Professor Fearing EECS192/Assignment #1 v1.01 Spring 2019

Due at beginning of class Tues Mar. 19.

(One assignment per team.)

The purpose of this assignment is to allow you to prototype track finding algorithms with a high level language before implementing on the BeagleBoneBlue. You are free to use signal processing libraries, floating point, etc. for initial prototyping. *Use Python 2.7 for compatibility.* However, remember that your line finding algorithm will need to work in pseudo-real-time on BeagleBoneBlue. The course instructors have not done any timing of python, numpy, OpenCV etc. on BeagleBoneBlue, so you will want to do some timing tests before planning a steering control loop around this line detection algorithm.

Typical strategies to use for finding the track include:

1. Frame subtraction and peak detection
2. smoothing followed by gradient detection (e.g. difference of Gaussians approximation to the Laplacian)
3. curve fitting, e.g. cubic spline or $\tanh^{-1}$.

A set of line scan data from EECS192 2016 Team 1: Fast and Curious is provided on Piazza for EE192 under "Resources", `natcar2016_team1.csv`. A Python template is provided `linescanplotsHW1.py` which will read the csv file (can easily be modified to read telemetry file by adding extra columns) and plot line camera data and velocity. The actual car run can be seen at `https://www.youtube.com/watch?v=_AMs9iixp5M`. A bit more than a complete lap is given in data, as you can see the steps right after the start line. This data has been pre-processed to control illumination. Note that the time sampling is not uniform as darker areas use longer exposure (this is not necessarily the best strategy). Also, the velocity data is quite noisy.

Complete the function `find_track(linescans)` which takes as input n frames of linescans of 128 values in the range 0...255 and returns:

[5 pts] a) `track_center_list` which is a length n list of the index in the range 0...127 corresponding to the center of the track in each frame. (`np.argmax()` is used as a starting point, but it is not at all robust.)

[2 pts] b) `track_found_list` which is a length n list of {10,100} , where 100=True and 10 = False if the track is visible for a particular frame.

[2 pts] c) `Cross_found_list` which is a length n list of {10,100}, **True** where 100=True and 10 = False if a crossing is present for a particular frame.

[1 pt] d) On BeagleBone use Linux, e.g. `time python linescanplotsHW1.py` to estimate average time to process a scan. Of course, comment out plotting code. (If using a different processor, report average time on laptop or your embedded processor).

Upload your completed Python code (it should generate the required plots when run) to bcourses, as well as .png plot of track center, track_found, cross_found as provided in `linescanplotsHW1.py`. Note avergae time per line scan. Your python function will be tested against another data set taken under similar conditions on a similar track.

The line scan data should be interpreted as given in the following table. For example, around scan 380, the track is out of frame, so `track_found` would be False.

Input track?	Input crossing?	output track found	output cross found
F	F	F	F
F	T	F	T
T	F	T	F
T	T	T	T