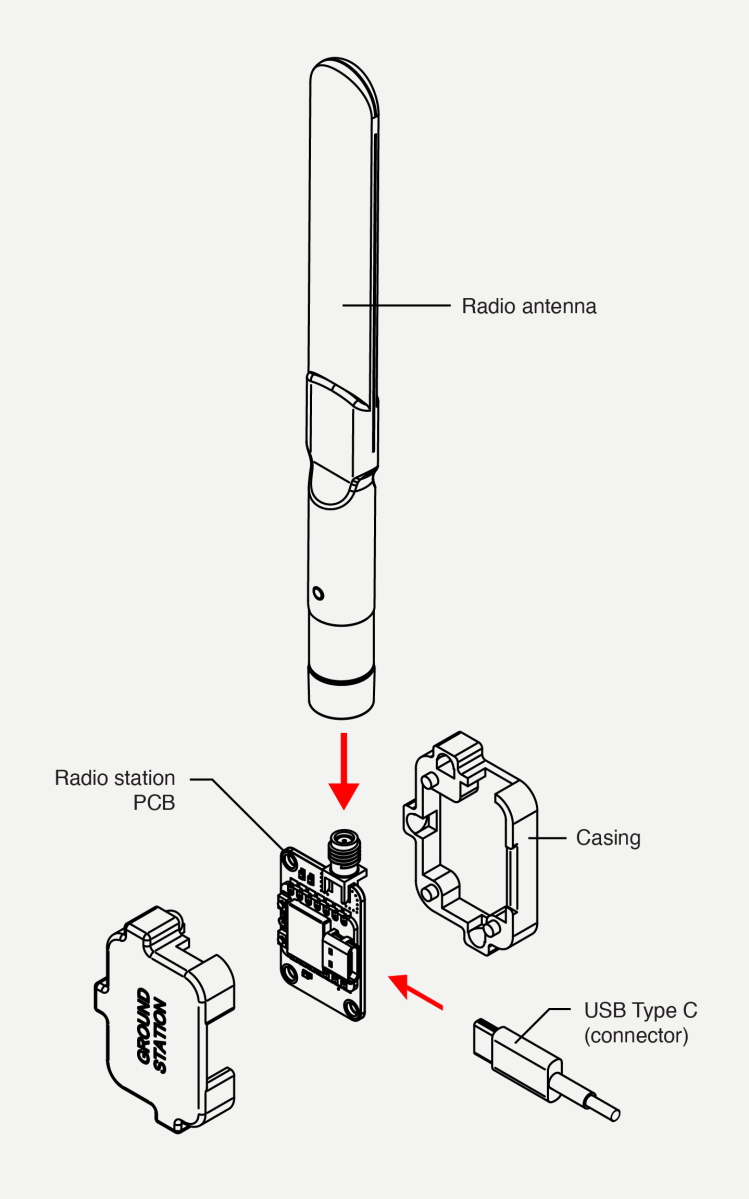
Outside the vehicle

Radio Station

Setup

Hardware

1. Assembling the radio station with casing.
2. Attach the radio antenna to the station and USB connector to your laptop.



Software

The radio station uses the Arduino IDE just like the other boards, and will need the RadioHead library installed from the github repository.

Before using the interface two applications need to be installed:

1. MATLAB for using the custom app.

2. VSCode (or any IDE) and the Teleplot extension installed for troubleshooting and redundancy.

How to use

Software - Live Race Engineer Interface

1. Ensure that the telemetry system is turned on. Can be verified by lights on the boards and the driver display should be displaying information if connected.
2. Connect the ground station to your laptop.
3. Open Teleplot and select the port and baudrate (115200 by default).
4. Use the Teleplot interface to verify that data is being received and monitor it.
5. If Teleplot worked, disconnect from it (only one application can access a given serial port at a time) open the MATLAB application and run it.
6. On the top left of the MATLAB application select the port and baudrate (same as Teleplot).
7. If this worked successfully you should be seeing data being plotted on the plots. Use the lap and quantity toggle switched to adjust the plots.

{% hint style="warning" %}

- Always start with Teleplot.
- The data that is being sent from the ground station to the laptop follows the format specified by Teleplot A:B§C
  - **A** is the name of the telemetry variable (be kind and avoid `:|` special chars in it!).
  - **B** is either the integer or floating point value to be plotted or a text format value to be displayed.
  - **C** is **optional** and is the unit of the telemetry (please avoid `,;:|.` special chars in it!).
- The MATLAB application will crash if you click on a toggle switch that has no datafeed assigned to it.
- The timer in the MATLAB application currently needs to be started manually. {% endhint %}

Development

How to edit/update

The MATLAB application works by taking reading the next message from the serial port which contains a single measurement in the Teleplot format, identifying its quantity (eg. Velocity, Super-capacitor voltage), storing it to it's corresponding cell array (an array of arrays more detail on it later), and updating the plots accordingly with the new data while also checking which checkbox is checked.

{% hint style="info" %} **Why cell arrays?**

Cell arrays were used to allow for various measurements per lap to be taken and stored in a dynamic structure. Where the index of the cell array corresponds to the lap number. Allowing to compare various laps measurements with each other based on measurement number. Meaning that you can only compare for example the 10th velocity measurement of lap 3 with lap 5 and not the velocity at 100 m from the start line of these laps. This is a big negative of the current version of the interface, but assuming that the vehicle has consistent lap times then this approach is acceptable. How to improve this is detailed in the future work. {% endhint %}

Main bits of the code

Whenever calling a function in the MATLAB app environmnent the first value you need to pass into it is always “app”.

▶ Details
Cell arrays

This is where the readings will be stored as discussed previously.

{% code overflow="wrap" %}

```
velocity = {}; %Cell array to store velocity readings of each lap
current = {}; %Cell array to store current readings of each lap
voltage = {}; %Cell array to store voltage readings of each lap
%Voltage values
superCapsOverallVoltage = {}; %Cell array to store the supercaps overall voltage
readings of each lap
beforeMotorControlVoltage = {}; %Cell array to store the before motor control
voltage readings of each lap
DCDCVoltage = {}; %Cell array to store the DC DC voltage readings of each lap
```

{% endcode %}

▶ Details
Start-up function

This is the function that allows for the serial port readings to be continuously (= every 1 ms) monitored and updated (highlighted in yellow), ultimately enabling the plots to be updated. The other stuff named timer are used to update the stopwatch.

Define these in `properties (Access = private)`

```
        myTimer %Timer
        Timer % Timer object
        TimerObject % Timer object
        StartTime % Start time
```

Function itself

```
        function startupFcn(app)
            % Query available serial ports
            serialInfo = serialportlist("available");
            app.serialPort.Items = serialInfo;
            if isempty(serialInfo)
                app.serialPort.Items = {'No Ports Available'};
                app.serialPort.Enable = 'off';
```

```
            else
                app.serialPort.Enable = 'on';
            end
            app.myTimer = timer('ExecutionMode', 'fixedRate', 'Period', 1,
'TimerFcn', @(src, event)myTimerCallback(app, src, event)); % Specify the callback
function
            % Start the timer
            start(app.myTimer);
            app.TimerObject = timer(...
                'ExecutionMode', 'fixedRate', ...
                'Period', 1, ... % Update every second
                'TimerFcn', @(~,~)updateTimer(app));
        end
```

As of now the way the code is setup you need to define a new add value function for each quantity that you want to be plotted and a function to update each of your plots.

▶ Details

Adding the new values

The function works by appending the value you give it to the end of the nth array in the cell array. Where n is the current lap number.

{% code overflow="wrap" %}

```
        function addVelocityValue(app, value)
            lapIndex = app.LapField.Value; % Get the current lap index from the
LapField

            if length(app.velocity) < lapIndex || isempty(app.velocity{lapIndex})
                % If the current lap's cell doesn't exist, initialize it with the
value
                app.velocity{lapIndex} = value;
            else
                % Otherwise, append the value to the existing cell data
                app.velocity{lapIndex} = [app.velocity{lapIndex}, value];
            end
        end
```

{% endcode %}

▶ Details

Plotting the new values

Single variable plots - For plots like velocity where you are only comparing its value through various laps the following function was created to plot it.

{% code overflow="wrap" %}

```
        function updateVelocityPlot(app)
            cla(app.velUIAxes);  % Clear the current axes
            hold(app.velUIAxes, 'on');
            checkedNodes = app.LapTree.CheckedNodes;  % Get the list of checked
nodes

            % Loop over each checked node
            for i = 1:numel(checkedNodes)
                node = checkedNodes(i);  % Get each checked node
                lapIndex = node.NodeData;  % Assuming NodeData stores the
corresponding lap index

                % Ensure lapIndex is valid and within bounds
                if isnumeric(lapIndex) && lapIndex > 0 && lapIndex <=
length(app.velocity)
                    % Check if the lap data exists and is not empty
                    if ~isempty(app.velocity{lapIndex})
                        plot(app.velUIAxes, app.velocity{lapIndex}, 'DisplayName',
sprintf('Lap %d', lapIndex), 'LineWidth',4);
                    end
                end
            end
            hold(app.velUIAxes, 'off');
            legend(app.velUIAxes, 'Location', 'best');
            title(app.velUIAxes, 'Velocity vs Measurements');
            xlabel(app.velUIAxes, 'Measurements');
            ylabel(app.velUIAxes, 'Velocity (m/s)');
        end
```

{% endcode %}

Multivariable plots - For plots where multiple quantities are being compared within a lap and over various laps the following function can be used. With the difference compared to the previous function being the switch which checks for which quantities should be plotted for the given lap.

{% code overflow="wrap" %}

```
        function updateCurrentPlot(app)
            cla(app.currentUIAxes);  % Clear the current axes
            hold(app.currentUIAxes, 'on');
            checkedLapNodes = app.LapTree.CheckedNodes;  % Get the list of checked
nodes from the LapTree
            checkedCurrentParameters = app.CurrentTree.CheckedNodes;  % Get the
list of checked voltage parameters


            % Loop over each checked lap node
            for i = 1:numel(checkedLapNodes)
                lapNode = checkedLapNodes(i);  % Get each checked lap node
```

```
                lapIndex = lapNode.NodeData;  % Assuming NodeData stores the
corresponding lap index

                % Ensure lapIndex is valid and within bounds
                if isnumeric(lapIndex) && lapIndex > 0 && lapIndex <=
length(app.velocity)
                    % Check if the lap data exists and is not empty

                    % Loop over each checked parameter node
                    for j = 1:numel(checkedCurrentParameters)
                        parameterNode = checkedCurrentParameters(j);
                        parameterIndex = parameterNode.NodeData;  % NodeData here
should correspond to the parameter index

                        switch parameterIndex
                            case 1  % Supercaps Overall Current
                                if ~isempty(app.superCapsOverallCurrent{lapIndex})
                                    plot(app.currentUIAxes,
str2double(app.superCapsOverallCurrent{lapIndex}), 'DisplayName', sprintf('SCO Lap
%d', lapIndex), 'LineWidth',4);
                                end
                            case 2  % Supercaps Charge Current
                                if ~isempty(app.superCapsChargeCurrent{lapIndex})
                                    plot(app.currentUIAxes,
str2double(app.superCapsChargeCurrent{lapIndex}), 'DisplayName', sprintf('SCC Lap
%d', lapIndex), 'LineWidth',4);
                                end
                            case 3  % Before Motor Control Current
                                if
~isempty(app.beforeMotorControlCurrent{lapIndex})
                                    plot(app.currentUIAxes,
str2double(app.beforeMotorControlCurrent{lapIndex}), 'DisplayName', sprintf('BMC
Lap %d', lapIndex), 'LineWidth',4);
                                end
                            % case 4  % DC DC Current
                            %     if ~isempty(app.DCDCCurrent{lapIndex})
                            %         plot(app.currentUIAxes,
str2double(app.DCDCCurrent{lapIndex}), 'DisplayName', sprintf('DCDC Lap %d',
lapIndex), 'LineWidth',4);
                            %     end
                        end
                    end
                end
            end
            hold(app.currentUIAxes, 'off');
            legend(app.currentUIAxes, 'Location', 'best');
            title(app.currentUIAxes, 'Current vs Measurements');
            xlabel(app.currentUIAxes, 'Measurements');
            ylabel(app.currentUIAxes, 'Current (A)');
        end
```

{% endcode %}

Putting it all together

{% hint style="info" %} You can click on the underlined code to reveal information about its use. {% endhint %}

```
function myTimerCallback(app, src, event)
            if (src.NumBytesAvailable > 0)
                        dataStr = readline(src);  % Reads data from the serial
port until a newline is received
                        % Remove non-ASCII characters except '§'
                        dataStr = regexprep(dataStr, '[\s>]|[^\x20-\x7E§]', '');
                        try
                            % Split the data into its components
                            splitData = strsplit(dataStr, '§');  % Split based on
the '§' delimiter
                            if numel(splitData) == 2
                                varNameValue = splitData{1};
                                %varUnit = splitData{2};

                                % Further split to separate variable name and
value
                                nameValueSplit = strsplit(varNameValue, ':');
                                    if numel(nameValueSplit) == 2
                                        varName = nameValueSplit{1};
                                        varValue = str2double(nameValueSplit{2});
% Convert value to double

                                        if strcmp(varName, 'Lap#')
                                            app.LapField.Value = varValue;
                                            handleNewLap(app);

                                        elseif strcmp(varName, 'Velocity')
                                            %disp('This is a velocity reading');
                                            addVelocityValue(app, varValue);
                                            updateVelocityPlot(app);

                                        %Supercaps Overall voltage update
                                        elseif strcmp(varName, 'SCOV')
                                            addSCOVoltageValue(app,
string(varValue));

                                            updateVoltagePlot(app);

                                        %before motor control voltage update
                                        elseif strcmp(varName, 'BMCV')
                                            addBMCVoltageValue(app,
string(varValue));

                                            updateVoltagePlot(app);

                                        %DC DC voltage update
                                        elseif strcmp(varName, 'DCDCV')
                                            addDCDCVoltageValue(app,
```

```
string(varValue));
                                        updateVoltagePlot(app);

                                    %Supercaps Overall current update
                                    elseif strcmp(varName, 'Curr1')
                                        addSCOCurrentValue(app,
string(varValue));
                                        updateCurrentPlot(app);

                                    %Supercaps Charge current update
                                    elseif strcmp(varName, 'Curr2')
                                        addSCCCurrentValue(app,
string(varValue));
                                        updateCurrentPlot(app);

                                    %Before Motor Control Current
                                    %update
                                    elseif strcmp(varName, 'Curr3')
                                        addBMCCurrentValue(app,
string(varValue));
                                        updateCurrentPlot(app);

                                    %DC DC Current update
                                    elseif strcmp(varName, 'DCDCC')
                                        addDCDCCurrentValue(app,
string(varValue));
                                        updateCurrentPlot(app);
                                    end
                                else
                                    disp('Error: Variable name and value not
in expected format.');
                                end
                            else
                                    disp('Error: Data not in expected format.');
                            end
                        catch
                            disp(['Error processing data: ', dataStr]);
                        end
                end
            end
```

Future work

Live Race Engineer Interface

The interface can be improved upon by creating a sophisticated database structure and querying the data from it. This would allow for any kind of comparison plot to be created. Furthermore, instead of sending the measurements from the vehicle to the ground station they could be uploaded to an online database from

which the readings can be accessed by any number of race engineers and from anywhere, allowing more people to do various kinds of data analysis simultaneously.

Off-track data analysis

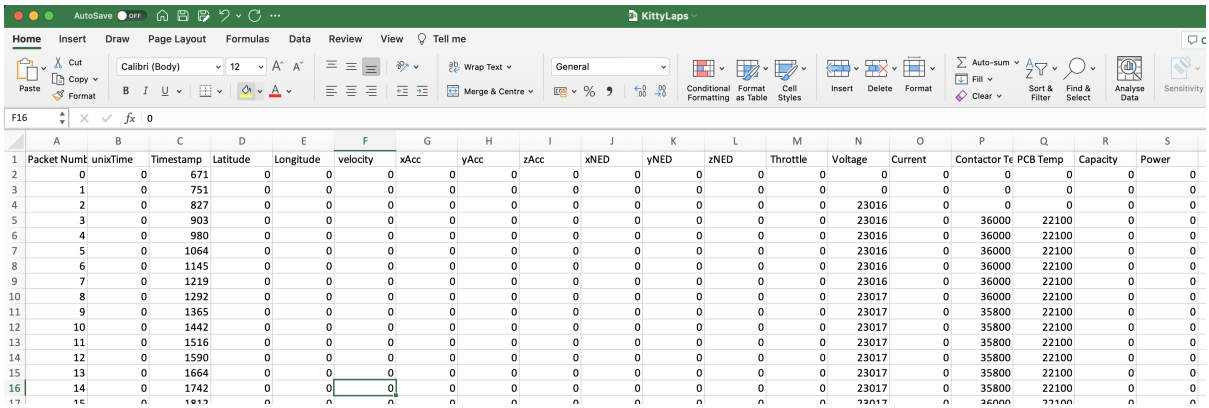
The following procedure applies to both hydrogen and battery electric MATLAB code, found on the Github repo.

The code first runs an overview of the testing data with the lap-by-lap performance. Then, the user inputs the lap numbers to allow for comparison of laps. Using the battery testing data, the tyre analysis can be performed automatically afterwards.

The hydrogen powertrain introduces further losses which I have not been able to estimate accordingly. In case of Hydrogen analysis, Milan can perform some simulations. His results can be plotted using the MilanPlots.mlx file.

Hydrogen and Battery Analysis Procedure:

1. Gather the testing data in excel and format it using the column order specific to each analysis. For this, please check the excel tables provided and match the title of the columns with the raw data from testing.



Packet Numb	unixTime	Timestamp	Latitude	Longitude	velocity	xAcc	yAcc	zAcc	xNED	yNED	zNED	Throttle	Voltage	Current	Contactor Te	PCB Temp	Capacity	Power
0	0	671	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	751	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	827	0	0	0	0	0	0	0	0	0	0	23016	0	0	0	0	0
3	0	903	0	0	0	0	0	0	0	0	0	0	23016	0	36000	22100	0	0
4	0	980	0	0	0	0	0	0	0	0	0	0	23016	0	36000	22100	0	0
5	0	1064	0	0	0	0	0	0	0	0	0	0	23016	0	36000	22100	0	0
6	0	1145	0	0	0	0	0	0	0	0	0	0	23016	0	36000	22100	0	0
7	0	1219	0	0	0	0	0	0	0	0	0	0	23016	0	36000	22100	0	0
8	0	1292	0	0	0	0	0	0	0	0	0	0	23017	0	36000	22100	0	0
9	0	1365	0	0	0	0	0	0	0	0	0	0	23017	0	35800	22100	0	0
10	0	1442	0	0	0	0	0	0	0	0	0	0	23017	0	35800	22100	0	0
11	0	1516	0	0	0	0	0	0	0	0	0	0	23017	0	35800	22100	0	0
12	0	1590	0	0	0	0	0	0	0	0	0	0	23017	0	35800	22100	0	0
13	0	1664	0	0	0	0	0	0	0	0	0	0	23017	0	35800	22100	0	0
14	0	1742	0	0	0	0	0	0	0	0	0	0	23017	0	35800	22100	0	0

Example of an excel sheet with telemetry data columns.

{% hint style="info" %} This data arrangement was specifically made for a hydrogen powertrain with the sensor mentioned in the 2024 configuration section.

If you are changing the sensors, you will need to modify the MATLAB code to account for that. Another (future work) solution is to make the plotting of the data automatic where if you upload a folder sectioned into multiple sensor folders from the receiver SD card,

e.g.

- config (folder)
  - README.txt
- sensor 1 (folder)
  - telemetry_date_time.txt
- sensor 2 (folder)
  - telemetry_date_time.txt

... then the program will detect and plot a graph for each sensor folder. {% endhint %}

2. Choose either the hydrogen or the electrical battery MATLAB code for analysis. In the first few lines, change the command to fit the correct file name: data = readtable('RandomName.xlsx');
3. Follow the command window and input the two lap numbers to allow for the comparison of laps.
4. All the results are now displayed. The next step is to run Milan Plots or the Tyre Analysis.

[^3]: Timer function callback, allows the plots to be continuously updated.

[^4]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^5]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^6]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^7]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^8]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^9]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^10]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^11]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^12]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^13]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^14]: Converts the Teleplot formatted measurement into VarName and VarValue.

[^15]: Checks whether VarName is “Lap#” and if so increments to lap number.

[^16]: Checks whether VarName is “Lap#” and if so increments to lap number.

[^17]: Checks whether VarName is “Lap#” and if so increments to lap number.

[^18]: Compares the VarName with the a predefined variable name and stores it VarValue to the corresponding cell array.

[^19]: Updates the plots.

[^20]: Compares the VarName with the a predefined variable name and stores it VarValue to the corresponding cell array.

[^21]: Updates the plots.

[^22]: Compares the VarName with the a predefined variable name and stores it VarValue to the corresponding cell array.

[^23]: Updates the plots.

[^24]: Compares the VarName with the a predefined variable name and stores it VarValue to the corresponding cell array.

[^25]: Updates the plots.

[^26]: Compares the VarName with the a predefined variable name and stores it VarValue to the corresponding cell array.

[^27]: Updates the plots.

[^28]: Compares the VarName with the a predefined variable name and stores it VarValue to the corresponding cell array.

[^29]: Updates the plots.

[^30]: Compares the VarName with the a predefined variable name and stores it VarValue to the corresponding cell array.

[^31]: Updates the plots.

[^32]: Compares the VarName with the a predefined variable name and stores it VarValue to the corresponding cell array.

[^33]: Updates the plots.