

Lab8 Wireless Links Documentation
2022

Alexandros Charisis

alexcharisis@gmail.com

Katsaros Ioannis Dimitrios

katsaros97g@gmail.com

TABLE CONTENTS

- [Getting Started](#)
- [Old Version](#)
- [New Version](#)
- [GUI](#)
- [Back-End Changes](#)

Getting Started

Lab 8 is a simulation of an indoor wireless network with one or several transmitters. Field, coverage, modulation and efficiency measurements are taken at specific points, across user defined lines, or on the whole floor. These measurements are based on the recreation of realistic phenomena. Its purpose is to provide information through various experiments with the intention of educating the user about wireless networks.

Old Version

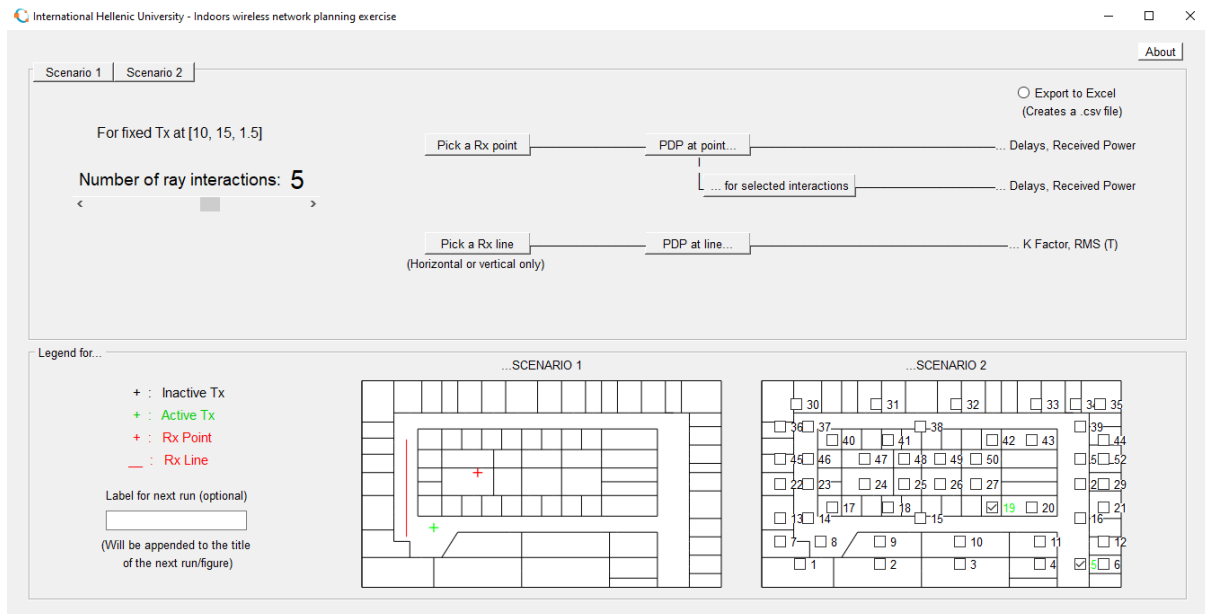
The previous version needs to be run on an old version of Matlab in a Windows 7 virtual environment. It is accessible only by downloading a very large file containing the virtual machine and all of the experiments.

New Version

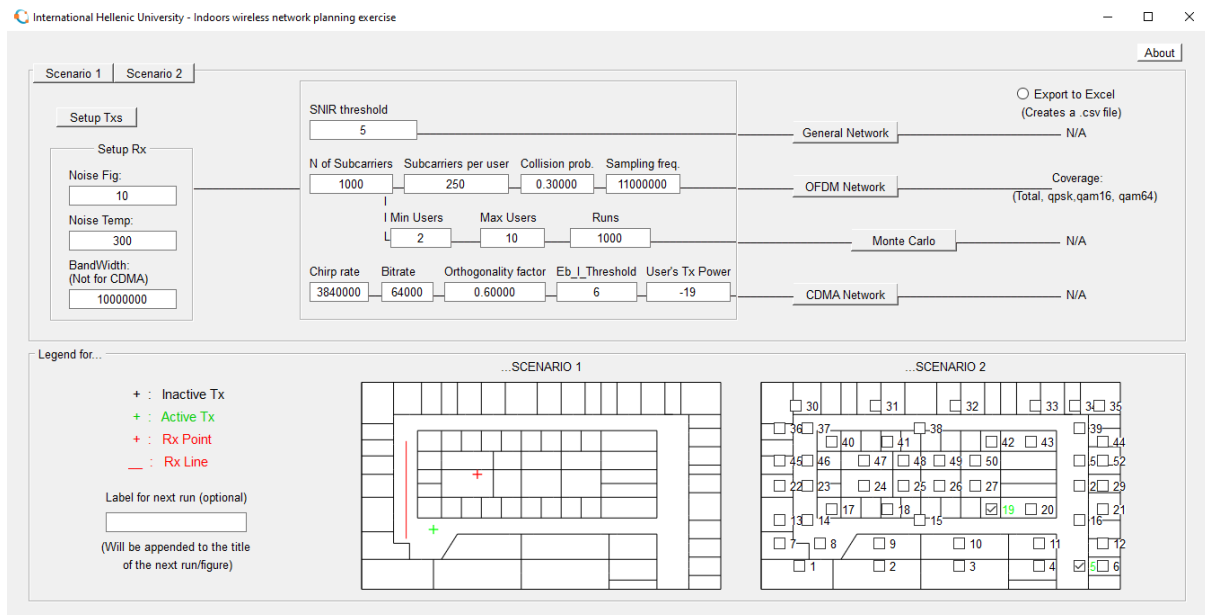
This version runs on Octave and respectively on the latest operating systems. It no longer requires a virtual machine reducing the computer's workload. It runs on the latest Octave version, no longer depending on an old version of Matlab. Since Octave is open source and free, it is easier for someone to access, maintain and add features to it.

GUI

One thing that differentiates the new version from the previous one is the revamped GUI, both in appearance (as seen below) and implementation.



Scenario 1



Scenario 2

The Octave gui initializes when the *show_mainDialog()* is called from the *mainDialog.m* file.

```
##### Main function, creating the whole GUI
function ret = show_mainDialog()
    wait_bar = waitbar(0, 'Generating GUI.');
```



```
    _scrSize = get(0, "screensize");
    _xPos = (_scrSize(3) - 1340)/2;
    _yPos = (_scrSize(4) - 658)/2;
```

The wait bar initializes at 0% when the gui is created through the *waitbar()* function. The screen's size, position on x and y are initialized right after.

```
% Create a struct with all the necessary objects.
mainDialog = struct( ...
    'figure', mainDialog, ...
    'about_button', about_button, ...
    'scenario_1', scenario_1, ...
    'interactions_slider', interactions_slider, ...
    'interactions_label', interactions_label, ...
    'rx_point_button', rx_point_button, ...
    'rx_line_button', rx_line_button, ...
    'pdp_point_button', pdp_point_button, ...
    'for_selected_interactions_button', for_selected_interactions_button, ...
    'pdp_line_button', pdp_line_button, ...
    'excel_1', excel_1, ...
    'scenario_2', scenario_2, ...
    'setup_txs', setup_txs, ...
    'setup_rx', setup_rx, ...
    'noise_fig_edit', noise_fig_edit, ...
    'noise_temp_edit', noise_temp_edit, ...
    'bandwidth_edit', bandwidth_edit, ...
    'GroupPanel_5', GroupPanel_5, ...
    'snir_edit', snir_edit, ...
    'n_of_subcarriers_edit', n_of_subcarriers_edit, ...
    'subcarriers_per_user_edit', subcarriers_per_user_edit, ...
    'collisions_prob_edit', collisions_prob_edit, ...
    'sampling_freq_edit', sampling_freq_edit, ...
    'min_users_edit', min_users_edit, ...
    'max_users_edit', max_users_edit, ...
    'runs_edit', runs_edit, ...
    'chirp_rate_edit', chirp_rate_edit, ...
    'bitrate_edit', bitrate_edit, ...
    'orth_factor_edit', orth_factor_edit, ...
    'eb_i_threshold_edit', eb_i_threshold_edit, ...
    'users_tx_power_edit', users_tx_power_edit, ...
    'general_network_button', general_network_button, ...
    'ofdm_network_button', ofdm_network_button, ...
    'monte_carlo_button', monte_carlo_button, ...
    'cdma_network_button', cdma_network_button, ...
    'excel_2', excel_2, ...
    'scenario_1_button', scenario_1_button, ...
    'scenario_2_button', scenario_2_button, ...
    'legends', legends, ...
    'axis1', axis1, ...
    'axis2', axis2, ...
    'next_run_edit', next_run_edit);
```

A struct called *mainDialog* is created that stores all the values from the gui that are necessary to run the simulation. The value names match the gui names for easier use. There is an exception with the axis1 and axis2 values which refer to the (image) where we set the Rx points and lines for axis1 and to the (image) with the checkboxes for axis2.

```
% Assign callbacks.
set (about_button, 'callback', @about_Callback);

set (scenario_1_button, 'callback', @scenario_1_button_Callback);
set (interactions_slider, 'callback', @interactions_slider_Callback);
set (rx_point_button, 'callback', @rx_point_button_Callback);
set (rx_line_button, 'callback', @rx_line_button_Callback);
set (pdp_point_button, 'callback', @pdp_point_button_Callback);
set (for_selected_interactions_button, 'callback', @for_selected_interactions_button_Callback);
set (pdp_line_button, 'callback', @pdp_line_button_Callback);

set (scenario_2_button, 'callback', @scenario_2_button_Callback);
set (setup_txs, 'callback', @setup_txs_Callback);
set (noise_fig_edit, 'callback', @noise_fig_edit_Callback);
set (noise_temp_edit, 'callback', @noise_temp_edit_Callback);
set (bandwidth_edit, 'callback', @bandwidth_edit_Callback);
set (snir_edit, 'callback', @snir_edit_Callback);
set (n_of_subcarriers_edit, 'callback', @n_of_subcarriers_edit_Callback);
set (subcarriers_per_user_edit, 'callback', @subcarriers_per_user_edit_Callback);
set (collisions_prob_edit, 'callback', @collisions_prob_edit_Callback);
set (sampling_freq_edit, 'callback', @sampling_freq_edit_Callback);
set (min_users_edit, 'callback', @min_users_edit_Callback);
set (max_users_edit, 'callback', @max_users_edit_Callback);
set (runs_edit, 'callback', @runs_edit_Callback);
set (chirp_rate_edit, 'callback', @chirp_rate_edit_Callback);
set (bitrate_edit, 'callback', @bitrate_edit_Callback);
set (orth_factor_edit, 'callback', @orth_factor_edit_Callback);
set (eb_i_threshold_edit, 'callback', @eb_i_threshold_edit_Callback);
set (users_tx_power_edit, 'callback', @users_tx_power_edit_Callback);
set (general_network_button, 'callback', @general_network_button_Callback);
set (ofdm_network_button, 'callback', @ofdm_network_button_Callback);
set (monte_carlo_button, 'callback', @monte_carlo_button_Callback);
set (cdma_network_button, 'callback', @cdma_network_button_Callback);

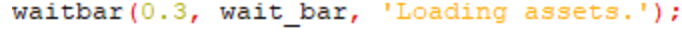
set (next_run_edit, 'callback', @next_run_edit_Callback);
```

A callback function is assigned to every button and text input of the gui. The names also match for easier use.

```

dlg = struct(mainDialog);

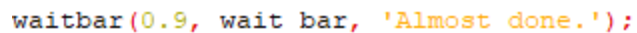
% Load assets.
waitbar(0.3, wait_bar, 'Loading assets.');
```



```

mainDialog.handles = init(wait_bar);

% Apply last changes on the GUI.
waitbar(0.9, wait_bar, 'Almost done.');
```



```

mainDialog = drawAxes(mainDialog);
mainDialog = createCheckboxes(mainDialog);

% Save into the main figure.
guidata(mainDialog.figure, mainDialog);

%
% The source code writed here will be executed when
% windows load. Work like 'onLoad' event of other languages.
%
close(wait_bar);
set(mainDialog.figure, 'visible', 'on');
ret = mainDialog;
end
```

The *mainDialog* struct is initialized and the wait bar progresses to 30%. The handles struct is initialized by the *init()* function in the *init.m* file. It also progresses the waitbar while initializing the values. After initialization is complete the waitbar progresses to 90%. The two images are drawn in the gui and the checkboxes are created. The data from the *mainDialog* struct is stored in the figure of the ui. Loading is complete, the wait bar closes and the *mainDialog* window is set to visible

```

% Enables all UI elements based on the choice type.
% Choice must be either 'off' or 'on'.
function enableUI(mainDialog, choice)
    % Global buttons
    set(mainDialog.about_button, 'enable', choice);
    set(mainDialog.scenario_1_button, 'enable', choice);
    set(mainDialog.scenario_2_button, 'enable', choice);
    % Scenario 1
    set(mainDialog.interactions_slider, 'enable', choice);
    set(mainDialog.rx_point_button, 'enable', choice);
    set(mainDialog.rx_line_button, 'enable', choice);
    set(mainDialog.pdp_point_button, 'enable', choice);
    set(mainDialog.for_selected_interactions_button, 'enable', choice);
    set(mainDialog.pdp_line_button, 'enable', choice);
    set(mainDialog.excel_1, 'enable', choice);
    % Scenario 2
    set(mainDialog.setup_txs, 'enable', choice);
    set(mainDialog.noise_fig_edit, 'enable', choice);
    set(mainDialog.noise_temp_edit, 'enable', choice);
    set(mainDialog.bandwidth_edit, 'enable', choice);
    set(mainDialog.snir_edit, 'enable', choice);
    set(mainDialog.n_of_subcarriers_edit, 'enable', choice);
    set(mainDialog.subcarriers_per_user_edit, 'enable', choice);
    set(mainDialog.collisions_prob_edit, 'enable', choice);
    set(mainDialog.sampling_freq_edit, 'enable', choice);
    set(mainDialog.min_users_edit, 'enable', choice);
    set(mainDialog.max_users_edit, 'enable', choice);
    set(mainDialog.runs_edit, 'enable', choice);
    set(mainDialog.chirp_rate_edit, 'enable', choice);
    set(mainDialog.bitrate_edit, 'enable', choice);
    set(mainDialog.orth_factor_edit, 'enable', choice);
    set(mainDialog.eb_i_threshold_edit, 'enable', choice);
    set(mainDialog.users_tx_power_edit, 'enable', choice);
    set(mainDialog.general_network_button, 'enable', choice);
    set(mainDialog.ofdm_network_button, 'enable', choice);
    set(mainDialog.monte_carlo_button, 'enable', choice);
    set(mainDialog.cdma_network_button, 'enable', choice);
    set(mainDialog.excel_2, 'enable', choice);
    set(mainDialog.next_run_edit, 'enable', choice);
end

```

The *enableUI()* function enables or disables all the gui elements based on the value of *choice*, on for enabled and off for disabled. This is used to prevent the ui values from changing while the simulation is running, a way to prevent possible errors.


```

% Changes the handles' interactions and updates the gui label.
function interactions_slider_Callback(src, data)
    mainDialog = guidata(gcf);

    number = get(src, "value");
    number = round(number);

    mainDialog.handles.interactions = number;
    set(mainDialog.interactions_label, "String", num2str(number));

    guidata(mainDialog.figure, mainDialog);
end

```

The *interactions_slider_Callback()* function gets the numeric value of the slider. It rounds the value to the closest integer, to use it as the number of ray interactions and saves it in the handles struct. Finally it changes the *interactions_label* to the new number and updates the ui with the new value through the *guidata()* function.

```

% Based on a mouse input, deletes the previous cross point,
% paints a new one on the new coordinate and updates the handles' Rx point.
function rx_point_button_Callback(src, data)
    mainDialog = guidata(gcf);

    %% Get input
    [x, y, buttons] = ginput(1);

    if ((x < 0) || (x > 45) || ...
        (y < 0) || (y > 45))
        return;
    endif

    Rx = [x y 1.500000];

    % delete previous
    delete(mainDialog.Rx_x);
    delete(mainDialog.Rx_y);
    % assign new value
    mainDialog.handles.rx_single = Rx;
    % draw Rx (red) cross
    mainDialog.Rx_x = line(mainDialog.axis1, [Rx(1)-0.6 Rx(1)+0.6], ...
                           [Rx(2) Rx(2)], 'color', [1 0 0], 'linewidth', 1);
    mainDialog.Rx_y = line(mainDialog.axis1, [Rx(1) Rx(1)], ...
                           [Rx(2)-1 Rx(2)+1], 'color', [1 0 0], 'linewidth', 1);

    guidata(mainDialog.figure, mainDialog);
end

```

The `rx_point_button_Callback()` function “moves” the rx point on the image. It deletes the old one and creates a new one. First the mouse click position is saved as a three value vector of its x position, y position and buttons which refers to the mouse button pressed, then it is checked. If it's out of bounds the function exits and nothing changes. If it's in bounds it creates an Rx vector with 3 values x, y and 1.5. The first two are used to position the point on the image. The 1.5 is the z axis value and it's set to 1.5 so the point is visible in front of the image. The previous point is deleted, the handles struct is updated with the new values of Rx and two lines are drawn on the image to create a cross.

```
% Based on 2 mouse inputs, deletes the previous line,
% paints a new one on the new coordinates and updates the handles' line.
function rx_line_button_Callback(src, data)
    mainDialog = guidata(gcf);

    %% Get input
    [x1, y1, buttons] = ginput(1);
    [x2, y2, buttons] = ginput(1);

    if ((x1 < 0) || (x1 > 45) || ...
        (y1 < 0) || (y1 > 45) || ...
        (x2 < 0) || (x2 > 45) || ...
        (y2 < 0) || (y2 > 45))
        return;
    endif

    % Horizontal or Vertical
    if ((x1-x2)^2 > (y1 - y2)^2)
        y2 = y1;
    else
        x2 = x1;
    endif

    Line = [x1 x2 y1 y2];

    % delete previous
    delete(mainDialog.line);
    % assign new value
    mainDialog.handles.line = Line;
    % draw Rx (red) line
    mainDialog.line = line(mainDialog.axis1, [Line(1:2)], ...
                           [Line(3:4)], 'color', [1 0 0]);

    guidata(mainDialog.figure, mainDialog);
end
```

The *rx_line_button_Callback()* moves the rx line on the image. It deletes the old one and creates a new one. First the two mouse click positions are saved as a three value vector each. The vector consists of the x position, y position and buttons which refers to the mouse button pressed. The vectors are checked. If they are out of bounds the function exits and nothing changes. If they are in bounds it compares the distance of the two points on each axis. The line is drawn on the longer one, by setting the values of the other axis to be the same. A Line vector is created with the x and y positions of the two points. The old line is deleted and the handles struct gets updated with the new values. The line is drawn and the gui gets updated through the *guidata()* function.

```
% Calls the corresponding function. If the Rx point or line is beyond coverage,
% an error log is displayed instead. If chosen, the function's returning values
% are written into a csv file.
function pdp_point_button_Callback(src, data)
    mainDialog = guidata(gcf);
    handles = mainDialog.handles;

    enableUI(mainDialog, 'off');
    try

        [k_Rice, RMS_T, TT_N, PP_N] = Plot_Rays_PDP_Single_Point(handles);

        if (get(mainDialog.excel_1, 'Value') == 1)
            data(:,1) = TT_N;
            data(:,2) = PP_N;
            csvwrite('PDP_single_point.csv', data);
        endif
    catch Exception
        errordlg({'The Rx point or line you selected is beyond coverage!', ...
            'No ray leads to it. Please select another one.'}, 'Oops!', 'modal')
    end

    enableUI(mainDialog, 'on');
end
```

The *pdp_at_point_button_Callback()* function disables the gui, calls the *Plot_Rays_PDP_Single_Point()* function and then re-enables the gui. If the export to excel button is checked it exports the results to a csv file. If the selected rx point is out of coverage it informs the user with a pop up dialog.

```
% Calls the corresponding function. If the Rx point or line is beyond coverage,
% an error log is displayed instead. If chosen, the function's returning values
% are written into a csv file.
```

```
function for_selected_interactions_button_Callback(src, data)
    mainDialog = guidata(gcf);
    handles = mainDialog.handles;

    enableUI(mainDialog, 'off');
    try
        [k_Rice, RMS_T, TT_N, PP_N] = Plot_Rays_PDP_Spec_Generation(handles);

        if (get(mainDialog.excel_1, 'Value') == 1)
            data(:,1) = TT_N;
            data(:,2) = PP_N;
            csvwrite('PDP_single_point_one_interaction.csv', data);
        endif
    catch MException
        errorDlg({'The Rx point or line you selected is beyond coverage!', ...
            'No ray leads to it. Please select another one.'}, 'Oops!', 'modal')
    end

    enableUI(mainDialog, 'on');
end
```

The *for_selected_interactions_button_Callback()* function disables the gui, calls the *Plot_Rays_PDP_Spec_Generation()* function and then re-enables the gui. If the export to excel button is checked it exports the results to a csv file. If the selected rx point is out of coverage it informs the user with a pop up dialog.

```
% Calls the corresponding function. If the Rx point or line is beyond coverage,
% an error log is displayed instead. If chosen, the function's returning values
% are written into a csv file.
```

```
function pdp_line_button_Callback(src, data)
    mainDialog = guidata(gcf);
    handles = mainDialog.handles;

    enableUI(mainDialog, 'off');
    try
        [k_Rice, RMS_T] = Plot_Rays_PDP_Along_Lines(handles);

        if (get(mainDialog.excel_1, 'Value') == 1)
            data(:,1) = k_Rice;
            data(:,2) = RMS_T;
            csvwrite('PDP_line.csv', data);
        endif
    catch MException
        errorDlg({'The Rx point or line you selected is beyond coverage!', ...
            'No ray leads to it. Please select another one.'}, 'Oops!', 'modal')
    end

    enableUI(mainDialog, 'on');
end
```

The *pdp_line_button_Callback()* function disables the gui, calls the *Plot_Rays_PDP_Along_Lines()* function and then re-enables the gui. If the export to excel button is checked it exports the results to a csv file. If the selected rx line is out of coverage it informs the user with a pop up dialog.

```
% Scrollbar for the 'Setup Txs' window's panel.  
% If the panel is smaller than the window, it does nothing.  
function setup_txs_slider_Callback(src, data, panel, original_pos)  
    value = get(src, 'Value');  
    pos = get(panel, 'Position');  
  
    if (pos(4) < 400)  
        return;  
    endif  
  
    set(panel, 'Position', [pos(1), original_pos+value, pos(3), pos(4)]);  
end
```

The *setup_txs_slider_Callback()* function creates a working scrollbar for the Setup Txs window.

```

% Saves the current changes done by the user in the 'Setup TxS' window.
% and closes the window.
% If any field is non-numerical it prompts an error dialog instead.
function save_changes_button_Callback(src, data, mainDialog, edit_fields)
    % check for any invalid values
    for [val, key] = edit_fields
        str = get(val, 'String');
        number = str2num(str);

        if isempty(number)
            set(val, 'string', '0');
            errordlg(['Non-numerical inputs are not allowed!' ...
                    '\nPlease input only numerical data.' ...
                    '\nCurrent changes not saved!']);
            return;
        endif
    endfor

    % save changes for each
    TxIDS = mainDialog.handles.TxIDS;
    for i=1:length(TxIDS)
        index = find(mainDialog.handles.TxIDS == TxIDS(i));

        pow = get(edit_fields.(['pow_edit_' num2str(TxIDS(i))]), 'String');
        gain = get(edit_fields.(['gain_edit_' num2str(TxIDS(i))]), 'String');

        mainDialog.handles.TxPows(index) = str2num(pow);
        mainDialog.handles.TxGains(index) = str2num(gain);
    endfor

    guidata(mainDialog.figure, mainDialog);
    % close current figure
    close();
end

```

The *save_changes_button_Callback()* function saves the changes done in the Setup TxS window and closes it. For every input field it checks if the string given is a numeric value. If a non numeric was given, a pop up dialog informs the user and the function exits without updating the handles struct. It iterates the TxIDS list from the handles struct and updates the values of each entry. Finally it updates the gui with the new values.

```

% Creates the 'Setup Txs' window that contains all the Txs
% currently active by the user. Provides an edit field for each,
% to change Power and Gain.
function setup_txs_Callback(src, data)
    mainDialog = guidata(gcf);

    total_ids = mainDialog.handles.TxIDS;
    total_pows = mainDialog.handles.TxPows;
    total_gains = mainDialog.handles.TxGains;

    % 5 margin, 20 for titles, 40 for each ID,
    % 40 margin, 30 for button + a little more (7)
    y_size = 5+20 + length(total_ids)*40 + 30+40+7;

    _scrSize = get(0, "screensize");
    _xPos = (_scrSize(3) - 270)/2;
    _yPos = (_scrSize(4) - y_size)/2;

```

The *setup_txs_Callback()* function creates the gui window that opens when the Setup Txs button is pressed. It retrieves the TxIDS, TxPows and TxGains from the handles struct and uses them to create the elements of the window. It sets the size on y axis based on how many TxIDS it gets from the handles. The window size and its position on screen are initialized. For each id it creates a row with its id, power input field and gain input field.

```

% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function noise_fig_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src, 'String');
    number = str2num(str);

    if isempty(number)
        set(src, 'string', '0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.NoiseFig = number;
    guidata(mainDialog.figure, mainDialog);
end

```

The *noise_fig_edit_Callback()* function saves the input from the Noise Fig input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window

and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function noise_temp_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.NoiseTemp = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *noise_temp_edit_Callback()* function saves the input from the Noise Temp input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function bandwidth_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.BW = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *bandwidth_edit_Callback()* function saves the input from the BandWidth input field to the handles struct. It checks if the given string can be converted to a

number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function snir_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.SINRthresh = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *snir_edit_Callback()* function saves the input from the SNIR Threshold input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function n_of_subcarriers_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.NofSubcTotal = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *n_of_subcarriers_edit_Callback()* function saves the input from the N of Subcarriers input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function subcarriers_per_user_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.NofSubcPerUser = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *subcarriers_per_user_edit_Callback()* function saves the input from the Subcarriers per user input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function collisions_prob_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.Probability = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *collisions_prob_edit_Callback()* function saves the input from the Collision prob input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function sampling_freq_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.F_sampl = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *sampling_freq_edit_Callback()* function saves the input from the Sampling freq input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function min_users_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.minUsers = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *min_users_edit_Callback()* function saves the input from the Min Users input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function max_users_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.maxUsers = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *max_users_edit_Callback()* function saves the input from the Max Users input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function runs_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.Runs = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *runs_edit_Callback()* function saves the input from the Runs input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function chirp_rate_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.chirp = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *chirp_rate_edit_Callback()* function saves the input from the Chirp rate input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function bitrate_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.bitrate = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *bitrate_edit_Callback()* function saves the input from the Bitrate input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function orth_factor_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.Orthogonality_Factor = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *orth_factor_edit_Callback()* function saves the input from the Orthogonality factor input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function eb_i_threshold_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.Eb_I_Threshold = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *eb_i_threshold_edit_Callback()* function saves the input from the Eb_I_Threshold input field to the handles struct. It checks if the given string can be

converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Updates the corresponding handles' field with the new value.
% If the value is non-numerical, an error dialog is prompted instead.
function users_tx_power_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    str=get(src,'String');
    number = str2num(str);

    if isempty(number)
        set(src,'string','0');
        errordlg('Input must be numerical');
        return;
    end

    mainDialog.handles.Tx_Power_User = number;
    guidata(mainDialog.figure, mainDialog);
end
```

The *users_tx_power_edit_Callback()* function saves the input from the User's Tx Power input field to the handles struct. It checks if the given string can be converted to a number. If a non numerical was given, it informs the user with a pop up window and exits without saving the changes to the handles, else it saves and updates the gui figure.

```
% Calls the corresponding function.
function general_network_button_Callback(src, data)
    mainDialog = guidata(gcf);
    handles = mainDialog.handles;

    enableUI(mainDialog, 'off');

    General_Parameters_Net(handles);

    enableUI(mainDialog, 'on');
end
```

The *general_network_button_Callback()* function disables the gui, calls the *General_Parameters_Net()* function and then re-enables the gui.

```

% Calls the corresponding function. If chosen, the function's
% returning values are written into a csv file.
function ofdm_network_button_Callback(src, data)
    mainDialog = guidata(gcf);
    handles = mainDialog.handles;

    enableUI(mainDialog, 'off');

    [percent_cover_OFDM, percent_qpsk, percent_qam16, percent_qam64] ...
        = OFDM_Network(handles);

    if (get(mainDialog.excel_2, 'Value') == 1)
        data=[percent_cover_OFDM, percent_qpsk, percent_qam16, percent_qam64]';
        csvwrite('OFDM_coverage.csv', data);
    endif

    enableUI(mainDialog, 'on');
end

```

The *ofdm_network_button_Callback()* function disables the gui, calls the *OFDM_Network()* function and then re-enables the gui. If the Export to Excel box is checked, the results are saved to a csv file.

```

% Calls the corresponding function.
function monte_carlo_button_Callback(src, data)
    mainDialog = guidata(gcf);
    handles = mainDialog.handles;

    enableUI(mainDialog, 'off');

    MonteCarlo_OFDM(handles);

    enableUI(mainDialog, 'on');
end

```

The *monte_carlo_button_Callback()* function disables the gui, calls the *MonteCarlo_OFDM()* function and then re-enables the gui.


```

% Calls the corresponding function. If the Tx_Power_User is greater than
% the max Tx Power in the handles, an error dialog is displayed instead.
function cdma_network_button_Callback(src, data)
    mainDialog = guidata(gcf);
    handles = mainDialog.handles;
    Tx_Power_User = handles.Tx_Power_User;
    TxPows = handles.TxPows;

    if (Tx_Power_User > max(TxPows))
        errorDlg(["User's Tx Power (CDMA only) must be less than" ...
            " the maximum Tx power for any transmitter"], ...
            'Input Error','modal');
        return;
    end

    enableUI(mainDialog, 'off');
    [percent_cov_CDMA] = CDMA_Network(handles);
    enableUI(mainDialog, 'on');
end

```

The *cdma_network_button_Callback()* function disables the gui, calls the *CDMA_Network()* function and then re-enables the gui. If the User's Tx Power is greater than the maximum Tx power in handles, a pop up dialog informs the user and the function exits without running the simulation.

```

% Swaps between Scenarios. Sets this panel to visible,
% and the other panels not visible.
function scenario_1_button_Callback(src, data)
    mainDialog = guidata(gcf);

    set(mainDialog.scenario_1, "visible", "on");
    set(mainDialog.scenario_2, "visible", "off");

    guidata(mainDialog.figure, mainDialog);
end

% Swaps between Scenarios. Sets this panel to visible,
% and the other panels not visible.
function scenario_2_button_Callback(src, data)
    mainDialog = guidata(gcf);

    set(mainDialog.scenario_1, "visible", "off");
    set(mainDialog.scenario_2, "visible", "on");

    guidata(mainDialog.figure, mainDialog);
end

```

The *scenario_1_button_Callback()* and *scenario_1_button_Callback()* functions alternate between the two scenario panels in the main window. They set the one to visible and the other to non-visible.

```
% Updates the handles' figures_title. The title that the next
% figures that will be plotted will have.
function next_run_edit_Callback(src, data)
    mainDialog = guidata(gcf);

    mainDialog.handles.figures_title = get(src, 'String');

    guidata(mainDialog.figure, mainDialog);
end
```

The *next_run_edit_Callback()* function saves the string given to Label for next run edit box to the handles struct and updates the gui.

```
% Draws the lines that make up the 'floor' on both axes.
% Also draws the initial Tx, Rx points and line on the first axis.
function mainDialog = drawAxes(mainDialog)
    % get line positions
    facet = ProloadfacetCCSR;

    % plot them on both axes
    for i=1:size(facet,2);
        pos = facet(i).position;
        line(mainDialog.axis1, pos(1:2, 1), pos(1:2, 2), 'color', [0 0 0]);
        line(mainDialog.axis2, pos(1:2, 1), pos(1:2, 2), 'color', [0 0 0]);
    end

    Tx = mainDialog.handles.tx_single;
    % draw Tx (green) cross
    mainDialog.Tx_x = line(mainDialog.axis1, [Tx(1)-0.6 Tx(1)+0.6], ...
        [Tx(2) Tx(2)], 'color', [0 1 0], 'linewidth', 1);
    mainDialog.Tx_y = line(mainDialog.axis1, [Tx(1) Tx(1)], [Tx(2)-1 Tx(2)+1], ...
        'color', [0 1 0], 'linewidth', 1);

    Rx = mainDialog.handles.rx_single;
    % draw Rx (red) cross
    mainDialog.Rx_x = line(mainDialog.axis1, [Rx(1)-0.6 Rx(1)+0.6], ...
        [Rx(2) Rx(2)], 'color', [1 0 0], 'linewidth', 1);
    mainDialog.Rx_y = line(mainDialog.axis1, [Rx(1) Rx(1)], ...
        [Rx(2)-1 Rx(2)+1], 'color', [1 0 0], 'linewidth', 1);

    Line = mainDialog.handles.line;
    % draw (red) line
    mainDialog.line = line(mainDialog.axis1, [Line(1:2)], ...
        [Line(3:4)], 'color', [1 0 0]);
end
```

The *drawAxes()* function loads the lines' coordinates from the *ProloadfacetCCSR.m* file and draws them twice, once for axis1 and once for axis2 image. It draws the green Tx cross, the red Rx cross and the red Rx line on axis1. The two crosses are drawn by 2 lines each, with 90 degrees angle.

```
% Changes the color of the label related to the checkbox pressed.
% The text color becomes green if activated, else black.
% Also updates the handles' TxIDS, TxPows and TxGains with
% the default values if activated or deletes them otherwise.
function checkbox_Callback(src, data)
    mainDialog = guidata(gcf);

    value = get(src, 'Value');
    number = get(src, 'String');

    if (value == 1)
        % change label color
        set(mainDialog.(['checkbox_label_' number]), 'ForegroundColor', [0 1 0]);

        % add values
        mainDialog.handles.TxIDS(end+1) = str2num(number);
        mainDialog.handles.TxPows(end+1) = -10;
        mainDialog.handles.TxGains(end+1) = 2;
    else
        % change label color
        set(mainDialog.(['checkbox_label_' number]), 'ForegroundColor', [0 0 0]);

        % delete stored values
        index = find(mainDialog.handles.TxIDS == str2num(number));
        mainDialog.handles.TxIDS(index) = [];
        mainDialog.handles.TxPows(index) = [];
        mainDialog.handles.TxGains(index) = [];
    endif

    guidata(mainDialog.figure, mainDialog);
end
```

The *checkbox_Callback()* function changes the color of the checked checkboxes' labels to green and unchecked to black. It updates the handles struct with the new TxIDS, TxPows and TxGains when a box is checked, or deletes them when unchecked.

```

% Populates the second axis by creating checkboxes
% on each position of the handles' TxPos.
% Also activates the initial Tx IDS.
function mainDialog = createCheckboxes(mainDialog)
    pos = get(mainDialog.axis2, 'Position');

    Tx_Pos = mainDialog.handles.Tx_Pos;

    % The weird 'Positions' with certain 'offsets' inside
    % the for loop, are calculated by eye and rigorous testing.
    % Changing the axis size, means that these are obsolete,
    % and need re-calculation.
    for i=1:size(Tx_Pos,1)
        %%% Checkboxes' labels

        % width of label: 7 for 1 to 9
        %                14 for two digit numbers
        width = 7;
        if (i > 9)
            width = 14;
        endif

        offset = 18;
        checkbox_label = uicontrol( ...
            'parent', mainDialog.figure, ...
            'Style', 'text', ...
            'Units', 'pixels', ...
            'BackgroundColor', [1 1 1], ...
            'FontAngle', 'normal', ...
            'FontName', 'Arial', ...
            'FontSize', 10, 'FontUnits', 'points', ...
            'FontWeight', 'normal', ...
            'ForegroundColor', [0 0 0], ...
            'HorizontalAlignment', 'left', ...
            'Position', [pos(1)+(offset*2)+Tx_Pos(i, 2)*400/45, ...
                        pos(2)+(offset+1)+Tx_Pos(i, 3)*230/45, width, 10], ...
            'String', num2str(i), ...
            'TooltipString', '');

        % Standard naming for labels: checkbox_label_N
        mainDialog.(['checkbox_label_' num2str(i)]) = checkbox_label;
    end
end

```

```

    %%% Checkboxes
    checkbox = uicontrol( ...
        'parent', mainDialog.figure, ...
        'Style', 'checkbox', ...
        'Units', 'pixels', ...
        'BackgroundColor', [1 1 1], ...
        'ForegroundColor', [1 1 1], ...
        'FontAngle', 'normal', ...
        'FontName', 'Arial', ...
        'FontSize', 10, 'FontUnits', 'points', ...
        'FontWeight', 'normal', ...
        'ForegroundColor', [0.000 0.000 0.000], ...
        'Position', [pos(1)+(offset)+Tx_Pos(i, 2)*400/45, ...
                     pos(2)+offset+Tx_Pos(i, 3)*230/45, 13, 13], ...
        'String', num2str(i), ...
        'TooltipString', '', ...
        'Min', 0, 'Max', 1, ...
        'Value', 0);

    set(checkbox, 'callback', @checkbox_Callback);

    % Standard naming for checkboxes: checkbox_N
    mainDialog(['checkbox' num2str(i)]) = checkbox;
endfor

% Activate the initial checkboxes
TxIDS = mainDialog.handles.TxIDS;
for i=1:length(TxIDS)
    set(mainDialog(['checkbox' num2str(TxIDS(i))]), 'Value', 1);
    set(mainDialog(['checkbox_label_' num2str(TxIDS(i))]), ...
        'ForegroundColor', [0 1 0]);
endfor
end

```

The *createCheckboxes()* function creates the checkboxes on axis2 based on the coordinates loaded from the handles struct. Then it activates the initial checkboxes given by the handles.

```

% Creates the 'About' window, showcasing info about its contributors.
function about_Callback(src, data)
    % window values
    width = 410;
    height = 210;
    _scrSize = get(0, "screensize");
    _xPos = (_scrSize(3) - width)/2;
    _yPos = (_scrSize(4) - height)/2;

    about_figure = figure (
        'resize', 'on', ...
        'windowstyle', 'normal', ...
        'MenuBar', 'none', ...
        'name', 'About', ...
        'NumberTitle', 'off', ...
        'Position', [_xPos, _yPos, width, height], ...
        'visible', 'off');

    % Side img
    image_axis = axes( ...
        'Units', 'pixels', ...
        'parent', about_figure, ...
        'FontAngle', 'normal', ...
        'FontName', 'Arial', ...
        'FontSize', 10, 'FontUnits', 'points', ...
        'FontWeight', 'normal', ...
        'Position', [0 55 120 120] );

    Img = imread('about_image.png');
    imshow(Img, 'parent', image_axis);

    % Texts
    about_labels_texts = {'Created for the', ...
        'International Hellenic University,', ....
        'Greece, November 2010,', ...
        '', ...
        'by C.Liaskos and G.Koutitas,', ...
        '(cliaskos@csd.auth.gr, g.koutitas@ihu.edu.gr)', ...
        '', ...
        'Updated by A.Charisis and I.Katsaros,', ...
        '(alexcharisis@gmail.com, katsaros97g@gmail.com)'};

```

```

% Height (y) for the first label.
y = 180;
% Make labels for each text
for i=1:length(about_labels_texts)
    uicontrol( ...
        'parent', about_figure, ...
        'Style', 'text', ...
        'Units', 'pixels', ...
        'BackgroundColor', [1 1 1], ...
        'FontAngle', 'normal', ...
        'FontName', 'Arial', ...
        'FontSize', 10, 'FontUnits', 'points', ...
        'FontWeight', 'normal', ...
        'ForegroundColor', [0 0 0], ...
        'HorizontalAlignment', 'left', ...
        'Position', [110 y 300 20], ...
        'String', about_labels_texts(i), ...
        'TooltipString', '');

    y -= 20;
endfor

set(about_figure, 'visible', 'on');
end

```

The *about_Callback()* creates an about window with information about the project, the creators and the contributors.

Back-End Changes

The changes mentioned below are about the functions called by the user in the back end such as *ProPlotReceivedRays* or *Plot_Rays_PDP_Single_Point*, since the GUI and its interactions are updated.

- First and foremost, changes have been made throughout all the relevant functions in order for them to run in Octave instead of the original MATLAB platform. Such changes can be an Octave function that is named differently or needs different parameters but does not change the output and does not affect any result, so they are not worth mentioning in detail.
- Continuing forward it is necessary to address the changes in the *handles (struct)* initialization function. As of this version, it does no longer contain methods to change each field's value as the responsibility is shifted into the GUI's callbacks. It contains all the previous fields with no difference in values and a new *figures_title* field, which in the previous version was passed on as an argument in each function, further complicating things. Lastly, a *wait_bar* (waitbar object) parameter is added (and is passed through the GUI initialization), to better keep track and inform the user about the initialization process.
- The **.mat** files **tempHB,1,2,3,4** storing check boxes, check boxes' texts, TxIDS, TxPow and TxGains respectively are omitted, since they are no longer needed with the new implementation and are updated directly into the *handles (struct)* and passed on the functions through that.

- The following functions' [Plot_Rays_PDP_Single_Point, Plot_Rays_PDP_Spec_Generation, Plot_Rays_PDP_Along_Lines, General_Parameters_Net, OFDM_Network, MonteCarlo_OFDM, CDMA_Network] parameters have been reduced to just the *handles (struct)*, since every argument passed on in the older version was a field of *handles*, thus making it an unnecessary implication. All previous parameters are now defined at the start of each function, maintaining their previous names. An example of that is:

```
function [k_Rice, RMS_T, TT_N, PP_N]=Plot_Rays_PDP_Single_Point(handles)
#####
Number = handles.interactions;
rx1 = handles.rx_single;
label = handles.figures_title;

#####

%inputs default
%Number=5;
%rx=[14.5 25 1.5]
Ray_PDP=handles.Ray_PDP;
ReceiveSphere=handles.ReceiveSphere;
[X,Y,Z]=meshgrid(0:0.25:45,0:0.25:45,1.5:0.25:1.5);
...
```

- All the functions mentioned above produce plots. To eliminate the risk of the plotting calls affecting any other *axis*, each of Octave's plotting call now contains the corresponding *axis* object. An example of this is **view(3)** changed to **view(axis1, 3)**, where axis1 is the wanted axis object.

The functions *ProPlotReceivedRays* and *ProCreateCity* have also received an extra parameter called *currentAxis* for the same purpose as it is used on plot/line/patch calls:

```

figure('Name', label);
axis2 = gca();
view(axis2, 3);
box(axis2, 'off');
surf(axis2, X,Y,Z,ReceivePower);
shading(axis2, 'interp');
hold on;
    ProCreateCity(axis2, 19);
%h=findobj('Type','Surf');
%%hh=findobj('Type','Patch');
%alpha(h,1)
%alpha(hh,0.1)
axis(axis2, "equal", "tight");
hold on;
TX=handles.tx_single;
stem3(axis2, TX(1),TX(2),TX(3)+1,'g');
ProPlotReceivedRays(axis2, II,GG,NN,Ray_PDP);
hold on;
plot3(axis2, ReceiveSphere(rx).position(1,1), ...);
colormap(axis2, "jet");
c = colorbar(axis2, 'ytick', -180:20:0);
set(c, 'position', [0.87, 0.1, 0.05, 0.8]);
title(axis2, 'Received Power');

```

Code snippet showcasing where 'axis' object is used.

- The default color pack of a plot in Octave is different than the original's, resulting in different visuals and difficulty in analyzing one, depending on the plot.
The command `colormap(axis#, "jet");` is used in each plot to face this.
- A particular problem (bug) was presented in Octave whereas using `axis(axis2, "equal", "tight");` resulted in the colorbar of the plot also being altered. To counter act this, the colorbar must be set hard coded in the desired location and with the desired dimensions. Furthermore, for the colorbar to have the same values/ticks as the original, they also have to be set manually, if needed.
- The `make_datatip_show_colormap_value()` function that used to give info of a 3D plot based of the mouse's position is now removed. It is not replaced with an updated one, due to the lack of the `datacursormode()` function in the Octave platform.

- Lastly, both *OFDM_Network* and *MonteCarlo_OFDM* functions had a time-consuming loop. In detail:

```

for i=1:size(Y,2)
    for q=1:size(X,1)
        par=0;
        for ii=1:N;
            par=par+log2(1+(10.^( continuing here....
        end
        gama_nn(q,i)=2.^((1/N).*par)-1;
    end
end

```

Octave (in contrast with Matlab) needs ~3 minutes to complete the computations.

As it is shown, in the innermost loop the *ii* variable is not used and can easily be removed. The loop does the same computation N times, adds up the result, and divides by N later, yielding the value of one of the repeated computations. Also, *gama_nn* matrix is not pre-allocated, creating performance issues.

Thus, the resulting code is:

```

gama_nn = zeros(size(Y,2), size(X,1));
for i=1:size(Y,2)
    for q=1:size(X,1)
        par=log2(1+(10.^( continuing here....
        gama_nn(q,i)=2.^par-1;
    end
end

```

Note that the original code was adding the same value N times and then divided it by N.

By omitting those computations there will also be a smaller rounding error due to float computations.