Overview: Several members of the UR math and cs faculties are working on a project that is developing diagnostic tools for minor traumatic brain injuries (concussions) using non-invasive testing strategies. One type of test we are analyzing is eye tracking data. Subjects track a target moving on a computer screen while an eye tracking apparatus records the gaze position, eye velocity and acceleration. From this data, a researcher can reconstruct the gaze track and compare it to the target track the subject was meant to follow. Various measures can be calculated that determine how well the subject's gaze matched the target's movements.

Eye movements, called *saccades*, are classified as either *primary*, meaning that they are in reaction to the movement of a target, or `corrective`, meaning that they follow a primary saccade and refine the position of the eye with respect to the target. There is a tendency to overshoot or undershoot the target on the primary saccade. Most saccade detection algorithms use the velocity of the eye to distinguish between fixations (period where the eye is at rest) and saccades. There are more sophisticated techniques, but our experience has been that just looking a velocity is sufficient for detecting saccades.

Quantities of interest are the amplitude of the saccade (how far did the eye move), the peak velocity, the onset time, completion time, and duration.

You can find a good, brief explanation of eye movement concepts here: `https://www.liverpool.ac.uk/~pcknox/teaching/Eymovs/params.htm`

You will have access to several anonymized files with data from these tests, and will produce a program that parses the files, calculates several measures on the data, and gives a visual representation of your findings. All of the data files are from horizontal step tests, that is, steps where the target maintained the same vertical position and movements were only from side to side, in discrete jumps.

Requirements:

1. You must supply a makefile with your project which builds the executable when "make" is typed in the source directory.
2. Each member of your team should be responsible for at least one class in your design. Responsibility should be clearly indicated in the source files.
3. Your application should allow the user select a data file to load for processing. See below for information on the file format.
4. Your program should be able to visually display the data contained in a data file similar to Figure 1. A full-sized .png of this figure is included in the tarball of supporting files for this assignment, since it it a bit hard to read here. It is fine for you to use open source libraries for plotting for this portion of the application. Because data exists for both eyes and both 2D dimensions, your program should allow the user to choose which eye to display, and which dimension, x or y. The first component of this graph is a line plot of the position of the selected eye in the selected dimension versus time. The position of the target for the same coordinate is also displayed in this plot. The units for the x axis are time, the units for the y axis are degrees (more on this later). The second component of the graph is the velocity of the selected eye in the selected dimension versus time. The units for the y axis are degrees/sec.
5. You should identify each saccade in the data file, and classify it as either primary or corrective. Collect the following information for each saccade:
    - time of onset,

- time peak velocity occurs,
- peak velocity,
- amplitude (difference between starting eye position and ending eye position), and
- time of completion.

You can calculate duration from this data as well.

6. You should develop your own measure for how closely the gaze track matches the target track. This could use information like how well the amplitude of the saccade matches the amplitude of the corresponding target movement, how close the final eye position matches the corresponding target position, whether there are corrective saccades, and what their magnitude is, and so on.

7. Your program should display a summary of the saccades that appear in the data in a dialog box. The format is up to you, but the display should include:

   - The name of the file containing the data.
   - The number of primary and corrective saccades present in the data.
   - The number of target movements present in the data.
   - The value of your measure of how well the gaze track matched the target track for this subject.
   - A list of all the saccades detected, with the the following information for each saccade:
     - Saccade number
     - Type (1 = primary, 2 = corrective)
     - Time in seconds of the peak velocity for the saccade
     - Time in seconds of the onset of the saccade
     - Time in seconds of the completion of the saccade
     - The amplitude of the saccade (difference between the start position and end position of the eye in the selected dimension, in degrees)
     - The peak velocity in degrees per second.
     - The duration in milliseconds.

     A header line for the tabular information would be nice. It should be possible to save this data to a file.
   - Your code should be carefully documented. There should be header comments for each class in the .h file that describe the general purpose of the class. Each method should have a comment along the lines of the Javadoc comments you are used to writing, with a description of the purpose of the method, the parameters required, and the return value. In fact, I don't mind if you just use the Javadoc format. We just won't be able to generate the HTML version from the comments. These should also be in the .h file, and you don't have to duplicate them in the .cpp file. Comments within methods describing how they work should also be provided.

8. You must supply a writeup describing your work. The writeup should contain:

   - A description of how to use your program.
   - A description of your measure for how well the subject's gaze tracked the target
   - A description of your saccade detection and classification schemes.
   - A description of any outstanding issues in your code.

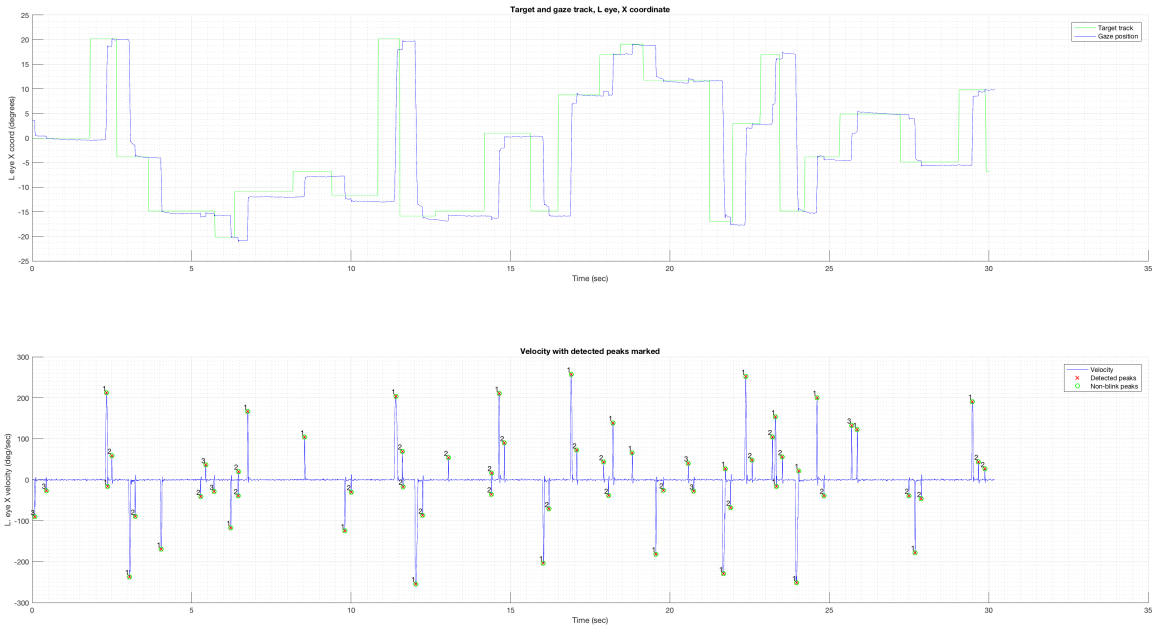

Figure 1: Display of an example data file.

Data file format: The data file is plain text. The file begins with a number of lines of preamble information. The only important information in the preamble is on the EVENTS line. The decimal number following the label "RATE" on this line is the number of samples per second, which could be important to some of the things you need to do. The rest can be ignored.

The actual test data begins with a line labeled "START". This line is followed by several lines starting with textual labels that we are not very concerned about. Lines beginning with a decimal number are samples. The test data ends with a line labeled "END".

Between the "START" and "END" lines, you will need to pay attention to the lines beginning with numbers and lines that begin with "MSG". The lines beginning with numbers are actual data samples, and they have the following format:

timestamp leye_x leye_y l_pupil reye_x reye_y r_pupil leye_xvel leye_yvel reye_xvel reye_yvel xres yres .

Where:

- timestamp is the time in milliseconds since the monitoring computer was started.
- leye_x is the x coordinate of the left eye gaze position in pixels.
- leye_y is the y coordinate of the left eye gaze position in pixels.
- l_pupil is the diameter of the left pupil in millimeters.
- reye_x is the x coordinate of the right eye gaze position in pixels.
- reye_y is the y coordinate of the right eye gaze position in pixels.
- r_pupil is the diameter of the right pupil in millimeters.

- leye_xvel is the x component of the left eye's velocity in degrees per second.
- leye_yvel is the y component of the left eye's velocity in degrees per second.
- reye_xvel is the x component of the right eye's velocity in degrees per second.
- reye_yvel is the y component of the right eye's velocity in degrees per second.
- xres is the resolution for the x dimension in pixels per degree.
- yres is the resolution for the y dimension in pixels per degree
- . is the end of record marker.

Note that you will need to normalize the position data to be in terms of degrees offset from the center of the screen. The screen is 1920 x 1200 pixels, and the original coordinate system for the position data has the origin at the lower left corner of the screen. The normalization is a two step process: first, transform the data so that the origin of the coordinate system is at the center of the screen (subtract 1920/2 from the x-coordinates, 1200/2 from the y-coordinates), then, divide by the appropriate resolution. Note that the resolution is computed by the EyeLink system, and tends to drift over the course of a session. It is a good idea to use the average of the resolutions for this final step, rather than using the value recorded with the sample.

Some samples are data from blinks, indicated by pupil size of 0. It is usually best to simply ignore or delete these samples. Experience has shown that there can be spurious eye movements on either side of a blink, so it makes sense to delete some number of samples on either side of the blink as well. Start with 50 samples on either side and work your way up if that does not seem sufficient.

The other lines to pay attention to are lines beginning with the string "MSG" and containing the string "TARGET_POS". From these lines, you will need the timestamp and the parenthesized x and y coordinates for the new target position on the screen. You will need to apply the same transformation to this position data to normalize it.

You will see that the velocity data is fairly noisy. It would be especially difficult to detect peaks from the raw data, so it is likely that you will want to apply some sort of smoothing to the velocity data. A simple moving average will probably be sufficient, but if you want to try more sophisticated methods, please do. I am fine for you to use open source libraries for this portion of your program as well.

My Matlab code for saccade detection detects peaks and then searches forward and back from the peaks to determine start and end times for saccades, and thus their duration. This is definitely not the only way to approach the problem. Others have worked from the other direction, detecting fixations and then classifying anything else as a saccade.

Submitting: Package all files required to build your system, including your makefile, along with your writeup in a tarball named `cmsc240_proj3_`*team*`.tgz` and upload it to the CS240_inbox shared folder on Box.