HOW-TO: Motion Tracking with PhaseSpace

Adam Smith (amsmith@ucsc.edu)

September 2005

Contents

1 Introduction

This document was created to give you a high level walk-through for using the PhaseSpace motion tracking system. You should read the referenced parts of the original documentation supplied with the system as part of each step of the process. Often, alternative procedures, tips, and pitfalls will be discussed here as an improvement over the original methods. You can find discussions of additional technical details, related tools, and location of important files (including electronic versions of the original documentation) in in the appendices.

2 About the Sensor

The PhaseSpace system is an active, optical, indoor, wide-area, motion tracking sensor. Simply put, it uses data from several cameras oriented about a common space to infer the precise 3D location of pattern-driven LED markers in real time. The original documentation, **PhaseSpace Motion Digitizer: Overview**, has a high level description of the system. From this document it is important that you understand the names and functions of the devices involved (cameras, LED drivers, LEDs, hub, server, and pre-wired calibration objects). The system setup and capture work-flow summaries presented are very general and are likely to be highly dependent on your specific need, so it is less important to understand these.

2.1 Capabilities

The PhaseSpace system is quite capable, but you should compare your requirements against the following to make sure its use is appropriate. The system is able to report the location of single markers and the pose of rigid bodies (collections of markers that are fixed relative to one other), over 90 markers simultaneously. A new estimate for the location of each point is reported at 480Hz with precision down to less than 1mm. Compared with other active optical trackers, the PhaseSpace system is fairly robust against dropouts due to occlusion because of the number of individual cameras involved.

2.2 Limitations

However, there are some not-very-obvious limitations to the system. Due to imperfect calibration, when a marker moves from zone to zone (that is when the set of detectors reporting the marker changes due to visibility) the system can report discontinuous motion. Also dependent on the quality of calibration, differing scale and orientation errors in different parts of the sensing space break down the linearity of the space — a one meter distance in the world is not always reported as a one meter distance by the system, but it is usually close (normally withing 3%). Because the LED drivers sync with the hub using radio you should take care not to shield or jam the (sometimes) two-way communication. Different hardware will perform differently, though direct line of site is usually not necessary.

3 Planning the Configuration

There is nothing in the original documentation about planning out the configuration of the sensor based on your requirements, so keep your attention focused here. Deciding where to place cameras ahead of time can save you reconfiguration time in the future and improve the quality of the data reported by the system if you take some important constrains into consideration. Also, you should decide how many cameras you want to deploy. The system can operate usefully with anywhere from four to sixteen cameras. The most general method is to simply place as many of the cameras as you can in a ring around the space in question, but often modifying this setup by pushing some cameras in and out or rotating others is beneficial. This process is very complex and there is no one right way to do it — though getting out a piece of paper, drawing it out and walking around the future location and visualizing it is a fine way to start!

3.1 Budgeting Space

Depending on the shape of the space you want to sense, a fully deployed system can take up vastly more room than you might expect. The cameras are most reliable at a range from 1m to 5m and within a conic field of view of about 50 deg. These numbers, of course, will vary with each generation of the hardware. As a result, you need to budget enough space to allow the cameras to be set far enough away from the action that they are both out of the minimum range and cover enough space side-to-side to track the action. Pulling the cameras back even more enhances the redundancy of the system as it usually allows the sensitive space of several cameras to overlap more, yielding a more stable reading, however, Due to the radial nature of optical sensors, they are much less sensitive to small details at long range. You should keep this balance of precision and robustness in mind when selecting a location for the cameras, often there will be a configuration that uses the sensitive space of the cameras and has generous overlapping. Some day you will actually be setting this up, and at that time you will have to worry about running cables and setting up tripods, but its more important to consider these issues (that you would still have even with magic wireless cameras that hung in space on their own).

3.2 Visibility Concerns

In order for the system report a marker location, the marker must be seen by at least three detectors. Since there are two detectors in each camera that means at least two cameras are necessary to localize a marker. Having the cameras' views meet at a perpendicular is more desirable than having only two nearly parallel views as it constrains the location of the marker better. This being said, when you are trying to create overlap between cameras, make sure you get overlap with cameras at different angles, not just cameras nearby that are nearly parallel. Because of possible discontinuities in motion when a marker loses or acquires visual contact with a marker you should arrange your cameras so, if possible, that the motion of the markers does not bring cross each cameras visibility boundary more than necessary (i.e. avoid placing cameras behind lattices, pillars, other obstacles, or along the line where tracked objects will likely stack and occlude one another). It is not important for every camera to see every part of the space you are interested it, it is more important that each camera has a clean view of its local part of the space.

4 Deploying the Cameras

This is the first step where you actually play with the hardware. Physically setting up the server is a fairly straightforward, uncreative process, however setting up the cameras is a bit more challenging (and I'm not just referring to all of the work you will be doing screwing in cameras, plugging in wires, and adjusting mounts above your head while standing on a step ladder). The original documentation, **PhaseSpace Motion Digitizer: Camera System**, covers the details of the setup process. Hold off on actually performing the setup until you have read the rest of this section though. Also, don't worry about the calibration procedure at this stage.

4.1 Rigidity

The detectors inside of each camera are very sensitive to angular changes. When you are deploying the cameras you should be mounting them rigidly to stable objects. If your world were perfect and you knew nothing would ever bump your cameras simpling setting them on the floor would be fine, but there are always little details like drafts, vibrations, clumsy visitors, clumsy you, or even experimental killer robots on the loose that you might come across — it really is best to secure the cameras tightly. Heavy duty tripods or ball heads mounted to lab benches are quite effective. Every time you suspect any one of the cameras has been bumped (or even if you just noticed degraded performance from an untouched system that was calibrated several days ago), you should recalibrate the system. Rigid mounting should reduce the need for this.

4.2 Interference

Even though the cameras are outfitted with optical filters to block out light of the wrong color, even though there is code in the server to reject signals of the wrong shape, and even though the markers are pulsed with a very specific pattern you still need to be aware of possible sources of interference. When the camera is analyzing the signal it gets from each of its detectors, it is looking for localized patches of pulsed light. Large blackbody radiators (halogen or incandescent, the sun), and their reflections, are generally not a problem. Arc lamps, and possibly fluorescent lights can confuse the camera if they are bright enough or focused enough. More common, however, is that reflections of real markers will confuse the sensor. Luckily, these problems are easily avoided. Normally, if the interference source is an overhead light, the cameras can simply be aimed downward until the light is out of view. If a camera must be aimed in a direction that it will see a light in you can setup a shade or diffuser in the path of the light to the camera. As for reflections, shiny spots can be covered with a diffuse layer (a sheet of paper). To check if you are really having problems due to interference of other light sources you can use the **usbscope** tool in image mode on the server once the cameras are deployed.

4.3 Cabling

The PhaseSpace uses Ethernet cable to connect all of the cameras together and to the hub (the four conductor cable is used as USB by the system). As you learned in the original documentation, the cameras are arranged into several chains in a deployment. While it is possible to put up to six and possibly seven cameras on a chain, the connection to the more distant cameras becomes unreliable. The hub allows up to four chains to be connected and you should always use all of these ports. Cable length has little effect on the reliability of the system compared to the number of hops in a chain. Because you have decided your camera placements ahead of time (a wise action in terms of data quality), you may find it hard to cable your setup without creating several custom cables from a spool. One trick is to use cameras near the server as relays and to leapfrog connections along the chain past other cameras in turn. Always try to keep the maximum number of cameras on a chain as low as possible, even if you have to run a longer cable. When you reset the hub, each of the cameras will be detected in the order they are connected. Sometimes, however, cameras at the end of the chain will not be detected (usually in long chains). If this is the case, check for loose cable connections along that chain and hit the reset button on the hub to try again.

4.4 Aiming

Ideally you already have a plan for the general aiming of the cameras. When you are mounting the cameras point them in the general direction you have planned for them then move yourself to the place where you intend to have the camera point, squatting if necessary, and look back at the camera. From here it should be easy to see which way the camera should be adjusted by looking which faces of the camera's housing are exposed. This is much easier than trying to visualize the centerline of the camera when you are standing next to it. Unfortunately, if you re-aim a camera you will have to recalibrate the system before you can get 3D data from it again. To simplify this process, you can set up a marker at the location in question using the wired LED driver and check the cameras new 2D view of the marker using **usbscope** or **scope2d** before recalibrating.

5 Calibrating the Sensor

Now that you have deployed all of the physical elements of the camera system its time to calibrate the system. A walk-through of the calibration process is covered in the **Calibration Wizard** section of the **PhaseSpace Motion Digitizer: Camera System** documentation. The on-screen instructions of **tkcalib** cover the basic setups of what you need to do. However, there are some subtle techniques that you should keep in mind while performing the calibration process.

5.1 Coarse Calibration

As you read in the original documentation, this part of the process gathers individual snapshots, from all of the cameras, of the calibration object in various poses in the space. Beyond just the goal of turning all of the boxes green (the squares at the top of the screen that show calibration status), your intention should be to show the system the space you intend to capture data in. Though it is possible to complete the coarse calibration process without moving the calibration object around the space, only pointing it in different directions, this only gives the system constrains in that small part of the space. Each snapshot is used to bring several cameras into agreement. You should purposively move the calibration object to places in the space your objects are likely to be and face the object in a direction the markers are likely to face from that place. If you constrain the system outside of the space you are interested, a solution that lowers the error at that point, and raises the average area in the rest of the space may be selected. If the process seems stuck, take note of which indicators have not turned green yet and try to include the corresponding cameras in your points. If the indicator never becomes green you should abort the calibration process and check what the camera is seeing with **usbscope** and re-aim that camera. Applying this technique makes the coarse calibration process take longer than it needs to when just lighting the green boxes, but it will improve the quality of the calibration greatly.

5.2 Incremental Calibration

This process is sometimes called the "dance" because of the motions involved. The original documentation has some simple instructions that are not so easy to understand. The basic idea is to wave the calibration wand around the space you are interested in so that system can tweak the calibration a bit to smooth boundary conditions and establish an overall scale for the space. Without worrying about the exact process, take the calibration wand out into the space and begin to twirl it around (don't forget to hit 'capture' first). The location of the markers on the wand is only sampled a handful of times a second during this process, so by keeping the wand in a constant twirling motion it should be easy to get a good distribution of data points. The system needs to see the wand in many different orientations in order to constrain the system properly along the x, y, and z axes. The display on the server will show you where you have been with the wand and you should try to not leave any gaps in the space you are interested in (or stray to far outside of it). If possible, try to get data from all of the cameras for each area of the space by facing the markers toward each of the cameras in turn while you twirl it. The calibration process does not care how graceful you are, it just needs data. That being said, it is always better to capture more data, take a few minutes to be thorough about your coverage of the space.

Important: When you look at the display of your captured motion, if you see any lines of motion that seem to fly out of the space you were working in stop and restart the incremental calibration process again. If these visible outlier points are integrated into the calibration process, the system may choose a solution that creates discontinuities in the space you care about in an effort to reduce error along the outlier points' paths. This, fortunately, is a rare occurrence. If, however, you see it happening often during your "dance" it is a symptom of poor coarse calibration and a better use of your efforts would be to repeat the course calibration process before continuing. Disregard the average error value displayed during calibration, often your eye is a better judge of what is a good calibration than this value.

6 Preparing Tracked Objects

Deploying the camera system was only half of the setup you will have to do to complete this process. The objects you are tracking need to be readied as well. The original documentation, **PhaseSpace Motion Digitizer: LED System**, discusses the purpose and setup of the LEDs, LED drivers and base station. In this document, disregard mention of **.phasespace** directory, the **PhaseSpace Motion Digitizer: Configuration Manager** manual covers how to make equivalent changes using a web interface. Beyond the mechanics of setting up a string of markers and powering on a LED driver, there is some technique to getting the objects you want tracked ready for data capture.

6.1 Marker Placement

Much like when you were planning out the camera configuration, you should make a plan for where the markers will be first, then worry about making sure the cable reaches second. There are three primary concerns with regard to marker placement. First, the markers should be oriented so that they will face the cameras directly. An orientation that will likely be seen by a few cameras well is better than one where more cameras will see it but not as directly (ex: placing markers on the side of an object instead of on the top when the cameras are arranged in a ring). Second, the markers should be placed as far apart as convenient. With a constant level of noise, further spaced markers will yield a better fix on the pose of an object. Finally, avoiding reflections is important. You should not place markers where reflections from other part of the object tracked will be seen by the cameras.

Rigidity: Though not really an issue of the exact placement of the markers, to ensure repeatability of your results make sure your markers are fixed rigidly to the objects to be tracked. Velcro pads can make reattaching markers easy, but a simple strap of tape over the marker (with a hole allowing the LED to be seen, of course) is sufficient to keep the 'give' in the Velcro from becoming an issue.

6.2 Rigid Body Definitions

The system can perform individual point tracking without any special setup, however, before it can extract a rigid body pose you must define the relative positions of markers on the object. The simplest way of doing this is using the program **rbcapture**. This program automatically builds a rigid body definition file (something.rb) from the reported locations of the markers – make sure to use the **-m** and **-o** options. Capture one or more rigid body definitions this way. Next, use **rbedit** to rotate the cluster of markers from its world pose to its ideal pose. If the sensor data was noisy during capture, you can repeat this process until you are happy with the results. Capturing the definitions with the object in different parts of the space will yield slightly different results depending on calibration. Later on, if your object is being reported with the wrong orientation, you can use this tool again to remap the local coordinate frame of the object. The **.rb** file created is actually human-readable text, but it should not be necessary to edit it by hand (unless, on the rare occasion, you have better data on the relative position of the markers than the sensor reports).

7 Viewing Sensor Data

At this point, your deployment of the PhaseSpace system is fully capable of supporting your application. However, before getting into collecting piles of data and analyzing it in detail it is a good idea to try some practice runs of the action you want track. During these practice runs, you can view the data in real time using some of the tools provided. Often you will will be able to make qualitative judgments about the motion of the markers just by watching the data go by and following the lines displayed in 3D. Also, you may notice problem areas with the sensor that overlap with areas you need critical data from. To simply view the live data without recording it there are two very effective methods.

7.1 Paint

The prettier and more intuitive of the methods is to use **paint**. This program is documented in **PhaseSpace Motion Digitizer: Software**, specific options in appendix **A**. For simplest use run the program with no arguments. Or, if you simply want track the pose of a rigid body, use the filename of the rigid body definition as an argument. In almost all cases you should hit the '5x' and 'points' buttons at startup. This will enlarge the displayed marker size (making them easier to see and click on) and start painting trails of the markers location using individual points. Using the 'lines' display mode looks similar but gives you less information. In 'points' mode you can look at the distance between points to compare the relative velocity between two pieces of the markers path where in 'lines' mode a slow and a fast motion would produce similar looking results. By default **paint** runs as a master program (it sets up the primary connection to the server and starts the sensor), but if you want the real-time viewing capabilities of **paint** at the same time as a master

program you write you can use the **-slave** option to allow it to connect as a slave. In this mode it simply displays the objects that were specified by the master program. There are many key commands to use while **paint** is running, the most important of which are 'N' which toggles drawing of the camera visibility cones and 'n' which toggles drawing of labels next to each detected marker to help you identify objects. Hitting 'g' will display a multi-resolution grid around the current location of a selected marker. Zooming in on this grid and looking a the jitter cloud of a marker can give you a good quick estimate of the magnitude of noise for that marker. If you want to clean up grids you have placed hit 'G'. Though not important for visual analysis of the data, the 'Q' to start and 'W' to stop spinning of the world display are nice for running fancy looking demos of the system tracking your objects.

7.2 Logging Tools

The other method is much less pretty but is very detailed in its output. You can use the locally built logging tools to get numeric (x, y, z) coordinates for individual points or (x, y, z, w, a, b, c) pose tuples for the calibration object in real time. The locations of the tools, **logrb**, **log3d**, and **log2d**, are listed in the appendix. By default these tools will try to run as masters by you can run them at the same time as **paint** using the **-slave** option on them. Another default action to override (only as master) is to use the **-f freq** option to set the sampling frequency of the system lower so you can actually read the data as it scrolls by. Of the tree tools you will likely only use **log3d** which gets the 3D location for a individual markers (one or many at a time). It is up to you to interpret this data, however because it is plain text it is easy to work with using standard Unix tools and some locally developed comma separated value processing tools discussed in the appendix.

8 Integrating Sensor into Your Analysis Framework

Now that you can see what data the system is putting out, its time to get it hooked up to be analyzed automatically. Depending on the amount of work you are willing to do and efficiency you need there are a few options for doing this. A common concern with both methods presented here is how to actually interpret the values you get. To start, the length measurements along x, y, and z are in millimeters. If you need them in some different units you can easily convert them yourself. All measurements are relative to the frame you picked during the final alignment phase of calibration. It is your responsibility to align the systems reference frame to fit your interpretation by using the **align** tool or making the necessary adjustments in your own code. There is no guarantee that $z+$ is up, for example.

8.1 CSV Streams

The first method is ideal for rapid application development. Basically, you use the logging tools as a real-time adapter from the sensor to a comma separated value stream. The logging tools print each line of data as soon as it is available and have no buffering enabled. There are a few considerations when processing the data this way. First, because the data is being streamed, you have to process it in order. If you suspect your stream processing application cannot process the data in real time you should consider using the **-f freq** option to have the system run at a slower rate, or have a separate thread read the stream, keeping some shared variables up to date with the latest information. Some of the logging tools may inject lines of metadata starting with a # so take care to filter them out. The CSV stream method allows you to easily process data without having to learn the PhaseSpace API, require linking to the PhaseSpace libraries, or handling many error cases and should probably be considered first when processing your data. Additionally, when working with CSV streams, you can redirect the immediate output of the logging tools into a file and later analyze it for different purposes without recapturing data. These files may be unnecessarily big compared to the PhaseSpace binary format but they are easy to trim, edit, and annotate using your favorite text editor.

8.2 PhaseSpace API

The PhaseSpace API is discussed in detail in the original documentation cleverly titled **PhaseSpace Software API**. When reading this document it is probably best to start with the code examples at the end, **example 1** if you are tracking points and **example 2** if you are tracking rigid bodies. A good starting point for writing your own application using the PhaseSpace API is to modify a copy of source of the logging tool most similar to your application. Using the API directly instead of the CSV streams eliminates a lot of unnecessary text assembly and parsing and allows you specific controls over the marker and rigid body configuration. To capture data in a way that it can be replayed back through your tools for re-analysis look at the locally built **rpdlog** tool. It works with the PhaseSpace binary "raw peak data" format.

8.3 Network Performance

You can think of the server as a glorified appliance used to manage the cameras but its also a capable processing platform. It is easy to have your main application run on your machine and stream data over the network, however, this constantly moves low level data to your application even when it might not be needed. If your application is large, it should be possible to segment the part that works with the sensor out and run it directly on the server. This process can process the sensor data live, and only report back important, possibly higher-level, information back to your main application. This pattern also frees your main application from being tied to the PhaseSpace libraries. Unfortunately the server is less than friendly to this practice. The server lacks a C compiler, several common interpreters, and many shared libraries. If your application can peacefully share the CPU, memory, and hard disk on that system, it may be a smart choice to run your code on the server.

A Electronic Resources at Ames

A.1 Documentation

https://nx.arc.nasa.gov/nx/dsweb/View/Collection-6447	PDFs of vendor supplied documentation
https://nx.arc.nasa.gov/nx/dsweb/View/Document-60165/document.html	login details for current system
https://nx.arc.nasa.gov/nx/dsweb/View/Collection-6441	links, notes, mail

A.2 Locally Built Tools

These tools were developed at Ames to help extract data from the sensor and analyze the result. Some of them are written in Perl and can be run directly while others require running **make** first. They are all located on the **phasespace** CVS repository accessible through **davey** at **/afs/ic-afs.arc.nasa.gov/project/Master**. The PhaseSpace libraries and headers are required for several of the tools, if **make** does not work on the first try look in **/afs/ic-afs.arc.nasa.gov/project/phasespace/**.

csvadapt	experimental adaptive exponential moving average filter
csvdelta	first order different stream processor
csvema	simple exponential moving average filter
csvfir	simple finite impulse response filter
csvpatch	gap filling table preprocessor
csvstat	general purpose statistical analyzer
csvscope	.rid to .csv format converter
detcameras	multi-detector debugging tool
logrb	calibration object rigid body pose logger
log2d	2D per-camera marker location logger
log3d	3D individual marker location logger
rpdlog	command line .rpd recording and playback tool

B Debugging Jumps and Jitter

Its likely that if you are carefully comparing the data from the sensor to the truth about where the markers are and what they did you will find some discrepancies. Unfortunately, additive white Gaussian noise is not a good model for these errors. Most error, however, can be characterized by two specific behaviors.

Jumps: Jumps are relatively large discontinuities in the reported motion of a marker in range of $1cm$ to $10cm$. They are caused by disparities in the modeling of the space resulting from slight calibration errors. Specifically, when set of detectors contributing to the localization of a marker changes, the resulting reported location may change even if the real world location has not. Jumps are most noticeable (and repeatable) when a marker is in motion and moves into or out of the visible space of a detector. If the motion is reversed, the jump should be as well. Most importantly, jumps are *not* random, and if a marker is carefully placed on a boundary between visibility zones it will likely jump between two specific location, spending short periods of time at both locations. Recalibration can reduce the magnitude of the jumps, and reducing the crossing of visibility boundaries by reorienting the motion of the object can reduce the number of jumps. To actually figure out which detector is causing the problem you can, with a test marker fixed rigidly near the location of the jump, cover up all of the detectors in the system one at a time to see which one affects the same jump. However, even with this information there is no general method to fix the situation for sure.

Jitter: The other common behavior is simple positional jitter. Jitter is always present as opposed to jumps which occur at specific locations in the space. Depending on locations of the detectors localizing a marker, tools like **paint** will display lumpy 'clouds' of points around the general location of the marker. Often these 'clouds' will be elongated along specific axes. Because each detector has a fixed angular resolution, how well a detector constrains a marker's location is based on how close it is. Sub-pixel location noise in each of the detectors coupled with a hard thresholding process in the image analysis produces this random jitter along an axis parallel to the detector. Depending on a complicated interaction between the orientations of the cameras this jitter can either cancel or amplify along other axes. When only a few detectors are involved in the localization of a marker the jitter clouds will be very skinny with a string covariance in the error. When more detectors are involved the interaction produces a more spherical, Gaussian-like, distribution. If the clouds have a bi-modal distribution there is likely some small magnitude jumping occurring as well as the simple jitter. Usually there is one offending detector contributing the dirtiest data to the marker localization. As with debugging jumps, cover up the detectors one at a time and watch the effect on the cloud shape to discover the problem detector. If the detector is close to the marker and is still producing bad data it is likely that there is some background interference affecting the detectors view of the marker or that the marker is being viewed from a grazing angle. To reduce the overall magnitude of the jitter, place the cameras closer to the action, so small movements will produce larger angular changes on the detector. Also, this error is fairly easily cleaned up by post processing the data with a low-pass filter. Explicit outlier rejection is not important here because the jitter is fairly localized.

C Quick Analysis Recipes with CSV Tools

Here are a few examples for using the locally built CSV tools to accomplish simple data analysis tasks:

Note: Piping data around in the way described below is not suitable for real-time analysis because many system tools use small buffers and the locally built CSV tools read all of their input before writing output.

```
##
## Building a report of noise for a single marker:
##

# run paint so we can see where the marker is
paint &
# once running, place the marker in a location to be tested
```

```
# capture 5000 data points into a csv file
log3d -slave -only 0 -csv | head -5000 > jitter-data.csv
# -slave allows it to run with paint already started
# -only 0 means sample only marker number id 0
# head -5000 prints the first 5000 lines of data
# > saves the data

# analyze the data and produce a human readable report of the statistics
tail +2 jitter-data.csv | csvpatch | csvstat | tee jitter-stats.txt
# tail +2 passes through the file skipping the first two lines of sensor
#         version and other data
# csvpatch fills holes in the table if the marker could not be seen for
#          some frames
# csvstat analyzes the data, computing mean, standard deviation and
#         covariance between each of the columns of data
# tee displays and saves the report

##
## Computing velocity and acceleration vectors and magnitudes of a marker:
##

# without using paint this time
# capture as much data as you like
log3d -only 0 -csv -f 480 | tee dirtyposition-data.csv
# -f 480 is same as default/max, set your desired frequency here
# tee will save the data until you hit ctrl+c

# remove first few extraneous lines from data file
# and patch holes (if any) in the table
cat dirtyposition-data.csv | tail +2 | csvpatch >  position-data.csv

# compute velocity data
cat position-data.csv | csvdelta > velocity-data.csv
cat position-data.csv | csvdelta -norm > speed-data.csv
# -norm computes the Euclidean distance between position vectors at each
#       step, producing one, positive, value as a result

# compute acceleration data
cat velocity-data.csv | csvdelta > acceleration-data.csv
cat velocity-data.csv | csvdelta -norm > accmagnitude-data.csv

# compute stats
cat velocity-data.csv | csvstat -quick > velocity-means.txt
cat acceleration-data.csv | csvstat -quick > acceleration-means.txt
cat speed-data.csv | csvstat > speed-mean.txt
cat accmagnitude-data.csv | csvstat > accmagnitude-mean.txt
# -quick means skip standard deviation and covariance calculations
# -quick has no affect on single column tables

# Be sure to interpret the results as millimeters per 1/480th of a second.
# *-data.csv files are suitable for viewing in a tool like gnuplot4
# Speed an acceleration magnitude values are less meaningful because jitter
```

```
# and jumps can throw these values off significantly -- real velocity and
# acceleration averages are useful however.

##
## Smoothing over noise to produce cleaner data:
##

# This example uses the data file position-data.csv created in the previous
# example (480Hz sampling frequency).

# smooth with a 0.1s window average
cat position-data.csv | csvfir -boxcar 48 > boxcar-data.csv

# smooth with a half-each-time exponential moving average
cat position-data.csv | csvema -alpha 0.5 > ema-data.csv

# Note: smoothing too much can destroy the features of the data you are
# trying to examine.  Try a few test runs of the data with different filter
# parameters and view the results in gnuplot to make sure you know what's
# going on.

##
## Viewing CSV data:
##

# launch gnuplot, type other commands at the internal prompt
gnuplot

# plot 3D trails (similar to paint) X vs Y vs Z
splot "position-data.csv" with dots

# plot one coordinate value vs time (use value 1, 2, or 3 for x, y, or z)
plot "position-data.csv" using 0:1 using dots
```