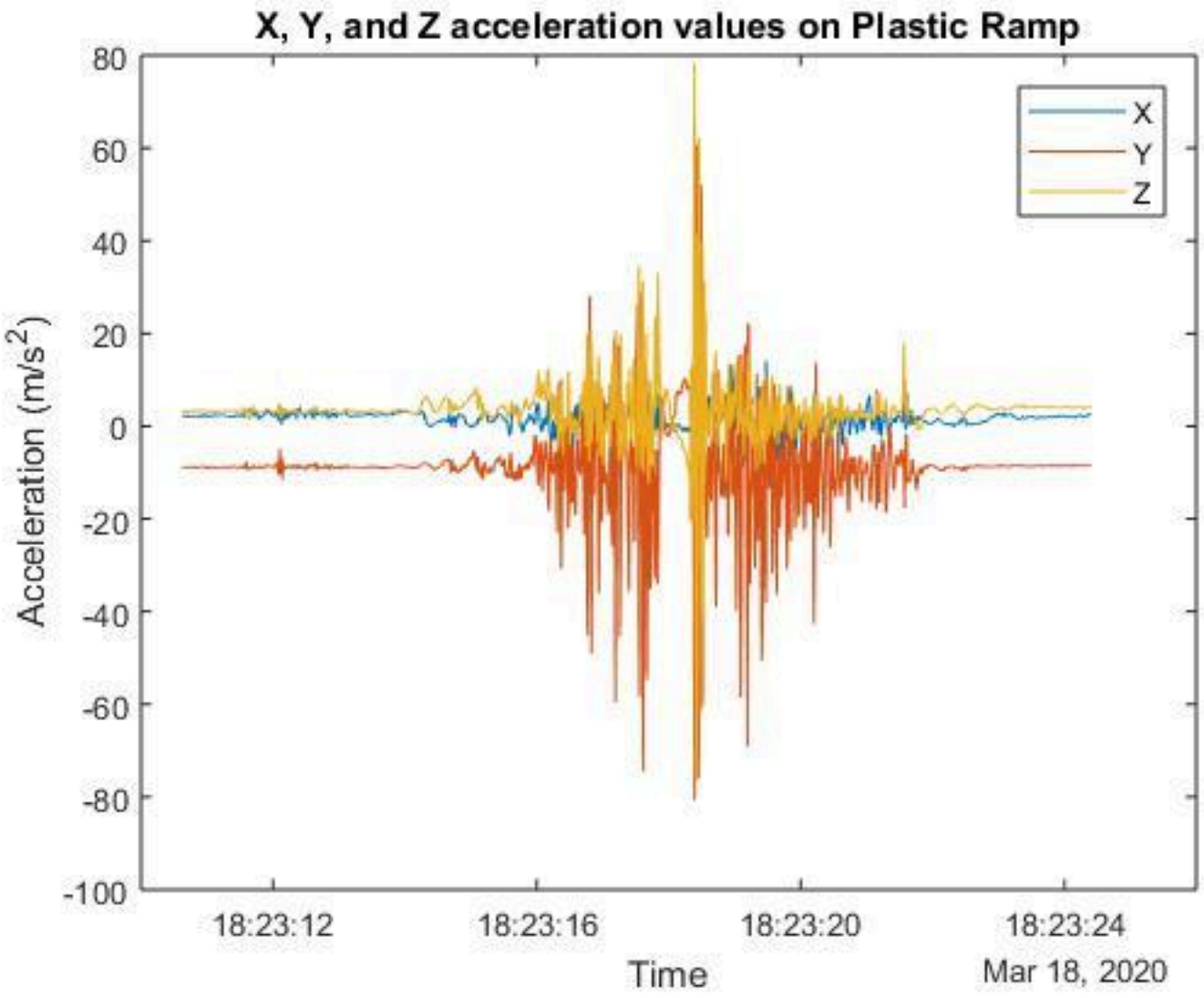


Can you see where the jump happens?

Recording Changes in Acceleration during Bunnyhops and Small Jumps

by Giancarlo Zaniolo

This is the second installment in a series of tests in which I use an accelerometer to collect data while performing various actions on my bike(s). The tests below were done to attempt to see if there are any noticeable patterns when going over small jumps.

Experiment:

For this experiment, I will be using Matlab Mobile to record accelerometer data while bunnyhopping (a completely self-propelled jump) and riding over a very small plastic jump.

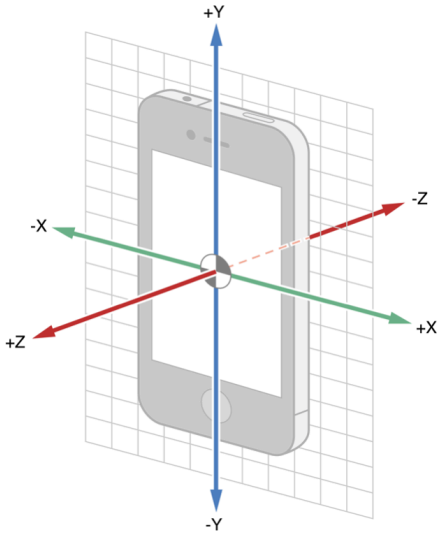
While much of the programming will be the same as that done in the first experiment, I will still explain the steps taken to get each graph, so feel free to pay more attention to the analysis.

Setup:

The setup from this experiment is the same as the seup from the first experiment.

```
imshow(imrotate(imread('IMG_2560.jpg'), 270))
```

Before analyzing the data, it is important to note the accelerometer's axes, and which directions of acceleration will produce which values for each axis.



Experiment 1: Bunnyhop

For the first test, I just rode my bike down my backyard, bunnyhoped, slowed down, and stopped the recording.

Loading in the Variables - Bunnyhop

First, the file with all of the variables recorded during the bunnyhop trial was loaded .

```
load bunnyhop001_Kink.mat
```

Then, the matrices for each of the individual axes are saved as their own variables so that more files can be loaded later on without losing the data for the bunnyhop trial.

```
xBunnyhop = Acceleration.X;
yBunnyhop = Acceleration.Y;
zBunnyhop = Acceleration.Z;
tBunnyhop = Acceleration.Timestamp;
```

Plotting the Raw Data - Bunnyhop

Using this raw data, I can plot a figure by graphing the values of the x, y, and z accelerometers in relation to time.

```
bunnyhopAllDirections = [xBunnyhop, yBunnyhop, zBunnyhop];

figure; plot(tBunnyhop, bunnyhopAllDirections)
title('X, Y, and Z acceleration values during bunnyhop')
xlabel('Time')
ylabel('Acceleration (m/s^2)')
legend('X', 'Y', 'Z')
```

When first looking at this graph, I could see that the y and z coordinates have nonzero initial acceleration values due to gravity, which remains consistant with the first experiment. As for observations relating to the actual jump, I can see that the most noticeable part of it was not the takeoff, or time in midair, but the landing, during which there is a large spike. During the bunnyhop, I also noticed that the y acceleration value, which normally was just over -10m/s^2 now fluctuated closer to 0.

Jump Identification Method 1

One thing that I hoped to be able to get from this graph is the time during which I was in midair, and by looking at the graph, I see that the two largest spikes are the ones from the takeoff and the landing. If I can find the times of these spikes and calculate the time between them, I should be able to calculate the time in midair.

```
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

disp("totalAirTime " + ((lowestValueIndex - secondLowestValueIndex))/100)
```

```
totalAirTime 0.6
```

We can see that I spent a total of 0.6 seconds in midair. Assuming that the largest negative peaks occur during takeoff and landing, this could theoretically be used to find out the air time of a jump. If it is not the case, it may be possible to use the lack of large changes in acceleration during the jump to isolate it, in conjunction with the takeoff and landing peaks.

I also made a graph to double check that the peaks were in the right place.

```
testYBunnyhop = yBunnyhop;
for i = 1 : length(testYBunnyhop)
    if i ~= lowestValueIndex && i ~= secondLowestValueIndex
        testYBunnyhop(i) = 0;
    else
        testYBunnyhop(i-1) = 70;
        testYBunnyhop(i) = -70;
    end

end

figure; plot(tBunnyhop, yBunnyhop)
hold on
plot(tBunnyhop, testYBunnyhop)
hold off
title('Side-by-side y-values for bunnyhop and testYBunnyhop')
xlabel('Time')
ylabel('Acceleration (m/s^2)')
legend('Bunnyhop', 'Spike Graph')
```

```
disp(lowestValue)
```

```
  -78.4466
```

```
disp(lowestValueIndex)
```

```
   952
```

```
disp(secondLowestValue)
```

```
  -38.7321
```

```
disp(secondLowestValueIndex)
```

```
   892
```

As seen in the graph, the calculated peaks are in the correct places.

Graphing the Magnitude of the Overall Vector in Relation to Time - Bunnyhop

Using the Pythagorean Theorem, I can find the magnitude of the overall vector, and then plot it to another graph.

```
netBunnyhopAllDirections = sqrt(xBunnyhop.^2 + yBunnyhop.^2 + zBunnyhop.^2);

figure; plot(tBunnyhop, netBunnyhopAllDirections)
title('absolute acceleration vector values during bunnyhop')
xlabel('Time')
ylabel('Acceleration (m/s^2)')
```

This graph shows many of the observations noticed in the first graph, although it is a little more difficult to see where the jump starts and ends, as while the acceleration value during the jump is lower than most of the values around it, it doesn't really hover around 0.

Graphing the Polar Coordinates in Relation to Time - Bunnyhop

Using the accelerometer data, I can also find the coordinates for if I were to plot the vector in spherical polar form: (ρ, θ, ϕ)

```
rhoBunnyhop = sqrt(xBunnyhop.^2 + yBunnyhop.^2 + zBunnyhop.^2);
thetaBunnyhop = atand(xBunnyhop./zBunnyhop);
phiBunnyhop = acosd(yBunnyhop./rhoBunnyhop);
```

I can then plot each of these as a function of time. (rhoBunnyhop is the same as the vector magnitude graph, so I won't replot it) As shown in the previous experiment, the theta and phi graphs tend to not display patterns very clearly, so I can plot the graph of only the takeoff and landing locations on the same window as the theta and phi graphs.

```
testYBunnyhop2 = yBunnyhop;
for i = 1 : length(testYBunnyhop2)
    if i ~= lowestValueIndex && i ~= secondLowestValueIndex
        testYBunnyhop2(i) = 10;
    else
        testYBunnyhop2(i) = 170;
    end

end

figure; plot(tBunnyhop, thetaBunnyhop)
hold on
plot(tBunnyhop, testYBunnyhop2)
hold off
title('theta value for bunnyhop')
xlabel('Time')
ylabel('Degrees')
legend('Theta', 'Spike Graph')
```

```
figure; plot(tBunnyhop, phiBunnyhop)
hold on
plot(tBunnyhop, testYBunnyhop2)
hold off
title('phi value for bunnyhop')
xlabel('Time')
ylabel('Degrees')
legend('Phi', 'Spike Graph')
```

Much like in trial 1, the theta value shows essentially no useful information, with the values being all over the place.

Since gravity causes the phi value to automatically sit at around 160 (the angle of the seatstay of the bike, where the phone is attatched to), it is difficult to see the distinct takeoff and landing, where the vector would be close to 180. However, we can see that the phi value tends to be less than 160 throughout the duration of the entire jump, which could be used in future jumps to determine the length of the airtime.

Measurement Angle Correction

After realizing that the phone itself was at an angle, I knew that it wouldn't be too hard to make it so that the y vector pointed straight up and down, and the z vector pointed forwards and backwards. By adding the component vectors of y and z in the "corrected y" direction, I could calculate the value of the "corrected y" vector, and repeat my calculations for z. According to Vital BMX, the seat tube angle of the 2018 Kink Whip (my BMX bike) is 71 degrees, so I based my calculations off of it.

```
yComponentCorrectedYBunnyhop = yBunnyhop * cosd(19);
zComponentCorrectedYBunnyhop = zBunnyhop * cosd(71);

correctedYBunnyhop = yComponentCorrectedYBunnyhop + zComponentCorrectedYBunnyhop;

yComponentCorrectedZBunnyhop = yBunnyhop * cosd(71);
zComponentCorrectedZBunnyhop = zBunnyhop * cosd(19);

correctedZBunnyhop = yComponentCorrectedZBunnyhop + zComponentCorrectedZBunnyhop;
```

I could see that this worked by plotting the z graph of the corrected values.

```
figure; plot(tBunnyhop, correctedZBunnyhop)
hold on
plot(tBunnyhop, zBunnyhop)
hold off
title('Z values of corrected vs normal bunnyhop vectors')
xlabel('Time')
ylabel('Acceleration (m/s^2)')
legend('corrected', 'normal')
```

The Z value, which originally sat around 5 when the bike was not in motion now sits at 0. The corrected vector values are not especially useful in this instance, since the regular y vector is already good enough to locate the

beginning and end of the jump, but in future instances, especially when I mount my phone on a new bike and the seat tube isn't as straight, knowing how to correct the vector directions can be handy.

Experiment 2: Plastic Ramp

For this test, I rode my bike down the same path in my backyard, but this time jumping using a small plastic ramp that I have.

Loading in the Variables and Plotting the Raw Data - Plastic Ramp

I repeated what I did for the bunnyhop analysis.

```
load plasticRamp_Kink.mat

xPlasticRamp = Acceleration.X;
yPlasticRamp = Acceleration.Y;
zPlasticRamp = Acceleration.Z;
tPlasticRamp = Acceleration.Timestamp;

plasticRampAllDirections = [xPlasticRamp, yPlasticRamp, zPlasticRamp];

figure; plot(tPlasticRamp, plasticRampAllDirections)
title('X, Y, and Z acceleration values on Plastic Ramp')
xlabel('Time')
ylabel('Acceleration (m/s^2)')
legend('X', 'Y', 'Z')
```

While the hop did seem more defined, I could already see that the method used to find the hop in the bunnyhop test would not work for the plastic ramp test. The second highest peak did not line up with the takeoff. I ended up coming up with some different methods to calculate air time.

Jump Identification Method 2

One thing that I noticed in this chart is that there was a lot less change in acceleration during the jump than while riding, so I tried to calculate the air time based on the lack of acceleration.

```
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
```

This algorithm works by calculating a "change in total acceleration" value and seeing it it passes a threshold. The "change in total acceleration" value is calculated like this:

The indexes of "array A" are shown below.

[1] [2] [3] [4] [5] [6] [7] [8]

$$|A(2) - A(3)| + |A(3) - A(4)| + |A(4) - A(5)| + |A(5) - A(6)| + |A(6) - A(7)|$$

The calculated value will then be used to determine the "state" of the bike at position 2.

The three possible states are "Still", "Moving", and "Airborne". If the value is greater than the assigned "threshold" value, it means that the bike is "Moving" over the bumpy ground, while if it is less, it means that the bike is either "Still" or "Airborne". Since an object falls 44 meters in 3 seconds, it is safe to say that if an object retains the "Airborne" status for over 3 seconds, the object is actually "Still", which is how my algorithm determines whether the bike is "Airborne" or "Still".

In order to see if the algorithm worked, we can plot modeArray and yPlasticRamp on the same graph.

```
modeArray = modeArray - 1;
modeArray = modeArray .* 30;


figure; plot(tPlasticRamp, yPlasticRamp)
hold on
plot(tPlasticRamp, modeArray)
hold off
```

```
title('Jump Detection Algorithm Test')
xlabel('Time')
ylabel('Acceleration (m/s^2)')
legend('Y', 'modeArray')
```

For the plastic ramp acceleration graph, this algorithm works very well and is able to detect multiple jumps (if there are multiple jumps), it does have its weaknesses. For example, it the ground is smooth enough, such as in a skatepark, or if there is a lot of shaking during the jump, the algorithm will not be able to detect jumps.

Smoothing out Data using Matlab's Built-in Function

While thinking of ways to manipulate the data further, I found a function built into matlab that smooths out curves, and I tried to apply it to the Plastic Ramp Jump data.

```
smoothedYPlasticRamp = smooth(yPlasticRamp);

figure; plot(tPlasticRamp, yPlasticRamp, ':');
hold on
plot(tPlasticRamp, smoothedYPlasticRamp,'-');
title('Rough vs Smooth yPlasticRamp')
xlabel('Time')
ylabel('Acceleration m/s^2')
legend('Rough', 'Smooth')
```

While this graph does not really provide anything useful in terms of the algorithms that I already have, it helped me see that on average, the g forces were more negative right before the jump.

Smoothing out Data using an Average of Nearby Values

It may be useful in future tests to calculate the average value around each point, and calculate where the average drops off the most, which is where the jump would theoretically be. For now I just graphed the averages to see what it would look like.

```
valueAvgArray = zeros(length(yPlasticRamp), 1);
for i = 11: (length(yPlasticRamp) - 10)
    valueAvg = 0;
    for j = -10 : 10
        valueAvg = valueAvg + (yPlasticRamp(i + j));
    end
    valueAvg = valueAvg / 21;
    valueAvgArray(i) = valueAvg;
end

figure; plot(tPlasticRamp, yPlasticRamp)
hold on
plot(tPlasticRamp, valueAvgArray)
hold off
title('Rough vs Averages')
xlabel('Time')
```

```
ylabel('Acceleration (m/s^2)')
legend('Y', 'valueAvgArray')
```

This smoothing function ended up working very well. With another round of smoothing, the curve could potentially look like there is virtually no interference.

As the jumps that I record get larger and larger, I will hopefully be able to focus more on the lack of forces during the jump and less on the interference, eventually allowing me to identify jumps using only the increased g-forces during takeoff and landing and the lack of g-forces during the jump. As for now, the jumps are still too small to notice either of these clearly.

Graphing the Magnitude of the Overall Vector in Relation to Time - Plastic Ramp

Next, I graphed the vector magnitude graph.

```
netPlasticRampAllDirections = sqrt(xPlasticRamp.^2 + yPlasticRamp.^2 + zPlasticRamp.^2);

figure; plot(tPlasticRamp, netPlasticRampAllDirections)
title('Vector Magnitude Values on the Plastic Ramp')
xlabel('Time')
ylabel('Acceleration (m/s^2)')
```

Nothing new or interesting really showed up on the vector magnitude graph. In future experiments, I might abandon the use of this graph, as it seems like it is just an inferior version of the correctedY graph, which has positive and negative values.

Graphing the Polar Coordinates in Relation to Time - Plastic Ramp

Next come the spherical polar form values:

```
rhoPlasticRamp = sqrt(xPlasticRamp.^2 + yPlasticRamp.^2 + zPlasticRamp.^2);
thetaPlasticRamp = atand(xPlasticRamp./zPlasticRamp);
phiPlasticRamp = acosd(yPlasticRamp./rhoPlasticRamp);

figure; plot(tPlasticRamp, thetaPlasticRamp)
title('Theta Value for the Plastic Ramp')
xlabel('Time')
ylabel('Degrees')
```

```
figure; plot(tPlasticRamp, phiPlasticRamp)
title('Phi Value for the Plastic Ramp')
xlabel('Time')
ylabel('Degrees')
```

These two graphs may have more use if interference can be smoothed out, but these tests will likely not be seen in future experiments.

Conclusion:

These tests helped me start to get a better idea of how to analyze patterns during a jump, and I was able to come up with a couple jump detectors, as well as ways to smooth out data. If I were to do the test again, I would do it on the smooth sidewalk in front of my house rather than my bumpy backyard, but it is unlikely that I will, since I think it would be more useful spending time on larger jumps, especially if I am focusing on finding patterns during jumps. For the next test, I will likely head to the skatepark to attempt drops and jumps, maybe bunnyhop while I'm there, and further on in the future, I might try to see the effects of riding down stairs on different bikes. During the skatepark test, it will be especially interesting trying to identify drops, as there is no large g-force during the takeoff.