Tracking with SWAMIS

Derek A. Lamb

7th June 2005

Chapter 1

Preliminaries

1.1 Introduction

If you have decided you want to track features in images, then you have come to the right place. SWAMIS (the Southwest Automatic Magnetic Interpretation Suite) is a freely available and documented feature tracking code. SWAMIS was designed to tracking moving magnetic features on the Sun, but there's no reason it can't be used for other applications. But because it was designed to track magnetic features, it can handle positive-valued and negative-valued features in your image sequence. Of course, if you only have positive values, then SWAMIS will only have to do half the work.

SWAMIS is not a stand-alone program into which you can feed in your image sequence and pop out results. Instead, it is a suite of Perl Data Language (PDL) programs run in sequence. So you will need to have PDL running on your computer. You might already have it–it now comes in the SolarSoft distribution tree, and some Linux distributions are coming with it installed. Indeed, the Mandrake 10.1 I just installed has this available to install just like any other package, though I found this out after I had already installed. If you already have PDL, then you can skip 1.3. If you already have PGPLOT installed, which is likely if you have done any scientific computing or plotting, you can also skip 1.2 and go straight to 1.4. This Guide only assumes that you have a working Linux distribution of some kind (access to the installation media may be necessary) and a working internet connection. If you want to install onto Windows using Cygwin, go ahead, but I don't know anything about it. Your experience with installation may be different–this is just an edited account of my experience installing PDL onto a new system, and hopefully it will be helpful to you.

1.2 Installing PGPLOT

When I performed this installation, I did Section 1.3 before Section 1.2, but the PDL installation might have gone smoother had PGPLOT been up and running, so I recommend doing it in the order presented here. PGPLOT is a standalone library, so the order will not affect it adversely.

Currently, SWAMIS diagnostics and visual monitoring is accomplished through the powerful PGPLOT graphics package. It would be extremely useful to have this installed. Future versions of SWAMIS might support the newer PLPLOT, but this is not a priority at the moment. Before you try to download this, make sure that you don't already have PGPLOT installed

```
ls /usr/local/pgplot
```

If it lists some files, then you already have it in the normal installation location. If it says "No such file or directory," then you need to get PGPLOT. To do this, go to the "External Libraries" link on the PDL website `http://pdl.perl.org` or go straight to the PGPLOT website `http://www.astro.caltech.edu/~tjp/pgplot/`. Download the file pgplot5.2.tar.gz to /usr/local/build. Take a second to do

```
which g77
which gcc
```

to make sure you have the correct compilers. For some reason my Mandrake 10.1 distribution didn't come with f77 or g77, which makes it a bit hard to compile PGPLOT, as a lot of it is written in Fortran. It's probably better to have g77 than f77 these days. I got the make file made, but after I said "make", it complained about not having a rule to make png.h. Then I found this PGPLOT installation script `http://www.boulder.swri.edu/swamis/PGPLOT_install` that we are redistributing here courtesy of Matt Newville at Argonne National Laboratory. It has a "kludge" in it that fixes the png driver. You can read about it in the PGPLOT readme file `http://www.boulder.swri.edu/swamis/README.PGPLOT`. So I downloaded the script, made it executable, and ran it:

```
chmod 774 PGPLOT_install
./PGPLOT_install --system=linux --with-png --with-gif
```

Note: you may or may not need a license from Unisys to use the GIF format. Since mine is for non-commercial educational use, I am assuming that a license will not be required. At any rate, the PNG support should be good enough. The script did its thing, and finished with the message "PGPLOT appears to be correctly installed. Good!" Warm fuzzies for everyone. Now you need to set some environment variables. I opened up my .bashrc file and added:

```
export PGPLOT_DIR=/usr/local/pgplot
export PGPLOT_DEV=/xserve
```

I created a .Xdefaults file in my home directory and added:

```
pgxwin.server.visible: false
```

This will take effect when you log out and log back in to your windowing system.

1.3 Installing PDL

First, make sure you have Perl. If you have any recent distribution worth it's salt, you will have this. To make sure, at the command line type:

```
which perl
```

If it says something in the affirmative like "/usr/bin/perl", you're in luck. If not, you either 1)don't have it installed, or 2)it's installed somewhere terrible that isn't in your path. Say

```
locate perl |less
```

to get some ideas about this. If it's not installed, use your local package manager or go to www.perl.org and follow the instructions there.

Second, make sure you have a recent version of perl. Say:

```
perl -v
```

And hopefully it will print you a cheery message saying that it is Perl version 5.8.x (where x is some number) or higher. If it says 5.6 or smaller, you probably want to upgrade, as support for 5.6 is deprecated in the newest version (2.4.2) of PDL and will probably disappear entirely in future versions.

At this point I assume you have perl installed. Now, go to pdl.perl.org and download PDL. I recommend getting the latest Stable Release, because then if by chance something goes wrong you'll know it's you, not some unstable code you got. You'll probably end up with a file called PDL-2.4.2.tar.gz in your home or /tmp directory. At this point I make myself root and move the thing to /usr/local/build (make the directory if you need to). Unzip and unpack it with

```
gunzip PDL-2.4.2.tar.gz;
tar -xvf PDL-2.4.2.tar;
```

Now you have a directory called /usr/local/build/PDL-2.4.2. This is good. A little bit of looking at the INSTALL file tells you some important information, including how to manipulate the installation configuration file (perldl.conf). In particular, I always go and remove the # from in front of WITH_BADVAL =>1 and put it in front of WITH_BADVAL =>0. This ensures you have badvalue support, which isn't critical for SWAMIS, but is a neat feature that is in general helpful to have (you can see the documentation about it later if you want). Now, to install it:

```
cpan install PDL
```

or if that doesn't work, as happened to me, use

```
perl -MCPAN -e shell;
cpan> install PDL
```

If it's your first time using CPAN you'll get a question asking if you want to configure it. My advice is to say yes, accept all the defaults until you get to the one that says something about "should I follow prerequisites?" or "should I get all the dependencies" (I'm not sure of the exact wording), and change it from "ask me first" to "yes, please do." We're relying on the software to do the Right Thing here, with the added benefit that you can take a break and get a sandwich without having to worry that the installation script is just sitting there doing nothing waiting for you to press a button. The script will go out and get a bunch of packages that will Enhance Your PDL Experience and generally make the thing usable. If you get some errors first consult the end of the INSTALL file to see if your error is there. It probably isn't. All these errors are submitted by space aliens who are using computers 3000 times as advanced as yours and they are all configured funnily so that nothing is actually supposed to work without a lot of tweaking. This supposedly makes them superior.

When I did this, CPAN gave me an error, something to the effect of "Installation seems impossible." Don't worry, it's not really impossible, the script just didn't get a lot of positive reinforcement growing up. You're probably just missing some libraries. I scrolled up and found about the most useful information I could:

> make[2]: *** No rule to make target
> '/usr/lib/perl5/5.8.5/i386-linux-thread-multi/CORE/EXTERN.h', needed
> by '_01syntax_t_6b3a.o'. Stop.
>
> make[2]: Leaving directory
> '/root/.cpan/build/Inline-0.44/C/_Inline_test/build/_01syntax_t_6b3a'
>
> A problem was encountered while attempting to compile and install your
> Inline C code. The command that failed was:
>
> make > out.make 2>&1
>
> The build directory was:
> /root/.cpan/build/Inline-0.44/C/_Inline_test/build/_01syntax_t_6b3a
>
> To debug the problem, cd to the build directory, and inspect the output
> files.

So I went to the directory it told me to go to, and basically found the same information in the output files as was printed above. Having determined that the "No rule to make target EXTERN.h" seemed to be the first error message, and that I did not have a file by that name anywhere on my system, I decided to see what Google said. I found that Mandrake 10.1 was missing the perl-devel RPM, so I downloaded it (and its dependencies) and installed it. Went back to the CPAN prompt and reran "install PDL", then it crunched for awhile and said it couldn't find the file glu.h. So I went to Mandrake's

rpm finder and found the packages with glu.h and installed them, then reran "install PDL". Lather, rinse, and repeat untill your system has all the packages it needs for a complete build. It might be a good ideal to get GSL, the Gnu Scientific Library, if you don't already have it. Get the GSL-devel package too, while you're there.

Now PDL is installed, and it will work. You can type "perldl" at the command line and get into the shell. But you are going to need some more things to get it to work nicely. Go back to your CPAN and type:

```
install Parse::RecDescent
install Text::Balanced
install Inline
install ExtUtils::F77
install Astro::FITS::Header
install Convert::UU
install PGPLOT
install Term::ReadKey
install Term::ReadLine
install Term::ReadLine::Perl
install Time::Timezone
install Time::ParseDate
install Time::JulianDay
install Time::CTime
install Time::DaysInMonth
```

You should be able to replace Term::ReadLine::Perl with Term::Readline::Gnu, though I'm not sure what the practical difference is. By this point most of the packages should say "*** is up to date," but this is a good last-minute check. As a final step, get the sample PDL startup file `http://www.boulder.swri.edu/swamis/sample-perldlrc` and copy it to your home directory as .perldlrc. You can edit the file however you like. I had to comment out the use PDL::GSL lines, but I'm not sure why yet.

1.4 Installing SWAMIS

Now you need to get Swamis itself. This is the last step, and then you're up and running. Go back to the SWAMIS website `http://www.boulder.swri.edu/swamis` and download the SWAMIS archive. It might be useful to make a "pdl" directory in your home directory for holding your programs. Unzip and unpack the archive there, and you're ready to go. Now wasn't that easy?

Chapter 2

Tracking

2.1 Tracking Methods, or How SWAMIS Works

This is slightly modified copy from a forthcoming ApJ article by DeForest *et al*.

2.1.1 Detection

SWAMIS uses a two-threshold signed discriminator with hysteresis in space and time. Pixels are subjected to the high threshold unless they are adjacent to any already- detected pixel, in which case they are subjected to the low threshold. The discrimination is iterated, dilating detected regions by comparison of their boundaries to the lower threshold, until no more points are added to the detected locus.

The high threshold is typically set to about 6 σ of the measured noise, and the low threshold is typically 2-3 σ. The discriminator keeps track of sign, yielding masks that are either 0, 1, or -1 depending on whether each feature is positive or negative. The threshold is specified to the code in absolute data value units: noise characteristics are determined in advance.

The neighborhood of a particular pixel in a dataset comprises up to 26 pixels in a 3×3 x,y,t cube, and SWAMIS can be switched to include different sorts of neighbors. Typically one does not consider the eight pixels that comprise the corners of the cube and that only share a single corner with the central pixel. The low threshold applies only to neighbors with the same sign as the adjacent region. The code can be made to allow neighbors only within the same frame as the test pixel.

SWAMIS uses recursion to extend features as far as possible into an output detection mask. First, all pixels above the high threshold are marked in the output mask, then multiple passes are made through each image. At each iteration, all pixels adjacent to newly-detected pixels are tested against the low threshold. Because neighbors are allowed on the time axis as well as the spatial axes, a recursive algorithm is used to step back and forth through the data as new points are detected (requiring points in the previous frame to be checked against the lower threshold). There is a user-settable limit on how far back this "look-behind" can traverse.

2.1.2 Identification

SWAMIS contains two identification methods: "clump" and "downhill". The clump method groups any adjacent same-sign pixels into one feature with one identification number. The downhill method finds local maxima in the absolute data and assigns each maximum a unique label number. Label value then flows from each maximum toward zero until it either reaches the end of the feature or reaches a valley. In the event that two or more identification numbers meet in a "valley", any pixel that could go to either is arbitrarily assigned to the first number to reach that pixel.

The clump identifier also begins by seeding each local maximum with a unique label, but allows label value to flow "uphill", so that contiguous blocks of detected pixels all end up carrying the lowest numerical label assigned to any of the local maxima within each block. This step produces a list of images, each of which contains features marked by integer labels that are unique with the frame. Positive features get positive labels; negative features get negative labels.

2.1.3 Association

SWAMIS scans through the identification frames and identifies intersections of same-sign features between each frame n and the successive frame n+1 For each feature f in n , a "forward intersection list" of intersecting features in n+1 is assembled. For the largest of those intersecting features, f'_{max}, a "reverse intersection list" of features in n is assembled. f' is assigned the same label as f if and only if the largest reverse intersector of f'_{max} is f. After all such features have been found and associated, the remaining unassociated features are presumed to be new and are assigned new globally unique labels. The output of this step is a set of images marked with globally unique labels that are persistent across frames.

2.1.4 Tabulation

SWAMIS separates the acts of associating features and tabulating feature histories into two separate passes. In the tabulation pass, SWAMIS calculates the flux, size, and x & y center-of-flux of each feature for each frame. Features with lifetimes less than 4 frames, maximum sizes less than 4 pixels, or x,y,t volumes less than 8 pixels are discarded (though the user is able to set these options). To avoid the swiss cheese problem, such discarded features are re-associated: if a discarded feature is touching another non-discarded feature of the same sign then all of its pixels are reassigned to the non-discarded feature. The output of this step is a table with feature ID running across the top and time step running vertically, containing, the location and size information for the feature for each frame. The associated-label images can be touched-up in place to save disk space.

2.1.5 Event Classification

SWAMIS' final pass through the data is an attempt to extract histories for all features. We determine how each feature is born and dies. The possible birth methods are:

emergence, appearance, and fragmentation. An emergence occurs if there is nearby opposing flux that also emerged within a settable window of time, or a nearby feature of opposite sign that increases in flux. An appearance is an origin with no nearby flux of either sign, or if nearby concentrations change in a way that does not conserve flux. If an origin coincides with a nearby (or reverse-intersecting) feature of the same sign, and the nearby feature loses flux, then it is classified as a fragmentation.

Conservation of flux is enforced only approximately, to allow for noise and the difficulty of calibrating total flux. The "slop" on delta-flux is settable by the user for each of these types of event.

The death events of submergence (or cancellation), disappearance, and merging, respectively, are the birth events run backwards in time. An additional event type, complex, is present for multiple-feature interactions, and an error type is present for those events that do not conserve flux.

2.2 Pipeline

Get the sample tracking pipeline `http://www.boulder.swri.edu/swamis/pipeline.pdl`. This is how we use it, but you're free to experiment. "00-data" is the directory that contains all your images. "01-mask", "02-id", "03-assoc", and "04-tab" are directories that contain images of the intermediate steps. My advice is to copy and paste the code directly into the command line so you know what is happening with each command. If you want to see the documentation for a particular program, you can type at the command line

```
perldl> ?frag_detect
```

or whatever the program name happens to be.

Chapter 3

Final Comments

Please pass on questions or suggestions for this document to Derek Lamb `mailto:derek@boulder.swri.edu`. Questions about SWAMIS operation can be directed either to Derek or to the main SWAMIS author, Craig DeForest `mailto:deforest@boulder.swri.edu`.