TECHNOLOGIES FOR SENSORS AND CLINICAL INSTRUMENTATION
Project report
Academic year 2022-2023

A. ALIVERTI, C. PAGANELLI
A. ANGELUCCI

Electrogoniometer for gait analysis
Potentiometer

Team 4:
BOUQUILLON Mylène, 10905159
DESIDE Guillaume, 10905967
MATERNE Sophie, 10872639
MEDRANO Estanislao, 10916969
PESKIR Aleksa, 10877580
SCHNIRER Tobias, 10912716

School of Industrial and Information Engineering
Biomedical Engineering - Master of Science

Table of contents

List of Tables

List of Figures

List of Codes

1 Introduction

1.1 Context analysis

Over the past century, a large amount of studies have focused on gait analysis to gain a comprehensive understanding of human walking, as it is the primary mean of human locomotion. To do so, the traditional method consists in measuring the angular variations of the body joints set in motion, in particular those of the lower limbs (the hip, knee and ankle) [1, 2, 4, 6, 8].

Continuous knee motion monitoring has become essential in various medical applications as it can provide in-depth insights into the functional status of patients. For example, it can assist clinical practitioners in assessing joint and muscular limitations, identifying individual therapeutic needs and developing accurate tailored rehabilitation plans after injury, optimizing athlete performances as well as providing feedback information for prosthetic control systems.

Amongst the various sensor types for continuous knee flexion and extension monitoring, potentiometers have been identified as promising devices with strong potential for measuring sagittal knee range of motion during daily activities (e.g. walking, jogging and stairs climbing). Indeed, as those are available in small sizes (about a few centimeters in diameter) and can withstand several million movement cycles before showing any signs of wear and tear, they meet most of the technological and physiological challenges encountered during the development of a gait analysis sensor. By challenges, one means that the sensor system should be easy to use, neither interfere with clothing nor restrict the patient's performances (e.g. use of clothing embedded-sensors).

This assignment aims to design a potentiometer-based electrogoniometer with an Arduino microcontroller to perform a gait analysis through real-time measurements of the knee angle. To achieve this, the potentiometer and the Arduino board are embedded into a knee sleeve thanks to two 3D-printed arms, one on each limb segment and both attached to the outer lateral side of the knee. The fixed axis of rotation of the potentiometer is aligned with the axis of rotation of the knee. A video of a demonstration showing the functionalities of both the software and the hardware of the implemented solution is available on the following One Drive. It also contains the 3D structure files as well as the `Arduino`, `Postprocessing` and `Python` codes, which are further explained in this report.

1.2 Phases of the walk

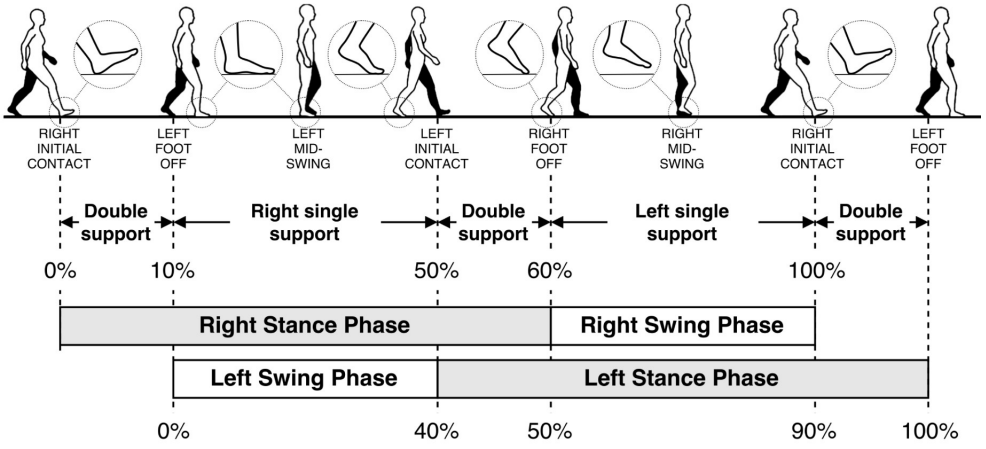
Gait analysis can be defined as the study of human locomotion by isolating the shortest repeatable task during gait in order to analyse and quantify how a person walks. This walking-unit being the time between one foot strike and the subsequent foot strike of the same foot.

The gait cycle (stride) can be broken down into two primary phases, the stance and the swing phases, which alternate for each lower limb. Those being themselves composed of different sub-phases as illustrated in Fig. 1a and 1b.

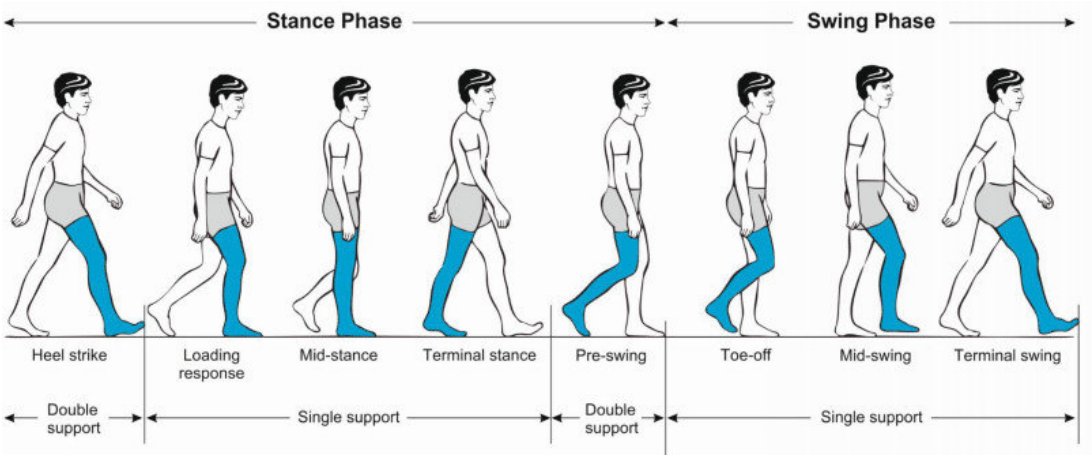
1.3 Knee joint motion

The following knee angle ranges describe the knee dynamics while walking on a flat surface:

- Initial contact: flexion up to about 5°.
- Load response: flexion up to about 15°.
- Mid-stance: flexion up to about 25°/30° (local maximum knee flexion) and start of extension.
- Terminal stance: extension up to about 5° (local maximum knee extension).
- Pre swing: rapid flexion up to about 35/40°.
- Initial swing: rapid flexion up to about 60° (global maximum knee flexion).
- Mid-swing: extension up to about 30°.
- Terminal swing: full extension (global maximum knee extension) and very small flexion before initial contact.



(a)



(b)

Fig. 1: Phases of the walk (a) [10] (b) [7].

1.4 Problem facing

As previously mentioned, the goal of this assignment is to develop a potentiometer-based electrogoniometer with an Arduino microcontroller board to measure the evolution of the knee angle. During its creation, a couple of issues have been encountered and solutions had to be found to overcome them. These are listed below:

- Wireless data sharing between the Arduino and the external device: to effectively monitor the knee angle evolution during the gait, real-time data transmission between the software and the hardware is highly recommended. Thus, the first idea was to send the data collected by the hardware to the software through a Wi-Fi connection. However, this idea had to be given up as it turned out that the chosen Wi-Fi module (ESP8266 01S esp-01S Wlan WiFi module with breadboard adapter) is not compatible with an Arduino micro and no compatible Wi-Fi module is currently available on the market at a reasonable price. Therefore, a Bluetooth module (HC-05 HC-06 Bluetooth Wireless RF-Transceiver module RS232 Serial TTL) has been used instead.
- Printed Circuit Board (PCB): due to the long delivery times, the PCB of the goniometer could not be ordered and then received in time for the deadline of this project. Thus, it has been designed (subsection 2.2.5) but not manufactured and the current sensor still contains some soldered cables.
- Phases of the walk: the first step in data analysis consists in identifying the phases of the walk in order to determine if the gait of the subject is impaired or not. Theoretically speaking, the knee joint motion can be divided into 8 stages according to the phases of the walk (Fig. 1a and 1b, subsection 1.3). However, for the sake of simplicity, only 5 phases will be considered while performing the gait analysis. Those being the different succession of flexion-extension periods. Remark that this assumption is not oversimplifying the gait analysis and still provides deep insights about the health state of the subject's walk.

2 Hardware

The hardware system is composed of a 3D structure and a few electronic components inserted into a knee brace.

2.1 3D structure

The main structure has been designed using the `SolidWorks` software. As shown in Fig. 2, it is composed of 2 main arms, both with a hole in the middle of one extremity for the insertion of the potentiometer. The latter being used for the measurement of the knee angle, so for the evaluation of the gait analysis. One arm of the structure [the upper one in Fig. 2] is aimed to fit on the upper part of the knee joint while the second one [the lower one in Fig. 2] fits the lower part. Using this design, it is actually possible to create the mechanical joint required to measure the movement of the leg at the knee position.

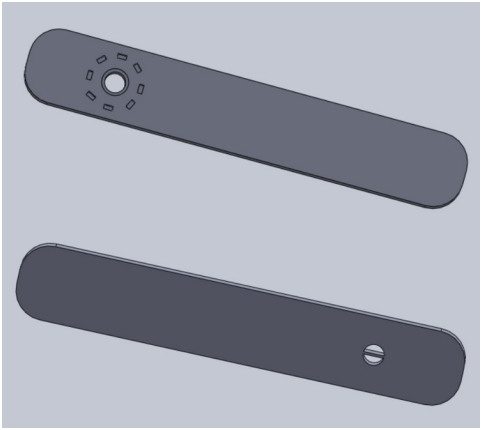


Fig. 2: Braces of the 3D structure.

The whole structure has been printed using a 3D printer and filled with a 50% density of plastic material, thereby allowing it to support the applied forces without presenting any signs of deformation or even breaking.

2.2 Electronics

2.2.1 List of components

The electronic components of the developed potentiometer-based goniometer are the following:

- Arduino micro: a microcontroller board based on the ATmega32U4.
- 10 kΩ angular potentiometer.
- Portable battery: the 5V power supply.
- HC-05 HC-06 Bluetooth Wireless RF-Transceiver module RS232 Serial TTL.

2.2.2 Arduino micro

Using its analog input pins, the Arduino micro (Fig. 3) is able to capture the changing resistance values from the potentiometer and then convert them into meaningful angle readings. The Arduino micro, with its compact size, robust performance and extensive community support, is highly suitable for the development of an electrogoniometer for gait analysis.

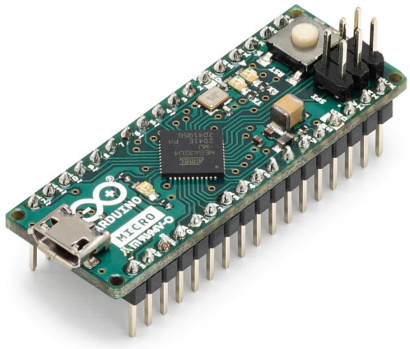


Fig. 3: Arduino micro [9].

2.2.3 Potentiometer

The angular potentiometer (Fig. 4) can be defined as a resistive position sensor that functions as an adjustable voltage divider and whose resistance varies linearly with the angular movement (so the angle θ) imposed by the system. Its connection with the Arduino board is mentionned in Tab. 1.

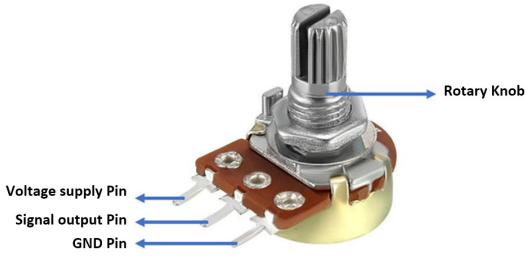


Fig. 4: Potentiometer [3].

Arduino	Potentiometer
A3	Signal output Pin
5V	Voltage supply Pin
GND	GND Pin

Tab. 1: Arduino - Potentiometer connections.

2.2.4 Wireless communication

The Bluetooth HC-05 module (Fig. 5), which is integrated into the wireless capabilities of the Arduino micro, improves the functioning of the developed goniometer. Indeed, it allows the wireless sharing of data in real-time between the hardware and the software, thereby improving the flexibility and ease of use of the sensor. The software could be any type of external devices, such as a computer, tablet or smartphone.

As displayed in Tab. 2, the Arduino board and the Bluetooth module are coupled through the Arduino serial pins TX and RX as well as the VCC and ground connections. The Bluetooth link with the device of interest can be established by proper configuration using the AT commands.

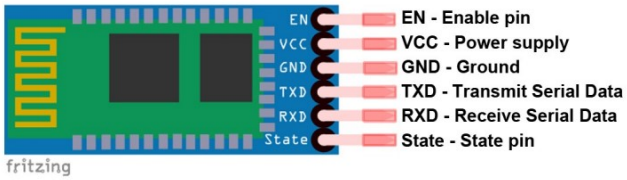


Fig. 5: Bluetooth module.

Arduino	Bluetooth
10	TX (= TXD)
11	RX (= RXD)
5V	VCC
GND	GND

Tab. 2: Arduino - Bluetooth connections.

2.2.5 Printed Circuit Board

A PCB (Fig. 7) has been designed for the purpose of providing electrical connections and mechanical support to the electrical components of the goniometer's circuit. Its schematic is displayed in Fig. 6.

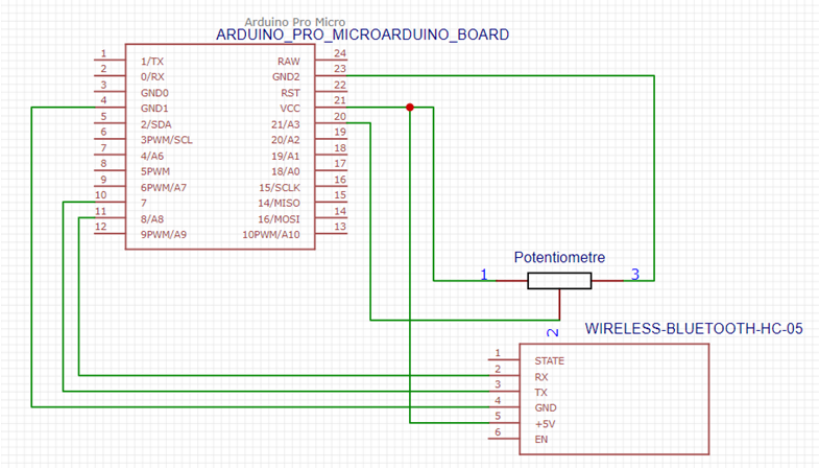


Fig. 6: PCB schematic of the sensor.

There are several advantages in the use of a PCB:

1. Space reduction: The circuit on a PCB can take up much less space than the individual components themselves, resulting in a condensed end-product. This is notable especially when designing portable devices as it is the case in the present project.

2. Connections: As the connections of the circuit are directly printed on the board, the use of cables is not necessary anymore, thereby removing most of the unwanted interferences and noises.

3. Encapsulation: PCBs make it almost impossible to touch 2 contacts at once with bare skin. This eliminates the chances of getting an electric shock from the device itself.

4. Efficiency and cost: PCBs allow for devices to be manufactured with fewer components, which in turn, tends to drastically reduce the costs.

5. Recycling: PCBs are easy to repair and recycle. If a specific component on the board fails, it is usually much easier to fix than if it was located elsewhere in the circuit. Moreover, when the board finally reaches the end of its lifespan, it can be recycled without any negative environmental effects.

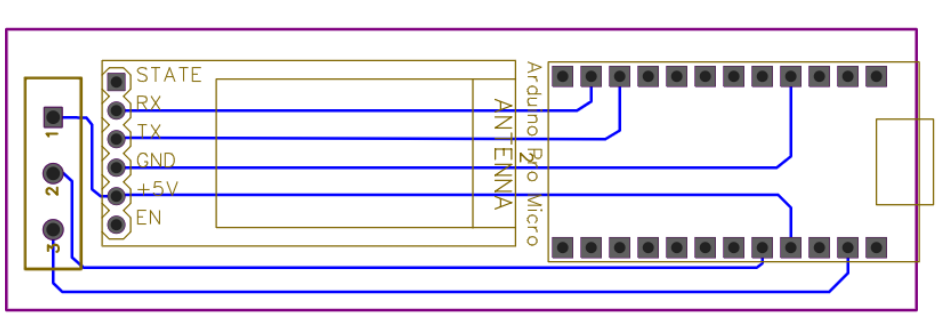


Fig. 7: PCB of the sensor.

2.2.6 Fritzing schematic of the assembly

The Frtizing schematic of the potentiometer-based goniometer is displayed in Fig. 8. It represents all the connections amongst the different electronic components and how they are coupled with the 3D structure.

2.3 Housing

The 3D structure and the electronic components are all embedded into a knee sleeve shown in Fig. 9. This type of housing makes the sensor easy to wear and thereby significantly simplifies the procedure of gait analysis. Remark that the accuracy of the results is slightly greater when there is no tissue folds between the knee brace and the knee of the subject, meaning that shorts/skirts/dresses (or even skinny pants) are more recommended than large pants to perform a gait analysis with this goniometer. Moreover, as the electronics can be removed from the knee sleeve without altering the sensor's functioning, one can insert them into another one to suit better the lower limb size of the subject undergoing the gait analysis.

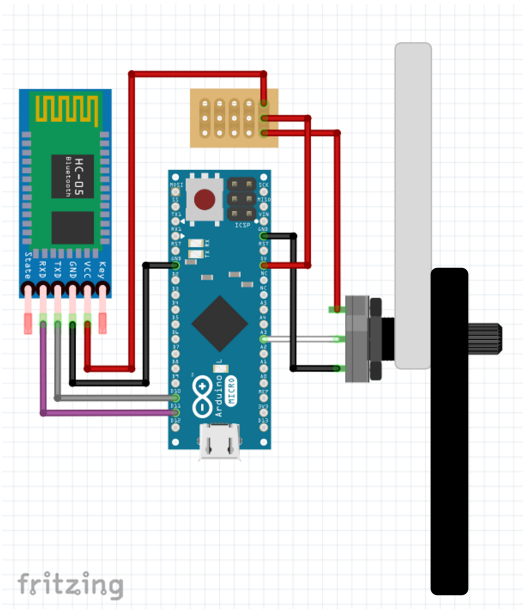


Fig. 8: Fritzing schematic of the assembly.

Fig. 9: Knee brace [5].

2.4 Final assembly

The final assembly of the goniometer as well as a couple of its zooms are respectively displayed in Fig. 10 and in Fig. 11, 12 and 13.

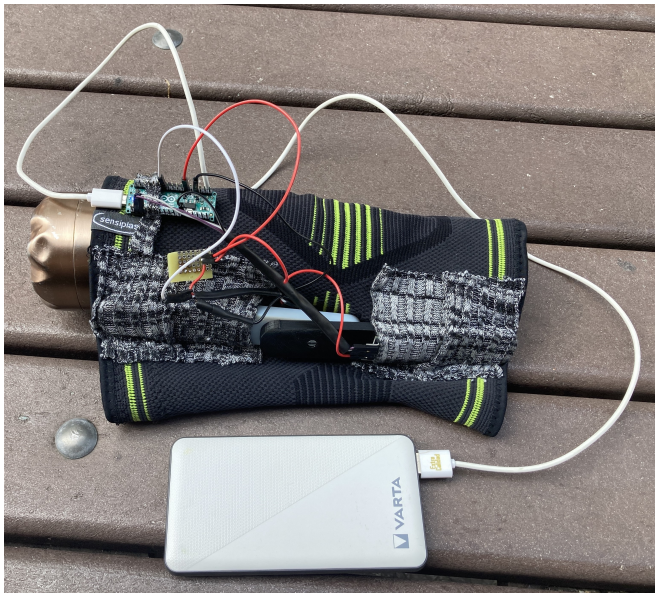


Fig. 10: Final assembly.

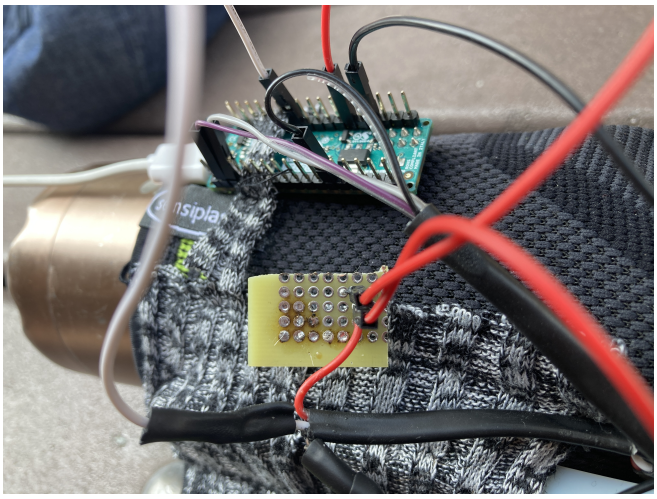


Fig. 11: Zoom on the connections on the final assembly.

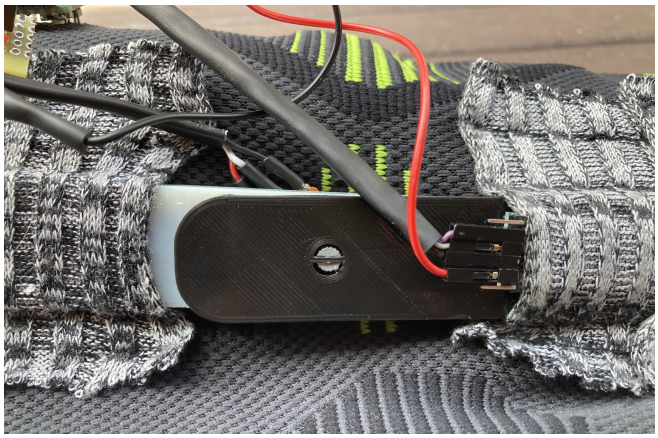


Fig. 12: Zoom on the knee joint on the final assembly.

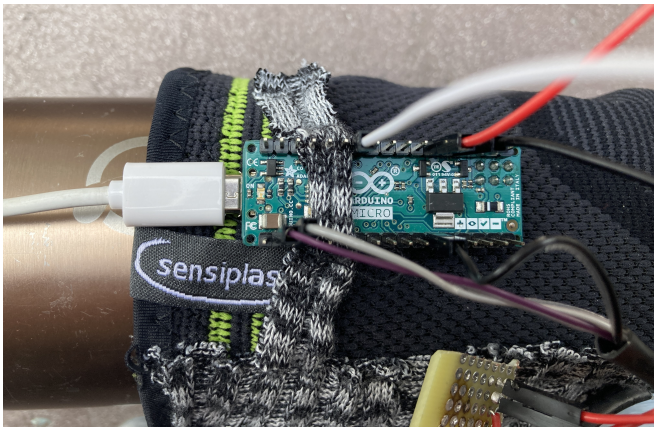


Fig. 13: Zoom on the Arduino micro on the final assembly.

3 Firmware

To manage the hardware and enable its communication with the software, firmware is required. Its main function consists in acquiring the analog output voltage of the potentiometer, converting it to a digital form, and sending the collected information to the software. The latter being in charge of additional processing and visualisation. More precisely, the firmware reads the angle value provided by the potentiometer under the form of voltage and sends it to the `Processing` code.

3.1 Firmware implementation

The firmware tasks have been implemented through the `Arduino` framework, which includes the necessary libraries, such as "SoftwareSerial" and "TimerOne", that enable the communication and timing functionalities. The code is reported in Listing 1 and its most relevant instructions are explained below. One should keep in mind that this code can also be used in other applications, such as volume control in multimedia devices.

```
#include <SoftwareSerial.h>
#include <TimerOne.h>

const int analogPin = A3;
SoftwareSerial bluetooth(10, 11);

const int numReadings = 20;
int readings[numReadings];  // the readings from the analog input
int total = 0;              // the running total
int average = 0;            // the average

float xn1 = 0.0;
float yn1 = 0.0;
float xn2 =0.0;

const int samplingFrequency = 30; // Sampling frequency in Hz
const long samplingInterval = 1000000 / samplingFrequency; // Sampling interval in
    microseconds

void setup() {
  pinMode(analogPin, INPUT); // Set analog pin as input
  Serial.begin(9600); // Initialize serial communication

  Timer1.initialize(samplingInterval); Timer1.attachInterrupt(readPotentiometer); //Timer
    initialization
  bluetooth.begin(9600); // Initialize Bluetooth communication
  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  }

}
void loop() {}

void readPotentiometer() {
  total = 0.0;
  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = analogRead(analogPin);
    total = total + readings[thisReading];
  }

   average = total / numReadings;

   float xn = average;
   float yn = -0.02305471*yn1  + 0.51152736*xn +  0.51152736*xn1 + 0.0*xn2;
   xn2 = xn1;  xn1 = xn; yn1 = yn;

  float angle = 0.41665476*yn-30.275; //from calibration
   bluetooth.println(angle);
}
```

Listing 1: `Arduino` code - main.ino.

- *Line 1* imports the "SoftwareSerial" library whose function is to allow the serial communication on other digital pins of the Arduino board, using software to replicate the functionality.
- *Line 2* imports the "TimerOne" library that enables the creation of a periodic interrupt (i.e. at a specific frequency).
- *Line 4* defines the "analogPin" constant (const int makes the "analogPin" read-only).
- *Line 5* initialises the Bluetooth connection.
- *Lines 7 and 14* initialise the variables used to filter the signal send by Bluetooth.
- *Lines 16 and 17* respectively set the angle values sampling frequency to 30 [Hz] and the time sampling interval to the inverse of the sampling frequency in [ms].
- *Lines 19 to 29* concern the "setup()" function that is called first when the sketch starts and runs only once. It sets the analog pin as an input and the data rate at 9600 [baud] (i.e. bits per second) for the serial communication (i.e. serial data transmission). Then, it initializes the "Timer1" with the value of the previously defined time sampling interval and attaches an interrupt to the "readPotentiometer" function further defined. It also initialises the Bluetooth communication to a data rate of 9600 [baud].
- *Lines 32 to 37* create the "readPotentiometer" function which is called each time an interrupt is triggered in order to read the analog value provided by the potentiometer itself.
- *Lines 41 to 43* apply a low-pass filter on the data in order to remove the frequencies above 10 [Hz] (Nyquist frequency of 15 [Hz]), thereby removing the interference.
- *Line 45* allows to map the analog value provided by the potentiometer into an angle value through the 0-1023 / 0-180 conversion. The conversion formula specific to the goniometer has been found by means of a calibration step (subsection 5.1).
- *Line 46* prints the measured angle on the serial monitor as well as on the Bluetooth module every sampling interval.

3.2 Firmware uploading

To upload the code described in Listening 1, the Arduino board needs to be connected to the external device through a USB cable. Then, the following different steps have to be performed:

- Open the Arduino IDE and select the correct board and port under the "Tools" menu.
- Copy and paste the firmware code into the Arduino IDE.
- Click on the "Upload" button to compile and upload the firmware into the Arduino board.
- Monitor the serial output in the Arduino IDE's serial monitor to ensure everything is correctly working.

3.3 Bluetooth communication

Finally, the last step consists in pairing the Arduino with the Bluetooth module and assuring their communication. This Bluetooth connection allows to send and receive data without any wires between the Arduino and the external device. To do so, the following steps have to be executed:

- Ensure that the Bluetooth module is powered on and in pairing mode.
- Enable "Bluetooth" on the external device and search for available devices.
- Locate and pair with the Bluetooth module from the list of available devices.
- Once paired, establish a serial communication connection with the Bluetooth module using a Bluetooth terminal app on the external device.

4 Software

To capture the data acquired by the potentiometer-based goniometer in real-time and display them in a user-friendly graphical interface, software is required. In the present case, this graphic interface, which has been coded in `Processing` language (Listing 5), displays the instantaneous knee angle and speed values as well as the time evolution of the knee angle (Fig. 14). To get these measurements, data acquisition (Listing 2) and processing (Listing 3) are first necessary. Remark that these are gathered in one unique code but, for a better understanding, they will be displayed into different sections and individually explained.

4.1 Data acquisition

The data acquisition section (Listing 2) of the `Processing` code consists in reading and storing data received via Bluetooth on the "COM14" serial port for further analysis. Regarding the code itself, it includes the 2 variables "waitForStart" and "Finish", which are used to wait for user input before proceeding with data acquisition. The user is expected to press the "S" (or "s") key to initiate data acquisition which stops when the "E" (or "e") key is pressed. Once the condition of receiving the appropriate user input is met, the code proceeds to read data from the serial port. A file named "test_post_analysis.txt" is created to store the acquired data.

```
void setup() {
  size(1900,1600);
  background(255);
  port = new Serial (this, "COM14", 9600);
  String filename = "test_post_analysis.txt";
  output = createWriter(filename);

   .... (hidden part of the code)
}

void draw() {
  .... (hidden part of the code)

  if (!waitForStart && !Finish){
    myString = port.readStringUntil(lf);
   if (myString != null) {
      angle_knee= float(myString);
      time += samplingTime;
  }
  .... (hidden part of the code)
  }
}
```

Listing 2: `Processing` code - data acquisition.

4.2 Data processing

The data processing section (Listing 3) of the `Processing` code calculates the speed and acceleration based on the previously acquired knee angle data. This enables to assign a given phase of the walk (subsection 1.4) to the gait currently being analysed. The algorithm behind this step is displayed in Algorithm 1 and explained just after.

```
void calculateSpeed(){
  if (angleList.size() > 1) {
    float currentAngle = angleList.get(angleList.size() - 1);
    float previousAngle = angleList.get(angleList.size() - 2);
    previous_angle_speed = angle_speed;
    angle_speed = (currentAngle - previousAngle);
  } else {
 previous_angle_speed = 0.0;
    angle_speed = 0.0;
  }
}
void calculateAcceleration() {
  if (angleList.size() > 2) {
```

```
14      float currentSpeed = angle_speed;
        float previousSpeed = previous_angle_speed;
16      previous_angle_acceleration = angle_acceleration;
        angle_acceleration = (currentSpeed - previousSpeed);
18    } else {
        previous_angle_acceleration =0.0;
20      angle_acceleration = 0.0;
    }
22  }
```

Listing 3: `Processing` code - data processing.

Algorithm 1 The ideal algorithm to assign the current phase of walk

```
current_state ← "phase 1"
if |current_angle - previous_angle| > 10 then
    print("out of range")
end if
switch current_phase do
    case "phase 1"
        if current_speed >= 0 then
            current_state ← "phase 1"
        else
            current_state ← "phase 2"
        end if
    case "phase 2"
        if current_speed <= 0 then
            current_state ← "phase 2"
        else
            current_state ← "phase 3"
        end if
    case "phase 3"
        if current_acc >= 0 then
            current_state ← "phase 3"
        else
            current_state ← "phase 4"
        end if
    case "phase 4"
        if current_speed >= 0 then
            current_state ← "phase 4"
        else
            current_state ← "phase 5"
        end if
    case "phase 5"
        if current_speed <= 0 then
            current_state ← "phase 5"
        else
            current_state ← "phase 1"
        end if
```

The current walk phase is determined by 2 (sometimes 3) factors: the phase of the previous time instant as well as the sign of angle speed and/or acceleration. To do so, the following steps have to be performed:

- The algorithm starts by checking if the difference between the current angle and the previous angle is greater than 10 [°] to ensure that the angle is within a reasonable acceptable range. If it is, the algorithm prints "out of range" and analyses the next new knee angle. If it is not, the algorithm then checks the current phase which can be either 1, 2, 3, 4 or 5.
- For each current phase, the algorithm checks the sign of the current speed and acceleration.
- The algorithm repeats this process until the current phase is 5 (the last one - end of the gait cycle). At that point, the algorithm starts over again at phase 1 (the first one - start of the gait cycle).

4.3 Real-time visualisation

The real-time graphical representation of the knee angle data provided by the software is displayed in Fig. 14. It consists of

- a continuous line graph showing the time evolution of the knee angle (on the right) coloured depending on in which phase the subject is (legend on the middle left), as well as
- a dynamical update of the numerical values of the knee angle and speed (on the upper and lower left).

This multi-parameters representation provides an immediate accurate feedback to the user, thereby allowing them to instantaneously notice when abnormal knee angle and speed values occur.

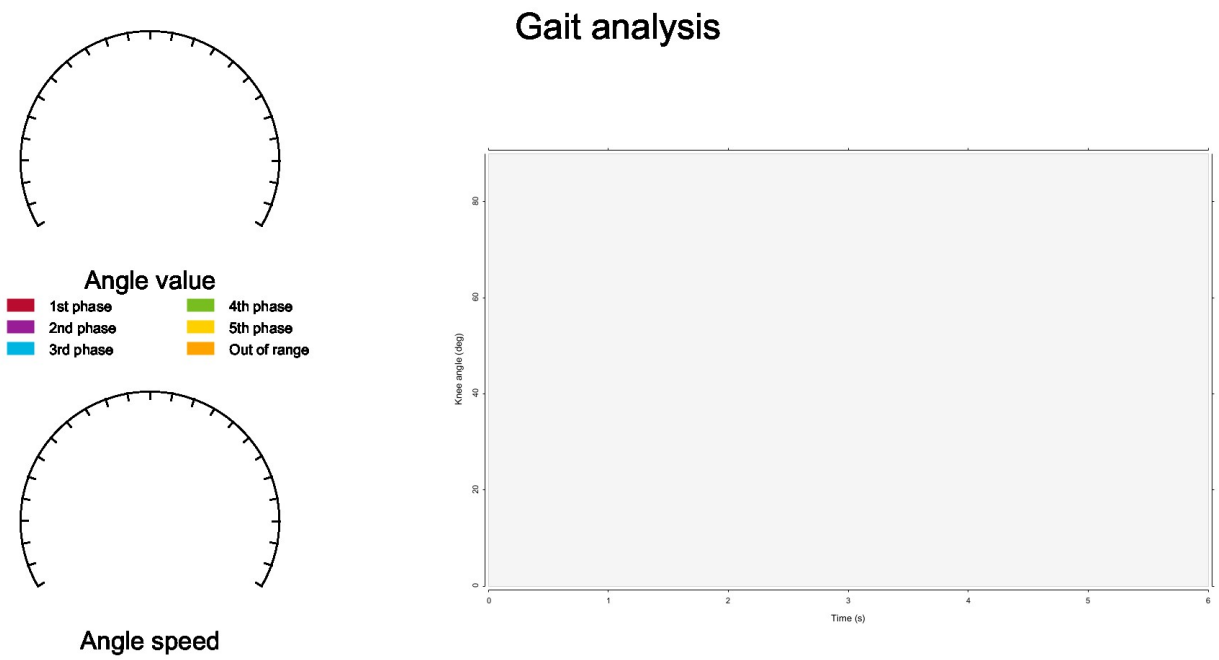


Fig. 14: Graphic interface of the system (without any results).

4.4 User interaction

The user interface has been created with the aim of being as simple and intuitive as possible for the user to interact with it. One way to achieve it consists in coding a few keyboard and mouse shortcuts (Listing 4) such as:

- Key "S" (or "s") resets the graph to the starting instant and sets the current phase to the first phase.
- Key "E" (or "e") pauses the running program.
- Key "C" (or "c") pauses the running program and clears the measurement data.
- Key "X" (or "x") exits the running program.
- A left mouse click sets the current phase to the first phase.

```
void keyPressed() {
  if (key == 'c' || key == 'C') {
    background(255);
    time = 0.0;
    waitForStart = true;
    Finish = false;
  }
  if (key == 's' || key == 'S') {
    waitForStart = false;
    time = 0.0;
    current_phase = 0;
  }
  if (key == 'e' || key == 'E') {
    Finish = true;
  }
  if(key == 'x' || key == 'X'){
     output.flush();
```

```
18      output.close();
        exit();
20    }
  }
22 void mousePressed() {
  output.println("new");
24 current_phase = 0;
  }
```

Listing 4: `Processing` code - user interaction controls.

5 Results

The main goal of this last section consists in comparing the theory with the practice, the theory being the presumed division and succession of the gait phases, based on the established literature and biomechanical models. On the other hand, practice concerns the results obtained in the field using the created potentiometer-based goniometer. As previously mentioned, this sensor has been designed with the purpose of measuring the knee joint angle during the gait cycle. Moreover, in the present case, the focus will be on the protocol of "walking on a flat surface".

5.1 Calibration

To calibrate the sensor, a linear regression has been performed by means of the least-squares criterion (independent linearity) as shown in the appendix (Listing 6). This calibration is subject specific. Indeed, to do so, the subject of interest was wearing the goniometer on its right knee and standing up. Then, by moving its right leg while keeping the rest of its body immobile, the 0 [°] and 90 [°] have been measured with a universal goniometer at the same time as the digital values of the potentiometer measured by the `Arduino` micro. Based on these values, the coefficients of the calibration line were calculated. To verify the validity of these coefficients, other angles have been checked.

5.2 Real-time analysis

Fig. 15 corresponds to a screenshot of the graphical user interface (GUI) displayed on the external device during a gait cycle analysis. The latter being performed on a given subject wearing the current potentiometer-based goniometer while walking on a flat surface. As mentioned before (subsection 4.3), the instantaneous knee angle and speed are printed on the left while the time evolution of the knee angle appears on the right. By eye-balling these results, one can immediately notice that the 5 phases succeed to each other in a correct order without omitting any of them. Regarding the out-of-range values, the subject did not produce any of them at that moment. However, as shown in the video of the demonstration joined to this report, those tend to appear when the subject starts to run. This can simply be explained by the criteria of the "10 [°]" used to discriminate the recorded data in real time. Nevertheless, even during the run, the sensor still provides a gait cycle curve with a meaningful trend, meaning that this system can be used for a global first analysis in that particular case.

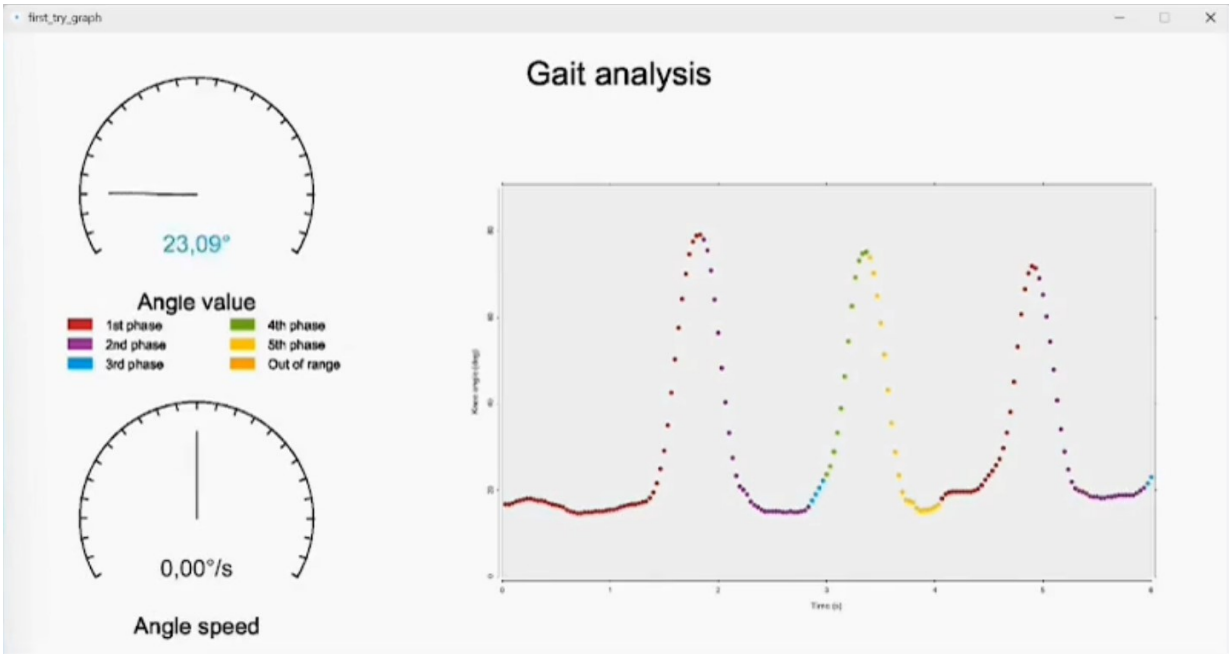


Fig. 15: Graphic interface of the system (with results).

5.3 Post-analysis

Fig. 16, obtained with the `Python` code reported in Listing 7, displays the comparison between the recorded and the theoretical knee angles (and so gait phases too) for a "walking on a flat surface" protocol. The 2 thin curves correspond to 2 different gait cycles of one single subject under examination, while the thick curve represents the range of expected values.

The collected data revealed that the sensor provides reliable measurements. Indeed, the succession of the real life 5 gait phases is in accordance with the theoretical one. These phases are the flexion-extension periods, and not the 8 classical phases as previously mentioned in subsection 1.4. Regarding the trend of the gait cycle curve, the real one perfectly mimics the expected curve except for the first peak (local maximum), which reaches higher angle values. More precisely, the measured knee angles for the first 2 phases are completely out of the theoretical range. Therefore, at first sight, they could be considered as being real out-of-range values (sign of abnormal gait). However, after some literature research, more testing and comparison with other subjects' outcomes, it resulted that this cosine-like curve is specific to the present goniometer. This proves that the out-of-range criteria should not be based on the theoretical knee angles (subsection 1.3) but rather on the speed and acceleration variations (i.e. slope and concavity) (Algorithm 1).

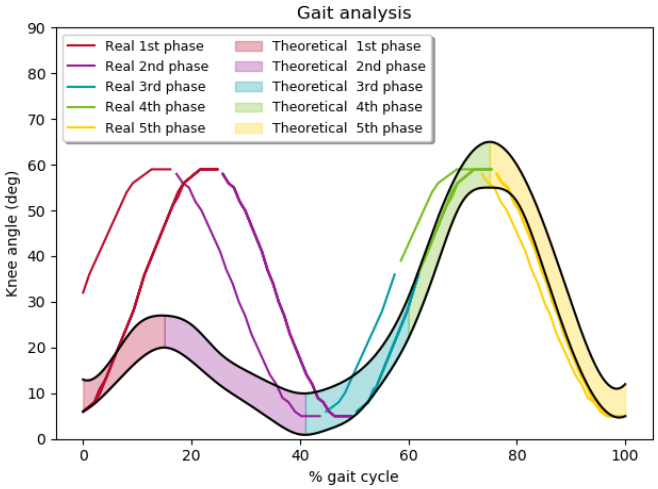


Fig. 16: Post-analysis of the gait analysis.

5.4 Limitations

It is significant to highlight that there were certain restrictions on the study related to the development of this potentiometer-based goniometer. The main one being the sample size of subjects. Indeed, the tiny amount of healthy volunteers could not accurately reflect the population's diversity in terms of age, gender, and pathology.

While potentiometers can be used for reliable gait analysis, they are still known to have a couple of limitations that need to be considered:

- **Single-axis measurement**: Potentiometers provide measurements along a single axis, which means they enable the capture of the joint only in one single plane. However, gait analysis often requires considering multiple planes simultaneously. Usually, those planes are the sagittal, frontal, and transverse ones.
- **Limited accuracy and precision**: The accuracy and precision of potentiometers can be affected by several factors such as electrical noise and mechanical hysteresis. Those can introduce errors and affect the reliability of the measurements obtained during gait analysis.
- **Invasive attachment**: Potentiometers require attaching a physical sensor to the body segment being measured. This could potentially cause discomfort or even alter the natural gait pattern.
- **Limited portability**: Potentiometers typically require wired connections to data acquisition systems, limiting the participant's mobility and potentially affecting the natural gait pattern.

To overcome these limitations, many gait analysis systems now incorporate other types of sensors, such as inertial measurement units (IMUs). Those are characterised by a wider range of motion and a low degree of invasiveness while providing measurements in multiple axes and offering wireless data transmission. Therefore, IMUs can complement or replace potentiometers to provide even more comprehensive and accurate gait analysis data.

5.5 Future work

Future studies might concentrate on verifying the current potentiometer-based goniometer in a broader / more varied population, including subjects with particular diseases and aberrant gaits. This would provide a more thorough grasp of how the sensor functions in real situations.

One additional task could involve the development and implementation of an application that, as the computer used up to now, would track the time evolution of the knee angle. Its main advantage is that it would make the gait analysis even more simple as this type of platform can be uploaded on any external device such as smartphones. This application would have the capability to store data, extract relevant information as well as display gait parameters.

Finally, to enhance the actual system's reliability, a screw could be added at the extremities of the 3D structure to assure a perfect connection between both arms without impairing their relative movement.

A Appendix

A.1 `Processing` code

```
1
import processing.serial.*;
3 import grafica.*;

5 Serial port;
////////////////////////////////////////////////////////////////////////////////
7 PrintWriter output;

9
////////////////////////////////////////////////////////////////////////////////
11
boolean waitForStart = true;
13
boolean Finish = false;
15
////////////////////////////////////////////////////////////////////////////////
17 int lf=10;

19 float time = 0.0; // initial time value

21 String myString = null;
//String myString ="10.4";
23 float angle=0.0;

25



27 int samplingFrequency = 30;
float samplingTime = 1.0/samplingFrequency;
29
float maxTime = 6.0; // maximum time value for the graph
31
float value = 0.0; // potentiometer value (0-1023)
33
float maxSpeed=5;
35


37 ArrayList<Float> angleList;

39 int sizeList = 10;

41
float angle_speed;
43 float previous_angle_speed;

45 float angle_acceleration;
float previous_angle_acceleration;
47


49 float angle_knee = 0.0; // initial angle value
float previous_angle_knee = 0.0;
51


53 int current_phase = 0;
int previous_phase = 0;
55



57


59 ////////////////////////////////////////////////////////////////////////////


61



63 float graphWidth = 1000.0; // width of the graph
float graphHeight = 600.0; // height of the graph
```

```
65
float margin = 700.0; // margin from the edge
67 float marginy = 200;

69
int dialSize = 360; // size of the speedometer dial
71 int dialAngle = 240; // angle of the speedometer dial (in degrees)
int dialStart = 150; // starting angle of the speedometer dial (in degrees)
73 int margin1 = 300;
int margin2=margin1+500;
75
int sizex=1900;
77 int sizey=1600;

79 ////////////////////////////////////////////////////////////////////////////

81 GPlot plot;
int step = 0;
83 int stepsPerCycle = int(maxTime)*samplingFrequency*100;

85 float Timedelta = 6.0;
float lowest_x = 0.0;
87 float upper_x = lowest_x + Timedelta;

89 GPointsArray points1 = new GPointsArray(stepsPerCycle);

91 ArrayList<Integer> pointColors;

93 ////////////////////////////////////////////////////////////////////////////////

95 void setup() {
  size(1900,1600);
97   background(255);
  port = new Serial (this, "COM14", 9600);
99
  String filename = "test_post_analysis.txt";
101   output = createWriter(filename);
  angleList = new ArrayList<Float>();
103   create_graph();
  pointColors = new ArrayList<Integer>(stepsPerCycle);
105 }

107
////////////////////////////////////////////////////////////////////////////////
109
void draw() {
111   stroke(0);

113   draw_graph();

115   // draw the speedometer dial
  draw_speedometer();
117


119


121   if (!waitForStart && !Finish){

123
    myString = port.readStringUntil(lf);
125
   if (myString != null) {
127       angle_knee= float(myString);
      //angle_knee=cos((time)*30.0/TWO_PI)*20.0+50.0;
129       time += samplingTime;
  }
131


133
```

```
  if(Math.abs( angle_knee - previous_angle_knee) < 30 || angleList.size() ==0){

    value = angle_knee;
    gait_analysis();
    draw_point();
    previous_angle_knee = angle_knee;
  }
  }

}

////////////////////////////////////////////////////////////////////////////
void gait_analysis(){

  output.println(value);
  angleList.add(angle_knee);
  if (angleList.size() > sizeList) {
        angleList.remove(0);
  }
  calculateSpeed();
  calculateAcceleration();
  println(current_phase,time, angle_knee,angle_speed,angle_acceleration,
    previous_angle_acceleration);
  CurrentPhase();
  //println(current_phase);
}


void calculateSpeed(){

  if (angleList.size() > 1) {
    float currentAngle = angleList.get(angleList.size() - 1);
    float previousAngle = angleList.get(angleList.size() - 2);
    previous_angle_speed = angle_speed;
    angle_speed = (currentAngle - previousAngle) ;/// deltaTime;
  } else {
 previous_angle_speed = 0.0;
    angle_speed = 0.0;
  }

}

void calculateAcceleration() {

  if (angleList.size() > 2) {
    float currentSpeed = angle_speed;
    float previousSpeed = previous_angle_speed;

    previous_angle_acceleration = angle_acceleration;
    angle_acceleration = (currentSpeed - previousSpeed) ;/// deltaTime;
  } else {
    previous_angle_acceleration =0.0;
    angle_acceleration = 0.0;
  }
}


void CurrentPhase(){
   previous_phase = current_phase;

  if (angleList.size()< 2){
    current_phase = 0;
  }
  else if(Math.abs( angle_knee - previous_angle_knee) <1.0){
    current_phase = current_phase;
  }
  else if (Math.abs( angle_knee - previous_angle_knee) > 10){
    current_phase = 5;
  }
```

```
  else if(current_phase == 0 && angle_speed >=0.0){
    current_phase = 0;
  }
  else if(current_phase == 0 && angle_speed <0.0){
    current_phase = 1;
  }
  else if(current_phase == 1 && angle_speed <=0.0){
    current_phase = 1;
  }
  else if(current_phase == 1 && angle_speed >0.0){
current_phase = 2;
  }
  else if(current_phase == 2 && angle_acceleration >=0.0){
    current_phase = 2;
  }
  else if(current_phase == 2 && angle_acceleration < 0.0){// && previous_angle_acceleration
    <=0.000){
    current_phase = 3;
  }
  else if(current_phase == 3 && angle_speed >=0.0){
    current_phase = 3;
  }
  else if(current_phase == 3 && angle_speed < 0.0){
    current_phase = 4;
  }
  else if(current_phase == 4 && angle_speed < 0.0){
    current_phase = 4;
  }
  else if(current_phase == 4 && angle_speed > 0.0){
  current_phase =0;
  output.println("new");
  }
  else{
   current_phase = current_phase;
  }


}


////////////////////////////////////////////////////////////////////////////:

void keyPressed() {
  if (key == 'c' || key == 'C') {
    background(255);
    for(int j =0;j<50;j++){
     for (int i=0; i < plot.getPointsRef().getNPoints();i++){
          plot.removePoint(0);
    }
    }
    pointColors.clear();
    angleList.clear();
    plot.setXLim(0, 0+Timedelta);
    lowest_x=0;
    upper_x= 0+Timedelta;
    time = 0.0;
    waitForStart = true;
    Finish = false;
    step=0;
  }
  if (key == 's' || key == 'S') {
    pointColors.clear();
    waitForStart = false;
    time = 0.0;
    current_phase = 0;
  }

  if (key == 'e' || key == 'E') {
    Finish = true;
```

```
  }
  if(key == 'x' || key == 'X'){

     output.flush();
     output.close();
     exit();
  }


}

void mousePressed() {
output.println("new");
current_phase = 0;
}


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


void create_graph(){
  plot = new GPlot(this);
  plot.setPos(margin, marginy);
  plot.setDim(graphWidth, graphHeight);


  // Set the plot limits (this will fix them)
  plot.setXLim(lowest_x, upper_x);
  plot.setYLim(0, 90);


  plot.getXAxis().setAxisLabelText("Time (s)");
  plot.getYAxis().setAxisLabelText("Knee angle (deg)");

  // Activate the panning effect
  plot.activatePanning();

}

void draw_graph(){


  plot.beginDraw();
  plot.drawBackground();
  plot.drawBox();
  plot.drawXAxis();
  plot.drawYAxis();
  plot.drawTopAxis();
  plot.drawRightAxis();
  plot.getMainLayer().drawPoints();
  plot.endDraw();



  //title
  textAlign(CENTER, TOP);
  fill(0);
  textSize(50);
  text("Gait analysis",sizex/2,30);

  //text("Time evolution of the knee angle", graphWidth/2 + margin, marginy-90);



  //phase legend
  stroke(255,255,255);
  fill(186,12,47);
  rect(margin-600,margin1+dialSize/2-40,40,20);
  fill(0);
```

```
  textAlign(LEFT,CENTER);
  textSize(20);
  text("1st phase",margin-540,margin1+dialSize/2-32);

  fill(152,29,151);
  rect(margin-600,margin1+dialSize/2-10,40,20);
  fill(0);
  textSize(20);
  text("2nd phase",margin-540,margin1+dialSize/2-2);

  fill(0,181,226);
  rect(margin-600,margin1+dialSize/2+20,40,20);
  fill(0);
  textSize(20);
  text("3rd phase",margin-540,margin1+dialSize/2+28);

  fill(120,190,32);
  rect(margin-350,margin1+dialSize/2-40,40,20);
  fill(0);
  textSize(20);
  text("4th phase",margin-290,margin1+dialSize/2-32);

  fill(255,209,0);
  rect(margin-350,margin1+dialSize/2-10,40,20);
  fill(0);
  textSize(20);
  text("5th phase",margin-290,margin1+dialSize/2-2);

  fill(255,163,0);
  rect(margin-350,margin1+dialSize/2+20,40,20);
  fill(0);
  textSize(20);
  text("Out of range",margin-290,margin1+dialSize/2+28);


}

void draw_speedometer(){

  pushMatrix();
  translate(margin1,margin1-50);
  //text("  ",margin1+90,margin1);
  strokeWeight(2);
  stroke(0);
  noFill();
  arc(0, 0, dialSize, dialSize, radians(dialStart), radians(dialStart + dialAngle));
  for (int i = 0; i <= dialAngle/10; i++) {
    float angle = radians(dialStart + i * 10);
    float x1 = (dialSize/2 - 10) * cos(angle);
    float y1 = (dialSize/2 - 10) * sin(angle);
    float x2 = dialSize/2 * cos(angle);
    float y2 = dialSize/2 * sin(angle);
    line(x1, y1, x2, y2);
  }
  fill(0);


  popMatrix();

  pushMatrix();
  translate(margin1,margin2-50);
  //text("  ",margin1+90,margin1);
  strokeWeight(2);
  stroke(0);
  noFill();
  arc(0, 0, dialSize, dialSize, radians(dialStart), radians(dialStart + dialAngle));
  for (int i = 0; i <= dialAngle/10; i++) {
    float angle = radians(dialStart + i * 10);
```

```
    float x1 = (dialSize/2 - 10) * cos(angle);
    float y1 = (dialSize/2 - 10) * sin(angle);
    float x2 = dialSize/2 * cos(angle);
    float y2 = dialSize/2 * sin(angle);
    line(x1, y1, x2, y2);
  }

  popMatrix();

  textSize(35);
  textAlign(CENTER, CENTER);
  text("Angle value",margin1,margin1-70+dialSize/2);
  text("Angle speed",margin1,margin2-70+dialSize/2);
}
void draw_point(){

  float handAngleSpeed=-12*angle_speed;

  textAlign(CENTER, CENTER);
  textSize(36);
  noStroke();
  fill(255);
  circle(margin1,margin2-50,dialSize*0.85); //white circle, overwriting the previous line, to
    draw a new line on top of it
  fill(255);
  rect(margin1-50,margin2,100,50);
  fill(0);
  text(nf(angle_speed, 0, 2)+"  /s", margin1, margin2+20);
  stroke(0);
  line(margin1,margin2-50, margin1-dialSize*0.375*sin(radians(handAngleSpeed)),margin2-50-
    dialSize*0.375*cos(radians(handAngleSpeed))); //drawing the speedometer hand

  float handAngle=120-4*value/3; //converting knee angle to speedometer angle

  // draw the value as text
  textAlign(CENTER, CENTER);
  textSize(36);
  noStroke();
  fill(255);
  circle(margin1,margin1-45,dialSize*0.85); //white circle, overwriting the previous line, to
    draw a new line on top of it
  fill(255);
  rect(margin1-40,margin1,80,50);

  //// Color according to angle
  if (current_phase == 0){
  fill(186,12,47);
  text(nf(value, 0, 2)+"  ", margin1, margin1+20);

  }
  else if (current_phase == 1){
  fill(152,29,151);
  text(nf(value, 0, 2)+"  ", margin1, margin1+20);

  }
   else if (current_phase == 2){
  fill(0,181,226);
  text(nf(value, 0, 2)+"  ", margin1, margin1+20);

  }
   else if (current_phase == 3){
  fill(120,190,32);
  text(nf(value, 0, 2)+"  ", margin1, margin1+20);

  }
   else if (current_phase == 4){
  fill(255,209,0);
  text(nf(value, 0, 2)+"  ", margin1, margin1+20);

```

```
  }
475  else{
  fill(255,163,0);
477  text(nf(value, 0, 2)+"  ", margin1, margin1+20);
  }
479  stroke(0);
  line(margin1,margin1-50, margin1-dialSize*0.375*sin(radians(handAngle)),margin1-50-dialSize
    *0.375*cos(radians(handAngle))); //drawing the speedometer hand
481

483  GPoint current_point = new GPoint(time,value);

485

  if (maxTime < time){
487

      lowest_x=lowest_x+ samplingTime/2.0;
489   upper_x= upper_x+ samplingTime/2.0;
      println("limit",lowest_x);
491   plot.setXLim(lowest_x, upper_x);

493

  }
495  if (step >= stepsPerCycle){
    plot.removePoint(0);
497   pointColors.remove(0);
  }
499

  if (current_phase == 0){
501  plot.addPoint(current_point);
   pointColors.add(color(186,12,47));
503

  }
505  else if (current_phase == 1){

507    plot.addPoint(current_point);
    pointColors.add(color(152,29,151));
509

  }
511  else if (current_phase == 2){
      plot.addPoint(current_point);
513  pointColors.add(color(0,181,226));

515  }
  else if (current_phase == 3){
517

    plot.addPoint(current_point);
519   pointColors.add(color(120,190,32));

521  }
  else if (current_phase == 4){
523

    plot.addPoint(current_point);
525   pointColors.add(color(255,209,0));

527  }
  else{
529     plot.addPoint(current_point);
    pointColors.add(color(255, 163, 0));
531  current_phase = previous_phase;
  }
533

  int[] arr = pointColors.stream().mapToInt(Integer::intValue).toArray();
535  plot.setPointColors(arr);
  step++;
537

}
```

Listing 5: `Processing` code - graphic interface.

A.2 Python code

```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import numpy as np


"""
Format du file:
First column with the digital value of thje potentiometer (0-1023)
Second column with the angle value measured (0-180)
"""
data = np.loadtxt('calibration.txt')


X = data[:, 0]
y = data[:, 1]

X=X.reshape(-1,1)
y=y.reshape(-1,1)



reg = LinearRegression().fit(X, y)


print("The coeeficient is: ", reg.coef_)
print("The intercept is: ", reg.intercept_)


y_pred = reg.predict(X)


plt.scatter(X, y, color="black")
plt.ylabel("angle(deg)")
plt.xlabel("Value of arduino analog read")
plt.plot(X,y_pred , color="blue", linewidth=3)



plt.show()
```

Listing 6: Phython code - calibration.

```


import matplotlib.pyplot as plt
import numpy as np
from scipy import interpolate


############################################################


def create_array(file_path):
    arrays = []
    values = []
    with open(file_path, 'r') as file:
        for line in file:
            line = line.strip()
            if line == 'new':
                array = np.array(values, dtype=float)
                arrays.append(array)
                values = []
            else:
                values.append(float(line))
    return arrays


```

```
def attribute_phase(lst_pourcentages, gait):
    lst_phases = [[[lst_pourcentages[0], gait[0]]], [], [], [], [], []]

    current_state = 0

    current_angle = gait[0]
    previous_angle = gait[0]

    current_speed = 0
    previous_speed = 0

    current_acc = 0
    previous_acc = 0

    for i in range(1, len(gait), 1):
        previous_angle = current_angle
        current_angle = gait[i]

        previous_speed = current_speed
        current_speed = current_angle - previous_angle

        previous_acc = current_acc
        current_acc = current_speed - previous_speed

        if np.abs(current_angle - previous_angle) > 9:
            lst_phases[5].append([lst_pourcentages[i], gait[i]])
        elif current_state == 0 and current_speed >= 0:
            lst_phases[0].append([lst_pourcentages[i], gait[i]])
            current_state = 0
        elif current_state == 0 and current_speed < 0:  # and previous_speed <0:
            lst_phases[1].append([lst_pourcentages[i], gait[i]])
            current_state = 1
        elif current_state == 1 and current_speed <= 0:
            lst_phases[1].append([lst_pourcentages[i], gait[i]])
            current_state = 1
        elif current_state == 1 and current_speed > 0:  # and previous_speed >0:
            lst_phases[2].append([lst_pourcentages[i], gait[i]])
            current_state = 2
        elif current_state == 2 and current_acc >= 0:
            lst_phases[2].append([lst_pourcentages[i], gait[i]])
            current_state = 2
        elif current_state == 2 and current_acc < 0:  # and previous_acc <0:
            lst_phases[3].append([lst_pourcentages[i], gait[i]])
            current_state = 3
        elif current_state == 3 and current_speed >= 0:
            lst_phases[3].append([lst_pourcentages[i], gait[i]])
            current_state = 3
        elif current_state == 3 and current_speed < 0:  # and previous_speed <0:
            lst_phases[4].append([lst_pourcentages[i], gait[i]])
            current_state = 4
        elif current_state == 4:
            lst_phases[4].append([lst_pourcentages[i], gait[i]])
            current_state = 4
        else:
            lst_phases[5].append([lst_pourcentages[i], gait[i]])
    return lst_phases


##############################################################

phase_1_pourcentage = np.linspace(0, 15, 100)
phase_2_pourcentage = np.linspace(15, 41, 100)
phase_3_pourcentage = np.linspace(41, 60, 100)
phase_4_pourcentage = np.linspace(60, 75, 100)
phase_5_pourcentage = np.linspace(75, 100, 100)

standard_values = np.array(
    [[0, 6, 13],
     [5, 11, 17],
```

```
94      [10, 17, 25],
        [15, 20, 27],
96      [20, 17, 25],
        [25, 12, 19],
98      [30, 8, 15],
        [35, 4, 12],
100     [40, 1, 10],
        [45, 2, 11],
102     [50, 5, 14],
        [55, 12, 20],
104     [60, 22, 31],
        [65, 37, 47],
106     [70, 52, 60, ],
        [75, 55, 65],
108     [80, 52, 60],
        [85, 37, 47],
110     [90, 20, 30],
        [95, 8, 15],
112     [100, 5, 12],
        ]
114 )

116 lower_line = interpolate.interp1d(standard_values[:, 0], standard_values[:, 1], kind='cubic')

118 upper_line = interpolate.interp1d(standard_values[:, 0], standard_values[:, 2], kind='cubic')

120 lst_pourcentages = np.linspace(0, 100, 1000)

122

##
    ##########################################################################################################

124 """
Format du file
126 column with the angle value and the different cycles divided by "new"
"""
128

file_path = 'post_analysis.txt'  # Replace with your file path
130 arrays = create_array(file_path)

132 ##########################################################################################################



134

lst_legend = ["Real 1st phase", "Real 2nd phase", "Real 3rd phase", "Real 4th phase", "Real 5
    th phase",
136               "out of range"]
lst_color = ["#BA0C2F", "#981D97", "#009CA6", "#78BE20", "#FFD100", "red"]
138

plt.title("Gait analysis")
140

for array, cz in zip(arrays, range(0, len(arrays))):
142     print(array)
        if len(array) != 0:
144         lst_pourcentages = np.linspace(0,100,len(array))
            lst_phases = attribute_phase(lst_pourcentages, array)
146         print(lst_phases)
            for l, leg, c in zip(lst_phases, lst_legend, lst_color):
148             if len(l) != 0:
                    lst_p = []
150                 lst_a = []
                    for j in l:
152                     lst_p.append(j[0])
                        lst_a.append(j[1])
154                 if cz == len(arrays)-1:
                        plt.plot(lst_p, lst_a, label=leg, color=c)
156                     plt.legend()
                    else:
158                     plt.plot(lst_p, lst_a, color=c)
```

```
##
    ##############################################################################################################


plt.plot(np.linspace(0, 100, 1000), lower_line(np.linspace(0, 100, 1000)), color="black")
plt.plot(np.linspace(0, 100, 1000), upper_line(np.linspace(0, 100, 1000)), color="black")

plt.fill_between(phase_1_pourcentage, lower_line(phase_1_pourcentage), upper_line(
    phase_1_pourcentage), color="#BA0C2F",
                 alpha=0.3, label="Theoretical  1st phase")
plt.fill_between(phase_2_pourcentage, lower_line(phase_2_pourcentage), upper_line(
    phase_2_pourcentage), color="#981D97",
                 alpha=0.3, label="Theoretical  2nd phase")
plt.fill_between(phase_3_pourcentage, lower_line(phase_3_pourcentage), upper_line(
    phase_3_pourcentage), color="#009CA6",
                 alpha=0.3, label="Theoretical  3rd phase")
plt.fill_between(phase_4_pourcentage, lower_line(phase_4_pourcentage), upper_line(
    phase_4_pourcentage), color="#78BE20",
                 alpha=0.3, label="Theoretical  4th phase")
plt.fill_between(phase_5_pourcentage, lower_line(phase_5_pourcentage), upper_line(
    phase_5_pourcentage), color="#FFD100",
                 alpha=0.3, label="Theoretical  5th phase")

plt.xlabel("% gait cycle")
plt.ylabel("Knee angle (deg)")
plt.ylim(0, 90)
plt.legend(loc='upper left',
           ncol=2, fancybox=True, shadow=True, prop={'size': 9})
plt.show()
print("over")
```

Listing 7: `Phython` code - post-analysis.

References

[1] C. Adel, Y. Benabid, and M. A. Louar. "Design and implementation of an electronic goniometer for gait analysis". In: *Computer Methods in Biomechanics and Biomedical Engineering* 20.sup1 (2017), S3–S4. DOI: 10.1080/10255842.2017.1382832.

[2] Christin Büttner, Thomas L. Milani, and Freddy Sichting. "Integrating a Potentiometer into a Knee Brace Shows High Potential for Continuous Knee Motion Monitoring". In: *Sensors* 21.6 (2021). ISSN: 1424-8220. DOI: 10.3390/s21062150. URL: https://www.mdpi.com/1424-8220/21/6/2150.

[3] Quartz Components. URL: https://quartzcomponents.com/products/10k-potentiometer (visited on 05/30/2023).

[4] Pankaj D. "Evaluation of Normal Gait Using Electro-Goniometer". In: *Journal of Scientific Industrial Research* 68 (2009), pp. 696–698. DOI: 10.1080/10255842.2017.1382832.

[5] Lidl. URL: https://www.lidl.co.uk/our-products (visited on 05/27/2023).

[6] E. Maranesi et al. "A goniometer-based method for the assessment of gait parameters". In: (2014), pp. 1–4. DOI: 10.1109/MESA.2014.6935539.

[7] Walter Pirker and Regina Katzenschlager. "Gait disorders in adults and the elderly: A clinical guide". In: *Wiener klinische Wochenschrift* 129 (Oct. 2016). DOI: 10.1007/s00508-016-1096-4.

[8] Andressa Rezende et al. "Polymer Optical Fiber Goniometer: A New Portable, Low Cost and Reliable Sensor for Joint Analysis". In: *Sensors* 18.12 (2018). ISSN: 1424-8220. DOI: 10.3390/s18124293. URL: https://www.mdpi.com/1424-8220/18/12/4293.

[9] Arduino Store. URL: https://store-usa.arduino.cc/products/arduino-micro (visited on 05/29/2023).

[10] Can Tunca et al. "Inertial Sensor-Based Robust Gait Analysis in Non-Hospital Settings for Neurological Disorders". In: *Sensors* 17.4 (2017). ISSN: 1424-8220. DOI: 10.3390/s17040825. URL: https://www.mdpi.com/1424-8220/17/4/825.