IMAGE PROCESSING with ADSS

Peter Perry
DSTO
(Contractor)

Basic Concepts

- Image Processing in stages
  - Each stage is (relatively) simple.
  - Many stages are reusable.
- Unix/Linux can run stages in Parallel
  - If each is a separate process.
  - How could they communicate?

Inter-Process Communication

- Unit processes have three 'automatic' files.
  - **stdin** – Standard Input
  - **stdout** – Standard Output
  - **stderr** – Standard Error
- Unix/Linux allows us to 'pipe' processes.
  - A pipe is an area of memory
  - Maintained by the operating system
  - One process writes to it (uses it as stdout)
  - Second process reads from it (as stdin)

Simple Pipe Example

- Can pipe stdout of one process to stdin of next
  - This is called a “filter”
  - cat *.c
  - cat *.c | grep image
  - cat *.c | grep image | grep input
  - cat *.c | grep image | grep input | sort
- Can put together quite powerful commands like this.

So to ADSS

- Write a program that
  - reads commands from stdin
  - Does some simple image processing (really well and quickly)
  - Writes its results to stdout
- Link the programs together using pipes
  - That is ADSS

Sample ADSS processing

- A synthetic aperture radar (SAR) image
  - Bright returns are usually man-made objects
- So find the bright pixels in the image.
  - Define “bright” statistically
  - Many will just be noise
  - Where there are clumps, it is probably a target.
- The cluster the clumps together
- Then select clumps based on some criteria

Our ADSS Chain

Command and Data Language

- Text-based language describes:
  - The processing steps
  - The module parameters
  - The image(s) to be processed

The Processing Steps

- (process command prescreen “ata”)
  (process command cluster “clusterer”)
  (process command discrim “simple-features”)
- (process create pipeline
  prescreen cluster discrim)

The Configuration

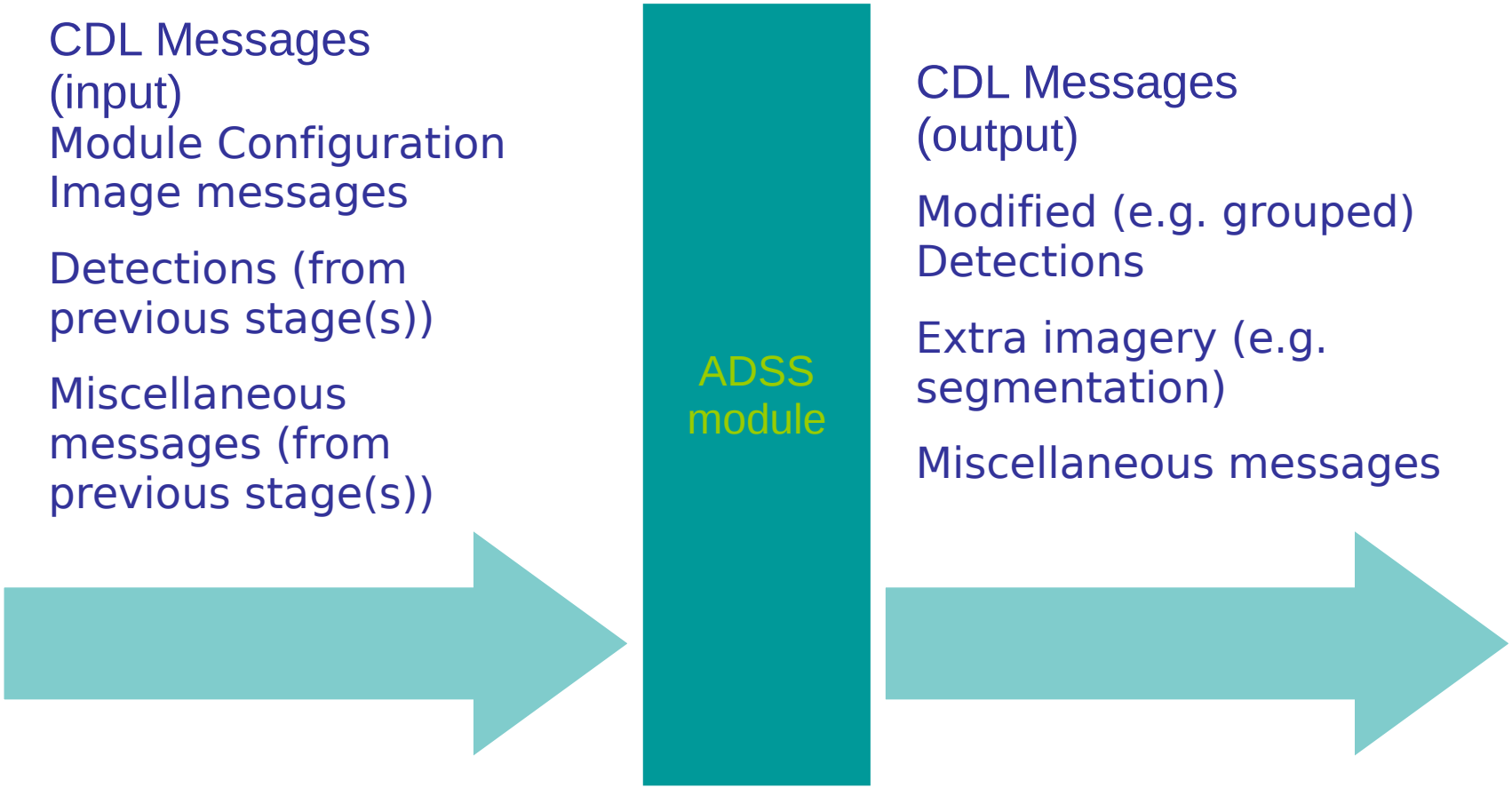
- (config prescreen
  (threshold-sd-over-mean 4.0)
  (guard-size 20)
  (outer-limit   25)
  (input-image main))
- (config cluster
  (cluster-distance 5)
  (input-image main))
- (config discrim
  (minimum-pixels 20))

The Image

- (input-image main isr “images/test-image.isr”)
- (available main 1024)
- (images-done)
- (exit)

Module Operation

Each module has a single input stream and single output stream



The Output

- The first stage will describe the bright pixels it found:
  - (detection (main (x 125) (y 115))
    (thresholded-value 4.78)
    (history prescreen))
- The second stage will cluster these:
  - (detection (main (x 100) (y 112))
    (bounds (x 98 8) (y 111 4))
    (rle-data (main (2 5) (0 3 2 3) (0 8) (4 2))
    (thresholded-value 5.2) (detected-pixels 21)
    (history cluster prescreen))

The Project

- Started 1997 by Dr Nick Redding
- I joined in January 2002
  - 8 modules supporting two image formats
- Currently a team of 10 contractors
  - Around 600 modules, 60 image formats
  - Nearly a million lines of code
- Plus several DSTO staff
- Plus collaborators in UK, Canada, USA

An ADSS Module

- Aim is to have each module
  - Self contained
  - Self documenting
- We achieve this through:
  - A module header which contains the documentation (LaTeX format).
  - And the interface specification (principally the parameters)
  - The module body implements the algorithm.

- /*** COPYRIGHT
  Copyright © Commonwealth of Australia 1993-2010
  ***/
- /*** AUTHOR
  Peter Perry, July 2010
  ***/
- /*** DOCUMENTATION
  A module to detect bright pixels.

  Bright pixels are detected by simple thresholding....
  ***/

```
/*** SPECIFICATION
(config (threshold ?real)
  (param threshold (type float) (init 2.0) store)
  (description “Threshold.”)
  (extended-description
    “This is the detection threshold.”))
(config (tile-size ?integer)
  (param tileSize (type int) (init 1000) store)
  (assert tileSize > 0)
  (description “Image processing tile size.”)
  (extended-description
    “The image will be processed in tiles of”
    “size $tileSize \times tileSize$.”))

(config default BASIC_CONFIGS)
```

- Specification section (cont):

```
(callback imagesReady)
(callback processTile)
(callback imagesDone)

(command default)

(global  imageFrag (type fragment))

***/
```

- The actual code:

```
#include “adss-module.hpp”
#include “absolute-threshold.inc”



tcdl_status imagesDone(void)
{
  frag_free(&moduleData.imageFrag);
  return CDL_OK;
}
```

```
tcdl_status imagesReady(void)
{
  frag_init(&moduleData.imageFrag,
            moduleData.tileSize,
            moduleData.tileSize,
            DTYPE_FLOAT, false,
            DSCALE_LIN_AMPL);
  cdl_set_mode(CDL_MODE_REGION,
                    NULL, NULL);
  cdl_set_geometry(moduleData.tileSize,
                        moduleData.tileSize,
                        0, 0, 1, 0, 0, 1, true);
  return CDL_OK;
}
```

```
tcdl_status processTile(tcdl_image *img,
  int x0, int y0, int w0, int h0,  /* input region */
  int x1, int y1, int w1, int h1) /* output region */
{
  int     x, y;
  float *p;

  frag_reshape(&moduleData.imageData,
                        w0, h0);
  ail_read_fragment(IMAGE_HANDLE(img),
                              &moduleData.imageData,
                              NULL, x0, y0, w0, h0);
  assign_frag_data(FLOAT, p,
                             moduleData.imageData);
```

```
  for (y = 0; y < w0; ++y)
    for (x = 0; x < h0; ++x) {
      if (*p >= moduleData.threshold) {
        cdl_generate_detection(x0 + x, y0 + y,
                               “thresholded-value, *p);
      }
       ++p;
    }
  return CDL_OK;
}
```

Comments

- As you can see, there is not much code. Nearly everything is done by the libraries
  - Conversion from external image format.
  - Loading image data into memory.
  - Working out the next tile to process
  - Generating the output (detection ..) message.
- I have made a few simplifications.
  - But this would work as printed.

Average Gray Absolute Difference Magnitude Map

Description

Average Gray Absolute Difference Magnitude Map

This module is capable of performing two forms of prescreening on the incoming image: Average Gray Absolute Difference Magnitude Map (AGADMM) and Average Gray Signed Difference Magnitude Map (AGSDMM). The prescreening method defaults to AGADMM, but can be explicitly selected via a CDL configuration message.

Its capability to detect differences in either direction make it suitable for use with electro-optic (EO) and infra-red (IR) imagery.

The module is implemented as a region processor.

The AGADMM algorithm has the following characteristics. In addition to the normal sliding window moved across the image, it has an inner window that varies in size at each outer window location; the inner window size is chosen to maximise the dissimilarity of the inner window, containing the target, from the outer window which contains the background.

Formally, this is expressed in the following way. Let the pixels of the inner window centred at (i, j) be denoted by Θ_{ij} and the pixels of the outer window, excluding those of the inner, be Ω_{ij} (also centred at (i, j)). Then the value of the AGADMM at the pixel on which the two windows are centred is given by the maximum absolute difference of the mean of Ω and Θ over all the possible inner window sizes centered in the outer window. If x_{ij} is the magnitude of the (i, j)-th pixel in the image, then the output d_{ij} can be expressed as

$$d_{ij} = \max_{\Theta_{ij}} \left| \frac{1}{|\Theta_{ij}|} \sum_{x_{kl} \in \Theta_{ij}} x_{kl} - \frac{1}{|\Omega_{ij}|} \sum_{x_{kl} \in \Omega_{ij}} x_{kl} \right| \tag{1}$$

where $|\cdot|$ denotes the cardinality of a set when appropriate. Essentially, the filter seeks to maximise the possible difference between the mean of the background pixels and that of a cluster of (hopefully brighter) target pixels.

AGADMM has three parameters to set at run time: the size of the outer window, and the maximum and minimum sizes of the inner window. Each of these parameters can be adjusted using the configuration parameters discussed in a following subsection. AGADMM has been implemented using column and row "caching" to capitalise on the minimal changes that occur between each placement of the inner and outer windows, making it a relatively fast and efficient algorithm.

The AGADMM algorithm was originally developed for optical imagery where the presence of a target can be indicated by both positive and negative differences between targets and their background. In contrast, in SAR imagery targets of interest are always indicated by a positive difference. Consequently, a variation of AGADMM was developed that replaces the absolute value operation of the previous equation (1) to give

$$d_{ij} = \max_{\Theta_{ij}} \operatorname{ramp} \left(\frac{1}{|\Theta_{ij}|} \sum_{x_{kl} \in \Theta_{ij}} x_{kl} - \frac{1}{|\Omega_{ij}|} \sum_{x_{kl} \in \Omega_{ij}} x_{kl} \right), \tag{2}$$

where ramp(·) denotes the function

$$\text{ramp}(x) = \begin{cases} x, & \text{if } x \geqslant 0; \\ 0, & \text{if } x < 0. \end{cases}$$

The AGADMM algorithm has been implemented with two possible detectors at its output: a global threshold on the filtered image, or a Neyman-Pearson detector. The principle behind the Neyman-Pearson detector is to choose the most likely hypothesis for a decision using the probability distributions for each class. In other words, the output y of the detector is assigned to either the target class T or clutter class C by the decision rule

$$\begin{cases} y \in T & \text{if } L(y) > \tau \\ y \in C & \text{if } L(y) \leqslant \tau \end{cases} \tag{3}$$

where τ is a weighting defined to be one here, and $L(y)$ is the likelihood ratio given by

$$L(y) = \frac{p_t(y)}{p_c(y)},$$

and $p_t(y)$ and $p_c(y)$ are the probability density functions of the target and clutter processes. Note that this is an optimal decision rule given $p_t(y)$ and $p_c(y)$. Training for the detector is provided by the `train-npd` module.

Complex image data is handled by conversion to amplitude.

Parameters (Alphabetical Listing)

`(algorithm` *symbol*`)`

Choose absolute or signed difference for detection.
The symbol must be one of:

`absolute-difference`	Absolute difference
`signed-difference`	Signed difference (detect bright targets)

`(detection-action` *symbol*`)`

Handling of incoming detection in detection processing mode.
The default is : `"pass"`
The symbol must be one of:

`pass`	Pass the incoming detection
`block`	Block the incoming detection
`pass-if`	Pass the incoming detection if this module tried to generate detections
`block-if`	Block the incoming detection if this module tried to generate detections
`pass-new`	Pass only detections this module generates.
`pass-both`	Pass both the original detection an detections this module generates.

`(inner-diameter-max` *integer*`)`

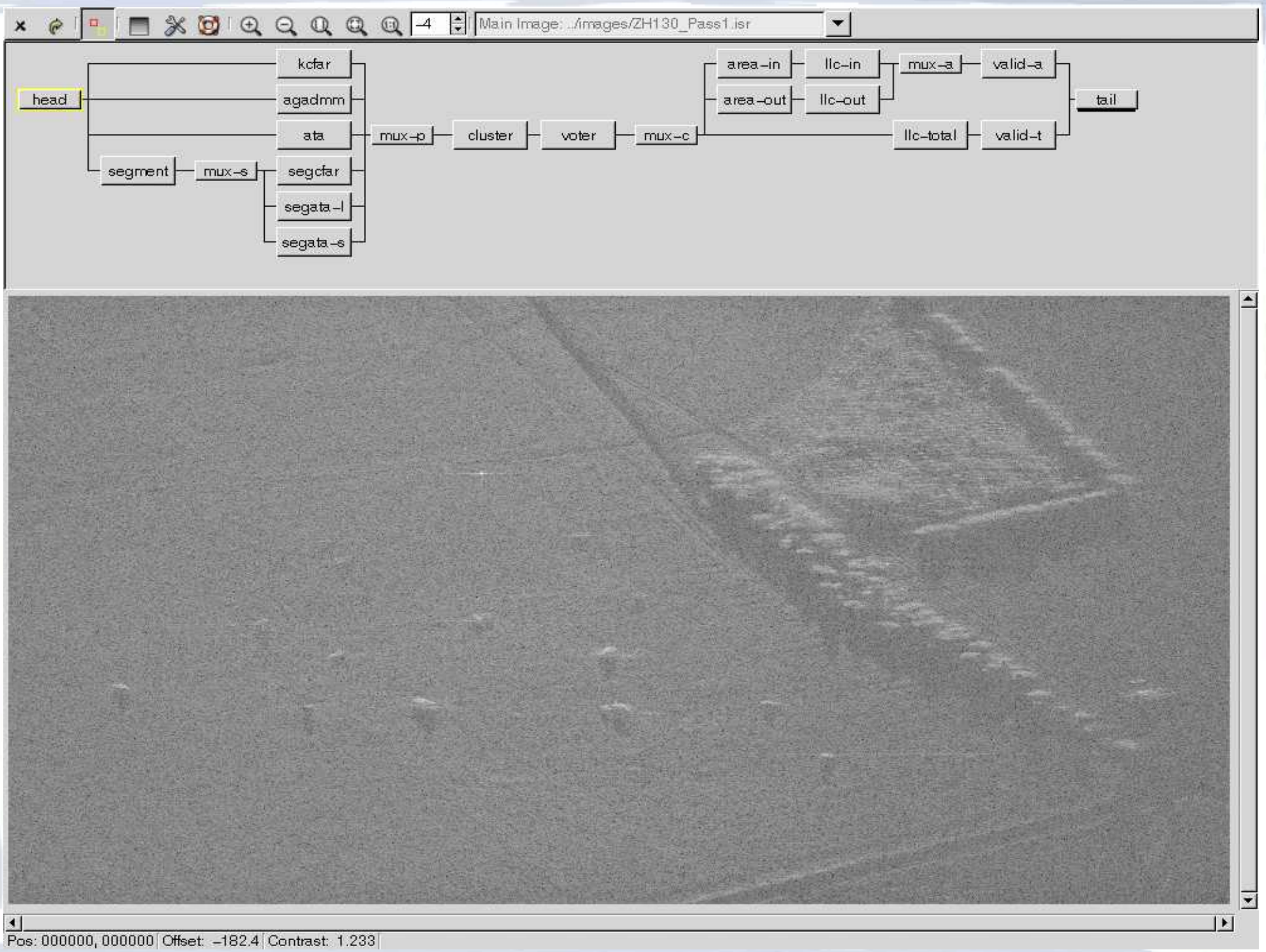
Filter maximum inner diameter.
The values are tested against : `((innerMax > 1))`

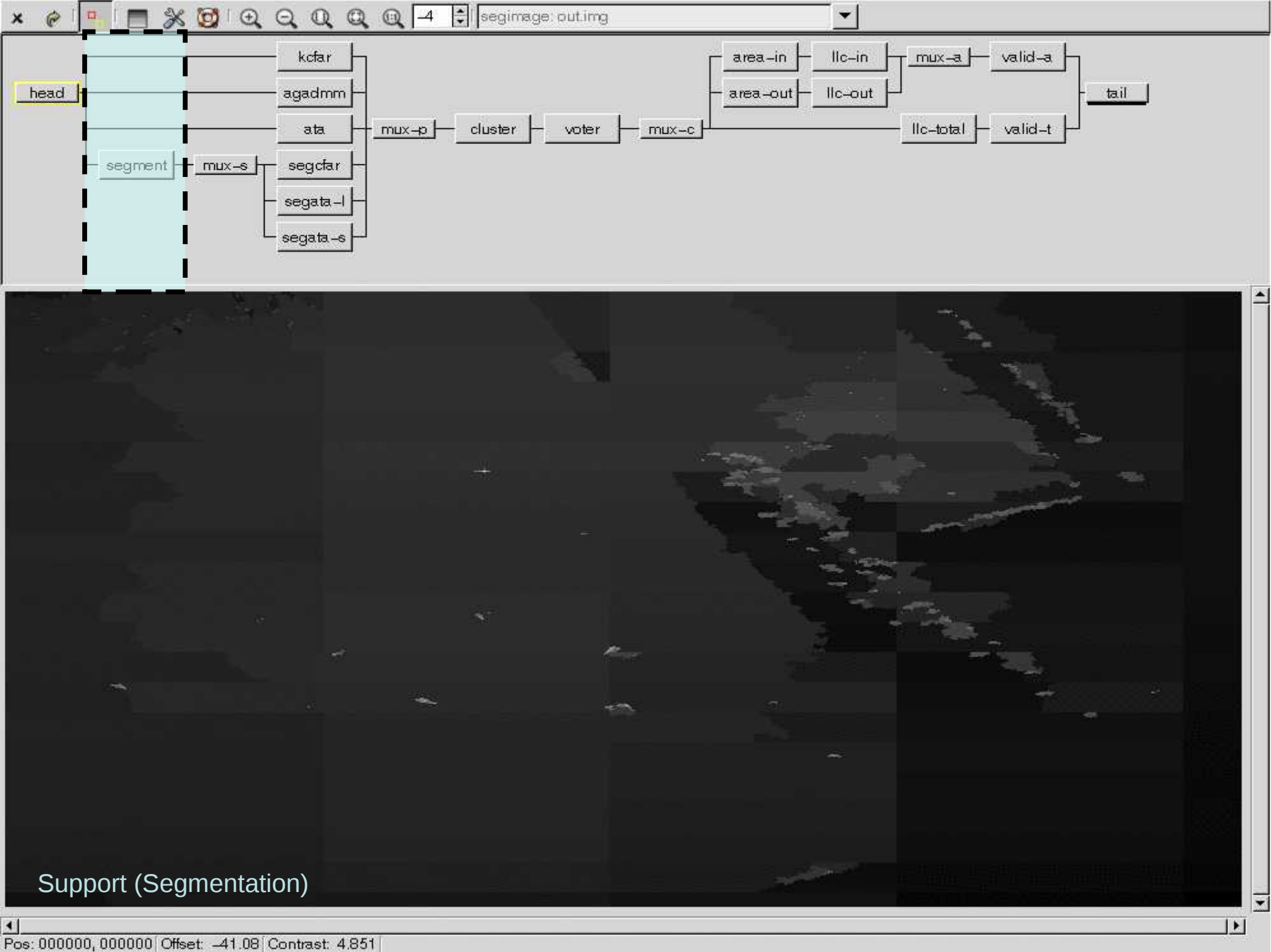
Application Areas

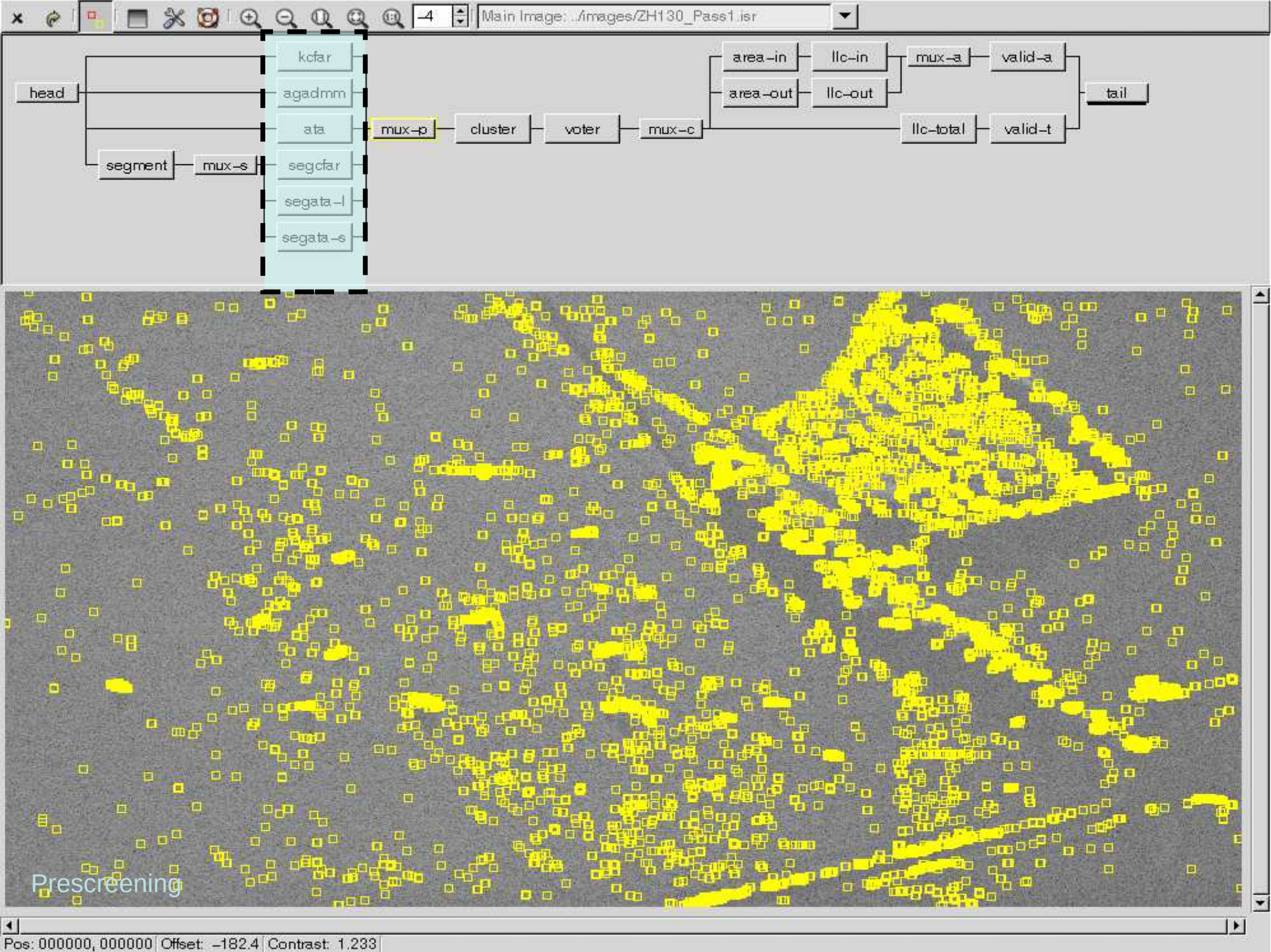
- Satellite and Wide Area Surveillance
  - Detecting ships.
- Video Processing
  - Detecting moving features.
  - Identifying vehicles (ellipse detection).
- Hyperspectral Image Processing
  - 128 spectral bands (cf 3 in a colour image)
- Image Fusion
  - Merging data from 2 or 3 different sensors

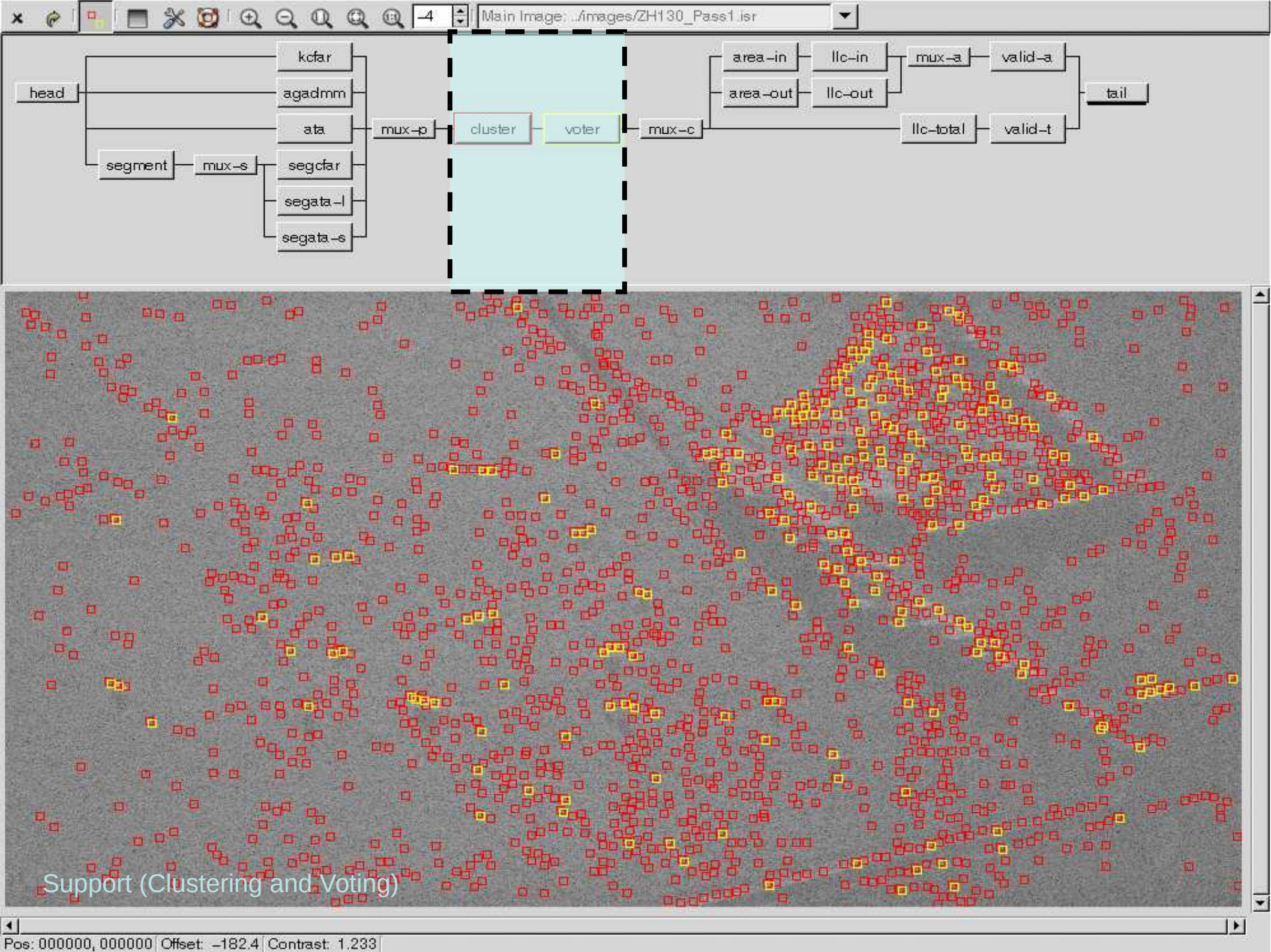
Application Areas (cont)

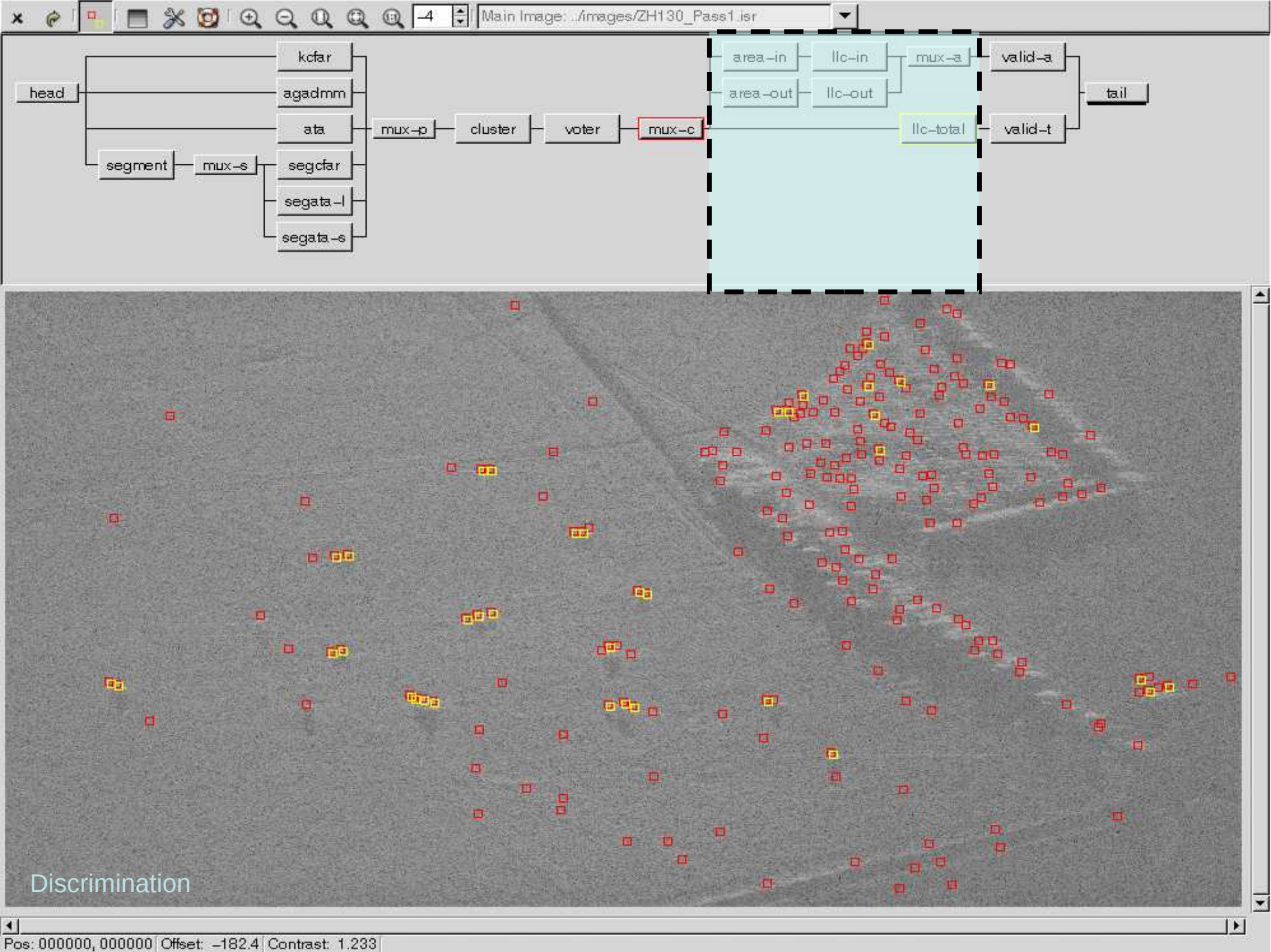
- Image Registration
  - Merging data from two or more surveys over the same region.
- Face and Gait Recognition
  - Identifying people in surveillance video.
- Warning of Abandoned Baggage

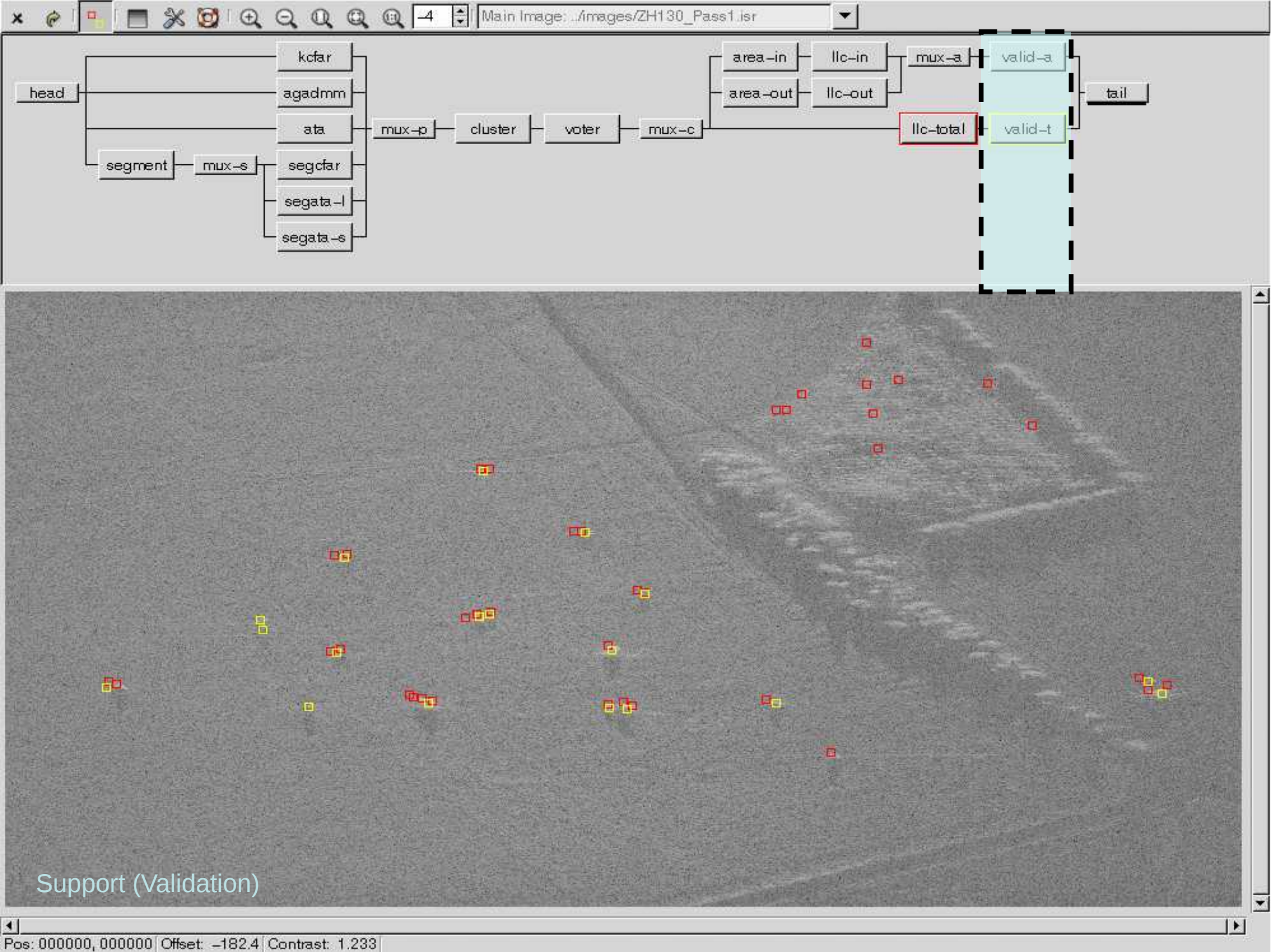

-4
Main Image: ../images/ZH130_Pass1.isr
head
kcfar
agadmm
ata
segment
mux-s
segcfar
segata-l
segata-s
mux-p
cluster
voter
mux-c
area-in
area-out
llc-in
llc-out
mux-a
valid-a
llc-total
valid-t
tail
Pos: 000000, 000000
Offset: -182.4
Contrast: 1.233


-4
segimage: out.img
head
kcfar
agadmm
ata
segment
mux-s
segcfar
segata-l
segata-s
mux-p
cluster
voter
mux-c
area-in
area-out
llc-in
llc-out
mux-a
valid-a
llc-total
valid-t
tail
Support (Segmentation)
Pos: 000000, 000000
Offset: -41.08
Contrast: 4.851


-4
Main Image: ../images/ZH130_Pass1.isr
head
kcfar
agadmm
ata
segment
mux-s
segcfar
segata-l
segata-s
mux-p
cluster
voter
mux-c
area-in
area-out
llc-in
llc-out
mux-a
valid-a
llc-total
valid-t
tail
Prescreening
Pos: 000000, 000000
Offset: -182.4
Contrast: 1.233


-4
Main Image: ../images/ZH130_Pass1.isr
head
kcfar
agadmm
ata
segment
mux-s
segcfar
segata-l
segata-s
mux-p
cluster
voter
mux-c
area-in
area-out
llc-in
llc-out
mux-a
llc-total
valid-a
valid-t
tail
Support (Clustering and Voting)
Pos: 000000, 000000
Offset: -182.4
Contrast: 1.233


-4
Main Image: ../images/ZH130_Pass1.isr
head
kcfar
agadmm
ata
segment
mux-s
segcfar
segata-l
segata-s
mux-p
cluster
voter
mux-c
area-in
area-out
llc-in
llc-out
mux-a
llc-total
valid-a
valid-t
tail
Discrimination
Pos: 000000, 000000
Offset: -182.4
Contrast: 1.233


-4
Main Image: ../images/ZH130_Pass1.isr
head
kcfar
agadmm
ata
segment
mux-s
segcfar
segata-l
segata-s
mux-p
cluster
voter
mux-c
area-in
area-out
llc-in
llc-out
llc-total
mux-a
valid-a
valid-t
tail
Support (Validation)
Pos: 000000, 000000
Offset: -182.4
Contrast: 1.233

Geographic Data Fusion within ADSS

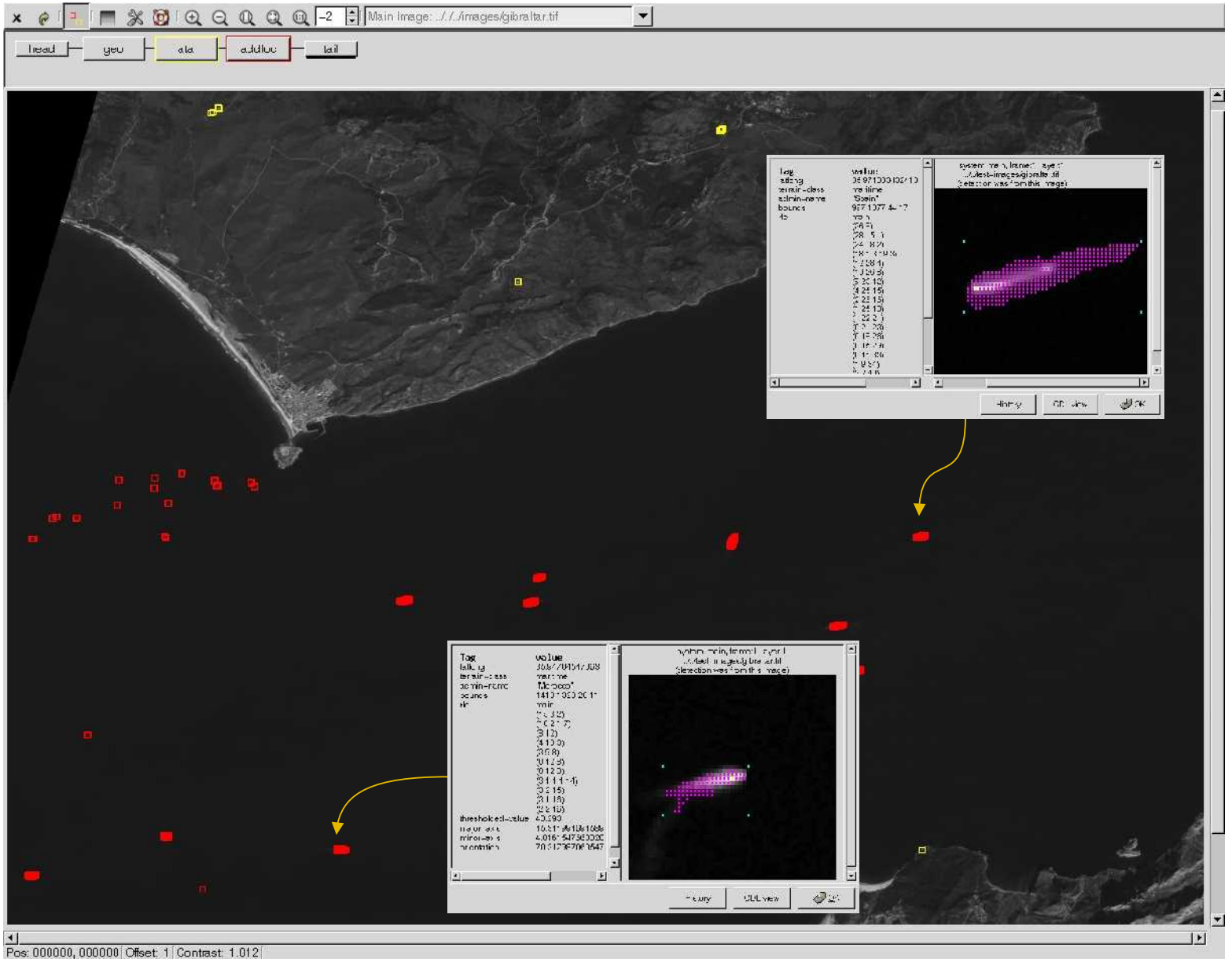


Image of the Straight of Gibraltar

Detections indicated by red and yellow boxes

Clicking on detections brings up dialog boxes showing geo-referencing data from the World Vector Shoreline Database

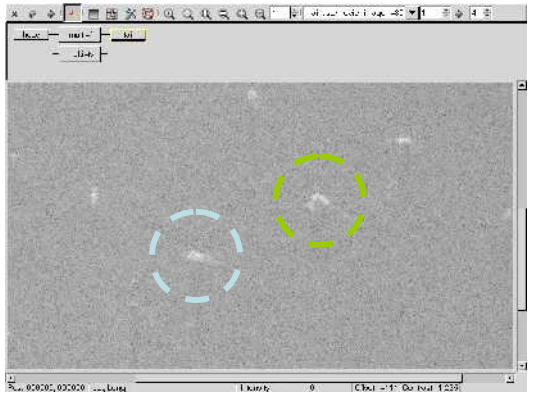
Can distinguish between targets in different territorial waters

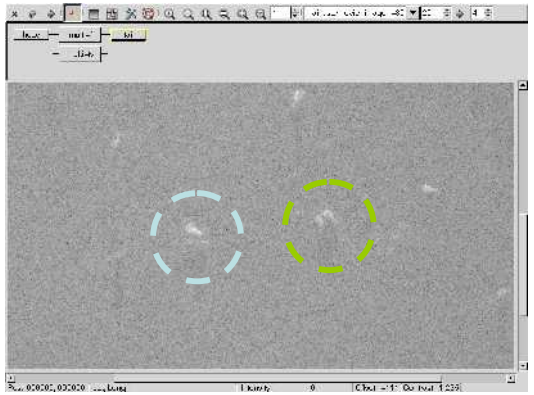
Terrestrial targets can be processed differently to maritime targets

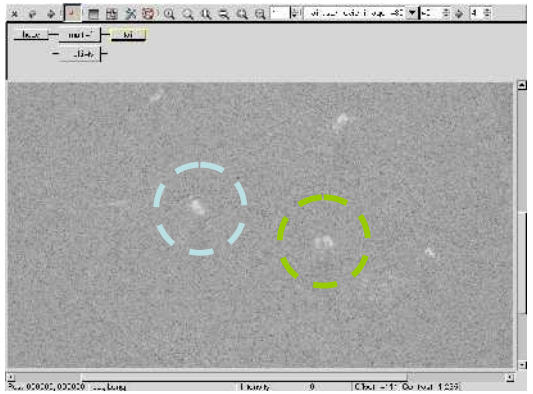
Red : Maritime Targets

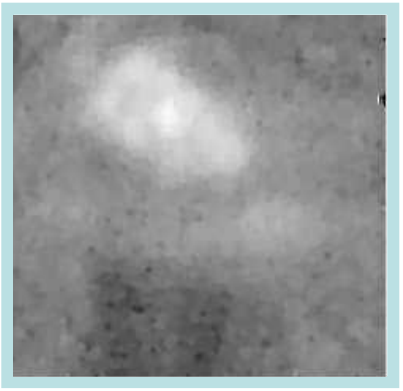
Yellow : Terrestrial Targets

Super Resolution – SAR Spot Sequence





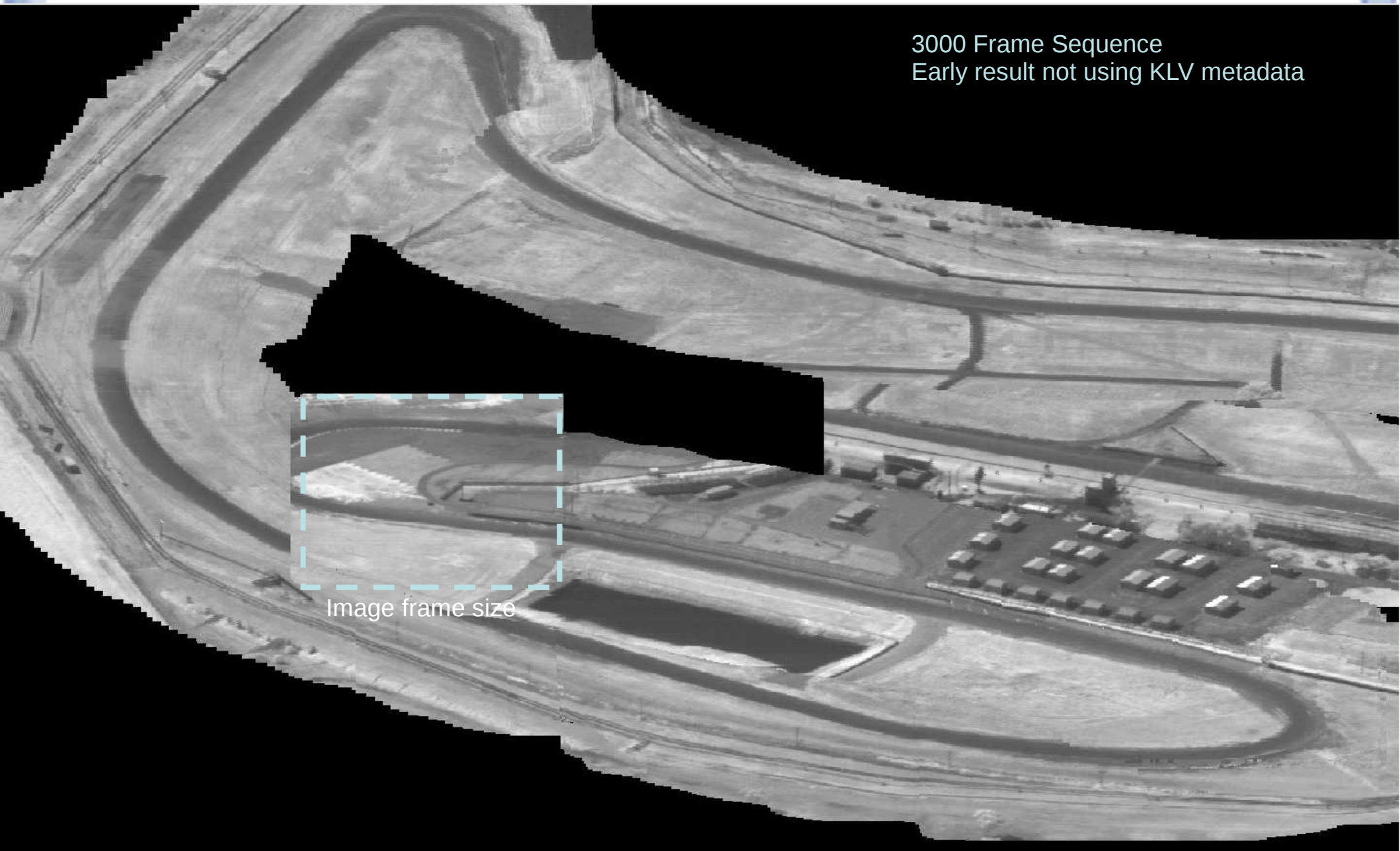




Super-resolved Targets

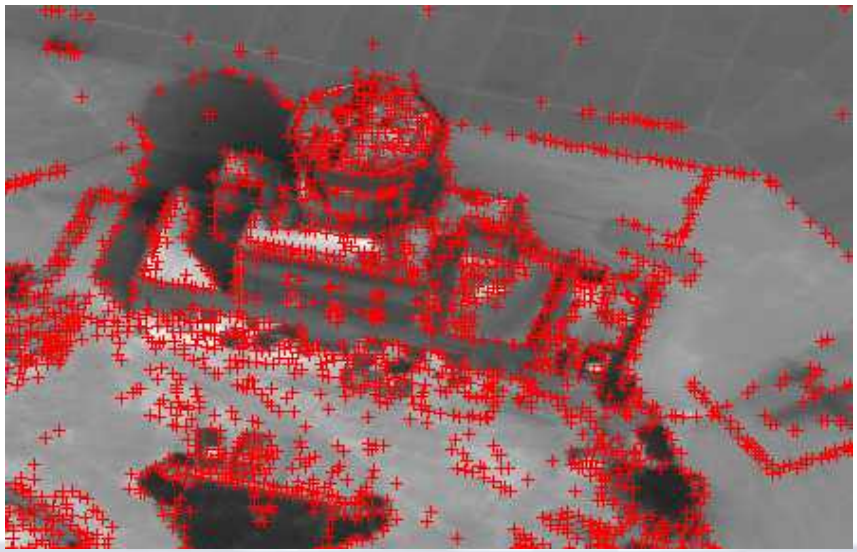
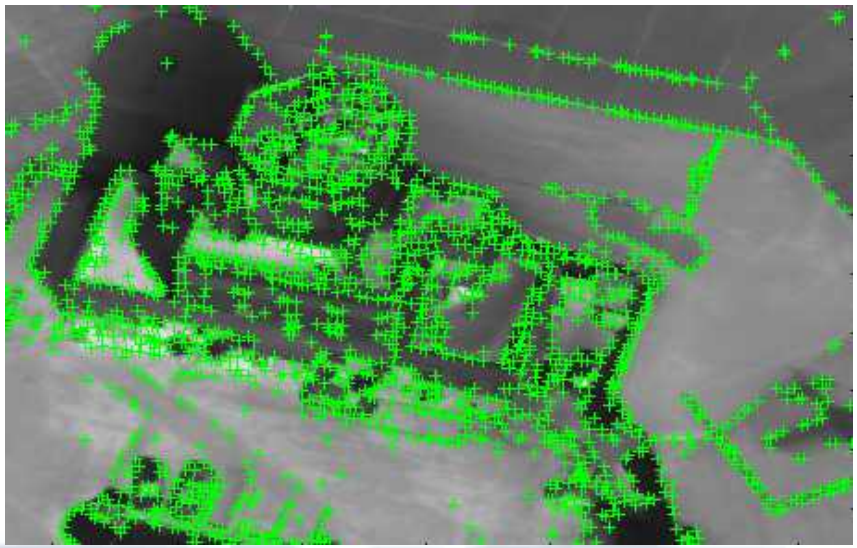
- Can obtain enhanced resolution of a target from a temporal sequence of images
- Information from multiple frames is used to build a single still image
- Can improve effective resolution of a sensor

Image Mosaic – MPEG Sequence



Example fly over of Parafield airfield control tower



Courtesy P. Torr's Structure and Motion Toolkit in Matlab


1:1:main-window - wheels.jpg
main: wheels.jpg
layer
Push
Monochrome
Pos: 000279, 000205
Lat, Long: 0:00:00.00N 0:00:00.00E
val: 64
Dets/pix: 22/156,510

Questions?