

---

# **strym Documentation**

***Release 0.1***

**Rahul Bhadani**

**Mar 11, 2020**



**CONTENTS:**

<b>1</b>	<b>Class <i>strymread</i></b>	<b>1</b>
<b>2</b>	<b>Class <i>strym</i></b>	<b>5</b>
<b>3</b>	<b>Functions</b>	<b>7</b>
	<b>Python Module Index</b>	<b>9</b>
	<b>Index</b>	<b>11</b>



## 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

- **csvfile** (*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()**

Retrieves 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()
```

**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** Timeseries data for wheel speed of front left tire from the CSV file

**Return type** *pandas.DataFrame*

**wheel\_speed\_fr()**

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

**Return type** *pandas.DataFrame*

**wheel\_speed\_rr()**

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

**Return type** *pandas.DataFrame*

**wheel\_speed\_rl()**

**Returns** Timeseries 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 longitudinal 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 longitudinal 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*

**integrate** (*df*, *init=0.0*, *integrator=<function cumtrapz>*)

Integrate a timeseries data using scipy.integrate.cumtrapz

**Parameters**

- **df** (*pandas.DataFrame*) – A two column Pandas data frame. First Column should have name ‘Time’ and Second Column Should be named ‘Message’
- **init** (*double*) – Initial conditions for integration. Default Value: 0.0.
- **integrator** (*function*) – Integrator method. By default, it is *scipy.integrate.cumptrapz*

**Returns df** – A two column Pandas data frame with first column named ‘Time’ and second column named ‘Message’

**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*



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 synchronously or in isochronous 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/JmcsIgroupData/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 received 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](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 visualize 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 *msg\_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

**dat: bytearray** byte data to be parsed

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

**Return type** *list*

## FUNCTIONS

`strym.ranalyze(df, title='Timeseries')`

A utility function to analyse rate of a timeseries data

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 dataframe. First column name must be named 'Time' and second column must be 'Message'
- **df2** (*pandas.DataFrame*) – Second timeseries dataframe. 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 timeseries DataFrame
- **dfnew2** (*pandas.DataFrame*) – Second new resampled timeseries DataFrame



## PYTHON MODULE INDEX

### S

`strym`, [1](#)



## A

`accely()` (*strym.strymread method*), 2

## C

`candb()` (*strym.strymread attribute*), 1  
`count()` (*strym.strymread method*), 2  
`csvfile()` (*strym.strymread attribute*), 1

## D

`dataframe()` (*strym.strymread attribute*), 1  
`dbcfile()` (*strym.strymread attribute*), 1

## F

`frequency()` (*strym.strymread method*), 3

## I

`integrate()` (*strym.strymread method*), 4  
`isolog()` (*strym.strym method*), 5

## K

`kill()` (*strym.strym method*), 6

## L

`lat_dist()` (*strym.strymread method*), 3  
`long_dist()` (*strym.strymread method*), 3

## M

`messageIDs()` (*strym.strymread method*), 1

## P

`parse_can_buffer()` (*strym.strym method*), 6  
`plt_speed()` (*strym.strymread method*), 3  
`plt_ts()` (*in module strym*), 7  
`process_received_data()` (*strym.strym method*), 5

## R

`ranalyze()` (*in module strym*), 7  
`rel_accel()` (*strym.strymread method*), 3

## S

`speed()` (*strym.strymread method*), 2  
`steer_angle()` (*strym.strymread method*), 2  
`steer_fraction()` (*strym.strymread method*), 3  
`steer_rate()` (*strym.strymread method*), 2  
`steer_torque()` (*strym.strymread method*), 2  
`strym` (*class in strym*), 5  
`strym` (*module*), 1  
`strymread` (*class in strym*), 1

## T

`trajectory()` (*strym.strymread method*), 4  
`ts_sync()` (*in module strym*), 7

## V

`violinplot()` (*in module strym*), 7

## W

`wheel_speed_fl()` (*strym.strymread method*), 3  
`wheel_speed_fr()` (*strym.strymread method*), 3  
`wheel_speed_rl()` (*strym.strymread method*), 3  
`wheel_speed_rr()` (*strym.strymread method*), 3

## Y

`yaw_rate()` (*strym.strymread method*), 2