Driving Behavior Analysis Software

* Ashraf Shaikh Mohammad([ashraf@iastate.edu](mailto:ashraf@iastate.edu)) , Shankar Sridhar([shankars@iastate.edu](mailto:shankars@iastate.edu))
* It is a software written in python 3 language which is used to parse CSV files
* It provides point based and segment-based scores.
* It combines all the segment scores and provides a score for the lap
* It plots graphs for acceleration, speed and jerk and saves it in local disk in a folder.
* It saves all the point and segment scores for this lap in a csv file
* It has 3 total python files – events\_graderv3.py, grading\_helper.py and trip\_graderv3.py
* Trip\_graderv3.py is the main file to run

**General instructions**

* Ensure python v3.x is installed in the system along with packages matplotlib which has to be downloaded externally
* Other used packages like sys, csv and math are in-built packages
* Does not require any IDE to run.
* Ensure path in trip\_grader.py are set accordingly to your system to load the log files in CSV format with correct names.
* Then just call trip\_graderv3.py in command line or IDE.
* If plots are required, ensure PLOT flag is set in events\_graderv3.py

# Events\_grader

* File which contains grading methods for regular and special segments
* Called by trip\_grader after end of a segment and uses the grading\_helpers to get info about zone limits, segment limits etc.
* Returns a score for the segment graded out of 100

## **Description**

The events\_grader takes all the necessary parameters such as speed, distance, accelerations, passed by value from trip\_grader and uses those points to calculate point-based score using the thresholds set in grading\_helpers and finally returns a score for the segment

## **Variables**

1. **DEBUG:** flag to turn on debugging print statements when grading
2. **PLOT:** flag to generate and save plots at the end of grading a segment
3. **START\_DIST, END\_DIST:** alias names for indexes 0 and 1. Just for meaningful name to call when used in segment or zone\_limits
4. **STOP\_SIGN:** flag to know the current zone graded has a stop sign and check for complete stops. Used when grading special segments
5. **SPEED\_LIMIT, JERK\_LIMIT\_REG, JERK\_LIMIT\_SP, POLY\_ACCN\_A etc.** are constants with index values for zone\_thresholds in grading\_helpers

For eg. zone\_thresholds[segment\_num][zone\_num][SPEED\_LIMIT] will access the zeroth value for that segment and zone number.

1. **CONV\_FACTOR:** Used for converting units of acceleration, deceleration or speed values wherever needed from km/h to that unit.

For eg. if it is 5/18, values multiplied with CONV\_FACTOR are converted from km/h to m/s.

If it is 1, then just indicates values are not converted and just remain same if multiplied with it.

## **Functions**

1. **Plot\_graphs()**
   1. Sets the file path to save
   2. Plots for speed, acceleration and jerk vs distance
   3. Based on segment counter and segment\_type, plots vertical lines for stop line or signal line and turn end or signal end and roundabout start/end at their distances
   4. Saves the figure in designated path in variable f\_name
   5. This function is called at the end of regular segment() or special\_segment().
   6. The PLOT flag in events\_grader is used to enable or disable calling this function
2. **get\_poly\_threshold()**
   1. takes in three arguments A,B,C required for a 2-degree polynomial equation and X for speed
   2. returns the acceleration value according to the parameters
3. **get\_exp\_threshold()**
   1. takes in two arguments A,B required for exponential equation and X for speed.
   2. Returns the acceleration value according to the coefficients supplied.
4. **Regular\_grading()**

Grades straight segments using polynomial regression and returns the segment score back to trip\_grader

**Parameters:**

1. **Speeds []** = list of actual speed values in km/h in this segment
2. **Accelerations []** = list of actual acceleration values in km/hs in this segment
3. **Jerks []** = list of actual jerk values in km/hs2 in this segment
4. **Distance\_intervals []** = list of point to point distance in meters in this segment
5. **Segment\_distance []** = list of distance covered till now at each point in this segment
6. **Total segment\_distance** = total distance travelled till now in this segment
7. **Segment\_counter** = current segment count. Starts from 0
8. **LAP\_NUM** = integer for the current lap number traversed

**Function local variables:**

1. **Segment\_score =** float to calculate score for the segment
2. **Num\_values =** integer to hold the number of points in this segment
3. **Zone\_counter =** integer to track and traverse through the zones in this segment. Used to traverse through zone\_limits and zone\_thresholds as index along with segment\_counter. Set to zero initially.
4. **Speed\_limits, accn\_limits, dccn\_limits, jerk\_limits\_positive, jerk\_limits\_negative =** list variables used to store various thresholds based on the current segment and zone number taken from zone\_thresholds. Acceleration and deceleration limits for each point are calculated by passing their respective coefficients from zone\_thresholds to functions get\_poly or get\_expo\_threshold functions.

These lists are used to plot the threshold limits in various plots for speed, acceleration and jerk

1. **Point\_file**  = file path to save the point scores in a csv file for this segment

**Algorithm:**

1. For each point in num\_values
2. Set speed\_weight to 100, accn and jerk weights to 0 initially
3. If current segment\_distance is greater than the zone end\_limit, increment zone\_counter until its within the limit
4. Get the speed\_limit for the current zone and append it to speed\_limits []
5. If speed above speed limit, calculate speed\_limit score for that point
6. Regular segments have three accn/dccn coefficients, so accn\_a,b and c coefficients are obtained from zone\_thresholds based on zone and segment counter. Similarly for dccn a,b and c
7. Acc\_limit and dccn\_limit values are obtained by passing the coefficients to get\_poly\_threshold() are appended to accn\_limits [] and dccn\_limits [] lists respectively.
8. If acceleration > 0 , i.e. positive acceleration, speed and acceleration weights are updated to 50 each.
9. If acceleration above the accn\_limit, score is calculated, otherwise accn\_score is set to 100
10. Steps 8 and 9 are repeated for deceleration if acceleration < 0 and stored in accn\_score
11. Jerk limit is obtained from zone threshold and appended to jerk positive and jerk negative limits lists, used later for plotting
12. If jerk value present, i.e. not zero in jerks[], update speed, accn and dccn weights to 33,34 and 33 respectively and jerk score is calculated and saved. If not jerk score is set to 100
13. Point score is obtained for this point by combining respective scores and weights of speed, acceleration and jerk
14. If DEBUG flag set, print debug statements
15. Segment\_score is added by the current point score times the distance
16. Point\_scores\_n\_weights is a common-separated string which has speed, jerk and acceleration score and the distance for this point and is written to point\_file
17. Repeat steps 2 to 16 for each point in num\_values
18. If num\_values is out of range, calculate segment\_score by dividing it with total\_segment\_distance
19. close point\_file
20. if PLOT is set, plot the graphs for this segment
21. If total segment score is less than 0, set it to zero and return it to trip\_grader.py
22. **Special\_grading()**

* Follows similar steps like regular\_grading.
* Uses expo\_thresholds instead of poly\_thresholds
* If the segment\_type is a stop, checks for stop sign and notes the minimum speed in the stop area

# Grading\_helpers

* Contains variables and constants used by other scripts
* Helps updating important parameters from a single file
* Keeps the variable clutter away from other files for a cleaner structure

## **Description**

This file is imported by both trip\_grader and events\_grader. It is designed in such a way that parameters pertaining to the route such as segment limits, zone\_limits, zone\_thresholds like speed limit, acceleration/deceleration coefficients are modifiable in the future without changing the grading algorithm or the trip\_grader which parses the file and calls the grading functions.

## **Variables**

1. **Field\_names:** for naming each column of the csv file read. Used when opening the csv file and can be used as a key to access the row data. For eg. rows[‘acceleration’]
2. **Segment\_type:** it is a python dictionary, with key being the segment­ number and its value which is a string which tells if the segment is start, straight, stop, signal, roundabout, turn and stop
3. **Segment\_limits:** gives the starting and ending distance in meters as a tuple
4. **Zone\_limits:** a zone is a part of a segment.

They are a range of distance in meters (x meters, y meters)

For straight segments, zones are based on different speed limits within a segment.

For eg. in segment\_limits[1], 20 to 70 m have the same 25 mph speed limit, so they have only one zone.

For special zones, it varies a bit.

For stop signs, there are 4 zones – the one where you approach, zone where you stop (approx. 10 m range), turn zone (all the stop signs have a turn followed by, in this route), leaving zone.

For stop lights, and roundabouts it will be three zones

1. **Zone\_thresholds:** It is a list of tuples for a segment. One list is for one segment.

Number of tuples in the list depends on number of zones.

Format of a tuple is **(speed limit(in km/h), accn coeff, dccn coeff,jerk threshold)**

It is made in this format so that thresholds belonging to a particular zone can be easily pulled from this tuple based on segment counter and zone counter during the run of the program.

These threshold coefficients were set by finding a closely fitting trendline on a acceleration vs speed graph with data obtained from driving logs, external articles as well as by taking into consideration of the context of the driving

Special segments follow exponential regression (y=Ae^Bx) and have only two coefficients A and B:

For acceleration on regular roads: y = 7.7871e-0.051x

For acceleration on turns: y = 14.477e-0.034x

For deceleration on regular roads: y = 13.028e-0.018x

For deceleration on turns: y = 8.28e-0.037x

Straight segments follow a 2 degree polynomial regression ( y=Ax2+Bx+C) and thus have three coefficients – A, B and C

For acceleration on city roads: y = 0.0012x2 - 0.248x + 13.117

For acceleration on highway: y = 0.0022x2 - 0.4956x + 30.047

For deceleration on city roads: y = 0.0012x2 - 0.248x + 13.117

For deceleration on highway: y = 0.0011x2 - 0.2187x + 12.461

1. Variables like REG\_SPECIAL\_ACCN\_ALPHA, and BETA are the threshold coefficients A and B respectively for special segments. Similarly, for straight segments, they are based on CITY or HIGHWAY and suffixed with A, B or C.

# Trip\_grader

* The main python script which must be run
* Loads the csv file and parses it
* Imports two other python files for calling grading related functions and variables
* Combines all the scores to produce the final score for the lap

## **Variables:**

1. **LAP\_NUM:** for choosing lap number so appropriate csv file can be loaded
2. **START\_DIST, END\_DIST:** index for 0th and 1st position of tuple. Useful in choosing first position of segment limit tuples
3. **Segment\_counter:** to track current segment count. Incremented every time segment limit is reached. Used by segment\_type to know the segment’s type whether straight or otherwise
4. **Speeds, accelerations, jerks, distance\_intervals, segment\_distances -** individual lists to store respective data when there is valid acceleration
5. **regular\_score, special\_score –** variables to store segment score after grading for their respective types

## **Description:**

It opens the log file in CSV format, parses each row and keeps adding the row data such as speeds, accelerations, jerk, distance etc. to their respective lists if the acceleration value in the row was valid (i.e. not ‘n/a’).

If the distance in row becomes greater than the current segment’s upper limit, it calls the corresponding grading function based on the segment\_type

Once done parsing the log file, it checks the data lists for any pending data and are graded accordingly as above.

The scores are then combined to produce the score for the lap

## **Algorithm**:

1. For every row in csv file:

1. Check If lap distance from csv file above the current segment limit based on segment\_counter.

If yes:

If segment\_type is “straight”

1. Set REGULAR flag to True
2. Call regular\_grading() by passing required parameters and save the score in regular\_score
3. Update regular\_distance by total\_segment\_distance
4. Update regular\_scores using current regular\_score

Else:

1. Set SPECIAL flag to True
2. Call special\_grading() by passing required parameters and save the score in special\_score
3. Update special\_distance by total\_segment\_distance
4. Update special\_scores using current special\_score

Reset variables

If no, proceed to 2:

1. If there is valid acceleration, i.e not ‘n/a’:
   1. Append necessary lists: speeds, accelerations, jerks, distance\_intervals and segment distances
   2. Update total segment distance

2. If speeds list is not empty after parsing the file, indicates last segment was not graded since file ended

Grade it based on segment\_type of the current segment\_counter as above

1. combine the scores using segment scores, segment weights and Importance weights