# Two Dimensional Localization of Impacts Using Acoustic Time Delay Measurement

**Joshua Send**
**Torrey Pines High School**

## Introduction

This project is an extension of a previous year's project that involved similar ideas and basic mechanisms. That time, the project's goal was to demonstrate the principle for localizing impacts along a linear object (a pipe) by using a simple proportion between arrival times, which converts into a distance along the pipe. Simpler (piezo) sensors were used, and only basic arrival time detecting algorithms. This year's new project aims to demonstrate the ability to determine the source of the vibration in two dimensions (on a plane surface) using arrival times of vibrations at three sensors to create hyperbolas that intersect at the source of the vibration, while also comparing more advanced detecting algorithms

## Comparison to The Previous Year

Some of the background theory behind last year's linear localization project is similar to this year's two-dimensional localization system. The general idea behind both is to measure arrival times of vibrations, and use these to localize the source of the vibration.

With the hardware, there are also some similarities, such as the use of analog-digital converters (ADC's) to create digital data from analog-output of the sensors. The microcontroller has also not changed, though this was an original goal that was not able to be accomplished. A major departure of the hardware is the switch from piezoelectric sensor to accelerometers. Accelerometers offer more choice and simplified use than piezoelectric sensors, which are not powered and generate extremely weak signals that require amplification and level shifting through operational amplifiers (which caused many headaches before). With a sensitive accelerometer mounted vertically so it measures +1G as a baseline, both of these needs are negated, while having a wider range of sensitivities available. Also, accelerometers are often mounted on breadboards or evaluation-boards, which allow extensions to directly attach ADC's close to the sensors, eliminating excess noise picked up by transmitting analog signals over long distances. Another change in hardware was the use of a single-point contact to the surface being monitored (a screw into the board, with the sensor attached to the top of the screw).

The software similarities only go as far as the software on the Arduino, which previously used two circular buffers and running averages for two sensors. Now, there are three of each for the three sensors. A few memory optimizations were made to squeeze more data storage room out of the Arduino's limited 2 KB memory. The computer software was totally rewritten to use a new graphics system (Python's Tkinter) which enables advanced graphical representations and interactions. Underneath this layer, the programming uses new object-oriented structuring; coordinate rotations; data filtering, scaling, and shifting; and discrete wavelet transforms, least squares line fits, and spline fits for data analysis.

Apart from the more sensitive sensors and the additional dimension, the major difference comes from implementing and comparing more advanced arrival detection algorithms. Ultimately, using high-frequency wavelet arrivals represented a major improvement over the previous simpler threshold methods. Also, the interactive and graphical display of data, arrival detections, and solutions are new this year.

Although the accuracy of this year's system in the basic detection mode is lower than the previous years', this is an expected result with the low temporal resolution the Arduino achieves with the additional sensor it has to monitor. Because this was known at the start of the project, a Leaf Labs© Maple controller with 72 MHz clock speed was acquired. However, even with weeks of work testing the third sensor, asking others for help, and multiple complete sensor replacements, the third sensor setup refused to work correctly with the Maple. Interestingly, each of the sensor replacements worked correctly with the Arduino. It is assumed that this discrepancy comes from the fact that the Maple runs at 3.3V (and so its I/O pins do too) while the Arduino operates using 5V – the voltage the ADC's are made to run at, though their datasheet claims 3V is enough to trigger usage. Because of the problem with the third sensor, the project reverted to using an Arduino microcontroller for data collection and monitoring.

**Materials**

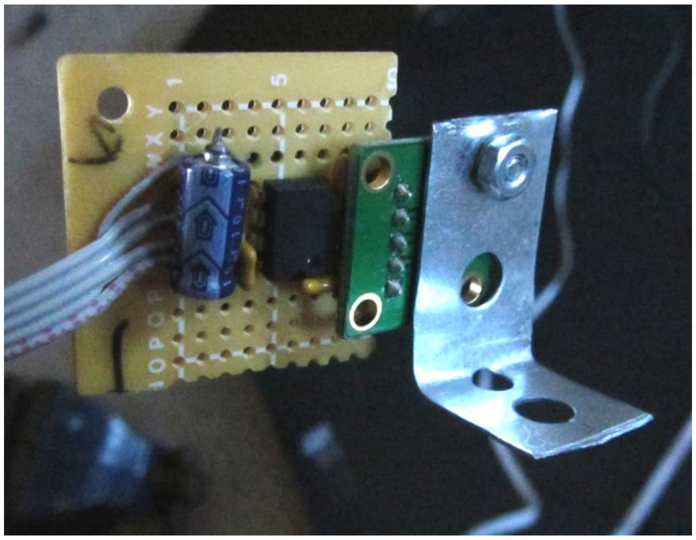
- 1 x Arduino Uno r3
- 3 x Burr-Brown ADS7818 DIP Analog-Digital Converter
- 3 x Analog ADXL203 (EB) analog accelerometer
- 9 x 0.1 µF ceramic capacitor
- 3 x 10 µF electrolytic capacitor
- 1 x Protoboard
- 1 x 60cm by 60cm Hard PVC Board
- 3 x 3" metal tape with holes
- 3 x Small screws and connectors
- Ribbon Cable
- Oscilloscope
- Arduino IDE
- Python 2.7
- Male to Female USB cable

- Roll of solid copper hookup wire
- Velleman VTSS5 soldering Iron
- RadioShack 60/40 Light-Duty Rosin-Core Solder
- Adhesive Tape
- Permanent marker
- 1/16” Drill Bit and Drill

**Procedure**

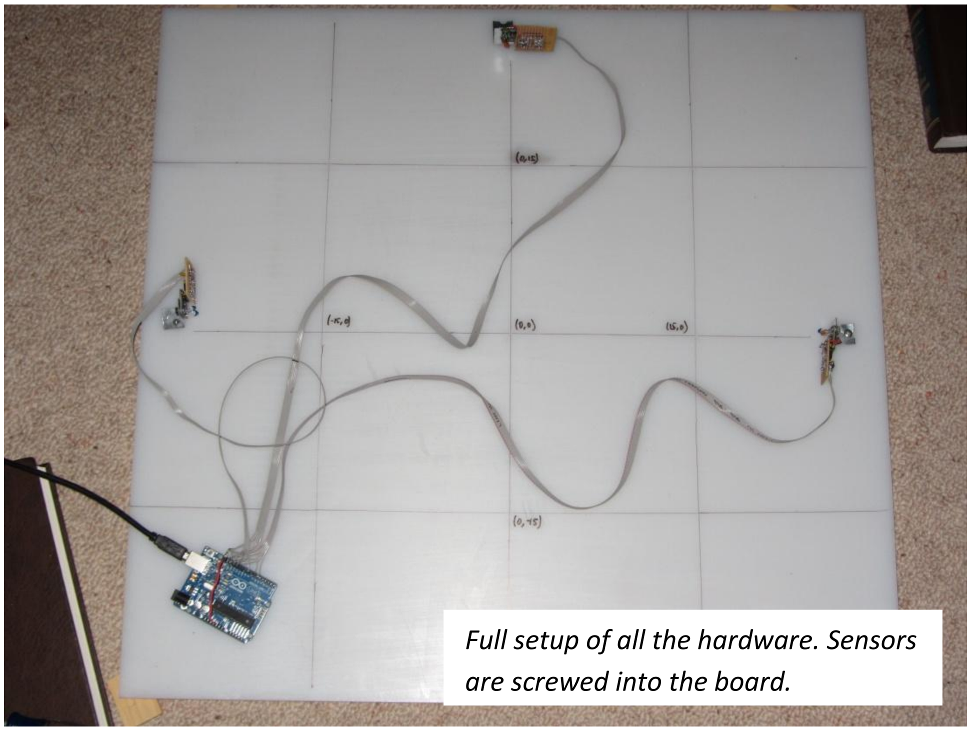
*All procedures performed by student*

1. Test ADXL203EB for adequate bandwidth and sensitivity
    - Connect oscilloscope to either X or Y axis output of the accelerometer, and connect grounds as appropriate.
    - Supply accelerometer with +5V and ground connections
    - Observe changes of level on the oscilloscope as selected axis is perpendicular to the ground and then parallel.
2. Create foot and three attachment boards for accelerometers
    - Bend metal tape to 90 degrees to make a foot
    - Screw together ADXL203EB and metal tape through one of the holes
    - Cut protoboard into 3 pieces large enough to hold ADC, and 4 capacitors
    - Connect the ADC to correct pins:
        - Vref (pin1) decoupled to ground with 0.1 μF ceramic capacitor
        - +In (pin2) to the X or Y axis of the ADXL203EB
        - -In (pin3) to ground



*Final assembly of the sensor and analog-digital converter components*

- GND (pin4) to ground
- CONV (pin5) to Arduino pin 6,7, and 8 for respective chips
- DATA (pin6) to Arduino pin 12
- CLK (pin7) to Arduino pin 13
- +VCC (pin8) to +5V and decoupled to ground two 0.1 µF and one 10 µF capacitors



*Full setup of all the hardware. Sensors are screwed into the board.*

- Ribbon cable was used to connect Arduino and protoboard

- Provide power to ADXL203, and ground it and the metal foot

3. Connect Arduino to computer via USB cable
4. Put screws through holes in metal tape, and screw whole assembly into the board at desired locations.
5. Software – Arduino C++
   a. SPI library parameters
      i. SPI.setBitOrder(MSBFIRST);
      ii. SPI.setClockDivider(SPI_CLOCK_DIV2);
      iii. SPI.setDataMode(SPI_MODE0);
   b. Collect data from ADC's one, two, and three (only first one shown)
      i. PORTB &= B11111011; // pull pin 10 of Aaduino LOW (ADC sample)
      ii. PORTB |= B00000100; // pull pin 10 of Arduino HIGH (ADC off)

iii. These are port manipulations to save CPU cycles

c. Calculate current total for set of data

d. Check each new value against the running total

i. Avoid averaging because it uses division, which is a slow process

e. If value exceeds threshold, send data to computer over USB, but not the calculated times

**Explanation of PC-side software**

Because the Python 2.7 program spans over 1100 lines, a general overview of what it does and how it works is required.

The program is structured in two big classes: class Scifair, and class newHyberolaClass. There are a few global functions outside of both, and global variables (mostly constants that are subject to change) are at the top of the file. Class Scifair is the encapsulating structure that contains everything for the graphics, serial checking, processing, and event handling. The newHyperbolaClass exists for the graphing of hyperbolas in an efficient manner. It requires a center for the hyperbola, and a point along the new x-axis. From these it generates the new, rotated and shifted coordinate system for the hyperbola that will be generated once input data such as foci and time difference is provided. Hyperbolas are generated with the yieldHorizontalHyperbolaPoints() function, which yields one set of points at a time to conserve memory. Before the point is passed back to the caller, it is translated back into the normal coordinate system.

Class Scifair's init() function mostly contains the setup calls required to create the graphics and initialize required variables. Notable are the 'outerFrame' frame, which contains all the graphics, the 'canvas' canvas, which is used to graph the hyperbolas, and the 'plot' canvas, which is used to plot the data. The class is documented using #---label--- notation. The first major label is "button related functions" which are the helper functions that the graphical buttons require to work. Next are "coordinate tools" which are functions that are used to convert human, (0,0) centered coordinate systems to the computer's (0,0) top left coordinate system for both the 'plot' and 'canvas' canvases. "Hyperbola Graphers" are functions that create and use the hyperbola class to graph the hyperbolas. "Object Removal and Timed Deletion" documents a few functions that delete graphical objects based on tags applied when they were created. "Iterative Solvers" are a set of functions that are unused at fair time, but are written for an expected extension of the project in which the sensors can be moved around arbitrarily and recalibrated to work as well as before using only a little input data (in essence, that problem comes down to doing the opposite as solving for impact locations – impact locations are known but sensor locations are not). "Basic

Drawing Functions" are and do exactly what the label describe, for both the 'canvas' and 'plot' canvases.

Second to last in this class is "Serial and Data processing", which is where much of the gruntwork of the system happens. Here serial data is read in from the Arduino, processed, cleaned, scaled, checked for validity, and sent to analysis functions. The results of those are sent to the plotting and hyperbola graphing functions to display the results. The next section is "Analysis Methods". Here the functions that process the data for arrival times of waves live: 'stdThreshold' uses a method similar to the Arduino's native code, comparing data points to an average and when this exceeds a threshold, it returns the associated time. The second detection method 'lifeFit' takes the data passed in, rectifies it over the average, and zeroes for values below the threshold. Differences between points are then generated to find relative maxima. The increasing maxima are used to apply a least-squares fit through the increasing maxima. The line created is intersected with the baseline average, and this point's x value is an approximation of the start of the impact wave. Finally, for the third detection method, the function 'wavelet' takes its input data (which has already been interpolated and spaced evenly with a spline fit), applies a discrete single-level wavelet transform (returning the amplitude coefficients versus time for the highest possible frequency band), rectifies the coefficients, applies a threshold, and then finds the time of the first non-zero coefficient.

The entire program can be found in Appendix B.

## A More Detailed Look at Analysis Algorithms and Dependencies

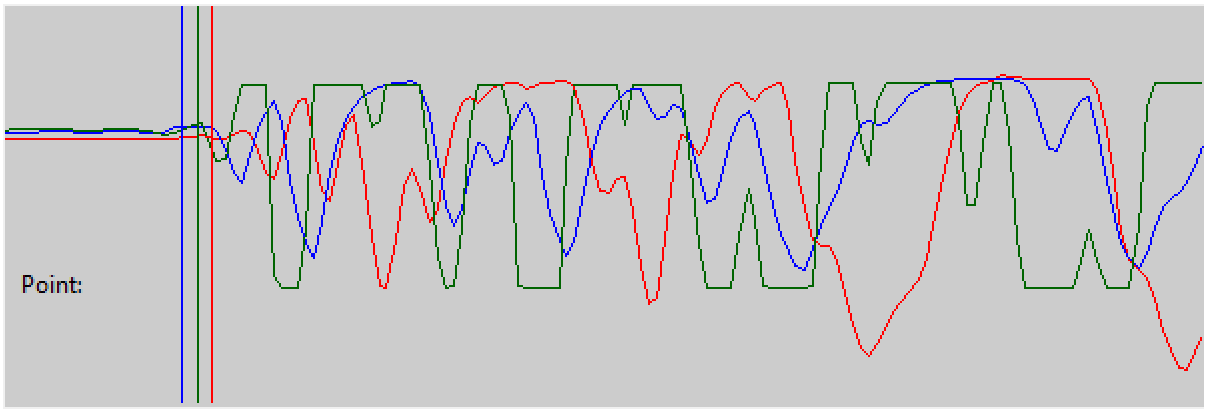
This section is intended to overview how the system uses various algorithms, and to give slightly more than surface-level explanations of how they work.

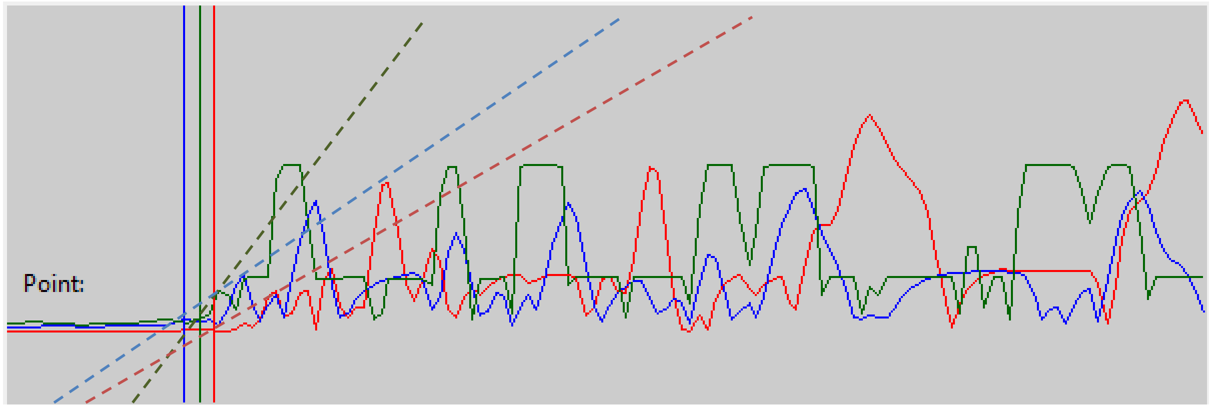
### Standard Threshold Analysis

There are three analysis methods that can be used in the system interactively. The first is a basic threshold. It works by taking the first 5 data points received from the controller, and using the average of them as a baseline for further comparison. Then it traverses the rest of the data set, while comparing each new data point with the baseline average. When a data point exceeds the baseline by too much (determined by a threshold variable), the time associated with that point is stored for later use. After a time has been determined from each stream of data (ie. each sensor's data), these times are sent to a different function to be turned into hyperbolas and plots for the graphical interface.

### Least Squares Line Fit Analysis

The next analysis algorithm is the least-squares line fit algorithm. A least-squares line fit essentially takes a set of points, and minimizes the average distance to each point (squared). The algorithm in this system employs it to find a straight-line fit through a set of relative maxima. To find the maxima, the system rectifies the data and does a series of differences. It then sends those points to a line fit function. The values of the line are returned, and intersected with a line representing the baseline average. The intersection's x-coordinate (time) approximates the time of arrival of the wave. For example:





Here, the top graph represents the basic data that the computer receives from the controller. The lower plot is the same data, rectified and shifted down (to see the upper ranges of the values). The hand-drawn line fits (dashed lines) through the maxima show that the system's estimates are on par with what one would expect. As before, the times are sent to the hyperbola graphing function. *Note: the vertical lines are where the system calculates the start of the wave to be*

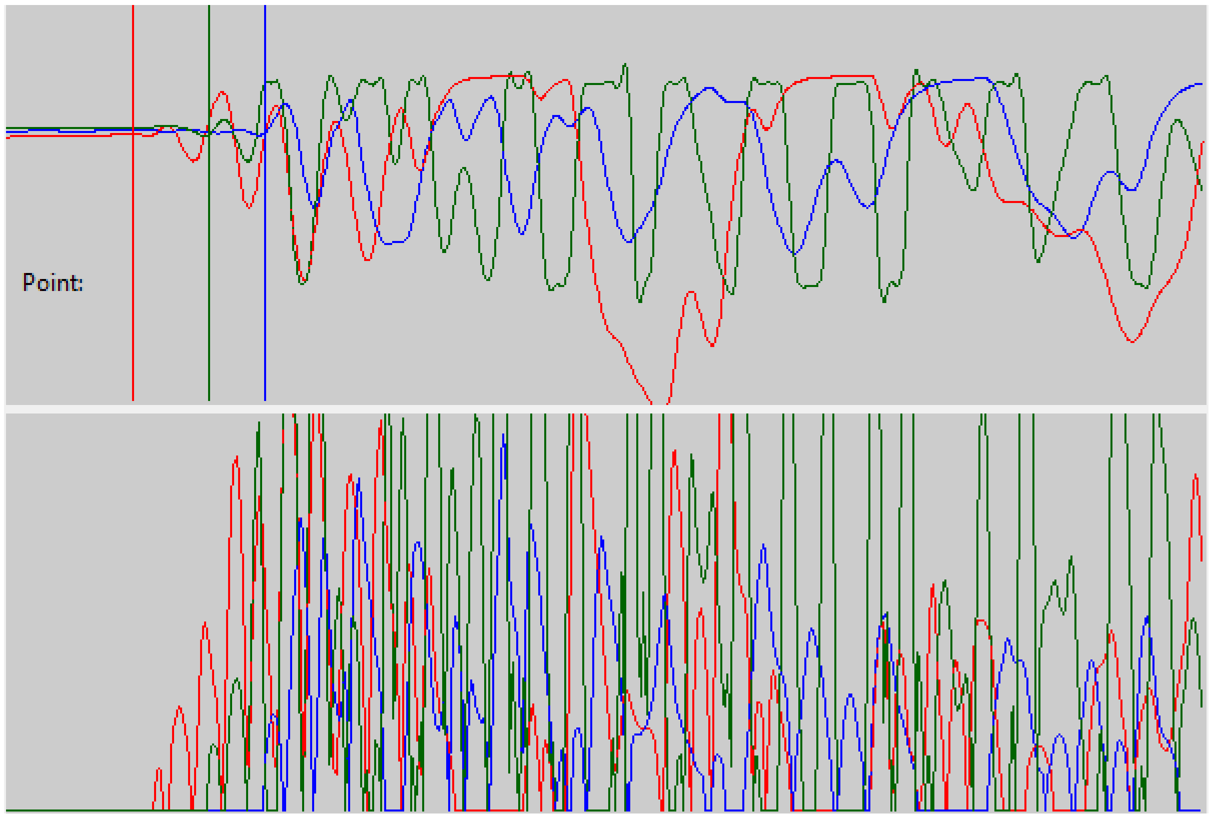
<u>Wavelet Analysis</u>

Finally, the most complicated analysis method is the discrete wavelet transform based algorithm. A wavelet transform is similar to a fourier transform in that it takes a set of data and returns frequency components. However, a fourier transform cannot handle rising or decaying frequencies over time. The wavelet transform (a relatively new invention) was made to fix this. Since FFTs (fast Fourier transforms) do not work over very short data segments, the wavelet methods use successive sequences of high- and low-pass filters. In the wavelet transform function 'wavedec' provided in the Pywavelets API, one feeds in data and the level of decomposition to do. The data is assumed to be evenly spaced, which is why a spline fit has to applied to the data beforehand.

A cubic spline fit such as the one used in this system (provided in the Scipy library) fits a smooth line consisting of cubic polynomial segments between points. With more points, it fits adjacent endpoints and derivatives to obtain a smooth interpolation. Using this, one can generate data at any point along the curve between given points. So, for the wavelet algorithm to work, the data is first fed into the spline fit function and then re-extracted at evenly spaced intervals. This is fed into the wavelet transform.

The level parameter of the transform indicates which frequency should be analyzed for. Level one returns the frequency corresponding with two data points per return value (which represents the amplitude of that frequency in the time covered by those two data). For example, 4 data points spaced 2 seconds apart would return two coefficients spanning 4 seconds each. Similarly, for 1024 points spanning 4 microseconds each, 512 coefficients covering 8 microseconds each would be returned. This situation is shown below.

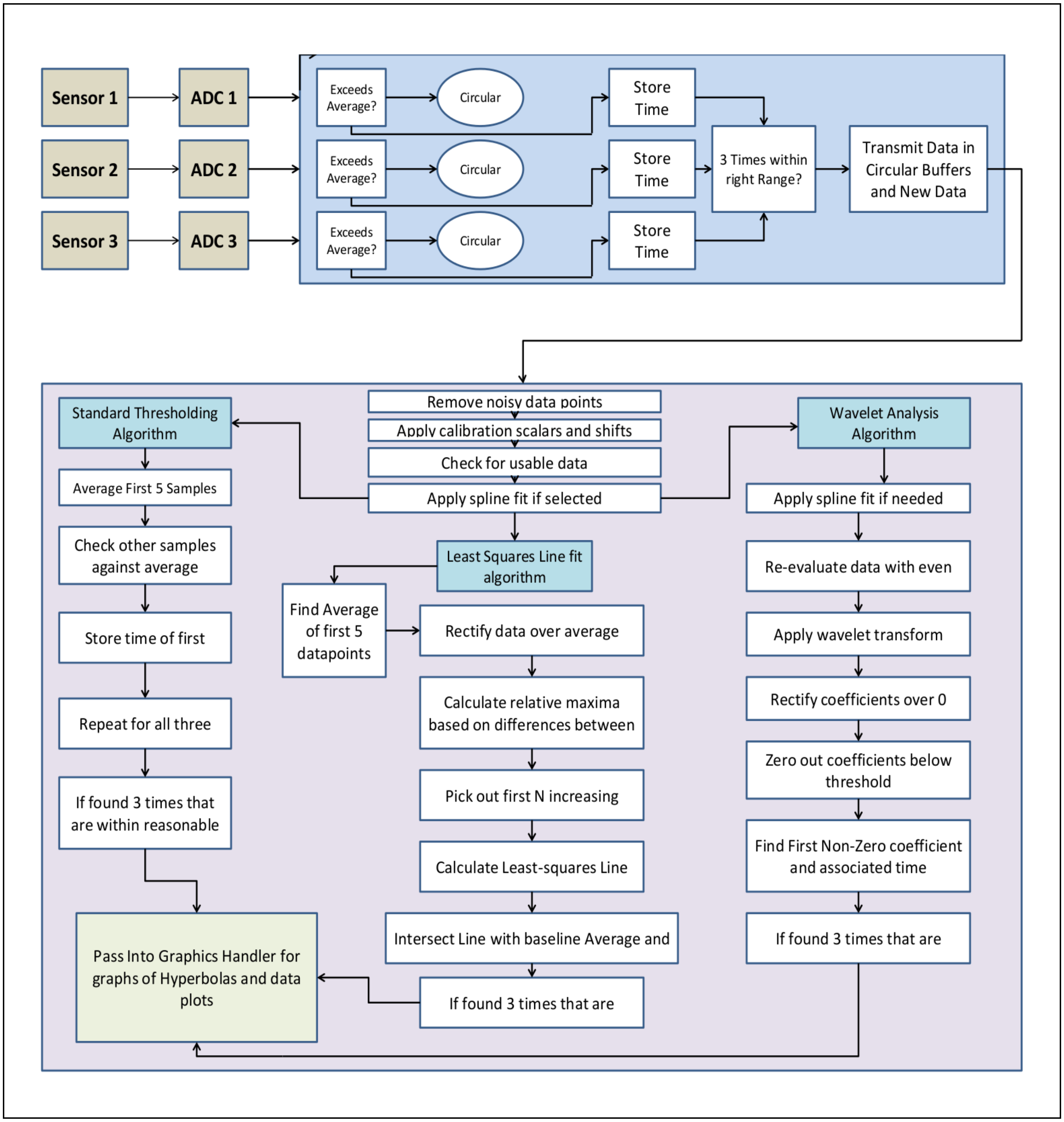
The wavelet analysis then takes the coefficients, rectifies them, and zeroes out the ones below a set threshold (default 5.7). This is the result, as displayed by the system:



The top graph is the acceleration data received by the system at the three sensors, and bottom graph shows the wavelet amplitude coefficients calculated and rectified. The entire record is 4.096 milliseconds long. The points graphed are also the ones that did not get zeroed out for being too small. The time of the first-nonzero coefficient is the start of the wave, and this value is passed into the same hyperbola graphing function as the other analysis methods. Note that the waves/oscillations analyzed by the wavelet are much higher frequency than the oscillations visible by eye in the upper panel.

This system's programming depends on a few open source libraries. One of them is NUMPY, the fundamental package for scientific computing in Python. Also needed was the Pyserial library to communicate with the Arduino, the Pywavelets library, which provided the discrete wavelet transform, and the SciPy library, which contained the spline fit function.

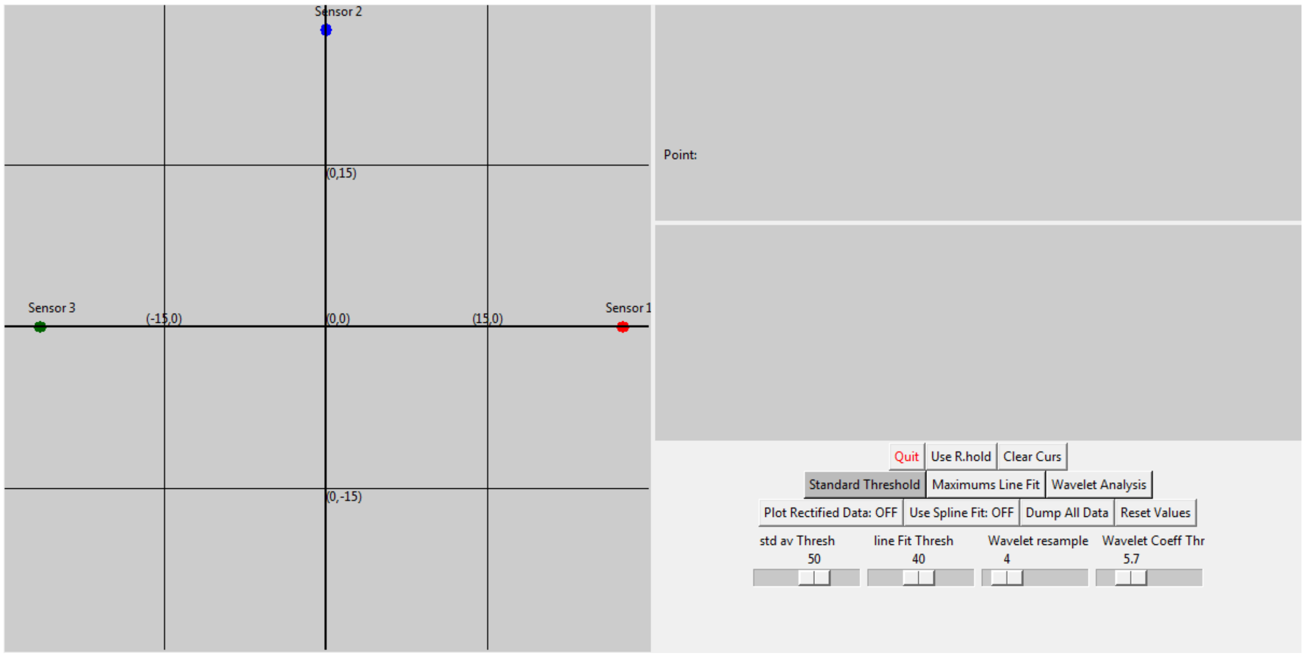
The next page is a flowchart of the large processes the system goes through to find the starting times of the accelerations.


Sensor 1
Sensor 2
Sensor 3
ADC 1
ADC 2
ADC 3
Exceeds Average?
Circular
Store Time
3 Times within right Range?
Transmit Data in Circular Buffers and New Data
Standard Thresholding Algorithm
Average First 5 Samples
Check other samples against average
Store time of first
Repeat for all three
If found 3 times that are within reasonable
Remove noisy data points
Apply calibration scalars and shifts
Check for usable data
Apply spline fit if selected
Least Squares Line fit algorithm
Find Average of first 5 datapoints
Rectify data over average
Calculate relative maxima based on differences between
Pick out first N increasing
Calculate Least-squares Line
Intersect Line with baseline Average and
If found 3 times that are
Wavelet Analysis Algorithm
Apply spline fit if needed
Re-evaluate data with even
Apply wavelet transform
Rectify coefficients over 0
Zero out coefficients below threshold
Find First Non-Zero coefficient and associated time
If found 3 times that are
Pass Into Graphics Handler for graphs of Hyperbolas and data plots

## Findings and Results

### Overview

Throughout this section, versions of this diagram will continue to appear:

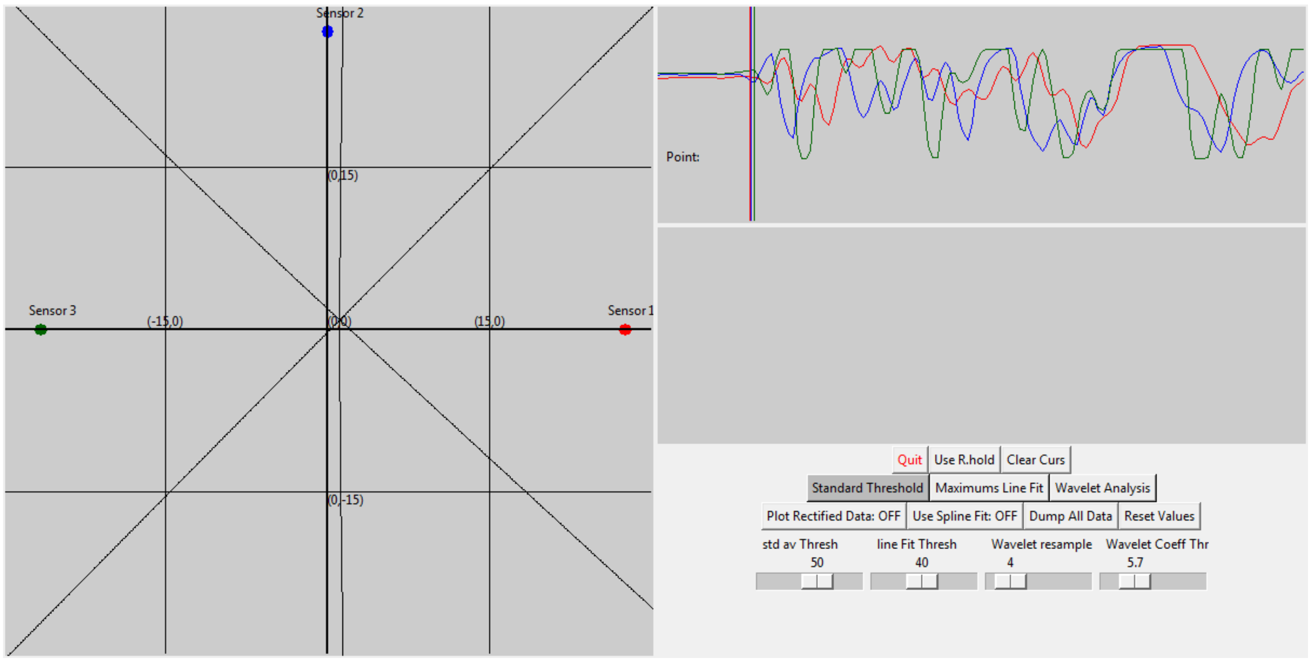


This is the format the system uses to output its data. The left half is the virtual representation of the actual board. Hyperbolas are plotted here. The top right frame is used to plot data vs time for all three sensors. The second plotting area on the right is only used when wavelet analysis is used. It displays rectified wavelet coefficients, versus time.

All of the graphs and plots are color coded. Red is always for sensor 1, blue is always for sensor 2, and green is always for sensor 3. The hyperbolas do not have a color because they are a composition of the data from two sensors.

## Examples of Analysis

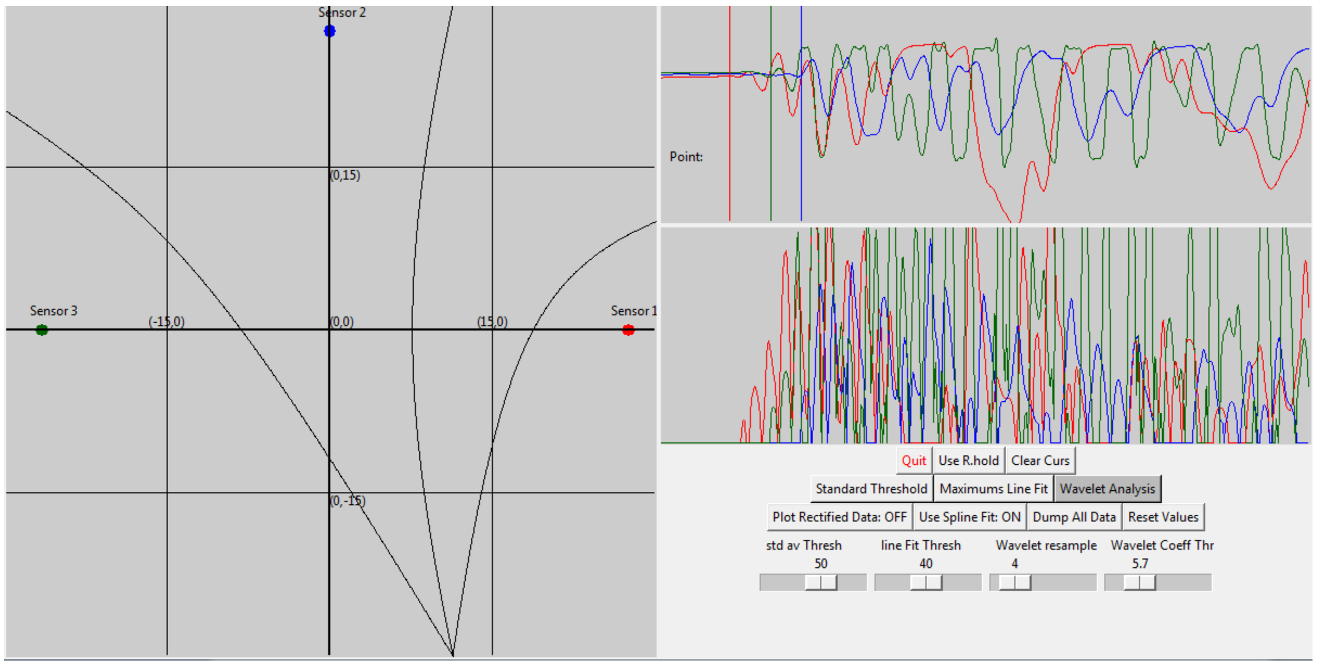
To start, here is an example of an accurate result given with the “standard threshold” method:



Plot Rectified Data: OFF
Use Spline Fit: OFF
std av Thresh 50
line Fit Thresh 40
Wavelet resample 4
Wavelet Coeff Thr 5.7

The pair of slightly curved lines that form an ‘x’ and the vertical slightly curved line are halves of the three hyperbolas generated by the calculated time delays. In the plotting window (top right) the data received from the Arduino is plotted against time. The three vertical lines spaced closely together are the indicators of when the system believes the waves start. Those times are used for the hyperbola graphs in the left frame. The intersection of the hyperbolas is the point the vibrations originated from.

This is a typical non-centered result using the “wavelet analysis” algorithm:



Plot Rectified Data: OFF
Use Spline Fit: ON

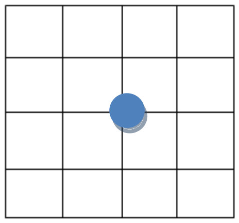
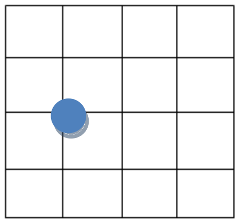
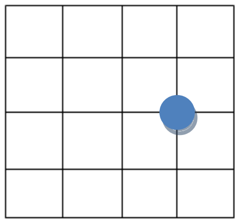
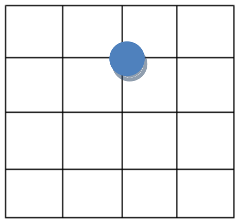
std av Thresh 50
line Fit Thresh 40
Wavelet resample 4
Wavelet Coeff Thr 5.7

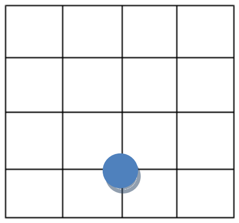
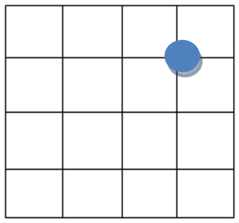
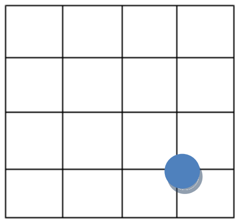
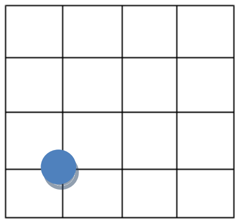
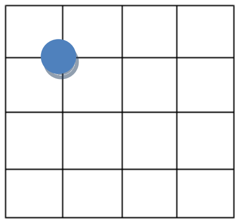
This example is a very accurate instance of the system’s function. The intersecting hyperbolas meet at exactly the place the impact was generated. Here the curvature of the hyperbolas is also clearly visible, and the time differences in arrival times are indicated clearly with the large separations between the colored vertical line indicators on the upper right plot.

Important in the last image is the use of the second plotting frame. It shows the coefficients that the single level discrete wavelet transform returned (though rectified). The algorithm has already zeroed out the wavelet coefficients with a value of less than 5.7, so all that remain are significant peaks. The two plots use the same time axis, so it is possible to verify that the wave start indicators are actually where the wavelets are not zero. Indeed, this is true upon close inspection (the red indicator is slightly in front of the first red wavelet because of pre-coded calibration values).

## Accuracy and Repeatability

Following is a table of mean errors and root mean square errors at select locations using the three analysis methods. There were at least 20 trials run at each location.

| Impact Location *(Grid is miniaturized representation of the board; blue dot is impact location)* | System configuration *Algorithm – algorithm options* | Average Error *(mean distance from actual location)* | Root Mean Square Error *(RMSE)* | # of Trials |
|---|---|---|---|---|
| Centered Impact  | Standard Threshold – no Spline fit, threshold 50 | 2.42 cm | 3.06 cm | 21 |
| | Standard Threshold – Spline fit, threshold 50 | 1.78 cm | 2.17 cm | 21 |
| | Standard Threshold – no Spline fit, threshold 35 | 2.46 cm | 3.18 cm | 21 |
| | Standard Threshold – Spline fit, threshold 35 | 2.53 cm | 2.81 cm | 21 |
| | Wavelet Analysis – Coefficient threshold 5.7 | 0.69 cm | 0.98 cm | 30 |
| | Least-Squares line fit – threshold 40 | 5.49 cm | 9.04 cm | 21 |
| Halfway Left  | Standard Threshold – no Spline fit, threshold 50 | 2.99 cm | 3.23 cm | 21 |
| | Standard Threshold – Spline fit, threshold 50 | 2.86 cm | 3.16 cm | 21 |
| | Standard Threshold – no Spline fit, threshold 35 | 3.29 cm | 4.89 cm | 21 |
| | Standard Threshold – Spline fit, threshold 35 | 3.48 cm | 5.57 cm | 21 |
| | Wavelet Analysis – Coefficient threshold 5.7 | 1.32 cm | 1.59 cm | 30 |
| Halfway Right  | Standard Threshold – no Spline fit, threshold 50 | 1.77 cm | 2.49 cm | 21 |
| | Standard Threshold – Spline fit, threshold 50 | 1.72 cm | 2.13 cm | 21 |
| | Standard Threshold – no Spline fit, threshold 35 | 15.29 cm | 15.75 cm | 21 |
| | Standard Threshold – Spline fit, threshold 35 | 1.52 cm | 1.73 cm | 21 |
| | Wavelet Analysis – Coefficient threshold 5.7 | 1.76 cm | 1.93 cm | 30 |
| Halfway up  | Standard Threshold – no Spline fit, threshold 50 | 2.73 cm | 3.22 cm | 21 |
| | Standard Threshold – Spline fit, threshold 50 | 2.32 cm | 2.82 cm | 21 |
| | Standard Threshold – no Spline fit, threshold 35 | 1.76 cm | 2.09 cm | 21 |
| | Standard Threshold – Spline fit, threshold 35 | 1.92 cm | 2.39 cm | 21 |
| | Wavelet Analysis – Coefficient threshold 5.7 | 1.30 cm | 1.84 cm | 30 |
| Halfway Down | Standard Threshold – no Spline fit, threshold 50 | 3.26 cm | 5.67 cm | 21 |
| | Standard Threshold – Spline fit, threshold 50 | 3.36 cm | 4.65 cm | 21 |
| | Standard Threshold – no Spline fit, threshold 35 | 2.78 cm | 3.37 cm | 21 |
| | Standard Threshold – Spline fit, threshold 35 | 2.86 cm | 4.26 cm | 21 |
| | Wavelet Analysis – Coefficient threshold 5.7 | 1.45 cm | 1.93 cm | 30 |

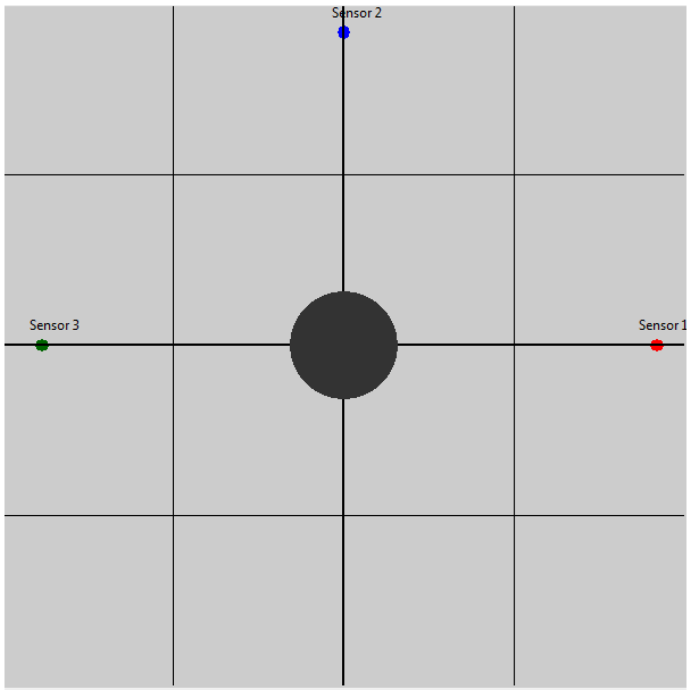
| | | | | |
|---|---|---|---|---|
|  | | | | |
| Upper Right Middle Vertex  | Standard Threshold – no Spline fit, threshold 50<br>Standard Threshold – Spline fit, threshold 50<br>Standard Threshold – no Spline fit, threshold 35<br>Standard Threshold – Spline fit, threshold 35<br><br>Wavelet Analysis – Coefficient threshold 5.7 | 4.39 cm<br>4.79 cm<br>3.84 cm<br>3.27 cm<br><br>2.65 cm | 5.00 cm<br>5.26 cm<br>4.45 cm<br>4.10 cm<br><br>3.04 cm | 21<br>21<br>21<br>21<br><br>30 |
| Lower Right Middle Vertex  | Standard Threshold – no Spline fit, threshold 50<br>Standard Threshold – Spline fit, threshold 50<br>Standard Threshold – no Spline fit, threshold 35<br>Standard Threshold – Spline fit, threshold 35<br><br>Wavelet Analysis – Coefficient threshold 5.7 | 7.42 cm<br>8.42 cm<br>8.77 cm<br>7.16 cm<br><br>3.64 cm | 8.25 cm<br>9.30 cm<br>9.96 cm<br>8.14 cm<br><br>4.08 cm | 21<br>21<br>21<br>21<br><br>30 |
| Lower Left Middle Vertex  | Standard Threshold – no Spline fit, threshold 50<br>Standard Threshold – Spline fit, threshold 50<br>Standard Threshold – no Spline fit, threshold 35<br>Standard Threshold – Spline fit, threshold 35<br><br>Wavelet Analysis – Coefficient threshold 5.7 | 11.23 cm<br>5.90 cm<br>8.31 cm<br>5.98 cm<br><br>2.05 cm | 11.9 cm<br>7.56 cm<br>10.3 cm<br>7.76 cm<br><br>2.32 cm | 21<br>21<br>21<br>21<br><br>30 |
| Upper Left Middle Vertex  | Standard Threshold – no Spline fit, threshold 50<br>Standard Threshold – Spline fit, threshold 50<br>Standard Threshold – no Spline fit, threshold 35<br>Standard Threshold – Spline fit, threshold 35<br><br>Wavelet Analysis – Coefficient threshold 5.7 | 5.94 cm<br>4.70 cm<br>6.13 cm<br>5.16 cm<br><br>2.60 cm | 6.72 cm<br>5.48 cm<br>7.21 cm<br>6.55 cm<br><br>3.54 cm | 21<br>21<br>21<br>21<br><br>30 |

Firstly, the least-squares line fit is only present in the first table entry ("Center") because it is a generally unreliable and inaccurate method. This is because of a few reasons, but mostly that regression analysis (same thing as least-squares fit) requires at least two points to estimate a line. Unfortunately, there is sometimes only one useful maximum, rendering the least-squares fit useless. As this happens more than a quarter of the time, this approach has less-than-ideal characteristics overall. Above, it also has a huge error in both mean and root mean square error at the center, which is generally the most accurate location on the board. Therefore, further analysis was omitted for that method.

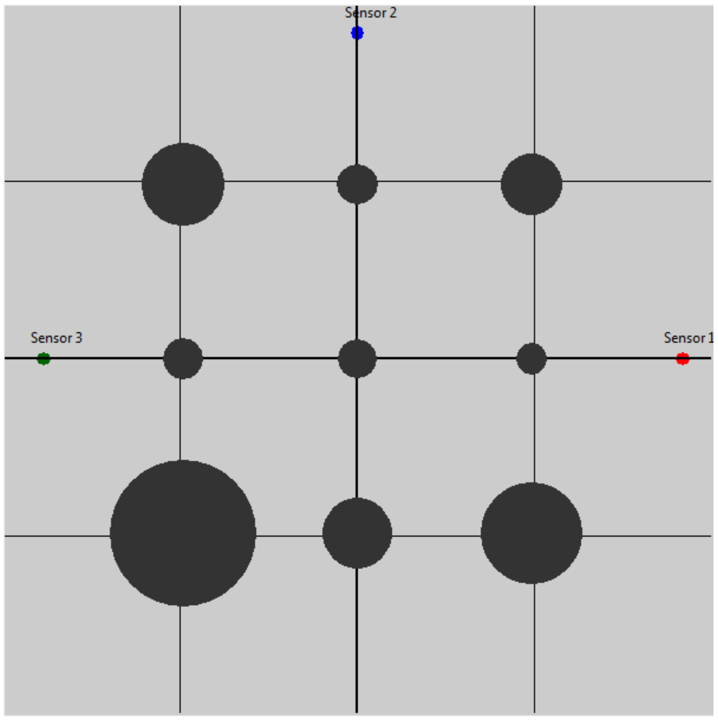
Within the standard threshold option, there were 4 different options explored. The data shows that applying a spline fit to the data generates better overall results. However changing the threshold from 50 to 35 yields better results in only a few cases, and it is not a consistent, large improvement. It can be concluded that the threshold (between 35 and 50) will have little effect on the results.

The results suggest that the wavelet transform approach is far more reliable than the standard threshold algorithm, as indicated by the lower RMSE than the other options at each position. While the wavelet method did get beaten at mean error for a few positions, it is generally more accurate and at all times much more reliable, making it the method of choice for this application.

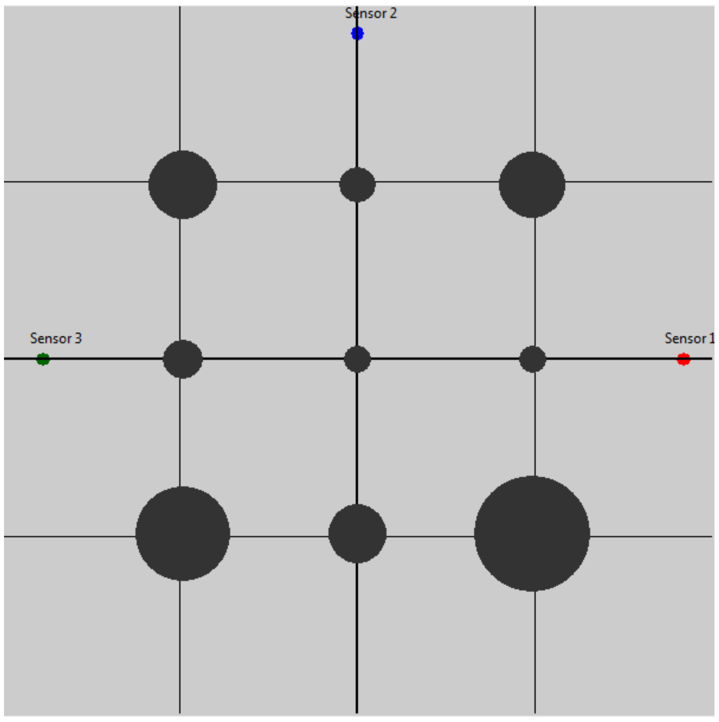
Here is a more digestible representation of the RMSE's of the 6 situations:



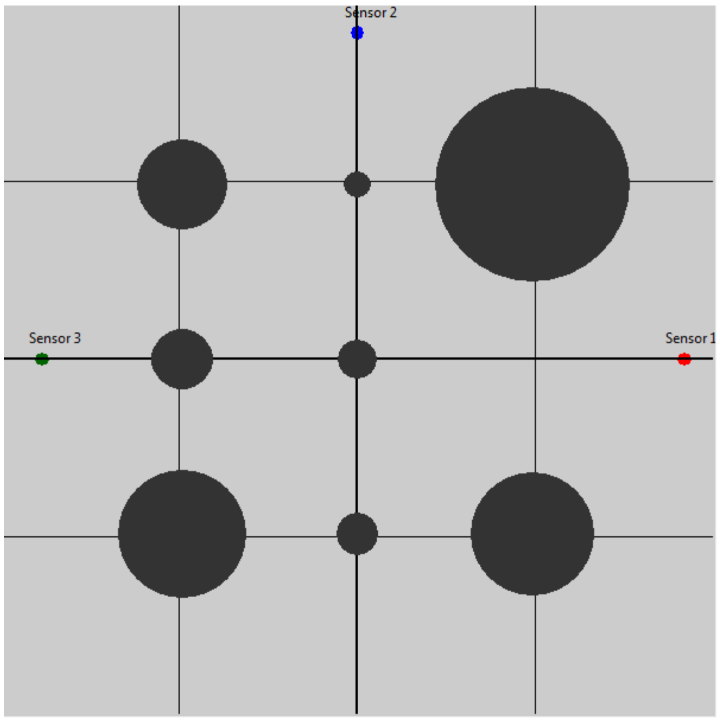
The plot to the left is a cutout of the left half the system graphics. The big, black circle centered at (0, 0) is the area covered by the RMSE of the Least-squares fit algorithm. This is hugely inconsistent in the area where it is supposed to be the most precise, which is why the rest of the points were not tested.



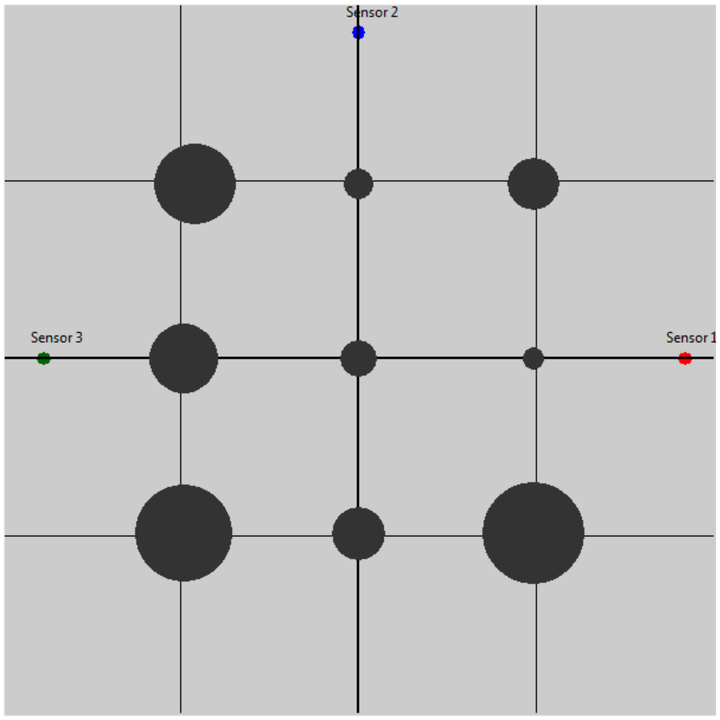
This plot shows the RMSE's of the standard thresholding method. Configurations used for this set of data were no spline fit, and a threshold of 50. The results on the left indicate more precisionalong the axes, especially the x-axis.std_spline_thesh50



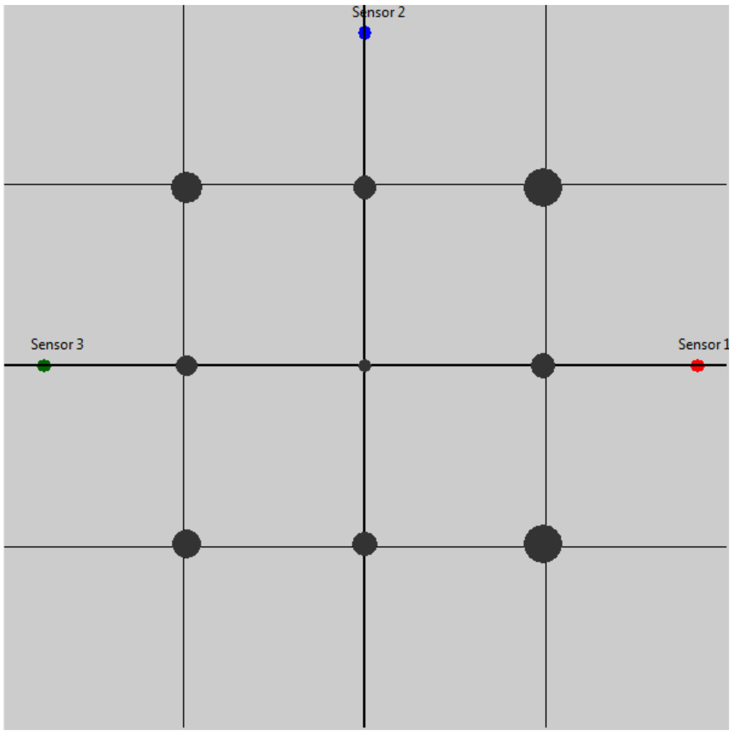
Here is almost the same configuration as above, except that a spline-fit was applied to the data. Evidently, this helped cut down the scattering of data and made all the locations more precise. Even the center (0, 0) gained some.



Configurations here were standard threshold algorithm with no spline fit and a threshold of 35. It seems like lowering the threshold led to an increase in the scattering of the results. The big black circle at the top right is about 15 centimeters in diameter, which is not a very satisfying result



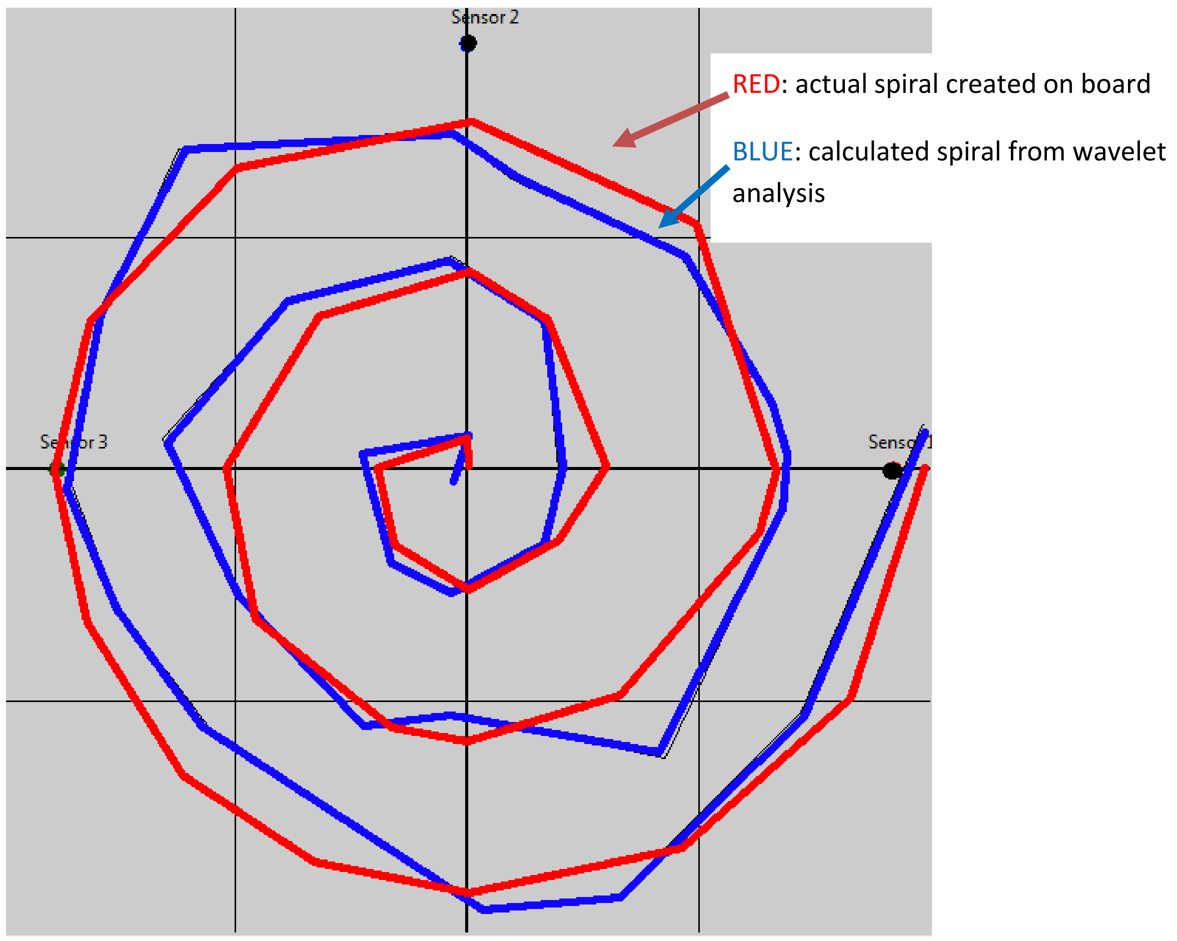
Applying a spline fit to the situation above seems to drastically improve the repeatability of the data. The large circle to the top right has shrunk to the about the average for results from the standard threshold analysis method.



Lastly, this is the same type of representation as before, but for a different analysis method, the ‘wavelet analysis’ algorithm. Just by observation of the diagram to the left, and comparison to the previous five, it is obvious that wavelets offer the best repeatability out of any of the previously attempted analysis.

## Accuracy and Repeatability at continuously changing targets

The following diagram is a cutout of the graphing section of the regular Python program.

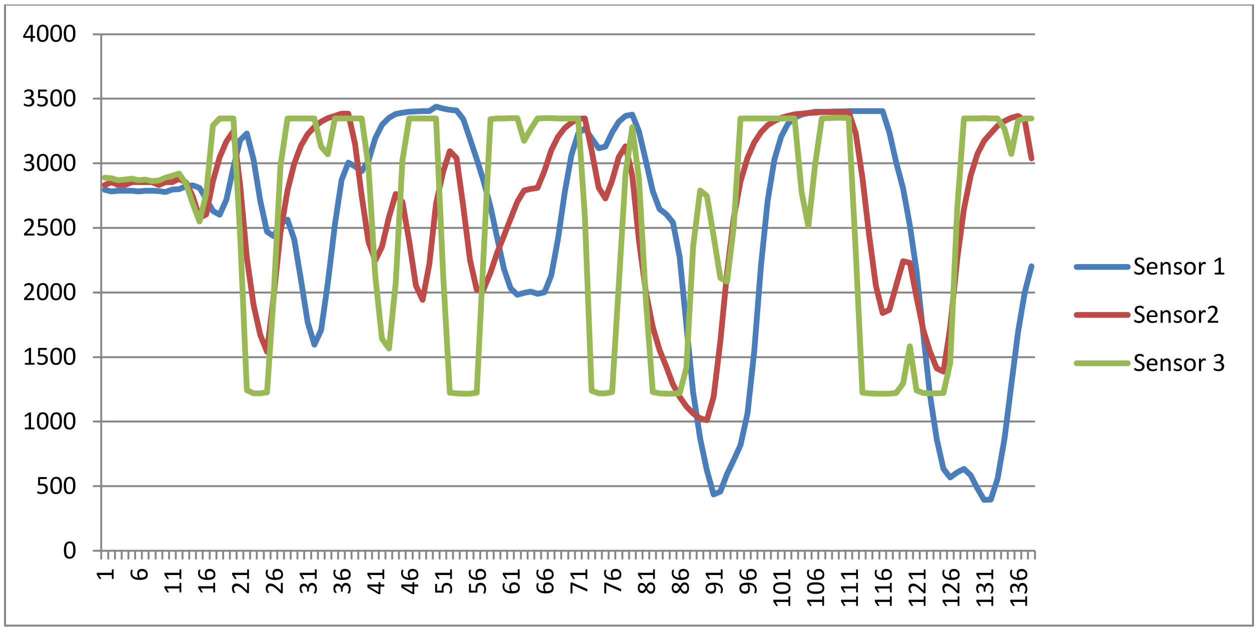


The red spiral is the actual spiral that was replicated onto the physical board with 31 impacts. Data analysis was done using the "wavelet analysis" method. The resulting intersections of the hyperbolas for each impact were recorded, and afterward reconnected on the plot, creating the blue spiral. In essence, this is a measure of how accurate the system is continuously and at varying positions, as opposed to repeated single-location impacts as in the chart presented earlier.

The recreated spiral indicates that it the system's wavelet analysis is actually quite accurate everywhere on the board. Toward the bottom left, the results became a little skewed, but then reconnected with the accuracy that the system displayed before as the spiral progressed.

A closer look at the Data

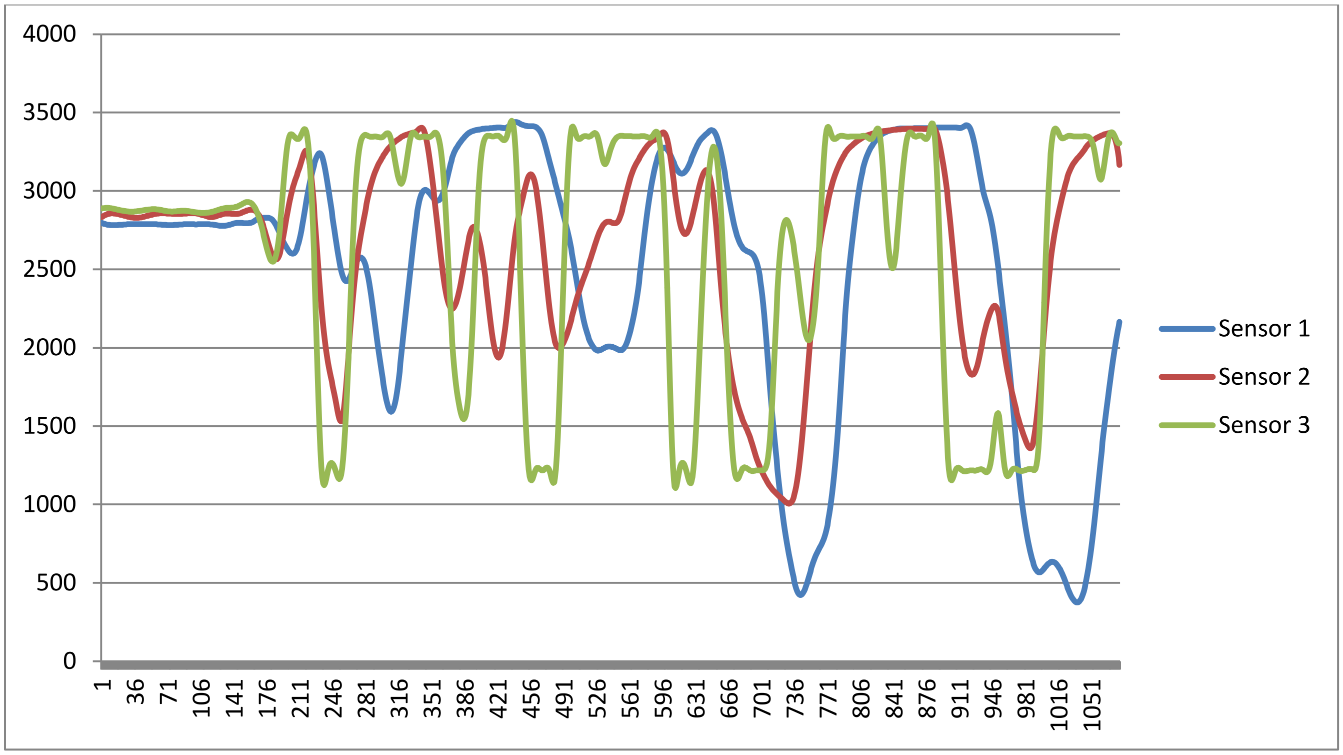
It is worth it to take a closer look at the data being processed and manipulated, and how certain options affect it. The following graph is of an impact near the center, using the wavelet approach (however, the analysis method does not directly affect the basic plot of the data).



It's a fairly straightforward graph. In it there are a few interesting details that can be picked out. For one, the time resolution is not very good, as seen by how jagged the graph is. This means that even with good software analysis, there is an inherent data insufficiency and temporal inaccuracy.

Secondly, it's important to note that all three sensors top out at about the value of 3300. This is a hardware challenge that could be overcome with a different accelerometer that can tolerate a greater range of accelerations. Interestingly, the green sensor also *bottoms* out at a certain value, which is not expected, and still unexplained. The first two sensors seem to have no problem going lower than value 1200.

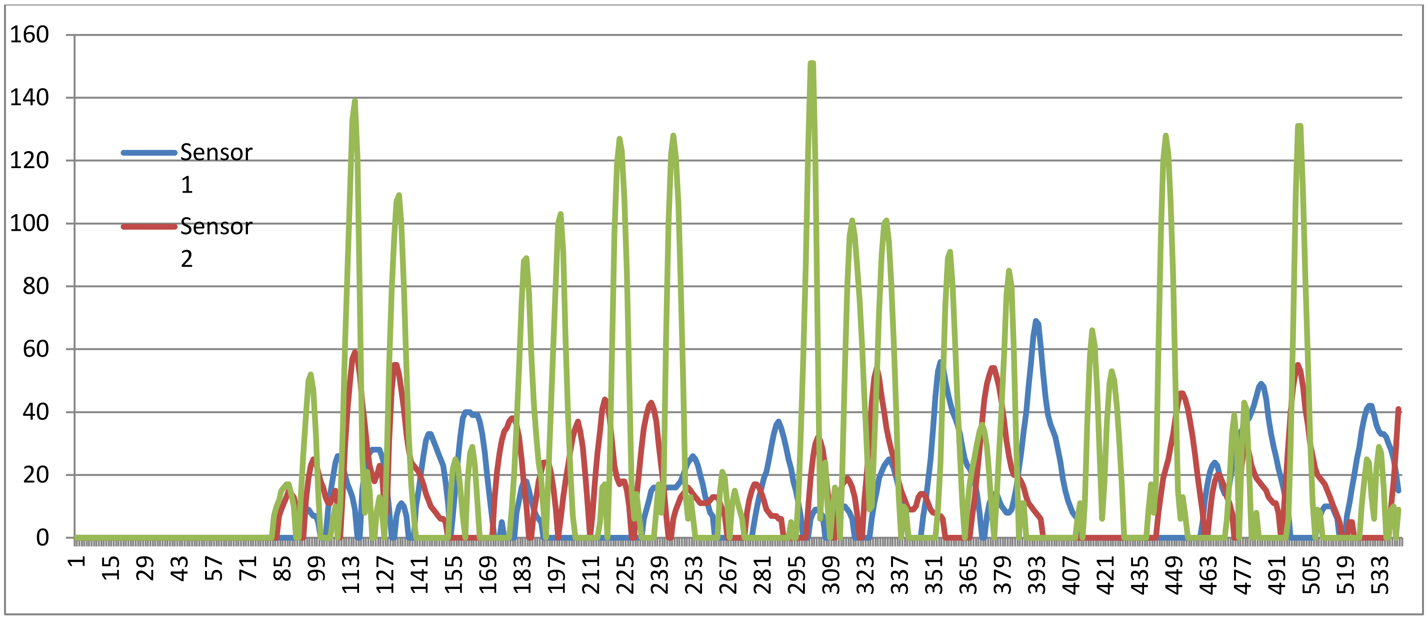
Though the wavelet analysis method required a spline fit to create regularly spaced data out of discrete, non-regular data, the spline can also be applied any of the methods, at the expense of about a second of processing power.



This is the same data after the application of a spline fit. Note that there are many more samples than before (about 6 times as many!). The spline method supplied in the Scipy library allows an arbitrary number of points to be generated from a set of data. In the program, this capability is used to smooth out data and access intermediate values.

One disadvantage of using the spline fit is the amplification of noise. Because the library tries to fit a cubic through points, small changes get accented, which is normally not desired. For this reason, calibration factors and shifts were added into the program. Real-time adjustable thresholds also help to make quick changes to the program to aid in accuracy.

One last interesting close-up is the wavelet coefficient data produced by the Pywavelets waveletdec() function. The following figure is of rectified and thresholded coefficients returned by that function. This means that any values closer to 0 than the threshold will be replaced with zero. In this case, the default threshold was used, 5.7.



The rectification and the threshold seem to be functioning as they should. The result is that the surviving coefficients point toward the start of the wave. Because of the thresholding and the inherent nature of wavelets, the wavelet-based data analysis is more robust with regard to gradual level changes and small noise.

Although it was originally guessed that the wavelet transform would pick up high frequency, low amplitude patters the eye could not see, results seem to indicate that the wavelet coefficients become significant about the same time the eye can start to tell a difference in the data, and not before. It is unknown if this is because there is not enough temporal resolution, there are actually no faster, low amplitude, high frequency waves produced, or that such waves simply decay too quickly to travel very far.

## Other Factors and Results

Throughout this project, hardware and physical components have been the single largest source of error. The switch to the Maple board failed, preventing faster sampling speeds and better resolution. The bad readings that come through are frequent, and have to be filtered out, slowing the rest of the program down.

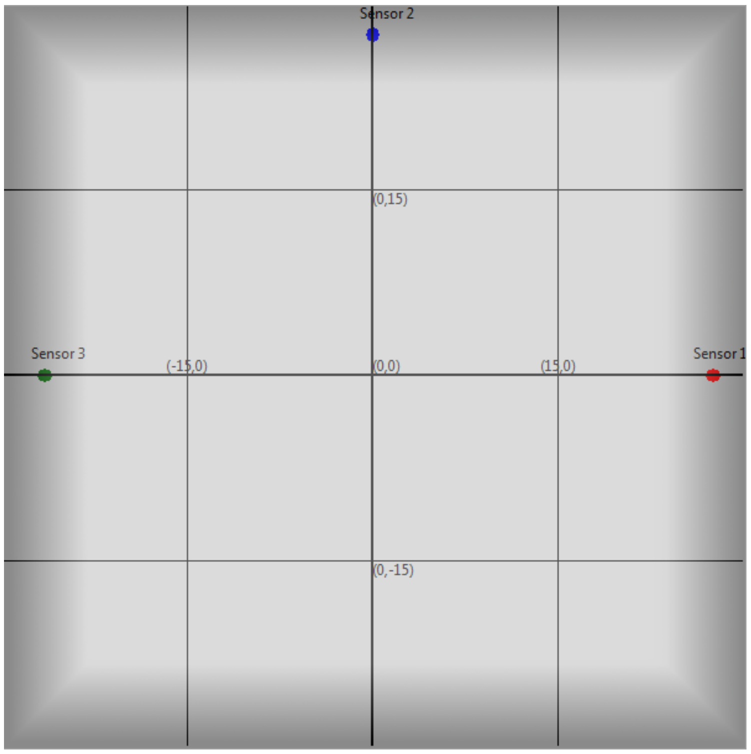
Significant about the random noise that cause bad readings is the event that occurs about 5% of the time. This is where the data that comes through is actually the tail end of the wave that needs be caught at the beginning. It is suspected that this occurs when the Arduino triggered data collection due to some noise, and then reads some interference to produce strange data the computer program does not know how to filter out.

Even more frequent are single spikes or drops in ADC readings that cause the Arduino to trigger data collection and transmission. Because the noisy values are almost always the same (the minimum 0 or the maximum 4095), the program has an exclude list to check and use as a filter. However, without this process, the system would fail. The spikes and drops cause bad readings about once every two seconds, often more frequently than actual data readings. The main problem with the noise is the need to devote precious processing time and power to the filtering process. While this is happening, or data is being captured on the Arduino or being transmitted to the computer, actual data that arrives a split second later is missed or discarded as well.

## Conclusions

This project is a success in its main original venture: to show that it is possible to locate an impact in 2-D purely based on the vibrations it causes. Although consistency and accuracy are two things that can still be refined, the idea has been shown to be viable.

*Inaccurate regions of the board*



A major conclusion that can be based on the data and results obtained in the previous section is that the wavelet-based analysis method using high-frequency oscillations produces the best results by drastic amounts. The standard threshold method produces respectable results (enough to show that the system can compute the location, but can also do it much better in other ways). Within the standard threshold method, changing the threshold parameter does not seem to produce much different results, but applying a spline fit before analyzing the data does provide a noticeable increase in accuracy and repeatability.

To the lighter areas to the right are the approximate areas where the standard threshold algorithm and wavelet analysis yield good results (least mean squares algorithm does not produce accurate results in most places). Wavelets can produce some close guesses up until the very edges of the board, but repeatability becomes a problem at that point too.

Future iterations of this project might include a fourth sensor to provide more accurate readings and push back the grey area shown here.

## Problems and Difficulties

Even though it was intended to be a largely computer science driven project, it originally also had a very complicated hardware aspect. This was finding the right accelerometer. In a high speed application such as this one, bandwidth and sensitivity are key. The sampling speed needed disqualified almost all reasonably priced digital accelerometers, which are often limited by the sampling speed of the analog-digital converter. Also, since common applications of accelerometers don't need fast sampling speeds at all, the easiest to find analog chips are made for tilt sensing (such as in phones). Adding on top of those restrictions the fact that it has to be handled with a regular soldering iron (so no surface mount chips alone) and not too expensive, it took several days to find a sensor that would satisfy all the conditions.

Actually, many of the remaining limitations of this project stem from some hardware incapability. Firstly, the sampling speed and number of samples the Arduino can hold at once leave much to be desired and naturally limit accuracy and repeatability. Also, some of the random noise could be eliminated by designing printed circuit boards for the accelerometers, capacitors, and ADC's in a smaller and more reliable package, and finding better ways of mounting the sensor setups to the board.

A final limitation of the project is the requirement for a hard surface the monitor, and a hard object to produce the impact. Applications could be greatly expanded if soft surfaces and soft objects could generate vibrations. However, this is inherently difficult to do because soft surfaces do not transmit vibrations very well, and soft objects do not generate high amplitude, long distance waves.

## Future studies

Besides the basics such as upgrading the microcontroller, and updating the hardware to be more reliable, the software could be optimized for speed. Currently it takes about a second for the standard threshold algorithm, and adding in a spline fit costs another second. The wavelet analysis can sometimes take up to three seconds to compute and display. If these times were reduced, results could be shown more promptly, and data could be fed into the computer more quickly, reducing the two or three second pause required between readings.

A separate study branching off of this one to consider would be a comparison between using accelerometers like this year and piezoelectric elements like before, and seeing which system produces more accurate results and more manageable hardware for the same goal. Also, a study of the effects of placing the board on stilts versus on the ground or on a supporting sheet of material and accuracy resulting from these changes could be done.

Because of this project's similarity to modern ultrasound ranging devices, naval sonar imaging, and microphone-array based 3-D localization of sounds in space, it is a good stepping stone into even more complicated projects. Most likely, the basic principles remain the same, and the algorithms and mathematics grown in scope.

<u>Applications</u>

If processing speed could be increased and accuracy could be ensured consistently at a high level, this system could find many useful applications. For instance, spacecraft could have a system of these on their metal skin that detect miniscule impacts caused by asteroids and debris crashing into the satellites. This would reduce diagnostic and repair time overall. Floors could be seeded with sensors to detect footsteps and determine where they are. Falling raindrops could be detected and turned into a model. Perhaps more interesting and applicable to everyday people is the possibility of mounting this system on house walls or paneling. The software could be reconfigured to detect taps or impacts on regions of the wall which serve different functionality such as turning on lights or heating. If the system is tuned to extreme accuracy, it could find a use as a vibration based keyboard or a cheap, DIY touch screen. Of course, things like tap and drag would not work, but these could be improvised by double taps to make connecting lines or something similar.

Overview of applications:

- Impact detection on space- and aircraft skins
- Intruder detection
- Lighting and general home control from a wall-mounted panel
    - Each area of the board could be configured for lights, heating, etc
    - Could be customized and changed easily at will
- Large invisible keyboard on a hard surface
- Cheap, large DIY touch[knock]screen
- General impact detection and localization

<u>Wrap up</u>

This year’s project implemented the full idea the previous year hoped to accomplish. Because of complications, difficulty, and inexperience, last year’s work was cut down to a small scale, but still provided some of the background detail needed to make this year’s work feasible. In the end, this project was also a learning experience. The results turned out to be a much more complicated than expected and created a system worthy of future time investment for improvements.

## Background Research

Although this project is based on similar hardware compared to last year, and the background research is similar to an extent, there are some interesting new things I included in the project that warrant their own research. The most notable things include the similarity of the system with the radio-wave based LORAN system used in pre-GPS time, and the decision to use accelerometers over last year's piezoelectric sensors.

However, starting with the basis of the system, a new decision had to be made regarding the microcontroller that was used. The standard Arduino Uno R3 that was used in the previous project due to its ubiquitousness was just barely fast enough for the two sensors that were employed then. Even with optimization, which was already included last year to make the system more accurate, the 48 to 52 microsecond response time (for 2 sensors) would not have been improved doubly to achieve the same response time for three sensors, which are required for this project.

Therefore, with speed requirements kept in mind, the selection process for a faster but equally user-friendly and capable interface was sought for. After looking at the more powerful Arduino Mega, the decision was struck to go with the Maple microcontroller, designed by Leaf Labs. It is built using the same pin layout as the Arduino in order to achieve compatibility and access the wide user base that the Arduino already has. However, although the Maple is the same size, the processing power is much greater. By specification, the Maple has a 72 MHz clock speed on its ARM Cortex M3 (citation), and 128 KB of flash memory. The Arduino has a 16 MHz clock and 32 KB of flash memory. The clock speed plays a huge role later in conversing with the analog-digital converter, as higher clock speed means faster data transfer and updates. However, as most of the processing time is spent calculating averages, storing data, and taking timestamps, there is more to be gained from the faster clock in internal operations than serial SPI communications.

Another advantage of moving to the Maple is that the extra processing power comes with a small learning curve. Only some prerequisite software is required, and the programming of the Maple is designed to be as similar as possible to the Arduino. Luckily, since much of the last project was gaining experience in the micro-world and high speed electronics, that knowledge will not have to be relearned very much.

Moving onto more hardware related concepts, the chip between the sensor and the digital processor is the analog-digital converter (ADC). This year, the choice was made to keep using the same through-hole ADC that was used before. This is the Burr Brown ADS7818, 500 KHz bandwidth analog-digital converter. It communicates using the SPI interface, which is convenient because the Maple has full SPI support built in. For reliability, the provided SPI library will be used rather than recreate it for experience, though this is a possible extension of the project that would be purely for learning.

Finally, the decision on what kind of sensor to use turned out to be the most challenging part of the hardware process. Last year, the piezoelectric sensor was the basis of converting the sound into electricity. This was both a simple and a complicated system. The advantages included that they were flexible and could be mounted to many different surfaces easily. On top of that, they have bandwidth (maximum sampling speed for analog sensors essentially) that is limited only by the sampling speed of the ADC. The main disadvantage of piezoelectric sensors is the need for an operational amplifier to both shift the level of the analog signal (which fluctuates into negative voltage, a range that ADC's can't normally read) and amplify it to a level in which the ADC gets reliable readings. As this year it was desired to work on more computer science aspects of the project, rather than electrical engineering, one advantage of the accelerometer is the absence of the complicated and confusing operational amplifier. Since the axis that is being polled is the vertical one, and accelerometers inherently measure the static acceleration due to gravity, this shifts the signal being received well into the positive voltage range. Besides this, accelerometers come in a huge variety of specifications and sensitivities, so it would be simple to eliminate the need for an amplifier by finding a sensor that is more sensitive to accelerations.	There are a few important points to note that work against the accelerometers. One is that they have limited bandwidth. This means they can physically only follow the sinusoid at the given maximum frequency, and beyond that they are no longer as accurate as the spec sheet gives. Another downside is the higher performance accelerometers generally only come as surface mount packages, making hand-soldering almost impossible without specialized equipment. However, this can be avoided by buying evaluation boards of the sensors, but this brings up another disadvantage, the cost. While piezoelectric sensors generally run sub-10 dollars, accelerometers with evaluation boards often go for more than 20 dollars. This cost can be driven down, if it comes to that, with large scale production and using machines to attach the accelerometers.

So, as a result of this comparison, the accelerometer was chosen. The main reason was that the accelerometer has fewer hardware components to fiddle with, and cuts down the electrical engineering work. All that is required is the same analog-digital converter (ADS7818) and the accelerometer. Much time was spent searching for the right accelerometer, and finally one was found with large bandwidth, simple usage, and availability pre-soldered onto an evaluation board.

Already, accelerometers have found widespread use in all kinds of applications. For instance, new smart phones use them to find out if they are being held vertically or horizontally. Accelerometer use falls into a few categories: inertial measurement of velocity and position, vibration and shock measurement, and measurement of gravity to determine orientation. The use in smart phones falls into the last category. Some other uses of accelerometers are in airbag deployment and crash detection, free fall detection, monitor health of moving machinery and double integration of acceleration to find location and movement. Recently, they have also been employed in heart monitoring and other health applications.

On to the conceptual side of the project, there is a striking resemblance with the basis of this project with the now outdated LORAN (LOng RAnge Navigation) system. Hyperbolic navigation concepts were developed in the 1930's but WWII rushed it into practical use. Although the British were the first to use a hyperbolic navigation system, when the Americans joined they were able to provide coverage of 30% of the earth's surface. LORAN C, the successor to LORAN A developed during the war, terminated operations in the 1970's after it was superseded by satellite based navigation such as GPS. However, the fact that there have been several modernization and revitalization attempts attests to its success and reliability.

LORAN, navigation based on hyperbolas rests up on the principle of multilateration, the method of turning time differences into locations. The basics of this project are essentially what the LORAN system used: construct hyperbolas based on time differences in arrival, and intersect various hyperbolas to get a fix on the source of the location. However, the system made in this project works on much smaller time scales, not only because of smaller differences (tens of centimeters instead of hundreds of kilometers) but also because sound travels faster in solids than gases. On the other hand, solids should be more consistent because their properties aren't very affected by various fluctuations in temperature, wind, and other inconsistencies. Another difference in the implementation of LORAN is that multiple signals could be sent and re-evaluated to improve the accuracy of location fixes, whereas this project can practically only be expected to generate one pulse. Also, LORAN signals had a specialized frequency and pulse shape that made their measurement more predictable. For this system, it is expected to receive seemingly quite random signals, with only the first edge of the data being of interest (until wavelet transform is implemented).

An interesting thing mentioned in the LORAN navy document is that navigators had to exercise extra caution when they knew they were located in the baseline extension of two stations. In other words, they were close to the line extending beyond the two foci (the two receiving stations). This is dangerous because as hyperbolas flatten out toward a line, they lose accuracy (a small change in the parameter causes a huge change in the result). In this project, similar dead-zones are predicted. For this reason, the sensors will almost always be placed at the edges of the board to maximize the good areas of detection. The best detection rates will no doubt be inside the triangle formed by the three sensors.

In comparison to the best known usage of vibrations and waves to determine where they came from, an element of confusion might come up (addressed later). Most people know how earthquakes' epicenters are found, and it is not too hard to figure out with a class in geometry and algebra.  During an earthquake, a seismograph is written by the detection machine, and it contains various types of waves. A scientist could then identify the fast P waves and the slower S waves (Pennington). With the speed of each type of wave known (from previous measurements), the time difference is translated into a distance. Once three different seismometers have such readings, it is a simple matter to triangulate the epicenter of the earthquake.

One thing to note that is important and a common source of confusion is the difference between triangulation and multilateration, and this will probably be the biggest misconception with this project. Triangulation uses time difference between two signals generated from the same event, which can be translated into distances and then circles and intersections, while multilateration only uses differences between arrival at two different sensors to construct hyperbolas and solve these. This project does not use triangulation like earthquakes. It uses multilateration.

Research was also done into vibration based localization work that has been done previously. Many of these systems rely on complicated models and advanced algorithms for the purpose of active damage detection. These systems tend to be tools of civil and mechanical engineers and are often applied to large structures for health monitoring and safety. These mathematically rigorous systems exceed the scope of the problem that needs to be solved here. If the system contained joints in 3-D space, continuous vibration measurement from which damage information needs to be extracted, or other much more complicated extensions, a model of the entire structure may have been the best plan of action.

Modern damage testing has various ranges of complication. These range the ones discussed above to simple impact detection. In some cases, like in car crashes, piezoelectric crash elements are used to determine which airbag to deploy. Nowadays accelerometers are also used to check for these fast decelerations, which deploys the airbag. In other cases, models are developed for the accelerometers to determine if damage has been done. Intuitively, a structure will have a natural frequency and characteristics that determine what the accelerometer reads. Then, if something happens to this structure, the responses change. Several difficulties arise with this: for one, actual damage releases very high frequency propagations that the system may not be designed to catch them. It may be that system is designed to watch the general response of the whole structure, which indicates general health, but cannot detect point damage. Therefore, systems are often designed to certain levels, from just detecting a change, to detection, localization, quantification of damage, and prediction of lifetime. This project is a simple level 2: detection and localization. It will not use an advanced model or 3-d localization, and only rely on time differences acquired from accelerometers.

Overall, this background research has given insight into possible failures and uses of a system such as the one that will be made. The learning about accelerometers and bandwidth and sensitivity was highly interesting, and revealed things like the difference between analog bandwidth and digital bandwidth.

**Bibliography**

"A Beginner's Guide to Accelerometers." *Dimension Engineering*.Web. Nov.-Dec. 2011. <http://www.dimensionengineering.com/accelerometers.htm>.

"Accelerometers and Accelerometer Info." *Omega*. Web. 05 Feb. 2012. <http://www.omega.com/prodinfo/accelerometers.html>.

*Chapter 12: LORAN Navigation*. N.p.: Military, n.d. N. pag. Print.

De Abreu, G. L. C, J. F. Ribeiro, and V. Steffen, Jr. "Finite Element Modeling of a Plate with Localized Piezoelectric Sensors and Actuators." (n.d.): n. pag. Web.

Hubbard, James E. Smart Skin Sensor for Real-time Side Impact Detection and Off-line Diagnostics. Trustees of Boston University, assignee. Patent 5797623. 25 Aug. 1998. Print.

Scott W. Doebling, Charles R. Farrar, and Michael B. Prime , "A SUMMARY REVIEW OF VIBRATION-BASED DAMAGE IDENTIFICATION METHODS ." (n.d.): n. pag. Web.

# Appendix A: Software

## Microcontroller (Arduino – C++)

```
#include <SPI.h>

const int ADSS1= 6;   //first AD is pin 10
const int ADSS2 = 7;   //second AD is pin
const int ADSS3 = 8;
const byte TMP = B00111111;

//adjustable constants
const long SERIALSPEED = 57600;   ////Serial connection speed
const int RUNS = 3;        //number of times to refill array while (re)setting vars
const byte LENGTH = 10;     //CAN'T BE BIGGER THAN 256 (byte)
const int FAST_LENGTH = 127; //can't go any longer, out of memory!s
const unsigned int MinThreshHold = (40) * LENGTH;   //modify the () only
//MinThresHold can't exceed 65536/LENGTH (for LENGTH = 50 that would be 1310)
//just treat the () as if it were the comparison #

//first impact variables (continously checking/evaluating)
 int dataAD1[LENGTH];
 int dataAD2[LENGTH];
 int dataAD3[LENGTH];
 unsigned long dataTimes[LENGTH];
 unsigned long tmpt;
 long totAD1 = 0;
 long totAD2 = 0;
 long totAD3 = 0;
 int valAD1 = 0;    //long?
 int valAD2 = 0;
 int valAD3 = 0;
 byte counter = 0; //max 256

 unsigned long impactTime1;
 unsigned long impactTime2;
 unsigned long impactTime3;

 unsigned long impT1, impT2, impT3;

//second impact variables (continously sampling, then evaluating)
 int fastAD1[FAST_LENGTH];
 int fastAD2[FAST_LENGTH];
 int fastAD3[FAST_LENGTH];
 unsigned int timeStamp[FAST_LENGTH]; //converting unsigned longs into unsigned ints by subtracting starting time to save
mem
 unsigned long t;     //for filling timestamp array

//success variable!
int successful = 0;

void setup() {
//configure pins as outputs
  pinMode(ADSS1, OUTPUT);
  pinMode(ADSS2, OUTPUT);
  pinMode(ADSS3, OUTPUT);
  //set up SPI interface
  SPI.setBitOrder(MSBFIRST);
  SPI.setClockDivider(SPI_CLOCK_DIV2);
  SPI.setDataMode(SPI_MODE0);
  SPI.begin();
  setVars();
}


void loop() {
while (true) {
//acquire data and assemble
  PORTD &= B10111111;
  byte MSB1 = SPI.transfer(0);
  byte LSB1 = SPI.transfer(0);
  PORTD |= B01000000;
  valAD1 = ((MSB1 & TMP) << 6) | (LSB1 >> 2);

  PORTD &= B01111111; // pull SS (pin 10) of arduino LOW (ADC sample)
  byte MSB2 = SPI.transfer(0);
  byte LSB2 = SPI.transfer(0);
  PORTD |= B10000000; // pull pin 10 of arduino to HIGH (ADC off) = PORT MANIPULATION
  valAD2 = ((MSB2 & TMP) << 6) | (MSB2 >> 2);

  PORTB &= B11111110; // pull SS (pin 9) of arduino LOW (ADC sample)
  byte MSB3 = SPI.transfer(0);
  byte LSB3 = SPI.transfer(0);
  PORTB |= B00000001; // pull SS (pin 9) of arduino HIGH (ADC off)
  valAD3 = ((MSB3 & TMP) << 6) | (LSB3 >> 2);

  impactTime1 = micros();
  tmpt = impactTime1;

//recording and math
        if (abs(totAD1-((long)valAD1*LENGTH)) >= MinThreshHold) {
          collectFast();  //fill data arrays fastAD1, fastAD2, fast AD3, timeStamp
          impT1 = impactTime1;
          successful = dataCrunch(fastAD2, fastAD3, dataAD2, dataAD3, totAD2, totAD3, 2,3); //crunch second data array
          if (successful == 1) {
            transmit(); //transmit over //Serial all the info along with timeDelay
          }
          setVars(); //reset the data arrays and totals
          continue;
        }

        //if it was the second ADC that registered the tap
        else if (abs(totAD2-((long)valAD2*LENGTH)) >= MinThreshHold) {
          collectFast();  //fill data arrays
          impT2 = impactTime1;
          successful = dataCrunch(fastAD1, fastAD3, dataAD1, dataAD3, totAD1, totAD3, 1,3); //crunch first data array
          if (successful == 1) {
            transmit(); //transmit over //Serial all the info along with timeDelay
          }
          setVars(); //reset the data arrays and totals
          continue;
        }

       else if (abs(totAD3-((long)valAD3*LENGTH)) >= MinThreshHold) {
          collectFast();
          impT3 = impactTime1;
          successful = dataCrunch(fastAD1, fastAD2, dataAD1, dataAD2, totAD1, totAD2, 1,2);
          if (successful == 1) {
            transmit();
        }
        setVars(); //reset the data arrays and totals
        continue;
      }

    totAD1 -= dataAD1[counter];  //subtract the next spot, which is the oldest value
    dataAD1[counter] = valAD1;                          //write into that spot
    totAD1 += valAD1;                                   //add the new value

    totAD2 -= dataAD2[counter];
    dataAD2[counter] = valAD2;
    totAD2 += valAD2;

    totAD3 -= dataAD3[counter];
    dataAD3[counter] = valAD3;
    totAD3 += valAD3;
    dataTimes[counter] = tmpt;
  counter++;                                          //increase counter
  if (counter == LENGTH) {                  //reset counter every LENGTH, faster than doing it every largest multiple of LENGTH
that fits in an int
      counter = 0;                          //reset to 0; all this is equivalent to counter%LENGTH but then counter would
overflow after 65536
    }
}

}
//end of loop()

void setVars() {
  int nums = LENGTH*RUNS; //to account for spike or drop at beginning

//reset arrays
  for (int x = 0; x < LENGTH; x++) {
    dataAD1[x] = 0;
    dataAD2[x] = 0;
    dataAD3[x] = 0;
    dataTimes[x] = 0;
  }

  totAD1 = 0;
  totAD2 = 0;
  totAD3 = 0;
  counter = 0;
    for (int i = 0; i < 100; i++) {
      PORTD &= B10111111;
      SPI.transfer(0);
      SPI.transfer(0);
      PORTD |= B01000000;
      PORTD &= B01111111; // pull SS (pin 10) of arduino LOW (ADC sample)
      SPI.transfer(0);
      SPI.transfer(0);
    PORTD |= B10000000;
     PORTB &= B11111110; // pull SS (pin 9) of arduino LOW (ADC sample)
      SPI.transfer(0);
      SPI.transfer(0);
      PORTB |= B00000001; //
    }

 // tmp1 = micros();
```

```
  for (int x = 0; x < nums; x++) {
    PORTD &= B10111111;
    byte MSB1 = SPI.transfer(0);
    byte LSB1 = SPI.transfer(0);
    PORTD |= B01000000;
    valAD1 = ((MSB1 & TMP) << 6) | (LSB1 >> 2);

    PORTD &= B01111111; // pull SS (pin 10) of arduino LOW (ADC sample)
    byte MSB2 = SPI.transfer(0);
    byte LSB2 = SPI.transfer(0);
    PORTD |= B10000000; // pull pin 10 of arduino to HIGH (ADC off) = PORT MANIPULATION
    valAD2 = ((MSB2 & TMP) << 6) | (LSB2 >> 2);

    PORTB &= B11111110; // pull SS (pin 9) of arduino LOW (ADC sample)
    byte MSB3 = SPI.transfer(0);
    byte LSB3 = SPI.transfer(0);
    PORTB |= B00000001; // pull SS (pin 9) of arduino HIGH (ADC off)
    valAD3 = ((MSB3 & TMP) << 6) | (LSB3 >> 2);

    t = micros();
    totAD1 -= dataAD1[counter];  //subtract the next spot, which is the oldest value
    dataAD1[counter] = valAD1;                    //write into the current spot
    totAD1 += valAD1;                             //add the new value

    totAD2 -= dataAD2[counter];
    dataAD2[counter] = valAD2;
    totAD2 += valAD2;

    totAD3 -= dataAD3[counter];
    dataAD3[counter] = valAD3;
    totAD3 += valAD3;

    dataTimes[counter] = t;
    counter++;                                  //increase counter
    if (counter == LENGTH) {              //reset counter ever LENGTH, faster than doing it every largest multiple of LENGTH that
fits in an int
      counter = 0;                        //reset to 0; all this is equivalent to counter%LENGTH but then counter would overflow
after 65536
    }
  }
  counter = 0;
  successful = 0; //reset success byte variable
}

//function to fill fast data arrays with data
void collectFast() {
    for (int x = 0; x < FAST_LENGTH; x++) {
      PORTD &= B10111111;
      byte MSB1 = SPI.transfer(0);
      byte LSB1 = SPI.transfer(0);
      PORTD |= B01000000;

      PORTD &= B01111111; // pull SS (pin 10) of arduino LOW (ADC sample)
      byte MSB2 = SPI.transfer(0);
      byte LSB2 = SPI.transfer(0);
      PORTD |= B10000000; // pull pin 10 of arduino to HIGH (ADC off) = PORT MANIPULATION

      PORTB &= B11111110; // pull SS (pin 9) of arduino LOW (ADC sample)
      byte MSB3 = SPI.transfer(0);
      byte LSB3 = SPI.transfer(0);
      PORTB |= B00000001; // pull SS (pin 9) of arduino HIGH (ADC off)

      //add data to arrays
      fastAD1[x] = ((MSB1 & TMP) << 6) | (LSB1 >> 2);
      fastAD2[x] = ((MSB2 & TMP) << 6) | (LSB2 >> 2);
      fastAD3[x] = ((MSB3 & TMP) << 6) | (LSB3 >> 2);
      timeStamp[x] = micros() - dataTimes[(counter)%LENGTH];
    }
}

int dataCrunch(int fastData_1[], int fastData_2[], int origData_1[], int origData_2[], long total_1, long total_2, int sensornum1,
int sensornum2) {

  int combData_1[FAST_LENGTH + LENGTH]; //for hooking together old data and new fast data
  int combData_2[FAST_LENGTH + LENGTH];

  //copy older and faster new data into single longer array
  for (int x = 0; x < LENGTH; x++) {
    combData_1[x] = origData_1[(counter+1+x)%LENGTH]; //assign oldest value first: counter is newest value, so
(counter+1)[oldest] + (x)[increment] = oldest to newest
    combData_2[x] = origData_2[(counter+1+x)%LENGTH];
  }

  for (int x = 0; x < FAST_LENGTH; x++) {
    combData_1[x+LENGTH] = fastData_1[x];
    combData_2[x+LENGTH] = fastData_2[x];
  }
  int found1 = 0;
  int found2 = 0;

  for (int x = LENGTH; x < (FAST_LENGTH+LENGTH); x++) {
    if (abs(total_1 - (combData_1[x]*LENGTH)) >= MinThreshHold && found1 != 1) { //if the current total-val*LENGTH is greater
than the set threshold
      impactTime2 = timeStamp[x-LENGTH];
      if (sensornum1 == 1) {
        impT1 = impactTime2;
      }
      else if (sensornum1 == 2) {
        impT2 = impactTime2;
      }
      else if (sensornum1 == 3) {
        impT2 = impactTime2;
      }
      found1 = 1;
    }
    if (abs(total_2 - (combData_2[x]*LENGTH)) >= MinThreshHold && found2 != 1) {
      impactTime3 = timeStamp[x-LENGTH];
      if (sensornum2 == 1) {
        impT1 = impactTime3;
      }
      else if (sensornum2 == 2) {
        impT2 = impactTime3;
      }
      else if (sensornum2 == 3) {
        impT3 = impactTime3;
      }
      found2 = 1;
    }
    if (found1 == 1 && found2 == 1) {
      return 1;
    }
    total_1 -= combData_1[x-LENGTH]; //remove oldest data in list
    total_1 += combData_1[x];        //add newer data

    total_2 -= combData_2[x-LENGTH];
    total_2 += combData_2[x];
  }
  return 0; //failed to find anything :(
}


int transmit() {
 Serial.begin(SERIALSPEED);
  for (int i = counter; i < counter+LENGTH; i++) {
    Serial.print(dataAD1[i%LENGTH]);
    Serial.print(" ");
    Serial.print(dataAD2[i%LENGTH]);
    Serial.print(" ");
    Serial.print(dataAD3[i%LENGTH]);
    Serial.print(" ");
    Serial.println(dataTimes[i%LENGTH]-dataTimes[counter]);
  }
  Serial.print(valAD1); //these were never stored in the array before checking but they need to be included anyway
  Serial.print(" ");
  Serial.print(valAD2);
  Serial.print(" ");
  Serial.print(valAD3);
  Serial.print(" ");
  Serial.println(impactTime1-dataTimes[counter]);
  for (int i = 0; i< FAST_LENGTH; i++) {
    Serial.print(fastAD1[i]);
    Serial.print(" ");
    Serial.print(fastAD2[i]);
    Serial.print(" ");
    Serial.print(fastAD3[i]);
    Serial.print(" ");
    Serial.println(timeStamp[i]); //subtration already factored in in collectFast
  }
  Serial.println("DONE");

  Serial.end();
    delay(100); //estimated amount of time needed for return to normal behavior
  return 1;
}
```

# Appendix B: Software

## Data Analysis – Python 2.7

```
from Tkinter import *
import numpy as np
import tkSimpleDialog
import serial
import math
import pywt
from scipy.interpolate import interp1d
from operator import add
from operator import sub


ser = serial.Serial('COM4', 57600,timeout=1)
exclude = (0,1023, 4094, 4095) #values qualified as bad data, remove that time's data from all 3 to maintain consistency
plotRectified = False
useSplineFit = False

#debugging options
VERBOSE = False
VERBOSE_VERBOSE = False
DUMP_FILE = "DUMP"

THRESHOLD = 50                                    #standard averaging algorithm's threshold
LINE_FIT_THRESHOLD = 40                #through-peak line fit threshold
MICROS_PER_SAMPLE = 4                  #wavelet sample spacing
WAVELET_THRESHOLD = 5.7                #wavelet algorithm's threshold
LEVEL_TO_USE = -1                                 #wavelet level to use, counting from the back (-1 -> highest resolution)

SCALE_FACTOR_1 = 1.2
SCALE_FACTOR_2 = 0.7
SCALE_FACTOR_3 = 0.6
RED_START_OFFSET = 30
RED_TWEAK = 84 #subtract from sensor 1 time... all sensor1 data is 'late' compared to other two, so all data is shifted left
BLUE_TWEAK = -14

AV_DIST = 8

INTERPOLATE_KIND = "cubic" #can be: ('linear','nearest', 'zero', 'slinear', 'cubic') or as an integer specifying the order of the
spline interpolator to use. Default is 'linear'.

defaults = {'THRESHOLD': 50, 'LINE_FIT_THRESHOLD':40, 'MICROS_PER_SAMPLE':4, 'WAVELET_THRESHOLD':5.7,
'LEVEL_TO_USE':-1}

length = 61 #width of entire board in cm
size = 600 #size of board in pixels
ratio = 600.0/length #for conversion between pixels and cm
                                                  #needs to be changed if square canvas is no longer length 700
sensors = {1:(28*ratio,0),2:(0,28*ratio),3:(-27*ratio,0)} #these are in humanXY already


sqrt = math.sqrt
cos = math.cos
```

```
sin = math.sin
atan = math.atan
pi = math.pi


colors  = {1:'red', 2:'blue', 3:'dark green'}


class SciFair:
        global THRESHOLD
        global LINE_FIT_THRESHOLD
        global MICROS_PER_SAMPLE
        global WAVELET_THRESHOLD
        def __init__(self, parent):
                self.parent = parent
                #----------------------------- Set up frames and canvases -----------------------------
                self.outerFrame = Frame(parent,width=1200, height=650)
                self.canvasFrame = Frame(self.outerFrame, width=size,height=size)
                self.plotFrame=Frame(self.outerFrame, width=600, height=600)
                self.canvas = Canvas(self.canvasFrame, width=600, height=600,bg="#CCCCCC")
                self.plot = Canvas(self.plotFrame, width=600, height = 200, bg="#CCCCCC")
                self.waveletsPlot = Canvas(self.plotFrame, width=600,  height=200, bg="#CCCCCC")
                self.controlFrame1 = Frame(self.plotFrame, width=600, height=50, bg="#DDDDDD")
                self.controlFrame2 = Frame(self.plotFrame, width=600, height=50, bg="#DDDDDD")
                self.controlFrame3 = Frame(self.plotFrame, width=600, height=50, bg="#DDDDDD")
                self.controlFrame4 = Frame(self.plotFrame, width=600, height=100, bg="#DDDDDD")
                #self.outerFrame.pack_propagate(0)
                self.outerFrame.pack()
                self.canvasFrame.pack(side=LEFT,anchor=N)
                self.plotFrame.pack(side=LEFT,anchor=N)
                self.controlFrame4.pack(side=BOTTOM, anchor=S)
                self.controlFrame3.pack(side=BOTTOM, anchor=S)
                self.controlFrame2.pack(side=BOTTOM, anchor=S)
                self.controlFrame1.pack(side=BOTTOM, anchor=S)
                self.canvas.pack(side=TOP, anchor=W)
                self.plot.pack(side=TOP)
                self.waveletsPlot.pack(side=TOP)

                #---------------------------------- Class variables ----------------------------------
                self.maxTime = 1150                             #estimated time needed to cross 54 cm
                self.speed = ratio*55.0/1050 #cm/us -> pixels/us
                self.lineX1 = 0
                self.lineY1 = 0
                self.lineX2 = 0
                self.lineY2 = 0
                self.rightClickState = 0
                self.firstTime = True
                self.data = []
                self.tmpData = []
                self.origData = []
                self.splinedData = []
                self.scaledTimeData = {}
                self.scaledWaveletData = {}
                self.rectifiedData = []
                self.waveletData = []
                self.scale = 1
                self.analysisType = "stdThresh"

                #-------------------------------------- Buttons -------------------------------------
```

```
			self.quitButton = Button(self.controlFrame1, text="Quit", fg="red", command = self.outerFrame.quit)
			self.quitButton.pack(side=LEFT, anchor=N)
			self.rToggle = Button(self.controlFrame1, text="Use R.hold",command = self.toggleRightClickFunc)
			self.rToggle.pack(side=LEFT, anchor=N)
			self.clrHyperb = Button(self.controlFrame1, text="Clear Curs", command = self.deleteCurs)
			self.clrHyperb.pack(side=LEFT, anchor=N)

			self.stdThresholdButton = Button(self.controlFrame2, text="Standard Threshold", command = 
self.analyzeThresh)
			self.slopeEstimate = Button(self.controlFrame2, text="Maximums Line Fit", command = self.analyzeLine)
			self.waveletButton = Button(self.controlFrame2, text="Wavelet Analysis", command = self.analyzeWavlet)
			self.stdThresholdButton.pack(side=LEFT, anchor=N)
			self.slopeEstimate.pack(side=LEFT, anchor=N)
			self.waveletButton.pack(side=LEFT, anchor=N)

			self.activStdThreshButton()

			self.plotRectifiedDataButton = Button(self.controlFrame3, text="Plot Rectified Data: %s" %("ON" if 
plotRectified else "OFF"), command = self.togglePlotRectified)
			self.plotRectifiedDataButton.pack(side=LEFT,anchor=N)

			self.useSplineFitButton = Button(self.controlFrame3, text="Use Spline Fit: %s" %("ON" if useSplineFit else 
"OFF"), command = self.toggleSplineUse)
			self.useSplineFitButton.pack(side=LEFT, anchor=N)

			self.dataDumpButton = Button(self.controlFrame3, text="Dump All Data", command = self.dataDump)
			self.dataDumpButton.pack(side=LEFT, anchor=S)

			self.avThreshSlider = Scale(self.controlFrame4, from_=5, to=80, orient=HORIZONTAL, label="std av Thresh", 
command = self.changeStdAvThresh)
			self.lineThreshSlider = Scale(self.controlFrame4, from_=5, to=80, orient=HORIZONTAL, label="line Fit 
Thresh" , command = self.changeLineFitThresh)
			self.waveletSamplingSlider = Scale(self.controlFrame4, from_=2, to=20, orient=HORIZONTAL, 
label="Wavelet resample spacing", command=self.changeWaveletSampleSpacing)
			self.waveletThreshSlider = Scale(self.controlFrame4, from_=1.0, to = 20.0, 
resolution=0.1,orient=HORIZONTAL, label="Wavelet Coeff Thresh", command=self.changeWaveletThresh)
			self.avThreshSlider.pack(side=LEFT)
			self.lineThreshSlider.pack(side=LEFT)
			self.waveletSamplingSlider.pack(side=LEFT)
			self.waveletThreshSlider.pack(side=LEFT)

			self.avThreshSlider.set(THRESHOLD)
			self.lineThreshSlider.set(LINE_FIT_THRESHOLD)
			self.waveletSamplingSlider.set(MICROS_PER_SAMPLE)
			self.waveletThreshSlider.set(WAVELET_THRESHOLD)

			self.resetValsButton = Button(self.controlFrame3, text="Reset Values", command = self.resetVars)
			self.resetValsButton.pack(side=LEFT)
			#-------------------------------------- Bindings ------------------------------------
			self.canvas.bind("<Button-3>", self.rightButton_canvas)
			self.plot.bind("<B3-Motion>", self.rightButton_plot)
			self.plot.bind("<Button-3>", self.rightButton_plot)
			self.waveletsPlot.bind("<B3-Motion>", self.rightButton_waveletsPlot)
			self.waveletsPlot.bind("<Button-3>", self.rightButton_waveletsPlot)

			#---------------------------------------- Labels --------------------------------------
			self.plot.create_text(10,140,text="Point:", anchor=W)

			self.canvasDisk(sensors[1],10, colors[1])
			self.canvasDisk(sensors[2],10, colors[2])
			self.canvasDisk(sensors[3],10, colors[3])
			self.drawFullLine((0,0,10,0), width=2);
			self.drawFullLine((0,0,0,10), width=2);

			self.canvas.create_text(map(add,self.canvasXY(sensors[1]),(-15,-10)), text="Sensor 1", anchor=SW)
			self.canvas.create_text(map(add,self.canvasXY(sensors[2]),(-10,-10)), text="Sensor 2", anchor=SW)
			self.canvas.create_text(map(add,self.canvasXY(sensors[3]),(-10,-10)), text="Sensor 3", anchor=SW)

			self.drawFullLine((-int(size/4),0,-int(size/4),10))
			self.drawFullLine((int(size/4),0,int(size/4),10))
			self.drawFullLine((0,-int(size/4),10,-int(size/4),10))
			self.drawFullLine((0,int(size/4),10,int(size/4),10))

			self.canvas.create_text(self.canvasXY((0,0)), text="(0,0)", anchor=SW)
			self.canvas.create_text(self.canvasXY((-int(size/4),0)),text="(-15,0)",anchor=S)
			self.canvas.create_text(self.canvasXY((int(size/4),0)),text="(15,0)",anchor=S)
			self.canvas.create_text(self.canvasXY((0,-int(size/4))),text="(0,-15)",anchor=NW)
			self.canvas.create_text(self.canvasXY((0,int(size/4))),text="(0,15)",anchor=NW)


			print "Waiting for Serial"
			#start cycle
			self.checkSerial()

		#---------------------------------------------------------------------------------------------
		#----------------------------------- Button-related Functions -------------------------------
		#---------------------------------------------------------------------------------------------
		def rightButton_canvas(self,event):
			self.parent.title("MOUSE \t\t Pixel: (%d,%d) \t\t Coordinate: (%d,%d) \t\t CM: (%d,%d)" %(event.x,event.y ,
	self.humanXY(event.x,event.y)[0],self.humanXY(event.x,event.y)[1] ,
	self.humanXY(event.x,event.y)[0]/ratio,self.humanXY(event.x,event.y)[1]/ratio))

		def rightButton_plot(self, event):
			self.plot.delete("dataLabel")
			closestKey = closestDictMatch(self.scaledTimeData,event.x)
			self.plotDisk(self.plotToHumanXY(event.x, event.y),2, tag="dataLabel")
			vals = self.scaledTimeData[closestKey]
			self.plot.create_text(10,150, text="Time: %d" %vals[3], tags="dataLabel", anchor=W)
			self.plot.create_text(10,160, text="%d" %vals[0], fill=colors[1], tags="dataLabel", anchor=W)
			self.plot.create_text(10,170, text="%d" %vals[1], fill=colors[2], tags="dataLabel", anchor=W)
			self.plot.create_text(10,180, text="%d" %vals[2], fill=colors[3], tags="dataLabel", anchor=W)

		def rightButton_waveletsPlot(self,event):
			self.waveletsPlot.delete("dataLabel")
			closestKey = closestDictMatch(self.scaledWaveletData,event.x)
			self.waveletPlotDisk(self.plotToHumanXY(event.x, event.y),2, tag="dataLabel") #don't need a new
	plotToHumanXY if two plots are same size
			vals = self.scaledWaveletData[closestKey]
			self.waveletsPlot.create_text(10,150, text="Time: %d" %vals[3], tags="dataLabel", anchor=W)
			self.waveletsPlot.create_text(10,160, text="%f" %vals[0], fill=colors[1], tags="dataLabel", anchor=W)
			self.waveletsPlot.create_text(10,170, text="%f" %vals[1], fill=colors[2], tags="dataLabel", anchor=W)
			self.waveletsPlot.create_text(10,180, text="%f" %vals[2], fill=colors[3], tags="dataLabel", anchor=W)

		def toggleRightClickFunc(self):
			if self.rightClickState == 0:
				self.rToggle.configure(text="Use R.click")
				self.canvas.unbind("<Button-3>")
				self.canvas.bind("<B3-Motion>",self.rightButton_canvas)
		else:
			self.rToggle.configure(text="Use R.hold")
			self.canvas.unbind("<B3-Motion>")
			self.canvas.bind("<Button-3>",self.rightButton_canvas)
		self.rightClickState = not self.rightClickState

	def togglePlotRectified(self):
		global plotRectified
		plotRectified = not plotRectified
		self.plotRectifiedDataButton.configure(text="Plot Rectified Data: %s" %("ON" if plotRectified else "OFF"))

	def toggleSplineUse(self):
		global useSplineFit
		useSplineFit = not useSplineFit
		self.useSplineFitButton.configure(text="Use Spline Fit: %s" %("ON" if useSplineFit else "OFF"))

	def dataDump(self):
		output = open("%s.txt" %DUMP_FILE,'w')
		output.write("RAW DATA: \n\n")
		for x in range(0,len(self.tmpData)):
			output.write("%d\t%d\t%d\t%d\n" %(self.tmpData[x][0], self.tmpData[x][1], self.tmpData[x][2],
self.tmpData[x][3]))

		output.write("\n\n\n CLEANED DATA: \n\n")
		for x in range(0,len(self.origData)):
			output.write("%d\t%d\t%d\t%d\n" %(self.origData[x][0], self.origData[x][1], self.origData[x][2],
self.origData[x][3]))

		output.write("\n\n\n RECTIFIED DATA: \n\n")
		for x in range(0,len(self.rectifiedData)):
			output.write("%d\t%d\t%d\t%d\n" %(self.rectifiedData[x][0], self.rectifiedData[x][1],
self.rectifiedData[x][2], self.rectifiedData[x][3]))

		output.write("\n\n\n SPLINED DATA \n\n")
		for x in range(0,len(self.splinedData)):
			output.write("%d\t%d\t%d\t%d\n" %(self.splinedData[x][0], self.splinedData[x][1],
self.splinedData[x][2], self.splinedData[x][3]))

		output.write("\n\n\n WAVELET COEFFS -- post rectification and thresholding \n\n")
		for x in range(0,len(self.waveletData)):
			output.write("%d\t%d\t%d\t%d\n" %(self.waveletData[x][0], self.waveletData[x][1],
self.waveletData[x][2], self.waveletData[x][3])	)

		output.close()

	def resetVars(self):
		global	THRESHOLD
		global LINE_FIT_THRESHOLD
		global MICROS_PER_SAMPLE
		global WAVELET_THRESHOLD
		global LEVEL_TO_USE
		self.resetValsButton.configure(text="Resetting")
		THRESHOLD = defaults["THRESHOLD"]
		LINE_FIT_THRESHOLD = defaults["LINE_FIT_THRESHOLD"]
		MICROS_PER_SAMPLE = defaults["MICROS_PER_SAMPLE"]
		WAVELET_THRESHOLD = defaults["WAVELET_THRESHOLD"]
		LEVEL_TO_USE = defaults["LEVEL_TO_USE"]
		self.avThreshSlider.set(THRESHOLD)
		self.lineThreshSlider.set(LINE_FIT_THRESHOLD)
		self.waveletSamplingSlider.set(MICROS_PER_SAMPLE)
		self.waveletThreshSlider.set(WAVELET_THRESHOLD)
		self.resetValsButton.configure(text="Reset Values")

	def activStdThreshButton(self):
		self.stdThresholdButton.configure(bg="#BBBBBB")
		self.slopeEstimate.configure(bg="#EEEEEE")
		self.waveletButton.configure(bg="#EEEEEE")
	def activLineFitButton(self):
		self.stdThresholdButton.configure(bg="#EEEEEE")
		self.slopeEstimate.configure(bg="#BBBBBB")
		self.waveletButton.configure(bg="#EEEEEE")
	def activWaveletButton(self):
		self.stdThresholdButton.configure(bg="#EEEEEE")
		self.slopeEstimate.configure(bg="#EEEEEE")
		self.waveletButton.configure(bg="#BBBBBB")

	#-------------------------------------- ANALYSIS CALLERS ------------------------------------
	def analyzeThresh(self):
		self.activStdThreshButton()
		self.analysisType = "stdThresh"
		self.analyze(self.analysisType)
	def analyzeLine(self):
		self.activLineFitButton()
		self.analysisType = "lineFit"
		self.analyze(self.analysisType)
	def analyzeWavlet(self):
		self.activWaveletButton()
		self.analysisType = "wavelet"
		self.analyze(self.analysisType)

	def drawFullLineCaller(self):
		self.drawFullLine(self.lineX1,self.lineY1,self.lineX2,self.lineY2)


	#-------------------------------------- SLIDERS ------------------------------------
	def changeStdAvThresh(self,event):
		if self.firstTime:
			return 0
		global THRESHOLD
		THRESHOLD = self.avThreshSlider.get()
		print "Threshold: ", THRESHOLD
		self.analysisType = "stdThresh"
		self.activStdThreshButton()
		self.analyze(self.analysisType)

	def changeLineFitThresh(self,event):
		if self.firstTime:
			return 0
		global LINE_FIT_THRESHOLD
		LINE_FIT_THRESHOLD = self.lineThreshSlider.get()
		print "Line Fit Threshold: ", LINE_FIT_THRESHOLD
		self.analysisType = "lineFit"
		self.activLineFitButton()
		self.analyze(self.analysisType)

	def changeWaveletSampleSpacing(self,event):
		if self.firstTime:
			return 0
		global MICROS_PER_SAMPLE
		MICROS_PER_SAMPLE = self.waveletSamplingSlider.get()
		print "Micros per sample: ", MICROS_PER_SAMPLE
		self.analysisType= "wavelet"
		self.activWaveletButton()
		self.analyze(self.analysisType)

	def changeWaveletThresh(self,event):
		if self.firstTime:
			return 0
		global WAVELET_THRESHOLD
		WAVELET_THRESHOLD = self.waveletThreshSlider.get()
		print "Wavelet Threshold: ", WAVELET_THRESHOLD
		self.analysisType = "wavelet"
		self.activWaveletButton()
		self.analyze(self.analysisType)


	#------------------------------------------------------------------------------------------
	#-------------------------------------- Coordinate Tools ------------------------------------
	#------------------------------------------------------------------------------------------
	def humanXY(self,*XY):
		if len(XY) == 1:
			y = int(self.canvas.cget("height"))/2 - XY[0][1]
			x = XY[0][0] - int(self.canvas.cget("width"))/2
		else:
			y = int(self.canvas.cget("height"))/2 - XY[1]
			x = XY[0] - int(self.canvas.cget("width"))/2
		return (x,y)

	def canvasXY(self,*XY):
		if len(XY) == 1:
			y = int(self.canvas.cget("height"))/2 - XY[0][1]
			x = XY[0][0] + int(self.canvas.cget("width"))/2
		else:
			y = int(self.canvas.cget("height"))/2 - XY[1]
			x = XY[0] + int(self.canvas.cget("width"))/2
		return (x,y)

	def plotXY(self, *XY):
		if len(XY) == 1:
			y = int(self.plot.cget("height"))/2 - XY[0][1]
			x = XY[0][0] + int(self.plot.cget("width"))/2
		else:
			y = int(self.plot.cget("height"))/2 - XY[1]
			x = XY[0] + int(self.plot.cget("width"))/2
		return (x,y)

	def plotToHumanXY(self,*XY):
		if len(XY) == 1:
			y = int(self.plog.cget("height"))/2 - XY[0][1]
			x = XY[0][0] - int(self.plot.cget("width"))/2
		else:
			y = int(self.plot.cget("height"))/2 - XY[1]
			x = XY[0] - int(self.plot.cget("width"))/2
		return (x,y)


	#------------------------------------------------------------------------------------------
	#------------------------------------ Hyperbola Graphers ----------------------------------
	#------------------------------------------------------------------------------------------

		#	Explicit x = f(y) solution for a hyperbola sqrt((x-a)^2+(y-b)^2)-sqrt((x-c)^2+(y-d)^2)=Tdiff*speed)
		#
		#	spaces are * it seems
		#	There is  +- here: ||
		#	y = (-4 a^2 b+4 a^2 d-sqrt((4 a^2 b-4 a^2 d-8 a b x+8 a d x+4 b^3-4 b^2 d-4 b c^2+8 b c x-4 b d^2-4
b T^2+4 c^2 d-8 c d x+4 d^3-4 d T^2)^2-4 (-4 b^2+8 b d-4 d^2+4 T^2) (-a^4+4 a^3 x-2 a^2 b^2+2 a^2 c^2-4 a^2 c x+2 a^2 d^2+2
a^2 T^2-4 a^2 x^2+4 a b^2 x-4 a c^2 x+8 a c x^2-4 a d^2 x-4 a T^2 x-b^4+2 b^2 c^2-4 b^2 c x+2 b^2 d^2+2 b^2 T^2-c^4+4 c^3 x-
2 c^2 d^2+2 c^2 T^2-4 c^2 x^2+4 c d^2 x-4 c T^2 x-d^4+2 d^2 T^2-T^4+4 T^2 x^2))+8 a b x-8 a d x-4 b^3+4 b^2 d+4 b c^2-8 b c
x+4 b d^2+4 b T^2-4 c^2 d+8 c d x-4 d^3+4 d T^2)/(2 (-4 b^2+8 b d-4 d^2+4 T^2))
		#
		#	y = (-4 a^2 b+4 a^2 d+sqrt((4 a^2 b-4 a^2 d-8 a b x+8 a d x+4 b^3-4 b^2 d-4 b c^2+8 b c x-4 b d^2-4
b T^2+4 c^2 d-8 c d x+4 d^3-4 d T^2)^2-4 (-4 b^2+8 b d-4 d^2+4 T^2) (-a^4+4 a^3 x-2 a^2 b^2+2 a^2 c^2-4 a^2 c x+2 a^2 d^2+2
a^2 T^2-4 a^2 x^2+4 a b^2 x-4 a c^2 x+8 a c x^2-4 a d^2 x-4 a T^2 x-b^4+2 b^2 c^2-4 b^2 c x+2 b^2 d^2+2 b^2 T^2-c^4+4 c^3 x-
2 c^2 d^2+2 c^2 T^2-4 c^2 x^2+4 c d^2 x-4 c T^2 x-d^4+2 d^2 T^2-T^4+4 T^2 x^2))+8 a b x-8 a d x-4 b^3+4 b^2 d+4 b c^2-8 b c
x+4 b d^2+4 b T^2-4 c^2 d+8 c d x-4 d^3+4 d T^2)/(2 (-4 b^2+8 b d-4 d^2+4 T^2))
		#
		#	Python syntax:
		#	y = (-4*a**2*b+4*a**2*d-sqrt(underSqrt)+8*a*b*x-8*a*d*x-4*b**3+4*b**2*d+4*b*c**2-
8*b*c*x+4*b*d**2+4*b*T**2-4*c**2*d+8*c*d*x-4*d**3+4*d*T**2)/(2*(-4*b**2+8*b*d-4*d**2+4*T**2))
		#
		#	y = (-4*a**2*b+4*a**2*d+sqrt(underSqrt)+8*a*b*x-8*a*d*x-4*b**3+4*b**2*d+4*b*c**2-
8*b*c*x+4*b*d**2+4*b*T**2-4*c**2*d+8*c*d*x-4*d**3+4*d*T**2)/(2*(-4*b**2+8*b*d-4*d**2+4*T**2))
	def drawInefficientHyperbola(self,point1,point2,diff):
		a = point1[0]
		b = point1[1]
		c = point2[0]
		d = point2[1]
		T = diff
		counter = 0
		for x in range(-3500,3500,1):
			x = x*0.1 #makes the range 0.1*start,0.1*end,step 0.1
			underSqrt = (4*a**2*b-4*a**2*d-8*a*b*x+8*a*d*x+4*b**3-4*b**2*d-4*b*c**2+8*b*c*x-
4*b*d**2-4*b*T**2+4*c**2*d-8*c*d*x+4*d**3-4*d*T**2)**2-4*(-4*b**2+8*b*d-4*d**2+4*T**2)*(-a**4+4*a**3*x-
2*a**2*b**2+2*a**2*c**2-4*a**2*c*x+2*a**2*d**2+2*a**2*T**2-4*a**2*x**2+4*a*b**2*x-4*a*c**2*x+8*a*c*x**2-
4*a*d**2*x-4*a*T**2*x-b**4+2*b**2*c**2-4*b**2*c*x+2*b**2*d**2+2*b**2*T**2-c**4+4*c**3*x-
2*c**2*d**2+2*c**2*T**2-4*c**2*x**2+4*c*d**2*x-4*c*T**2*x-d**4+2*d**2*T**2-T**4+4*T**2*x**2)
			if underSqrt >= 0:
				y1 = (-4*a**2*b+4*a**2*d-sqrt(underSqrt)+8*a*b*x-8*a*d*x-
4*b**3+4*b**2*d+4*b*c**2-8*b*c*x+4*b*d**2+4*b*T**2-4*c**2*d+8*c*d*x-4*d**3+4*d*T**2)/(2*(-4*b**2+8*b*d-
4*d**2+4*T**2))
				y2 = (-4*a**2*b+4*a**2*d+sqrt(underSqrt)+8*a*b*x-8*a*d*x-
4*b**3+4*b**2*d+4*b*c**2-8*b*c*x+4*b*d**2+4*b*T**2-4*c**2*d+8*c*d*x-4*d**3+4*d*T**2)/(2*(-4*b**2+8*b*d-
4*d**2+4*T**2))
				if counter == 0:
					prev1 = (x,y2)
					prev2 = (x,y2)
					counter += 1
					continue
				self.line(None,prev1[0],prev1[1],x,y1)
				self.line(None,prev2[0],prev2[1],x,y2)
				prev1 = (x,y1)
				prev2 = (x,y2)

	def drawHyperbola(self,point1,point2,diff, tag=None):
			cntr = midPt(point1,point2)
			newSys = newHyberolaClass(cntr, point1)
```

```
                    if VERBOSE_VERBOSE
:################### VERBOSE OPTION -- draw X,Y coords hyperb's use (rotated, shifted) ####################
                              self.drawFullLine((cntr + newSys.xRef),color='red', dashing=1, width=3)
                              self.drawFullLine((cntr + newSys.yRef),color='red', dashing=1, width=3)
                    pointGen = newSys.yieldHorizontalHyperbolaPoints(dist(cntr,point1),diff)
                    prev = pointGen.next()
                    while prev == "sentinel":
                              prev = pointGen.next()
                    for x in pointGen:
                              if x == "sentinel":
                                        prev = pointGen.next()
                                        x = pointGen.next()
                                        continue
                              self.line(tag,prev,x)
                              prev = x


          #-----------------------------------------------------------------------------------------
          #------------------------------- Object Removal and timed deletion ------------------------
          #-----------------------------------------------------------------------------------------
          def delTaggedAs(self,tag):
                    self.canvas.delete(tag)
                    self.plot.delete(tag)
                    self.waveletsPlot.delete(tag)
          def deleteCurs(self):
                    self.canvas.delete("cur")
                    self.plot.delete("cur")
                    self.waveletsPlot.delete("cur")
          def deleteAnalysis(self):
                    self.canvas.delete("analysis")
                    self.plot.delete("analysis")
                    self.waveletsPlot.delete("analysis")
          def deleteAfter(self,t,tags=None): #takes milliseconds
                    self.parent.after(t,self.delTaggedAs(tags))

          #-----------------------------------------------------------------------------------------
          #------------------------------------ Iterative Solvers -----------------------------------
          #-----------------------------------------------------------------------------------------
          def iterSolution2D(self,start,end,func, accuracy=0.1):
                    step = 10
                    endPt = end[:]
                    angle = math.atan2(end[1] - start[1], end[0]-start[0])
                    getPtAtDist = lambda d: (start[0] + d*cos(angle),start[1] + d*sin(angle))
                    getStepBreakdown = lambda st: (st*cos(angle),st*sin(angle))
                    addStep = lambda p: map(add,p,getStepBreakdown(step))
                    sols = []
                    startVal = func(start)
                    end = addStep(start)
                    endVal = func(end)
                    for x in range(0,int(dist(start,endPt)/step)):
                              startCp = tuple(start[:])
                              endCp = tuple(end[:])
                              print "step: %f\nstart: %s\nend: %s\n val at start: %f\nval at end: %f\n\n" %(step, str(start),
str(end), startVal,endVal)
                              if startVal == 0:
                                        if start not in sols: sols.append(start)
                              elif endVal == 0:
                                        if end not in sols: sols.append(end)
                              elif startVal*endVal < 0:
```

```
                    step = 0.5*step
                    midPt = getPtAtDist(step)
                    midVal = func(midPt)
                    while abs(midVal) > accuracy:
                        if startVal*midVal < 0:
                            end = midPt[:]
                            endVal = midVal
                        if endVal*midVal < 0:
                            start = midPt[:]
                            startVal = midVal
                        step = 0.5*step
                        midPt = getPtAtDist(step)
                        midVal = func(midPt)
                    print "FOUND:", midPt, midVal
                    sols.append(midPt)
                    step = 5
                start = addStep(startCp)
                startVal = func(start)
                end = addStep(endCp)
                endVal = func(end)
                if len(sols) == 3:
                    print "BREKAING"
                    print sols
                    break
        return sols


    def iterSolution1D(self,start,end,func, accuracy=0.1): #assumes func never throws a sqrt(negative number) error &&
func is a in form lambda pt: f(p[0],p[1]) - sol (which = 0)
        startCloseness = func(start)
        endCloseness = func(end)
        print startCloseness, endCloseness
        if startCloseness*endCloseness > 0: #they are both either + or both -
            return None #no convergence in this range
        pt = 0.5*(end-start) + start
        midCloseness = func(pt)
        while abs(midCloseness) > accuracy: #while not within one of the answer
            print startCloseness, endCloseness
            print pt, midCloseness
            if startCloseness*midCloseness < 0: #if these two points contain 0 in their span
                end = pt
                endCloseness = midCloseness
                pt = 0.5*(end-start) + start
                midCloseness = func(pt)
            elif endCloseness*midCloseness < 0:     #or if these two points contain 0 in their span
                start = pt
                startCloseness = midCloseness
                pt = 0.5*(end-start) + start
                midCloseness = func(pt)
        return pt


    #this function uses polar to check the circle at even intervals (var step)
    #the lambda 'pt' converts the polar coord into rectangular coords the func accepts, taking into account the shift that
needs to be applied (don't want to mess with shifted polar circles)
    def findOnRadius(self,point,radius,func, accuracy=0.1):
        theta = 0
        endTheta = pi/8
        shift = point
```

```
		print point,radius
		pointsFound = 0
		step = pi/8
		points = []
		pt = lambda t: map(add,polToRect(radius,t),shift)
		for x in range(1,32):
			val = func(pt(theta))
			valEnd = func(pt(endTheta))
			print "FOR LOOP PRINT\n\nstep: %f\ntheta: %f\nendTheta: %f\n val at theta: %f\nval at
endTheta: %f\n\n" %(step, theta, endTheta,val,valEnd)
			if (val*valEnd) < 0: #if there's convergence on this range (if * is neg)
				print "step: %f\ntheta: %f\nendTheta: %f\n val at theta: %f\nval at endTheta: %f\n\n"
%(step, theta, endTheta,val,valEnd)
				while abs(val) > accuracy: #while haven't found answer with accuracy <= 0.1
					midPt = ((pt(theta)[0]+pt(endTheta)[0])/2,(pt(theta)[1]+pt(endTheta)[1])/2)
					print "step: %f\ntheta: %f\nendTheta: %f\n val at theta: %f\nval at endTheta:
%f\n\n" %(step, theta, endTheta,val,valEnd)
					if val == 0:
						if pt(theta) not in points: points.append(pt(theta))
					elif valEnd == 0:
						if pt(endTheta) not in points: points.append(pt(endTheta))
					elif val*func(midPt) < 0: #if 0 is between starting theta and mid
						endTheta = endTheta - 0.5*step
						valEnd = func(pt(endTheta))
					elif valEnd*func(midPt) < 0:
						theta = theta + 0.5*step
						val = func(pt(theta))
					step = 0.5*step
					raw_input()
				pointsFound += 1
				print "FOUND: ", pt(theta)
				points.append(pt(theta))
			if pointsFound == 2:
				print "breaking"
				break
			step = pi/8
			theta = step*x
			endTheta = step*(1+x)
		return points

	#-------------------------------------------------------------------------------------------
	#--------------------------------- Basic Drawing Functions ---------------------------------
	#-------------------------------------------------------------------------------------------
	def line(self,tag,*points): #takes two points at a time
		if len(points) == 2:
			coords = (points[0][0],points[0][1], points[1][0], points[1][1])
		else:
			coords = (points[0] ,points[1], points[2], points[3])
		self.canvas.create_line(self.canvasXY(coords[0],coords[1])+self.canvasXY(coords[2],coords[3]),tags = tag)

	def drawFullLine(self,coords, color='black', dashing=None, width=1.0): #takes two points as ((x,y),(x,y)) or (x,y,x,y)
		canvasYmax = int(self.canvas.cget('height'))/2
		canvasYmin = -canvasYmax
		dx = coords[0] - coords[2]
		dy = coords[1] - coords[3]
		if dx == 0 and dy == 0: #invalid
			return 0
		if dx == 0: #if vertical line
		self.canvas.create_line(self.canvasXY(coords[0],canvasYmax)+self.canvasXY(coords[0],canvasYmin),fill=color,
dash=dashing, width=width)
				return 0
			if dy == 0: #if horizontal
				self.canvas.create_line(self.canvasXY(-
int(self.canvas.cget('width'))/2,coords[1])+self.canvasXY(int(self.canvas.cget('width'))/2,coords[1]),fill=color, dash=dashing,
width=width)
				return 0

			#comes from y-b=m(x-a) using (x,y) and (a,b)
			xAtYMax = (canvasYmax-coords[1])/(dy/dx) + coords[0]
			xAtYMin = (canvasYmin-coords[1])/(dy/dx) + coords[0]
			self.canvas.create_line(self.canvasXY(xAtYMax,canvasYmax)+self.canvasXY(xAtYMin,canvasYmin),fill=color,
dash=dashing, width=width)
			return 0

	def canvasDisk(self,center, radius, color='red', tag=None):
		diagonal = cos(45)*radius
		topLeft = self.canvasXY(center[0]-diagonal,center[1]-diagonal)
		bottomRight = self.canvasXY(center[0]+diagonal, center[1] + diagonal)
		self.canvas.create_oval(topLeft[0],topLeft[1],bottomRight[0],bottomRight[1],width=0,fill=color, tags=tag)

	def plotDisk(self,center,radius,color='purple', tag=None):
		diagonal = cos(45)*radius
		topLeft = self.plotXY(center[0]-diagonal, center[1]-diagonal)
		bottomRight = self.plotXY(center[0]+diagonal, center[1]+diagonal)
		self.plot.create_oval(topLeft[0], topLeft[1],  bottomRight[0], bottomRight[1], width=0, fill=color, tags=tag)

	def waveletPlotDisk(self,center,radius,color='purple', tag=None):
		diagonal = cos(45)*radius
		topLeft = self.plotXY(center[0]-diagonal, center[1]-diagonal)
		bottomRight = self.plotXY(center[0]+diagonal, center[1]+diagonal)
		self.waveletsPlot.create_oval(topLeft[0], topLeft[1],  bottomRight[0], bottomRight[1], width=0, fill=color,
tags=tag)

	#---------------------------------------------------------------------------------------------
	#---------------- Misc Functions - most likely deprecated in current version ----------------
	#---------------------------------------------------------------------------------------------
	def getLineCoords(self):
		self.lineX1 = tkSimpleDialog.askfloat("Input","Enter first X Cood: ", initialvalue=0)
		self.lineY1 = tkSimpleDialog.askfloat("Input","Enter first Y Cood: ", initialvalue=0)
		self.lineX2 = tkSimpleDialog.askfloat("Input","Enter second X Cood: ", initialvalue=0)
		self.lineY2 = tkSimpleDialog.askfloat("Input","Enter second Y Cood: ", initialvalue=0)


	#---------------------------------------------------------------------------------------------
	#-------------------------------- Serial and Data Processing -------------------------------
	#---------------------------------------------------------------------------------------------

	#Interface, Signal, etc
	def checkSerial(self):
		if ser.inWaiting() > 1:
			self.firstTime = False
			self.tmpData = []
			a = ser.readline()
			while "DONE" not in a:
				self.tmpData.append(a)
				a = ser.readline()
				self.processData()
			self.parent.after(100,self.checkSerial)


	def processData(self):
			for x in range(0,len(self.tmpData)):
				self.tmpData[x] = self.tmpData[x].split(" ")
				for i in range(0,len(self.tmpData[x])):
					self.tmpData[x][i] = int(self.tmpData[x][i].rstrip().strip())
			self.tmpData = self.cleanData(self.tmpData)

			#now we need to recheck data to make sure values that were 'signals' before were not bad data that is now
removed
			if not self.checkEachForSignal(self.tmpData):
				print "discarding"
				return 0

			self.origData = [x[:] for x in self.tmpData]
			if VERBOSE:				#################### VERBOSE OPTION -- print cleaned data
received from controller ####################
				print "---IN PROCESS DATA: cleaned accepted data:---"
				print self.origData
				print "---END PROCESS DATA data PRINT---"
			if useSplineFit:
				data1 = [(x[0],x[3]) for x in self.origData]
				data2 = [(x[1],x[3]) for x in self.origData]
				data3 = [(x[2],x[3]) for x in self.origData]
				splined1 = self.splineFit(data1, MICROS_PER_SAMPLE)
				splined2 = self.splineFit(data2, MICROS_PER_SAMPLE)
				splined3 = self.splineFit(data3, MICROS_PER_SAMPLE)
				self.splinedData = [(splined1[i][0], splined2[i][0], splined3[i][0], splined1[i][1]) for i in
range(len(splined1))] #(data1,data2,data3,time)
				if VERBOSE_VERBOSE:			#################### VERBOSE_VERBOSE OPTION --
print splined data received ####################
					print "---IN PROCESS DATA: splined data:---"
					print self.splinedData
					print "---END PROCESS DATA splined data PRINT---"
				self.data = [x[:] for x in self.splinedData]
			else:
				self.data = [x[:] for x in self.origData]
			self.analyze(self.analysisType)


	def cleanData(self, info):
			datas = [x[:] for x in info]
			data1, data2, data3 = [i[0] for i in datas],[i[1] for i in datas],[i[2] for i in datas]
			pops = []
			for i in range(1,len(datas)-1):
				if list(set(datas[i]) & set(exclude)) != []:
					pops.append(i)
				#if self.data[i][0] in exclude: #and (self.data[i-1][0]/100) == 28 and (self.data[i+1][0]/100) == 28:
				#	pops.append(i)
				#elif self.data[i][1] in exclude: #and (self.data[i-1][1]/100) == 28 and (self.data[i+1][1]/100) == 28:
				#	pops.append(i)
				#elif self.data[i][2] in exclude: #and (self.data[i-1][2]/100) == 28 and (self.data[i+1][2]/100) == 28:
				#	pops.append(i)
			pops = list(set(pops)) #cut out duplicates
			if VERBOSE:
				print "POPPING",
					print pops
					print "ELEMENTS TO BE POPPED:"
					for x in pops: print datas[x]
					print "END POPS"
				counter = 0
				for i in pops:
					del datas[i-counter]
					counter += 1

				for x in range(0,12): #RED SLOW DATA SHIFT DOWN HAPPENS HERE
					datas[x][0] -= RED_START_OFFSET

				av1, av2, av3 = sum(data1[0:AV_DIST])/AV_DIST, sum(data2[0:AV_DIST])/AV_DIST,
sum(data3[0:AV_DIST])/AV_DIST

				for x in range(0,len(datas)):
					datas[x][0] = (datas[x][0]-av1)*SCALE_FACTOR_1 + av1
					datas[x][1] = (datas[x][1]-av2)*SCALE_FACTOR_2 + av2
					datas[x][2] = (datas[x][2]-av3)*SCALE_FACTOR_3 + av3

				return datas


			def analyze(self,typ): #operates on self.data
				global plotRectified
				self.deleteCurs()
				if typ == "stdThresh":
					times = self.stdThreshold(self.data)
					self.drawPlots(self.data)
				elif typ == "lineFit":
					results = self.lineFit(self.data)
					if plotRectified:
						self.drawPlots(self.rectifiedData, shift=2000) #shift parameter is how much to shift the
plot DOWN
					else:
						self.drawPlots(self.data)
					if results == None:
						return 0
					times = results[0]
					coeffs = results[1]

				elif typ == "wavelet":
					if not useSplineFit: #need splineFit for this analysis
						self.toggleSplineUse()
						data1 = [(x[0],x[3]) for x in self.origData]
						data2 = [(x[1],x[3]) for x in self.origData]
						data3 = [(x[2],x[3]) for x in self.origData]
						splined1 = self.splineFit(data1, MICROS_PER_SAMPLE)
						splined2 = self.splineFit(data2, MICROS_PER_SAMPLE)
						splined3 = self.splineFit(data3, MICROS_PER_SAMPLE)
						self.splinedData = [(splined1[i][0], splined2[i][0], splined3[i][0], splined1[i][1]) for i in
range(len(splined1))] #(data1,data2,data3,time)
						self.data = [x[:] for x in self.splinedData]
					times = self.wavelet(self.data)
					self.drawPlots(self.data)
					self.drawWavelets(self.waveletData)

				times = (times[0] - RED_TWEAK, times[1] - BLUE_TWEAK, times[2])
				diffs = [times[0]-times[1],times[1]-times[2],times[0]-times[2]]
```

```
		if max([abs(x) for x in diffs]) > self.maxTime+100:
			self.deleteAnalysis()
			print "-----ANALYSIS TYPE: %s-----" %typ
			self.canvas.create_text(self.canvasXY(0,0),text="???", font=("Helvetica","72"), tags=("cur"))
			print "###DATA DUMP###"
			#for i in self.data: #dump data
			#		print i
			print "Times", times
			print "###END DATA DUMP###"
			return 0
		self.deleteAnalysis()
		self.drawEstimateIndicators(times)
		try:
			self.drawHyperbolas(diffs)
		except UnboundLocalError:
				self.canvas.create_text(self.canvasXY(0,0), text="Could Not Plot -- Error 700_unbound
sol var", font=("Helvetica","20"), tags=("cur"))
				return 0


	#draws the graphic in the plot canvas
	#uses the self.data variable
	def drawPlots(self,data, shift=0):
		datas = []
		lastTime = data[-1][-1]
		d = lastTime
		self.scale = float(self.plot.cget("width"))/d
		shift = -shift/(4200/int(self.plot.cget("height")))
		self.scaledTimeData= {}
		for x in range(0,len(self.data)):
			datas.append([])
			datas[x].append(data[x][0]/(4200/int(self.plot.cget("height"))))
			datas[x].append(data[x][1]/(4200/int(self.plot.cget("height"))))
			datas[x].append(data[x][2]/(4200/int(self.plot.cget("height"))))
			datas[x].append((data[x][3])*self.scale)
			self.scaledTimeData[int(datas[x][-1])] = (data[x][0],data[x][1], data[x][2], data[x][3])

		pt1 = datas[0][0]
		pt2 = datas[0][1]
		pt3 = datas[0][2]
		t = datas[0][3]
		for x in datas:
			self.plot.create_line(t, int(self.plot.cget("height"))- pt1-shift, x[3], int(self.plot.cget("height"))-x[0]-
shift, fill=colors[1], tags=("cur","plots"))
			self.plot.create_line(t, int(self.plot.cget("height"))- pt2-shift, x[3], int(self.plot.cget("height"))-x[1]-
shift, fill=colors[2], tags=("cur","plots"))
			self.plot.create_line(t, int(self.plot.cget("height"))- pt3-shift, x[3], int(self.plot.cget("height"))-x[2]-
shift, fill=colors[3], tags=("cur","plots"))
			pt1 = x[0]
			pt2 = x[1]
			pt3 = x[2]
			t = x[3]

	def drawWavelets(self,data, shift=0):
		lastTime = data[-1][-1]
		shift = -shift/(4200/int(self.waveletsPlot.cget("height")))
		scaledWavelets = []
		self.scaledWaveletData = {}
		for x in range(0,len(data)):
			scaledWavelets.append([])
			#print scaledWavelets[x]
			#print data[x][0]
			scaledWavelets[x].append(data[x][0]*3.0)
			scaledWavelets[x].append(data[x][1]*3.0)
			scaledWavelets[x].append(data[x][2]*3.0)
			scaledWavelets[x].append((data[x][3])*self.scale)
			self.scaledWaveletData[int(scaledWavelets[x][-1])] = (data[x][0],data[x][1],data[x][2],data[x][3])
		pt1 = scaledWavelets[0][0]
		pt2 = scaledWavelets[0][1]
		pt3 = scaledWavelets[0][2]
		t = scaledWavelets[0][3]
		for x in scaledWavelets:
			self.waveletsPlot.create_line(t, int(self.waveletsPlot.cget("height"))- pt1-shift, x[3], 
int(self.waveletsPlot.cget("height"))-x[0]-shift, fill=colors[1], tags=("cur","plots"))
			self.waveletsPlot.create_line(t, int(self.waveletsPlot.cget("height"))- pt2-shift, x[3], 
int(self.waveletsPlot.cget("height"))-x[1]-shift, fill=colors[2], tags=("cur","plots"))
			self.waveletsPlot.create_line(t, int(self.waveletsPlot.cget("height"))- pt3-shift, x[3], 
int(self.waveletsPlot.cget("height"))-x[2]-shift, fill=colors[3], tags=("cur","plots"))
			pt1 = x[0]
			pt2 = x[1]
			pt3 = x[2]
			t = x[3]

	def drawEstimateIndicators(self, times):
		self.plot.create_line(self.scale*(times[0]),0,self.scale*(times[0]),int(self.plot.cget("height")),fill=colors[1], 
tags=("cur","analysis", "plots"))
		self.plot.create_line(self.scale*(times[1]),0,self.scale*(times[1]),int(self.plot.cget("height")),fill=colors[2], 
tags=("cur","analysis", "plots"))
		self.plot.create_line(self.scale*(times[2]),0,self.scale*(times[2]),int(self.plot.cget("height")),fill=colors[3], 
tags=("cur","analysis","plots"))

	def drawHyperbolas(self, diffs):
		self.drawHyperbola(sensors[1],sensors[2],diffs[0]*self.speed+0.1,("cur","analysis"))
		self.drawHyperbola(sensors[2],sensors[3],diffs[1]*self.speed+0.1,("cur","analysis"))
		self.drawHyperbola(sensors[1],sensors[3],diffs[2]*self.speed+0.1,("cur","analysis"))


	#-------------------------------------------------------------------------------------------
	#------------------------------------ ANALYSIS METHODS ------------------------------------
	#-------------------------------------------------------------------------------------------

	def stdThreshold(self,data):
		global BLUE_TWEAK
		global RED_TWEAK
		RED_TWEAK = 85
		BLUE_TWEAK = -20
		data = data[:]
		data1, data2, data3, times = [],[],[],[]
		for x in data:
			data1.append(int(x[0]))
			data2.append(int(x[1]))
			data3.append(int(x[2]))
			times.append(int(x[3]))

		av1, av2, av3 = sum(data1[0:AV_DIST])/AV_DIST, sum(data2[0:AV_DIST])/AV_DIST, 
sum(data3[0:AV_DIST])/AV_DIST
		found1, found2, found3 = False, False, False
```

```
            for i in range(AV_DIST,len(data1)):
                if abs(data1[i] - av1) > THRESHOLD and not found1:
                    impactTime1 = times[i]
                    found1 = True
                if abs(data2[i] - av2) > THRESHOLD and not found2:
                    impactTime2 = times[i]
                    found2 = True
                if abs(data3[i] - av3) > THRESHOLD and not found3:
                    impactTime3 = times[i]
                    found3 = True
                if found1 and found2 and found3: break
            if not found1 or not found2 or not found3: #if one of them didn't have a value found
                return (9000,9000,9000) #9000 will be caught and displayed as ??? in the analyze function
            print (impactTime1, impactTime2, impactTime3)
            return (impactTime1, impactTime2, impactTime3)


        def checkEachForSignal(self,data):
            data = data[:]
            data1, data2, data3 = [i[0] for i in data],[i[1] for i in data],[i[2] for i in data]

            av1, av2, av3 = sum(data1[0:AV_DIST])/AV_DIST, sum(data2[0:AV_DIST])/AV_DIST,
    sum(data3[0:AV_DIST])/AV_DIST
            found1, found2, found3 = False, False, False
            for i in range(AV_DIST,len(data1)):
                if abs(data1[i] - av1) > THRESHOLD:
                    found1 = True
                if abs(data2[i] - av2) > THRESHOLD:
                    found2 = True
                if abs(data3[i] - av3) > THRESHOLD:
                    found3 = True
                if found1 and found2 and found3: break
            if not(found1 and found2 and found3): return False
            else: return True


        def lineFit(self,datas):
            maxPeaks = 7
            data = datas[:]
            data1, data2, data3, times = [],[],[],[]
            for x in data:
                data1.append(int(x[0]))
                data2.append(int(x[1]))
                data3.append(int(x[2]))
                times.append(int(x[3]))

            av1, av2, av3 = sum(data1[0:AV_DIST])/AV_DIST, sum(data2[0:AV_DIST])/AV_DIST,
    sum(data3[0:AV_DIST])/AV_DIST
            for i in range(AV_DIST,len(data)):
                data1[i] = av1 + abs(data1[i]-av1) #rectify with average as baseline
                data2[i] = av2 + abs(data2[i]-av2)
                data3[i] = av3 + abs(data3[i]-av3)

            self.rectifiedData = [zip(data1, data2, data3, times)][0]
            diffs1, diffs2, diffs3 = [],[],[]
            for i in range(AV_DIST,len(data)):
                if abs(data1[i]-av1) > LINE_FIT_THRESHOLD: #if there's a significant difference
                    diffs1.append(data1[i]-data1[i-1])
                else:
                    diffs1.append(0)
                if abs(data2[i]-av2) > LINE_FIT_THRESHOLD:
                    diffs2.append(data2[i]-data2[i-1])
                else:
                    diffs2.append(0)
                if abs(data3[i]-av3) > LINE_FIT_THRESHOLD:
                    diffs3.append(data3[i]-data3[i-1])
                else:
                    diffs3.append(0)

            maxs1, maxs2, maxs3 = [],[],[]
            for i in range(1,len(diffs1)):
                if diffs1[i] < 0 and diffs1[i-1] >= 0:
                    maxs1.append(i-1)
                if diffs2[i] < 0 and diffs2[i-1] >= 0:
                    maxs2.append(i-1)
                if diffs3[i] < 0 and diffs3[i-1] >= 0:
                    maxs3.append(i-1)
            if VERBOSE_VERBOSE:
        #################### VERBOSE_VERBOSE OPTION -- print data, differences, calculated maxima
    ####################
                for x in range(0,len(diffs1)):
                    print data1[x+AV_DIST], diffs1[x],
                    if x in maxs1:
                        print 'MAX'
                    else: print ""
                print ""
                for x in range(0,len(diffs2)):
                    print data2[x+AV_DIST], diffs2[x],
                    if x in maxs2:
                        print "MAX"
                    else: print ""
                print ""
                for x in range(0,len(diffs3)):
                    print data3[x+AV_DIST], diffs3[x],
                    if x in maxs3:
                        print "MAX"
                    else: print ""

            #subtracting AV_DIST in all these because diffs and maxs start at index AV_DIST of the original data list
    (max's values [indices] are AV_DIST above the start of the data)
            maxPts1, maxPts2, maxPts3 = [(times[maxs1[0]+AV_DIST],data1[maxs1[0]+AV_DIST])] ,
    [(times[maxs2[0]+AV_DIST],data2[maxs2[0]+AV_DIST])] , [(times[maxs3[0]+AV_DIST],data3[maxs3[0]+AV_DIST])]
            if len(maxPts1) > maxPeaks:
                maxPts1 = maxPts1[:maxPeaks]
            if len(maxPts2) > maxPeaks:
                maxPts2 = maxPts2[:maxPeaks]
            if len(maxPts3) > maxPeaks:
                maxPts3 = maxPts3[:maxPeaks]
            counter = 1
            while data1[maxs1[counter]+AV_DIST] - data1[maxs1[counter-1]+AV_DIST] > -50:
                maxPts1.append((times[maxs1[counter]+AV_DIST],data1[maxs1[counter]+AV_DIST]))
                counter += 1
            counter = 1
            while data2[maxs2[counter]+AV_DIST] - data2[maxs2[counter-1]+AV_DIST] > -100:
                maxPts2.append((times[maxs2[counter]+AV_DIST],data2[maxs2[counter]+AV_DIST]))
                counter += 1
            counter = 1
```

```
		while data3[maxs3[counter]+AV_DIST] - data3[maxs3[counter-1]+AV_DIST] > -100:
			maxPts3.append((times[maxs3[counter]+AV_DIST],data3[maxs3[counter]+AV_DIST]))
			counter += 1
		print "MaxTimes1: ", [x[0] for x in maxPts1]
		print "MaxTimes2: ", [x[0] for x in maxPts2]
		print "MaxTimes3: ", [x[0] for x in maxPts3]

		lineFit1 = leastSquaresFit([(x[1],x[0]) for x in maxPts1]) #returns (b,m) which are y = mx+b
		lineFit2 = leastSquaresFit([(x[1],x[0]) for x in maxPts2])
		lineFit3 = leastSquaresFit([(x[1],x[0]) for x in maxPts3])
		if lineFit1 == None or lineFit2 == None or lineFit3 == None:
				self.canvas.create_text(self.canvasXY(0,0),text="Can't find enough points", 
font=("Helvetica","36"), tags=("cur"))
				return None
		#calculate impact times by intersecting line generated by least squares fit and y = av (baseline)
		impactTime1 = (av1 - lineFit1[0])/lineFit1[1]
		impactTime2 = (av2 - lineFit2[0])/lineFit2[1]
		impactTime3 = (av3 - lineFit3[0])/lineFit3[1]

		print (impactTime1, impactTime2, impactTime3)
		return ((impactTime1, impactTime2, impactTime3), (lineFit1, lineFit2, lineFit3)) #return (b,m) in case that 
line will be plotted too


	def wavelet(self,data):
		global BLUE_TWEAK
		global RED_TWEAK
		BLUE_TWEAK = 0
		level = 1
		data1, data2, data3 = [],[],[]
		for x in data:
			data1.append(x[0])
			data2.append(x[1])
			data3.append(x[2])
		wvlt1 = [list(i) for i in list(pywt.wavedec(data1, "db1", level=level))]
		wvlt2 = [list(i) for i in list(pywt.wavedec(data2, "db1", level=level))]
		wvlt3 = [list(i) for i in list(pywt.wavedec(data3, "db1", level=level))]

		del wvlt1[0] #first set of coeffs not used
		del wvlt2[0]
		del wvlt3[0]

		coeffs1 = wvlt1[LEVEL_TO_USE][:]
		coeffs2 = wvlt2[LEVEL_TO_USE][:]
		coeffs3 = wvlt3[LEVEL_TO_USE][:]

		coeffs1 = rectifyData(coeffs1, 0) #rectify the coefffs
		coeffs2 = rectifyData(coeffs2, 0)
		coeffs3 = rectifyData(coeffs3, 0)

		for x in range(0,len(coeffs1)):
			if coeffs1[x] < WAVELET_THRESHOLD: #threshold the coeffs
				coeffs1[x] = 0
			if coeffs2[x] < WAVELET_THRESHOLD:
				coeffs2[x] = 0
			if coeffs3[x] < WAVELET_THRESHOLD:
				coeffs3[x] = 0

		self.waveletData = []
		for x in range(0,len(coeffs1)):
			self.waveletData.append((coeffs1[x], coeffs2[x], coeffs3[x], data[x][3]*2))

		firstValLoc1 = getFirstNonZeroIndex(coeffs1)
		firstValLoc2 = getFirstNonZeroIndex(coeffs2)
		firstValLoc3 = getFirstNonZeroIndex(coeffs3)
		t1 = firstValLoc1*MICROS_PER_SAMPLE*2
		t2 = firstValLoc2*MICROS_PER_SAMPLE*2
		t3 = firstValLoc3*MICROS_PER_SAMPLE*2
		print (t1, t2, t3)
		return (t1,t2,t3)


	#-------------------------------------------------------------------------------------------
	#-------------------------------------- Other Math Stuff ------------------------------------
	#-------------------------------------------------------------------------------------------
	def splineFit(self,d, sampleSpacing):			#func can only take one set of [(y,t)...] data at a time
		ys = [i[0] for i in d] #copy data out
		ts = [i[1] for i in d]
		func = interp1d(ts,ys, kind=INTERPOLATE_KIND)
		end = int(ts[-1]/(2*sampleSpacing))*2*sampleSpacing #gives an integer close to 4.1k and divisible by the 
spacing
		data = [(func(t), t) for t in range(0,end,sampleSpacing)]
		return data


#-------------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------
#--------------------------------- Hyperbola Generation Class -------------------------------
#-------------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------

#make new coord sys to graph simple (moved/rotated origin) hyperbolas centered at the origin
#for vert hyperbola centered at (0,0) and foci at b, and difference T:
#x = +- sqrt((16y**2*b**2 - 8*y*b*T**2 + T**4)/(4*T**2) - y**2 + 2*y*b - b**2)
#only takes vertical distance to focus (half distance to other focus) and time difference

class newHyberolaClass():
	def __init__(self,origin,ptX):
		#ptX = (ptX[0]/dist(origin,ptX),ptX[1]/dist(origin,ptX)) #make unit length
		self.originShift = origin
		self.angle = math.atan2(float(ptX[1])-origin[1],float(ptX[0])-origin[0])
		self.xRef = ptX
		self.yRef = tuple(map(add,rotatePt_origin((float(ptX[0])-origin[0],float(ptX[1])-origin[1]),pi/2),origin))

	def yieldVerticalHyperbolaPoints(self,d,diff):  #d must be positive
		T = diff
		b = d
		start = int(T/2)
		xPos = lambda y: sqrt((16*y**2*b**2 - 8*y*b*T**2 + T**4)/(4*T**2) - y**2 + 2*y*b - b**2)
		xNeg = lambda y: -sqrt((16*y**2*b**2 - 8*y*b*T**2 + T**4)/(4*T**2) - y**2 + 2*y*b - b**2)
		for y in range(350,start,-1):
			try:
				sol = xNeg(y) #start in the upper left solutions
			except ValueError: pass #print "failed", y,(16*y**2*b**2 - 8*y*b*T**2 + T**4)/(4*T**2) - y**2 + 
2*y*b - b**2
			if sol < -500:
				continue
			yield self.transToNormalCoords((sol,y))
		for y in range(start,350):
			try:
				sol = xPos(y) #upper right
			except ValueError: pass #print "failed", y,(16*y**2*b**2 - 8*y*b*T**2 + T**4)/(4*T**2) - y**2 +
2*y*b - b**2
			if sol > 500:
				break
			yield self.transToNormalCoords((sol,y))
		yield "sentinel"
		for y in range(-350,-1*start,):
			try:
				sol = xNeg(y) #lower left
			except ValueError: pass #print "failed", y,(16*y**2*b**2 - 8*y*b*T**2 + T**4)/(4*T**2) - y**2 +
2*y*b - b**2
			if sol < -500:
				continue
			yield self.transToNormalCoords((sol,y))
		for y in range(-1*start,-350,-1):
			try:
				sol = xPos(y) #lower left
			except ValueError: pass #print "failed", y,(16*y**2*b**2 - 8*y*b*T**2 + T**4)/(4*T**2) - y**2 +
2*y*b - b**2
			if sol > 500:
				break
			yield self.transToNormalCoords((sol,y))

	def yieldHorizontalHyperbolaPoints(self,d,diff):
		a = diff/2
		c = abs(d)
		xPos = lambda y: sqrt(a**2 * (1+(y**2/(c**2 - a**2))))
		xNeg = lambda x: -1*xPos(x)
		if a >= 0:
			for y in range(-650,650):
				try:
					sol = xNeg(y)
				except ValueError: pass #print "failed", y; continue
				if sol > 700 or sol < -700:
					continue
				yield self.transToNormalCoords((sol,y))
		if a <= 0:
			for y in range(-650,650):
				try:
					sol = xPos(y)
				except ValueError: pass #print "failed", y; continue
				if sol > 700 or sol < -700:
					continue
				yield self.transToNormalCoords((sol,y))


	def transToNewCoords(self,point):
		x = point[0]
		y = point[1]
		x = x - self.originShift[0] #shift to new origin
		y = y - self.originShift[1]
		newPt = rotatePt_origin((x,y), -self.angle)
		return newPt
	def transToNormalCoords(self,point):
		newPt = rotatePt_origin(point,self.angle)
		x = newPt[0] + self.originShift[0]
```

```
                y = newPt[1] + self.originShift[1]
                return (x,y)


#------------------------------------------------------------------------------------------
#------------------------------------------------------------------------------------------
#------------------------- Math, Support, and Misc global Functions -----------------------
#------------------------------------------------------------------------------------------
#------------------------------------------------------------------------------------------

#default places is 100 places... need this accuracy for iterative solvers
def truncateFloat(num,places=100):
        return int(10**(places)*num)/(10.0**(places))

def rectToPolar(*rectPt):
        if len(rectPt) == 1:
                rectPt = [x for sub in rectPt for x in sub]
        r = sqrt(rectPt[0]**2 + rectPt[1]**2)
        theta = math.atan2(rect[1],rect[0])
        return (truncateFloat(r),truncateFloat(theta))

def polToRect(*polarPt):
        if len(polarPt) == 1:
                rectPt = [x for sub in polarPt for x in sub]
        x = polarPt[0]*cos(polarPt[1])
        y = polarPt[0]*sin(polarPt[1])
        return (truncateFloat(x),truncateFloat(y))

def dist(*points):
        if len(points) == 2:
                points = [x for sub in points for x in sub]
        return sqrt((points[0]-points[2])**2 + (points[1]-points[3])**2)

def rotatePt_origin(pt,angle): #rotate point around origin, angle in radians
        return (truncateFloat(pt[0]*cos(angle) - pt[1]*sin(angle),5),truncateFloat(pt[0]*sin(angle) + pt[1]*cos(angle),5))

def midPt(*points):
        if len(points) == 2:
                points = [x for sub in points for x in sub]
        return((points[0]+points[2])/2,(points[1]+points[3])/2)

#takes a dictionary with keys being numbers and finds closet key to
#returns the closest key
def closestDictMatch(dictionary, val):
        closenesses = [(abs(key - val),key) for key in dictionary]
        return min(closenesses)[1]

def getFirstNonZeroIndex(l):
        for i in range(0,len(l)):
                if l[i] != 0:
                        return i

def leastSquaresFit(points):
        if len(points) == 1:
                return None
        ys = [float(p[0]) for p in points]
        xs = [float(p[1]) for p in points]
        y2s = [float(y**2) for y in ys]
        x2s = [float(x**2) for x in xs]
```

```
    xys = [float(x*y) for x,y in zip(xs,ys)]

    if VERBOSE_VERBOSE:                                                                 ####################
VERBOSE_VERBOSE OPTION -- print data, differences, calculated maxima ####################
        print "----LEAST SQUARES FIT----"
        print "points:", points
        print "X's: ", xs
        print "Y's: ", ys
        print "X^2: ", x2s
        print "Y^2: ", y2s
        print "-------END LSQ FIT-------"
    n = float(len(xs))
    b = (sum(ys)*sum(x2s) - sum(xs)*sum(xys))/(n*sum(x2s) - sum(xs)**2)
    m = (n*sum(xys) - sum(xs)*sum(ys))/(n*sum(x2s) - sum(xs)**2)
    if VERBOSE:
        print "---LSQ FIT (b,m)---"
        print "b,m: %f %f" %(b,m)
        print "---END LSQF (b,m)---"
    return (b,m)


def rectifyData(data,val): #data to be rectified about value val
    data = [val+abs(x-val) for x in data]
    return data

def main():
    root = Tk()
    root.minsize(850,600)
    app = SciFair(root)
    root.mainloop()

if __name__ == '__main__':
    main()
```

# Appendix C: Data

**Expedited and Formatted using various Python Scripts**

## Single-location Repeated Impact Data Results

CENTER:
LSQ_fit_thresh40:
average: ((0.000000,0.000000),5.491404)
rms: ((0.000000,0.000000),9.043545),
std_noSpline_thresh35:
average: ((0.000000,0.000000),2.458058)
rms: ((0.000000,0.000000),3.178050),
std_noSpline_thresh50.:
average: ((0.000000,0.000000),2.424539)
rms: ((0.000000,0.000000),3.057777),
std_spline_thresh35:
average: ((0.000000,0.000000),2.537142)
rms: ((0.000000,0.000000),2.810694),
std_spline_thresh50.:
average: ((0.000000,0.000000),1.776948)
rms: ((0.000000,0.000000),2.167948),
wvlt_thresh5_7:
average: ((0.000000,0.000000),0.696431)
rms: ((0.000000,0.000000),0.982607),

halfDown:
std_noSpline_thresh35:
average: ((0.000000,-15.000000),2.773451)
rms: ((0.000000,-15.000000),3.368976),
std_noSpline_thresh50.:
average: ((0.000000,-15.000000),3.256238)
rms: ((0.000000,-15.000000),5.674504),
std_spline_thresh35:
average: ((0.000000,-15.000000),2.858057)
rms: ((0.000000,-15.000000),4.260282),
std_spline_thresh50:
average: ((0.000000,-15.000000),3.356842)
rms: ((0.000000,-15.000000),4.652956),
wvlt_thresh5_7:
average: ((0.000000,-15.000000),1.453501)
rms: ((0.000000,-15.000000),1.929803),

halfLeft:
std_noSpline_thresh35:
average: ((-15.000000,0.000000),3.290389)
rms: ((-15.000000,0.000000),4.888763),
std_noSpline_thresh50:
average: ((-15.000000,0.000000),2.993962)
rms: ((-15.000000,0.000000),3.232646),
std_Spline_thresh35:
average: ((-15.000000,0.000000),3.479215)
rms: ((-15.000000,0.000000),5.572253),
std_Spline_thresh50:
average: ((-15.000000,0.000000),2.863379)
rms: ((-15.000000,0.000000),3.162278),
wvlt_thresh5_7:
average: ((-15.000000,0.000000),1.322406)
rms: ((-15.000000,0.000000),1.597412),

halfRight:
std_noSpline_thresh35:
average: ((15.000000,15.000000),15.299464)
rms: ((15.000000,15.000000),15.749603)
,
std_noSpline_thresh50:
average: ((15.000000,0.000000),1.768347)
rms: ((15.000000,0.000000),2.489980),
std_spline_thresh35:
average: ((15.000000,0.000000),1.524540)
rms: ((15.000000,0.000000),1.732051),
std_spline_thresh50:
average: ((15.000000,0.000000),1.723718)
rms: ((15.000000,0.000000),2.133073),
wvlt_thresh5_7:
average: ((15.000000,0.000000),1.755211)
rms: ((15.000000,0.000000),1.929803),

halfUp:
std_noSpline_thresh35:
average: ((0.000000,15.000000),1.758086)
rms: ((0.000000,15.000000),2.085665),
std_noSpline_thresh50:
average: ((0.000000,15.000000),2.728530)
rms: ((0.000000,15.000000),3.224903),
std_spline_thresh35:
average: ((0.000000,15.000000),1.921704)
rms: ((0.000000,15.000000),2.387467),
std_spline_thresh50:
average: ((0.000000,15.000000),2.315379)
rms: ((0.000000,15.000000),2.819574),
wvlt_thresh5_7:
average: ((0.000000,15.000000),1.303184)
rms: ((0.000000,15.000000),1.838290),

middleLowerLeft:
std_noSpline_thresh35:
average: ((-15.000000,-15.000000),8.311593)
rms: ((-15.000000,-15.000000),10.312614),
std_noSpline_thresh50:
average: ((-20.000000,-15.000000),11.225026)
rms: ((-20.000000,-15.000000),11.897479),
std_spline_thresh35:
average: ((-15.000000,-15.000000),5.975542)
rms: ((-15.000000,-15.000000),7.755643),
std_spline_thresh50:
average: ((-15.000000,-15.000000),5.896837)
rms: ((-15.000000,-15.000000),7.559762),
wvlt_thresh5_7:
average: ((-15.000000,-15.000000),2.045593)
rms: ((-15.000000,-15.000000),2.311888),

middleLowerRight:
std_noSpline_thresh35:
average: ((15.000000,-15.000000),8.767645)
rms: ((15.000000,-15.000000),9.964939),
std_noSpline_thresh50:
average: ((15.000000,-15.000000),7.418434)
rms: ((15.000000,-15.000000),8.252272),
std_spline_thresh35:
average: ((15.000000,-15.000000),7.164840)
rms: ((15.000000,-15.000000),8.136338),
std_spline_thresh50:

```
		average: ((15.000000,-
15.000000),8.422934)
		rms: ((15.000000,-
15.000000),9.300538),
wvlt_coeff5_7:
		average: ((15.000000,-
15.000000),3.637138)
		rms: ((15.000000,-
15.000000),4.081075),

middleUpperLeft:
std_noSpline_thresh35:
		average: ((-
15.000000,15.000000),6.134214)
		rms: ((-
15.000000,15.000000),7.218033),
std_noSpline_thresh50:
		average: ((-
15.000000,15.000000),5.940019)
		rms: ((-
15.000000,15.000000),6.723095),
std_spline_thresh35:
		average: ((-
14.000000,15.000000),5.155489)
		rms: ((-
14.000000,15.000000),6.557439),
std_spline_thresh50:
		average: ((-
15.000000,15.000000),4.701515)
		rms: ((-
15.000000,15.000000),5.477226),
wvlt_thresh5_7:
		average: ((-
15.000000,15.000000),2.604420)
		rms: ((-
15.000000,15.000000),3.537971),

middleUpperRight:
std_noSpline_thresh35:
		average:
((15.000000,15.000000),3.842858)
		rms:
((15.000000,15.000000),4.449719),
std_noSpline_thresh50:
		average:
((15.000000,15.000000),4.388703)
		rms:
((15.000000,15.000000),5.000000),
std_spline_thresh35:
		average:
((15.000000,15.000000),3.271140)
		rms:
((15.000000,15.000000),4.098780),
std_spline_thresh50:
		average:
((15.000000,15.000000),4.793891)
		rms:
((15.000000,15.000000),5.263079),
wvlt_thresh5_7:
		average:
((15.000000,15.000000),2.654459)
		rms:
((15.000000,15.000000),3.039964),
```

Raw Data For Single Point Tests

LSQ_fit_thresh40.txt
0, 0
5, 5
-1, 0
5, -2
2, -1
0, 0
0, -8
1, 0
4, -4
3, -6
3, -3
0, -2
0, -29
1, -7
-1, 0
2, 0
std_noSpline_thresh35.txt
0, 0
-3, 3
-2, 2
-3, 3
0, 0
-2, 0
-2, 2
-1, 4
0, 0
-1, 3
-2, 7
-1, 0
0, 0
0, 4
-1, 0
-1, 0
-1, 5
-1, 2
-2, 0
-1, 1
-1, -3
0, 0
std_noSpline_thresh50..txt
0, 0
0, 0
0, 3
0, 4
0, 0
-1, 0
0, 2
-3, 3
-1, 0
-1, 4
-3, 5
-1, 3
-1, 0
-1, 1
-3, 3
-1, 0
-1, 1
-3, 3
0, 5
-1, 1
0, 0
-2, 2
std_spline_thresh35.txt
0, 0
0, -3
-1, 0
-3, 0
-2, 0
-3, 4
-2, 2
0, -3
0, 3
-3, -1
-1, -3
1, -3
-2, 3
0, 2
-2, -1
-4, -1
0, 1
-2, 0
-2, 0
-1, 0
-2, 0
-1, 0
std_spline_thresh50..txt
0, 0
-3, 3
1, -1
-2, 1
-1, 1
-2, 1
-1, 1
0, 0
0, 1
0, 0
-1, 0
-1, 1
-1, 0
-3, 4
-2, 0
0, 1
-2, -1
-1, 0
-2, 1
-2, 0
-2, 1
-2, 1
wvlt_thresh5_7.txt
0, 0
0, 1
0, 0
0, -1
-1, -1
0, -1
1, 0
0, 0
0, 0
0, 0
1, 0
-1, 0
-1, 1
0, 0
0, -1
0, 0
0, -1
2, -1
0, -2
0, -1
0, 0
0, 0
1, -1
0, 0
0, 0
-1, -1
0, 0
1, 0
0, 0
0, -1
0, 0
std_noSpline_thresh35.txt
0, -15
0, -15
-3, -12
-1, -17
0, -13
0, -14
-1, -13
-2, -11
-1, -13
-1, -19
0, -17
0, -14
0, -7
-1, -16
0, -14
-1, -16
-3, -12
-2, -12
0, -11
1, -14
-4, -15
-3, -13
std_noSpline_thresh50..txt
0, -15
0, -14
0, -14
1, -29
0, -14
0, -15
0, -15
0, -15
0, -13
-3, -13
-1, -15
-3, -13
-1, -15
0, -13
0, -14
-1, -16
-2, -21
-1, -12
-1, -13
-2, -11
-1, -16
-2, 3
std_spline_thresh35.txt
0, -15
0, -16
0, -17
-2, -14
0, -16
0, -16
-1, -15
-2, -12
0, -17
0, -17
-1, -14
-1, -17
0, -16
0, -9
0, -18
2, -29
0, -18
-1, -15
-2, -20
0, -15
-6, -15
0, -14
std_spline_thresh50.txt
0, -15
0, -15
-2, -11
-1, -16
-4, -13
-2, -13
0, -17
0, -15
-2, -19
-2, -19
0, -17
-2, -19
-2, -21
-2, -11
-2, -17
0, -15
0, -16
-5, -15
2, -29
0, -15
-1, -11
0, -17
wvlt_thresh5_7.txt
0, -15
0, -15
0, -14
1, -14
1, -18
0, -20
0, -13
0, -15
2, -16
0, -14
0, -18
1, -16
0, -15
-2, -18
-1, -15
-2, -16
-1, -14
1, -13
0, -14
1, -16
-2, -16
0, -17
0, -15
0, -14
0, -15
0, -15
1, -17
0, -14
0, -15
0, -15
0, -13
std_noSpline_thresh35.txt
-15, 0
-17, 0
-16, -1
-15, 0
-14, -1
-16, 4
-16, 3
-17, 2
-17, 3
-5, -15
-17, 2
-16, 2
-15, 3
-16, 4
-15, 2
-17, 3
-16, 3
-16, 3
-16, 3
-17, 2
-16, 0
-16, -1
std_noSpline_thresh50.txt
-15, 0
-18, 0
-17, 2
-16, 2
-17, 2
-18, 0
-16, -3
-16, 0
-17, -1
-16, 4
-15, 2
-17, 3
-17, -1
-16, 2
-18, 4
-16, 2
-13, -1
-17, -1
-16, 4
-17, 4
-17, 3
-17, 4
std_Spline_thresh35.txt
-15, 0
-17, -1
-16, 2
-17, 2
-16, -1
-17, 2
-12, 1
-17, 1
-18, 2
-16, 1
-16, 1
-16, 3
-17, 2
-17, 4
-17, 3
-17, 2
-9, -21
-17, 1
-17, 1
-16, 2
-17, 2
-16, 1
std_Spline_thresh50.txt
-15, 0
-18, 1
-16, 0
-16, -1
-17, 3
-17, 3
-17, 0
-15, 1
-20, 4
-17, 3
-16, 2
-18, 2
-18, 1
-16, 2
-16, 3
-17, 3
-17, 2
-17, 3
-17, 2
-17, 2
-17, 0
-16, 2
wvlt_thresh5_7.txt
-15, 0
-17, 1
-16, 0
-15, 0
-16, 0
-16, 1
-15, 0
-16, 1
-16, 0
-16, 0
-14, -2
-16, 0
-16, 0
-16, 1
-14, -1
-17, 0
-17, 0
-16, 0
-16, 0
-15, 0
-16, 0
-16, 1
-17, 1
-14, -1
-16, 0
-16, 0
-15, 0
-17, 1
-16, 0
-18, 3
-17, 0
std_noSpline_thresh35.txt
15, 015, 1
15, 1
12, -1
15, 0
13, 0
15, -2
14, -3
16, 0
15, 0
14, 1
13, -2
14, -2
15, 2
14, 1
13, -1
13, -1

```
13, -2
14, 0
14, 2
14, 1
15, 2
13, -1
std_noSpline_
thresh50.txt
15, 0
16, -6
16, 2
15, 0
15, 0
15, 1
15, 0
15, 0
15, -4
14, 0
16, -3
15, 0
15, -4
14, 0
14, 0
12, -2
12, -1
15, 1
15, 0
16, 2
13, -1
14, 1
std_spline_thr
esh35.txt
15, 0
14, 1
15, -3
16, -1
15, 1
13, -1
14, -2
15, 0
15, -2
16, 0
16, 1
16, 2
16, 0
15, 1
13, -1
13, -2
13, 0
15, 1
14, 0
15, 1
16, 0
15, 1
std_spline_thr
esh50.txt
15, 0
15, -2
15, 0
12, -1
15, -3
15, 2
14, -2
16, -4
12, -2
13, -2
15, 0
15, 1
16, 0
15, 1
14, -1
15, 1
16, 1
14, -1
15, -3
15, 0
15, 1
15, 1
wvlt_thresh5_
7.txt
15, 0
15, 1
14, 0
15, 0
13, -1
14, 0
14, 0
15, -3
15, -2
15, -2
14, 0
13, 0
13, -2
13, -1
15, -1
15, -2
14, 1
14, 0
17, 0
13, -1
14, -2
14, -1
15, -2
13, -2
13, 0
14, -1
14, -1
15, -2
12, -1
14, -2
15, 1
std_noSpline_
thresh35.txt
0, 15
0, 15
1, 13
1, 11
0, 13
0, 14
0, 13
1, 12
-1, 12
0, 14
0, 13
0, 16
0, 13
-1, 15
1, 13
0, 13
0, 15
0, 12
0, 13
-1, 15
0, 14
0, 14
std_noSpline_
thresh50.txt
0, 15
-2, 13
-3, 16
-2, 14
-2, 16
-1, 16
-1, 15
-1, 15
-3, 17
-2, 15
-5, 16
-3, 16
0, 13
-2, 14
0, 17
-2, 17
-3, 17
-2, 16
6, 13
-2, 15
-6, 13
0, 15
std_spline_thr
esh35.txt
0, 15
1, 13
-1, 15
-1, 15
0, 13
0, 13
-1, 14
-1, 15
-2, 14
-4, 12
-2, 12
-1, 13
-1, 14
0, 13
-2, 10
0, 14
-2, 13
-1, 15
0, 15
-1, 15
-1, 15
-1, 15
std_spline_thr
esh50.txt
0, 15
0, 16
-6, 20
-2, 16
-3, 15
-2, 15
0, 17
0, 14
-1, 16
-1, 16
-4, 14
-2, 15
-1, 15
-2, 16
-1, 15
-2, 17
-3, 16
-2, 16
-2, 15
-3, 16
-1, 15
-2, 15
wvlt_thresh5_
7.txt
0, 15
-1, 14
0, 14
-2, 16
0, 15
-2, 14
0, 15
0, 15
0, 16
-5, 18
-1, 15
0, 14
0, 14
0, 15
-1, 14
0, 15
0, 16
0, 14
1, 14
-3, 18
0, 14
-2, 16
0, 14
0, 15
0, 14
-2, 17
1, 16
-1, 14
-1, 15
1, 14
0, 15
std_noSpline_
thresh35.txt
-15, -15
-10, -30
-14, -29
-14, -16
-15, -17
-14, -18
-9, -29
-14, -17
-15, -29
-14, -16
-10, -29
-16, -21
-16, -18
-17, -22
-11, -30
-13, -29
-13, -15
-13, -29
-13, -15
-14, -28
-15, -20
-15, -23
std_noSpline_
thresh50.txt
-20, -15
-14, -18
-14, -19
-14, -21
-14, -28
-13, -26
-10, -15
-8, -20
-11, -27
-11, -18
-8, -24
-10, -22
-13, -26
-14, -20
-16, -28
-15, -19
-11, -18
-10, -23
-16, -27
-10, -11
-6, -24
-12, -16
std_spline_thr
esh35.txt
-15, -15
-14, -16
-14, -25
-16, -19
-15, -22
-13, -29
-15, -28
-15, -18
-14, -16
-15, -18
-14, -17
-15, -19
-15, -29
-12, -15
-16, -18
-16, -21
-15, -26
-13, -13
-15, -17
-15, -17
-14, -30
-15, -18
std_spline_thr
esh50.txt
-15, -15
-14, -17
-14, -18
-15, -19
-16, -18
-13, -29
-15, -20
-15, -18
-12, -25
-16, -17
-14, -12
-16, -19
-15, -18
-15, -19
-17, -18
-13, -29
-16, -17
-14, -29
-14, -18
-10, -12
-16, -19
-16, -30
wvlt_thresh5_
7.txt
-15, -15
-16, -16
-15, -17
-15, -16
-17, -18
-15, -13
-16, -17
-15, -16
-17, -16
-15, -17
-16, -16
-15, -16
-15, -17
-16, -12
-15, -15
-15, -16
-15, -16
-16, -18
-14, -12
-15, -17
-15, -17
-16, -18
-15, -17
-15, -17
-15, -17
-16, -16
-16, -17
-18, -19
-15, -16
-16, -12
-15, -17
std_noSpline_
thresh35.txt
15, -15
20, -29
17, -27
17, -25
17, -25
14, -28
14, -17
15, -20
17, -21
16, -16
14, -26
18, -29
15, -20
15, -24
16, -28
15, -23
16, -27
15, -23
11, -15
17, -25
14, -28
14, -15
std_noSpline_
thresh50.txt
15, -15
13, -21
17, -26
15, -22
17, -20
14, -19
12, -15
13, -27
17, -17
12, -22
10, -20
15, -27
15, -26
14, -21
14, -13
13, -26
16, -25
12, -11
12, -19
13, -21
13, -24
15, -26
std_spline_thr
esh35.txt
15, -15
17, -28
19, -26
15, -17
13, -23
18, -25
17, -24
```

18, -25
17, -23
18, -19
18, -19
19, -22
14, -21
18, -24
14, -15
13, -18
15, -26
14, -22
14, -22
14, -24
14, -18
14, -14
std_spline_thresh50.txt
15, -15
10, -28
13, -23
14, -23
18, -27
15, -24
11, -11
13, -21
12, -20
19, -28
16, -17
18, -26
13, -22
14, -24
14, -26
17, -18
17, -23
17, -18
17, -27
16, -24
16, -27
14, -19
wvlt_coeff5_7.txt
15, -15
18, -18
14, -17
13, -19
18, -19
13, -17
16, -21
13, -19
15, -17
13, -14
16, -20
12, -18
13, -15
15, -20
15, -19
18, -19
15, -16
13, -15
16, -22
14, -15
15, -16
13, -19
14, -20
14, -16
14, -16
13, -15
16, -20
13, -19
17, -19
16, -21
17, -18
std_noSpline_thresh35.txt
-15, 15
-14, 12
-14, 13
-15, 14
-15, 12
-10, 7
-14, 12
-13, 11
-12, 9
-15, 12
-8, 9
-5, 5
-9, 7
-16, 12
-13, 8
-10, 7
-14, 11
-9, 7
-10, 9
-15, 12
-13, 11
-9, 7
std_noSpline_thresh50.txt
-15, 15
-13, 11
-10, 7
-9, 7
-12, 10
-10, 9
-10, 8
-9, 8
-10, 8
-12, 9
-13, 10
-18, 12
-17, 13
-13, 12
-9, 8
-10, 7
-14, 12
-14, 14
-11, 10
-13, 13
-16, 14
-16, 11
std_spline_thresh35.txt
-14, 15
-16, 15
-15, 13
-16, 14
-9, -2
-16, 15
-13, 11
-11, 8
-14, 11
-14, 11
-15, 12
-15, 12
-16, 12
-6, 6
-11, 6
-13, 12
-15, 12
-17, 12
-10, 8
-15, 11
-15, 11
-14, 11
std_spline_thresh50.txt
-15, 15
-15, 11
-11, 9
-15, 13
-15, 13
-12, 11
-15, 12
-15, 13
-14, 8
-17, 13
-15, 13
-12, 11
-9, 9
-11, 7
-22, 17
-14, 12
-16, 15
-11, 9
-12, 12
-10, 9
-14, 14
-10, 10
wvlt_thresh5_7.txt
-15, 15
-20, 18
-19, 19
-20, 18
-22, 22
-19, 17
-16, 15
-16, 15
-16, 14
-15, 14
-15, 14
-16, 15
-16, 15
-17, 15
-19, 18
-16, 14
-16, 15
-19, 16
-20, 18
-16, 14
-17, 16
-16, 16
-17, 15
-16, 15
-15, 16
-16, 15
-19, 17
-16, 15
-15, 15
-15, 15
-19, 16
std_noSpline_thresh35.txt
15, 15
21, 16
15, 11
12, 9
12, 8
11, 10
15, 16
15, 12
16, 14
16, 13
16, 12
15, 12
15, 11
15, 13
14, 9
16, 11
15, 13
13, 13
14, 10
15, 14
12, 9
13, 14
std_noSpline_thresh50.txt
15, 15
19, 14
13, 8
16, 14
17, 16
21, 16
13, 14
14, 10
18, 13
17, 13
18, 14
15, 12
17, 15
18, 14
13, 9
13, 9
17, 12
11, 6
13, 10
12, 8
17, 13
19, 15
std_spline_thresh35.txt
15, 15
21, 22
17, 16
13, 9
12, 8
16, 16
20, 20
16, 13
16, 11
16, 12
17, 16
16, 16
15, 16
16, 13
17, 13
16, 15
17, 15
16, 12
16, 16
15, 12
15, 11
15, 16
std_spline_thresh50.txt
15, 15
15, 11
12, 8
19, 17
12, 12
14, 11
15, 13
16, 16
15, 13
14, 9
12, 10
12, 8
13, 9
16, 13
13, 9
13, 10
10, 10
17, 13
11, 11
19, 16
12, 11
13, 9
wvlt_thresh5_7.txt
15, 15
14, 14
14, 12
15, 15
14, 14
12, 11
16, 15
14, 13
14, 14
12, 11
13, 11
13, 12
15, 15
15, 14
16, 15
12, 13
13, 14
12, 13
13, 13
14, 12
12, 13
13, 12
14, 12
14, 12
13, 14
14, 14
12, 15
14, 14
13, 11
14, 12
12, 12

## Spiral Data

Actual Impact Points:
0, 0
0, 2
-6, 0
-5, -5
0, -8
6, -5
9, 0
5, 10
0, 13
-10, 10
-16, 0
-14, -10
-5, -17
0, -18
10, -15
19, -4
20, 0
19, 4
15, 16
4, 21
0, 23
-15, 20
-25, 10
-27, -1
-25, -10
-19, -20
-10, -26
0, -28
14, -25
25, -15
30, 0

Calculated Impact points
-1, -1
0, 2
-7, 1
-5, -6
-1, -8
5, -5
6, 0
5, 10
-1, 14
-12, 11
-20, 2
-15, -8
-7, -20
-1, -16
13, -19
21, -2
21, 1
20, 4
14, 14
3, 19
-1, 22
-19, 21
-24, 10
-26, -1
-23, -9
-17, -17
-8, -23
1, -29
10, -28
22, -16
30, 3