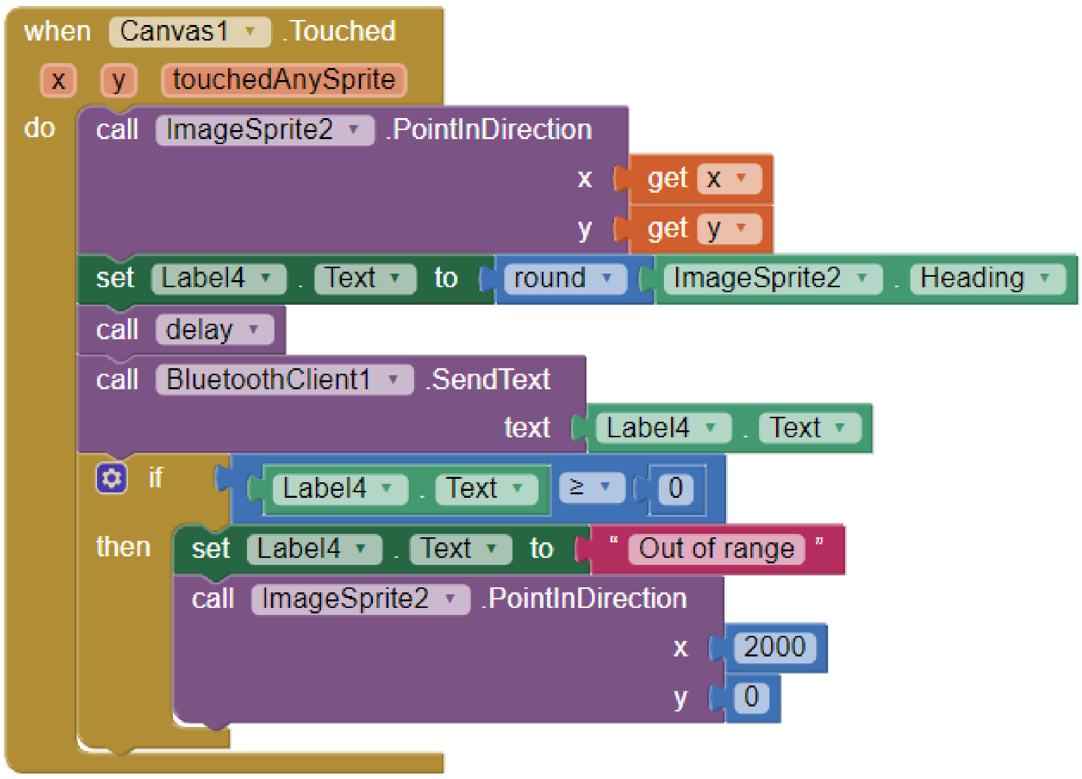
First a global delay function is generated. This function is necessary because without it the program would run continuously and the executed commands would not be able to be delivered to the actuator wireless testing module. This procedure uses the system's time which is a counter that counts every millisecond. To generate a twenty milliseconds delay a value of 20 is added to the current system's time and that new value is compared with the actual system's time until it catches up in 20 milliseconds. The delay happens because this comparison is tested on a while loop containing no instructions to do. The next function created generates a list of available Bluetooth devices, denominated clients, and their respective addresses to select from. During this procedure the user can select the wireless actuator testing module. After the user picks a Bluetooth client, the application gets connected to it and sets the label to "Connected". These functions are described on Figure 4.11.



```
initialize global delay to 0

to delay
do  set global delay to  call Clock1 .SystemTime
    set global delay to  get global delay + 20
    while test  call Clock1 .SystemTime < get global delay
    do

when BluetoothAddressesList .BeforePicking
do  set BluetoothAddressesList . Elements to  BluetoothClient1 . AddressesAndNames

when BluetoothAddressesList .AfterPicking
do  if    call BluetoothClient1 .Connect
                              address  BluetoothAddressesList . Selection
    then  set Label4 . Text to " Connected "
          set Label4 . TextColor to 🟩
```

Figure 4.11: Global delay function and Bluetooth connection function.

What followed was the creation of a canvas space that was used to move the smart brace actuator to a certain angular position. This was achieved by making an indicator image sprite point in direction to the x and y coordinates of the section where the canvas space was touched and then using that action to send a value in degrees to the actual Bluetooth client. Since the actuator is not set to move in a range of 360 degrees, a function to limitate the range, return the leg to the initial position and warn the users that they are out of range was created. This function is shown in Figure 4.12 .

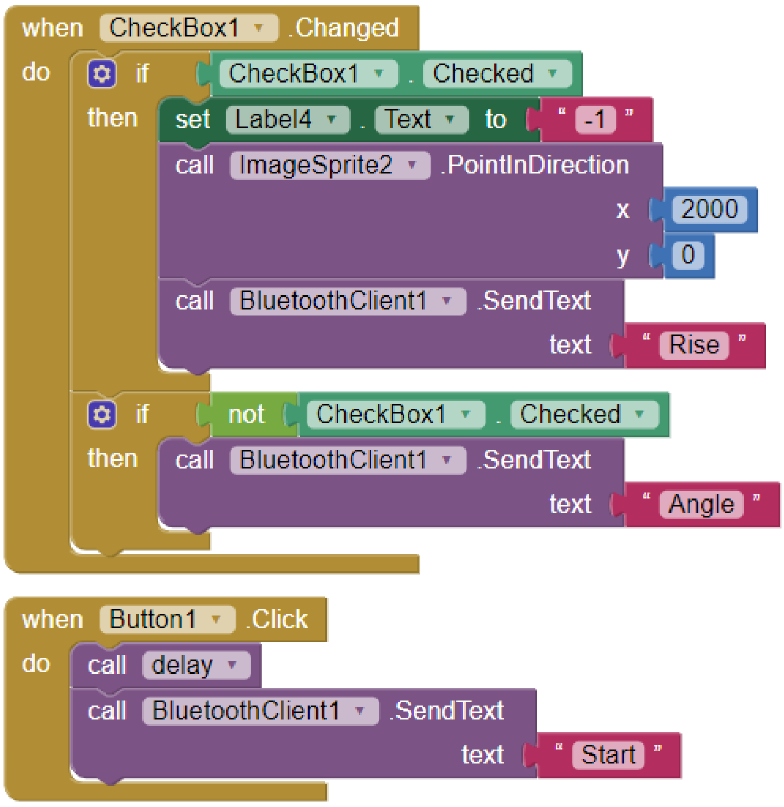


```
when Canvas1 .Touched
  x  y  touchedAnySprite
do  call ImageSprite2 .PointInDirection
                                   x  get x
                                   y  get y
    set Label4 . Text to  round  ImageSprite2 . Heading
    call delay
    call BluetoothClient1 .SendText
                             text  Label4 . Text
    if    Label4 . Text  ≥  0
    then  set Label4 . Text to  " Out of range "
          call ImageSprite2 .PointInDirection
                                         x  2000
                                         y  0
```

.

Figure 4.12: Function to move the actuator to a given angular position

Finally a function to switch from the angular position selection to the leg rise cycle generation was designed by sending messages to the current Bluetooth client. The check box object switches between the mentioned functions and a button object starts the leg rise cycle by sending the command “Start”. If the check box is checked the indicator returns to the origin and the command “Rise” is sent and when it is not checked the command “Angle” is sent. This function is illustrated in Figure 4.13.



```
when CheckBox1 .Changed
do  if    CheckBox1 . Checked
    then  set Label4 . Text to " -1 "
          call ImageSprite2 .PointInDirection
                                    x  2000
                                    y  0
          call BluetoothClient1 .SendText
                               text " Rise "
    if    not CheckBox1 . Checked
    then  call BluetoothClient1 .SendText
                               text " Angle "

when Button1 .Click
do  call delay
    call BluetoothClient1 .SendText
                         text " Start "
```

Figure 4.13: Function for switching between control modalities.

In order to read the commands sent by the mobile application, the Arduino Nano was programmed to receive such commands through serial communication as a string. There are two main states for the wireless actuator testing denominated Rises and Angle. These states are set up by the check box object on the Android application. When one of them is selected, the program switches to the selected state. If the Angle mode is selected the header label on the Android app is set to the angle corresponding to the section where the canvas space was touched. This value is used to determine the direction of movement and calculate the number of steps the actuator should move to reach the given position. If the Rises mode is selected, the program waits until the start button is pressed to start a leg rise cycle. The leg rise cycle is composed by a flexion and extension. This code can be found on Appendix A.

4.2.3 Experimental intention of movement detection

To detect the intention of movement and algorithm based on an ANN was designed, trained and adapted for control purposes. This algorithm was made with the Keras Python library which is intended for the creation of neural network models. To be able to use a neural network for the movement intention detection of the smart brace user the possible sources of information, that would reveal what is the desired movement, were identified and analyzed.

These sources are the angular displacement Δθ of the encoder and the binary states of the interrupters initially placed on the exoskeleton's foot support belt and base. These sources of data are used as the input features of the model. Based on the leg movement evaluation it was assumed that when the user has the intention to flex the leg, the angle difference is negative and smaller than some threshold value of -15. Oppositely, if the user's intention is to extend the leg, the angle difference would be positive and bigger than some threshold value of 15. Both of these assumptions are based on the fact that it's possible that the user will not press any of the buttons but still try to move, therefore the angle displacement Δθ will always be considered to detect the intention of movement. If the angle difference is negative and at the same time the Button 1 placed on the foot belt is pressed, the system should become more sensitive given that it would have more information to decide that the user is trying to flex the leg. This would be achieved by reducing the threshold value from -15 to -10. The same would happen if Button 2 is pressed while trying to extend the leg, the threshold value would be reduced from 15 to 10. Besides these instances there could be some accidental button presses opposite to the direction of movement, however since the encoder's position is always considered a movement should be possible anyways but with the peculiarity that I would be activated after surpassing a bigger threshold of -25 when contracting and accidentally pressing Button 2 and 25 when extending and accidentally pressing Button 1. If a given movement is in between the range of the positive and negative thresholds it would not be sufficient for the system to determine if the user wants to move the leg at all so no movement would be generated. The three possible actions after detecting an intention are to flex, extend or stop the smart brace movement. These outcomes were used as different classes for classifying the intention of the user and a simplified estimation about the relevance of the classes with the labels Very high, High ,Very low and Low was defined on Table 4..

Table 4.3: Expected Class outputs based on deduced input features values.

	Input Features			Classes		
Instance	Button1	Button2	Δθ	Flex	Extend	Stop
1	1	0	Δθ <-10	Very high	Very low	Low
2	1	0	-10<Δθ<25	Low	Very low	High
3	1	0	Δθ>25	Low	High	Low
4	0	1	Δθ<-25	High	Low	Low
5	0	1	-25<Δθ<10	Very low	Low	High
6	0	1	Δθ>10	Very low	Very high	Low
7	0	0	Δθ<-15	High	Low	Low
8	0	0	-15<Δθ<15	Low	Low	Very high
9	0	0	Δθ>15	Low	High	Low

To train an ANN for classification purposes a decent amount of training and testing data is required, however this data is not available at the moment since to collect it the smart brace would have to be tested several times by many users under many circumstances to collect the necessary data for training; data mining is beyond the scope of this project. However, it’s still possible to test the model’s potential of detecting the intention of movement by generating synthetic data based on the input features and the expected class outputs defined above. This is done with the intention of transfer learning to the live data provided by the smart brace’s sensors[23]. To achieve this the data was generated using the following equation:

$$P_x = (1 - P_a) \times w \tag{4.12}$$

Where P_x is the probability of the estimated class,
P_a— the probability of the class with preference.
w— a scaling factor dependent on the relevance of the estimated class.

To create the data the thresholds were placed on a table in such a way that at the threshold value the class with preference has a probability value close to 0.5, which represents half of the total probability. Then the probability of the class with preference is extended incrementally by a factor of 0.2. This increment is parallel to the angle displacement until the value reached is equal to the total probability of 1, from this point the probabilities of the other classes are set to zero. The class with preference is the one that has the highest probability to be the outcome depending on the input. For instance, on Table 4.4. as the value of angle displacement decreases the probability of flexing increases from the threshold value of -10 at which the probability value of the Flex class is bigger compared to the Extend and Stop classes. The values of the Extend and Stop classes are calculated by subtracting the total probability of 1 and the value of the class with preference, times a scaling factor. The scaling factor was chosen by means of the observations made on Table 4.3. Note that the sum of scaling factors is equal to one, meaning that the remaining probability after the subtraction is divided proportionally to the scaling factor.

Table 4.4: Example of generated synthetic data for when the button 1 is pressed.

Button1	Button2	Δθ	Flex		Extend		Stop	
1	0	-25	Flex	1	Pext =	0	Pstop =	0
1	0	-20	class is	0.84	(1-Pa)w	0.04	(1-Pa)w	0.12
1	0	-15	Pa	0.64		0.09		0.27
1	0	-10		0.44	w=0.25	0.14	w=0.75	0.42
1	0	-5		0.24		0.19		0.57
1	0	0		0.07		0.23		0.7
1	0	5	Pflex =	0.01	Pext =	0.01	Stop	0.98
1	0	10	(1-Pa)w	0.07	(1-Pa)w	0.15	class is	0.78
1	0	15		0.14		0.28	Pa	0.58
1	0	20	w=0.333	0.21	w=0.666	0.41		0.38
1	0	25		0.27		0.55		0.18

Table 4.5: Example of generated synthetic data for when the button 2 is pressed.

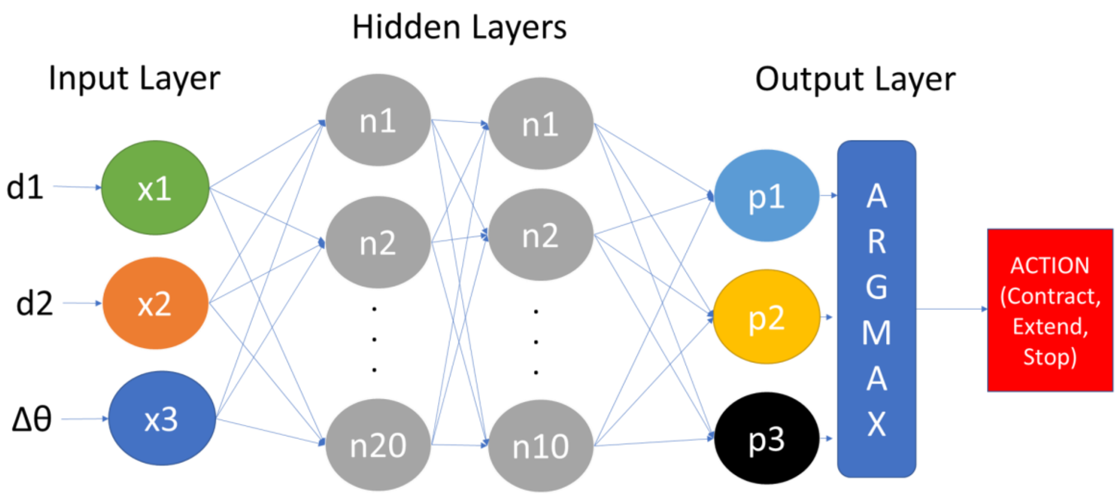
Input Features			Classes					
Button1	Button2	Δθ		Flex		Extend		Stop
0	1	-25	Pflex =	0.55	Pext =	0.27	Stop	0.18
0	1	-20	(1-Pa)w	0.41	(1-Pa)w	0.21	class is	0.38
0	1	-15		0.28		0.14	Pa	0.58
0	1	-10	w=0.666	0.15	w=0.333	0.07		0.78
0	1	-5		0.01		0.01		0.98
0	1	0	Pflex=	0.24	Extend	0.04	Pstop=	0.72
0	1	5	(1-Pa)w	0.19	class is	0.24	(1-Pa)w	0.57
0	1	10		0.14	Pa	0.44		0.42
0	1	15	w=0.25	0.09		0.64	w=0.75	0.27
0	1	20		0.04		0.84		0.12
0	1	25		0.00		1.00		0.00

Table 4.6: Example of generated synthetic data for when no buttons are pressed.

Input Features			Classes					
Button1	Button2	Δθ		Flex		Extend		Stop
0	0	-25	Flex	0.8	Pext =	0.08	Pstop =	0.12
0	0	-20	class is	0.6	(1-Pa)w	0.16	(1-Pa)w	0.24
0	0	-15	Pa	0.4		0.24		0.36
0	0	-10		0.2	w=0.4	0.32	w=0.6	0.48
0	0	-5		0		0.4		0.6
0	0	0		0		0		1
0	0	5	Pstop =	0.4	Extend	0	Pstop =	0.6
0	0	10	(1-Pa)w	0.32	class is	0.2	(1-Pa)w	0.48
0	0	15		0.24	Pa	0.4		0.36
0	0	20	w=0.4	0.16		0.6	w=0.6	0.24
0	0	25		0.08		0.8		0.12

The neural network receives three input features, what happens inside is that each value from the input layer is distributed to each one of the 20 neurons from the first hidden layer giving each connection or synapse between neurons a random weight. The same happens inside the hidden layers giving each connection between the first layer of 20 neurons and the second one of 10 neurons a random weight until it is mapped to the output layer with random weights too. The output layer then shows a result using a probability distribution. Then the decision with the highest probability of whether the user wants to contract, extend, or stop the movement is selected by means of an Argmax function. This structure is shown on Figure 4.14.

Figure 4.14: Architecture of the neural network



Each synapse of a neuron represents that the neuron takes each input value multiplied by each of their random connection weight and sums it all together plus the bias, then the result provided by the neuron activation function will return a binary value of 0 or 1 using a ReLU activation function depending on the value of the calculated sum. Each neuron undergoes this process until each of the three final decision neurons get a value. A probability value between 0 and 1 from a Softmax function is the output of the neural network. The value with the highest the probability represents the chosen decision, the sum of those three values equals 1 which makes for the whole distribution of probabilities.

At first, the neural network won't get the expected outputs correctly so that's why it needs to get trained by calculating each error made. This is done by comparing the actual output value with the expected one. Then an adjustment vector is calculated by multiplying the difference of the compared outputs by the derivatives of the Relu function evaluated on the outputs gotten. To get new weights for each neuron a dot product calculation is made between the adjustment vector and the input layer. This process iterates many times until the network becomes more accurate by adjusting each weight and bias applied on the neuron.[24]

Initially, a CSV file is uploaded with the synthetically generated data to train the network by separating the input data from the output values. The network is built sequentially, layer by layer. First, it is declared that the input layer has 3 neurons with 3 input values which will be densely connected to those 20 of the first hidden layer created. This is the mapping process which builds each one of the synapses between these two layers that will return an output value of 0 or 1 for each neuron after undergoing the ReLu activation function process. The same process is made when the second hidden layer is created with 10 neurons, with the singularity that a dropout of 20% is applied. This implicates that 20% of the neurons are randomly disconnected as neighbor neurons tend to rely on the specialization of each other, which could cause the model to become specialized only for the training data set given. Then the output layer is created with three neurons densely connected to those of the last hidden layer which weren't ignored, returning a final output value between 0 and 1 after undergoing the Softmax activation function to get the probability distribution of them. This neural network compiles the three existing categories to adjust itself by comparing the outputs and calculating the error made, it iterates this process 1000 times or epochs. The Adam gradient descent optimizer is used, where after getting a prediction the algorithm goes back again make the necessary adjustments to the weights based on how accurate the model was, to get new predictions until the model gets more accurate. To see the full code for training the neural network go to Appendix B.

4.2.4 Control modalities

The algorithms were written in the Python programming language and use functions of several libraries. The PyModbus library is used to read the encoder degrees and buttons and activate the stepper motor by Modbus RTU communication, the NumPy library for mathematical operations with arrays and Keras to initialize the ANN and load the pretrained weights. For the process of leg rises three control modalities were proposed.

During the movement analysis stage the main characteristics of the desired movements were investigated. To assist the leg extension and flexions involved in the leg rises the encoder was programmed to detect the intention of movement through the use of an angular threshold. The angular threshold is created after the initial angular position is read and it is defined as the initial position plus an activation angle of 15 degrees, however this parameter can be changed to make the system less or more sensitive. If the angular threshold is surpassed the stepper motor generates an angular increment with the set speed in *Rpm*; the speed on the knee joint would be reduced due to the gear ratio. This process is repeated until the leg reaches the final position of 120 degrees, once it happens the stepper motor waits 5 seconds and slowly returns to the initial position with a given speed. This code, included in Appendix D, is represented in a commented section.

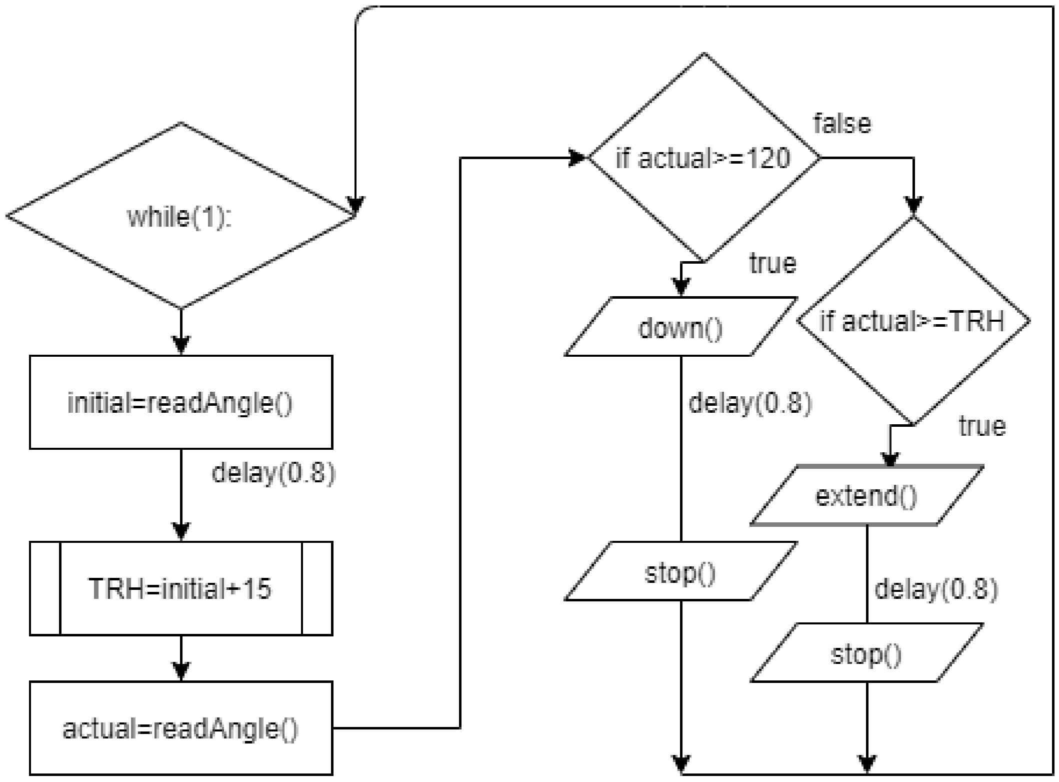


Figure 4.15: Simplified flowchart of the basic leg rises control algorithm.

For the continuous leg rises algorithm the angular position of the encoder sensor is read, followed by the state of the buttons. In the case that the position read is smaller than 120 degrees and the first button is pressed ,the smart brace generates a leg extension, if instead the button 2 is pressed a flexion movement is generated. When the position is greater than 120 degrees the system generates a corrective movement given by a leg flexion. This code can be found on Appendix C.

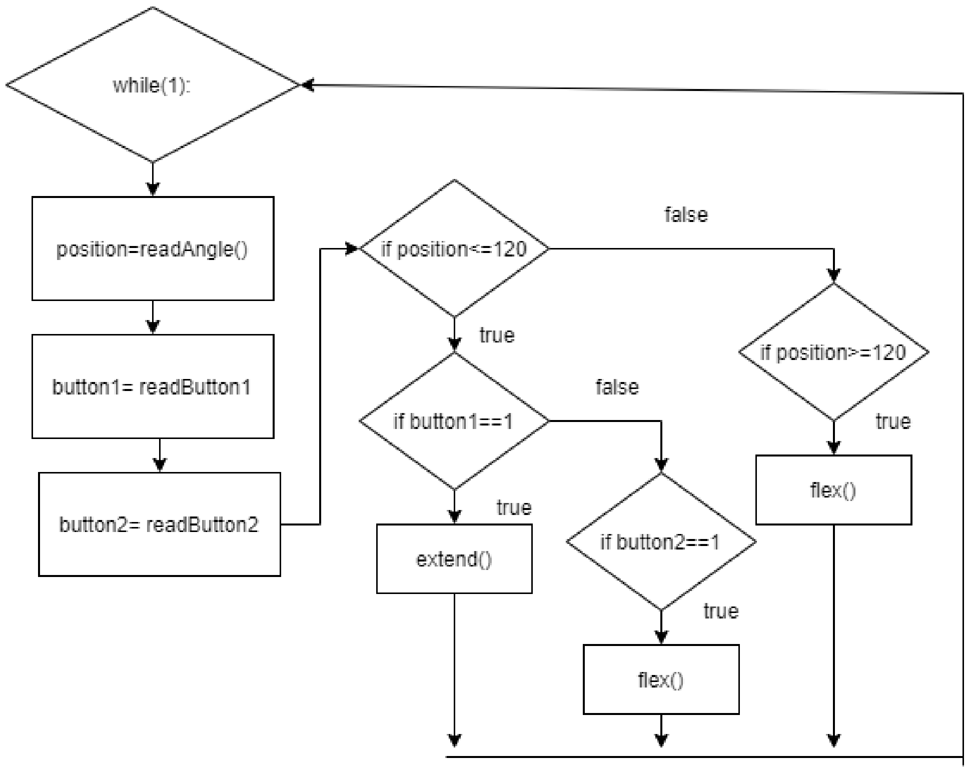


Figure 4.16: Simplified flowchart for the continuous leg rises algorithm.

For the ANN based leg rises modality the model was initialized and the weights were loaded into it. Then the initial angular position of the encoder is read followed by a delay. The purpose of the delay is to give the system some time to identify whether the position has changed, this wouldn't be possible without it. After this, the actual position is read and the angular displacement Δθ is determined by calculating the actual position minus the initial position. Next, the state of the input buttons is read and put into an array with the angular displacement Δθ. This array is then passed through the trained neural network and an Argmax function. The Argmax function returns the index of the class with the highest probability. This index value is assigned to the action variable and it is the one that is read by the if statements; it determines the action to be taken by the system. If the action is equal to 0 the flexion movement is generated. If the action equals 1, the extension movement is the outcome. Finally when the action equals 2, the stop action is selected. These functions are followed by a delay to allow the system to preform them. The flowchart for this algorithm is described on Figure 4.17 .

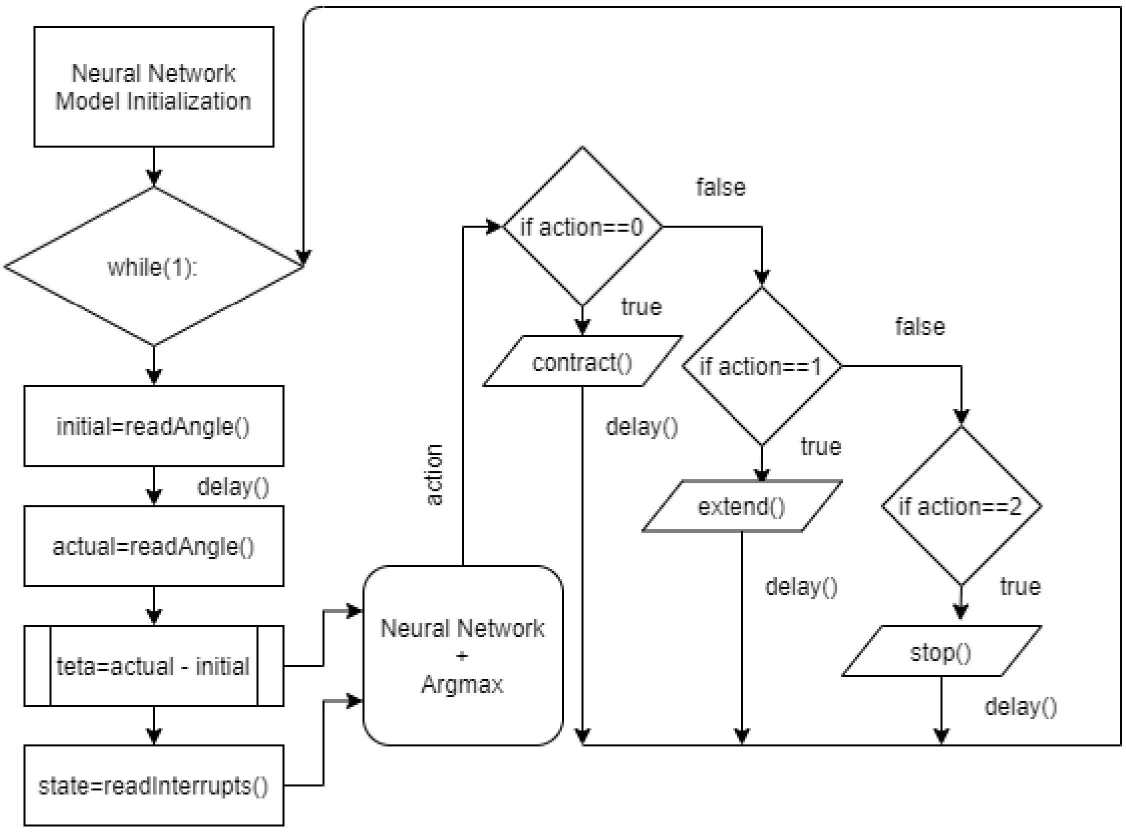


Figure 4.17: Simplified flowchart of the leg rises control algorithm based on an ANN.

To aid the process of standing and walking the exoskeleton uses an algorithm based on a finite state machine approach as described on Figure 4.18. This approach has different states and transitions that follow a logical sequence. In the beginning the person would be sitting, this state is defined as Sitting State. If the button is pressed the smart brace would start to create an extension movement that takes the user from the Sitting State to the Upright Standing State. From this point if the user presses the button again the system starts to generate a simplified gait cycle defined as Walking State which is composed of a leg flexion followed by a leg extension, in a ROM of 120 to 60 degrees and at the set speed in *Rpm*. When this cycle ends the user is back at the Upright Standing State. If during this cycle the user causes a movement that is beyond the defined ROM the system moves to the Error State. In order to return to the Walking State a correction is generated depending on the boundary that was surpassed. This code is fully included on Appendix E.

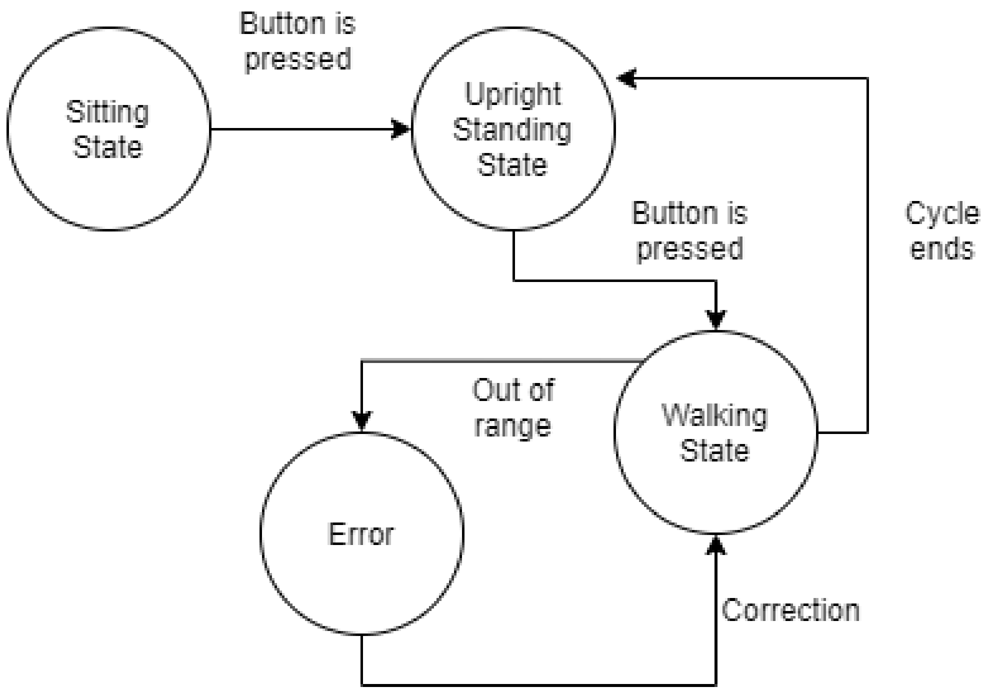


Figure 4.18: Finite state machine algorithm for standing and walking.

4.3 Implementation and testing of prototype

The process of prototype construction and testing consists of two sections where the actuator system, control system prototype were implemented and then tested under different circumstances where they could potentially fail. The outcomes of such tests were reported and then the relevant parameters were adjusted to get the best possible performance.

4.3.1 Control and actuator system implementation

First the selected stepper motor was incorporated to the existing motor support of the smart brace by four M5 screws with hexagonal head of 5mm of diameter and 20mm of length and the provided dented belt was placed as shown on Figure 4.19.

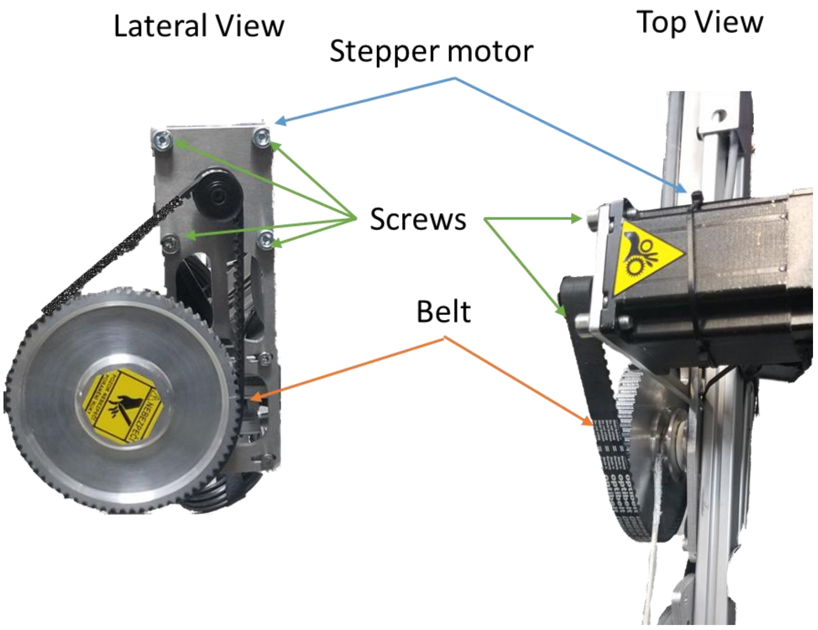


Figure 4.19: Incorporation of the stepper motor and dented belt.

Then the selected encoder was fixed by three M1 screws, of 1 mm of diameter and 20mm of length, and six nuts nuts made to fit the same diameter, Figure 4.20.

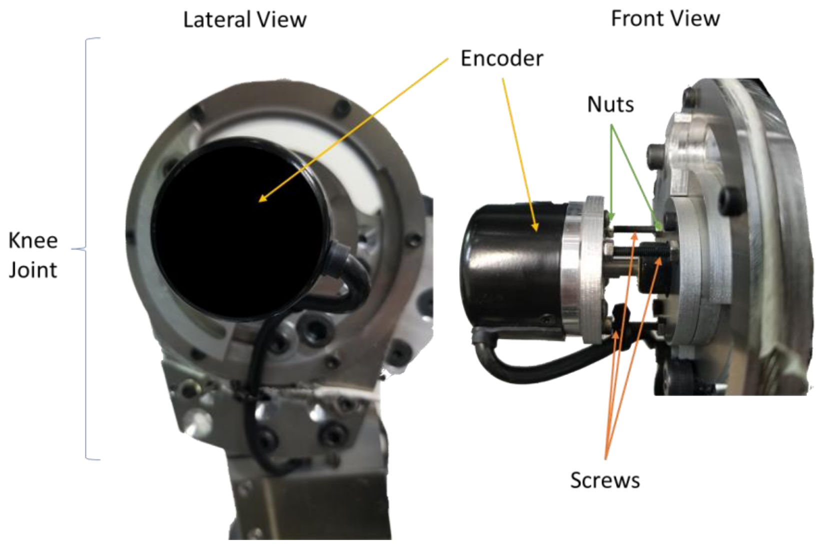


Figure 4.20: Incorporation of the encoder.

The next step was to build the control system. This was achieved by placing the STM32 microcontroller board, button circuits, some pins for connection of the sensors, emergency switch and stepper motor driver on a prototyping circuit board and soldering them following the proposed schematic on the Subchapter 4.1.5. Finally the power supply and the stepper motor phases were connected. The final prototype is illustrated on Figure 4.21.

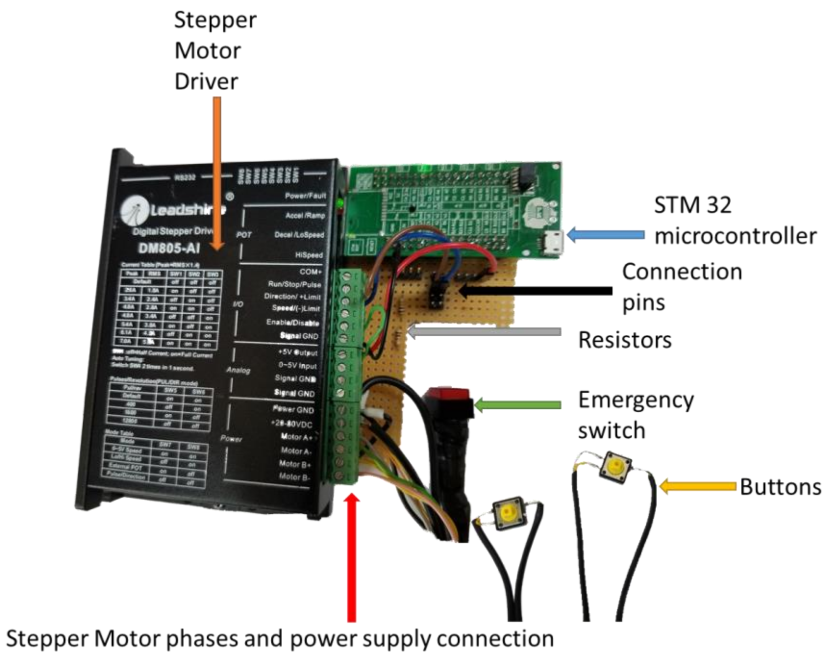


Figure 4.21: Control system prototype.

After this, the wireless actuator testing module was soldered to a prototyping circuit board based on the circuit proposed on the Subchapter 4.1.6. It includes the Bluetooth module, Arduino Nano, resistors and transistors and finally a 4 pin connector for powering the module and sending the signals to the stepper motor driver. This module is depicted on Figure 4.22.

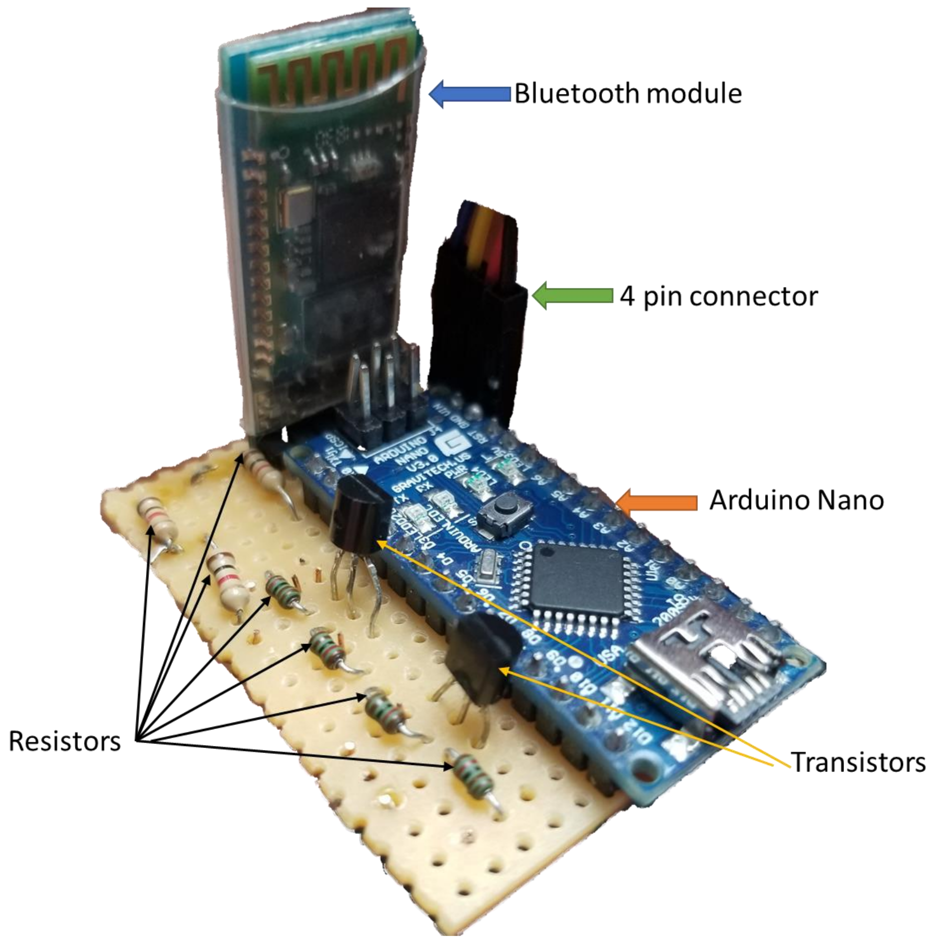


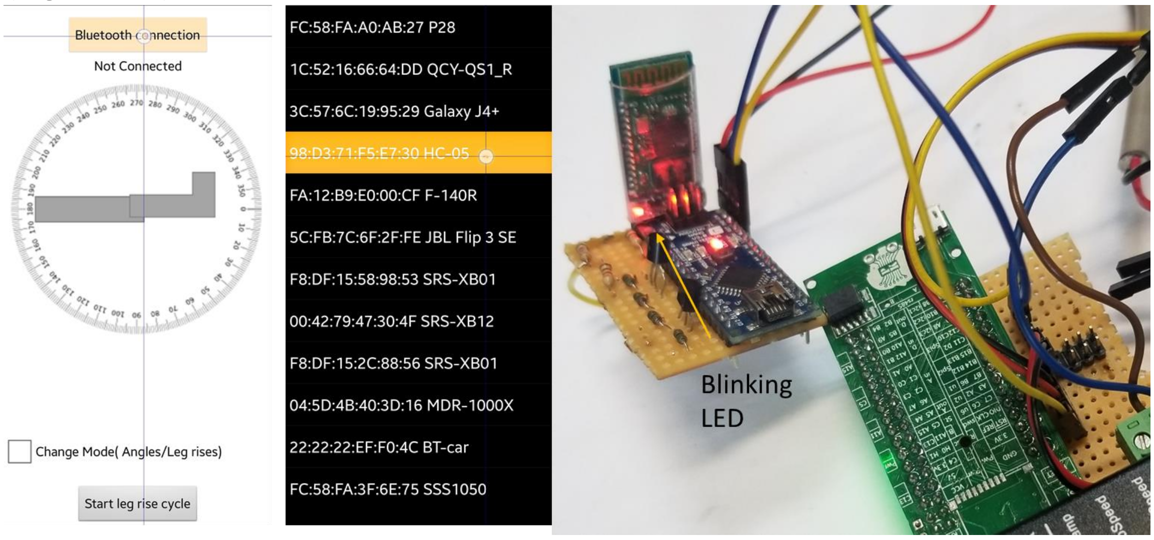
Figure 4.22: Wireless actuator testing module.

4.3.2 Testing of complete system

At first a continuity test was performed to evaluate if the components of the control system were properly soldered or connected and if no short circuits were created during the process. This was done by using a multimeter in continuity test mode and placing its leads at the ends of the measured terminals or pins; the state of the connection was evaluated and the necessary corrections were applied.

The Android application was compiled and tested based on the proposed structure on subchapter 4.2.2. On the canvas space at the center, an image showing the angles where the exoskeleton can move was inserted and on top of that an indicator image sprite, symbolizing the smart brace's shank, was centered to match the pivot point of the leg. At the bottom a check box to change the testing modality and a button to start the leg rise modality were inserted. This structure was evaluated on a real Android smartphone (Samsung Galaxy S8+) by means of enabling the developer options to show the layout bounds.

Then the prototyped actuator testing wireless module was assessed by a connecting it to the stepper motor driver and trying to stablish a Bluetooth communication 20 times with the Android application for actuator testing. The number of times the connection was successful versus the times it didn't work were recoded to evaluate the performance of the system. The pointer location and show taps Android developer options were used to demonstrate the functionality of the application. The Bluetooth module is selected and the communication is stablished when its LED starts to blink intermittently every second (Figure 4.23).



```
FC:58:FA:A0:AB:27 P28
1C:52:16:66:64:DD QCY-QS1_R
3C:57:6C:19:95:29 Galaxy J4+
98:D3:71:F5:E7:30 HC-05
FA:12:B9:E0:00:CF F-140R
5C:FB:7C:6F:2F:FE JBL Flip 3 SE
F8:DF:15:58:98:53 SRS-XB01
00:42:79:47:30:4F SRS-XB12
F8:DF:15:2C:88:56 SRS-XB01
04:5D:4B:40:3D:16 MDR-1000X
22:22:22:EF:F0:4C BT-car
FC:58:FA:3F:6E:75 SSS1050
```

Figure 4.23: Testing of Bluetooth connection with the actuator testing module.

After the connection was stablished some commands were sent to the exoskeleton to test if the leg rise cycle was activated and if the desired angular position was achieved. These results were compared in the same way as the Bluetooth connectivity. To prevent the smart brace from falling while testing, some weights were placed on the foot link to keep it grounded as shown on Figure 4.24.

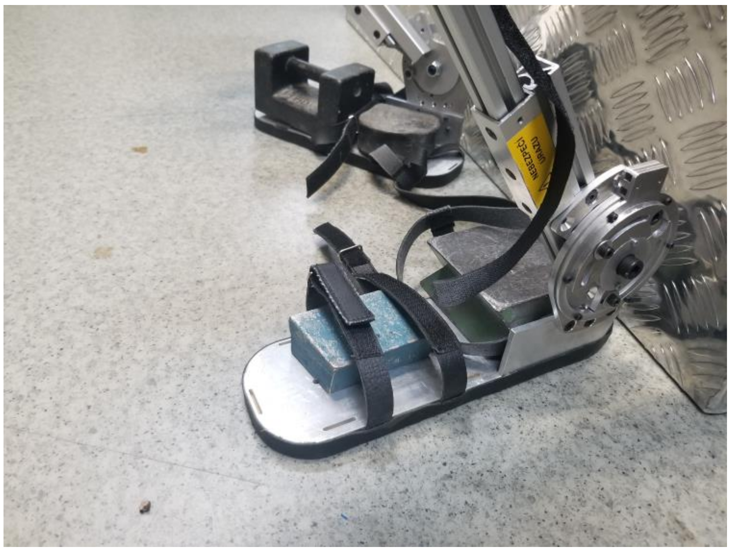


Figure 4.24: Weights placed on the foot link.

Then the neural network was trained with the generated synthetic data.
To evaluate the control algorithms a human subject tried the smart brace and time-angle graphs were implemented by reading the angular position of the encoder while the control algorithms were running.

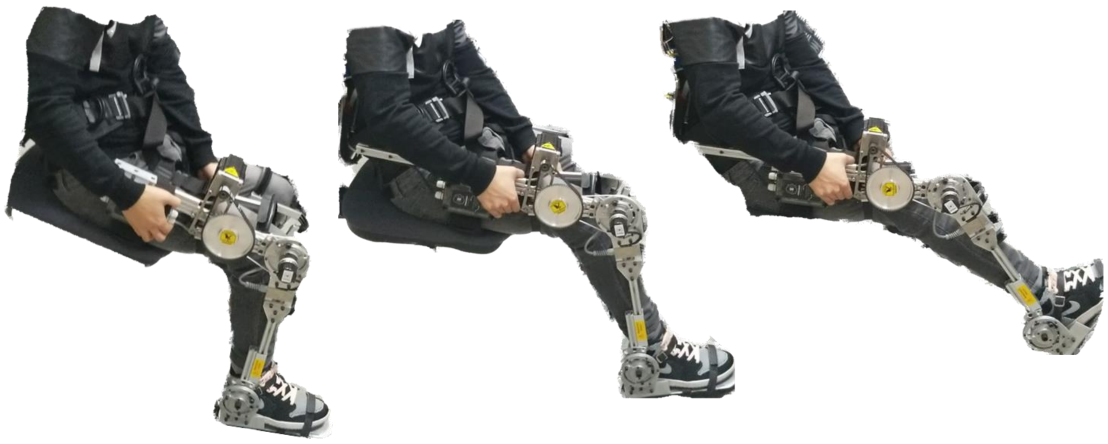


Figure 4.25: Exoskeleton aided leg rise on a human subject.

5 Results

The results contain tables, graphs and bar charts with examples of final values.

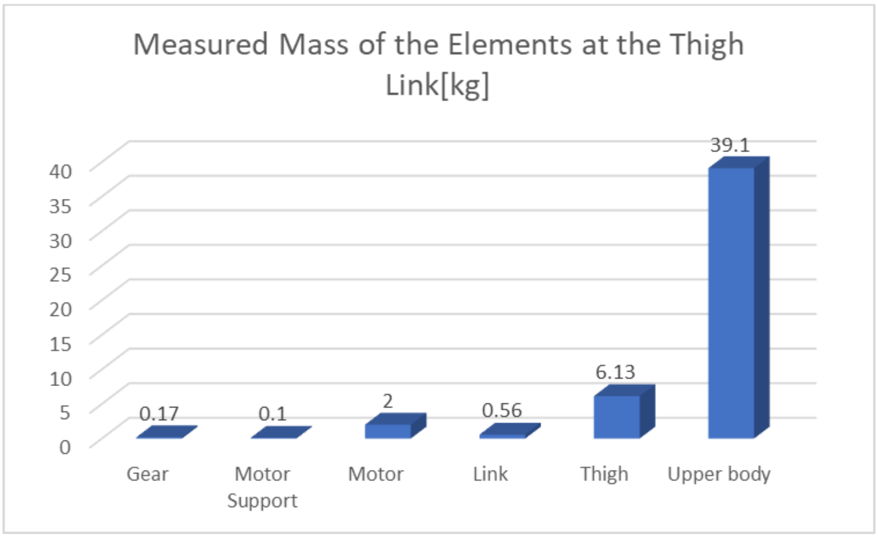


Figure 5.1: Measured mass of elements that exert a weight on the thigh link of the exoskeleton.

Table 5.1: Forces and distances of elements with respect to the brace's knee joint.

Element	Force [N]	Distance from joint to COM [m]	Distributed Load [N/m]	Load segment extension[m]
Gear	1.666	0.14	-	-
Motor Support	0.98	0.19	-	-
Motor	19.6	0.19	-	-
Link	5.488	0.32	8.57	From 0 to 0.64
Thigh	60.074	0.24	127.06	From 0 to 0.48
Upper body	383.18	0.56	2375.94	From 0.48 to 0.64

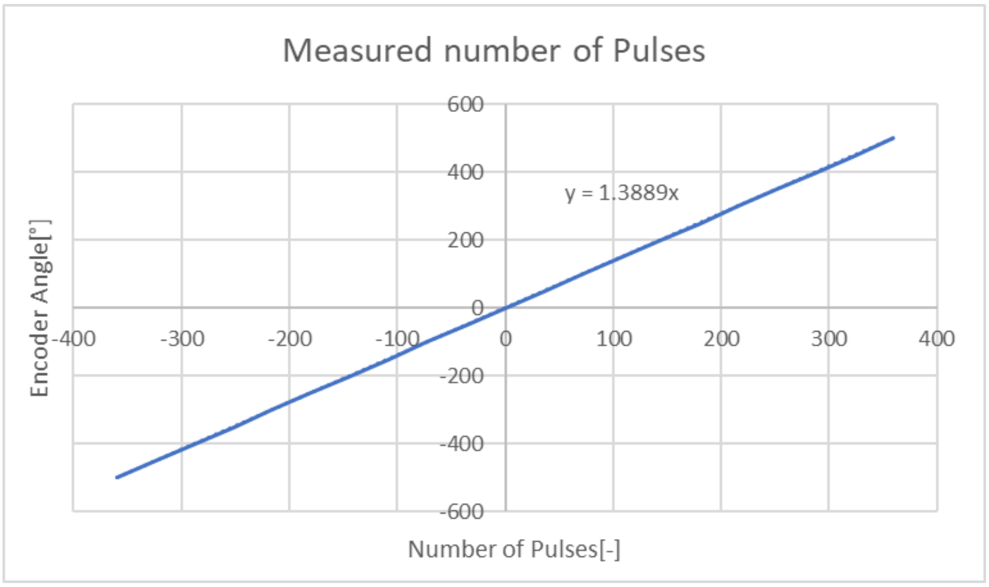


Figure 5.2: Measured number of pulses to reach a given encoder Angle.

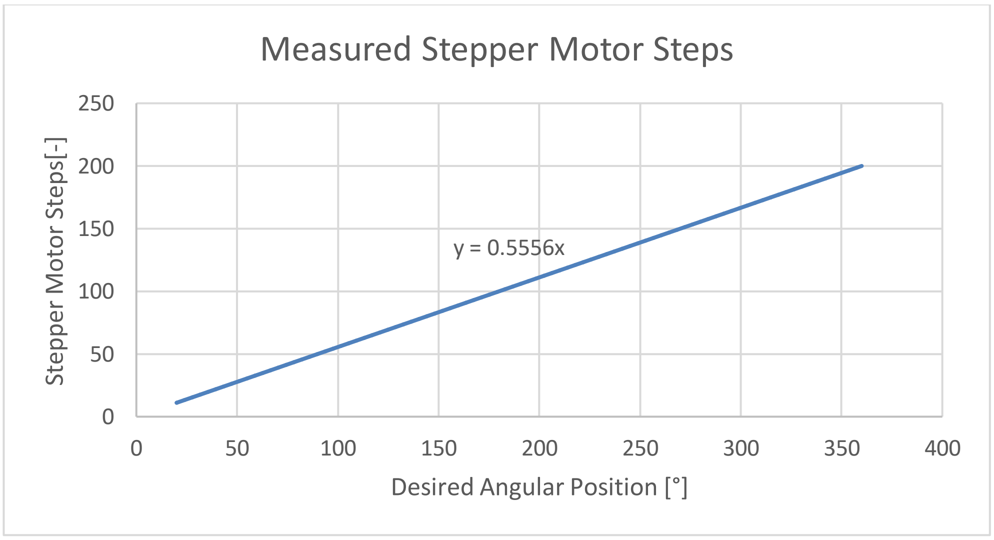


Figure 5.3: Measured number of stepper motor pulses to reach a desired angular position.

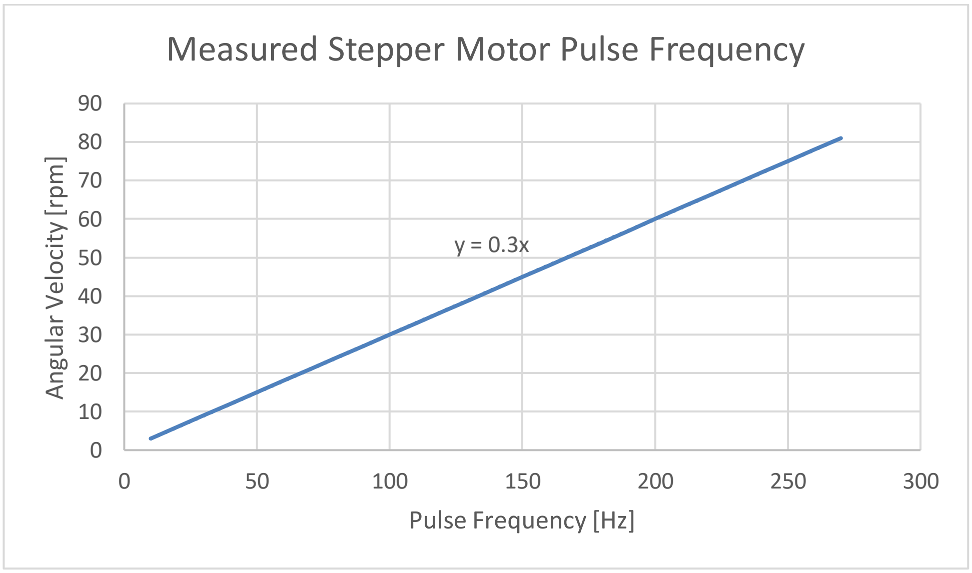


Figure 5.4: Measured *Rpm* at a given pulse frequency of the stepper motor.

Table 5.2: Continuity test results of the control system prototype

Measurement Between		Result	Improvement
GND terminal of Stepper driver	GND terminal of power supply	Continuous	
GND terminal of Stepper driver	GND terminal of STM32	Continuous	
GND terminal of Stepper driver	GND terminal of encoder	Continuous	
GND terminal of Stepper driver	Button 1 resistor	Pin of the button got broken	A new button was soldered and tested
GND terminal of Stepper driver	Button 2 resistor	Continuous	
+24V terminal of Stepper driver	Vcc terminal of power supply	Continuous	
+5V terminal of Stepper driver	+5V terminal of STM32	Continuous intermittently	Pin soldered again
+5V terminal of Stepper driver	+5V terminal of encoder	Continuous	
+5V terminal of Stepper driver	Button 1 input pin	Continuous	
+5V terminal of Stepper driver	Button 2 input pin	Continuous	
Enable pin of STM32	Enable pin of Stepper driver	Continuous	
Pulse pin of STM32	Pulse pin of Stepper driver	Continuous	
Direction pin of STM32	Direction pin of Stepper driver	Continuous	
Pulse counter pin of STM32	Pulse pin of STM32	Short circuit due to exposed cable	New longer cable was soldered
Encoder A pin of STM32	Encoder Phase A	Continuous	
Encoder B pin of STM32	Encoder Phase B	Continuous	
Interrupt 1 terminal of STM32	Button 1 output	Continuous	
Interrupt 2 terminal of STM33	Button 2 output	Continuous	

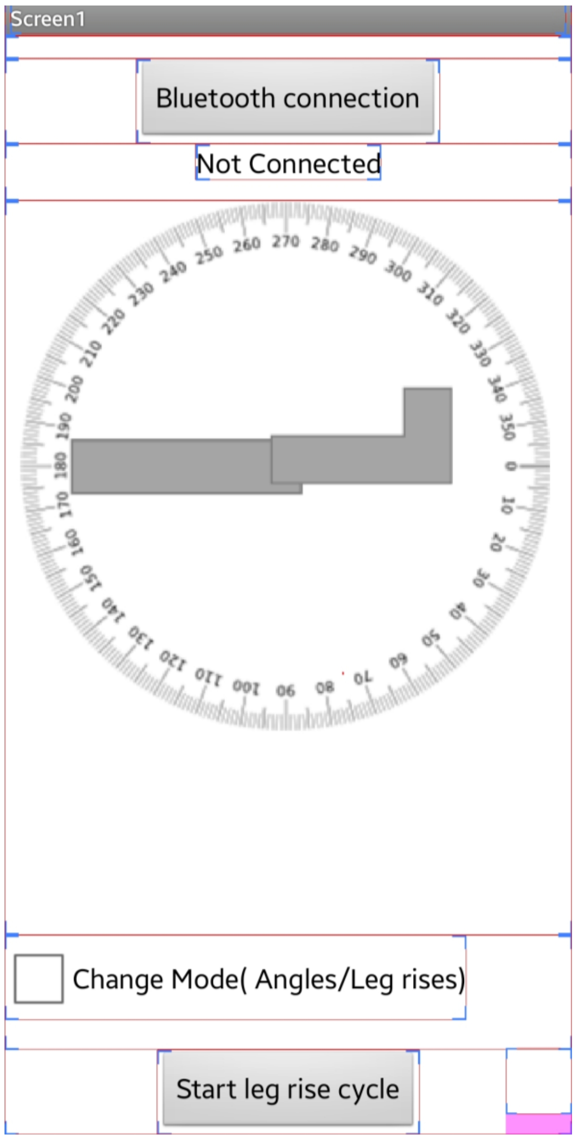


Figure 5.6: Result of the compiled layout of the wireless actuator testing application.

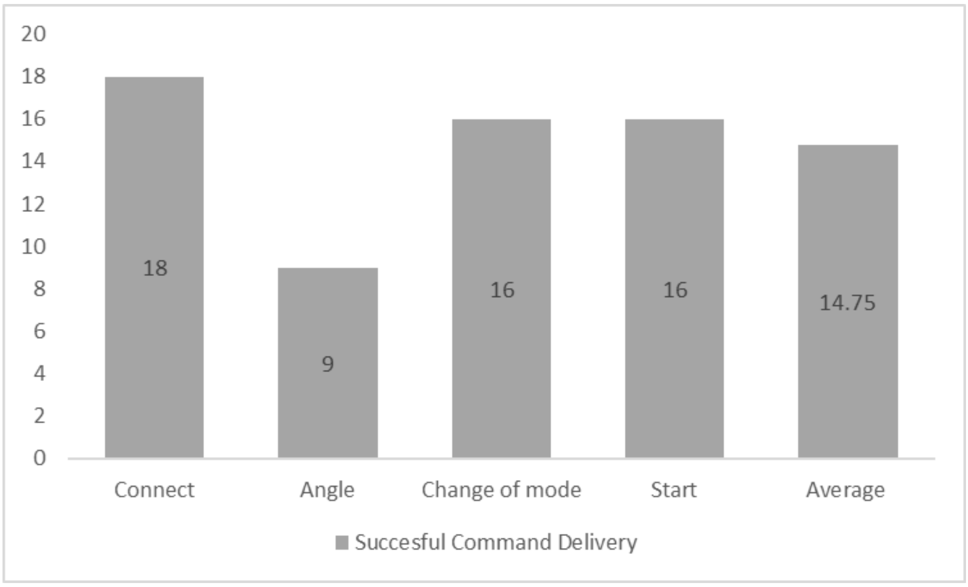


Figure 5.7: Testing of command delivery to the wireless actuator control module.

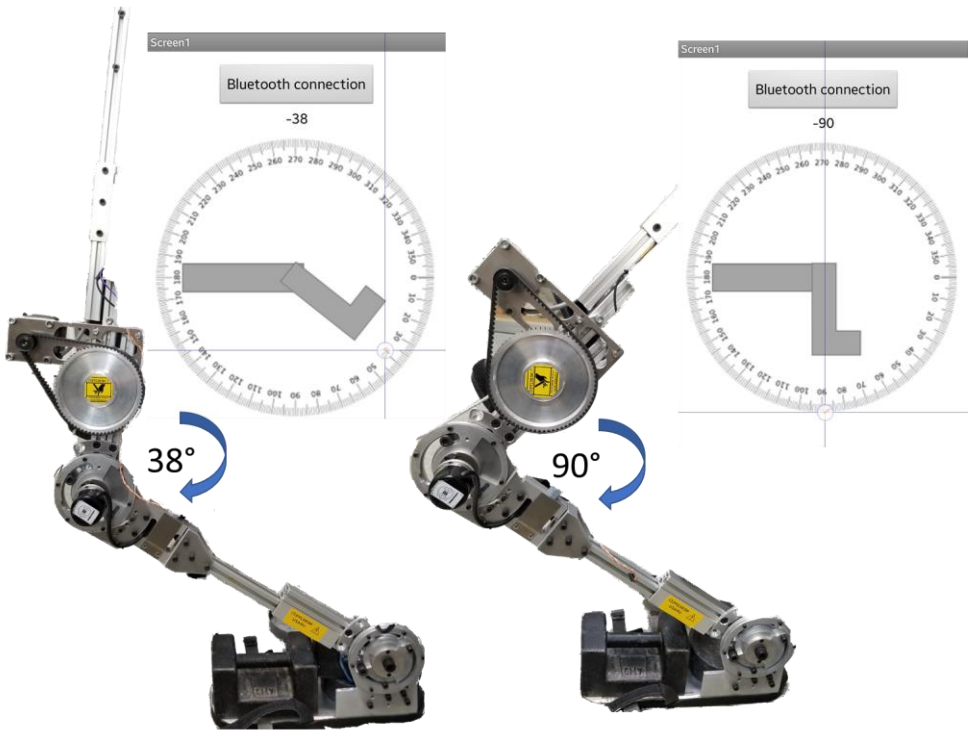


Figure 5.8: Testing of angle command delivery to the exoskeleton.

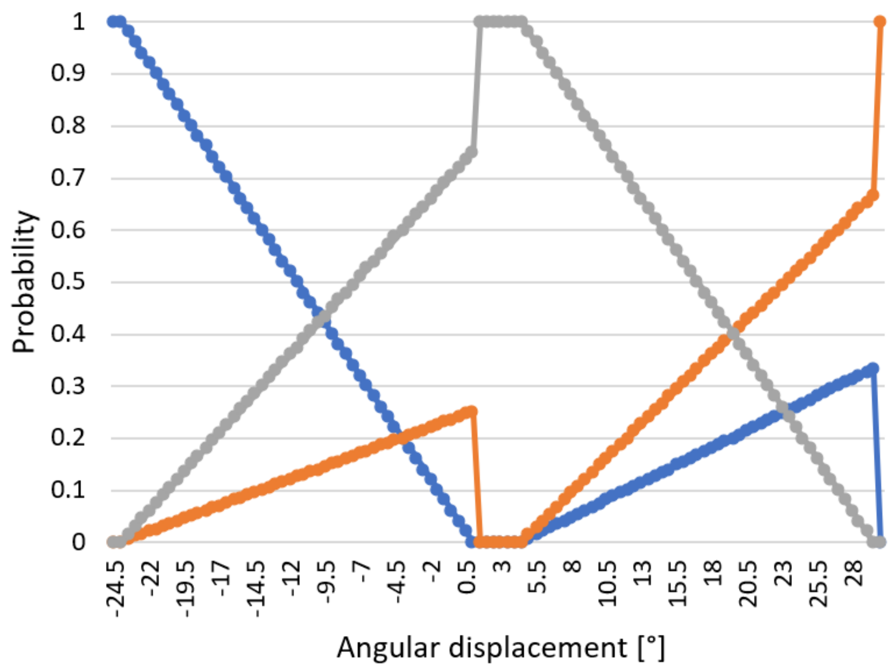


Figure 5.9: Graph of synthetic data generated for button 1 pressed.

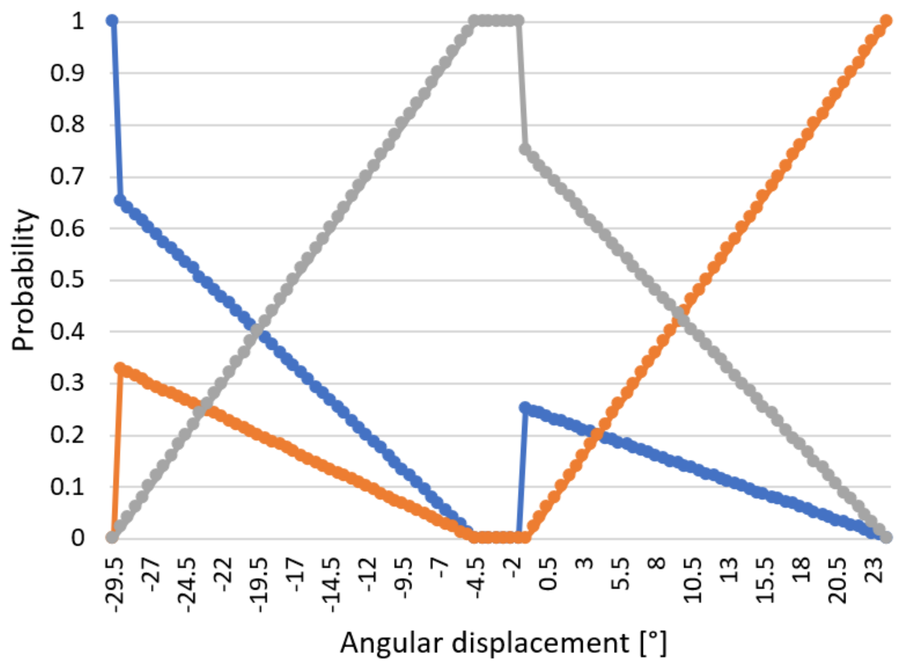


Figure 5.10: Graph of synthetic data generated for button 2 pressed.

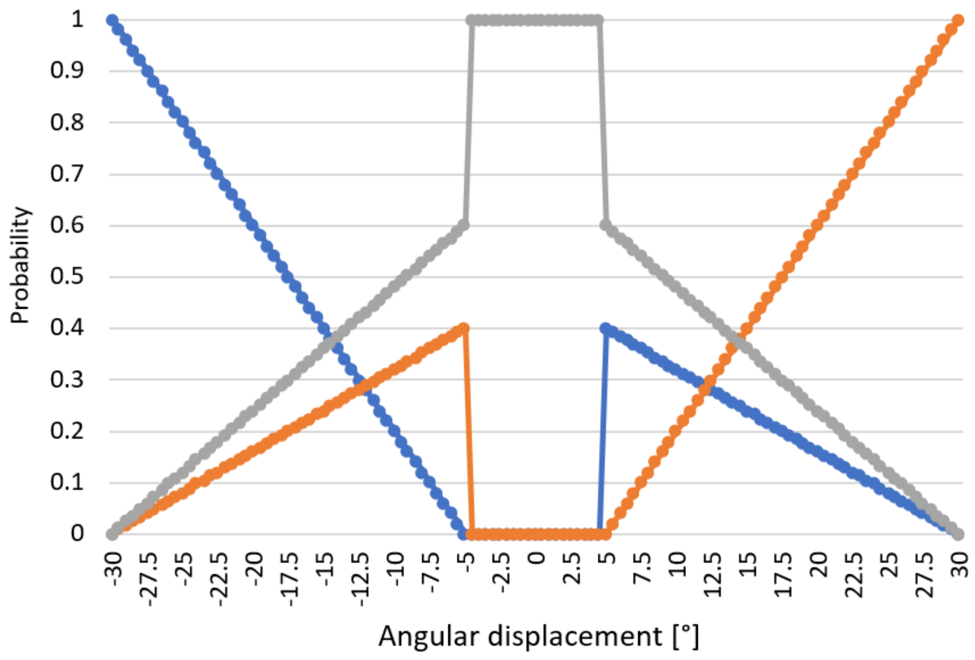


Figure 5.11: Graph of synthetic data generated for when no buttons are pressed.

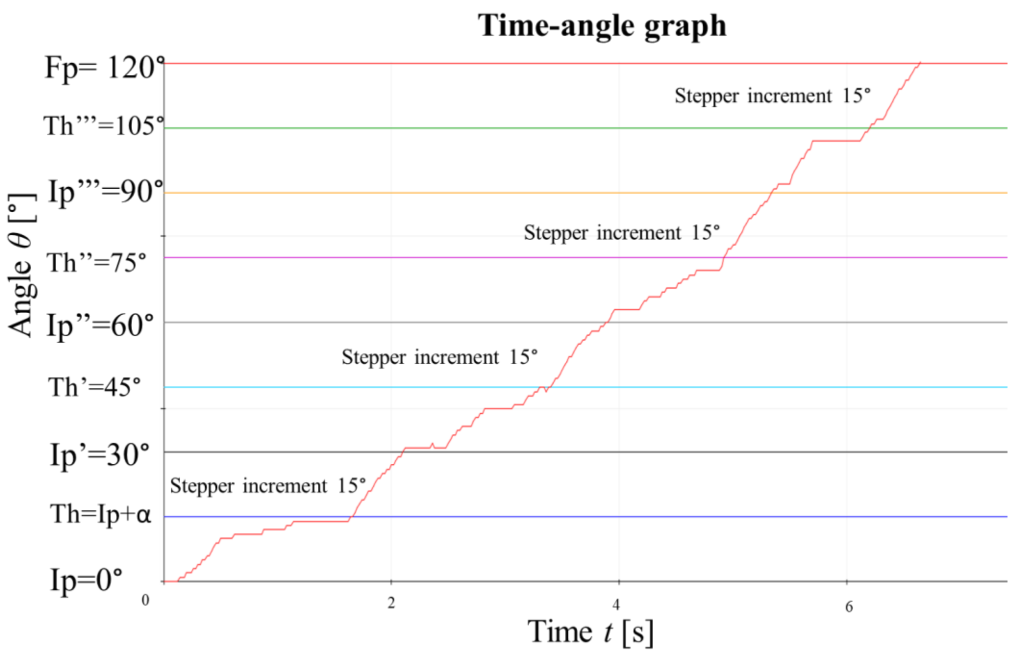


Figure 5.12: Time-angle graph of the leg rise function activated by thresholds.

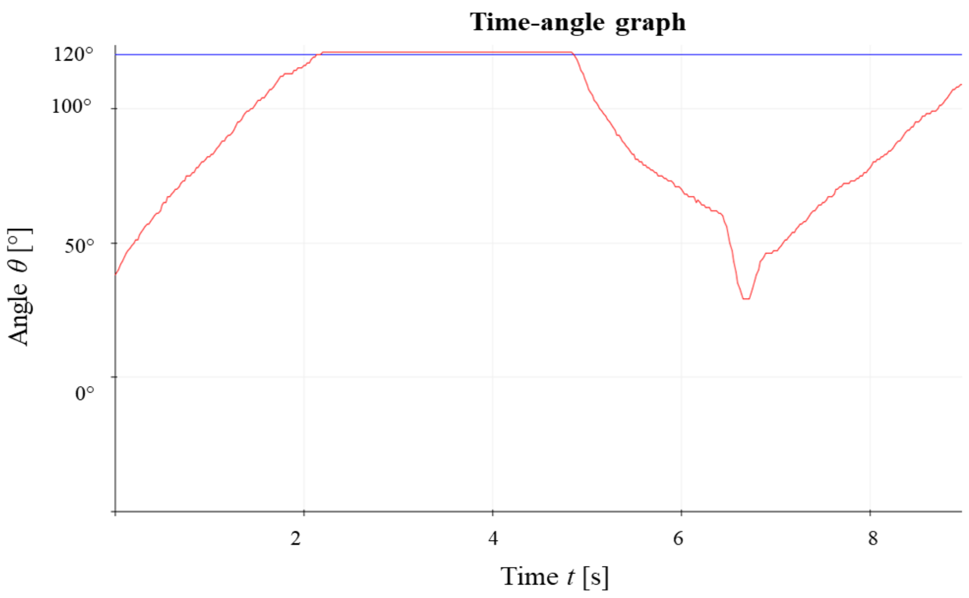


Figure 5.13: Time-angle graph of the walking algorithm generating a correction when the negative limit is passed,

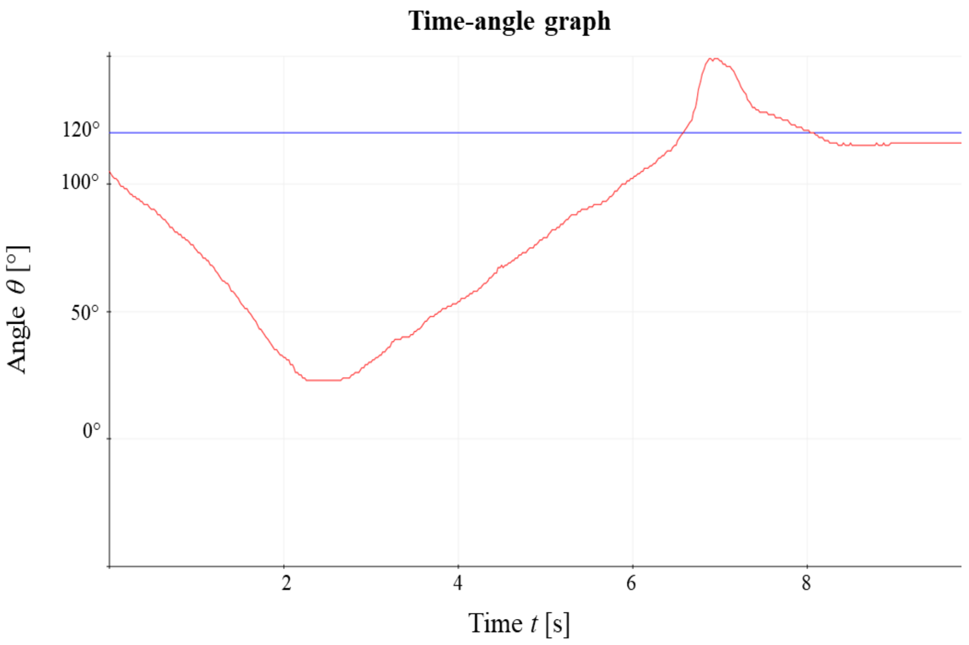


Figure 5.14: Time-angle graph of the walking algorithm generating a correction when the positive limit is passed,

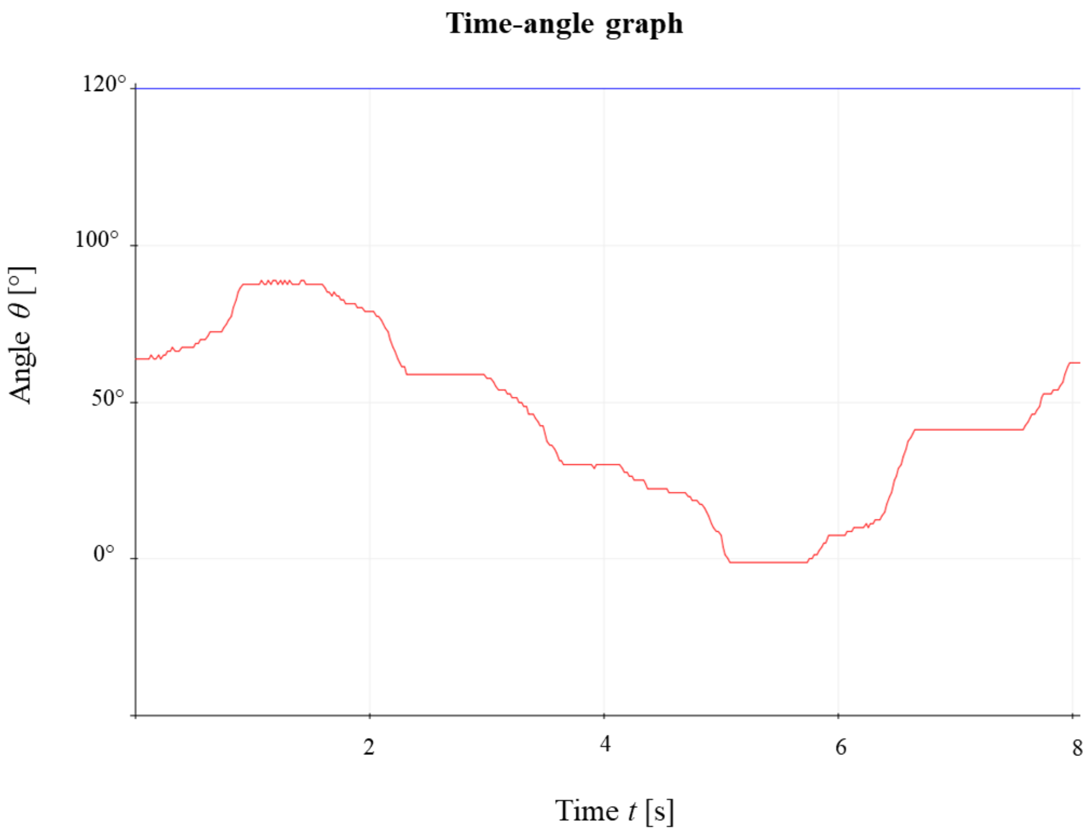


Figure 5.15: ANN mediated leg rise.

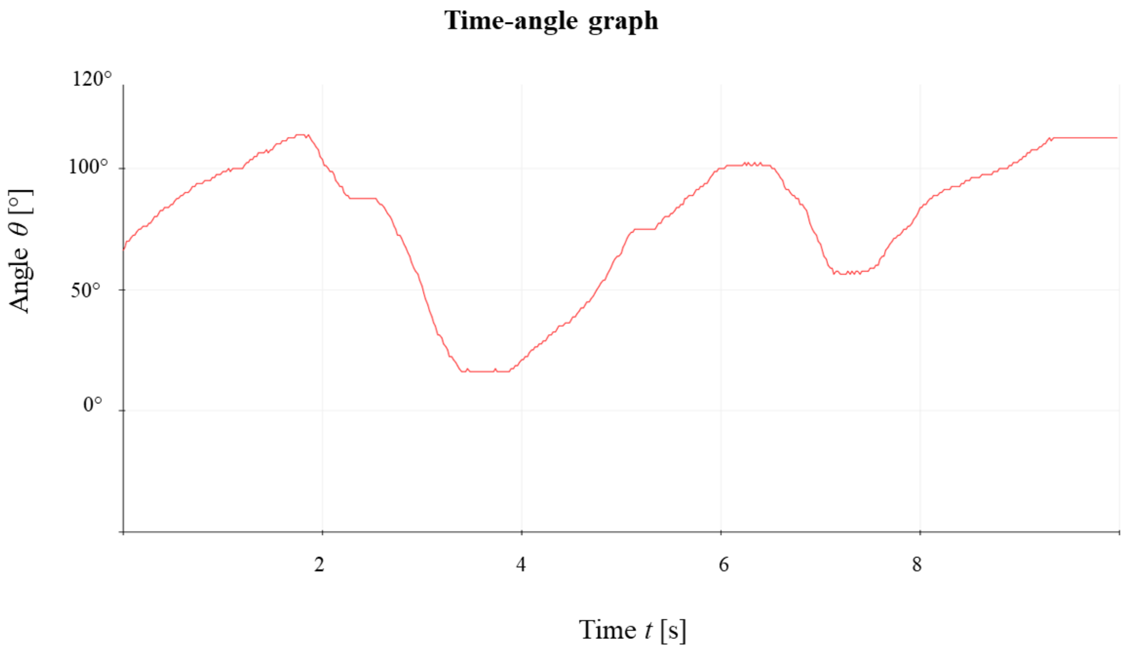


Figure 5.16: Continuous leg rises algorithm.

6 Discussion

The outcome of this project is a functional lower extremity smart brace control and actuator system which allows several control modalities and wireless communication. The calculated torque value that was used to select the stepper motor was a rough approximation which didn’t take in account the inertia of the elements involved on the movement or the friction of the system. However it was still able to produce a supportive torque which was the main goal.

During this process it was determined that the accuracy of the neural network was affected mainly by the number of epochs applied when training it. It was observed that the higher the number of epochs, the higher the accuracy. A value of 1000 epochs was chosen to train the network, which produced an accuracy of 98.16. Since the neural network was trained with an experimental synthetically generated data the accuracy value should just be used as a reference to perform adjustments to the network but not as a relevant indicator of the accuracy of the network.

The first operational mode based on leg rises was approached with three control modalities. The first one mediated by setting an angular threshold and increasing the position step by step worked as expected, however for it to be usable the speed of increment had to be very slow, compared to the other control modalities for leg rises. The continuous leg rises one, mediated by two buttons and an angular limit, worked well with the peculiarity that its performance was better when the step resolution was increased for microstepping of the stepper motor. This is because at the time when the user presses one button continuously the program tries to initialize the sent instruction continuously and it creates undesired vibrations. For the las one based on an ANN the delay and speed had to be increased for it to work optimally. The buttons that were placed on the foot link increased the sensitivity of the system as expected, however since the foot link is not fixed, there were some accidental button presses, so a new approach for placing the buttons on the foot link should be further investigated, perhaps if the foot link is immobilized the outcome would be better.

The second type of movement, standing up and walking, worked as expected when the limits were surpassed by creating a corrective movement. However it could be improved by allowing the user to go from the standing state back to the sitting state and also by replicating the full gait cycle and not just the phases 3, 4 and 5 a better outcome could be achieved.

The Bluetooth testing module faced some connectivity issues when trying to receive commands, this could have been because the delivered voltage was not enough.

The control code is functional but it could be improved by creating a visual interface not just for actuator testing but for control of the whole system, where the therapists could easily set up the parameter for rehabilitation depending on the patient’s needs. The mechanical system didn’t have an actuator system to aid the hip joint, which is also involved in the movement. In a future version the addition of such actuator at the hip position may allow a smoother movement with less strain in the patient’s leg.

7 Conclusion

Although there are certain disadvantages of using exoskeletons such as limited mobility, possible failures, the fact that they require constant maintenance and high costs, they are very helpful devices. They could be used to improve the range of movement limitation caused by several knee complications. Also, exoskeletons could reduce the number of therapists needed, and allow even the most disabled patients to regain some degree of mobility. The advantages of using them may include, increased strength, increased resistance, protection against impacts, performance of demanding jobs easily and provision of vital data for improvements. For example, the data provided by the included sensors could be used as a feedback to know the level of progress after a series of rehabilitation sessions. If this approach is combined with machine learning algorithms, the system could become adaptive, meaning that it would incrementally become specialized on a particular recovery modality or patient, outperforming other available systems that do not possess such flexible capabilities.

The ease of customization of such robotic systems may provide much more personalized and precise rehabilitation routines to recover the ROM, since they allow quantifying and studying the rehabilitation process in greater depth, especially about what the exoskeleton and the patient are doing jointly. At the same time, the fact that these devices can be upgraded through software updates or through the addition of modular components such as a wireless module, represents a huge advantage given that it would allow them to keep improving steadily and not becoming obsolete. As a result the wellbeing of patients would be improved and the overall process of rehabilitation would be eased.

References

[1] VITECKOVA, Slavka, KUTILEK, Patrik, BOISBOISSEL, Gérard De, KRUPICKA, Radim, GALAJDOVA, Alena, KAULER, Jan, LHOTSKA, Lenka and SZABO, Zoltan. Empowering lower limbs exoskeletons: state-of-the-art. *Robotica*. 2018. Vol. 36, no. 11p. 1743–1756. DOI 10.1017/s0263574718000693.

[2] CHEN, Bing, ZI, Bin, WANG, Zhengyu, QIN, Ling and LIAO, Wei-Hsin. Knee exoskeletons for gait rehabilitation and human performance augmentation: A state-of-the-art. *Mechanism and Machine Theory*. 2019. Vol. 134, p. 499–511. DOI 10.1016/j.mechmachtheory.2019.01.016.

[3] COOKE, J R, 1988. Stepper Motors: Principles and Characteristics. Proceedings of the Institution of Mechanical Engineers, Part D: Transport Engineering [online]. vol. 202, no. 2, pp. 111–117. Retrieved z: doi:10.1243/pime_proc_1988_202_163_02

[4] HALICIOGLU, Recep, L Canan DULGER and A Tolga BOZDANA, 2015. Mechanisms, classifications, and applications of servo presses: A review with comparisons. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture [online]. vol. 230, no. 7, pp. 1177–1194. Retrieved z: doi:10.1177/0954405415600013

[5] ANAM, Khairul and AL-JUMAILY, Adel Ali. Active Exoskeleton Control Systems: State of the Art. *Procedia Engineering*. 2012. Vol. 41, p. 988–994. DOI 10.1016/j.proeng.2012.07.273.

[6] SHAMAEI, Kamran, Massimo CENCIARINI, Albert A. ADAMS, Karen N. GREGORCZYK, Jeffrey M. SCHIFFMAN and Aaron M. DOLLAR, 2014. Design and Evaluation of a Quasi-Passive Knee Exoskeleton for Investigation of Motor Adaptation in Lower Extremity Joints. IEEE Transactions on Biomedical Engineering [online]. vol. 61, no. 6, pp. 1809–1821. Retrieved z: doi:10.1109/tbme.2014.2307698

[7] JUNG, Jun-Young, Wonho HEO, Hyundae YANG and Hyunsub PARK, 2015. A Neural Network-Based Gait Phase Classification Method Using Sensors Equipped on Lower Limb Exoskeleton Robots. Sensors [online]. vol. 15, no. 11, pp. 27738–27759. Retrieved z: doi:10.3390/s151127738

[8] BUTTON, Kate, IQBAL, Arshi S., LETCHFORD, Robert H. and DEURSEN, Robert W.m. Van. Clinical effectiveness of knee rehabilitation techniques and implications for a self-care treatment model. *Physiotherapy*. 2012. Vol. 98, no. 4p. 287–299. DOI 10.1016/j.physio.2011.08.003.

[9] MUTSUZAKI, Hirotaka, TAKEUCHI, Ryoko, MATAKI, Yuki and WADANO, Yasuyoshi. Target range of motion for rehabilitation after total knee arthroplasty. *Journal of Rural Medicine*. 2017. Vol. 12, no. 1p. 33–37. DOI 10.2185/jrm.2923.

[10] Use of Knee Extension Device During Rehabilitation of a Patient with Type 3 Arthrofibrosis after ACL Reconstruction. *North American Journal of Sports Physical Therapy : NAJSPT*. August 2006.

[11] LENSSEN, Anton F, CRIJNS, Yvonne Hf, WALTJÉ, Eddie Mh, ROOX, George M, STEYN, Mike Ja Van, GEESINK, Ruud Jt, BRANDT, Piet A Van Den and BIE, Rob A De. Effectiveness of prolonged use of continuous passive motion (CPM) as an adjunct to physiotherapy following total knee arthroplasty: Design of a randomised controlled trial [ISRCTN85759656]. *BMC Musculoskeletal Disorders*. 2006. Vol. 7, no. 1. DOI 10.1186/1471-2474-7-15.

[12] FORAN, Jared R.H. Total Knee Replacement Exercise Guide - OrthoInfo - AAOS. FISCHER, Stuart J. (ed.), *OrthoInfo* [online]. February 2017. [Accessed 7 January 2020]. Available from: https://orthoinfo.aaos.org/en/recovery/total-knee-replacement-exercise-guide/

[13] *Total Knee Replacement Exercise Guide* [online]. February 2017. Orthoinfo. [Accessed 7 January 2020]. Available from: https://orthoinfo.aaos.org/globalassets/figures/a00301f05.jpg

[14] *Total Knee Replacement Exercise Guide* [online]. February 2017. Orthoinfo. [Accessed 7 January 2020]. Available from: https://orthoinfo.aaos.org/globalassets/figures/a00301f06.jpg

[15] NEUMANN, Donald A., KELLY, Elisabeth Roen, KIEFER, Craig L., MARTENS, Kimberly and GROSZ, Claudia M. *Kinesiology of the musculoskeletal system: foundations for rehabilitation*. St. Louis, MO : Elsevier, 2017.

[16] LOUDON, Janice K. BIOMECHANICS AND PATHOMECHANICS OF THE PATELLOFEMORAL JOINT. *International journal of sports physical therapy* [online]. December 2016. [Accessed 8 January 2020]. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5095937/#__ffn_sectitle

[17] SCHAFER, R. C. CHAPTER 2: MECHANICAL CONCEPTS AND TERMS. In : *Clinical biomechanics: muscoskeletal actions and reactions*. Baltimore : Williams & Wilkins, 1987.

[18] PORTER, Stuart. Chapter 6: Biomechanics. In : *Tidys Physiotherapy*. Edinburgh : Churchill Livingstone, 2008. p. 152–153.

[19] Biomechanics. *Musculoskeletal Key* [online]. 7 January 2017. [Accessed 9 January 2020]. Available from: https://musculoskeletalkey.com/biomechanics-2/

[20] Experimentální systém robotického exoskeletu kostry dolních končetin ESRE-OMI. *Exoskelet - Technická podpora* [online]. [Accessed 8 January 2020]. Available from: https://rm.prokyber.cz/projects/zak-vyr/wiki/Exoskelet?fbclid=IwAR3cwWWbFVnqyxQj2PimZZCyDEIyZAbkdiytzKJiGIfruIDV04DKIKPECYo

[21] MALTHUS, COWARD, TJ, SCHOFIELD, STEVENS, COWAN, G. DANAEI, LIN, FEZEU, B. BALKAU, KENGNE, MJ, JK, AP and JCN, 1970. The weight of nations: an estimation of adult human biomass*BMC Public Health* [online] [accessed. 21. May 2020]. Retrieved z: https://bmcpublichealth.biomedcentral.com/articles/10.1186/1471-2458-12-439

[22] TÖZEREN Aydin, 1999. *Human body dynamics: classical mechanics and human movement*. New York: Springer.

[23] T. A. Le, A. G. Baydin, R. Zinkov and F. Wood, "Using synthetic data to train neural networks is model-based reasoning," 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 3514-3521, doi: 10.1109/IJCNN.2017.7966298.

[24] SHARMA, Aditya, 2017. Understanding Activation Functions in Deep Learning*Learn OpenCV* [online] [accessed. 21. May 2020]. Retrieved z: https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

Appendix A: Code for wireless actuator control testing

```
const int stepPin = 6;
const int dirPin = 5;
String initialState = "";
int label=0;
int currentAngle=0;
int lastAngle=0;
int angle=0;
int displacement=0;
String actualState = "Angle";
boolean dirRotation = HIGH;
int rotSpeed = 1500;

void setup() {
  pinMode(stepPin,OUTPUT);
  pinMode(dirPin,OUTPUT);
  Serial.begin(9600);
}
void loop() {
  delayMicroseconds(1);
  if(Serial.available() > 0){ // if data is being sent do:
    initialState = Serial.readString(); // Read data from the serial port
 }
 if (actualState == "Rise"){ //if the rise command is sent, generate a le
g cycle
  if (initialState == "Start") {//Start if button is pressed
    delay(10);
    if (dirRotation == HIGH) {
      dirRotation = LOW;
    }  //start leg rise cycle
    digitalWrite(dirPin,dirRotation);
    delay(1000);
      for(int i = 0; i <= 200; i++) {
      digitalWrite(stepPin,HIGH);
      delayMicroseconds(10000);
      digitalWrite(stepPin,LOW);
      delayMicroseconds(10000);
      }
     digitalWrite(dirPin,HIGH);
     delay(1000);
      for(int i = 0; i <= 200; i++) {
      digitalWrite(stepPin,HIGH);
      delayMicroseconds(10000);
      digitalWrite(stepPin,LOW);
      delayMicroseconds(10000);
      }

    initialState = "";
  }

  if (initialState == "Angle"){
    actualState = initialState;
  }

}

else if (actualState == "Angle"){
label = initialState.toInt();
Serial.println(angle);
Serial.println(initialState);
if (label < 0 ){
 label = 360+label;
}
currentAngle = map(label,0,359,0,200);
digitalWrite(dirPin,LOW); // move to selected angle
 if (currentAngle != lastAngle){
   if(currentAngle > lastAngle){
     displacement = (currentAngle - lastAngle);
     for(int i = 0; i < displacement; i++) {
     digitalWrite(stepPin,HIGH);
     delayMicroseconds(10000);
     digitalWrite(stepPin,LOW);
     delayMicroseconds(10000);
     }
   }
   if(currentAngle < lastAngle){
     displacement = (lastAngle - currentAngle);
     digitalWrite(dirPin,HIGH);
     for(int i = 0; i < displacement; i++) {
     digitalWrite(stepPin,HIGH);
     delayMicroseconds(10000);
     digitalWrite(stepPin,LOW);
     delayMicroseconds(10000);
     }
   }
 }
 lastAngle = currentAngle;
 if (initialState == "Rise"){
   actualState = initialState;
 }
}
```

Appendix B: Code for neural network training

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
import numpy

dataset = numpy.loadtxt("exoskeleton.csv", dtype=float, delimiter="," )
X = dataset[:,:3]
Y = dataset[:,3:]
#create neural network
model = Sequential()
model.add(Dense(3, input_dim=3, activation='relu')) # 1 input layer
model.add(Dense(20, activation='relu'))#first hidden layer
model.add(Dense(10, activation='relu'))#second hidden layer
model.add(Dropout(.2))
model.add(Dense(3, activation='softmax')) # softmax class probability

# compile the neural network, adam gradient descent
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=
['accuracy'])

# call the function to fit to the data (training the network)
model.fit(X, Y, epochs = 1000, batch_size=10)
scores= model.evaluate(X,Y)
print("\n%s: %.2f%%"% (model.metrics_names[1],scores[1]*100))
model.save('weights2.h5')
```

Appendix C: Code for continuous leg rises

```
from ModbusHandler import modbusHandler
from datetime import datetime
import time #the initial conditions are defined here:
mbClient = modbusHandler(Method = "rtu", Port = "/dev/ttyUSB0", Stopbits
= 1 , Bytesize = 8, Parity = 'E', Baudrate = 460800)
enable=1
rpm=60
run=1
st0p=0
angle=300
angle1=-300
rpm2=10
smoothExt=20
smoothFlex=-20

def angMSB(a) :
 angleArray= mbClient.floatToMod(a)
 angleMSB=angleArray[0]
 return angleMSB

def angLSB(a) :
 angleArray= mbClient.floatToMod(a)
 angleLSB=angleArray[1]
 return angleLSB

def extend() :

  mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #ena
ble
  mbClient.write_register(addresS = 40357, valuE =rpm2, uniT = 1) #rpm
  mbClient.write_register(addresS = 40358, valuE =angMSB(smoothExt), uniT
 = 1)#angleMSB
  mbClient.write_register(addresS = 40359, valuE =angLSB(smoothExt), uniT
 = 1)#angleLSB
  mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
  mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run
  time.sleep(0.0001)
  stop()
  mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #ena
ble
  mbClient.write_register(addresS = 40357, valuE =rpm, uniT = 1) #rpm
  mbClient.write_register(addresS = 40358, valuE =angMSB(angle), uniT = 1
)#angleMSB
  mbClient.write_register(addresS = 40359, valuE =angLSB(angle), uniT = 1
)#angleLSB
  mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
  mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run


def stop() :

 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #enab
le
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = st0p, uniT = 1) #run

def flex() :
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #enab
le
 mbClient.write_register(addresS = 40357, valuE =rpm2, uniT = 1) #rpm
 mbClient.write_register(addresS = 40358, valuE =angMSB(smoothFlex), uniT
 = 1)#angleMSB
 mbClient.write_register(addresS = 40359, valuE =angLSB(smoothFlex), uniT
 = 1)#angleLSB
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run
 time.sleep(0.0001)
 stop()
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #enab
le
 mbClient.write_register(addresS = 40357, valuE =rpm, uniT = 1) #rpm
 mbClient.write_register(addresS = 40358, valuE =angMSB(angle1), uniT = 1
)#angleMSB
 mbClient.write_register(addresS = 40359, valuE =angLSB(angle1), uniT = 1
)#angleLSB
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run
def smooth() :
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #enab
le
 mbClient.write_register(addresS = 40357, valuE =rpm2, uniT = 1) #rpm
 mbClient.write_register(addresS = 40358, valuE =angMSB(smoothFlex), uniT
 = 1)#angleMSB
 mbClient.write_register(addresS = 40359, valuE =angLSB(smoothFlex), uniT
 = 1)#angleLSB
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run
while(1):

  result = mbClient.read_holding_registers(addresS =40350, counT = 2, uni
T = 1)
  position= mbClient.modToFloat(result.registers[0],result.registers[1])
  result = mbClient.read_discrete_inputs(addresS =10001, counT = 10, uniT
 = 1)
  print (position)
```

```
print(result.bits)
button1=result.bits[0]
button2=result.bits[3]
if(position <= 120):

    if button1 ==1:
     print("extend")
     extend()

    if button2==1:
     print("flex")
     flex()

if(position >=120):
    time.sleep(0.01)
    flex()
```

Appendix D: Code for leg rises with an ANN

```
from ModbusHandler import modbusHandler
from datetime import datetime
import time
from keras.models import Sequential
from keras.layers import Dense, Dropout
import numpy

# The neural network is initialized
model = Sequential()
model.add(Dense(3, input_dim=3, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='relu'))

model.add(Dropout(.2))
model.add(Dense(3, activation='softmax'))

model.load_weights('weights2.h5')
# communication is stablished
mbClient = modbusHandler(Method = "rtu", Port = "/dev/ttyUSB0", Stopbits
= 1 , Bytesize = 8, Parity = 'E', Baudrate = 460800)
gearRatio=11
enable=1
rpm=20
run=1
st0p=0
angle=20*gearRatio
angle1=-20*gearRatio
angleReturn=-120*gearRatio

def neural_Network(a,b,c):
 array1=numpy.array([[a,b,c]])
 predictions = model.predict(array1)
 # output our model's predictions.
 return predictions
def angMSB(a) :
 angleArray= mbClient.floatToMod(a)
 angleMSB=angleArray[0]
 return angleMSB
def angLSB(a) :
 angleArray= mbClient.floatToMod(a)
 angleLSB=angleArray[1]
 return angleLSB
```

```
def up() :#this function moves the leg up
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #enab
le
 mbClient.write_register(addresS = 40357, valuE =rpm, uniT = 1) #rpm
 mbClient.write_register(addresS = 40358, valuE =angMSB(angle), uniT = 1)
#angleMSB
 mbClient.write_register(addresS = 40359, valuE =angLSB(angle), uniT = 1)
#angleLSB
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run

def stop() :#this function stops the leg

 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #enab
le
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = st0p, uniT = 1) #run

def down() : #this function moves the leg down

 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #enab
le
 mbClient.write_register(addresS = 40357, valuE =rpm, uniT = 1) #rpm
 mbClient.write_register(addresS = 40358, valuE =angMSB(angle1), uniT = 1
)#angleMSB
 mbClient.write_register(addresS = 40359, valuE =angLSB(angle1), uniT = 1
)#angleLSB
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run

def Return() :#this function moves the to origin

 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1) #enab
le
 mbClient.write_register(addresS = 40357, valuE =rpm, uniT = 1) #rpm
 mbClient.write_register(addresS = 40358, valuE =angMSB(angleReturn), uni
T = 1)#angleMSB
 mbClient.write_register(addresS = 40359, valuE =angLSB(angleReturn), uni
T = 1)#angleLSB
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run

while(1):

    result = mbClient.read_holding_registers(addresS =40350, counT = 2, u
niT = 1)
    deg=mbClient.modToFloat(result.registers[0],result.registers[1])
    time.sleep(1)
    result2 = mbClient.read_holding_registers(addresS =40350, counT = 2, 
uniT = 1)
    deg2=mbClient.modToFloat(result2.registers[0],result2.registers[1])
    teta=deg2-deg
    print(deg)
    result = mbClient.read_discrete_inputs(addresS =10001, counT = 10, un
iT = 1)
    button1=result.bits[0]
    button2=result.bits[3]
    out_array=[result.bits[0],result.bits[3],teta]
    print(out_array)
    predicted=neural_Network(button1,button2,teta)
    print(predicted)
    action=numpy.argmax(predicted)
    print(action)
    if(action==0):
      print('down')
      down()
      time.sleep(2)

    if(action==1):
      print('up')
      up()
      time.sleep(2)

    if(action==2):
      print('stop')
      stop()
# code for basic leg rises
# while(1):
#  result = mbClient.read_holding_registers(addresS =40350, counT = 2, un
iT = 1)
#  initial= mbClient.modToFloat(result.registers[0],result.registers[1])
#  TRH=initial+15)
#  result1 = mbClient.read_holding_registers(addresS =40350, counT = 2, u
niT = 1)
#  actual= mbClient.modToFloat(result1.registers[0],result1.registers[1])
#  print(actual)
#  if(actual>=120):
#       print('down')
#       time.sleep(5)
#       Return()
#       time.sleep(2)
#  if(actual<120):
#      if(actual>=TRH):
#       print('up')
#       up()
#       time.sleep(1)
```

Appendix E: Code for standing up and walking

```
from ModbusHandler import modbusHandler
from datetime import datetime
import time #the initial conditions are defined here:
mbClient = modbusHandler(Method = "rtu", Port = "/dev/ttyUSB0", Stopbits
= 1 , Bytesize = 8, Parity = 'E', Baudrate = 460800)
gearRatio=11
enable=1
rpm=10
run=1
st0p=0
angle=5*gearRatio
angle1=-5*gearRatio
rpm2=3
smoothExt=10
smoothFlex=-10
def angMSB(a) :
 angleArray= mbClient.floatToMod(a)
 angleMSB=angleArray[0]
 return angleMSB
def angLSB(a) :
 angleArray= mbClient.floatToMod(a)
 angleLSB=angleArray[1]
 return angleLSB
def stop() :
 mbClient.write_register(addresS = 40356, valuE = enable, uniT= 1)#enable
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = st0p, uniT = 1) #run
def stand() :
 mbClient.write_register(addresS = 40356, valuE = enable, uniT= 1)#enable
 mbClient.write_register(addresS = 40357, valuE =rpm, uniT = 1) #rpm
 mbClient.write_register(addresS = 40358, valuE =angMSB(120), uniT = 1)
#angleMSB
 mbClient.write_register(addresS = 40359, valuE =angLSB(120), uniT = 1)
#angleLSB
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)#mode
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1) #run

def walk() :

 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1)
 mbClient.write_register(addresS = 40357, valuE =rpm2, uniT = 1)
 mbClient.write_register(addresS =40358,valuE =angMSB(smoothFlex),uniT=1)
 mbClient.write_register(addresS = 40359,valuE=angLSB(smoothFlex),uniT=1)
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1)
 time.sleep(0.0001)
 stop()
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1)
 mbClient.write_register(addresS = 40357, valuE =rpm, uniT = 1)
 mbClient.write_register(addresS = 40358, valuE =angMSB(angle1),uniT = 1)
 mbClient.write_register(addresS = 40359, valuE =angLSB(angle1),uniT = 1)
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1)
 time.sleep(0.0001)
 stop()
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1)
 mbClient.write_register(addresS = 40357, valuE =rpm2, uniT = 1)
 mbClient.write_register(addresS = 40358, valuE=angMSB(smoothExt),uniT=1)
 mbClient.write_register(addresS = 40359, valuE=angLSB(smoothExt),uniT=1)
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1)
 time.sleep(0.0001)
  stop()
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1)
 mbClient.write_register(addresS = 40357, valuE =rpm, uniT = 1)
 mbClient.write_register(addresS = 40358, valuE =angMSB(angle), uniT = 1)
 mbClient.write_register(addresS = 40359, valuE =angLSB(angle), uniT = 1)
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1)
def plusreturn() :
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1)
 mbClient.write_register(addresS = 40357, valuE =rpm2, uniT = 1)
 mbClient.write_register(addresS = 40358, valuE=angMSB(angle),uniT=1)
 mbClient.write_register(addresS = 40359, valuE=angLSB(angle),uniT=1)
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1)
def minusreturn() :
 mbClient.write_register(addresS = 40356, valuE = enable, uniT = 1)
 mbClient.write_register(addresS = 40357, valuE =rpm2, uniT = 1)
 mbClient.write_register(addresS = 40358,valuE=angMSB(smoothFlex),uniT=1)
 mbClient.write_register(addresS = 40359,valuE=angLSB(smoothFlex),uniT=1)
 mbClient.write_register(addresS = 40360, valuE = 2, uniT = 1)
 mbClient.write_register(addresS = 40361, valuE = run, uniT = 1)
# sitting state
while(1):
 result = mbClient.read_discrete_inputs(addresS =10001, counT =10,uniT=1)
 if(result.bits[0]==1)
   stand()
   time.sleep(3)
   # standing state

   while(1):

    result = mbClient.read_holding_registers(addresS =40350, counT = 2, u
niT = 1)
```

```
    position= mbClient.modToFloat(result.registers[0],result.registers[1]
)
    result = mbClient.read_discrete_inputs(addresS =10001, counT = 10, un
iT = 1)
    print (position)
    print(result.bits)
    button1=result.bits[0]

    if(position < 120 and position > 60):
      # walking state
      if button1 ==1:
       print("walk")
       walk()
      # back to standing state
    if(position >= 120):
      # error
      print("out of range")
      time.sleep(0.01)
      # correction
      minusreturn()
      # back to walking state
    if(position<= 60):
      # error
      print("out of range")
      time.sleep(0.01)
      # correction
      plusreturn()
      # back to walking state
```

Appendix F:Content of the enclosed CD

- Key words
- Abstract in Czech
- Abstract in English
- Scan of the assignment of the topic of the diploma thesis
- The complete diploma thesis
- Datasheet of used components
- Codes
- Data from calculations