Sprout & Co.

August 28, 2011

Street Bump Solution

Team Members:

Michael Nagle

Eliot Kristan

Benjamin Hitov

Raphael Luckom

Table of Contents

Section I. Description of Algorithm

A. General outline of our approach

Our approach was to use the magnitude of acceleration spikes along the phone's z-axis (the vertical axis as the car is traveling forward) to identify impacts, and then use other filters to distinguish potholes from other road events. We based our approach on a report called the Pothole patrol [pdf] (written by MIT's Computer Science department) that used 400-Hz accelerometer data to map road hazards. It identified spikes in the z-axis acceleration as evidence of bumps, and then used other filters to distinguish potholes from other events.

The other filters we tested were:

- **Minimum speed filter** – to make sure the car is actually traveling (as opposed to an impact caused by slamming a door in a stationary car)
- **Z-axis / x-axis ratio filter** – this compares the acceleration along the z-axis to the acceleration along the x-axis, the axis going between the wheels of the car. The idea behind it is that a pothole event usually only affects one wheel of the car (or two on the same side), therefore a z-spike produced by a pothole should be accompanied by a spike in the right or left direction as the car is traveling forward, depending on which wheel hit the pothole.
- **Z-axis / speed filter –** this compares the z-axis acceleration to the speed of the car. This is making sure that the large spike in z-acceleration came not just from driving quickly over something insignificant, but that the ratio is high enough to claim that something's off with the road.

We found that these filters did not improve the algorithm's performance and chose not to implement them.

Our filters identify spikes by measuring the standard deviation of each acceleration indicator in a trace. Our threshold values are calculated in terms of standard deviations (e.g. taking all points whose z-acceleration is more than 2 standard deviations outside the mean.) This helps account for differences between cars and ways of positioning the phone–each point is evaluated against its own data set, rather than an arbitrary acceleration value which may not capture the differences between cars (like car suspension systems, phone positions, etc.)

Using the Rutherford data as a test case, we found setting the threshold value for the z-acceleration at 3.45 successfully returned the 3 identified pothole points on the map and no other points. We found that the GPS location we returned was not precisely on top of the point in the given map, but that the actual site contains six potholes in a string and we believe the location is slightly different due to this.

B. Severity – picking an algorithm and on-site evaluations

General Description

Once we have an identified a pothole location, we need to assign it a severity score from

1 to 100. We start with a location that we have already determined to be a pothole. Then we take the mean of the z-acceleration values across the entire data set and subtract that from the specific point's z-acceleration value. We then take the absolute value of this number – recognizing that a car may be physically moving up or down after hitting a pothole – and convert it to a 1 - 100 scale by multiplying it by 10. This gives a reasonable estimate of the severity of the pothole on a 1 - 100 scale.

Process

In order to figure out how to turn these values into a 1 - 100 scale, we decided to do some human scoring of potholes first, and then approximate those scores algorithmically. We started by taking a trip to the pothole sites (as we're local to Boston) to try and come up with a method of measurement. We initially went to Hyde Park, and took photos with rulers and tennis balls to size the potholes – the ruler measured length / width, and the tennis ball was a referent for depth.

See http://www.flickr.com/photos/connors934/tags/potholes/

Then, after the new data release, we went to the Rutherford site to come up with 1 - 100 scale scores. On the Rutherford trip, we scored based on width, length, and depth of pothole, and when appropriate, length of bumpiness / shock to the car (such as the strip of uneven road underneath the overpass at Rutherford.) Photos of this trip can be seen here: http://bit.ly/phEt2z . We just used a tennis ball to take rough measures, and did not use a ruler, as we did not have much time on the busy intersection. The subjective scoring process is explained over video here:

http://www.youtube.com/iameliot#p/u/2/mg9vW3yjlMc

and we came up with the following subjective scores for the three events at the Rutherford site:

- Point 1 (the most northern point) was a set of three 10 - 15s followed by three 20 - 25s
- Point 2 (the northern underpass point) was a 50 – a long strip of damaged pavement
- Point 3 (the southern underpass point) was a 35 – a lesser strip of damaged pavement & smaller potholes

We then used this as a way to turn our computed acceleration values into a 1 - 100 scale. Our current algorithm returns these three pothole points successfully – and no false positives – and returns the severity scores as:

- Point 1 computed: 20.99
- Point 2 computed: 40.14
- Point 3 computed: 20.06

These values broadly mirror our subjective scores for the potholes we observed.

Section II. Challenges with the new data from revision 1.3

A. Issues encountered

As we examined possible threshold values for each indicator, we encountered a challenge: there were some traces with little or no recorded data near potholes indicated on the supplied maps. No matter what threshold values we used, we didn't find potholes we knew were there.

When we looked at the data itself, we found a possible reason: the Street Bump app polls the phone's accelerometers once every four seconds. When it detects an acceleration above a certain level, it records more frequently. Early versions of the app used a default configuration that provided many such 'windows' of high-frequency data points, making it more likely that all potholes would be recorded. As of version 1.3, however, the default settings raised the threshold for triggering more frequent polling. This new threshold results in few – if any – high-frequency-windows in most traces, reducing the chance that any pothole event will be recorded in detail.

Additionally, this higher-threshold triggering also precludes truly accurate pinpointing of the potholes themselves; over a four-second window, a car traveling at a slow 10mph will cover just under 59 feet. This suggests that based on the likely data, it will be impossible to provide the 'within five yards' (15 feet) of accuracy requested, regardless of the filters used.

After observing these constraints, we began looking for a method of finding the best threshold values for our filters. We found that while it was possible to 'tune' the filters to report only correct potholes for some traces, the threshold values that gave good results in one case did not necessarily give good results in others. The low frequency of the data points contributed significantly to the uncertainty; for a given trace it was difficult to tell whether the algorithm was 'missing' a pothole or whether the data simply gave no indication it was there. This suggested that what we needed was a better way to evaluate the data itself, rather than a way to find potholes under the assumption that the data was sufficient and complete. See the description of the Street Bump Visualization presented in Section III to learn more about our evaluation tool.

B. Recommendations for data and phone settings

We recorded several runs around the Rutherford Ave. site using a Samsung Galaxy S phone, some of which are on YouTube (see http://bit.ly/nWsJjx & http://bit.ly/q0Q6P6) and several of which are uploaded to Street Bump's site under the user "nagle" - http://apps.citizapps.com/postgeo/ID:0.5582044972973915 (tracks 4 through 13). A description of each track is as follows::

- Track 4 - Phone on the floor in same position as new challenge data. All settings at default with accel threshold of .5
- Track 5 - Phone on floor w/ .3 threshold starting from northerly pothole region
- Track 6 - Phone on floor at .2 thresh

- Track 7 - Phone mounted on top of dash, .2 threshold
- Track 9 - Phone mounted on top of dash, .2 threshold
- Track 10 - Phone mounted on top of dash, .1 threshold
- Track 11 - Phone mounted on ash tray going back to .5 default threshold. Short track to test theory if the phone positioned on the floor was the critical variable between old datasets and new datasets window triggers.
- Track 12 - Phone mounted on top of dash .15 threshold
- Track 13 - Phone mounted on top of dash and .15 threshold., 1.5 loops around the track

Taking several runs with different acceleration threshold values in the app's settings, we believe that an ideal setting would be 0.15 (which map can be seen here: http://maps.google.com/maps?f=q&source=s_q&hl=en&geocode=&q=http://apps.citizapps.com/mapgeo/ID:0.5582044972973915/13).

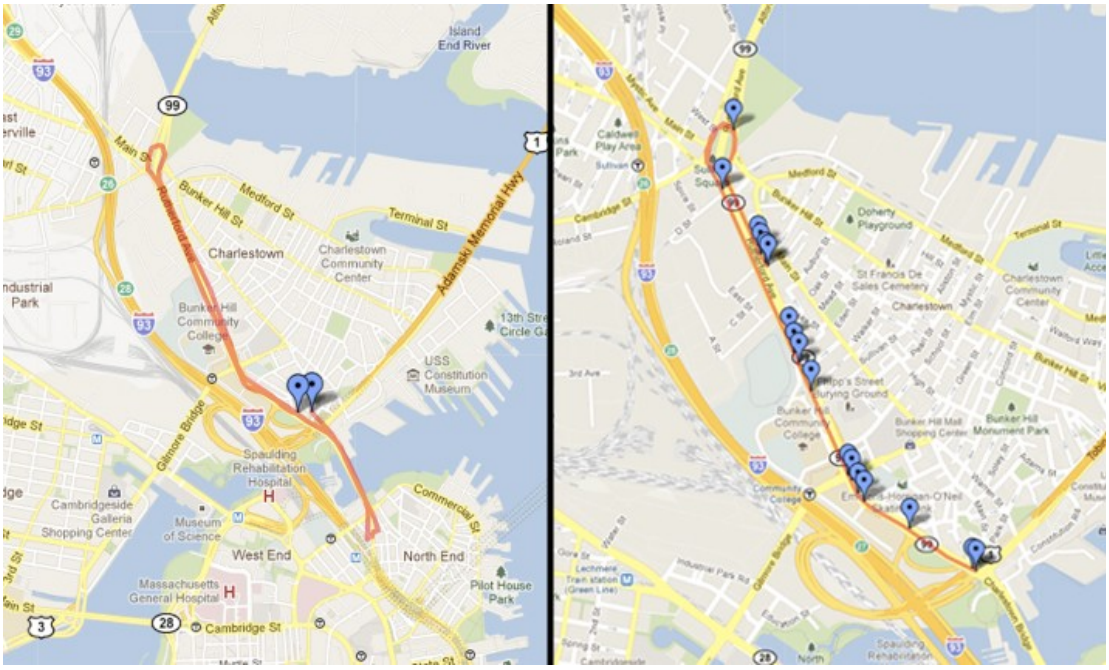

Illustration 1: The Rutherford track on the left was recorded with the default Street Bump app settings while the version on the right was recorded with an Acceleration Threshold value of .15. The blue markers indicate when event windows are triggered and higher frequency sampling occurs.

We also found significant differences in the app's behavior depending on whether it was on the carpet (http://bit.ly/r2qNir) or dashboard of the car (http://bit.ly/r9LSbQ). We found the Android was much more sensitive when on the dashboard of the car – we expect this is due to the carpet damping the accelerometers. We would recommend either having a universal suggestion to users for common placement of the Android when running Street Bump (likely somewhere in reach of the car charger) or seeing if there is a meaningful way to detect the dampening of the surface and accordingly adjust for it.

Section III. The Street Bump visualization.

(Please see http://sb-filter.appspot.com for the visualization)

In trying to both get better intuition for the filters we were using and pick appropriate threshold values, we chose to make a simple visualization of the Bump data. The visualization is a simple concept – it is an overlay of a Bump data set onto a Google map, with sliders for each filter that lets you see the effect of the filter on the Street Bump data set.

On the map, known potholes (based on supplied data) are displayed as points and a different color marker is used to indicate any data points which pass through the current set of filters, as adjusted by the sliders. The key idea is that you can then can adjust the filter thresholds until the events found by the filters line up as nearly as possible with the known potholes (or until it becomes clear that the data is insufficient to find them.)

We think that this is not only a useful exploration tool for understand filters more intuitively, but could also help in seeing what thresholds should be used in the Bump app as defaults so that sufficient data is always recorded to filter down to the pothole events.

The visualization is programmed in Python and relies heavily on Google Maps and jQuery with AJAX interactions for dynamic updating. It is currently hosted on Google App Engine.

To get started using the application you first upload a Street Bump CSV file. You will see all of the points of the track displayed on the Google Maps map. Once uploaded you can enable the various filters by checking the adjacent check boxes. Once enabled, sliding the threshold slider will change the threshold of each filter and automatically update the points on the Google Maps map. Clicking onto a single point in the interface will give you the severity score of that location.

Known location markers are automatically plotted on the map. Yellow markers as well as red markers with a star in them represent known pothole locations. White markers represent non-pothole locations.

We have included the source code for this web app in our submission as the file “webapp.zip”.

Section IV. Other Insights and Strategies

A. Introduction – techniques which require higher sample rates or were otherwise unimplemented

Our algorithm became substantially simpler after the update to version 1.3 was released. This was because with the usual sampling rate being once-every-four-seconds, we found it was difficult to apply many of the techniques we'd initially included.

We mention them here to spur ideas for more precise algorithmic techniques and to share what we think are potentially valuable strategies. We also believe that techniques like these make a strong case for reducing the default accelerometer threshold in the Street Bump app and raising the sampling rate of data collected.

We include in this section a few suggestions and additional techniques that we think could make useful additions to Street Bump.

B. Patterns in orientation

One of the interesting differences between Street Bump and the paper we cited by MIT, Pothole Patrol, is that Street Bump runs on an Android which is not fixed to any surface, whereas the Pothole Patrol setup mounted accelerometers securely to seven taxis. While the non-standard positioning and possibility for the phone to move around seems a difficulty at first, it appears there's a benefit to this. On pothole events, the phone is likely to flip up slightly, in a way that is revealed in the orientation data around the event.

Specifically, it appeared that on a pothole event, noticeable spikes would appear in the OrientZ and OrientY columns of data, and that these could be use to narrow in on the exact data point which corresponded to a pothole (where the maximum / minimum of the spike was).

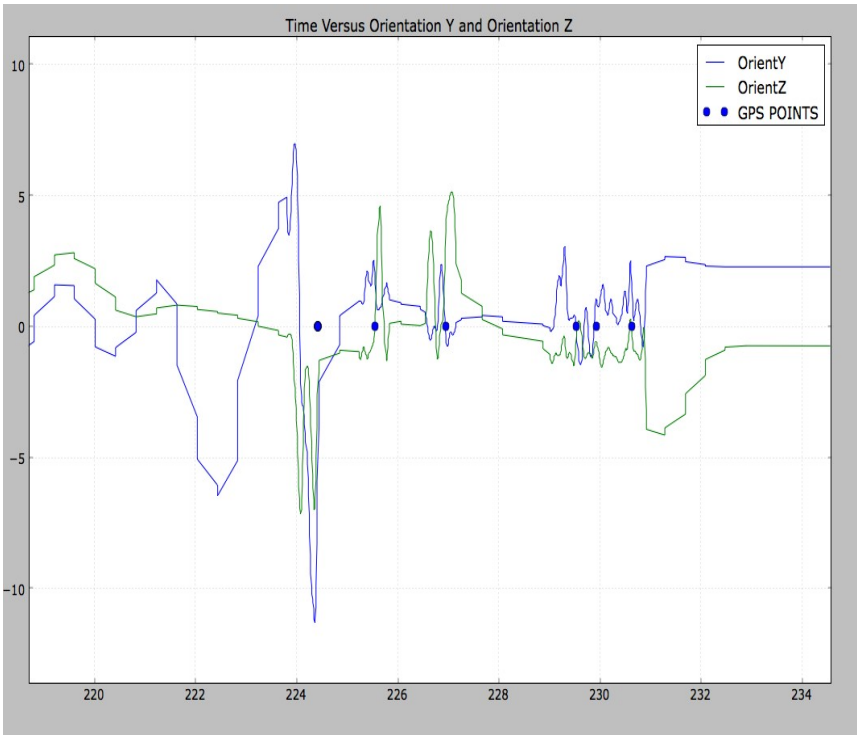

Illustration 2: Above is a graph of a section of the Hyde-Park-Round-1.csv data that displays the orientation y and orientation z values in relation to time. The data has also been put through a high pass filter which normalizes it to 0 as well as a triangular smoothing function. You can see in the graph that with the higher frequency sampling there are distinct spikes and patterns.

C. High-pass filtering

A useful technique with higher sampled data is the ability to put it through a high-pass filter. This filters for events of a higher frequency than the filter threshold – which both identifies impulses (such as spikes in acceleration, or orientation, per the last section) and will also remove low-pass events. This means that the data will then get centered around zero, as the average of the data will be seen as a very low-frequency signal to be removed.

Additionally, variances in acceleration that happen over a few minutes – such as the car being on a sloped road for a while – should also be filtered out as a low-frequency component. This ultimately leaves us with data that consists of impulse events that will be separated out from variances in the car or the particular road.

D. Windowing

Another technique we explored was capturing data into “windows” each of which would be separately analyzed. This is based off of the observation that Street Bump collects

data more frequently when the set accelerometer threshold is crossed – and so a noticeably high sample rate surrounds at least one bump event. The idea would be to break the data from one large data sets to several "windows" – each of which is known to contain at least one (though possibly multiple) bump events.

This makes the analysis substantially easier – we know that non-eventful data will not be captured in a window, and we can then look for peaks in acceleration as well as orientation to identify singular bump events. We can also use the window to gauge pothole severity – by looking at the mean acceleration of the window as well as its length.

This process depends on sampling at a high enough rate to trigger the accelerometer threshold frequently, but is a way to break down the large data sets into more manageable chunks. This would be as simple as setting the default accelerometer threshold rate to 0.15 – no major overhaul would be needed to achieve this.

E. Adjusting for different phones' orientations

As the scope of the Innocentive challenge appeared not to be to make the algorithm work universally for any user, but rather to do well with the given datasets, we ultimately chose not to implement a function to normalize or otherwise standardize orientation.

We felt that was ultimately a pretty complex problem; what follows is our best idea for tackling the problem:

The goal is to find a rotation vector by which to rotate all data points by so that the average acceleration of the dataset is the closest possible fit to gravity (meaning a value of 9.8 in the AccelZ column and 0 in the AccelX and AccelY column– or as close as possible.) The idea is to assume the phone stays in some orientation for the whole trip, and to rotate it to a standard position that is effectively face up.

So rotate the dataset by all possible rotation vectors, look at the average acceleration, and see which comes out closes to gravity.

This is a naive way of doing the algorithm – optimization techniques such as gradient descent would be far more efficient in determining the correct rotation vector. (With gradient descent, you'd minimize the distance of the resultant mean acceleration vector to the gravity vector, so that you found out what the best rotation vector was.)

The key idea is to observe that most acceleration events will be met with an equal and opposite event (speeding up results in braking, and hitting a pothole and going up will be met by coming back down.) We don't know how well this holds up in practice, but it was our guiding idea. It may be that OrientX data gets ignored for this, since that will naturally change as the car turns.

F. A non-algorithmic suggestion: voice confirmation of potholes

The natural thing that comes up when telling people about working on Street Bump is something like "couldn't you just crowdsource finding potholes?"

Usually, they mean "couldn't people just take photos of potholes and send them in, GPS

stamped and everything?" Many people say they'd love to do this if they knew it was going to the city.

One obvious problem with this is that not all potholes are easy to photograph. The potholes under Rutherford Ave.'s overpass are a clear example of this. There's nowhere to stop the car to take photos, the lighting is already tough because of the overpass, and they run for quite a long stretch – capturing this would be difficult. (We tried unsuccessfully to video tape it several times!)

Another problem is that some people think you are a troublemaker if you start photographing and measuring potholes in the street. This video shows us a friendly visit from the Boston Police Department while we are measuring potholes on Rutherford Ave. – http://www.youtube.com/iameliot#p/u/0/DyaxVLmtSGk

Another approach would be to let the Street Bump app have a voice-confirmation feature. It would beep whenever a high-acceleration event happened, and then prompt the user to somehow indicate whether the event was a pothole, perhaps by saying "Yes / No / I don't know." With voice recognition native to the Android platform, this seems like a natural integration, and could be a powerful filter for identifying potholes (especially when used across many cars.)

Appendix A: Pseudo code and other explanations of what's underneath our hood

Find_Potholes.py

The algorithm is included as attachment "alg_submission.zip". Once unzipped, navigating to the directory "alg_submission" in terminal with Python running on your system and inputting "python find_potholes.py" will execute the script and print the JSON object to screen.

Variables

- **dataset_file** represents the filename of the csv input file.
- **reader** is a list of lists representing the data. Each entry in reader is a list representing a data point.
- **headers** represents the first line in the csv file; the line with the headers. After the headers variable has been set, the first line of the reader variable is discarded, so it contains only the data.

Algorithm

The first step cleans up the data by removing any rows with fewer than the normal number of columns and converting all values to floating point numbers so they can be manipulated more accurately.

After the data has been prepared, the next lines control which filters from filters.py will be applied to find pothole events. We have used only the z-spike filter in the current implementation, as it gave the most consistent results. Other filters could be added in a chain–using the output of earlier filters as the input for later ones. The remaining points are assigned to the variable filtered_points.

Next, the algorithm applies the scoring function (described in section I.B.) to the remaining points.

Finally, the algorithm organizes the points remaining into a JSON object requested by the City of Boston and prints it to screen.

general_std_peaks (Z-spike filter function)

We've created a single function (general_std_peaks) for finding spikes outside an arbitrary number of standard deviations from the mean. Its arguments are:

- **data**: a list of lists, each element of which represents a data point
- **target_column**: an integer representing the index within each data point of the indicator to be measured. Defaults to 5, which is the index of the z-acceleration in the data points.
- **devs**: a float representing the standard deviation threshold; for instance devs=3.4 will return points outside 3.4 standard deviations of the mean.

Defaults to 3.4, which we found to be the most accurate on the self-test dataset (though it did not give good results on all datasets, as previously mentioned).

- **mode**: The mode variable determines whether the filter returns all points outside the specified number of standard deviations for the indicator, or only those above or below the number of standard deviations. When mode is set to 'both', the function returns all points outside the specified standard deviations. When set to 'high', the function only returns points above mean by more than the specified number of standard deviations, and when set to 'low', only those below the mean. Defaults to ‘both’.

Appendix B: Media & Other Resources

Flickr Stream of pothole measuring pictures at the Hyde Park site:

http://www.flickr.com/photos/connors934/tags/potholes/

Picasa album with more photos of potholes at the Rutherford Ave. site:

https://picasaweb.google.com/115081280190429273690/RutherfordStreetBumpPhotos?authuser=0&authkey=Gv1sRgCLuRkYupxZPviwE&feat=directlink#

Youtube channel including videos of pothole measurement at the Rutherford Ave. site:

http://www.youtube.com/iameliot

Street Bump recorded tracks under user “nagle”:

http://apps.citizapps.com/postgeo/ID:0.5582044972973915

Assembla workspace and code repository:

http://www.assembla.com/code/sprout-sb/subversion/nodes