6.835 MiniProject 1: Early Sketch Understanding – Stroke Segmentation

Due: 5:00PM Friday, February 21nd, 2020

This exercise is intended to give you some hands-on experience in dealing with pen-based computing. We will supply real data (pen strokes) and some basic Python code as a foundation. The exercise then involves implementing the corner finding approach in the attached paper by Stahovich [1].

As the paper is at times unclear on the approach it actually uses, we have outlined it for you. Nevertheless, you should read the paper carefully, and may discover that it is useful to refer to it as you work on this project, in order to understand some of the details.

You will no doubt discover that it is not always easy to figure out what the system described in the paper actually did; sadly this is typical of papers in the research literature. As a result it's useful to learn how to dig out the details: if you want to do research, you need to be able to at least reproduce other people's results. Even if you simply want to use an idea contained in a paper, you have to be able to decipher it in detail. Sometimes this is more work than it should be, but knowing how to do this is an important skill.

Things to note about this project:

- It will likely take you longer than you expect, so start soon. Don't put it off, or you will not get it done in time. Please take advantage of Piazza and office hours to get your questions answered.
- We expect you to implement what's outlined here, not to reproduce everything described in the papers. If you get inspired you are of course welcome to go beyond what we outline, but make sure you do at least what is listed below. In some parts of the assignment, we have simplified the procedures from the paper.
- This exercise is about getting down to the details of dealing with real pen data. You will confront a collection of issues and details that must be dealt with if you want to make sense of pen input, and get to see how this actually works.

1 Before You Start

This project requires coding with Python 3.7. If you are not familiar with Python, you may not be ready for this course. Consider getting more experience and taking this course next year. If you simply need a review of the language, refer to the numerous tutorials available online.

2 Getting Started

Download the project file package (`MiniProject1.zip`) from the course web page. The folder contains some Python support code and the dataset `strokes_data.py`.

2.1 Setup

Using the command prompt, create a virtual machine to hold the dependencies of the mini project. Note: A virtual environment is not necessary, but is a good habit for developers to follow. On a Mac, type the following in Terminal:

```
cd MiniProject1/
virtualenv venv -p python3
source venv/bin/activate
```

For Windows, use these commands[1]:

```
cd MiniProject1/
mkvirtualenv venv -p python3
```

Now the virtual machine should be up and running (you should see the `(venv)` at the front of the terminal line). Next, install the dependencies using Pip:

```
pip install -r requirements.txt
```

2.2 Starter Code

A `Stroke` object (defined in `Stroke.py`) contains arrays x,y,and t (time), representing the position and timestamp of each consecutive sampled datapoint in the stroke. You can visualize all 12 strokes by running `python eval_all_strokes.py` from your terminal. This script will create Stroke objects, send them to be segmented (using the segmentation function you will create), and then graph the raw stroke and segments. Right now, stroke segmentation is unimplemented, so you should see all the strokes as they appear in Figure 1. You can

[1] See this tutorial for additional info: http://timmyreilly.azurewebsites.net/python-pip-virtualenv-installation-on-windows/

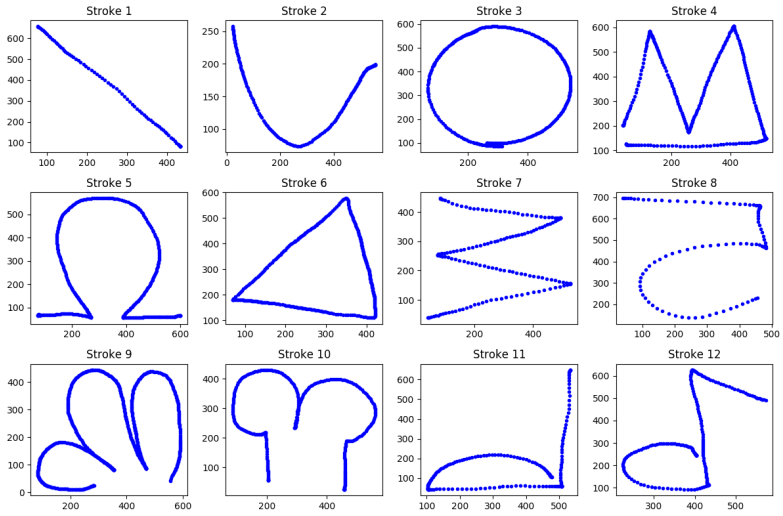

Figure 1: The 12 strokes you will segment

also visualize individual strokes using the command `python eval_stroke.py <stroke_number>`, where the stroke number is a number between 1 and 12.

In Part 3, you will modify only the following file for this assignment: `stroke_segmentation.py`. In Part 4, you will modify only the following file for this assignment: `classify_strokes.py`. Please DO NOT modify any other file provided to you. In `stroke_segmentation.py`, you will find the following:

- A list of parameters. You will need to change the values of these parameters as you write your code. Do not change the names of the parameters.
- Unimplemented functions. In this assignment, you will fill out these functions. Each function corresponds to a part of the implementation outlined below. The arguments and return values are defined for you in the functions' annotations. Do not change the functions' inputs or outputs (there is an exception for the optional extra credit `merge` function, described later in more detail). While all of these functions must be implemented, you are also free to create additional helper functions if you find it useful to do so.
- The `segment_stroke` function. This function ties together all of the functions you will implement. It is the function called by `eval_all_strokes.py` and eval_stroke.py to display your segmented strokes. The `segment_stroke` function is implemented for you, and you should not need to modify it. However, if you are completing the optional extra credit, you may need to make additions to the `segment_stroke` function. Do not modify the function arguments or returned values.

The `segment_stroke` function has the following structure:

```
def segment_stroke(stroke):
    ...
    returns segpoints, segtypes
```

- `stroke` is a stroke data structure represented by the Stroke class.

- `segpoints` is a length N array of the indices of points at which the `stroke` is segmented, including the start/endpoints of the stroke.
- `segtypes` is a length N-1 binary array indicating the type of each segment: 0 for a line, and 1 for an arc.

3 Segmenting Strokes

The main part of the assignment is to identify corners and classify the segments between corners as lines or arcs. There are a number of parameters that will have to be adjusted to improve performance, so be sure to devote time to reasoning through the parameters you choose. Using the Stahovich paper as a guide, your stroke segmentation should follow the steps outlined in the parts below.

1. *Complete the* `compute_cumulative_arc_lengths` *function.* Construct an array of the cumulative arc lengths between each pair of consecutive sampled points in the stroke.
2. *Complete the* `compute_smoothed_pen_speeds` *function.* Construct an array of smoothed pen speeds at each point. The size of the window over which smoothing occurs is one of the parameters you will need to adjust (parameter `PEN_SMOOTHING_WINDOW`).
3. *Complete the* `compute_tangents` *function.* Construct an array of tangents. To do this, fit a linear regression line to a window of points surrounding each point, and note the slopes. The size of this window is one of the parameters you will need to adjust[2] (parameter `TANGENT_WINDOW`).
4. Define curvature, which is the change in orientation (angle) over change in position along the arc length.
   (a) *Complete the* `compute_angles` *function.* Convert the slopes to angles using the arc tangent function (`atan`).
   (b) *Complete the* `plot_angles` *function.* To see a problem that crops up with using arc tangent (and with using angles in general), plot the arctangent of the slopes that you obtained for stroke 5 – the "Ω" symbol, and briefly explain what you think is going on. As this phenomenon this will corrupt your estimates of the curvature (change in angle over change in position along the arc length), they will have to be corrected.
   (c) *Complete the* `correct_angles` *function.* Write and test a method to correct the phenomenon you found in (b). Use your `plot_angles` function to make sure that the phenomenon was corrected as you expected.

[2]Numpy provides `np.linalg.lstsq()`, a built-in function for performing least-squares linear regression.

(d) *Complete the* `compute_curvatures` *function.* Finally, compute the array of curvatures by again using a least squares line fit as described in the paper. You will need to adjust the `CURVATURE_WINDOW` parameter. Hint: Be careful when assigning curvature to consecutive overlapping points (e.g. points with the same position at consecutive times).

5. Identify corners. The paper is a little unclear on this, but the basic idea is to have two separate criteria, both of which contribute points:

   (a) *Complete the* `compute_corners_using_speed_alone` *function.* Using speed alone, select local minima of speed that are lower than a threshold percentage of the average speed (`SPEED_THRESHOLD_1`)

   (b) *Complete the* `compute_corners_using_curvature_and_speed` *function.* Identify local maxima of curvature, and accept them if the speed at that point is below a threshold (`SPEED_THRESHOLD_2`), even if the point is not a speed minimum, and the curvature at the point is above a threshold (`CURVATURE_THRESHOLD`). Note that the `CURVATURE_THRESHOLD` parameter is given in degrees per pixel, but `atan` returns values in radians.

   You may find the `detect_peaks` function helpful for this part of the assignment.

6. *Complete the* `combine_corners` *function.* Combine the two sets of points identified in step 5, with any nearly coincident points merged. You will need to decide what "nearly coincident" should mean, and decide which of the two points to keep. Use the `MINIMUM_DISTANCE_BETWEEN_CORNERS` parameter to adjust the distance metric for nearly coincident points.

7. You should now have identified a set of segmentation points. You will now classify each segment as either a line or a curve. (For this assignment, you do not need to consider elliptical curves, only circular curves). For each segment:

   (a) *Complete the* `compute_linear_error` *and* `compute_circular_error` *functions.* Fit a line using linear regression, and fit a circle using the provided code, `circle_fit.py`. Then, calculate the residual errors for the line fit and the circle fit.

   (b) *Complete the* `compute_subtended_angle` *function.* Find the angle subtended by the arc of the circle.

   (c) *Complete the* `choose_segment_type` *function.* Based on the residual errors found in (a) and the angle subtended by the arc of the circle found in (b), classify each segment as a line or curve (0 for a line, and 1 for an arc). For this part, you will need to modify the `MINIMUM_ARC_ANGLE` parameter.

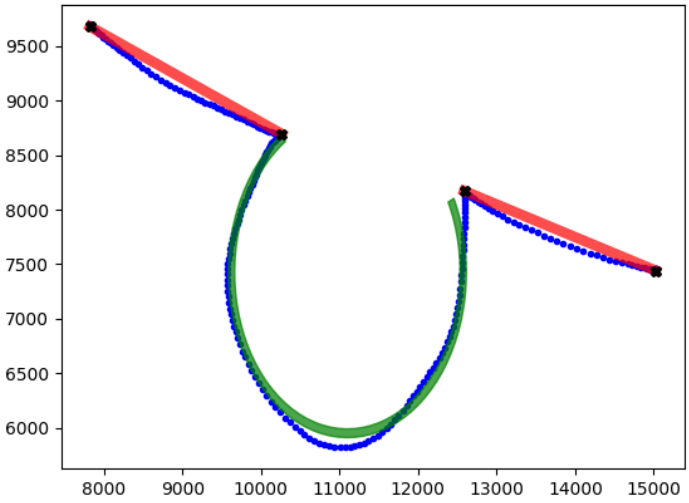

Figure 2: An example system output (not obtained using default parameters)

8. You can view the output on all given shapes in the dataset using `eval_all_strokes`, or on a single shape using `eval_stroke`, which will both call the `segment_stroke` function. Run these scripts from the command line:
   `python eval_all_strokes.py`
   `python eval_stroke.py <stroke number>`
   In this viewer, segments classified as lines are rendered in red, and segments classified as arcs are rendered in green. Corners are indicated with black x's. An example of reasonable system output is shown in Figure 2.

9. In your code, you should have at least 8 parameters:

Parameter	**Value used in the paper**
Size of window for smoothing penspeed	5 total (2 points on each side)
Size of window for computing tangent	11
Size of window for computing curvature	11
Speed threshold 1	25% of average
Curvature threshold	.75 degree / pixel
Speed threshold 2	80% of average
Minimum allowed distance between two corners	unspecified
Minimum arc angle	36 degrees

   Using these values, evaluate your system and print out the results with `eval_all_strokes.py`. Experiment with different values for these parameters, and print out the best results you can obtain, indicating the parameter values that you have chosen.

10. **[Extra Credit - Optional]** *Complete the* `merge` *function.* Now, we will take our segmentation one step further and attempt to Merge adjacent segments. Do so according to the following rules:

    (a) If a segment is shorter than 20% of the length (`MERGE_LENGTH_THRESHOLD`) of its adjacent segment, or, if adjacent segments are of the same type, try to merge them

(b) If error of fit of new segment is $< 10\%$ (`MERGE_FIT_ERROR_THRESHOLD`) of the sum of fit errors of the original 2 segments, use the new segment

You are free to change these percentages to make your system optimal. In order to make the `merge` function work, you may need to change the arguments of the function. Be sure to annotate your function and change the call the the `merge` function within `segment_stroke`.

4 Classifying Strokes via Template Matching

Now that you have segmented each stroke into lines and arcs, you can now represent each stroke in a reduced manner by using the coordinates of the segment endpoints and the curve midpoints.

We can now perform symbol recognition and classify the strokes using template-matching methods similar to those described in *An image-based, trainable symbol recognizer for hand-drawn sketches* [2]. You can read this paper in the provided `TemplateMatching.pdf`. You will build on the stroke segmentation from Part 3 to perform a modified template-matching based recognizer.

In `template_data.py` you will find symbol templates. A `Template` object (defined in `Template.py`) contains a string *name* describing the corresponing symbol, and arrays x, and y, representing the set of normalized x and y coordinates of line and curve endpoints, plus the midpoint of each curve.

Your job is to perform stroke recognition by matching each stroke to the correct symbol class template. You will implement the functions in `classify_strokes.py`. Please do not modify any additional files.

You can evaluate these functions by running `evaluate_all_strokes.py` and `eval_stroke.py`. The classification output should print out green if correct, and red if incorrect.

1. *Complete the* `normalize_segpoints` *function.* Given a `Stroke` object, return the normalized x and y coordinates of each distinct relevant point that will be used for template matching. Relevant points would be the set of line endpoints, curve endpoints, and the curve segment midpoints of the segmented stroke. We want to normalize strokes by scaling the points, so that the each strokes' minimum x and y value is at 0, and the maximum x and y value is at 1. That way, they are at the same scale as the given templates.

2. *Complete the* `calculate_MHD` *function to calculate the Modified Hausdorf Distance (MHD).* The formula for the Modified Hausdorf Distance is outlined in Section 4.2 the paper *An image-based, trainable symbol recognizer for hand-drawn sketches*. This assignment is a modified version of the paper.

3. *Complete the* `classify_stroke` *function*, where given a stroke and list of `Template` objects, you will return the *name* of the template that has the minimal MHD.

5 Submission

Please submit the following in a zipped file to the course website in the Homework section:

1. A zipped folder containing your source code. Please submit all the files used to run the code in this project. Do NOT include your `venv` folder.
2. A color image of the results of your system with default parameters.
3. A color image of the results of the best parameter values you could identify.
4. A PDF containing the following:
   (a) The answer to the question in step 4b, regarding the subtlety of arc tangent, and why this will corrupt the estimates of the curvature.
   (b) Your definition of "nearly coincident", and how you chose the value for the parameter (*i.e.* the minimum allowed distance between two corners).
   (c) Compare the results of your chosen parameter values with those of the default. If yours turn out to be better, explain what you did, and why it works better. Be sure to include the values of the best parameters you found (corresponding to the image of your best results). If you changed your parameters to use radians instead of degrees, also specify this in your writeup.
   (d) The papers describe several additional steps to improve this system. Which of these (or what other steps) would be the most effective in improving your system as it now stands?
   (e) What advantages and disadvantages does the MHD-based template matching sketch recognizing system provide? How does this compare to other sketch recognition systems?
   (f) If you implemented the extra credit or any other extensions, please describe what you did and your results in your writeup. Include any relevant pictures.

References

[1] Stahovich, Segmentation of pen strokes using pen speed. In *AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural.*
[2] Kara, Stahovich, An image-based, trainable symbol recognizer for hand-drawn sketches, 2005.