strym Documentation

Release 0.1

Rahul Bhadani

Mar 21, 2020

CONTENTS:

CHAPTER

ONE

CLASS *STRYMREAD*

class strym.**strymread**(*csvfile*, *dbcfile=''*, ***kwargs*)

strymread reads the logged CAN data from the given CSV file. This class provides several utilities functions

Parameters

- **csvfie** (*str*) – The CSV file to be read
- **dbcfile** (*str*) – The DBC file which will provide codec for decoding CAN messages

dbcfile

The filepath of DBC file

Default Value = ''

Type *string* **optional**

csvfile

The filepath of CSV Data file

Type *string*

dataframe

Pandas dataframe that stores content of csvfile as dataframe

Type *pandas.Dataframe*

candb

CAN database fetched from DBC file

Type *cantools.db*

Returns Returns an object of type *strymread*

Return type *strymread*

Example

```
>>> import strym
>>> from strym import strymread
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> dbcfile = 'newToyotacode.dbc'
>>> csvdata = '2020-03-20.csv'
>>> r0 = strymread(csvfile=csvlist[0], dbcfile=dbcfile)
```

messageIDs()

Retreives list of all messages IDs available in the given CSV-formatted CAN data file.

Returns A python list of all available message IDs in the given CSV-formatted CAN data file.

Return type *list*

count()
A utility function to plot counts for each Message ID as bar graph

Example

```
>>> import strym
>>> from strym import strymread
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> dbcfile = 'newToyotacode.dbc'
>>> csvdata = '2020-03-20.csv'
>>> r0 = strymread(csvfile=csvlist[0], dbcfile=dbcfile)
>>> ro.count()
```

start_time()
start_time retrieves the the human-readable time when logging of the data started

Returns Human-readable string-formatted time.

Return type *str*

end_time()
end_time retrieves the the human-readable time when logging of the data was stopped.

Returns Human-readable string-formatted time.

Return type *str*

triptime()
triptime retrieves total duration of the recording for given CSV-formatted log file.

Returns Duration in seconds.

Return type *double*

triplength(*time=-1*)
triplength returns total distance travelled while logging CAN data.

Alternative, one can provide a second argument *time* to query how much distance was traveled in, say 50 seconds from start.

Parameters **time** (*double*) – Provide a valid elapsed time in seconds to query how much distance was traveled *time* seconds since the logging of data was started.

driving_characteristics()
driving_characteristics provides driving characteristics for the given driving data in the form of python dictionary.

Currently, the dictionary contains following metadata from the driving data

- File name of CSV-formatedd CAN data file
- Associated DBC file used
- Start time of the trip in human-readable date format
- End time of the trip in human-readable date format
- Total duration of the trip

- Total distance traveled in meters
- Total distance traveled in kilometers
- Total distance traveled in miles

Returns A python dictionary containing driving metadata

Return type *dictionary*

speed()

Returns Timeseries speed data from the CSV file

Return type *pandas.DataFrame*

Example

```
>>> import strym
>>> from strym import strymread
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> dbcfile = 'newToyotacode.dbc'
>>> csvdata = '2020-03-20.csv'
>>> r0 = strymread(csvfile=csvlist[0], dbcfile=dbcfile)
>>> speed = r0.ts_speed()
```

accely()

Returns Timeseries data for acceleration in y-direction from the CSV file

Return type *pandas.DataFrame*

steer_torque()

Returns Timeseries data for steering torque from the CSV file

Return type *pandas.DataFrame*

yaw_rate()

Returns Timeseries data for yaw rate from the CSV file

Return type *pandas.DataFrame*

steer_rate()

Returns Timeseries data for steering rate from the CSV file

Return type *pandas.DataFrame*

steer_angle()

Returns Timeseries data for steering angle from the CSV file

Return type *pandas.DataFrame*

steer_fraction()

Returns Timeseries data for steering fraction from the CSV file

Return type *pandas.DataFrame*

wheel_speed_fl()

Returns Timeseeries data for wheel speed of front left tire from the CSV file

Return type *pandas.DataFrame*

wheel_speed_fr()

Returns Timeseeries data for wheel speed of front right tire from the CSV file

Return type *pandas.DataFrame*

wheel_speed_rr()

Returns Timeseeries data for wheel speed of rear right tire from the CSV file

Return type *pandas.DataFrame*

wheel_speed_rl()

Returns Timeseeries data for wheel speed of rear left tire from the CSV file

Return type *pandas.DataFrame*

rel_accel(*track_id*)
utility function to return timeseries relative acceleration of detected object from radar traces of particular track id

Parameters **track_id** (*numpy array*) –

Returns Timeseries longitduinal distance data from the CSV file

Return type *pandas.DataFrame*

long_dist(*track_id*)
utility function to return timeseries longitudinal distance from radar traces of particular track id

Parameters **track_id** (*numpy array*) –

Returns Timeseries longitduinal distance data from the CSV file

Return type *pandas.DataFrame*

lat_dist(*track_id*)
utility function to return timeseries lateral distance from radar traces of particular track id

Parameters **track_id** (*numpy array*) –

Returns Timeseries lateral distance data from the CSV file

Return type *pandas.DataFrame*

plt_speed()
Utility function to plot speed data

frequency()
Retrieves the frequency of each message in a pandas.Dataframe()

MessageID	MeanRate	MedianRate	RateStd	MaxRate	MinRate	RateIQR

Returns Returns the a data frame containing mean rate, std rate, max rate, min rate, rate iqr

Return type *pandas.DataFrame*

trajectory(*x_init=0.0*, *y_init=0.0*, *data_rate=50.0*)
A simple trajectory tracing function based on CAN data

Parameters

- **x_init** (*double*) – Initial X-coordinate of the vehicle
- **y_init** (*double*) – Initial Y-coordinate of the vehicle
- **data_rate** (*double*) – Rate at which message are sampled.

Returns A pandas Dataframe with three columns: Time, X, Y, Vx, Vy

Return type *pandas.DataFrame*

CHAPTER

TWO

CLASS *STRYM*

class strym.**strym**(*dbcfile*, ***kwargs*)

strym class records data from Comm AI Panda and visualize in real time. The constructor first gets an "USB context" by creating *USBContext* instance. Then, it browses available USB devices and open the one whose manufacturer is COMMA.AI. One right device is identified, *strym* creates a device handle, enables automatic kernel driver detachment and claim interface for I/O operation.

Read and Write for USB devices are either done synchronosly or in isochronos mode.

If your interest is merely in capturing data, you should perform synchronous mode.

For (almost) real time visualization, isochronous mode is the way to go.

Parameters

- **dbcfile** (*string*) –
- **path of can database file in order to decode the message** (*Provide*) –
- **kwargs** – Arbitrary keyword arguments.

  path: ***string*** Specify the path/folder where data will be saved. By default path is set to *~/CyverseData/JmscslgroupData/PandaData*

See also:

https://github.com/gotmc/libusb

https://pypi.org/project/libusb1/

https://vovkos.github.io/doxyrest/samples/libusb/index.html

https://github.com/vpelletier/python-libusb1

https://www.beyondlogic.org/usbnutshell/usb4.shtml

process_received_data(*transfer*)

process_received_data function implements a callback that processes the reeceived data from USB in isochronous mode. Once data is extracted from buffer, it is saved in the object's data variable. The data is used to update the plot in the real time.

isolog(*visualize*, *msg_type*, *attribute_num*, ***kwargs*)

isoviz() function will log everything in asynchronous manner but only visualize specific attribute of the given message. Upon pressing ctrl-C, the logging will terminate and SIGINT signal handler will create a plot and save in two formats: python's pickle format and pdf.

isoviz is responsible handling data transfer in the isochronous mode and parsing through callback function *process_received_data*

See https://vovkos.github.io/doxyrest/samples/libusb/group_libusb_asyncio.html?highlight=transfer#details-group-libusb-asyncio for more detail

Parameters

- **visualize** (*bool*) – specifies whether to visaulize while logging the CAN data
- **msg_type** (*string*) – specifies a valid message type from the DBC file
- **attribute_num** (*int*) – select the specific attribute from the given *mgs_type* to be displayed
- ****kwargs** – Arbitrary keyword arguments.

  log: ***enumeration: {info, debug}*** set log level to info and debug

  match: ***enumeration: {exact, in}*** how the message type and specified attribute should be matched for visualization. *exact* specifies exact match, *in* specifies substring matching.

kill(*sig*)

kill catches SIGINT or CTRL-C while recording the data and closes the comma ai device connection

parse_can_buffer(*dat*)

parse_can_buffer parses the can data received through the USB device and returns list of message ID, message and bus number

Parameters **dat** (*bytearray*) – byte data to be parsed

Returns Returns a list containing message ID, message and bus number

Return type *list*

CHAPTER

THREE

FUNCTIONS

strym.**ranalyze**(*df*, *title='Timeseries'*)
A utility function to analyse rate of a timeseries data

Parameters **title** (*str*) – A descriptive string for this particular analysis

strym.**plt_ts**(*df*, *title=''*)
A utility function to plot a timeseries

strym.**violinplot**(*df*, *title='Violin Plot'*)
A violin plot to show the data distribution

strym.**ts_sync**(*df1*, *df2*, *rate=50*)
Time-synchronize and resample two time-series dataframes of varying, non-uniform sampling.

In a non-ideal condition, the first time of *df1* timeseries dataframe will not be same as the first time of *df2* dataframe.

In that case, we will calculate the value of message at the latest of two first times of *df1* and *df2* using linear interpolation method. Call the latest of two first time as *latest_first_time*.

Similarly, we will calculate the value of message at the earliest of two end times of *df1* and *df2* using linear interpolation method. Call the latest of two first time as *earliest_last_time*.

Linear interpolation formula is

$$X_i = \frac{X_A - X_B}{a - b}(i - b) + X_B$$

Next, we will truncate anything beyond [*latest_first_time*, *earliest_last_time*]

Once we have common first and last time in both timeseries dataframes, we will use cubic interpolation to do uniform sampling and interpolation of both time-series dataframe.

Parameters

- **df1** (*pandas.DataFrame*) – First timeseries datframe. First column name must be named 'Time' and second column must be 'Message'
- **df2** (*pandas.DataFrame*) – Second timeseries datframe. First column name must be named 'Time' and second column must be 'Message'
- **rate** (*double*) – New uniform sampling rate

Returns

- **dfnew1** (*pandas.DataFrame*) – First new resampled timseries DataFrame
- **dfnew2** (*pandas.DataFrame*) – Second new resampled timseries DataFrame

PYTHON MODULE INDEX

S

INDEX