ArtUs RF Data Control implemented in Python, MATLAB and LabVIEW

User Manual

TELEMED

Ultrasound Medical Systems

Internet page: https://www.pcultrasound.com/
Information E-mail: info@pcultrasound.com
Support E-mail: support@pcultrasound.com

Contents

Introduction

User Manual describes dynamic link library (DLL) which allows to call TELEMED SDK [2] functions from other programming languages and 3 sample programs for experiments. The DLL functions retrieves B mode image data and beamformed RF Data. There are three sample programs provided prepared in Python, MATLAB and LabVIEW which in principle are copies of ArtUs RF Data Control [1] program and allows similar functionality: 1) to adjust ultrasound B mode imaging parameters such as transmit focus, frequency, imaging depth, gain of received signal etc, 2) to adjust beamformed RF data stream window in B mode image frame, to select RF Data source point and to 3) record scanned RF Data from Cine loop to bin files as it is possible by using ArtUs RF Data control program. The goal of this package to provide possibilities for researchers to work in platform which is convenient (MATLAB, Python, LabVIEW or other which is capable to call C++ DLL library which links to TELEMED SDK) and to receive image data and beamformed RF Data in real-time. The package contains C++ DLL wrapper and 3 sample programs prepared for Python, MATLAB and LabVIEW.

1. DLL usgfw2wrapper.dll purpose and description of the library functions

DLL usgfw2wrapper.dll is a C++ program which operates as a wrapper between TELEMED SDK usgfw2 library [2] written in C++ and other programming platforms which are capable to call DLLs, such as MATLAB, Python or LabView. For the explicit knowledge on TELEMED's SDK we invite you to read [2].

The DLL library allows to initialize ultrasound scanner and to acquire ultrasound image and RF data in real-time. Majority of ultrasound imaging controls such as scanning frequency selection, imaging depth, gain could be implemented by using the library. Currently the library contains following functions, which could be called from external software:

Function	Description	Input/Output
on_init()	Function initializes variables, objects and handles used in further calls.	None/None (Void)
init_ultrasound_usgfw2()	Function creates main Usgfw2 library object.	None/returns int32, value 3 in case of success for error handling
find_connected_probe()	Function detects connected ultrasound probe.	None/return int32, value 101 in case of success (if probe detected)
data_view_function()	Function creates main ultrasound scanning object for detected probe.	None/return int32, value 100 in case of success (if probe detected)
mixer_control_function(int32, int32, int32, int32, int32, int32, int32)	Function creates B mixer control. Image width and height are passed into a function by first 4 parameters.	Input parameters are: int32 – left coordinate of imaging rectangle (i.e. 0) int32 – top coordinate of imaging rectangle (i.e. 0) int32 – right coordinate of imaging rectangle (i.e. 512) int32 – bottom coordinate of imaging rectangle (i.e. 512) int32 – R int32 – G int32 – B – RGB values for image background, (0,0,0) - black Output Return int32, value 89 in case of success
get_resolution(singlePtr, singlePtr)	Function returns pixel size (resolution) in x and y dimensions in mm for imaging axis.	Input: Pointer to float value (resolution X) Pointer to float value (resolution Y)
return_pixel_values(uint32Ptr)	Function returns buffer with pixel values of current frame in BGRA order, for further	Input: Pointer to initialized uint32 array of size

	imaging purposes array must be reshaped.	(w*h*4), where w – image width, h – image height
Run_ultrasound_scanning()	Function to run scanning	None/None (Void)
Freeze_ultrasound_scanning()	Function to freeze ultrasound scanning	None/None (Void)
Stop_ultrasound_scanning()	Function to stop ultrasound scanning	None/None (Void)
Close_and_release()	Function releases all initialized variables, objects, controls and variables.	None/None (Void)
set_callback_Sample_Grabber()	Function to set call back for retrieving RF Data.	None/None (Void)
set_callback_scan_converter()	Function to set call back for retrieving B mode image data.	None/None (Void)
on_probe_button_pressed(int32, int32, int32, int32, int32, int32, int32)	Function releases and reinitializes ultrasonography controls and recreates main ultrasound scanning object (data view) for detected probe.	Input: int32 – left coordinate of imaging rectangle (i.e. 0) int32 – top coordinate of imaging rectangle (i.e. 0) int32 – right coordinate of imaging rectangle (i.e. 512) int32 – bottom coordinate of imaging rectangle (i.e. 512) int32 – R int32 – G int32 – B – RGB values for image background, (0,0,0) – black.
UsgQualProp_control()	Function creates ultrasonography quality properties control	None/None (Void)
B_dynamic_range()	Function creates Dynamic range control for B mode	None/None (Void)
B_DynamicRangeSetPrevNext(int32, longPtr)	Function changes dynamic range value by one-step to desired direction.	Input: int32 direction and step (i.e. -1, +1) longPtr – pointer to actual dynamic range value
frequency_control()	Function creates scanning frequency control.	None/None (Void)
B_FrequencySetPrevNext(int32, int32Ptr, longPtr)	Function changes B mode frequency value by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), int32Ptr – pointer to actual frequency value in Hz, longPtr – pointer to actual frequency number from available list of frequencies.
gain_control()	Function creates B mode gain control.	None/None (Void)

B_GainSetByIdx(int32, longPtr)	Function changes B mode gain to a desired value.	Input: int32 – desired gain value, longPtr – pointer to actual gain value.
focus_control()	Function creates B mode transmit focus control.	None/None (Void)
B_FocusSetPrevNext(int32, longPtr, longPtr, longPtr)	Function changes B mode transmit focal depth.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual focal depth value, longPtr – pointer to value of number of possible focal zones, longPtr – pointer to index of active focal zone.
lines_density()	Function creates B mode lines density control.	None/None (Void)
B_LinesDensitySetPrevNext(int32, longPtr)	Function changes B mode lines density value by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual Lines Density value (8 – Low, 16 – Medium, 22 – Standard S, 24 – Standard, 32 – High)
power_control()	Function creates B mode power control.	None/None (Void)
B_PowerSetByIdx(int32, longPtr)	Function changes B mode power to a desired value.	Input: int32 – power value (range 0…20), longPtr – returns actual power value in dB.
int32 steering_angle()	Function creates B mode steering angle control if available for selected probe.	Output: int32 – returns 1 if control was created (linear probe), or zero otherwise.
B_SteeringAngleSetPrevNext(int32, longPtr)	Function changes B mode steering angle value by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual steering angle value in degrees.
view_area()	Function creates B mode view area control.	None/None (Void)
B_view_areaSetPrevNext(int32, longPtr)	Function changes View Area value by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual view area value.
TGC_control()	Function creates B mode TGC control.	None/None (Void)
adjust_TGC(int32, int32, longPtr)	Function changes TGC value at the selected depth to a desired value.	Input: int32 – number of TGC slider,

		int32 – amplification value in percent for selected slider, longPtr – pointer to a depth value which corresponds to selected slider.
image_orientation()	Function creates image orientation control.	None/None (Void)
ChangeScanDirection(int32)	Function changes scan direction.	Input: int32 – passed 0 or 1, 1 – if to change scan direction and 0 – if to keep the same.
depth_control()	Function creates imaging depth control.	None/None (Void)
DepthSetPrevNext(int32, longPtr)	Function changes imaging depth value by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual depth value.
wide_view_angle()	Function creates angle control for wide view scanning mode.	None/None (Void)
WideViewAngleSetPrevNext(int32, longPtr)	Function changes wide view angle value by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual wide view angle value.
compound_angle()	Function creates compound mode angle control.	None/None (Void)
CompoundAngleSetPrevNext(int32, longPtr)	Function changes compound angle value by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual compound angle value.
compound_frames_number()	Function creates compound mode frames number control.	None/None (Void)
CompoundFramesSetPrevNext(int32, longPtr)	Function changes compound frames number value by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual compound frames number value.
CompoundSubframeSetPrevNext(int32, int32Ptr)	Function changes compound subframe index by one-step to desired direction.	Input: int32 – direction and step (i.e. -1, +1), longPtr – pointer to actual compound subframe index.
RF_data_controls()	Function creates controls for RF data handling (source point, RF data window etc.).	None/None (Void)
RF_Data_source_ID(int32)	Function sets RF Data Source Point ID for desired RF Data stream.	Input: int32 – ID of desired RF Data Stream (1 – Beamformer output, 2 –

		TFC filter output, 3 – Angular apodization output, 4 – Hilbert filter output).
RF_WindowMove(int32, int32)	Function adjusts RF Data window position by a predefined step.	Input: int32 – dx value (step to move to x direction) int32 – dy value (step to move to y direction)
RF_WindowSize(int32, int32)	Function adjusts RF Data window size by a predefined step.	Input: int32 – dx value (size to adjust to x direction) int32 – dy value (size to adjust to y direction)
RF_lines_get(longPtr, longPtr)	Function returns number of scanning lines for beginning and end of RF window.	Input: longPtr – pointer to Line1 (start window) value, longPtr – pointer to Line2 (end window) – value.
RF_window_get(longPtr, longPtr, longPtr, longPtr, int32Ptr)	Function returns through pointers the position of current RF Window.	Input: longPtr – pointer to RF window start depth value, longPtr – pointer to RF window end depth value, longPtr – pointer to RF window start line index, longPtr – pointer to RF window end line index, int32Ptr – pointer to number of lines in a window value.
RF_window_set(long, long, long, long)	Function sets the position of RF Window.	Input: long – RF Window start depth value, long – RF Window end depth value, long – RF Window start line value, long – RF Window end line value.
cine_loop_controls()	Function creates cine loop controls.	None/None (Void)
get_cine_frame(int32)	Function retrieves frame from cine buffer with a specified index.	Input: int32 – index of frame to retrieve from cine buffer.
get_cine_interval(int64Ptr, int64Ptr, int64Ptr)	Function returns indexes of available frames of actual cine interval.	Input: int64Ptr – pointer to actual position index, int64Ptr – start index of actual cine interval, int64Ptr – end index of actual cine interval.
scan_type_control()	Function creates scan type (standard, wide view, compound) selection control.	None/None (Void)

turn_on_scan_type(int32)	Function to select active type of scanning.	Input: int32 – parameter to set type of scanning (0 – standard, 1 – wide view, 2 - compound).
get_sampling_period(int32Ptr)	Function returns sampling period value in ns.	Input: int32Ptr – pointer to sampling period value in ns.
long get_probe_code()	Function returns code of active probe listed in the registry.	Output: long – returns probe code.
return_RF_data(int16Ptr, int32, int32Ptr, doublePtr, int32Ptr, int32Ptr)	Function retrieves RF data of current frame from RF Data Window.	Input: int16Ptr – pointer to array for RF Data receiving, int32 – size of actual RF data frame to retrieve in elements (number of scanlines in window X number of samples per scan line), int32Ptr – pointer to RF data frame number, doublePtr – pointer to time stamp value for each RF data frame, int32Ptr – pointer to RF lines number in actual data frame, int32Ptr – pointer to samples number for each scanline in actual RF frame.
return_pixel_values2(uint32Ptr, int32Ptr, doublePtr)	Function returns B mode image pixel values buffer (in contrary to function return_pixel_values there is no possibility to receive duplicate frames, only some can be missing).	Input: uint32Ptr – pointer to image data array (w × h × 4 size), int32Ptr – pointer to actual frame number, doublePtr – pointer to time stamp value of actual frame.
start_write_RF_to_bin_file(int32Ptr, int32, int32)	Function starts writing RF Data to bin file from cine loop memory (option from memory to disc available only for the DLL).	Input: int32Ptr – pointer to path and filename of recorded file converted into int32 data type, int32 – length of path and filename in symbols of int32 type, int32 – number of frames to record.
stop_write_RF_to_bin_file()	Function stops writing RF Data to bin file.	None/None (Void)
convert_units_to_pixels(int32, int32, int32, int32, singlePtr, singlePtr, singlePtr, singlePtr)	Function converts scanlines and depth values to coordinates in pixels in image	Input: int32 – actual scanline index, int32 – subframe index,

	frame (function is actual for RF Window plotting).	int32 – start depth value in mm, int32 – end depth value in mm, singlePtr – pointer to X1 value in pixels, singlePtr – pointer to Y1 value in pixels, singlePtr – pointer to X2 value in pixels singlePtr – pointer to Y2 value in pixels.

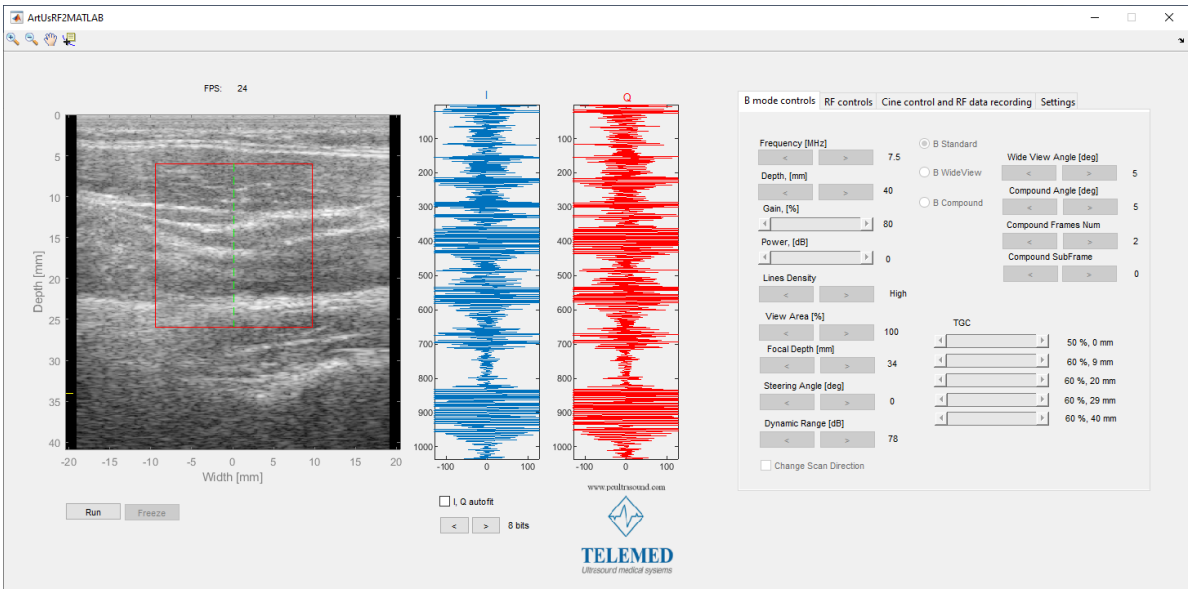
Please note: the function ***return_pixel_values*** calls SDK function ***GetCurrentBitmap*** and that means that there is a possibility of missing or duplicate frames which were originally obtained during scanning if imaging speed in external calling program do not matches scanning speed of ultrasound scanner.

2. MATLAB, Python and LabView sample programs which implements ArtUs RF Data Control functions

The chapter presents description of sample programs which has similar functionality as ArtUs RF Data Control program [1] just prepared for MATLAB (**ArtUsRF2MATLAB**), Python (**Python ArtUs RF Data Control**) and LabVIEW (**ArtUsRF2labview**) users. These sample programs have similar structure of programming – calls TELEMED SDK functions through DLL, which contents were described in previous chapter.

2.1. ArtUsRF2MATLAB sample program

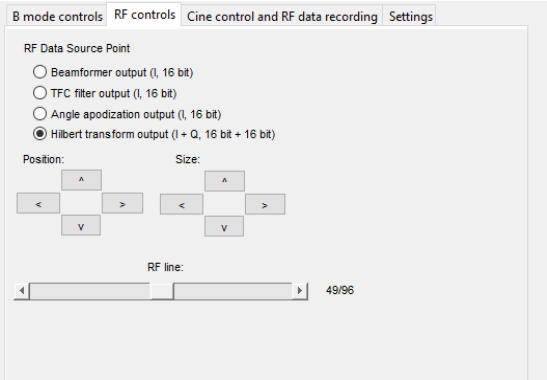
ArtUsRF2MATLAB sample program has similar functionality to ArtUs RF Data Control program written in C++ [1]. It is possible to control scanning parameters, to adjust RF Data window, review data stored in cine loop memory and to record RF Data to bin files from cine loop (structure of the data files is exactly the same as described [1]). The front panel of GUI is shown below. The GUI has B mode imaging window and RF Data plotting windows, FPS indicator and controls arranged in tabs. There are 4 tabs in total: 1) B mode controls, 2) RF controls, 3) Cine control and RF data recording, 4) Settings.



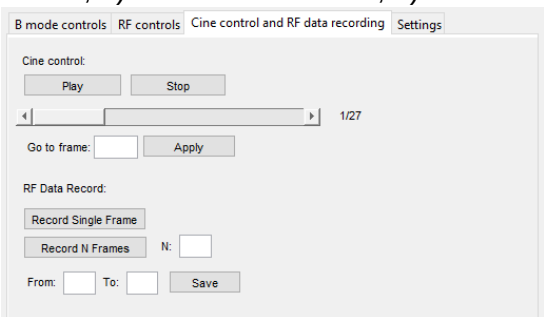
Tab **B mode controls** is shown below:

The controls here allow to adjust B mode imaging parameters such as 1) scanning **frequency**, 2) imaging **depth**, 3) **gain** of received signal, 4) acoustic **power**, 5) **lines density**, 6) **view area**, 7) **focal depth**, 8) **steering angle**, 9) **dynamic range**, 10) Time gain control (**TGC**) and 11) scan type selection (**standard**, **wide view** and **compound**) with corresponding controls.

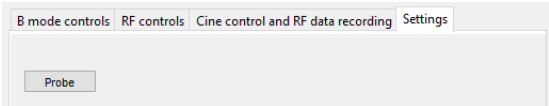
Second tab **RF controls** (below) contains RF Data related parameters: 1) **RF Data Source Point** (Beamformer output, TFC filter output, Angle apodization output and Hilbert transform output), 2) RF Data Window controls for adjustment of and **Position** and **Size** and 3) **RF line** which is displayed on screen selection.



Third tab **Cine control and RF data recording** contains controls for revision of scanned data and saving RF data from cine loop memory to BIN files. Note, that ArtUs RF Data Control program [1] has an option to record RF Data to disc during scanning in real-time, meanwhile for the MATLAB, LabView and Python you can only record RF Data retrospectively from cine loop memory to keep RF Data acquisition and imaging speed as high as possible for sample programs. Cine controls contains 1) **Play**, 2) **Stop** buttons and possibility to move through stored frames by **slider** and by entering **Go to frame** value. RF Data could be recorded in 3 ways: 1) **Record Single Frame**, 2) **Record N Frames**, 3) **Save** frames **From-To**.

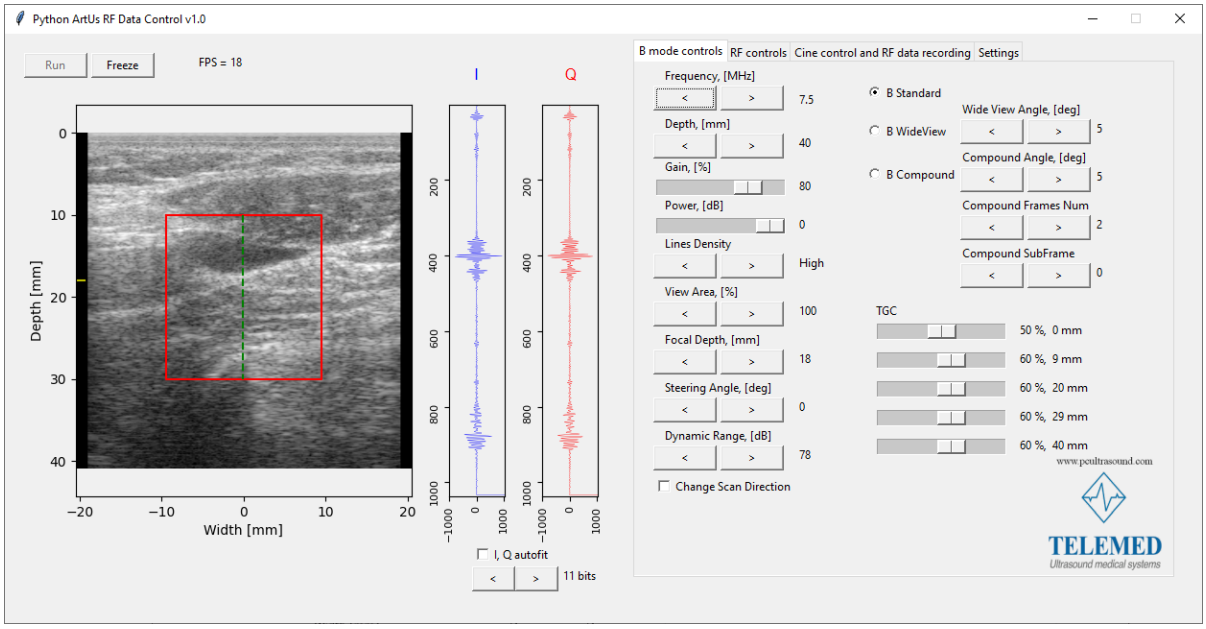


Fourth tab **Settings** are reserved for the future and currently has single **Probe** button, which calls on_probe_button_pressed… DLL function.



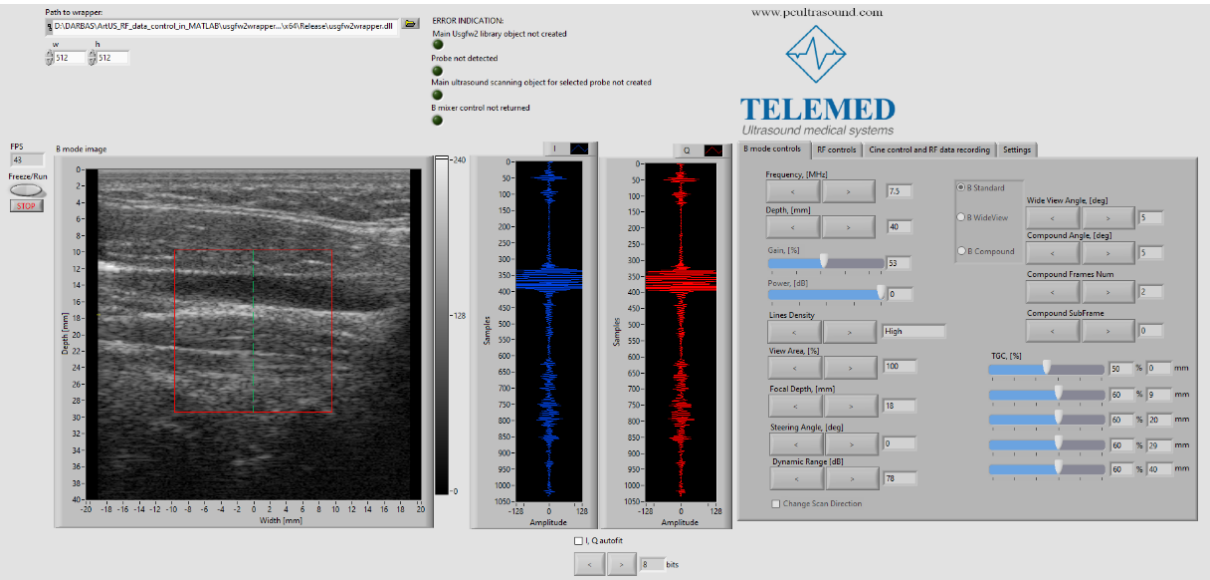
2.2. Python ArtUs RF Data Control sample program

Front panel of sample program prepared for Python users is show bellow. The sample program is organized exactly the same as MATLAB sample presented in previous chapter and will not be described in details in this subsection. Python sample program provides the lowest FPS (about 20 frames per second) in comparison to MATLAB and LabView samples due to slower built-in imaging functions.



2.3. ArtUsRF2labview sample program

Front panel of sample program prepared for LabVIEW users is show bellow. The sample program is organized exactly the same as MATLAB and Python samples presented in previous chapters and will not be described in details in this subsection, because the functionality is exactly the same.



2.4. Example of source code fragments for Python sample program

This subsection provides source code with comments of Python sample program to get knowledge on structure of the sample program. MATLAB and LabVIEW sample programs just repeats the functionality, only DLL functions calling routines slightly differs, and there is no sense to analyse them separately.

1. **Firstly, mandatory libraries and packages are imported, the main package for GUI building is *tkinter*:**

```
import tkinter
from tkinter import ttk
from tkinter import *
from tkinter import filedialog
from tkinter import Menu
import pdb;
import struct
import os
import math
import scipy
import numpy
import matplotlib
import sys
import ctypes
from ctypes import *
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from matplotlib.pyplot import imshow, show, draw, pause
import time
from winreg import *
from datetime import datetime
from tkinter import filedialog
from tkinter import messagebox
os.environ['SDL_HINT_WINDOWS_ENABLE_MESSAGELOOP'] = "0"
```

2. **Secondly, usgfw2 library objects are created, probe detected and scanning object for the probe created**

```
usgfw2.on_init()
ERR = usgfw2.init_ultrasound_usgfw2()
if (ERR == 2):
     print('Main Usgfw2 library object not created');
     sys.exit()

ERR = usgfw2.find_connected_probe()

if (ERR != 101):
    print('Probe not detected')
    print(ERR)
    sys.exit()

ERR = usgfw2.data_view_function()

if (ERR < 0):
     print('Main ultrasound scanning object for selected probe not created')
     sys.exit()

ERR = usgfw2.mixer_control_function(0,0,w,h,0,0,0)

if (ERR < 0):
     print('B mixer control not returned');
     sys.exit()
```

3. **Ultrasonography controls are created and call back functions for image and RF Data acquisition are set**

```
#---------------Scan converter call back for B mode image acquisition ----
```

```
usgfw2.set_callback_scan_converter()

#---------------Frequency control-------------------------------------------

usgfw2.frequency_control()
freq = c_int(0)
freq_no = c_long(0)

usgfw2.B_FrequencySetPrevNext(0, ctypes.pointer(freq), ctypes.pointer(freq_no));

frequency = str((freq.value)/1000000);

frequency_name = ttk.Label(tab1, text = 'Frequency, [MHz]')
frequency_name.place(x = 30, y = 5)

if (len(frequency)>3):
 if ((frequency[-1]) == '1'):
    frequency_label = ttk.Label(tab1, text = frequency[:-1] + ' ITHI')
 elif ((frequency[-1]) == '2'):
    frequency_label = ttk.Label(tab1, text = frequency[:-1] + ' THI')
 else:
    frequency_label = ttk.Label(tab1, text = frequency)
else:
    frequency_label = ttk.Label(tab1, text = frequency)

frequency_label.place(x = 170, y = 30)


#---------------Depth control-----------------------------------------
usgfw2.depth_control()
depth = ctypes.c_int(0)
usgfw2.DepthSetPrevNext(0, ctypes.pointer(depth))

depth_name = ttk.Label(tab1, text = 'Depth, [mm]')
depth_name.place(x = 30, y = 55)
depth_label = ttk.Label(tab1, text = depth.value)
depth_label.place(x = 170, y = 75)

#---------------Gain control-----------------------------------------
usgfw2.gain_control()
gain = ctypes.c_int(0)

usgfw2.B_GainSetByIdx(70, ctypes.pointer(gain))

gain_name = ttk.Label(tab1, text = 'Gain, [%]')
gain_name.place(x = 30, y = 100)
global gain_label
gain_label = ttk.Label(tab1, text = gain.value)
gain_label.place(x = 170, y = 120)

#---------------Power control-----------------------------------------
usgfw2.power_control()

power = ctypes.c_int(0)
usgfw2.B_PowerSetByIdx(20,ctypes.pointer(power));
power_name = ttk.Label(tab1, text = 'Power, [dB]')
power_name.place(x = 30, y = 140)
global power_label
power_label = ttk.Label(tab1, text = power.value)
power_label.place(x = 170, y = 160)

#---------------Lines density control ---------------------------------
usgfw2.lines_density();
lines_density = ctypes.c_int(0)
usgfw2.B_LinesDensitySetPrevNext(0, ctypes.pointer(lines_density));

if (lines_density.value == 8):
   string_LD = 'Low';

if (lines_density.value == 16):
   string_LD = 'Medium';

if (lines_density.value == 22):
   string_LD = 'Standard S';
```

```
if (lines_density.value == 24):
    string_LD = 'Standard';

if (lines_density.value == 32):
    string_LD = 'High';

lines_density_name = ttk.Label(tab1, text = 'Lines Density')
lines_density_name.place(x = 30, y = 180)
lines_density_label = ttk.Label(tab1, text = string_LD)
lines_density_label.place(x = 170, y = 200)

# --------------- View area control-----------------------------------------
usgfw2.view_area()
view_area = ctypes.c_int(0)
view_area_direction = 5;
usgfw2.B_view_areaSetPrevNext(view_area_direction, ctypes.pointer(view_area));

view_area_name = ttk.Label(tab1, text = 'View Area, [%]')
view_area_name.place(x = 30, y = 230)
view_area_label = ttk.Label(tab1, text = view_area.value)
view_area_label.place(x = 170, y = 250)
```

……

4. Call back functions for buttons and other controls are programmed

```
#---------------Frequency buttons-----------------------------------------
def frequency_button_down(usgfw2, frequency_label):

    frequency_label.config(text="")
    usgfw2.B_FrequencySetPrevNext(-1, ctypes.pointer(freq), ctypes.pointer(freq_no));
    frequency = str((freq.value)/1000000);

    if (len(frequency)>=3):
     if ((frequency[-1]) == '1'):
      frequency_label.config(text = frequency[:-1] + ' ITHI')
     elif ((frequency[-1]) == '2'):
      frequency_label.config(text = frequency[:-1] + ' THI')
     else:
      frequency_label.config(text = frequency)
    else:
      frequency_label.config(text = frequency)


def frequency_button_up(usgfw2, frequency_label):

    frequency_label.config(text="")
    usgfw2.B_FrequencySetPrevNext(+1, ctypes.pointer(freq), ctypes.pointer(freq_no));
    frequency = str((freq.value)/1000000);

    if (len(frequency)>=3):
     if ((frequency[-1]) == '1'):
      frequency_label.config(text = frequency[:-1] + ' ITHI')
     elif ((frequency[-1]) == '2'):
      frequency_label.config(text = frequency[:-1] + ' THI')
     else:
      frequency_label.config(text = frequency)
    else:
      frequency_label.config(text = frequency)


#---------------Depth buttons-----------------------------------------
def depth_button_down(usgfw2, depth_label):
   depth_label.config(text="")
   usgfw2.DepthSetPrevNext(-1, ctypes.pointer(depth))
   depth_label.config(text=depth.value)

   TGC1_label.config(text="")
   usgfw2.adjust_TGC(0, tgc_slider1.get(), ctypes.pointer(TGC_depth1));
   TGC1_label.config(text = str(tgc_slider1.get()) + ' %,  ' + str(TGC_depth1.value) + ' mm')

   TGC2_label.config(text="")
   usgfw2.adjust_TGC(1, tgc_slider2.get(), ctypes.pointer(TGC_depth2));
   TGC2_label.config(text = str(tgc_slider2.get()) + ' %,  ' + str(TGC_depth2.value) + ' mm')
```

```
    TGC3_label.config(text="")
    usgfw2.adjust_TGC(2, tgc_slider3.get(), ctypes.pointer(TGC_depth3));
    TGC3_label.config(text = str(tgc_slider3.get()) + ' %,  ' + str(TGC_depth3.value) + ' mm')

    TGC4_label.config(text="")
    usgfw2.adjust_TGC(3, tgc_slider4.get(), ctypes.pointer(TGC_depth4));
    TGC4_label.config(text = str(tgc_slider4.get()) + ' %,  ' + str(TGC_depth4.value) + ' mm')

    TGC5_label.config(text="")
    usgfw2.adjust_TGC(4, tgc_slider1.get(), ctypes.pointer(TGC_depth5));
    TGC5_label.config(text = str(tgc_slider5.get()) + ' %,  ' + str(TGC_depth5.value) + ' mm')

def depth_button_up(usgfw2, depth_label):
    depth_label.config(text="")
    usgfw2.DepthSetPrevNext(+1, ctypes.pointer(depth))
    depth_label.config(text=depth.value)

    TGC1_label.config(text="")
    usgfw2.adjust_TGC(0, tgc_slider1.get(), ctypes.pointer(TGC_depth1));
    TGC1_label.config(text = str(tgc_slider1.get()) + ' %,  ' + str(TGC_depth1.value) + ' mm')

    TGC2_label.config(text="")
    usgfw2.adjust_TGC(1, tgc_slider2.get(), ctypes.pointer(TGC_depth2));
    TGC2_label.config(text = str(tgc_slider2.get()) + ' %,  ' + str(TGC_depth2.value) + ' mm')

    TGC3_label.config(text="")
    usgfw2.adjust_TGC(2, tgc_slider3.get(), ctypes.pointer(TGC_depth3));
    TGC3_label.config(text = str(tgc_slider3.get()) + ' %,  ' + str(TGC_depth3.value) + ' mm')

    TGC4_label.config(text="")
    usgfw2.adjust_TGC(3, tgc_slider4.get(), ctypes.pointer(TGC_depth4));
    TGC4_label.config(text = str(tgc_slider4.get()) + ' %,  ' + str(TGC_depth4.value) + ' mm')

    TGC5_label.config(text="")
    usgfw2.adjust_TGC(4, tgc_slider1.get(), ctypes.pointer(TGC_depth5));
    TGC5_label.config(text = str(tgc_slider5.get()) + ' %,  ' + str(TGC_depth5.value) + ' mm')

#---------------Gain slider----------------------------------------
def gain_slider_click(event):
     gain_label.config(text="")
     usgfw2.B_GainSetByIdx(gain_slider.get(), ctypes.pointer(gain))
     gain_label.config(text=gain.value)
……..
```

5. The main RUN button which calls imaging function to update displayed data of B mode image and RF data

```
     def clicked_RUN(usgfw2):

          # ---- Enable controls when run -------------------------
         WideView_angle_button_minus.configure(state = 'normal')
         WideView_angle_button_plus.configure(state = 'normal')
         Compound_angle_button_minus.configure(state = 'normal')
         Compound_angle_button_plus.configure(state = 'normal')
         Compound_frames_button_minus.configure(state = 'normal')
         Compound_frames_button_plus.configure(state = 'normal')
         Compound_subframes_button_minus.configure(state = 'normal')
         Compound_subframes_button_plus.configure(state = 'normal')
         frequency_button_minus.configure(state = 'normal')
         frequency_button_plus.configure(state = 'normal')
         depth_button_plus.configure(state = 'normal')
         depth_button_minus.configure(state = 'normal')
         gain_slider.configure(state = 'normal')
         power_slider.configure(state = 'normal')
         lines_density_button_minus.configure(state = 'normal')
         lines_density_button_plus.configure(state = 'normal')
         view_area_button_minus.configure(state = 'normal')
         view_area_button_plus.configure(state = 'normal')
         focal_depth_button_minus.configure(state = 'normal')
         focal_depth_button_plus.configure(state = 'normal')
         tgc_slider1.configure(state = 'normal')
```

```
        tgc_slider2.configure(state = 'normal')
        tgc_slider3.configure(state = 'normal')
        tgc_slider4.configure(state = 'normal')
        tgc_slider5.configure(state = 'normal')
        radiobutton1.configure(state = 'normal')
        radiobutton2.configure(state = 'normal')
        radiobutton3.configure(state = 'normal')
        RF_window_right.configure(state = 'normal')
        RF_window_left.configure(state = 'normal')
        RF_window_up.configure(state = 'normal')
        RF_window_down.configure(state = 'normal')
        RF_window_size_rght.configure(state = 'normal')
        RF_window_size_lft.configure(state = 'normal')
        RF_window_size_upp.configure(state = 'normal')
        RF_window_size_dwn.configure(state = 'normal')
        radiobutton_beamformer_output.configure(state = 'normal')
        radiobutton_TFC_poutput.configure(state = 'normal')
        radiobutton_apodization_output.configure(state = 'normal')
        radiobutton_hilbert_output.configure(state = 'normal')
        if (available_steering_angle == 1):
          steering_angle_button_minus.configure(state = 'normal')
          steering_angle_button_plus.configure(state = 'normal')
        dynamic_range_button_minus.configure(state = 'normal')
        dynamic_range_button_plus.configure(state = 'normal')
        Scan_direction.configure(state = 'normal')

        btn.configure(state = 'disabled')
        btn1.configure(state = 'normal')

        Go_to_frame_button.configure(state = 'disabled')
        Go_to_frame_editbox.configure(state = 'disabled')
        Cine_frame_slider.configure(state = 'disabled')
        Stop_Cine_Loop.configure(state = 'disabled')
        Play_Cine_Loop.configure(state = 'disabled')

        RF_Data_record_single_frame_button.configure(state = 'disabled')
        RF_Data_N_frames_button.configure(state = 'disabled')
        RF_Data_N_frames_editbox.configure(state = 'disabled')
        RF_Data_From_frames_editbox.configure(state = 'disabled')
        RF_Data_To_frames_editbox.configure(state = 'disabled')
        RF_Data_Save_button.configure(state = 'disabled')

        run_loop = 1;
        global focal_depth_marker, buffer_as_numpy_array, RF_line, RFimg, RF_buffer_as_numpy_array, RFimg2,
number_of_samples_in_window_for_beam, Q_component_label, ax, difference1,reshaped_array, X_axis, Y_axis, p_array, p1,
p2, img, res_X, res_Y, fig, plt1, P1, P2, P3, P4, RF_Row_to_show, RFimg, RFimg2, FPS_label,Q_component_label,
RF_window, RF_line

        usgfw2.Run_ultrasound_scanning()
        p_array = (ctypes.c_uint*w*h*4)()

        p1 = ctypes.c_int(0);
        p2 = ctypes.c_float(0.0);

        usgfw2.return_pixel_values2(ctypes.pointer(p_array), ctypes.pointer(p1), ctypes.pointer(p2))
        buffer_as_numpy_array = np.frombuffer(p_array, np.uint)
        reshaped_array = np.reshape(buffer_as_numpy_array,(w, h, 4))

        fig, ax = plt.subplots(1,3, gridspec_kw={'width_ratios':[7.5,1.25,1.25]})
        fig.subplots_adjust(left=0.1, bottom=0.1, right=0.95, top=0.95, wspace=0.25, hspace=None)

        fig.patch.set_facecolor((0.94, 0.94, 0.94))
        ax[0].set_facecolor((0.94, 0.94, 0.94))
        img = ax[0].imshow(reshaped_array[::-1,:,0], cmap="gray",vmin=0, vmax=255,origin='lower',
                extent =[np.amin(X_axis), np.amax(X_axis), np.amax(Y_axis), np.amin(Y_axis)])

        ax[0].set_xlabel('Width [mm]')
        ax[0].set_ylabel('Depth [mm]')
        ax[0].axis('equal')

        # ------------- RF window and RF line drawing -------------------------------------------
        rect_position_x_1 = np.zeros(shape=(line2.value - line1.value + 1, 1));
        rect_position_y_1 = np.zeros(shape=(line2.value - line1.value + 1, 1));
        rect_position_x_2 = np.zeros(shape=(line2.value - line1.value + 1, 1));
        rect_position_y_2 = np.zeros(shape=(line2.value - line1.value + 1, 1));
```

```
        for i in range (line1.value,line2.value + 1):

          X1pix1line = ctypes.c_float(0.0)
          Y1pix1line = ctypes.c_float(0.0)
          X2pix1line = ctypes.c_float(0.0)
          Y2pix1line = ctypes.c_float(0.0)

          usgfw2.convert_units_to_pixels(i,SubFrameIndex.value, depth1.value, depth2.value, ctypes.pointer(X1pix1line),
ctypes.pointer(Y1pix1line), ctypes.pointer(X2pix1line), ctypes.pointer(Y2pix1line))

          rect_position_x_1[i - line1.value] = (X1pix1line.value-difference1)*res_X.value;
          rect_position_y_1[i - line1.value] = (Y1pix1line.value)*res_Y.value;

          rect_position_x_2[i - line1.value] = (X2pix1line.value-difference1)*res_X.value;
          rect_position_y_2[i - line1.value] = (Y2pix1line.value)*res_Y.value;

        rect_position_x = np.concatenate((rect_position_x_1, rect_position_x_2[::-1]), axis=None)
        rect_position_y = np.concatenate((rect_position_y_1, rect_position_y_2[::-1]), axis=None)
        rect_position_x = np.concatenate((rect_position_x, rect_position_x[0]), axis=None)
        rect_position_y = np.concatenate((rect_position_y, rect_position_y[0]), axis=None)

        RF_window, = ax[0].plot(rect_position_x,rect_position_y, color='red')

        #RF_Row_to_show = int(np.fix(lines_number.value/2));
        #old_RF_Row_to_show = RF_Row_to_show;

        X1pixRFline = ctypes.c_float(0.0)
        Y1pixRFline = ctypes.c_float(0.0)
        X2pixRFline = ctypes.c_float(0.0)
        Y2pixRFline = ctypes.c_float(0.0)
        Line = int(RF_Row_to_show + line1.value)
        usgfw2.convert_units_to_pixels(Line,SubFrameIndex.value, depth1.value, depth2.value, ctypes.pointer(X1pixRFline),
ctypes.pointer(Y1pixRFline), ctypes.pointer(X2pixRFline), ctypes.pointer(Y2pixRFline))

        XRFline = [(X1pixRFline.value-difference1)*res_X.value, (X2pixRFline.value-difference1)*res_Y.value];
        YRFline = [Y1pixRFline.value*res_X.value, Y2pixRFline.value*res_Y.value];
        RF_line, = ax[0].plot(XRFline,YRFline,'g--');

        I_component_label = Label(window, text = "I",fg="blue", font = 14)
        I_component_label.place(x = 487, y = 30)

        Q_component_label = Label(window, text = "",fg="red", font = 14)
        Q_component_label.place(x = 581, y = 30)

        focal_depth_marker, = ax[0].plot([np.amin(X_axis), np.amin(X_axis)+1],[focal_depth.value, focal_depth.value],'y')

        # ---------------------------------------------------------------------------
        P1 = ctypes.c_int(0)
        P2 = ctypes.c_double(0);
        P3 = ctypes.c_int(0);
        P4 = ctypes.c_int(0);

        if (source_ID == 4):
          p_array_RF = (ctypes.c_int16*number_of_samples_in_window_for_beam*lines_number.value*2)()

usgfw2.return_RF_data(ctypes.pointer(p_array_RF),number_of_samples_in_window_for_beam*lines_number.value*2,ctypes.p
ointer(P1),ctypes.pointer(P2),ctypes.pointer(P3),ctypes.pointer(P4))
        else:
          p_array_RF = (ctypes.c_int16*number_of_samples_in_window_for_beam*lines_number.value)()

usgfw2.return_RF_data(ctypes.pointer(p_array_RF),number_of_samples_in_window_for_beam*lines_number.value,ctypes.poi
nter(P1),ctypes.pointer(P2),ctypes.pointer(P3),ctypes.pointer(P4))

        RF_buffer_as_numpy_array = np.frombuffer(p_array_RF, np.int16)

        RF_data_to_show =
RF_buffer_as_numpy_array[RF_Row_to_show*number_of_samples_in_window_for_beam+3:RF_Row_to_show*number_of_s
amples_in_window_for_beam + number_of_samples_in_window_for_beam]

        y_axis_RF = range(1, number_of_samples_in_window_for_beam-2)
        RFimg, = ax[1].plot(RF_data_to_show, y_axis_RF,color = 'blue',linewidth=0.25)
        ax[1].invert_yaxis()
        ax[1].axis(xmin=-50, xmax = 50, ymin = number_of_samples_in_window_for_beam-1,ymax = 1)
        ax[1].tick_params(axis='both', which='major', labelsize=8, rotation=90)
        ax[1].set_facecolor((0.94, 0.94, 0.94))
```

```
        ax[2].tick_params(axis='both', which='major', labelsize=8, rotation=90)
        ax[2].set_facecolor((0.94, 0.94, 0.94))
        ax[2].axis('off')

        RFimg2, = ax[2].plot(0,0,color = 'red',linewidth=0.25)
        if (source_ID == 4):
          RF_data_to_show2 = RF_buffer_as_numpy_array[(number_of_samples_in_window_for_beam*lines_number.value) +
(RF_Row_to_show*number_of_samples_in_window_for_beam+3):(number_of_samples_in_window_for_beam*lines_number.v
alue)+RF_Row_to_show*number_of_samples_in_window_for_beam + number_of_samples_in_window_for_beam]
          ax[2].axis('on')
          RFimg2, = ax[2].plot(RF_data_to_show2, y_axis_RF,color = 'red',linewidth=0.25)
          Q_component_label.config(text = "Q")

        FPS_label = ttk.Label(window, text = 'FPS =')
        FPS_label.place(x = 200, y = 20)

usgfw2.RF_window_get(ctypes.pointer(depth1),ctypes.pointer(depth2),ctypes.pointer(line1),ctypes.pointer(line2),ctypes.pointer
(lines_number))

        old_depth1 = depth1.value;
        old_depth2 =depth2.value;
        old_line1 = line1.value;
        old_line2 = line2.value;

        matplotlib.pyplot.close()
        global plt1
        plt1 = FigureCanvasTkAgg(fig,window)

        while (run_loop==1):

         global stop_imaging
         if (stop_imaging == 1):
           stop_imaging = 0;
           break;

         matplotCanvas(window,ax, difference1,reshaped_array[:,:,0:3], X_axis, Y_axis, p_array, p1, p2, img, usgfw2, res_X,
res_Y, fig, plt1, P1, P2, P3, P4, RF_Row_to_show, RFimg, RFimg2, FPS_label,Q_component_label, RF_window, RF_line,
focal_depth_marker)
```

6. The main imaging function *matplotCanvas* for image and RF data updating in axes

```
def matplotCanvas(window,ax, difference1,reshaped,X_axis, Y_axis, p_array, p1, p2, img, usgfw2, res_X, res_Y, f, plt1, P1,
P2, P3, P4, RF_Row_to_show, RFimg, RFimg2, FPS_label, Q_component_label, RF_window, RF_line, focal_depth_marker):

    start_time = time.time()

    window.update()

    global RF_buffer_as_numpy_array
    if (stop_imaging == 1):
      return


    usgfw2.RF_window_get(ctypes.pointer(depth1), ctypes.pointer(depth2), ctypes.pointer(line1), ctypes.pointer(line2),
ctypes.pointer(lines_number))
    usgfw2.get_resolution(ctypes.pointer(res_X),ctypes.pointer(res_Y))

    RF_line_slider.config(from_=0, to=lines_number.value-1)
    RF_line_slider_label.config(text = str(RF_Row_to_show+1)+ '/' + str(lines_number.value))

    if (RF_Row_to_show+1 > lines_number.value and (lines_number.value-RF_Row_to_show+1)!=0):
        RF_Row_to_show = int((lines_number.value-1)/2)
        RF_line_slider.set(RF_Row_to_show)
    if ((lines_number.value-RF_Row_to_show+1)==0):
        RF_Row_to_show = lines_number.value-1
        RF_line_slider.set(RF_Row_to_show)

    if (w % 2 == 0):
        k = 0;
        for i in range (-w//2, w//2+1):
         if (i<0):
```

```
        j=i+0.5
        X_axis[k] = j*res_X.value
        k = k+1
       else:
        if (i>0):
          j=i-0.5
          X_axis[k] = j*res_X.value
          k = k+1
    else:
       for i in range (-w//2, w//2):
        X_axis[i+w/2 + 1] = i*res_X.value

    for i in range (0,h-1):
       Y_axis[i] = i*res_Y.value - Y1pix1line_0.value*res_Y.value;

    rect_position_x_1 = np.zeros(shape=(line2.value - line1.value + 1, 1));
    rect_position_y_1 = np.zeros(shape=(line2.value - line1.value + 1, 1));
    rect_position_x_2 = np.zeros(shape=(line2.value - line1.value + 1, 1));
    rect_position_y_2 = np.zeros(shape=(line2.value - line1.value + 1, 1));
    for i in range (line1.value,line2.value + 1):

     X1pix1line = ctypes.c_float(0.0)
     Y1pix1line = ctypes.c_float(0.0)
     X2pix1line = ctypes.c_float(0.0)
     Y2pix1line = ctypes.c_float(0.0)

     usgfw2.convert_units_to_pixels(i,SubFrameIndex.value, depth1.value, depth2.value, ctypes.pointer(X1pix1line),
ctypes.pointer(Y1pix1line), ctypes.pointer(X2pix1line), ctypes.pointer(Y2pix1line))

     rect_position_x_1[i - line1.value] = (X1pix1line.value-difference1)*res_X.value;
     rect_position_y_1[i - line1.value] = (Y1pix1line.value)*res_Y.value;

     rect_position_x_2[i - line1.value] = (X2pix1line.value-difference1)*res_X.value;
     rect_position_y_2[i - line1.value] = (Y2pix1line.value)*res_Y.value;

    rect_position_x = np.concatenate((rect_position_x_1, rect_position_x_2[::-1]), axis=None)
    rect_position_y = np.concatenate((rect_position_y_1, rect_position_y_2[::-1]), axis=None)
    rect_position_x = np.concatenate((rect_position_x, rect_position_x[0]), axis=None)
    rect_position_y = np.concatenate((rect_position_y, rect_position_y[0]), axis=None)

    RF_window.set_xdata(rect_position_x);
    RF_window.set_ydata(rect_position_y - Y1pix1line_0.value*res_Y.value);

    X1pixRFline = ctypes.c_float(0.0)
    Y1pixRFline = ctypes.c_float(0.0)
    X2pixRFline = ctypes.c_float(0.0)
    Y2pixRFline = ctypes.c_float(0.0)
    Line = int(RF_Row_to_show + line1.value)
    usgfw2.convert_units_to_pixels(Line,SubFrameIndex.value, depth1.value, depth2.value, ctypes.pointer(X1pixRFline),
ctypes.pointer(Y1pixRFline), ctypes.pointer(X2pixRFline), ctypes.pointer(Y2pixRFline))

    XRFline = [(X1pixRFline.value-difference1)*res_X.value, (X2pixRFline.value-difference1)*res_Y.value];
    YRFline = [Y1pixRFline.value*res_X.value - Y1pix1line_0.value*res_Y.value, Y2pixRFline.value*res_Y.value -
Y1pix1line_0.value*res_Y.value];

    RF_line.set_xdata(XRFline);
    RF_line.set_ydata(YRFline);

    focal_depth_marker.set_xdata([np.amin(X_axis), np.amin(X_axis)+1]);
    focal_depth_marker.set_ydata([focal_depth.value, focal_depth.value]);


    usgfw2.return_pixel_values2(ctypes.pointer(p_array), ctypes.pointer(p1), ctypes.pointer(p2))
    buffer_as_numpy_array = np.frombuffer(p_array, np.uint)
    reshaped_array = np.reshape(buffer_as_numpy_array,(w, h, 4))

    img.set_data(reshaped_array[::-1,:,0])
    img.set_extent([np.amin(X_axis), np.amax(X_axis), np.amax(Y_axis), np.amin(Y_axis)])

    # ---------- RF imaging ----------------------------------------------------------------
    number_of_samples_in_window_for_beam = int(numpy.fix(2*((depth2.value-
depth1.value)/1000)/1540/((sampling_period_ns.value)*1e-9)))
    if (source_ID == 4):
      p_array_RF = (ctypes.c_int16*number_of_samples_in_window_for_beam*lines_number.value*2)()
```

```
usgfw2.return_RF_data(ctypes.pointer(p_array_RF),number_of_samples_in_window_for_beam*lines_number.value*2,ctypes.p
ointer(P1),ctypes.pointer(P2),ctypes.pointer(P3),ctypes.pointer(P4))
      ax[2].axis('on')
    else:
      p_array_RF = (ctypes.c_int16*number_of_samples_in_window_for_beam*lines_number.value)()

usgfw2.return_RF_data(ctypes.pointer(p_array_RF),number_of_samples_in_window_for_beam*lines_number.value,ctypes.poi
nter(P1),ctypes.pointer(P2),ctypes.pointer(P3),ctypes.pointer(P4))
      ax[2].axis('off')

    RF_buffer_as_numpy_array = np.frombuffer(p_array_RF, np.int16)

    RF_data_to_show =
RF_buffer_as_numpy_array[RF_Row_to_show*number_of_samples_in_window_for_beam+3:RF_Row_to_show*number_of_s
amples_in_window_for_beam + number_of_samples_in_window_for_beam]
    y_axis_RF = range(1, number_of_samples_in_window_for_beam-2)

    if (source_ID == 4):
     RF_data_to_show2 = RF_buffer_as_numpy_array[(number_of_samples_in_window_for_beam*lines_number.value) +
(RF_Row_to_show*number_of_samples_in_window_for_beam+3):(number_of_samples_in_window_for_beam*lines_number.v
alue)+RF_Row_to_show*number_of_samples_in_window_for_beam + number_of_samples_in_window_for_beam]
     RFimg2.set_xdata(RF_data_to_show2)
     RFimg2.set_ydata(y_axis_RF)
     if (AutoSC == 1):
      ax[2].axis(xmin=np.amin(RF_data_to_show2)-5, xmax = np.amax(RF_data_to_show2)+5, ymin =
number_of_samples_in_window_for_beam-1,ymax = 1)
     else:
      ax[2].axis(xmin=-2**bits/2, xmax = 2**bits/2, ymin = number_of_samples_in_window_for_beam-1,ymax = 1)
     Q_component_label.config(text = "Q")
    else:
     RFimg2.set_xdata(0)
     RFimg2.set_ydata(0)
     Q_component_label.config(text = "   ")

    RFimg.set_xdata(RF_data_to_show)
    RFimg.set_ydata(y_axis_RF)
    if (AutoSC == 1):
      ax[1].axis(xmin=np.amin(RF_data_to_show)-5, xmax = np.amax(RF_data_to_show)+5, ymin =
number_of_samples_in_window_for_beam-1,ymax = 1)
    else:
      ax[1].axis(xmin=-2**bits/2, xmax = 2**bits/2, ymin = number_of_samples_in_window_for_beam-1,ymax = 1)

    plt1.draw()
    plt1.get_tk_widget().place(x=10,y=50)

    FPS_label.configure(text = 'FPS = ' + str(round(1/(time.time() - start_time))))
```

7. All ultrasonography controls used in Python interface are declared and programmed

```
# ---------------- Ultrasonography controls ------------------------------------------------
btn = Button(window, text="Run", command=lambda:clicked_RUN(usgfw2), height = 1, width = 8)
btn.place(x=20, y=20)

btn1 = Button(window, text="Freeze", command=lambda:clicked_freeze(usgfw2), height = 1, width = 8)
btn1.place(x=90, y=20)

# ---------------- B mode controls ------------------------------------------------
frequency_button_minus = Button(tab1, text="<", command=lambda:frequency_button_down(usgfw2, frequency_label), height
= 1, width = 8)
frequency_button_minus.place(x=20, y=25)

frequency_button_plus = Button(tab1, text=">", command=lambda:frequency_button_up(usgfw2, frequency_label), height = 1,
width = 8)
frequency_button_plus.place(x=90, y=25)

depth_button_minus = Button(tab1, text="<", command=lambda:depth_button_down(usgfw2, depth_label), height = 1, width =
8)
depth_button_minus.place(x=20, y=75)

depth_button_plus = Button(tab1, text=">", command=lambda:depth_button_up(usgfw2, depth_label), height = 1, width = 8)
depth_button_plus.place(x=90, y=75)
```

```
gain_slider = Scale(tab1, from_=0, to=90, orient=HORIZONTAL,command=gain_slider_click,showvalue = 0, length = 135)
gain_slider.set(70)
gain_slider.place(x=20, y=120)

power_slider = Scale(tab1, from_=0, to=20, orient=HORIZONTAL,command=power_slider_click,showvalue = 0, length = 135)
power_slider.set(20)
power_slider.place(x=20, y=160)

lines_density_button_minus = Button(tab1, text="<", command=lambda:lines_density_button_down(usgfw2,
lines_density_label), height = 1, width = 8)
lines_density_button_minus.place(x=20, y=200)

lines_density_button_plus = Button(tab1, text=">", command=lambda:lines_density_button_up(usgfw2, lines_density_label),
height = 1, width = 8)
lines_density_button_plus.place(x=90, y=200)

view_area_button_minus = Button(tab1, text="<", command=lambda:view_area_button_down(usgfw2, view_area_label), height
= 1, width = 8)
view_area_button_minus.place(x=20, y=250)

view_area_button_plus = Button(tab1, text=">", command=lambda:view_area_button_up(usgfw2, view_area_label), height = 1,
width = 8)
view_area_button_plus.place(x=90, y=250)

focal_depth_button_minus = Button(tab1, text="<", command=lambda:focal_depth_button_down(usgfw2, focal_depth_label),
height = 1, width = 8)
focal_depth_button_minus.place(x=20, y=300)

focal_depth_button_plus = Button(tab1, text=">", command=lambda:focal_depth_button_up(usgfw2, focal_depth_label), height
= 1, width = 8)
focal_depth_button_plus.place(x=90, y=300)

steering_angle_button_minus = Button(tab1, text="<", command=lambda:steering_angle_button_down(usgfw2,
steering_angle_label), height = 1, width = 8)
steering_angle_button_minus.place(x=20, y=350)

steering_angle_button_plus = Button(tab1, text=">", command=lambda:steering_angle_button_up(usgfw2,
steering_angle_label), height = 1, width = 8)
steering_angle_button_plus.place(x=90, y=350)

available_steering_angle = usgfw2.steering_angle()
if (available_steering_angle == 0):
   steering_angle_button_minus.configure(state = 'disabled')
   steering_angle_button_plus.configure(state = 'disabled')
else:
   steering_angle_button_minus.configure(state = 'normal')
   steering_angle_button_plus.configure(state = 'normal')
```

2.5. Installation and starting notes

To run sample programs the path to DLL file must be provided (for MATLAB sample path to header file must be declared as well).

MATLAB sample DLL calling example (112-118 lines of code):

```
PATH =
'D:\DARBAS\ArtUS_RF_data_control_in_MATLAB\usgfw2wrapper_C++_sources\usgfw2wrapper\x64\Release
\usgfw2wrapper.dll'; % path to dll library which operates between MATLAB and usgfw2 SDK

hfile =
'D:\DARBAS\ArtUS_RF_data_control_in_MATLAB\usgfw2wrapper_C++_sources\usgfw2wrapper\usgfw2wrapp
er.h'; % path to h file of library which operates between MATLAB and usgfw2 SDK

% checking if library is loaded and loading if not
if not(libisloaded('USGFW2MATLABWRAPPER'))
   loadlibrary(PATH,hfile,'alias', 'USGFW2MATLABWRAPPER');
   libfunctions USGFW2MATLABWRAPPER -full
```

An example of calling DLL function in MATLAB

```
calllib('USGFW2MATLABWRAPPER','on_init'); % function initializes variables used in scripts
```

PYTHON sample DLL calling example (line 93 of code):

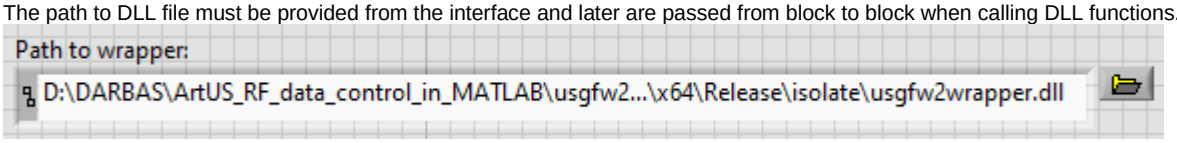
```
usgfw2 =
cdll.LoadLibrary('D:/DARBAS/ArtUS_RF_data_control_in_MATLAB/usgfw2wrapper_C++_sources/usgfw2wr
apper/x64/Release/usgfw2wrapper.dll')
```

An example of calling DLL function in Python

```
usgfw2.on_init()
```

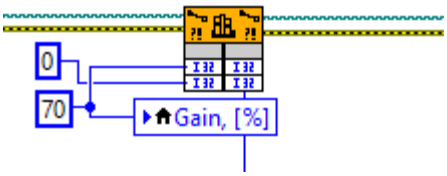
LabVIEW sample DLL calling example:

The path to DLL file must be provided from the interface and later are passed from block to block when calling DLL functions.

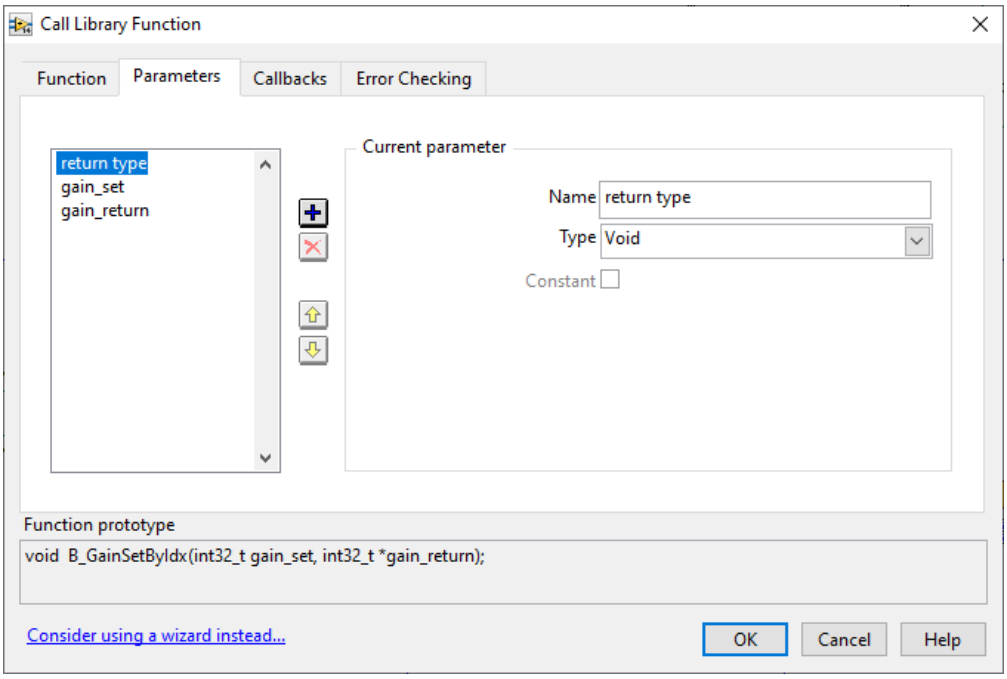


An example of calling DLL function in LabVIEW:

Call Library Function blocks are used for the purpose. The parameters for function inputs and outputs must be selected individually for each function of the DLL.



In this example ultrasonography Gain control are set to 70 % value. Function has void return type and two parameters to pass (*gain_set* and *gain_return*).



Please note: currently wrapper DLL was compiled for x64 bit version, so TELEMED SDK x64 must be installed as well. For successfully running MATLAB samples Microsoft Windows SDK 7 must be installed (x64 version), Download Microsoft Windows SDK for Windows 7 and .NET

Framework 4 from Official Microsoft Download Center, because some header files are needed for running MATLAB samples. You must have an installation in the following path "C:\Program Files\Microsoft SDKs\Windows\v7.1\Include", or if the installation path slightly differs you have to recompile the usgfw2wrapper project including the DIR to your Microsoft Windows SDK (x64). MATLAB scripts were prepared by using 2020a version (for older versions it might be necessary to install MinGW-w64 C/C++ compiler).

3. Revision history

Version	Date	Description of Revision	Revision author
1.0.0	20/09/2022	Initial Release	A. Sakalauskas

4. References

[1] ArtUs RF Data Control User Manual.

[2] Echo Blaster 64, Echo Blaster 128, LS64, LS128, ClarUs, SmartUs, MicrUs and ArtUs Series Ultrasound Systems Ultrasonography for Windows II Software Development Kit (Usgfw2 SDK) Programmer's Guide.