

```
1:  /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8:  #ifndef HARDWARESETTINGS_H
9:  #define HARDWARESETTINGS_H
10:
11:  #include <QDialog>
12:  #include "VisionSoil.h"
13:  #include <sys/utsname.h>
14:  #include <string>
15:
16:  namespace Ui {
17:  class HardwareSettings;
18:  }
19:
20:  class HardwareSettings : public QDialog {
21:      Q_OBJECT
22:
23:  public:
24:      explicit HardwareSettings(QWidget *parent = 0,
25:                               SoilAnalyzer::SoilSettings *soilsetting = 0);
26:      ~HardwareSettings();
27:
28:  private slots:
29:      void on_sb_HDRframes_editingFinished();
30:
31:      void on_sb_lightLevel_editingFinished();
32:
33:      void on_cb_encoderInv_clicked(bool checked);
34:
35:      void on_cb_enableRainbow_clicked(bool checked);
36:
37:  private:
38:      Ui::HardwareSettings *ui;
39:      SoilAnalyzer::SoilSettings *soilSetting = nullptr;
40:  };
41:
42: #endif // HARDWARESETTINGS_H
```

```
1:  /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8:  #include "hardwaresettings.h"
9:  #include "ui_hardwaresettings.h"
10:
11: HardwareSettings::HardwareSettings(QWidget *parent,
12:                                     SoilAnalyzer::SoilSettings *soilsetting)
13:     : QDialog(parent), ui(new Ui::HardwareSettings) {
14:     ui->setupUi(this);
15:     if (soilsetting != 0) {
16:         soilSetting = soilsetting;
17:     } else {
18:         soilSetting = new SoilAnalyzer::SoilSettings;
19:     }
20:
21:     ui->cb_enableRainbow->setChecked(soilSetting->enableRainbow);
22:     ui->cb_encoderInv->setChecked(soilSetting->encInv);
23:     ui->sb_HDRframes->setValue(soilSetting->HDRframes);
24:     ui->sb_lightLevel->setValue(soilSetting->lightLevel);
25:
26:     // Get system info
27:     struct utsname unameData;
28:     uname(&unameData);
29:
30:     ui->label_machinename->setText(tr(unameData.machine));
31:     ui->label_nodename->setText(tr(unameData.nodename));
32:     ui->label_releasename->setText(tr(unameData.release));
33:     ui->label_systemname->setText(tr(unameData.sysname));
34:     ui->label_versionname->setText(tr(unameData.version));
35:     std::string arch = static_cast<std::string>(unameData.machine);
36:     if (arch.find("armv7l") == string::npos) {
37:         ui->cb_enableRainbow->setDisabled(true);
38:         ui->cb_encoderInv->setDisabled(true);
39:         ui->sb_lightLevel->setDisabled(true);
40:         ui->label_ll->setDisabled(true);
41:     }
42: }
43:
44: HardwareSettings::~HardwareSettings() { delete ui; }
45:
46: void HardwareSettings::on_sb_HDRframes_editingFinished() {
47:     soilSetting->HDRframes = ui->sb_HDRframes->value();
48: }
49:
50: void HardwareSettings::on_sb_lightLevel_editingFinished() {
51:     soilSetting->lightLevel = static_cast<float>(ui->sb_lightLevel->value());
52: }
53:
54: void HardwareSettings::on_cb_encoderInv_clicked(bool checked) {
55:     soilSetting->encInv = checked;
56: }
57:
58: void HardwareSettings::on_cb_enableRainbow_clicked(bool checked) {
59:     soilSetting->enableRainbow = checked;
60: }
```

```
1: /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #include "vsagui.h"
9: #include <QApplication>
10:
11: int main(int argc, char *argv[]) {
12:     QApplication a(argc, argv);
13:     VSAGUI w;
14:     w.show();
15:
16:     return a.exec();
17: }
```

```
1:  /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #ifndef VSAGUI_H
9: #define VSAGUI_H
10:
11: // #define DEBUG
12:
13: #include <string>
14: #include <vector>
15: #include <sys/utsname.h>
16: #include <stdlib.h>
17:
18: #include <QMainWindow>
19: #include <QtGui>
20: #include <QWidget>
21: #include <QErrorMessage>
22: #include <QGraphicsScene>
23: #include <QGraphicsPixmapItem>
24: #include <QPixmap>
25: #include <QLabel>
26: #include <QProgressBar>
27: #include <QFileDialog>
28: #include <qcustomplot.h>
29:
30: #include <opencv2/core.hpp>
31: #include <opencv2/highgui.hpp>
32:
33: #include <boost/filesystem.hpp>
34: #include <boost/signals2.hpp>
35: #include <boost/bind.hpp>
36: #include <boost/serialization/serialization.hpp>
37:
38: #include "opencvqt.h"
39: #include "visionsettings.h"
40: #include "hardwaresettings.h"
41:
42: #include "Hardware.h"
43: #include "SoilMath.h"
44: #include "Vision.h"
45: #include "VisionDebug.h"
46: #include "VisionSoil.h"
47:
48: class QErrorMessage;
49:
50: namespace Ui {
51: class VSAGUI;
52: }
53:
54: class VSAGUI : public QMainWindow {
55:     Q_OBJECT
56:
57: public:
58:     explicit VSAGUI(QWidget *parent = 0);
59:     ~VSAGUI();
60:
61: private slots:
62:
63:     void on_SnapshotButton_clicked();
64:
65:     void on_actionSave_triggered();
66:
67:     void on_actionLoad_triggered();
68:
69:     void on_AnalyzeButton_clicked();
70:
71:     void on_actionNew_triggered();
72:
73:     void on_actionImport_triggered();
74:
75:     void on_actionExport_triggered();
76:
77:     void on_actionSegmentation_Settings_triggered();
78:
79:     void on_actionSave_Settings_triggered();
80:
81:     void on_actionLoad_Settings_triggered();
82:
83:     void on_actionRestore_Default_triggered();
```

```
84:
85: void on_SegmentButton_clicked();
86:
87: void on_verticalSlider_sliderReleased();
88:
89: void on_OffsetSlider_valueChanged(int value);
90:
91: void on_OffsetSlider_sliderReleased();
92:
93: void on_actionHardware_Settings_triggered();
94:
95: void on_actionCheese_2_triggered();
96:
97: void on_actionImport_RGB_Snapshot_triggered();
98:
99: void on_actionExport_RGB_Snapshot_triggered();
100:
101: private:
102:     Ui::VSAGUI *ui;
103:     QErrorMessage *errorMessageDialog;
104:     VisionSettings *settingWindow;
105:     HardwareSettings *hsettingWindow;
106:
107:     SoilAnalyzer::SoilSettings *sSettings;
108:
109:     boost::signals2::connection finished_sig;
110:     boost::signals2::connection progress_sig;
111:     boost::signals2::connection visionprogress_seg;
112:
113:     void on_vision_update(float prog, string statusText);
114:
115:     SoilAnalyzer::Sample *SoilSample = nullptr;
116:     SoilMath::NN *NeuralNet = nullptr;
117:
118:     cv::Mat *OrigImg;
119:     QProgressBar *progressBar = nullptr;
120:     QLabel *statusLabel = nullptr;
121:
122:     bool runFromBBB = false;
123:     bool Segmented = false;
124:
125:     void LoadSample();
126:     std::vector<std::string> webCams;
127:     void SetMatToMainView(cv::Mat &img);
128:     void CreateNewSoilSample();
129: };
130:
131: #endif // VSAGUI_H
```

```
1:  /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8:  #ifndef VISIONSETTINGS_H
9:  #define VISIONSETTINGS_H
10:
11:  #include <QDialog>
12:
13:  #include "VisionSoil.h"
14:
15:  namespace Ui {
16:  class VisionSettings;
17:  }
18:
19:  class VisionSettings : public QDialog {
20:  Q_OBJECT
21:
22:  public:
23:      explicit VisionSettings(QWidget *parent = 0,
24:                              SoilAnalyzer::SoilSettings *soilsetting = 0);
25:      ~VisionSettings();
26:
27:      SoilAnalyzer::SoilSettings *soilSetting = nullptr;
28:
29:  private slots:
30:
31:      void on_cb_use_adaptContrast_stateChanged(int arg1);
32:
33:      void on_sb_adaptContrKernel_editingFinished();
34:
35:      void on_sb_adaptContrastFactor_editingFinished();
36:
37:      void on_cb_useBlur_stateChanged(int arg1);
38:
39:      void on_sb_blurMask_editingFinished();
40:
41:      void on_rb_useLight_toggled(bool checked);
42:      void on_cb_ignoreBorder_stateChanged(int arg1);
43:
44:      void on_cb_fillHoles_stateChanged(int arg1);
45:
46:      void on_sb_sigmaFactor_editingFinished();
47:
48:      void on_rb_useOpen_toggled(bool checked);
49:
50:      void on_rb_useClose_toggled(bool checked);
51:
52:      void on_rb_useErode_toggled(bool checked);
53:
54:      void on_rb_useDilate_toggled(bool checked);
55:
56:      void on_sb_morphMask_editingFinished();
57:
58:  private:
59:      Ui::VisionSettings *ui;
60:  };
61:
62: #endif // VISIONSETTINGS_H
```

```

1:  /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8:  #include "vsagui.h"
9:  #include "ui_vsagui.h"
10:
11: VSAGUI::VSAGUI(QWidget *parent) : QMainWindow(parent), ui(new Ui::VSAGUI) {
12:     // Determine if the program is run from an ARM (BBB) device
13:     struct utsname unameData;
14:     uname(&unameData);
15:     std::string arch = static_cast<std::string>(unameData.machine);
16:     if (arch.find("armv7l") != string::npos) {
17:         runFromBBB = true;
18:     }
19:
20:     // Startup the UI
21:     ui->setupUi(this);
22:     SoilSample = new SoilAnalyzer::Sample;
23:     SoilSample->connect_Progress(
24:         boost::bind(&VSAGUI::on_vision_update, this, _1, _2));
25:     NeuralNet = new SoilMath::NN;
26:     NeuralNet->LoadState("NeuralNet/Default.NN");
27:     sSettings = new SoilAnalyzer::SoilSettings;
28:     sSettings->LoadSettings("Settings/Default.ini");
29:     ui->OffsetSlider->setValue(sSettings->thresholdOffsetValue);
30:
31:     // Init the status bar
32:     progressBar = new QProgressBar(ui->statusBar);
33:     progressBar->setAlignment(Qt::AlignLeft);
34:     progressBar->setMaximumSize(640, 19);
35:     ui->statusBar->addWidget(progressBar);
36:     progressBar->setValue(0);
37:
38:     statusLabel = new QLabel(ui->statusBar);
39:     statusLabel->setAlignment(Qt::AlignRight);
40:     statusLabel->setMinimumSize(600, 19);
41:     statusLabel->setMaximumSize(600, 19);
42:     ui->statusBar->addWidget(statusLabel);
43:     statusLabel->setText(tr("First Grab"));
44:
45:     // Get HDR snapshot of the sample or normal shot when HDR grab falters or
46:     // test image if normal grab falters
47:     Hardware::Microscope microscope;
48:
49:     finished_sig = microscope.connect_Finished([&]() {
50:         SetMatToMainView(SoilSample->OriginalImage);
51:         this->statusLabel->setText(tr("New Image Grabbed"));
52:     });
53:     progress_sig = microscope.connect_Progress(
54:         [&](int &prog) { this->progressBar->setValue(prog); });
55:
56:     try {
57:         if (microscope.IsOpened()) {
58:             microscope.GetHDRFrame(SoilSample->OriginalImage, sSettings->HDRframes);
59:         }
60:     } catch (Hardware::Exception::MicroscopeNotFoundException &em) {
61:         try {
62:             errorMessageDialog->showMessage(
63:                 tr("Microscope not found switching to first default Cam!"));
64:             if (microscope.AvailableCams().size() > 0) {
65:                 microscope.openCam(0);
66:             }
67:         } catch (Hardware::Exception::MicroscopeNotFoundException &em2) {
68:             // display error dialog no cam found and show default test image
69:             errorMessageDialog->showMessage(
70:                 tr("Microscope not found switching to test image!"));
71:             SoilSample->OriginalImage = cv::imread("/Images/SoilSample1.png");
72:         }
73:     } catch (Hardware::Exception::CouldNotGrabImageException &ei) {
74:         // HDRFrame not working switching to normal grab
75:         try {
76:             errorMessageDialog->showMessage(
77:                 tr("HDR Grab failed switching to normal grab!"));
78:             microscope.GetFrame(SoilSample->OriginalImage);
79:         } catch (Hardware::Exception::CouldNotGrabImageException &ei2) {
80:             // show default test image and error dialog
81:             errorMessageDialog->showMessage(
82:                 tr("Normal Grab failed switching to test image!"));
83:             SoilSample->OriginalImage = cv::imread("/Images/SoilSample1.png");

```

```
84:     }
85: }
86: }
87:
88: void VSAGUI::SetMatToMainView(cv::Mat &img) {
89:     QImage *qOrigImg = new QImage(OpenCVQT::Mat2QImage(img));
90:     QPixmap *qOrigPix = new QPixmap(QPixmap::fromImage(*qOrigImg));
91:     ui->MainImg->clear();
92:     ui->MainImg->setPixmap(*qOrigPix);
93:     ui->MainImg->show();
94: }
95:
96: VSAGUI::~VSAGUI() { delete ui; }
97:
98: void VSAGUI::on_SnapshotButton_clicked() {
99:     Hardware::Microscope microscope;
100:     CreateNewSoilSample();
101:     this->StatusLabel->setText(tr("Grabbing new Image!"));
102:     finished_sig = microscope.connect_Finished([&]() {
103:         SetMatToMainView(SoilSample->OriginalImage);
104:         this->StatusLabel->setText(tr("New Image Grabbed"));
105:     });
106:     progress_sig = microscope.connect_Progress(
107:         [&](int &prog) { this->progressBar->setValue(prog); });
108:     microscope.GetHDRFrame(SoilSample->OriginalImage, sSettings->HDRframes);
109: }
110:
111: void VSAGUI::on_vision_update(float prog, string statusText) {
112:     int progress = prog * 100;
113:     this->progressBar->setValue(progress);
114:     this->StatusLabel->setText(tr(statusText.c_str()));
115: }
116:
117: void VSAGUI::on_actionSave_triggered() {
118:     QString fn = QFileDialog::getSaveFileName(
119:         this, tr("Save Soil Sample"), tr("/home/"),
120:         tr("Soil Samples (*.VSS);; Soil Particles (*.VPS);; All Files (*)"));
121:     if (!fn.isEmpty()) {
122:         if (!fn.contains(tr(".VSS"))) {
123:             fn.append(tr(".VSS"));
124:         }
125:         std::string filename = fn.toStdString();
126:         SoilSample->Save(filename);
127:     }
128: }
129:
130: void VSAGUI::on_actionLoad_triggered() {
131:     QString fn = QFileDialog::getOpenFileName(
132:         this, tr("Load Soil Sample"), tr("/home/"),
133:         tr("Soil Samples (*.VSS);; Soil Particles (*.VPS);; All Files (*)"));
134:     if (!fn.isEmpty() && fn.contains(tr("VSS"))) {
135:         delete SoilSample;
136:         SoilSample = new SoilAnalyzer::Sample;
137:         std::string filename = fn.toStdString();
138:         SoilSample->Load(filename);
139:         SoilSample->connect_Progress(
140:             boost::bind(&VSAGUI::on_vision_update, this, _1, _2));
141:         SetMatToMainView(SoilSample->OriginalImage);
142:     }
143: }
144:
145: /*!
146:  * \brief VSAGUI::on_AnalyzeButton_clicked Analyze the sample
147:  */
148: void VSAGUI::on_AnalyzeButton_clicked() {
149:     SoilSample->Analyse(*NeuralNet);
150:     SetMatToMainView(SoilSample->RGB);
151: }
152:
153: void VSAGUI::on_actionNew_triggered() {
154:     CreateNewSoilSample();
155:     on_SnapshotButton_clicked();
156: }
157:
158: void VSAGUI::on_actionImport_triggered() {
159:     QString fn = QFileDialog::getOpenFileName(
160:         this, tr("Import Neural Network"), tr("/home/"),
161:         tr("Neural Net (*.NN);;All Files (*)"));
162:     if (!fn.isEmpty()) {
163:         std::string filename = fn.toStdString();
164:         NeuralNet->LoadState(filename);
165:     }
166: }
```



```

167:
168: void VSAGUI::on_actionExport_triggered() {
169:     QString fn = QFileDialog::getSaveFileName(
170:         this, tr("Export Neural Network"), tr("/home/"),
171:         tr("Neural Net (*.NN);;All Files (*)"));
172:     if (!fn.isEmpty()) {
173:         if (!fn.contains(tr(".NN"))) {
174:             fn.append(tr(".NN"));
175:         }
176:         std::string filename = fn.toStdString();
177:         NeuralNet->SaveState(filename);
178:     }
179: }
180:
181: void VSAGUI::on_actionSegmentation_Settings_triggered() {
182:     settingWindow = new VisionSettings(0, sSettings);
183:     settingWindow->exec();
184: }
185:
186: void VSAGUI::on_actionSave_Settings_triggered() {
187:     QString fn =
188:         QFileDialog::getSaveFileName(this, tr("Save Settings"), tr("/home/"),
189:                                     tr("Settings (*.ini);;All Files (*)"));
190:     if (!fn.isEmpty()) {
191:         if (!fn.contains(tr(".ini"))) {
192:             fn.append(tr(".ini"));
193:         }
194:         std::string filename = fn.toStdString();
195:         sSettings->SaveSettings(filename);
196:     }
197: }
198:
199: void VSAGUI::on_actionLoad_Settings_triggered() {
200:     QString fn =
201:         QFileDialog::getOpenFileName(this, tr("Load Settings"), tr("/home/"),
202:                                     tr("Settings (*.ini);;All Files (*)"));
203:     if (!fn.isEmpty()) {
204:         std::string filename = fn.toStdString();
205:         sSettings->LoadSettings(filename);
206:     }
207: }
208:
209: void VSAGUI::on_actionRestore_Default_triggered() {
210:     std::string filename = "Settings/Default.ini";
211:     sSettings->LoadSettings(filename);
212: }
213:
214: void VSAGUI::on_SegmentButton_clicked() {
215:     SoilSample->PrepImg(sSettings);
216:     SetMatToMainView(SoilSample->RGB);
217: }
218:
219: void VSAGUI::on_verticalSlider_sliderReleased() {
220:     sSettings->thresholdOffsetValue = ui->OffsetSlider->value();
221:     if (SoilSample->imgPrepped) {
222:         SoilSample->PrepImg(sSettings);
223:     }
224: }
225: void VSAGUI::on_OffsetSlider_valueChanged(int value) {
226:     sSettings->thresholdOffsetValue = ui->OffsetSlider->value();
227: }
228:
229: void VSAGUI::on_OffsetSlider_sliderReleased() {}
230:
231: void VSAGUI::on_actionHardware_Settings_triggered() {
232:     hsettingWindow = new HardwareSettings(0, sSettings);
233:     hsettingWindow->exec();
234: }
235:
236: /*!
237:  * \brief VSAGUI::on_actionCheese_2_triggered Load the cheese program which can
238:  * stream the webcam
239:  */
240: void VSAGUI::on_actionCheese_2_triggered() {
241:     // Get the name of the individual cams
242:     std::vector<std::string> availCams = Hardware::Microscope::AvailableCams();
243:     uint i = 0;
244:     for_each(availCams.begin(), availCams.end(), [&](std::string &c) {
245:         // If the current iterator is the Microscope start cheese
246:         if (C.compare(MICROSCOPE_NAME) == 0) {
247:             std::string bashStr = "cheese --device=/dev/video";
248:             bashStr.append(std::to_string(i));
249:             std::system(bashStr.c_str());

```

```
250:         return;
251:     }
252:     i++;
253: });
254: }
255:
256: /*!
257: * \brief VSAGUI::on_actionImport_RGB_Snapshot_triggered Imports an RGB image to
258: * be used for
259: */
260: void VSAGUI::on_actionImport_RGB_Snapshot_triggered() {
261:     this->statusLabel->setText(tr("Importing new Image!"));
262:
263:     // Create the new SoilSample
264:     CreateNewSoilSample();
265:     // Show the filedialog and import the RGB image
266:     QString fn = QFileDialog::getOpenFileName(
267:         this, tr("Load Image"), tr("/home/"),
268:         tr("Image (*.jpg *.JPG *.png *.PNG *.gif *.GIF *.bmp *.BMP *.ppm *.PPM)"));
269:     if (!fn.isEmpty()) {
270:         std::string filename = fn.toStdString();
271:         SoilSample->OriginalImage = cv::imread(filename, 1);
272:         if (SoilSample->OriginalImage.channels() != 3) {
273:             errorMessageDialog->showMessage(tr("No RGB image"));
274:             on_actionImport_RGB_Snapshot_triggered();
275:         }
276:         SetMatToMainView(SoilSample->OriginalImage);
277:         this->statusLabel->setText(tr("New Image Imported"));
278:     }
279: }
280:
281: /*!
282: * \brief VSAGUI::CreateNewSoilSample Create the new Soil Sample
283: */
284: void VSAGUI::CreateNewSoilSample() {
285:     delete SoilSample;
286:     SoilSample = new SoilAnalyzer::Sample;
287:     SoilSample->connect_Progress(
288:         boost::bind(&VSAGUI::on_vision_update, this, _1, _2));
289: }
290:
291: /*!
292: * \brief VSAGUI::on_actionExport_RGB_Snapshot_triggered Export the RGB snapshot
293: */
294: void VSAGUI::on_actionExport_RGB_Snapshot_triggered()
295: {
296:     QString fn =
297:         QFileDialog::getSaveFileName(this, tr("Load Image"), tr("/home/"),
298:             tr("PPM Image (*.ppm *.PPM)"));
299:     if (!fn.isEmpty()) {
300:         if (!fn.contains(tr(".ppm"))) {
301:             fn.append(tr(".ppm"));
302:         }
303:         std::string filename = fn.toStdString();
304:         cv::imwrite(filename, SoilSample->OriginalImage);
305:     }
306: }
307: }
```

```
1:  /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8:  #include "visionsettings.h"
9:  #include "ui_visionsettings.h"
10:
11: VisionSettings::VisionSettings(QWidget *parent,
12:                               SoilAnalyzer::SoilSettings *soilsetting)
13:   : QDialog(parent), ui(new Ui::VisionSettings) {
14:   ui->setupUi(this);
15:   if (soilsetting != 0) {
16:       soilSetting = soilsetting;
17:   } else {
18:       soilSetting = new SoilAnalyzer::SoilSettings;
19:   }
20:
21:   // Read Segmentation Values from setting file
22:   ui->cb_fillHoles->setChecked(soilSetting->fillHoles);
23:   ui->cb_ignoreBorder->setChecked(soilSetting->ignorePartialBorderParticles);
24:   ui->cb_useBlur->setChecked(soilSetting->useBlur);
25:   if (!soilSetting->useBlur) {
26:       ui->sb_blurMask->setEnabled(false);
27:   }
28:   ui->cb_use_adaptContrast->setChecked(soilSetting->useAdaptiveContrast);
29:   if (!soilSetting->useAdaptiveContrast) {
30:       ui->sb_adaptContrastFactor->setEnabled(false);
31:       ui->sb_adaptContrKernel->setEnabled(false);
32:   }
33:   switch (soilSetting->typeOfObjectsSegmented) {
34:   case Vision::Segment::Bright:
35:       ui->rb_useDark->setChecked(false);
36:       ui->rb_useLight->setChecked(true);
37:       break;
38:   case Vision::Segment::Dark:
39:       ui->rb_useDark->setChecked(true);
40:       ui->rb_useLight->setChecked(false);
41:       break;
42:   }
43:   switch (soilSetting->morphFilterType) {
44:   case Vision::MorphologicalFilter::CLOSE:
45:       ui->rb_useClose->setChecked(true);
46:       ui->rb_useDilate->setChecked(false);
47:       ui->rb_useErode->setChecked(false);
48:       ui->rb_useOpen->setChecked(false);
49:       break;
50:   case Vision::MorphologicalFilter::OPEN:
51:       ui->rb_useClose->setChecked(false);
52:       ui->rb_useDilate->setChecked(false);
53:       ui->rb_useErode->setChecked(false);
54:       ui->rb_useOpen->setChecked(true);
55:       break;
56:   case Vision::MorphologicalFilter::ERODE:
57:       ui->rb_useClose->setChecked(false);
58:       ui->rb_useDilate->setChecked(false);
59:       ui->rb_useErode->setChecked(true);
60:       ui->rb_useOpen->setChecked(false);
61:       break;
62:   case Vision::MorphologicalFilter::DILATE:
63:       ui->rb_useClose->setChecked(false);
64:       ui->rb_useDilate->setChecked(true);
65:       ui->rb_useErode->setChecked(false);
66:       ui->rb_useOpen->setChecked(false);
67:       break;
68:   }
69:
70:   ui->sb_adaptContrastFactor->setValue(soilSetting->adaptContrastKernelFactor);
71:   ui->sb_adaptContrKernel->setValue(soilSetting->adaptContrastKernelSize);
72:   ui->sb_blurMask->setValue(soilSetting->blurKernelSize);
73:   ui->sb_morphMask->setValue(soilSetting->filterMaskSize);
74:   ui->sb_sigmaFactor->setValue(soilSetting->sigmaFactor);
75: }
76:
77: VisionSettings::~VisionSettings() { delete ui; }
78:
79: void VisionSettings::on_cb_use_adaptContrast_stateChanged(int arg1) {
80:     soilSetting->useAdaptiveContrast = static_cast<bool>(arg1);
81: }
82:
83: void VisionSettings::on_sb_adaptContrKernel_editingFinished() {
```

```
84:     int val = ui->sb_adaptContrKernel->value();
85:     if (val % 2 == 0) {
86:         ui->sb_adaptContrKernel->setValue(val + 1);
87:     }
88:     soilSetting->adaptContrastKernelSize = ui->sb_adaptContrKernel->value();
89: }
90:
91: void VisionSettings::on_sb_adaptContrastFactor_editingFinished() {
92:     soilSetting->adaptContrastKernelFactor =
93:         static_cast<float>(ui->sb_adaptContrastFactor->value());
94: }
95:
96: void VisionSettings::on_cb_useBlur_stateChanged(int arg1) {
97:     soilSetting->useBlur = static_cast<bool>(arg1);
98: }
99:
100: void VisionSettings::on_sb_blurMask_editingFinished() {
101:     int val = ui->sb_blurMask->value();
102:     if (val % 2 == 0) {
103:         ui->sb_blurMask->setValue(val + 1);
104:     }
105:     soilSetting->blurKernelSize = ui->sb_blurMask->value();
106: }
107:
108: void VisionSettings::on_rb_useLight_toggled(bool checked) {
109:     if (checked) {
110:         soilSetting->typeOfObjectsSegmented = Vision::Segment::Bright;
111:     } else {
112:         soilSetting->typeOfObjectsSegmented = Vision::Segment::Dark;
113:     }
114: }
115:
116: void VisionSettings::on_cb_ignoreBorder_stateChanged(int arg1) {
117:     soilSetting->ignorePartialBorderParticles = static_cast<bool>(arg1);
118: }
119:
120: void VisionSettings::on_cb_fillHoles_stateChanged(int arg1) {
121:     soilSetting->fillHoles = static_cast<bool>(arg1);
122: }
123:
124: void VisionSettings::on_sb_sigmaFactor_editingFinished() {
125:     soilSetting->sigmaFactor = static_cast<float>(ui->sb_blurMask->value());
126: }
127:
128: void VisionSettings::on_rb_useOpen_toggled(bool checked) {
129:     if (checked) {
130:         soilSetting->morphFilterType = Vision::MorphologicalFilter::OPEN;
131:     }
132: }
133:
134: void VisionSettings::on_rb_useClose_toggled(bool checked) {
135:     if (checked) {
136:         soilSetting->morphFilterType = Vision::MorphologicalFilter::CLOSE;
137:     }
138: }
139:
140: void VisionSettings::on_rb_useErode_toggled(bool checked) {
141:     if (checked) {
142:         soilSetting->morphFilterType = Vision::MorphologicalFilter::ERODE;
143:     }
144: }
145:
146: void VisionSettings::on_rb_useDilate_toggled(bool checked) {
147:     if (checked) {
148:         soilSetting->morphFilterType = Vision::MorphologicalFilter::DILATE;
149:     }
150: }
151:
152: void VisionSettings::on_sb_morphMask_editingFinished() {
153:     int val = ui->sb_morphMask->value();
154:     if (val % 2 == 0) {
155:         ui->sb_morphMask->setValue(val + 1);
156:     }
157:     soilSetting->filterMaskSize = ui->sb_morphMask->value();
158: }
```

```
1:  /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8:  #ifndef OPENCVQT_H
9:  #define OPENCVQT_H
10: #include <QImage>
11: #include <opencv2/core.hpp>
12: #include <vector>
13:
14: class OpenCVQT {
15: public:
16:     OpenCVQT();
17:     static QImage Mat2QImage(const cv::Mat &src) {
18:         QImage dest;
19:         if (src.channels() == 1) {
20:             cv::Mat destRGB;
21:             std::vector<cv::Mat> grayRGB(3, src);
22:             cv::merge(grayRGB, destRGB);
23:             dest = QImage((uchar *)destRGB.data, destRGB.cols, destRGB.rows,
24:                          destRGB.step, QImage::Format_RGB888);
25:         } else {
26:             dest = QImage((uchar *)src.data, src.cols, src.rows, src.step,
27:                          QImage::Format_RGB888);
28:             dest = dest.rgbSwapped();
29:         }
30:         return dest;
31:     }
32: };
33:
34: #endif // OPENCVQT_H
```