Chapter 3 – WR Teacher source code

This chapter is a guide to read and understand WRTeacher's source code.

3.1 AudioWidget

It provides controls to edit the sound buffer:
- Move cursor left / right.
- Set start and stop points.
- Cut borders.
- Zoom sound buffer.

Header file : AudioWidget.h

Includes GLAudio control (which displays the sound buffer), Qt PushButton and ScrollView window
Defines a class AudioWidget, with Qt signal/slot support; a destructor has been implemented to delete PushButtons and ScrollView allocated in the costructor
This widget emits signal “BufChamged” when the user, acting on the “Cut Borders” PushButton, modifies the sound buffer.
The “UpdateBuffer” slot fires when the buffer is modified from an external widget (RecorderWidget, which records a new soud, or RecorderConrolsWidget, which can erase the buffer)
mpDataBuffer and mBufSize are, respectively, a pointer to the sound buffer and the size, in bytes, of the buffer.
Finally, buttons and a scrollview widget are defined as pointers: they will be allocated in the constructor and deleted in the destructor.

Source code : AudioWidget.cpp

The constructor sets the widget's size, allocates buttons and ScrollView and sets its positions and sizes.
Bitmap images displayed on the buttons are read from disk, and must be in the same directory of WRTeacher's executable.
The ScrollView widget contains the GLAudio control, which size can be greater than ScrollView widget's: in this case, ScrollView widget automatically provides a scrollbar.
This module only connects edit buttons to GLAudio's slots.
UpdateBuffer / BufChanged signal/slot connections are handled by RecorderWidget; AudioWidget only updates his buffer pointer and size when RecorderWidget orders to do it.
The destructor releases memory allocated in the constructor.

3.2 FileIo

This module displays a modal window to let him choose wether to save or load whistle, noise and microphone's igain.
Functions ExistsWhistle, ExistsNoise and ExistsMiclevel must be called before displaying the window and they are used to activate the RadioButtons.
After the user clicks the “OK” button, functions Whistle, Noise and Miclevel can be used to check the RadioButtons status.

Header file : FileIo.h

Includes the Qt standard dialog window (DialogWidget subclesses it) and uses RadioButtons.
Defines a struct which is always written at the beginning of the .wrs file.
This struct containes igain (0 – 100 or -1 if not present), whistle and noise buffer sizes (0 if not present).

```
typedef struct AudioHeaderType //Header of file
```

```
{ int MicLevel; // -1 if not present
  long BufSizeWhistle; // 0 if not present
  long BufSizeNoise; // 0 if not present
} AudioHeaderType;
```

SetWhistle(), SetNoise() and SetMicLevel() slots fire when the user changes the state of a RadioButton.
Finally, controls are defined: whistle, noise, miclevel RadioButtons and the “OK” PushButton.

Source code : FileIo.cpp

Constructor defines RadioButtons and the “OK” PushButton.
When the user presses “OK”, the widget is automatically hidden and its exec() method terminates returning QDialog::Accepted;
Destructor just frees allocated memory.
SetWhistle(), SetNoise() and SetMicLevel() slots save the state of the RadioButtons when it is changed by the user.
ExistsWhistle(), ExistNoise() and ExistsMicLevel() functions activate RadioButtons, while Wistle(), Noise() and MicLevel() functions return the user's choiches

3.3 GlAudio

This is an **OpenGL** module that displays the sound waveform.
It has got a cursor (yellow vertical line) a selection range (drawn with a dark background), and its zoom can be changed from 1:1 to 4:1.
This module also operates on the sound buffer, deleting samples external to the selected range, when the slot DeleteBorders() is fired by pressing the “Delete Borders” PushButton in the AudioWidget interface.

Header file : GlAudio.h

It subclasses the QGLWidget class.
By default, the paintGL(), initializeGL() and resizeGL() events must be also subclassed.
The SetPosition() function is used to initialize or move the cursor's position.
SetStartHere(), SetStopHere(), ZoomIn(), ZoomOut(), MoveLeft(), MoveRight() and DeleteBorders() slots are fired by pressing the corresponding PushButtons in the AudioWidget interface.
When UpdateBuffer slot is fired by the RecorderWidget module, this module just refreshes its buffer pointer (mpDataBuffer) and size (mNumSamples).
The BufChanged signal is emitted when the DeleteBorders slot has terminated the execution of its code.
Since the area of this control is fixed, as the user cannot resize it dragging its borders (see the initializeGL() function), so the resizeGL() function has not been implemented.

Source code: GlAudio.cpp

The constructor is used for reset at startup, destructor is empty, as this module does not allocate any resource.
The initializeGL() function is called almost once before the control's area is painted.
This function sets the OpenGL viewport (which can be smaller than the client area) and projection's parameters.
The paintGL() function repaints the viewport and it is fired every time the public slot updateGL() method is called.
In this module, the size of the viewport can change, when the user changes the zoom by pressing the “Zoom In” and “Zoom Out” PushButtons in the AudioWindow interface, so initializeGL() must be reacalled every time before drawing, to adjust the viewport's size.
As previously discussed, there is a slot for every PushButton in the area; all these slots do the required job and redraw the viewport
The mousePressEvent provides a shortcut to define the selection range.
When the user left-clicks or right-clicks on the client area, start sample and stop sample (respectively) are set; the user's selection snaps every LENGTH samples.

3.4 GlBars

This is an **OpenGL** module that draws the periodogram.

Header file : GlBars.h

GlBars simply subclasses QGLWidget and its functions paintGL() and initializeGL().
The function resizeGL() has not been implemented, as GlBars does not allow to change its own size.
The Draw() function gets LENGTH samples of audio, calculates the periodogram and displays it.

Source code : GlAudio.cpp

The constructor just sets to 0 all periodogram's samples.
The initializeGL() function follows the standard steps: viewport setting, projection setup and modelview matrix setup.
The paintGL() fuction is also quite simple.
It clears the client area, draws a background and, finally, draws 32 bars, one for each periodogram's sample.
The Draw() function calculates all periodogram's samples, stores them in the private member mP[] and updates the client area, callling updateGL().

3.5 GlDataBars

This **OpenGL** module provides a graphical visualization of data separation in a bar-style graph and lets the user operate on the frequency mask.

Header file : GlDataBars.h

The FrequencyMaskChanged() signal is emitted when the user changes the frequency mask by clicking on a marker or on a bar in the graph, in order to tell TeacherWidget module to syncronize its frequency mask.
The AutoSelect() function automatically selects (if possible) the 3 periodogram's samples that give a better data separation.
The mousePressEvent() traps a left-click on a sample and toggles its bit in the frequency mask.

Source code : GlDataBars.cpp

The constructor sets the frequency mask to 0 (no samples used) and also sets to 0 all the bars in the graph.
The initializeGL() function sets viewport and projection and defines a display list to draw a marker just calling glCallList() (see paintGL()).
The paintGL() is quite simple: refer to the GlBars module for details.
Frequency mask is drawn in the lower part of the area through an array of 32 markers: each marker is grey (RGB=127,127,127) if it is not selected, and yellow (RGB=255,255,0) if it is selected.
The mousePressEvent() traps a left-click, calculates the “clicked” sample and toggles its bit in the frequency mask.
Obviously, the client area must be redrawn; the FrequencyMaskChanged() signal must also be emitted in order to comunicate the changing to the TeacherWidget module.
The Autoselect() function tries to select the 3 periodogram's samples whose data separation is higher.
It uses an iterative method: starting from threshold = 50%, it calculates how many samples produce a data separation higher than treshold; if they are more than 3, threshold is moved up, otherwise it is moved down.
This method operates subdividing the interval in 2 sub-intervals of the same length, and stops when exactly 3 samples are selected.
It can happen that some samples produce the same data separation, resulting in threshold to oscillate around this value: to avoid this, the number of iterations is limited to 100, after whom AutoSelect() returns a frequency mask which is the best approximation that can be done; the user can modify the frequency mask later.

3.6 GlDataViewer

This **OpenGL** module visualizes in a 3D space the examples (green=whistle, red=noise) and the perceptron's surfaces.

Header file : GlDataViewer.h

The DrawData() function updates examples and the perceptron according to changes done by the TeacherWidget module, then redraws the client area.
The SetFrequencies() function just changes the 3 frequencies and redraws the client area: examples are left unchanged.
Left(), Right(), ZoomIn(), ZoomOut(), Up(), Down() slots are fired by pressing the PushButtons under the Perceptron Viewer window in the TeacherWidget module; they move the camera.
Standard QGLWidget's methods are subclassed, as in the previous modules...
A camera is defined in this module. It points the origin and its position is fixed through a radius (distance from the origin) an horizontal angle (from -90° to +180°) and a vertical angle (from 0° to 90°)
The 3 axis (x,y and z) represent the 3 periodogram's samples used by the perceptron.
Examples are accessed through a pointer to the first element (mpExample) and the number of examples (mNumExamples).

Source code : GlDataViewer.cpp

The constructor resets the camera position.
The initializeGL() function sets up the viewport and prepares OpenGL for using blending.
The point size is set to 5 to make the points (examples) more visible.
The camera is placed in the correct position, then examples and surfaces are drawn.
The color of the surfaces is transprent to let the user see the examples through them.
Since this module works in a Log-Log-Log space, values smaller then 1 are clamped to 1 (0 in the Log axis)
As said before, the DrawData() function refreshes examples and perceptron.
Previously discussed slots, one for each PushButton, must redraw the viewport, as they have changed the camera's position.
The SetFrequencies() function analyzes the frequency mask and retrieves the first 3 selected frequencies, setting then as axis in the graph.
If less than 3 frequencies are selected, the graph will result reduced to a plane or a line.
If more than 3 frequencies are selected, higher frequencies are ignored.

3.7 GlMonitor

This **OpenGL** module displays the peak level read from the microphone and displays it.
The code is quite simple to understand – refer to previous OpenGL modules for details.

Header file : GlMonitor.h

This is a very simple module; Implementation is identical to the previous OpenGL modules.

Source code : GlMonitor.cpp

We can see the standard initializeGL() and paintGL() functions.
SetValue() and ResetValue() functions update the internal variables “max” and “value” and redraw the viewport.

3.8 main.cpp

This module defines the main WRTeacher's window and shows it.

Source code : main.cpp

Class WindowWidget is the main window class.
When the RecorderWidget module changes the whistle buffer or the noise buffer, the TeacherWidget module must update its own buffer poiners and sizes.
Before starting the application, WRTeacher verifies if the graphical environment has got thread support.

3.9 RecorderControls

This module provides a set of tool to let the user control the recorder:
- Monitor PushButton
- Rec PushButton
- Stop PushButton
- Play PushButton
- Reset PusButton
- MicLevel Slider

Header file : RecorderControls.h

The GetMicVolume() and ChangeVolume() functions set and get the current IGAIN level.
Play(), Rec(), Monitor(), Stop() and Reset() signals are emitted when the user presses the PushButtons.

Source code : RecorderControls.cpp

The constructor defines PhshButtons and the Slider.
The PushButtons fire some slots that controls the PushButton's steate and emit the corresponding signal.
When the user moves the Sllider, igain is changed according to the changing.
MonitorClicked(), PlayClicked(), StopClicked() and RecClicked() slots activate / deactivate the PushButtons according to the recorder's state.
The destructor just releases the allocated memory.

3.10 RecorderWidget

Defines the recorder's interface.

Heder file : RecorderWidget.h

Acting on mpTab (see sorce code – constructor), the user can change the active buffer.
All the operations (rec, play, reset) are applyed to the active buffer.
We have 3 threads that do the main recorder's operations:
- During play:
  Play thread – plays the sound
  The widget's timer (see timerEvent()) draws the periodogram through the GLBars widget.
- During rec:
  Rec threads – records from the microphone
- During Monitor:
  Monitor threads – gets 64 samples from the microphone.
  The widget's timer (see timerEvent()) draws the peak value in the GlMonitor window.

Since Play, Rec and Monitor threads are mutually exclusive, they are launched as main_trhread.
RecorderWidget is a simple POSIX implementation of the Thread Class; for instance, the play thread is launched as follows:
Play() calls StartPlayThread(), passing this as its arguent.
StartPlayThread() calls this->PlayLoop() and, after its termination, terminates the thread calling pthread_exit().
PlayLoop() writes the buffer to the SoundBlaster.

The UpdateBuffer() slot and the BufChanged() signal must be duplicated, as RecorderWidget module works on two buffers (Whistle and Noise).
For the same reason, buffers and their lengths are also duplicated.
The ChangeActiveBuffer() slot is fired when the user, acting on mpTab, changes the active buffer.
The recorder can be in Stop, Rec, Monitor ort Play status: tRecStatus contains the recorder's status.

Source code : RecorderWidget.cpp

When the Rec loop is launched, the active buffer is erased, a new buffer is allocated to contain 30 seconds of recorded audio, and it is filled whith blocks of LENGTH samples, read from the SoundBlaster, until the buffer terminates or the user presses the “Stop” PushButton in the RecorderControls module.
The PlayLoop() function is similar to the RecLoop() function.
Some blank samples are played before the sound, to fill the system's internal buffer.
The PlayLoop() function also updates the global variable gSample; the Draw loop (slower than the Play loop) calculates the periodogram using 64 samples starting from the gSample-th sample in the active buffer.
The Monitor loop does not require a perfect syncronism with the read audio: it simply reads LENGTH samples from the microphone continuously.
StartPlayThread(), StartRecThread() and StartMonitorThread() functions just start the corresponding threads.
Play(), Rec() and Monitor() slots (fired when the user presses a PushButton in the RecorderControls module) call the previous functions to start threads.
The Stop() slot blocks the execution until the main thread (play, rec or monitor) is terminated, in order to prevent the user from launching a new operation when the previous loop is not already terminated.
This module owns the Whistle and Noise buffers: it must create and delete them, and it controls the signal/slot mechanism from it and its slave modules.
SaveToFile() and LoadFromFile() slots are fired when the user presses the “Save To File” and “Load From File” PushButtons.

3.11 TeacherWidget

This module defines the lerarning widget interface and integrates a control which let the user test the perceptron before saving it.

Header file : TeacherWidget.h

The learning thread and the stop thread are launched as follows:
- The StartLearn() and StartTest() functions start the two threads, launching StartLearningThread() and StartTestThread(), respectively.
- The StartLearnThread() and StartTestThread() functions launch the LearnLoop() and TestLoop() functions, respectively, and, afters its terminations, exit from the thread calling pthread_exit().
- The LearnLoop() and TestLoop() functions do the job.

UpdateBufferWhistle() and UpdateBufferNoise() slots are fired when the recorder changes a buffer.
The Analyze() function reads the whistle buffer and the noise buffer (from the RecorderWidget module) and calculates data separation for each periodogram's sample (see Chapter 2 for details); then, it draws it in the GlDataBars window.
The AutoSelectFrequencies() function tries to select the 3 periodogram's samples that better separate the data.
The Learning loop stops when the user presses the “Stop” PushButton in the “Perceptron” section or when the perceptron completes a learning cycle on the examples without incorrect outputs.
The test loop terminates when the user presses the “Stopt” PushButton.
The ToggleDoppler() slot is fired when the user changes the sate of the “Doppler compensation” RadioButton in the “Perceptron” section.
Than we have Save/Load handling slots.
The FrequencyMaskChanged() slot is fired when the user, acting on the GlDataBars window, changes the frequency mask.
The GenerateExamples() slot reads the whistle and thw noise sounds (from the RecorderWidget module).

The UpdateTurnOnThreshold() and UpdateTurnOffThreshold slots are fired when the user, acting on the SpinBoxes in the “Perceptron” section, changes the turnon threshold or the turnoff threshold.
Buffers are accessed, as in the previous modules, through a pointer to the buffer (mpDataBufferWhistle and mpDataBufferNoise) and the buffer size (mBufSizeWhistle and mBufSizeNoise).
This module owns:
– the perceptron
– the examples
– the frequency mask
– the “Use doppler” flag

Source code : TeacherWidget.cpp

The constructor defines the widget's sections:
– The “Frequency selection” section
– The “Perceptron” section
– The “Viewer” section and its control PushButtons
– The Save / Load section.
– The Test section and its PushButtons.
The connection between the RecorderWidget and the TeacherWidget modules is done by the main.cpp source.
The Analyze() function operates in two steps.
In the first step, the function analyzes all the two buffers and finds out:
– the smaller value of each periodogram's sample produced by the whistle sound.
– the greater value of each periodogram's sample produced by the noise sound.
In the second step, the function calcluates, for each periodogram's sample, the fraction of examples external to the intersection of the whistle and noise sets.
Finally, examples are generated.
NB: When the recorder section changes one of its buffers (whistle or noise), the learning part is not syncronized until the Analyze() function is launched.
The GenerateExamples() function just translates the buffers into the example format (see Perc.h for details)
The background color of the mpTestBox widget is red if the perceptron detects no whistle and it is green when the perceptron detects the whistle.
The SavePerceptron() and the LoadPerceptron() functions just call the respective perceptron's functions, displaying the Perc's error message if they encunter problems.
When a new perceptron is loaded from a file, the LoadPerceptron() functions updates the interface in order to keep syncronization with the perceptron.

3.12 TimeSpecs

This module lets the user change the time specifics to be used during the recognition.

Header file : TimeSpecs.h

This widget subclasses Qdialog.
The SetPerceptron() function sets the perceptron used during recognition, while the SetTimes() function sets the time specifics to be used during recognition; these internal time specifics are modified through the SpinBoxes in this window and are different from perceptron's ones (the perceptron gets these time specifics when it is saved, through the GetTimes() function).
The SpinChanged() slot is fired when the user changes the value of one of the SpinBoxes.
The StartTest() and StopTest() functions start and stop the widget's timer (see TimerEvent()), which is used to simulate the robot's main timer.
This module also owns the Whistle Counter (mpCounter).

Source code : TimeSpecs.cpp

The constructor creates the dialog and resets time specifics to their standard values.
The destructor only frees allocated memory.
When the SpinChanged slot is fired, all time specifics are translated in milliseconds and shown on the right of the corresponding SpinBoxes.
The timerEvent() function simulates the main robot's timer.
Since Periodograms() function is used to calculate simultaneously the periodograms of 2 sequences, this function has two different behaviours: the first time it just reads gBuffer1, while the second time it reads gBuffer2, calculates the two periodograms and processes the two periodograms through the whistle counter mpCounter. Finally, it fills of green the corresponding TestBox when the whistle counter emits a mesage.
Then, previously discussed slots are implemented. When the “Save” PushButton is pressed, the widget hides himself and emits the accept() signal.
NB if the user presses the “Cancel” PushButton, the perceptron will keep using his old time specifics.