Education Associate Final Report

Adam Smith, UC Santa Cruz (amsmith@cs.ucsc.edu)
Sponsor: Terry Fong, Intelligent Robotics Group

Project: Wide Area Motion Tracking with PhaseSpace

June 20 to September 30, 2005

Abstract

This report outlines my experience with the Education Associates program this summer working with the Intelligent Robotics Group. I was tasked with evaluating and improving a wide area indoor motion tracking system made by PhaseSpace for the purpose of tracking humans and robots working together in the framework of the Peer2Peer project. I discuss my *educational experience* in terms of theoretical (ideas, concepts, and algorithms) input and output as well as practical (procedures, techniques, and other technical experience) input and output.

Contents

1 Introduction

This summer I spent a lot of time thinking of systems of input and output, be that a collection of log parsing tools or a system of linear equations minimizing the sum of squared errors while localizing a point in 3D given an arbitrary number of planar constrains. In all of these systems, some information goes into a magic box, something happens, and something different comes out. I think its fitting, given that my mind is still stuck in this mode, to organize my final report on my experience this summer as a mapping of what bits of information came into head and what bits of information (code, documentation, concepts) I put out. Of course, there are other inputs (food from Tuesday IRG lunches) and other outputs (dusty, crumpled balls of painters' tape that I accidentally left in the Moonscape) but I'll try not to emphasize too many of these fun but not-so-educational bits of experience.

2 Theory: Ideas, Concepts, and Algorithms

I've been pretty picky about what I put in this theory section as a lot of what I've dealt with could be considered ideas, concepts or algorithms. I make the distinction here between those ideas that are applicable to many general problems and those that were tied to specific applications.

2.1 Input

My primary mode of *inputting theory* was to find a nice PDF and lean back in the one of twenty or so office chairs in N260-107D and soak in the information throug one of the two giant CRT monitors attached to my work station. Often I'd have to keep a web browser open viewing Wikipedia on the other monitor to help me chase down what some of the unfamiliar methods the paper's author was assuming I already knew. I often began reading certain topics because I thought their application would be able to improve the PhaseSpace system, however, I often followed tangents and found myself reading interesting, educational documents about other topics that really had no application back the problem at hand, but were exciting nonetheless.

2.1.1 Filtering

"Filtering" turns out to be a much bigger topic than I had imagined – especially when I include the topic of "estimation." I was looking to apply a good filter to three specific aspects of the PhaseSpace system. First, I wanted to clean up the noisy photo-detector data being processed by the cameras. Second, I wanted to smooth over noise and reject outlier points in the 3D position of the markers reported over time. Third, I wanted smooth motion for the reported pose (position and rotation) of the rigid bodies. Without listing the countless names and key terms surrounding the filters I learned about here, I can say I did some pretty hefty reading from theory to implementation and analyzing results. I was happy to see that a lot of what I read about looking at signals in the frequency domain and applying certain filters overlapped a lot with my intuitive knowledge coming from my experience playing with electronic music hardware and developing an image codec a year or so back.

2.1.2 Linear Algebra

I had taken a few linear algebra classes in the past, and I can derive the Pythagorean Theorem for any finite dimensional inner product space, but I had yet to actually apply what I knew to a real world application. A lot of the filters and methods for pose estimation I read about were essentially powered by solving a system of linear equations in the end. To really get a feel for why these methods worked I found some good (and by good I mean rigorous) explanations and review of the processes that were used.

2.1.3 Algorithmic Constraints of FPGAs

Being a mostly software-kind-of-guy I usually don't spend too much time thinking about how all of the algorithms I think about actually get run on real hardware. The image filter I was developing for the PhaseSpace cameras was going to have to run on the cameras themselves if it was going to be useful in the real world. It turns out despite their bulk processing power FPGAs have some important limitations that I was not aware of. After learning a bit more about how they operate and looking at a few simple applications I made some major changes to my image filter so that it could still work efficiently in the FPGAs. Most importantly, I organized the processing into a per-pixel pipeline that needed only limited information about nearby pixels and unrolled and broke apart the ugly convolution I had planned into a few simpler parts that when combined together yielded a very similar result with much less work.

2.1.4 SIFT, SLAM, and other STUFF

The rest of the ideas I absorbed were less related to my work but I found myself spending several hours exploring them anyway (often while my lunch was digesting or I was at a roadblock in the main project). SIFT, or scale-invariant feature transform, is a method of extracting recognizable features from an image so that they can be recognized in other images even if the camera position has changed or the image is of a different size. SLAM, simultaneous localization and mapping, is the term for the problem (often in robotics) of building a map of a new environment while at the same time trying to figure out where you are in that map. I read several articles relating these techniques together and several considering them on their own. I know some of these ideas are already implemented on the current rovers and it was exciting too see what progress others are making with robots rolling around their lab, exploring offices and recognizing circular hallways.

2.2 Output

The process of *outputting theory* was a little bit more involved than the input process. My two somewhat-theoretical contributions were presented at the PhaseSpace office. For both of these methods I communicated the ideas in ad-hoc scribbling on either whiteboards or graph paper using generous hand-waving. There was plenty of example code, test data, and analysis graphs to back it up, but I hadn't written up the idea itself in any formal way.

2.2.1 Efficient two stage image filter for consistent peak detection

One of the problems I encountered while analyzing the PhaseSpace system was that slight imperfections in the optics of the cameras were causing clusters of spikes to appear on the cameras' photo-detector instead of smooth lumps. Depending on the exact distribution and height of the spikes the location of a marker reported by the camera could jump around even when the marker had not actually moved. This was creating overall jitter in the system that was very noticeable at the edge of the field of view of many cameras. I came up with an image filtering method, that should have an efficient implementation on the camera's FPGAs that turns the spiked input from the detectors into a smoother, marker-likelihood graph. Instead of looking for a certain shape using the matched-filter techniques I had originally planned to use, I experimented with a bank of filters, each looking for certain features of good peak shapes and combining their results in a second stage filter. The input spikes proved too unpredictable so I settled on a simpler solution that simply applied a high-pass filter followed by a low-pass filter on the magnitude of the output of the first filter. In simpler words, it looks for clusters of strong high frequency noise. I found this method to yield significantly more consistent results in the face of shifting spike patterns that did any of the other methods I tried.

2.2.2 Constraint weighting for graceful boundary conditions

Another major problem with the way the PhaseSpace system was implemented is that it trusts the input from each of its (usually) sixteen cameras equally whenever they report data. In this setup, a known noisy camera, a far away camera, a camera with intermittent view of the a marker, and close, clean, camera with a consistent view of a marker all contribute equal information to the localization of the marker in 3D. The method I proposed applies a weight to the constrains imposed by the information from each camera based on how much that camera is to be trusted given its position and history. Initially, simple factors including where the marker was in the camera's field of view and approximate distance from the camera would be used to weight-down constraints. Even with just these adjustments we would eliminate the harsh jumps that appear in the reported location of the marker when the marker moves out of the field of view of one camera or into the field of view of another one. In-place occlusions could still cause certain constrains to flicker in and out with this setup even when the marker is in a good position relative to the camera. To combat this case I proposed tracking how long a marker has been visible by a camera and fade in the weight of the constraint applied by that camera until the camera is trusted, and if the camera looses sight of a marker, trust the camera's last known constraint a little further in time while its weight fades out. This method cannot make up for errors in calibration or camera modeling, but it can turn the effect of these errors into smooth sliding or warping of the reported space rather than harsh position jumps.

3 Practice: Procedures, Techniques, and Other Technical Experience

3.1 Input

Practical input took place in several ways, though most were variations on "reading the manual" – something I used to avoid when I was younger.

3.1.1 PhaseSpace

I got a lot of hands-on experience working with the PhaseSpace system hardware. This was hands-on in the sense that I edited configuration files and source code and also in the sense that I got up on a step-ladder and added new camera mounts with a screw driver. I had the system setup process memorized by the end and managed to document a few helpful insights about it in my how-to. Beyond what was covered in the manual I learned some tricks and techniques from our local Anne and some of the guys from PhaseSpace for marker preparation, calibration, cabling, aiming, etc. I also learned the procedures that had been used to evaluate the system before the summer and got familiar with the resulting data. In some informal conversations with the engineers at PhaseSpace I learned a lot about the internal workings of and a some of the design choices made in developing their system.

3.1.2 Visualeyez

Visualeyez, the *other* optical motion tracking system I worked with during the summer, had its own setup process, special considerations for preparing subjects to be tracked, and calibration. I also got familiar with the test procedures and results that Dan had come up with when the system first arrived. Much of what I learned about this system went into making my test procedures for the PhaseSpace more fair, and not overly-specific to the design of the system.

3.1.3 ICE

ICE, or Internet Communications Engine, is the middle-ware solution chosen to link together most of the low level components in the P2P project. This framework was all new to me (although I was familiar with the idea of what it did beforehand). In order to accomplish what I did I had to learn to write both client and server ends of an ICE application as well as help decide how an indirect proxy server work function to allow access to data from both PhaseSpace and Visualeyez at the same time.

3.1.4 GNU Make

Though I thought I was pretty good with it already, I learned a lot more about GNU Make this summer. I was comfortable writing Makefiles that worked in different environments but looking back they were unnecessarily redundant and often took a lot of typing just to get the default action to be performed. This summer, I learned a lot of new techniques for writing Makefiles with GNU Make (most notably implicit rules) and now think of it more as of a logic programming engine than a build system. I used Makefiles to automate a lot of my testing and analysis – managing the whole process from data collection to including generated graphs in the final analysis.

3.1.5 Inline::C

Nagging Systems Group for a week to get this installed turned out to be a great time-saver for me. Inline::C is a module for Perl that lets the programmer call C functions from inside a Perl script without having to go through the process of manually creating interface files. To have low-level control of the image filter I was writing I needed to be coding in C. In development, I went through the edit-compile-link-run cycle many times to get the filter doing what I wanted. Using Inline::C, this reduced to a simple edit-run cycle and all of the dirty work was taken care of for me. All of my testing an analysis code was written in Perl where it was easy to modify to test different applications.

3.1.6 Dynamic Linking

Most of the software in the world starts its life as source code, gets compiled into object files, gets linked into an executable, then gets loaded into memory by the operating system and run. In classes, I had written a compiler and an executable loader, but I never really got hands on experience with what happens at the linking stage. This summer I took some time to figure out what really happens, especially regarding dynamic linking. The Inline::C programs I wrote generated little dynamic-link shared libraries when they were run for the first time. Using these and my new found knowledge of dynamic linking I was able to speed up my testing process by writing code that simply linked against some libraries I was building at runtime rather than having to be recompiled each time, slowing down my development cycle.

3.1.7 Octave/Gnuplot

I am an only occasional user of Mathematica, so when it came time to do some serious analysis I looked, instead, use whatever tools I had on the local system to help me out. Octave, a free GNU clone of Matlab, took care of a lot of number crunching for me. Paired with Gnuplot (which was also new to me) I had a powerful way to view and manipulate the multi-dimensional data that was coming to me from the motion trackers. These tools served me well for doing interactive computation as well as being very useful in automating my test procedures. My Makefiles managed hooking up Octave, Gnuplot, Perl and LaTeXall in the right way to give me nice reports of the tests I ran with a single command line.

3.2 Output

Practical output is simply the results of all the actions I took, though most of it was embodied in a handful of programs and documents that I wrote.

3.2.1 Data Analysis Tools

Mostly as a side-effect to all of the data analyzed, I ended up creating several generic tools to help me automate my work. Instead of creating a monolithic analysis engine, I decided to structure my tools as filters, little building blocks which I could assemble together in different ways when the need arose. The most general of these tools were my comma separated value parsing tools. The program "csvstat" read a bunch of numbers in CSV format from standard input, crunched them, and wrote interesting statistics to standard out. Any time I reported a mean, standard deviation, covariance, or correlation in a report those numbers came from "csvstat." Another tool, "csvdelta," computed row-wise differences for a stream of input and wrote it to standard out – instantly transforming position data into velocity data, or whatever the application demanded. I had an option for this program that would compute Euclidean distances between successive vectors producing a single column result for any number of dimensions. Finally, "csvpatch" was a utility for filling in missing values in a table. Many of the data files I worked with had holes when markers were discovered and lost over time.

3.2.2 Data Extraction Tools

The analysis tools are no good unless they have some data to operate on. To get data out of the PhaseSpace system and into my analysis pipeline, I wrote several extraction tools. The logging tools ("log2d," "log3d," "logrb" and "csvscope") hooked into the PhaseSpace system using its C interface and printed out the data it got to standard out where I could route and manipulate it with the standard UNIX tools. My tools weren't limited to PhaseSpace though, I had an "extract-marker" tool for Visualeyez that would convert the sensor's text dump format into something parse-able by the rest of my tools.

3.2.3 Data Filtering Tools

In the tradition of the simple UNIX filter programs that I had been writing, I wrote more tools to test out my signal filtering ideas on live data. The simplest, "csvfir" and "csvema" applied a finite impulse response and exponential moving average, respectively, to data they read in. These tools proved more useful for testing in general than they did for improving the motion tracking systems at all. Another tool, "csvadaptema" was supposed to be a reactive, adaptive EMA-based filter, but it was never really very successful – an idea that I had wanted to experiment with nonetheless.

The two (one-dimensional) image filters I produced (using Inline::C) were “cfilt-bandpass” and “cfilt-treble.” The bandpass filter did was it’s name says, looking for medium frequency humps and calling them a marker. It turned out not to perform as well as I had planned because many other light sources would have the same characteristic hump that the PhaseSpace markers did. The “treble” filter, counter-intuitively, works by “looking for noise” and calling clumps of it a marker. None of the background interference sources I encountered were sharp enough points to create high-frequencies in the photo-detector signal, so looking for these seemed to single out the markers from the rest better. I had some simple Perl code wrapped around these filters to see their response to single frames of data at a time, but to get a good evaluation of them I needed to apply them to whole streams of data recorded from the detectors. I created another tool, “ridfilter,” to filter the data in the raw-image-data files. This tool linked against the C parts of my filters at runtime and thus didn’t require recompilation each time I changed a piece of the filter.

3.2.4 PhaseSpace Test Procedures

The first big document I produced, “PhaseSpace Motion Digitizer Test Procedures” described a process for evaluating any changes to the system in the future. I went beyond the procedures that had already been followed to test the sensor and created several new processes and documented how to use the automated analysis system. I made a point to declare all of the assumptions made by each of the tests in their descriptions. In creating these I realized that I needed to add more options to my data analysis tools. After (and during) creating this document I went through several test runs to, first, make sure my procedure documentation made sense, and second, to collect data about the system so it can be compared to future iterations of the system. This was the first major document I had written using LaTeXand I had to get familiar with a few more commands before finishing it.

3.2.5 PhaseSpace How-to

The other large document I produced was “How-To: Motion Tracking with PhaseSpace.” It was a high-level walk-through for using the motion tracking system. It was intended to compliment the documentation that came with the system, adding tips and techniques I had figured out from my own experience with the system. In it, I recorded a dump of the important bits of knowledge I had discovered this summer. One of the bigger contributions that I talked about in the how-to was the the idea of using the logging tools I had created as a sort of text API. Being able to record, manipulate, and analyze data from the sensor without having to write any software – just composing UNIX filter programs together – greatly reduces the effort required to use the system. I think for future use of the PhaseSpace system at Ames this method will be the smarter choice (especially in terms of researcher labor). I even dedicated a section of the appendix to code (shell) examples using this method. It is my hope that the PhaseSpace documentation included with the system will improve over time, but until then hopefully my document can help fill in the details.

3.2.6 System Evaluations

In a less formal manner (text files and email) I submitted evaluations of the two motion tracking systems. For the PhaseSpace system, I reported interesting statistics for different types of sub-regions of the space tracked by the system. Also, I tried to analyze the causes of the inaccuracies I discovered and came up with an interesting model. I compared the space reported by the system to a fractured grid on paper. In this analogy (which is best explained by an image I can't find anymore), measurements given by the system are locally consistent until certain boundaries are crossed where large discontinuities are observed. What I really claim is that there is a very complex six-dimensional function warping the space based on the position an calibration of all of the cameras. In terms of precision the PhaseSpace system was more than sufficient for our needs but the large scale discontinuities destroyed the accuracy of the system. The Visualeyez system was much more well behaved, though my analysis reported that the space it covered was much less than that of PhaseSpace and it was much more susceptible to occlusion problems because only two viewpoints were involved. Without a better knowledge of what was going on inside the Visualeyez system I was only able to draw simple conclusions.

3.2.7 Demonstrations

The least demanding of my duties, although probably the most visible, was getting the motion tracking equipment ready and showing it up during demonstrations. For this I had to make sure that the cameras were well distributed around the space so that a mobile mob of interested audience members wouldn't block the path of the light from the markers to the cameras. I made sure markers were securely attached (using velcro and painters tape) to whichever rovers we happened to be showing off that day. And finally, I made sure the robots left pretty colored trails on the screen when they drove around. K10, in its best handwriting, was programmed to spell out its name in the marker trails in the PhaseSpace viewer.

4 Conclusion

This summer was, indeed, a great learning experience but it also changed the way I think. In retrospect, I think the best preparation for this job I had came from taking natural sciences classes in school. Understanding the scientific method helped me organize my work more than having a lot of programming experience. After all, a lot of what I learned this summer would have been basic background knowledge for an electrical engineer, not new material for a computer scientist (in training) like me. Since I've started grad school this year I've been reading papers with different questions in my mind than I used to. Beyond looking to see what cool new idea they have, I look to make sure their claims are supported by their evidence and that their tests measure something that is relevant to the the problem they propose to solve. Knowledge, curiosity, and rationality are some of my most important values. My experience this summer enriched all of these.