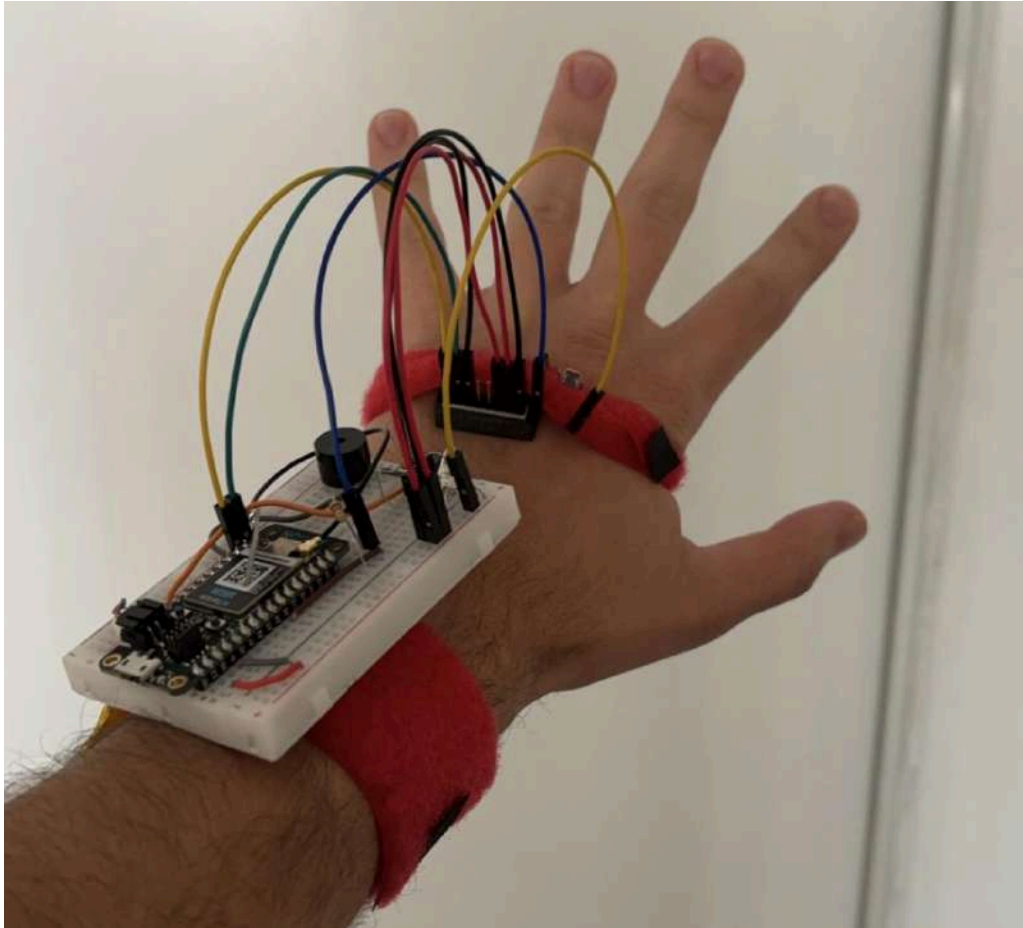


Final Project: Developer Documentation



December 8, 2023

Brad Barakat
bbarakat@usc.edu

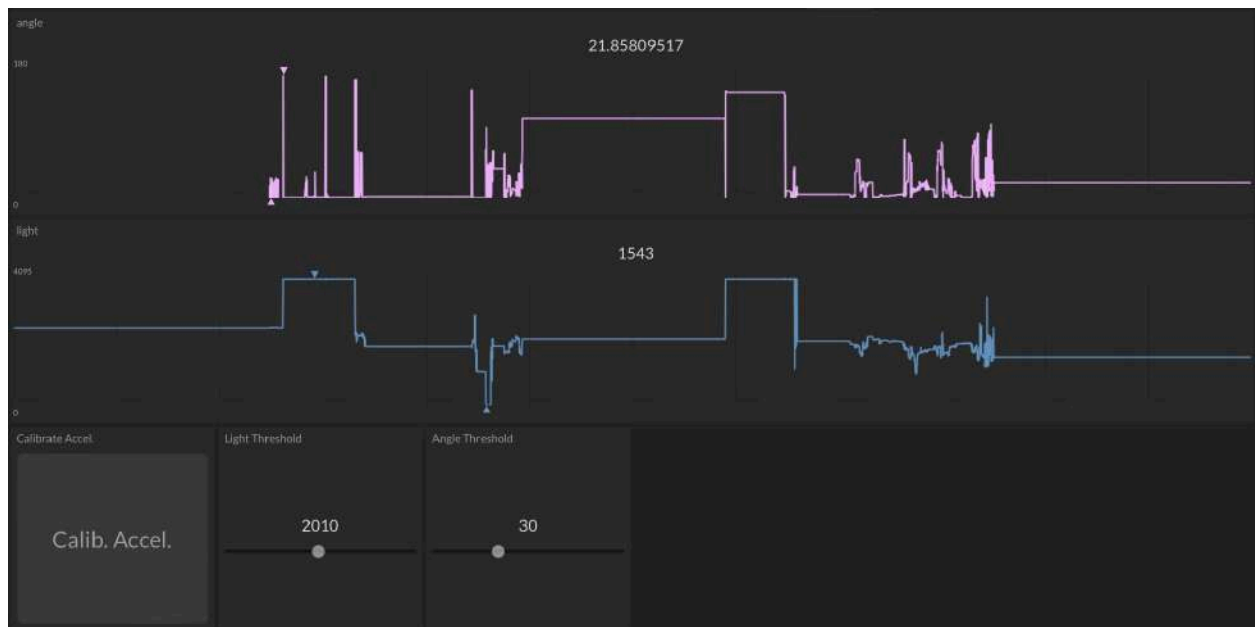
Table of Contents

Table of Contents	2
Overview	3
Wiring Diagram and Components	4
Device Setup Instructions	5
Accelerometer Calibration Instructions	5
SPAVV Setup Instructions	5
Imported Libraries	6
Particle Libraries	6
C++ Libraries	6
Code Functions	6
accel_calib.cpp	6
itp348_project_barakat_brad.cpp	7
Appendix	9

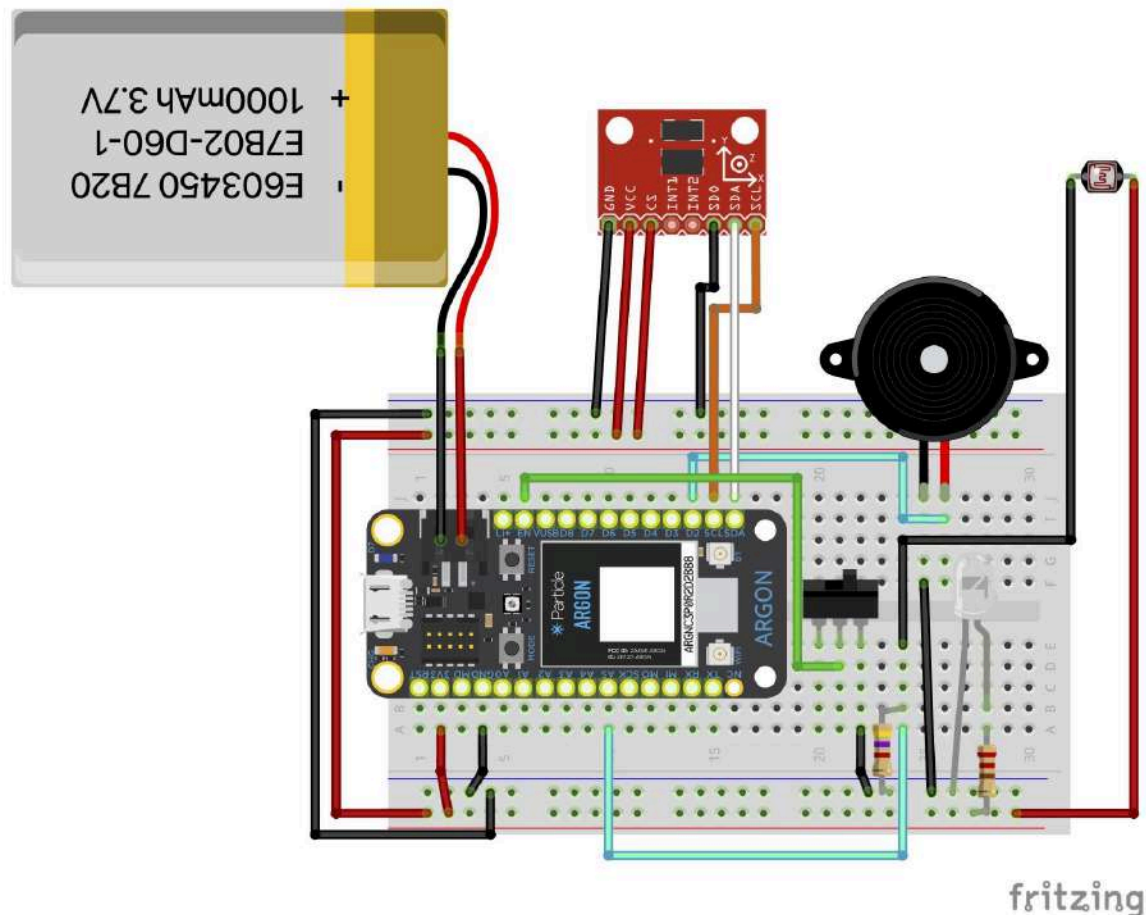
Overview

The Smart Posture Aide for Violinists and Violists (SPAVV) is a wearable device connected to the cloud. It has an accelerometer that goes on the back of your left hand, along with a relative light sensor, and a main breadboard that goes on your left forearm (as shown on the title page image). The breadboard has a speaker, power switch, and an always-on LED. The accelerometer is used to detect changes in wrist angle, the LED and light sensor is used to show the relative proximity between the back of the left hand and the left forearm, and the speaker plays a pitch depending on whether the angle threshold, light threshold, or both are exceeded. The SPAVV also graphs the change in angle and the light sensor output on an Initial State dashboard. The dashboard allows the user to set the angle and light thresholds, as well as calibrate the base orientation of the accelerometer.

Initial State Dashboard Link: <https://go.init.st/uue3as7>



Wiring Diagram and Components



The components in the wiring diagram above may be rearranged as long as it does not affect the connections between them (for example, the accelerometer's 3V and GND wires could connect to the other +/- strip of the breadboard), and the wire colors are meant to differentiate between different purposes (power, ground, clock, etc.). The LiPo battery allows you to use the SPAVV without being tethered to a computer or outlet by a USB cable.

Component	Location	Purpose
3-Axis Accelerometer	The back of the left hand	Output the direction of gravity relative to the back of the left hand
Relative Light Sensor	The back of the left hand	Determine if the back of the left hand is too close to the LED on the forearm (the left wrist is bent too much)

Power Switch	Main breadboard	Turn the device on and off to prevent unnecessary battery loss and cloud data transfers
LED	Main breadboard	Provide constant light for the relative light sensor
Piezo Buzzer	Main breadboard	Play a note with the pitch depending on if the angle, light, or both thresholds are exceeded

Device Setup Instructions

If you are starting with an uncalibrated accelerometer (the gains and offsets are not known), and you wish to find them, the device setup will have two processes: accelerometer calibration and SPAVV setup. Note that the same wiring can be kept for both processes.

Accelerometer Calibration Instructions

1. Flash the accel_calib workspace to the Argon and open the Serial Monitor
 - a. The workspace was configured for deviceOS@5.4.1
2. Open the Initial State dashboard and follow the instructions in the Serial Monitor
3. Save the gains and outputs, and update the src/itp348_project_barakat_brad.cpp file in the itp348_project_barakat_brad workspace

SPAVV Setup Instructions

1. Use velcro to attach the main breadboard to your left forearm, and the accelerometer and light sensor to the back of your left hand
 - a. In the picture on the title page, a 3D-printed piece was used to house the accelerometer
2. Get in proper posture (see Appendix) and then flash the itp348_project_barakat_brad workspace to the Argon
 - a. The workspace was configured for deviceOS@5.4.1
 - b. If you wish to set the current wrist angle to be the new zero, press the “Calib. Accel.” button on the Initial State dashboard and hold the position until the angle becomes close to zero
3. Open the Initial State dashboard to view the graphs and set thresholds

Imported Libraries

Particle Libraries

- ADXL345_Sparkfun_Particle
- ArduinoJson

C++ Libraries

- cmath

Code Functions

Below are the functions of the code made for this project. It is worth noting that the import statements, pin labels, global variables, and enums are at the top of each file, above the subroutines. This document will describe the function's purpose, but the comments in the code also describe the parameters and outputs.

accel_calib.cpp



- `void waitForCalibSignal()`
 - This function waits for Initial State input (this is a blocking function!)
- `void getAxisAccelForCalib(int& axis_accel, const int& dir)`
 - This function finds the raw acceleration along an axis
- `void myHandler(const char *event, const char *data)`
 - This function handles the integration response when data is queried from Initial State
- `void setup()`
 - Starts up serial communication, sets up the ADXL345 accelerometer, and subscribes to the Initial State webhook
- `void loop()`
 - This function has two parts, both of which send instructions through serial:
 - Obtains the gains and offsets for the accelerometer
 - Takes more acceleration measurements after the gains and offsets are found and applied
 - An infinite loop is used for this section to prevent the `loop()` function from returning to the calibration section

itp348_project_barakat_brad.cpp

- `float TwoNumOp(float a, float b, ArrOp op)`
 - This function performs arithmetic with two floats
 - This function may seem useless, but it's needed for the functions below that do arithmetic with arrays of floats
- `void arrayOperator(float* arrA, float Bi, float* arrRes, int resLen, ArrOp op)`
 - This function performs arithmetic with an array of floats and a float
- `void arrayOperator(float* arrA, float* arrB, float* arrRes, int resLen, ArrOp op)`
 - This function performs arithmetic with two arrays of floats
- `float findVectMag2D(float x, float y)`
 - This function finds the magnitude of a 2D vector
- `void measAccel(float* cxyz)`
 - This function finds the acceleration and applies the gains and offsets
- `void testAccel(float* xyz)`
 - This function tests `measAccel()`
- `float findAngle(float* cxyz)`
 - This function finds the angle between the current downward vector and the baseline gravity vector
- `void makeJsonObj(String field, float value, time32_t epoch)`
 - This function makes a JSON object that will be part of the data packet sent to Initial State
- `void appendOrPublishJson()`
 - This function appends or publishes the data packet sent to Initial State, depending on the size
- `int analogReadWithinLims(int pin)`
 - This function reads from an analog pin and makes sure the value is within analog limits (for the Argon, 0-4095)
- `void myHandler(const char *event, const char *data)`

- This function handles the integration response when data is queried from Initial State
- `void setup()`
 - Starts up serial communication, sets up the ADXL345 accelerometer, subscribes to the Initial State webhook, and sets the first calibration epoch (the angle is calibrated right away)
- `void loop()`
 - This function has multiple parts
 - Finds the wrist angle
 - Finds the light sensor output
 - Checks to see if the angle needs to be calibrated
 - If so, it calibrates the angle in a non-blocking way
 - Checks to see if the angle and/or light thresholds were exceeded
 - If so, it makes the buzzer play a note for half a second, with the pitch depending on whether angle, light, or both are exceeded
 - Appends to the JSON and publishes it
 - Reads in data from Initial State

Appendix

Good Wrist Posture	Bad Wrist Posture
 A photograph showing a violinist's left hand holding the neck of a violin. The wrist is in a neutral, straight position, aligned with the forearm. The hand is wearing a red sensor band and a small electronic device with wires attached.	 A photograph showing a violinist's left hand holding the neck of a violin. The wrist is bent significantly backward (hyperextended) relative to the forearm. The hand is wearing the same red sensor band and electronic device as in the first image.