ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba

Manual de Código

Grado en Ingeniería Informática
(Doble mención computación + computadores)

Sistema de mapeo de interiores mediante mediciones láser.

Autor
Manuel Rafael Navarro Fuentes

Director
Dr. Rafael Muñoz Salinas

Octubre, 2020

UNIVERSIDAD D CÓRDOBA

Índice

Capítulo 1

Introducción

El contenido del fichero "LMS.zip" se compone de una serie de ficheros de cabecera y ficheros de código, así como ficheros de interfaz, de proyecto y recursos de Qt. Los ficheros se han clasificado en las siguientes categorías:

- **Proyecto de Qt:** el fichero LMS.pro fichero servirá para crear el makefile necesario de forma automática
- **Inicio del programa:** para ello se usará el fichero main.cpp.
- **Ventana principal:** la ventana principal de la aplicación se compone de 3 ficheros, mainwindow.h, mainwindow.cpp y mainwindow.ui.
- **Modificación de parámetros:** para modificar loar parámetros de forma gráfica y almacenarlos se usan los ficheros parameters.cpp, parameters.h y parameters.ui.
- **Creación del mapa real:** para crear el mapa real se precisan los ficheros work.h y work.cpp.
- **Simulaciones:** para correr simulaciones se utilizan los ficheros simulation.h y simulation.cpp.
- **Funcionalidades para SLAM:** se utilizará el fichero slam.hpp.
- **Levenberg Marquardt:** el fichero levmarq.h contendrá el algoritmo genérico de Levenberg Marquardt.
- **Recursos:** el fichero resources.qrc contendrá el código de recurso del icono de la aplicación.

Capítulo 2

Proyecto de Qt

2.1. LMS.pro

Listing 2.1: LMS.pro

```
1  QT       += core gui
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++11
6
7  # The following define makes your compiler emit warnings if you use
8  # any Qt feature that has been marked deprecated (the exact warnings
9  # depend on your compiler). Please consult the documentation of the
10 # deprecated API in order to know how to port your code away from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
13 # You can also make your code fail to compile if it uses deprecated
       APIs.
14 # In order to do so, uncomment the following line.
15 # You can also select to disable deprecated APIs only up to a certain
       version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
       the APIs deprecated before Qt 6.0.0
17
18 unix {
19     CONFIG += link_pkgconfig
20     PKGCONFIG += opencv
21 }
22
23 SOURCES += \
24     main.cpp \
25     mainwindow.cpp \
26     parameters.cpp \
27     sdk/src/arch/linux/net_serial.cpp \
```

```
28      sdk/src/arch/linux/net_socket.cpp \
29      sdk/src/arch/linux/timer.cpp \
30      sdk/src/hal/thread.cpp \
31      sdk/src/rplidar_driver.cpp \
32      simulation.cpp \
33      work.cpp
34
35  HEADERS += \
36      mainwindow.h \
37      parameters.h \
38      sdk/include/rplidar.h \
39      sdk/include/rplidar_cmd.h \
40      sdk/include/rplidar_driver.h \
41      sdk/include/rplidar_protocol.h \
42      sdk/include/rptypes.h \
43      sdk/src/arch/linux/arch_linux.h \
44      sdk/src/arch/linux/net_serial.h \
45      sdk/src/arch/linux/thread.hpp \
46      sdk/src/arch/linux/timer.h \
47      sdk/src/hal/abs_rxtx.h \
48      sdk/src/hal/assert.h \
49      sdk/src/hal/byteops.h \
50      sdk/src/hal/event.h \
51      sdk/src/hal/locker.h \
52      sdk/src/hal/socket.h \
53      sdk/src/hal/thread.h \
54      sdk/src/hal/types.h \
55      sdk/src/hal/util.h \
56      sdk/src/rplidar_driver_TCP.h \
57      sdk/src/rplidar_driver_impl.h \
58      sdk/src/rplidar_driver_serial.h \
59      sdk/src/sdkcommon.h \
60      simulation.h \
61      slam.hpp \
62      work.h
63
64  FORMS += \
65      mainwindow.ui \
66      parameters.ui
67
68
69  TARGET = LMS
70
71  # Default rules for deployment.
72  qnx: target.path = /tmp/$${TARGET}/bin
73  else: unix:!android: target.path = /opt/$${TARGET}/bin
74  !isEmpty(target.path): INSTALLS += target
75
```

```
76 DISTFILES +=
77
78 RESOURCES += ./resources.qrc
```

Capítulo 3

Inicio del programa

3.1. main.cpp

Listing 3.1: main.cpp

```
1  #include "mainwindow.h"
2  #include <iostream>
3
4  #include <QApplication>
5
6  int main(int argc, char *argv[])
7  {
8      QApplication a(argc, argv);
9      MainWindow w;
10     w.show();
11
12     return a.exec();
13 }
```

Capítulo 4

Ventana principal

4.1. mainwindow.h

Listing 4.1: mainwindow.h

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include "parameters.h"
5  #include "simulation.h"
6  #include "work.h"
7
8  #include "slam.hpp"
9  #include "sdk/include/rplidar.h"
10
11 #include <QThreadPool>
12 #include <QPoint>
13 #include <QMainWindow>
14
15
16 QT_BEGIN_NAMESPACE
17 namespace Ui { class MainWindow; }
18 QT_END_NAMESPACE
19
20 class MainWindow : public QMainWindow
21 {
22     Q_OBJECT
23
24 public:
25     MainWindow(QWidget *parent = nullptr);
26     ~MainWindow();
27
28
29
30 private:
```

```
31      parameters *param; //Parameters window to ask and store parameters
            needed by the software.
32      QThreadPool *pool; //Pool to launch and manage threads.
33      Work work; //Class work for the real sensor scenario.
34      Simulation simulation; //Class simulation for simulated scenarios.
35      QPoint Q1; //First point to calculate distance between two points
            of the map.
36      QPoint Q2; //Second point to calculate distance between two points
            of the map.
37      int points; //Number of points clicked (0, 1 or 2).
38
39
40
41  private slots:
42      //This slot launches the work thread in order to begin creating a
            new real map.
43      void on_b_new_map_clicked(); //Slot executed when the button to
            create a new map is clicked.
44
45      //This slot saves an image of the real map that is beeing created.
46      void on_b_save_map_clicked(); //Slot executed when the button to
            save the map is clicked.
47
48      //This slot saves the readings of the LiDAR in a text file for
            simulate it later.
49      void on_b_save_sim_clicked(); //Slot executed when the button to
            save the readings for simulation is clicked.
50
51      //This slot resets the software, finishing all threads and setting
            default values to the parameters.
52      void on_b_restart_clicked(); //Slot executed when the button reset
            is clicked.
53
54      //This slot loads an specified by the user simulation.
55      void on_b_load_clicked(); //Slot executed when the button to load
            simulation is clicked.
56
57      //This slot stops or starts or continues the simulation, depending
            on the current state of the simulation.
58      void on_b_start_stop_clicked(); //Slot executed when the button to
            start or stop the simulation is clicked.
59
60      //This slot show the parameters window in order to let the user
            set the parameters.
61      void btnaction(); //Slot executed when the button to set the
            parameters is clicked.
62
63      //This slot calculates and shows the distance between two points
```

```
        of the map previously clicked by the user.
64      void on_b_distance_clicked(); //Slot executed when the button to
        calculate distance between two points is clicked.
65
66      //This slot saves the coordinates of a point clicked by the user
        and makes sure there is only a maximum of 2 points active.
67      void on_map_button_clicked(); //Slot executed when the users
        clicks a point in the map to calculate distance.
68
69      //This slot sets the speed for the simulation process.
70      void on_text_spd_textChanged(const QString &arg1); //Slot executed
         when the text which sets the speed is changed.
71
72      //This slot sets the pixmap of a Qlabel in order to show the map
        in real time.
73      void print_img(const QImage &img); //Slot executed when the
        threads (work or simulation) need to print the image of the map
        .
74
75      //This slot shows a message to make the user know that the
        simulation has finished.
76      void sim_finished(); //Slot executed when a simulation finishes.
77
78      //This slot shows an error message to the user when the driver of
        the LiDAR cannot connect to the specified serial port.
79      void error_port(); //Slot executed when the driver of the LiDAR
        cannot connect to the specified serial port.
80
81
82  private:
83      Ui::MainWindow *ui; //Graphic user interface.
84      int count; //Variable used to count the number of readings once a
        simulation is loaded.
85  };
86  #endif // MAINWINDOW_H
```

4.2. mainwindow.cpp

Listing 4.2: mainwindow.cpp

```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  #include "ui_parameters.h"
4  #include "parameters.h"
5  #include "work.h"
6  #include <simulation.h>
7
8  #include <cstdio>
```

```
9  #include <iostream>
10 #include <fstream>
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <math.h>
14 #include <unistd.h>
15
16 #include "slam.hpp"
17 #include "levmarq.h"
18 #include "sdk/include/rplidar.h"
19
20 #include <opencv2/highgui.hpp>
21 #include <opencv2/core/core.hpp>
22 #include <opencv2/imgproc.hpp>
23
24 #include <QRunnable>
25 #include <QObject>
26 #include <QFileDialog>
27 #include <QThread>
28 #include <QProcess>
29 #include <QThreadPool>
30 #include <QMessageBox>
31 #include <QMouseEvent>
32 #include <QString>
33
34
35 MainWindow::MainWindow(QWidget *parent)
36     : QMainWindow(parent)
37     , ui(new Ui::MainWindow)
38 {
39     ui->setupUi(this);
40     //this->setStyleSheet("background-color: rgb(30,50,56);");
41     connect(ui->b_set_parameters,SIGNAL(clicked()),this,SLOT(btnaction
           ())); // creating connections to activate the window parameters
            once the user clicks the button.
42
43     param = new parameters; //Assigning a new parameters to the class
           param.
44
45     pool = QThreadPool::globalInstance(); //Creating a new pool for
           the trheads.
46
47     points = 0; //Initialising the number of points clicked.
48
49     work.setAutoDelete(false); //The thread is not deleted after
           calling run();
50     connect(&work, &Work::print_img,this, &MainWindow::print_img); //
           Connecting the slot print_img of the mainwindow from the signal
```

```
        print_img of the work class.
51      connect(&work, &Work::error_port,this, &MainWindow::error_port);
        //Connecting the slot error_port of the mainwindow from the
        signal error_port of the work class.
52
53      simulation.setAutoDelete(false); //The thread is not deleted after
         calling run();
54      connect(&simulation, &Simulation::print_img,this, &MainWindow::
        print_img); //Connecting the slot print_img of the mainwindow
        from the signal print_img of the simulation class.
55      connect(&simulation, &Simulation::sim_finished,this, &MainWindow::
        sim_finished); //Connecting the slot sim_finished of the
        mainwindow from the signal sim_finished of the simulation class
        .
56
57      setWindowIcon(QIcon(":/pixil-layer-Background.png"));
58
59      work.init(); //Initialising work.
60      simulation.init(); //Initialising simulation.
61  }
62
63  MainWindow::~MainWindow()
64  {
65      //Setting running conditions to false, freeing memory and waiting
        for threads.
66      work.setRunning(false);
67      simulation.setRunning(false);
68      simulation.setFinished(true);
69      pool->waitForDone();
70      if(simulation.started())
71          delete[] simulation.readings_;
72      delete ui;
73  }
74
75  void MainWindow::on_b_new_map_clicked()
76  {
77      //When there is no simulation nor real map running and there is no
         map alredy created, the user can start a new map.
78      if (!simulation.running() && !work.running() && !work.finished()
        && !work.started())
79      {
80          //The real map state is set as started, the parameters are set
             and the thread is launched.
81          work.setStarted(true);
82          work.set_parameters(param->getPoint(), param->getWidth(),
            param->getHeight(), param->getXVariation(), param->
            getYVariation(), param->getAngleVariation(), param->
            getWeight(), param->getBaudrate(), param->getPort(), param
```

```
            ->getLevmarq(), param->getRangeX(), param->getRangeY(),
            param->getRangeAngle(), param->getIgnoreHuman());
83          pool->start(&work);
84      }
85      else
86      {
87          //If the state of the software does not mach the conditions, a
                warning message is shown.
88          QMessageBox msg;
89          msg.setText("There is a simulation running or a map beeing
            created. If the map has finished, reset and start maping
            again");
90          msg.exec();
91      }
92  }
93
94  void MainWindow::on_b_save_map_clicked()
95  {
96      if (work.running()) //If a real map is beeing created, then the
        image of that map can be saved.
97      {
98          //The state of save is set as true in order to block the
            thread while the user saves the image.
99          work.setSave(true);
100
101         //A file dialog to ask the user for the name of the file is
            shown and the image is saved.
102         QString fileName = QFileDialog::getSaveFileName(this, tr("Save
                Map"), ".", tr("Image Files (*.png)"));
103         cv::imwrite(fileName.toStdString()+".png",work.getImgMap());
104
105         //The state of save is now set to false again and the thread
            stops running.
106         work.setSave(false);
107         work.setRunning(false);
108
109         //The real map state is now set to finished.
110         work.setFinished(true);
111     }
112     else if (simulation.started()) //If there is a simulaion started,
        a warning message is shown.
113     {
114         QMessageBox msg;
115         msg.setText("There is a simulation running. To save a
            simulation map, click stop or let the simulation finish.");
116         msg.exec();
117     }
118     else if (work.finished()) //If the real map has been created, a
```

```
        warning message is shown.
119     {
120         QMessageBox msg;
121         msg.setText("A map has just finished and it has been saved.
               Reset and then start a new map");
122         msg.exec();
123     }
124     else //If there is no simulation nor real map beeing created, a
        warning message is shown.
125     {
126         QMessageBox msg;
127         msg.setText("There is no map beeing created.");
128         msg.exec();
129     }
130 }
131
132
133 void MainWindow::on_b_save_sim_clicked()
134 {
135     //The user will be able to save the readings only if a real map
        has finished, the real map is not running and no simulation has
         started.
136     if (work.finished() && !work.running() && !simulation.started())
137     {
138         //A file dialog will be shown to choose the name and route of
            the file to save to.
139         QString fileName = QFileDialog::getSaveFileName(this, tr("Save
             simulation"), ".", tr("Text Files (*.txt)"));
140         std::ofstream f_w;
141             //The file is opened.
142             f_w.open(fileName.toStdString()+".txt");
143
144             if (f_w.is_open())
145             {
146                 //If the file is opened, the readings from the work
                    class are written into the file.
147                 for (int i = 0; i < (int)work.getCountReadings(); i++)
148                 {
149                     //Angle from where the ray was captured.
150                     int write = work.readings_[i].angle_z_q14;
151                     f_w << write << "\n";
152                     //Distance the ray traveled.
153                     write = work.readings_[i].dist_mm_q2;
154                     f_w << write << "\n";
155                     //Quality of the reading.
156                     write = work.readings_[i].quality;
157                     f_w << write << "\n";
158                     //Number of the ray.
```

```
159                     write = work.readings_[i].flag;
160                     f_w << write << "\n";
161                 }
162                 //f_w << "\n";
163             }
164
165             //Once the file is written, it is closed.
166             f_w.close();
167     }
168     else if(simulation.running()) //If there is a simulation running,
            a warning message is shown
169     {
170         QMessageBox msg;
171         msg.setText("You cannot save data from simulation, you should
                already have the file.");
172         msg.exec();
173     }
174     else //If none of the conditions are met, it means there is a real
            map beeing created and the user needs to save it to proceed.
175     {
176         QMessageBox msg;
177         msg.setText("Save the map and then save the data for
                simulation.");
178         msg.exec();
179     }
180
181
182 }
183
184
185 void MainWindow::on_b_restart_clicked()
186 {
187     //If a simulation has started and is not running, it resets the
            program.
188     if (simulation.started() && !simulation.running())
189     {
190         //The elementos of the UI are reset to their initial state, as
                well as the param class and the simulation class.
191         ui->map->setText("MAP");
192         ui->simu_finished->setText("");
193         ui->text_spd->setText("100");
194         ui->l_point1->setStyleSheet("");
195         ui->t_point1->setStyleSheet("");
196         ui->l_point1->setText("");
197         ui->t_point1->setText("");
198         ui->l_point2->setStyleSheet("");
199         ui->t_point2->setStyleSheet("");
200         ui->l_point2->setText("");
```

```
201         ui->t_point2->setText("");
202         ui->lab_distance->setStyleSheet("");
203         ui->lab_distance->setText("");
204         simulation.clear();
205         param->clear();
206     }
207     //If there is a simulation started and running, a warning message
        is shown.
208     else if (simulation.started() && simulation.running())
209     {
210         QMessageBox msg;
211         msg.setText("You cannot reset when a simulation is running.\
            nStop the simulation and try again");
212         msg.exec();
213     }
214     //If no simulation or map has started or finished, there is
        nothing to reset.
215     else if (!simulation.started() && !work.finished() && !work.
        running())
216     {
217         QMessageBox msg;
218         msg.setText("There is nothing to reset.");
219         msg.exec();
220     }
221     else if(work.running()) //If there is a real map running, it does
        not reset.
222     {
223         QMessageBox msg;
224         msg.setText("You cannot reset when a map is beeing created.\
            nSave the map and try again");
225         msg.exec();
226     }
227     //If the simulation has not started and a real map is finished, it
        resets the program.
228     else if(!simulation.running() && work.finished())
229     {
230         //The elementos of the UI are reset to their initial state, as
            well as the param class and the work class.
231         ui->map->setText("MAP");
232         ui->simu_finished->setText("");
233         ui->text_spd->setText("100");
234         ui->l_point1->setStyleSheet("");
235         ui->t_point1->setStyleSheet("");
236         ui->l_point1->setText("");
237         ui->t_point1->setText("");
238         ui->l_point2->setStyleSheet("");
239         ui->t_point2->setStyleSheet("");
240         ui->l_point2->setText("");
```

```
241         ui->t_point2->setText("");
242         ui->lab_distance->setStyleSheet("");
243         ui->lab_distance->setText("");
244         work.clear();
245         param->clear();
246     }
247
248 }
249
250
251
252
253 void MainWindow::on_b_load_clicked()
254 {
255     //If there is no real map beeing created or finished and no
          simulation finished or
256      if (!work.running() && !work.finished() && !simulation.running()
          && !simulation.finished() && !simulation.started())
257      {
258          //A file dialog is opened to ask the user for the path and
              file name to load.
259         QString fileName = QFileDialog::getOpenFileName(this, tr("Open
              simulation"), ".", tr("Text Files (*.txt)"));
260
261          std::ifstream f_r;
262          std::string line;
263
264          f_r.open(fileName.toStdString());
265
266          //If the file is opened, the readings are saved into a buffer
               and the number of readings is updated.
267          if (f_r.is_open())
268          {
269              simulation.readings_ = (
                    rplidar_response_measurement_node_hq_t*) std::malloc(
                    sizeof(rplidar_response_measurement_node_hq_t)*200000)
                    ;
270              count = 0;
271              while(getline(f_r, line))
272              {
273                  if (count < 200000)
274                  {
275                      simulation.readings_[count].angle_z_q14 = stoi(
                            line);
276                      getline(f_r, line);
277                      simulation.readings_[count].dist_mm_q2 = stoi(
                            line);
278                      getline(f_r, line);
```

```
279                     simulation.readings_[count].quality = stoi(line);
280                     getline(f_r, line);
281                     simulation.readings_[count].flag = stoi(line);
282                 }
283                 count++;
284             }
285             simulation.setCount(count);
286             simulation.setLoaded(true);
287             QMessageBox msg;
288             msg.setText("Data loaded!");
289             msg.exec();
290         }
291         f_r.close();
292 
293     }
294     else if (work.running())
295     {
296         QMessageBox msg;
297         msg.setText("A real map is beeing created. You cannot load a
              simulation.");
298         msg.exec();
299     }
300     else
301     {
302         QMessageBox msg;
303         msg.setText("A simulation is active. You cannot load a new
              simulation.\n\n Reset and then load.");
304         msg.exec();
305     }
306 }
307 
308 void MainWindow::on_b_start_stop_clicked()
309 {
310     if(!simulation.finished()) //If simulation is not finished, some
          other requirements are checked.
311     {
312         //If there is a simulation running and no map beeing created,
              the simulation stops.
313         if (simulation.running() && !work.running())
314         {
315             simulation.setRunning(false);
316         }
317         //If the simulation has started but is not running, it means
              that the simulation is stopped. If no map is beeing created
              , the simulation continues.
318         else if (!simulation.running() && !work.running() &&
              simulation.started())
319         {
```

```
320             simulation.setRunning(true);
321         }
322         //If the simulation has been loaded, there is no map beeing
               created and simulation is not started or running, then it
               starts.
323         else if (!simulation.running() && !work.running() && !
               simulation.started() && simulation.loaded())
324         {
325             simulation.setRunning(true);
326             simulation.setStarted(true);
327             simulation.set_parameters(param->getPoint(), param->
                   getWidth(), param->getHeight(), param->getXVariation(),
                    param->getYVariation(), param->getAngleVariation(),
                   param->getWeight(), ui->text_spd->text().toDouble(),
                   param->getLevmarq(), param->getRangeX(), param->
                   getRangeY(), param->getRangeAngle(), param->
                   getIgnoreHuman());
328             pool->start(&simulation);
329         }
330         //If there is a map beeing created, a warning message is shown
               .
331         else if (work.running() || work.finished())
332         {
333             QMessageBox msg;
334             msg.setText("You cannot start or stop the simulation
                   because a real map is beeing created or just finished."
                   );
335             msg.exec();
336         }
337         //If there is no simulation loaded, a warning message is shown
               .
338         else if (!simulation.loaded())
339         {
340             QMessageBox msg;
341             msg.setText("There is no simulation data loaded.");
342             msg.exec();
343         }
344
345     }
346     else
347     {
348         QMessageBox msg;
349         msg.setText("The simulation has finished.");
350         msg.exec();
351     }
352 }
353
354 void MainWindow::btnaction()
```

```
355 {
356
357     //The parameters window will only be available when there is no
            map beeing created or simulation running.
358     if (!work.running() && !simulation.running() && !simulation.
            finished() && !work.finished())
359     {
360         param->show();
361         param->setAttribute( Qt::WA_QuitOnClose, false ); //When the
                main window is closed, the window param will also close.
362     }
363     else
364     {
365         QMessageBox msg;
366         msg.setText("You cannot modify the parameters after starting a
                map (real or simulated).");
367         msg.exec();
368     }
369 }
370
371
372
373 void MainWindow::on_map_button_clicked()
374 {
375     //If there are no points clicked, then it counts as the first
            point, and the visual elements are set to visualize the first
            point.
376     if (points%2 == 0)
377     {
378         Q1 = ui->map_button->mapFromGlobal(ui->map_button->cursor().
                pos());//local coordinates
379         ui->l_point1->setStyleSheet("background-color: rgb(54, 104,
                117); color: rgb(238, 238, 236);");
380         ui->t_point1->setStyleSheet("background-color: rgb(54, 104,
                117); color: rgb(238, 238, 236);");
381         ui->l_point1->setText(QString("(%1,%2)").arg(Q1.x()/(800.0/
                param->getWidth())).arg(Q1.y()/(800.0/param->getHeight())))
                ;
382         ui->t_point1->setText("Point 1");
383         points++;
384         ui->l_point2->setStyleSheet("");
385         ui->t_point2->setStyleSheet("");
386         ui->l_point2->setText("");
387         ui->t_point2->setText("");
388     }
389     //If there is one point clicked, the visual elements are set to
            visualize the second point.
390     else if (points%2 == 1)
```

```
391     {
392         Q2 = ui->map_button->mapFromGlobal(ui->map_button->cursor().
                pos());//local coordinates
393         ui->l_point2->setStyleSheet("background-color: rgb(54, 104,
                117); color: rgb(238, 238, 236);");
394         ui->t_point2->setStyleSheet("background-color: rgb(54, 104,
                117); color: rgb(238, 238, 236);");
395         ui->l_point2->setText(QString("(%1,%2)").arg(Q2.x()/(800.0/
                param->getWidth())).arg(Q2.y()/(800.0/param->getHeight()))
                ;
396         ui->t_point2->setText("Point 2");
397         points++;
398     }
399 }
400
401
402 void MainWindow::on_b_distance_clicked()
403 {
404     //If there are only 0 or 1 points clicked, a warning message is
            shown.
405     if (points == 0)
406     {
407         QMessageBox msg;
408         msg.setText("Click two points on the map and then click this
                button to get the distance between them.\n");
409         msg.exec();
410     }
411     else if (points == 1)
412     {
413         QMessageBox msg;
414         msg.setText("You need to click another point before
                calculating\n");
415         msg.exec();
416     }
417     //If two points are clicked, the distance is calculated and shown
            on its appropiate visual element.
418     else if (points == 2)
419     {
420         points = 0;
421         double distance = sqrt(pow(Q1.x()/(800.0/param->getWidth())-Q2
                .x()/(800.0/param->getWidth()),2)+pow(Q1.y()/(800.0/param->
                getHeight())-Q2.y()/(800.0/param->getHeight()),2));
422         ui->lab_distance->setStyleSheet("background-color: rgb(54,
                104, 117);color: rgb(238, 238, 236);");
423         ui->lab_distance->setText(QString("Distance: %1 cm").arg(
                distance));
424     }
425 }
```

```
426
427
428
429
430 void MainWindow::on_text_spd_textChanged(const QString &arg1)
431 {
432     simulation.setSpeed(arg1.toDouble());
433 }
434
435 void MainWindow::sim_finished()
436 {
437    ui->simu_finished->setText("Simulation finished");
438 }
439
440
441 void MainWindow::error_port()
442 {
443     QMessageBox msg;
444     msg.setText("Error, cannot bind to the specified serial port.\n");
445     msg.exec();
446     pool->waitForDone();
447 }
448
449 void MainWindow::print_img(const QImage &img)
450 {
451     ui->map->setPixmap(QPixmap::fromImage(img.rgbSwapped()));
452 }
```

4.3. mainwindow.ui

Listing 4.3: mainwindow.cpp

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3   <class>MainWindow</class>
4   <widget class="QMainWindow" name="MainWindow">
5    <property name="geometry">
6     <rect>
7      <x>0</x>
8      <y>0</y>
9      <width>1075</width>
10     <height>875</height>
11    </rect>
12   </property>
13   <property name="sizePolicy">
14    <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
15     <horstretch>0</horstretch>
16     <verstretch>0</verstretch>
```

```
17     </sizepolicy>
18    </property>
19    <property name="minimumSize">
20     <size>
21      <width>1075</width>
22      <height>875</height>
23     </size>
24    </property>
25    <property name="maximumSize">
26     <size>
27      <width>1075</width>
28      <height>875</height>
29     </size>
30    </property>
31    <property name="windowTitle">
32     <string>LMS</string>
33    </property>
34    <property name="styleSheet">
35     <string notr="true">background-color: rgb(30,50,56);</string>
36    </property>
37    <widget class="QWidget" name="centralwidget">
38     <widget class="QPushButton" name="b_new_map">
39      <property name="geometry">
40       <rect>
41        <x>850</x>
42        <y>50</y>
43        <width>200</width>
44        <height>25</height>
45       </rect>
46      </property>
47      <property name="toolTip">
48       <string>If the device does not stop running and no map is started
             after a few seconds, restart the app.</string>
49      </property>
50      <property name="styleSheet">
51       <string notr="true">background-color: rgb(0, 96, 100);
52 color: rgb(238, 238, 236);</string>
53      </property>
54      <property name="text">
55       <string>New map (Start mapping)</string>
56      </property>
57     </widget>
58     <widget class="QPushButton" name="b_save_map">
59      <property name="geometry">
60       <rect>
61        <x>850</x>
62        <y>80</y>
63        <width>200</width>
```

```
64        <height>25</height>
65       </rect>
66      </property>
67      <property name="toolTip">
68       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;Activating
            this button will stop the mapping&lt;/p&gt;&lt;/body&gt;&lt;/
            html&gt;</string>
69      </property>
70      <property name="styleSheet">
71       <string notr="true">background-color: rgb(0, 96, 100);
72  color: rgb(238, 238, 236);</string>
73      </property>
74      <property name="text">
75       <string>Save map</string>
76      </property>
77     </widget>
78     <widget class="QPushButton" name="b_load">
79      <property name="geometry">
80       <rect>
81        <x>850</x>
82        <y>200</y>
83        <width>200</width>
84        <height>25</height>
85       </rect>
86      </property>
87      <property name="styleSheet">
88       <string notr="true">background-color: rgb(74, 96, 107);
89  color: rgb(238, 238, 236);</string>
90      </property>
91      <property name="text">
92       <string>Load simulation</string>
93      </property>
94     </widget>
95     <widget class="QPushButton" name="b_save_sim">
96      <property name="geometry">
97       <rect>
98        <x>850</x>
99        <y>110</y>
100       <width>200</width>
101       <height>25</height>
102      </rect>
103     </property>
104     <property name="styleSheet">
105      <string notr="true">background-color: rgb(0, 96, 100);
106 color: rgb(238, 238, 236);</string>
107     </property>
108     <property name="text">
109      <string>Save data for simulation</string>
```

```
110     </property>
111    </widget>
112    <widget class="QLabel" name="label">
113     <property name="geometry">
114      <rect>
115       <x>850</x>
116       <y>230</y>
117       <width>50</width>
118       <height>25</height>
119      </rect>
120     </property>
121     <property name="styleSheet">
122      <string notr="true">background-color: rgb(74, 96, 107);
123 color: rgb(238, 238, 236);</string>
124     </property>
125     <property name="text">
126      <string>Speed:</string>
127     </property>
128    </widget>
129    <widget class="QLineEdit" name="text_spd">
130     <property name="geometry">
131      <rect>
132       <x>901</x>
133       <y>230</y>
134       <width>132</width>
135       <height>25</height>
136      </rect>
137     </property>
138     <property name="styleSheet">
139      <string notr="true">background-color: rgb(74, 96, 107);
140 color: rgb(238, 238, 236);</string>
141     </property>
142     <property name="text">
143      <string>100</string>
144     </property>
145     <property name="alignment">
146      <set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
147     </property>
148    </widget>
149    <widget class="QLabel" name="label_2">
150     <property name="geometry">
151      <rect>
152       <x>1035</x>
153       <y>230</y>
154       <width>15</width>
155       <height>25</height>
156      </rect>
157     </property>
```

```
158     <property name="styleSheet">
159      <string notr="true">background-color: rgb(74, 96, 107);
160 color: rgb(238, 238, 236);</string>
161     </property>
162     <property name="text">
163      <string>%</string>
164     </property>
165    </widget>
166    <widget class="QPushButton" name="b_set_parameters">
167     <property name="geometry">
168      <rect>
169       <x>850</x>
170       <y>320</y>
171       <width>200</width>
172       <height>25</height>
173      </rect>
174     </property>
175     <property name="toolTip">
176      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;Using this
          button is optional, all the parameters needed have a default
          value.&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
177     </property>
178     <property name="styleSheet">
179      <string notr="true">background-color: rgb(89, 126, 143);
180 color: rgb(238, 238, 236);</string>
181     </property>
182     <property name="text">
183      <string>Set parameters</string>
184     </property>
185    </widget>
186    <widget class="QLabel" name="map">
187     <property name="geometry">
188      <rect>
189       <x>25</x>
190       <y>25</y>
191       <width>800</width>
192       <height>800</height>
193      </rect>
194     </property>
195     <property name="font">
196      <font>
197       <pointsize>72</pointsize>
198      </font>
199     </property>
200     <property name="cursor">
201      <cursorShape>CrossCursor</cursorShape>
202     </property>
203     <property name="mouseTracking">
```

```
204       <bool>true</bool>
205      </property>
206      <property name="text">
207       <string>MAP</string>
208      </property>
209      <property name="alignment">
210       <set>Qt::AlignCenter</set>
211      </property>
212     </widget>
213     <widget class="QPushButton" name="b_start_stop">
214      <property name="geometry">
215       <rect>
216        <x>850</x>
217        <y>260</y>
218        <width>200</width>
219        <height>25</height>
220       </rect>
221      </property>
222      <property name="styleSheet">
223       <string notr="true">background-color: rgb(74, 96, 107);
224 color: rgb(238, 238, 236);</string>
225      </property>
226      <property name="text">
227       <string>Start/Stop</string>
228      </property>
229     </widget>
230     <widget class="QPushButton" name="b_restart">
231      <property name="geometry">
232       <rect>
233        <x>850</x>
234        <y>140</y>
235        <width>200</width>
236        <height>25</height>
237       </rect>
238      </property>
239      <property name="styleSheet">
240       <string notr="true">background-color: rgb(0, 96, 100);
241 color: rgb(238, 238, 236);</string>
242      </property>
243      <property name="text">
244       <string>Reset</string>
245      </property>
246     </widget>
247     <widget class="QLabel" name="simu_finished">
248      <property name="geometry">
249       <rect>
250        <x>840</x>
251        <y>570</y>
```

```
252         <width>150</width>
253         <height>25</height>
254        </rect>
255       </property>
256       <property name="styleSheet">
257        <string notr="true">color: rgb(132, 207, 176);
258 </string>
259       </property>
260       <property name="text">
261        <string/>
262       </property>
263       <property name="alignment">
264        <set>Qt::AlignCenter</set>
265       </property>
266      </widget>
267      <widget class="QPushButton" name="map_button">
268       <property name="geometry">
269        <rect>
270         <x>25</x>
271         <y>25</y>
272         <width>800</width>
273         <height>800</height>
274        </rect>
275       </property>
276       <property name="cursor">
277        <cursorShape>CrossCursor</cursorShape>
278       </property>
279       <property name="mouseTracking">
280        <bool>true</bool>
281       </property>
282       <property name="text">
283        <string/>
284       </property>
285       <property name="flat">
286        <bool>true</bool>
287       </property>
288      </widget>
289      <widget class="QLabel" name="t_point1">
290       <property name="geometry">
291        <rect>
292         <x>850</x>
293         <y>410</y>
294         <width>100</width>
295         <height>25</height>
296        </rect>
297       </property>
298       <property name="toolTip">
299        <string>Click two points on the map and you will get the distance
```

```
        </string>
300     </property>
301     <property name="styleSheet">
302      <string notr="true"/>
303     </property>
304     <property name="text">
305      <string/>
306     </property>
307     <property name="alignment">
308      <set>Qt::AlignCenter</set>
309     </property>
310    </widget>
311    <widget class="QLabel" name="lab_distance">
312     <property name="geometry">
313      <rect>
314       <x>850</x>
315       <y>470</y>
316       <width>200</width>
317       <height>25</height>
318      </rect>
319     </property>
320     <property name="text">
321      <string/>
322     </property>
323     <property name="alignment">
324      <set>Qt::AlignCenter</set>
325     </property>
326    </widget>
327    <widget class="QPushButton" name="b_distance">
328     <property name="geometry">
329      <rect>
330       <x>850</x>
331       <y>380</y>
332       <width>200</width>
333       <height>25</height>
334      </rect>
335     </property>
336     <property name="styleSheet">
337      <string notr="true">background-color: rgb(54, 104, 117);
338 color: rgb(238, 238, 236);</string>
339     </property>
340     <property name="text">
341      <string>Get distance between points</string>
342     </property>
343    </widget>
344    <widget class="QLabel" name="t_point2">
345     <property name="geometry">
346      <rect>
```

```
347         <x>950</x>
348         <y>410</y>
349         <width>100</width>
350         <height>25</height>
351        </rect>
352       </property>
353       <property name="toolTip">
354        <string>Click two points on the map and you will get the distance
             </string>
355       </property>
356       <property name="styleSheet">
357        <string notr="true"/>
358       </property>
359       <property name="text">
360        <string/>
361       </property>
362       <property name="alignment">
363        <set>Qt::AlignCenter</set>
364       </property>
365      </widget>
366      <widget class="QLabel" name="l_point1">
367       <property name="geometry">
368        <rect>
369         <x>850</x>
370         <y>440</y>
371         <width>100</width>
372         <height>25</height>
373        </rect>
374       </property>
375       <property name="toolTip">
376        <string>Click two points on the map and you will get the distance
             </string>
377       </property>
378       <property name="styleSheet">
379        <string notr="true"/>
380       </property>
381       <property name="text">
382        <string/>
383       </property>
384       <property name="alignment">
385        <set>Qt::AlignCenter</set>
386       </property>
387      </widget>
388      <widget class="QLabel" name="l_point2">
389       <property name="geometry">
390        <rect>
391         <x>950</x>
392         <y>440</y>
```

```
393         <width>100</width>
394         <height>25</height>
395        </rect>
396       </property>
397       <property name="toolTip">
398        <string>Click two points on the map and you will get the distance
              </string>
399       </property>
400       <property name="styleSheet">
401        <string notr="true"/>
402       </property>
403       <property name="text">
404        <string/>
405       </property>
406       <property name="alignment">
407        <set>Qt::AlignCenter</set>
408       </property>
409      </widget>
410     </widget>
411     <widget class="QMenuBar" name="menubar">
412      <property name="geometry">
413       <rect>
414        <x>0</x>
415        <y>0</y>
416        <width>1075</width>
417        <height>22</height>
418       </rect>
419      </property>
420     </widget>
421     <widget class="QStatusBar" name="statusbar"/>
422    </widget>
423    <resources/>
424    <connections/>
425   </ui>
```

Capítulo 5

Modificación de los parámetros

5.1. parameters.h

Listing 5.1: parameters.h

```
1  #ifndef PARAMETERS_H
2  #define PARAMETERS_H
3
4  #include <QWidget>
5  #include "slam.hpp"
6  #include <string>
7
8  namespace Ui {
9  class parameters;
10 }
11
12 class parameters : public QWidget
13 {
14     Q_OBJECT
15
16 private:
17     Ui::parameters *ui; //Graphic user interface.
18
19     //Variables that store the parameters needed by the SLAM
          algorithms.
20         //range_... is used by brute force and means the range where
              to try the position of the system.
21         //..._var_ is used by Levenberg Marquardt and means the
              variation of the variable between two estimations.
22         //w_ represents the weight of the previous value of a pixel.
23     int width_,height_, baudrate_, range_x_, range_y_, range_angle_;
24     double angle_var_, x_var_, y_var_, w_;
25     slam::Position p_; //Initial position of the system.
26     char *port_; //The port where the LiDAR will be connected.
27     bool levmarq_; //Represent whether to use Levenberg Marquardt or
```

```
        brute force.
28      bool ignore_human_;
29
30  public:
31      explicit parameters(QWidget *parent = nullptr);
32      ~parameters();
33
34      //Observers
35
36      //Returns the width of the map.
37      inline int getWidth() const {return this->width_;}
38
39      //Returns the height of the map.
40      inline int getHeight() const {return this->height_;}
41
42      //Returns the baudrate that is going to be used to connect to the
            LiDAR.
43      inline int getBaudrate() const {return this->baudrate_;}
44
45      //Returns the range of X to optimize the position with the SLAM
            brute force algorithm.
46      inline int getRangeX() const {return this->range_x_;}
47
48      //Returns the range of Y to optimize the position with the SLAM
            brute force algorithm.
49      inline int getRangeY() const {return this->range_y_;}
50
51      //Returns the range of the angle to optimize the position with the
             SLAM brute force algorithm.
52      inline int getRangeAngle() const {return this->range_angle_;}
53
54      //Returns the initial point of the system in the map.
55      inline slam::Position getPoint() const {return this->p_;}
56
57      //Returns the variation of the angle between two estimations used
            by the SLAM levmarq algorithm.
58      inline double getAngleVariation() const {return this->angle_var_;}
59
60      //Returns the variation of X between two estimations used by the
            SLAM levmarq algorithm.
61      inline double getXVariation() const {return this->x_var_;}
62
63      //Returns the variation of Y between two estimations used by the
            SLAM levmarq algorithm.
64      inline double getYVariation() const {return this->y_var_;}
65
66      //Returns the weight of the previous value value of a pixel
67      inline double getWeight() const {return this->w_;}
```

```
68
69      //Returns the port which the LiDAR driver will try to connect to.
70      inline char * getPort() const {return this->port_;}
71
72      //Returns whether the levmar algorithm or the brute force
            algorithm will be used.
73          // 1 for levmarq, 0 for brute force.
74      inline bool getLevmarq() const {return this->levmarq_;}
75
76      //Returns whether the human should be ignored or not while mapping
            .
77      inline bool getIgnoreHuman() const {return this->ignore_human_;}
78
79
80      //Modifiers
81
82      //Sets the width of the map.
83      inline void setWidth(const int &width) {this->width_ = width;}
84
85      //Sets the height of the map.
86      inline void setHeight(const int &height) {this->height_ = height;}
87
88      //Sets the baudrate to connect to the LiDAR.
89      inline void setBaudrate(const int baudrate) {this->baudrate_ =
            baudrate;}
90
91      //Sets the range of X that the brute force algorithm will use in
            order to estimate the position.
92      inline void setRangeX(const int range_x) {this->range_x_ = range_x
            ;}
93
94      //Sets the range of Y that the brute force algorithm will use in
            order to estimate the position.
95      inline void setRangeY(const int range_y) {this->range_y_ = range_y
            ;}
96
97      //Sets the range of the angle that the brute force algorithm will
            use in order to estimate the position.
98      inline void setRangeAngle(const int range_angle) {this->
            range_angle_ = range_angle;}
99
100     //Sets the initial position of the system.
101     inline void setPoint(const slam::Position P) {this->p_ = P;}
102
103     //Sets the variation of the angle between two estimations that the
            Levenberg Marquardt algorithm will use.
104     inline void setAngleVariation(const double angle_var) {this->
            angle_var_ = angle_var;}
```

```
105
106     //Sets the variation of X between two estimations that the
            Levenberg Marquardt algorithm will use.
107     inline void setXVariation(const double &x_var) {this->x_var_ =
            x_var;}
108
109     //Sets the variation of Y between two estimations that the
            Levenberg Marquardt algorithm will use.
110     inline void setYVariation(const double &y_var) {this->y_var_ =
            y_var;}
111
112     //Sets the weight of the preivous value of a pixel.
113     inline void setWeight(const double &w) {this->w_ = w;}
114
115     //Sets the port which the driver of the LiDAR will use to connect
            to.
116     inline void setPort(const char* &port) {strcpy(this->port_,port);}
117
118     //Sets the algorithm that will be used:
119         // 1 for levmarq, 0 for brute force.
120     inline void setLevmarq(const bool &levmarq) {this->levmarq_ =
            levmarq;}
121
122     void clear();//Frees memory and sets default values to parameters.
123
124 private slots:
125     void on_b_accept_clicked(); //Slot to accept the modification of
            the parameters.
126
127     //Slot that executes when the algorithm is changed, and it loads
            the default values for each algorithm.
128     void on_algorithm_currentIndexChanged(int index);
129
130     void on_b_cancel_clicked(); //Slot to decline the modification of
            the parameters.
131
132     //Slot that executes when the user changes the parameter of
            ignoring human or not and it changes the state of that variable
            .
133     void on_ignore_human_stateChanged(int arg1);
134 };
135
136 #endif // PARAMETERS_H
```

5.2. parameters.cpp

Listing 5.2: parameters.cpp

```
1  #include "parameters.h"
2  #include "ui_parameters.h"
3  #include <iostream>
4  #include <QMessageBox>
5
6
7  parameters::parameters(QWidget *parent) :
8      QWidget(parent),
9      ui(new Ui::parameters)
10 {
11     ui->setupUi(this);
12
13     //Default values for the parameters.
14     p_.setX(400);
15     p_.setY(400);
16     p_.setAngle(0);
17     x_var_= 2;
18     y_var_= 2;
19     angle_var_ = 4;
20     range_x_ = 20;
21     range_y_ = 20;
22     range_angle_ = 80;
23     width_= 800;
24     height_ = 800;
25     baudrate_ = 115200;
26     port_ = (char*) std::malloc(sizeof(char)*256);
27     strcpy(port_,"/dev/ttyUSB0");
28     w_ = 0.7;
29     levmarq_ = false;
30     ignore_human_ = true;
31
32     //Initial algorithm: Brute force. Levenberg Marquardt not visible.
33     ui->algorithm->setCurrentIndex(0);
34     ui->l_levmarq->setVisible(false);
35    /* ui->x_lev->setVisible(false);
36     ui->y_lev->setVisible(false);
37     ui->a_lev->setVisible(false);
38     ui->text_a_var->setVisible(false);
39     ui->text_x_var->setVisible(false);
40     ui->text_y_var->setVisible(false);*/
41
42     ui->text_x->setText("");
43     ui->text_y->setText("");
44     ui->text_x_var->setText("");
45     ui->text_y_var->setText("");
46     ui->text_wi->setText("");
47     ui->text_h->setText("");
```

```
48      ui->text_baud->setText("");
49      ui->text_port->setText("");
50      ui->text_we->setText("");
51      ui->text_a_var->setText("");
52      ui->range_Y->setText("");
53      ui->range_x->setText("");
54      ui->range_angle->setText("");
55
56
57  }
58
59  parameters::~parameters()
60  {
61      delete ui;
62  }
63
64
65  void parameters::on_b_accept_clicked()
66  {
67      //If the users accepts, the parameters are set according to some
            requirements.
68      //If one or many of those parameters are not in the allowed range,
            it will show a warning message to the user to change them.
69      bool error = false;
70      if (ui->text_x_var->text().length() > 0)
71      {
72          if (ui->text_x_var->text().toInt() > 0)
73              x_var_= ui->text_x_var->text().toInt();
74          else
75              error = true;
76      }
77
78      if (ui->text_y_var->text().length() > 0)
79      {
80          if (ui->text_y_var->text().toInt() > 0)
81              y_var_= ui->text_y_var->text().toInt();
82          else
83              error = true;
84      }
85
86      if (ui->text_wi->text().length() > 0)
87      {
88          if (ui->text_wi->text().toInt() > 0)
89              width_= ui->text_wi->text().toInt();
90          else
91              error = true;
92      }
93
```

```
94      if (ui->text_h->text().length() > 0)
95      {
96          if (ui->text_h->text().toInt())
97              height_ = ui->text_h->text().toInt();
98          else
99              error = true;
100     }
101
102     if (ui->text_x->text().length() > 0)
103     {
104         if (ui->text_x->text().toInt() > 0 && ui->text_x->text().toInt
                () < width_)
105             p_.setX(ui->text_x->text().toInt());
106         else
107             error = true;
108     }
109     else
110     {
111         p_.setX(width_/2);
112     }
113
114     if (ui->text_y->text().length() > 0)
115     {
116         if (ui->text_y->text().toInt() > 0 && ui->text_y->text().toInt
                () < height_)
117             p_.setY(ui->text_y->text().toInt());
118         else
119             error = true;
120     }
121     else
122     {
123         p_.setY(height_/2);
124     }
125
126     if (ui->text_baud->text().length() > 0)
127     {
128         if (ui->text_baud->text().toInt() > 0)
129             baudrate_ = ui->text_baud->text().toInt();
130         else
131             error = true;
132     }
133
134     if (ui->text_port->text().length() > 0)
135         strcpy(port_,ui->text_port->text().toStdString().c_str());
136
137     if (ui->text_we->text().length() > 0)
138     {
139         if (ui->text_we->text().toDouble() >= (0-0.00001) && ui->
```

```
            text_we->text().toDouble() <= (1+0.00001))
140             w_ = ui->text_we->text().toDouble();
141         else
142             error = true;
143     }
144
145     if (ui->text_a_var->text().length() > 0)
146     {
147         if (ui->text_a_var->text().toDouble() > 0)
148             angle_var_ = ui->text_a_var->text().toDouble();
149         else
150             error = true;
151     }
152
153     if (ui->range_Y->text().length() > 0)
154     {
155         if (ui->range_Y->text().toInt() > 0)
156             range_y_ = ui->range_Y->text().toInt();
157         else
158             error = true;
159     }
160
161     if (ui->range_x->text().length() > 0)
162     {
163         if (ui->range_x->text().toInt() > 0)
164             range_x_ = ui->range_x->text().toInt();
165         else
166             error = true;
167     }
168
169     if (ui->range_angle->text().length() > 0)
170     {
171         if (ui->range_angle->text().toInt() > 0)
172             range_angle_ = ui->range_angle->text().toInt();
173         else
174             error = true;
175     }
176
177     if (error)
178     {
179         QMessageBox msg;
180         msg.setText("A parameter or many of them have an invalid value
                . Check the parameters in order to correct them.");
181         msg.exec();
182     }
183     else
184         this->close();
185 }
```

```
186
187
188 void parameters::clear()
189 {
190     //Default values for parameters.
191     p_.setX(400);
192     p_.setY(400);
193     p_.setAngle(0);
194     x_var_= 3;
195     y_var_= 3;
196     width_= 800;
197     height_ = 800;
198     baudrate_ = 115200;
199     range_x_ = 20;
200     range_angle_ = 60;
201     range_y_ = 20;
202     ignore_human_ = true;
203     strcpy(port_,"/dev/ttyUSB0");
204     w_ = 0.7;
205     angle_var_ = 4;
206     levmarq_ = false;
207
208
209     ui->text_x->setText("");
210     ui->text_y->setText("");
211     ui->text_x_var->setText("");
212     ui->text_y_var->setText("");
213     ui->text_a_var->setText("");
214     ui->text_wi->setText("");
215     ui->text_h->setText("");
216     ui->text_baud->setText("");
217     ui->text_port->setText("");
218     ui->text_we->setText("");
219     ui->algorithm->setCurrentIndex(0);
220     ui->ignore_human->setChecked(true);
221     ui->ignore_human->setText("Yes");
222
223 }
224
225
226 void parameters::on_algorithm_currentIndexChanged(int index)
227 {
228     //When the users chooses another algorithm, the information shown
        changes in order to
229     //let the user change the parameters for the chosen algorithm.
230     if (index == 0)
231     {
232         levmarq_ = false;
```

```
233         ui->l_levmarq->setVisible(false);
234        /* ui->text_x_var->setVisible(false);
235         ui->text_y_var->setVisible(false);
236         ui->text_a_var->setVisible(false);
237         ui->a_lev->setVisible(false);
238         ui->x_lev->setVisible(false);
239         ui->y_lev->setVisible(false);*/
240
241         ui->l_brute->setVisible(true);
242         ui->x_brute->setVisible(true);
243         ui->y_brute->setVisible(true);
244         ui->angle_brute->setVisible(true);
245         ui->range_Y->setVisible(true);
246         ui->range_x->setVisible(true);
247         ui->range_angle->setVisible(true);
248
249         ui->text_x_var->setText("");
250         ui->text_y_var->setText("");
251         ui->text_a_var->setText("");
252         ui->range_x->setText("");
253         ui->range_Y->setText("");
254         ui->range_angle->setText("");
255         x_var_ = 2;
256         y_var_ = 2;
257         angle_var_ = 4;
258     }
259     else
260     {
261         levmarq_ = true;
262         ui->l_levmarq->setVisible(true);
263         ui->text_x_var->setVisible(true);
264         ui->text_y_var->setVisible(true);
265         ui->text_a_var->setVisible(true);
266         ui->a_lev->setVisible(true);
267         ui->x_lev->setVisible(true);
268         ui->y_lev->setVisible(true);
269
270         ui->l_brute->setVisible(false);
271         ui->x_brute->setVisible(false);
272         ui->y_brute->setVisible(false);
273         ui->angle_brute->setVisible(false);
274         ui->range_Y->setVisible(false);
275         ui->range_x->setVisible(false);
276         ui->range_angle->setVisible(false);
277
278         ui->text_x_var->setText("");
279         ui->text_y_var->setText("");
280         ui->text_a_var->setText("");
```

```
281         x_var_ = 1;
282         y_var_ = 1;
283         angle_var_ = 1.5;
284     }
285 }
286
287 void parameters::on_b_cancel_clicked()
288 {
289     //If the users cancels, every textBox is set to no text.
290     ui->algorithm->setCurrentIndex(0);
291     ui->text_x->setText("");
292     ui->text_y->setText("");
293     ui->text_x_var->setText("");
294     ui->text_y_var->setText("");
295     ui->text_wi->setText("");
296     ui->text_h->setText("");
297     ui->text_baud->setText("");
298     ui->text_port->setText("");
299     ui->text_we->setText("");
300     ui->text_a_var->setText("");
301     ui->range_Y->setText("");
302     ui->range_x->setText("");
303     ui->range_angle->setText("");
304     ui->ignore_human->setChecked(true);
305     ui->ignore_human->setText("Yes");
306
307     this->close();
308 }
309
310 void parameters::on_ignore_human_stateChanged(int arg1)
311 {
312     if (arg1 == 0)
313     {
314         ui->ignore_human->setText("No");
315         ignore_human_ = false;
316     }
317     else if (arg1 == 2)
318     {
319         ignore_human_ = true;
320         ui->ignore_human->setText("Yes");
321     }
322 }
```

5.3. parameters.ui

Listing 5.3: parameters.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>parameters</class>
 <widget class="QWidget" name="parameters">
  <property name="enabled">
   <bool>true</bool>
  </property>
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>878</width>
    <height>510</height>
   </rect>
  </property>
  <property name="minimumSize">
   <size>
    <width>0</width>
    <height>0</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>10000</width>
    <height>10000</height>
   </size>
  </property>
  <property name="focusPolicy">
   <enum>Qt::NoFocus</enum>
  </property>
  <property name="windowTitle">
   <string>Set parameters</string>
  </property>
  <property name="windowIcon">
   <iconset>
    <normaloff>../media/interfaz/opciones.png</normaloff>../media/
       interfaz/opciones.png</iconset>
  </property>
  <property name="styleSheet">
   <string notr="true">background-color: rgb(89, 126, 143);
</string>
  </property>
  <widget class="QLabel" name="label">
   <property name="geometry">
    <rect>
     <x>60</x>
     <y>70</y>
     <width>171</width>
```

```
48       <height>17</height>
49      </rect>
50     </property>
51     <property name="toolTip">
52      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
           style=&quot; font-weight:600;&quot;&gt;Initial position of the
           system, recommended value: middle of map, angle 0. Default:
           (400,400,0)&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</
           string>
53     </property>
54     <property name="styleSheet">
55      <string notr="true">color: rgb(238, 238, 236);</string>
56     </property>
57     <property name="text">
58      <string>Initial Position (integer):</string>
59     </property>
60    </widget>
61    <widget class="QLabel" name="label_2">
62     <property name="geometry">
63      <rect>
64       <x>60</x>
65       <y>100</y>
66       <width>151</width>
67       <height>17</height>
68      </rect>
69     </property>
70     <property name="toolTip">
71      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
           style=&quot; font-weight:600;&quot;&gt;Size of map in cm.
           Default: 800x800.&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&
           gt;</string>
72     </property>
73     <property name="styleSheet">
74      <string notr="true">color: rgb(238, 238, 236);</string>
75     </property>
76     <property name="text">
77      <string>Size of map (integer):</string>
78     </property>
79    </widget>
80    <widget class="QLabel" name="x_lev">
81     <property name="geometry">
82      <rect>
83       <x>60</x>
84       <y>270</y>
85       <width>171</width>
86       <height>17</height>
87      </rect>
88     </property>
```

```
89     <property name="toolTip">
90      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
           style=&quot; font-weight:600;&quot;&gt;Variation of X between
           to tests to optimize position in SLAM. Recommended values: 1 to
            3. Default: 1&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;
          </string>
91     </property>
92     <property name="styleSheet">
93      <string notr="true">color: rgb(238, 238, 236);</string>
94     </property>
95     <property name="text">
96      <string>Variation of X (integer):</string>
97     </property>
98    </widget>
99    <widget class="QLabel" name="a_lev">
100    <property name="geometry">
101     <rect>
102      <x>60</x>
103      <y>330</y>
104      <width>181</width>
105      <height>17</height>
106     </rect>
107    </property>
108    <property name="toolTip">
109     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
           style=&quot; font-weight:600;&quot;&gt;Variation of Angle(theta
           ) between to test to optimize position in SLAM. Recommended
           values: 0.4 to 3.0. Default: 1.5&lt;/span&gt;&lt;/p&gt;&lt;/
           body&gt;&lt;/html&gt;</string>
110    </property>
111    <property name="styleSheet">
112     <string notr="true">color: rgb(238, 238, 236);</string>
113    </property>
114    <property name="text">
115     <string>Variation of angle (float):</string>
116    </property>
117   </widget>
118   <widget class="QLabel" name="y_lev">
119    <property name="geometry">
120     <rect>
121      <x>60</x>
122      <y>300</y>
123      <width>171</width>
124      <height>17</height>
125     </rect>
126    </property>
127    <property name="toolTip">
128     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
```

```
        style=&quot; font-weight:600;&quot;&gt;Variation of Y between
        to tests to optimize position in SLAM. Recommended values: 1 to
         3. Default: 1&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;
        </string>
129    </property>
130    <property name="styleSheet">
131     <string notr="true">color: rgb(238, 238, 236);</string>
132    </property>
133    <property name="text">
134     <string>Variation of Y (integer):</string>
135    </property>
136   </widget>
137   <widget class="QLabel" name="label_6">
138    <property name="geometry">
139     <rect>
140      <x>60</x>
141      <y>130</y>
142      <width>231</width>
143      <height>17</height>
144     </rect>
145    </property>
146    <property name="toolTip">
147     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
        style=&quot; font-weight:600;&quot;&gt;Weight of the occupation
         assigned to the last value of a pixel. A high value means that
         the new information has low importance. Recommended values:
        0.4 to 0.7 Default: 0.7&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt
        ;/html&gt;</string>
148    </property>
149    <property name="styleSheet">
150     <string notr="true">color: rgb(238, 238, 236);</string>
151    </property>
152    <property name="text">
153     <string>Weight of last pixel value (float):</string>
154    </property>
155   </widget>
156   <widget class="QLineEdit" name="text_x">
157    <property name="geometry">
158     <rect>
159      <x>320</x>
160      <y>70</y>
161      <width>71</width>
162      <height>25</height>
163     </rect>
164    </property>
165    <property name="styleSheet">
166     <string notr="true">color: rgb(238, 238, 236);</string>
167    </property>
```

```
168   </widget>
169   <widget class="QLineEdit" name="text_wi">
170    <property name="geometry">
171     <rect>
172      <x>320</x>
173      <y>100</y>
174      <width>71</width>
175      <height>25</height>
176     </rect>
177    </property>
178    <property name="styleSheet">
179     <string notr="true">color: rgb(238, 238, 236);</string>
180    </property>
181   </widget>
182   <widget class="QLineEdit" name="text_x_var">
183    <property name="geometry">
184     <rect>
185      <x>260</x>
186      <y>270</y>
187      <width>171</width>
188      <height>25</height>
189     </rect>
190    </property>
191    <property name="styleSheet">
192     <string notr="true">color: rgb(238, 238, 236);</string>
193    </property>
194   </widget>
195   <widget class="QLineEdit" name="text_y_var">
196    <property name="geometry">
197     <rect>
198      <x>260</x>
199      <y>300</y>
200      <width>171</width>
201      <height>25</height>
202     </rect>
203    </property>
204    <property name="styleSheet">
205     <string notr="true">color: rgb(238, 238, 236);</string>
206    </property>
207   </widget>
208   <widget class="QLineEdit" name="text_a_var">
209    <property name="geometry">
210     <rect>
211      <x>260</x>
212      <y>330</y>
213      <width>171</width>
214      <height>25</height>
215     </rect>
```

```
216     </property>
217     <property name="styleSheet">
218      <string notr="true">color: rgb(238, 238, 236);</string>
219     </property>
220    </widget>
221    <widget class="QLineEdit" name="text_we">
222     <property name="geometry">
223      <rect>
224       <x>300</x>
225       <y>130</y>
226       <width>191</width>
227       <height>25</height>
228      </rect>
229     </property>
230     <property name="styleSheet">
231      <string notr="true">color: rgb(238, 238, 236);</string>
232     </property>
233    </widget>
234    <widget class="QPushButton" name="b_accept">
235     <property name="geometry">
236      <rect>
237       <x>780</x>
238       <y>480</y>
239       <width>89</width>
240       <height>25</height>
241      </rect>
242     </property>
243     <property name="styleSheet">
244      <string notr="true">background-color: rgb(98, 195, 177);</string>
245     </property>
246     <property name="text">
247      <string>Accept</string>
248     </property>
249     <property name="default">
250      <bool>true</bool>
251     </property>
252    </widget>
253    <widget class="QPushButton" name="b_cancel">
254     <property name="geometry">
255      <rect>
256       <x>680</x>
257       <y>480</y>
258       <width>89</width>
259       <height>25</height>
260      </rect>
261     </property>
262     <property name="styleSheet">
263      <string notr="true">background-color: rgb(224, 147, 147);</string>
```

```
264     </property>
265     <property name="text">
266      <string>Cancel</string>
267     </property>
268    </widget>
269    <widget class="QLineEdit" name="text_y">
270     <property name="geometry">
271      <rect>
272       <x>420</x>
273       <y>70</y>
274       <width>71</width>
275       <height>25</height>
276      </rect>
277     </property>
278     <property name="styleSheet">
279      <string notr="true">color: rgb(238, 238, 236);</string>
280     </property>
281    </widget>
282    <widget class="QLineEdit" name="text_h">
283     <property name="geometry">
284      <rect>
285       <x>420</x>
286       <y>100</y>
287       <width>71</width>
288       <height>25</height>
289      </rect>
290     </property>
291     <property name="styleSheet">
292      <string notr="true">color: rgb(238, 238, 236);</string>
293     </property>
294    </widget>
295    <widget class="QLabel" name="label_7">
296     <property name="geometry">
297      <rect>
298       <x>300</x>
299       <y>70</y>
300       <width>16</width>
301       <height>17</height>
302      </rect>
303     </property>
304     <property name="styleSheet">
305      <string notr="true">color: rgb(238, 238, 236);</string>
306     </property>
307     <property name="text">
308      <string>X</string>
309     </property>
310    </widget>
311    <widget class="QLabel" name="label_8">
```

```
312     <property name="geometry">
313      <rect>
314       <x>300</x>
315       <y>100</y>
316       <width>16</width>
317       <height>17</height>
318      </rect>
319     </property>
320     <property name="styleSheet">
321      <string notr="true">color: rgb(238, 238, 236);</string>
322     </property>
323     <property name="text">
324      <string>W</string>
325     </property>
326    </widget>
327    <widget class="QLabel" name="label_9">
328     <property name="geometry">
329      <rect>
330       <x>400</x>
331       <y>100</y>
332       <width>16</width>
333       <height>17</height>
334      </rect>
335     </property>
336     <property name="styleSheet">
337      <string notr="true">color: rgb(238, 238, 236);</string>
338     </property>
339     <property name="text">
340      <string>H</string>
341     </property>
342    </widget>
343    <widget class="QLabel" name="label_10">
344     <property name="geometry">
345      <rect>
346       <x>400</x>
347       <y>70</y>
348       <width>16</width>
349       <height>17</height>
350      </rect>
351     </property>
352     <property name="styleSheet">
353      <string notr="true">color: rgb(238, 238, 236);</string>
354     </property>
355     <property name="text">
356      <string>Y</string>
357     </property>
358    </widget>
359    <widget class="QLabel" name="label_12">
```

```
360     <property name="geometry">
361      <rect>
362       <x>60</x>
363       <y>400</y>
364       <width>131</width>
365       <height>17</height>
366      </rect>
367     </property>
368     <property name="styleSheet">
369      <string notr="true">color: rgb(238, 238, 236);</string>
370     </property>
371     <property name="text">
372      <string>Baudrate (integer):</string>
373     </property>
374    </widget>
375    <widget class="QLabel" name="label_13">
376     <property name="geometry">
377      <rect>
378       <x>40</x>
379       <y>370</y>
380       <width>71</width>
381       <height>21</height>
382      </rect>
383     </property>
384     <property name="font">
385      <font>
386       <pointsize>16</pointsize>
387      </font>
388     </property>
389     <property name="text">
390      <string>Sensor</string>
391     </property>
392    </widget>
393    <widget class="QLabel" name="label_14">
394     <property name="geometry">
395      <rect>
396       <x>60</x>
397       <y>430</y>
398       <width>91</width>
399       <height>17</height>
400      </rect>
401     </property>
402     <property name="toolTip">
403      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
          style=&quot; font-weight:600;&quot;&gt;Port which the system is
           connected to. Default: /dev/ttyUSB0&lt;/span&gt;&lt;/p&gt;&lt
          ;/body&gt;&lt;/html&gt;</string>
404     </property>
```

```
405     <property name="styleSheet">
406      <string notr="true">color: rgb(238, 238, 236);</string>
407     </property>
408     <property name="text">
409      <string>Port (string):</string>
410     </property>
411    </widget>
412    <widget class="QLineEdit" name="text_baud">
413     <property name="geometry">
414      <rect>
415       <x>200</x>
416       <y>400</y>
417       <width>171</width>
418       <height>25</height>
419      </rect>
420     </property>
421     <property name="styleSheet">
422      <string notr="true">color: rgb(238, 238, 236);</string>
423     </property>
424    </widget>
425    <widget class="QLineEdit" name="text_port">
426     <property name="geometry">
427      <rect>
428       <x>200</x>
429       <y>430</y>
430       <width>171</width>
431       <height>25</height>
432      </rect>
433     </property>
434     <property name="styleSheet">
435      <string notr="true">color: rgb(238, 238, 236);</string>
436     </property>
437    </widget>
438    <widget class="QLabel" name="label_15">
439     <property name="geometry">
440      <rect>
441       <x>40</x>
442       <y>40</y>
443       <width>141</width>
444       <height>21</height>
445      </rect>
446     </property>
447     <property name="font">
448      <font>
449       <pointsize>16</pointsize>
450      </font>
451     </property>
452     <property name="text">
```

```
453      <string>Map and SLAM</string>
454     </property>
455    </widget>
456    <widget class="QLabel" name="label_16">
457     <property name="geometry">
458      <rect>
459       <x>10</x>
460       <y>10</y>
461       <width>261</width>
462       <height>20</height>
463      </rect>
464     </property>
465     <property name="font">
466      <font>
467       <italic>true</italic>
468      </font>
469     </property>
470     <property name="text">
471      <string>All fields in this window are optional</string>
472     </property>
473    </widget>
474    <widget class="QLabel" name="l_levmarq">
475     <property name="geometry">
476      <rect>
477       <x>40</x>
478       <y>230</y>
479       <width>321</width>
480       <height>31</height>
481      </rect>
482     </property>
483     <property name="font">
484      <font>
485       <pointsize>16</pointsize>
486      </font>
487     </property>
488     <property name="text">
489      <string>Levenberg Marquardt algorithm</string>
490     </property>
491    </widget>
492    <widget class="QLabel" name="label_11">
493     <property name="geometry">
494      <rect>
495       <x>60</x>
496       <y>190</y>
497       <width>81</width>
498       <height>17</height>
499      </rect>
500     </property>
```

```
501     <property name="toolTip">
502      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
            style=&quot; font-weight:600;&quot;&gt;Weight of the occupation
             assigned to the last value of a pixel. A high value means that
             the new information has low importance. Recommended values:
            0.4 to 0.7 Default: 0.7&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt
            ;/html&gt;</string>
503     </property>
504     <property name="styleSheet">
505      <string notr="true">color: rgb(238, 238, 236);</string>
506     </property>
507     <property name="text">
508      <string>Algorithm</string>
509     </property>
510    </widget>
511    <widget class="QComboBox" name="algorithm">
512     <property name="geometry">
513      <rect>
514       <x>300</x>
515       <y>190</y>
516       <width>191</width>
517       <height>25</height>
518      </rect>
519     </property>
520     <item>
521      <property name="text">
522       <string>Brute force</string>
523      </property>
524     </item>
525     <item>
526      <property name="text">
527       <string>Levenberg Marquardt</string>
528      </property>
529     </item>
530    </widget>
531    <widget class="QLineEdit" name="range_x">
532     <property name="geometry">
533      <rect>
534       <x>690</x>
535       <y>270</y>
536       <width>171</width>
537       <height>25</height>
538      </rect>
539     </property>
540     <property name="toolTip">
541      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
            style=&quot; font-weight:600;&quot;&gt;From -(value/2) to (
            value/2)&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</
```

```
        string>
542    </property>
543    <property name="styleSheet">
544     <string notr="true">color: rgb(238, 238, 236);</string>
545    </property>
546   </widget>
547   <widget class="QLineEdit" name="range_angle">
548    <property name="geometry">
549     <rect>
550      <x>690</x>
551      <y>330</y>
552      <width>171</width>
553      <height>25</height>
554     </rect>
555    </property>
556    <property name="styleSheet">
557     <string notr="true">color: rgb(238, 238, 236);</string>
558    </property>
559   </widget>
560   <widget class="QLabel" name="y_brute">
561    <property name="geometry">
562     <rect>
563      <x>450</x>
564      <y>330</y>
565      <width>211</width>
566      <height>17</height>
567     </rect>
568    </property>
569    <property name="toolTip">
570     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
        style=&quot; font-weight:600;&quot;&gt;From -(value/2) to (
        value/2). Measure: cm. Default: 20&lt;/span&gt;&lt;/p&gt;&lt;/
        body&gt;&lt;/html&gt;</string>
571    </property>
572    <property name="styleSheet">
573     <string notr="true">color: rgb(238, 238, 236);</string>
574    </property>
575    <property name="text">
576     <string>Range to estimate Y (integer):</string>
577    </property>
578   </widget>
579   <widget class="QLabel" name="angle_brute">
580    <property name="geometry">
581     <rect>
582      <x>450</x>
583      <y>300</y>
584      <width>231</width>
585      <height>17</height>
```

```
586      </rect>
587     </property>
588     <property name="toolTip">
589      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
            style=&quot; font-weight:600;&quot;&gt;From -(value/2) to (
            value/2). Measure: degrees. Default: 60&lt;/span&gt;&lt;/p&gt;&
            lt;/body&gt;&lt;/html&gt;</string>
590     </property>
591     <property name="styleSheet">
592      <string notr="true">color: rgb(238, 238, 236);</string>
593     </property>
594     <property name="text">
595      <string>Range to estimate angle (integer):</string>
596     </property>
597    </widget>
598    <widget class="QLineEdit" name="range_Y">
599     <property name="geometry">
600      <rect>
601       <x>690</x>
602       <y>300</y>
603       <width>171</width>
604       <height>25</height>
605      </rect>
606     </property>
607     <property name="toolTip">
608      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
            style=&quot; font-weight:600;&quot;&gt;From -(value/2) to (
            value/2)&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</
            string>
609     </property>
610     <property name="styleSheet">
611      <string notr="true">color: rgb(238, 238, 236);</string>
612     </property>
613    </widget>
614    <widget class="QLabel" name="x_brute">
615     <property name="geometry">
616      <rect>
617       <x>450</x>
618       <y>270</y>
619       <width>211</width>
620       <height>17</height>
621      </rect>
622     </property>
623     <property name="toolTip">
624      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
            style=&quot; font-weight:600;&quot;&gt;From -(value/2) to (
            value/2). Measure: cm. Default: 20&lt;/span&gt;&lt;/p&gt;&lt;/
            body&gt;&lt;/html&gt;</string>
```

```
625     </property>
626     <property name="styleSheet">
627      <string notr="true">color: rgb(238, 238, 236);</string>
628     </property>
629     <property name="text">
630      <string>Range to estimate X (integer):</string>
631     </property>
632    </widget>
633    <widget class="QLabel" name="l_brute">
634     <property name="geometry">
635      <rect>
636       <x>40</x>
637       <y>230</y>
638       <width>211</width>
639       <height>31</height>
640      </rect>
641     </property>
642     <property name="font">
643      <font>
644       <pointsize>16</pointsize>
645      </font>
646     </property>
647     <property name="text">
648      <string>Brute force algorithm</string>
649     </property>
650    </widget>
651    <widget class="QLabel" name="label_17">
652     <property name="geometry">
653      <rect>
654       <x>60</x>
655       <y>160</y>
656       <width>101</width>
657       <height>17</height>
658      </rect>
659     </property>
660     <property name="toolTip">
661      <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
          style=&quot; font-weight:600;&quot;&gt;Weight of the occupation
           assigned to the last value of a pixel. A high value means that
           the new information has low importance. Recommended values:
          0.4 to 0.7 Default: 0.7&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt
          ;/html&gt;</string>
662     </property>
663     <property name="styleSheet">
664      <string notr="true">color: rgb(238, 238, 236);</string>
665     </property>
666     <property name="text">
667      <string>Ignore human: </string>
```

```
668     </property>
669    </widget>
670    <widget class="QCheckBox" name="ignore_human">
671     <property name="geometry">
672      <rect>
673       <x>300</x>
674       <y>160</y>
675       <width>51</width>
676       <height>23</height>
677      </rect>
678     </property>
679     <property name="text">
680      <string>Yes</string>
681     </property>
682     <property name="checked">
683      <bool>true</bool>
684     </property>
685    </widget>
686   </widget>
687   <tabstops>
688    <tabstop>text_x</tabstop>
689    <tabstop>text_y</tabstop>
690    <tabstop>text_wi</tabstop>
691    <tabstop>text_h</tabstop>
692    <tabstop>text_we</tabstop>
693    <tabstop>algorithm</tabstop>
694    <tabstop>text_x_var</tabstop>
695    <tabstop>text_y_var</tabstop>
696    <tabstop>text_a_var</tabstop>
697    <tabstop>range_x</tabstop>
698    <tabstop>range_Y</tabstop>
699    <tabstop>range_angle</tabstop>
700    <tabstop>text_baud</tabstop>
701    <tabstop>text_port</tabstop>
702    <tabstop>b_cancel</tabstop>
703    <tabstop>b_accept</tabstop>
704   </tabstops>
705   <resources/>
706   <connections/>
707  </ui>
```

Capítulo 6

Creación del mapa real

6.1. work.h

Listing 6.1: work.h

```
1  #ifndef WORK_H
2  #define WORK_H
3
4  #include<QObject>
5  #include<QRunnable>
6  #include<unistd.h>
7  #include<QTimer>
8  #include <QThread>
9  #include "slam.hpp"
10 #include "sdk/include/rplidar.h"
11
12
13 class Work : public QObject, public QRunnable
14 {
15     Q_OBJECT
16
17 private:
18     bool running_; //Represents whether the real map is beeing created
           or not.
19     bool error_; //Used to specify if an error has occurred during the
           connection to the LiDAR.
20     bool started_; //Represents whether the thread has started
          creating the real map or not.
21     bool finished_; //Represents whether the real map is finished or
          not.
22     bool save_; //Represents whether the thread has to be blocked
          while the map is saved by the main window.
23     bool levmarq_; //Represents whether to use Levenberg Marquardt
          algorithm or brute force.
24     bool ignore_human_;
```

```
25      int start_; //Represents whether it is the first frame or not.
26      int count_readings_; //Number of total readings from the start.
27      slam::Position P_; //Position of the system.
28
29      //Variables that store the parameters needed by the SLAM
            algorithms.
30          //range_... is used by brute force and means the range where
                to try the position of the system.
31          //..._var_ is used by Levenberg Marquardt and means the
                variation of the variable between two estimations.
32          //w_ represents the weight of the previous value of a pixel.
33      int width_,height_, baudrate_, range_x_, range_y_, range_angle_;
34      double x_var_, y_var_, angle_var_,w_;
35      char *port_;//The port where the LiDAR will be connected.
36      cv::Mat mat_; //Matrix to store the visual map while it is saved
            as an image.
37
38
39  public:
40      rplidar_response_measurement_node_hq_t * readings_; //Readings
            from the LiDAR to save to file.
41
42      void init(); //Initialises bool variables.
43      void run(); //Function that will run as a thread.
44
45
46      //Observers
47
48      //Returns whether the real map is beeing created or not.
49      inline bool running() const {return this->running_;}
50
51      //Returns whether the creation of a map has started or not.
52      inline bool started() const {return this->started_;}
53
54      //Returns whether an error has occurred during the connection to
            the LiDAR or not.
55      inline bool error() const {return this->error_;}
56
57      //Returns whether the thread should be stopped in order ot let the
             main window save the map.
58      inline bool save() const {return this->save_;}
59
60      //Returns whether the creation of a map has finished or not.
61      inline bool finished() const {return this->finished_;}
62
63      //Returns whether the algorithm used is Levenberg Marquart or
            Brute force.
64          // 1 for levmarq, 0 for brute force.
```

```
65      inline bool levmarq() const {return this->levmarq_;}
66
67      //Returns whether it is the first frame or not.
68      inline int start() const {return this->start_;}
69
70      //Returns the number of total readings from the start.
71      inline int getCountReadings() const {return this->count_readings_
           ;}
72
73      //Returns the visual map as a cv::Mat.
74      inline cv::Mat getImgMap() const {return this->mat_;}
75
76      //Modifiers
77
78      //Allows to set the state of the creation of the map to running or
            not running.
79      inline void setRunning(const bool &running) {this->running_ =
           running;}
80
81      //Allows to set the state of the creation of a map to started or
           not started.
82      inline void setStarted(const bool &started) {this->started_ =
           started;}
83
84      //Allows to set the state of the creation of a map to finished or
           unfinished.
85      inline void setFinished(const bool &finished) {this->finished_ =
           finished;}
86
87      //Allows to set the state of the creation of a map to saving or
           not saving.
88      inline void setSave(const bool &save) {this->save_ = save;}
89
90      //Sets the algorithm as levmarq or brute force.
91      inline void setLevmarq(const bool &levmarq) {this->levmarq_ =
           levmarq;}
92
93      //Allows to set the variable start_ to check if it is the first
           frame or not.
94      inline void setStart(const int &start) {this->start_ = start;}
95
96      //Sets the number of readings from the LiDAR from the start of the
            creation of the map.
97      inline void setCountReadings(const int &count) {this->
           count_readings_ = count;}
98
99      //Sets the parameters needed by the SLAM algorithms.
100     void set_parameters(slam::Position p, int width, int height, int
```

```
        x_var, int y_var, double angle_var, double w, int baudrate,
        char *port, bool levmarq, int range_x, int range_y, int
        range_angle, bool ignore_human);
101
102     void clear(); //Frees memory and sets default values to parameters
        .
103
104 signals:
105     void print_img(const QImage &img); //This signal is sent every
        time the map is updated and ready to be printed.
106     void error_port(); //This signal is sent when the driver of the
        LiDAR could not connect to the specified serial port.
107
108 };
109
110 #endif // WORK_H
```

6.2. work.cpp

Listing 6.2: work.cpp

```
1  #include "work.h"
2
3  #include "sdk/include/rplidar.h"
4  #include "levmarq.h"
5  #include "slam.hpp"
6
7  #include<QThread>
8  #include <QImage>
9
10
11 //Error function to optimize using Levenberg Marquardt algorithm.
12 void error_2(const aruco::LevMarq<double>::eVector &sol, aruco::
       LevMarq<double>::eVector &err, const slam::Map &map, const double *
       dist, const double *theta, const int count)
13 {
14     err.resize(1);
15     slam::Position P(sol(0), sol(1), sol(2));
16     err(0) = map.fit(P, dist, theta, count);
17 }
18
19
20
21 //Function to check on RPLIDAR health status
22 bool checkRPLIDARHealth(rp::standalone::rplidar::RPlidarDriver * drv)
23 {
24     u_result     op_result;
25     rplidar_response_device_health_t healthinfo;
```

```
26
27      op_result = drv->getHealth(healthinfo);
28      if (IS_OK(op_result)) { // the macro IS_OK is the preperred way to
            judge whether the operation is succeed.
29          printf("RPLidar health status : %d\n", healthinfo.status);
30          if (healthinfo.status == RPLIDAR_STATUS_ERROR) {
31              fprintf(stderr, "Error, rplidar internal error detected.
                    Please reboot the device to retry.\n");
32              // enable the following code if you want rplidar to be
                    reboot by software
33               drv->reset();
34              return false;
35          } else {
36              return true;
37          }
38
39      } else {
40          fprintf(stderr, "Error, cannot retrieve the lidar health code:
                %x\n", op_result);
41          return false;
42      }
43  }
44
45  void Work::init()
46  {
47      running_ = false;
48      started_ = false;
49      finished_ = false;
50      error_ = false;
51      save_ = false;
52      levmarq_ = true;
53      start_ = 0;
54      ignore_human_ = true;
55  }
56
57
58  void Work::run()
59  {
60      u_result op_result; //Variable to check the result of a complete
            reading.
61      count_readings_ = 0; //Number of total readings from the start of
            the creation of the map.
62      double dist[8192]; //Buffer to store the distance of rays in a
            reading.
63      double theta[8192]; //Buffer to store the angle from where a ray
            was taken in a reading.
64      slam::Map map(width_, height_); //The map that will be created.
65
```

```
66      // create the driver instance
67     rp::standalone::rplidar::RPlidarDriver * drv = rp::standalone::
           rplidar::RPlidarDriver::CreateDriver(rp::standalone::rplidar::
           DRIVER_TYPE_SERIALPORT);
68     if (!drv) {
69         fprintf(stderr, "insufficent memory, exit\n");
70         exit(-2);
71     }
72
73     rplidar_response_device_info_t devinfo;
74     bool connectSuccess = false;
75     // make connection...
76         if(!drv)
77             drv = rp::standalone::rplidar::RPlidarDriver::CreateDriver(
                   rp::standalone::rplidar::DRIVER_TYPE_SERIALPORT);
78         else
79             drv->reset();
80         if (IS_OK(drv->connect(port_, baudrate_)))
81         {
82             op_result = drv->getDeviceInfo(devinfo);
83
84             if (IS_OK(op_result))
85             {
86                 connectSuccess = true;
87             }
88             else
89             {
90                 delete drv;
91                 drv = NULL;
92             }
93         }
94
95
96     //If it was not able to connect, there was a problem with the port.
97     if (!connectSuccess) {
98         emit error_port();
99         error_ = true;
100    }
101    else
102    {
103        // print out the device serial number, firmware and hardware
               version number..
104        printf("RPLIDAR S/N: ");
105        for (int pos = 0; pos < 16 ;++pos) {
106            printf("%02X", devinfo.serialnum[pos]);
107        }
108
109        printf("\n"
```

```
110                 "Firmware Ver: %d.%02d\n"
111                 "Hardware Rev: %d\n"
112                 , devinfo.firmware_version>>8
113                 , devinfo.firmware_version & 0xFF
114                 , (int)devinfo.hardware_version);
115     }
116 
117 
118     // check health...
119     if (!error_)
120     {
121         if (!checkRPLIDARHealth(drv)) {
122             error_ = true;
123         }
124     }
125 
126 
127 
128 
129     //If there were not errors, the creating of the map starts.
130     if(!error_)
131     {
132         drv->startMotor();
133         // start scan...
134         drv->startScan(0,1);
135         running_ = true;
136         readings_ =(rplidar_response_measurement_node_hq_t*) std::
               malloc(sizeof(rplidar_response_measurement_node_hq_t)
               *200000);
137     }
138     else
139     {
140         started_ = false;
141         running_ = false;
142         error_ = false;
143     }
144 
145      while(running_)
146      {
147 
148          rplidar_response_measurement_node_hq_t nodes[8192]; //Readings
                from a complete spin.
149          size_t   count = sizeof(nodes)/sizeof(
               rplidar_response_measurement_node_hq_t); //Number of
               readings per spin.
150          int x=0,y=0; //Variables to calculate where to print a line in
                the map.
151 
```

```
152
153
154         op_result = drv->grabScanDataHq(nodes, count);
155
156         if (IS_OK(op_result)) {
157
158             drv->ascendScanData(nodes, count); //Sorts the measures by
                    angle
159
160
161             map.undrawSystem(P_);
162
163             //Calculates the distance in cm and the angle in degrees.
164             //It also saves the readings in the buffer for saving them
                    later.
165             for (int pos = 0; pos < (int)count ; ++pos) {
166                 dist[pos] = nodes[pos].dist_mm_q2 / 10.f / (1 << 2);
167                 theta[pos] = nodes[pos].angle_z_q14 * 90.f / (1 << 14)
                        ;
168                 count_readings_++;
169                 if (count_readings_ < 200000)
170                 {
171                     readings_[count_readings_].angle_z_q14 = nodes[pos
                            ].angle_z_q14;
172                     readings_[count_readings_].dist_mm_q2 = nodes[pos
                            ].dist_mm_q2;
173                     readings_[count_readings_].quality = nodes[pos].
                            quality;
174                     readings_[count_readings_].flag = nodes[pos].flag;
175                 }
176             }
177
178
179             //If is not the first frame, the algorithm begins.
180             if (start_ == 1)
181             {
182                 //If the selected algorithm is Levenberg Marquardt,
                        then a solver is created.
183                 //The params for the solver would be to stop after
                        2000 iterations or after the error is under
                        0.000001.
184                 //The variation of each variable (x,y,angle) is algo
                        set.
185                 if (levmarq_)
186                 {
187                     aruco::LevMarq<double> Solver;
188                     aruco::LevMarq<double>::eVector sol(3);
189                     Solver.setParams(500,0.000001);
```

```
190                     sol(0) = P_.getX();
191                     sol(1) = P_.getY();
192                     sol(2) = P_.getAngle();
193                     Solver.setDervEpsilon({x_var_, y_var_, angle_var_
                            });
194                     Solver.solve(sol, bind(error_2, std::placeholders
                            ::_1, std::placeholders::_2, map, dist, theta,
                            count));
195                     P_.setX(sol(0));
196                     P_.setY(sol(1));
197                     P_.setAngle(sol(2));
198                 }
199                 //If brute force is the chosen algorithm, the software
                        will execute this code.
200                 else
201                 {
202                     map.bruteForce(P_, dist, theta, count, range_x_,
                            range_y_, range_angle_, x_var_, y_var_,
                            angle_var_);
203                 }
204
205
206             }
207
208             //Once the position is estimated, this code calculates
                    where to print lines using trigonometry to obtain the x
                    and y coordinates in the map and prints the lines.
209             for (int pos = 0; pos < (int)count ; ++pos)
210             {
211                 //If the software should ignore the human, then
                        ignores some angles of the readings.
212                 if (ignore_human_)
213                 {
214                     if(theta[pos] >125 || theta[pos] < 55)
215                     {
216                         x = (cos( (theta[pos]+P_.getAngle()) * PI /
                                180.0 ) * dist[pos]) + P_.getX();
217                         y = (sin( (theta[pos]+P_.getAngle()) * PI /
                                180.0 ) * dist[pos]) + P_.getY();
218
219                         if (x <= width_ && y <= height_ && dist[pos
                                ]>0.1)
220                             map.lineToObject(P_.getX(), P_.getY(), x,
                                    y, w_);
221                     }
222                 }
223                 else
224                 {
```

```
225                     x = (cos( (theta[pos]+P_.getAngle()) * PI / 180.0
                            ) * dist[pos]) + P_.getX();
226                     y = (sin( (theta[pos]+P_.getAngle()) * PI / 180.0
                            ) * dist[pos]) + P_.getY();
227
228                     if (x <= width_ && y <= height_ && dist[pos]>0.1)
229                         map.lineToObject(P_.getX(), P_.getY(), x, y,
                                w_);
230                 }
231             }
232
233
234             //The map is updated with the new position of the system
                    and the new lines printed.
235             map.update(P_);
236
237             //From now on, the software will know that it is not the
                    first frame.
238             start_ = 1;
239
240             //The image is sent to the main window as a pixmap to be
                    printed in the Qlabel.
241             mat_ = map.getMap();
242             cv::resize(mat_,mat_,cv::Size(800,800));
243             QImage img((const uchar*)mat_.data, mat_.cols, mat_.rows,
                    mat_.step, QImage::Format_RGB888);
244             emit(print_img(img.copy()));
245
246
247             //The image is saved into the mat_ buffer in order to save
                     it from the main window while save_ is active
248             mat_ = map.getMap();
249             while(save_){}
250         }
251     }
252
253     drv->stop();
254     drv->stopMotor();
255     rp::standalone::rplidar::RPlidarDriver::DisposeDriver(drv);
256     drv = NULL;
257     return;
258 }
259
260
261 void Work::set_parameters(slam::Position p, int width, int height, int
        x_var, int y_var, double angle_var, double w, int baudrate, char *
       port, bool levmarq, int range_x, int range_y, int range_angle, bool
        ignore_human)
```

```
262 {
263     P_.setX(p.getX());
264     P_.setY(p.getY());
265     P_.setAngle(p.getAngle());
266     x_var_= x_var;
267     y_var_= y_var;
268     width_= width;
269     height_ =height;
270     w_ = w;
271     angle_var_ = angle_var;
272     baudrate_ = baudrate;
273     levmarq_ = levmarq;
274     range_x_ = range_x;
275     range_y_ = range_y;
276     range_angle_ = range_angle;
277     ignore_human_ = ignore_human;
278     port_ = (char*) std::malloc(sizeof(char)*256);
279     strcpy(port_, port);
280 }
281
282
283 void Work::clear()
284 {
285     //Default values for the parameters and free memory.
286     set_parameters(slam::Position(400,400,0), 800, 800, 1, 1, 1.5,
           0.7, 115200, "/dev/ttyUSB0", true, 20, 20, 60, true);
287     running_ = false;
288     start_ = 0;
289     error_ = false;
290     started_ = false;
291     finished_ = false;
292     count_readings_ = 0;
293
294
295     for (int i = 0; i < 200000; i++)
296     {
297         readings_[i].angle_z_q14 = 0;
298         readings_[i].dist_mm_q2 = 0;
299         readings_[i].quality = 0;
300         readings_[i].flag = 0;
301     }
302
303     delete[] readings_;
304 }
```

Capítulo 7

Simulaciones

7.1. simulation.h

Listing 7.1: simulation.h

```
1  #ifndef SIMULATION_H
2  #define SIMULATION_H
3
4  #include<QObject>
5  #include<QRunnable>
6  #include<unistd.h>
7  #include<QTimer>
8  #include<QThread>
9  #include <QImage>
10 #include "sdk/include/rplidar.h"
11 #include "slam.hpp"
12 #include "levmarq.h"
13
14 class Simulation : public QObject, public QRunnable
15 {
16     Q_OBJECT
17
18 private:
19     bool running_; //Represents whether the simulation is running or
          not.
20     bool started_; //Represents whether the simulation has started or
          not.
21     bool finished_; //Represents whether the simulation has finished
          or not.
22     bool loaded_; //Represents whether the data of a simulation has
          been loaded or not.
23     bool levmarq_; //Represents whether to use Levenberg Marquardt
          algorithm or brute force.
24     bool ignore_human_; //Represents wheter to ignore the human or not
           while mapping.
```

```
25      int start_; //Represents whether it is the first frame or not.
26      slam::Position P_;//Position of the system.
27
28      //Variables that store the parameters needed by the SLAM
           algorithms.
29          //range_... is used by brute force and means the range where
               to try the position of the system.
30          //..._var_ is used by Levenberg Marquardt and means the
               variation of the variable between two estimations.
31          //The speed of the simulation will be proportional to the
               variable speed_.
32          //w_ represents the weight of the previous value of a pixel.
33      int width_,height_, count_, range_x_, range_y_, range_angle_;
34      double x_var_, y_var_, angle_var_,w_, speed_;
35      cv::Mat mat_; //Used for sending the visual map as image to the
           main window.
36
37
38  public:
39      rplidar_response_measurement_node_hq_t * readings_; //Readings
           from the LiDAR read from file.
40
41      void init(); //Initialises bool variables.
42      void run(); //Function that will run as a thread.
43
44
45      //Observers
46
47      //Returns whether the simulation is running or not.
48      inline bool running() const {return this->running_;}
49
50      //Returns whether the creation of a map has started or not.
51      inline bool started() const {return this->started_;}
52
53      //Returns whether the creation of a map has finished or not.
54      inline bool finished() const {return this->finished_;}
55
56      //Returns whether the data for simulation has been loaded or not.
57      inline bool loaded() const {return this->loaded_;}
58
59      //Returns whether the algorithm used is Levenberg Marquart or
           Brute force.
60          // 1 for levmarq, 0 for brute force.
61      inline bool levmarq() const {return this->levmarq_;}
62
63      //Returns whether it is the first frame or not.
64      inline int start() const {return this->start_;}
65
```

```
66      //Returns the total number of readings read from the simulation
            file.
67      inline int getCount() const {return this->count_;}
68
69      //Returns the speed at which the simulation is run.
70      inline double getSpeed() const {return this->speed_;}
71
72
73      //Modifiers
74
75      //Allows to set the state of the simulation to running or not
            running.
76      inline void setRunning(const bool &running) {this->running_ =
            running;}
77
78      //Allows to set the state of the simulation to started or not
            started.
79      inline void setStarted(const bool &started) {this->started_ =
            started;}
80
81      //Allows to set the state of the simulationcreation of a map to
            finished or unfinished.
82      inline void setFinished(const bool &finished) {this->finished_ =
            finished;}
83
84      //Allows to set the state of the simulation to loaded or not
            loaded.
85      inline void setLoaded(const bool &loaded) {this->loaded_ = loaded
            ;}
86
87      //Sets the algorithm as levmarq or brute force.
88      inline void setLevmarq(const bool &levmarq) {this->levmarq_ =
            levmarq;}
89
90      //Allows to set the variable start_ to check if it is the first
            frame or not.
91      inline void setStart(const int &start) {this->start_ = start;}
92
93      //Sets the number of readings from the LiDAR read from the
            simulation file.
94      inline void setCount(const int &count) {this->count_ = count;}
95
96      //Sets the speed of the simulation.
97      inline void setSpeed(const double &speed) {this->speed_ = speed;}
98
99      //Sets the parameters needed by the SLAM algorithms.
100     void set_parameters(slam::Position p, int width, int height, int
            x_var, int y_var, double angle_var, double w, double speed,
```

```
        bool levmarq, int range_x, int range_y, int range_angle, bool
        ignore_human);
101
102     void clear(); //Frees memory and sets default values to parameters
        .
103
104
105 signals:
106     void print_img(const QImage &img); //This signal is sent every
        time the map is updated and ready to be printed.
107     void sim_finished(); //Signal sent to make the main window aware
        that the simulations has finished.
108
109 };
110
111 #endif // SIMULATION_H
```

7.2. simulation.cpp

Listing 7.2: simulation.cpp

```
1  #include "simulation.h"
2  #include "slam.hpp"
3  #include "levmarq.h"
4  #include<QThread>
5  #include<QMessageBox>
6
7
8  //Error function to optimize using Levenberg Marquardt algorithm.
9  void error(const aruco::LevMarq<double>::eVector &sol, aruco::LevMarq<
       double>::eVector &err, const slam::Map &map, const double *dist,
       const double *theta, const int count)
10 {
11     err.resize(1);
12     slam::Position P(sol(0), sol(1), sol(2));
13     err(0) = map.fit(P, dist, theta, count);
14 }
15
16
17 void Simulation::init()
18 {
19     running_ = false;
20     started_ = false;
21     finished_ = false;
22     loaded_ = false;
23     start_ = 0;
24     ignore_human_ = true;
25 }
```

```
26
27
28 void Simulation::run()
29 {
30
31     slam::Map map(width_,height_); //Map that will be created.
32     int   count = count_; //Count of reading stages.
33     int pos = 0, prev_pos = 0; //Counters for managing readings.
34     int x = 0, y = 0; //Variables to calculate where to print a line
          in the map.
35     double *dist_ =(double*) std::malloc(sizeof(double)*8192); //
          Vector of distances from the laser ray.
36     double *theta_ =(double*) std::malloc(sizeof(double)*8192); //
          Vector of angles from the laser ray.
37
38
39
40     //While the map is not finished and there are still readings (pos
          < count).
41     while(pos < count && !finished_)
42     {
43         //while the simulation is stopped, save the map.
44         while(!running_ && !map.getMap().empty() && !finished_){cv::
              imwrite("map_sim_stop.png",map.getMap());}
45         //If the user has reset while the simulation was stopped,
              leave.
46         if (!started_){break;}
47
48         //Undraw the system from the map in order to clear that space.
49         map.undrawSystem(P_);
50
51         //Manage readings.
52         prev_pos = pos;
53         //Total readings in this stage.
54         int t_stg_readings = 0;
55
56         //While the angle is not superior to 358, get the angle and
              the distance and save it into a buffer.
57         for (t_stg_readings = 0; readings_[pos].angle_z_q14 * 90.f /
              (1 << 14) < 358.0 ; t_stg_readings++)
58         {
59             //Angle in degrees
60             theta_[t_stg_readings] = readings_[pos].angle_z_q14 * 90.f
                  / (1 << 14);
61             //Distance in cm
62             dist_[t_stg_readings] = readings_[pos].dist_mm_q2 / 10.f /
                  (1 << 2);
63             pos++;
```

```
64          }
65          //If it is not the first reading and there has been readings (
               pos-prev_pos != 0), the position is estimated.
66          if (start_ == 1 && pos-prev_pos != 0)
67          {
68              //If the selected algorithm is Levenberg Marquardt, then a
                   solver is created.
69              //The params for the solver would be to stop after 2000
                   iterations or after the error is under 0.000001.
70              //The variation of each variable (x,y,angle) is algo set.
71              if (levmarq_)
72              {
73                  aruco::LevMarq<double> Solver;
74                  aruco::LevMarq<double>::eVector sol(3);
75                  Solver.setParams(2000,0.000001);
76                  sol(0) = P_.getX();
77                  sol(1) = P_.getY();
78                  sol(2) = P_.getAngle();
79                  Solver.setDervEpsilon({x_var_, y_var_, angle_var_});
80                  Solver.solve(sol, bind(error, std::placeholders::_1,
                       std::placeholders::_2, map, dist_, theta_, pos-
                       prev_pos));
81                  P_.setX(sol(0));
82                  P_.setY(sol(1));
83                  P_.setAngle(sol(2));
84              }
85              else
86              //If brute force is the chosen algorithm, the software
                   will execute this code.
87              {
88                  map.bruteForce(P_, dist_, theta_, pos-prev_pos,
                       range_x_, range_y_, range_angle_, x_var_, y_var_,
                       angle_var_);
89              }
90          }
91
92          //Once the position is estimated, this code calculates where
               to print lines using trigonometry to obtain the x and y
               coordinates in the map and prints the lines.
93          for (int i = 0; i < t_stg_readings ; i++)
94          {
95              //If the software should ignore the human, then ignores
                   some angles of the readings.
96              if (ignore_human_)
97              {
98                  if(theta_[i] >125 || theta_[i] < 55)
99                  {
100                     x = (cos( (theta_[i]+P_.getAngle()) * PI / 180.0 )
```

```
                    * dist_[i]) + P_.getX();
101                 y = (sin( (theta_[i]+P_.getAngle()) * PI / 180.0 )
                    * dist_[i]) + P_.getY();
102
103                 if (x <= 1000 && y <= 1000 && map.getPixel(x,y)
                    [2]<192 && dist_[i]>0.1)
104                     map.lineToObject(P_.getX(), P_.getY(), x, y,
                        w_);
105             }
106         }
107         else
108         {
109             x = (cos( (theta_[i]+P_.getAngle()) * PI / 180.0 ) *
                    dist_[i]) + P_.getX();
110             y = (sin( (theta_[i]+P_.getAngle()) * PI / 180.0 ) *
                    dist_[i]) + P_.getY();
111             //printf("theta (degrees): %03.2f Dist (cm): %08.2f
                    Pos: %d X: %d Y: %d \n", theta, dist[pos], pos,
                    points[pos].x,points[pos].y);
112
113             if (x <= 1000 && y <= 1000 && map.getPixel(x,y)[2]<192
                    && dist_[i]>0.1)
114                 map.lineToObject(P_.getX(), P_.getY(), x, y, w_);
115         }
116     }
117
118     //The map is updated with the new position of the system and
            the new lines printed.
119     map.update(P_);
120
121     //The image is sent to the main window as a pixmap to be
            printed in the Qlabel.
122     mat_ = map.getMap();
123     cv::resize(mat_,mat_,cv::Size(800,800));
124
125     QImage img((const uchar*)mat_.data, mat_.cols, mat_.rows, mat_
        .step, QImage::Format_RGB888);
126     emit(print_img(img.copy()));
127
128     //The time this process is asleep is proportional to the speed
            that the users sets.
129     usleep((10000/speed_)*1000);
130
131     //From now on, the software will know it is not the first
            frame.
132     start_ = 1;
133
134     //If there were some readings regarding, this code ignores
```

```
            them.
135         while (readings_[pos].angle_z_q14 * 90.f / (1 << 14) > 358.0)
136             pos++;
137     }
138
139
140     //If pos >= count it means the simulation should finish as there
            are no more readings regarding.
141     //The image of the map is saved.
142     if (pos >= count)
143     {
144         running_ = false;
145         finished_ = true;
146         emit sim_finished();
147         cv::imwrite("map_sim_finished.png",map.getMap());
148     }
149
150 }
151
152
153
154 void Simulation::set_parameters(slam::Position p, int width, int
        height, int x_var, int y_var, double angle_var, double w, double
        speed, bool levmarq, int range_x, int range_y, int range_angle,
        bool ignore_human)
155 {
156     P_.setX(p.getX());
157     P_.setY(p.getY());
158     P_.setAngle(p.getAngle());
159     x_var_= x_var;
160     y_var_= y_var;
161     width_= width;
162     height_ =height;
163     w_ = w;
164     angle_var_ = angle_var;
165     speed_ = speed;
166     range_x_ = range_x;
167     range_y_ = range_y;
168     range_angle_ = range_angle;
169     levmarq_ = levmarq;
170     ignore_human_ = ignore_human;
171 }
172
173 void Simulation::clear()
174 {
175     //Default values for parameters and free memory.
176     set_parameters(slam::Position(400,400,0), 800, 800, 1, 1, 1.5,
            0.7, 100, true, 20, 20, 60, true);
```

```
177     running_ = false;
178     finished_ = false;
179     loaded_ = false;
180     start_ = 0;
181     started_ = false;
182     count_ = 0;
183 
184     for (int i = 0; i < 200000; i++)
185     {
186         readings_[i].angle_z_q14 = 0;
187         readings_[i].dist_mm_q2 = 0;
188         readings_[i].quality = 0;
189         readings_[i].flag = 0;
190     }
191 
192     delete[] readings_;
193 }
```

Capítulo 8

Funcionalidades para SLAM

8.1. slam.hpp

Listing 8.1: slam.hpp

```
1  #ifndef _SLAMHPP_
2  #define _SLAMHPP_
3
4  #include <opencv2/highgui.hpp>
5  #include <opencv2/core/core.hpp>
6  #include <opencv2/imgproc.hpp>
7  #include <cstdlib>
8
9
10 #define PI 3.14159265
11
12
13 namespace slam
14 {
15     //Clas to store and manage the position of a system in order to
          implement SLAM.
16     class Position
17     {
18         private:
19             int x_, y_; //Stores X and Y variables of the system.
20             double angle_; //Stores the angle of the system.
21
22         public:
23
24         //Constructors
25             Position(int x=0, int y=0, double angle=0) //Default
                contrstuctor. Accepts the three components of the
                position.
26             {
27                 this->setX(x);
```

```
28                  this->setY(y);
29                  this->setAngle(angle);
30              }
31
32              Position(const Position & p) //Overloaded constructor,
                    accepts another position as parameter.
33              {
34                  this->setX(p.getX());
35                  this->setY(p.getY());
36                  this->setAngle(p.getAngle());
37              }
38
39          //Observers
40              inline const int & getX() const {return this-> x_;} //
                    Returns the component X of the system.
41              inline const int & getY() const {return this-> y_;} //
                    Returns the component Y of the system.
42              inline const double & getAngle() const {return this->
                    angle_;} //Returns the component angle of the system.
43
44          //Modifiers
45              //Sets the component X of the system. Recieves a const int
                    by reference.
46              inline void setX(const int & x) {this->x_ = x;}
47              //Sets the component Y of the system. Recieves a const int
                    by reference.
48              inline void setY(const int & y) {this->y_ = y;}
49              //Sets the component angle of the system. Recieves a const
                    int by reference.
50              inline void setAngle(const double & angle) {this->angle_ =
                    angle;}
51
52              //Operator to allow assignment using '='.
53              slam::Position &operator = (const slam::Position p)
54              {
55                  this->setX(p.getX());
56                  this->setY(p.getY());
57                  this->setAngle(p.getAngle());
58                  return *this;
59              }
60      };
61
62      //Class to store and perform SLAM using a occupation matrix (oc_)
            and a visual matrix (map_).
63      //Needs a slam::Position in order to locate the system.
64      //The occupation work as follows:
65          // 0 for unknown space.
66          // 1 to 255 for %of occupation.  1 means low occupation and
```

```
            255 means high occupation.
67      class Map
68      {
69          private:
70              cv::Mat map_;
71              cv::Mat oc_;
72
73          public:
74
75          //Constructors
76              //Default constructor given X (width) and Y(height)
                    components.
77              //Initiates every pixel to 0;
78              Map(const int &x = 800, const int &y = 800)
79              {
80                  cv::Mat map(x, y, CV_8UC3);
81                  this->setMap(map);
82                  cv::Mat oc(x, y, CV_8UC1);
83                  this->oc_ = oc;
84
85                  for(int i=0;i<this->getRows();i++)
86                      for(int j=0;j<this->getCols();j++)
87                      {
88                          this->setValues(i,j,0,0,0);
89                          this->setValuesOc(i,j,0);
90                      }
91              }
92
93              Map(cv::Mat map) {this->setMap(map);} //Overloaded
                    constructor given a map
94
95          //Observers
96              inline cv::Mat getMap() const {return this-> map_;} //
                    Returns the visual map as cv::Mat.
97              inline cv::Mat getOc() const {return this-> oc_;} //
                    Returns the occupation map as cv::Mat.
98              inline int getRows() const {return this->map_.rows;} //
                    Returns the rows (height) as int.
99              inline int getCols() const {return this->map_.cols;} //
                    Returns the columns (width) as int.
100
101             //Returns the values for a pixel from the visual map using
                     the coordinates.
102             //If the map is bigger than the point, it returns the
                    point. If not, it returns the first point of the map.
103             inline const cv::Vec3b & getPixel(const int x, const int y
                    ) const
104             {
```

```
105             if (this->getRows() >= y && this->getCols() >= x)
106                 return map_.at<cv::Vec3b>(y,x);
107             else
108                 return map_.at<cv::Vec3b>(0,0);
109
110         }
111
112         //Overloaded function to obtain the values of a pixel
                using a Points instead of the coordinates.
113         //If the map is bigger than the point, it returns the
                point. If not, it returns the first point of the map.
114         inline const cv::Vec3b & getPixel(const cv::Point & pos)
                const
115         {
116             if (this->getRows() >= pos.y && this->getCols() >= pos
                    .x)
117                 return map_.at<cv::Vec3b>(pos);
118             else
119                 return map_.at<cv::Vec3b>(0,0);
120
121         }
122
123         //Returns the value of a channel of a pixel using the
                coordinates and the channel defined by an int.
124         //If the map is bigger than the point, it returns the
                point. If not, it returns -1.
125         int getPixel_channel(const int x, const int y, const int
                channel) const
126         {
127             if (this->getRows() >= y && this->getCols() >= x)
128                 return this->map_.at<cv::Vec3b>(y,x)[channel];
129             else
130                 return -1;
131         }
132
133         //Returns the value of a channel of a pixel using the
                coordinates and the channel defined by a char.
134         //If the map is bigger than the point, it returns the
                point. If not, it returns -1.
135         int getPixel_channel(const int x, const int y, const char
                channel) const
136         {
137             if (channel == 'b' && this->getRows() >= y && this->
                    getCols() >= x)
138                 return this->map_.at<cv::Vec3b>(y,x)[0];
139             else if (channel == 'g' && this->getRows() >= y &&
                    this->getCols() >= x)
140                 return this->map_.at<cv::Vec3b>(y,x)[1];
```

```
141                 else if (channel == 'r' && this->getRows() >= y &&
                        this->getCols() >= x)
142                     return this->map_.at<cv::Vec3b>(y,x)[2];
143                 else
144                     return -1;
145             }
146 
147 
148             //Returns the values for a pixel from the visual map using
                    the coordinates.
149             //If the map is bigger than the point, it returns the
                    point. If not, it returns the first point of the map.
150             inline const uchar & getPixelOc(const int x, const int y)
                    const
151             {
152                 if (this->getRows() >= y && this->getCols() >= x && y
                        >=0 && x>=0)
153                     return oc_.at<uchar>(y,x);
154                 else
155                     return oc_.at<uchar>(0,0);
156 
157             }
158 
159             //Overloaded function to obtain the values of a pixel
                    using a Points instead of the coordinates.
160             //If the map is bigger than the point, it returns the
                    point. If not, it returns the first point of the map.
161             inline const uchar & getPixelOc(const cv::Point & pos)
                    const
162             {
163                 if (this->getRows() >= pos.y && this->getCols() >= pos
                        .x && pos.y>=0 && pos.x>=0)
164                     return oc_.at<uchar>(pos);
165                 else
166                     return oc_.at<uchar>(0,0);
167 
168             }
169 
170         //Modifiers
171             inline void setMap(const cv::Mat & map) {this->map_ = map
                    ;} //Assigns the visual map to the specified cv::Mat.
172             inline void setOc(const cv::Mat & map) {this->oc_ = map;}
                    //Assigns the occupation map to the specified cv::Mat.
173 
174             //Frees the memory of the two maps.
175             inline void release(){
176                 map_.release();
177                 oc_.release();
```

```
178             }
179
180             //Resizes both maps according to the specified width and
                    height.
181             inline void resize(const int &width, const int &height){
182                 map_.create(width, height, CV_8UC3);
183                 oc_.create(width, height, CV_8UC1);
184             }
185
186             //Sets the rgb values of a pixel in the visual map given
                    the values and coordinates.
187             inline void setValues(const int & x, const int & y, const
                    int & r, const int & g, const int & b)
188             {
189                 if (this->getRows() >= y && this->getCols() >= x && y
                        >=0 && x>=0)
190                 {
191                     map_.at<cv::Vec3b>(y,x)[0] = b;
192                     map_.at<cv::Vec3b>(y,x)[1] = g;
193                     map_.at<cv::Vec3b>(y,x)[2] = r;
194                 }
195
196             }
197
198             //Sets the rgb values of a pixel in the visual map given
                    the values and the cv::Point.
199             inline void setValues(const cv::Point & pos, const int & r
                    , const int & g, const int & b)
200             {
201                 if (this->getRows() >= pos.y && this->getCols() >= pos
                        .x && pos.y>=0 && pos.x>=0)
202                 {
203                     map_.at<cv::Vec3b>(pos)[0] = b;
204                     map_.at<cv::Vec3b>(pos)[1] = g;
205                     map_.at<cv::Vec3b>(pos)[2] = r;
206                 }
207             }
208
209             //Sets the occupation value of a pixel in the occuation
                    map given the value and coordinates.
210             inline void setValuesOc(const int & x, const int & y,
                    const int & value)
211             {
212                 if (this->getRows() >= y && this->getCols() >= x && y
                        >=0 && x>=0)
213                 {
214                     oc_.at<uchar>(y,x) = value;
215                 }
```

```
216
217             }
218
219             //Sets the occupation value of a pixel in the occuation
                    map given the value and the cv::Point.
220             inline void setValuesOc(const cv::Point & pos, const int &
                    value)
221             {
222                 if (this->getRows() >= pos.y && this->getCols() >= pos
                        .x && pos.y>=0 && pos.x>=0)
223                 {
224                     oc_.at<uchar>(pos) = value;
225                 }
226             }
227
228             //Sets all the values for all the pixels on both matrixes
                    to 0.
229             inline void set_zeros()
230             {
231                 for(int i=0;i<this->getRows();i++)
232                     for(int j=0;j<this->getCols();j++)
233                     {
234                         this->setValues(i,j,0,0,0);
235                         this->setValuesOc(i,j,0);
236                     }
237             }
238
239
240
241             //Function to draw a line in the map from the position of
                    the system to an objectgiven two points and the weight
                    of last value.
242             void lineToObject(const int & x0, const int & y0, const
                    int & x1, const int & y1, const double & w)
243             {
244                 if(x0 > 0 && x0 < getCols() && y0 > 0 && y0 < getRows
                        () && x1 > 0 && x1 < getCols() && y1 > 0 && y1 <
                        getRows() && w >= 0 && w <= 1)
245                 {
246                     cv::Point p0(x0,y0), p1(x1,y1);
247
248                     //Line iterator from opencv that implements the
                            Bresenham algorithm.
249                     cv::LineIterator it(oc_, p0, p1, 8);
250
251                     int value = 0;
252
253                     //The iterator is iterated to take into account
```

```
                    all the pixels in the line between the system
                    and the object.
254                 for(int i = 0; i < it.count - 3; i++, ++it)
255                 {
256                     value = rint(getPixelOc(it.pos())*w +
                            127*(1.0-w));
257                     setValuesOc(it.pos(), value);
258                 }
259
260                 value = rint(getPixelOc(it.pos())*w + 191*(1.0-w))
                        ;
261                 setValuesOc(it.pos(), value);
262                 it++;
263                 value = rint(getPixelOc(it.pos())*w + 223*(1.0-w))
                        ;
264                 setValuesOc(it.pos(), value);
265                 it++;
266                 value = rint(getPixelOc(it.pos())*w + 255*(1.0-w))
                        ;
267                 setValuesOc(it.pos(), value);
268                 it++;
269                 value = rint(getPixelOc(it.pos())*w + 223*(1.0-w))
                        ;
270                 setValuesOc(it.pos(), value);
271                 it++;
272                 value = rint(getPixelOc(it.pos())*w + 191*(1.0-w))
                        ;
273                 setValuesOc(it.pos(), value);
274             }
275
276         }
277
278         //Function to draw a line with a desired colour from one
               point to another
279         void line(const int x0, const int y0, const int x1, const
               int y1, const int r, const int g, const int b)
280         {
281             cv::Point p0(x0,y0), p1(x1,y1);
282
283             cv::LineIterator it(this->map_, p0, p1, 8);
284
285             for(int i = 0; i < it.count; i++, ++it)
286                 this->setValues(it.pos(), r, g, b);
287         }
288
289
290         //Function that calculates the fitness for a specified
               position according to the occupation map.
```

```
291             //Recieves the data necesary to compare the point with the
                    map, which are:
292                 //try_P: the point to evaluate.
293                 //Current_dist: the distance readings obtained from
                        the LiDAR.
294                 //theta: the angle from which the distances were taken
                        .
295                 //count: the number of distance readings.
296             //Returns the fitness from 0 to 1 where 0 means the best
                    fitness and 1 means the worst fitness.
297             double fit(const slam::Position try_P, const double *
                    current_dist, const double *theta,  int count) const
298             {
299                 double fitness = 0.0;
300                 int x = 0, y = 0;
301
302                 for (int pos = 0; pos < count; pos++)
303                 {
304                     //Si la medici n fue v lida (la distancia medida
                            fue mayor a 0.1), se tiene en cuenta.
305                     if (current_dist[pos] > 0.1)
306                     {
307                         x = (cos( (theta[pos]+try_P.getAngle()) * PI /
                                180.0 ) * current_dist[pos]) + try_P.getX
                                ();
308                         y = (sin( (theta[pos]+try_P.getAngle()) * PI /
                                180.0 ) * current_dist[pos]) + try_P.getY
                                ();
309                         if(x < getCols() && y < getRows())
310                             fitness+= ((int)getPixelOc(x,y));
311                     }
312                 }
313
314                 return (255.0*count - fitness)/(255.0*count);
315             }
316
317             //Function that applies brute force to estimate the
                    position and orientation of the system in the map.
318                 //P is the points to be calculated
319                 //dist is a buffer with the distances from the LiDAR
320                 //theta is a buffer with the angle from where the
                        distances were taken.
321                 //count is the number of rays or measures.
322                 //range_x is the range where to try with the X
                        component.
323                 //range_y is the range where to try with the Y
                        component.
324                 //range_angle is the range where to try with the angle
```

```
                component.
325         void bruteForce(slam::Position &P, const double *dist,
               const double *theta, const int &count, const int &
               range_x, const int &range_y, const int &range_angle,
               const int & x_var, const int &y_var, const double &
               angle_var)
326         {
327             //Variables to try and optimize the position of the
                   system in the map.
328             slam::Position try_P(P.getX()-20, P.getY()-20, P.
                   getAngle()-30);
329             double opt = 0.0;
330             double opt_try = 0.0;
331             slam::Position prev_P(P);
332             //First value of optimization.
333             opt = fit(prev_P, dist, theta, count);
334
335             //The following code will test the different positions
                    of de system in a given range.
336             //From (current position - range_y_/2) to (current
                   position + range_y/2)
337             for (int i = 0; i < range_y; i=i+y_var){
338                 try_P.setY(prev_P.getY() - range_y/2 + i);
339
340                 //From (current position - range_x_/2) to (current
                        position + range_x/2)
341                 for (int j = 0; j < range_x; j=j+x_var){
342                     try_P.setX(prev_P.getX() - range_x/2 + j);
343
344                     //From (current angle - range_angle_/2) to (
                           current angle + range_angle/2)
345                     for (int k = 0; k < range_angle; k=k+angle_var
                           ){
346                         try_P.setAngle(prev_P.getAngle()-
                               range_angle/2+k);
347                         opt_try = fit(try_P, dist, theta, count);
348
349                         //If the new value is better than the
                               previous one, the position is updated.
350                         if (opt_try < opt)
351                         {
352                             P = try_P;
353                             opt = opt_try;
354                         }
355                     }
356                 }
357             }
358         }
```

```
359
360
361
362             //Draws the system in the visual map.
363             void drawSystem(slam::Position P, const int r, const int g
                    , const int b)
364             {
365                 for (int i = 0; i < 5; i++)
366                     for (int j = 0; j < 5; j++)
367                         setValues(P.getX()-2+i,P.getY()-2+j, r, g, b);
368
369                 int x = (cos( (P.getAngle()+270) * PI / 180.0 ) * 10)
                        + P.getX();
370                 int y = (sin( (P.getAngle()+270) * PI / 180.0 ) * 10)
                        + P.getY();
371                 line(P.getX(), P.getY(), x, y, r, g, b);
372             }
373
374             //Undraws the system from the visual map.
375             void undrawSystem(slam::Position P)
376             {
377                 for (int i = 0; i < 5; i++)
378                     for (int j = 0; j < 5; j++)
379                         setValues(P.getX()-2+i,P.getY()-2+j, 60, 60,
                                60);
380
381                 int x = (cos( (P.getAngle()+270) * PI / 180.0 ) * 10)
                        + P.getX();
382                 int y = (sin( (P.getAngle()+270) * PI / 180.0 ) * 10)
                        + P.getY();
383                 line(P.getX(), P.getY(), x, y, 60, 60, 60);
384             }
385
386             //Updates the visual map according to the occupation map.
                    It also draws the system in the visual map.
387             void update(slam::Position P)
388             {
389                 for(int i=0;i<this->getRows();i++)
390                     for(int j=0;j<this->getCols();j++)
391                     {
392                         if (this->getPixelOc(i,j) != 0)
393                         {
394                             //If the value of occupation is under 161,
                                    we take it as unoccupied.
395                             if (this->getPixelOc(i,j) < 161)
396                                 this->setValues(i,j,60,60,60);
397                             //If the value is under 30, we take it as
                                unknown.
```

```
398                             else if (this->getPixelOc(i,j) < 30)
399                                 this->setValues(i,j,0,0,0);
400                             //If the value is over 222, we take it as
                                   occupied.
401                             else if (this->getPixelOc(i,j) > 222)
402                                 this->setValues(i,j,255,0,0);
403                             //If the value is between 161 and 222, we
                                   take it as parcially occupied.
404                             else if (this->getPixelOc(i,j) > 160)
405                                 this->setValues(i,j,80,0,0);
406                         }
407                     }
408                 this->drawSystem(P, 0, 0, 255);
409             }
410
411             //Operator to allow assignment using '='.
412             slam::Map &operator = (const slam::Map m)
413             {
414                 this->setMap(m.getMap());
415                 this->setOc(m.getOc());
416                 return *this;
417             }
418     };
419
420
421 }
422
423 #endif
```

Capítulo 9

Levenberg Marquardt

9.1. levmarq.h

Listing 9.1: levmarq.h

```
1  /**
2  Copyright 2017 Rafael Mu oz Salinas. All rights reserved.
3
4  Redistribution and use in source and binary forms, with or without
       modification, are
5  permitted provided that the following conditions are met:
6
7     1. Redistributions of source code must retain the above copyright
          notice, this list of
8        conditions and the following disclaimer.
9
10    2. Redistributions in binary form must reproduce the above
          copyright notice, this list
11       of conditions and the following disclaimer in the documentation
             and/or other materials
12       provided with the distribution.
13
14 THIS SOFTWARE IS PROVIDED BY Rafael Mu oz Salinas ''AS IS'' AND ANY
       EXPRESS OR IMPLIED
15 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
       MERCHANTABILITY AND
16 FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
       Rafael Mu oz Salinas OR
17 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
       EXEMPLARY, OR
18 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
       SUBSTITUTE GOODS OR
19 SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
       HOWEVER CAUSED AND ON
20 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
```

```
   TORT (INCLUDING
21 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
   SOFTWARE, EVEN IF
22 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
23
24 The views and conclusions contained in the software and documentation
   are those of the
25 authors and should not be interpreted as representing official
   policies, either expressed
26 or implied, of Rafael Mu oz Salinas.
27 */
28
29 #ifndef ARUCO_MM_LevMarq_H
30 #define ARUCO_MM_LevMarq_H
31 #include <Eigen/Core>
32 #include <Eigen/Cholesky>
33 #include <functional>
34 #include <iostream>
35 #include <cmath>
36 #include <ctime>
37 #include <cstring>
38 #include <vector>
39 #include <chrono>
40 #include <iomanip>
41 namespace aruco{
42 // Levenberg-Marquardt method for general problems Inspired in
43 //@MISC\{IMM2004-03215,
44 //    author       = "K. Madsen and H. B. Nielsen and O. Tingleff",
45 //    title        = "Methods for Non-Linear Least Squares Problems (2
   nd ed.)",
46 //    year         = "2004",
47 //    pages        = "60",
48 //    publisher    = "Informatics and Mathematical Modelling,
   Technical University of Denmark, {DTU}",
49 //    address      = "Richard Petersens Plads, Building 321, {DK-}2800
    Kgs. Lyngby",
50 //    url          = "http://www.ltu.se/cms_fs/1.51590!/
   nonlinear_least_squares.pdf"
51 //}
52 template<typename T>
53 class   LevMarq{
54 public:
55
56
57     typedef     Eigen::Matrix<T,Eigen::Dynamic,1>  eVector;
58     typedef     std::function<void(const eVector  &, eVector &)> F_z_x;
59     typedef     std::function<void(const eVector  &,  Eigen::Matrix<T,
        Eigen::Dynamic,Eigen::Dynamic> &)> F_z_J;
```

```
60      LevMarq();
61      /**
62      * @brief Constructor with parms
63      * @param maxIters maximum number of iterations of the algoritm
64      * @param minError to stop the algorithm before reaching the max
            iterations
65      * @param min_step_error_diff minimum error difference between two
            iterations. If below this level, then stop.
66      * @param tau parameter indicating how near the initial solution is
             estimated to be to the real one. If 1, it means that it is
            very far and the first
67      * @param der_epsilon increment to calculate the derivate of the
            evaluation function
68      * step will be very short. If near 0, means the opposite. This
            value is auto calculated in the subsequent iterations.
69      */
70      LevMarq(int maxIters,double minError,double min_step_error_diff=0,
            double tau=1 ,double der_epsilon=1e-3);
71
72      /**
73    * @brief setParams
74    * @param maxIters maximum number of iterations of the algoritm
75    * @param minError to stop the algorithm before reaching the max
        iterations
76    * @param min_step_error_diff minimum error difference between two
        iterations. If below this level, then stop.
77    * @param tau parameter indicating how near the initial solution is
        estimated to be to the real one. If 1, it means that it is very
        far and the first
78    * @param der_epsilon increment to calculate the derivate of the
        evaluation function
79    * step will be very short. If near 0, means the opposite. This value
        is auto calculated in the subsequent iterations.
80    */
81      void setParams(int maxIters,double minError,double
            min_step_error_diff=0,double tau=1 ,double der_epsilon=1e-3);
82      void setDervEpsilon(const std::vector<double> derEp);
83
84      /**
85    * @brief solve  non linear minimization problem ||F(z)||, where F(z)=
        f(z) f(z)^t
86    * @param z  function params 1xP to be estimated. input-output.
        Contains the result of the optimization
87    * @param f_z_x evaluation function  f(z)=x
88    *          first parameter : z :  input. Data is in double precision
        as a row vector (1xp)
89    *         second parameter : x :  output. Data must be returned in
        double
```

```
90  * @param f_J  computes the jacobian of f(z)
91  *          first parameter : z :  input. Data is in double precision
       as a row vector (1xp)
92  *          second parameter : J :  output. Data must be returned in
       double
93  * @return final error
94  */
95     double solve(  eVector  &z, F_z_x , F_z_J);
96     /// Step by step solve mode
97
98
99     /**
100     * @brief init initializes the search engine
101     * @param z
102     */
103    void init(eVector  &z, F_z_x );
104    /**
105     * @brief step gives a step of the search
106     * @param f_z_x error evaluation function
107     * @param f_z_J Jacobian function
108     * @return error of current solution
109     */
110    bool step(  F_z_x f_z_x , F_z_J  f_z_J);
111    bool step(  F_z_x f_z_x);
112    /**
113     * @brief getCurrentSolution returns the current solution
114     * @param z output
115     * @return error of the solution
116     */
117    double getCurrentSolution(eVector  &z);
118    /**
119     * @brief getBestSolution sets in z the best solution up to this
          moment
120     * @param z output
121     * @return  error of the solution
122     */
123    double getBestSolution(eVector  &z);
124
125    /**  Automatic jacobian estimation
126 * @brief solve  non linear minimization problem ||F(z)||, where F(z)=
       f(z) f(z)^t
127 * @param z  function params 1xP to be estimated. input-output.
       Contains the result of the optimization
128 * @param f_z_x evaluation function  f(z)=x
129 *          first parameter : z :  input. Data is in double precision
       as a row vector (1xp)
130 *          second parameter : x :  output. Data must be returned in
       double
```

```
131  * @return final error
132  */
133     double solve(  eVector  &z, F_z_x );
134     //to enable verbose mode
135     bool & verbose(){return _verbose;}
136 
137     //sets a callback func call at each step
138     void setStepCallBackFunc(std::function<void(const eVector  &)>
           callback){_step_callback=callback;}
139     //sets a function that indicates when the algorithm must be stop.
           returns true if must stop and false otherwise
140     void setStopFunction( std::function<bool(const eVector  &)>
           stop_function){_stopFunction=stop_function;}
141 
142     void calcDerivates(const eVector & z ,  Eigen::Matrix<T,Eigen::
           Dynamic,Eigen::Dynamic> &, F_z_x);
143 private:
144     int _maxIters;
145     double _minErrorAllowed,_der_epsilon,_tau,_min_step_error_diff;
146     bool _verbose;
147     //--------
148     eVector curr_z,x64;
149     std::vector<double> _devEpsilonV;
150     double currErr,prevErr,minErr ;
151     Eigen::Matrix<T,Eigen::Dynamic,Eigen::Dynamic> I,J;
152     double mu,v;
153     std::function<void(const eVector  &)> _step_callback;
154     std::function<bool(const eVector  &)> _stopFunction;
155 
156 
157 };
158 
159 
160 
161 template<typename T>
162 LevMarq<T>::LevMarq(){
163     _maxIters=1000;_minErrorAllowed=0;_der_epsilon=1e-3;_verbose=false
           ;_tau=1;v=5;_min_step_error_diff=0;
164 }
165 /**
166 * @brief Constructor with parms
167 * @param maxIters maximum number of iterations of the algoritm
168 * @param minError to stop the algorithm before reaching the max
      iterations
169 * @param min_step_error_diff minimum error difference between two
      iterations. If below this level, then stop.
170 * @param tau parameter indicating how near the initial solution is
      estimated to be to the real one. If 1, it means that it is very far
```

```
     and the first
171 * @param der_epsilon increment to calculate the derivate of the
     evaluation function
172 * step will be very short. If near 0, means the opposite. This value
     is auto calculated in the subsequent iterations.
173 */
174 template<typename T>
175 LevMarq<T>::LevMarq(int maxIters,double minError,double
     min_step_error_diff,double tau ,double der_epsilon ){
176     _maxIters=maxIters;_minErrorAllowed=minError;_der_epsilon=
        der_epsilon;_verbose=false;_tau=tau;v=5;_min_step_error_diff=
        min_step_error_diff;
177 }
178
179 /**
180 * @brief setParams
181 * @param maxIters maximum number of iterations of the algoritm
182 * @param minError to stop the algorithm before reaching the max
     iterations
183 * @param min_step_error_diff minimum error difference between two
     iterations. If below this level, then stop.
184 * @param tau parameter indicating how near the initial solution is
     estimated to be to the real one. If 1, it means that it is very far
     and the first
185 * @param der_epsilon increment to calculate the derivate of the
     evaluation function
186 * step will be very short. If near 0, means the opposite. This value
     is auto calculated in the subsequent iterations.
187 */
188 template<typename T>
189 void LevMarq<T>::setParams(int maxIters,double minError,double
     min_step_error_diff,double tau ,double der_epsilon){
190     _maxIters=maxIters;
191     _minErrorAllowed=minError;
192     _der_epsilon=der_epsilon;
193     _tau=tau;
194     _min_step_error_diff=min_step_error_diff;
195
196 }
197 template<typename T>
198 void LevMarq<T>::setDervEpsilon(const std::vector<double> derEp){
199     _devEpsilonV=derEp;
200 }
201
202 template<typename T>
203 void  LevMarq<T>:: calcDerivates(const eVector & z ,  Eigen::Matrix<T,
     Eigen::Dynamic, Eigen::Dynamic> &J ,  F_z_x f_z_x)
204 {
```

```
205     for (int i=0;i<z.rows();i++) {
206         eVector zp(z),zm(z);
207         double epsilon=_der_epsilon;
208         if(i<_devEpsilonV.size()) epsilon=_devEpsilonV[i];
209         zp(i)+=epsilon;
210         zm(i)-=epsilon;
211         eVector xp,xm;
212         f_z_x( zp,xp);
213         f_z_x( zm,xm);
214         eVector dif=(xp-xm)/(2.f*epsilon);
215         J.middleCols(i,1)=dif;
216     }
217 }
218
219 template<typename T>
220 double  LevMarq<T>:: solve(  eVector  &z, F_z_x f_z_x){
221     return solve(z,f_z_x,std::bind(&LevMarq::calcDerivates,this,std::
        placeholders::_1,std::placeholders::_2,f_z_x));
222 }
223 template<typename T>
224 bool  LevMarq<T>:: step(  F_z_x f_z_x){
225     return step(f_z_x,std::bind(&LevMarq::calcDerivates,this,std::
        placeholders::_1,std::placeholders::_2,f_z_x));
226 }
227
228 template<typename T>
229 void LevMarq<T>::init(eVector  &z, F_z_x f_z_x ){
230     curr_z=z;
231     I.resize(z.rows(),z.rows());
232     I.setIdentity();
233     f_z_x(curr_z,x64);
234     minErr=currErr=prevErr=x64.cwiseProduct(x64).sum();
235     J.resize(x64.rows(),z.rows());
236     mu=-1;
237
238
239 }
240
241
242 #define splm_get_time(a,b) std::chrono::duration_cast<std::chrono::
    duration<double>>(a-b).count()
243
244
245 template<typename T>
246 bool LevMarq<T>::step( F_z_x f_z_x, F_z_J f_J){
247
248     f_J(curr_z,J);
249     Eigen::Matrix<T,Eigen::Dynamic,Eigen::Dynamic> Jt=J.transpose();
```

```
250     Eigen::Matrix<T,Eigen::Dynamic,Eigen::Dynamic> JtJ=(Jt*J);
251
252     eVector  B=-Jt*x64;
253     if(mu<0){//first time only
254         int max=0;
255         for(int j=1;j<JtJ.cols();j++) if (JtJ(j,j)>JtJ(max,max)) max=j
                ;
256         mu=JtJ(max,max)*_tau;
257     }
258
259     double gain=0,prev_mu=0;
260     int ntries=0;
261     bool isStepAccepted=false;
262     do{
263         //add/update dumping factor to JtJ.
264         //very efficient in any case, but particularly if initial dump
              does not produce improvement and must reenter
265         for(int j=0;j<JtJ.cols();j++)  JtJ(j,j)    += mu-prev_mu;//
             update mu
266         prev_mu=mu;
267         eVector delta=   JtJ.ldlt().solve(B);
268         eVector  estimated_z=curr_z+delta;
269         //compute error
270         f_z_x(estimated_z,x64);
271         auto err=x64.cwiseProduct(x64).sum();
272         auto L=0.5*delta.transpose()*((mu*delta) - B);
273         gain= (err-prevErr)/ L(0,0) ;
274         //get gain
275         if (gain>0){
276             mu=mu*std::max(double(0.33),1.-pow(2*gain-1,3));
277             v=5.f;
278             currErr=err;
279             curr_z=estimated_z;
280             isStepAccepted=true;
281         }
282         else{ mu=mu*v; v=v*5;}
283
284     }while(gain<=0 && ntries++<5);
285
286     if (_verbose) std::cout<<std::setprecision(5) <<"Curr Error="<<
         currErr<<" AErr(prev-curr)="<<prevErr-currErr<<" gain="<<gain<<
         " dumping factor="<<mu<<std::endl;
287     //    //check if we must move to the new position or exit
288     if ( currErr<prevErr)
289         std::swap ( currErr,prevErr );
290
291      return isStepAccepted;
292
```

```
293 }
294
295
296 template<typename T>
297 double  LevMarq<T>:: getCurrentSolution(eVector  &z){
298
299     z=curr_z;
300     return currErr;
301 }
302 template<typename T>
303 double  LevMarq<T>::solve( eVector  &z, F_z_x  f_z_x, F_z_J f_J){
304
305     init(z,f_z_x);
306
307     if( _stopFunction){
308         do{
309             step(f_z_x,f_J);
310             if (_step_callback) _step_callback(curr_z);
311         }while(!_stopFunction(curr_z));
312
313     }
314     else{
315         //intial error estimation
316         int mustExit=0;
317         for ( int i = 0; i < _maxIters && !mustExit; i++ ) {
318             if (_verbose)std::cerr<<"iteration "<<i<<"/"<<_maxIters<<
                    "  ";
319             bool isStepAccepted=step(f_z_x,f_J);
320             //check if we must exit
321             if ( currErr<_minErrorAllowed ) mustExit=1;
322             if( fabs( prevErr -currErr)<=_min_step_error_diff  || !
                    isStepAccepted) mustExit=2;
323             //exit if error increment
324             if (currErr<prevErr  )mustExit=3;
325             //            if (  (prevErr-currErr)  < 1e-5 )  mustExit=
                    true;
326             if (_step_callback) _step_callback(curr_z);
327         }
328
329 //        std::cout<<"Exit code="<<mustExit<<std::endl;
330     }
331     z=curr_z;
332     return currErr;
333
334 }
335
336 }
337
```

```
338 #endif
```

Capítulo 10

Recursos

10.1. resources.qrc

Listing 10.1: resources.qrc

```
1 <!DOCTYPE RCC><RCC version="1.0">
2 <qresource>
3   <file>./pixil-layer-Background.png</file>
4 </qresource>
5 </RCC>
```