Bluetooth Music Player with Lighting and LCD display

Friday
Minghe Ren (sawyermh), Zhiwei Tang (zwtang), Zhiyu Wang (wangzy95)

Abstract

As there are already many smart home music player products in the market, we plan to make one ourselves with some additional interesting features. This product is mainly comprised of four features: LCD control and display panel, Bluetooth MP3 player, IR sensors to detect movements, and multiple LED-strip lighting modes. The device can get the number of people in the room and automatically switch on/off the music as well as lighting according to the presence of the people. The user is also able to control the lighting effect, choose music, change the number of people via a touch screen. The actual product achieves all the planned features and works robustly.

1. Introduction

In our project, we build a home-use Bluetooth MP3 player with some additional features. Firstly, we have a LED-strip with different light modes integrated into our system. When the MP3 is playing music, the LED-strip is also shining along with the music. Secondly, we're using IR sensors to detect and count the number of people in the room. We there is no one in this room, the MP3 will automatically shut down, or turn on when a person steps into the room firstly. Then, we have an LCD display panel to show user information like song list, lighting mode and the number of people. Also, users can use the touch screen to control the device like choosing songs, the LED lighting mode or changing people number. The motion from a user (i.e. entering or exiting the door) will trigger IR sensors and the IR sensors then send a signal back to the driver. The driver is responsible send signals to control the lighting, meanwhile, keep a record of user status. At the application level, there is an LCD process to keep reading the status of the driver. If something changes in the user level or kernel level, the process will send commands to control the Bluetooth MP3 player accordingly. The LED lighting has several lighting effect options like breathing light, flashing or simply illuminate the room. We can use buttons on the LCD screen to start/stop of the music as well as to specify the music to play. The start/stop of the music can also be controlled by the presence of people in the room. We install the madplay and aplay in our system for the MP3 decoder module and the Bluetooth module respectively.

We have three pieces of code in our project in total. One is kernel driver written in C which is responsible for receiving IR sensor signals, controlling LED, store information, read and write handling. The other one is the Qt code written in c++ which is responsible for displaying information and sending control signals to the kernel driver or the MP3 player module. The last one is a shell script which contains all the command to control the MP3 player (madplay and aplay commands).

Our motivation is from smart home MP3 player on the market like Google home, Alexa. We want to try something simple but practical to initial attempt. We come up with a scenario that would make the MP3 more fun, just like we said in the project proposal: "Imaging after a long day work or study, when you step into your house, all the sudden, the music starts to play and LED starts shining. This is your happy hour...".

We use motion control in our project which is more convenient and practical. In the future, we can also add more features like voice control, connecting and controlling the furniture. There is a lot we can do to improve our product and this device has also great potential market value.

For the result of the project, we realized all the features we planned in advance. Firstly, the MP3 decoder module and Bluetooth module work very well. The motion sensor is able to successfully detect and determine the direction of motion (coming in or getting out). The number of people will be counted automatically by IR sensors. However, due to the type of the sensor, it has a cooldown time of 2 seconds, that means there should be a 2 seconds gap for each person's movement to be detected. By using the LCD touch screen we can correctly send commands to the kernel module and display user information with 1s delay. To make the system more robust, we add reset button and we can adjust the number of people at LCD. We have around 1s delay to turn on or shut down the MP3 by motion triggering. Overall, the delay for the whole system is 1s.

2. Design Flow

Our project includes mainly four functions: music playing with Bluetooth speaker, IR sensor module, LED lighting display, and Qt LCD control. To achieve music playing function, we have madplay, aplay, and other required API installed on Gumstix. With these player software, we can connect the gumstix with a Bluetooth speaker, encode mp3 files into waves, and send the wave to the speaker. For the IR sensors and LED lighting display part, we developed a kernel module which can handler IR sensor signals with interrupts. By designing interrupt handler functions, most corresponding functions can be triggered. These functions include LED lighting display, we chose 2 general gpio and 1 gpio with PWM function as signal output. These 3 pins are connected with MOSFETs whose outputs are connected with the LED light strip. In the following paragraphs, the command and data flow from Qt GUI to hardware devices will be described to explain how our project works in detail.

The first one is mp3 music playing and playback control flow. We will use the start button as an example of the control flow. When the start button is clicked, the text of the current Item in the song list will be collected. And in the start button handler function, a command will be executed by calling system() function. The command executed is like "ash /media/card/project/scripts/music_control.sh -s /media/card/project/mp3s/" plus the song name. And in the music_control shell, madplay and aplay will be called to decode the mp3 file and send the wave to Bluetooth speaker.

1.Button clicked	2.Button handler function	3. system execute "music_control.sh"	4. madplay & aplay called	5. Bluetooth speaker starts playing

table1. music play control flow

The second is about the interaction between Qt and mygpio driver. In the mygpio driver, signals from IR sensors will be handled with Interrupts. Everytime IR sensors are triggered by an entering move, the people counting variable in mygpio driver will add one. Similarly, the leaving move will decrease people

counting variable. The Qt module will read mygpio once every one second, getting the people counting variable and displaying it on the screen. Besides the entering and leaving move, the “+”, “-”, and “reset” button on the LCD screen could also change the people counting variable. Their data flow will be explained in the next paragraph. Except for the people count variable, Qt module also reads the current display mode of LED strips. We designed 5 modes for LED to display, the current mode number will also be shown on the LCD screen. The reading operation is accomplished by setting a QTimer.

1. QTimer timeout, Timer handler function called.	2. Read current LED display mode & people number.	3. Edit the content in QLineEdit displayed on GUI.

table2. Qt reading flow

The third one is about written to mygpio device from Qt module. As mentioned in the last paragraph, the “+”, “-”, and “reset” button could make a change to the people counting variable in mygpio device. This is done by executing a command which could direct write to the device, like “echo p+ > /dev/mygpio”. In the mygpio::read function, this command will be handled and add one to the people counting variable. Similarly, “-”, “reset”, and “change” button are also implemented in this way.

1.Button clicked	2.Button handler function	3. system execute “echo p+ >/dev/mygpio”	4. In mygpio, read function parse the text	5. People counting number plus 1.

table3. People number control & mode control flow

The visualization of these control flows is shown as the picture below.

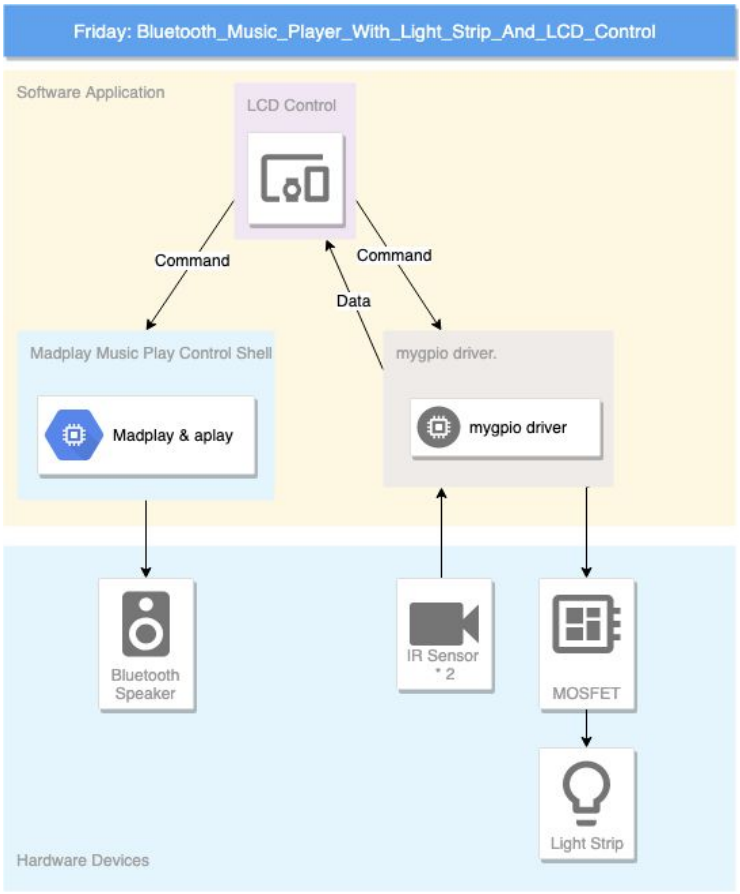

Fig 1. Control Flows Diagram

The overall contributions for this project are shown below:

MP3 decoder and Bluetooth module: Minghe Ren 33%, Zhiyu Wang 33%, Zhiwei Tang 33%;
Kernel Module: Minghe Ren;
Qt module: Zhiyu Wang;
Music control Scripts: Zhiwei Tang;

Overall percentage: Minghe Ren 33%, Zhiyu Wang 33%, Zhiwei Tang 33%;

3. Project Details (around 3-5 pages, including pictures and graphs)

a. **LED Module**

We bought a LED-strip online as the strobe lights module in our project. The LED strip equips with 3 inputs controlling 3 basic color lights in the LED-strip: red, blue, green. One problem we are facing is that to fully light up the LEDs, we need to provide 12V voltage to the LED-strip whereas the voltage output from gumstix board is not enough. To solve this problem we utilize the MOSFETs to take over the actual control of the LED-Strip [1]. On the one hand, we use the 12V power supply connecting the LED-strip. On the other hand, we connect also the control

signal generated from gumstix board's GPIO connecting the MOSFET gate pin. As a result, there are a total of 3 MOSFETs in use and each of them behaves like a switch connecting one certain LED color input on the strip.

Due to the limited number of PWM in the gumstix board, we only use one PWM signal output to control one certain color in the LED-strip (green). For the other two (red and blue), we use normal GPIO outputs.

b. LED Circuitry

If there is no voltage at the gate input, the MOSFET behaves like a very large resistor, which means that no current is flowing and the corresponding color on the LED-Strip will not lighten up. If, on the other hand, if 5V is applied, the MOSFET behaves like a very small resistor and current can flow. The PWM turns a MOSFET on and off very fast and the optical effect is that a color seems darker or brighter. Each MOSFET thus controls one of the three colors. We set 3 GPIO pins (16, 28, 29) as outputs to control GRB colors.

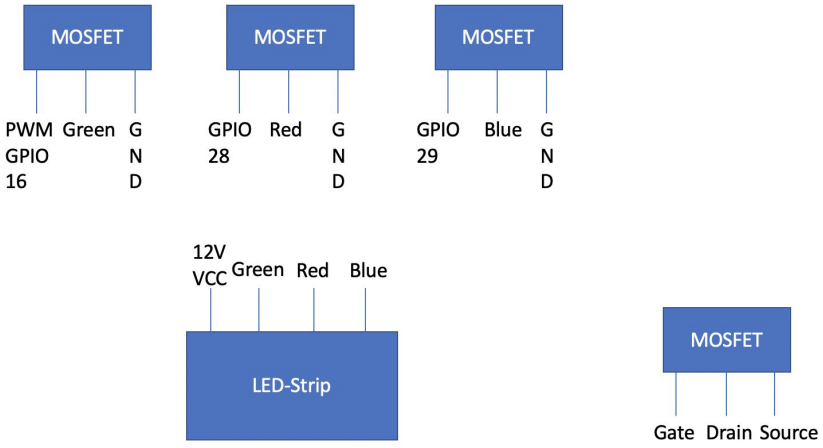

Fig 2. LED Circuitry

c. **IR Sensor Module**

We use two IR sensors to detect whether people coming in or coming out of the room. We judge these two scenarios by observing the triggering order of IR sensors. We use two GPIOs to receive the signals sent by IR sensors. If one IR sensor detects motion, it sends a high-level signal. Each GPIO input is handled by an interrupt handler and triggered by the rising edge.

One challenge we're facing is that the IR sensors we bought are very sensitive and this may cause problem when determining the direction of movement. We use the paper to wrap around IR sensors to reduce the noise.

d. **IR Sensor Circuitry**

We use the VCC provided by gumstix board as the power supply to IR sensors and two GPIO inputs to receive the triggered signals.

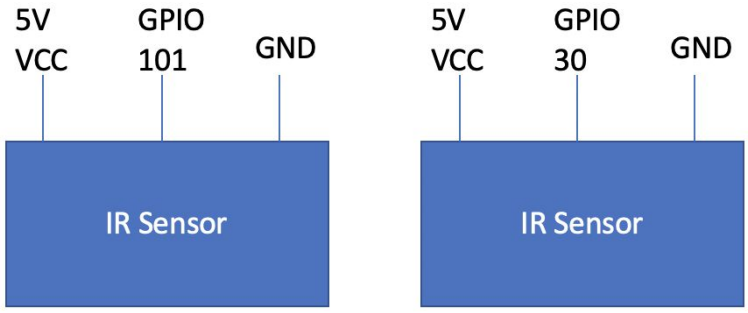

Fig 3. IR Sensor Circuitry

e. **Device Driver (Kernel Module)**

The kernel driver is responsible for LED and IR sensor modules. The LED module has different lighting mode and the IR sensor is used to keep a record of people in the room. By using IR sensors, when there is someone firstly coming into the room, the MP3 starts to play automatically and turns off when no one in the room. All the functions defined in the kernel module are shown below:

```
/* Declaration of functions */
static int mygpio_open(struct inode *inode, struct file *filp);
static int mygpio_release(struct inode *inode, struct file *filp);
static ssize_t mygpio_read(struct file *filp, char *buf, size_t count, loff_t *f_pos);
static ssize_t mygpio_write(struct file *filp, const char *buf, size_t count, loff_t *f_pos);
static void mygpio_exit(void);
static int mygpio_init(void);
void _TimerHandler(unsigned long data);
void _TimerHandler1(unsigned long data);
irqreturn_t gpio_irq0(int irq, void *dev_id, struct pt_regs *regs);
irqreturn_t gpio_irq1(int irq, void *dev_id, struct pt_regs *regs);
```

Fig 4. Driver functions

In the kernel module, we use a periodic timer to keep everything updated. Upon each timer interval, we check the status of play mode and number of people. The different play modes stand for different LED lighting patterns. Mode zero is to turn off the LED. Mode one is the breathing light with utilizing the PWM. Mode two and three are strobe lights with slow and fast speed respectively. Mode four is simply white light. This is defined in “TimerHandler” function in mykernel.c.

There are two GPIO interrupt handler functions: gpio_irq0, gpio_irq1. They are responsible to handle the signals sent by IR sensors. When either of them is triggered by a rising-edge signal, they will check whether the other one is triggered to determine the direction of movement. Also, a new timer function called “TimerHandler1” is set to step in. The aim of this new timer handler is to change the value of a global variable named “numofpeople”, which keeps a record of people in the room. This will only happen when both IR sensors are triggered inside the new time handler and the timer waiting time is set to 1.5s. When someone comes in, this value will add one or minus one otherwise.

In the mygpio_write function, user can send commands to the kernel module in order to change the play mode of our MP3 player, add or decrease the number of people, and reset the system to initial state. In the mygpio_read function, the user can acquire information such as current play mode and number of people in the room from the kernel.

f. **Qt Module**

For the Qt module for this project, specifically, there are mainly three code files except for the Makefile. In this section, the implementation detail about these three files: main.cpp, Friday_gui.cpp, and Friday_gui. will be described. Also to show the specific idea, pieces of code will be displayed to help explain the meaning of the code.

1) main.cpp

The main.cpp file could be found at the following path in GitHub repository: EC535_project/ project_qt/main.cpp. In this file, all required libraries are included here, mainly the libraries for QT widgets. And what the main function does is creating an instance of the customized class Friday_gui which is named “window”, and showing it on the LCD screen. The reason why the main function should be kept simple is no one want the GUI frozen when some function is running. Designing all functions in the Friday_gui class will avoid the showing process being interrupted by other threads in the main function.

EC535_project/ project_qt/main.cpp

```
int main(int argc, char *argv[]){
      QApplication app(argc, argv);
      Friday_gui window;
      window.showFullScreen();
      window.show();

      return app.exec();
}
```

Fig 5. main function

2) Friday_gui.h

The Friday_gui.h file could also be found at the same path with main.cpp. This is a header file, which consists of header file definition check, required libraries, and also the definition of Friday_gui class.
The first thing about the Friday_gui class is it is inherited from QWidget class. Usually when a customized class is defined for Qt Application, a QWidget class is a good parent class to inherit from. Different from normal C++ class define, a Qt class includes public, private and private slots members. In the public member, only the constructor of Friday_gui class is defined. For private members, the following components are used.

- 1 QTimer used to call update_handler() function in a period of time.
- 1 QGridLayout used to place all components in place.
- 1 QListWidget used to display the music in store.
- 1 QStringList used to keep the songs read from SD card.
- Playback group: 4 QPushButtons to accomplish start, play/pause, next song, and prev song functions. 1 QLineEdit to display the current playing music.
- Lighting mode group: 1 QPushButton to change mode. 1 QLineEdit used to display current lighting mode.
- People counting group: 3 QPushButtons to add, minus, and reset the number of people in the room. 1 QLineEdit to display the current number of people.
- variables:
    - state, type int, used to track current music playing state.
        - state == 0: means no song is playing.
        - state == 1: means some song is playing now.
        - state == 2: means some song was playing, but it is paused now.
    - mode, type int, used to track current lighting mode. Totally, 4 types are provided.
    - csong, type int, current song index, used to track the index in the song list. The value represents the index of the currently playing song in the song list.
    - dirName, type QString, hardcoded directory to the music folder.
    - dir, type QDir, the directory variable.

EC535_project/ project_qt/Friday_gui.cpp
Friday_gui class:

```
private:
      QTimer* timer;
      QGridLayout* layout;
      QListWidget* songs;
      QStringList* songList;
      QPushButton* play;
      QPushButton* start;
      QPushButton* next;
      QPushButton* prev;
      QLineEdit* nowplaying;
```

```
    QLineEdit* cmode;
    QPushButton* change;

    QLineEdit* cpeople;
    QPushButton* reset;
    QPushButton* add;
    QPushButton* minus;

    // Variables for song name retrieval and script calls
    int state, mode, csong;
    QString dirName;
    QDir dir;
```

Fig 6. Friday_gui class private members

For the members of the private slots, all the handler functions are declared here. Except for the timer_update() function, the other functions are all handler function of buttons.

EC535_project/ project_qt/Friday_gui.cpp
Friday_gui class:

```
private slots:
    void play_handler();
    void pause_handler();
    void start_handler();
    void stop_handler();
    void next_handler();
    void prev_handler();
    void change_handler();
    void reset_handler();
    void add_handler();
    void minus_handler();
    void timer_update();
```

Fig 7. Friday_gui class private slots functions.

3) Friday_gui.cpp

The Friday_gui.cpp file could also be found at the same path with main.cpp. This is the most important file for this music play and light display Qt module. This cpp file consists of the constructor of Friday_gui class and the definition of all handler functions. Their functions will be explained in this section.

i. **Friday_gui constructor**
   1. **Reading songs from the directory, filling the QListWidget with songs read from the target folder.**
   2. **Creating objects for QPushButtons, QLineEdits, and QTimer. Linking the buttons with handler functions defined later.**
   3. **Designing the layout of GUI by putting objects into the grid layout object.**
   4. **Assigning an original value for tracking variables, like state, mode, csong, and uiplay.**

ii. **Handler functions definition.**
   1. **void Friday_gui::timer_update()**

      The timer_update() function will be called every one second, which is the period set for timer object. It reads people count and current light display mode from the mygpio device. According to the count number and lighting mode, the corresponding QLineEdits will also be modified to show current values.

   2. **void Friday_gui::change_handler()**

      The change_handler() function will be called when the change button is clicked. In the change_handler function, mode value will add one and also modulo with the size of modes. Since in our design, there are a total of 5 modes, mode value will be limited into from 0 to 4. Besides, the current mode value will also be written into the mygpio device with the format “echo m*”. * represents the mode value.

   3. **void Friday_gui::reset_handler()**

      The reset_handler() function will be called when the reset button is clicked. What the reset_handler() function does is sending the reset command to mygpio device, and when mygpio kernel module receive the command, it will reset both mode value and people count value.

   4. **void Friday_gui::add_handler()**

      The add_handler() function will be called when the “+” button is clicked. It is used to add 1 to the total people count, by sending the add command to mygpio device.

   5. **void Friday_gui::minus_handler()**

      The minus_handler() function will be called when the “-” button is clicked. It is used to minus 1 to the total people count, by sending the minus command to mygpio device.

   6. **void Friday_gui::next_handler()**

The next_handler() function will be called when the “next” button is clicked. This is part of playback function of music player. In this function, the csong index will be added 1, and nowplaying QLineEdit will be changed to the current song name. Also based on the current state of music player, corresponding music playback handlers will be called to achieve function. For example, if the current state equals to 1, means some song is playing now, Stop_handler() will be called to stop the music playing. And the next song will be selected and set as CurrentItem. After that, the start_handler() function will start playing the currently selected song.

7. **void Friday_gui::prev_handler()**

   The prev_handler() function will be called when the “prev” button is clicked. Similar to next handler, the logic of music transforming is the same, the only thing different is csong index will minus 1, and if it is less than 0, it will be set to the last index in songList.

8. **void Friday_gui::minus_handler()**

   The minus_handler() function will be called when the “-” button is clicked. It is used to minus 1 to the total people count, by sending the minus command to mygpio device.

9. **void Friday_gui::play_handler()**

   The play_handler() function will be called when the “play/pause” button is clicked. When the current state is 0, the text on the play button will be “Play”, by clicking it, the current item in Song list will be played by executing command “ash music_control.sh”, this will call a bash file with arguments. Also, the play button’s text will be set to “pause”. Then when the current state is 1, the text on the play button will be “pause”, by clicking it, current playing song will be paused and its text will be set as “play”.

10. **void Friday_gui::start_handler()**

    The start_handler() function will be called when the “start” button is clicked. Its function is ro get the current Item from Song List, convert its content to string and play it with madplay module. Also since the song is playing after the operation, the text on the play button should be set to “pause”. If there is already some song playing, this function will stop current playing first, and start another one by calling itself again.

11. **void Friday_gui::stop_handler()**

No button is connected with stop_handler() function. It is only called by other playback function to stop the current playing song. Also after calling stop_handler() function, the state will be set to 0, which means no song is playing now.

g. **MADplay mp3 decoder**

MAD is a high-quality MPEG audio decoder, it can decode mp3 format decoded file into 24-bit PCM format to be able to feed into the ALSA for the speaker to play. It also provides the command MAD play provides an API so that we can customize it to build our mp3 player.
In our implementation, we made a shell file and executed by according handlers. This means when we press certain button on the GUI on the LCD display, the handler will pass two arguments to the shell script, the first argument chooses the operating mode(i.e. start, play, pause and stop operations), the second argument specifies the music file to play, the shell file will perform the decoding and music playing operations accordingly.
In the shell file, it switches operation modes and plays a certain music file after receiving different arguments from the handler. Figure 8 shows the handler for controlling the music play operating mode. Figure 9 shows part of the shell script which does the music start/stop operations.

```
void gui::play_handler()
{
        // If  a song is paused, play it and set state to playing
        if(state == 2 || state == 0)
        {
                int status = system("ash /media/card/project/scripts/music_control.sh -g");
                state = 1;
                uiplay = 1;
                play->setText("pause");
        }else if(state == 1){
                int status = system("ash /media/card/project/scripts/music_control.sh -p");
                state = 2;
                uiplay = 0;
                play->setText("Play");
        }
}
```

Figure 8. Music playing mode handler

```
case "$1" in
        # Case: Start a new song
        -s)
        echo start $2
        madplay $2 -r 44100 --output=wave:- | aplay -D d80 & > output
        ;;

        # Case: Stop the current song
        -k)
        echo stop $2
        killall madplay
        ;;
```

Figure 9. Music decode and play control

4. Summary

In this project, we successfully build a smart home music player with music playing, lighting control as well as people sensing and counting features. The device can be fully controlled by an LCD screen and the final product can perform its functions robustly. The remaining challenges may be adding more features to make it more competitive in the market. Possible features may be voice control to make it more convenient to be used, we can also add face recognition function to help with the home safety, we can also make it be able to connect to the internet to download the new song as well as news or weather information. In order to realize these features, we may need to improve the user interface on the LCD screen and also configurations of the internet connection. For the marketability, the product can be used by college students to control their decoration lights in the dorm and also play songs for leisure time. The people sensing and counting feature can help save energy when nobody is in the room and the people number information is useful for music volume control, lighting and heating adjustments. The device is also user-friendly as the user interface on the LCD screen clearly shows the music playing status, lighting pattern and people number information, all the functions can also be controlled by the touch screen.

References

[1] How to control a RGB LED-Strip with a Raspberry Pi. https://dordnung.de/raspberrypi-ledstrip/
[2] Basic Bluetooth Audio with LCD Controls. https://github.com/saribe0/EC535-Basic-Bluetooth-Audio-with-LCD-Controls
[3] Qt Documentation Archives, Qt Examples. https://doc.qt.io/archives/qt-4.8/all-examples.html
[4] What's the purpose of QWidget's parent? https://stackoverflow.com/questions/8798313/what-is-the-purpose-of-qwidgets-parent