Gradient-Free Boundary Tracking*

Zhong Hu
Faculty Advisor: Todd Wittman
August 2007
UCLA Department of Mathematics

Section I. Introduction

In my paper, my main objective is to track objects in hyperspectral images, since hyperspectral images enable us to extract much more detailed information than standard RGB images. Classical segmentation methods, such as snakes and active contours, look at all the pixels in the image. For large multi-channel images, this is very expensive. Ideally, the number of pixels tracked should be proportional to the length of the boundary. Our solution is to simply "walk" the boundary in a systematic manner. A tracking algorithm is developed based on the robotic path planning algorithm developed by Jin and Bertozzi [1]. The algorithm is first tested on simple binary images and then extended to tracking objects in hyperspectral images.

Section II. Algorithm

The basic tracking algorithm is outlined below.

```
Input: Angular velocity w, tracking velocity V

Ask the user to input a point (x,y) in the image
if (x,y) is inside the boundary
        set d = 1
else
        set d = -1
Initialize $\theta = 0$
For fixed number of iterations
        Set $\theta = \theta + d * w$
            $x = x + V * \cos \theta$
            $y = y + V * \sin \theta$
        if (x,y) lies outside the image domain
                End the program
        if we have made a full circle
                $V = 2 * V$
        if we cross the boundary
                Store the boundary point
                $d = -d$
```

The algorithm I have developed provides a method to track boundaries of an arbitrary image. To determine whether the image is inside or outside the boundary, we create a binary signal d(x,y), such that at a certain threshold, d(x,y)=1 if (x,y) is inside the boundary and d(x,y)= -1 if (x,y) is outside the boundary. Next, the user picks an arbitrary point on

* This research is supported by NGA NURI grant HM1582-06-BAA-004

the image, which will be called (x,y). The number of iterations could either be fixed or a stopping criterion can be created. For each iteration, we move V units. If inside the boundary, the angle θ is increased by w, the angular velocity. If outside the boundary, the angle θ is decreased by w. The location x=x+V*cosθ and y=y+V*sinθ will be our new location.

If the initial point is more than V units away from the boundary, the algorithm will fail to track the boundary. In this case, the algorithm will repeatedly trace out a circle. To overcome this problem, if we detect that a full circle was made, the size of V is increased until we find the boundary point. Then it is set back to the original value of V. Fig. 1 illustrates the importance of this correction.

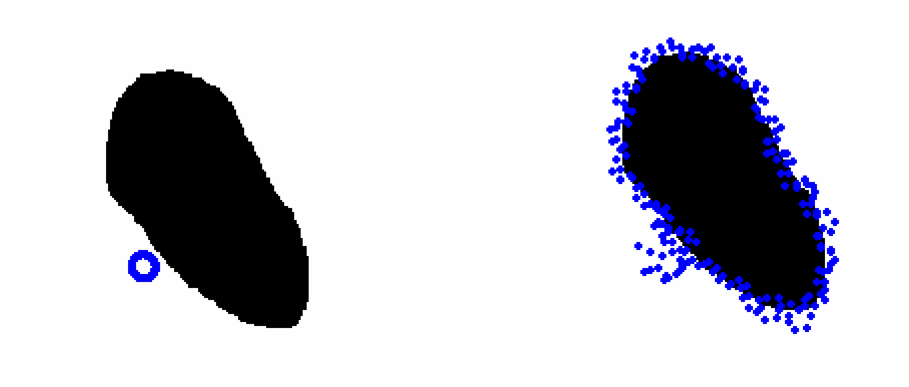
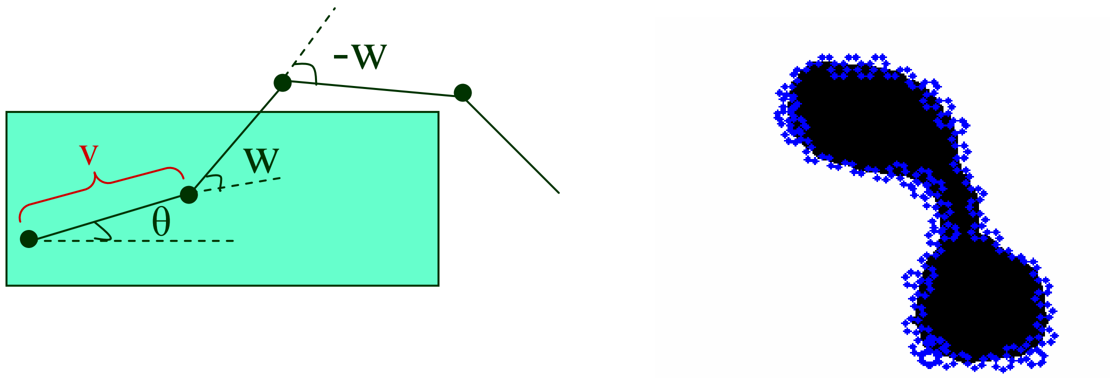


Fig. 1 Left: Without step V correction Right: With step V correction

When the value of d changes sign, this indicates that we have crossed the boundary of the object. We store the boundary point as the midpoint of the current value of (x,y) and the previous (x,y) point. Fig. 2 shows how the algorithm works and also provides an example of how it applies on a binary image.



ig. 2 Left: Illustration of the algorithm Right: Example of applying the algorithm

Section III. Performance

The term Percent Pixels on Boundary (PPB) is defined by:

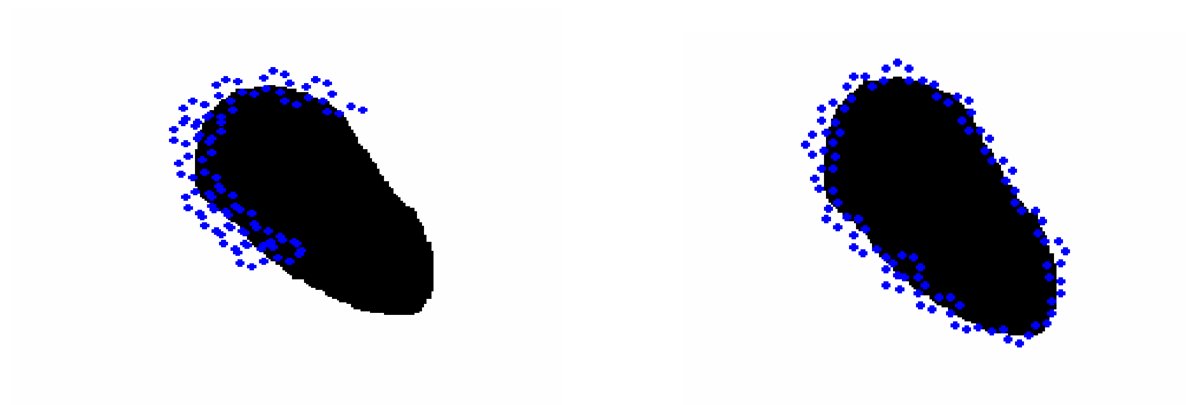
$$\mathrm{PPB} = \frac{\#\,\text{boundary points detected}}{\text{Total}\,\#\,\text{pixels tested}}$$

PPB is used to measure the performance of the algorithm. We want to detect as many boundary points as possible while minimizing the number of pixels tested. Thus, it makes sense that the higher the PPB, the higher efficiency is for tracing the boundary. Applying the algorithm on the same image with the same number of iterations and parameters, the PPB varies slightly depending on the initial point. In Fig. 3 left, we can see that the crossing angle θ is large, which indicates that the tracking is not efficient. To improve our tracking, we apply the angle correction proposed in [1]. After we cross the boundary, θ is updated

$$\theta = \theta + d\,\frac{w(t_2 - t_1) - 2\theta_{ref}}{2}$$

where t_1 and t_2 are times of the last two boundary crossings and θ_{ref} is a parameter.

From Fig. 3 right, the angle θ is much flatter once it passes the boundary. Also, the PPB with angle correction and θ_{ref} 0.5 is higher than without angle correction. Moreover, angle correction tracks more of the image boundary with the same number of iterations.



Speed V= 5, w=1, PPB=0.21, 100 iterations   Speed V= 5, w=1, PPB=0.42,100 iterations

Fig. 3 Left: Without angle correction   Right : With angle correction

If substantial noise is present in the images, our tracking algorithm fails to track the boundary. To overcome this, we apply two independent cumulative sum (CUSUM) filters as in [1]:

$$U(t) = \begin{cases} 0 & \text{if } t = 0 \\ \min(\overline{\mathrm{U}},\ \max(0,\ \ \mathrm{A}(x,\ \ y)\ -\ \mathrm{B}\ -\ c_u\ +\ \mathrm{U}) & \text{if } t > 0 \end{cases}$$

$$L(t) = \begin{cases} 0 & \text{if } t = 0 \\ \max(\ \overline{\mathrm{L}},\ \min(0,\ \ \mathrm{A}(x,\ \ y)\ -\ \mathrm{B}\ +\ c_l\ +\ \mathrm{L}) & \text{if } t > 0 \end{cases}$$

where $\overline{\mathrm{U}}$ and $\overline{\mathrm{L}}$ are the accumulation thresholds, A is the image, A(x,y) is the intensity at location (x,y), B is the threshold for the image A, and c_u and c_l are the "dead-zone" parameters. CUSUM filters deal with noise by minimizing the detection delay and keeping the probability of false record low. U(t) is called the "high-side filter" and L(t) is called the "low-side filter". If we move inside the boundary and the intensity is above B, U(t) increases as the accumulation of the measurement increases. When U(t) is greater than $\overline{U}$, we have detected a boundary crossing. The same idea holds for L(t).

In Figure 4, the left image shows the failure of the original algorithm to trace the boundary with Gaussian noise of variance 0.1. The image on the right shows the results using CUSUM filters by choosing $\overline{U}=50$, $\overline{L}=-50$, $c_u=2$, $c_l=2$, and B=100 to the image with Gaussian noise with variance 0.1. Figure 5 shows a plot of U versus L.

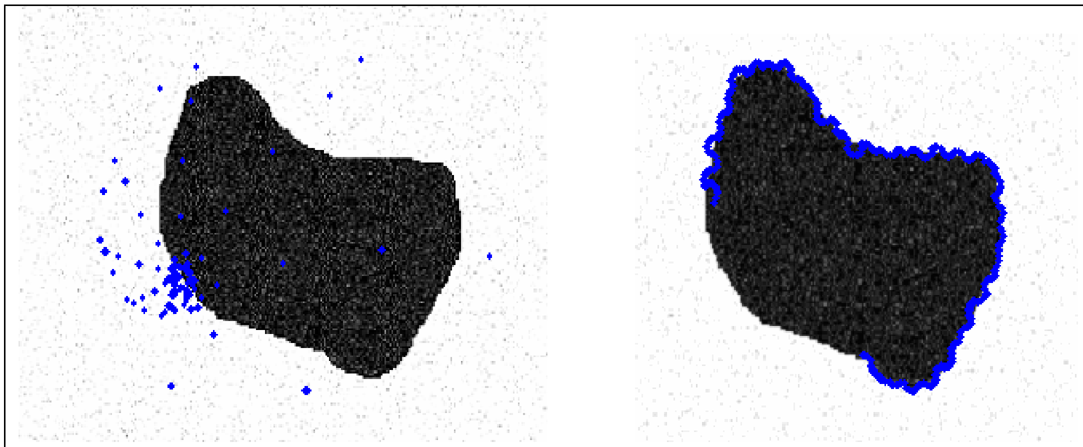


Fig. 4 Left: Without CUSUM filters Right: With CUSUM filters

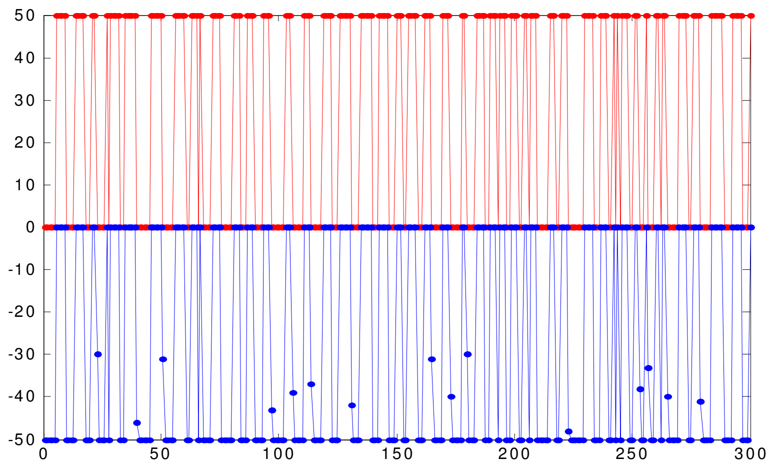
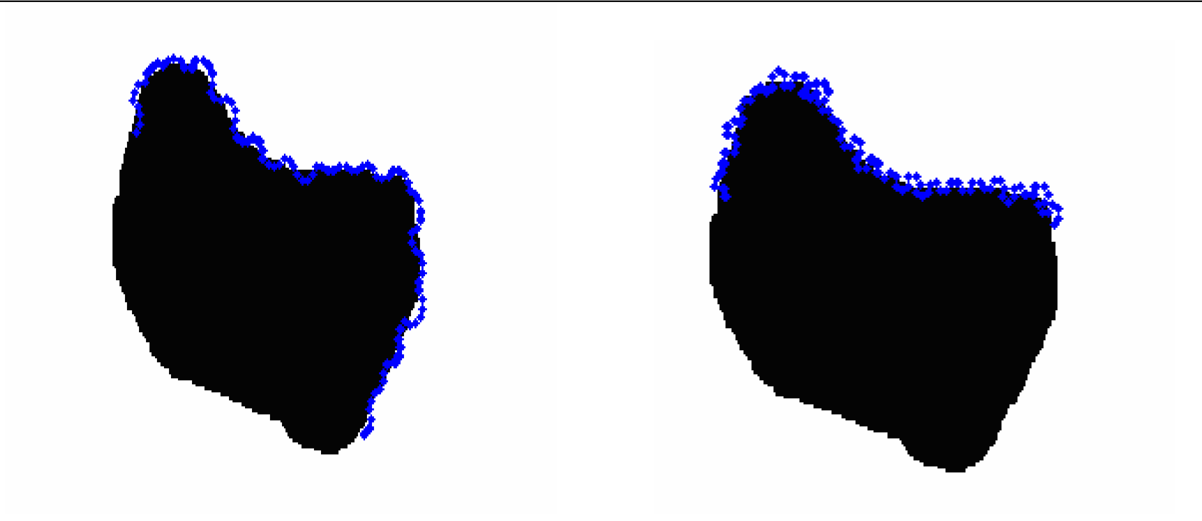


Fig. 5 The plot of U (red) versus L (blue) over time.

First, observe that CUSUM performs best on images with noise. Also, CUSUM helps us track the boundary more smoothly. The performance of using CUSUM filters is better possibly due to the noise. To verify the above results, we test the tracking algorithm both with and without the CUSUM filter on a noise-free image. The result coincides with the above results (see Fig. 6).



V=3, PPB=0.297,150 iterations V=3, PPB=0.43, 150 iterations

Fig. 6 Left: With CUSUM filters Right: Without CUSUM filters

Section IV. Boundary tracking on hyperspectral images

In order to track the water in a hyperspectral image, we need a water detector. Instead of using a water detector, we are going to use a human activity detector to differentiate natural from man-made regions. We use Normalized Difference Vegetation Index (NDVI):

$$\text{NDVI} = \frac{\rho_{nir} - \rho_{red}}{\rho_{nir} + \rho_{red}}$$

where ρ_{nir} and ρ_{red} are the reflectance values in the near-infrared and red bands. At a certain threshold, NDVI does a good job differentiating land from water (see Fig. 7).

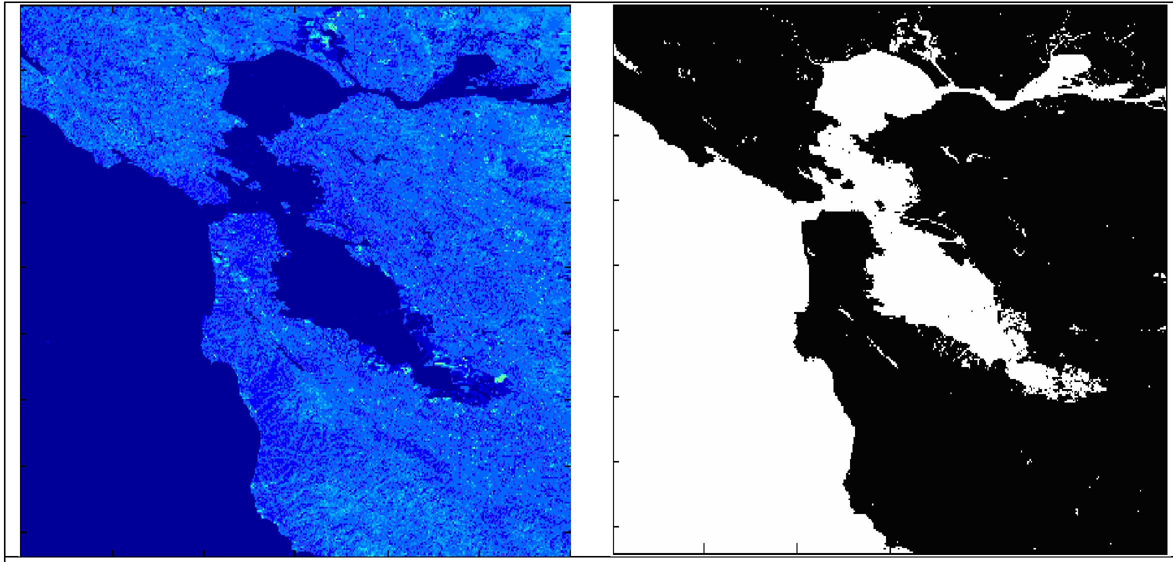
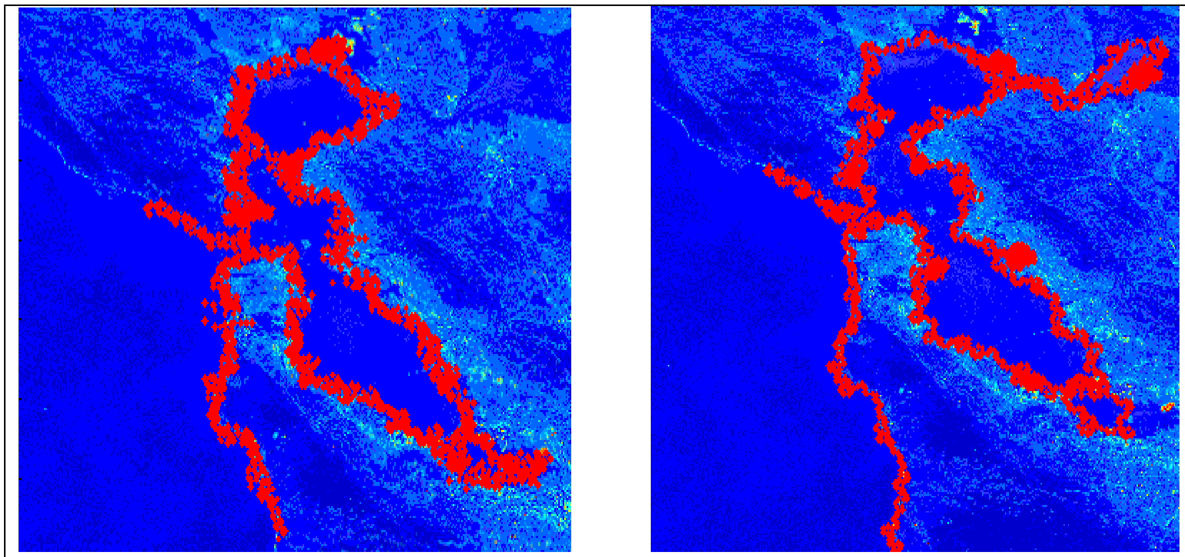


Fig. 7 Left: A Near Infrared band in San Francisco Bay
Right: NDVI of San Francisco Bay

Fig. 8 shows the results of using NDVI to trace the boundary on the green band of the hyperspectral image of San Francisco Bay. Both of the algorithms use the same w, initial point, and same number of iterations, but different V. The smaller V traces the boundary more accurately, but it takes a longer time to run.



a) V=25, Runtime=38.8 sec **b) V=15, Runtime=20.2 sec**

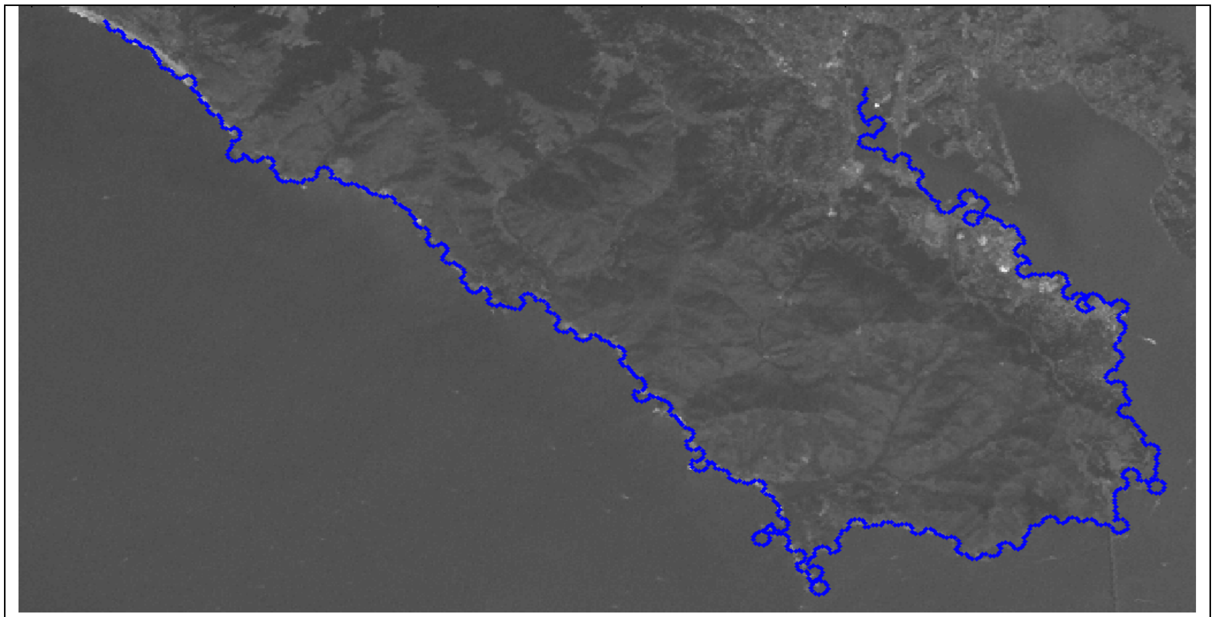
Fig. 8 Boundary tracking on a hyperspectral image of San Francisco Bay

The size of the hyperspectral image of San Francisco Bay is large, about 4050 by 3000 pixels. Thus, it is reasonable to use large units of V to track the boundary. To see the details of tracing the boundary, let's perform the algorithm on a small portion of the image and use small units of V. Also, it is a good idea to use the CUSUM filters since there may be some noise in the NDVI image. From Fig. 9, using the method of CUSUM filters with the right parameters seems to trace the boundary much better than the method without the CUSUM filters. However, the PPB with CUSUM filters is smaller than the one without CUSUM filters, which indicates that the PPB may not be ideal in measuring how well it can trace the boundary. Also, the CUSUM filters acts similar to the angle correction method in that the boundaries appear smoother, which indicates that tracking is more efficient.

References

[1] Z. Jin and A. L. Bertozzi, "Environmental Boundary Tracking and Estimation Using Multiple Autonomous Vehicles," UCLA CAM preprint, 2007.

[2] C. Unsalan and K. L. Boyer, "A System to Detect Houses and Residential Street Networks in Multispectral Satellite Images," ICPR04, 18 Dec. 2004


800 iterations with CUSUM, V=3, PPB = 0.33

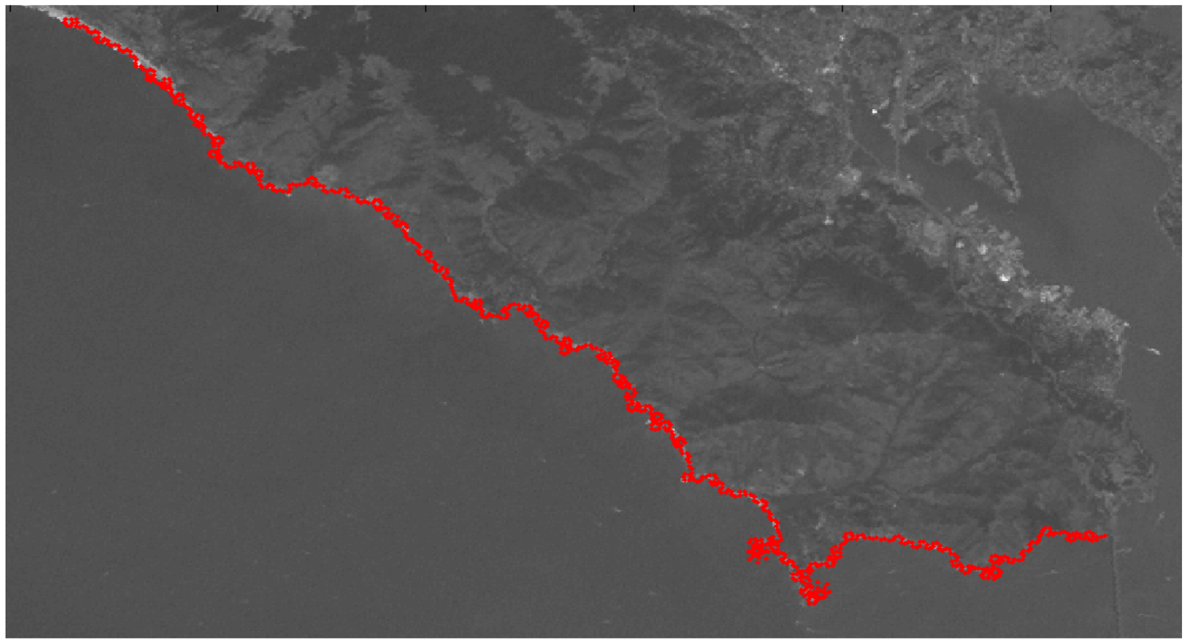

800 iterations without CUSUM, V=3, PPB = 0.42

Fig. 9 Effect of CUSUM filters on boundary tracking of 500x600 portion of San Francisco Bay image.