Feature Tracking with YAFTA

Feature tracking amounts to, essentially, two tasks:

1. identifying features in an image; and
2. matching identified features between images.

Making feature tracking useful to answer scientific questions imposes an additional task:

3. analysis of tracking results.

YAFTA (Yet Another Feature Tracking Algorithm) is a suite of IDL programs that accomplishes these tasks using an intuitive, modular approach. Though relatively simple, these routines can appear complex to new users. This document aims familiarize the uninitiated with their use. Below, I first outline how each of these three tasks are accomplished, in general terms; then I give specific examples.

The code, supporting software, and electronic versions of this documentation can be downloaded from *http://solarmuri.ssl.berkeley.edu/∼welsch/public/software/YAFTA/* .

1. Identifying Features

"Features" are collections of pixels in a 2-D, (N_x, N_y) image array that are grouped according to some criterion. YAFTA is optimized for application to magnetograms (line-of-sight, as in Figure 1, or the normal component of vector magnetograms), and groups collections of pixels together that are *convex* in absolute field strength. Simply put, pixels that lie on the same "hills" in absolute field strength are grouped together. (Pixels lying on saddle points are grouped with pixels in the direction of the steepest uphill gradient.)

This grouping is accomplished with a flux-ranked, downhill labelling algorithm (Welsch and Longcope 2003) , RANKDOWN.PRO, which generates a label mask. A label mask is an (N_x, N_y) array in which the pixels bear the label of the feature, if any, with which each pixel has been grouped. Pixels below a user-set threshold in flux density are not grouped, and receive a zero label in the mask. YAFTA uses signed masks: the labels in the mask bear the sign of the magnetic flux of the pixels in the group, as in Figure 2. The IDL routine LABEL_REGION.PRO can also be used to group contiguous pixels — i.e., all positive-flux pixels above threshold that touch each other.

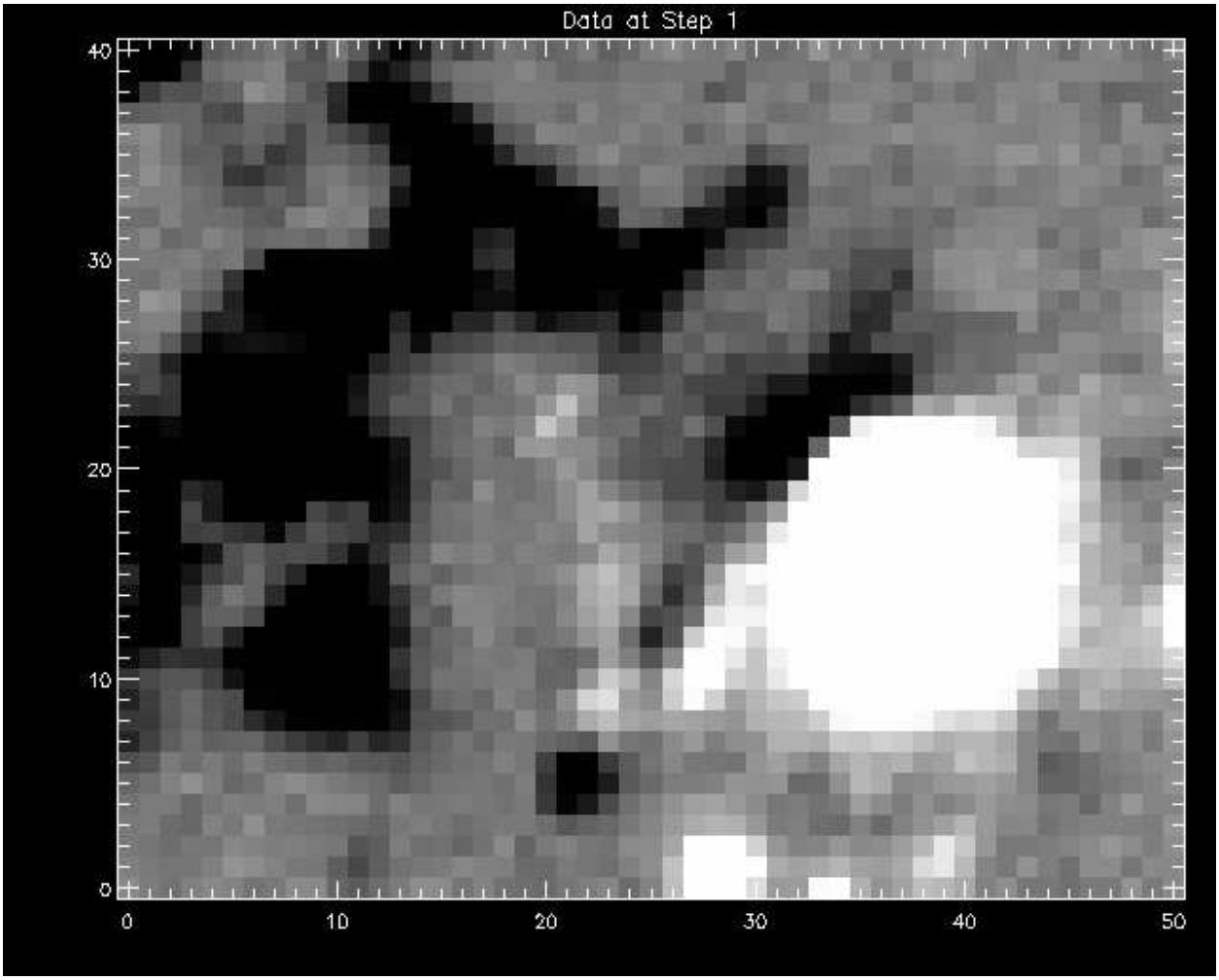

Fig. 1.— This constrast-enhanced, grayscale sub-image of an MDI full-disk, line-of-sight magnetogram, taken on 10 May 1997 at 00:04:05 UT, shows the large, positive-flux (white) sunspot in AR 8038, and nearby weaker, negative flux concentrations (black). This figure was generated by YAFTA_DISPLAY.PRO (see Table 4), using the command "YAFTA_display,image,/aspect".

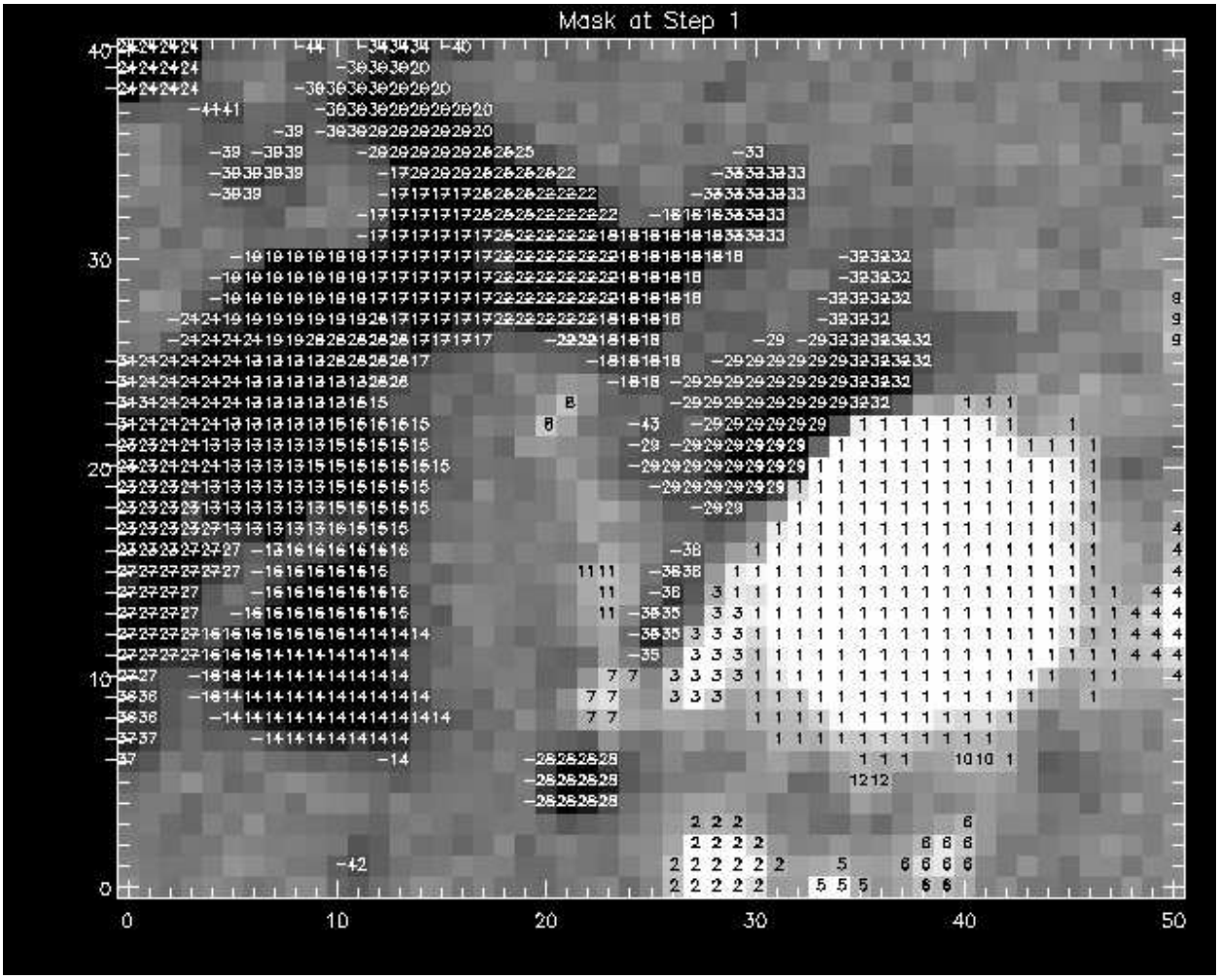

Fig. 2.— A mask of pixels from the magnetogram image Figure 1 grouped by convexity with RANKDOWN.PRO. Pixels below a 30G threshold in absolute flux density were not labelled. Negative (positive) pixels' labels are shown in white (black) for contrast. The minimum absolute flux density threshold passed to RANKDOWN.PRO is user-set. This figure was generated using the IDL routines YAFTA_DISPLAY.PRO and DISPLAY_LABELS.PRO (see Table 4), using the commands "YAFTA_display,image,/aspect" and "display_labels,image,mask".

After grouping pixels into features, CREATE_FEATURES.PRO analyzes groupings that match given criteria (minimum number of pixels, or peak field strength above a "peak threshold"), computes various quantities (detailed in Table 4) to characterize each such feature, and stores these quantities in an IDL structure associated with that feature. All the features' structures from a given step are concatenated in a 1-D array of structures. Figure 3 shows features identified from the mask shown in Figure 2.

2. Matching Features

Matching of features between two successive images is accomplished with MATCH_FEATURES_V*.PRO, which uses overlapping pixels from each image's mask, and the array of structures from each image, to match features. Features can fragment, merge, appear, and disappear. (While, ideally, each positive-flux feature would appear or disappear simultaneously with a an equal amount of negative flux, in practice unipolar features are often observed to appear or disappear without any nearby oppositely-signed counterpart [see Lamb and DeForest 2004]. This might arise from the coalescence or dispersal of flux across the magnetograph threshold.)

In the matching process, the labels stored in features' structures in the current time step are changed, as are the corresponding pixels in that time step's mask, via association with features from the previous step. Figure 4 shows features identified from the next image in the MDI sequence, and Figure 5 shows the relabelled figures after matching with the features in Figure 3.

Each feature's "source" and "terminous" are recorded in the .src and .trm fields, respectively, of that feature's structure (see Table 4). During the matching process at a given step, both the .src fields from the current step's features and the .trm fields from the previous step's features are updated. Features that do not appear, disappear, fragment, or merge, between steps are termed "one-to-one" matches. These features "propagate" themselves: their .label, .src, and .trm fields are identical.

Usually (depending upon image cadence), most features are matched one-to-one between steps. When a feature fragments into more than one "child," the largest child (by absolute flux) keeps the label of the "parent." An analogous procedure is used when many parent features merge into one child: the label of that parent with the greatest flux is passed on to the single child. Figures 3 and 5 show that features 15 and 11, for instance, have merged.

Because the matching process is complex, modifications to the feature matching routine are likely. Hence, each version of this routine carries a version number: MATCH_FEATURES_V01.PRO,

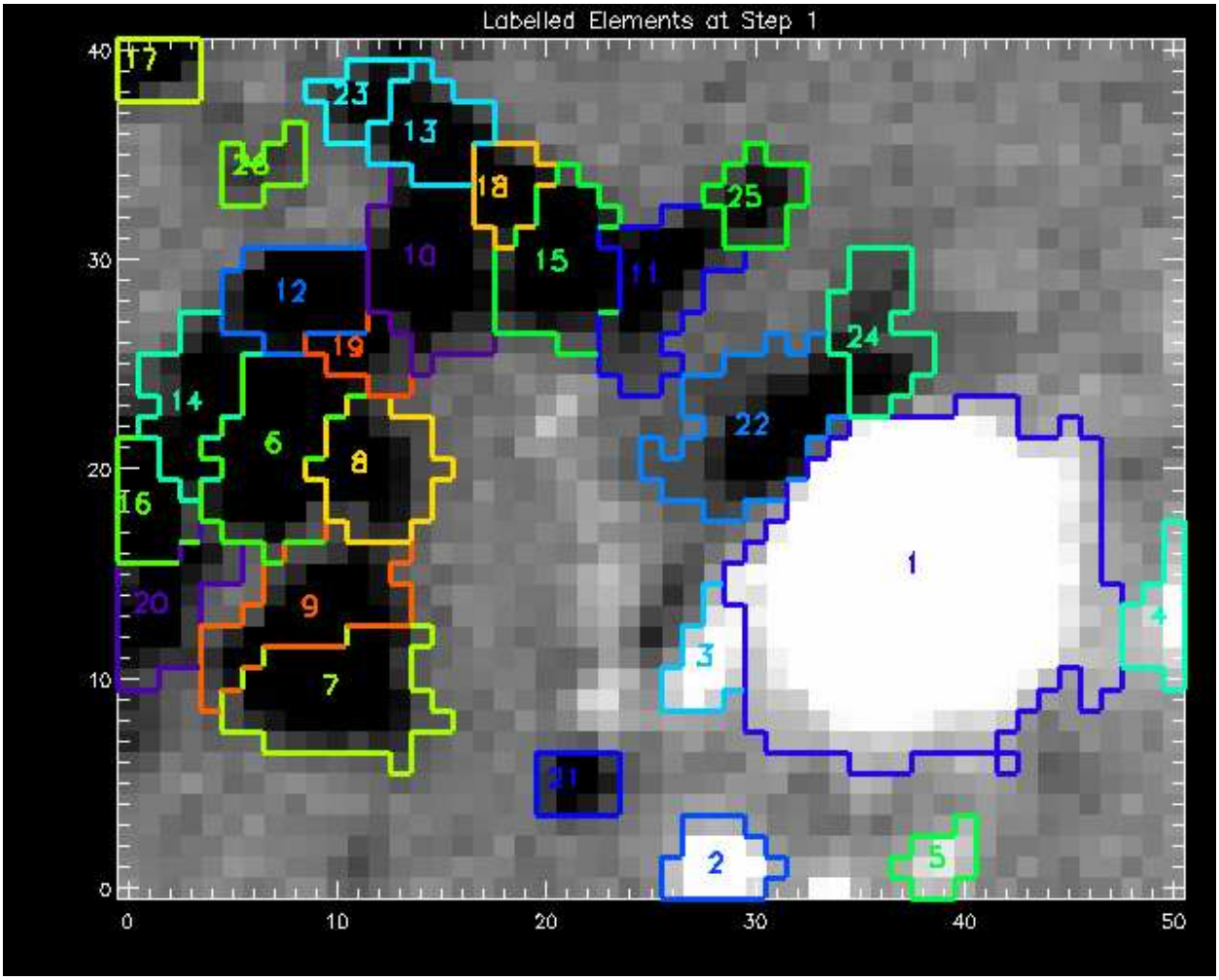

Fig. 3.— Flux elements, defined by CREATE_FEATURES.PRO, are outlined by colored contours, with their numerical labels plotted at their magnetic centers-of-flux. Note that pixel groupings identified in Figure 2 with less than 10 pixels are not identified as features. (The minimum-size threshold passed to CREATE_FEATURES.PRO is user-set.) The negative flux identified as features 22 and 24 probably results from a projection effect — AR 8038 lies to the east of disk center, and the penumbral field of its spot (feature 1) points away from the instrument, and lies essentially tangential to the solar surface. This figure was generated using the IDL routines YAFTA_DISPLAY.PRO PLOT_EDGES.PRO, and PLOT_LABELS.PRO (see Table 4), using the commands "YAFTA_display,image,/aspect", "plot_edges,mask", and "plot_labels,features".

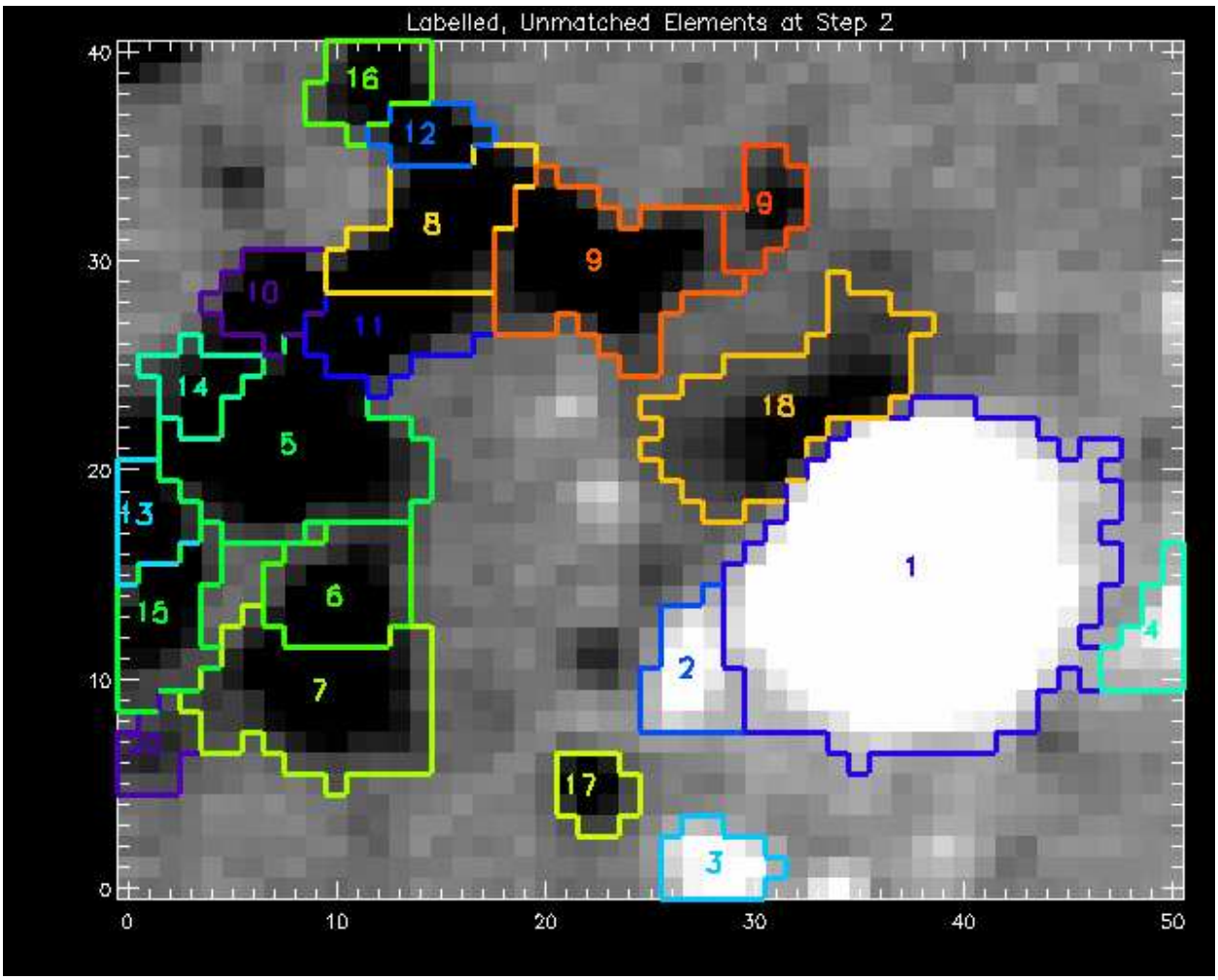

Fig. 4.— Features identified in the corresponding sub-region of the next MDI full-disk magnetogram, taken 96 minutes later. These features have not been matched with those in Figure 3.

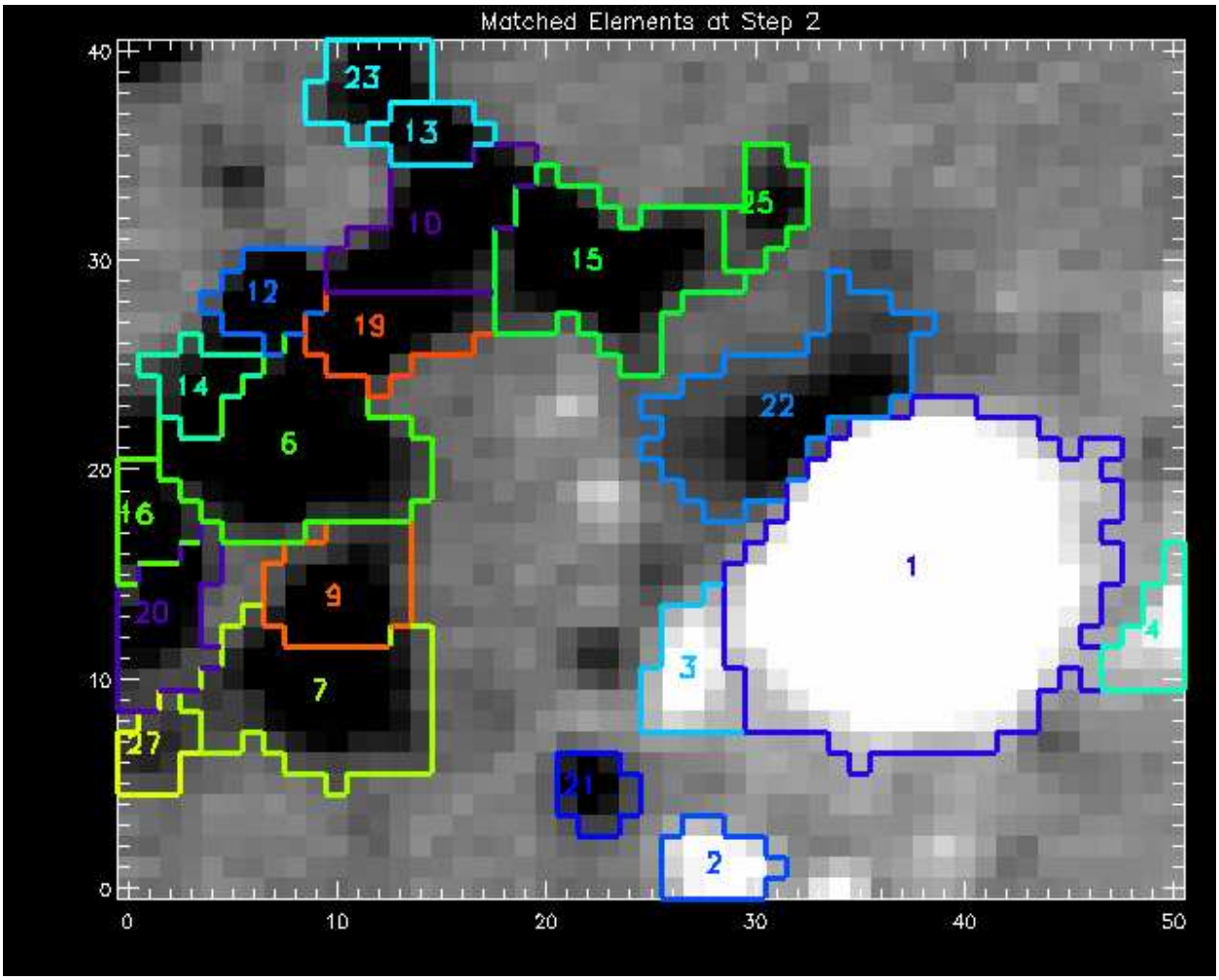

Fig. 5.— Relabelled features from Figure 4 after applying MATCH_FEATURES_V01.PRO, to identify correspoding features between Figures 3 and 4. Comparison of this image with Figure 4 shows that some features remain essentially unchanged (2,3,21, etc.), while others merge (e.g., 15 and 11), appear (27), or disappear (5, 26). No features fragment over this time interval. The slow image cadence has allowed relatively large changes in the features' structures to occur between tracking steps.

MATCH_FEATURES_V02.PRO, denoted here by MATCH_FEATURES_V*.PRO, etc.

3. Analyzing Tracking Results

Feature tracking results can be analyzed in many ways, so I only present a few examples here. In Table 4, I list the codes used directly for tracking, and some codes that are useful for displaying results. (Comment lines in these codes give examples of their use.) In order for these codes to run correctly, IDL must call these non-standard routines, so make sure these routines are in the file path IDL will search. In Table 4, I describe the quantities stored by CREATE_FEATURES.PRO for each feature at each time step.

I have also provided several IDL files for tutorial purposes. The main level IDL code YAFTA_TEST.PRO loads in data from YAFTA_INPUT.SAV, and runs the tracking routines, and generates the output data stored in YAFTA_OUTPUT.SAV. By restoring YAFTA_OUTPUT.SAV, one can emulate the commands below. To facilitate entering these commands in IDL, I have stored them in a separate file, YAFTA_IDL.PRO, stripped of any text except for comment lines.

- To find the structures associated with a feature, say the feature labelled # 8, and plot the feature's unsigned flux as a function of time, one could use

  IDL> index8 = where(features.label eq 8, n8)
  IDL> if n8 ne 0 then flux8 = features(index8).phi
  IDL> if n8 ne 0 then step8 = features(index8).step
  IDL> if n8 ne 0 then plot, step8, flux8

  This could be done another way with less code,

  IDL> index8 = where(features.label eq 8, n8)
  IDL> if n8 ne 0 then feat8 = features(index8)
  IDL> if n8 ne 0 then plot, feat8.step, feat8.phi

  To plot the signed flux, one can multiply the .phi field by the .sign field,

  IDL> if n8 ne 0 then plot, feat8.step, feat8.phi*feat8.sign

- A feature that first appears at step $t = k$ where no features previously existed (and is therefore not a result of fragmentation [see below]) has $-k$ in its source field. (All

features from the initial tracking step have .src set to 0.) Analogously, features that disappear at a step $t = k$ have .trm set to $-k$. (All features from the final tracking step have .trm set to 0.) To find appearances, and calculate their contribution to the average flux (excluding all features from the first step), one could use

```
IDL> apps = where(features.src lt 0, n_apps)
IDL> if n_apps ne 0 then appflux = total(features(apps).phi)
IDL> notfirst = features(where(features.step gt 0))
IDL> if n_apps ne 0 then flux = total(notfirst.phi)
IDL> if n_apps ne 0 then print, appflux, flux
```

- A feature at step $t = k$ that derived from a feature from step $t = k - 1$ (either as a one-to-one match, or as a fragment) has the label of the feature from $t = k - 1$ in its .src field. In a fragmentation, though, only the largest fragment (by flux) keeps its .src label, so fragments can be found by searching for .src fields that are positive and differ from .label fields. (Recall that all features from the initial step have a zero in their .src fields!) So, to find the average unsigned flux contained in a fragment, use:

```
IDL> where_frag = where(features.label ne features.src and $
IDL> features.src gt 0)
IDL> if n_frags ne 0 then frags = features(where_frag)
IDL> if n_frags ne 0 then fragflux = total(frags.phi)/n_frags
```

  This can be compared to the average unsigned flux of all elements,

```
IDL> avgflux= total(features.phi)/n_elements(features.phi)
IDL> print,'Avg. Unsigned Flux in All Features, in Fragments:'
IDL> print,avgflux,fragflux
```

- The 1-D pixel addresses of each feature are also stored in that feature's structure (as a string variable), and can be used to visualize a feature's shape at a given step.

```
IDL> mask_str = features(13).mask_str
IDL> addresses = long(strsplit(mask_str,/extract))
IDL> newmask = all_masks(*,*,0)
IDL> newmask(*,*) = 0
IDL> newmask(addresses) = 13
IDL> YAFTA_display, img, /aspect
IDL> plot_edges, newmask
```

This "feature mask" can also be visualized in isolation,

```
IDL> YAFTA_display, newmask, /asp
```

and even dilated, using IDL's DILATE.PRO,

```
IDL> dilmask = newmask
IDL> dilmask(addresses) = 10
IDL> s1 = replicate(1,3,3) ; dilates to nearest neighbors
IDL> dilate1 = 9*fix(dilate(dilmask, s1))
IDL> s2 = replicate(1,5,5) ; dilates to 2 nearest neighbors
IDL> dilate2 = 8*fix(dilate(dilmask, s2))
IDL> YAFTA_display, dilmask+dilate1+dilate2, /asp
IDL> plot_edges, newmask
```

One useful application of YAFTA (and feature tracking in general) is to study the process of cancellation. Perhaps the first step in using YAFTA for cancellation analysis is to determine when cancellation is occurring. The IDL program FIND_NEARBY.PRO provides a potentially useful scheme, as outlined below, for finding closely spaced pairs of oppositely signed features.

1. Input the array of feature structures and mask from one or more time steps.
2. For each time step, construct a "dilated mask" of negative features, using IDL's DILATE.PRO to dilate the mask of each negative feature, and combining them into a single mask. The number of pixels by which to dilate is a free parameter.
3. (Since dilated feature masks can overlap with masks from other features [dilated or otherwise], pixel labels that conflict with previously assigned labels are not assigned.)
4. Loop over all positive-flux features from the same time step.
5. In the loop, dilate each feature's mask, and multiply the result by the dilated negative mask. If the product array of the dilated masks contains non-zero elements, then the masks overlap, and overlapping labels and pixels are found.
6. Record either the indices (default) or labels of the overlapping features in one array, and the corresponding overlapping pixel addresses (as character strings) in another.

This routine only *identifies* cases in which the dilated masks overlap; determining whether cancellation actually occurred during that step or not is a different matter. Other criteria for cancellation need to be applied at this point, e.g.,

- Do the features' absolute fluxes decline at that step? If so, by how much?
- Do the features' centers-of-flux approach each other?

Notably, both of these criteria can be undermined by evolution away from the site of overlap — e.g., by merging with a same-sign feature away from the overlapping region. Further analysis of cancellation requires more detailed examination of the features identified by FIND_NEARBY.PRO.

4. Known Problems & Bugs

New users should be aware of these programs' potential shortcomings.

1. When run at the main program level (as in YAFTA_TEST.PRO), these routines create variables that are not destroyed after program completion. Consequently, errors can be encountered running these procedures repeatedly in the same IDL session without first deleting or renaming variables. ALternatively, these programs can be incorporated in a procedure, which will destroy local variables upon completion.
2. A bug exists on some architectures (so far, only DEC stations), in which IDL will not concatenate anonymous structures with identical fields, as done in CREATE_FEATURES.PRO. One easy work around is to us named structures instead. To do so, modify one line of code in CREATE_FEATURES.PRO, replacing

$$\text{feature} = \{\text{label:label\_j}, \cdots$$

with

$$\text{feature} = \{\text{feature,label:label\_j}, \cdots$$

Field	**Content**
label	longword integer label, i; changes when matching occurs
size	integer # of pixels grouped into feature i
step	integer tracking step at which this structure was created
src	integer, i's source, from matching: if negative, appeared at step (-src); if positive, label of parent feature from previous step [a]
trm	integer, i's terminus, from matching: if negative, disappeared at step (-src); if positive, label of child feature from next step [b]
sign	integer +/- 1 gives i's polarity
phi	abs(signal) × pixel area, dA, summed over i's N_i pixels, $\Phi_i = (\sum_{j=1}^{N_i} \lvert B_j \rvert) dA$
maxb	absolute max. field in feature; useful for dual thresholding
x	i's flux-weighted average x coord., $x_i = (\sum_{j=1}^{N_i} x_j B_j) dA/\Phi_i$
y	i's flux-weighted average x coord., $y_i = (\sum_{j=1}^{N_i} y_j B_j) dA/\Phi_i$
mask_str	string variable containing feature's pixel addresses, retrieve via addresses = long(strsplit(mask_str,/extract)) in IDL
x2	2nd moment of spatial distribution, $x2_i = (\sum_{j=1}^{N_i} (x_i - x_j)^2 B_j) dA/\Phi_i$
y2	2nd moment of spatial distribution, $y2_i = (\sum_{j=1}^{N_i} (y_i - y_j)^2 B_j) dA/\Phi_i$
xy	2nd moment of spatial distribution, $xy_i = (\sum_{j=1}^{N_i} (x_i - x_j)(y_i - y_j) B_j) dA/\Phi_i$
vx	given LCT velocities, vx is i's avg. horizontal velocity, $vx_i = (\sum_{j=1}^{N_i} vx_j B_j) dA/\Phi_i$
vx2	given LCT velocities, $vx2_i = (\sum_{j=1}^{N_i} (vx_i - vx_j)^2 B_j) dA/\Phi_i$
vy	given LCT velocities, vy is i's avg. vertical velocity, $vy_i = (\sum_{j=1}^{N_i} vy_j B_j) dA/\Phi_i$
vy2	given LCT velocities, $vy2_i = (\sum_{j=1}^{N_i} (vy_i - vy_j)^2 B_j) dA/\Phi_i$
vxvy	given LCT velocities, $vxvy_i = (\sum_{j=1}^{N_i} (vx_i - vx_j)(vy_i - vy_j) B_j) dA/\Phi_i$

[a] Parent-child .src fields are identical for one-to-one matches, but differ when a parent fragments into $M > 1$ children.

[b] Parent-child .trm fields are identical for one-to-one matches, but differ when $M > 1$ parents merge into one child.

IDL Program	**Purpose**
RANKDOWN.PRO	groups pixels (by convexity) into features, generates mask
PAD_ARRAY.PRO	called by RANKDOWN.PRO, pads an input array with rows and columns of zeros
CREATE_FEATURES.PRO	analyzes features, creates structure for each
MATCH_FEATURESV_01.PRO	matches features between time steps
YAFTA_DISPLAY.PRO	a renamed copy of DISPLAY.PRO, a wrapper for IDL's TVSCL.PRO, generates images (with titles, proper aspect ratio, etc.) easily
IMGSCL_FEN.PRO	called by YAFTA_DISPLAY.PRO
IMGEXP.PRO	called by YAFTA_DISPLAY.PRO
DISPLAY_LABELS.PRO	overlays mask's labels, via xyouts, on displayed image
TVREAD.PRO	reads current IDL window into various graphics file formats
PLOT_EDGES.PRO	overlays features' outlines, colored by label, on displayed image
PLOT_LABELS.PRO	overlays features' labels, colored by label, on displayed image
FIND_CANCEL.PRO	analyzes YAFTA results to find cancellations, and estimates cancelled flux
YAFTA_INPUT.SAV	IDL save file of MDI full-disk images of AR 8038, to test YAFTA
YAFTA_TEST.PRO	main level IDL routine that tracks data from YAFTA_INPUT.SAV, and stores results in YAFTA_OUTPUT.SAV
YAFTA_OUTPUT.SAV	IDL save file generated by YAFTA_TEST.PRO
YAFTA_IDL.PRO	main level IDL code, extracted from text of this document, for easily entering IDL commands above

REFERENCES

Lamb, D. A. and DeForest, C. E. 2004, American Astronomical Society Meeting Abstracts 204.

Welsch, B. T. and Longcope, D. W. 2003, ApJ 588.