University of Liverpool

ELEC362 AS2 report

Author: Yifan Wei
ID: 201376878
Department: Department of Electrical Engineering and Electronics

December 23, 2019

Contents

1 How the program works

The requirement of this project was to implement a Qt-based GUI app which could allow the user to read an graph image from the file then get some points to calculate the point value based on the axis information.

The main function of this project could be sum up as the following functions:

1. Load an image from the file browser
2. Get the information of axis
3. User select point need to calculate/ App selected points automatically and calculate
4. Export file with the information of those points

This is the state graph designed for implementing those requirements.

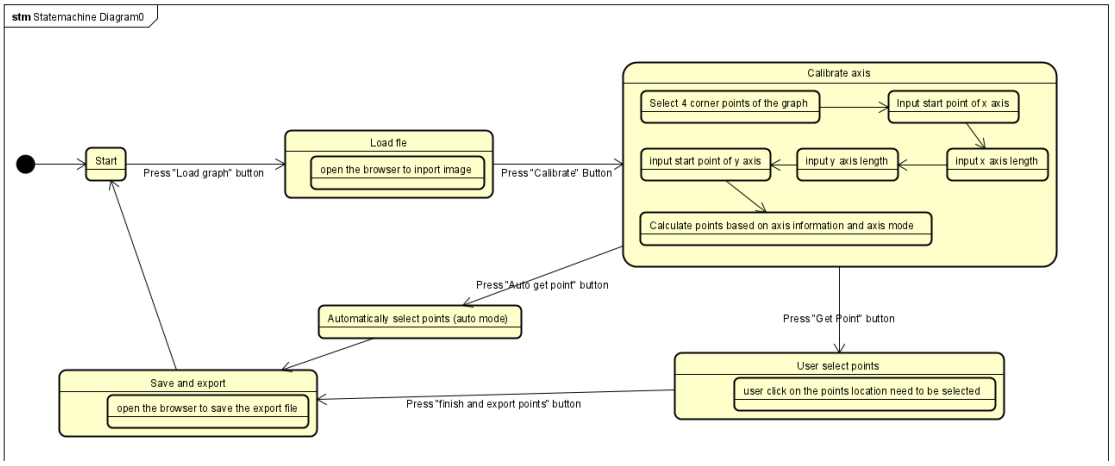

Figure 1: This is the state graph of the app

1.1 App main structure

The project app is based on a Qdialog object which could display a window to the screen and every button and rendering things are based on this project.

1.2 File load

The image rendered by Qimage which support pixel level operation. When pressing the button load the file load will be opened and then after open the file, it would be passed to a Qimage object and rendered in the QDialog object.

1.3 Calibrating axis

The graph showed in the project instruction is as follow: To calibrate, the axis

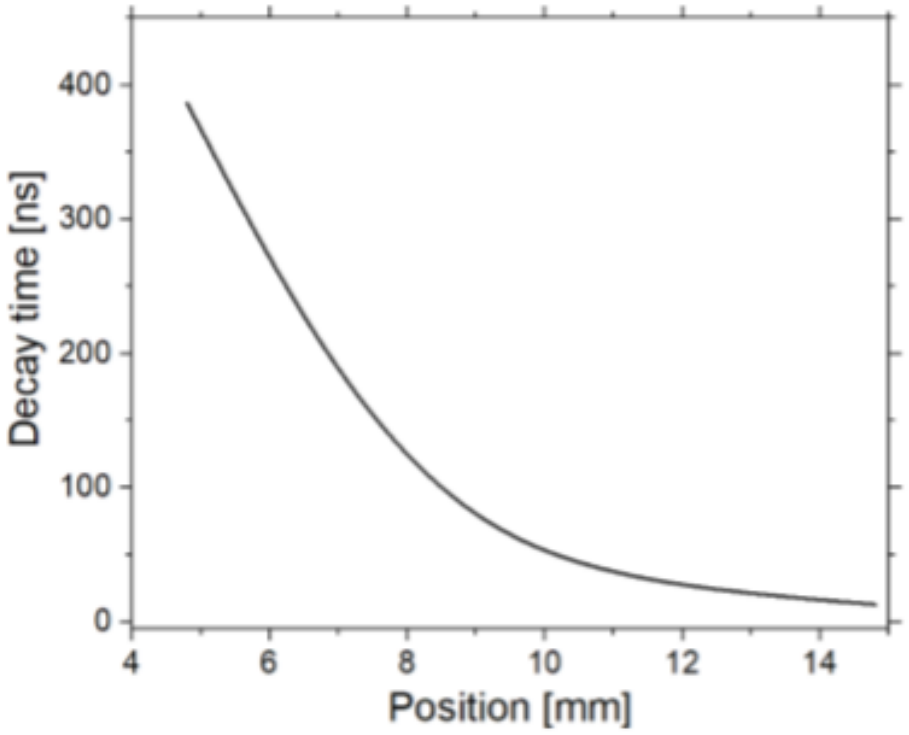

Figure 2: Graph example

unit length is required for calculating the point real number. The user needs to input the segment length of the axis first. From the example graph the axis may not start with 0 so the program will open a small QDialog window to let the user to type in the start point of the graph.

After that the program will get all the information needed for mapping and locating points from QDialog coordinate to the graph coordinate.

Then the selecting of the axis mode is also needed for the final calculation. Linear and log axis could be selected by three methods:

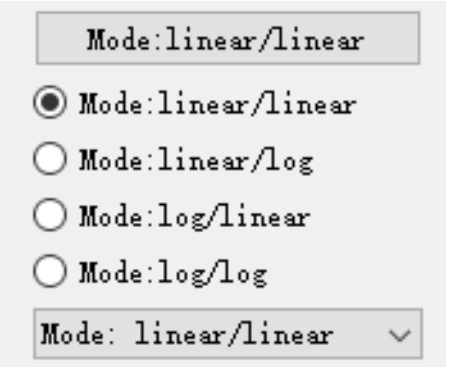

Figure 3: Mode selecting

1. First is supported by push button which could change the axis mode of x and y successively: Linear-Linear-¿Linear-Log-¿Log-Linear-¿Log-Log.

2. Second are 4 Qridiobutton with these 4 modes could be selected by the user. (The reason of the design to select 4 peak points of the graph area will be described in the following paragraph)

3. Third is a QComble button including these 4 modes could be selected by the user by clicking on it.

All those three-mode selecting methods button will be refresh to the right state by the flag variable to store the current mode when the user change any of them.

1.4 calculate points

The calculation based on the data acquired by calibrating. after calibration, the system gets 4 peak points of the graph. the system coordinate is from upper left corner of the window but the graph starts from the lower left corner so they need to fit each other during the calculation. For any points on x the coordinate is same which could be directly calculated by the distance between two points divided by convert ratio calculated by windows local position of mouse click and user inputted segment length in the graph. but for y, the final value will be the max value of axis subtract the value calculated as same methods of x. Moreover, to deal with the starting point of each axis could not be 0 then add those values to the initial value of these axis. The basic calculation could be described as the following picture:

after that, those values will be calculated by the mode's selection of the axis. For log axis, the value will be calculated by the power of 10. After that, the final

```
x_value=starting_value_X_+ x_value/calibration_factor_x_;//calculate point valu
y_value=(starting_value_Y_+legend_length_Y_)- y_value/calibration_factor_y_;//c
```

Figure 4: Calculate point

values will be saved in two vectors for further used.

1.5 Export file

The final value of X and Y was saved in two vectors points_x_ and points_y_ from the early calculation steps. When the user clicked save and export file the calculated point information will be saved in a txt file with user selected path and name.

1.6 Auto mode

The auto mode is the requirement of Task2 which could make the program to get points automatically. My method to implement that was based on pixel calculations. The calibrating step to get the 4 points of edge peaks of the curve area is the preparation of this step. The user could input the point number n she would like to get. then x axis will be divided to n+1 segment then used loop to check every pixel with lower sum of RGB which could be black then get the point for the further calculation same as user selected points. To divide x is because for every function there is only one y value for each x so when find the black point for the x value, other pixels will no need to be checked.

1.7 User friendly

To make it more user friendly, Qpainter was used to paint circle on the user click point for calibrating to make it easy to remember which points are be calibrated.

2 User instructions

The using process could be described as the following list:

1. Run the App
2. Click on “Load graph button” to Load the graph from the file browser

3. Click on “Calibrate” to enter calibration mode.
4. Selecting the four corner points of the curve area for calibrating.
5. Followed the instructions on the windows to enter the length of the graph area and the starting value for x and y axis.
6. Selecting mode based on the graph type by clicking the buttons for mode.
7. Click on “Get Point” button then click on the point would like to get value on the graph or click on “Auto get point” then enter the point number would like to get then automatically get those points.
8. Click “finish and export points” to export points information got to the txt file.(auto mode no need to do this step)

Those process could also be seen in the state graph.

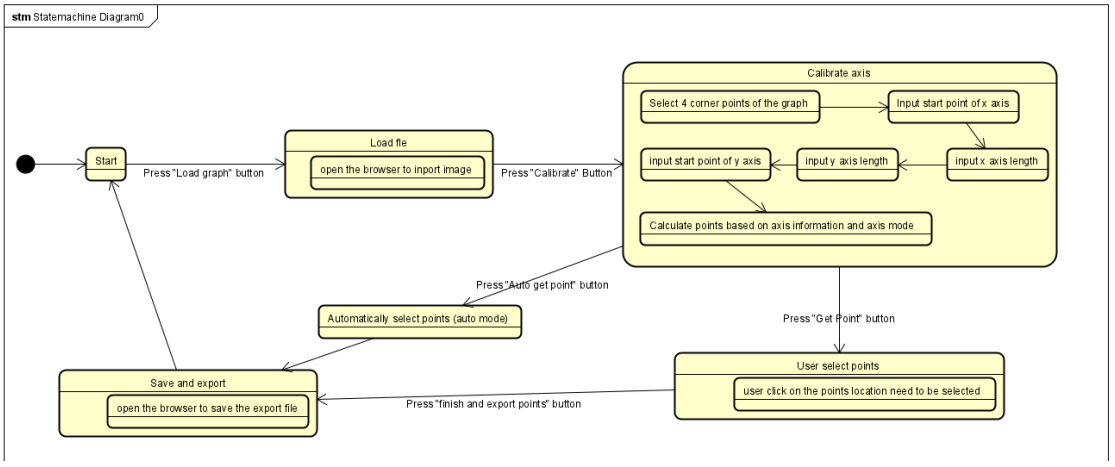

Figure 5: User state

3 Testing and verification attempts

3.1 Testing

1. Open the file, only load file button is available as setting. (Figure:6)
2. Select load image; the system opens the browser for user to open File. (Figure:7)

3. when the image success loaded, calibration button is available. (Figure:8)
4. When clicked on calibrating, the window guides the user to input the axis information (x y length and starting point). (Figure:9)
5. Then get point and auto mode are available. (Figure:10)
6. use get point then click any point on the image, the system will get the point then display it on the LCD object. (Figure:11)
7. select save and export points the browser will open for user to save File. (Figure:12)
8. check the saved file; it includes the point location with first column x and second column y. (Figure:13)
9. Testing the auto mode it will ask user the point number the user would like to get, input 10. (Figure:14)
10. Check the output file, if there are 10 points selected correctly. (Figure:15)
11. Set up another image by excel with log to check the system then comparing the value got by the system with initial value, (Figure:16)

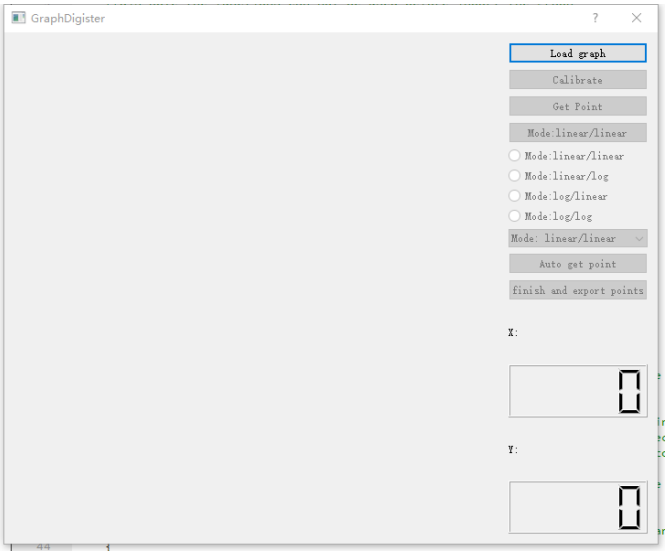

Figure 6:

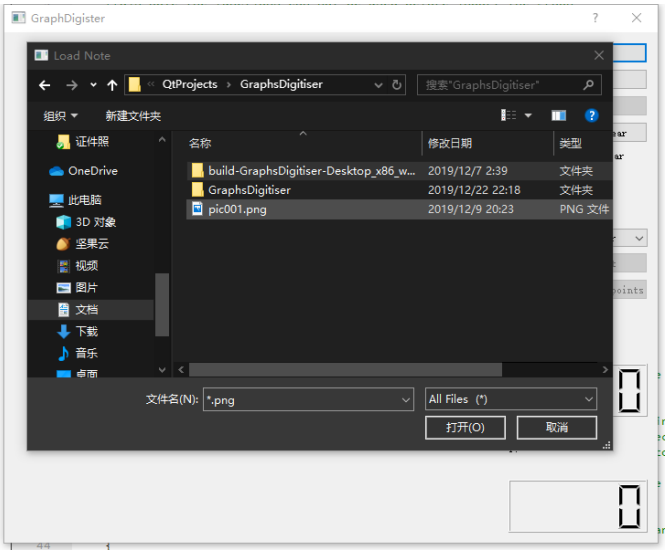

Figure 7:

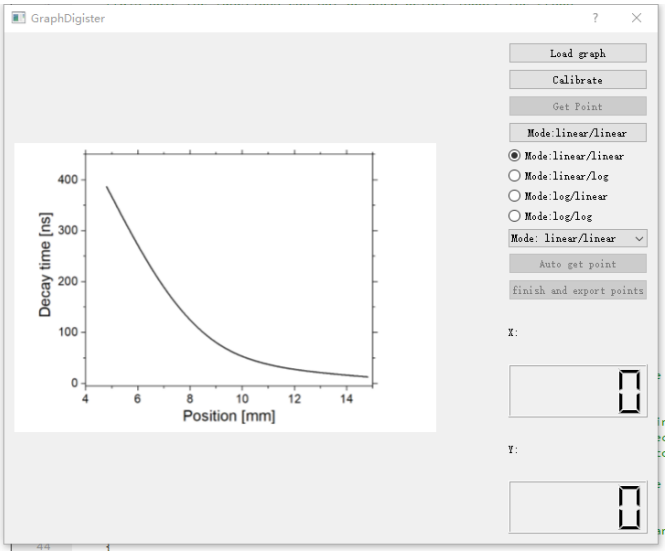

Figure 8:

4 Overall notes

There are several special features in my design, this paragraph will describe them in detail:

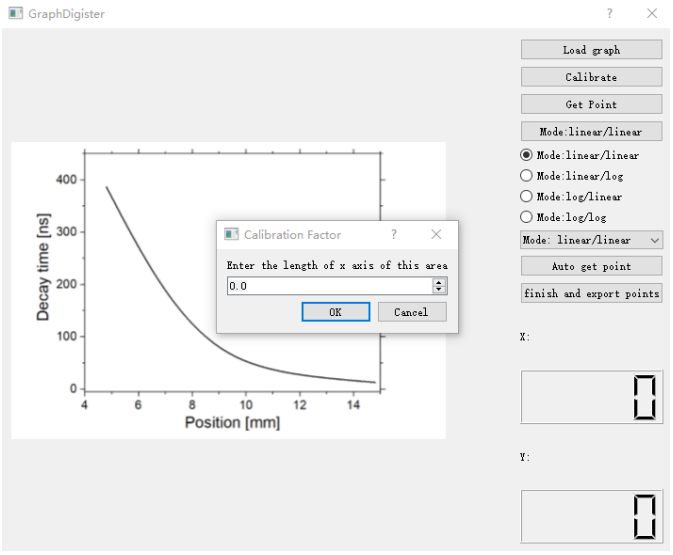

Figure 9:

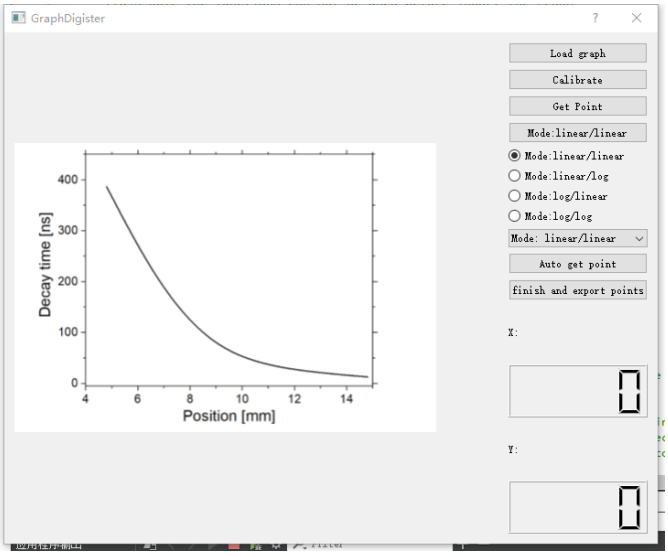

Figure 10:

4.1 4 Points Calibrating

The calibration let the user to select 4 edge point because the 4 point will get the area of the whole graph which could be used to define the selecting area in auto mode for task 2 because from adding the x and y number of the 4 points, the biggest and least point will be get for the biggest sum and least sum which define the range of the image area. Moreover such 4 points could locate the length of

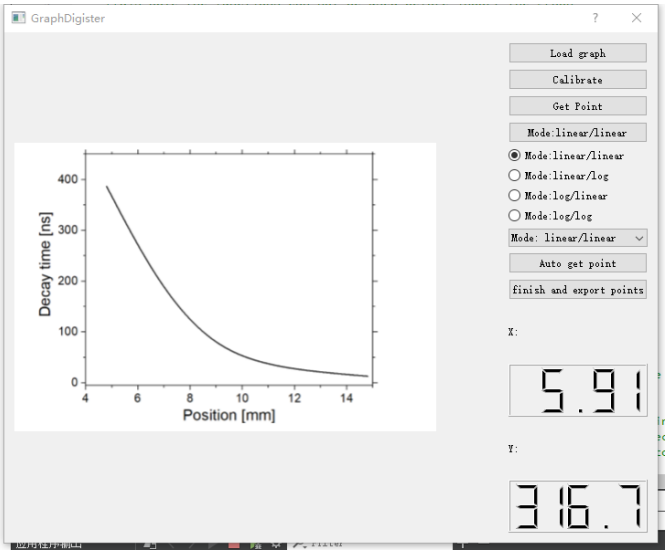

Figure 11:

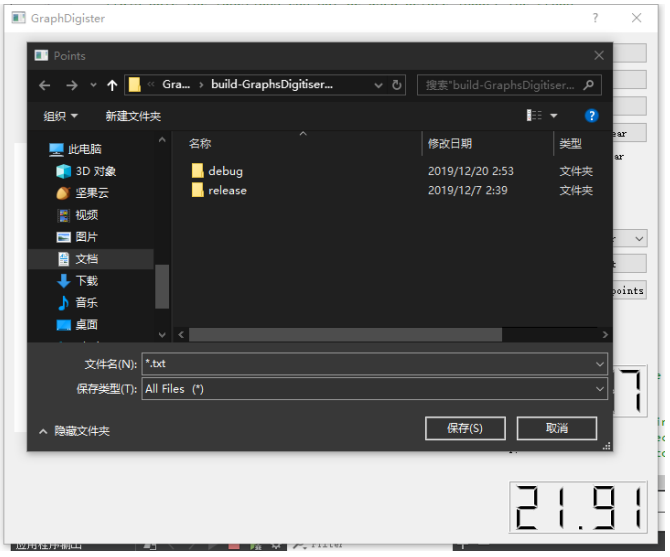

Figure 12:

axis which will be used in calibrating. For the user is easier to select 4 edge point than get randomly 2 point on each area.

4.2 Uable buttons

My code through unable button to avoid potential bugs. For example, when the user first opens the app, only load image button is available because without image

Figure 13:

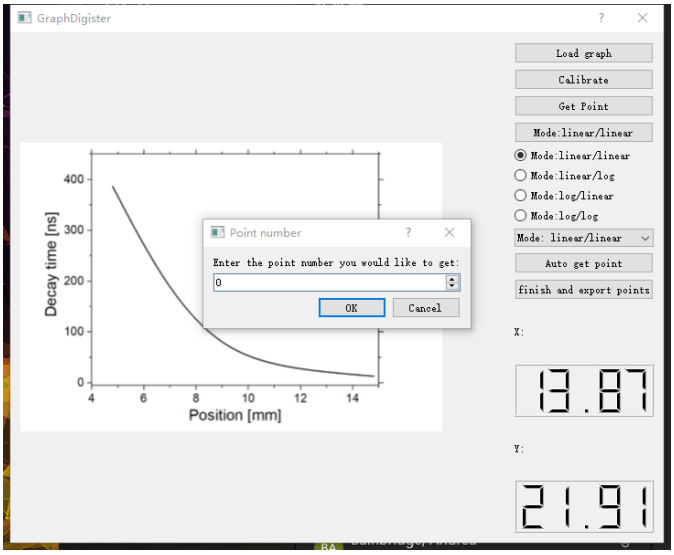

Figure 14:

every other operation are mean less. Those setting will make the user more tend to operation in a right sequence.

4.3 Viriable protect

Sometimes the user may go wrong input or forget to input any variable in mistake. the system will initial all the variable as a default value to prevent stack overflow caused by null pointer.

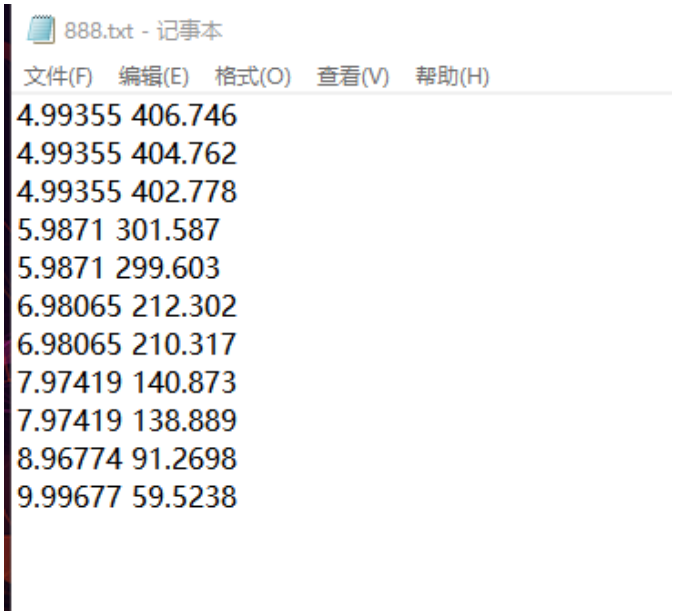

Figure 15:

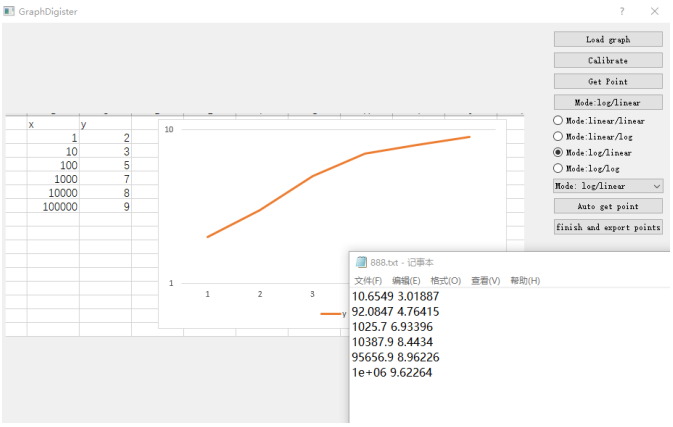

Figure 16:

4.4 LCD object setting

There are two LCD object in the windows to display the value of current point to make it more clearly and user friendly.

4.5 Referesh

When export the points information, all the variable will be set to default value for the import of next image

5 Appendix

```
1  //graphicsdigister.h
2  #ifndef GRAPHDIGITISER_H
3  #define GRAPHDIGITISER_H
4
5  #include <QDialog>
6  #include <QGraphicsScene>
7  #include <QFileDialog>
8  #include <QGraphicsPixmapItem>
9  #include <QImage>
10 #include <QMouseEvent>
11 #include <QVector>
12 #include <QPointF>
13 #include <QInputDialog>
14 #include <QtMath>
15 #include <QDebug>
16 #include <QMessageBox>
17 #include <algorithm>
18 #include <math.h>
19 #include <QPainter>
20
21 //definate a struc for saving point and save it
22 struct MyPoint {
23     double value;
24     QPointF point;
25 };
26
27 namespace Ui {
28 class GraphsDigitiser;
29 }
30
31 class GraphsDigitiser : public QDialog
32 {
33     Q_OBJECT
34
35 public:
36     explicit GraphsDigitiser(QWidget *parent = nullptr);
37     ~GraphsDigitiser();
38
39     void mousePressEvent(QMouseEvent *event);
40
```

```
41      void CalibrationCheck(); // Function to check if the 4 point
            calibration is finished or not
42
43      double CalculatePointX(QVector<QPointF>& clickedPoints); //
            Function to calculate the overall distance between all points
            in a vector
44      double CalculatePointY(QVector<QPointF>& clickedPoints); //
            Function to calculate the overall distance between all points
            in a vector
45
46
47  private slots:
48      //functions trigger by click
49      void on_Load_file_clicked();
50
51      void on_Mode_select_clicked();
52
53      void on_Mode_clicked();
54
55      void on_Mode_2_clicked();
56
57      void on_Mode_3_clicked();
58
59      void on_Mode_4_clicked();
60
61      void on_Export_clicked();
62
63      void on_Calibrate_clicked();
64
65      void on_Get_point_clicked();
66
67      void on_Mode_select_list_activated(const QString &arg1);
68
69      void on_Auto_get_point_clicked();
70
71  private:
72      Ui::GraphsDigitiser *ui;
73      QGraphicsScene *scene=new QGraphicsScene();
74
75      //clear objects and prepare for the next graph digitiser
76      void RefreshValues();//flush the variables saving points
77      void ResetButtons();//reset th button status preparing for the
            next use
```

```
78
79      //functions and variables used for calibrating
80      QPointF min_point_;  // saving the zero point
81      QPointF max_point_;  // saving max point used to know the range
            of graph
82      QVector<MyPoint> calibration_vector_;  // Vector for saving the 4
             calibration points
83      MyPoint my_point_tmp_;//tempary variable to save point into the
            QVector<MyPoint> CalibrationVector
84      bool is_calibrating=false; // flag to indicate whether in
            calibrtion mode or not
85      double legend_length_Y_=0; // Varaible to save the physical
            length of the legend on the picture
86      double legend_length_X_=0; // Varaible to save the physical
            length of the legend on the picture
87      double starting_value_X_=0; //save the starting value in zero
            point of x axis
88      double starting_value_Y_=0; //save the starting value in zero
            point of y axis
89      double calibration_factor_x_=0; // Varaible to save the
            calibration factor of x
90      double calibration_factor_y_=0; // Varaible to save the
            calibration factor of y
91
92      //functions and variables for selecting and calculating points
93      QVector<QPointF> points_vector_; // Vector for saving the points
            selected
94      bool is_selecting_=false; // flag to indicate whether the app is
            in distance measuring mode or not
95      double CalculateDistance(QVector<QPointF>& clickedPoints);//used
            to calculate the distance of two point;
96      void CalculateAndDisplayPoints();//to calculate points and save
            the value
97      std::vector<double> points_x_;//save the final value of x values
            of points
98      std::vector<double> points_y_;//save the final value of y values
            of points
99      void SavePoints();//save the information of points
100
101     //These variables and functions are used to changing image mode
102     int mode_flag_=1; //this flag used to save the current graphic
            calculate mode(liner or log)
```

```
103     void RefreshAxisModeChoosingPushButton();//this is to check the
            current graph mode setting(liner/log) and change the text on
            the push button
104     void RefreshAxisModeChoosingRadioButton();//this is to check the
            current graph mode setting(liner/log) and change the text on
            the radio button
105     void RefreshAxisModeChoosingComboBox();//this is to check the
            current graph mode setting(liner/log) and change the text on
            the ComboBox
106
107     //These fuctions and variables will be used in Auto maticially
            getting points for Task 2
108     void AutoSelectPoint();
109     double segment_x_;//the segment length which devide the graph to
            many equal divide
110     double segment_y_;//the segment length which devide the graph to
            many equal divide
111     int auto_points_num_=10;//the point number auto mode will  g e t
            default number is 10
112     QImage image_;//the image in this variable will be checked in
            auto mode
113     QPointF tmpPoint_;//this point is a tmp point used to save points
114
115     //These functions are used to make the program more user friendly
116     void paintEvent(QPaintEvent *); // Implements the drawing events
            which used to calibrate
117     QColor *pcolor=new QColor(); // Variable to hold hte colour
118     QPainter *paint;
119     //QPoint point_location_;
120 };
121
122 #endif // GraphsDigitiser_H
123
124 //graphicsdigister.cpp
125
126 #include "graphsdigitiser.h"
127 #include "ui_graphsdigitiser.h"
128
129 GraphsDigitiser::GraphsDigitiser(QWidget *parent) : QDialog(parent),
        ui(new Ui::GraphsDigitiser)
130 {  // Called once at the start of the application
131     ui->setupUi(this);
```

```
132     //ui->graphicsView->setScene(scene); // connecting the Scene to
            the View
133
134     //dis able the functions can not be used before inport the graph
135     ui->Calibrate->setEnabled(false);
136     ui->Get_point->setEnabled(false);
137     ui->Auto_get_point->setEnabled(false);
138     ui->Export->setEnabled(false);
139     ui->Mode->setEnabled(false);
140     ui->Mode_2->setEnabled(false);
141     ui->Mode_3->setEnabled(false);
142     ui->Mode_4->setEnabled(false);
143     ui->Mode_select->setEnabled(false);
144     ui->Mode_select_list->setEnabled(false);
145 }
146
147 GraphsDigitiser::~GraphsDigitiser()
148 { // Called once at the end of the application
149
150     delete ui;//distruct ui to free the momery
151
152     delete scene;//distruct scene to free the momery
153
154 }
155
156 void GraphsDigitiser::mousePressEvent(QMouseEvent *event)
157 {   // Called whenever there is a mouse click on the application
158     if(is_calibrating) // This part of code is executed in the
            calibration phase only
159     {
160         //get points
161         my_point_tmp_.point=event->pos();//get the point location
                from the ui window
162         my_point_tmp_.value=my_point_tmp_.point.x()+my_point_tmp_.
                point.y();//record the value for each point X+Y used for
                next sorting to define the location
163         calibration_vector_.push_back(my_point_tmp_);  // Add the
                point object to the calibration vector
164
165         CalibrationCheck(); // A function to verify that the
                calibration is done and to calculate the Lengend's lengths
166     }
167
```

```
168     if(is_selecting_) // This part of code is executed in the
            selecting points and calculating step
169     {
170
171         points_vector_.push_back(event->pos()); // Add the point
                object to the points saving
172
173         // To measure point including with the mode and information
                of axis
174         if (points_vector_.size()>0) {
175             //if there are points saved do the following operation
176             CalculateAndDisplayPoints();//calculate the reality value
                    of the selected point
177         }
178     }
179     //point_location_=event->pos();
180 }
181
182 void GraphsDigitiser::CalculateAndDisplayPoints(){
183     double x_value=0; // A variable to record x value
184     double y_value=0; // A variable to record y value
185
186     x_value=points_vector_.back().x()-min_point_.x();//calculate the
            physical x distance from the point to the zero point
187     y_value=points_vector_.back().y()-min_point_.y();//calculate the
            physical y distance from the point to the zero point
188
189     x_value=starting_value_X_+ x_value/calibration_factor_x_;//
            calculate point value based on coordinates
190     y_value=(starting_value_Y_+legend_length_Y_)- y_value/
            calibration_factor_y_;//calculate point value based on
            coordinates, the local coordinate is different to the graph
            for y so there needs to be subtract from the max value
191
192     //calculate the true value based on mode selection
193     if(mode_flag_==2){
194         y_value=pow(10, y_value);//calculate y as log mode
195     }
196     else if(mode_flag_==3){
197         x_value=pow(10, x_value);//calculate x as log mode
198     }
199     else if(mode_flag_==4){
200         x_value=pow(10, x_value);//calculate x as log mode
```

```
201             y_value=pow(10, y_value);//calculate y as log mode
202         }
203
204         //display points
205         ui->lcdNumber->display(x_value); // Displaying x value to the LCD
206         ui->lcdNumber_2->display(y_value); // Displaying y value to the
                LCD
207         points_x_.push_back(x_value);//save the x value of such point
                into the vector
208         points_y_.push_back(y_value);//save the y value of such point
                into the vector
209
210     }
211
212     void GraphsDigitiser::SavePoints()
213     {
214         // Opening the save file dialog:
215         QString filename=QFileDialog::getSaveFileName(this,tr("Points"),
                tr("*.txt"));       //print the filename and format
216
217         // Creating the file to be saved (and setting it in the write
                mode):
218         QFile file(filename);
219
220         file.open(QIODevice::WriteOnly);
221
222         // "Flushing" the text into a saved file:
223         QTextStream out(&file); // to prevent "copies" of our file
224
225         //out <<"Points Information";//used for out put debug
226
227         for(unsigned int i=0;i<points_x_.size();i++){
228             out << points_x_[i]<< " " <<points_y_[i]<< "\n";//out put the
                    points information x for the first row and y for the
                    second row
229         }
230
231         //Close the file in the end.
232         file.close();
233     }
234
235     void GraphsDigitiser::CalibrationCheck()
236     {
```

```
237     if (calibration_vector_.size()==4) // We need only 4 points to
            finish calibration
238
239     {   is_calibrating=false; // Switching off calibration as soon as
            we have 4 points
240
241         bool ok; // This Boolean variable is True if the user accepts
                the dialog and false if they rejec the dialog (press "
            Cancel" intead of "OK")
242
243         // Shows dialogs to ask the user for the length of the
            legends
244         legend_length_X_=QInputDialog::getDouble(this, tr("
            Calibration Factor"),tr("Enter the length of x axis of
            this area"), 0, 0, 1000, 1, &ok);
245         starting_value_X_=QInputDialog::getDouble(this, tr("
            Calibration Factor"),tr("Enter the starting valur of x"),
            0, 0, 1000, 1, &ok);
246         legend_length_Y_=QInputDialog::getDouble(this, tr("
            Calibration Factor"),tr("Enter the length of y axis of
            this area"), 0, 0, 1000, 1, &ok);
247         starting_value_Y_=QInputDialog::getDouble(this, tr("
            Calibration Factor"),tr("Enter the starting valur of y"),
            0, 0, 1000, 1, &ok);
248
249         // sort the intervals in increasing order to get the zero
            point and max point to get the range of the picture
250         std::sort(calibration_vector_.begin(), calibration_vector_.
            end(), [](auto &a, auto &b){ return a.value <b.value; });
            //Sort the points by the value from least
251         min_point_=calibration_vector_[0].point;//save the zero point
                of the graph
252         max_point_=calibration_vector_[3].point;//save the maxpoint
            of the graph
253
254         double xdistance=max_point_.x()-min_point_.x();//calculate
            the x distance of the max and zero point
255         double ydistance=max_point_.y()-min_point_.y();//calculate
            the y distance of the max and zero point
256
257         calibration_factor_x_=xdistance/legend_length_X_; //
            Calculates the calibration factor for x axis
```

```
258         calibration_factor_y_=ydistance/legend_length_Y_; //
                Calculates the calibration factor for y axis
259
260         calibration_vector_.clear();//cleared CalibrationVector
                because it already finished it mission and prepare for the
                 next recalibration
261     }
262 }
263
264 void GraphsDigitiser::on_Load_file_clicked()
265 {// Called whenever the user clicks on "Load file"
266
267     //enable the buttons whrn finish loading
268     ui->Calibrate->setEnabled(true);
269     ui->Mode->setEnabled(true);
270     ui->Mode_2->setEnabled(true);
271     ui->Mode_3->setEnabled(true);
272     ui->Mode_4->setEnabled(true);
273     ui->Mode_select->setEnabled(true);
274     ui->Mode_select_list->setEnabled(true);
275
276     RefreshValues();//refresed the points saves for the last time
277
278     QString filename=QFileDialog::getOpenFileName(this,tr("Load Note"
            ),tr("*.png")); // Allows the user to choose an input file
279
280     //QImage image(filename); // Saves the file in a a QPixmap object
281     QImage image(filename); // Saves the file in a a QPixmap object
282
283     image_=image;//save the image for auto mode
284
285     //scene->addPixmap(image); // Adds the pixmap to the scene
286     ui->label->setPixmap(QPixmap::fromImage(image));
287     ui->label->show();
288 }
289
290 void GraphsDigitiser::on_Export_clicked()
291 {
292     SavePoints();//save points when export on clicked
293     ResetButtons();//reset button for the next image
294 }
295
296 void GraphsDigitiser::RefreshAxisModeChoosingPushButton()
```

```
297 {
298     if(mode_flag_==1){
299         ui->Mode_select->setText("Mode:linear/linear");//change the
                text on the button to "Mode:linear/linear"
300     }
301     else if(mode_flag_==2){
302         ui->Mode_select->setText("Mode:linear/log");//switch to "Mode
                :linear/log"
303     }
304     else if(mode_flag_==3){
305         ui->Mode_select->setText("Mode:log/linear");//switch to "Mode
                :log/linear"
306     }
307     else if(mode_flag_==4){
308         ui->Mode_select->setText("Mode:log/log");//switch to "Mode:
                log/log"
309     }
310 }
311
312 void GraphsDigitiser::RefreshAxisModeChoosingRadioButton()
313 {
314     if(mode_flag_==1){
315         ui->Mode->setChecked ( true );//switch on the push button "
                Mode: linear/linear"
316     }
317     else if(mode_flag_==2){
318         ui->Mode_2->setChecked ( true );//switch on the push button "
                Mode: linear/log"
319     }
320     else if(mode_flag_==3){
321         ui->Mode_3->setChecked ( true );//switch on the push button "
                Mode: log/linear"
322     }
323     else if(mode_flag_==4){
324         ui->Mode_4->setChecked ( true );//switch on the push button "
                Mode: log/log"
325     }
326 }
327
328 void GraphsDigitiser::RefreshAxisModeChoosingComboBox()
329 {
330     if(mode_flag_==1){
```

```
331         ui->Mode_select_list->setCurrentIndex(0);//switch on the
                combobox model to "Mode:linear/linear"
332     }
333     else if(mode_flag_==2){
334         ui->Mode_select_list->setCurrentIndex(1);//switch on the
                combobox model to "Mode:linear/log"
335     }
336     else if(mode_flag_==3){
337         ui->Mode_select_list->setCurrentIndex(2);//switch on the
                combobox model to "Mode:log/linear"
338     }
339     else if(mode_flag_==4){
340         ui->Mode_select_list->setCurrentIndex(3);//switch on the
                combobox model to "Mode:log/log"
341     }
342 }
343
344 void GraphsDigitiser::AutoSelectPoint()
345 {
346     segment_x_=(max_point_.x()-min_point_.x())/(auto_points_num_+1);
347     segment_y_=(max_point_.y()-min_point_.y())/(auto_points_num_+1);
348
349     //QColor a=image_.pixelColor(floor(tmpPoint_.x()),floor(tmpPoint_
            .y()));
350     //QColor a=image_.pixelColor(tmpPoint_.x()-11,tmpPoint_.y()-122)
            ;//get the pixel color information
351     QColor tmp_color;//save tempory color data in calculating
352     QPoint tmp_point;//save tempory point data in calculating
353     int tmp_x;
354     for(int i=1;i<auto_points_num_;i++)//outer loop search x with
            user selected number
355     {
356         for(int j=min_point_.ry()+segment_y_;j<max_point_.ry()-
                segment_y_;j++)//+ and - segment y to make sure not to get
                 the points on the axis
357         {//inner loop search every y on x axis
358             tmp_x=i*segment_x_+min_point_.x();//set x coordinate by
                    the segment
359             tmp_color=image_.pixelColor(tmp_x-11,j-122);//get the
                    pixel color information with image location
                    calibrating
360             if(tmp_color.red()+tmp_color.blue()+tmp_color.green()
                    <400){//judge the color based on rgb value, <400 could
```

```
                    be considered as black
361                 tmp_point.setX(tmp_x);//set x to the location get
                        from the loop
362                 tmp_point.setY(j);//set y to the location get from
                        the loop
363                 points_vector_.push_back(tmp_point);//save this point
                         to the vector to save them for further actions
364                 CalculateAndDisplayPoints();//calculate the reality
                        value of the selected point
365                 continue;//for each x have a single y for each
                        function so when get point then finish this local
                        loop to save time
366             }
367         }
368     }
369     SavePoints();//save and export points data
370 }
371
372 void GraphsDigitiser::on_Mode_select_clicked()
373 {
374     mode_flag_++;
375     if(mode_flag_>4){
376         mode_flag_=1; //to prevent the flag overflow
377     }
378     if(mode_flag_==1){
379         ui->Mode_select->setText("Mode:linear/linear");//change the
                text on the button to "Mode:linear/linear"
380     }
381     else if(mode_flag_==2){
382         ui->Mode_select->setText("Mode:linear/log");
383     }
384     else if(mode_flag_==3){
385         ui->Mode_select->setText("Mode:log/linear");
386     }
387     else if(mode_flag_==4){
388         ui->Mode_select->setText("Mode:log/log");
389     }
390     RefreshAxisModeChoosingRadioButton();//refresh the radio button
            set of same function
391     RefreshAxisModeChoosingComboBox();//refresh the function on the
            combobox to the defult model
392 }
393
```

```
394 void GraphsDigitiser::on_Mode_clicked()
395 {
396     mode_flag_=1;//set flag to 1
397     RefreshAxisModeChoosingPushButton();//refresh other button
398     RefreshAxisModeChoosingComboBox();//refresh the function on the
        combobox to the defult model
399 }
400
401 void GraphsDigitiser::on_Mode_2_clicked()
402 {
403     mode_flag_=2;//set flag to 2
404     RefreshAxisModeChoosingPushButton();//refresh other button
405     RefreshAxisModeChoosingComboBox();//refresh the function on the
        combobox to the defult model
406 }
407
408 void GraphsDigitiser::on_Mode_3_clicked()
409 {
410     mode_flag_=3;//set flag to 3
411     RefreshAxisModeChoosingPushButton();//refresh other button
412     RefreshAxisModeChoosingComboBox();//refresh the function on the
        combobox to the defult model
413 }
414
415 void GraphsDigitiser::on_Mode_4_clicked()
416 {
417     mode_flag_=4;//set flag to 4
418     RefreshAxisModeChoosingPushButton();//refresh other button
419     RefreshAxisModeChoosingComboBox();//refresh the function on the
        combobox to the defult model
420 }
421
422 void GraphsDigitiser::on_Calibrate_clicked()
423 {
424     is_calibrating=true;//switch on the flag of is_calibrating to
        activate calibrating function in mouse event
425     ui->Get_point->setEnabled(true);
426     ui->Auto_get_point->setEnabled(true);
427     ui->Export->setEnabled(true);
428 }
429
430 void GraphsDigitiser::on_Get_point_clicked()
431 {
```

```
432     is_selecting_=true;//switch on the flag of is_selecting to
            activate selecting function in mouse event
433 }
434
435 void GraphsDigitiser::RefreshValues()
436 {
437     is_calibrating=false; // refresh flag is_calibrating
438     is_selecting_=false; // refresh flag false
439     mode_flag_=1; //refresh flag model
440     points_vector_.clear(); // Vector for saving the points selected
441
442     //refresh buttons
443     RefreshAxisModeChoosingPushButton();//refresh the text on the
            push button to the defult model
444     RefreshAxisModeChoosingRadioButton();//refresh the text on the
            radio button to the defult model
445     RefreshAxisModeChoosingComboBox();//refresh the function on the
            combobox to the defult model
446
447     points_x_.clear();//clear the vector to save x values
448     points_y_.clear();//clear the vector to save y values
449
450 }
451
452 void GraphsDigitiser::ResetButtons()
453 {
454
455     //dis able the functions can not be used before import the graph
456     ui->Calibrate->setEnabled(false);
457     ui->Get_point->setEnabled(false);
458     ui->Auto_get_point->setEnabled(false);
459     ui->Export->setEnabled(false);
460     ui->Mode->setEnabled(false);
461     ui->Mode_2->setEnabled(false);
462     ui->Mode_3->setEnabled(false);
463     ui->Mode_4->setEnabled(false);
464     ui->Mode_select->setEnabled(false);
465     ui->Mode_select_list->setEnabled(false);
466 }
467
468 void GraphsDigitiser::on_Mode_select_list_activated(const QString &
        arg1)
469 {
```

```
470     //the function for the QComboButton which allow the user to
            choose mode
471     if (arg1=="Mode: linear/linear"){
472         mode_flag_=1; //set model flag to 1 when user choose Mode:
                linear/linear
473         RefreshAxisModeChoosingPushButton();//refresh other button
                related to mode
474         RefreshAxisModeChoosingRadioButton();//refresh other button
                related to mode
475     }else if(arg1=="Mode: linear/log"){
476         mode_flag_=2; //set model flag to 2 when user choose Mode:
                linear/log
477         RefreshAxisModeChoosingPushButton();//refresh other button
                related to mode
478         RefreshAxisModeChoosingRadioButton();//refresh other button
                related to mode
479     }else if(arg1=="Mode: log/linear"){
480         mode_flag_=3; //set model flag to 3 when user choose Mode:
                log/linear
481         RefreshAxisModeChoosingPushButton();//refresh other button
                related to mode
482         RefreshAxisModeChoosingRadioButton();//refresh other button
                related to mode
483     }else if(arg1=="Mode: log/log"){
484         mode_flag_=4; //set model flag to 4 when user choose Mode:
                log/log
485         RefreshAxisModeChoosingPushButton();//refresh other button
                related to mode
486         RefreshAxisModeChoosingRadioButton();//refresh other button
                related to mode
487     }
488 }
489
490 void GraphsDigitiser::on_Auto_get_point_clicked()
491 {
492     //The user could select the point number for the auto mode
493     bool ok; // This Boolean variable is True if the user accepts the
            dialog and false if they reject the dialog (press "Cancel"
            instead of "OK")
494     auto_points_num_=QInputDialog::getInt(this, tr("Point number"),tr
            ("Enter the point number you would like to get:"), 0, 0, 1000,
             1, &ok);
495
```

```
496     //select point based on pixel information automatically
497     AutoSelectPoint();//get point and export automatically
498     ResetButtons();//refresh the button status for the next use
499 }
500
501 void GraphsDigitiser::paintEvent(QPaintEvent *)
502 { // This function is called implicitly in the constructor, and
        whoever there is update() or repaint() function
503
504     QPainter painter(this);//this used to paint points selected
505     // make the graph in a good shape
506     painter.setRenderHint(QPainter::Antialiasing, true);
507     // setting the pen
508     painter.setPen(QPen(QColor(0, 160, 230), 2));
509     // setting the color
510     painter.setBrush(QColor(255, 160, 90));
511     //paint all the points user selected saved in the vector
512     if(!points_vector_.isEmpty()){
513         for (int i=0;i<points_vector_.size();i++) {
514             painter.drawEllipse(points_vector_[i].x(),points_vector_[
                    i].y(),10,10);
515         }
516     }
517
518     QPainter painter2(this);//this used to paint calibrate points
            selected
519     // make the graph in a good shape
520     painter2.setRenderHint(QPainter::Antialiasing, true);
521     // setting the pen
522     painter2.setPen(QPen(QColor(0, 160, 230), 2));
523     // setting the color
524     painter2.setBrush(QColor(160, 32, 240));
525     //paint all the points user selected for calibration
526     if(!calibration_vector_.isEmpty()){
527         for (int i=0;i<calibration_vector_.size();i++) {
528             painter2.drawEllipse(calibration_vector_[i].point.x(),
                    calibration_vector_[i].point.y(),10,10);
529         }
530     }
531
532     //painter.drawEllipse(point_location_.x(),point_location_.y()
            ,10,10);
533 }
```

```
534
535
536 //main.cpp
537
538 #include "graphsdigitiser.h"
539 #include <QApplication>
540
541
542
543 int main(int argc, char *argv[])
544 {
545     QApplication a(argc, argv);
546     GraphsDigitiser w;
547     w.show();
548
549     return a.exec();
550 }
```

References