2) The subtlety in using angles, and specifically arctangent, is that the atan function will return a value between $\pi/2$ and $-\pi/2$. Thus, a line curving counterclockwise might transition suddenly from and angle of $\pi/2 - \varepsilon$ to $-\pi/2$ instead of to $\pi/2 + \varepsilon$. Because of this, the curvature, as the derivative of the angle, will spike.

3) I define "nearly coincident" as being two corners within 200px of one another. This minimum distance was determined empirically, and seems to prevent small deviations within the "slow" range near a corner from causing additional edges.

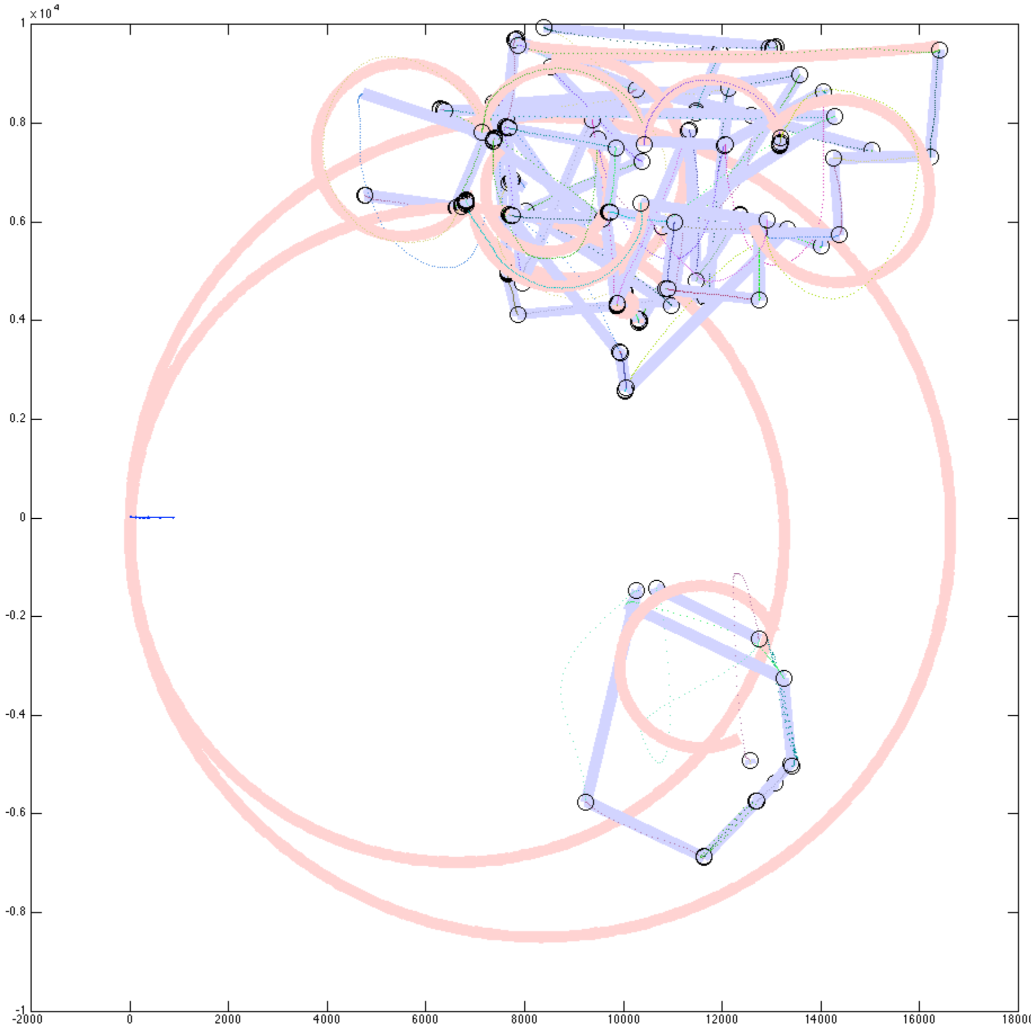
6) The most major changes to the parameters were simply boosting the minimum distance, and adding a threshold distance away from the ends of the stroke for another corner. People are noisy and wobbly, and especially so near the start and end of a stroke, when they are slowing down, speeding up, and deciding what to do. Dropping nearby corners removed many edges that seem unintentional.

```
speedSmoothingWindow = 6;           % points (starts at 2 on each side)
tangentWindow = 10;                 % points to regress to find tangent
firstSpeedThresholdPercent = .25;   % percent of average speed
curvatureThreshold = .75;           % degrees per pixel
secondSpeedThresholdPercent = .8;   % percent of average speed
minimumCornerDistance = 200;        % distance between allowable corners
minimumArcAngle = 36;               % degrees for loosest arc

minimumStartDistance = 500;
```

7) My system is an unfinished implementation of this paper. While correctAngleCurve.m appears to be simply unnecessary given the other constraints I've included in my system, a fully-implemented selection of curve type would improve the results. After that, a learning system that tweaked the parameters based on user behavior might further refine the behavior.

evalAll - default parameters



evalAll - optimized parameters