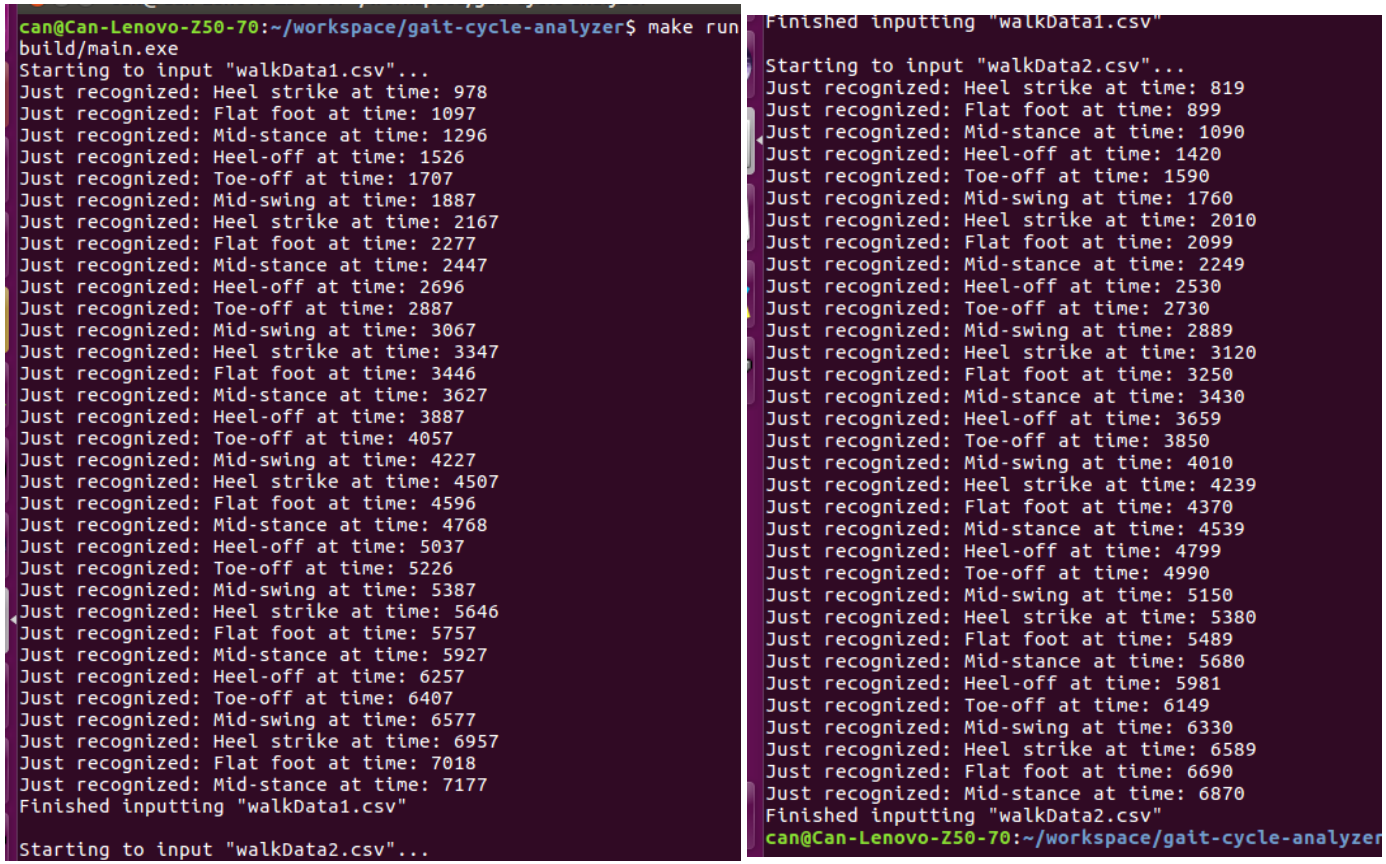
Software/Controls Design Challenge

Submission by: Can Cui

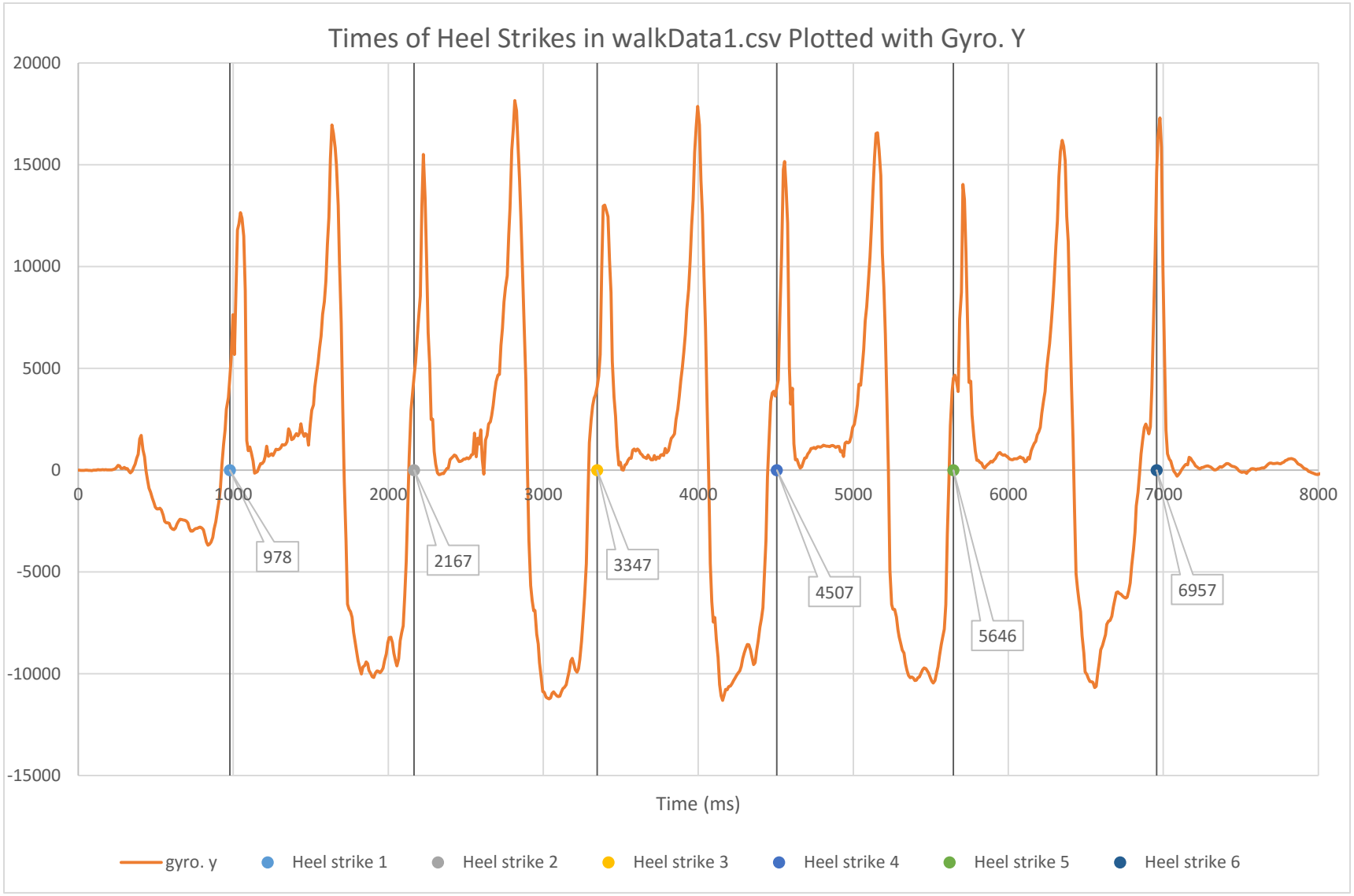
Overview

My algorithm for this challenge identifies the beginning of each gait cycle (heel strike) as well as every other phase (foot flat, mid-stance, heel-off, toe-off, and mid-swing) in real time. It works fully for both the given datasets. The algorithm is implemented using C++.

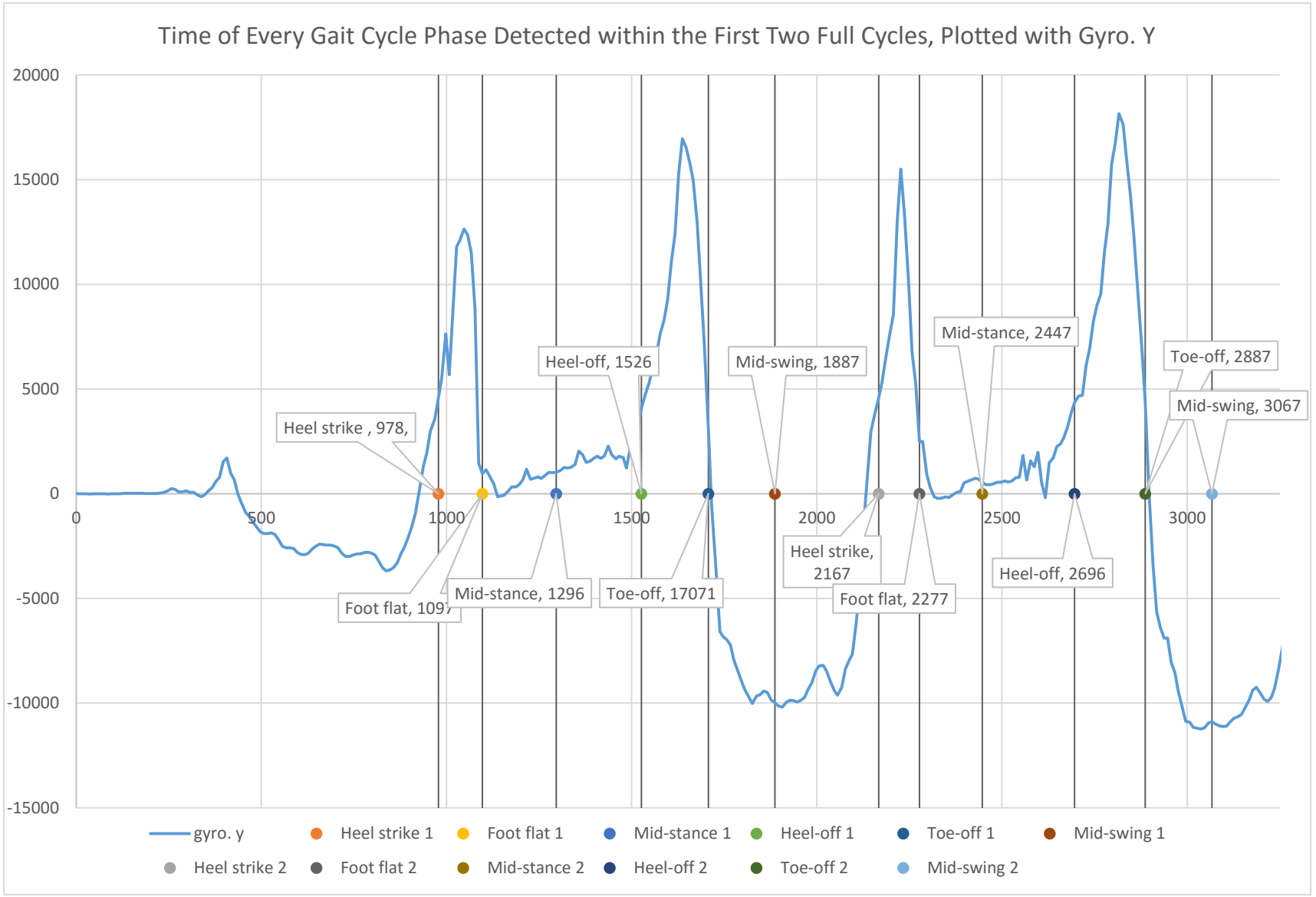
This is the output for walkData1.csv (left) and walkData2.csv (right):



This first graph indicates where the heel strikes were detected in walkData1.csv:

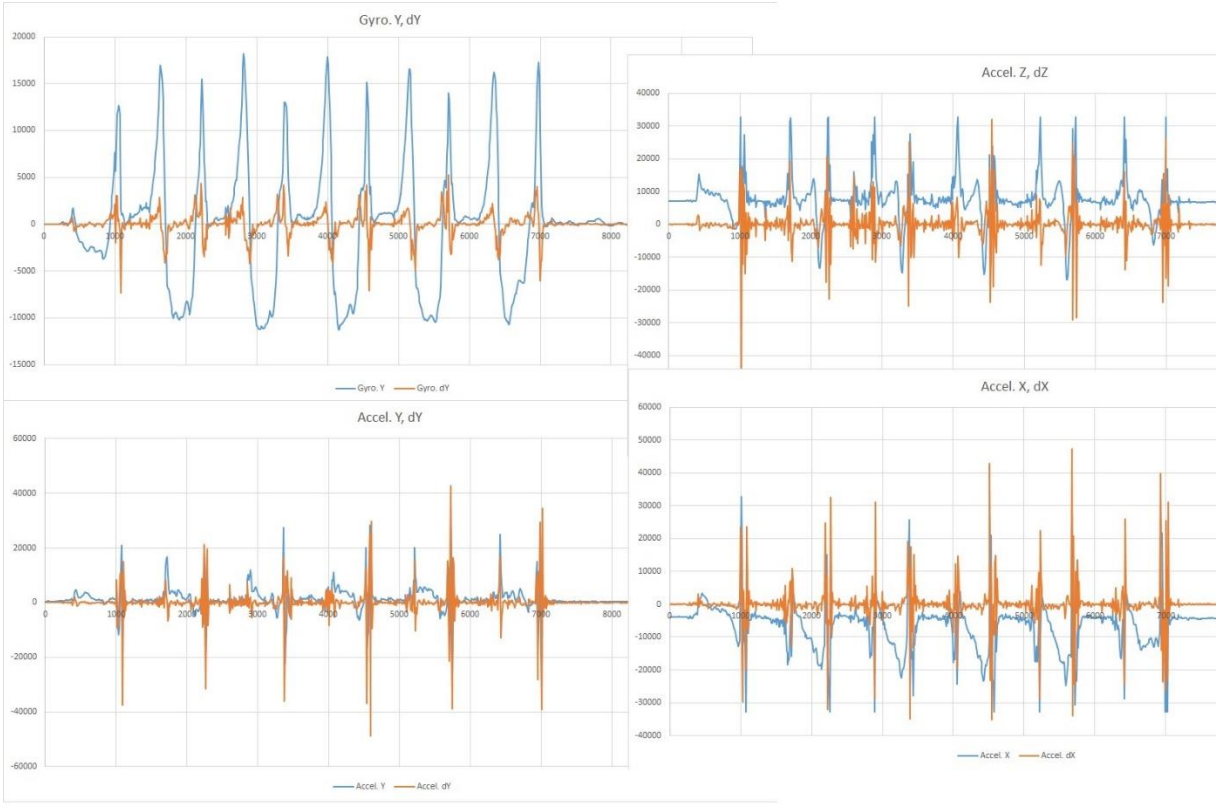


This next graph indicates where every phase was detected in the first two full cycles of walkData1.csv.



Approach

My first step was to figure out where the phases of the gait cycle occurred in the given data. I decided to visually examine the data from accel. X, accel. Y, accel, Z, and gyro. Y from both datasets. I did this by creating graphs of these data as well as their delta values over time. The picture on the right is one set of graphs I examined. Blue represents the signal and orange represents their delta values.



I then looked at a diagram of the phases of the gait cycle, watched a slow motion YouTube video of the gait cycle, and walked around in my apartment while watching my feet. While I was doing these, I also imagined the acceleration and angular acceleration that was occurring in the location the IMU was placed, and tried to correlate these with the features in my graphs. My conclusion about where the phases of the gait cycle were located in the data is reflected by the graph of where my algorithm identified the phases, on page 3. The following is a quick explanation for how I determined which features corresponded to each phase:

Heel strike

Heel strike has to be proceeded by negative angular acceleration around the y-axis because of the tip of the foot rising up near the end of mid-swing. It is accompanied by a spike in acceleration on the z-axis from the impact.

Foot flat

Foot flat must come after heel strike. It is proceeded by a positive spike in angular acceleration around the y-axis from the tip of the foot moving down until it hits the ground.

Mid-stance

Mid-stance must be proceeded by foot flat. During this phase, there is little acceleration on any axis since the foot is planted.

Heel-off

Heel-off must be proceeded by mid-stance. There is increasing angular acceleration about the y-axis from the heel lifting up while the front of the foot is still touching the ground.

Toe-off

Toe-off must be proceeded by heel-off. Angular acceleration about the y-axis approaches zero since the heel is no longer lifting with respect to the front of the foot, as the foot is about to swing forwards.

Mid-swing

Mid-swing must be proceeded by toe-off. There is a negative dip in acceleration along the x-axis as the foot swings forwards. There is also a negative dip in angular acceleration about the y-axis as the foot transitions from toe-off to heel strike.

I isolated gyro. Y and accel. Y as two signals which together contained all the information needed to identify all the phases of the gait cycle. Note that the features my algorithm looks for may not be the same as the features I just described for each phase, and may even seem trivial sometimes. This is because my algorithm looks instead for more distinct features in the signal that occur at the same time as the features I described. Using more distinct signals reduces the chances of false results.

At a high level, my algorithm works as follows:

1. Intake data points in real time
    a. This is done by calling a function and passing the data points as the arguments. Additionally, a struct for gait phase information is passed by reference so that the function can return information on the gait cycle phase when a new phase is detected
2. Identify features such as spikes, troughs, and areas of little activity in the data
    a. This capability is encapsulated by a class called *Feature_extractor*
3. Every time a new feature is identified, all the recently-identified features are compared against the features required to advance to the next state (next gait cycle phase)
    a. This is encapsulated in a class called *Gait_cycle_classifier* , which contains instances of *Feature_extractor* (one for each signal)
    b. *Gait_cycle_classifier* uses a state machine to organize and store the criteria required to advance from one phase to the next

    c. For example, the features required to move from heel strike to foot flat are (1) a positive edge through a high threshold in the gyro. Z signal within the last 150ms and (2) a negative edge through a high threshold in the gyro. Z signal within the last 30ms
    d. When the current state advances to the next, the criteria to advance to the next stage is updated by the state machine
4. When a new phase is detected, the information regarding what phase it is and when it was detected are passed to the caller

I've included my source code with annotations in this package. The algorithm is contained within gait_cycle_classifier.cpp/.hpp and feature_extractor.cpp/.hpp. The main.cpp file is used purely to demonstrate the functionality of my algorithm. I've also included the makefile so it's easy to compile and test my algorithm ("make" to compile, "make run" to run). Note that to run the program, both walkData1.csv and walkData2.csv must be in a folder in the working directory called "datasets". The executable file was built on a Ubuntu machine.

Technical Highlights

- All functions use fixed-point arithmetic and frequently-called functions have constant time complexity. There are no loops in a standard function call to input data
- The algorithm is encapsulated inside a class, so it's less messy to implement
- It's easy to add additional phases or criteria for each phase because of the use of a state machine
- The algorithm identifies all 6 phases of the gait cycle in both walkData1.csv and walkData2.csv using the same settings, despite the features being slightly different

Notes

You may have noticed that the *Feature_extractor* class is capable of recognizing many features that aren't used to determine the gait cycle phases. Although feature detection still runs with constant time complexity, it is slightly slower because of the extra features it is monitoring for. I left it this way for two reasons:

1. The class can now be reused for many other applications where features need to be extracted from signals, like EMG analysis
2. I wasn't sure initially how many features and what type of features I would need to use to reliably recognize each phase of the gait cycle. I prepared for the worst-case scenario and assumed I would need to use many types of features. This way, I wouldn't have to go back, modify, and retest existing code.

Libraries Used

- I used an open-source CSV parser (in references) so I could quickly focus on the actual algorithm instead of parsing a .csv file.

- I used the std::vector container to store recently-identified features because its functionality was the closest to what I needed. Again, I used this instead of writing my own data structure so I could focus on the algorithm. Given more time, I would have crafted a custom data structure which suits my application exactly.
- I used std::string to provide human-readable strings for each phase. This would not be necessary in an embedded application.

References

- http://www.physio-pedia.com/Gait
- https://sourceforge.net/p/cccsvparser/wiki/Home/
- https://www.youtube.com/watch?v=5j4YRHf6Iyo