# Handwriting Capture Exercises

Use any programming environment you feel suitable for these tasks. Matlab/Octave/SciLab is one choice as it has callable primitives to capture stroke coordinate sequences as a matrix. There are API’s for MS Windows and Macintosh OSX that can be used from C/C++. Java, Python, ….

1. Use the MatLab program “get\_pencil\_curve.m” as a starting point and create “stylusPoints” which captures the sequence of (x,y) coordinates from hand written text and draws the written path in a window.
   1. Don’t quit on penUp/penDown events add color attribute (B-W) to captured cursor locations and continue to trace both inked and uninked moves.
   2. Make the procedure “callable” and have it pass the continuous stream of coordinates to the calling routine.
2. Write a callable routine “strokes” that processes the output from stylus-points and returns the cursor location sequence as a sequence of stroke matrices containing cursor locations between “critical points”
   1. Change in color (penup/pendown)
   2. Minimum in distance moved between cursor locations (of smoothed cursor sequence to avoid false minima). This identifies points of maximum curvature since the stylus must obey Newton’s laws of motion.
3. Compress the data stream into a smaller space. One method is to “Vector Quantize” each stroke as described below, but feel free to come up with alternative stroke compression schemes.
   1. Write “vectors” which:
      1. Accepts the stroke sequence of cursor locations within strokes.
      2. Compresses[[1]](#endnote-1) each stroke into a short sequence of vector moves that trace the stroke minimizing both “mean-square-error” and number of vectors (Note that there is a trade-off here, parameterize weighting of error vs number of vectors and experiment to find a good compromise) using a fixed vocabulary of vectors:
         1. 256 (8 bits) equally spaced angles
         2. 128 (7 bits) 2k lengths (more than needed, but allows for scaling)
         3. 2 colors (B, W – 1 bit)
      3. Outputs the vector sequence per stroke representing the written path.
4. Write “drawPath” which draws a smoothed set of strokes in a window from the vector sequences for comparison with the original path of cursor locations.

1. I favor a “beam search” algorithm. Start with the best k vectors from the initial point. Then extend each to a sequence of 2 vectors (yielding k2 sequences), then prune back to the best k sequences and repeat until reaching the end of the stroke. This is a space and time limited search that finds a good solution, but not the “optimal” solution. [↑](#endnote-ref-1)