List of Data Manipulation Methods and Algorithms Used in the Bike Accelerometer Project

by Giancarlo Zaniolo

Modifying the Raw Data to Produce Different Graphs

This section covers all of the ways that the raw data was manipulated to display different representations of the actions performed in each experiment.

Total Vector Magnitude

```
netExperimentAllDirections = sqrt(xExperiment.^2 + yExperiment.^2 + zExperiment.^2);
```

```
Unrecognized function or variable 'xExperiment'.
```

First seen in: Experiment 1

Description: This value shows the magnitude of the net vector of the measured x, y, and z vectors.

Utility: This graph is able to preserve the choppiness of the graphs, if that is that you are looking for, but it cannot tell you which direction the force is in, and is therefore not very useful when developing algorithms to interperet patterns in the data.

Polar Form Graphing

Rho

```
rhoExperiment = sqrt(xExperiment.^2 + yExperiment.^2 + zExperiment.^2);
```

First seen in: Experiment 1

Description: This is the same as the total vector magnitude.

Utility: This is the same as the total vector magnitude.

Theta

```
thetaExperiment = atand(xExperiment./zExperiment);
```

First seen in: Experiment 1

Description: This value shows the theta value if the net vector were to be plotted in polar spherical form.

Utility: Compared to the vertical acceleration forces, the horizontal acceleration forces are almost nothing, and because the plane where the theta value lies on is parallel to the ground, most of the data obtained from this means nothing. The only case where this could potentially be used is if the bike were to be set down on its side.

Phi

```
phiExperiment = acosd(yExperiment./rhoExperiment);
```

First seen in: Experiment 1

Description: This value shows the phi value if the net vector were to be plotted in polar spherical form.

Utility: This value's main use is that it can show you more or less what angle the phone is tilted at before using the y/z correction algorithm, but it does have its limitations that prevent it from being a good measurement for data analysis. Since the phi plane is always pretty much perpendicular to the ground, it is very hard to see when there is a stronger downward force than gravity. Also, because this is a measure of the angle, and not of the magnitude of the force, when the bike is in free-fall and the forces are very small, the phi value just sorta goes crazy.

Data Correction Algorithms

Y/Z Tilt Correction

```
yComponentCorrectedYBunnyhop = yBunnyhop * cosd(phoneAngle);
zComponentCorrectedYBunnyhop = zBunnyhop * cosd(90 - phoneAngle);

correctedYBunnyhop = yComponentCorrectedYBunnyhop + zComponentCorrectedYBunnyhop;

yComponentCorrectedZBunnyhop = yBunnyhop * cosd(90 - phoneAngle);
zComponentCorrectedZBunnyhop = zBunnyhop * cosd(phoneAngle);

correctedZBunnyhop = yComponentCorrectedZBunnyhop + zComponentCorrectedZBunnyhop;
```

First seen in: Experiment 2

Description: This algorithm corrects for any tilt affecting the y and z axes if provided the angle at which the measurement device is tilted.

Utility: This algorithm will be especially useful when mounting the measurement device to a surface that is not perpendicular to the ground, since it allows the y-value to be kept perpendicular to the ground. This allows the corrected y-axis value, which has both positive and negative values, to be used instead doing a vector magnitude calculation, which provides the magnitude of the force, but no direction.

Bump Interference Correction

Matlab's Built-in Smooth Function

```
smoothedExperiment = smooth(yExperiment);
```

First seen in: Experiment 2

Description: This is Matlab's built-in algorithm for smoothing out data.

Utility: This function does a good job at smoothing out the data, but it still leaves a little bit of interference, no matter how many times you loop the data through. It does a good job at keeping the height of the peaks even after multiple smoothings of the data. This would be a good function to use if you want to measure what forces the bike was more realistically feeling during the jump.

Smoothing Data Using an Average of Nearby Values

```
numberOfSmoothings = 100;
numberOnEachSide = 1;
```

```
valueAvgArray = zeros(length(yExperiment), 1);
valueAvgArray2 = yExperiment;

for k = 1 : numberOfSmoothings
    for i = 1 : numberOnEachSide
    valueAvg = 0;
    for j = (-i + 1) : numberOnEachSide
        valueAvg = valueAvg + (valueAvgArray2(i + j));
    end
    valueAvg = valueAvg / (numberOnEachSide + i);
    valueAvgArray(i) = valueAvg;
    end
    for i = numberOnEachSide + 1: (length(valueAvgArray2) - numberOnEachSide)
        valueAvg = 0;
        for j = -numberOnEachSide : numberOnEachSide
            valueAvg = valueAvg + (valueAvgArray2(i + j));
        end
        valueAvg = valueAvg / ((2 * numberOnEachSide) + 1);
        valueAvgArray(i) = valueAvg;
    end
    for i = (numberOnEachSide - 1) : -1: 0
        valueAvg = 0;
        for j = -numberOnEachSide : i
            valueAvg = valueAvg + (valueAvgArray2(length(valueAvgArray2) - i + j));
        end
        valueAvg = valueAvg / (numberOnEachSide + i + 1);
        valueAvgArray(length(valueAvgArray2) - i) = valueAvg;
    end
    valueAvgArray2 = valueAvgArray;
end
```

First seen in: Experiment 2

Description: This smoothing algorithm takes the average of the x points to the right the central point, the central point, and the x points to the right of the central point, and can be repeated as many times as needed.

Utility: This function does a good job at smoothing out the data, and the more you loop re-smooth the data, the more smooth it becomes. However, the more times you smooth it, the more rounded and less prominent the peaks become. This would be a good function to use if you want to make sure there is minimal interference, but the general shape of the jump is still there.

Data Analysis Algorithms

Jump Detection Algorithms

Jump Detection Algorithm 1 (Two Largest Peaks)

```
lowestValue = 0;
lowestValueIndex = 0;
secondLowestValue = 0;
secondLowestValueIndex = 0;
for i = 2: (length(yBunnyhop)-1)
    %detects if it is a peak
    if yBunnyhop(i) < yBunnyhop(i-1)
```

```
        if yBunnyhop(i) < yBunnyhop(i+1)
            %checks to see if it qualifies for one of the lowest value
            %slots
            if yBunnyhop(i) < secondLowestValue
                if yBunnyhop(i) < lowestValue
                    %checs to see if new lowest value is at least 25 100ths
                    %of a second away from old lowest, if it is, old lowest
                    %replaces old second lowest
                    %makes sure lowest is the lowest within a 50 100ths of
                    %a second range
                    if abs(i - lowestValueIndex) > 25
                        secondLowestValue = lowestValue;
                        secondLowestValueIndex = lowestValueIndex;
                    end
                    lowestValue = yBunnyhop(i);
                    lowestValueIndex = i;
                else
                    if yBunnyhop(i) < secondLowestValue
                        %only sets new second lowest value if it is far
                        %enough from lowest
                        if abs(i - lowestValueIndex) > 25
                            secondLowestValue = yBunnyhop(i);
                            secondLowestValueIndex = i;
                        end
                    end
                end
            end
        end
    end
end
```

First seen in: Experiment 2

Description: This jump detection algorithm finds the takeoff and landing of the jump by finding the the lowest and second lowest acceleration peaks.

Utility: This algorithm is only useful in super specific scenarios: The takeoff and landing have to be the largest peaks, meaning that drops will not be visible, and the flat ground has to be smooth enough for the second largest peak to be the takeoff. Aditionally, there can only be 1 jump. It is unlikely that this exact algorithm will be used in the future.

Jump Detection Algorithm 2 (Disturbance-Based)

```
airCounter = 0;
modeArray = zeros(length(yPlasticRamp), 1);
%you are most likely still when you start the test, and this starts off the chain reaction wher
modeArray(1) = 1;
threshold = 16;

for i = 2: (length(yPlasticRamp) - 5)
    sumOfPoints = 0;
    for j = 0: 4
        sumOfPoints = sumOfPoints + abs(yPlasticRamp(i + j) - yPlasticRamp(i + j + 1));
    end
```

```
    %if it passes the threshold (moving on bumpy enough ground)
    if sumOfPoints > threshold
        %if in "air" for less than 0.25 secs before jump lands, read over
        %all "air"s with 2s
        if airCounter < 25
            for j = i: -1: (i - airCounter)
                modeArray(j) = 2;
            end
        end
        modeArray(i) = 2;
        airCounter = 0;
        %if it doesnt pass the threshold (either still or in air)
    elseif sumOfPoints < threshold
            %if its stopped it stays stopped
            if modeArray(i - 1) == 1 || length(yPlasticRamp) - i < 300
                modeArray(i) = 1;
            %if its still for too long it counts it as stopped, reads over
            %all "air"s with 1s
            elseif airCounter >= 300
                for j = i: -1: (i - airCounter)
                    modeArray(j) = 1;
                end
            else
                modeArray(i) = 3;
            end
            airCounter = airCounter + 1;
    end
end
```

First seen in: Experiment 2

Description: This jump detection algorithm defines the time in the air based on if there is a lot of shaking on the graph, caused by bumps on the ground.

Utility: This algorithm is also only useful in specific scenarios, but it has one key advantage over algorithm 1: it works even with multiple jumps. However, it has the disadvantage that it only works if the bike is bumping around when it is moving, which means that it will likely not work as well on smooth surfaces like skateparks, or when the measurement device is mounted to a bike with suspension. Also, this algorithm will not work well on bump interference-corrected data, so it is unlikely that this will be used outside of very specific scenarios.