Identification of mechatronic systems

Dual-tone multi-frequency (DTMF) – Initial report

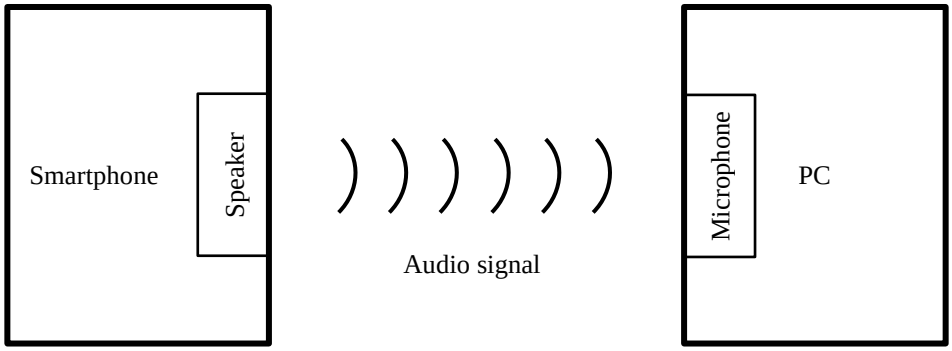
First and last name	Faculty	Field of study	Group
Jakub Kokosiński Michał Kubik	IMiR	IME	Thursday 11:30

Academic year	Semester	Date of execution
2023/24	VIII	04-06-2024

TABLE OF CONTENTS

1 INITIAL ASSUMPTIONS/SCOPE OF WORK

In the scope of the project, the objective is to develop MATLAB code that facilitates user interaction with PC actions using audio signals generated by a smartphone's numerical keypad. This involves capturing audio input from the smartphone's microphone, processing it to filter out noise, and analysing the signal to decode the DTMF frequency pairs corresponding to pressed buttons. The decoded digits or symbols will then be utilized to trigger various actions on the PC.



Basic system representation

Additionally, the software will feature a graphical user interface (GUI) allowing the user to start and stop the MATLAB script. The GUI will display real-time visualizations including:

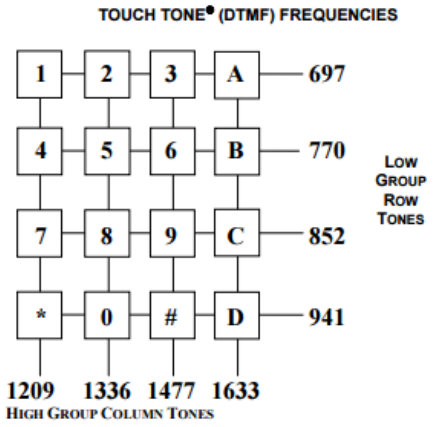
- Time characteristics of the input signal,
- Frequency characteristics of the filtered signal (Goertzel algorithm),
- Output signal representing the decoded DTMF symbols.

The additional goal of the project is to utilize these decoded DTMF symbols to control a video game, providing an interactive and engaging experience for the user.

2 DTMF OVERVIEW

DTMF, or Dual-Tone Multi-Frequency, is a signalling system used in telecommunication and various applications to encode digits and symbols onto the audio signal transmitted over phone lines. It employs a combination of two simultaneous frequencies to represent each symbol.

The DTMF system consists of a grid of frequencies, with one frequency from a high-frequency group and one from a low-frequency group, forming a unique pair for each digit (0-9) and some additional symbols (* and #). The high-frequency group typically consists of frequencies ranging from 1209 Hz to 1633 Hz, while the low-frequency group ranges from 697 Hz to 941 Hz. When a button on a telephone keypad is pressed, it generates a combination of two frequencies corresponding to that button's position on the grid. These frequency pairs are then decoded at the receiving end to interpret the pressed digit or symbol. DTMF decoding typically involves filtering and analysing the received audio signal to detect the presence of specific frequency combinations.



Source: sigidwiki.com, username: Cartoonman (https://www.sigidwiki.com/wiki/User:Cartoonman)

Applications of DTMF include telephone systems for dialling numbers, interactive voice response systems (IVR), automated customer service systems, remote control systems, and more.

3 MATLAB CODE DESCRIPTION

At the beginning of the main loop in our Matlab script, we record an audio sample using a previously initialized audio recorder object and defined recording parameters. The default values of these parameters are:

- sampling rate – fs – 16 kHz,
- time duration of one sample – recDuration – 0,39 sec,
- bits per sample – recNBits – 16,
- number of audio channels – recNChannels – 1 (mono).

```
recObj = audiorecorder(app.fs, app.recNBits, app.recNChannels);
%% Record audio
recordblocking(recObj, app.recDuration);
%% Set signal parameters
audio = getaudiodata(recObj);

dt = 1/app.fs;
s = app.recDuration/dt;
timeVector = 0:dt:app.recDuration-dt;
```

In the next step, we calculate the frequency response with Goertzel algorithm which involves recurrent signal processing and is expressed by the following equation:

$$s[n] = x[n] + 2\cos(\omega_k)\, s[n-1] - s[n-2];\ init\ values\ are{:}\ s[-1] = 0\ and\ s[-2] = 0$$

where:

- k – frequency index,
- n – signal sample index,
- N – signal length,
- $\omega_k = {}^{2\pi k}/_{N}$ – representing frequency k. in form required for Goertzel algorithm

After $s[n]$ is calculated for every n, parameters S_1 and S_2 can be determined by the following equations:

$$S_1 = s[N-1]$$
$$S_2 = s[N-2]$$

And finally, algorithm's output:

$$X[k] = S_1 - e^{-j\omega_k} S_2$$

where:

- $e^{-j\omega_k} = \cos(\omega_k) - jsin(\omega_k)$

Luckily it is not required to calculate all steps presented above because there is a predefined *gortzel()* function in Matlab. For the frequencies of our interest – *k* – which are DTMF-defined tones, the Matlab script used to determine signal's DFT may look like the one presented below.

```
dtmfFrequencies = [697 770 852 941 1209 1336 1477];
freqIndices = round(dtmfFrequencies/app.fs*s) + 1;
dftAudio = goertzel(audio, freqIndices);
dftAudio = abs(dftAudio');
```

After acquiring the amplitudes values for the given frequencies, we determine two frequencies with the highest amplitudes – one from the low tones group (697 Hz, 770 Hz, 852 Hz, 941 Hz) and one from the high tones group (1209 Hz, 1336 Hz, 1477 Hz).

```
[~, greatestLowFreqPosition] = max(dftAudio(1:4));
[~, greatestHighFreqPosition] = max(dftAudio(5:7));
greatestHighFreqPosition = greatestHighFreqPosition + 4;
```

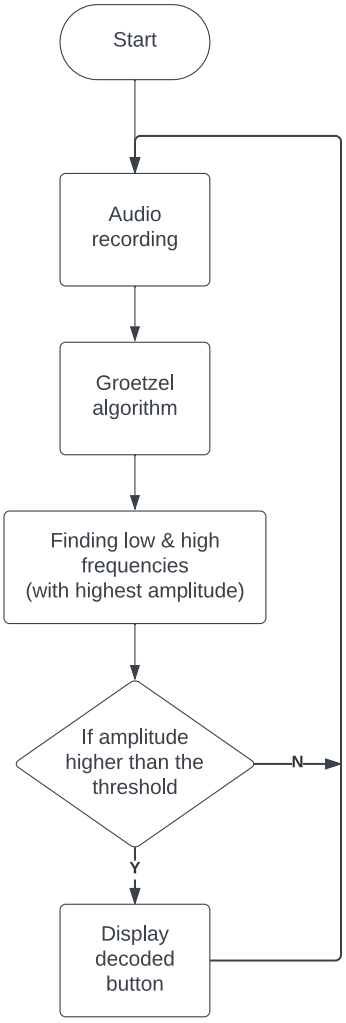
In the next step, these amplitudes are compared to defined threshold in `app.audioThreshold` variable.

```
freqAnswers = zeros(1, 7);
freqAnswers(greatestLowFreqPosition) = dftAudio(greatestLowFreqPosition) >=
app.audioThreshold;
freqAnswers(greatestHighFreqPosition) = dftAudio(greatestHighFreqPosition) >=
app.audioThreshold;
freqAnswers = double(freqAnswers);
```

Finally, two acquired frequencies are checked against values in the `freqPossibleAnswers` array to determine which button was pressed.

```
freqPossibleAnswers = [
%[697 770 852 941 1209 1336 1477]
  [1   0   0   0   1    0    0];    % 1
  [1   0   0   0   0    1    0];    % 2
  [1   0   0   0   0    0    1];    % 3
  [0   1   0   0   1    0    0];    % 4
  [0   1   0   0   0    1    0];    % 5
  [0   1   0   0   0    0    1];    % 6
  [0   0   1   0   1    0    0];    % 7
  [0   0   1   0   0    1    0];    % 8
  [0   0   1   0   0    0    1];    % 9
  [0   0   0   1   1    0    0];    % *
  [0   0   0   1   0    1    0];    % 0
  [0   0   0   1   0    0    1]];   % #
```

All steps of the described script are presented on the flowchart below.



Main loop flowchart

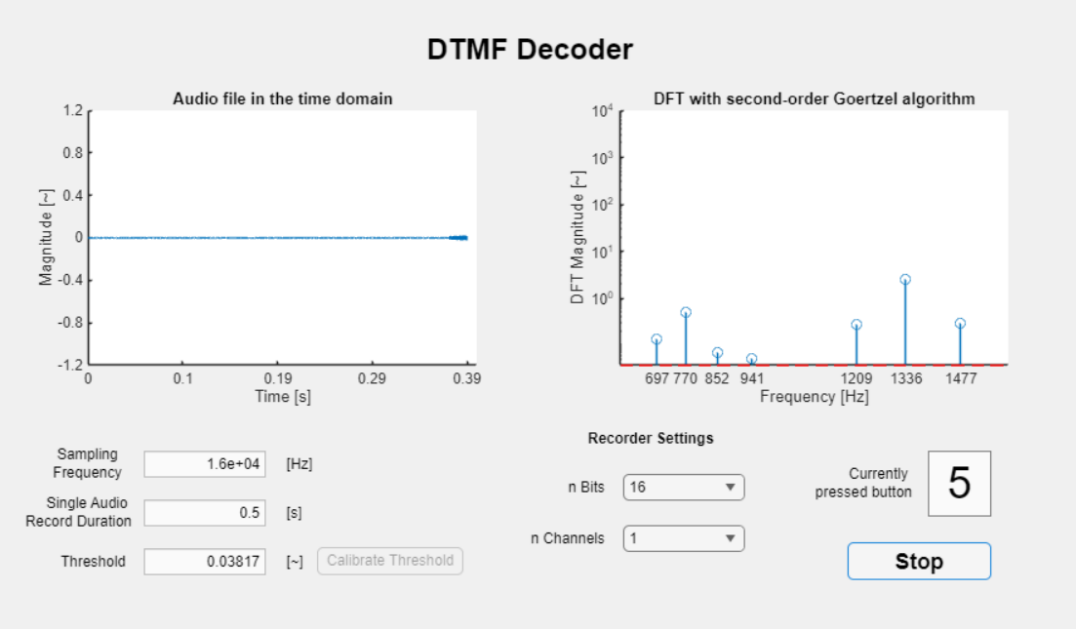
4 GRAPHICAL USER INTERFACE

For better visualisation of the DTMF concept and more enjoyable user experience, we have created a Graphical User Interface, using build-in Matlab addon called *AppDesigner*.
The GUI has been designed in a way to satisfy the base requirements of our project (chapter #1) which are as follows:

- Time characteristics of the input signal,
- Frequency characteristics of the filtered signal (Goertzel algorithm),
- Output signal representing the decoded DTMF symbols.

Additionally we added some quality of life features:

- Ability to change important code variables from GUI level,
- Threshold calibration button, that can automatically set the optimal threshold level for the current ambient noise.



Graphical User Interface

The main focus of this application was the ability to monitor and showcase the registered signals in real-time. The first plot shows us the recorded audio sample in the time domain.
The second plot is of the post-processed signal, after Goertzel algorithm with the main Touch-Tone DTMF frequencies (see Chapter #2) shown on the X axis. Another one of the main features of the app is the "Current pressed button" window, showing in real-time the current input decoded from the recorded audio sample.

As for the additional content, it can be seen below the two main plots. We have the ability to quickly change, in order:

- Sampling frequency
- Single Audio Record Duration
- Threshold
- Number of bits
- Number of channels

We also have the ability to automatically calibrate the best possible threshold in regards of the current ambient noise, by pressing the "Calibrate Threshold" button. To be able to change any of the previously mentioned variables, the app recording has to be stopped by pressing the START/STOP button.

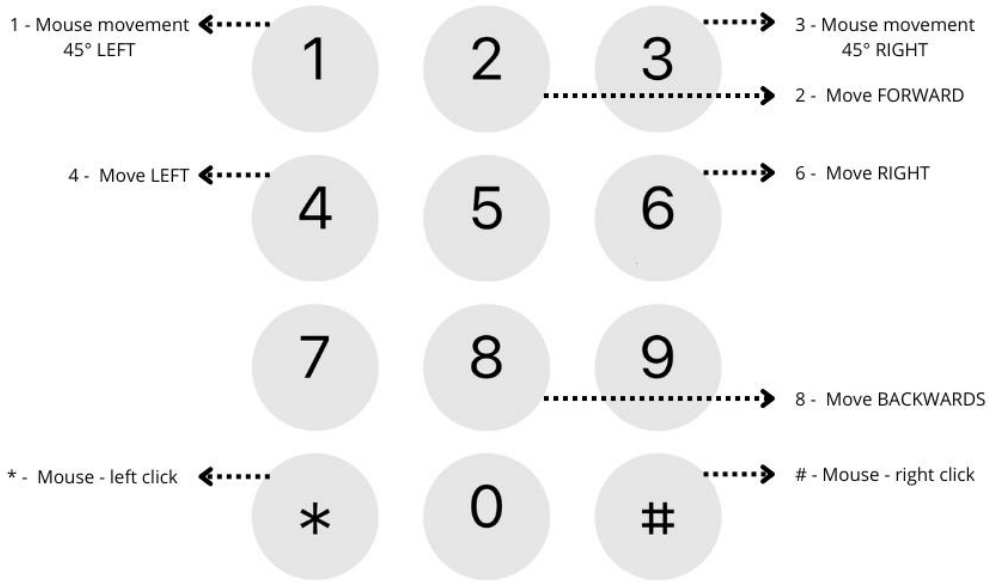
5 DTMF – GAME CONTROL

The main goal of our project was not only decoding DTMF inputs thru or Matlab application, but we wanted to use the obtained inputs to create a platform for wireless video game control.

By defining additional commands in the Matlab code, we are able to assign computer inputs for each decoded DTMF button instance. Standard phone DTMF keyboard has 12 buttons available, so we can prepare 12 unique input responses, which gives us some good options for controlling video games. For this particular project, our game of choice was a very popular game called "Minecraft", which we deemed an interesting test example of the abilities of our DTMF control system.

The game itself requires quite an advance input setup to be able to play it on even a basic level. Even though we are limited in our number of decoder DTMF buttons we managed to obtain a decent level of control without using all of the available buttons. The biggest obstacle in the definition of the game inputs was the lack of precise mouse movement which are essential in a first-person-view game such as Minecraft.

Below is a list of the used DTMF buttons and the assigned inputs for each one:



For the DTMF game control to work correctly both application: DTMF decoder and Minecraft, have to run simultaneously. After pressing the START button on the GUI we can click and start-up Minecraft, and after pressing the desired button on our phone we should see a reaction in game.

Basic setup of the Minecraft game and our DTMF Decoder GUI

6 REFERENCES

1. https://www.mathworks.com/help/signal/ref/goertzel.html
2. Giergiel J., Uhl T., Identyfikacja układów mechanicznych, PWN, Warszawa 1990
3. Uhl T., Kurowski P., Zastosowanie środowisk MATLAB Siglab do analizy sygnałów, CCATIE, Kraków 1998

7 MATLAB SCRIPT

All Matlab source files were uploaded to GitHub Repository: github.com/jako645/DTMF-in-Matlab.

```
classdef dtmfApp < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        CalibrateThresholdButton        matlab.ui.control.Button
        nChannelsDropDown               matlab.ui.control.DropDown
        nChannelsDropDownLabel          matlab.ui.control.Label
        nBitsDropDown                   matlab.ui.control.DropDown
        nBitsDropDownLabel              matlab.ui.control.Label
        DTMFDecoderLabel                matlab.ui.control.Label
        RecorderSettingsLabel           matlab.ui.control.Label
        DurationLabel                   matlab.ui.control.Label
        FsLabel                         matlab.ui.control.Label
        ThresholdLabel                  matlab.ui.control.Label
        SingleAudioRecordDurationEditField  matlab.ui.control.NumericEditField
        SingleAudioRecordDurationEditFieldLabel  matlab.ui.control.Label
        SamplingFrequencyEditField  matlab.ui.control.NumericEditField
        SamplingFrequencyEditFieldLabel  matlab.ui.control.Label
        StartButton                     matlab.ui.control.Button
        ThresholdEditField              matlab.ui.control.NumericEditField
        ThresholdEditFieldLabel         matlab.ui.control.Label
        CurrentlypressedbuttonEditField  matlab.ui.control.EditField
        CurrentlypressedbuttonEditFieldLabel  matlab.ui.control.Label
        UIAxesTime                      matlab.ui.control.UIAxes
        UIAxesDFT                       matlab.ui.control.UIAxes
    end

    properties (Access = private)
        recDuration = 0.39;          % for recording object
        fs = 16000;
        recNBits = 16;
        recNChannels = 1;

        audioThreshold = 0.5;        % for signal processing

        recordingEnabled = false;   % for stopping dtmf's while loop
        pressedButton = 'n';
    end

    methods (Access = private)

    end
```

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

        % Value changed function: ThresholdEditField
        function ThresholdEditFieldValueChanged(app, event)
            app.audioThreshold = app.ThresholdEditField.Value;
        end

        % Button pushed function: StartButton
        function StartButtonPushed(app, event)
            import java.awt.Robot;
            import java.awt.event.*;
            robot = Robot();
            robot.delay(2000);

            if ~app.recordingEnabled
                app.recordingEnabled = true;
                app.CurrentlypressedbuttonEditField.Value = "";
                app.StartButton.Text = "Stop";

                % Turn off other components
                app.ThresholdEditField.Editable = "off";
                app.SamplingFrequencyEditField.Editable = "off";
                app.SingleAudioRecordDurationEditField.Editable = "off";
                app.nBitsDropDown.Editable = "off";
                app.nChannelsDropDown.Editable = "off";
                app.CalibrateThresholdButton.Enable = "off";

                recObj = audiorecorder(app.fs, app.recNBits, app.recNChannels);

                while app.recordingEnabled
                    %% Record audio
                    recordblocking(recObj, app.recDuration);
                    %% Set signal parameters
                    audio = getaudiodata(recObj);

                    dt = 1/app.fs;
                    s = app.recDuration/dt;
                    timeVector = 0:dt:app.recDuration-dt;

                    %% Apply Groetzel algorithm
                    dtmfFrequencies = [697 770 852 941 1209 1336 1477];
                    freqIndices = round(dtmfFrequencies/app.fs*s) + 1;
                    dftAudio = goertzel(audio, freqIndices);
                    dftAudio = abs(dftAudio');

                    %% Get two frequencies with the greatest magnitude
                    [~, greatestLowFreqPosition] = max(dftAudio(1:4));
                    [~, greatestHighFreqPosition] = max(dftAudio(5:7));
                    greatestHighFreqPosition = greatestHighFreqPosition + 4;

                    %% Check threshold
                    freqAnswers = zeros(1, 7);
                    freqAnswers(greatestLowFreqPosition) =
dftAudio(greatestLowFreqPosition) >= app.audioThreshold;
                    freqAnswers(greatestHighFreqPosition) =
dftAudio(greatestHighFreqPosition) >= app.audioThreshold;
```

```
                    freqAnswers = double(freqAnswers);

                    %% Get output
                    freqPossibleAnswers = [
                       %[697 770 852 941 1209 1336 1477]
                        [1   0   0   0   1    0    0];    % 1
                        [1   0   0   0   0    1    0];    % 2
                        [1   0   0   0   0    0    1];    % 3
                        [0   1   0   0   1    0    0];    % 4
                        [0   1   0   0   0    1    0];    % 5
                        [0   1   0   0   0    0    1];    % 6
                        [0   0   1   0   1    0    0];    % 7
                        [0   0   1   0   0    1    0];    % 8
                        [0   0   1   0   0    0    1];    % 9
                        [0   0   0   1   1    0    0];    % *
                        [0   0   0   1   0    1    0];    % 0
                        [0   0   0   1   0    0    1]];   % #

                    % Pause variable
                    pause_var = 1.5;

                    % freqAnswers has two '1' - button was pressed
                    if freqAnswers == freqPossibleAnswers(1, 1:end)
                        app.pressedButton = '1';
                        robot.mouseMove(300, 540);
                    elseif freqAnswers == freqPossibleAnswers(2, 1:end)
                        app.pressedButton = '2';
                        robot.keyPress(KeyEvent.VK_W);
                        pause(pause_var);
                        robot.keyRelease(KeyEvent.VK_W);
                    elseif freqAnswers == freqPossibleAnswers(3, 1:end)
                        app.pressedButton = '3';
                        robot.mouseMove(1800, 540);
                    elseif freqAnswers == freqPossibleAnswers(4, 1:end)
                        app.pressedButton = '4';
                        robot.keyPress(KeyEvent.VK_A);
                        pause(pause_var);
                        robot.keyRelease(KeyEvent.VK_A);
                    elseif freqAnswers == freqPossibleAnswers(5, 1:end)
                        app.pressedButton = '5';
                    elseif freqAnswers == freqPossibleAnswers(6, 1:end)
                        app.pressedButton = '6';
                        robot.keyPress(KeyEvent.VK_D);
                        pause(pause_var);
                        robot.keyRelease(KeyEvent.VK_D);
                    elseif freqAnswers == freqPossibleAnswers(7, 1:end)
                        app.pressedButton = '7';
                    elseif freqAnswers == freqPossibleAnswers(8, 1:end)
                        app.pressedButton = '8';
                        robot.keyPress(KeyEvent.VK_S);
                        pause(pause_var);
                        robot.keyRelease(KeyEvent.VK_S);
                    elseif freqAnswers == freqPossibleAnswers(9, 1:end)
                        app.pressedButton = '9';
                    elseif freqAnswers == freqPossibleAnswers(10, 1:end)
                        app.pressedButton = '*';

robot.mousePress(java.awt.event.InputEvent.BUTTON1_DOWN_MASK);
                        pause(1);
```

```
robot.mouseRelease(java.awt.event.InputEvent.BUTTON1_DOWN_MASK);
                    elseif freqAnswers == freqPossibleAnswers(11, 1:end)
                        app.pressedButton = '0';
                    elseif freqAnswers == freqPossibleAnswers(12, 1:end)
                        app.pressedButton = '#';

robot.mousePress(java.awt.event.InputEvent.BUTTON3_DOWN_MASK);
                        pause(0.5);

robot.mouseRelease(java.awt.event.InputEvent.BUTTON3_DOWN_MASK);
                    else
                        app.pressedButton = 'n'; % none button was pressed
                    end

                    %% Display answer
                    % Time plot
                    maxScaleValue = 1.2;
                    app.UIAxesTime.XTick = linspace(0, app.recDuration, 5);
                    app.UIAxesTime.YTick = linspace(-maxScaleValue,
maxScaleValue, 7);
                    app.UIAxesTime.YLim = [-maxScaleValue maxScaleValue];
                    plot(app.UIAxesTime, timeVector, audio)

                    % FFT plot
                    app.UIAxesDFT.XTick = dtmfFrequencies;
                    app.UIAxesDFT.XLim = [600 1600];
                    app.UIAxesDFT.YTick = [1, 10, 100, 1000, 10000];
                    app.UIAxesDFT.YLim = [0 10000];
                    stem(app.UIAxesDFT, dtmfFrequencies, abs(dftAudio))
                    hold(app.UIAxesDFT, 'on');
                    plot(app.UIAxesDFT, [0 2000], [app.audioThreshold
app.audioThreshold], 'r--')
                    hold(app.UIAxesDFT, 'off');

                    % Displaying char
                    if app.pressedButton ~= 'n'
                        app.CurrentlypressedbuttonEditField.Value =
app.pressedButton;
                    else
                        app.CurrentlypressedbuttonEditField.Value = '-';
                    end
                    pause(0.01)
                end
            else
                app.recordingEnabled = false;
                app.StartButton.Text = "Start";

                % Turn on other components
                app.ThresholdEditField.Editable = "on";
                app.SamplingFrequencyEditField.Editable = "on";
                app.SingleAudioRecordDurationEditField.Editable = "on";
                app.nBitsDropDown.Editable = "on";
                app.nChannelsDropDown.Editable = "on";
                app.CalibrateThresholdButton.Enable = "on";
            end
        end

        % Value changed function: SamplingFrequencyEditField
```

```
        function SamplingFrequencyEditFieldValueChanged(app, event)
            app.fs = app.SamplingFrequencyEditField.Value;
        end

        % Value changed function: SingleAudioRecordDurationEditField
        function SingleAudioRecordDurationEditFieldValueChanged(app, event)
            app.recDuration = app.SingleAudioRecordDurationEditField.Value;
        end

        % Value changed function: nChannelsDropDown
        function nChannelsDropDownValueChanged(app, event)
            app.recNChannels = app.nChannelsDropDown.Value;
        end

        % Value changed function: nBitsDropDown
        function nBitsDropDownValueChanged(app, event)
            app.recNBits = app.nBitsDropDown.Value;
        end

        % Button pushed function: CalibrateThresholdButton
        function CalibrateThresholdButtonPushed(app, event)
            app.CalibrateThresholdButton.Text = "Wait";
            app.StartButton.Enable = "off";

            %% Record audio
            recObj = audiorecorder(app.fs, app.recNBits, app.recNChannels);
            recordingDuration = 2.5;
            recordblocking(recObj, recordingDuration);

            %%  Set required parameters
            audio = getaudiodata(recObj);
            dt = 1/app.fs;
            s = recordingDuration/dt;

            %% Apply Groetzel algorithm
            dtmfFrequencies = [697 770 852 941 1209 1336 1477];
            freqIndices = round(dtmfFrequencies/app.fs*s) + 1;
            dftAudio = goertzel(audio, freqIndices);
            dftAudio = abs(dftAudio');

            %% Get max amplitude
            maxMagnitude = max(dftAudio);

            %% Set new threshold
            safeSpaceFactor = 1.1;  % [*100%]
            app.ThresholdEditField.Value = safeSpaceFactor * maxMagnitude;
            app.audioThreshold = safeSpaceFactor * maxMagnitude;

            %% Clean up
            recObj.delete;
            app.CalibrateThresholdButton.Text = "Calibrate Threshold";
            app.StartButton.Enable = "on";
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
```

```
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 892 527];
            app.UIFigure.Name = 'MATLAB App';

            % Create UIAxesDFT
            app.UIAxesDFT = uiaxes(app.UIFigure);
            title(app.UIAxesDFT, 'DFT with second-order Goertzel algorithm')
            xlabel(app.UIAxesDFT, 'Frequency [Hz]')
            ylabel(app.UIAxesDFT, 'DFT Magnitude [~]')
            zlabel(app.UIAxesDFT, 'Z')
            app.UIAxesDFT.YLim = [0 1];
            app.UIAxesDFT.XTick = [];
            app.UIAxesDFT.YTick = [];
            app.UIAxesDFT.YScale = 'log';
            app.UIAxesDFT.YMinorTick = 'on';
            app.UIAxesDFT.ZTick = [];
            app.UIAxesDFT.Position = [471 193 366 260];

            % Create UIAxesTime
            app.UIAxesTime = uiaxes(app.UIFigure);
            title(app.UIAxesTime, 'Audio file in the time domain')
            xlabel(app.UIAxesTime, 'Time [s]')
            ylabel(app.UIAxesTime, 'Magnitude [~]')
            zlabel(app.UIAxesTime, 'Z')
            app.UIAxesTime.XTick = [];
            app.UIAxesTime.YTick = [];
            app.UIAxesTime.ZTick = [];
            app.UIAxesTime.Position = [35 193 366 260];

            % Create CurrentlypressedbuttonEditFieldLabel
            app.CurrentlypressedbuttonEditFieldLabel = uilabel(app.UIFigure);
            app.CurrentlypressedbuttonEditFieldLabel.HorizontalAlignment =
'right';
            app.CurrentlypressedbuttonEditFieldLabel.WordWrap = 'on';
            app.CurrentlypressedbuttonEditFieldLabel.Position = [664 102 89 54];
            app.CurrentlypressedbuttonEditFieldLabel.Text = 'Currently pressed
button';

            % Create CurrentlypressedbuttonEditField
            app.CurrentlypressedbuttonEditField = uieditfield(app.UIFigure,
'text');
            app.CurrentlypressedbuttonEditField.Editable = 'off';
            app.CurrentlypressedbuttonEditField.HorizontalAlignment = 'center';
            app.CurrentlypressedbuttonEditField.FontSize = 36;
            app.CurrentlypressedbuttonEditField.Position = [766 102 52 54];

            % Create ThresholdEditFieldLabel
            app.ThresholdEditFieldLabel = uilabel(app.UIFigure);
            app.ThresholdEditFieldLabel.HorizontalAlignment = 'right';
            app.ThresholdEditFieldLabel.Position = [50 54 58 22];
            app.ThresholdEditFieldLabel.Text = 'Threshold';

            % Create ThresholdEditField
            app.ThresholdEditField = uieditfield(app.UIFigure, 'numeric');
            app.ThresholdEditField.Limits = [0 10000];
```

```
            app.ThresholdEditField.ValueChangedFcn = createCallbackFcn(app,
@ThresholdEditFieldValueChanged, true);
            app.ThresholdEditField.Position = [123 54 100 22];
            app.ThresholdEditField.Value = 0.5;

            % Create StartButton
            app.StartButton = uibutton(app.UIFigure, 'push');
            app.StartButton.ButtonPushedFcn = createCallbackFcn(app,
@StartButtonPushed, true);
            app.StartButton.FontSize = 18;
            app.StartButton.FontWeight = 'bold';
            app.StartButton.Position = [700 50 118 31];
            app.StartButton.Text = 'Start';

            % Create SamplingFrequencyEditFieldLabel
            app.SamplingFrequencyEditFieldLabel = uilabel(app.UIFigure);
            app.SamplingFrequencyEditFieldLabel.HorizontalAlignment = 'right';
            app.SamplingFrequencyEditFieldLabel.WordWrap = 'on';
            app.SamplingFrequencyEditFieldLabel.Position = [22 124 83 43];
            app.SamplingFrequencyEditFieldLabel.Text = 'Sampling Frequency';

            % Create SamplingFrequencyEditField
            app.SamplingFrequencyEditField = uieditfield(app.UIFigure,
'numeric');
            app.SamplingFrequencyEditField.Limits = [0 100000];
            app.SamplingFrequencyEditField.ValueChangedFcn =
createCallbackFcn(app, @SamplingFrequencyEditFieldValueChanged, true);
            app.SamplingFrequencyEditField.Position = [123 134 100 22];
            app.SamplingFrequencyEditField.Value = 16000;

            % Create SingleAudioRecordDurationEditFieldLabel
            app.SingleAudioRecordDurationEditFieldLabel = uilabel(app.UIFigure);
            app.SingleAudioRecordDurationEditFieldLabel.HorizontalAlignment =
'right';
            app.SingleAudioRecordDurationEditFieldLabel.WordWrap = 'on';
            app.SingleAudioRecordDurationEditFieldLabel.Position = [22 89 91
33];
            app.SingleAudioRecordDurationEditFieldLabel.Text = 'Single Audio
Record Duration';

            % Create SingleAudioRecordDurationEditField
            app.SingleAudioRecordDurationEditField = uieditfield(app.UIFigure,
'numeric');
            app.SingleAudioRecordDurationEditField.Limits = [0 1.5];
            app.SingleAudioRecordDurationEditField.ValueChangedFcn =
createCallbackFcn(app, @SingleAudioRecordDurationEditFieldValueChanged, true);
            app.SingleAudioRecordDurationEditField.Position = [123 94 100 22];
            app.SingleAudioRecordDurationEditField.Value = 0.5;

            % Create ThresholdLabel
            app.ThresholdLabel = uilabel(app.UIFigure);
            app.ThresholdLabel.Position = [240 54 25 22];
            app.ThresholdLabel.Text = '[~]';

            % Create FsLabel
            app.FsLabel = uilabel(app.UIFigure);
            app.FsLabel.Position = [240 134 26 22];
            app.FsLabel.Text = '[Hz]';
```

```
            % Create DurationLabel
            app.DurationLabel = uilabel(app.UIFigure);
            app.DurationLabel.Position = [240 94 25 22];
            app.DurationLabel.Text = '[s]';

            % Create RecorderSettingsLabel
            app.RecorderSettingsLabel = uilabel(app.UIFigure);
            app.RecorderSettingsLabel.FontWeight = 'bold';
            app.RecorderSettingsLabel.Position = [487 155 108 22];
            app.RecorderSettingsLabel.Text = 'Recorder Settings';

            % Create DTMFDecoderLabel
            app.DTMFDecoderLabel = uilabel(app.UIFigure);
            app.DTMFDecoderLabel.FontSize = 24;
            app.DTMFDecoderLabel.FontWeight = 'bold';
            app.DTMFDecoderLabel.Position = [355 470 174 32];
            app.DTMFDecoderLabel.Text = 'DTMF Decoder';

            % Create nBitsDropDownLabel
            app.nBitsDropDownLabel = uilabel(app.UIFigure);
            app.nBitsDropDownLabel.HorizontalAlignment = 'right';
            app.nBitsDropDownLabel.Position = [466 115 35 22];
            app.nBitsDropDownLabel.Text = 'n Bits';

            % Create nBitsDropDown
            app.nBitsDropDown = uidropdown(app.UIFigure);
            app.nBitsDropDown.Items = {'16', '8', '24'};
            app.nBitsDropDown.ValueChangedFcn = createCallbackFcn(app,
@nBitsDropDownValueChanged, true);
            app.nBitsDropDown.Position = [516 115 100 22];
            app.nBitsDropDown.Value = '16';

            % Create nChannelsDropDownLabel
            app.nChannelsDropDownLabel = uilabel(app.UIFigure);
            app.nChannelsDropDownLabel.HorizontalAlignment = 'right';
            app.nChannelsDropDownLabel.Position = [435 73 66 22];
            app.nChannelsDropDownLabel.Text = 'n Channels';

            % Create nChannelsDropDown
            app.nChannelsDropDown = uidropdown(app.UIFigure);
            app.nChannelsDropDown.Items = {'1', '2'};
            app.nChannelsDropDown.ValueChangedFcn = createCallbackFcn(app,
@nChannelsDropDownValueChanged, true);
            app.nChannelsDropDown.Position = [516 73 100 22];
            app.nChannelsDropDown.Value = '1';

            % Create CalibrateThresholdButton
            app.CalibrateThresholdButton = uibutton(app.UIFigure, 'push');
            app.CalibrateThresholdButton.ButtonPushedFcn =
createCallbackFcn(app, @CalibrateThresholdButtonPushed, true);
            app.CalibrateThresholdButton.Position = [265 54 120 23];
            app.CalibrateThresholdButton.Text = 'Calibrate Threshold';

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

    % App creation and deletion
```

```
    methods (Access = public)

        % Construct app
        function app = dtmfApp

            % Create UIFigure and components
            createComponents(app)

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

            if nargout == 0
                clear app
            end
        end

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

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