Developing a Wearable Internet of Things (IoT) Platform for Gait Classification

Written by: Kavin Balakrishnan, Arunan Elamaran, Michael Davydov, Ryan Tabrizi, Keshav Vyas

Introduction

Upon introduction to the STMicroelectronics© 32-bit chips and learning the basics of signal processing, machine learning, and data science, our team developed an algorithm to classify various walking motions.

Abstract

We have developed a gait classification system through using MATLAB's graphing analysis features and STMicroelectronics' SensorTile Platform under the guidance of UCLA professor William Kaiser. The system can distinguish stand still, neutral walking, stair ascent, and stair descent as the four primary gaits. The system requires the user to walk a certain distance over a calibration period. The extracted values are utilized in acquiring features from the sensor data. The features for the algorithm will be selected from normalized accelerometer and gyroscope data.

Rig Setup

The "rig" was the most crucial physical aspect. We needed to establish a reliable, consistent platform to hold the ST board firmly in place. The rig was used for both data acquisition (getting data so we can analyze it) and the testing of our algorithm. In addition, to get the best results, a relatively upright position with minimal cushioning was preferable, as to prevent the dampening of any jerks in the motions that could serve as potential features. Additionally, we wanted to ensure that the SensorTile's sensors could pick up the forces and vibrations of gait with a measure of the highest quality for easier processing and feature decision making. Our first model involved placing the system into a fanny pack, and stuffing it with rags and other material to prevent the system from shifting. Later, to increase accuracy, we assembled a rig using an old belt, attached by screws, to a block of styrofoam. We poked holes in this styrofoam block to accommodate the end pins of the SensorTile and its main board. By fitting it in the block and using tape to bind these parts together, we were left with what we considered sturdy and reliable hardware. It ended up lasting us the entire way without failure. Pictures of the rig are shown in figures 2-4.

Data Acquisition

An STMicroelectronics© 32 chip, along with a nucleo board and sensortile, was initially coded to output several values: time count, the gyroscope x values, gyroscope y values, gyroscope z values, accelerometer x values, accelerometer y values, and accelerometer z values. The team chose to focus on classifying 3 main motions: normal walk, stair ascent, and stair descent.

To record the data, we used the program Putty, which served as our console to system workbench and converted values into a CSV format as well as a modified program from the "E96C: Internet of Things" course from UCLA

that monitored gyroscope and accelerometer values on the SensorTile. To ensure that the graphs were both precise and accurate, the tester wore the rig tightly around their waist and performed the motions evenly without excessive variance (analyzing variance is needed to make a robust program that can account for it). To make the graphs easily comparable however, the same number of steps would be performed for each motion for a similar time interval.

The first set of training data that was taken did not display any obvious indications that could be used for data analysis. This data was too coarse and the sampling rate was changed from 100 to 50. Another set of data was taken using a camera and with a new sampling rate in hopes of being able to match the stride with the plotted waveforms. We were then able to conclude that the peaks in the waveforms occurred when the feet contacted the ground. Through several more trials, we adjusted the baud rate down to 5. A new tester was designated to collect 3 sets of each, normal walk, stair ascent, and stair descent. This resulted in the high quality data being eventually used for analysis.

After acquiring the data from the SensorTile, we uploaded the csv format to an excel spreadsheet for normalization. For each data set (acceleration x, y, and z, and gyroscope x, y, z) we determined the mean of the absolute values of the set and divided the original values by that mean to obtain the normalized values. Using the normalized values, we found that the waveform pattern was more distinguished and it was easier to identify features within the waveforms.

MatLab Graph Analysis

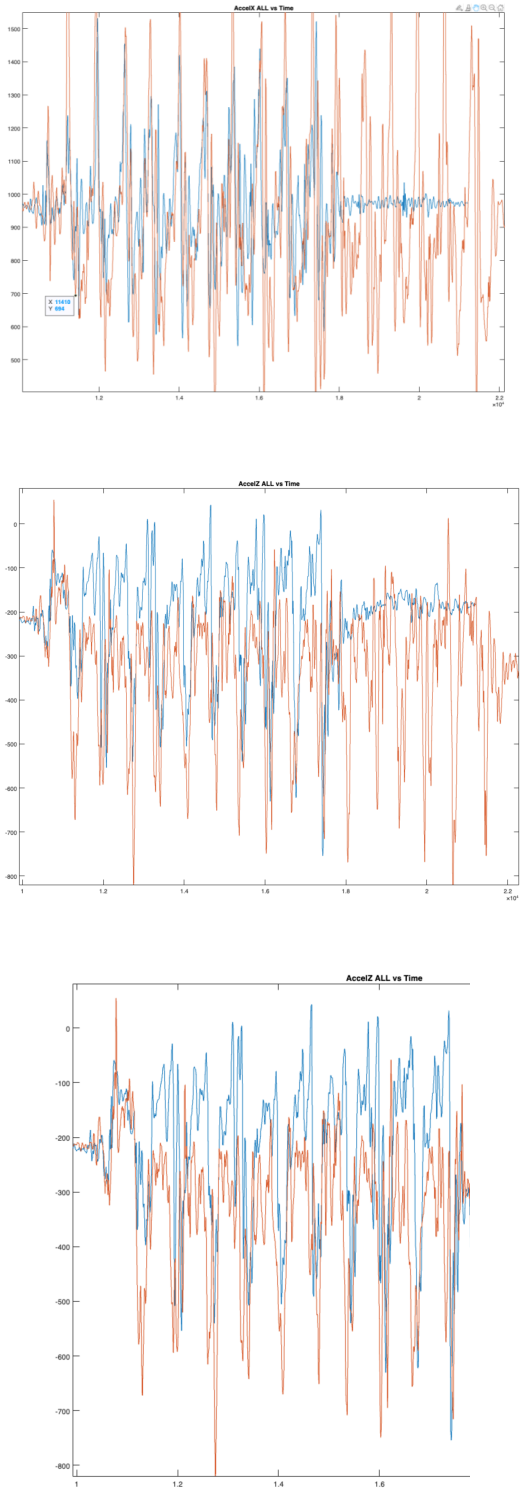

Figure 1: Each graph plots the values of our top 3 features along with the best data for ascending and descending we could find.

Once the three gaits were obtained, we created graphs to qualitatively and quantitatively compare the gaits so as to identify potential features. We analyzed features on the respective graphs such as the average, peak height, the average troph height, the overall average value, and the amplitude of the waveform. To determine which features we would use in our classification model, our goal was to find the feature that had the most distinct differences among different graphs. It is was then discussed that it would be easier to compare all the sensor values for mathematical relationships if:

1. Each set of data was matched to have a similar start and end point as the "normal walk" data set for a better comparison of the waveform by time.
2. The means of each of the 6 sensor values was taken across the "normal walk value". The means were then used to divide each of the other sensor values in order to make all data relative to normal walk.

This would still retain the shape of each graph but allow for better comparisons to be made such as with acceleration and gyroscope values. We decided against using time as the feature, because we found that we could find patterns in the graph without having to use the time interval. The most robust feature that we found was the means of mean peak value. In other words, we found multiple mean peak values within a set, over a set time interval, and then calculate the mean of those means. The other feature we found was the vertical translation of the waveform graphs. We found that the waveform graphs for ascent vs normal walk and descent vs normal walk for acceleration z was simply a vertical translation, either up or down, from the normal walk graph.

Choosing a Dynamic Decision Tree Over a Neural Network

Most importantly, we were able to determine that a significant amount of the motion classification can be done through a decision tree structure. Such a structure can differentiate between level or non-level walk. Features we have derived through extensive testing include finding the means of mean peak value matrices.

We decided to utilize a dynamic decision tree model for our classification algorithm. This differs from a typical decision tree as it will have values that can be adapted to a specific person based on a training phase in which data is acquired. The main similarity though is that it will classify the motions based on predetermined true or false conditions rather than nodes and error like a neural network. Initially, we decided that we were going to use a neural network with 3 distinct features obtained from the data. However, when analyzing the data we found direct relationships between the behaviors of the graphs and the specific motions. When testing both the neural network model with the two features we had found and the dynamic decision tree that used the peaks in the graphs as its inputs, we found that the dynamic decision tree model was more robust and gave a higher classification accuracy than the neural network. Additionally, by using the dynamic decision tree, we can use values extracted from data collected during a training period rather than just hard coded (constant) values.

Setting up the Algorithm Program

Along with using MatLab, we also decided to write analysis programs to help us find features that we could not easily spot or determine. We chose to write our code in C using Xcode. We developed the code to better suit our needs and more efficiently find features with less work needed from our part. At first, we had to use CSVs which we had already normalized on Google

Sheets (as we did when using MatLab). However, we soon developed the system to perform the normalization itself and to need no preprocessed input except the raw CSVs which were recorded during training.

Upon finding potential features, we then decided to first test them using an algorithm on Xcode. We made sure to create the algorithm such that it would exactly mirror what we would create on SystemWorkbench. The only difference with the Xcode algorithm was that it would receive pre-recorded data stored in files while the SystemWorkbench algorithm would record its own data before analyzing it. Also when developing the Xcode version of the algorithm, we made sure to write the code in a fashion that we could easily make it not only readable, but also modifiable.

To write our code on SystemWorkbench, we just translated what we wrote in Xcode. We also wrote the necessary terminal print and LED code in order to ensure that the system is user-friendly. When initial problems arose when transitioning between the two systems (from Xcode to SystemWorkbench), we would isolate the problem in SystemWorkbench and then see how it was implemented in our Xcode algorithm. Then, based on the similarities and differences between the two systems, we would determine the proper fix.

When transferring the code from Xcode to SystemWorkbench, an error arose in terms of storage: while transitioning from feature finding to algorithm writing, although we removed the axes we did not need, the code setup required that we save the arrays of each axis prior to its analysis. This meant storing several arrays with thousands of elements. To solve this problem, we decided to immediately analyze the collected data rather than storing it. This way, we could reuse just one two dimensional array for each motion rather than having multiple to store the data of the multiple motions.

Motion Classification Algorithm Testing and Testing Method For Extremes

To test the motions, we would first perform the training period and then multiple sets of the three motions. The first version of the dynamic decision tree had an extremely low tolerance. It was only able to correctly classify motions if they were done almost exactly like the set values recorded during the transition period, meaning the feature did not have enough room to account for variation. After testing the system and reviewing the results, we determined that in the following iterations we would need to implement functions that handled edge cases. We invented a “smooth or rough” system that dealt with two types of extremes in our gait. Testing procedures would happen like normal, but the motion would change. Smooth gait was regular, slow, and evenly timed. Our decision tree should be able to recognize these motions to a reasonable degree of certainty as this was in the first iteration. Rough was fast and unevenly timed. Since our system is based on comparing data to a “normal,” uneven movements give data that is too noisy to process. At the moment, we are yet to write a robust enough noise filtering algorithm, or find one that suits our needs. However, that version of the system often failed to classify motion properly regardless of it being performed smoothly or roughly. We were getting consistently incorrect results with both. Out of our three gait classifications—walking, ascent, and descent—walking was getting correctly classified almost all of the time. This told us that our job for classifying walking was largely done, but we needed to find more robust features that classified between ascent and descent.

Improvements Made to the Algorithm

After testing the system through the "rough and smooth method," we noticed that although Normal Walk was properly being classified, the system was consistently being inaccurate in differentiating between Stair Ascent and Stair Descent. After further testing, we determined that we would have to replace the classifying feature for Ascent and Descent.

Rather than solely relying on the AX feature, we created a new feature that is based on the raw AZ values (not normalized). We were able to determine that we could accurately classify the New Motion as Ascent or Descent depending on which way the AZ values leaned (if the average was closer to that of Ascent, we would classify the motion as Ascent.)

SystemWorkbench Problems

While engaging in the testing and refinement of our decision tree classifier, we experienced software-breaking System Workbench errors. This put a halt to our progress for a short period, for the frequency of these errors and their severity was entirely random. Many times, System Workbench needed to be restarted or even completely reinstalled. Still, the errors would persist. In addition, the error messages were always cryptic and vague, which made it impossible to debug. We even considered moving the entire project to the new CUBE IDE at one point, as we suspected that System Workbench was simply showing its age and little could be done for our problem. We were able to avoid this issue by running System Workbench on a completely different device on a different network. We ended the rest of the testing and refinement of our final product on this new device.

Accuracy/Results

As mentioned earlier in the testing section, the algorithm was tested for smooth (regular) walking motions and for rough motions. Smooth motion is best identifiable as motion at a regular, smooth pace, whereas "rough" motion is at a faster pace than normal walking and harder jolts.

Our system works best with 90+% accuracy when the motions performed are smooth. When rough motion is performed, our system correctly classifies the motion with 75% accuracy.

How to Use the System

First attach the belt to your waist such that the wires coming out of the STM board towards your right side. (see picture below) Center the STM board on the middle of your waist. Enter the System Workbench IDE with the imported program files. Go to Project, then Clean. Clean all Projects. Then enter the main folder and click on DataLog to highlight it Then go to Project → Build Project. Then go to Run→ Debug As and select the one with the blue icon on the left and says "Ac6 STM32 C/C++ Application". Then run the program by pressing the green play button on the debug page. When the orange LED on the blue SensorTile board turns on, perform a normal walking motion (walking on level ground). Stop when the light turns off and prepare for the next motion, stair ascent. When the light turns on again, start walking up stairs at a walking pace. Once the light turns off, prepare for the next motion, stair descent. When the light turns on, walk down the stairs until the light turns off. At this point, the training period is now complete. The user can now perform whichever motion from above, along with standing still, during each period. After each period, the program will classify the motion that was performed and display it on the terminal screen, and also through blinks from the LED. 1, 2, 3, and 4 blinks correlate with stand still, normal walk, stair ascent, and stair descent respectively.

When the user wishes to stop the system, they can simply press the red stop button, or perform 3 stand stills in a row.

Further Improvement

The algorithm code is set up such that changes can easily be made. One method through which we ensure this is by making use of the "#define" feature. By doing this, one experimenting with this code can simply change a value for a specified property of the algorithm at the line of definition. This has the effect of a variable, but does not take up the storage space that a variable would.

The features we found can be further developed into input nodes for a neural network. This way, rather than relying on true or false conditions, the code can weigh the strength of the features to classify the motion, thus making the system significantly more robust. Additionally, other features can be explored, such as vertical displacement and periodic gait motion without the reliance of time. Through further development, this system can detect far more than different gaits and push the boundaries of motion detection and measurement.

Links to the Githubs:
To see the all the development code used to make the algorithm:
https://github.com/rtabrizi/STMotionExploration

To see the actual algorithm code itself and also the data acquisition code:
https://github.com/ArunanElamaran/STM_Walk_Motion_Classification

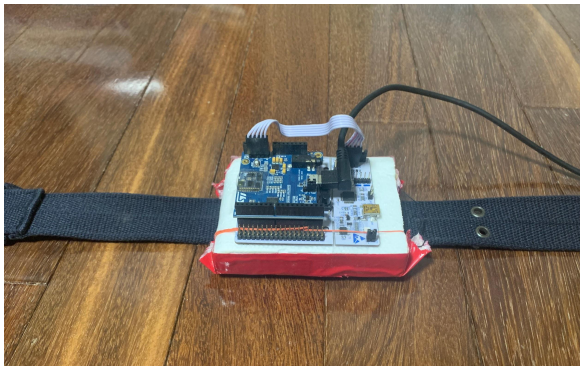

Figure 4: This features the orientation of how we attached the ST board to the belt.

.

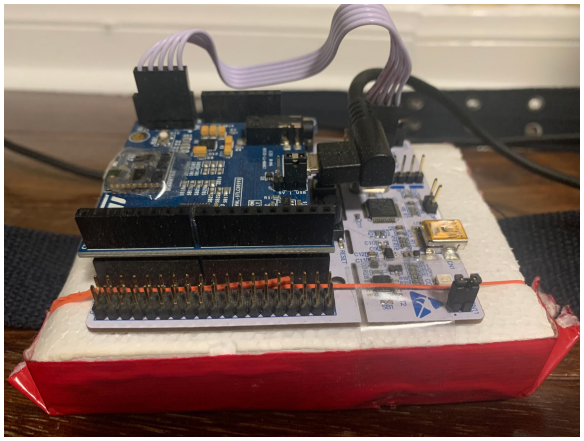

Figure 2: This photo shows more closely the way that we got the board to stay firmly in place even while motions were being performed.

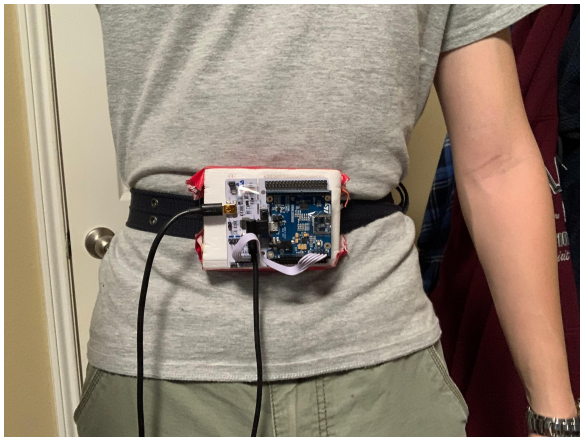

Figure 3: We placed the rig in the center of our front side on our hips.