

# Multi-Point Boundary Tracking For Atomic Force Microscopy

*Annie I-An Chen, Jef Huang, Samuel Lim*  
*Faculty Advisor: Todd Wittman*  
*UCLA Department of Mathematics*  
*August 2009*

---

## ABSTRACT

This paper presents several approaches for efficient imaging on the atomic force microscope (AFM) using boundary tracking. In addition to the refinement and generalization of existing algorithms such as those of Andersson<sup>[1][2]</sup> and Chen<sup>[3]</sup>, a new multi-point model is proposed. In this model, variables are calculated according to a set of consecutive measurements rather than updated on a point-by-point basis. Simulations on MATLAB, as well as techniques and challenges for actual implementation on the AFM, are also discussed.

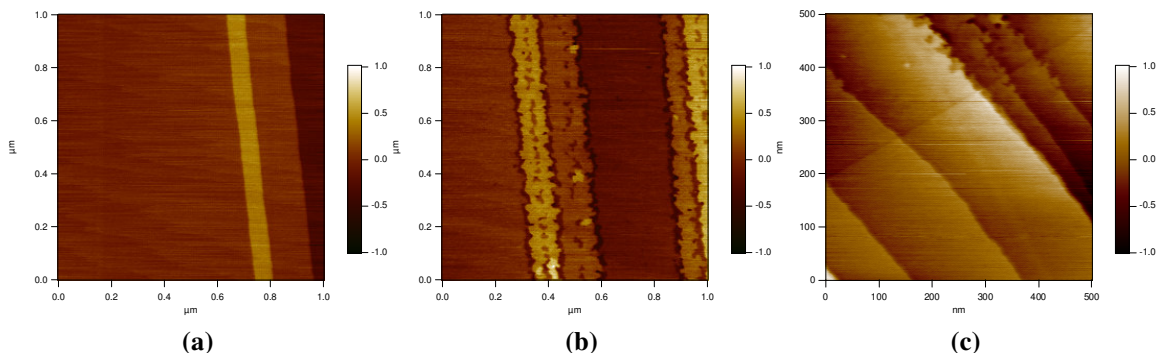
---

## I. INTRODUCTION

The atomic force microscopes (AFM) use a needle-cantilever system to locally interact with samples. This particular mechanism is capable of nano-scale imaging, beyond limitations of light/electron wavelengths.

The current method of AFM data-collection is the raster scan. As the controller moves the needle tip over the entire sample field in a preset path, a height or amplitude reading is returned point by point. Based on these readings, an image is formed. The user has some levels of control over the imaging process, including scan speed, image size, and resolution. Scan speed determines how fast the needle tip moves across the sample; image size determines the dimensions of the sample field to be imaged; resolution determines the dimensions of the returned image. The time required for obtaining a complete image is dependent on these three variables. A high scan speed allows the image to be completed more quickly, but speeds beyond certain thresholds effect the quality of the results. In some cases, the sample itself may be damaged by the needle's interaction. Similarly, a low resolution image can obtained relatively quickly, but the level of detail is compromised.

With the current method, there is a clear trade-off between quality and speed. The scanning of a detailed image is typically on the order of minutes. However, for certain applications, most of the information obtained by the raster scan is irrelevant. In fact, some scientific purposes require only data on the boundary of a sample. This is clearly demonstrated in the oxidation of potassium bromide, as shown in Figure 1. Thus, a reduction in data-collection time is possible by actively tracking specific features in the sample field.



**Figure 1. Oxidation of Potassium Bromide Obtained by Raster Scan**

(a) Image of static sample before oxidation. (b) Image of static sample after oxidation. (c) Image obtained during the oxidation process. Since the oxidation process occurs on the order of minutes, the raster scan is unable to capture accurate images of the moving sample.

Several point-by-point algorithms for boundary tracking exist, including those proposed by Andersson<sup>[1]</sup> and Chen<sup>[3]</sup>. However, these existing methods are not directly applicable to the AFM controller because it does not operate on a point-by-point basis. Rather, the controller takes an array of x and y voltages, which corresponds to positions on the sample field, and moves the needle to successive positions at a rate that is on the magnitude of kilohertz.

In this paper, we present multi-point boundary tracking algorithms for AFM imaging. We also performed simulations and used the results as bases for implementation on the AFM.

## II. PREVIOUS WORKS

### 1. Andersson's Smooth Trajectory (AST)

Some solutions to the problem of rapid AFM imaging for string-like samples have been proposed by Andersson, including local raster scanning with segments of straight lines<sup>[1]</sup> and local non-raster scanning with smooth, sinusoidal trajectories<sup>[2]</sup>. The latter, henceforth referred to as Andersson's Smooth-Trajectory Algorithm (AST), is more favorable because it minimizes sharp turns in the trajectory, thus reducing the chances of damaging the AFM needle. AST mounts sinusoidal waves on an estimated boundary

whose curvature can be viewed as a constant over a short arc length. The tip trajectory can be expressed as

$$x_{tip}(t) = x_d(s(t)) + A \sin(\omega s) q_2(s(t))$$

where  $x_{tip}$  is the needle tip trajectory,  $x_d$  is the estimated boundary,  $s(t)$  is the arc length as a function of time,  $A$  is the amplitude of the mounted sine wave,  $\omega$  is the spatial frequency of oscillation, and  $q_2$  is the unit vector normal to the estimated boundary at that point.

Each time the needle tip crosses the boundary, the crossing point is recorded as a “boundary point.” Two variables are updated accordingly:

- (1) The curvature of the estimated boundary, updated based on Heron’s formula
- (2) The tangent direction,  $q_1$ , of the estimated boundary, updated based on the two most recent crossing points.

As a result, each time the needle crosses the boundary,  $x_d$  and  $q_2$  are updated, and a new needle tip trajectory is calculated.

In order to maintain stability for the needle tip, the tip is moved at a constant speed, i.e.  $v_{tip} = \frac{dx_{tip}}{dt}$  is constant. Under this restriction, it is possible to solve for a unique relationship between  $s$  and  $t$  and thus determine the tip position with respect to time. The solution, however, is computationally expensive because it requires inverting a nontrivial integral. The suggestion was to do such calculations off-line and construct a look-up table, which takes up memory in exchange for runtime.

In our implementation and testing of AST with sinusoidal trajectories, we made two major modifications:

**(1) The tangent direction of the boundary is estimated with 3 points instead of 2.**

Since AST assumes the boundary to have constant curvature over the arc length in consideration (i.e. it can be approximated as a circle passing through the three previous boundary points), it appears more reasonable to us that the boundary should still be approximated by the same circle beyond the most recent boundary point. Therefore,  $q_1$  should be tangent to the circle (Figure 2.b), and not just tangent to the two most recent boundary points (Figure 2.a).

The advantage of this modification is its extension of the assumption for the sample: the modified model is not only valid for string-like sample, but also for samples with larger curvatures. In our experiments with curving boundaries, three-point estimation works significantly better than two-point estimation in that it tracks the boundaries more closely, and has less chances of losing of them.

This adjustment does not affect the performance of the original AST, which was designed for string-like samples. If the samples are string-like, its curvatures are small,

and the three most recent boundary points will be almost on the same line, yielding a tangent direction close to that calculated from 2 points.

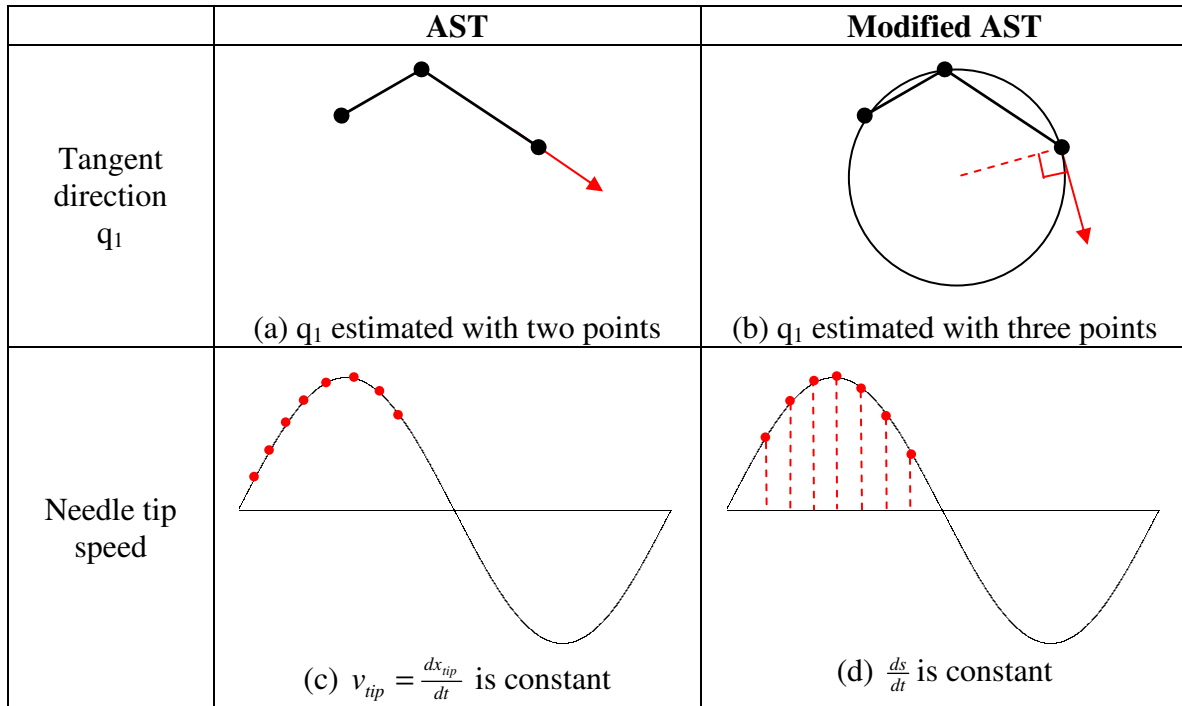


Figure 2. Comparison between the original and modified AST algorithms.

## (2) The needle speed is not constant.

In our model, we avoided the inversion of the integral by omitting the condition that the needle tip is moving at a constant speed (Figure 2.c). Instead, we assumed the projection of the needle trajectory onto the arc length of the boundary to be constant, i.e.  $\frac{ds}{dt}$  is constant. This yields a much more concise formula of the sine wave that is mounted (Figure 2.d).

There are two benefits for removing this constant-speed restriction. First of all, calculation is greatly simplified, and look-up tables are not needed because calculations are relatively trivial. In addition, the resulting trajectory of the needle tip travels slightly faster when it is near the boundary, and slower when it is away from the boundary turning back. This protects the needle by giving it more time when making a larger change in direction near the sine wave peaks.

However, there is a drawback to this adjustment: since the path consists is discretized, the distance between points would be greater for larger  $\omega$  (high-frequency sine waves), introducing a greater error when calculating the boundary point (Figure 3). The unit arc length and  $\omega$  have to be chosen carefully.

## 2. Chen's Circular Path (CCP)

Alex Chen proposed a very simple form of point-by-point boundary tracking<sup>[3]</sup> inspired by environmental boundary tracking. The basic algorithm has three main parameters:  $d$ , the turning direction;  $\omega$ , the turning angle; and  $v$ , the step size. The image is divided into two regions:  $\Omega_1$  and  $\Omega_2$ . The variable  $d$  is 1 in one region and -1 in the other. For each iteration, the trajectory turns by an angle of  $\omega d$ , and travels a distance  $v$ . Then, the algorithm checks for a boundary crossing. In the basic case of a black and white image, a boundary crossing has occurred if the value of the current point is on the opposite side of the given threshold than the previous point.

This particular method of crossing detection is not particularly useful when noise exists in the image. To that end, Chen's most significant contribution is the implementation of the CUSUM filter. This filter uses high-end and low-end thresholds to find boundary crossings. At each step, the filter adds the measurement of the current position to a cumulative sum. Once the sum reaches a certain threshold, a boundary crossing is identified. The use of the CUSUM filter greatly increases the algorithm's robustness to noise at the cost of delays.

Once a boundary crossing is identified,  $d$  is updated to be  $-d$ . In other words, the trajectory begins to head back towards the boundary once a crossing has been detected. As this particular algorithm tracks a boundary, the path taken is a series of semi-polygons. By setting the parameters  $v$  and  $\omega$  to be relatively small, the path resembles a series of semi-circles which weaves around the boundary. We will thus refer to this algorithm as the Chen's circular-path (CCP) algorithm henceforth.

## 3. Generalization

An underlying common framework can still be found between AST and CCP. In the context of discrete needle tip positions, CCP successively steers the needle tip, changing the direction by a fixed amount and moving it a constant distance. The original AST also makes constant-size steps, but varies the directions so that the resulting points fall on a sine wave mounted on a circle. Therefore, both can be viewed as algorithms for generating the direction angle as a function of time, while fixing the step size.

This led us to consider other angle functions, which result in various trajectory shapes. Under this general framework, we are no longer limited to segments of sinusoidal or circular paths. Various shapes were examined, including U-shapes, shapes that resemble ellipses, etc. We did not conclude on a particular shape that worked exceedingly well. However, it is worth noting that shapes longer in the normal direction than in the direction tangent to the boundary (i.e. "thinner" shapes) are denser and cross the boundary more frequently. On the other hand, "wider" shapes (i.e. shapes that travel

more in the tangent direction than in the normal direction) are able to move along the boundary faster.

### III. MULTI-POINT BOUNDARY TRACKING

AST and CCP are fundamentally point-by-point algorithms. It is impractical to apply such algorithms to the AFM due to the delay of its software-controller interface. There is an overhead introduced by sending commands from the software to the controller. According to our experiments with the Asylum MFP3D AFM, this overhead is approximately 25 milliseconds. This delay greatly hinders the performance of point-by-point boundary tracking methods. Sending multiple points at a time to the controller minimizes the effects of this communication delay. In the existing raster scan, multiple points corresponding to horizontal scan lines are sent to the controller each time, thus making the overhead almost negligible. Similarly, we wish to implement a multi-point method with boundary tracking.

Some complications exist when transitioning from point-by-point to multi-point boundary tracking. If multiple points are sent to the controller at once, the needle tip will move a relatively long distance before its trajectory is updated. Thus, if a boundary crossing occurs within this distance, correcting the trajectory based on the information of a boundary crossing is not possible until the end of that path. Since both AST and CCP update the trajectory immediately after a boundary crossing, they are not directly applicable to the multi-point problem. A simple solution would be moving the needle tip back to the crossing point before continuing with the updated path. However, doing so is not only inefficient, but may also damage the needle by forcing it to take a sharp turn. On the other hand, the final position in a multi-point array can potentially be far from the boundary. Continuing from that position without returning to the boundary point runs the risk of losing track of the boundary. An effective multi-point algorithm should be efficient in software-controller communication while maintaining the ability to track the boundary.

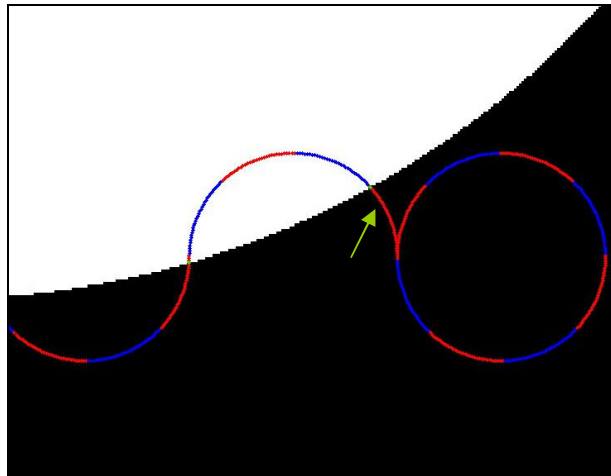
#### 1. Shark-Fin

One of our approaches uses a modified version of CCP. In this method, the key variables used are  $d$ , the heading direction;  $\bar{\omega}$ , a set of turning angles;  $v$ , the step size; and  $n$ , the number of points sent each iteration. The variable  $d$  takes on a value of 1 or -1, similar to CCP. Instead of using a fixed turning angle and moving one step, we calculate a set of  $n$  points at a time.

In these terms, the original CCP is the special case where  $\bar{\omega}$  is a set of angles equal in magnitude, and  $n$  is equal to 1. The simplest modification we can make is to change  $n$ , and send multiple points to the controller at once. In each iteration, a set of  $n$

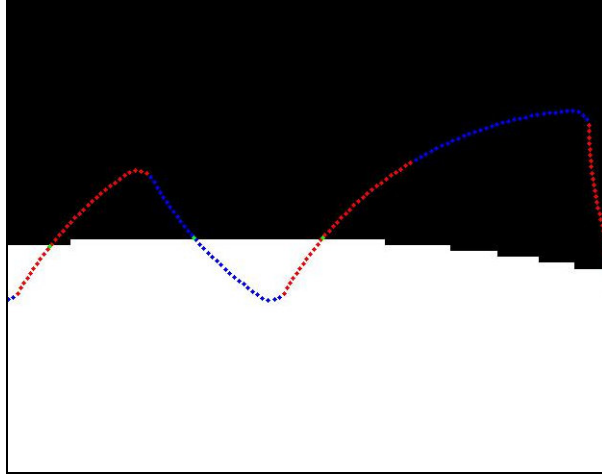
points is calculated by repeatedly moving a distance of  $v$  and turning by  $d$  multiplied by one element of  $\bar{\omega}$ . This entire set of points, which is equivalent to a segment of a circle, is then sent to the controller. Data corresponding to all  $n$  points is then collected. If a boundary crossing is discovered anywhere within the set of  $n$  measurements,  $d$  is updated to  $-d$ , so that the next set of angles will be “turned” the opposite direction. In ideal cases, the trajectory is expected to be a set of partial-circles moving along the boundary. Our initial testing was done by sending  $1/8$  of  $\bar{\omega}$  at a time, which corresponds to a  $\pi/4$  arc.

This particular method works if the boundary is relatively straight, but problems can easily arise depending on the location of a boundary crossing. In the case where a crossing happens near the beginning of a multi-point segment, the trajectory can possibly never return to the boundary, as depicted in Figure 4.



**Figure 4. The trajectory makes a full circle without returning to the boundary because a crossing occurred near the beginning of a segment (green arrow).**

Several improvements are possible, including sending smaller segments at a time, or moving back to the boundary point before continuing. These two solutions are not ideal, since the former would increase the scanning time and the latter can potentially damage the needle. A better solution is to make a series of relatively sharp turns at the iteration after a crossing. We can assume the worst case scenario to be one where the crossing occurs at the very beginning of a segment. In order to steer the trajectory back towards the boundary, a sharp turn is required at the end of that segment. If a particular segment corresponds to an arc of angle  $\alpha$ , making an additional turn of  $\alpha$  at the end of that segment sets the trajectory to a circle that passes through the first point of the segment. In fact, any  $\alpha' \geq \alpha$  achieves the same effect of bringing the trajectory back to the boundary. To avoid forcing the hypothetical needle tip to take a damaging sharp turn, the chosen angle  $\alpha'$  is evenly distributed amongst the first  $k$  points after then boundary-crossing segment (in other words, we add  $\alpha' / k$  to the first  $k$  points of  $\bar{\omega}$ ). According to our experiment using synthetic data, as shown in Figure 5, this modification successfully steers the needle tip back to the boundary in a trajectory shaped like shark fins.



**Figure 5. An additional turn of  $\alpha' = \frac{\pi}{2}$  is evenly distributed over the first  $k=5$  points of every segment following a crossing. In this example,  $\alpha = \frac{\pi}{4}$ .**

The arc angle  $\alpha$  should be chosen with careful consideration. Longer segments reduce the communication delay, but large  $\alpha$  prevents the trajectory from moving forward along the boundary.

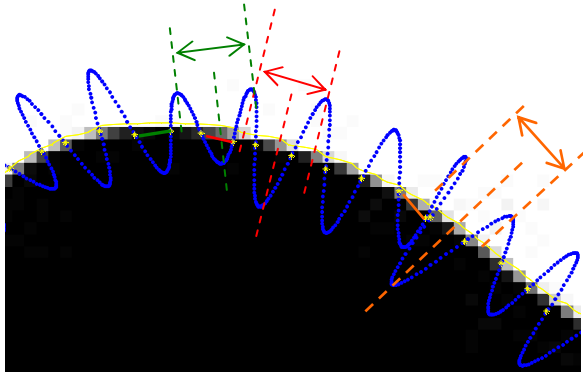
## 2. Half-Period

An alternative way is to generate a set of points that forms a longer segment on the trajectory. We considered points that form half periods of cosine waves, inspired by AST, and points that form half-periods of quarter-circles, inspired by CCP.

The major question in travelling longer segments at a time is where the starting and ending points of a segment are with respect to the boundary. Starting or stopping near the boundary is undesirable. Instead, we would like the points of crossing to be in the middle of the segments, so that each segment can be expected to contain exactly one boundary crossing. As a result, each segment starts and ends at points relatively far away from the boundary. The amplitude and direction of the trajectory should be calculated with this in mind.

Therefore, we chose to mount segments of half-period cosine waves (i.e.  $\cos \theta$ ,  $0 \leq \theta < \pi$ ). Its anticipated crossing point is in the middle of the half-period segment. The estimated boundary is the straight line linking the two most recent boundary points. Each new segment has an amplitude and a heading direction. The amplitude is the distance from the current needle tip position to the estimated boundary. The heading direction is parallel to the boundary line. To prevent the trajectory from reversing directions, the heading direction is only updated every other segment. In other words, the trajectory can be viewed as groups of two connected half-cosine waves with the same heading direction but different amplitudes.

The frequency of the cosine wave, which determines the oscillation intensity, is chosen according to the sample size. We also set a minimum value for the amplitude of the cosine waves. This prevents the amplitude from becoming arbitrarily small and lowers the possibility of losing the boundary.

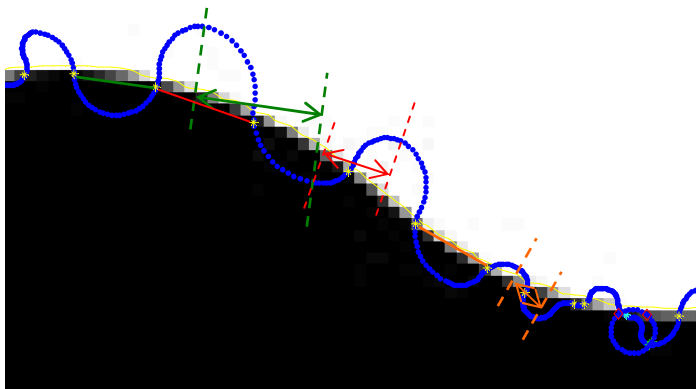


**Figure 6. Half-period cosine waves.**

The heading direction is updated every other half-period (solid lines), forming groups of two half-cosine waves (between dashed lines) with same heading directions but different amplitudes.

The frequency (or equivalently, the period, indicated by double-sided arrows) is a global variable that can be altered to control the intensity of the scan.

Another option is to use half-periods of quarter circles. Each segment consists of a pair of quarter circles concaving toward opposite directions. Again, the estimated boundary is the straight line connecting the two most recent boundary points. Each half-period has a radius, which is equal to the distance from the current tip position to the estimated boundary line, and a heading direction, which is parallel to the boundary line.



**Figure 7. Half-period waves of quarter-circles.**

For every half-period wave of two quarter circles (between dashed lines), if a new boundary point is found, the heading direction (solid lines) and the radius (half the length of double-sided arrows) is updated.

#### IV. IMPLEMENTATION OF ALGORITHMS

The multi-point tracking algorithms, as well as three initial boundary detection methods, are incorporated into a finite-state machine:

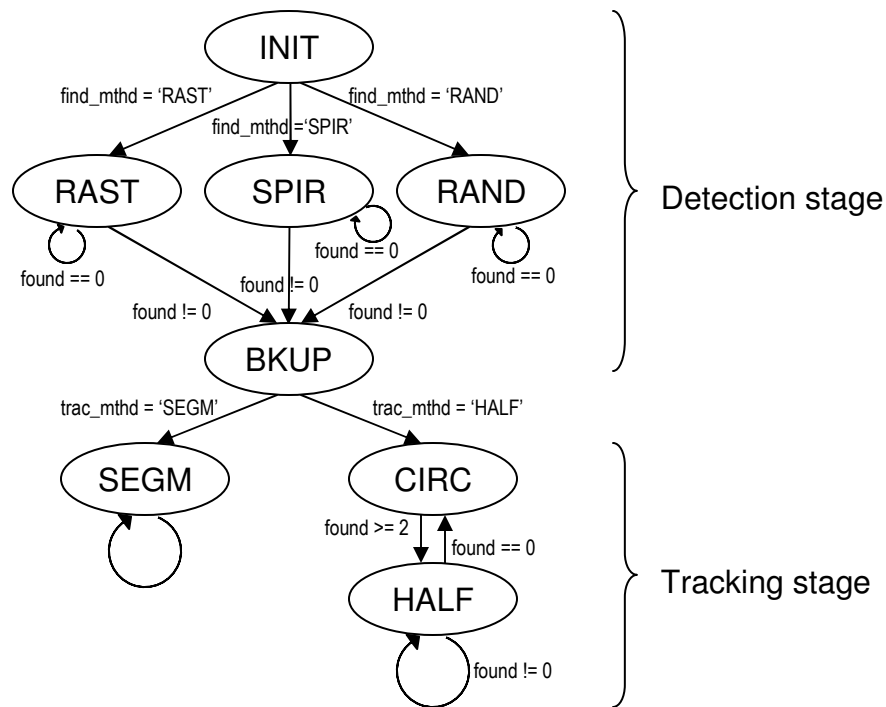


Figure 8. State graph of multi-point boundary detection and tracking.

State name	Description
<b>Finding stage:</b> locate one boundary point	
INIT	INITialization: calculate boundary threshold
RAST	RASTer: perform raster scan
SPIR	SPIRAl: perform spiral scan
RAND	RANDOm: form random lines across the field
BKUP	BacK-UP: travel in a straight line to the most recent boundary point
<b>Tracking stage:</b> scan around boundary	
SEGM	SEGMents: take measurements in increments of small segments
CIRC	CIRClE: draw a circle that encloses the most recent boundary point
HALF	HALF-periods: take measurements in increments of half-periods
Variable Name	Description
find_mthd	Choice of 3 method for finding: { 'RAST', 'SPIR', 'RAND' }
trac_mthd	Choice of 2 method for tracking: { 'SEGM', 'HALF' }
found	Number of boundary crossings in the previous set of measurements

Figure 9. Table of states and important variables

**INITialization:** In this state, the value corresponding to the boundary in the set of possible measurements is defined.

**RASTer scan:** This state performs a raster scan on the. The raster scan is a thorough but slow way of detection.

**SPIRAl scan:** This state sends sets of points in a spiral path. Given that the spiraling radius is not too sparse, this method is capable of finding the boundary more efficiently than the raster scan.

**RANDom search scan:** This state generates straight paths that turn by random angles upon reaching the border of the field.

**Back-UP:** When a boundary is found, the program enters this state. The needle point is steered back in a straight path to the point where the boundary was crossed.

**SEGMENTS:** This state performs the shark fin tracking algorithm.

**HALF-periods:** This state performs the half-period tracking algorithms with either half-cosine waves or pairs of quarter circles.

**CIRCLE:** Since half-period trajectories depend on a boundary estimated from the two most recent boundary points, at least two boundary points are needed before half-period tracking can start. Our solution is to steer the needle tip in a circle around the boundary point found in the detection stage. Since the circle encloses at least a point on a continuous boundary, it is guaranteed to cross the boundary at least twice while traversing its circumference (provided that the circle is not completely on one side of the boundary.) The boundary is then estimated by linking the two most recent boundary points.

This is also the correction state of the half-period methods: if no boundary points are found within a half-period segment, it is possible that the program has lost track of the boundary. Therefore, it comes back to this state in order to relocate the boundary. If no boundary points are found on the circle, the program returns to this state and moves the needle tip in a larger circle.

## **V. RESULTS**

### **1. MATLAB Simulation**

We created a MATLAB program with a graphic user interface. Most of our simulations were performed with this program. The results are shown in the following table:

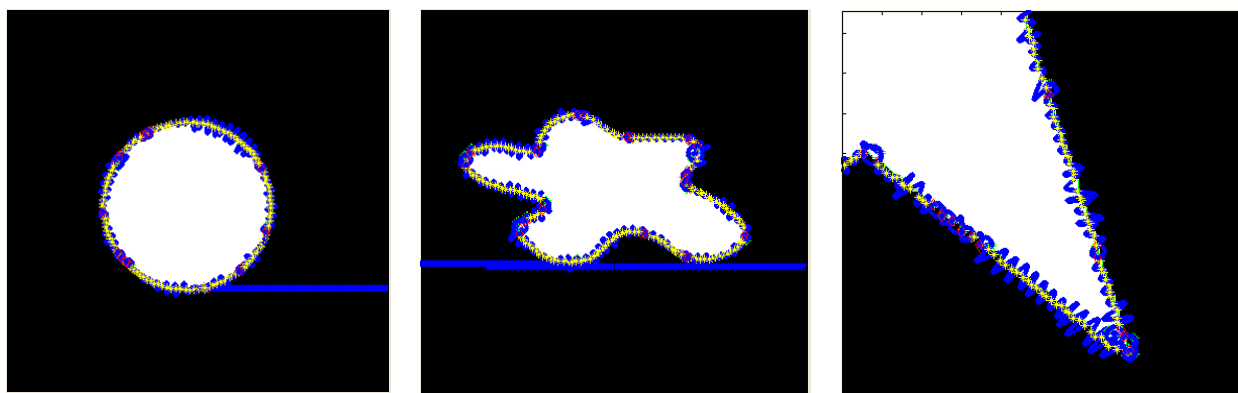


(a) 177 sends.

(b) 261 sends.

(c) 74 sends.

**Figure 10. Shark fin tracking algorithm.**  $V=0.1227$ , Circular Division = 8.

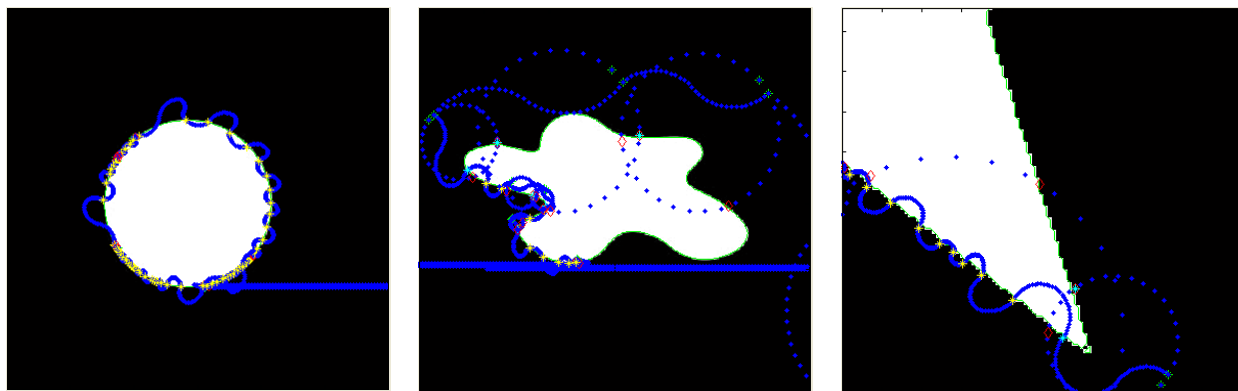


(a) 132 sends.

(b) 190 sends.

(c) 61 sends.

**Figure 11. Half-period tracking algorithm with half-cosine waves.** Min Amp=0.5, Freq=1.



(a) 102 sends

(b) Boundary tracking failed.

(c) Boundary tracking failed.

**Figure 12. Half-period tracking algorithm with quarter circles.** Min Amp=0.5, Freq=1.

For these samples, the half-period algorithm traverses the entire boundary in fewer iterations than the shark fin algorithm. However, the former requires several correction circles, while the latter does not lose track of the boundary as easily.

All of our testing with the shark fin method was done by traversing  $1/8$  of a circle per iteration. The results are rather promising: for a relatively smooth boundary using  $\alpha = \frac{\pi}{4}$ , there is typically one boundary detection per one or two segments.

Upon thorough testing, we conclude that half-periods of cosine waves are a better choice than quarter-circle waves, because we can control how sparse or dense we would like the scanning to be, whereas the quarter-circle trajectory becomes more sparse as the radius increases. The former yields satisfactory performance in tracking, while the latter loses track of the boundary easily if the boundary does not resemble a straight line.

## 2. AFM Testing

We had the privilege of working with an AFM in Lawrence Berkeley National Laboratory. The AFM is manufactured by Asylum Research. The company also provides interfacing software, written in Igor Pro.

During our time there, we were able to exercise basic needle control so that it performs the raster scan. However, due to time constraints and limited access to the AFM, we have not yet been able to implement and test other parts of our algorithm.

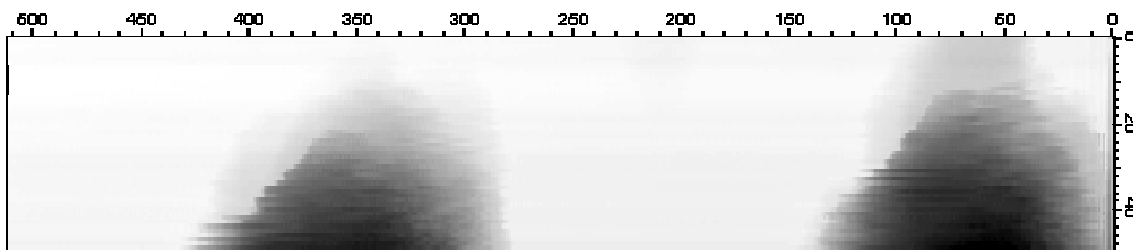


Figure . Resulting image of two plateaus, obtained by our implementation of the raster scan.

## VI. CONCLUSION

In the previous sections, we described the motivation for efficient AFM imaging. Existing algorithms were presented, along with reasons that make them inapplicable to the AFM. We then developed methods for boundary detection and tracking that allows for multi-point software-controller communication. Simulation on MATLAB and implementation on the AFM were also discussed.

## VII. ACKNOWLEDGEMENTS

The authors would like to thank Todd Wittman and Andrea Bertozzi for devoting so much time and interest in our project. We would also like to thank Paul Ashby for his kind assistance on the AFM at the Lawrence Berkeley National Laboratory.

## VIII. REFERENCES

- [1] S. B. Andersson, “**Curve tracking for rapid imaging in AFM,**” IEEE Transactions on Nanobioscience, vol. 6, no. 4, pp. 354–361, 2007.
- [2] P. I. Chang and S. B. Andersson, “**Smooth trajectories for imaging string-like samples in AFM: A preliminary study,**” in Proc American Control Conference, 2008, pp.3207.
- [3] Alex Chen, “**Image Segmentation Through Efficient Boundary Sampling,**” UCLA CAM preprint, 2009.