**BIOE/CS 494 Lab 5: Typing Glove**

Allison Bayro1 Frank Errichiello2 Alessandro Gozzi1 Tomasz Hulka2

1Department of Bioengineering, University of Illinois at Chicago, Chicago IL, 60607

2Department of Computer Science, University of Illinois at Chicago, Chicago IL, 60607

*This lab report for BIOE 494 describes the design and results of a carpal-tunnel monitoring typing glove. This wearable device force-sensing resistors, a three-axis accelerometer, and a flex sensor to monitor wrist flexion and use as a typing glove. This wearable is accompanied by a user-friendly interface to allow such monitoring and typing. The Arduino and Processing Integrated Development Environment was used to collect and utilize this wearable interactively on the developed typing user interface.*

1. **INTRODUCTION**

Currently, there is an expanded interest in the structural and functional capabilities of textiles able to measure and monitor vital signs and movement [1]. For example, certain knitted fabric sensors are able to detect the increase or decrease of resistance when stretched [1]. In this laboratory, a typing glove was developed with four embedded force-sensing resistors (FSRs) at the fingers to control typing and gestures as well as a three-axis accelerometer and gyroscope to allow selection of words. Further, a flex sensor was embedded with the glove at the wrist to monitor the angle flexion of the wrist as long hours of keyboard or mouse misuse may lead to or aggravate carpal tunnel syndrome [2]. Specfically, wrist angles greater than 20 degrees have the potential to bring about carpal tunnel syndrome [3]. This aggravation is notably severe wrist flexion angle that deviate from a straight line (Figure 1).

A picture containing square

Description automatically generated

*Figure 1: Wrist flexion angles (a, c, d, and f) that may result in carpal tunnel syndrome [4]*

The developed wearable typing gloves utilizes four FSRs within the fingertips to detect changes in applied pressure. Specially, FSRs determine changes in resistance due to a voltage divider setup [5]. If the value of the resistance change was higher than a predefined threshold, the program determined this to be a “touch”. Also, an embedded flex sensor utilizes a similar voltage divider circuit configuration in order to detect resistance changes to monitor wrist angle flexion [6]. Further, a three-axis accelerometer and gyroscope were used as a means to select sections of work in order to cut, copy, delete or paste. The MPU-6050 three-axis accelerometer and gyroscope measured the x, y, and z direction and acceleration in order to evaluate motion of the hand as opposed to stillness.

1. **METHODS**
   1. *Hardware*

The wearable typing glove was designed using several electronic components. Two DFRobot FireBeetle controllers were used as central and peripheral boards for wireless programming. The peripheral board was powered by a 3.7V, 1100 mAh lithium polymer battery. Further, four FSRs were used to determine changes in pressure at the fingertips by detecting resistance changes. These pressure sensors were electrically configured in a voltage divider setup using a 3.3kOhm resistors. The first finger, middle finger, ring finger, and pinky finger controlled the up, down, left, and right arrow keys, respectively. Pressure on the first and middle finger FSRs at once indicated a selection. Similarly, a flex sensor was utilized to measure changes in wrist angle flexion by detecting resistance changes. The flex sensor was also configured in a voltage divider setup using 3.3kOhm resistor. Finally, an accelerometer was implemented in the design to detect changes in motion to select words on the interface. The complete electrical circuit configuration is shown in Figure 2.

*Diagram

Description automatically generated*

*Figure 2: Fritzing diagram of wearable typing glove. \*Note: An Arduino Microcontroller is shown in the diagram, however, a DFRobot FireBeetle Board was used in the actual design.*

* 1. *Software*
     1. *Typing Interface*

In the typing interface, a keyboard is displayed on the lower half of the screen. The keyboard is laid out in the QWERTY style with lower-case letters, numbers and symbols similar to standard keyboard. Using the shift key, the user is able to change the lower-case letters to upper-case. The movement between different keys is controlled by the FSRs as described in the hardware methods section. The detection of the accelerometer movement allows the user to highlight words in order to copy, paste, or delete. The P5.js and Graphical User Interface code is included in Appendix B & C.

* + 1. *Carpal Tunnel Monitoring:*

The software monitors the wrist angle of the user via the flex sensor. Upon measuring an angle greater than \_\_\_ degree for over three seconds, the interface displays a message to fix the position of the wrist. A graph displays the wrist angle flexion over time to allow users to monitor they wrist position.

* 1. *Device Housing*

To assemble the electronics as a wearable typing glove, the pressure sensors were taped using double sided tape to the fingertips excluding the thumb. The wire was wrapped via athletic tape to prevent tangling. The flex sensor was embedded within the inside of the glove using double sided tape as well. The peripheral DRFRobot and LiPo battery were housed in a 3D-printed PLA box with a strap to be secured on the wrist.

1. **RESULTS**
   1. *Assembly*

The final assembly of the device shown in Figure 3.

**A**

*Figure 3. Design of the wearable typing glove (a) the electronics and (b) placement on the user*

* 1. *Interactive Home Screen*

The home screen is shown in Figure 4.

*Figure 4. Home screen of the wearable typing glove user interface*

* 1. *Typing Interface*

The results of the typing interface are shown in Figure 5. The user is able to select letters, numbers, or symbols using the FSRs located on the fingertips and accelerometer motion.

*Figure 5. Typing Interface.*

* 1. *Carpal Tunnel Monitoring*

The results of the implementation carpal tunnel monitoring feature are shown in Figure 6.

*Figure 6. Results of carpal tunnel monitoring*

1. CONCLUSION

The developed wearable typing glove, and accompanying user interface provided a wireless, user-friendly method to type on a keyboard and monitor the development of carpal tunnel. Specifically, the FSRs allowed the user to toggle between and select letters, numbers and symbols on the keyboard. The motion of the accelerometer provides the capability of highlighting words in order to copy, paste, cut or delete. The flex sensor monitored the angle of the wrist in order to prevent and aggravate carpal tunnel syndrome. Processing was used to develop GUI, hence, providing user interaction.

For future consideration, the wearable typing glove can incorporate algorithms to allow various combinations of pressure on the FSRs to represent letters, numbers of symbols, rather than toggling about a keyboard. Further, the material of the glove may be redesign with sensors embedded rather than taped to allow long-term use. Also, an additional flex sensor can be incorporated to monitor the wrist flexion in downwards and upwards direction.

**REFERENCES**

**[**1]E. Strazdienė, P. Blaževič, A. Vegys, and K. Dapkūnienė, “New Tendencies of Wearable Electronics Application in Smart Clothing,” *ELEKTRONIKA*.

[2] D. M. Rempel, P. J. Keir, and J. M. Bach, “Effect of wrist posture on carpal tunnel pressure while typing,” *Journal of Orthopaedic Research*, vol. 26, no. 9, pp. 1269–1273, 2008.

[3] C.-W. Liu, C.-H. Chen, C.-L. Lee, M.-H. Huang, T.-W. Chen, and M.-C. Wang, “Relationship Between Carpal Tunnel Syndrome and Wrist Angle in Computer Workers,” *The Kaohsiung Journal of Medical Sciences*, vol. 19, no. 12, pp. 617–622, 2003.

[4] “Effect of Wrist Deviation on Median Nerve Cross-Sectional ...” [Online]. Available: https://www.researchgate.net/publication/267903091\_Effect\_of\_Wrist\_Deviation\_on\_Median\_Nerve\_Cross-Sectional\_Area\_at\_Proximal\_Carpal\_Tunnel\_Level. [Accessed: 05-Apr-2021].

[5] “Short Flex Sensor,” *adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/1070?gclid=CjwKCAjwx6WDBhBQEiwA\_dP8rYtsgtTLQsejZ77O4vHX-fyE\_hjr6VMBmRIilMEfpWkQ3PBe9BNF9hoCKwAQAvD\_BwE. [Accessed: 04-Apr-2021].

[6] *Force Sensitive Resistor Hookup Guide*. [Online]. Available: https://learn.sparkfun.com/tutorials/force-sensitive-resistor-hookup-guide/all. [Accessed: 17-Feb-2021].

**Appendix A:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This is a library for the MPR121 12-channel Capacitive touch sensor

Designed specifically to work with the MPR121 Breakout in the Adafruit shop

----> https://www.adafruit.com/products/

These sensors use I2C communicate, at least 2 pins are required

to interface

Adafruit invests time and resources providing this open source code,

please support Adafruit and open-source hardware by purchasing

products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.

BSD license, all text above must be included in any redistribution

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <Wire.h>

#include "Adafruit\_MPR121.h"

#ifndef \_BV

#define \_BV(bit) (1 << (bit))

#endif

// You can have up to 4 on one i2c bus but one is enough for testing!

Adafruit\_MPR121 cap = Adafruit\_MPR121();

// Keeps track of the last pins touched

// so we know when buttons are 'released'

uint16\_t lasttouched = 0;

uint16\_t currtouched = 0;

boolean up = 0;

boolean down = 0;

boolean right = 0;

boolean left = 0;

boolean select = 0;

boolean back = 0;

void setup() {

Serial.begin(115200);

while (!Serial) { // needed to keep leonardo/micro from starting too fast!

delay(10);

}

Serial.println("Adafruit MPR121 Capacitive Touch sensor test");

// Default address is 0x5A, if tied to 3.3V its 0x5B

// If tied to SDA its 0x5C and if SCL then 0x5D

if (!cap.begin(0x5A)) {

Serial.println("MPR121 not found, check wiring?");

while (1);

}

Serial.println("MPR121 found!");

}

void loop() {

// Get the currently touched pads

currtouched = cap.touched();

for (uint8\_t i=0; i<6; i++) {

if (cap.filteredData(i) < 150) {

// Serial.print(i);

if (i == 0) {

left = 1;

}

if (i == 1) {

up = 1;

}

if (i == 2) {

right = 1;

}

if (i == 3) {

down = 1;

}

if (i == 4) {

select = 1;

}

if (i == 5) {

back = 1;

}

}

}

String toPrint = String(left) + "$"

+ String(up) + "$" + String(right) + "$" + String(down) + "$"

+ String(select) + "$" + String(back);

Serial.print(toPrint);

Serial.println();

up = 0;

down = 0;

right = 0;

left = 0;

select = 0;

back = 0;

// put a delay so it isn't overwhelming

delay(10);

}

**Appendix B:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<meta name="description" content="UIC CS494 SP20 Module 3">

<meta name="author" content="Tomasz Hulka">

<title>TheraSole - Gait Analysis App</title>

<!-- CDNS and more -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/5.0.0-alpha2/css/bootstrap.min.css" integrity="sha384-DhY6onE6f3zzKbjUPRc2hOzGAdEf4/Dz+WJwBvEYL/lkkIsI3ihufq9hk9K4lVoK" crossorigin="anonymous">

<link href="stylesheet.css" rel="stylesheet">

<script src="https://stackpath.bootstrapcdn.com/bootstrap/5.0.0-alpha2/js/bootstrap.bundle.min.js" integrity="sha384-BOsAfwzjNJHrJ8cZidOg56tcQWfp6y72vEJ8xQ9w6Quywb24iOsW913URv1IS4GD" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/5.0.0-alpha2/js/bootstrap.min.js" integrity="sha384-5h4UG+6GOuV9qXh6HqOLwZMY4mnLPraeTrjT5v07o347pj6IkfuoASuGBhfDsp3d" crossorigin="anonymous"></script>

<script src="https://kit.fontawesome.com/d9f17fec2b.js" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/p5@1.2.0/lib/p5.js"></script>

<script language="javascript" type="text/javascript" src="https://cdn.jsdelivr.net/npm/p5.serialserver@0.0.28/lib/p5.serialport.js"></script>

<script src="sketch.js"></script>

<script src="p5.sound.js"></script>

<script src="grafica.js"></script>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.0/font/bootstrap-icons.css">

</head>

<body>

<div class="d-flex" id="wrapper">

<!-- Sidebar -->

<div class="bg-light border-right" id="sidebar-wrapper">

<div class="sidebar-heading">Features</div>

<div class="list-group list-group-flush" id="sbar">

<a href="#" data-screen="home" class="list-group-item list-group-item-action bg-light">Home <i class="fas fa-home"></i></a>

<a href="#" data-screen="walking" class="list-group-item list-group-item-action bg-light">Walking <i class="fas fa-shoe-prints"></i></a>

<a href="#" data-screen="profile" class="list-group-item list-group-item-action bg-light">Profile <i class="fas fa-user"></i></a>

<a href="#" data-screen ="motion" class="list-group-item list-group-item-action bg-light">Motion <i class="far fa-clock"></i></a>

<a href="#" data-screen="therapy" class="list-group-item list-group-item-action bg-light">Therapy <i class="fas fa-user-md"></i></a>

</div>

</div>

<!-- /#sidebar-wrapper -->

<!-- Page Content -->

<div id="page-content-wrapper">

<!-- <nav class="navbar navbar-expand-lg navbar-light bg-light border-bottom fixed-top"> -->

<nav class="navbar navbar-expand-lg navbar-light bg-light border-bottom">

<button class="btn btn-primary" id="menu-toggle" style="margin-left: 5px;">View Menu</button>

</nav>

<!-- Home page -->

<div class="container-fluid screen mt-1" id="home">

<div class="row ml-5 mr-5">

<div class="col-md-12 text-center">

<h1 class="mt-4 mb-2">Welcome to TheraSole<i style="font-size: 20px;" class="far fa-copyright"></i></h1>

</div>

<div class="col-md-4 m-auto text-center">

<div class="card">

<img class="card-img-top" src="https://www.tekscan.com/sites/default/files/f-scan-inset.jpg" style="max-height: 400px;" alt="Card image cap">

<div class="card-body">

<p class="card-text">

<h5>TheraSole<i style="font-size: 10px;" class="far fa-copyright"></i> is a wearable smart shoe insole that incorporates force sensors and accelerators for the purpose of analyzing the user's gait. This responsive interface creates stunning visualizations and metrics that can be viewed on any device</h5>

</p>

</div>

</div>

</div>

</div>

<div class="row mt-5 ml-3 mr-3">

<div class="col-md-12 text-center">

<h3><b>We support four different modes with our device</b></h3>

</div>

<div class="col-md-1"></div>

<div class="col-md-5" id="modeInfo">

<div class="row">

<div class="col-md-6 mt-2 mb-2" id="modeInfo">

<div class="card">

<img class="card-img-top" src="https://www.active.com/Assets/Running/620/Finding+Your+Perfect+Run+Stride+620.jpg" style="height: 204px;" alt="Card image cap">

<div class="card-body">

<h5 class="card-title"><b>Walking:</b> <i class="fas fa-shoe-prints"></i></h5>

<p class="card-text">After collecting data for a 2 minute period, learn your cadence and step count. Displays force pad visualization and gyroscopic data.</p>

</div>

</div>

</div>

<div class="col-md-6 mt-2 mb-2" id="modeInfo">

<div class="card">

<img class="card-img-top" src="https://i.pinimg.com/originals/5e/1d/b0/5e1db03ff5d0fdeeb9c2397ae97820c3.png" style="height: 204px;" alt="Card image cap">

<div class="card-body">

<h5 class="card-title"><b>Profile:</b> <i class="fas fa-user"></i></h5>

<p class="card-text">Analyzes five gait profiles for 30 seconds each and calculates the Medial Force Percentage. Displays the current walking profile.</p>

</div>

</div>

</div>

</div>

</div>

<div class="col-md-5 " id="modeInfo">

<div class="row">

<div class="col-md-6 mt-2 mb-2" id="modeInfo">

<div class="card">

<img class="card-img-top" src="https://ak6.picdn.net/shutterstock/videos/1016312956/thumb/1.jpg?ip=x480" style="height: 204px;" alt="Card image cap">

<div class="card-body">

<h5 class="card-title"><b>Motion:</b> <i class="far fa-clock"></i></h5>

<p class="card-text">Determines whether the user is in motion or standing still.<br><br></p>

</div>

</div>

</div>

<div class="col-md-6 mt-2 mb-2" id="modeInfo">

<div class="card">

<img class="card-img-top" src="https://santecares.com/wp-content/uploads/2019/04/4-9-1.jpg" style="height: 204px;" alt="Card image cap">

<div class="card-body">

<h5 class="card-title"><b>Therapy:</b> <i class="fas fa-user-md"></i></h5>

<p class="card-text">Ensures that the gait cycle of a single foot is correct for those undergoing physical therapy.<br><br></p>

</div>

</div>

</div>

</div>

</div>

<div class="col-md-1"></div>

<div class="col-md-12 text-center mt-2 mb-5">

<h5><span style="font-style: italic;">To use these modes access the menu by the button in the top left.</span>

<i class="bi bi-arrow-up-left-circle-fill"></i> </h5>

</div>

</div>

</div>

<div class="container-fluid screen mt-1" id="walking">

<div class="row ml-5 mr-5">

<div class="col-md-12 text-center">

<h2 class="mt-4"> Walking <i class="fas fa-shoe-prints"></i></h2>

<div class="col-md-4 m-auto text-center">

<p>

This mode helps you learn your cadence and step count.

</p>

<h1 id="time" class="mt-4"></h1>

<h3 id="steps" class="mt-4"></h1>

<!-- <h6 class="mb-0">Stride Length: </h6> -->

<button class="btn btn-primary" id="startWalking" style="margin-left: 5px;" onclick="startActivity()">Start tracking</button>

<div class="col-md-6" id="currentData">

<!-- <h1 id="status"></h1> -->

</div>

</div>

</div>

<div class="col-md-12 m-auto text-center" id="walkingCanvas">

</div>

<div class= "col-md-6 text-center" id="walkingGraph1"></div>

<div class= "col-md-6 text-center" id="walkingGraph2"> </div>

<div class= "col-md-6 text-center" id="walkingGraph3"> </div>

<div class= "col-md-6 text-center" id="walkingGraph4"> </div>

</div>

</div>

<div class="container-fluid screen mt-1" id="profile">

<div class="row ml-5 mr-5">

<div class="col-md-12 text-center">

<h2 class="mt-4"> Profile <i class="fas fa-user"></i> </h2>

<p>

This mode analyzes five gait profiles for 30 seconds each and calculates the Medial Force Percentage. Displays the current walking profile.

</p>

<div class="col-md-4 m-auto text-center">

<h1 class="mt-4"></h1>

<h6></h6>

<!-- <button class="btn btn-primary" id="start-motion" style="margin-left: 5px;">Start tracking</button> -->

</div>

</div>

</div>

<div id="testingRow" class="row ml-5 mr-5 collapse">

<div class="col-md-12 text-center">

<h3>Lets test your calibrations</h3>

<p>To test please click start tracking</p>

<button class="btn btn-primary" id="btnTesting" style="margin-left: 5px;" onclick="startGaitTest()">Start Tracking</button>

</div>

</div>

<div id="calibrationRow" class="row ml-5 mr-5">

<div class="col-md-1"></div>

<div class="col-md-4">

<div class="row">

<div class="col-md-6 mt-2 mb-2" id="modeInfo">

<div class="card">

<div class="card-body">

<h5 id="normalGaitText" class="card-title"><b>Calibrate Normal Gait:</b> <i class="far fa-times-circle text-danger"></i></h5>

<p class="card-text">This is the state of walking normally.<br><br></p>

<button class="btn btn-primary" id="btnCalNormal" style="margin-left: 5px;" onclick="startGaitCalibration('normal')">Calibrate Normal</button>

</div>

</div>

</div>

<div class="col-md-6 mt-2 mb-2" id="modeInfo">

<div class="card">

<div class="card-body">

<h5 class="card-title" id="tiptoeGaitText"><b>Calibrate Tip-Toe Gait:</b> <i class="far fa-times-circle text-danger"></i></h5>

<p class="card-text">This is the state of walking on your tip toes.<br><br></p>

<button class="btn btn-primary" id="btnCalTipToe" style="margin-left: 5px;" onclick="startGaitCalibration('tiptoe')">Calibrate Tip-Toe</button>

</div>

</div>

</div>

</div>

</div>

<div class="col-md-2 mt-2 mb-2">

<div class="card">

<div class="card-body">

<h5 id="heelGaitText" class="card-title"><b>Calibrate Heel Gait:</b> <i class="far fa-times-circle text-danger"></i></h5>

<p class="card-text">This is the state of walking on your heels.<br><br></p>

<button class="btn btn-primary" id="btnCalHeel" style="margin-left: 5px;" onclick="startGaitCalibration('heel')">Calibrate Heel</button>

</div>

</div>

</div>

<div class="col-md-4">

<div class="row">

<div class="col-md-6 mt-2 mb-2" id="modeInfo">

<div class="card">

<div class="card-body">

<h5 id="intoeGaitText" class="card-title"><b>Calibrate In-Toe Gait:</b> <i class="far fa-times-circle text-danger"></i></h5>

<p class="card-text">This is the state of walking with your toes facing in.<br><br></p>

<button class="btn btn-primary" id="btnCalInToe" style="margin-left: 5px;" onclick="startGaitCalibration('intoe')">Calibrate In-Toe</button>

</div>

</div>

</div>

<div class="col-md-6 mt-2 mb-2" id="modeInfo">

<div class="card">

<div class="card-body">

<h5 id="outtoeGaitText" class="card-title"><b>Calibrate Out-Toe Gait:</b> <i class="far fa-times-circle text-danger"></i></h5>

<p class="card-text">This is the state of walking with your toes facing out.<br><br></p>

<button class="btn btn-primary" id="btnCalOutToe" style="margin-left: 5px;" onclick="startGaitCalibration('outtoe')">Calibrate Out-Toe</button>

</div>

</div>

</div>

</div>

</div>

<div class="col-md-1"></div>

</div>

<div class="row ml-5 mr-5">

<div class="col-md-12 text-center" id="currentTime">

<h1 id="time"></h1>

<h1 id="status"></h1>

</div>

<!-- <div class="col-md-6" id="currentData">

</div> -->

<div class="col-md-6 m-auto text-center" class="col-md-6" id="profileCanvas">

</div>

</div>

</div>

<div class="container-fluid screen mt-1" id="therapy">

<div class="row ml-5 mr-5">

<div class="col-md-12 text-center">

<h2 class="mt-4"> Therapy <i class="fas fa-user-md"></i> </h2>

<div class="col-md-4 m-auto text-center">

<h6 class="mb-0"></h6>

<p>

This mode ensures that you are walking heel to toe.

</p>

</div>

<h1 id="status"></h1>

</div>

<!-- <div class="col-md-6" id="currentData">

<h1 class="mt-4" id="time"></h1>

<h1 id="data"></h1>

<p id="myC"></p>

<p id="myS"></p>

</div> -->

<div class="col-md-6 m-auto text-center" class="col-md-6" id="therapyCanvas">

</div>

</div>

</div>

<div class="container-fluid screen mt-1" id="motion">

<div class="row ml-5 mr-5">

<div class="col-md-12 text-center">

<h2 class="mt-4"> Motion <i class="far fa-clock"></i></h2>

<div class="col-md-4 m-auto text-center">

<p>

This mode determines whether the user is in motion or standing still.

</p>

<!-- <button class="btn btn-primary" id="start-motion" style="margin-left: 5px;">Start tracking</button> -->

</div>

<div class="col-md-6 m-auto text-center">

<h1 id="status"></h1>

<h4 id="timeInMotion"></h4>

<h4 id="timeNotInMotion"></h4>

</div>

</div>

</div>

</div>

</div>

<!-- /#page-content-wrapper -->

</div>

<!-- /#wrapper -->

<script>

function startActivity(){

walkingStarted = true;

}

function startGaitTest(){

testingCalibration = true;

}

function startGaitCalibration(gaitType){

let btnCalNormal = document.getElementById('btnCalNormal');

let btnCalTipToe = document.getElementById('btnCalTipToe');

let btnCalHeel = document.getElementById('btnCalHeel');

let btnCalInToe = document.getElementById('btnCalInToe');

let btnCalOutToe = document.getElementById('btnCalOutToe');

btnCalNormal.disabled = true;

btnCalTipToe.disabled = true;

btnCalHeel.disabled = true;

btnCalInToe.disabled = true;

btnCalOutToe.disabled = true;

switch(gaitType){

case "normal":

calibratingNormal = true;

break;

case "tiptoe":

calibratingTipToe = true;

break;

case "heel":

calibratingHeel = true;

break;

case "intoe":

calibratingInToe = true;

break;

case "outtoe":

calibratingOutToe = true;

break;

}

}

// let myp5;

document.addEventListener('DOMContentLoaded', function() {

//myp5 = new p5(dummyP5, 'dummyDiv');

// menu button handler

document.querySelector('#menu-toggle').addEventListener("click", (event) => {

event.preventDefault();

window.scrollTo({ top: 0, behavior: 'smooth' });

document.querySelector('#wrapper').classList.toggle("toggled");

})

}, false);

// // motion button handler

// document.querySelector('#start-motion').addEventListener("click", (event) => {

// startMotionMode();

// })

//

// sidebar tab code below

//

document.querySelector("#home").style.display = "block";

let menuOptions = document.querySelectorAll('div#sbar a');

// handle clicks on each tab on the lefthand side

menuOptions.forEach( (anchor) => {

anchor.addEventListener("click", (event) => {

document.querySelectorAll(".screen").forEach((screen) => {

screen.style.display = "none"; // get rid of the open screens

})

let screen = event.target.getAttribute("data-screen");

switch(screen){

case "motion":

startMotionMode();

break;

case "profile":

startProfileMode();

break;

case "therapy":

startTherapyMode();

break;

case "walking":

startWalkingMode();

break;

}

let targetScreen = document.querySelector("#" + screen);

targetScreen.style.display = "block"; // display only the selected screen

document.querySelector('#wrapper').classList.toggle("toggled");

})

})

function startProfileMode(){

document.querySelector("#profile #status").innerHTML = "Walking profiles not configured";

setTimeout(startProfile, 0);

}

function startMotionMode() {

document.querySelector("#motion #status").innerHTML = "Please wait";

setTimeout(startMotion, 3000);

}

function startTherapyMode(){

document.querySelector("#therapy #status").innerHTML = "Please wait";

setTimeout(startTherapy, 3000);

}

function startWalkingMode() {

//document.querySelector("#walking #status").innerHTML = "Please wait";

graphFSRs();

setTimeout(startWalking, 0);

}

</script>

</body>

</html>

**Appendix C:**

let serial;

let latestData = "waiting for data";

let arduinoOutput = "";

let dataReady = false; // set to true once the first valid string is displayed

let heelReading = 0;

let mmReading = 0;

let lfReading = 0;

let mfReading = 0;

let inActivity = false;

let xAccel = 0;

let yAccel = 0;

let zAccel = 0;

let walkingStarted = false;

let calibratingNormal = false;

let calibratingTipToe = false;

let calibratingHeel = false;

let calibratingInToe = false;

let calibratingOutToe = false;

let testingCalibration = false;

function setup() {

serial = new p5.SerialPort();

serial.list();

serial.open('/dev/tty.usbmodem123456781'); //change per your system

serial.on('connected', serverConnected);

serial.on('list', gotList);

serial.on('data', gotData);

serial.on('error', gotError);

serial.on('open', gotOpen);

serial.on('close', gotClose);

noCanvas();

}

function startProfile() {

let m = new p5((sketch) => {

let currentGaitProfile;

let footImage;

let clientW = parentWidth(document.querySelector('#profileCanvas'))

let clientH;

let startTime;

let currentTime;

let firstLoop = true;

let state = 0;

let oldState = 0;

let heelReadingBundle = {

numReadings: 0,

numSteps: 0,

mfpAvg: 0,

heelAvg: 0,

mmAvg: 0,

lfAvg: 0,

mfAvg: 0,

finished: false,

};

let tiptoeReadingBundle = {

numReadings: 0,

numSteps: 0,

mfpAvg: 0,

heelAvg: 0,

mmAvg: 0,

lfAvg: 0,

mfAvg: 0,

finished: false,

};

let intoeReadingBundle = {

numReadings: 0,

numSteps: 0,

mfpAvg: 0,

heelAvg: 0,

mmAvg: 0,

lfAvg: 0,

mfAvg: 0,

finished: false,

};

let outtoeReadingBundle = {

numReadings: 0,

numSteps: 0,

mfpAvg: 0,

heelAvg: 0,

mmAvg: 0,

lfAvg: 0,

mfAvg: 0,

finished: false,

};

let normalReadingBundle = {

numReadings: 0,

numSteps: 0,

mfpAvg: 0,

heelAvg: 0,

mmAvg: 0,

lfAvg: 0,

mfAvg: 0,

finished: false,

};

let testingReadingBundle = {

numReadings: 0,

numSteps: 0,

mfpAvg: 0,

heelAvg: 0,

mmAvg: 0,

lfAvg: 0,

mfAvg: 0,

finished: false,

};

sketch.preload = function () {

footImage = loadImage('Images/Sole.svg');

currentGaitProfile = '';

currentTime = 0;

startTime = 0;

}

sketch.setup = function() {

currentTime = millis();

if(clientW > 700){

clientH = windowHeight \* .5;

cv = sketch.createCanvas(clientW, clientH);

}

else{

clientH = windowHeight \* .25;

cv = sketch.createCanvas(clientW, clientH);

}

// let cv = sk.createCanvas(document.querySelector(''))

cv.id("profileCV");

}

sketch.draw = function(){

sketch.background(255);

sketch.background(footImage);

currentTime =millis();

// update logic below using globals

let mfp = parseFloat(((mmReading + mfReading) \* 100) / (heelReading + mfReading + mmReading + lfReading + .001));

//document.querySelector('#profile h6').innerHTML = 'You are currently';

//document.querySelector('#profile #currentData').innerHTML = arduinoOutput;

// colors the circles on the foot drawing, with color intensity mapped to fsr reading

sketch.noStroke();

sketch.fill(3, 252, 32, (heelReading / 1023) \* 255);

sketch.circle(clientW/4.5,clientH/2,heelReading/6);

sketch.fill(246, 250, 12, (mmReading / 1023) \* 255);

sketch.circle(clientW/1.75,clientH/2.75,mmReading/6);

sketch.fill(10, 67, 252, (mfReading / 1023) \* 255);

sketch.circle(clientW/1.4,clientH/2.25,mfReading/6);

sketch.fill(242, 7, 39, (lfReading / 1023) \* 255);

sketch.circle(clientW/1.70,clientH/1.5,lfReading/6);

// if(mfp <= 22 && (mfReading <= 100 && lfReading <= 100)){

// currentGaitProfile = "Walking on your heel";

// }

// else if(mfp >= 40 && mfp <= 60){

// if(heelReading <= 100 && mfReading >= 100 && mmReading >= 100 && lfReading >= 100){

// currentGaitProfile = "Tip Toeing";

// }

// else if(heelReading >= 100 && mfReading >= 100 && mmReading >= 100 && lfReading <= 200){

// currentGaitProfile = "Intoeing";

// }

// else if(heelReading >= 100 && mfReading <= 100 && mmReading >= 100 && lfReading >= 100){

// currentGaitProfile = "Outoeing";

// }

// else{

// currentGaitProfile = "Walking Normally"

// }

// }

// //let gait = sketch.checkCurrentGait(testingReadingBundle);

// if (testingCalibration) {

// document.querySelector('#profile #status').innerHTML = currentGaitProfile;

// }

if(calibratingNormal){

if(firstLoop){

startTime = currentTime;

firstLoop = false;

}

currentTime = millis();

let realTime = (currentTime - startTime);

let time = msToTime(realTime);

timeString = time.timeString;

document.querySelector('#profile #time').innerHTML = timeString;

if(time.secs >= 30){

calibratingNormal = false;

sketch.calculateAvgs(normalReadingBundle);

console.log(normalReadingBundle);

document.getElementById('normalGaitText').innerHTML = '<b>Normal Profile Calibrated </b> <i class="far fa-check-circle text-success"></i>';

sketch.setCalibrationButtons();

firstLoop = true;

}

if(mfp <= 70){

normalReadingBundle.mfpAvg += mfp;

normalReadingBundle.heelAvg += heelReading;

normalReadingBundle.mmAvg += mmReading;

normalReadingBundle.lfAvg += lfReading;

normalReadingBundle.mfAvg += mfReading;

normalReadingBundle.numReadings += 1;

}

oldState = state;

if(mfp >= 0 && mfp <= 70) {

state = 0;

} else {

state = 1;

}

if (oldState != state) {

// change in state of foot

if(state === 0){

normalReadingBundle.numSteps += 1;

}

}

}

if(calibratingTipToe){

if(firstLoop){

startTime = currentTime;

firstLoop = false;

}

currentTime = millis();

let realTime = (currentTime - startTime);

let time = msToTime(realTime);

timeString = time.timeString;

document.querySelector('#profile #time').innerHTML = timeString;

if(time.secs >= 30){

calibratingTipToe = false;

sketch.calculateAvgs(tiptoeReadingBundle);

console.log(tiptoeReadingBundle);

document.getElementById('tiptoeGaitText').innerHTML = '<b>Tip-Toe Profile Calibrated </b> <i class="far fa-check-circle text-success"></i>';

sketch.setCalibrationButtons();

firstLoop = true;

}

if(mfp <= 70){

tiptoeReadingBundle.mfpAvg += mfp;

tiptoeReadingBundle.heelAvg += heelReading;

tiptoeReadingBundle.mmAvg += mmReading;

tiptoeReadingBundle.lfAvg += lfReading;

tiptoeReadingBundle.mfAvg += mfReading;

tiptoeReadingBundle.numReadings += 1;

}

oldState = state;

if(mfp >= 0 && mfp <= 70) {

state = 0;

} else {

state = 1;

}

if (oldState != state) {

// change in state of foot

if(state === 0){

tiptoeReadingBundle.numSteps += 1;

}

}

}

if(calibratingHeel){

if(firstLoop){

startTime = currentTime;

firstLoop = false;

}

currentTime = millis();

let realTime = (currentTime - startTime);

let time = msToTime(realTime);

timeString = time.timeString;

document.querySelector('#profile #time').innerHTML = timeString;

if(time.secs >= 30){

calibratingHeel = false;

sketch.calculateAvgs(heelReadingBundle);

console.log(heelReadingBundle);

document.getElementById('heelGaitText').innerHTML = '<b>Heel Profile Calibrated </b> <i class="far fa-check-circle text-success"></i>';

sketch.setCalibrationButtons();

firstLoop = true;

}

if(mfp <= 70){

heelReadingBundle.mfpAvg += mfp;

heelReadingBundle.heelAvg += heelReading;

heelReadingBundle.mmAvg += mmReading;

heelReadingBundle.lfAvg += lfReading;

heelReadingBundle.mfAvg += mfReading;

heelReadingBundle.numReadings += 1;

}

oldState = state;

if(mfp >= 0 && mfp <= 70) {

state = 0;

} else {

state = 1;

}

if (oldState != state) {

// change in state of foot

if(state === 0){

heelReadingBundle.numSteps += 1;

}

}

}

if(calibratingInToe){

if(firstLoop){

startTime = currentTime;

firstLoop = false;

}

currentTime = millis();

let realTime = (currentTime - startTime);

let time = msToTime(realTime);

timeString = time.timeString;

document.querySelector('#profile #time').innerHTML = timeString;

if(time.secs >= 30){

calibratingInToe = false;

sketch.calculateAvgs(intoeReadingBundle);

console.log(intoeReadingBundle);

document.getElementById('intoeGaitText').innerHTML = '<b>In-Toe Profile Calibrated </b> <i class="far fa-check-circle text-success"></i>';

sketch.setCalibrationButtons();

firstLoop = true;

}

if(mfp <= 70){

intoeReadingBundle.mfpAvg += mfp;

intoeReadingBundle.heelAvg += heelReading;

intoeReadingBundle.mmAvg += mmReading;

intoeReadingBundle.lfAvg += lfReading;

intoeReadingBundle.mfAvg += mfReading;

intoeReadingBundle.numReadings += 1;

}

oldState = state;

if(mfp >= 0 && mfp <= 70) {

state = 0;

} else {

state = 1;

}

if (oldState != state) {

// change in state of foot

if(state === 0){

intoeReadingBundle.numSteps += 1;

}

}

}

if(calibratingOutToe){

if(firstLoop){

startTime = currentTime;

firstLoop = false;

}

currentTime = millis();

let realTime = (currentTime - startTime);

let time = msToTime(realTime);

timeString = time.timeString;

document.querySelector('#profile #time').innerHTML = timeString;

if(time.secs >= 30){

calibratingOutToe = false;

sketch.calculateAvgs(outtoeReadingBundle);

console.log(outtoeReadingBundle);

document.getElementById('outtoeGaitText').innerHTML = '<b>Out-Toe Profile Calibrated </b> <i class="far fa-check-circle text-success"></i>';

sketch.setCalibrationButtons();

firstLoop = true;

}

if(mfp <= 70){

outtoeReadingBundle.mfpAvg += mfp;

outtoeReadingBundle.heelAvg += heelReading;

outtoeReadingBundle.mmAvg += mmReading;

outtoeReadingBundle.lfAvg += lfReading;

outtoeReadingBundle.mfAvg += mfReading;

outtoeReadingBundle.numReadings += 1;

}

oldState = state;

if(mfp >= 0 && mfp <= 70) {

state = 0;

} else {

state = 1;

}

if (oldState != state) {

// change in state of foot

if(state === 0){

outtoeReadingBundle.numSteps += 1;

}

}

}

if(testingCalibration){

if(firstLoop){

startTime = currentTime;

firstLoop = false;

}

document.querySelector('#profile #btnTesting').disabled = true;

currentTime = millis();

let realTime = (currentTime - startTime);

let time = msToTime(realTime);

timeString = time.timeString;

document.querySelector('#profile #time').innerHTML = timeString;

if(mfp <= 70){

testingReadingBundle.mfpAvg += mfp;

testingReadingBundle.heelAvg += heelReading;

testingReadingBundle.mmAvg += mmReading;

testingReadingBundle.lfAvg += lfReading;

testingReadingBundle.mfAvg += mfReading;

testingReadingBundle.numReadings += 1;

}

oldState = state;

if(mfp >= 0 && mfp <= 70) {

state = 0;

} else {

state = 1;

}

if (oldState != state) {

// change in state of foot

if(state === 0){

testingReadingBundle.numSteps += 1;

sketch.calculateAvgs(testingReadingBundle);

let gait = sketch.checkCurrentGait(testingReadingBundle);

document.querySelector('#profile #status').innerHTML = gait;

testingReadingBundle.mfpAvg = 0;

testingReadingBundle.heelAvg = 0;

testingReadingBundle.mmAvg= 0;

testingReadingBundle.lfAvg= 0;

testingReadingBundle.mfAvg = 0;

testingReadingBundle.numReadings = 0;

}

}

}

}

sketch.setCalibrationButtons = function () {

let btnCalNormal = document.getElementById('btnCalNormal');

let btnCalTipToe = document.getElementById('btnCalTipToe');

let btnCalHeel = document.getElementById('btnCalHeel');

let btnCalInToe = document.getElementById('btnCalInToe');

let btnCalOutToe = document.getElementById('btnCalOutToe');

btnCalNormal.disabled = normalReadingBundle.finished;

btnCalHeel.disabled = heelReadingBundle.finished;

btnCalTipToe.disabled = tiptoeReadingBundle.finished;

btnCalInToe.disabled = intoeReadingBundle.finished;

btnCalOutToe.disabled = outtoeReadingBundle.finished;

if(normalReadingBundle.finished &&

heelReadingBundle.finished &&

tiptoeReadingBundle.finished &&

intoeReadingBundle.finished &&

outtoeReadingBundle.finished)

{

document.getElementById('calibrationRow').classList.add('collapse');

document.getElementById('testingRow').classList.remove('collapse');

}

}

sketch.checkCurrentGait = function(bundle){

let heelReadings = {

MFPBelow: heelReadingBundle.mfpAvg - 5,

MFPAbove: heelReadingBundle.mfpAvg + 5,

HeelBelow: heelReadingBundle.heelAvg - 50,

HeelAbove: heelReadingBundle.heelAvg + 50,

MMBelow: heelReadingBundle.mmAvg - 50,

MMAbove: heelReadingBundle.mmAvg + 50,

MFBelow: heelReadingBundle.mfAvg - 50,

MFAbove: heelReadingBundle.mfAvg + 50,

LFBelow: heelReadingBundle.lfAvg - 50,

LFAbove: heelReadingBundle.lfAvg + 50,

}

let tipToeReadings = {

MFPBelow: tiptoeReadingBundle.mfpAvg - 5,

MFPAbove: tiptoeReadingBundle.mfpAvg + 5,

HeelBelow: tiptoeReadingBundle.heelAvg - 50,

HeelAbove: tiptoeReadingBundle.heelAvg + 50,

MMBelow: tiptoeReadingBundle.mmAvg - 50,

MMAbove: tiptoeReadingBundle.mmAvg + 50,

MFBelow: tiptoeReadingBundle.mfAvg - 50,

MFAbove: tiptoeReadingBundle.mfAvg + 50,

LFBelow: tiptoeReadingBundle.lfAvg - 50,

LFAbove: tiptoeReadingBundle.lfAvg + 50,

}

let normalReadings = {

MFPBelow: normalReadingBundle.mfpAvg - 5,

MFPAbove: normalReadingBundle.mfpAvg + 5,

HeelBelow: normalReadingBundle.heelAvg - 50,

HeelAbove: normalReadingBundle.heelAvg + 50,

MMBelow: normalReadingBundle.mmAvg - 50,

MMAbove: normalReadingBundle.mmAvg + 50,

MFBelow: normalReadingBundle.mfAvg - 50,

MFAbove: normalReadingBundle.mfAvg + 50,

LFBelow: normalReadingBundle.lfAvg - 50,

LFAbove: normalReadingBundle.lfAvg + 50,

}

let intoeReadings = {

MFPBelow: intoeReadingBundle.mfpAvg - 5,

MFPAbove: intoeReadingBundle.mfpAvg + 5,

HeelBelow: intoeReadingBundle.heelAvg - 50,

HeelAbove: intoeReadingBundle.heelAvg + 50,

MMBelow: intoeReadingBundle.mmAvg - 50,

MMAbove: intoeReadingBundle.mmAvg + 50,

MFBelow: intoeReadingBundle.mfAvg - 50,

MFAbove: intoeReadingBundle.mfAvg + 50,

LFBelow: intoeReadingBundle.lfAvg - 50,

LFAbove: intoeReadingBundle.lfAvg + 50,

}

let outtoeReadings = {

MFPBelow: outtoeReadingBundle.mfpAvg - 5,

MFPAbove: outtoeReadingBundle.mfpAvg + 5,

HeelBelow: outtoeReadingBundle.heelAvg - 50,

HeelAbove: outtoeReadingBundle.heelAvg + 50,

MMBelow: outtoeReadingBundle.mmAvg - 50,

MMAbove: outtoeReadingBundle.mmAvg + 50,

MFBelow: outtoeReadingBundle.mfAvg - 50,

MFAbove: outtoeReadingBundle.mfAvg + 50,

LFBelow: outtoeReadingBundle.lfAvg - 50,

LFAbove: outtoeReadingBundle.lfAvg + 50,

}

let similiarMFPAVGBelow = (tipToeReadings.MFPBelow + intoeReadings.MFPBelow + 10)\*.5 - 5;

let similiarMFPAVGAbove = (tipToeReadings.MFPAbove + intoeReadings.MFPAbove - 10)\*.5 + 5;

console.log(similiarMFPAVGBelow);

console.log(similiarMFPAVGAbove);

let gait = '';

if(bundle.mfpAvg >= heelReadings.MFPBelow && bundle.mfpAvg <= heelReadings.MFPAbove){

return "Heel";

}

if(bundle.mfpAvg >= outtoeReadings.MFPBelow && bundle.mfpAvg <= outtoeReadings.MFPAbove){

return "Out Toe";

}

if(bundle.mfpAvg >= normalReadings.MFPBelow && bundle.mfpAvg <= normalReadings.MFPAbove){

return "Normal";

}

if(bundle.mfpAvg >= similiarMFPAVGBelow && bundle.mfpAvg <= similiarMFPAVGAbove){

if((bundle.heelAvg >= tipToeReadings.HeelBelow && bundle.heelAvg <= tipToeReadings.HeelAbove)){

return 'Tip Toe';

}

else{

return 'In Toe';

}

}

gait = document.querySelector('#profile #status').innerHTML;

return gait;

}

sketch.calculateAvgs = function (bundle) {

bundle.finished = true;

bundle.mfpAvg = bundle.mfpAvg / bundle.numReadings;

bundle.heelAvg = bundle.heelAvg / bundle.numReadings;

bundle.mmAvg = bundle.mmAvg / bundle.numReadings;

bundle.lfAvg = bundle.lfAvg / bundle.numReadings;

bundle.mfAvg = bundle.mfAvg / bundle.numReadings;

console.log(bundle);

}

sketch.windowResized = function () {

console.log ("dummyP5 got a window resize");

clientW = parentWidth(document.querySelector('#profileCanvas'));

sketch.background(footImage);

if(clientW > 700){

clientH = windowHeight \* .5;

sketch.resizeCanvas(clientW, clientH);

}

else{

clientH = windowHeight \* .25;

sketch.resizeCanvas(clientW, clientH);

}

}

},'profileCanvas')

}

function startWalking() {

let m = new p5((sketch) => {

let stepCount;

let state = 0;

let oldState = 0;

let startTime;

let currentTime;

let firstLoop = true;

let finished = false;

let timeString = '';

let clientW = parentWidth(document.querySelector('#walkingCanvas'))

sketch.preload = function () {

footImage = loadImage('Images/Sole.svg');

stepCount = 0;

startTime = 0;

currentTime = 0;

}

sketch.setup = function () {

currentTime = millis();

sketch.noCanvas();

};

sketch.draw = function(){

sketch.background(255);

// update logic below using globals

currentTime = millis();

if(walkingStarted){

if(firstLoop){

startTime = currentTime;

firstLoop = false;

}

currentTime = millis();

let realTime = (currentTime - startTime);

let time = msToTime(realTime);

timeString = time.timeString;

if(time.mins == 2){

walkingStarted = false;

finished = true;

}

let mfp = parseFloat(((mmReading + mfReading) \* 100) / (heelReading + mfReading + mmReading + lfReading + .001));

oldState = state;

if(mfp >= 0 && mfp <= 70) {

state = 0;

} else {

state = 1;

}

if (oldState != state) {

// change in state of foot

if(state === 0){

stepCount+=1;

}

}

document.querySelector('#walking #steps').innerHTML = 'Steps: ' + stepCount;

}

document.querySelector('#walking #time').innerHTML = timeString;

if(finished){

let cadence = stepCount / 2;

let stride = 35.5;

let stepLength = stride / 2 + ' inches';

let strideLength = stride + ' inches';

let final = `Cadence: ${cadence} <br />

Step Length: ${stepLength} <br />

Stride Length: ${strideLength}`

document.querySelector('#walking #currentData').innerHTML = final;

}

else{

//document.querySelector('#walking #currentData').innerHTML = latestData;

}

}

sketch.windowResized = function () {

clientW = parentWidth(document.querySelector('#walkingCanvas'));

sketch.background(footImage);

if(clientW > 700){

sketch.resizeCanvas(clientW, windowHeight \* .5);

}

else{

sketch.resizeCanvas(clientW, windowHeight \* .25);

}

}

},'walkingCanvas')

}

function startTherapy() {

let m = new p5((sketch) => {

let footImage;

let clientW = parentWidth(document.querySelector('#therapyCanvas'))

let startTime;

let currentTime;

let walkingCorrectly = true;

let firstLoop = true;

let state = 0;

let oldState = 0;

let timeString = '';

let readHeel = false;

let readTip = false;

let heelTime;

let tipTime;

let countBad = 0;

let countGood = 0;

let mySound; // when a user is not walking correctly

let steps = []; // keep track of all the steps taken

sketch.preload = function () {

footImage = loadImage('Images/Sole.svg');

mySound = loadSound('chime.mp3');

currentTime = 0;

startTime = 0;

}

sketch.setup = function() {

currentTime = millis();

if(clientW > 700){

clientH = windowHeight \* .5;

cv = sketch.createCanvas(clientW, clientH);

}

else{

clientH = windowHeight \* .25;

cv = sketch.createCanvas(clientW, clientH);

}

// let cv = sk.createCanvas(document.querySelector(''))

cv.id("therapyCV");

}

sketch.draw = function(){

sketch.background(255);

sketch.background(footImage);

if (firstLoop) {

startTime = currentTime;

firstLoop = false;

}

currentTime = millis();

let realTime = (currentTime - startTime);

let time = msToTime(realTime);

timeString = time.timeString;

//document.querySelector('#therapy #time').innerHTML = timeString;

let mfp = parseFloat(((mmReading + mfReading) \* 100) / (heelReading + mfReading + mmReading + lfReading + .001));

oldState = state;

//document.querySelector('#therapy #myC').innerHTML = mfp;

// mfp is used to determine if a user is stepping, along with additional parameters

if(mfp >= 0 && mfp <= 70 && (heelReading > 400 || mfReading > 100)) {

state = 0;

if (!readHeel) {

if (heelReading > 400) {

heelTime = realTime;

readHeel = true;

}

}

if (!readTip) {

if (mfReading > 100) {

tipTime = realTime;

readTip = true;

}

}

} else {

state = 1;

}

if (oldState != state) {

// change in state of foot

if(state === 0){

console.log("New step");

if (readHeel && readTip) {

if (abs(heelTime - tipTime) < 1500 && abs(heelTime - tipTime) > 2) {

steps.push({timeOfHeel: heelTime, timeOfTip: tipTime})

console.log(steps[steps.length - 1]);

}

}

readHeel = false;

readTip = false;

}

}

// for (let i = 0; i < steps.length; i++) {

// console.log(steps[i])

// }

let numSteps = steps.length;

if (numSteps >= 3) {

walkingCorrectly = sketch.correctForm(steps[numSteps - 1]) && sketch.correctForm(steps[numSteps - 2]) && sketch.correctForm(steps[numSteps - 3]);

}

if (!walkingCorrectly) {

if(!mySound.isLooping()) {

mySound.loop();

}

document.querySelector('#therapy #status').innerHTML = 'You are not walking correctly';

document.querySelector('#therapy #status').classList.remove("text-success");

document.querySelector('#therapy #status').classList.add("text-danger");

} else {

mySound.stop();

document.querySelector('#therapy #status').innerHTML = 'Correct gait cycle. Keep it up!';

document.querySelector('#therapy #status').classList.remove("text-danger");

document.querySelector('#therapy #status').classList.add("text-success");

}

//document.querySelector('#therapy #data').innerHTML = latestData;

//document.querySelector('#therapy #data').innerHTML = arduinoOutput;

// colors the circles on the foot drawing, with color intensity mapped to fsr reading

sketch.noStroke();

sketch.fill(3, 252, 32, (heelReading / 1023) \* 255);

sketch.circle(clientW/4.5,clientH/2,heelReading/6);

sketch.fill(246, 250, 12, (mmReading / 1023) \* 255);

sketch.circle(clientW/1.75,clientH/2.75,mmReading/6);

sketch.fill(10, 67, 252, (mfReading / 1023) \* 255);

sketch.circle(clientW/1.4,clientH/2.25,mfReading/6);

sketch.fill(242, 7, 39, (lfReading / 1023) \* 255);

sketch.circle(clientW/1.70,clientH/1.5,lfReading/6);

}

sketch.windowResized = function () {

console.log ("dummyP5 got a window resize");

clientW = parentWidth(document.querySelector('#therapyCanvas'));

sketch.background(footImage);

if(clientW > 700){

clientH = windowHeight \* .5;

sketch.resizeCanvas(clientW, clientH);

}

else{

clientH = windowHeight \* .25;

sketch.resizeCanvas(clientW, clientH);

}

}

sketch.correctForm = function (s) {

if (s.timeOfHeel < s.timeOfTip) {

return true;

}

return false;

}

},'therapyCanvas')

}

function startMotion() {

let m = new p5((sketch) => {

let timeInMotion;

let timeNotInMotion;

let startTime;

let currentTime;

sketch.preload = function () {

timeInMotion = 0;

timeNotInMotion = 0;

startTime = 0;

currentTime = 0;

}

sketch.setup = function () {

currentTime = millis();

sketch.noCanvas();

};

sketch.draw = function () {

if (inActivity) {

startTime = currentTime;

currentTime = millis();

timeInMotion += (currentTime - startTime);

document.querySelector('#motion #status').innerHTML = "In Motion";

document.querySelector('#motion #status').classList.remove("text-secondary");

document.querySelector('#motion #status').classList.add("text-success");

} else {

startTime = currentTime;

currentTime = millis();

timeNotInMotion += (currentTime - startTime);

document.querySelector('#motion #status').innerHTML = "Standing Still";

document.querySelector('#motion #status').classList.add("text-secondary");

document.querySelector('#motion #status').classList.remove("text-success");

}

if (millis() / 1000) {

document.querySelector('#motion #timeInMotion').innerHTML = "Moving for " + String(parseInt(timeInMotion / 1000)) + " seconds total";

document.querySelector('#motion #timeNotInMotion').innerHTML = "Still for " + String(parseInt(timeNotInMotion / 1000)) + " seconds total";

}

}

})

return m;

}

function parentWidth(elem) {

return elem.clientWidth;

}

function serverConnected() {

print("Connected to Server");

}

function gotList(thelist) {

print("List of Serial Ports:");

for (let i = 0; i < thelist.length; i++) {

print(i + " " + thelist[i]);

}

}

function gotOpen() {

print("Serial Port is Open");

}

function gotClose(){

print("Serial Port is Closed");

latestData = "Serial Port is Closed";

}

function gotError(theerror) {

print(theerror);

}

function gotData() {

let currentString = serial.readLine();

trim(currentString);

if (!currentString) return;

//console.log(currentString);

latestData = "Getting Data";

// only update our stored data if the string is valid

if (validString(currentString)) {

arduinoOutput = currentString;

updateValues();

}

}

// returns if the string is formatted correctly

// probably unneeded but added thanks to the values instilled in me by the great Professor John Lillis

function validString(data) {

let splitData = data.split('$');

if (splitData.length != 8) {

console.log("bad string length:" + splitData.length);

return false;

}

for (let i = 0; i < splitData.length; i++) {

if (i < 5) {

if (isNaN(parseInt(splitData[i]))) {

console.log("not an integer: " + splitData[i]);

return false;

}

}

else { // handles accel data

if (isNaN(parseFloat(splitData[i]))) {

console.log("not a float: " + splitData[i]);

return false;

}

}

}

if (parseInt(splitData[4]) != 0 && parseInt(splitData[4]) != 1) {

console.log("not a boolean: " + splitData[4] );

return false;

}

// must be valid

return true;

}

function graphFSRs() {

console.log("Here main scope");

let g1 = new p5((sketch) => {

let plot;

let points = [];

let clientW = parentWidth(document.querySelector('#walkingGraph1'))

let cv;

let timeStarted;

let firstLoop = true;

sketch.setup = function() {

sketch.createCanvas(500, 300);

//sketch.background(3, 252, 32);

plot = new GPlot(sketch);

plot.setPos(0, 0);

//plot.setDim(500, 500);

plot.getTitle().setText("Heel Reading");

plot.getXAxis().getAxisLabel().setText("Time (sec) ");

plot.getYAxis().getAxisLabel().setText("Force");

}

sketch.draw = function() {

if (walkingStarted) {

if (firstLoop) {

timeStarted = millis();

firstLoop = false;

}

document.getElementById('startWalking').disabled = true;

plot.addPoint((millis() - timeStarted)/ 1000, heelReading, "(" + msToTime(millis()) + " , " + str(heelReading) + ")");

plot.setPointColor(color(3, 252, 32));

// Draw the plot

plot.beginDraw();

plot.drawBackground();

plot.drawBox();

plot.drawXAxis();

plot.drawYAxis();

plot.drawTitle();

plot.drawGridLines(GPlot.BOTH);

plot.drawLines();

plot.drawPoints();

plot.endDraw();

}

}

},'walkingGraph1')

let g2 = new p5((sketch) => {

let plot;

let points = [];

let clientW = parentWidth(document.querySelector('#walkingGraph2'))

let cv;

let timeStarted;

let firstLoop = true;

sketch.setup = function() {

sketch.createCanvas(500, 300);

//sketch.background(3, 252, 32);

plot = new GPlot(sketch);

plot.setPos(0, 0);

//plot.setDim(500, 500);

plot.getTitle().setText("Medial Midfoot Reading");

plot.getXAxis().getAxisLabel().setText("Time (sec) ");

plot.getYAxis().getAxisLabel().setText("Force");

}

sketch.draw = function() {

if (walkingStarted) {

if (firstLoop) {

timeStarted = millis();

firstLoop = false;

}

plot.addPoint((millis() - timeStarted)/ 1000, mmReading, "(" + msToTime(millis()) + " , " + str(mmReading) + ")");

plot.setPointColor(color(246, 250, 12));

// Draw the plot

plot.beginDraw();

plot.drawBackground();

plot.drawBox();

plot.drawXAxis();

plot.drawYAxis();

plot.drawTitle();

plot.drawGridLines(GPlot.BOTH);

plot.drawLines();

plot.drawPoints();

plot.endDraw();

}

}

},'walkingGraph2')

let g3 = new p5((sketch) => {

let plot;

let points = [];

let clientW = parentWidth(document.querySelector('#walkingGraph3'))

let cv;

let timeStarted;

let firstLoop = true;

sketch.setup = function() {

sketch.createCanvas(500, 300);

//sketch.background(3, 252, 32);

plot = new GPlot(sketch);

plot.setPos(0, 0);

//plot.setDim(500, 500);

plot.getTitle().setText("Lateral Midfoot Reading");

plot.getXAxis().getAxisLabel().setText("Time (sec) ");

plot.getYAxis().getAxisLabel().setText("Force");

}

sketch.draw = function() {

if (walkingStarted) {

if (firstLoop) {

timeStarted = millis();

firstLoop = false;

}

plot.addPoint((millis() - timeStarted)/ 1000, lfReading, "(" + msToTime(millis()).secs + " , " + str(lfReading) + ")");

plot.setPointColor(color(242, 7, 39));

// Draw the plot

plot.beginDraw();

plot.drawBackground();

plot.drawBox();

plot.drawXAxis();

plot.drawYAxis();

plot.drawTitle();

plot.drawGridLines(GPlot.BOTH);

plot.drawLines();

plot.drawPoints();

plot.endDraw();

}

}

},'walkingGraph3')

let g4 = new p5((sketch) => {

let plot;

let points = [];

let clientW = parentWidth(document.querySelector('#walkingGraph4'))

let cv;

let timeStarted;

let firstLoop = true;

sketch.setup = function() {

sketch.createCanvas(500, 300);

//sketch.background(3, 252, 32);

plot = new GPlot(sketch);

plot.setPos(0, 0);

//plot.setDim(500, 500);

plot.getTitle().setText("Medial Forefoot Reading");

plot.getXAxis().getAxisLabel().setText("Time (sec) ");

plot.getYAxis().getAxisLabel().setText("Force");

}

sketch.draw = function() {

if (walkingStarted) {

if (firstLoop) {

timeStarted = millis();

firstLoop = false;

}

plot.addPoint((millis() - timeStarted)/ 1000, mfReading, "(" + msToTime(millis()).secs + " , " + str(mfReading) + ")");

plot.setPointColor(color(10, 67, 252));

// Draw the plot

plot.beginDraw();

plot.drawBackground();

plot.drawBox();

plot.drawXAxis();

plot.drawYAxis();

plot.drawTitle();

plot.drawGridLines(GPlot.BOTH);

plot.drawLines();

plot.drawPoints();

plot.endDraw();

}

}

},'walkingGraph4')

}

function updateValues() {

let splitData = arduinoOutput.split('$');

heelReading = parseInt(splitData[0]);

mmReading = parseInt(splitData[1]);

lfReading = parseInt(splitData[2]);

mfReading = parseInt(splitData[3]);

inActivity = !!(parseInt(splitData[4]));

xAccel = parseFloat(splitData[5]);

yAccel = parseFloat(splitData[6]);

zAccel = parseFloat(splitData[7]);

}

function draw() {

// Polling method

/\*

if (serial.available() > 0) {

let data = serial.read();

ellipse(50,50,data,data);

}

\*/

}

function msToTime(ms) {

let min = 0;

let sec = 0;

let timeVal = '';

if(ms > 60000){

min = parseInt(ms / 60000);

sec = parseInt((ms - (min \* 60000))/1000);

if(min < 10){

timeVal = '0' + min + ':';

}

else{

timeVal = min + ':';

}

if(sec < 10){

timeVal += '0' + sec;

}

else{

timeVal += sec;

}

}

else{

timeVal = '00:';

sec = parseInt(ms / 1000);

if(sec < 10){

timeVal += '0' + sec;

}

else{

timeVal += sec;

}

}

var time = {

mins: min,

secs: sec,

timeString: timeVal

}

return time;

}