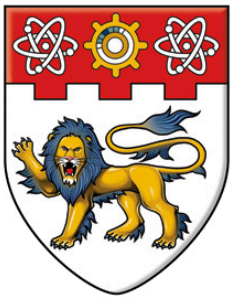


NANYANG TECHNOLOGICAL UNIVERSITY

DEVELOP A FRAMEWORK TO INTEGRATE THE MATLAB GUI APPS

Nonny Florentine

School Of Computer Engineering

2016

NANYANG TECHNOLOGICAL UNIVERSITY

DEVELOP A FRAMEWORK TO INTEGRATE THE MATLAB GUI APPS

Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Computer Engineering
of the Nanyang Technological University

by

Nonny Florentine

School Of Computer Engineering

2016

Abstract

The development of the Graphical User Interfaces (GUIs) has been found to be a essential component in learning of Digital Signal Processing (DSP) topics. In this project, we will be developing a set of DSP software learning tool in the form of GUI by using the MATLAB GUIDE. These software learning tool is called the MATLAB GUI apps. Initially our aim was to develop a framework of the application called the GUIDE Educational Tool (GET) app to integrate these MATLAB GUI apps such that when the user runs the main code, the system will detect all the available MATLAB GUI apps. However this framework was initially acceptable until which the idea of integrating other courses came about. 'What if the user wants to add new MATLAB GUI apps from other courses?'. The answer is we need a one-stop platform that can facilitate the various MATLAB GUI apps from the different courses available. Also to make this set of learning tool more versatile, we want the next user to be able to add new MATLAB GUI apps without ever changing the main code. The structure of the one-stop platform was implemented by using an XML-based architecture. This report will include the implementation of the framework and describing the features of the DSP MATLAB GUI apps that have been developed. At the end of this project, a total of 8 DPS MATLAB GUI apps were developed. Finally, to make sure that the functionality of the system is working we created a new course called the 'Digital Communications'. This course will be the example for a user on how to create new MATLAB GUI apps. Two MATLAB GUI apps were developed under this course.

Acknowledgements

This dissertation would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to Prof.Chng Eng Siong, my final year project supervisor whose sincerity and encouragement I will never forget. Prof.Chng Eng Siong has been my inspiration as I hurdle all the obstacles in the completion this project.

I am deeply indebted to Mr.Vu Tuan Thanh, had kind concern and consideration regarding my final year project progress.

The project will not possible to finish without Mr.Kyaw Zin Tun, had supported me throughout my final year project.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

List of Tables

List of Figures

Table of Contents

1. Introduction

1.1. Background

In the institutes where laboratory facilities are not that much available, and industries are located in remote areas, Personal Computer (PC) can be used to facilitate science and engineering education. Programming and simulation tools can be used widely for preparing such a PC-based setting for students. The teaching methods in educational field especially in the engineering courses have changed over the years from the simple "lecture-only" format to a more integrated "lecture-laboratory" environment. However, for effective teaching in the lecture component must also make extensive use of computer-based explanations, examples, and exercises. There are now some books, courses and even hardware or software learning tool that can bring up to speed on the learning process. PC-based software helps in several areas the learning process.

A modern software tool that is mostly used in engineering field is MATLAB (short for MATrix LABoratory). The use of MATLAB is increasing day by day [1]. Science and engineering students use this software broadly for educational purposes. One of the course's that greatly facilitates one's ability to demonstrate concepts using MATLAB is Digital Signal Processing (DSP).

MATLAB supports developing applications with Graphical User Interface (GUI) features. The GUI tool in MATLAB definitely provide an excellent learning process when used at multiple educational levels. GUI are a common sight in the digital age. They have become ubiquitous to the point of being invisible. They are used every day

without being noticed, on ATM's, websites and PC's. GUI's provide a pictorial faceplate that allows users to call various functions and edit their parameters without ever touching a program's actual code.

Nowadays, there are many educational MATLAB apps that are in the form of GUIs. One of the example is the **Educational Matlab GUIs** that provides a set of MATLAB GUI apps that were developed by the members of the Central Signal and Information Processing, a part of the School of Electrical and Computer Engineering at the Georgia Institute of Technology [1]. The reason that they develop the MATLAB GUI apps is to accompany the texbook – Signal Processing First. DSP covers a huge list of topics and mostly the MATLAB GUI apps are standalone. No doubt that in any institute, it is better to teach students learning DSP using the MATLAB GUI apps. However the MATLAB GUI apps that have been developed mostly were in the form of individual GUIs. Thus, they have no one-stop platform to access all the MATLAB GUI apps. It will be easier for the user to navigate the MATLAB GUI apps without manually looking at the apps individually. What we want to achieve in this project is not just buidling the DSP MATLAB GUI apps to aid the students in learning DSP but also to develop a framework to integrate those MATLAB GUI apps into one set of a learning tool.

1.2. Proposal

In the first contribution, we propose a framework for the multiple DSP MATLAB GUI apps to be integrated such that it is easy for the user to navigate the apps. Figure 1.1 shows one of the DSP MATLAB GUI app that has been developed for this project. This framework was initially acceptable until which the idea of integrating other courses came about. For example, in the Digital Communications course also use MATLAB GUI apps as a software-learning tool. Thus, in the second contribution, we propose an enhanced version of the framework that not only allows us to integrate those MATLAB DSP GUI apps but also able to integrate other MATLAB GUI apps from any courses.

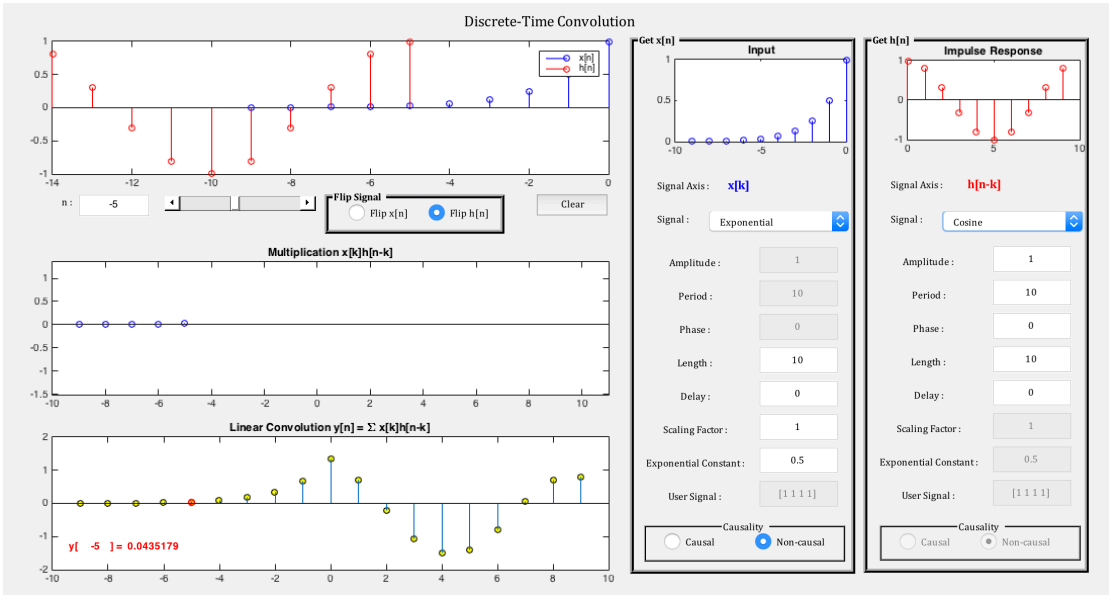


Figure 1.1: DSP MATLAB GUI app – Discrete-time Convolution

1.3. Objective

The goal of this project is to develop a framework that serve as a platform to access all the MATLAB GUI apps.

In this project, the following 2 objectives will be achieved:

- To develop a framework of the learning tool called **Guide Educational Tool (GET)**
- To implement the MATLAB GUI apps under the DSP course using MATLAB GUIDE

1.4. Report Organisation

This report is divided into seven chapters and an overview of each chapter is as follows:

- Chapter 1 provides an introduction of the project that includes the background, proposal and the objective of the project.
- Chapter 2 discuss on the related works that were used in this project
- Chapter 3 provides an overview of the proposed approach and system specification of the GET framework
- Chapter 4 focus on the implementation of the GET framework
- Chapter 5 discuss on the DSP MATLAB GUI apps that has been developed
- Chapter 6 concludes the report with future direction and lesson learnt

2. Literature Review

2.1. Related Works

Example of MATLAB GUI apps that is already available, **Educational MATLAB GUIs** developed by the members of the Center for Signal and Information Processing (CSIP), a part of the School of Electrical and Computer Engineering at the Georgia Institute of Technology. There are total of 11 GUI apps created using MATLAB for learning and teaching signal processing concepts. The GUI apps cover concepts such as Fourier series, Phasors and complex numbers, Continuous and discrete LTI systems, Continuous and discrete convolution, Sampling, Lowpass filter design, Relation between Pole-zero diagram and frequency response. Furthermore, in the **MATLAB File Exchange**, where we can find most programs and MATLAB GUI apps, they are all in the form of individual GUIs [2]. However, so far there is no simple framework that could integrate those GUI apps.

2.1.1. Project theory

The project theory for the **Educational MATLAB GUIs** is to accompany the textbook **Signal Processing First** so that students can appreciate and understand further about the Signal Processing [3]. For the MATLAB File Exchange, where users share their programs and MATLAB GUI apps, we can easily get the ideas to develop the DSP MATLAB GUI apps that we are going to develop in this project.

2.1.2. Project Similarity

The idea of the MATLAB GUI apps that we developed in our project was influenced from the GUI apps of the **Educational MATLAB GUIs**. For example the Discrete-time convolution app, it provides various plot options for enabling the tool to be effectively used as a lecture aid in a classroom environment.

3. Proposed Approach and System Specifications

3.1. System Structure

The idea of GET system structure is very simple. What we want to achieve in developing the GET is when the user runs the main code of the GET, the system will automatically search the available course, topic and follow by the sub-topic to access the MATLAB GUI apps. In addition, user did not need to modify any part of the code in the main code whenever new MATLAB GUI apps are added. Thus we designed the GET structure as shown in Figure 3.1 to achieve our objective.

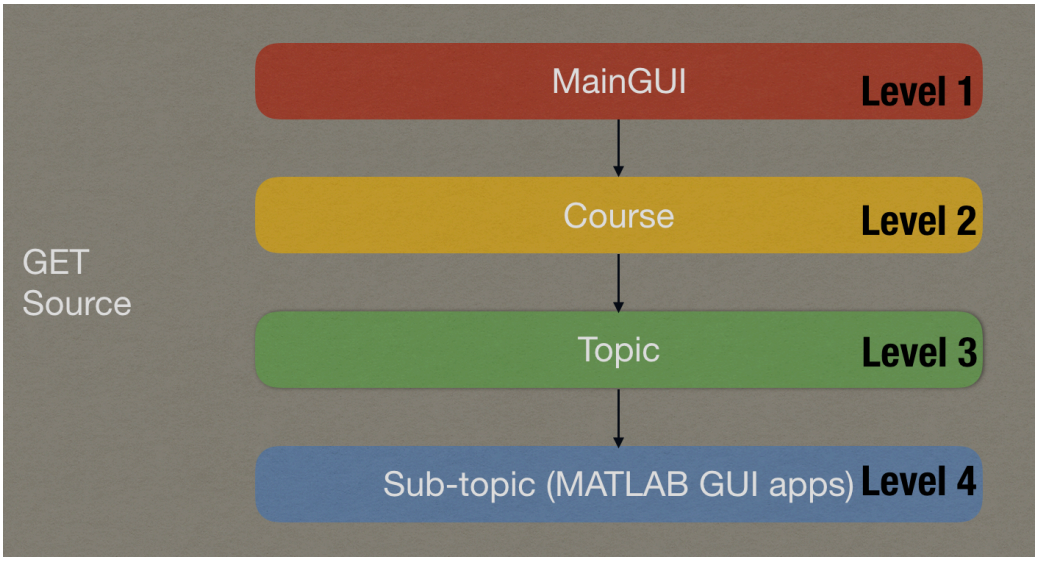


Figure 3.1: GET System Structure

There are 4 hierarchy levels in the GET structure – MainGUI, Course, Topic and Sub-Topic. The MainGUI consist of the main code (Main GUI app in short MainGUI) and it functions as a platform to access the various MATLAB GUI apps. The MATLAB GUI apps will be classified as level 3 that is the Sub-Topic. The Course and the Topic serves as a connection from the MainGUI to the MATLAB GUI apps.

This also led us in considering the structure of the Current Folder in the MATLAB. Although it is not a huge cause of concern, by doing this the future user will be easier to navigate the folders. Figure 3.2 shows the structure of the Current Folder. The folder of the GET will be named ‘Source’.

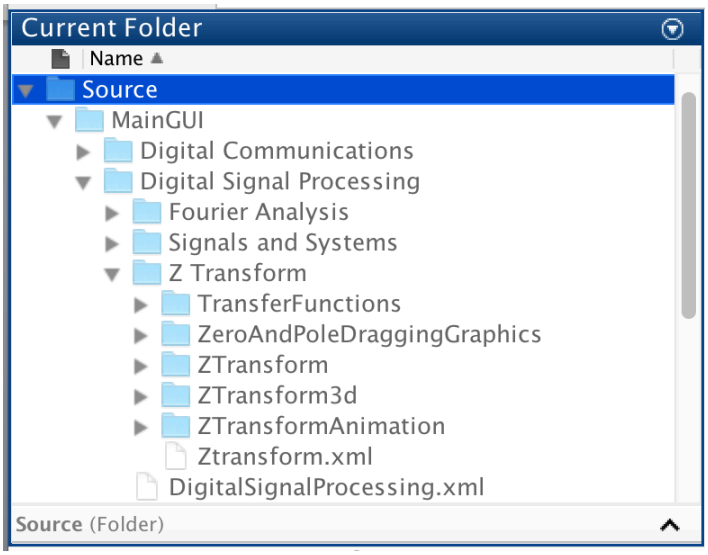


Figure 3.2: The GET Current Folder (Source)

3.2. Definitions

DSP:	Digital Signal Processing
GET:	Guide Educational Tools
GUI:	Graphical User Interface
MATLAB GUI app:	The learning tool (application) in the form of GUI (subtopic level)
MainGUI:	The main graphical user interface that act as a framework that integrates all the MATLAB GUI apps
MainGUI.fig:	The GET figure file
MainGUI.m:	The GET m-file
*.m:	Extension of the GUI code file
*.fig:	Extension of the GUI figure file
*.xml:	Extension of the xml file

3.3. Development Strategy

Agile approach was used in this project, as every new MATLAB GUI apps that is being implemented has to be tested straight away after developing the program. Agile approach promotes adaptive planning and encourages rapid identification to problems at stage with its iterative working style. The agile methodology developed fast where project scope is flexible and requirements may change.

3.4. System Architecture

Figure 3.3 shows the system architecture for the GET. In this project, an XML-based system architecture is used. The GET will include the MainGUI as the platform to access all the MATLAB GUI apps. The MATLAB GUI apps are classified under the Sub-Topic level. The 1st level and 2nd level, which are the Course and the Topic respectively, serves as a connection to the MATLAB GUI apps. To connect all the levels we will use *.xml file such that it will link the MATLAB GUI apps according to which topic and course they belong to. We will discuss the implementation in detail about the *.xml file in Chapter 4.

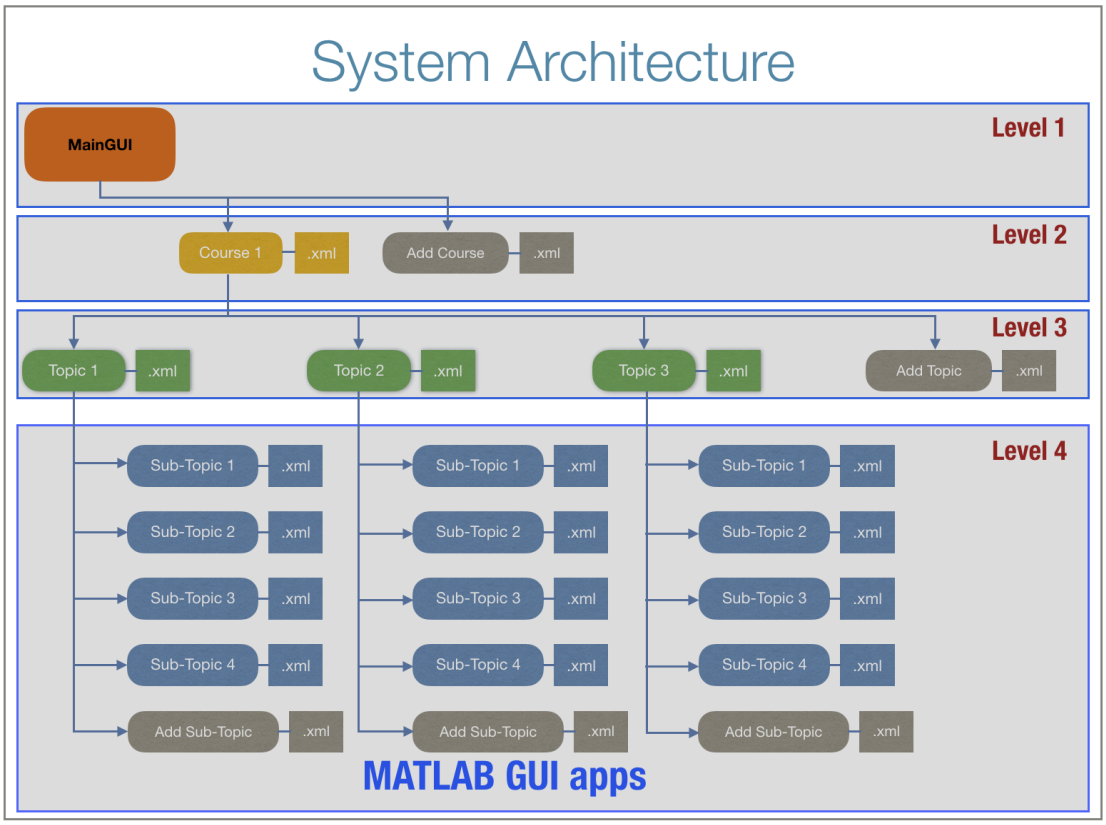


Figure 3.3: GET System Architecture

3.5. Summary of System Workflow

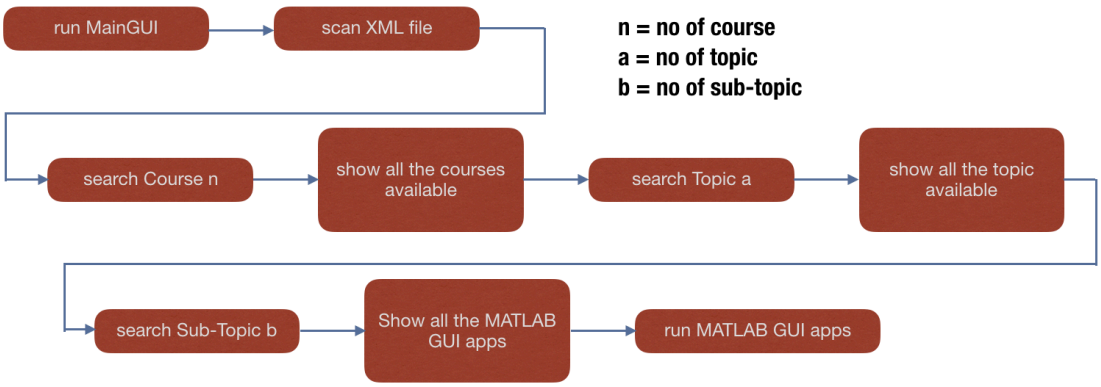


Figure 3.4: System Workflow

The figure above provides the summary of the entire system workflow.

3.6. Tools and Technologies

The Table 3.1 below provides an overview of the environment setup of this project.

Hardware	8 GB 1600 MHz DDR3, 2.4 GHz Intel Core i5
Operating System	OS X El Capitan Version 10.11.3
MATLAB Version	8.5.0.197613 (R2015a)

Table 3.1: Environment setup

3.6.1. MATLAB GUI

The MATLAB GUI apps can be built in two ways:

1. Using GUIDE (GUI Development Environment)
2. Coding from MATLAB editor

In this project we will be using GUIDE to develop the DSP MATLAB GUI apps as well as the MainGUI (GET main code). GUIDE is MATLAB's Graphical User Interface (GUI) Development Environment. It provides a set of tools for creating graphical user interfaces (GUIs) [4]. These tools simplify the process of programming layout the components on a figure. Thus for the ease of the programmers we will be using MATLAB GUIDE to implement all the GUI apps.

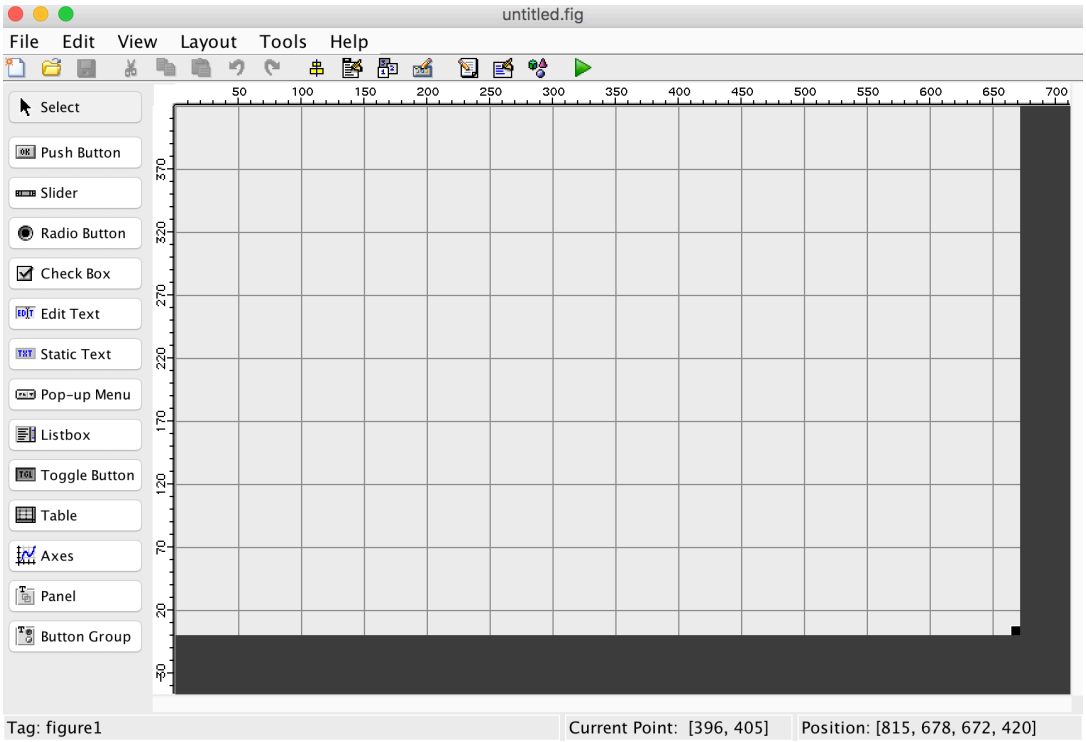


Figure 3.5: Layout editor of MATLAB GUIDE

Generally, creating GUI apps in GUIDE, it automatically generates two files which are the figure (*.fig) file where it contains a complete description of the GUI figure file and the GUI components and the code (*.m) file where it contains the code that controls the GUI. The *.fig is the extension of GUI figure files and *.m is the extension of the GUI code files. The *.m file will provide a framework for the implementation of the callbacks, the functions that execute when users activate a component in the GUI.

Any changes in the *.fig file will be made in the Layout editor as shown in Figure 3.5. The initialization of the code the callbacks are done in the *.m file. The programming of the behaviour of the GUI also to be done in the *.m file.

Thus we can conclude that the three basic tasks in process of implementing a GUI is first by designing the GUI in a piece of paper. The second task is by laying out a GUI where MATLAB implement GUI apps as figure windows containing various styles of the components using the layout editor of the MATLAB GUIDE. The last task is programming the GUI by editing the code in the *.m file, where each object must be program to perform the intended action when activated by the user of GUI.

3.6.2. MATLAB GUI Components

The various styles of the GUI components are:

(a) Push Button

(b) Slider

(c) Radio Button

(d) Check Box

(e) Edit Text

(f) Static Text

(g) Pop-up Menu

(h) Listbox

(i) Toggle Button

(j) Table

(k) Axes

(l) Panel

(m) Button Group

These components may vary depending on the version of MATLAB. The listed components above were based on the MATLAB version R2015a. We will be using the components above extensively when building the MATLAB GUI apps.

3.6.3. MATLAB m-file (*.m)

When all the GUI compoenents are added, the rest of the task is coding the *.m file. Codes are written under various functions. Functions are generated automatically. The opening function and callback functions are most important. Functions can be accessed as shown in Figure 3.6.

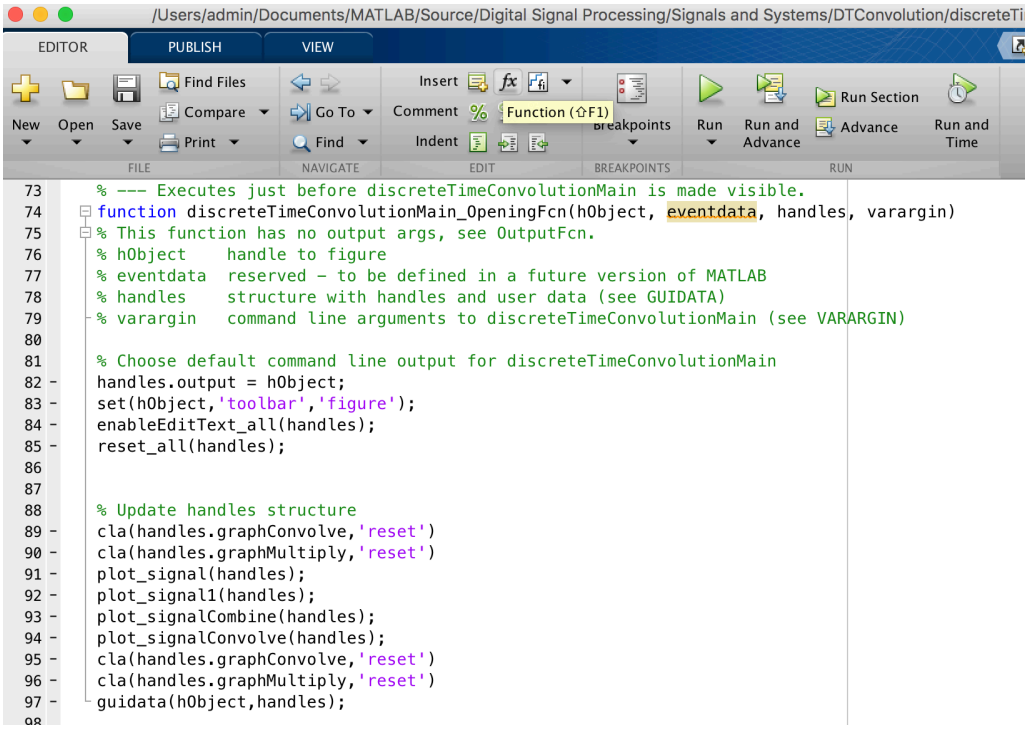


Figure 3.6: Accessing functions in *.m file

Coding the *.m file is a crucial steps in buidling the MATLAB GUI apps. As a developer or user for the MATLAB GUI apps, we would want the GUI apps to function properly.

3.6.4. Signal Processing Toolbox Functions

The Signal Processing Toolbox is a collection of tools built on the MATLAB numeric computing environment. The toolbox supports a wide range of signal processing operations, from waveform generation to filter design and implementation, parametric modeling, and spectral analysis. The Table 3.2 below shows some the functions from Signal Processing Toolbox that used in implementing the DSP MATLAB GUI apps.

Functions	**Description**
sin	Sine of argument in radians
stem	Plot discrete sequence data
max	Largest elements in array
min	Smallest elements in array
conv	Convolution and polynomial multiplication
double	Double precision
freqz	Frequency response of digital filter
zplane	Zero pole plot
tf2zp	Convert transfer function filter parameters to zero-pole-gain form
zp2tf	Convert zero-pole-gain filter parameters to transfer function form
fft/ ifft	Fast Fourier Transform /Inverse Fast Fourier Transform
angle	Phase angle

Table 3.2: Some Signal Processing Toolbox function

4. The Implementation of The GET Framework

4.1. The GET Graphical User Interface

To implement the framework of the GET, first we need to design the user interface of the MainGUI using the Tools and Technologies mentioned in the Chapter 3. The MainGUI that is classified as level 1, includes the *.fig file and the *.m file. Figure 4.1 shows the GET figure file called the MainGUI.fig. The MainGUI provides a simple interface such that user can select the courses, topics and follow by the sub-topics that include a description and a snapshot for the user to refer.

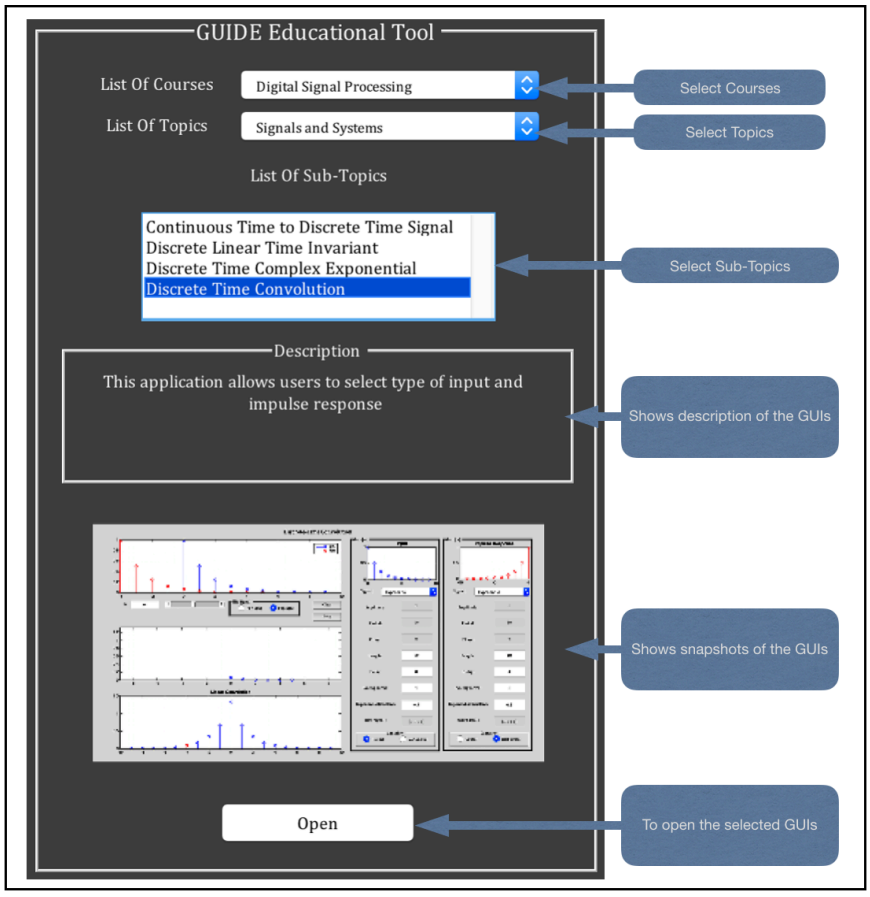


Figure 4.1: MainGUI.fig

In the MainGUI.fig consists of the following GUI components and its functions as shown in Table 4.1 below:

GUI components	**Functions**
Two pop-up menu	For selecting courses and topics
List Box	For showing the sub-topics
Static text	For showing descriptions
Axes	For showing snapshots
Push button	Open button to link the selected GUIs

Table 4.1: MainGUI Components and Functions

The two pop-up menus are for the list of courses and topics respectively. The list box is for the list of sub-topics. Each sub-topic has its own description and snapshot. The push button is for opening new MATLAB GUI apps depending which sub-topic that has been chosen.

4.2. The GET Functionalities

After the MainGUI.fig components have been added, we need to code the functionalities of the MainGUI. That means we need to code in the *.m file. The MainGUI *.m file is called the MainGUI.m. But before coding the functionalities in the *.m file, first we will again look at the workflow of the MainGUI.

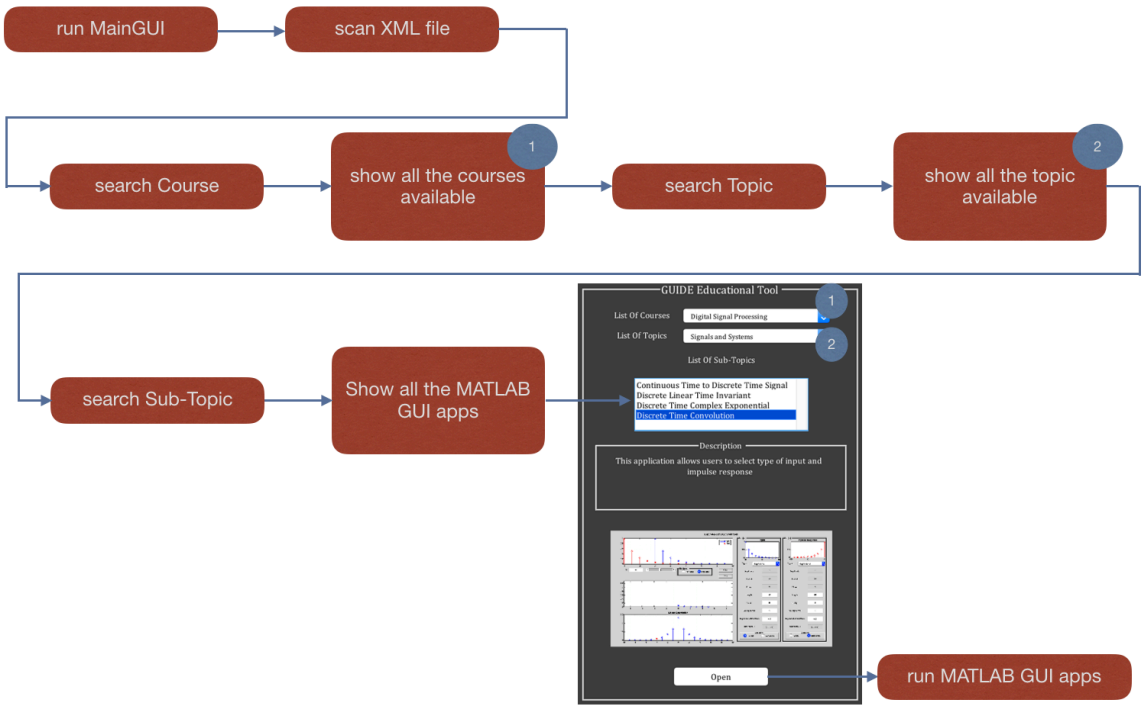


Figure 4.2: MainGUI Workflow

Figure 4.2 shows the workflow of the MainGUI. When the user runs the MainGUI, the program (MainGUI.m) will automatically search the entire available course and follow by the topic and the sub-topic by scanning all the *.xml files. The list of the sub-topics, which are the MATLAB GUI apps, will appear accordingly from the

course and the topic that has been chosen by the user. Practically if a MATLAB GUI app is available, its following course and topic are also available.

We discussed earlier that an XML-based system architecture is used to implement the GET framework. We designed a simple template that serves our purpose in building the MainGUI. Each of the level: Courses, Topics and Sub-Topics are assigned to 1 *.xml file template. The template has 1 root and 7 branches (elements). The root is called <main_gui> and the 7 branches are <type>, <topic>, <subtopic>, <title>, <function>, <description> and <snapshot>. The template of the *.xml file is shown in Figure 4.3.

```
<main_gui>
<type>course</type>
<course>Enter the course name here</course>
<topic>Leave blank</topic>
<subtopic>Leave blank</subtopic>
<function>Leave blank</function>
<description>Leave blank</description>
<snapshot>Leave blank</snapshot>
</main_gui>
```

```
<main_gui>
<type>topic</type>
<course>Enter the course name here</course>
<topic>Enter the topic name here</topic>
<subtopic>Leave blank</subtopic>
<function>Leave blank</function>
<description>Leave blank</description>
<snapshot>Leave blank</snapshot>
</main_gui>
```

```
<main_gui>
<type>subtopic</type>
<course>Enter the course name here</course>
<topic>Enter the topic name here</topic>
<subtopic>Enter the sub-topic name here</subtopic>
<function>The *.m file (MATLAB GUI app)</function>
<description>The desription of the app</description>
<snapshot>The snapshot file of the app</snapshot>
</main_gui>
```

Figure 4.3: The template of the *.xml file

The name of the xml structure is the <main_gui>. The <type> element has 3 inputs, which are the level 2, 3 and 4 in the GET structure: 'course' as the 2nd level, 'topic' as the 3rd level and 'subtopic' as the 4th level (shown in Table 4.2). The MATLAB GUI apps that specified under the Sub-Topic level it has the <type> subtopic for its *.xml file.

Level	**<type>**
1st Level: Course	Course
2nd Level: Topic	Topic
3rd Level: Sub-Topic (MATLAB GUIs)	Subtopic

Table 4.2: The input for <type> element

Figure 4.4 shows the *.xml files for level 2, 3 and 4. This set of *.xml files belongs to 'Transfer Function' from the 'Z Transform' topic and 'Digital Signal Processing' course. The 3 elements <function>, <description> and <snapshot> are only applicable for the <type> subtopic. The 3 elements mentioned can leave blank for the <type> course and topic. The input for the element <function> is specifically for the *.m file. For each GUI apps that the user develops, each must strictly specify its topic and course correctly.

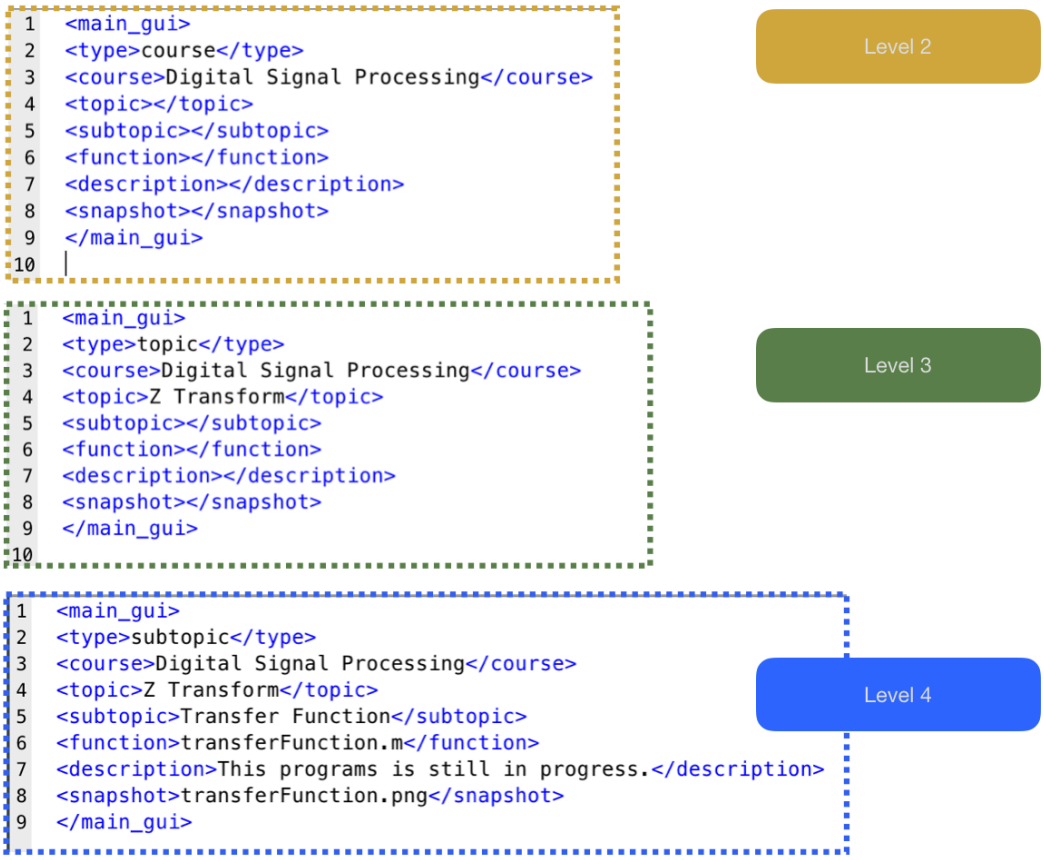


Figure 4.4: A complete set of *.xml files

Refer to Appendix A for the full code of the MainGUI. To implement this functionality, two functions are integrated in the MainGUI.m code. The two functions are rdir.m and xml2struct.m. These functions are available online (MATLAB File Exchange) and we use these functions for the following purpose:

1. To scan all the available *.xml files under the <main_gui>
2. To convert an Extensible Markup Language (XML) file into a MATLAB structure for easy access to the data

4.3. The Procedure for Adding New MATLAB GUI Apps

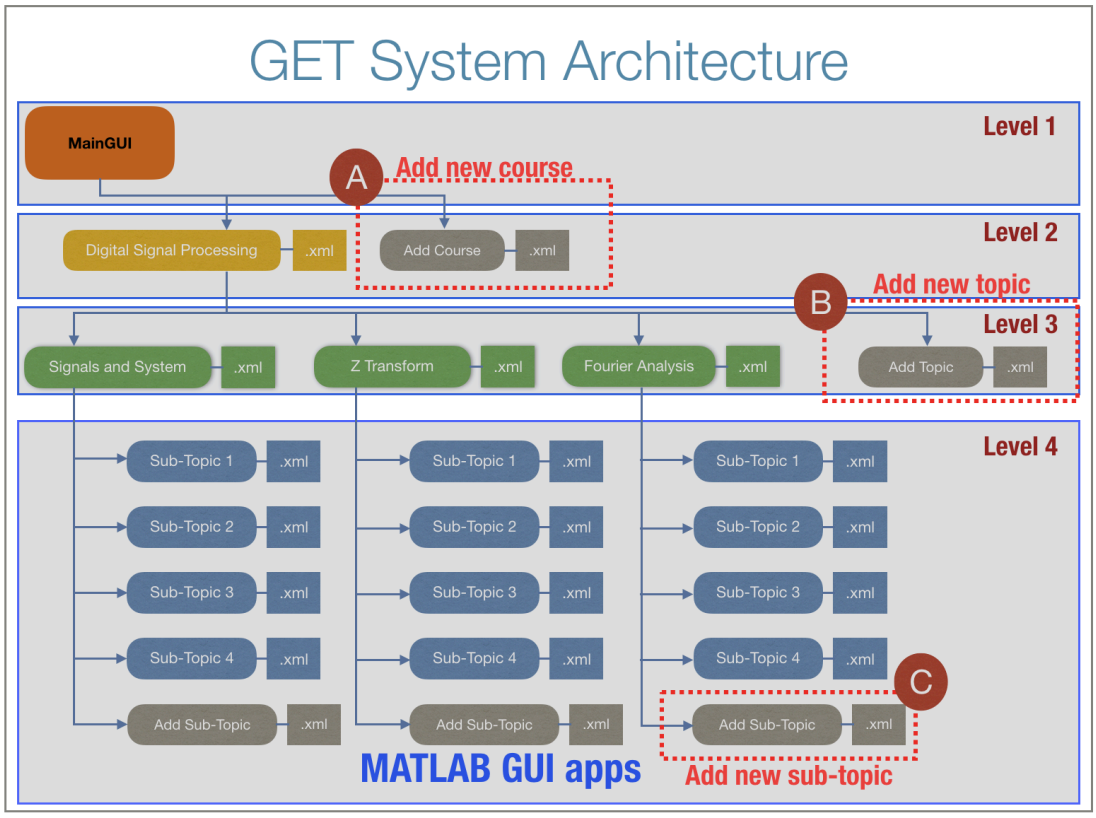


Figure 4.5: Current GET System Architecture

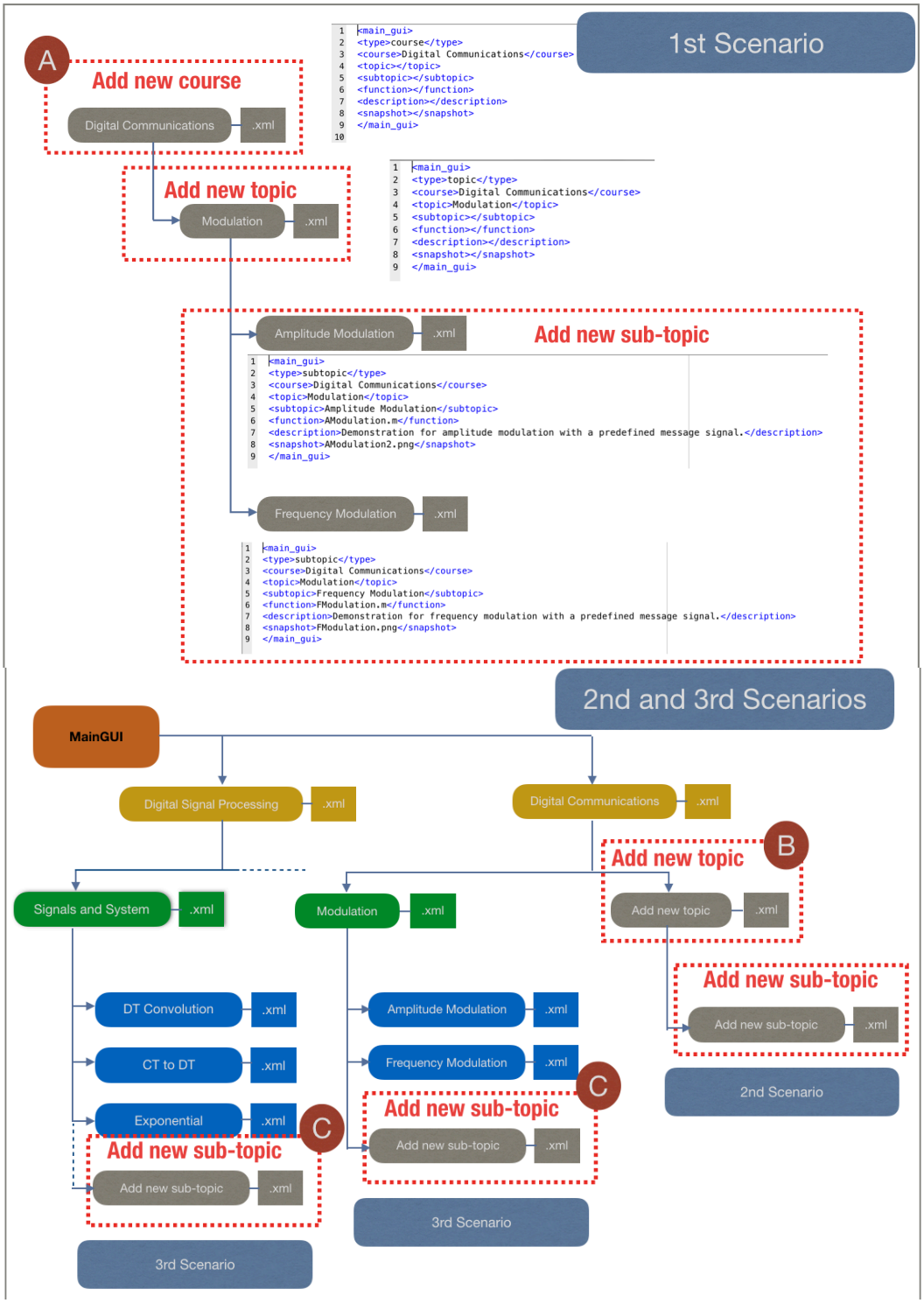


Figure 4.7: Scenario 1, 2 and 3

Figure 4.5 and Figure 4.6 shows the GET system architecture and the procedure on how to add course, topic and sub-topic in the following 3 scenarios. Mainly user wants to add the MATLAB GUI apps, which are listed under the sub-topic. There are 3 scenarios that can possibly happen. Firstly, adding new MATLAB GUI apps with new topic and new course. Secondly, adding new MATLAB GUI apps with new topic and with the existing course. Lastly, adding new MATLAB GUI apps with the existing course and topic.

In the GET current system, it has one course – Digital Signal Processing and three topics that are available. Assuming that each topic has 4 sub-topics, meaning that it has 4 MATLAB GUI apps for each topic. In the 1st scenario, Figure 4.6 shows an example on how to add new course, new topic and new sub-topic. The new course name is 'Digital Communications'. Its *.xml file will be in the same level as the Digital Signal Processing Course. Thus, the *.xml file for the input <type> element will be the 'course'. For the purpose of this project, we developed 2 examples of the MATLAB GUI apps for this course so as to make sure that the functionalities of the MainGUI is working. The developed MATLAB GUI apps under the course 'Digital Communications' are 'Amplitude Modulation' and 'Frequency Modulation'. These two apps are classified under the topic 'Modulation'.

In the 2nd scenario and 3rd scenario, the course is already available and user only wants to add the MATLAB GUI apps. In the 2nd scenario, user adds the MATLAB GUI apps with no available topic and has to add the new matching topic for the MATLAB GUI apps. In the 3rd scenarios, user adds a new MATLAB GUI app with the existing course and topics that are already available.

4.4. The Structure of the GET Current Folder

The current folders of the GET are structured as shown in Figure 4.7.

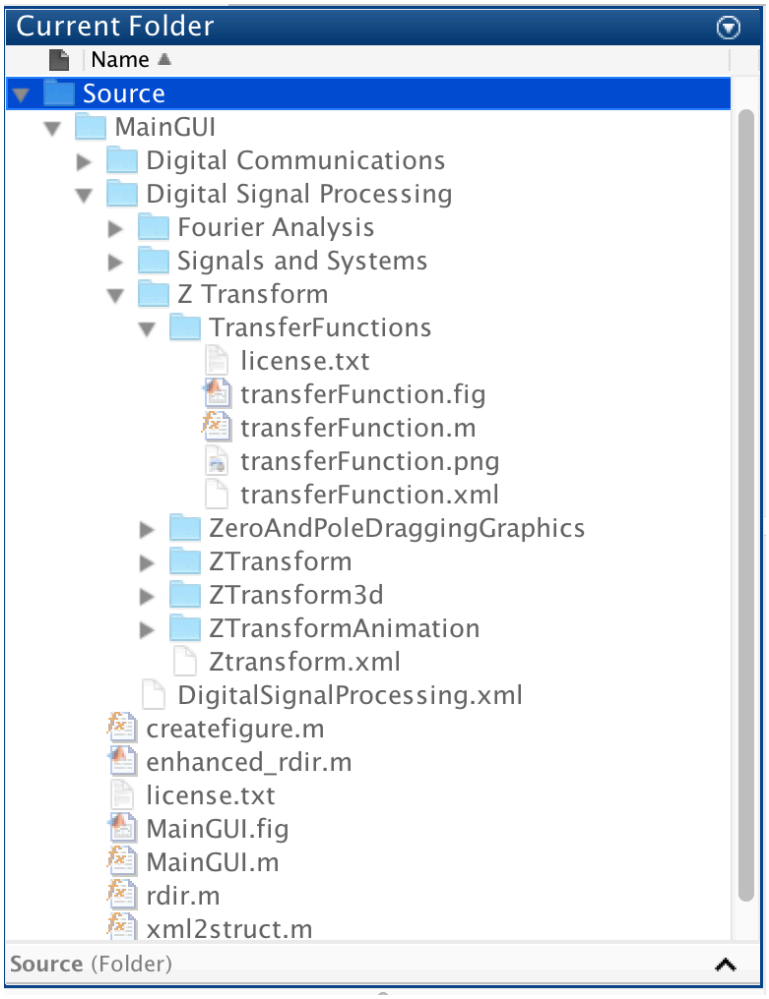


Figure 4.7: GET Current Folder

As we mentioned in the Chapter 3, we structured the current folder in the same way as the GET system structure. The 1st level folder contained the main code (MainGUI.m) and the figure file (MainGUI.fig) as well as the other two functions – rdir.m and xml2struct.m. The 2nd level is the course folder that contained the *.xml file of the <type> course and the 3rd level is the topic folder that contained the *.xml file of the <type> topic. The last level is the sub-topic folder that contained the MATLAB GUI

apps as well as the *.xml file of the <type> subtopic. In the case that the user put in the content of the *.fig file and the *.m file in the wrong folder, the MainGUI would still be able to allocate these files as long as the content of the *.xml file is correct.

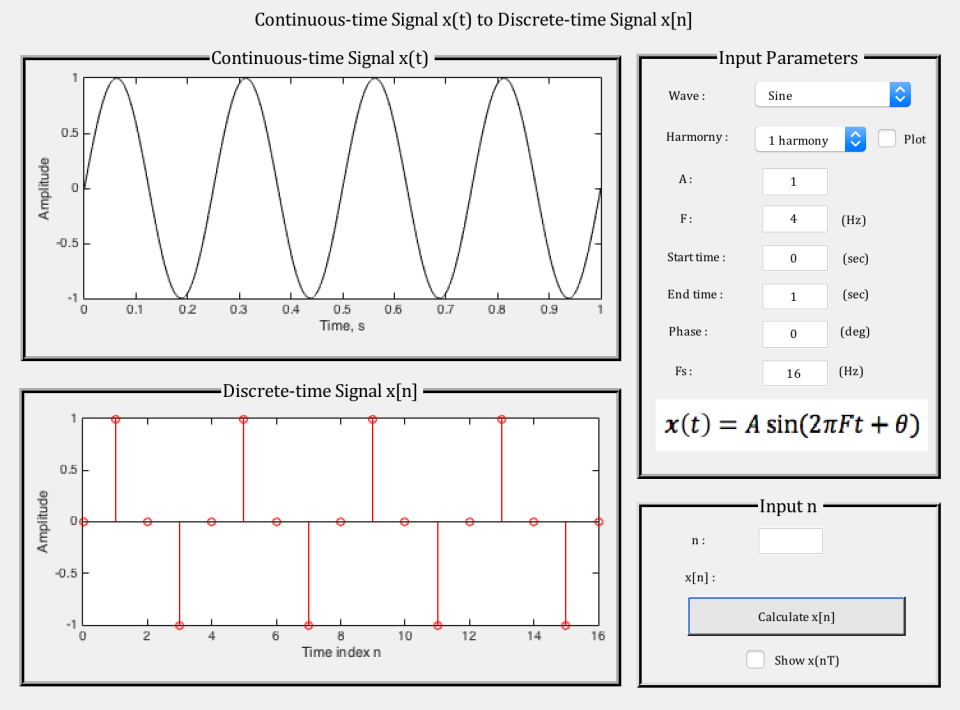
5. MATLAB GUI Apps

5.1. The completed MATLAB GUI apps for DSP

The three topics under DSP course that has been developed are Signals and Systems, Z-Transform and Fourier Analysis. The following list are the completed MATLAB GUI apps:

- Continuous-time Signal x(t) to Discrete-time Signal x[n]
- Discrete-time Convolution
- Discrete-time Complex Exponential
- Transfer Function
- Z-Transform zero and pole dragging graphics
- Fourier series animation version 1
- Fourier series animation version 2
- Discrete Fourier Transform

Continuous-time Signal x(t) to Discrete-time Signal x[n]

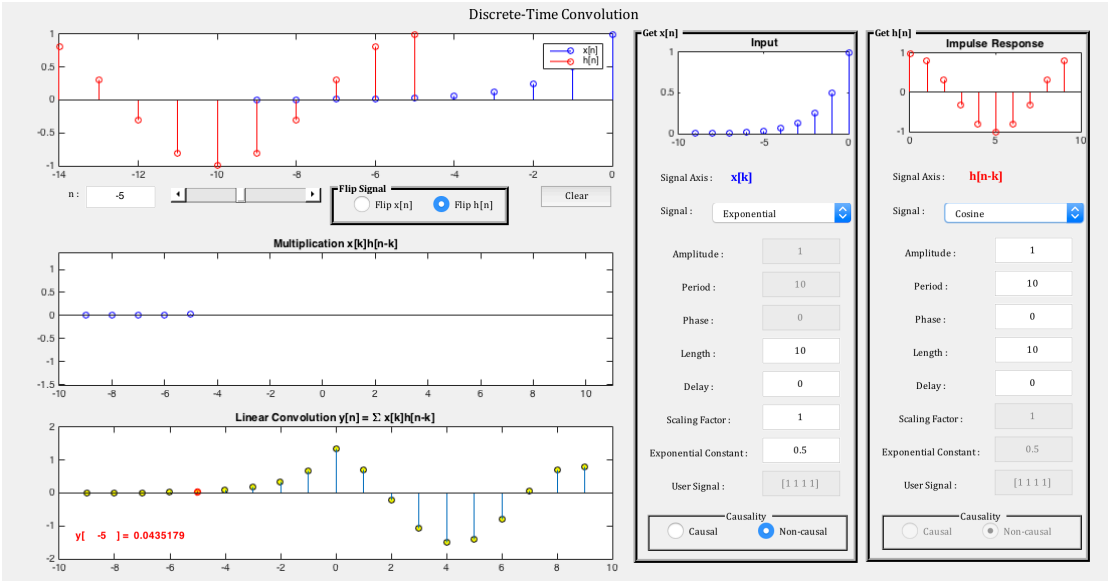


The **Continuous-time Signal x(t) to Discrete-time Signal x[n]** app is an application that allows to create a set of periodic continuous time signals and convert to discrete time signals.

The features:

- Users can change the input frequency and sampling rate values.
- Able to show the value of x[n] on the graph.

Discrete-time Convolution

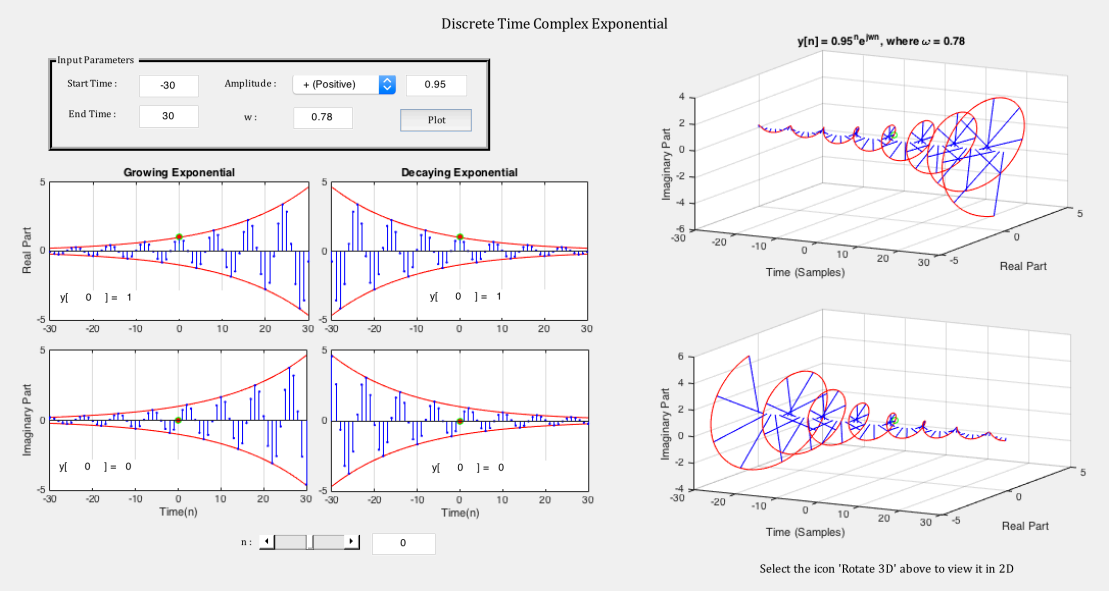


The **Discrete-time Convolution** app is an application that helps visualize the process of discrete-time convolution.

The features:

- Users can choose from a variety of different signals: sine, cosine, exponential, pulse, unit sample and user signal.
- Users can observe in the multiplication and the linear convolution graph as the n input value is being shifted
- Various plot options enable the tool to be effectively used as a lecture aid in a classroom environment.

Discrete-time Complex Exponential

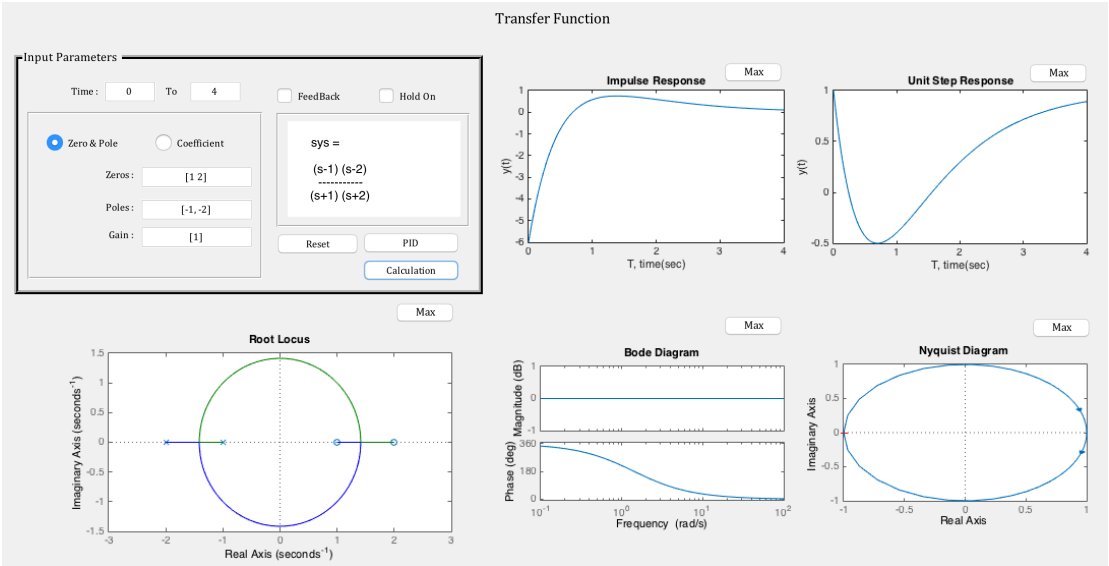


The **Discrete-time Complex Exponential** app is an application that allows users to visualize the complex exponential sequence evolving in 2D and 3D.

The features:

- Users can change the amplitude and the digital angular frequency values.
- Able to show the y[n] value on the graph

Transfer Function

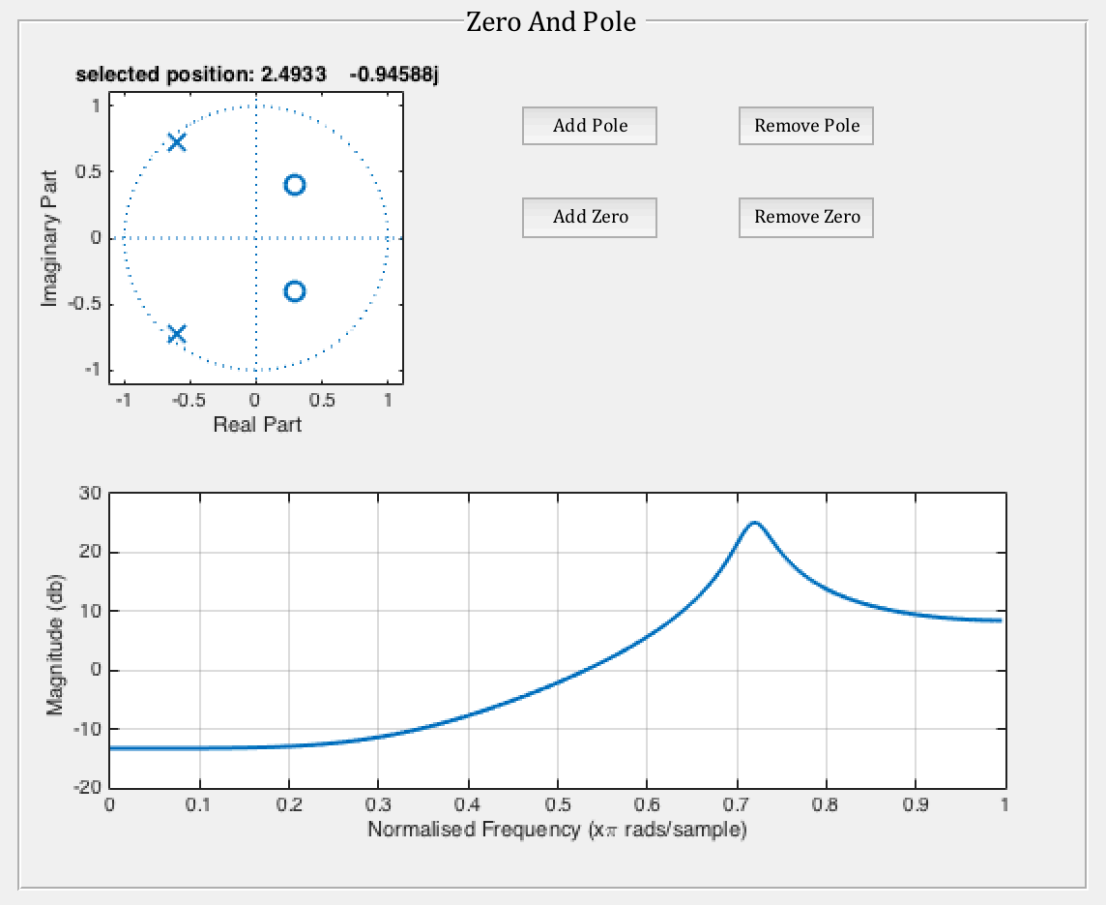


The **Transfer Function** app is an application that allows user to visualize the impulse response and the unit step response from the input of zeros and poles.

The features:

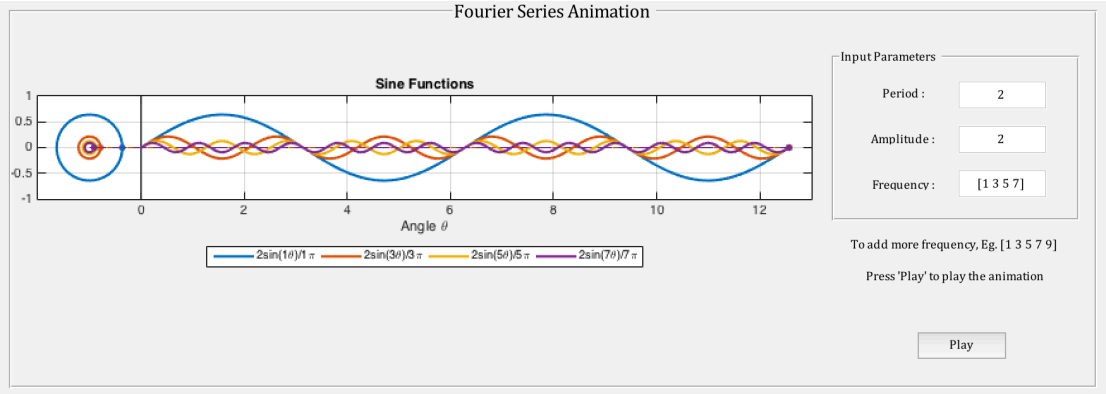
- Provides results from a step input, impulse input and diagrams of Root Locus, Bode, and Nyquist.
- Able to see multi-results of the transfer function that user insert.
- Shows the math equation of the transfer function on the figure.
- Furthermore, users can zoom in/out on the figure by using 'max' button.

Z-Transform zero and pole dragging graphics

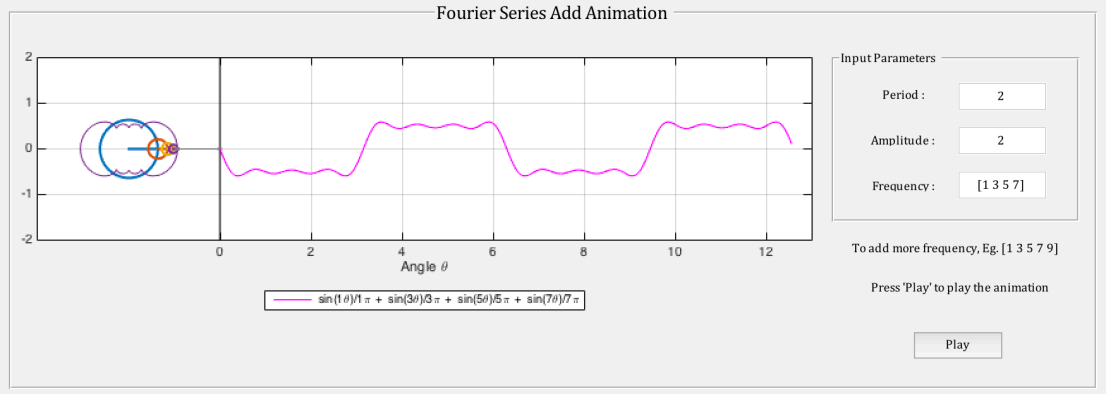


The **Z-Transform zero and pole dragging graphics** app is an application where user can drag the zeros and poles in the graph and the magnitude graph will update automatically.

Fourier series animation version 1



Fourier series animation version 2

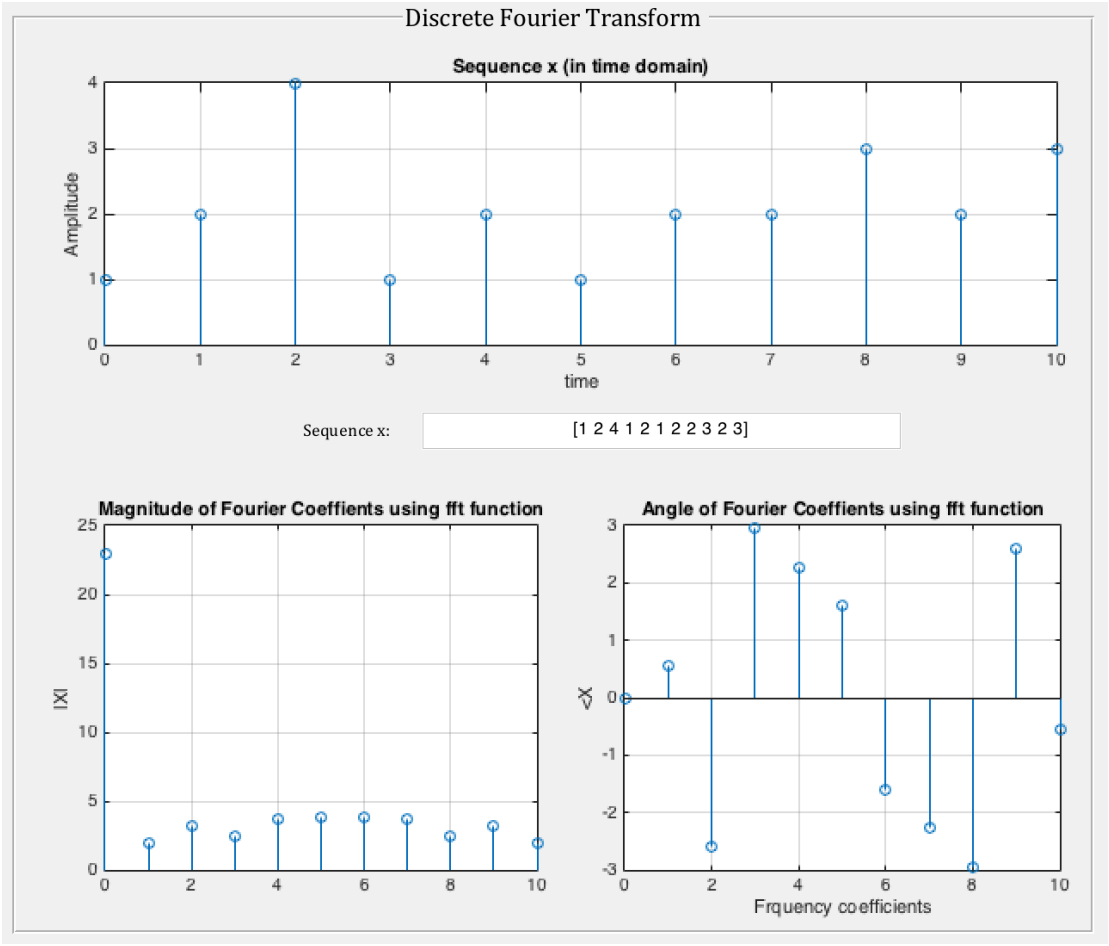


The **Fourier series animation version 1** is an application demonstrating Fourier series representation of periodic signals. The **Fourier series animation version 2** is an application demonstrating Fourier series representation of periodic signals as a sum of sinusoidal functions

The features of the two apps:

- Users can change the period, amplitude and frequency values.
- Simple animation that only plays when the play button is pressed.

Discrete Fourier Transform



The **Discrete Fourier Transform** app is an application that allow users to visualize the magnitude and angle of the Fourier Coefficients.

6. Conclusion

6.1. Summary of Achievements

In this project, the framework of the GET was implemented. When the user want to add new MATLAB GUI apps, no changing of the main code is needed. The user simply follows the procedure on how to add new MATLAB GUI app.

In conclusion, a total of 8 DPS MATLAB GUI apps were developed. A new course – Digital Communications was also created in order to make sure that the functionality of the MainGUI is working fine. The MainGUI is able to do an automatically search of all the MATLAB GUI apps that are available in the GET system.

6.2. Reflection and Future Work

In this project, future research should explore following directions: 1) improvement on the design and the functionality of the GET; 2) improvement on the DSP MATLAB GUI app and 3) development of a new MATLAB GUI app for the Digital Communication course.

References

[1] Educational MATLAB GUIs, retrieved from: http://users.ece.gatech.edu/mcclella/matlabGUIs/

[2] Matlab File Exchange, retrieved from: http://www.mathworks.com/matlabcentral/fileexchange/

[3] James H. McClellan, Mark A. Yoder, and Ronald W. Schafer. *Signal Processing First,* 1st edition. Pearson/Prentice Hall, 2003

[4] Using MATLAB to develop standalone graphical user interface (GUI) software packages for educational purposes, retrieved from: http://cdn.intechopen.com/pdfs-wm/11608.pdf

[5] Alan V. Oppenheim, Ronald W. Schafer, John R. Buck, *Discrete-Time Signal Processing*, Prentice Hall, ISBN 0-13-754920-2

[6] Ingle, Vinay K.; John G. Proakis , *Digital Signal Processing Using MATLAB* , 3rd edition. Stamford, CT: CL Engineering, ISBN 1111427372.

[7] Gilat, Amos, *MATLAB: An Introduction with Applications 2nd Edition*. John Wiley & Sons. *ISBN 978-0-471-69420-5*.

[8] XML Tutorial, retrieved from: http://www.w3schools.com/xml/xml_tree.asp

[9] John G.Proakis and Dimitris G. Manolakis. *Digital Signal Processing: Principles, Algorithms and Applications,* 3rd edition. New Jersey: Prentice Hall, 1995

[10] Sanjit K. Mitra, *Digital Signal Processing: a Computer-Based Approach*, 2nd edition. Irwin: McGraw-Hill, 2001

Appendix A: MainGUI.m

```
% Final Year Project CE
% Author:           Nonny Florentine (U1320766B)
%
% MainGUI
%
% Lastest update:   15 March 2016

function varargout = MainGUI(varargin)
% MAINGUI MATLAB code for MainGUI.fig
%      MAINGUI, by itself, creates a new MAINGUI or raises the
existing
%      singleton*.
%
%      H = MAINGUI returns the handle to a new MAINGUI or the handle
to
%      the existing singleton*.
%
%      MAINGUI('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in MAINGUI.M with the given input
arguments.
%
%      MAINGUI('Property','Value',...) creates a new MAINGUI or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before MainGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to MainGUI_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help MainGUI

% Last Modified by GUIDE v2.5 16-Mar-2016 01:56:26

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @MainGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @MainGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before MainGUI is made visible.
function MainGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to MainGUI (see VARARGIN)

% Choose default command line output for MainGUI
handles.output = hObject;

% ------------------------------------------------------------------%
% The MainGUI
% ------------------------------------------------------------------%

% scan xml file
if isunix
    xmlinfo = rdir(['../','/**/*.xml']);
else
    xmlinfo = rdir(['..\','\**\*.xml']);
end

numCourse = 0;

for i=1:length(xmlinfo)
    xmlstruct = xml2struct(xmlinfo(i).name);
    switch lower(xmlstruct.main_gui.type.Text)
        case 'course' % 2nd Level
            numCourse = numCourse + 1;
            handles.course_group(numCourse) = xmlstruct.main_gui;
    end
end

for i=1:length(xmlinfo)
    xmlstruct = xml2struct(xmlinfo(i).name);
    [path,name, ext] = fileparts(xmlinfo(i).name);
    switch lower(xmlstruct.main_gui.type.Text)
        case 'topic' % 3rd Level
            for y = 1:length(handles.course_group)
                if
strcmp(handles.course_group(y).course.Text,xmlstruct.main_gui.course.
Text)
                    if
isfield(handles.course_group(y),{'topic_group'})
                        xmlstruct.main_gui.path = path;
                        handles.course_group(y).topic_group(end+1) =
xmlstruct.main_gui;

%handles.course_group(y).topic_group(end+1).path = path;
                    else
                        handles.course_group(y).topic_group(1) =
xmlstruct.main_gui;
                        handles.course_group(y).topic_group(1).path =
path;
                    end
```

```
                end
            end

    end
end



for i=1:length(xmlinfo)
    xmlstruct = xml2struct(xmlinfo(i).name);
    [path,name, ext] = fileparts(xmlinfo(i).name);
    switch lower(xmlstruct.main_gui.type.Text)

        case 'subtopic' % 4th Level
            course_index = 0;
            % find a course
            for y = 1:length(handles.course_group)
                if
strcmp(handles.course_group(y).course.Text,xmlstruct.main_gui.course.
Text)
                    course_index = y;
                end
            end

            % find a topic
            for g =
1:length(handles.course_group(course_index).topic_group)
                if
strcmp(handles.course_group(course_index).topic_group(g).topic.Text,x
mlstruct.main_gui.topic.Text)
                    if
isfield(handles.course_group(course_index).topic_group(g),{'subTopic_
group'})
                        xmlstruct.main_gui.path = path;

handles.course_group(course_index).topic_group(g).subTopic_group(end+
1) = xmlstruct.main_gui;

%handles.course_group(module_index).topic_group(g).subTopic_group(end
+1).path = path;
                    else

handles.course_group(course_index).topic_group(g).subTopic_group(1) =
xmlstruct.main_gui;

handles.course_group(course_index).topic_group(g).subTopic_group(1).p
ath = path;
                    end
                end
            end

        end

end

% ------------------------------------------------------------------%
```

```
% ------------------------------------------------------------------%
% For the Main GUI, 1st pop-up menu, 2nd pop-up menu and the list box
% ------------------------------------------------------------------%

course = get(handles.coursePopupMenu,'value');
topic = get(handles.topicPopupMenu,'value');

% populate module
for i=1:length(handles.course_group) % length of course_group = 1 2
    courseStr{i} = [handles.course_group(i).course.Text];
end
set(handles.coursePopupMenu,'String',courseStr);



populateTopics(handles,course)
populateSubTopics(hObject,handles,course,1,1)

function populateTopics (handles, course)
% populates topic
ii=1;
for i=1:length(handles.course_group(course).topic_group)
    if
strcmp(handles.course_group(course).course.Text,handles.course_group(
course).topic_group(i).course.Text)
        topicStr{i} =
[handles.course_group(course).topic_group(i).topic.Text];
        ii=ii+1;
    end
end

set(handles.topicPopupMenu,'String',topicStr);


function populateSubTopics(hObject,handles,course,topic,subTopic)
% populates subtopic
count=0;
for k = 1:length(handles.course_group(course).topic_group)
       if
strcmp(handles.course_group(course).course.Text,handles.course_group(
course).topic_group(k).course.Text)
            count=count+1;
            if count==topic
                 for
i=1:length(handles.course_group(course).topic_group(k).subTopic_group
)
                     if
strcmp(handles.course_group(course).topic_group(k).topic.Text,handles
.course_group(course).topic_group(k).subTopic_group(i).topic.Text)
                        subTopicStr{i} =
[handles.course_group(course).topic_group(topic).subTopic_group(i).su
btopic.Text];
                     end
                 end
                 realTopic=k;
            end
       end
end
```

```
% variables in handles for button value
handles.realSubmodule=realTopic;
handles.realSubsubmodule=subTopic;

set(handles.subTopicList,'String',subTopicStr);

display_subTopic(handles, course, topic, subTopic);

% ------------------------------------------------------------------%

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes MainGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = MainGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in topicPopupMenu.
function topicPopupMenu_Callback(hObject, eventdata, handles)
% hObject    handle to topicPopupMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
topicPopupMenu contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
topicPopupMenu


% get the current topic to show
topic = get(handles.topicPopupMenu,'value');
course = get(handles.coursePopupMenu,'value');
subTopic = get(handles.subTopicList,'value');

set(handles.startText,'visible','off');
set(handles.subTopicList,'value',1);
populateSubTopics(hObject,handles,course,topic,1)

display_subTopic(handles, course, topic, subTopic);
guidata(hObject, handles);

%% Display the information of the MATLAB GUI app
function display_subTopic(handles,course, topic, subTopic)
% display description
```

```
set(handles.desText,'String',handles.course_group(course).topic_group
(topic).subTopic_group(subTopic).description.Text);
% display snapshot
if
~isempty(handles.course_group(course).topic_group(topic).subTopic_gro
up(subTopic).snapshot.Text)
    axes(handles.snapshotAxes)

imshow([handles.course_group(course).topic_group(topic).subTopic_grou
p(subTopic).path '/'
handles.course_group(course).topic_group(topic).subTopic_group(subTop
ic).snapshot.Text]);
end


% --- Executes during object creation, after setting all properties.
function topicPopupMenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to topicPopupMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in subTopicList.
function subTopicList_Callback(hObject, eventdata, handles)
% hObject    handle to subTopicList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
subTopicList contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
subTopicList
topic = get(handles.topicPopupMenu,'value');
subTopic = get(handles.subTopicList,'value');
course = get(handles.coursePopupMenu,'value');

populateSubTopics(hObject,handles,course,topic,subTopic)

set(handles.startText,'visible','off');
display_subTopic(handles,course,topic,subTopic);
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function subTopicList_CreateFcn(hObject, eventdata, handles)
% hObject    handle to subTopicList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


% --- Executes on button press in logout.
function logout_Callback(hObject, eventdata, handles)
% hObject    handle to logout (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in openBtn.
function openBtn_Callback(hObject, eventdata, handles)
% hObject    handle to openBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
topic = get(handles.topicPopupMenu,'value');
subTopic = get(handles.subTopicList,'value');
course = get(handles.coursePopupMenu,'value');

if
~isempty(handles.course_group(course).topic_group(topic).subTopic_gro
up(subTopic).function.Text)
    funcstr =
[handles.course_group(course).topic_group(topic).subTopic_group(subTo
pic).path '/'
handles.course_group(course).topic_group(topic).subTopic_group(subTop
ic).function.Text];
    eval_func =
str2func(handles.course_group(course).topic_group(topic).subTopic_gro
up(subTopic).function.Text(1:end-2));

addpath(handles.course_group(course).topic_group(topic).subTopic_grou
p(subTopic).path);
    eval_func();
end


% --- Executes during object creation, after setting all properties.
function snapshotAxes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to snapshotAxes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate snapshotAxes


% --- Executes on selection change in coursePopupMenu.
function coursePopupMenu_Callback(hObject, eventdata, handles)
% hObject    handle to coursePopupMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
topic = get(handles.topicPopupMenu,'value');
subTopic = get(handles.subTopicList,'value');
course = get(handles.coursePopupMenu,'value');

set(handles.topicPopupMenu,'value',1);
set(handles.subTopicList,'value',1);

populateTopics(handles,course)
```

```
populateSubTopics(hObject,handles,course,1,1)

set(handles.startText,'visible','off');

guidata(hObject, handles);
% Hints: contents = cellstr(get(hObject,'String')) returns
coursePopupMenu contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
coursePopupMenu


% --- Executes during object creation, after setting all properties.
function coursePopupMenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coursePopupMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function startText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to startText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

Appendix B: The xml file template

The xml file template.

Name of the xml structure: <main_gui>

The 3 inputs for the element <type> are: ‘course’, ‘topic’ and ‘subtopic’

The template:

```
<main_gui>
<type></type>
<course></course>
<topic></topic>
<subtopic></subtopic>
<function></function>
<description></description>
<snapshot></snapshot>
</main_gui>
```