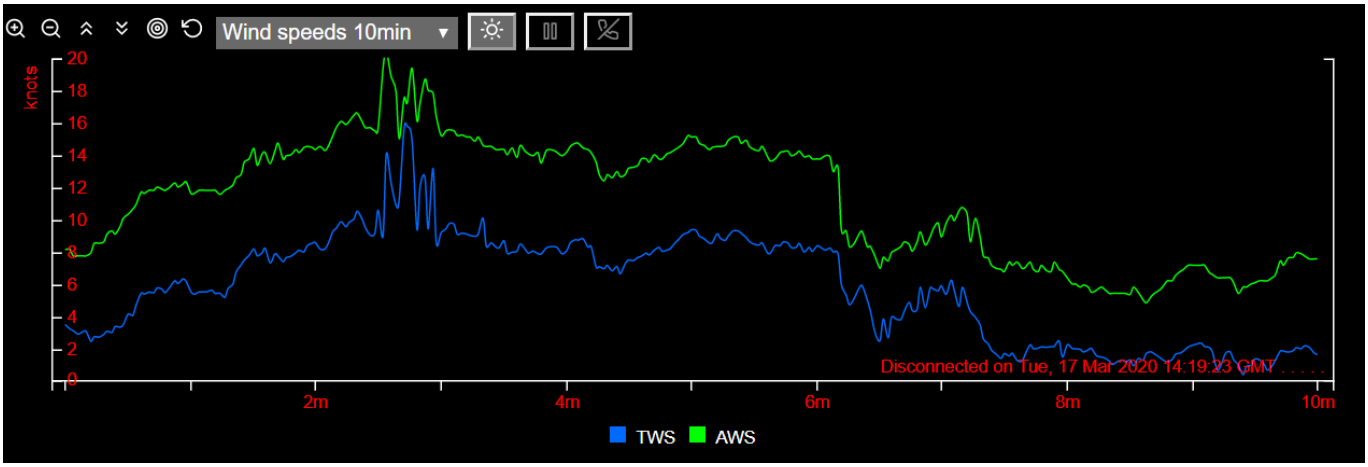
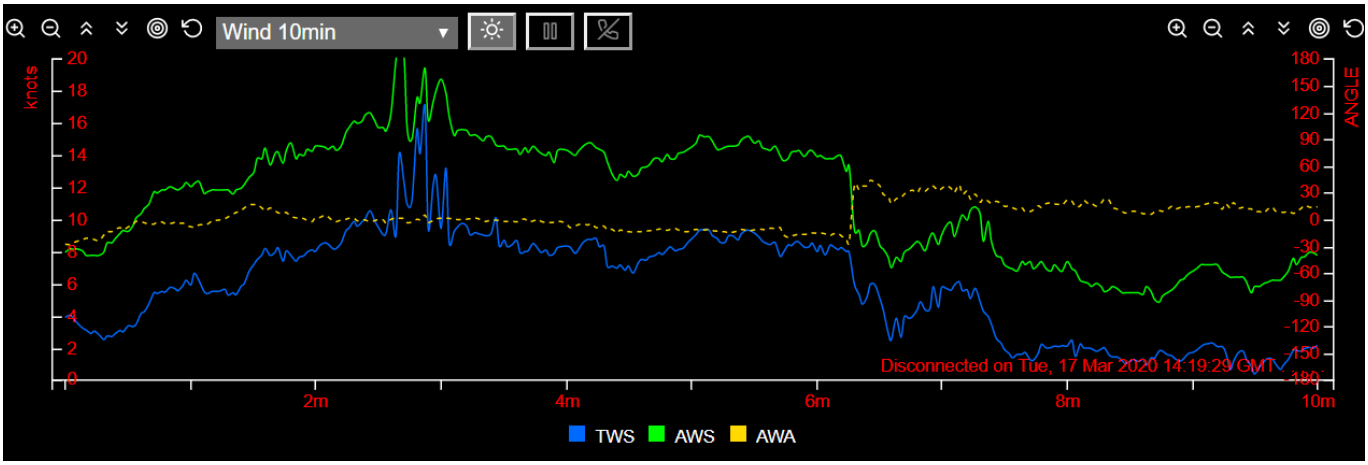
Signalk-stripcharts: generate strip charts from Signal K live boat data.

A stripchart displays the most recent boat data (from one or more Signal K paths) as a graph along a time axis (x-axis). Legends identify by abbreviations what paths are charted. Here is a chart with a 10 minutes time window plotting the true wind speed (TWS) and apparent wind speed (AWS):



The above chart has a y-axis on the left hand side. Optionally, as shown on the chart below (right hand side), a y2-axis may also be defined with its own unit. The dotted line corresponds to the plot according to the y2 axis, here the apparent wind angle (AWA).



The browser window can display two such charts on top of each other. For instance you may have wind speeds on the top chart and boat speeds on the bottom chart, or wind speeds with two different time windows, say 10 minutes and 24 hours. Those two charts can be selected with a dropdown list from a set of active charts.

Now it's time to install and start using the default charts specifications provided.

Table of contents:

- **Installation**
  - **Install as Webapp from Signal K Dashboard**
  - **Install on a client device**

    - **Install on a node server - typically the boat Signal K server**
- **Basic usage**
    - **y- and y2-axis buttons**
    - **Drop-down list**
    - **Main buttons**
- **How it works**
- **Customization**
    - **Chart specifications**
        - Derived data
        - Legends and lines colors
    - **Unit conversion**
    - **General options**
- **Browser compatibility**
- **CPU and memory requirements**
- **Troubleshooting**
- **Functional improvements**
- **Technical improvements**
- **Credits**

Table of contents generated with markdown-toc

Installation

Signalk-stripcharts comes with all required dependencies (including c3 charting library and d3 visualization library).

Choose one of the following 3 methods as applicable or desired:

Install as Webapp from Signal K Dashboard

If Signal K node server (https://github.com/SignalK/signalk-server-node) is installed on your server:

- login to the Signal K dashboard
- open the left hand side menu
- select `Appstore`, then `Available` and find `Signalk-Stripcharts` (if not found, it is probably already installed)
- click on the install icon next to the version number
- when installation is finished, click on the dashboard `Restart` button

You can now start signalk-stripcharts launcher from the dashboard `Webapps` menu.

Install on a client device

- Download the ZIP file from https://github.com/edefalque/signalk-stripcharts and unzip in a folder.

(by default, it will be installed in a folder named `signalk-stripcharts-master`; rename folder `signalk-stripcharts`)

Install on a node server - typically the boat Signal K server

- Use npm: `[sudo] npm install signalk-stripcharts`

This will install Signalk-stripcharts under the closest node_modules folder higher up in the hierarchy.

Basic usage

Start the launch menu:

- **If installed as a Signal K Webapps**: from your client or on the Signal K server, access the Signal K dashboard (e.g. *`mysignalkserveripaddr`*`:3000/signak` or `localhost:3000/admin`), then start Signalk-Stripcharts from the dashboard `Webapps` menu.

You may also:

- **If installed on a (Signal K) node server**: from your client browser enter url *`mynodeipaddr:port`*`/signalk-stripcharts`
- **If installed on a client**: double click on *`...path...`*`/signalk-stripcharts/index.html`

On the launch menu you can choose the Signal K server (same or distinct from the node server where signalk-stripcharts is installed) and choose a set of charts, then push the `Launch Signalk-stripcharts` button. If you are connected to the Internet try first with the default selections of the launch menu.

When the charts are displayed:

- hover on a legend: the corresponding plot is highlighted, the others dimmed; the corresponding Signal K path is displayed above the legend
- hover on a plot line, a tooltip is diplayed with the values corresponding to the legends
- by clicking on a legend you can toggle the dim status of the legend and of the corresponding plot

You may wish to bookmark the launched page(s) for easier later launching. Modify the query parameters `server` & `specs` as needed, e.g. if you have defined your own charts specifications (see later section "How it works").

y- and y2-axis buttons

These buttons affect the size and position of the plotted lines corresponding to the axis:

From left to right:

- zoom in
- zoom out
- raise the plot lines (y-axis range is shifted)
- lower the plot lines
- try to center the plot lines at present time
- reset to the initial scale and position settings

Drop-down list

You can select any chart in the set for display. If you choose `none` the other chart will be displayed on the whole window (similar effect if the same chart is selected twice).

Main buttons

From left to right:

- toggle day/night display
- toggle pause/play plotting
- disconnect from Signal K server (reload the page if you want to connect again)

They apply to the whole window.

How it works

A chart contents and rendition is governed by the **chart specifications**, **unit conversion** and some **general options**.

In Signal K, data types are identified as paths: e.g. the true wind speed is identified by the path `environment.wind.speedTrue`.

A chart is primarily specified by:

- a name
- the length of the time axis (`timeWindow`); typically 10 minutes, 2 hours or 24 hours
- the averaging period (`avgInterval` explained below)
- the y axis range and unit (unit conversion is taken care of)
- the same properties for the optional y2 axis
- the list of Signal K paths to be charted and corresponding abbreviated legends

For each path the charted value can be the average (`AVG`), the maximum (`MAX`) and/or the minimum (`MIN`) of the values received from Signal K during the averaging period. The averaging period (`avgInterval`) should be set such that the ratio `timeWindow`/`avgInterval` is between 300 and 1000 (i.e. a reasonable number of plots along the `timeWindow`); `avgInterval` must be longer than the sampling period defined in the Signal K subscription period (`subscribePolicy.period` option), which is typically 1 second. By default, the chart is refreshed every `avgInterval` (provided that some new data has come for that chart). When data stops coming for a particular path, it is "hotdecked" from the last data received until new data arrives; hotdecking stops after `hotdeckSec` seconds (default 60 in options) and thereafter the corresponding data will be 0 (in Signal K unit) indicating probably that the corresponding instrument was disconnected or switched off.

A chart specification is provided as a Javascript object. Here is the specification for the chart shown at the top of this document:

```
const Wind_speeds_10min =
    // for javascript variables naming rules see
https://javascript.info/variables#variable-naming
```

```
    { stripChartName: "Wind_speeds_10min",   // use preferably the same as the
containing object name
        timeWindow: 600,            // 10min in seconds
        avgInterval: 2,             // 2 sec
        intervalsPerRefresh: 2,     // default 1
        x: { label: "min/sec ago", tickCount: 11 },   // a tick every min (10, + 1
for zero)
        y: { unit: "Knot" },
        paths:
        [
            { path: "environment.wind.speedTrue", AVG: "TWS" },  // average is
plotted, legend is "TWS"
            { path: "environment.wind.speedApparent", AVG: "AWS" }
        ]
    }
;
```

By default, the chart is refreshed every `avgInterval` (provided that some new data has come); however refreshing can be made slower with the `intervalsPerRefresh` property in order to spare some processing.

Related chart specification objects are grouped into a set.

When the application is started as an url, the following "query" parameters must be provided:

- `server` with the address and port of the Signal K server (defaults to the address & port the page is loaded from)
- `specs` with the file name containing the chart specifications set (without .js extension).

The application subscribes to the Signal K server deltas for all the paths in the set. The values are then continuously collected and aggregated for all charts in a set (using the Streaming WebSocket API).

Different sources may provide data for the same path; as needed, the `path` property may be extended to filter a specific source, e.g. `navigation.speedOverGround[gps.1]`. Sources corresponding to a path on your Signal K server can be obtained from the launch menu provided that the corresponding instrument is on and connected. A certain path may be listed several times in the chart specs, e.g.:

```
{ path: "navigation.speedOverGround[gps.1]", AVG: "SOG1" },
{ path: "navigation.speedOverGround[gps.2]", AVG: "SOG2" },
{ path: "navigation.speedOverGround", AVG: "SOGx" },
```

The first two lines will collect data each respectively from `gps.1` and `gps.2`. The third line will collect SOG data from any other sources. If a line collect data from different poorly calibrated sources, the line plotted might be "jumpy" (note: the average computation will be influenced by the frequency at which the data is provided by the two sources).

At any one time, two charts can be displayed as selected by the user from drop down lists. If the user selects `none` in one of the drop down lists, the remaining chart covers the whole window area.

Charts can be paused. When paused, data collection continues "behind the scene". And the charts display will catch up when "unpaused". When the chart window is minimized or not in view, the charts are paused likewise and will catch up when brought to view. This minimizes processing load.

There is no persistency: when the window is closed it is disconnected from Signal K and the buffers are "lost". In order to reconnect reload the page.

The charts specifications sets are each stored in a .js file in the `./specs` folder. The following specifications files are provided at installation:

- `sail.js`
- `environment.js`
- `engines.js`

Each of them can be started in its own browser tab and run concurrently.

Additionally, `sources_filtering_example.js` is provided (read comments in the file before running).

In a specification file, a chart specification can be derived from another chart and only the properties that differ need to be specified (e.g. a two-hours chart can be derived from a ten-minutes chart, with most of the properties inherited). Inheritance is provided at the first level of properties only.

Signalk-stripcharts buffers having no persistency, they cannot be used to store the history. Persistency and more powerful charting capabilities can be provided with InfluxDB and Grafana as explained here (https://github.com/tkurki/signalk-to-influxdb/blob/master/README.md) or could be provided with the help of the history capability of Signal K if available.

Customization

Currently, customization is easy if the package is installed on the client, but less if it is installed on a server as the specs files may then be less easily accessible. If installed on a Raspberry PI from Signal K Appstore, they will probably be in `$HOME/.signalk/node_modules/signalk-stripcharts/specs/`.

Chart specifications

The specifications files are installed in `signalk-stripcharts/specs/`. Ample comments are provided for those features that were not explained above.

New specifications files can be easily derived from those provided at installation. Copy them under another name and edit them with a text editor, or better with a code editor such as Geany or Visual Studio Code.

The launch menu has a button that lists all paths and sources currently provided by the selected Signal K server. Switch on all your instruments and systems in order to obtain a full list of what you can chart.

Derived data

Some path values are provided to Signal K by the instruments; some other path values may need to be derived. E.g. true wind speed, if not provided by the instruments, can be derived from apparent wind speed, apparent wind angle and speed over ground.

Use the following Signal K dashboard menus:

- `Server/Plugin Config/Derived Data` in order to configure and start path derivation from existing paths; see https://github.com/SignalK/signalk-derived-data (you may have to instal the plugin first from Signal K `Appstore`)
- `Data Browser` in order to visualize the active and derived paths with their current values

Legends and lines colors

Colors can be specified per legend at the bottom of the chart specifications file in order to insure consistency accross multiple charts of a same set. If not provided colors will be assigned automatically by the c3 library. See `sail.js` for how to assign colors.

Unit conversion

`signalk-stripcharts/specs/units` provides the list of Signal K units and charting units (not yet a complete set), with the conversion factors and algorithms. It also provides the following default properties for the y and y2 axis as a function of the charting unit:

- `label`
- range (`min` and `max`)
- `tick.count` and `tick.format`

Those can be overridden in the chart specs.

A special unit `Percent` is provided. It allows to plot values of different units on a same "Percent" y or y2 axis by providing reference values in the Signal K unit for 0% and for 100%. See `engines.js` for an example with comments.

General options

Options governing all charts are given in `signalk-stripcharts/js/stripcharts_options.js`. See comments in the file. Some of those comments explain how time is managed in signalk-stripcharts.

Any of the following options can also be entered as query parameters after the url when launching stripcharts.html: `timeTolSec`, `logTypes`, `hotdeckSec`, `period`, `format`, `policy` and `minPeriod`. They will then override the corresponding values in `stripcharts_options.js`.

Invalid option values will be replaced by defaults.

Browser compatibility

Signalk-stripcharts uses ECMAScript 2015 (ES6).

It was mostly developed on Raspbian Chromium 72.0.3626.121 and Windows 10 Chrome 80.0.3987.132.

It seems to also work fine on:

- Windows 10 with Edge 18362 and with Firefox-ESR 68.6.0esr
- Raspbian with Firefox-ESR 52.9.0
- Androïd with Chrome 78.0.3904.108 (on mobile phone, preferably use the landscape orientation and select `none` for one of the charts)

Recent non-ESR versions of Firefox show some svg rendition problems.

CPU and memory requirements on the client side

On a Raspberry Pi 3B+ with signalk-stripcharts executing on a Chromium browser: a short burst of approximately 50% cpu consumption is observed when 2 charts corresponding to 6 paths are refreshed on the window (typically every 4 seconds for 10 minute timeWindow charts, or every 10 seconds for 2 hour timeWindow charts); this is only when the window tab is visible and "playing". Augment `avgInterval` and/or `intervalsPerRefresh` if you want to reduce the frequency of those bursts.

Between refreshing bursts, managing the data collection and aggregation for 10 charts, each with approximately 3 paths, takes less than 1% CPU. For the same numbers of charts and paths, memory footprint is between 40K and 60K at all time.

(measurements above are from the browser Task manager)

Troubleshooting

Syntax errors in the charts specifications files will be caught by javascript. Logical errors in the specs and errors in the data returned by Signal K may also be reported. Inspect the browser console as needed.

Some tracing options are provided in `./js/stripcharts_options.js`. Tracing occurs below the displayed charts.

Functional improvements

- [v] Filter by source: in a chart specs, at path level, specify source~~s wanted as an array~~
- ☐ ~~Filter out from the specs the path/sources which are never provided by the signalk server~~
- ☐ Provide a better way to create/manage custom specs files and preserve them when installing new versions of the package
- ☐ Units, add some missing units and conversions: Volt, Ampere, Watt, Joule, Coulomb, Ratio(0-n), Ratio(0%-100%), Cubic_meter, Cubic_meter_per_second, ...

Technical improvements

Signalk-stripcharts is essentially an html/javascript front-end application. It was developed with no or very limited prior knowledge of Linux, Javascript, HTML, CSS, Signal K, Node.js and other js-related tools. Therefore much technical improvement is of course possible. In particular the following improvements could be considered without structural changes:

- ☐ Transpiling (Babel)
- ☐ Minification.
- ☐ Retrieval of boat data is inspired from a simple Signal K websocket example. Possibly it could be replaced by the more robust https://github.com/SignalK/signalk-js-client

Any hints are most welcome.

Credits

The following open source or free software contributions have been especially inspiring:

- NavMonPC stripcharts: http://www.navmonpc.com/
- Signal K: http://signalk.org/
- c3.js: https://c3js.org/

Thank you to all who have shown the way!