Virtual oscilloscope project using Matlab®

Mateus Valverde Gasparino

May 17, 2020

1 Objectives

The objective of this project is to gain experience in analyzing signals, as well as in developing a GUI based tool for visualization of data based on a virtual oscilloscope.

2 Introduction

Practical oscilloscopes were invented in the early twentieth century as there was a dire need to enable engineers to visualize signals with high frequecies. Today all oscilloscopes are digital, capable of sampling a signal with very high sampling frequency in the Giga (10^9) samples per second among four channels. They are also capable of performing calculations on the signal such as the DC and AC_RMS components, peak-peak/amplitude, rise/fall times, period/frequency as well as a Fast Fourier Transform. In this project, you are going to build a simulated version of a digital oscilloscope, see Fig. 2 using Matlab's app designer.

3 Equipment

To acquire a signal, we will connect an output from a signal generator to an analog input port of the National Instruments (NI) 6009. The signal from the generator can be a sine wave, square wave or a triangular wave, superimposed on a DC offset.

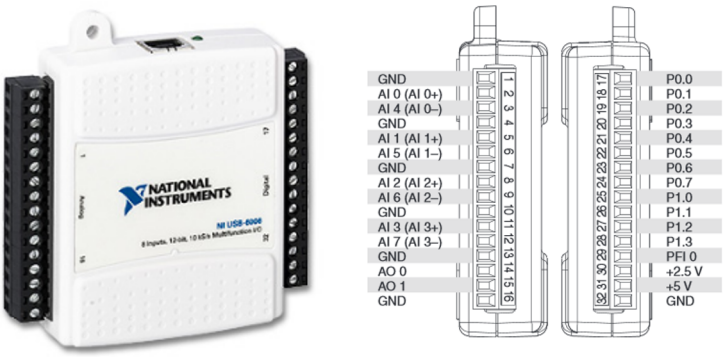

Figure 1: National Instruments NI6009 data acquisition module. Full specs can be found here.

4 Programming

In this project we are using Matlab, a powerful computation and visualization tool used in industry and academia. Matlab® has a data acquisition toolbox that makes use of National Instruments' NIDAQ drivers that allow it to communicate with hardware. It also has a graphical programming environment called Appdesigner.

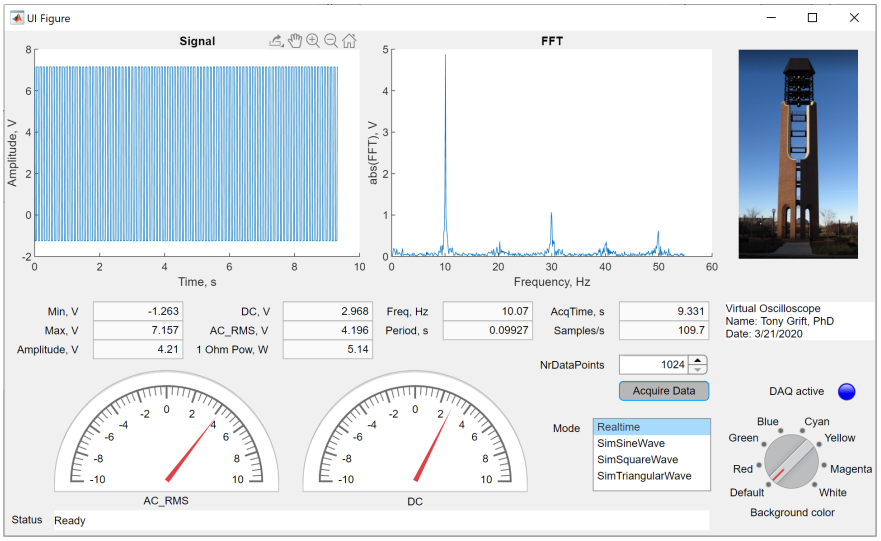

Figure 2: Example design view of the Virtual Oscilloscope.

The Matlab's Appdesigner environment was used to create a virtual oscilloscope app based on the example given by the professor, as shown in Fig. 2. The app should be able to get real-time data from the NI DAQ and process the data to calculate useful information from the signal, similar to a real oscilloscope.

5 Submission

My app was created with a very different design from Professor's example. I tried to design it to have a clean appearance, with less features in a single page. Therefore, I divided the App in six pages (or tabs), so the user can navigate between them.

The idea of dividing the app in pages came from cell phone apps, which try to make it easier for the user to understand the program's functionality by dividing the different functions in different pages.

Because of it, this app has a large number of objects and functions, so I'll describe them by dividing the explanation for each different page. Figure 3 shows all the objects used in this App.

Figure 3: Component browser showing the list of UI components.

5.1 Properties

The properties that are being used in this app are:

```
properties (Access = private)
    kMax = 1000;
    DaqSession
    SimSignal   = load('SineWaveData_8192.dat');
    simTime     = 0:0.00929:8192*0.00929;
    simArray    = zeros(1, 1000);
    RTSignal    = zeros(1, 1000);
    timeArray   = zeros(1, 1000);
    SamplingRate = 1000;
    filterType  = 'low';
    freqArray
    Signal
    FFTSignal
    Filter
    FilteredSignal
    FilteredFFT
    analogChannel = 1;
    PlotFFTwFilter = false;
    PlotFilteredSignal = false;
    PlotFilteredFFT = false;
    GREEN       = [0 1 0];
    RED         = [1 0 0];
end
```

Where **DaqSession** is the property that carries the National Instruments methods to acquire data. **SimSignal** is the property that contains the signal from the recorded data. **simTime** is the time array in respect to the recorded data. **simArray** is the array that contains part of the recorded data with number of points as specified by the user. **RTSignal** is the array that contains the real time signal with number of points as specified by the user. **timeArray** is the array that contains the timestamps of data that are plotted. **SamplingRate** is the sampling rate specified by the user. **filterType** is the filter type specified by user. **freqArray** is the array of frequencies to be plot after FFT calculation. **Signal** is the signal in time space to be plotted.

FFTSignal is the FFT transformation of the signal to be plotted. **Filter** is the filter model to be plotter and used to filter the FFT signal. **FilteredSignal** is the signal after filtered by filter and transformed back to the time domain. **FilteredFFT** is the FFT transformed signal after being filtered. **analogChannel** is the DAq analog channel chosen by user. **PlotFFTwFilter** is a bool variable that is true if the user wants to plot FFT and filter at the same time in the Filter page. **PlotFilteredSignal** is flag variable that is true if the user wants to plot the filtered signal in the same axes as the input signal. **PlotFilteredFFT** is a flag that is true if the user wants to plot the filtered FFT in the same axes as the non-filtered FFT in the FFT page. **GREEN** is the value representation for green color. **RED** is the value representation for red color.

5.2 The pages layout

In order to simplify the number of features in a single page, I chose a layout organized in tabs to design my app. The GUI object used to perform this kind of design is the tab panel, where I put the tabs in the top of the panel and hid them outside the app window space, so the user don't have access to it.

Instead of using the tabs to choose the pages, I opted to use states buttons, because in my opinion they look better, are more flexible to be customized and they look more similar to a real digital oscilloscope with lateral buttons. Therefore, to make them change when the user hits a button, I needed to create callbacks functions, one for each button, where for each button pressed, all the other buttons should be unpressed and the accordingly tab selected.

The following code snippet shows how this artifice using callbacks works:

```
% Value changed function: MainButton
function MainButtonValueChanged(app, event)
    app.MainButton.Value = true;
    app.SignalButton.Value = false;
    app.FFTButton.Value = false;
    app.FilterButton.Value = false;
    app.SaveButton.Value = false;
    app.AboutButton.Value = false;
    app.TabGroup.SelectedTab = app.MainViewTab;
```

```
end

% Value changed function: SignalButton
function SignalButtonValueChanged(app, event)
    app.MainButton.Value = false;
    app.SignalButton.Value = true;
    app.FFTButton.Value = false;
    app.FilterButton.Value = false;
    app.SaveButton.Value = false;
    app.AboutButton.Value = false;
    app.TabGroup.SelectedTab = app.SignalTab;
end

% Value changed function: FFTButton
function FFTButtonValueChanged(app, event)
    app.MainButton.Value = false;
    app.SignalButton.Value = false;
    app.FFTButton.Value = true;
    app.FilterButton.Value = false;
    app.SaveButton.Value = false;
    app.AboutButton.Value = false;
    app.TabGroup.SelectedTab = app.FFTTab;
end

% Value changed function: FilterButton
function FilterButtonValueChanged(app, event)
    app.MainButton.Value = false;
    app.SignalButton.Value = false;
    app.FFTButton.Value = false;
    app.FilterButton.Value = true;
    app.SaveButton.Value = false;
    app.AboutButton.Value = false;
    app.TabGroup.SelectedTab = app.FilterTab;
end

% Value changed function: SaveButton
function SaveButtonValueChanged(app, event)
    app.MainButton.Value = false;
    app.SignalButton.Value = false;
```

```
    app.FFTButton.Value = false;
    app.FilterButton.Value = false;
    app.SaveButton.Value = true;
    app.AboutButton.Value = false;
    app.TabGroup.SelectedTab = app.SaveTab;
end
```

5.3 Main page

The main page is the initial page that is shown to the user. In this page, the user can select the type of data that will be used, if it will be in real time or simulated mode. For simulated data, it uses recorded data from a sinusoidal signal, a square signal or a saw signal, saved as *.dat* file.

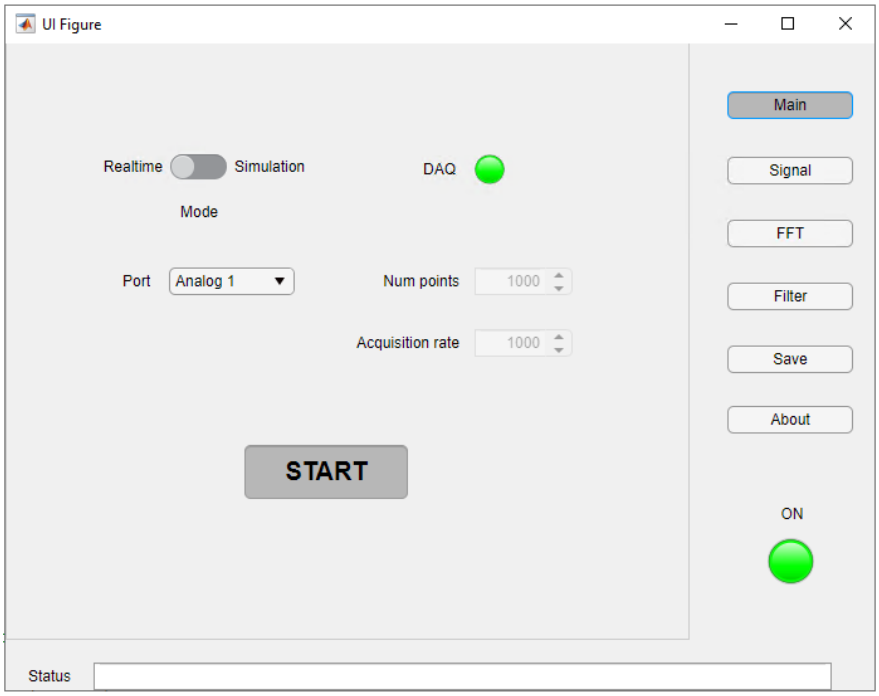

Figure 4: Main page.

The startup function is executed when the App is opened. In the startup function *clc* and *close all* are called to close and clean everything from Matlab. Then, a try/catch statement is called to try opening the Daq session, if an error occurs, then that means the DAQ is not connected to the computer, and the catch part is executed, turning the DAC lamp red and displaying in the status bar *"NI DAQ is not connected"*.

```
% Code that executes after component creation
function startupFcn(app)
    clc
    close all

    try
        app.DaqSession = daq.createSession('ni');
        release(app.DaqSession)
        app.DaqSession.Rate = app.SamplingRate;

        % ========================= Setup Analog Input ==============================
        Ch_input = app.DaqSession.addAnalogInputChannel('Dev1',0:3,'Voltage');
        set(Ch_input,'TerminalConfig','SingleEnded');
        app.DACLamp.Color = app.GREEN;
    catch
        app.StatusEditField.Value = 'NI DAQ is not connected';
        app.DACLamp.Color = app.RED;
    end
end
```

In the main page, there is a **START** button that takes care of the main loop. When the **START** button is pressed or unpressed, a callback function is called. If the button is pressed, **app.OFFLamp** turns green and its label displays "ON", however, if the button is unpressed, the **app.OFFLamp** turns red and its label displays "OFF". Whenever the callback function is called the **mainLoop()** function is called.

```
% Value changed function: STARTButton
function STARTButtonValueChanged(app, event)
    value = app.STARTButton.Value;
    if(value)
        app.OFFLamp.Color = app.GREEN;
        app.OFFLampLabel.Text = "ON";
        app.AcquisitionrateSpinner.Enable = 'off';
        app.NumpointsSpinner.Enable = 'off';
    else
        app.OFFLamp.Color = app.RED;
        app.OFFLampLabel.Text = "OFF";
```

```
        app.AcquisitionrateSpinner.Enable = 'on';
        app.NumpointsSpinner.Enable = 'on';
    end

    app.mainLoop(value);
end
```

The main loop function is a function that keeps running while the **START** button is pressed:

```
function mainLoop(app, value)
    try
        while value
            value = app.STARTButton.Value;
            app.PlotSignal();
            drawnow();
        end
    catch
        disp('App was closed.')
    end
end
```

The function has a try/catch statement to avoid errors when the app is closed while the while loop is running. The loop runs while $value = true$, in other words, while the button is pressed. While running, the loop keeps calling the function **app.PlotSignal()**, that is a very long function to perform all the calculations and plots. It also calls **drawnow()** to make sure everything is plotted and the program still have access to the callback even when inside the main loop.

The **app.PlotSignal()** function is responsible to make all the calculations and I chose to use a loop function because I wanted my app to work just like a real oscilloscope, that keeps updating the data in real-time while a signal is being fed to the data acquisition device.

The first part of the **app.PlotSignal()** function is used to decide which signal will be displayed, if it is the real time-signal or the simulated signal. In case of the real-time signal, the **startForeground(app.DaqSession)** is used. The function returns two important array: the data array and the timestamp array. The data array is a matrix with dimension equal to the number

of ports by the number of points. So, in the next line I copy only the line referent to the input port chosen by the user. Finally, the data is copied to the **app.Signal** property and timestamps are copied to **app.timeArray**. For the real-time signal, the **app.SamplingRate** comes from the user input.

If the simulated signal is selected, then we copy from **app.SimSignal** and **app.timeArray** only the number of points chosen by the user to the same **app.Signal** and **app.timeArray**, respectively. For the simulated signal, the **app.SamplingRate** is always 107.64, which was given beforehand.

```
if(strcmp(app.ModeSwitch.Value, 'Realtime'))
    %input_data = app.DaqSession.inputSingleScan;
    [data,timestamps,~] = startForeground(app.DaqSession);

    app.RTSignal = data(:,app.analogChannel);
    app.timeArray = timestamps;
    %app.Signal = data;
    app.Signal = app.RTSignal;

    % Sampling frequency
    app.SamplingRate = app.DaqSession.Rate;
else
    app.simArray = app.SimSignal(1:app.kMax);
    app.timeArray = app.simTime(1:app.kMax);

    app.Signal = app.simArray;

    % Sampling frequency
    app.SamplingRate = 107.64;
end
```

Other important parts inside this function will be explained in the following sections.

5.4 Signal page

The signal page is the most important page to analyze the data. It shows the input data and all the calculated values for the signal. Figure 5 shows the real-time signal input, acquired using the computer with IP 128.174.141.57, while Figure 6 shows the simulated input using the sinusoidal signal given by the professor.

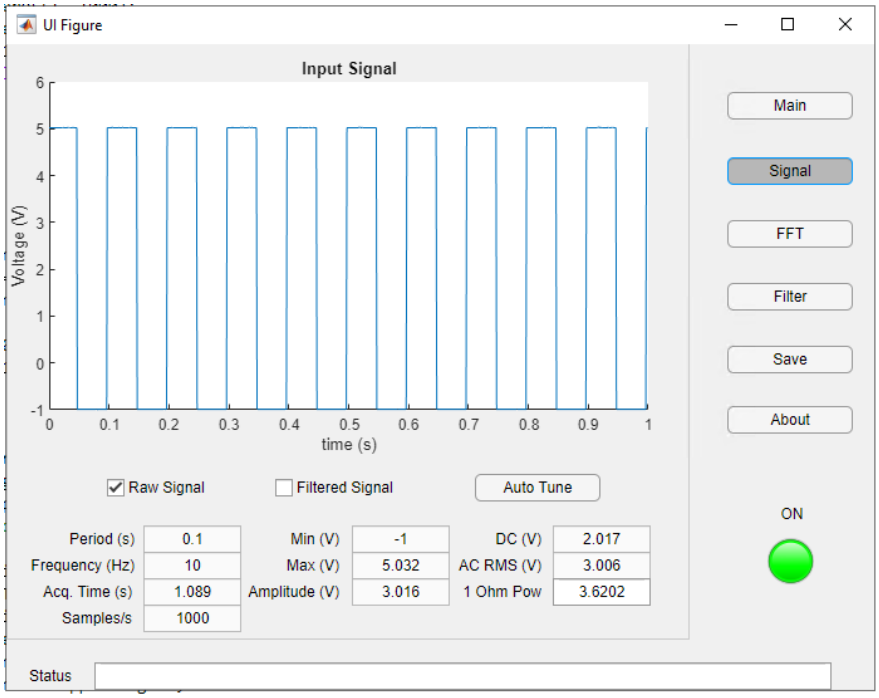

Figure 5: Signal page with the real time signal.

To perform all the calculations , the following piece of code is executed:

```
% Signal length
L = length(app.Signal);

app.FFTSignal = fft(app.Signal);
app.freqArray = app.SamplingRate*(0:L-1)/L;

% Make sure filter was created according the input signal
app.createFilter();

%app.FFTSignal
```

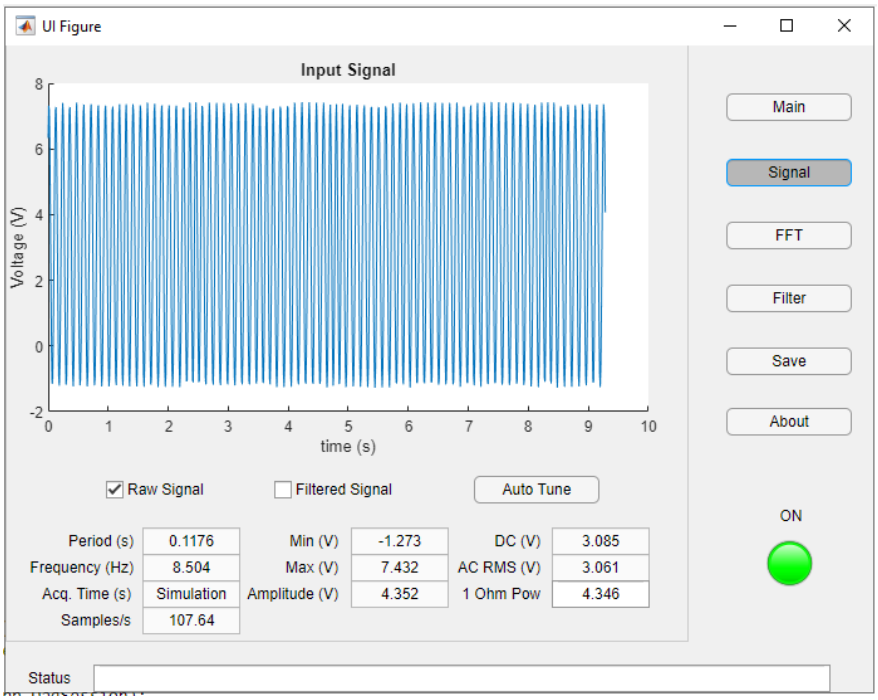

Figure 6: Signal page with the sinusoidal simulated signal.

```
%app.Filter
%size(app.FFTSignal(:).*app.Filter(:))

app.FilteredFFT = app.FFTSignal(:).*app.Filter(:);
app.FilteredSignal = ifft(app.FilteredFFT);

%% Signal tab
if(app.TabGroup.SelectedTab == app.SignalTab)
    y = [];

    if(app.RawSignalCheckBox.Value)
    y = [y, app.Signal];
    end

    if(app.FilteredSignalCheckBox.Value)
    y = [y, app.FilteredSignal];
    end

    if(~isempty(y))
    plot(app.UIAxes, app.timeArray, y);
```

```
    end

    % Frequency calculation
    [pks, locs] = findpeaks(abs(app.FFTSignal));
    freq_peak = max(pks(locs > 1));
    freq = app.freqArray(min(locs(pks==freq_peak)));
    if(length(pks) <= 1)
        freq = 0;
    end
    app.FrequencyHzEditField.Value = freq;

    % Period calculation
    app.PeriodsEditField.Value = 1/freq;

    % Amplitude calculation
    app.AmplitudeVEditField.Value = (max(app.Signal) - min(app.Signal))/2;
    app.MaxVEditField.Value = max(app.Signal);
    app.MinVEditField.Value = min(app.Signal);

    % DC value
    app.DCVEditField.Value = mean(app.Signal);

    % AC RMS
    app.ACRMSVEditField.Value = sqrt(mean((app.Signal - mean(app.Signal)).*
        (app.Signal - mean(app.Signal))));
end
```

First, some important variables are calculated: the signal length array **L**, the **app.FFTSignal** from the Fourier Transform using the Matlab's FFT function on the input signal **app.Signal**, and the **app.freqArray** by knowing the sampling rate and creating a array from 0 Hz to (Sampling Rate * signal length-1) Hz. The sampling rate is chosen by the user, using **app.AcquisitionrateSpinner** in the Main Page.

Then, to save processing time, I divided the function using *if* statements, that are executed depending on the selected tab. So, if the **Main Tab** is selected, then it'll only calculate and plot the values related to the Signal Page.

So, to perform the plot, first we check the two boxes: **app.RawSignalCheckBox** and

app.FilteredSignalCheckBox. If **app.RawSignalCheckBox** is checked, then it concatenates the raw signal to a variable **y**, and if **app.FilteredSignalCheckBox** is checked, the filtered signal is also concatenated to the same variable. This alllows me to plot only the signal the user wants to visualize. To plot the signals, I used the Matlab's plot function with **app.UIAxes** as parameter, which is the plot window located in the Signal Page.

Next, the usual oscilloscope measurements are calculated and plotted using edit fields. For the signal frequency, I used the Fourier transform and analyzed the peaks using the Matlab's function **findpeaks()**. Then, ignoring the frequency 0 Hz, which is the DC component, I used the **max()** function to find the maximum amplitude of the peaks. The frequency representing this peak is the signal's frequency, and it's displayed using the **app.FrequencyHzEditField**.

```
% Frequency calculation
[pks, locs] = findpeaks(abs(app.FFTSignal));
freq_peak = max(pks(locs > 1));
freq = app.freqArray(min(locs(pks==freq_peak)));
if(length(pks) <= 1)
    freq = 0;
end
app.FrequencyHzEditField.Value = freq;
```

Next, the period is calculated simply by $1/Frequency$, and it's displayed using the **app.PeriodsEditField**.

```
% Period calculation
app.PeriodsEditField.Value = 1/freq;
```

Amplitude, **Maximum Voltage** and **Minimum Voltage** are calculated together. Maximum voltage uses the max function to get the highest voltage in the signal, while minimum voltage uses the min function to get the lowest voltage. Amplitude is calculated from the half of the difference between maximum and minimum voltages, as shown in the following code snippet. These calculated values are displayed using **app.AmplitudeVEditField**, **app.MaxVEditField** and **app.MinVEditField** objects.

```
% Amplitude calculation
app.AmplitudeVEditField.Value = (max(app.Signal) - min(app.Signal))/2;
app.MaxVEditField.Value = max(app.Signal);
app.MinVEditField.Value = min(app.Signal);
```

The **DC** value is calculated using Matlab's mean() function, which calculates the average of the input signal. After calculating the DC value, we can subtract it from the input signal, and calculate the RMS value using its definition equation for discrete signals:

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} y_i}$$

```
% DC value
app.DCVEditField.Value = mean(app.Signal);
% AC RMS
app.ACRMSVEditField.Value = sqrt(mean((app.Signal - mean(app.Signal)).*(app.Signal - mean(app.Signal)
```

To calculate the power dissipated in a 1ω resistor, we sum the DC and AC RMS voltages squared and takes its square root, since:

$$Pow = \frac{U^2}{R} = \frac{U_{DC}^2 + U_{ACRMS}^2}{1}$$

The last feature in the signal page is the **Auto Tune** button. When pressed, it calls a callback function associated with the button, which changes the axis limits of **app.UIAxes** one period of the wave multiplied by 1.1 to guarantee one entire period of wave can be seen.

```
% Value changed function: AutoTuneButton
function AutoTuneButtonValueChanged(app, event)
    value = app.AutoTuneButton.Value;
    if(value)
        app.UIAxes.XLim = [0 app.PeriodsEditField.Value*1.1];
    else
        app.UIAxes.XLim = [0 inf];
```

```
    end
end
```

5.5 FFT page

The FFT page is the page used to plot the Fourier transform of the input signal and the filtered signal. As shown before, the Fourier transform of the signal is calculated using Matlab's fft() function and the Fourier transform of the filtered signal is calculated by the multiplication a filter (that will be presented in the next section) by the Fourier transform of the input signal. The following code snippet shows how the operations are performed:

```
app.FFTSignal = fft(app.Signal);
app.freqArray = app.SamplingRate*(0:L-1)/L;

% Make sure filter was created according the input signal
app.createFilter();

app.FilteredFFT = app.FFTSignal(:).*app.Filter(:);
```

Then, with both FFTs calculated, it is possible to plot them using an UIAxis. First, an *if* structure checks if the tab **app.TabGroup** is selected. If it is, then it checks if **app.FilteredFFTCheckBox** was selected. If the checkbox was not selected, then it only plots the non-filtered FFT, if it was, then it plots both filtered and non-filtered. The FFT signals are plotted using Matlab's plot() function using the **app.UIAxes_2**, which is the plot window located in the FFT page.

```
%% FFT tab
if(app.TabGroup.SelectedTab==app.FFTTab)
    if(app.PlotFilteredFFT)
        plot(app.UIAxes_2, app.freqArray, abs(app.FFTSignal)/L, app.freqArray, abs(app.FilteredFFT)/L
    else
        plot(app.UIAxes_2, app.freqArray, abs(app.FFTSignal)/L);
    end
end
```

Figure 7 shows the Fourier transform of the real-time input signal.

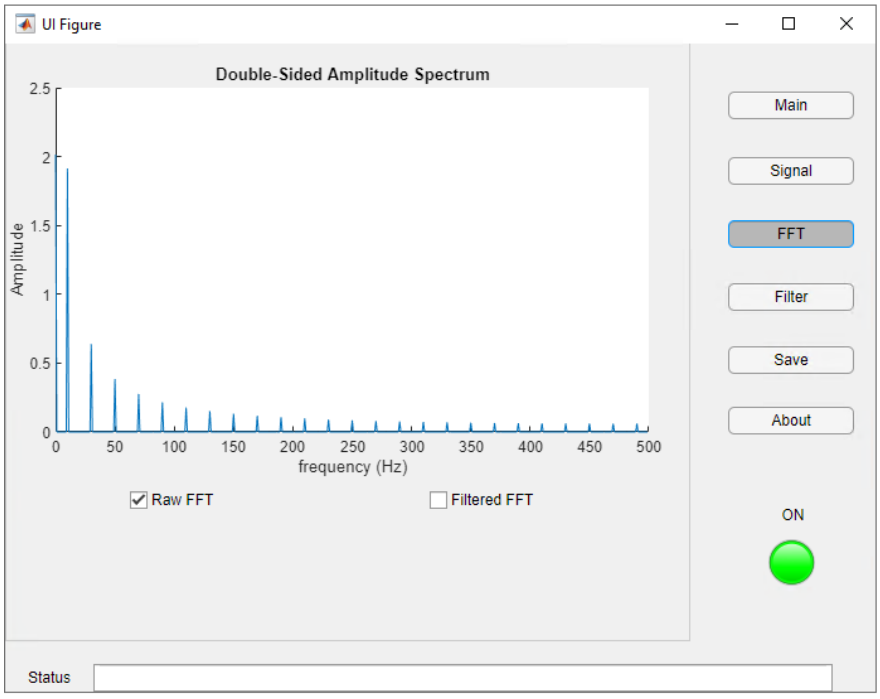

Figure 7: FFT page showing the Fourier transform of the input real-time square wave signal.

5.6 Filter page

The last page with features to analyze and process the input signal is the **Filter** page. In the Filter page it is possible to design a low-pass or high-pass filter using one of the models available. As shown in the FFT page section, the function **app.createFilter()** is constantly being called to create the filter and its functionality is shown in the following code snippet:

```
function createFilter(app)
    cutoffFreq = app.CutoffFreqSpinner.Value;
    filterOrder = app.OrderSpinner.Value;
    % Limit cutoff frequence
    cutoffFreq = min(cutoffFreq, length(app.FFTSignal)/2);

    if(strcmp(app.ModelDropDown.Value, 'Brick'))
        if(strcmp(app.filterType, 'low'))
            app.Filter = [ones(1,round(cutoffFreq)), zeros(1,length(app.FFTSignal)-2*round(cutoffFrec
```

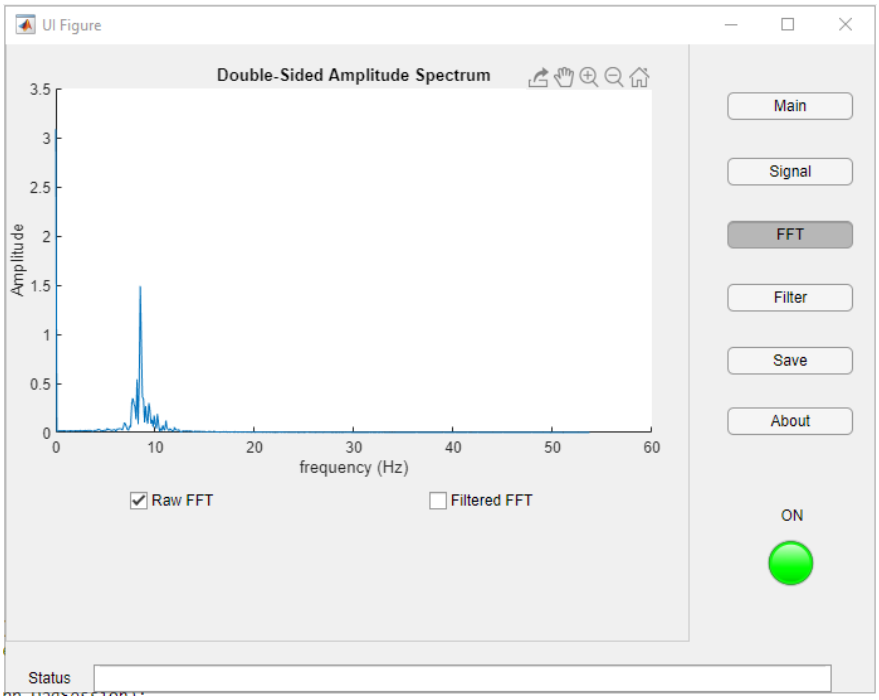

Figure 8: FFT page with the Fourier transform of the simulated sine wave signal.

```
        elseif(strcmp(app.filterType, 'high'))
            app.Filter = [zeros(1,round(cutoffFreq)), ones(1,length(app.FFTSignal)-2*round(cutoffFreq
        end

    elseif(strcmp(app.ModelDropDown.Value, 'Butterworth'))
        [b,a] = butter(filterOrder, cutoffFreq/(app.SamplingRate/2), app.filterType);
        h = freqz(b,a,app.freqArray/app.freqArray(end)*2*pi);
        app.Filter = h(:);

    elseif(strcmp(app.ModelDropDown.Value, 'Chebyshev'))
        [b,a] = cheby1(filterOrder, 3, cutoffFreq/(app.SamplingRate/2), app.filterType);
        h = freqz(b,a,app.freqArray/app.freqArray(end)*2*pi);
        app.Filter = h(:);
    end
end
```

So first, I get the cutoff frequency and the filter order. These values are inputs from user, using the **app.CutoffFreqSpinner** and the **app.OrderSpinner**. Then I make sure the cutoff frequency is not bigger than a half of the maximum frequency

```
cutoffFreq = min(cutoffFreq, length(app.FFTSignal)/2);
```

because this is the maximum frequency necessary to filter the signal, since the discrete Fourier transform of the signal is mirrored, and therefore it shows twice the frequencies of the input signal. then I chose three different filter models to use in this app: **brick filter**, **Butterworth** and **Chebyshev**.

Using a drop down list, the user can choose between one of these three filter models. An *if* structure is used to decide which part o the filter function is executed to create the filter, accordingly to the user's preferences. If the **brick filter** is chosen, then the following part of the code is executed:

```
if(strcmp(app.ModelDropDown.Value, 'Brick'))
    if(strcmp(app.filterType, 'low'))
        app.Filter = [ones(1,round(cutoffFreq)), zeros(1,length(app.FFTSignal)-2*round(cutoffFreq)),
            ones(1,round(cutoffFreq))];
    elseif(strcmp(app.filterType, 'high'))
        app.Filter = [zeros(1,round(cutoffFreq)), ones(1,length(app.FFTSignal)-2*round(cutoffFreq)),
            zeros(1,round(cutoffFreq))];
    end
```

Basically, it creates an array of ones and zeros with size equal to the FFT transform. The number of ones and zeros varies with the cutoff frequency. The user can also choose if the filter is a **low-pass** or **high-pass** filter. In case of a **low-pass** filter, the first condition is executed, otherwise, the second condition inverts the ones and zeros in the array. An example of brick low-pass filter is shown in Fig. 9 with the simulated sine wave signal.

The other two filters, **Butterworth** and **Chebyshev** are very similar. Using the Butterworth filter as example, Matlab has a function to create the filter, as shown in the code snippet:

```
[b,a] = butter(filterOrder, cutoffFreq/(app.SamplingRate/2), app.filterType);
```

Where **app.filterType** tells if the filter is a low-pass or high-pass filter. This option is input by user by **app.TypeDropDown_2**, and **app.filterType** changes the values according to

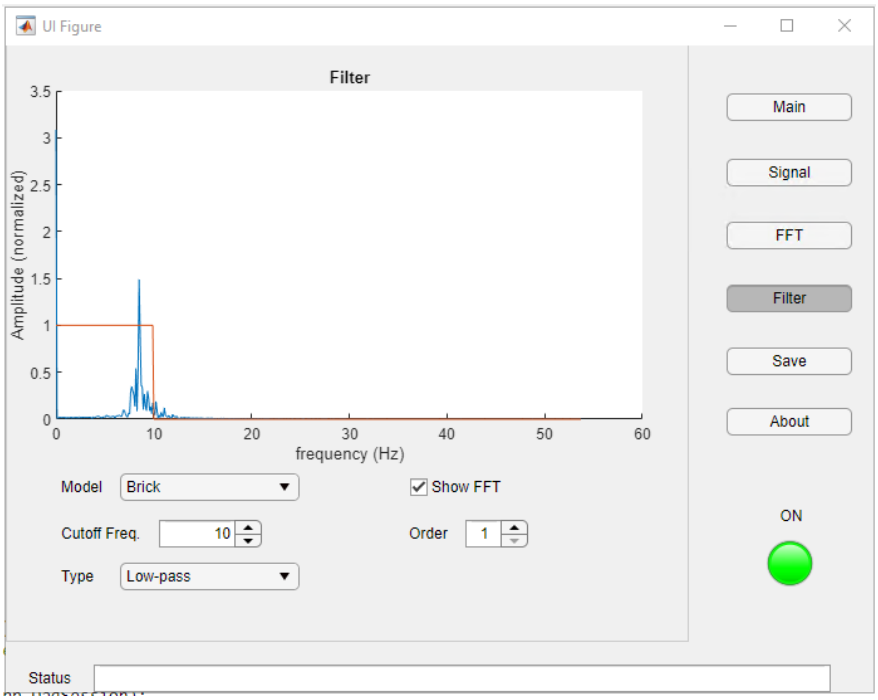

Figure 9: Filter page with the FFT of simulated sine wave signal with a brick low-pass filter with cutoff frequency of 10 Hz.

the following callback function:

```
% Value changed function: TypeDropDown_2
function TypeDropDown_2ValueChanged(app, event)
    value = app.TypeDropDown_2.Value;
    if(strcmp(value, 'Low-pass'))
        app.filterType = 'low';
    elseif(strcmp(value, 'High-pass'))
        app.filterType = 'high';
    end
end
```

The butter() function also needs the cutoff frequency, but a normalized cutoff frequency that goes from 0 to 1. To calculate the normalized cutoff frequency we need:

$$F_{normalized} = \frac{F}{SamplingRate/2}$$

Where F is the non-normalized cutoff frequency. The output of the butter function is the filter transfer function in form of numerator and denominator coefficients. Then, to transform them to an array of frequency response I used the Matlab's function **freqz()**, which parameters are the numerator and denominator coefficients of the filter transfer function, and the normalized frequency array in rad/s, from 0 to 2π, as shown in the following code snippet:

```
h = freqz(b,a,app.freqArray/app.freqArray(end)*2*pi);
```

Finally, as shown in the FFT page section, the filter is constantly multiplying the input FFT signal to calculate the filtered FFT. Both filter and non-filtered FFTs are plotted in **app.UIAxes_3**, similar to what happens with the FFT page. If the user checked the box to plot the non-filtered FFT, then the first condition is executed and both arrays are plotted, otherwise, only the filter is plotted.

```
%% Filter tab
if(app.TabGroup.SelectedTab==app.FilterTab)
    if(app.PlotFFTwFilter)
        plot(app.UIAxes_3, app.freqArray(1:length(app.FFTSignal)/2),
            abs(app.FFTSignal(1:length(app.FFTSignal)/2))/L,
            app.freqArray(1:length(app.FFTSignal)/2), app.Filter(1:length(app.FFTSignal)/2));
    else
        size(app.Filter)
        plot(app.UIAxes_3, app.freqArray(1:length(app.FFTSignal)/2),
            app.Filter(1:length(app.FFTSignal)/2));
    end
end
```

Figure 10 shows a Chevyshev filter designed to filter the real-time square wave input. The designed filter has a cutoff frequency of 50 Hz.

5.7 Save page

The **save page** is a tab to export the signal. It has only two objects: a text field and a button. The user can use the text field to write a filename to save the a .mat document containing all the

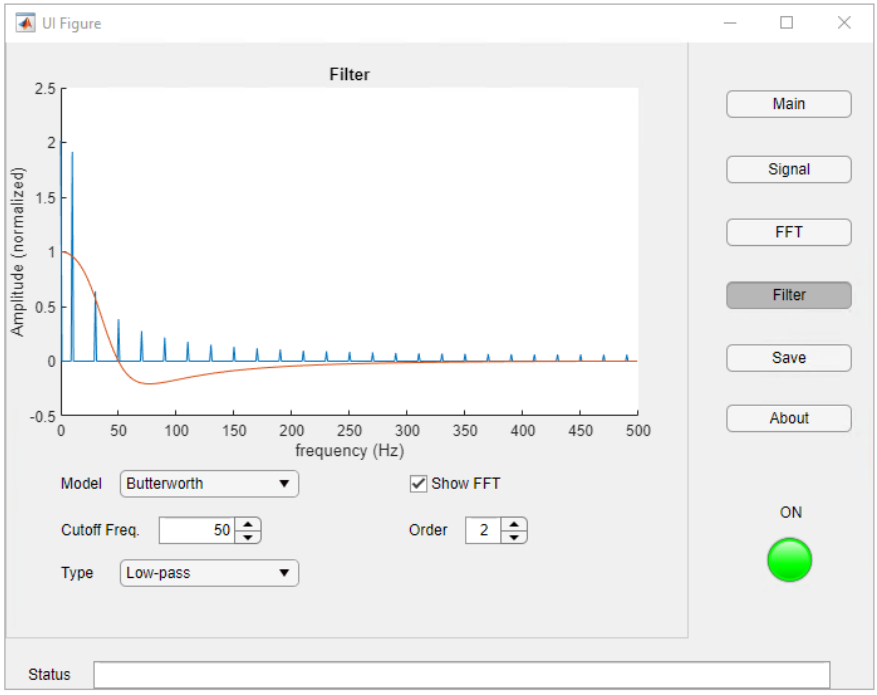

Figure 10: Filter page showing the FFT of the input square signal with a 2^{nd} order Butterworth low-pass filter with cutoff frequency of 50 Hz.

input signal points and its timestamps. By clicking on **Save**, a callback function is called:

```
function SaveButton_2Pushed(app, event)
    filename = app.FilenameEditField.Value + ".mat";
    time = app.timeArray;
    signal = app.Signal;
    save(filename, 'time', 'signal');
end
```

The function uses the **app.FilenameEditField.Value**, which is the filename input by user, and concatenates it with ".mat". Then the Matlab's function **save()** saves the input signal and timestamps to a *.mat* file with the chosen filename.

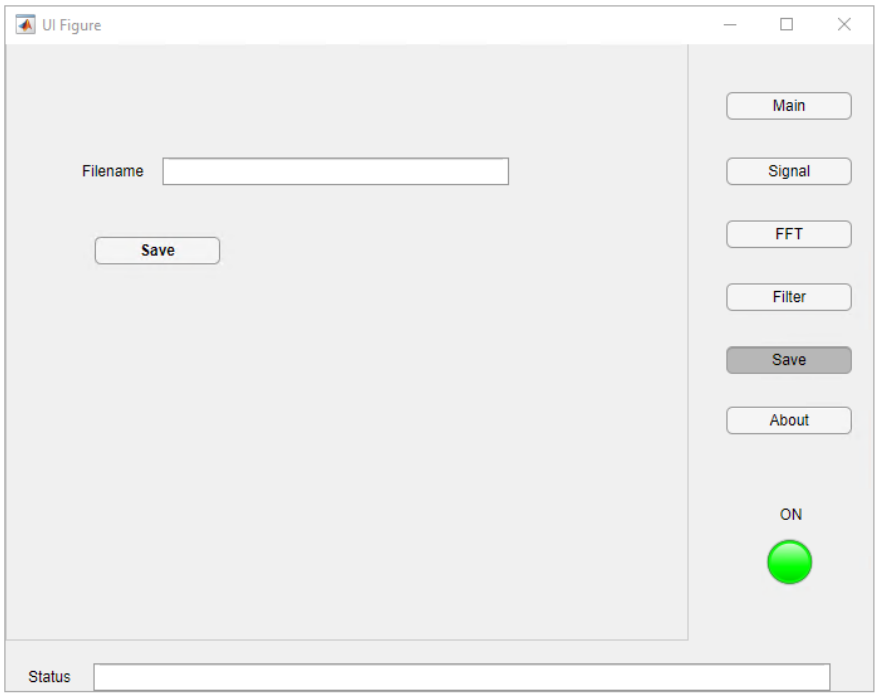

Figure 11: Save page.

5.8 About page

The **about page** is the last page in the app and it's the simplest one. There is a picture of me and there are some labels to display texts about my studies, to tell why the app was developed and show a link to my GitHub page: `github.com/matval/digital_oscilloscope`. After the project's submission deadline I'll post my code at this page, so anyone can download and use the app and contribute to make it better.

6 Conclusion

The app is capable of analyzing an input data, either in real-time mode or in simulated mode, where a *.dat* file containing the signal must be used to provide the data. The app can successfully get useful data from the signal wave, just as a real digital oscilloscope. Also, it presents different tools with a very different design, organized in pages, which simplifies the understanding for the user and provides a clean interface.

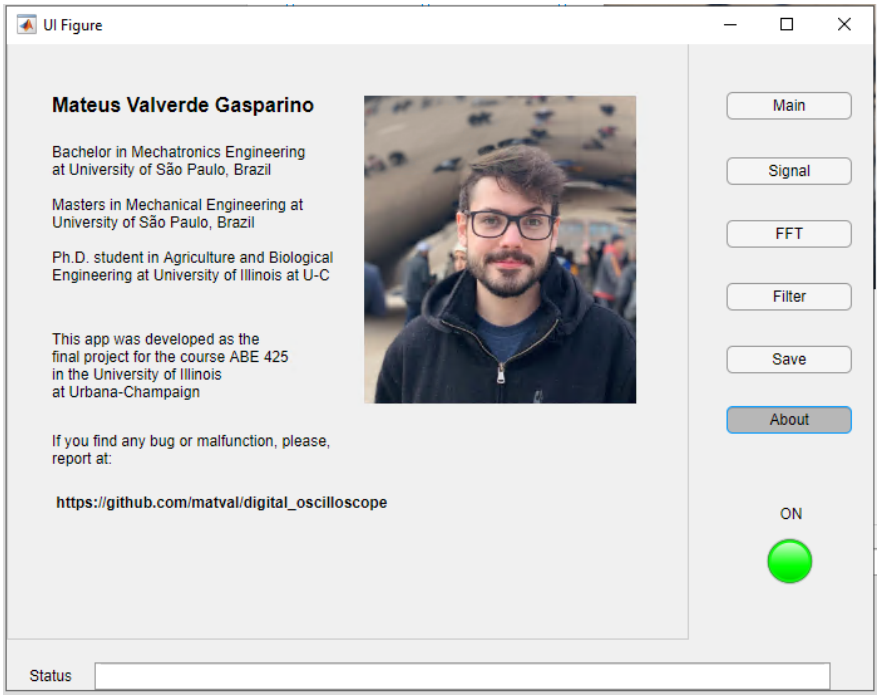

Figure 12: About page.

Although not encountered yet, some minor bugs are possible to happen, since it's still a prototype GUI with lots of lines of code, and not extensively tested. Therefore, several artifices to avoid crashing and bugs were used, such as the *try/catch* statements and *if* tests to verify if the input data and data structures are working as planned.

Even though, if any bug or malfunction is encountered, I link was provided to report fails, as well as get access to the algorithm.

7 Overall code

The overall code for the App:

```
classdef oscilloscope_project < matlab.apps.AppBase

% Properties that correspond to app components
properties (Access = public)
UIFigure                    matlab.ui.Figure
```

```
TabGroup                      matlab.ui.container.TabGroup
MainViewTab                   matlab.ui.container.Tab
DAQLampLabel                  matlab.ui.control.Label
DAQLamp                       matlab.ui.control.Lamp
PortDropDownLabel             matlab.ui.control.Label
PortDropDown                  matlab.ui.control.DropDown
ModeSwitchLabel               matlab.ui.control.Label
ModeSwitch                    matlab.ui.control.Switch
SimsignalDropDownLabel        matlab.ui.control.Label
SimsignalDropDown             matlab.ui.control.DropDown
STARTButton                   matlab.ui.control.StateButton
NumpointsSpinnerLabel         matlab.ui.control.Label
NumpointsSpinner              matlab.ui.control.Spinner
AcquisitionrateSpinnerLabel   matlab.ui.control.Label
AcquisitionrateSpinner        matlab.ui.control.Spinner
SignalTab                     matlab.ui.container.Tab
UIAxes                        matlab.ui.control.UIAxes
RawSignalCheckBox             matlab.ui.control.CheckBox
FilteredSignalCheckBox        matlab.ui.control.CheckBox
FrequencyHzEditFieldLabel     matlab.ui.control.Label
FrequencyHzEditField          matlab.ui.control.NumericEditField
AmplitudeVEditFieldLabel      matlab.ui.control.Label
AmplitudeVEditField           matlab.ui.control.NumericEditField
DCVEditFieldLabel             matlab.ui.control.Label
DCVEditField                  matlab.ui.control.NumericEditField
PeriodsEditFieldLabel         matlab.ui.control.Label
PeriodsEditField              matlab.ui.control.NumericEditField
MaxVEditFieldLabel            matlab.ui.control.Label
MaxVEditField                 matlab.ui.control.NumericEditField
MinVEditFieldLabel            matlab.ui.control.Label
MinVEditField                 matlab.ui.control.NumericEditField
ACRMSVEditFieldLabel          matlab.ui.control.Label
ACRMSVEditField               matlab.ui.control.NumericEditField
AutoTuneButton                matlab.ui.control.StateButton
OhmPowEditFieldLabel          matlab.ui.control.Label
OhmPowEditField               matlab.ui.control.EditField
AcqTimesEditFieldLabel        matlab.ui.control.Label
AcqTimesEditField             matlab.ui.control.EditField
SamplessEditFieldLabel        matlab.ui.control.Label
```

```
SamplessEditField           matlab.ui.control.EditField
FFTTab                      matlab.ui.container.Tab
UIAxes_2                    matlab.ui.control.UIAxes
RawFFTCheckBox              matlab.ui.control.CheckBox
FilteredFFTCheckBox         matlab.ui.control.CheckBox
FilterTab                   matlab.ui.container.Tab
UIAxes_3                    matlab.ui.control.UIAxes
ModelDropDownLabel          matlab.ui.control.Label
ModelDropDown               matlab.ui.control.DropDown
ShowFFTCheckBox             matlab.ui.control.CheckBox
CutoffFreqSpinnerLabel      matlab.ui.control.Label
CutoffFreqSpinner           matlab.ui.control.Spinner
TypeDropDown_2Label         matlab.ui.control.Label
TypeDropDown_2              matlab.ui.control.DropDown
OrderSpinnerLabel           matlab.ui.control.Label
OrderSpinner                matlab.ui.control.Spinner
SaveTab                     matlab.ui.container.Tab
FilenameEditFieldLabel      matlab.ui.control.Label
FilenameEditField           matlab.ui.control.EditField
SaveButton_2                matlab.ui.control.Button
AboutTab                    matlab.ui.container.Tab
Image                       matlab.ui.control.Image
label_name                  matlab.ui.control.Label
label_project               matlab.ui.control.Label
label_bug                   matlab.ui.control.Label
label_resume                matlab.ui.control.Label
label_git                   matlab.ui.control.Label
OFFLampLabel                matlab.ui.control.Label
OFFLamp                     matlab.ui.control.Lamp
StatusEditFieldLabel        matlab.ui.control.Label
StatusEditField             matlab.ui.control.EditField
MainButton                  matlab.ui.control.StateButton
SignalButton                matlab.ui.control.StateButton
FFTButton                   matlab.ui.control.StateButton
FilterButton                matlab.ui.control.StateButton
SaveButton                  matlab.ui.control.StateButton
AboutButton                 matlab.ui.control.StateButton
end
```

```
properties (Access = private)
kMax = 1000;
DaqSession
SimSignal   = load('SineWaveData_8192.dat');
simTime     = 0:0.00929:8192*0.00929;
simArray    = zeros(1, 1000);
RTSignal    = zeros(1, 1000);
timeArray   = zeros(1, 1000);
SamplingRate = 1000;
filterType  = 'low';
freqArray
Signal
FFTSignal
Filter
FilteredSignal
FilteredFFT
analogChannel = 1;
CutoffFreq  = inf;
PlotFFTwFilter = false;
PlotFilteredSignal = false;
PlotFilteredFFT = false;
GREEN       = [0 1 0];
RED         = [1 0 0];
acquisitionTime
end

methods (Access = private)
function PlotSignal(app)
if(strcmp(app.ModeSwitch.Value, 'Realtime'))
%input_data = app.DaqSession.inputSingleScan;
tic;
[data,timestamps,~] = startForeground(app.DaqSession);
app.AcqTimesEditField.Value = string(toc);
app.RTSignal = data(:,app.analogChannel);
app.timeArray = timestamps;
app.Signal = data;
app.Signal = app.RTSignal;
```

```
% Sampling frequency
app.SamplingRate = app.DaqSession.Rate;
else
app.simArray = app.SimSignal(1:app.kMax);
app.timeArray = app.simTime(1:app.kMax);

app.AcqTimesEditField.Value = 'Simulation';

app.Signal = app.simArray;

% Sampling frequency
app.SamplingRate = 107.64;
end

% Signal length
L = length(app.Signal);

app.FFTSignal = fft(app.Signal);
app.freqArray = app.SamplingRate*(0:L-1)/L;

app.SamplessEditField.Value = string(app.SamplingRate);

% Make sure filter was created according the input signal
app.createFilter();

%app.FFTSignal
%app.Filter
%size(app.FFTSignal(:).*app.Filter(:))

app.FilteredFFT = app.FFTSignal(:).*app.Filter(:);
app.FilteredSignal = ifft(app.FilteredFFT);

%% Signal tab
if(app.TabGroup.SelectedTab == app.SignalTab)
y = [];

if(app.RawSignalCheckBox.Value)
y = [y, app.Signal];
end
```

```
if(app.FilteredSignalCheckBox.Value)
y = [y, app.FilteredSignal];
end

if(~isempty(y))
plot(app.UIAxes, app.timeArray, y);
end

% Frequency calculation
[pks, locs] = findpeaks(abs(app.FFTSignal));
freq_peak = max(pks(locs > 1));
freq = app.freqArray(min(locs(pks==freq_peak)));
if(length(pks) <= 1)
freq = 0;
end
app.FrequencyHzEditField.Value = freq;

% Period calculation
app.PeriodsEditField.Value = 1/freq;

% Amplitude calculation
app.AmplitudeVEditField.Value = (max(app.Signal) - min(app.Signal))/2;
app.MaxVEditField.Value = max(app.Signal);
app.MinVEditField.Value = min(app.Signal);

% DC value
app.DCVEditField.Value = mean(app.Signal);

% AC RMS
app.ACRMSVEditField.Value = sqrt(mean((app.Signal - mean(app.Signal)).*(app.Signal - mean(app.Signal)
app.OhmPowEditField.Value = string(sqrt(app.DCVEditField.Value^2 + app.ACRMSVEditField.Value^2));
end

%% FFT tab
if(app.TabGroup.SelectedTab==app.FFTTab)
if(app.PlotFilteredFFT)
plot(app.UIAxes_2, app.freqArray(1:length(app.FFTSignal)/2), abs(app.FFTSignal(1:length(app.FFTSignal
else
```

```
plot(app.UIAxes_2, app.freqArray(1:length(app.FFTSignal)/2), abs(app.FFTSignal(1:length(app.FFTSignal
end
end

%% Filter tab
if(app.TabGroup.SelectedTab==app.FilterTab)
if(app.PlotFFTwFilter)
plot(app.UIAxes_3, app.freqArray(1:length(app.FFTSignal)/2), abs(app.FFTSignal(1:length(app.FFTSignal
else
size(app.Filter)
plot(app.UIAxes_3, app.freqArray(1:length(app.FFTSignal)/2), app.Filter(1:length(app.FFTSignal)/2));
end
end
end

function createFilter(app)
cutoffFreq = app.CutoffFreqSpinner.Value;
filterOrder = app.OrderSpinner.Value;
% Limit cutoff frequence
cutoffFreq = min(cutoffFreq, length(app.FFTSignal)/2);

brickSize = round(cutoffFreq*length(app.FFTSignal)/app.SamplingRate);

if(strcmp(app.ModelDropDown.Value, 'Brick'))
if(strcmp(app.filterType, 'low'))
app.Filter = [ones(1, brickSize), zeros(1,length(app.FFTSignal)-2*brickSize), ones(1, brickSize)];
elseif(strcmp(app.filterType, 'high'))
app.Filter = [zeros(1,round(cutoffFreq)), ones(1,length(app.FFTSignal)-2*round(cutoffFreq)), zeros(1,
end

elseif(strcmp(app.ModelDropDown.Value, 'Butterworth'))
[b,a] = butter(filterOrder, cutoffFreq/(app.SamplingRate/2), app.filterType);
h = freqz(b,a,app.freqArray/app.freqArray(end)*2*pi);
app.Filter = h(:);

elseif(strcmp(app.ModelDropDown.Value, 'Chebyshev'))
[b,a] = cheby1(filterOrder, 3, cutoffFreq/(app.SamplingRate/2), app.filterType);
h = freqz(b,a,app.freqArray/app.freqArray(end)*2*pi);
app.Filter = h(:);
```

```
end

end

function mainLoop(app, value)
try
while value
value = app.STARTButton.Value;
app.PlotSignal();
drawnow();
end
catch
disp('App was closed.')
end
end
end


% Callbacks that handle component events
methods (Access = private)

% Code that executes after component creation
function startupFcn(app)
clc
close all

try
app.DaqSession = daq.createSession('ni');
release(app.DaqSession)
app.DaqSession.Rate = app.SamplingRate;

% ========================== Setup Analog Input ==============================
Ch_input = app.DaqSession.addAnalogInputChannel('Dev1',0:3,'Voltage');
set(Ch_input,'TerminalConfig','SingleEnded');
app.DAQLamp.Color = app.GREEN;
catch
app.StatusEditField.Value = 'NI DAQ is not connected';
app.DAQLamp.Color = app.RED;
end
```

```
end

% Value changed function: MainButton
function MainButtonValueChanged(app, event)
app.MainButton.Value = true;
app.SignalButton.Value = false;
app.FFTButton.Value = false;
app.FilterButton.Value = false;
app.SaveButton.Value = false;
app.AboutButton.Value = false;
app.TabGroup.SelectedTab = app.MainViewTab;
end

% Value changed function: SignalButton
function SignalButtonValueChanged(app, event)
app.MainButton.Value = false;
app.SignalButton.Value = true;
app.FFTButton.Value = false;
app.FilterButton.Value = false;
app.SaveButton.Value = false;
app.AboutButton.Value = false;
app.TabGroup.SelectedTab = app.SignalTab;
end

% Value changed function: FFTButton
function FFTButtonValueChanged(app, event)
app.MainButton.Value = false;
app.SignalButton.Value = false;
app.FFTButton.Value = true;
app.FilterButton.Value = false;
app.SaveButton.Value = false;
app.AboutButton.Value = false;
app.TabGroup.SelectedTab = app.FFTTab;
end

% Value changed function: FilterButton
function FilterButtonValueChanged(app, event)
app.MainButton.Value = false;
app.SignalButton.Value = false;
```

```
app.FFTButton.Value = false;
app.FilterButton.Value = true;
app.SaveButton.Value = false;
app.AboutButton.Value = false;
app.TabGroup.SelectedTab = app.FilterTab;
end

% Value changed function: SaveButton
function SaveButtonValueChanged(app, event)
app.MainButton.Value = false;
app.SignalButton.Value = false;
app.FFTButton.Value = false;
app.FilterButton.Value = false;
app.SaveButton.Value = true;
app.AboutButton.Value = false;
app.TabGroup.SelectedTab = app.SaveTab;
end

% Value changed function: ModeSwitch
function ModeSwitchValueChanged(app, event)
value = app.ModeSwitch.Value;

if(strcmp(value, 'Realtime'))
app.PortDropDown.Visible = 'on';
app.PortDropDownLabel.Visible = 'on';
app.SimsignalDropDown.Visible = 'off';
app.SimsignalDropDownLabel.Visible = 'off';
app.AcquisitionrateSpinner.Enable = 'on';
else
app.PortDropDown.Visible = 'off';
app.PortDropDownLabel.Visible = 'off';
app.SimsignalDropDown.Visible = 'on';
app.SimsignalDropDownLabel.Visible = 'on';
app.AcquisitionrateSpinner.Enable = 'off';
end
end

% Value changed function: SimsignalDropDown
function SimsignalDropDownValueChanged(app, event)
```

```
value = app.SimsignalDropDown.Value;
%path = mfilename('fullpath');
%path = erase(path,"\oscilloscope_project");
path = '';

if(strcmp(value, 'Sine wave'))
fullPath = fullfile(path, 'SineWaveData_8192.dat');
elseif(strcmp(value, 'Square wave'))
fullPath = fullfile(path, 'SquareWaveData_8192.dat');
elseif(strcmp(value, 'Saw wave'))
fullPath = fullfile(path, 'TriangularWaveData_8192.dat');
end

signal = load(fullPath);
app.SimSignal = signal;
end

% Value changed function: STARTButton
function STARTButtonValueChanged(app, event)
value = app.STARTButton.Value;
if(value)
app.OFFLamp.Color = app.GREEN;
app.OFFLampLabel.Text = "ON";
app.AcquisitionrateSpinner.Enable = 'off';
app.NumpointsSpinner.Enable = 'off';
else
app.OFFLamp.Color = app.RED;
app.OFFLampLabel.Text = "OFF";
app.AcquisitionrateSpinner.Enable = 'on';
app.NumpointsSpinner.Enable = 'on';
end

app.mainLoop(value);
end

% Value changed function: ShowFFTCheckBox
function ShowFFTCheckBoxValueChanged(app, event)
value = app.ShowFFTCheckBox.Value;
app.PlotFFTwFilter = value;
```

```
end

% Value changed function: FilteredFFTCheckBox
function FilteredFFTCheckBoxValueChanged(app, event)
value = app.FilteredFFTCheckBox.Value;
app.PlotFilteredFFT = value;
end

% Value changed function: PortDropDown
function PortDropDownValueChanged(app, event)
value = app.PortDropDown.Value;

if(strcmp(value, 'Analog 0'))
app.analogChannel = 1;
elseif(strcmp(value, 'Analog 1'))
app.analogChannel = 2;
elseif(strcmp(value, 'Analog 2'))
app.analogChannel = 3;
elseif(strcmp(value, 'Analog 3'))
app.analogChannel = 4;
end
end

% Value changed function: NumpointsSpinner
function NumpointsSpinnerValueChanged(app, event)
value = app.NumpointsSpinner.Value;
app.kMax = value;
app.simArray = zeros(1, value);
app.RTSignal = zeros(1, value);
app.timeArray = zeros(1, value);

if(strcmp(app.ModeSwitch.Value, 'Realtime'))
if(app.STARTButton.Value)
app.StatusEditField.Value = "Please, stop acquisition data before changing number of points.";
else
app.DaqSession.NumberOfScans = value;
end
end
end
```

```
% Value changed function: AboutButton
function AboutButtonValueChanged(app, event)
app.MainButton.Value = false;
app.SignalButton.Value = false;
app.FFTButton.Value = false;
app.FilterButton.Value = false;
app.SaveButton.Value = false;
app.AboutButton.Value = true;
app.TabGroup.SelectedTab = app.AboutTab;
end

% Value changed function: AcquisitionrateSpinner
function AcquisitionrateSpinnerValueChanged(app, event)
value = app.AcquisitionrateSpinner.Value;
if(app.STARTButton.Value)
app.StatusEditField.Value = "Please, stop acquisition data before changing acquisition rate.";
else
while(~app.DaqSession.IsDone())
app.StatusEditField.Value = "Waiting for data acquisition to end the collection before changing acqui
pause(0.1);
end

app.StatusEditField.Value = "";
app.DaqSession.Rate = value;
end
end

% Button pushed function: SaveButton_2
function SaveButton_2Pushed(app, event)
filename = app.FilenameEditField.Value + ".mat";
time = app.timeArray;
signal = app.Signal;
save(filename, 'time', 'signal');
end

% Value changed function: TypeDropDown_2
function TypeDropDown_2ValueChanged(app, event)
value = app.TypeDropDown_2.Value;
```

```
if(strcmp(value, 'Low-pass'))
app.filterType = 'low';
elseif(strcmp(value, 'High-pass'))
app.filterType = 'high';
end
end

% Value changed function: AutoTuneButton
function AutoTuneButtonValueChanged(app, event)
value = app.AutoTuneButton.Value;
if(value)
app.UIAxes.XLim = [0 app.PeriodsEditField.Value*1.1];
else
app.UIAxes.XLim = [0 inf];
end
end
end

% Component initialization
methods (Access = private)

% Create UIFigure and components
function createComponents(app)

% Create UIFigure and hide until all components are created
app.UIFigure = uifigure('Visible', 'off');
app.UIFigure.Position = [100 100 691 514];
app.UIFigure.Name = 'UI Figure';

% Create TabGroup
app.TabGroup = uitabgroup(app.UIFigure);
app.TabGroup.Position = [1 41 544 501];

% Create MainViewTab
app.MainViewTab = uitab(app.TabGroup);
app.MainViewTab.Title = 'Main View';
app.MainViewTab.BackgroundColor = [0.9412 0.9412 0.9412];

% Create DAQLampLabel
```

```
app.DAQLampLabel = uilabel(app.MainViewTab);
app.DAQLampLabel.HorizontalAlignment = 'right';
app.DAQLampLabel.Position = [326 363 32 22];
app.DAQLampLabel.Text = 'DAQ';

% Create DAQLamp
app.DAQLamp = uilamp(app.MainViewTab);
app.DAQLamp.Position = [373 362 24 24];

% Create PortDropDownLabel
app.PortDropDownLabel = uilabel(app.MainViewTab);
app.PortDropDownLabel.HorizontalAlignment = 'right';
app.PortDropDownLabel.Position = [87 274 28 22];
app.PortDropDownLabel.Text = 'Port';

% Create PortDropDown
app.PortDropDown = uidropdown(app.MainViewTab);
app.PortDropDown.Items = {'Analog 0', 'Analog 1', 'Analog 2', 'Analog 3'};
app.PortDropDown.ValueChangedFcn = createCallbackFcn(app, @PortDropDownValueChanged, true);
app.PortDropDown.Position = [130 274 100 22];
app.PortDropDown.Value = 'Analog 0';

% Create ModeSwitchLabel
app.ModeSwitchLabel = uilabel(app.MainViewTab);
app.ModeSwitchLabel.HorizontalAlignment = 'center';
app.ModeSwitchLabel.Position = [136 329 36 22];
app.ModeSwitchLabel.Text = 'Mode';

% Create ModeSwitch
app.ModeSwitch = uiswitch(app.MainViewTab, 'slider');
app.ModeSwitch.Items = {'Realtime', 'Simulation'};
app.ModeSwitch.ValueChangedFcn = createCallbackFcn(app, @ModeSwitchValueChanged, true);
app.ModeSwitch.Position = [131 366 45 20];
app.ModeSwitch.Value = 'Realtime';

% Create SimsignalDropDownLabel
app.SimsignalDropDownLabel = uilabel(app.MainViewTab);
app.SimsignalDropDownLabel.HorizontalAlignment = 'right';
app.SimsignalDropDownLabel.Visible = 'off';
```

```
app.SimsignalDropDownLabel.Position = [54 225 61 22];
app.SimsignalDropDownLabel.Text = 'Sim signal';

% Create SimsignalDropDown
app.SimsignalDropDown = uidropdown(app.MainViewTab);
app.SimsignalDropDown.Items = {'Sine wave', 'Square wave', 'Saw wave'};
app.SimsignalDropDown.ValueChangedFcn = createCallbackFcn(app, @SimsignalDropDownValueChanged, true);
app.SimsignalDropDown.Visible = 'off';
app.SimsignalDropDown.Position = [130 225 100 22];
app.SimsignalDropDown.Value = 'Sine wave';

% Create STARTButton
app.STARTButton = uibutton(app.MainViewTab, 'state');
app.STARTButton.ValueChangedFcn = createCallbackFcn(app, @STARTButtonValueChanged, true);
app.STARTButton.Text = 'START';
app.STARTButton.FontSize = 20;
app.STARTButton.FontWeight = 'bold';
app.STARTButton.Position = [190 112 130 43];

% Create NumpointsSpinnerLabel
app.NumpointsSpinnerLabel = uilabel(app.MainViewTab);
app.NumpointsSpinnerLabel.HorizontalAlignment = 'right';
app.NumpointsSpinnerLabel.Position = [295 274 66 22];
app.NumpointsSpinnerLabel.Text = 'Num points';

% Create NumpointsSpinner
app.NumpointsSpinner = uispinner(app.MainViewTab);
app.NumpointsSpinner.Limits = [10 10000];
app.NumpointsSpinner.ValueChangedFcn = createCallbackFcn(app, @NumpointsSpinnerValueChanged, true);
app.NumpointsSpinner.Position = [373 274 80 22];
app.NumpointsSpinner.Value = 1000;

% Create AcquisitionrateSpinnerLabel
app.AcquisitionrateSpinnerLabel = uilabel(app.MainViewTab);
app.AcquisitionrateSpinnerLabel.HorizontalAlignment = 'right';
app.AcquisitionrateSpinnerLabel.Position = [273 225 88 22];
app.AcquisitionrateSpinnerLabel.Text = 'Acquisition rate';

% Create AcquisitionrateSpinner
```

```
app.AcquisitionrateSpinner = uispinner(app.MainViewTab);
app.AcquisitionrateSpinner.Limits = [10 10000];
app.AcquisitionrateSpinner.ValueChangedFcn = createCallbackFcn(app, @AcquisitionrateSpinnerValueChang
app.AcquisitionrateSpinner.Position = [373 225 80 22];
app.AcquisitionrateSpinner.Value = 1000;

% Create SignalTab
app.SignalTab = uitab(app.TabGroup);
app.SignalTab.Title = 'Signal';
app.SignalTab.BackgroundColor = [0.9412 0.9412 0.9412];

% Create UIAxes
app.UIAxes = uiaxes(app.SignalTab);
title(app.UIAxes, 'Input Signal')
xlabel(app.UIAxes, 'time (s)')
ylabel(app.UIAxes, 'Voltage (V)')
app.UIAxes.Position = [1 148 518 316];

% Create RawSignalCheckBox
app.RawSignalCheckBox = uicheckbox(app.SignalTab);
app.RawSignalCheckBox.Text = 'Raw Signal';
app.RawSignalCheckBox.Position = [80 110 83 22];
app.RawSignalCheckBox.Value = true;

% Create FilteredSignalCheckBox
app.FilteredSignalCheckBox = uicheckbox(app.SignalTab);
app.FilteredSignalCheckBox.Text = 'Filtered Signal';
app.FilteredSignalCheckBox.Position = [214 110 99 22];

% Create FrequencyHzEditFieldLabel
app.FrequencyHzEditFieldLabel = uilabel(app.SignalTab);
app.FrequencyHzEditFieldLabel.HorizontalAlignment = 'right';
app.FrequencyHzEditFieldLabel.Position = [14 48 88 22];
app.FrequencyHzEditFieldLabel.Text = 'Frequency (Hz)';

% Create FrequencyHzEditField
app.FrequencyHzEditField = uieditfield(app.SignalTab, 'numeric');
app.FrequencyHzEditField.Editable = 'off';
app.FrequencyHzEditField.HorizontalAlignment = 'center';
```

```
app.FrequencyHzEditField.Position = [109 48 78 22];

% Create AmplitudeVEditFieldLabel
app.AmplitudeVEditFieldLabel = uilabel(app.SignalTab);
app.AmplitudeVEditFieldLabel.HorizontalAlignment = 'right';
app.AmplitudeVEditFieldLabel.Position = [187 27 78 22];
app.AmplitudeVEditFieldLabel.Text = 'Amplitude (V)';

% Create AmplitudeVEditField
app.AmplitudeVEditField = uieditfield(app.SignalTab, 'numeric');
app.AmplitudeVEditField.Editable = 'off';
app.AmplitudeVEditField.HorizontalAlignment = 'center';
app.AmplitudeVEditField.Position = [275 27 78 22];

% Create DCVEditFieldLabel
app.DCVEditFieldLabel = uilabel(app.SignalTab);
app.DCVEditFieldLabel.HorizontalAlignment = 'right';
app.DCVEditFieldLabel.Position = [384 69 42 22];
app.DCVEditFieldLabel.Text = 'DC (V)';

% Create DCVEditField
app.DCVEditField = uieditfield(app.SignalTab, 'numeric');
app.DCVEditField.Editable = 'off';
app.DCVEditField.HorizontalAlignment = 'center';
app.DCVEditField.Position = [435 69 78 22];

% Create PeriodsEditFieldLabel
app.PeriodsEditFieldLabel = uilabel(app.SignalTab);
app.PeriodsEditFieldLabel.HorizontalAlignment = 'right';
app.PeriodsEditFieldLabel.Position = [44 69 58 22];
app.PeriodsEditFieldLabel.Text = 'Period (s)';

% Create PeriodsEditField
app.PeriodsEditField = uieditfield(app.SignalTab, 'numeric');
app.PeriodsEditField.Editable = 'off';
app.PeriodsEditField.HorizontalAlignment = 'center';
app.PeriodsEditField.Position = [109 69 78 22];

% Create MaxVEditFieldLabel
```

```
app.MaxVEditFieldLabel = uilabel(app.SignalTab);
app.MaxVEditFieldLabel.HorizontalAlignment = 'right';
app.MaxVEditFieldLabel.Position = [218 48 47 22];
app.MaxVEditFieldLabel.Text = 'Max (V)';

% Create MaxVEditField
app.MaxVEditField = uieditfield(app.SignalTab, 'numeric');
app.MaxVEditField.Editable = 'off';
app.MaxVEditField.HorizontalAlignment = 'center';
app.MaxVEditField.Position = [275 48 78 22];

% Create MinVEditFieldLabel
app.MinVEditFieldLabel = uilabel(app.SignalTab);
app.MinVEditFieldLabel.HorizontalAlignment = 'right';
app.MinVEditFieldLabel.Position = [221 69 44 22];
app.MinVEditFieldLabel.Text = 'Min (V)';

% Create MinVEditField
app.MinVEditField = uieditfield(app.SignalTab, 'numeric');
app.MinVEditField.Editable = 'off';
app.MinVEditField.HorizontalAlignment = 'center';
app.MinVEditField.Position = [275 69 78 22];

% Create ACRMSVEditFieldLabel
app.ACRMSVEditFieldLabel = uilabel(app.SignalTab);
app.ACRMSVEditFieldLabel.HorizontalAlignment = 'right';
app.ACRMSVEditFieldLabel.Position = [355 48 71 22];
app.ACRMSVEditFieldLabel.Text = 'AC RMS (V)';

% Create ACRMSVEditField
app.ACRMSVEditField = uieditfield(app.SignalTab, 'numeric');
app.ACRMSVEditField.Editable = 'off';
app.ACRMSVEditField.HorizontalAlignment = 'center';
app.ACRMSVEditField.Position = [435 48 78 22];

% Create AutoTuneButton
app.AutoTuneButton = uibutton(app.SignalTab, 'state');
app.AutoTuneButton.ValueChangedFcn = createCallbackFcn(app, @AutoTuneButtonValueChanged, true);
app.AutoTuneButton.Text = 'Auto Tune';
```

```
app.AutoTuneButton.Position = [373 110 100 22];

% Create OhmPowEditFieldLabel
app.OhmPowEditFieldLabel = uilabel(app.SignalTab);
app.OhmPowEditFieldLabel.HorizontalAlignment = 'right';
app.OhmPowEditFieldLabel.Position = [352 27 74 22];
app.OhmPowEditFieldLabel.Text = '1 Ohm Pow';

% Create OhmPowEditField
app.OhmPowEditField = uieditfield(app.SignalTab, 'text');
app.OhmPowEditField.HorizontalAlignment = 'center';
app.OhmPowEditField.Position = [435 27 78 22];
app.OhmPowEditField.Value = '0';

% Create AcqTimesEditFieldLabel
app.AcqTimesEditFieldLabel = uilabel(app.SignalTab);
app.AcqTimesEditFieldLabel.HorizontalAlignment = 'right';
app.AcqTimesEditFieldLabel.Position = [24 27 76 22];
app.AcqTimesEditFieldLabel.Text = 'Acq. Time (s)';

% Create AcqTimesEditField
app.AcqTimesEditField = uieditfield(app.SignalTab, 'text');
app.AcqTimesEditField.Editable = 'off';
app.AcqTimesEditField.HorizontalAlignment = 'center';
app.AcqTimesEditField.Position = [109 27 78 22];
app.AcqTimesEditField.Value = '0';

% Create SamplessEditFieldLabel
app.SamplessEditFieldLabel = uilabel(app.SignalTab);
app.SamplessEditFieldLabel.HorizontalAlignment = 'right';
app.SamplessEditFieldLabel.Position = [38 6 62 22];
app.SamplessEditFieldLabel.Text = 'Samples/s';

% Create SamplessEditField
app.SamplessEditField = uieditfield(app.SignalTab, 'text');
app.SamplessEditField.Editable = 'off';
app.SamplessEditField.HorizontalAlignment = 'center';
app.SamplessEditField.Position = [109 6 78 22];
app.SamplessEditField.Value = '0';
```

```
% Create FFTTab
app.FFTTab = uitab(app.TabGroup);
app.FFTTab.Title = 'FFT';

% Create UIAxes_2
app.UIAxes_2 = uiaxes(app.FFTTab);
title(app.UIAxes_2, 'Double-Sided Amplitude Spectrum')
xlabel(app.UIAxes_2, 'frequency (Hz)')
ylabel(app.UIAxes_2, 'Amplitude')
app.UIAxes_2.Position = [1 131 520 328];

% Create RawFFTCheckBox
app.RawFFTCheckBox = uicheckbox(app.FFTTab);
app.RawFFTCheckBox.Text = 'Raw FFT';
app.RawFFTCheckBox.Position = [99 101 71 22];
app.RawFFTCheckBox.Value = true;

% Create FilteredFFTCheckBox
app.FilteredFFTCheckBox = uicheckbox(app.FFTTab);
app.FilteredFFTCheckBox.ValueChangedFcn = createCallbackFcn(app, @FilteredFFTCheckBoxValueChanged, t
app.FilteredFFTCheckBox.Text = 'Filtered FFT';
app.FilteredFFTCheckBox.Position = [337 101 87 22];

% Create FilterTab
app.FilterTab = uitab(app.TabGroup);
app.FilterTab.Title = 'Filter';

% Create UIAxes_3
app.UIAxes_3 = uiaxes(app.FilterTab);
title(app.UIAxes_3, 'Filter')
xlabel(app.UIAxes_3, 'frequency (Hz)')
ylabel(app.UIAxes_3, 'Amplitude (normalized)')
app.UIAxes_3.Position = [1 142 513 316];

% Create ModelDropDownLabel
app.ModelDropDownLabel = uilabel(app.FilterTab);
app.ModelDropDownLabel.Position = [44 112 38 22];
app.ModelDropDownLabel.Text = 'Model';
```

```
% Create ModelDropDown
app.ModelDropDown = uidropdown(app.FilterTab);
app.ModelDropDown.Items = {'Brick', 'Butterworth', 'Chebyshev'};
app.ModelDropDown.Position = [91 112 143 22];
app.ModelDropDown.Value = 'Brick';

% Create ShowFFTCheckBox
app.ShowFFTCheckBox = uicheckbox(app.FilterTab);
app.ShowFFTCheckBox.ValueChangedFcn = createCallbackFcn(app, @ShowFFTCheckBoxValueChanged, true);
app.ShowFFTCheckBox.Text = 'Show FFT';
app.ShowFFTCheckBox.Position = [322 112 77 22];

% Create CutoffFreqSpinnerLabel
app.CutoffFreqSpinnerLabel = uilabel(app.FilterTab);
app.CutoffFreqSpinnerLabel.Position = [44 75 69 22];
app.CutoffFreqSpinnerLabel.Text = 'Cutoff Freq.';

% Create CutoffFreqSpinner
app.CutoffFreqSpinner = uispinner(app.FilterTab);
app.CutoffFreqSpinner.Limits = [1 Inf];
app.CutoffFreqSpinner.Position = [122 75 84 22];
app.CutoffFreqSpinner.Value = 1;

% Create TypeDropDown_2Label
app.TypeDropDown_2Label = uilabel(app.FilterTab);
app.TypeDropDown_2Label.Position = [44 41 32 22];
app.TypeDropDown_2Label.Text = 'Type';

% Create TypeDropDown_2
app.TypeDropDown_2 = uidropdown(app.FilterTab);
app.TypeDropDown_2.Items = {'Low-pass', 'High-pass'};
app.TypeDropDown_2.ValueChangedFcn = createCallbackFcn(app, @TypeDropDown_2ValueChanged, true);
app.TypeDropDown_2.Position = [91 41 143 22];
app.TypeDropDown_2.Value = 'Low-pass';

% Create OrderSpinnerLabel
app.OrderSpinnerLabel = uilabel(app.FilterTab);
app.OrderSpinnerLabel.Position = [321 75 36 22];
```

```
app.OrderSpinnerLabel.Text = 'Order';

% Create OrderSpinner
app.OrderSpinner = uispinner(app.FilterTab);
app.OrderSpinner.Limits = [1 Inf];
app.OrderSpinner.HorizontalAlignment = 'center';
app.OrderSpinner.Position = [366 75 52 22];
app.OrderSpinner.Value = 1;

% Create SaveTab
app.SaveTab = uitab(app.TabGroup);
app.SaveTab.Title = 'Save';

% Create FilenameEditFieldLabel
app.FilenameEditFieldLabel = uilabel(app.SaveTab);
app.FilenameEditFieldLabel.HorizontalAlignment = 'right';
app.FilenameEditFieldLabel.Position = [55 362 55 22];
app.FilenameEditFieldLabel.Text = 'Filename';

% Create FilenameEditField
app.FilenameEditField = uieditfield(app.SaveTab, 'text');
app.FilenameEditField.Position = [125 362 276 22];

% Create SaveButton_2
app.SaveButton_2 = uibutton(app.SaveTab, 'push');
app.SaveButton_2.ButtonPushedFcn = createCallbackFcn(app, @SaveButton_2Pushed, true);
app.SaveButton_2.FontWeight = 'bold';
app.SaveButton_2.Position = [71 299 100 22];
app.SaveButton_2.Text = 'Save';

% Create AboutTab
app.AboutTab = uitab(app.TabGroup);
app.AboutTab.Title = 'About';

% Create Image
app.Image = uiimage(app.AboutTab);
app.Image.Position = [285 185 217 251];
app.Image.ImageSource = 'Photo 25-12-18 13 03 50.jpg';
```

```
% Create label_name
app.label_name = uilabel(app.AboutTab);
app.label_name.FontSize = 16;
app.label_name.FontWeight = 'bold';
app.label_name.Position = [36 414 215 22];
app.label_name.Text = 'Mateus Valverde Gasparino';

% Create label_project
app.label_project = uilabel(app.AboutTab);
app.label_project.Position = [36 186 195 63];
app.label_project.Text = {'This app was developed as the '; 'final project for the course ABE 425';

% Create label_bug
app.label_bug = uilabel(app.AboutTab);
app.label_bug.Position = [36 137 230 28];
app.label_bug.Text = {'If you find any bug or malfunction, please,'; 'report at:'};

% Create label_resume
app.label_resume = uilabel(app.AboutTab);
app.label_resume.Position = [36 283 232 112];
app.label_resume.Text = {'Bachelor in Mechatronics Engineering'; 'at University of Sao Paulo, Brazil

% Create label_git
app.label_git = uilabel(app.AboutTab);
app.label_git.FontWeight = 'bold';
app.label_git.Position = [36 98 276 22];
app.label_git.Text = ' https://github.com/matval/digital_oscilloscope';

% Create OFFLampLabel
app.OFFLampLabel = uilabel(app.UIFigure);
app.OFFLampLabel.HorizontalAlignment = 'center';
app.OFFLampLabel.Position = [612 130 29 22];
app.OFFLampLabel.Text = 'OFF';

% Create OFFLamp
app.OFFLamp = uilamp(app.UIFigure);
app.OFFLamp.Position = [607 85 37 37];
app.OFFLamp.Color = [1 0 0];
```

```
% Create StatusEditFieldLabel
app.StatusEditFieldLabel = uilabel(app.UIFigure);
app.StatusEditFieldLabel.HorizontalAlignment = 'center';
app.StatusEditFieldLabel.Position = [16 1 40 22];
app.StatusEditFieldLabel.Text = 'Status';

% Create StatusEditField
app.StatusEditField = uieditfield(app.UIFigure, 'text');
app.StatusEditField.HorizontalAlignment = 'center';
app.StatusEditField.Position = [71 1 587 22];

% Create MainButton
app.MainButton = uibutton(app.UIFigure, 'state');
app.MainButton.ValueChangedFcn = createCallbackFcn(app, @MainButtonValueChanged, true);
app.MainButton.Text = 'Main';
app.MainButton.Position = [575 455 100 22];
app.MainButton.Value = true;

% Create SignalButton
app.SignalButton = uibutton(app.UIFigure, 'state');
app.SignalButton.ValueChangedFcn = createCallbackFcn(app, @SignalButtonValueChanged, true);
app.SignalButton.Text = 'Signal';
app.SignalButton.Position = [575 403 100 22];

% Create FFTButton
app.FFTButton = uibutton(app.UIFigure, 'state');
app.FFTButton.ValueChangedFcn = createCallbackFcn(app, @FFTButtonValueChanged, true);
app.FFTButton.Text = 'FFT';
app.FFTButton.Position = [575 353 100 22];

% Create FilterButton
app.FilterButton = uibutton(app.UIFigure, 'state');
app.FilterButton.ValueChangedFcn = createCallbackFcn(app, @FilterButtonValueChanged, true);
app.FilterButton.Text = 'Filter';
app.FilterButton.Position = [575 303 100 22];

% Create SaveButton
app.SaveButton = uibutton(app.UIFigure, 'state');
app.SaveButton.ValueChangedFcn = createCallbackFcn(app, @SaveButtonValueChanged, true);
```

```
app.SaveButton.Text = 'Save';
app.SaveButton.Position = [575 253 100 22];

% Create AboutButton
app.AboutButton = uibutton(app.UIFigure, 'state');
app.AboutButton.ValueChangedFcn = createCallbackFcn(app, @AboutButtonValueChanged, true);
app.AboutButton.Text = 'About';
app.AboutButton.Position = [575 205 100 22];

% Show the figure after all components are created
app.UIFigure.Visible = 'on';
end
end

% App creation and deletion
methods (Access = public)

% Construct app
function app = oscilloscope_project

% Create UIFigure and components
createComponents(app)

% Register the app with App Designer
registerApp(app, app.UIFigure)

% Execute the startup function
runStartupFcn(app, @startupFcn)

if nargout == 0
clear app
end
end

% Code that executes before app deletion
function delete(app)

% Delete UIFigure when app is deleted
delete(app.UIFigure)
```

```
end
end
end
```