Multi-Sun Centroiding

Jeren Suzuki

Last Edited August 28, 2013

Contents

Table 1. Final data structure of solar region

Name	Type	Value	Notes
XPOS	FLOAT	210.522	Rough calculation using a simple masking method
YPOS	FLOAT	166.702	”
REG	INT	1	Region ID #: 1 is 100%, 2 is 50%, 3 is 25%
THRESH	FLOAT	106.000	Threshold calculated from sorting array and taking derivatives. Used in both finding rough X-Y center as well as the threshold for limb-fitting.
PARTIAL	FLOAT	0.	Flag that determines if the solar region is cut off on the edge or not. 0 means that it is not cut off
XSTRIPS	STRUCTURE	-> WHOLEXSTRIPS Array[5]	Strucutre containing the strips of whole solar data bound by a cropped region chosen by XPOS and YPOS
YSTRIPS	STRUCTURE	-> WHOLEYSTRIPS Array[5]	”
LIMBXSTRIPS	STRUCTURE	-> LIMBXSTRIPS Array[5]	LIMBSTRIPS contains a pair of arrays, ENDPOINTS and STARTPOINTS that mark the limbs of each strip of data from X/YSTRIPS
LIMBYSTRIPS	STRUCTURE	-> LIMBYSTRIPS Array[5]	”
LIMBXPOS	FLOAT	210.710	Center calculated from LIMBXSTRIPS
LIMBYPOS	FLOAT	167.172	”

1 Introduction

`beta.pro` loads an image with any combination from a list of suns below and returns the center positions of the whole suns only. The only user-interaction is the selection of which region to analyze. The rest of the process is automatic.

- R1 & R2
- R2 & R3
- R1 & R3
- R1 & R2 & R3
- R1 & partial R2
- R1 & partial R3
- R2 & partial R3
- partial R1 & R2
- partial R1 & R3
- R1 & partial R2 & R3
- partial R1 & R2 & R3
- R1 & partial R2 & partial R3
- partial R1 & partial R2 & R3
- partial R1 & R2 & partial R3

Table 1 breaks down the resulting strucutre of the centroiding program:

2 Centroiding a Solar Image

The current method to find the centers of any solar image is the following:

1. Load Image
2. Read parameters from pblock.txt
3. Sort image and cut off top .1% of pixels (top 1% was actually too much)
4. Smooth, take deriv, smooth again, take deriv again of sorted array, find peaks that correspond to difference solar regions and their thresholds

5. Mask image above thresholds to find centers of every shape, regardless of partial or not
6. Scan border of image looking for consecutive pixels above lowest threshold
7. If more than 5 pixels in a row, marks x and y position and tags nearest solar center as “partial”
8. Crop remaining whole suns
9. Extract 5 strips centered around cropped solar center for both X and Y direction
10. Extract a pair of limb strips for each long strip
11. Fit 2D polynomial to limb profile
12. Mark position where fitted polynomial crosses threshold
13. Use positions to find chord lengths of sun
14. Average midpoints of chords to find limb-fitted centers
15. Using the limb-fitted centers, recrop the image so that every pixel in the image is part of the solar disk
16. Analyze the cropped image for fiducials
17. Using the fiducial positions, we compare the solar positions we calculated to a position defined by the physical setup.
18. Not directly in the workflow, but we also use the solar centers to find the rotation angle/orientation of the setup. This kind of calculation is increasingly worse with less whole solar regions
19. HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM

3 Issues

The current method of finding thresholds in the image is robust, yet slow. The problem doesn’t lie in the second derivative, but the smoothing of the data so the derivaties return unusable data. As of now, the best smoothing filter is also the slowest, `ts_smooth()`. Figure **??** shows the difference that `ts_smooth()` makes. smoothedarr is the sorted array soothed with `ts_smooth()`. The arrays look a *LOT* worse if we use `smooth(sortedarray)` instead of `ts_smooth(sortedarray)` (See Figure 1).

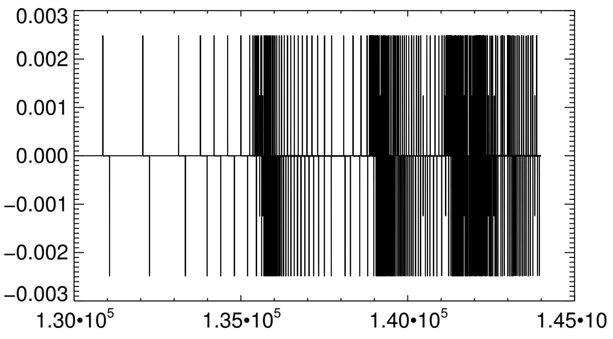
We need a faster way to find the peaks, either through an alternative smoothing method or a new peak-finding method.

The current method of associating a region number to a sun is using IDL’s `label_region` and `histogram`. In the future we are going to port over the code from IDL to C++ so I shouldn’t rely so heavily on complicated built-in IDL functions. As of now, however, the code is fully functional and further development will be halted in favor of working on other parts of the project.

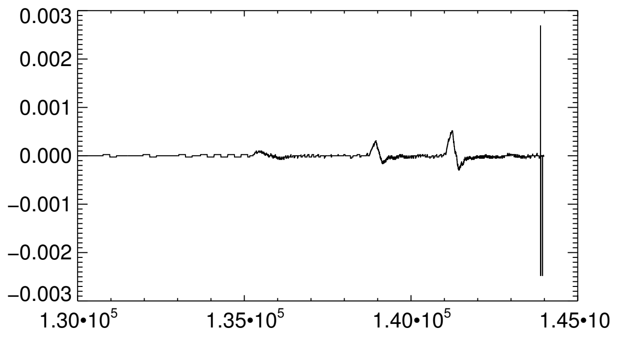
Furthermore, we need to figure out the distinguishing factor that identifies if there is some “global” darkening that affects the entire image. The reason is that if we have one image of a 100% and 50% sun in some shade that results in an image that *looks* like a 50% and 25% sun, how do we tell the difference?

We have yet to investigate this issue, but there is a possibility that a fiducial may lie on the edge of the solar limb where we are taking strips of data and muck up our polynomial fitting. We have not seen how much of an effect it would have or if it should be something we are worried about at all. One possible solution is if the program notices a fiducial when trying to fit a polynomial to the solar limb, that it look at the next column/row over until the fiducial doesn’t interfere with the data.

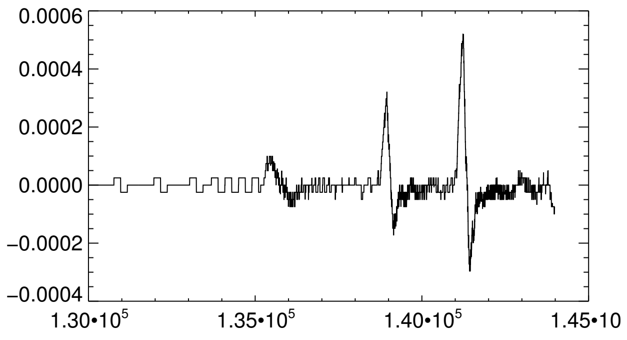
Currently we are using 5 chords, separated by 5 pixels apart, these are rather arbitrary numbers, perhaps I should do a case study that looks at the optimal numbers to use. Actually, a case study of all the parameters I use, but since we’re halting development on the code, this may not happen in the near future.



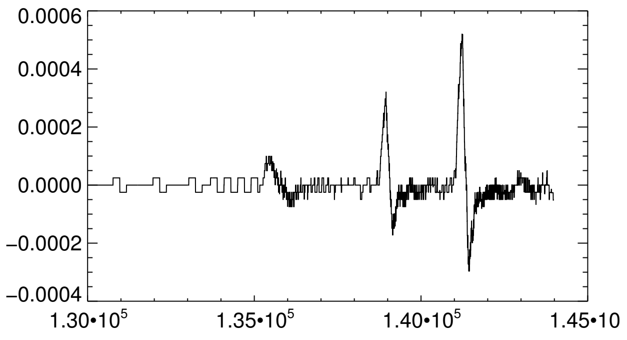
(a) `deriv(deriv(smoothedarr))`



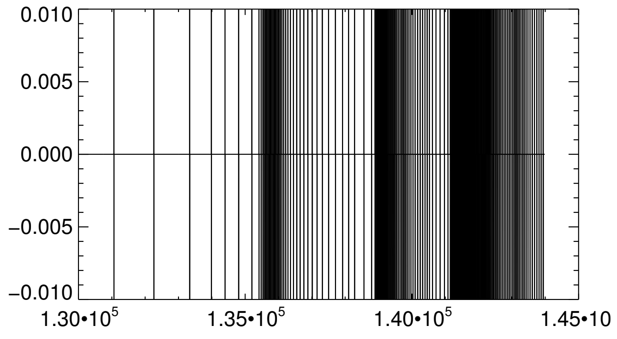
(b) `deriv(smooth(deriv(smoothedarr)))`



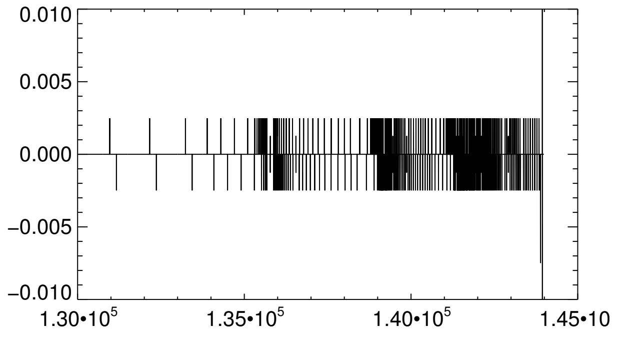
(c) `deriv(smooth(deriv(smoothedarr),/edge_wrap))`



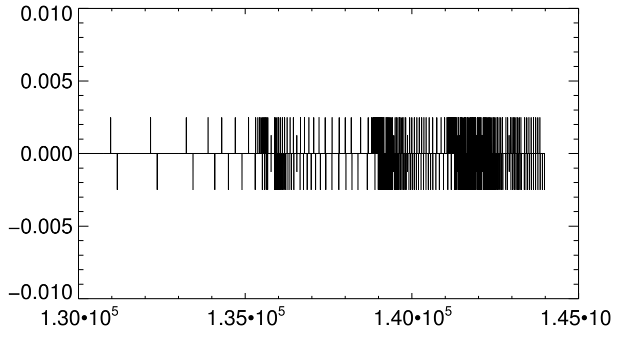
(d) `deriv(ts_smooth(deriv(smoothedarr)))`



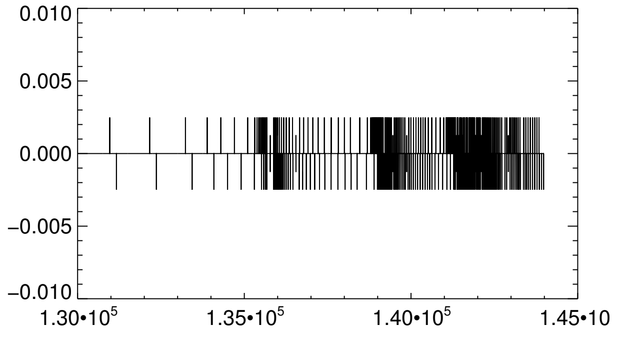
(e) `deriv(deriv(smooth(sortedarray)))`



(f) `deriv(smooth(deriv(smooth(sortedarray))))`

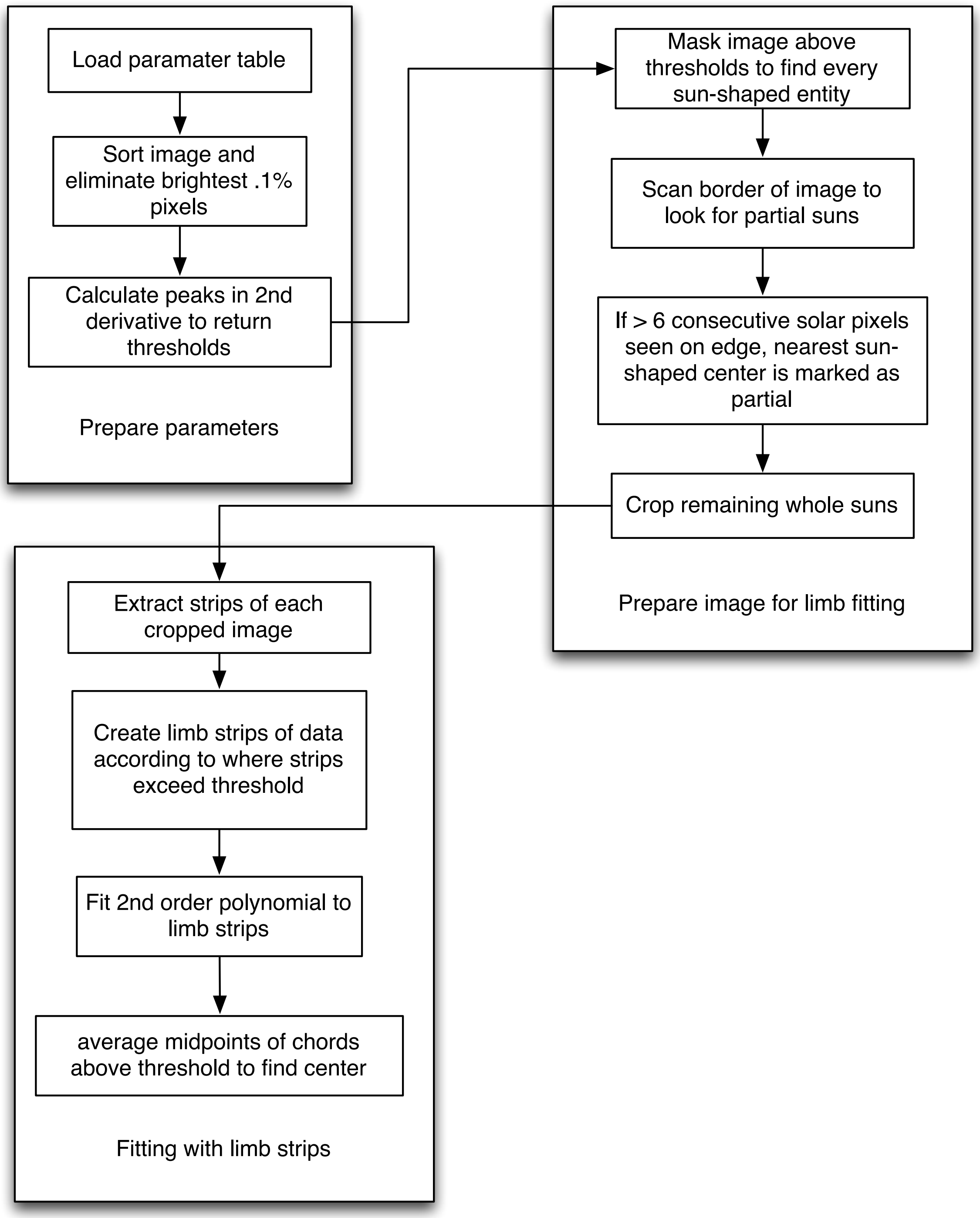


(g) `deriv(smooth(deriv(smooth(sortedarray)),/edge_wrap))`



(h) `deriv(ts_smooth(deriv(smooth(sortedarray))))`

Figure 1


Load paramater table
Sort image and eliminate brightest .1% pixels
Calculate peaks in 2nd derivative to return thresholds
Prepare parameters
Mask image above thresholds to find every sun-shaped entity
Scan border of image to look for partial suns
If > 6 consecutive solar pixels seen on edge, nearest sun-shaped center is marked as partial
Crop remaining whole suns
Prepare image for limb fitting
Extract strips of each cropped image
Create limb strips of data according to where strips exceed threshold
Fit 2nd order polynomial to limb strips
average midpoints of chords above threshold to find center
Fitting with limb strips

4 Psuedo Issues

One recurring topic that is brought up with each iteration of this software is how fast/accurate it is. The problem is that we don't have a baseline "This is where the center actually is" location of the solar centers. This makes it difficult to see how accurate our results are. We have compared limb-fitted results to simple mask-centroiding results to gauge accuracy, although a more reliable centroiding function may involve fitting the solar data to a 2D Gaussian.