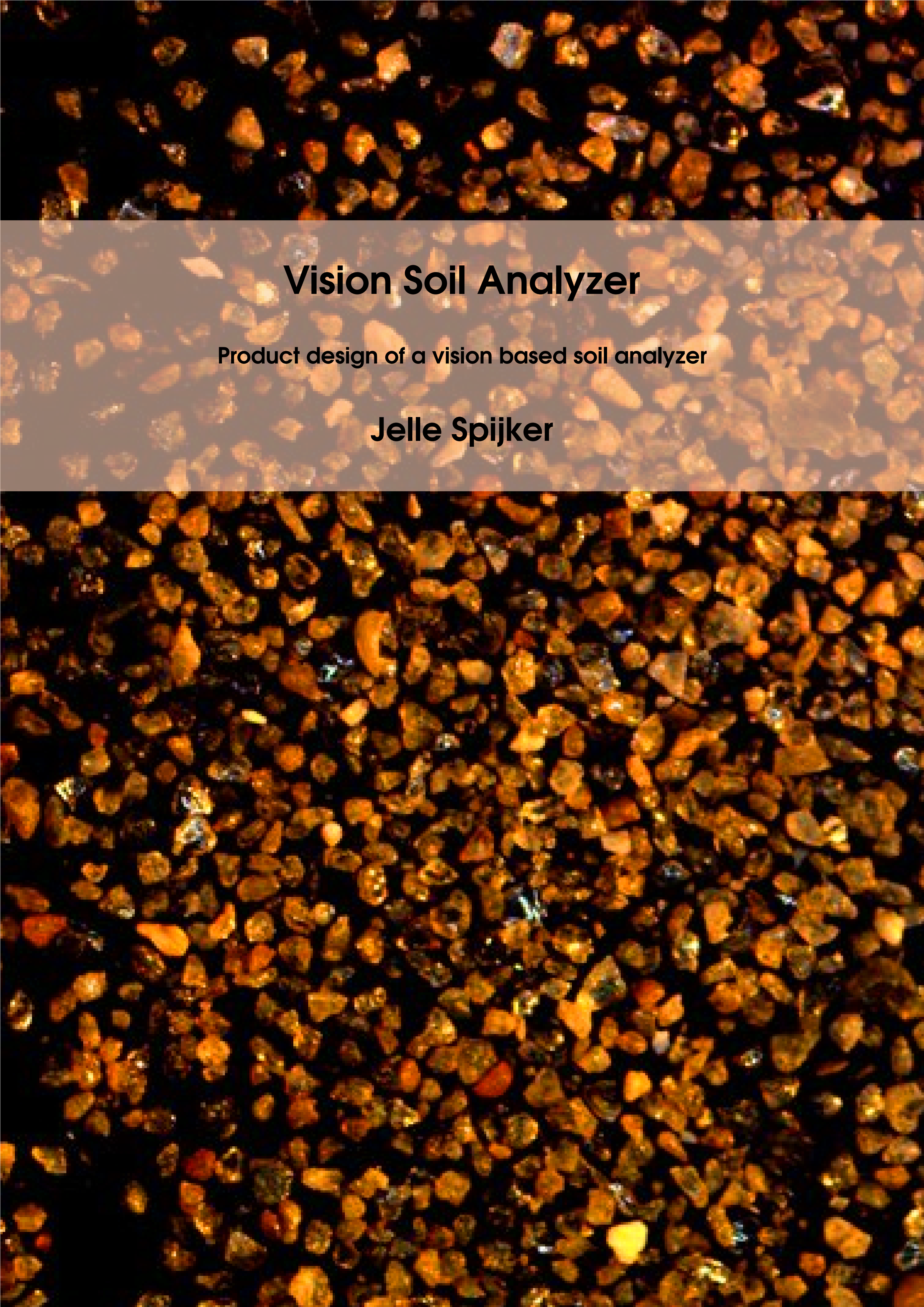


Vision Soil Analyzer

Product design of a vision based soil analyzer

Jelle Spijker

PUBLISHED BY ROYAL IHC

WWW.IHCMERWEDE.COM
WWW.MTIHOLLAND.COM
WWW.HAN.NL

First printing, September 2015

Contents

1. Introduction

This project finds its roots in the minor Embedded Vision Design @ HAN, hereafter named EVD. During this minor an embedded device was developed which analyses soil samples using a microscope. This Vision Soil Analyzer hereafter refereed to as VSA, analyzes samples using the optical properties. It gives an user information on color, texture and structure.

This is developed in collaboration with Royal IHC and MTI Holland. Royal IHC is one of Holland major shipyard companies and specializes in dredging and offshore. MTI Holland BV is royal IHC dredging knowledge center. They're worldwide leading centre of expertise in the area of translating knowledge of dredging, mining and deep-sea mining processes into the specification, design and application of equipment.

Both companies have an interests in knowing the properties of soil, be it to advise their customers or to further facilitate their own research and services. Current methods, like the Particle Size Analysis using a sieve and hydrometer are time consuming and non portable. To facilitate quick, accurate and on location soil research an embedded device has been developed. This VSA analyzes soil samples using a microscope and gives the user acceptable and quick results on the soil visual properties.

Quick and reliable results are a welcome addition into any laboratory, this combined with a device that is light and portable gives it's users an added benefit of shortened logistical operations for their soil samples. This results in some serious time benefits.

During the first period of the minor a basic prototype has been developed. This prototype ran in Matlab on a X64 desktop computer and was a first test case for the algorithms and idea's. In the second period this prototype is developed on an ARMv7 embedded Linux device and is compiled in C++. The goal of the software is to analyze soil samples and presenting the user with information regarding it's color, texture and structure.

Information regarding the color of a sample is presented to the user in the CIE Lab and Redness Index color-models. These color models show correlation between different soil properties, such as iron content and fertility. Conversion between different color-models are CPU intensive, because each pixel will be transformed using multiple algorithms. It's therefore paramount that calculations are done with an minimum of machine instructions and with acceptable errors.

Texture information is presented to a user via a Particle Size Distribution, hereafter named PSD. This is a cumulative function representing the ratio of different particle sizes in the soil sample. Due to the nature of a two dimensional digital image numerous problems arise. These are overlap of smaller particles by bigger particles, this gives a distortion in the PSD results, because the smaller particle is registered as part of the bigger particle. And another problem is the fact that soil particles are three dimensional. but the image is two dimensional.

Information about the structure of the soil is extrapolated from the individual particles shapes. These shapes are describes in the frequency domain, using a Fast Fourier Transform and fed into a Neural Network which classifies these shapes into standard soil categories. These are time consuming operations and therefore should be done with a minimum of machine instructions and efficient programming.

This wiki / product documentation gives the developer(s) and customers, namely MTI and IHC a tool to further the development of the VSA in to a full fledge market ready product. The development environment and the used protocols are described in order to guard the quality of the work. The product itself is designed by determining a global IPO Input-Process-Output diagram. This leads to the functional specifications. To illustrate the working of the device further the User Interface will be designed which will be supplemented with a short manual. All the above design tools will come together in a detailed IPO. Correct working of the device is guaranteed with various testing protocols. The current working principles follows a set global workflow. The vision related algorithms are describe in order to determine the most efficient working order. This results in the complete image processing steps

The following project setup is proposed for the release candidate. Future release will follow the roadmap

Design

2. Functional Design

2.1 Global Input-Proces-Output

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

2.2 Specifications

2.2.1 Functional requirements

Name Description
Word Definition
Comment Elaboration

2.2.2 Technical requirements

Name Description
Word Definition
Comment Elaboration

3. User Interface

4. Manuals

5. Technical Design

5.1 Hierarchical structure

This is an example of theorems.

5.2 Architecture

This is a theorem consisting of several equations.

5.3 Detailed Input-Process-Output schematics

This is a theorem consisting of just one line.

5.3.1 Led driver

5.3.2 Global position unit

5.3.3 Controller

6. Vision design

Realization

7. Technical Realization

8. Vision realization

This chapter describes the used vision processing techniques. The current prototype and work flow is developed to allow for different routines. The user has multiple options and strategies available to achieve optimum results. Each of these are explained in the sequential subsection below. It begins with the acquisition of image(s), which are then enhanced to allow for optimal segmentation of pixels related to sand particles. These pixels are used to determine the features of each particle, which serve as input for the classification algorithms.

8.1 Image acquisition

A thorough review of the current literature [1] identified three properties that can be used in vision based analyzing. These properties are structure (shape), color and texture (size). When looking closely at sand sample, you notice a multitude of shapes, colors and sizes, each particle is unique and differs from its neighbor. This diversity brings it own challenges. The shape of a particle determines how it will rest on the sample plate. The color and the translucency of the particle, determines how easily it can be segmented or identified from the background. Whilst the size determines the needed focus depth of the microscope.

> R In samples, where the particles show a huge spread in size, compared to the mean size, there will be a noticeable difference in focus, between big and small particles.

Acquisition strategies

The first prototype is developed in such a way that multiple acquisition strategies can be implemented. Each of these tackle different challenges. The quality of the acquired image is the biggest factor in the successful extraction of a particle, but in order to make any valid claim about the sample, a certain amount of particles have to examined. To determine the minimum sample size , the following equation can be derived:

Let the reliability be 95% $\therefore z = 1.96$, the probability be $P = 50\%$ and the accuracy be $\alpha = 5\%$; consider the function:

$$z\sqrt{\frac{p \times (1-P)}{n}} \leq \alpha \rightarrow n \geq \frac{-p \times (P-1) \times z^2}{\alpha^2} \quad (8.1)$$

This brings the minimum amount of particles to 384. With the predefined range of particle sizes ($0.2[mm] \leq P_size \leq 2[mm]$ where P defines a particle) and the limited work area under the microscope, multiple shots have to be taken. Where the sample is rearranged. Between fifteen and twenty shots are usually enough.

R The process of rearranging the particles, will be automated in the future. Student of the minor Offshore & Construction taught at the University of Applied Sciences Rotterdam will work on this challenge. This is done on the RDM Campus. This minor starts in September 2015. Their product will serve as input for the second prototype. Their assignment is described in appendix A and will be executed under the auspice of MTI Holland and the author.

Acquisition

Each sample is placed in a light condition room, and laid out on a semitransparent white acrylate plate. The sample can be illuminated with a bright field light source, where the light is aimed directly at an object or the particle can be lit with back lighting. See the course notes [3] for a more in-depth description. The choice for back lighting can be made because translucent particle are harder to segment in a bright field light. The trade off is extra processing time.

After the sample is placed in the light condition room, the microscope takes a image with bright field illumination and, if the option is selected, another one with back lighting. Hereafter the sample is rearranged, this is a manual procedure. Once the sample is rearranged a new set of shots is taken. Each image that is acquired from the microscope is defined by a matrix were the values are triples for the RGB (red, green and blue) values and these are defined by an unsigned byte.

Each image is stored in a vector using a custom container. This container consists of a bright field image, back light image and a SI-conversion factor . Each time the height is changed, the microscope has to be calibrated so that the relation between pixel and [mm] can be determined. This is done by taking a shot of a disc with known dimensions. A single euro cent can serve for this purpose.

R The image is stored in the OpenCV matrix (cv::Mat) container. This container is designed to handle image processing data and routines. It makes use of memory management and smart pointers to handle the data effectively.

8.2 Image enhancement

Image enhancement prepares the RGB image for conversion to a binary image. It eliminates noise and brings out wanted features, by using filters.

Intensity image

The first step in this process step is the conversion from the RGB color space to an scalar valued image which represent the luminosity, also known as a intensity image. This luminosity is calculated using a weighted average and is done for bright field and back lit images.

Let $\mathbf{I}$ and $\mathbf{R},\mathbf{G},\mathbf{B}$ be a matrices with dimensions $n \times m$ derived from the color matrix **RGB** with dimensions $n \times m \times 3$; The weighted average can be calculated with the following equation:

$$\mathbf{I} = 0.2126 \times \mathbf{R} + 0.7152 \times \mathbf{G} + 0.0722 \times \mathbf{B} \tag{8.2}$$

Adaptive contrast stretch

After the conversion from RGB to an intensity image, the user has the choice to apply an adaptive contrast stretch to the bright field images. This process is used to enhance the contrast of the intensity image. For every pixel and its surrounding area the mean and standard deviation are calculated. If the value of the pixel is above or below the mean than the following rule is used to determine the new value: $\mathbf{I}_{n,m} = \mathbf{I}_{n,m} \times \alpha \pm \sigma$, where α is a scaling factor and σ is the standard deviation of the old pixel value with it's neighboring kernel pixels.

Blur

As a second enhancement the user can apply a blurring operation to the bright field images, in essence the opposite of the contrast stretch. The blur operation also determines the mean for every pixels within a given area: the kernel. The mean value of the kernel is assigned to the pixel.

Cropping

The above operations described in the paragraph 8.2 and 8.2, leave the border pixels unaffected in their calculations. This offset is determined by half of the biggest kernel size. These pixels are discarded for the next step. The enhanced intensity matrix is used for particle segmentation, see section 8.3. Whilst the intensity matrix of the bright field image is used for the conversion to the CIE La*b* colorspace, as explained in section 8.3.1.

8.3 Feature extraction

In order to tell something about the individual particles, they first have to be identified and separated from the background. This is done with the enhanced intensity matrix. Which is taken from the bright field matrix or if available the back lit intensity matrix.

Segmentation

The images are segmented by calculating a threshold value. This value is determined by using the Otsu threshold. Xu et al. [2] describe that the Otsu threshold is equal to the average of the mean levels of two classes partitioned by this threshold. This threshold can be iteratively determined.

Let $\vec{h}$ be a vector of dimension 256 which represent a count of values in the enhanced intensity matrix $\mathbf{I}$ with dimensions $m \times n$

$$\frac{1}{t}\sum_{i=1}^{t}\vec{h}_i = t - \frac{1}{256-t}\sum_{i=t}^{256}\vec{h}_i \tag{8.3}$$

8.3.1 CIE La*b* extraction

8.3.2 Fast Fourier Descriptors

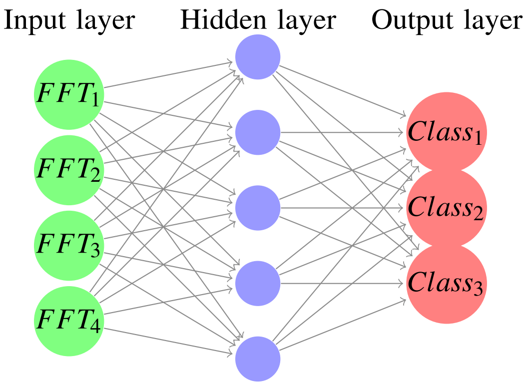
8.3.3 SI equivalent diameter

8.4 Classification

8.4.1 Roundness using Hu-moments

8.4.2 Angularity using a Neural Network

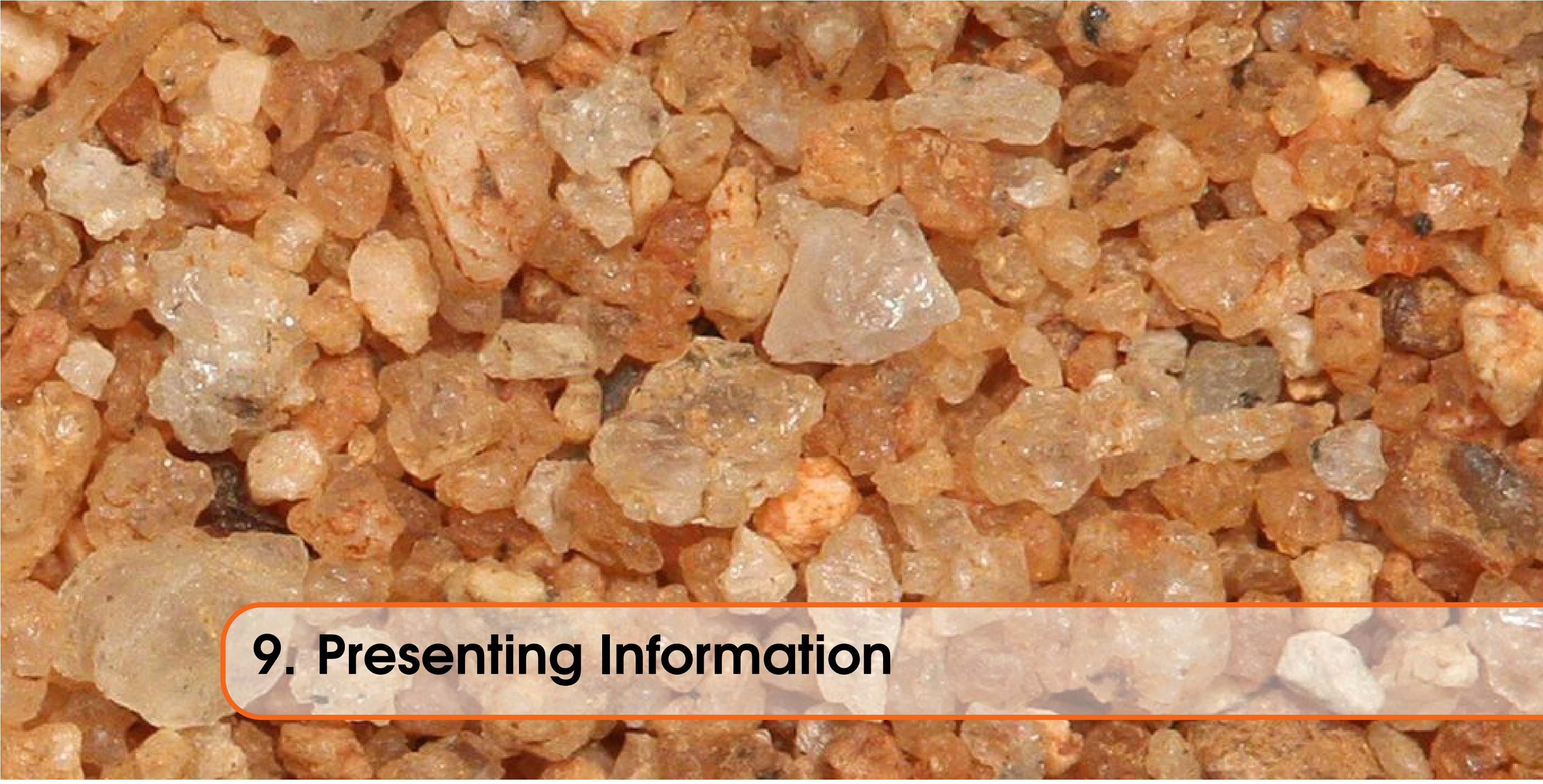
Angularity of particle can be described as



This is an example of examples.

III

Verification



9. Presenting Information

9.1 Table

Treatments	**Response 1**	**Response 2**
Treatment 1	0.0003262	0.562
Treatment 2	0.0015681	0.910
Treatment 3	0.0009271	0.296

Table 9.1: Table caption

9.2 Figure

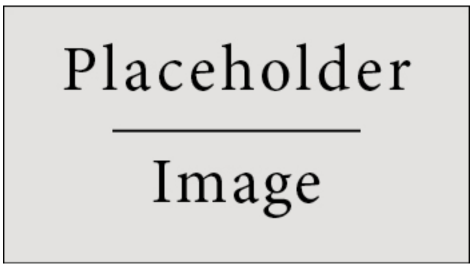
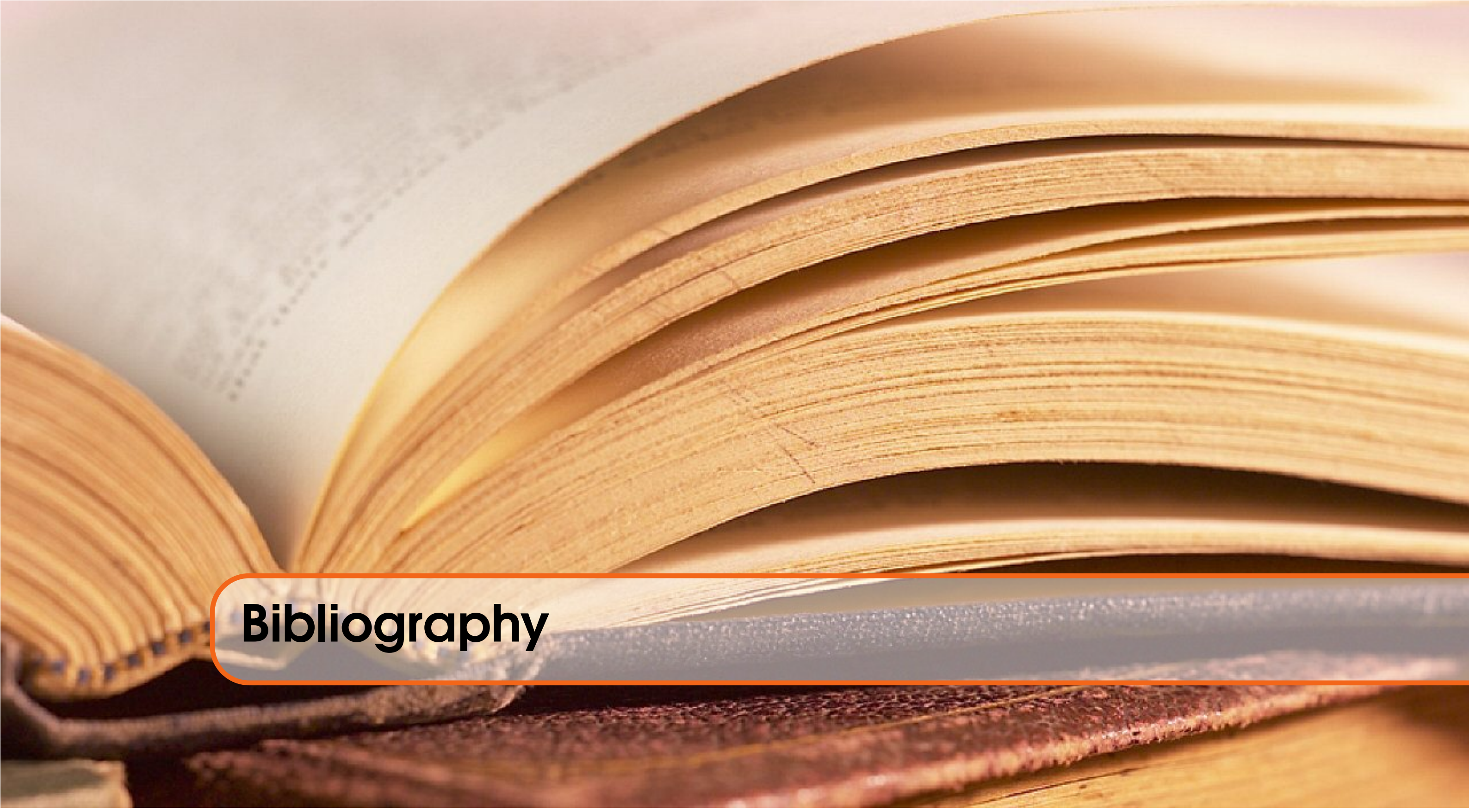


Figure 9.1: Figure caption

IV

Addenda



Bibliography

Books

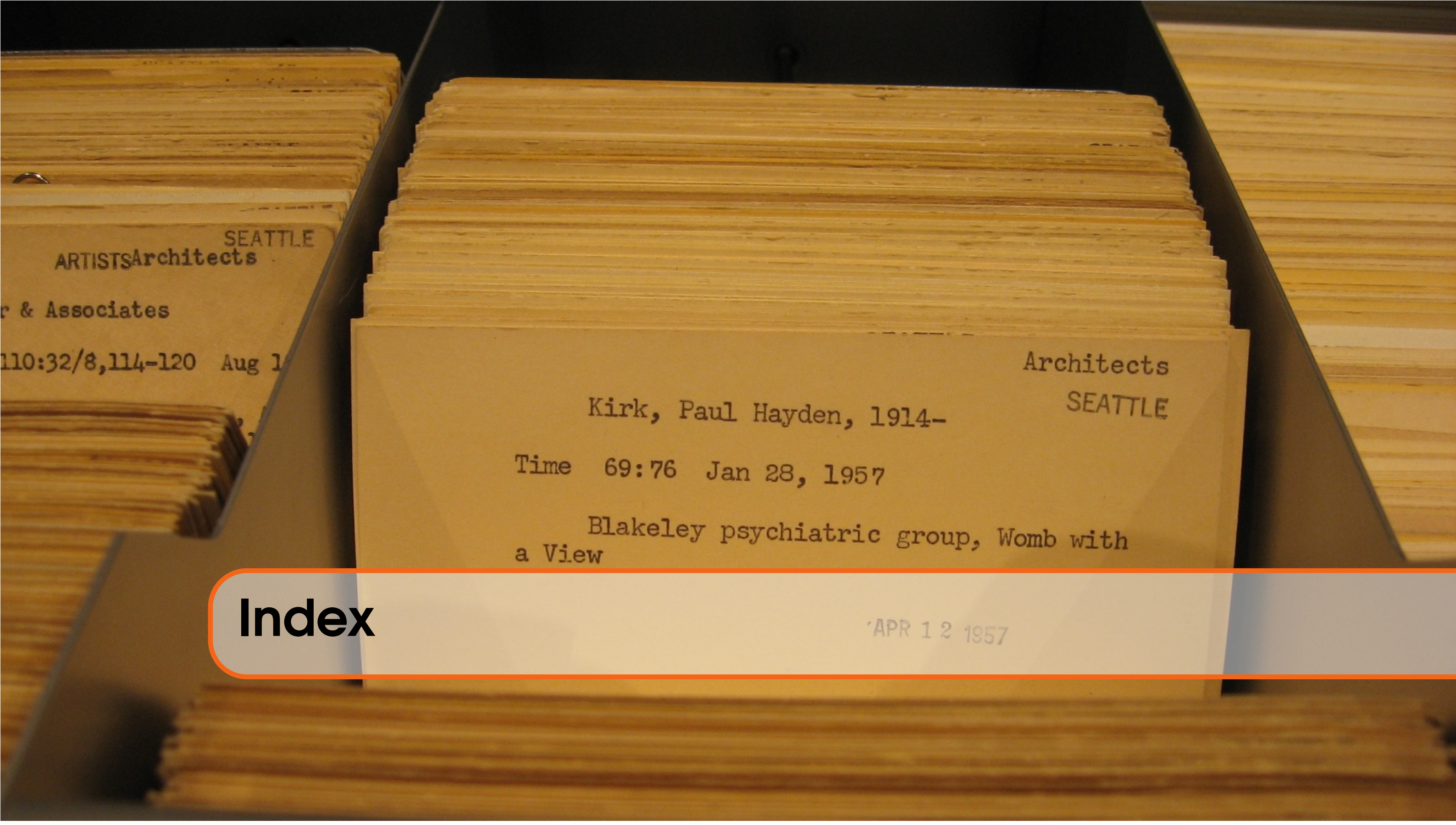
[3] ir. P.A.C. Ypma. *Course Notes EVD2*. University of applied sciences, Sept. 2, 2014. 71 pages (cited on page 26).

Reports

[1] Jelle Spijker. *Optische kenmerken van grond gebruikt bij computer vision*. Literatuurstudie. HAN University of applied sciences, 2014 (cited on page 25).

Articles

[2] Xiangyang Xu et al. “Characteristic analysis of Otsu threshold and its applications”. In: *Pattern Recognition Letters* 32.7 (2011), pages 956 –961. ISSN: 0167-8655. DOI: http://dx.doi.org/10.1016/j.patrec.2011.01.021. URL: http://www.sciencedirect.com/science/article/pii/S0167865511000365 (cited on page 27).


Architects
SEATTLE
Kirk, Paul Hayden, 1914-
Time 69:76 Jan 28, 1957
Blakeley psychiatric group, Womb with
a View
APR 1 2 1957

Index

A. RDM campus: Student project

Subject: RDM Student project
Author: Jelle Spijker

Introduction

This project finds its roots in the minor Embedded Vision Design (EVD) taught at the university of applied sciences HAN. During this minor a portable embedded device was developed which analyses soil samples using a microscope. This Vision Soil Analyser hereafter referred to as VSA, analyses soil samples using the optical properties. It's main function is: Presenting quantifiable information to a user on the properties of soil: such as colour, texture and structure.

The VSA takes a snapshot from a soil sample, which is placed under a microscope in an closed environment. This digital image is analysed using a multitude of computer vision algorithms. Statistical data is presented to the user in the form a Particle Size Distribution (PSD) and a histogram of the shape classification. The PSD is obtained by calculating the number of pixels for each individual particle, whilst shape classification is determined by describing the contour of each individual particle as mathematical function which undergoes a transformation to the frequency domain. This complex vector then serves as input for an Artificial Neural Network (ANN) where the output classifies each particle in a certain category.

The prototype developed during the minor EVD will serve as a basis for a graduation project of that same student, which initialized the project. This is done for his main course mechanical engineering at the HAN. This graduation project is done under the auspices of MTI Holland. The goal during this second stage is to develop a field ready prototype. In conjunction with the necessary documentation (Technical Dossier). Due to the scale of the project, several key problems are identified and separated from the main project. These problems can be tackled by separated student groups.

Problem description

Due to the transformation from 3D particles to a discrete 2D image certain data is lost. This degradation of data introduces errors in the statistical data. One of the forms of degradations is the overlap of bigger particle onto smaller particles. These particles are identified as an particle with at least the size and the contour of the biggest particles. Thus giving false negatives for the smaller particles and often false positives for the bigger particle.

A solution that will be explored during this stage is the execution of multiple analysis of the same discrete particle population. This will result in an accurate statistical representation of the soil sample placed under the microscope.

The project that the RDM students can tackle can be described as follow:

> Design and build a prototype with which the placement of particles, relative to each other and ranging in sizes from 0.02 - 2 [mm] are randomly changed in a time span of 1 [sec], which is tightly integrated with the main prototype.

The prototype is to be CE compliant and should be build according to technical specifications. It should be described in a Technical Dossier, containing all necessary documents such as: technical drawings (according to mono system), bill of materials, calculation, analysis and design reports.

B. Literature review

Literatuurstudie

Optische kenmerken van grond gebruikt bij computer vision

Opdrachtgever: Royal IHC

Opleverdatum: **3 november 2014**

Jelle Spijker

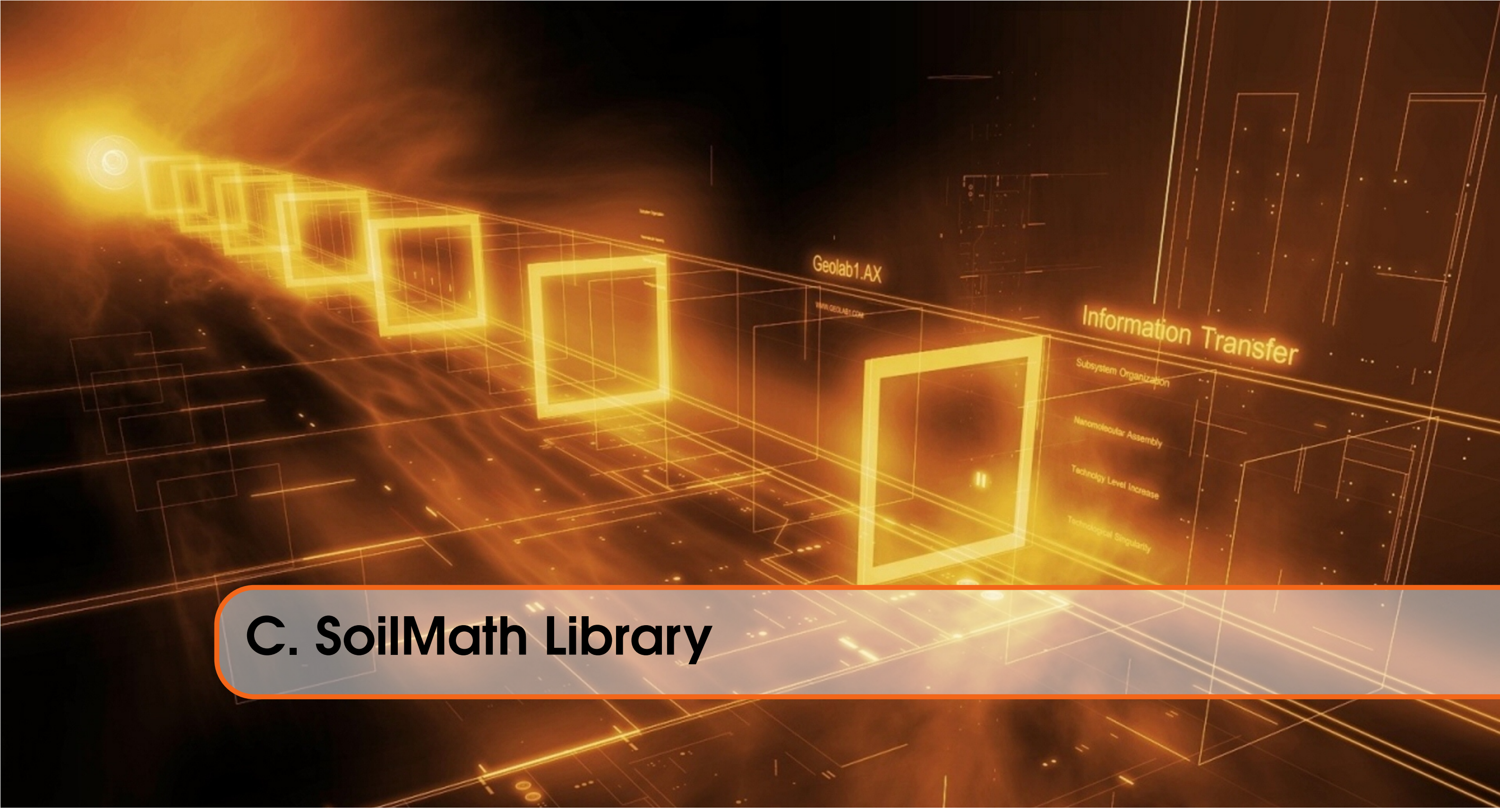
Datum 2 november 2014
Revisie 20141102

Contact gegevens:

Jelle Spijker (495653) – 06-43272644 – Spijker.Jelle@gmail.com

Disclaimer HAN:

Door ondertekening van dit voorblad, bevestigen wij dat het – door ons ingeleverd(e) werkstuk/rapport/scriptie (verder te noemen “product”) – zelfstandig en zonder enige externe hulp door ons is vervaardigd en dat wij op de hoogte zijn van de regels omtrent onregelmatigheden/fraude zoals die vermeld staan in het opleidingsstatuut.
In delen van het product, die letterlijk of bijna letterlijk zijn geciteerd uit externe bronnen (zoals internet, boeken, vakbladen enz.) is dit door ons via een verwijzing conform APA-norm (b.v. voetnoot) expliciet kenbaar gemaakt in het geciteerde tekstdeel (cursief gedrukt).



C. SoilMath Library

C.0.1 Genetic Algorithm Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

//! Genetic Algorithmes used for optimization problems
/*!
  * Use this class for optimization problems. It's currently optimized for
 * Neural Network optimzation
  */
#pragma once

#include <bitset>
#include <random>
#include <string>
#include <algorithm>
#include <chrono>
#include <math.h>
#include <list>

//#include "NN.h"
#include "SoilMathTypes.h"
#include "MathException.h"

#include <QtCore/QObject>
#include <QDebug>
```

```
#include <QThread>
#include <QtConcurrent>

#include <boost/bind.hpp>

namespace SoilMath {

class GA : public QObject {
  Q_OBJECT

public:
  float MutationRate = 0.075f; /**< mutation rate*/
  uint32_t Elitisme = 4;       /**< total number of the elite bastard*/
  float EndError = 0.001f;     /**< acceptable error between last itteration*/
  bool Revolution = true;

  /*!
 * \brief GA Standard constructor
 */
  GA();

  /*!
   * \brief GA Construction with a Neural Network initializers
   * \param nnfunction the Neural Network prediction function which results will
   * be optimized
   * \param inputneurons the number of input neurons in the Neural Network don't
   * count the bias
   * \param hiddenneurons the number of hidden neurons in the Neural Network
   * don't count the bias
   * \param outputneurons the number of output neurons in the Neural Network
   */
  GA(NNfunctionType nnfunction, uint32_t inputneurons, uint32_t hiddenneurons,
     uint32_t outputneurons);

  /*!
   * \brief GA standard de constructor
   */
  ~GA();

  /*!
   * \brief Evolve Darwin would be proud!!! This function creates a population
   * and itterates
   * through the generation till the maximum number off itterations has been
   * reached of the
   * error is acceptable
```

```
   * \param inputValues complex vector with a reference to the inputvalues
   * \param weights reference to the vector of weights which will be optimized
   * \param rangeweights reference to the range of weights, currently it doesn't
   * support indivudal ranges
   * this is because of the crossing
   * \param goal target value towards the Neural Network prediction function
   * will be optimized
   * \param maxGenerations maximum number of itterations default value is 200
   * \param popSize maximum number of population, this should be an even number
   */
  void Evolve(const InputLearnVector_t &inputValues, Weight_t &weights,
              MinMaxWeight_t rangeweights, OutputLearnVector_t &goal,
              uint32_t maxGenerations = 200, uint32_t popSize = 30);
signals:
  void learnErrorUpdate(double newError);

private:
  NNfunctionType NNfuction; /**< The Neural Net work function*/
  uint32_t inputneurons;    /**< the total number of input neurons*/
  uint32_t hiddenneurons;   /**< the total number of hidden neurons*/
  uint32_t outputneurons;   /**< the total number of output neurons*/

  MinMaxWeight_t rangeweights;
  InputLearnVector_t inputValues;
  OutputLearnVector_t goal;

  float minOptim = 0;
  float maxOptim = 0;
  uint32_t oldElit = 0;
  float oldMutation = 0.;
  std::list<double> last10Gen;
  uint32_t currentGeneration = 0;
  bool revolutionOngoing = false;

  /*!
   * \brief Genesis private function which is the spark of live, using a random
   * seed
   * \param weights a reference to the used Weight_t vector
   * \param rangeweights pointer to the range of weights, currently it doesn't
   * support indivudal ranges
```

```
     * \param popSize maximum number of population, this should be an even number
     * \return
     */
    Population_t Genesis(const Weight_t &weights, uint32_t popSize);

    /*!
     * \brief CrossOver a private function where the partners mate with each other
     * The values or PopMember_t are expressed as bits or ar cut at the point
     * CROSSOVER
     * the population members are paired with the nearest neighbor and new members
     * are
     * created pairing the Genome_t of each other at the CROSSOVER point.
     * Afterwards all
     * the top tiers partners are allowed to mate again.
     * \param pop reference to the population
     */
    void CrossOver(Population_t &pop);

    /*!
     * \brief Mutate a private function where individual bits from the Genome_t
     * are mutated
     * at a random uniform distribution event defined by the MUTATIONRATE
     * \param pop reference to the population
     */
    void Mutate(Population_t &pop);

    /*!
     * \brief GrowToAdulthood a private function where the new population members
     * serve as the
     * the input for the Neural Network prediction function. The results are
     * weight against
     * the goal and this weight determine the fitness of the population member
     * \param pop reference to the population
     * \param inputValues a InputLearnVector_t with a reference to the inputvalues
     * \param rangeweights pointer to the range of weights, currently it doesn't
     * support individal ranges
     * \param goal a Predict_t type with the expected value
     * \param totalFitness a reference to the total population fitness
     */
    void GrowToAdulthood(Population_t &pop, float &totalFitness);

```

```
/*!
 * \brief SurvivalOfTheFittest a private function where a battle to the death
 * commences
 * The fittest population members have the best chance of survival. Death is
 * instigated
 * with a random uniform distibution. The elite members don't partake in this
 * desctruction
 * The ELITISME rate indicate how many top tier members survive this
 * catastrophic event.
 * \param inputValues a InputLearnVector_t with a reference to the inputvalues
 * \param totalFitness a reference to the total population fitness
 * \return
 */
bool SurvivalOfTheFittest(Population_t &pop, float &totalFitness);

/*!
 * \brief PopMemberSort a private function where the members are sorted
 * according to
 * there fitness ranking
 * \param i left hand population member
 * \param j right hand population member
 * \return true if the left member is closser to the goal as the right member.
 */
static bool PopMemberSort(PopMember_t i, PopMember_t j) {
  return (i.Fitness < j.Fitness);
}

/*!
 * \brief Conversion of the value of type T to Genome_t
 * \details Usage: Use <tt>ConvertToGenome<Type>(type, range)</tt>
 * \param value The current value wich should be converted to a Genome_t
 * \param range the range in which the value should fall, this is to have a
 * Genome_t
 * which utilizes the complete range 0000...n till 1111...n
 */
template <typename T>
inline Genome_t ConvertToGenome(T value, std::pair<T, T> range) {
  uint32_t intVal = static_cast<uint32_t>(
      (UINT32_MAX * (range.first + value)) / (range.second - range.first));
  Genome_t retVal(intVal);
  return retVal;
```

```
  }

  /*!
   * \brief Conversion of the Genome to a value
   * \details Usage: use <tt>ConvertToValue<Type>(genome,
      range)
   * \param gen is the Genome which is to be converted
   * \param range is the range in which the value should
      fall
   */
  template <typename T>
  inline T ConvertToValue(Genome_t gen, std::pair<T, T>
     range) {
    T retVal =
        range.first +
        (((range.second - range.first) * static_cast<T>(gen.
           to_ulong())) /
         UINT32_MAX);
    return retVal;
  }
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "GA.h"

namespace SoilMath {
GA::GA() {}

GA::GA(NNfunctionType nnfunction, uint32_t inputneurons,
    uint32_t hiddenneurons,
       uint32_t outputneurons) {
  this->NNfuction = nnfunction;
  this->inputneurons = inputneurons;
  this->hiddenneurons = hiddenneurons;
  this->outputneurons = outputneurons;
}

GA::~GA() {}

void GA::Evolve(const InputLearnVector_t &inputValues,
    Weight_t &weights,
                MinMaxWeight_t rangeweights,
                   OutputLearnVector_t &goal,
                uint32_t maxGenerations, uint32_t popSize) {
  minOptim = goal[0].OutputNeurons.size();
  minOptim = -minOptim;
  maxOptim = 2 * goal[0].OutputNeurons.size();
```

```
  oldElit = Elitisme;
  oldMutation = MutationRate;
  this->inputValues = inputValues;
  this->rangeweights = rangeweights;
  this->goal = goal;

  // Create the population
  Population_t pop = Genesis(weights, popSize);
  float totalFitness = 0.0;
  for (uint32_t i = 0; i < maxGenerations; i++) {
    CrossOver(pop);
    Mutate(pop);
    totalFitness = 0.0;
    GrowToAdulthood(pop, totalFitness);
    if (SurvivalOfTheFittest(pop, totalFitness)) {
      break;
    }
  }
  weights = pop[0].weights;
}

Population_t GA::Genesis(const Weight_t &weights, uint32_t
    popSize) {
  if (popSize < 1)
    return Population_t();

  Population_t pop;
  unsigned seed = std::chrono::system_clock::now().
      time_since_epoch().count();
  std::default_random_engine gen(seed);
  std::uniform_real_distribution<float> dis(rangeweights.
      first,
                                            rangeweights.
                                                second);

  for (uint32_t i = 0; i < popSize; i++) {
    PopMember_t I;
    for (uint32_t j = 0; j < weights.size(); j++) {
      I.weights.push_back(dis(gen));
      I.weightsGen.push_back(
          ConvertToGenome<float>(I.weights[j], rangeweights)
              );
    }
    pop.push_back(I);
  }
  return pop;
}

void GA::CrossOver(Population_t &pop) {
  Population_t newPop; // create a new population
  PopMember_t newPopMembers[2];
  SplitGenome_t Split[2];

  for (uint32_t i = 0; i < pop.size(); i += 2) {

    for (uint32_t j = 0; j < pop[i].weights.size(); j++) {
```

```
      // Split A
      Split[0].first = std::bitset<CROSSOVER>(
          pop[i].weightsGen[j].to_string().substr(0,
              CROSSOVER));
      Split[0].second = std::bitset<GENE_MAX - CROSSOVER>(
          pop[i].weightsGen[j].to_string().substr(CROSSOVER,
                                                  GENE_MAX -

                                                      CROSSOVER
                                                      ));

      // Split B
      Split[1].first = std::bitset<CROSSOVER>(
          pop[i + 1].weightsGen[j].to_string().substr(0,
              CROSSOVER));
      Split[1].second = std::bitset<GENE_MAX - CROSSOVER>(
          pop[i + 1].weightsGen[j].to_string().substr(
              CROSSOVER,
                                                      GENE_MAX
                                                          -
                                                        CROSSOVER
                                                        ));

      // Mate A and B to AB and BA
      newPopMembers[0].weightsGen.push_back(
          Genome_t(Split[0].first.to_string() + Split[1].
              second.to_string()));
      newPopMembers[1].weightsGen.push_back(
          Genome_t(Split[1].first.to_string() + Split[0].
              second.to_string()));
    }
    newPop.push_back(newPopMembers[0]);
    newPop.push_back(newPopMembers[1]);
    newPopMembers[0].weightsGen.clear();
    newPopMembers[1].weightsGen.clear();
  }

  // Allow the top tiers population partners to mate again
  uint32_t halfN = pop.size() / 2;
  for (uint32_t i = 0; i < halfN; i++) {
    for (uint32_t j = 0; j < pop[i].weights.size(); j++) {
      Split[0].first = std::bitset<CROSSOVER>(
          pop[i].weightsGen[j].to_string().substr(0,
              CROSSOVER));
      Split[0].second = std::bitset<GENE_MAX - CROSSOVER>(
          pop[i].weightsGen[j].to_string().substr(CROSSOVER,
                                                  GENE_MAX -

                                                      CROSSOVER
                                                      ));

      Split[1].first = std::bitset<CROSSOVER>(
          pop[i + 2].weightsGen[j].to_string().substr(0,
              CROSSOVER));
      Split[1].second = std::bitset<GENE_MAX - CROSSOVER>(
          pop[i + 2].weightsGen[j].to_string().substr(
```

```
                CROSSOVER,
                                                            GENE_MAX
                                                                -
                                                              CROSSOVER
                                                              ));

        newPopMembers[0].weightsGen.push_back(
            Genome_t(Split[0].first.to_string() + Split[1].
                second.to_string()));
        newPopMembers[1].weightsGen.push_back(
            Genome_t(Split[1].first.to_string() + Split[0].
                second.to_string()));
      }
      newPop.push_back(newPopMembers[0]);
      newPop.push_back(newPopMembers[1]);
      newPopMembers[0].weightsGen.clear();
      newPopMembers[1].weightsGen.clear();
    }
    pop = newPop;
}

void GA::Mutate(Population_t &pop) {
  unsigned seed = std::chrono::system_clock::now().
      time_since_epoch().count();
  std::default_random_engine gen(seed);
  std::uniform_real_distribution<float> dis(0, 1);

  std::default_random_engine genGen(seed);
  std::uniform_int_distribution<int> disGen(0, (GENE_MAX -
      1));

  QtConcurrent::blockingMap<Population_t>(pop, [&](
      PopMember_t &P) {
    for (uint32_t j = 0; j < P.weightsGen.size(); j++) {
      if (dis(gen) < MutationRate) {
        P.weightsGen[j][disGen(genGen)].flip();
      }
    }
  });
}

void GA::GrowToAdulthood(Population_t &pop, float &
    totalFitness) {

  QtConcurrent::blockingMap<Population_t>(pop, [&](
      PopMember_t &P) {
    // std::for_each(pop.begin(), pop.end(), [&](PopMember_t
        &P) {
    for (uint32_t j = 0; j < P.weightsGen.size(); j++) {
      P.weights.push_back(ConvertToValue<float>(P.weightsGen
          [j], rangeweights));
    }
    Weight_t iWeight(P.weights.begin(),
                     P.weights.begin() + ((inputneurons + 1)
                         * hiddenneurons));
    Weight_t hWeight(P.weights.begin() + ((inputneurons + 1)
```

```
* hiddenneurons),
                        P.weights.end());

    for (uint32_t j = 0; j < inputValues.size(); j++) {
      Predict_t results = NNfuction(inputValues[j], iWeight, hWeight,
                                          inputneurons, hiddenneurons, outputneurons);
      // See issue #85
      bool allGood = true;
      float fitness = 0.0;
      for (uint32_t k = 0; k < results.OutputNeurons.size(); k++) {
        bool resultSign = std::signbit(results.OutputNeurons[k]);
        bool goalSign = std::signbit(goal[j].OutputNeurons[k]);
        fitness += results.OutputNeurons[k] / goal[j].OutputNeurons[k];
        if (resultSign != goalSign) {
          allGood = false;
        }
      }
      fitness += (allGood) ? results.OutputNeurons.size() : 0;
      P.Fitness += fitness;
    }
  });

  for_each(pop.begin(), pop.end(), [&](PopMember_t &P) {
    P.Fitness /= inputValues.size();
    totalFitness += P.Fitness;
  });
}

bool GA::SurvivalOfTheFittest(Population_t &pop, float &totalFitness) {
  bool retVal = false;
  uint32_t decimationCount = pop.size() / 2;

  unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
  std::default_random_engine gen(seed);

  std::sort(pop.begin(), pop.end(),
            [](const PopMember_t &L, const PopMember_t &R) {
              return L.Fitness < R.Fitness;
            });

  float maxFitness = pop[pop.size() - 1].Fitness * pop.size();
  uint32_t i = Elitisme;
  while (pop.size() > decimationCount) {
    if (i == pop.size()) {
      i = Elitisme;
```

```
    }
    std::uniform_real_distribution<float> dis(0, maxFitness);
    if (dis(gen) > pop[i].Fitness) {
      totalFitness -= pop[i].Fitness;
      pop.erase(pop.begin() + i);
    }
    i++;
  }

  std::sort(pop.begin(), pop.end(),
            [](const PopMember_t &L, const PopMember_t &R) {
              return L.Fitness > R.Fitness;
            });

  float learnError = 1 - ((pop[0].Fitness - minOptim) / (maxOptim - minOptim));

  // Viva la Revolution
  if (currentGeneration > 9) {
    double avg = 0;
    for_each(last10Gen.begin(), last10Gen.end(), [&](double &G) { avg += G; });
    avg /= 10;
    double minMax[2] = {avg * 0.98, avg * 1.02};
    if (learnError > minMax[0] && learnError < minMax[1]) {
      if (!revolutionOngoing) {
        qDebug() << "Viva la revolution!";
        oldElit = Elitisme;
        Elitisme = 0;
        oldMutation = MutationRate;
        MutationRate = 0.25;
        revolutionOngoing = true;
      }
    } else if (revolutionOngoing) {
      qDebug() << "Peace has been restort";
      Elitisme = oldElit;
      MutationRate = oldMutation;
      revolutionOngoing = false;
    }
    last10Gen.pop_front();
    last10Gen.push_back(learnError);
  } else {
    last10Gen.push_back(learnError);
  }
  currentGeneration++;
  emit learnErrorUpdate(static_cast<double>(learnError));
  if (learnError < EndError) {
    retVal = true;
  }
  return retVal;
}
}
```

C.0.2 Fast Fourier Transform Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <vector>
#include <complex>
#include <cmath>
#include <valarray>
#include <array>
#include <deque>
#include <queue>
#include <iterator>
#include <algorithm>
#include <stdint.h>
#include <opencv2/core.hpp>
#include "SoilMathTypes.h"
#include "MathException.h"

namespace SoilMath {
/*!
 * \brief Fast Fourier Transform class
 * \details Use this class to transform a black and white blob presented as a
 * cv::Mat with values 0 or 1 to a vector of complex values representing the Fourier
 * Descriptors.
 */
class FFT {
public:
  /*!
 * \brief Standard constructor
 */
  FFT();

  /*!
  * \brief Standard deconstructor
  */
  ~FFT();

  /*!
   * \brief Transforming the img to the frequency domain and returning the
   * Fourier Descriptors
   * \param img contour in the form of a cv::Mat type CV_8UC1. Which should
   * consist of a continous contour. \f$ \{ img \in \mathbb{Z} | 0 \leq img \leq
```

```
     * 1 \} \f$
     * \return a vector with complex values, represing the contour in the
     * frequency domain, expressed as Fourier Descriptors
     */
    ComplexVect_t GetDescriptors(const cv::Mat &img);

private:
    ComplexVect_t
        fftDescriptors; /**< Vector with complex values which represent the
                                descriptors*/
    ComplexVect_t
        complexcontour; /**< Vector with complex values which represent the
                                contour*/
    cv::Mat Img;        /**< Img which will be analysed*/

    /*!
     * \brief Contour2Complex a private function which translates a continous
     * contour image
     * to a vector of complex values. The contour is found using a depth first
     * search with
     * extension list. The alghorithm is based upon <a
     * href="http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-4-search-depth-first-hill-climbing-beam/">MIT
     * opencourseware
     * 6-034-artificial-intelligence lecture 4</a>
     * \param img contour in the form of a cv::Mat type CV_8UC1. Which should
     * consist of a continous contour. \f$ \{ img \in \mathbb{Z} | 0 \leq img \leq
     * 1 \} \f$
     * \param centerCol centre of the contour X value
     * \param centerRow centre of the contour Y value
     * \return a vector with complex values, represing the contour as a function
     */
    ComplexVect_t Contour2Complex(const cv::Mat &img, float centerCol,
                                  float centerRow);
    /*!
     * \brief Neighbors a private function returning the neighboring pixels which
     * belong to a contour
     * \param O uchar pointer to the data
     * \param pixel current counter
     * \param columns total number of columns
     * \param rows total number of rows
     * \return
     */
```

```cpp
  iContour_t Neighbors(uchar *O, int pixel, uint32_t columns, uint32_t rows);

  /*!
   * \brief fft a private function calculating the Fast Fourier Transform
   * let \f$ m \f$ be an integer and let \f$ N=2^m \f$ also
   * \f$ CA=[x_0,\ldots,x_{N-1}] \f$ is an \f$ N \f$ dimensional complex vector
   * let \f$ \omega=\exp({-2\pi i\over N}) \f$
   * then \f$ c_k={\frac{1}{N}}\sum_{j=0}^{j=N-1}CA_j\omega^{jk} \f$
   * \param CA a \f$ CA=[x_0,\ldots,x_{N-1}] \f$ is an \f$ N \f$ dimensional
   * complex vector
   */
  void fft(ComplexArray_t &CA);

  /*!
   * \brief ifft
   * \param CA
   */
  void ifft(ComplexArray_t &CA);
};
}
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "FFT.h"

namespace SoilMath {
FFT::FFT() {}

FFT::~FFT() {}

ComplexVect_t FFT::GetDescriptors(const cv::Mat &img) {
  if (!fftDescriptors.empty()) {
    return fftDescriptors;
  }

  complexcontour = Contour2Complex(img, img.cols / 2, img.rows / 2);

  // Supplement the vector of complex numbers so that N = 2^m
  uint32_t N = complexcontour.size();
  double logN = log(static_cast<double>(N)) / log(2.0);
  if (floor(logN) != logN) {
    // Get the next power of 2
```

```
    double nextLogN = floor(logN + 1.0);
    N = static_cast<uint32_t>(pow(2, nextLogN));

    uint32_t i = complexcontour.size();
    // Append the vector with zeros
    while (i++ < N) {
      complexcontour.push_back(Complex_t(0.0, 0.0));
    }
  }

  ComplexArray_t ca(complexcontour.data(), complexcontour.size());
  fft(ca);
  fftDescriptors.assign(std::begin(ca), std::end(ca));
  return fftDescriptors;
}

iContour_t FFT::Neighbors(uchar *O, int pixel, uint32_t columns,
                          uint32_t rows) {
  long int LUT_nBore[8] = {-columns + 1, -columns, -columns - 1, -1,
                           columns - 1, columns, 1 + columns, 1};
  iContour_t neighbors;
  uint32_t pEnd = rows * columns;
  uint32_t count = 0;
  for (uint32_t i = 0; i < 8; i++) {
    count = pixel + LUT_nBore[i];
    while (count >= pEnd && i < 8) {
      count = pixel + LUT_nBore[++i];
    }
    if (i >= 8) {
      break;
    }
    if (O[count] == 1)
      neighbors.push_back(count);
  }
  return neighbors;
}

ComplexVect_t FFT::Contour2Complex(const cv::Mat &img, float centerCol,
                                   float centerRow) {
  uchar *O = img.data;
  uint32_t pEnd = img.cols * img.rows;

  std::deque<std::deque<uint32_t>> sCont;
  std::deque<uint32_t> eList;

  // Initialize the queue
  for (uint32_t i = 0; i < pEnd; i++) {
    if (O[i] == 1) {
      std::deque<uint32_t> tmpQ;
      tmpQ.push_back(i);
      sCont.push_back(tmpQ);
```

```
      break;
    }
  }

  if (sCont.front().size() < 1) {
    throw Exception::MathException(EXCEPTION_NO_CONTOUR_FOUND,
                                   EXCEPTION_NO_CONTOUR_FOUND_NR);
  } // Exception handling

  uint32_t prev = -1;

  // Extend path on queue
  for (uint32_t i = sCont.front().front(); i < pEnd;) {
    iContour_t nBors =
        Neighbors(0, i, img.cols, img.rows); // find neighboring pixels
    std::deque<uint32_t> cQ = sCont.front(); // store first queue;
    sCont.erase(sCont.begin());              // erase first queue from beginning
    if (cQ.size() > 1) {
      prev = cQ.size() - 2;
    } else {
      prev = 0;
    }
    // Loop through each neighbor
    for (uint32_t j = 0; j < nBors.size(); j++) {
      if (nBors[j] != cQ[prev]) // No backtracking
      {
        if (nBors[j] == cQ.front() && cQ.size() > 8) {
          i = pEnd;
        } // Back at first node
        if (std::find(eList.begin(), eList.end(), nBors[j]) ==
            eList.end()) // Check if this current route is extended elsewhere
        {
          std::deque<uint32_t> nQ = cQ;
          nQ.push_back(nBors[j]); // Add the neighbor to the queue
          sCont.push_front(nQ);   // add the sequence to the front of the queue
        }
      }
    }
    if (nBors.size() > 2) {
      eList.push_back(i);
    } // if there are multiple choices put current node in extension List
    if (i != pEnd) {
      i = sCont.front().back();
    } // If it isn't the end set i to the last node of the first queue
    if (sCont.size() == 0) {
```

```
        throw Exception::MathException(EXCEPTION_NO_CONTOUR_FOUND,
                                        EXCEPTION_NO_CONTOUR_FOUND_NR);
      }
    }

    // convert the first queue to a complex normalized vector
    Complex_t cPoint;
    ComplexVect_t contour;
    float col = 0.0;
    // Normalize and convert the complex function
    for_each(
        sCont.front().begin(), sCont.front().end(),
        [&img, &cPoint, &contour, &centerCol, &centerRow, &col](uint32_t &e) {
          col = (float)((e % img.cols) - centerCol);
          if (col == 0.0) {
            cPoint.real(1.0);
          } else {
            cPoint.real((float)(col / centerCol));
          }
          cPoint.imag((float)((floorf(e / img.cols) - centerRow) / centerRow));
          contour.push_back(cPoint);
        });

    return contour;
  }

  void FFT::fft(ComplexArray_t &CA) {
    const size_t N = CA.size();
    if (N <= 1) {
      return;
    }

    //!< Divide and conquor
    ComplexArray_t even = CA[std::slice(0, N / 2, 2)];
    ComplexArray_t odd = CA[std::slice(1, N / 2, 2)];

    fft(even);
    fft(odd);

    for (size_t k = 0; k < N / 2; ++k) {
      Complex_t ct = std::polar(1.0, -2 * M_PI * k / N) * odd[k];
      CA[k] = even[k] + ct;
      CA[k + N / 2] = even[k] - ct;
    }
  }

  void FFT::ifft(ComplexArray_t &CA) {
    CA = CA.apply(std::conj);
    fft(CA);
    CA = CA.apply(std::conj);
    CA /= CA.size();
```

```
}
}
```

C.0.3 Neural Network Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <stdint.h>
#include <vector>
#include <string>
#include <fstream>

#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/serialization/version.hpp>

#include "GA.h"
#include "MathException.h"
#include "SoilMathTypes.h"
#include "FFT.h"

#include <QtCore/QObject>

namespace SoilMath {
/*!
 * \brief The Neural Network class
 * \details This class is used to make prediction on large data set. Using self
 * learning algoritmes
 */
class NN : public QObject {
  Q_OBJECT

public:
  /*!
 * \brief NN constructor for the Neural Net
 * \param inputneurons number of input neurons
 * \param hiddenneurons number of hidden neurons
 * \param outputneurons number of output neurons
 */
  NN(uint32_t inputneurons, uint32_t hiddenneurons, uint32_t outputneurons);

  /*!
   * \brief NN constructor for the Neural Net
   */
  NN();

  /*!
```

```
     * \brief ~NN virtual deconstructor for the Neural Net
     */
    virtual ~NN();

    /*!
     * \brief Predict The prediction function.
     * \details In this function the neural net is setup and the input which are
     * the complex values descriping the contour in the frequency domein serve as
     * input. The absolute value of these im. number because I'm not interrested
     * in the orrientation of the particle but more in the degree of variations.
     * \param input vector of complex input values, these're the Fourier
     * descriptors
     * \return a real valued vector of the output neurons
     */
    Predict_t Predict(ComplexVect_t input);

    /*!
     * \brief PredictLearn a static function used in learning of the weights
     * \details It starts a new Neural Network object and passes all the
     * paramaters in to this newly created object. After this the predict function
     * is called and the value is returned. This work around was needed to pass
     * the neural network to the Genetic Algorithm class.
     * \param input a complex vector of input values
     * \param inputweights the input weights
     * \param hiddenweights the hidden weights
     * \param inputneurons the input neurons
     * \param hiddenneurons the hidden neurons
     * \param outputneurons the output neurons
     * \return
     */
    static Predict_t PredictLearn(ComplexVect_t input, Weight_t inputweights,
                                  Weight_t hiddenweights, uint32_t inputneurons,
                                  uint32_t hiddenneurons, uint32_t outputneurons);

    /*!
     * \brief SetInputWeights a function to set the input weights
     * \param value the real valued vector with the values
     */
    void SetInputWeights(Weight_t value) { iWeights = value; }

    /*!
     * \brief SetHiddenWeights a function to set the hidden weights
```

```
     * \param value the real valued vector with the values
     */
    void SetHiddenWeights(Weight_t value) { hWeights = value; }

    /*!
     * \brief SetBeta a function to set the beta value
     * \param value a floating value ussualy between 0.5 and 1.5
     */
    void SetBeta(float value) { beta = value; }
    float GetBeta() { return beta; }

    /*!
     * \brief Learn the learning function
     * \param input a vector of vectors with complex input values
     * \param cat a vector of vectors with the know output values
     * \param noOfDescriptorsUsed the total number of descriptos which should be
     * used
     */
    void Learn(InputLearnVector_t input, OutputLearnVector_t cat,
               uint32_t noOfDescriptorsUsed);

    /*!
     * \brief SaveState Serialize and save the values of the Neural Net to disk
     * \details Save the Neural Net in XML valued text file to disk so that a
     * object can
     * be reconstructed on a latter stadia.
     * \param filename a string indicating the file location and name
     */
    void SaveState(std::string filename);

    /*!
     * \brief LoadState Loads the previouse saved Neural Net from disk
     * \param filename a string indicating the file location and name
     */
    void LoadState(std::string filename);

    Weight_t iWeights; /**< a vector of real valued floating point input weights*/
    Weight_t hWeights; /**< a vector of real valued floating point hidden weight*/

    uint32_t MaxGenUsedByGA = 200;
    uint32_t PopulationSizeUsedByGA = 30;
    float MutationrateUsedByGA = 0.075f;
    uint32_t ElitismeUsedByGA = 4;
```

```
  float EndErrorUsedByGA = 0.001;
  float MaxWeightUsedByGA = 50;
  float MinWeightUSedByGa = -50;

  uint32_t GetInputNeurons() { return inputNeurons; }
  void SetInputNeurons(uint32_t value);

  uint32_t GetHiddenNeurons() { return hiddenNeurons; }
  void SetHiddenNeurons(uint32_t value);

  uint32_t GetOutputNeurons() { return outputNeurons; }
  void SetOutputNeurons(uint32_t value);

  bool studied =
      false; /**< a value indicating if the weights are a results of a
                 learning curve*/

signals:
  void learnErrorUpdate(double newError);

private:
  GA *optim = nullptr;
  std::vector<float> iNeurons; /**< a vector of input values, the bias is
                                  included, the bias is included and
                                  is the first value*/
  std::vector<float>
      hNeurons; /**< a vector of hidden values, the bias is included and
                    is the first value*/
  std::vector<float> oNeurons; /**< a vector of output values*/

  uint32_t hiddenNeurons = 50; /**< number of hidden neurons minus bias*/
  uint32_t inputNeurons = 20;  /**< number of input neurons minus bias*/
  uint32_t outputNeurons = 18; /**< number of output neurons */
  float beta; /**< the beta value, this indicates the steepness of the sigmoid
                 function*/

  friend class boost::serialization::access; /**< a private friend class so the
                                                serialization can access all
                                                the needed functions */
  /*!
   * \brief serialization function
```

```
   * \param ar the object
   * \param version the version of the class
   */
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {
    if (version == 0) {
      ar &BOOST_SERIALIZATION_NVP(inputNeurons);
      ar &BOOST_SERIALIZATION_NVP(hiddenNeurons);
      ar &BOOST_SERIALIZATION_NVP(outputNeurons);
      ar &BOOST_SERIALIZATION_NVP(iWeights);
      ar &BOOST_SERIALIZATION_NVP(hWeights);
      ar &BOOST_SERIALIZATION_NVP(beta);
      ar &BOOST_SERIALIZATION_NVP(studied);
      ar &BOOST_SERIALIZATION_NVP(MaxGenUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(PopulationSizeUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(MutationrateUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(ElitismeUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(EndErrorUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(MaxWeightUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(MinWeightUSedByGa);
    }
  }
};
}
BOOST_CLASS_VERSION(SoilMath::NN, 0)
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "NN.h"
using namespace std;

namespace SoilMath {
NN::NN() { beta = 0.666; }

NN::NN(uint32_t inputneurons, uint32_t hiddenneurons, uint32_t outputneurons) {
  // Set the number of neurons in the network
  inputNeurons = inputneurons;
  hiddenNeurons = hiddenneurons;
  outputNeurons = outputneurons;
  // Reserve the vector space
  iNeurons.reserve(inputNeurons + 1);  // input neurons + bias
  hNeurons.reserve(hiddenNeurons + 1); // hidden neurons + bias
  oNeurons.reserve(outputNeurons);     // output neurons

  beta = 0.666;
}
```

```

NN::~NN()
{
  if (optim != nullptr) {
      delete optim;
    }
}

void NN::LoadState(string filename) {
  std::ifstream ifs(filename.c_str());
  boost::archive::xml_iarchive ia(ifs);
  ia >> boost::serialization::make_nvp("NeuralNet", *this);
}

void NN::SaveState(string filename) {
  std::ofstream ofs(filename.c_str());
  boost::archive::xml_oarchive oa(ofs);
  oa << boost::serialization::make_nvp("NeuralNet", *this);
}

Predict_t NN::PredictLearn(ComplexVect_t input, Weight_t inputweights,
                           Weight_t hiddenweights, uint32_t inputneurons,
                           uint32_t hiddenneurons, uint32_t outputneurons) {
  NN neural(inputneurons, hiddenneurons, outputneurons);
  neural.studied = true;
  neural.SetInputWeights(inputweights);
  neural.SetHiddenWeights(hiddenweights);
  return neural.Predict(input);
}

Predict_t NN::Predict(ComplexVect_t input) {
  if (input.size() != inputNeurons) {
    throw Exception::MathException(EXCEPTION_SIZE_OF_INPUT_NEURONS,
                                   EXCEPTION_SIZE_OF_INPUT_NEURONS_NR);
  }
  if (!studied) {
    throw Exception::MathException(EXCEPTION_NEURAL_NET_NOT_STUDIED,
                                   EXCEPTION_NEURAL_NET_NOT_STUDIED_NR);
  }

  iNeurons.clear();
  hNeurons.clear();
  oNeurons.clear();

  // Set the bias in the input and hidden vector to 1 (real number)
  iNeurons.push_back(1.0f);
  hNeurons.push_back(1.0f);

```

```
  Predict_t retVal;
  uint32_t wCount = 0;

  // Init the network
  for (uint32_t i = 0; i < inputNeurons; i++) {
    iNeurons.push_back(static_cast<float>(abs(input[i])));
  }
  for (uint32_t i = 0; i < hiddenNeurons; i++) {
    hNeurons.push_back(0.0f);
  }
  for (uint32_t i = 0; i < outputNeurons; i++) {
    oNeurons.push_back(0.0f);
  }

  for (uint32_t i = 1; i < hNeurons.size(); i++) {
    wCount = i - 1;
    for (uint32_t j = 0; j < iNeurons.size(); j++) {
      hNeurons[i] += iNeurons[j] * iWeights[wCount];
      wCount += hNeurons.size() - 1;
    }
    hNeurons[i] = 1 / (1 + pow(2.71828f, (-hNeurons[i] * beta)));
  }

  for (uint32_t i = 0; i < oNeurons.size(); i++) {
    wCount = i;
    for (uint32_t j = 0; j < hNeurons.size(); j++) {
      oNeurons[i] += hNeurons[j] * hWeights[wCount];
      wCount += oNeurons.size();
    }
    oNeurons[i] =
        (2 / (1.0f + pow(2.71828f, (-oNeurons[i] * beta)))) -
        1; // Shift plus scale so the learning function can be calculated
  }

  retVal.OutputNeurons = oNeurons;
  retVal.ManualSet = false;
  return retVal;
}

void NN::Learn(InputLearnVector_t input, OutputLearnVector_t cat,
               uint32_t noOfDescriptorsUsed __attribute__((unused))) {
  if (optim == nullptr) {
      optim = new SoilMath::GA(PredictLearn, inputNeurons, hiddenNeurons, outputNeurons);
    }
  connect(optim, SIGNAL(learnErrorUpdate(double)), this, SIGNAL(learnErrorUpdate(double)));

  optim->Elitisme = ElitismeUsedByGA;
  optim->EndError = EndErrorUsedByGA;
  optim->MutationRate = MutationrateUsedByGA;
```

```
  ComplexVect_t inputTest;
  std::vector<Weight_t> weights;
  Weight_t weight(((inputNeurons + 1) * hiddenNeurons) +
                      ((hiddenNeurons + 1) * outputNeurons),
                  0);
  // loop through each case and adjust the weights
  optim->Evolve(input, weight,
                MinMaxWeight_t(MinWeightUSedByGa,
                    MaxWeightUsedByGA), cat,
                MaxGenUsedByGA, PopulationSizeUsedByGA);

  this->iWeights = Weight_t(
      weight.begin(), weight.begin() + ((inputNeurons + 1) *
          hiddenNeurons));
  this->hWeights = Weight_t(
      weight.begin() + ((inputNeurons + 1) * hiddenNeurons),
          weight.end());
  studied = true;
}

void NN::SetInputNeurons(uint32_t value) {
  if (value != inputNeurons) {
    inputNeurons = value;
    iNeurons.clear();
    iNeurons.reserve(inputNeurons + 1);
    studied = false;
  }
}

void NN::SetHiddenNeurons(uint32_t value) {
  if (value != hiddenNeurons) {
    hiddenNeurons = value;
    hNeurons.clear();
    hNeurons.reserve(hiddenNeurons + 1);
    studied = false;
  }
}

void NN::SetOutputNeurons(uint32_t value) {
  if (value != outputNeurons) {
    outputNeurons = value;
    oNeurons.clear();
    oNeurons.reserve(outputNeurons);
    studied = false;
  }
}
}
```

C.0.4 Statistical Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#define MAX_UINT8_VALUE 256
#define VECTOR_CALC 1

#include <stdint.h>
#include <utility>
#include <vector>
#include <cstdlib>
#include <cmath>
#include <limits>
#include <typeinfo>
#include <string>

#include <fstream>

#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/version.hpp>
#include <boost/math/distributions/students_t.hpp>

#include "MathException.h"
#include "SoilMathTypes.h"
#include "CommonOperations.h"

namespace SoilMath {

/*!
 * \brief Stats class
 * \details Usage Stats<type1, type2, type3>Stats() type 1, 2 and 3 shoudl be of
 * the same value and concecuative in size
 */
template <typename T1, typename T2, typename T3> class Stats {
public:
  bool isDiscrete = true; /**< indicates if the data is discrete or real*/

  T1 *Data = nullptr;         /**< Pointer the data*/
  uint32_t *bins = nullptr; /**< the histogram*/
  double *CFD = nullptr;     /**< the CFD*/
  bool Calculated = false;  /**< indication if the data has been calculated*/
  float Mean = 0.0;          /**< the mean value of the data */
```

```
    uint32_t n = 0;              /**< number of data points*/
    uint32_t noBins = 0;         /**< number of bins*/
    T1 Range = 0;                /**< range of the data*/
    T1 min = 0;                  /**< minimum value*/
    T1 max = 0;                  /**< maximum value*/
    T1 Startbin = 0;             /**< First bin value*/
    T1 EndBin = 0;               /**< End bin value*/
    T1 binRange = 0;             /**< the range of a single bin*/
    float Std = 0.0;             /**< standard deviation*/
    T3 Sum = 0;                  /**< total sum of all the data values*/
    uint16_t Rows = 0;           /**< number of rows from the data matrix*/
    uint16_t Cols = 0;           /**< number of cols from the data matrix*/
    bool StartAtZero = true;     /**< indication of the minimum value starts at zero
                                     or could be less*/
    double *BinRanges = nullptr;
    double HighestPDF = 0.;

    uint32_t *begin() { return &bins[0]; }     /**< pointer to the first bin*/
    uint32_t *end() { return &bins[noBins]; } /**< pointer to the last + 1 bin*/

    /*!
     * \brief WelchTest Compare the sample using the Welch's Test
     * \details (source:
     * http://www.boost.org/doc/libs/1_57_0/libs/math/doc/html/math_toolkit/stat_tut/weg/st_eg/two_sample_students_t.html)
     * \param statComp Statiscs Results of which it should be tested against
     * \return
     */
    bool WelchTest(SoilMath::Stats<T1, T2, T3> &statComp) {
      double alpha = 0.05;
      // Degrees of freedom:
      double v = statComp.Std * statComp.Std / statComp.n +
                 this->Std * this->Std / this->n;
      v *= v;
      double t1 = statComp.Std * statComp.Std / statComp.n;
      t1 *= t1;
      t1 /= (statComp.n - 1);
      double t2 = this->Std * this->Std / this->n;
      t2 *= t2;
      t2 /= (this->n - 1);
      v /= (t1 + t2);
      // t-statistic:
      double t_stat = (statComp.Mean - this->Mean) /
                      sqrt(statComp.Std * statComp.Std / statComp.n +
                           this->Std * this->Std / this->n);
      //
```

```
    // Define our distribution, and get the probability:
    //
    boost::math::students_t dist(v);
    double q = cdf(complement(dist, fabs(t_stat)));

    bool rejected = false;
    // Sample 1 Mean == Sample 2 Mean test the NULL hypothesis, the two means
    // are the same
    if (q < alpha / 2)
      rejected = false;
    else
      rejected = true;
    return rejected;
  }

  /*!
   * \brief Stats Constructor
   * \param rhs Right hand side
   */
  Stats(const Stats &rhs)
      : bins{new uint32_t[rhs.noBins]{0}}, CFD{new double[rhs.noBins]{}},
        BinRanges{new double[rhs.noBins]{}} {
    this->binRange = rhs.binRange;
    this->Calculated = rhs.Calculated;
    this->Cols = rhs.Cols;
    this->EndBin = rhs.EndBin;
    this->isDiscrete = rhs.isDiscrete;
    this->max = rhs.max;
    this->Mean = rhs.Mean;
    this->min = rhs.min;
    this->n = rhs.n;
    this->noBins = rhs.noBins;
    this->n_end = rhs.n_end;
    this->Range = rhs.Range;
    this->Rows = rhs.Rows;
    this->Startbin = rhs.Startbin;
    this->Std = rhs.Std;
    this->Sum = rhs.Sum;
    std::copy(rhs.bins, rhs.bins + rhs.noBins, this->bins);
    std::copy(rhs.CFD, rhs.CFD + rhs.noBins, this->CFD);
    std::copy(rhs.BinRanges, rhs.BinRanges + rhs.noBins, this->BinRanges);
    this->Data = rhs.Data;
    this->StartAtZero = rhs.StartAtZero;
    this->HighestPDF = rhs.HighestPDF;
  }

  /*!
   * \brief operator = Assigmnet operator
   * \param rhs right hand side
   * \return returns the right hand side
   */
  Stats &operator=(Stats const &rhs) {
    if (&rhs != this) {
```

```
    Data = rhs.Data;

    if (bins != nullptr) {
      delete[] bins;
      bins = nullptr;
    }
    if (CFD != nullptr) {
      delete[] CFD;
      CFD = nullptr;
    }
    if (BinRanges != nullptr) {
      delete[] BinRanges;
      BinRanges = nullptr;
    }

    bins = new uint32_t[rhs.noBins];     // leak
    CFD = new double[rhs.noBins];        // leak
    BinRanges = new double[rhs.noBins]; // leak
    this->binRange = rhs.binRange;
    this->Calculated = rhs.Calculated;
    this->Cols = rhs.Cols;
    this->EndBin = rhs.EndBin;
    this->isDiscrete = rhs.isDiscrete;
    this->max = rhs.max;
    this->Mean = rhs.Mean;
    this->min = rhs.min;
    this->n = rhs.n;
    this->noBins = rhs.noBins;
    this->n_end = rhs.n_end;
    this->Range = rhs.Range;
    this->Rows = rhs.Rows;
    this->Startbin = rhs.Startbin;
    this->Std = rhs.Std;
    this->Sum = rhs.Sum;
    this->Data = &rhs.Data[0];
    std::copy(rhs.bins, rhs.bins + rhs.noBins, this->bins);
    std::copy(rhs.CFD, rhs.CFD + rhs.noBins, this->CFD);
    std::copy(rhs.BinRanges, rhs.BinRanges + rhs.noBins,
        this->BinRanges);
    this->StartAtZero = rhs.StartAtZero;
    this->HighestPDF = rhs.HighestPDF;
  }
  return *this;
}

/*!
 * \brief Stats Constructor
 * \param noBins number of bins with which to build the histogram
 * \param startBin starting value of the first bin
 * \param endBin end value of the second bin
 */
Stats(int noBins = 256, T1 startBin = 0, T1 endBin = 255) {
  min = std::numeric_limits<T1>::max();
```

```
    max = std::numeric_limits<T1>::min();
    Range = std::numeric_limits<T1>::max();
    Startbin = startBin;
    EndBin = endBin;
    this->noBins = noBins;
    bins = new uint32_t[noBins]{0};    // leak
    CFD = new double[noBins]{};        // leak
    BinRanges = new double[noBins]{}; // leak

    if (typeid(T1) == typeid(float) || typeid(T1) == typeid(
        double) ||
        typeid(T1) == typeid(long double)) {
      isDiscrete = false;
      binRange = static_cast<T1>((EndBin - Startbin) /
          noBins);
    } else {
      isDiscrete = true;
      binRange = static_cast<T1>(round((EndBin - Startbin) /
          noBins));
    }
  }

  /*!
   * \brief Stats constructor
   * \param data Pointer to the data
   * \param rows Number of rows
   * \param cols Number of Columns
   * \param noBins Number of bins
   * \param startBin Value of the start bin
   * \param startatzero bool indicating if the bins should
      be shifted from zero
   */
  Stats(T1 *data, uint16_t rows, uint16_t cols, int noBins =
      256,
        T1 startBin = 0, bool startatzero = true) {
    min = std::numeric_limits<T1>::max();
    max = std::numeric_limits<T1>::min();
    Range = max - min;

    Startbin = startBin;
    EndBin = startBin + noBins;
    StartAtZero = startatzero;

    if (typeid(T1) == typeid(float) || typeid(T1) == typeid(
        double) ||
        typeid(T1) == typeid(long double)) {
      isDiscrete = false;
    } else {
      isDiscrete = true;
    }

    Data = data;
    Rows = rows;
    Cols = cols;
    bins = new uint32_t[noBins]{0};
    CFD = new double[noBins]{};
```

```
    BinRanges = new double[noBins]{};
    this->noBins = noBins;
    if (isDiscrete) {
      BasicCalculate();
    } else {
      BasicCalculateFloat();
    }
  }

  /*!
   * \brief Stats Constructor
   * \param data Pointer the data
   * \param rows Number of rows
   * \param cols Number of Columns
   * \param mask the mask should have the same size as the data a value of zero
   * indicates that the data pointer doesn't exist. A 1 indicates that the data
   * pointer is to be used
   * \param noBins Number of bins
   * \param startBin Value of the start bin
   * \param startatzero indicating if the bins should be shifted from zero
   */
  Stats(T1 *data, uint16_t rows, uint16_t cols, uchar *mask, int noBins = 256,
        T1 startBin = 0, bool startatzero = true) {
    min = std::numeric_limits<T1>::max();
    max = std::numeric_limits<T1>::min();
    Range = max - min;

    Startbin = startBin;
    EndBin = startBin + noBins;
    StartAtZero = startatzero;

    if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
        typeid(T1) == typeid(long double)) {
      isDiscrete = false;
    } else {
      isDiscrete = true;
    }

    Data = data;
    Rows = rows;
    Cols = cols;
    bins = new uint32_t[noBins]{0};
    CFD = new double[noBins]{};
    BinRanges = new double[noBins]{};
    this->noBins = noBins;
    if (isDiscrete) {
      BasicCalculate(mask);
    } else {
      BasicCalculateFloat(mask);
    }
  }
```

```
  /*!
   * \brief Stats Constructor
   * \param binData The histogram data
   * \param startC start counter
   * \param endC end counter
   */
  Stats(T2 *binData, uint16_t startC, uint16_t endC) {
    noBins = endC - startC;
    Startbin = startC;
    EndBin = endC;
    uint32_t i = noBins;

    if (typeid(T1) == typeid(float) || typeid(T1) == typeid(
        double) ||
        typeid(T1) == typeid(long double)) {
      isDiscrete = false;
      throw Exception::MathException(
          EXCEPTION_TYPE_NOT_SUPPORTED,
                                    EXCEPTION_TYPE_NOT_SUPPORTED_NR
                                        );
    } else {
      isDiscrete = true;
    }

    bins = new uint32_t[noBins]{0};
    CFD = new double[noBins]{};
    BinRanges = new double[noBins]{};
    while (i-- > 0) {
      bins[i] = binData[i];
      n += binData[i];
    }
    BinCalculations(startC, endC);
  }

  ~Stats() {
    Data == nullptr;
    if (bins != nullptr) {
      delete[] bins;
      bins = nullptr;
    }
    if (CFD != nullptr) {
      delete[] CFD;
      CFD = nullptr;
    }
    if (BinRanges != nullptr) {
      delete[] BinRanges;
      BinRanges = nullptr;
    }
  }

  /*!
   * \brief BasicCalculateFloat execute the basic float data
       calculations
   */
  void BasicCalculateFloat() {
```

```
    float sum_dev = 0.0;
    n = Rows * Cols;
    for (uint32_t i = 0; i < n; i++) {
      if (Data[i] > max) {
        max = Data[i];
      }
      if (Data[i] < min) {
        min = Data[i];
      }
      Sum += Data[i];
    }
    binRange = (max - min) / noBins;
    uint32_t index = 0;
    Mean = Sum / (float)n;
    Range = max - min;

    if (StartAtZero) {
      for (uint32_t i = 0; i < n; i++) {
        index = static_cast<uint32_t>(Data[i] / binRange);
        if (index == noBins) {
          index -= 1;
        }
        bins[index]++;
        sum_dev += pow((Data[i] - Mean), 2);
      }
    } else {
      for (uint32_t i = 0; i < n; i++) {
        index = static_cast<uint32_t>((Data[i] - min) /
            binRange);
        if (index == noBins) {
          index -= 1;
        }
        bins[index]++;
        sum_dev += pow((Data[i] - Mean), 2);
      }
    }
    Std = sqrt((float)(sum_dev / n));
    getCFD();
    Calculated = true;
  }

  /*!
   * \brief BasicCalculateFloat execute the basic float data
       calculations with a
   * mask
   * \param mask uchar mask type 0 don't calculate, 1
      calculate
   */
  void BasicCalculateFloat(uchar *mask) {
    float sum_dev = 0.0;
    n = Rows * Cols;
    uint32_t nmask = 0;
    for (uint32_t i = 0; i < n; i++) {
      if (mask[i] != 0) {
        if (Data[i] > max) {
          max = Data[i];
```

```
        }
        if (Data[i] < min) {
          min = Data[i];
        }
        Sum += Data[i];
        nmask++;
      }
    }
    binRange = (max - min) / noBins;
    uint32_t index = 0;
    Mean = Sum / (float)nmask;
    Range = max - min;
    if (StartAtZero) {
      for (uint32_t i = 0; i < n; i++) {
        if (mask[i] != 0) {
          index = static_cast<uint32_t>(Data[i] / binRange);
          if (index == noBins) {
            index -= 1;
          }
          bins[index]++;
          sum_dev += pow((Data[i] - Mean), 2);
        }
      }
    } else {
      for (uint32_t i = 0; i < n; i++) {
        if (mask[i] != 0) {
          index = static_cast<uint32_t>((Data[i] - min) /
              binRange);
          if (index == noBins) {
            index -= 1;
          }
          bins[index]++;
          sum_dev += pow((Data[i] - Mean), 2);
        }
      }
    }
    Std = sqrt((float)(sum_dev / nmask));
    getCFD();
    Calculated = true;
  }

  /*!
   * \brief BasicCalculate execute the basic discrete data
      calculations
   */
  void BasicCalculate() {
    double sum_dev = 0.0;
    n = Rows * Cols;
    for (uint32_t i = 0; i < n; i++) {
      if (Data[i] > max) {
        max = Data[i];
      }
      if (Data[i] < min) {
        min = Data[i];
      }
      Sum += Data[i];
```

```
    }
    binRange = static_cast<T1>(ceil((max - min) /
        static_cast<float>(noBins)));
    if (binRange == 0) {
      binRange = 1;
    }
    Mean = Sum / (float)n;
    Range = max - min;

    uint32_t index;
    if (StartAtZero) {
      std::for_each(Data, Data + n, [&](T1 &d) {
        index = static_cast<uint32_t>(d / binRange);
        if (index == noBins) {
          index -= 1;
        }
        bins[index]++;
        sum_dev += pow((d - Mean), 2);
      });
    } else {
      std::for_each(Data, Data + n, [&](T1 &d) {
        index = static_cast<uint32_t>((d - min) / binRange);
        if (index == noBins) {
          index -= 1;
        }
        bins[index]++;
        sum_dev += pow((d - Mean), 2);
      });
    }
    Std = sqrt((float)(sum_dev / n));
    getCFD();
    Calculated = true;
  }

  /*!
   * \brief BasicCalculate execute the basic discrete data
      calculations with
   * mask
   * \param mask uchar mask type 0 don't calculate, 1
      calculate
   */
  void BasicCalculate(uchar *mask) {
    double sum_dev = 0.0;
    n = Rows * Cols;
    uint32_t nmask = 0;
    uint32_t i = 0;
    std::for_each(Data, Data + n, [&](T1 &d) {
      if (mask[i++] != 0) {
        if (d > max) {
          max = d;
        }
        if (d < min) {
          min = d;
        }
        Sum += d;
        nmask++;
```

```
      }
    });
    binRange = static_cast<T1>(ceil((max - min) / static_cast<float>(noBins)));
    Mean = Sum / (float)nmask;
    Range = max - min;

    uint32_t index;
    if (StartAtZero) {
      i = 0;
      std::for_each(Data, Data + n, [&](T1 &d) {
        if (mask[i++] != 0) {
          index = static_cast<uint32_t>(d / binRange);
          if (index == noBins) {
            index -= 1;
          }
          bins[index]++;
          sum_dev += pow((d - Mean), 2);
        }
      });
    } else {
      i = 0;
      std::for_each(Data, Data + n, [&](T1 &d) {
        if (mask[i++] != 0) {
          index = static_cast<uint32_t>((d - min) / binRange);
          if (index == noBins) {
            index -= 1;
          }
          bins[index]++;
          sum_dev += pow((d - Mean), 2);
        }
      });
    }
    Std = sqrt((float)(sum_dev / nmask));
    getCFD();
    Calculated = true;
  }

  /*!
   * \brief BinCalculations excute the cacluations with the
      histogram
   * \param startC start counter
   * \param endC end counter
   */
  void BinCalculations(uint16_t startC, uint16_t endC __attribute__((unused))) {
    float sum_dev = 0.0;
    // Get the Sum
    uint32_t i = 0;
    for_each(begin(), end(), [&](uint32_t &b) { Sum += b * (startC + i++); });

    // Get Mean
    Mean = Sum / (float)n;

```

```
    // Get max
    for (int i = noBins - 1; i >= 0; i--) {
      if (bins[i] != 0) {
        max = i + startC;
        break;
      }
    }

    // Get min
    for (uint32_t i = 0; i < noBins; i++) {
      if (bins[i] != 0) {
        min = i + startC;
        break;
      }
    }

    // Get Range;
    Range = max - min;

    // Calculate Standard Deviation
    i = 0;
    for_each(begin(), end(), [&](uint32_t &b) {
      sum_dev += b * pow(((i++ + startC) - Mean), 2);
    });
    Std = sqrt((float)(sum_dev / n));
    getCFD();
    Calculated = true;
  }

  uint32_t HighestFrequency() {
    uint32_t freq = 0;
    std::for_each(begin(), end(), [&](uint32_t &B) {
      if (B > freq) {
        freq = B;
      }
    });
    return freq;
  }

  void GetPDFfunction(std::vector<double> &xAxis, std::
      vector<double> &yAxis,
                          double Step, double start = 0, double
                              stop = 7) {
    uint32_t resolution;
    resolution = static_cast<uint32_t>(((stop - start) /
        Step) + 0.5);

    xAxis.push_back(start);
    double yVal0 = (1 / (Std * 2.506628274631)) *
                      exp(-(pow((start - Mean), 2) / (2 * pow(
                          Std, 2))));
    yAxis.push_back(yVal0);
    HighestPDF = yVal0;
    for (uint32_t i = 1; i < resolution; i++) {
      double xVal = xAxis[xAxis.size() - 1] + Step;
      xAxis.push_back(xVal);
```

```
      double yVal = (1 / (Std * 2.506628274631)) *
                    exp(-(pow((xVal - Mean), 2) / (2 * pow(
                        Std, 2))));
      yAxis.push_back(yVal);
      if (yVal > HighestPDF) {
        HighestPDF = yVal;
      }
    }
  }

protected:
  uint32_t n_end = 0; /**< data end counter used with mask*/

  /*!
   * \brief getCFD get the CFD matrix;
   */
  void getCFD() {
    uint32_t *sumBin = new uint32_t[noBins];
    sumBin[0] = bins[0];
    CFD[0] = (static_cast<double>(sumBin[0]) / static_cast<
        double>(n)) * 100.;
    for (uint32_t i = 1; i < noBins; i++) {
      sumBin[i] = (sumBin[i - 1] + bins[i]);
      CFD[i] = (static_cast<double>(sumBin[i]) / static_cast
          <double>(n)) * 100.;
      if (CFD[i] > HighestPDF) {
        HighestPDF = CFD[i];
      }
    }
    delete[] sumBin;
  }

  friend class boost::serialization::access; /**<
      Serialization class*/

  /*!
   * \brief serialize the object
   * \param ar argument
   * \param version
   */
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {
    if (version == 0) {
      ar &isDiscrete;
      ar &n;
      ar &noBins;
      for (size_t dc = 0; dc < noBins; dc++) {
        ar &bins[dc];
      }
      for (size_t dc = 0; dc < noBins; dc++) {
        ar &CFD[dc];
      }
      for (size_t dc = 0; dc < noBins; dc++) {
        ar &BinRanges[dc];
      }
      ar &Calculated;
```

```
      ar &Mean;
      ar &Range;
      ar &min;
      ar &max;
      ar &Startbin;
      ar &EndBin;
      ar &binRange;
      ar &Std;
      ar &Sum;
      ar &Rows;
      ar &Cols;
      ar &StartAtZero;
      ar &HighestPDF;
    }
  }
};
}

typedef SoilMath::Stats<float, double, long double>
    floatStat_t; /**< floating Stat type*/
typedef SoilMath::Stats<uchar, uint32_t, uint64_t>
    ucharStat_t; /**< uchar Stat type*/
typedef SoilMath::Stats<uint16_t, uint32_t, uint64_t>
    uint16Stat_t; /**< uint16 Stat type*/
typedef SoilMath::Stats<uint32_t, uint32_t, uint64_t>
    uint32Stat_t; /**< uint32 Stat type*/
BOOST_CLASS_VERSION(floatStat_t, 0)
BOOST_CLASS_VERSION(ucharStat_t, 0)
BOOST_CLASS_VERSION(uint16Stat_t, 0)
BOOST_CLASS_VERSION(uint32Stat_t, 0)
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include "Stats.h"
#include <boost/serialization/base_object.hpp>

namespace SoilMath {
class PSD : public SoilMath::Stats<double, double, long double> {
private:
  uint32_t DetBin(float value) {
    uint32_t i = noBins - 1;
    while (i > 0) {
      if (value > BinRanges[i]) {
        return i;
      }
      i--;
```

```
    }
    return 0;
  }

  void BasicCalculatePSD() {
    float sum_dev = 0.0;
    n = Rows * Cols;
    for (uint32_t i = 0; i < n; i++) {
      if (Data[i] > max) {
        max = Data[i];
      }
      if (Data[i] < min) {
        min = Data[i];
      }
      Sum += Data[i];
    }
    uint32_t index = 0;
    Mean = Sum / (float)n;
    Range = max - min;
    for (uint32_t i = 0; i < n; i++) {
      index = DetBin(Data[i]);
      bins[index]++;
      sum_dev += pow((Data[i] - Mean), 2);
    }
    Std = sqrt((float)(sum_dev / n));
    getCFD();
    Calculated = true;
  }
  friend class boost::serialization::access;

  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {
    if (version == 0) {
      ar &boost::serialization::base_object<
          SoilMath::Stats<double, double, long double>>(*
             this);
    }
  }

public:
  PSD() : SoilMath::Stats<double, double, long double>() {}

  PSD(double *data, uint32_t nodata, double *binranges,
     uint32_t nobins,
      uint32_t endbin)
      : SoilMath::Stats<double, double, long double>(nobins,
          0, endbin) {
    std::copy(binranges, binranges + nobins, BinRanges);
    Data = data;
    Rows = nodata;
    Cols = 1;

    BasicCalculatePSD();
  }
};
}
```

```
BOOST_CLASS_VERSION(SoilMath::PSD, 0)
```

C.0.5 General project files

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-06-06T11:59:21
#
#-------------------------------------------------

QT       += core gui concurrent
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = SoilMath
TEMPLATE = lib
VERSION = 0.9.8

DEFINES += SOILMATH_LIBRARY
QMAKE_CXXFLAGS += -std=c++11
unix:!macx: QMAKE_RPATHDIR += $$PWD/../../../build/install/

@
CONFIG(release, debug|release):DEFINES += QT_NO_DEBUG_OUTPUT
@

SOURCES += \
    NN.cpp \
    GA.cpp \
    FFT.cpp

HEADERS += \
    Stats.h \
    Sort.h \
    SoilMathTypes.h \
    SoilMath.h \
    NN.h \
    MathException.h \
    GA.h \
    FFT.h \
    CommonOperations.h \
    predict_t_archive.h \
    Mat_archive.h \
    psd.h

#opencv
LIBS += -L/usr/local/lib -lopencv_core -lopencv_highgui
INCLUDEPATH += /usr/local/include/opencv
INCLUDEPATH += /usr/local/include

#boost
DEFINES += BOOST_ALL_DYN_LINK
INCLUDEPATH += /usr/include/boost
LIBS += -L/usr/lib/x86_64-linux-gnu/ -lboost_serialization -
   lboost_iostreams

#Zlib
LIBS += -L/usr/local/lib -lz
INCLUDEPATH += /usr/local/include
```

```

unix {
    target.path = $PWD/../../../build/install
    INSTALLS += target
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \brief Collection of the public SoilMath headers
 * Commonpractice is to include this header when you want to add Soilmath
 * routines
 */
#pragma once

#include "Stats.h"
#include "Sort.h"
#include "FFT.h"
#include "NN.h"
#include "GA.h"
#include "CommonOperations.h"
#include "SoilMathTypes.h"
#include "psd.h"
#include "Mat_archive.h"
#include "predict_t_archive.h"
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#define COMMONOPERATIONS_VERSION 1

#include <algorithm>
#include <stdint.h>
#include <math.h>
#include <vector>

namespace SoilMath {
inline uint16_t MinNotZero(uint16_t a, uint16_t b) {
  if (a != 0 && b != 0) {
    return (a < b) ? a : b;
  } else {
    return (a > b) ? a : b;
```

```
  }
}

inline uint16_t Max(uint16_t a, uint16_t b) { return (a > b) ? a : b; }

inline uint16_t Max(uint16_t a, uint16_t b, uint16_t c, uint16_t d) {
  return (Max(a, b) > Max(c, d)) ? Max(a, b) : Max(c, d);
}

inline uint16_t Min(uint16_t a, uint16_t b) { return (a < b) ? a : b; }

inline uint16_t Min(uint16_t a, uint16_t b, uint16_t c, uint16_t d) {
  return (Min(a, b) > Min(c, d)) ? Min(a, b) : Min(c, d);
}

static inline double quick_pow10(int n) {
  static double pow10[19] = {1, 10, 100, 1000, 10000, 100000, 1000000, 10000000,
                             100000000, 1000000000, 10000000000, 100000000000,
                             1000000000000, 10000000000000, 100000000000000,
                             1000000000000000, 10000000000000000,
                             100000000000000000, 1000000000000000000};
  return pow10[(n >= 0) ? n : -n];
}


// Source:
// http://martin.ankerl.com/2012/01/25/optimized-approximative-pow-in-c-and-cpp/
static inline double fastPow(double a, double b) {
  union {
    double d;
    int x[2];
  } u = {a};
  u.x[1] = (int)(b * (u.x[1] - 1072632447) + 1072632447);
  u.x[0] = 0;
  return u.d;
}

static inline double quick_pow2(int n) {
  static double pow2[256] = {
      0,    1,    4,    9,    16,   25,   36,   49,   64,   81,
      100,  121,  144,  169,  196,  225,  256,  289,  324,  361,
      400,  441,  484,  529,  576,  625,  676,  729,  784,  841,
      900,  961,  1024, 1089, 1156, 1225, 1296, 1369,
```

```
            1444,  1521,
        1600,  1681,  1764,  1849,  1936,  2025,  2116,  2209,
            2304,  2401,
        2500,  2601,  2704,  2809,  2916,  3025,  3136,  3249,
            3364,  3481,
        3600,  3721,  3844,  3969,  4096,  4225,  4356,  4489,
            4624,  4761,
        4900,  5041,  5184,  5329,  5476,  5625,  5776,  5929,
            6084,  6241,
        6400,  6561,  6724,  6889,  7056,  7225,  7396,  7569,
            7744,  7921,
        8100,  8281,  8464,  8649,  8836,  9025,  9216,  9409,
            9604,  9801,
        10000, 10201, 10404, 10609, 10816, 11025, 11236,
          11449, 11664, 11881,
        12100, 12321, 12544, 12769, 12996, 13225, 13456,
          13689, 13924, 14161,
        14400, 14641, 14884, 15129, 15376, 15625, 15876,
          16129, 16384, 16641,
        16900, 17161, 17424, 17689, 17956, 18225, 18496,
          18769, 19044, 19321,
        19600, 19881, 20164, 20449, 20736, 21025, 21316,
          21609, 21904, 22201,
        22500, 22801, 23104, 23409, 23716, 24025, 24336,
          24649, 24964, 25281,
        25600, 25921, 26244, 26569, 26896, 27225, 27556,
          27889, 28224, 28561,
        28900, 29241, 29584, 29929, 30276, 30625, 30976,
          31329, 31684, 32041,
        32400, 32761, 33124, 33489, 33856, 34225, 34596,
          34969, 35344, 35721,
        36100, 36481, 36864, 37249, 37636, 38025, 38416,
          38809, 39204, 39601,
        40000, 40401, 40804, 41209, 41616, 42025, 42436,
          42849, 43264, 43681,
        44100, 44521, 44944, 45369, 45796, 46225, 46656,
          47089, 47524, 47961,
        48400, 48841, 49284, 49729, 50176, 50625, 51076,
          51529, 51984, 52441,
        52900, 53361, 53824, 54289, 54756, 55225, 55696,
          56169, 56644, 57121,
        57600, 58081, 58564, 59049, 59536, 60025, 60516,
          61009, 61504, 62001,
        62500, 63001, 63504, 64009, 64516, 65025};
  return pow2[(n >= 0) ? n : -n];
}

static inline long float2intRound(double d) {
  d += 6755399441055744.0;
  return reinterpret_cast<int &>(d);
}

/*!
 * \brief calcVolume according to ISO 9276-6
 * \param A
 * \return
```

```
 */
static inline float calcVolume(float A) {
  return (pow(A, 1.5)) / 10.6347f;
}

static inline std::vector<float> makeOutput(uint8_t value, uint32_t noNeurons) {
  std::vector<float> retVal(noNeurons, -1);
  retVal[value - 1] = 1;
  return retVal;
}

/*!
 * \brief calcDiameter according to ISO 9276-6
 * \param A
 * \return
 */
static inline float calcDiameter(float A) {
  //return sqrt((4 * A) / M_PI);
  return 1.1283791670955 * sqrt(A);
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */
#pragma once

#define GENE_MAX 32  /**< maximum number of genes*/
#define CROSSOVER 16 /**< crossover location*/

#include <stdint.h>
#include <bitset>
#include <vector>
#include <complex>
#include <valarray>
#include <array>

typedef unsigned char uchar;   /**< unsigned char*/
typedef unsigned short ushort; /**< unsigned short*/
typedef unsigned int uint32_t;

typedef std::complex<double> Complex_t;       /**< complex vector of doubles*/
typedef std::vector<Complex_t> ComplexVect_t; /**< vector of Complex_t*/
typedef std::valarray<Complex_t> ComplexArray_t; /**< valarray of Complex_t*/
typedef std::vector<uint32_t> iContour_t;        /**< vector of uint32_t*/
typedef std::bitset<GENE_MAX> Genome_t; /**< Bitset
```

```
    represssenting a genome*/
typedef std::pair<std::bitset<CROSSOVER>, std::bitset<GENE_MAX - CROSSOVER>>
    SplitGenome_t; /**< a matted genome*/

typedef std::vector<float> Weight_t;      /**< a float vector */
typedef std::vector<Genome_t> GenVect_t; /**< a vector of genomes*/
typedef struct PopMemberStruct {
  Weight_t weights;         /**< the weights the core of a population member*/
  GenVect_t weightsGen;     /**< the weights as genomes*/
  float Calculated = 0.0; /**< the calculated value*/
  float Fitness = 0.0;      /**< the fitness of the population member*/
} PopMember_t;              /**< a population member*/
typedef std::vector<PopMember_t> Population_t; /**< Vector with PopMember_t*/
typedef std::pair<float, float>
    MinMaxWeight_t; /**< floating pair weight range*/

typedef struct Predict_struct {
  uint8_t Category = 1; /**< the category number */
  float RealValue = 1.;  /**< category number as float in order to estimate how
                           precise to outcome is*/
  float Accuracy = 1.;   /**< the accuracy of the category*/
  std::vector<float> OutputNeurons; /**< the output Neurons */
  bool ManualSet = true;
} Predict_t;                          /**< The prediction results*/
typedef Predict_t (*NNfunctionType)(
    ComplexVect_t, Weight_t, Weight_t, uint32_t, uint32_t,
    uint32_t); /**< The prediction function from the Neural Net*/

typedef std::vector<ComplexVect_t>
    InputLearnVector_t; /**< Vector of a vector with complex values*/
typedef std::vector<Predict_t> OutputLearnVector_t; /**< vector with results*/
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

// Source:
// http://stackoverflow.com/questions/16125574/how-to-serialize-opencv-mat-with-boost-xml-archive
```

```
#pragma once

#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/access.hpp>
#include <opencv/cv.h>
#include <opencv2/core.hpp>

namespace boost {
namespace serialization {
/*!
 * \brief serialize Serialize the openCV mat to disk
 */
template <class Archive>
inline void serialize(Archive &ar, cv::Mat &m, const unsigned int version __attribute__((unused))) {
  int cols = m.cols;
  int rows = m.rows;
  int elemSize = m.elemSize();
  int elemType = m.type();

  ar &cols;
  ar &rows;
  ar &elemSize;
  ar &elemType; // element type.

  if (m.type() != elemType || m.rows != rows || m.cols != cols) {
    m = cv::Mat(rows, cols, elemType, cv::Scalar(0));
  }

  size_t dataSize = cols * rows * elemSize;

  for (size_t dc = 0; dc < dataSize; dc++) {
    ar &m.data[dc];
  }
}
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

// Source:
// http://stackoverflow.com/questions/16125574/how-to-serialize-opencv-mat-with-boost-xml-archive
#pragma once

#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
```

```
#include <boost/serialization/access.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/serialization/complex.hpp>
#include "SoilMathTypes.h"

namespace boost {
namespace serialization {
/*!
 * \brief serialize Serialize the openCV mat to disk
 */
template <class Archive>
inline void serialize(Archive &ar, Predict_t &P, const unsigned int version __attribute__((unused))) {
  ar &P.Accuracy;
  ar &P.Category;
  ar &P.OutputNeurons;
  ar &P.RealValue;
}
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#define EXCEPTION_MATH "Math Exception!"
#define EXCEPTION_MATH_NR 0
#define EXCEPTION_NO_CONTOUR_FOUND                                            \
  "No continuous contour found, or less then 8 pixels long!"
#define EXCEPTION_NO_CONTOUR_FOUND_NR 1
#define EXCEPTION_SIZE_OF_INPUT_NEURONS                                       \
  "Size of input unequal to input neurons exception!"
#define EXCEPTION_SIZE_OF_INPUT_NEURONS_NR 2
#define EXCEPTION_NEURAL_NET_NOT_STUDIED "Neural net didn't study exception!"
#define EXCEPTION_NEURAL_NET_NOT_STUDIED_NR 3
#define EXCEPTION_TYPE_NOT_SUPPORTED                                          \
  "Type not supported for operation exception!"
#define EXCEPTION_TYPE_NOT_SUPPORTED_NR 4

#pragma once
#include <exception>
#include <string>

namespace SoilMath {
namespace Exception {
class MathException : public std::exception {
public:
```

```
  MathException(std::string m = EXCEPTION_MATH, int n = EXCEPTION_MATH_NR)
      : msg(m), nr(n){};
  ~MathException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
  const int *id() const _GLIBCXX_USE_NOEXCEPT { return &nr; }

private:
  std::string msg;
  int nr;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#include <stdint.h>

namespace SoilMath {
/*!
 * \brief The Sort template class
 */
class Sort {
public:
  Sort() {}
  ~Sort() {}

  /*!
   * \brief QuickSort a static sort a Type T array with i values
   * \details Usage: QuickSort<type>(*type , i)
   * \param arr an array of Type T
   * \param i the number of elements
   */
  template <typename T> static void QuickSort(T *arr, int i) {
    if (i < 2)
      return;

    T p = arr[i / 2];
    T *l = arr;
    T *r = arr + i - 1;
    while (l <= r) {
      if (*l < p) {
        l++;
      } else if (*r > p) {
```

```
      r--;
    } else {
      T t = *l;
      *l = *r;
      *r = t;
      l++;
      r--;
    }
  }
  Sort::QuickSort<T>(arr, r - arr + 1);
  Sort::QuickSort<T>(l, arr + i - l);
}

/*!
 * \brief QuickSort a static sort a Type T array with i values where the key
 * are also changed accordingly
 * \details Usage: QuickSort<type>(*type *type , i)
 * \param arr an array of Type T
 * \param key an array of 0..i-1 representing the index
 * \param i the number of elements
 */
template <typename T> static void QuickSort(T *arr, T *key, int i) {
  if (i < 2)
    return;

  T p = arr[i / 2];

  T *l = arr;
  T *r = arr + i - 1;

  T *lkey = key;
  T *rkey = key + i - 1;

  while (l <= r) {
    if (*l < p) {
      l++;
      lkey++;
    } else if (*r > p) {
      r--;
      rkey--;
    } else {
     if (*l != *r) {
        T t = *l;
        *l = *r;
        *r = t;

        T tkey = *lkey;
        *lkey = *rkey;
        *rkey = tkey;
      }

      l++;
      r--;

```

```
        lkey++;
        rkey--;
      }
    }
    Sort::QuickSort<T>(arr, key, r - arr + 1);
    Sort::QuickSort<T>(l, lkey, arr + i - 1);
  }
};
}
```

D. Hardware Library

D.0.1 Microscope Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class Microscope
Interaction with the USB 5 MP microscope
*/

#pragma once

#include <stdint.h>
#include <vector>
#include <string>
#include <utility>
#include <algorithm>

#include <sys/stat.h>
#include <sys/utsname.h>
#include <sys/ioctl.h>
#include <fstream>
#include <fcntl.h>

#include <linux/videodev2.h>
#include <linux/v4l2-controls.h>
#include <linux/v4l2-common.h>

```

```
#include <boost/filesystem.hpp>
#include <boost/regex.hpp>

#include <opencv2/photo.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/core.hpp>

#include "MicroscopeNotFoundException.h"
#include "CouldNotGrabImageException.h"

namespace Hardware {
class Microscope {
public:
  enum Arch { ARM, X64 };

  enum PixelFormat { YUYV, MJPG };

  struct Resolution_t {
    uint16_t Width = 2048;
    uint16_t Height = 1536;
    PixelFormat format = PixelFormat::MJPG;
    std::string to_string() {
      std::string retVal = std::to_string(Width);
      retVal.append(" x ");
      retVal.append(std::to_string(Height));
      if (format == PixelFormat::MJPG) {
          retVal.append(" - MJPG");
        }
      else {
          retVal.append(" - YUYV");
        }
      return retVal;
    }
    uint32_t ID;
  };

  struct Control_t {
    std::string name;
    int minimum;
    int maximum;
    int step;
    int default_value;
    int current_value;
    uint32_t ID = V4L2_CID_BASE;
    bool operator==(Control_t &rhs) {
      if (this->name.compare(rhs.name) == 0) {
        return true;
      } else {
        return false;
      }
    }
    bool operator!=(Control_t &rhs) {
      if (this->name.compare(rhs.name) != 0) {
        return true;
      } else {
```

```
        return false;
      }
    }
  };

  typedef std::vector<Control_t> Controls_t;

  struct Cam_t {
    std::string Name;
    std::string devString;
    uint32_t ID;
    std::vector<Resolution_t> Resolutions;
    uint32_t delaytrigger = 1;
    Resolution_t *SelectedResolution = nullptr;
    Controls_t Controls;
    int fd;
    bool operator==(Cam_t const &rhs) {
      if (this->ID == rhs.ID || this->Name == rhs.Name) {
        return true;
      } else {
        return false;
      }
    }
    bool operator!=(Cam_t const &rhs) {
      if (this->ID != rhs.ID && this->Name != rhs.Name) {
        return true;
      } else {
        return false;
      }
    }
  };

  std::vector<Cam_t> AvailableCams;
  Cam_t *SelectedCam = nullptr;
  Arch RunEnv;

  Microscope();
  Microscope(const Microscope &rhs);

  ~Microscope();

  Microscope operator=(Microscope const &rhs);

  bool IsOpened();
  bool openCam(Cam_t *cam);
  bool openCam(int &cam);
  bool openCam(std::string &cam);

  bool closeCam(Cam_t *cam);

  void GetFrame(cv::Mat &dst);
  void GetHDRFrame(cv::Mat &dst, uint32_t noframes = 3);

  Control_t *GetControl(const std::string name);
  void SetControl(Control_t *control);

```

```
  Cam_t *FindCam(std::string cam);
  Cam_t *FindCam(int cam);

private:
  cv::VideoCapture *cap = nullptr;

  std::vector<cv::Mat> HDRframes;

  std::vector<Cam_t> GetAvailableCams();
  Arch GetCurrentArchitecture();
  int fd;
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "Microscope.h"

namespace Hardware {

Microscope::Microscope() {
  RunEnv = GetCurrentArchitecture();
  AvailableCams = GetAvailableCams();
  for_each(AvailableCams.begin(), AvailableCams.end(), [](
      Cam_t &C) {
    C.SelectedResolution = &C.Resolutions[C.Resolutions.size
        () - 1];
  });
}

Microscope::Microscope(const Microscope &rhs) {
  std::copy(rhs.AvailableCams.begin(), rhs.AvailableCams.end
      (),
            this->AvailableCams.begin());
  this->RunEnv = rhs.RunEnv;
  this->SelectedCam = rhs.SelectedCam;
  this->cap = rhs.cap;
  this->fd = rhs.fd;
  this->HDRframes = rhs.HDRframes;
}

Microscope::~Microscope() { delete cap; }

Microscope::Arch Microscope::GetCurrentArchitecture() {
  struct utsname unameData;
  Arch retVal;
  uname(&unameData);
  std::string archString = static_cast<std::string>(
      unameData.machine);
```

```
  if (archString.find("armv7l") != string::npos) {
    retVal = Arch::ARM;
  } else {
    retVal = Arch::X64;
  }
  return retVal;
}

std::vector<Microscope::Cam_t> Microscope::GetAvailableCams() {
  const string path_ss = "/sys/class/video4linux";
  const string path_ss_dev = "/dev/video";
  std::vector<Cam_t> retVal;
  struct v4l2_queryctrl queryctrl;
  struct v4l2_control controlctrl;

  // Check if there're videodevices installed
  // Itterate through the cams
  for (boost::filesystem::directory_iterator itr(path_ss);
       itr != boost::filesystem::directory_iterator(); ++itr) {
    string videoln = itr->path().string();
    videoln.append("/name");
    if (boost::filesystem::exists(videoln)) {
      Cam_t currentCam;
      std::ifstream camName;
      camName.open(videoln);
      std::getline(camName, currentCam.Name);
      camName.close();
      currentCam.ID =
          std::atoi(itr->path().string().substr(28, std::string::npos).c_str());

      // Open Cam
      currentCam.devString = path_ss_dev + std::to_string(currentCam.ID);
      if ((currentCam.fd = open(currentCam.devString.c_str(), O_RDWR)) == -1) {
        throw Exception::MicroscopeException(EXCEPTION_NOCAMS,
                                                                  EXCEPTION_NOCAMS_NR);
      }

      // Get controls
      memset(&queryctrl, 0, sizeof(queryctrl));
      memset(&controlctrl, 0, sizeof(controlctrl));
      for (queryctrl.id = V4L2_CID_BASE; queryctrl.id < V4L2_CID_LASTP1;
           queryctrl.id++) {

        if (ioctl(currentCam.fd, VIDIOC_QUERYCTRL, &queryctrl) == 0) {
          if (!(queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)) {
            Control_t currentControl;
```

```
          currentControl.ID = queryctrl.id;
          currentControl.name = (char *)queryctrl.name;
          currentControl.minimum = queryctrl.minimum;
          currentControl.maximum = queryctrl.maximum;
          currentControl.default_value = queryctrl.default_value;
          currentControl.step = queryctrl.step;
          controlctrl.id = queryctrl.id;
          if (ioctl(currentCam.fd, VIDIOC_G_CTRL, &controlctrl) == 0) {
            currentControl.current_value = controlctrl.value;
          }
          currentCam.Controls.push_back(currentControl);
        }
      } else {
        if (errno == EINVAL)
          continue;
        throw Exception::MicroscopeException(EXCEPTION_QUERY,
                                             EXCEPTION_QUERY_NR);
      }
    }

    // Get image formats
    struct v4l2_format format;
    memset(&format, 0, sizeof(format));

    uint32_t width[5] = {640, 800, 1280, 1600, 2048};
    uint32_t height[6] = {480, 600, 960, 1200, 1536};

    uint32_t ResolutionID = 0;

    // YUYV
    for (uint32_t i = 0; i < 5; i++) {
      format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
      format.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
      format.fmt.pix.width = width[i];
      format.fmt.pix.height = height[i];
      int ret = ioctl(currentCam.fd, VIDIOC_S_FMT, &format);
      if (ret != -1 && format.fmt.pix.height == height[i] &&
          format.fmt.pix.width == width[i]) {
        Resolution_t res;
        res.Width = format.fmt.pix.width;
        res.Height = height[i];
        res.ID = ResolutionID++;
        res.format = PixelFormat::YUYV;
        currentCam.Resolutions.push_back(res);
      }
    }

    // MJPEG
    for (uint32_t i = 0; i < 5; i++) {
```

```
        format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        format.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
        format.fmt.pix.width = width[i];
        format.fmt.pix.height = height[i];
        int ret = ioctl(currentCam.fd, VIDIOC_S_FMT, &format);
        if (ret != -1 && format.fmt.pix.height == height[i] &&
            format.fmt.pix.width == width[i]) {
          Resolution_t res;
          res.Width = format.fmt.pix.width;
          res.Height = format.fmt.pix.height;
          res.ID = ResolutionID++;
          res.format = PixelFormat::MJPG;
          currentCam.Resolutions.push_back(res);
        }
      }

      close(currentCam.fd);
      retVal.push_back(currentCam);
    }
  }

  for (uint32_t i = 0; i < retVal.size(); i++) {
    if (retVal[i].Resolutions.size() == 0) {
      retVal.erase(retVal.begin() + i);
      i--;
    }
  }

  return retVal;
}

bool Microscope::IsOpened() {
  if (cap == nullptr) {
    return false;
  } else {
    return cap->isOpened();
  }
}

bool Microscope::openCam(Cam_t *cam) {
  for (uint32_t i = 0; i < AvailableCams.size(); i++) {
    if (AvailableCams[i] == *cam) {
      closeCam(SelectedCam);
      SelectedCam = cam;
      cap = new cv::VideoCapture(SelectedCam->ID);
      if (!cap->isOpened()) {
        throw Exception::MicroscopeException(EXCEPTION_NOCAMS,
                                                                 EXCEPTION_NOCAMS_NR);
      }
      cap->set(CV_CAP_PROP_FRAME_WIDTH, SelectedCam->SelectedResolution->Width);
      cap->set(CV_CAP_PROP_FRAME_HEIGHT,
```

```
                  SelectedCam->SelectedResolution->Height);
      for (Controls_t::iterator it = SelectedCam->Controls.begin();
           it != SelectedCam->Controls.end(); ++it) {
        SetControl(&*it);
      }
      return true;
    }
  }
  return false;
}

bool Microscope::openCam(std::string &cam) { return openCam(FindCam(cam)); }

bool Microscope::openCam(int &cam) { return openCam(FindCam(cam)); }

Microscope::Cam_t *Microscope::FindCam(int cam) {
  for (uint32_t i = 0; i < AvailableCams.size(); i++) {
    if (cam == AvailableCams[i].ID) {
      return &AvailableCams[i];
    }
  }
  return nullptr;
}

Microscope::Cam_t *Microscope::FindCam(string cam) {
  for (uint32_t i = 0; i < AvailableCams.size(); i++) {
    if (cam.compare(AvailableCams[i].Name) == 0) {
      return &AvailableCams[i];
    }
  }
  return nullptr;
}

bool Microscope::closeCam(Cam_t *cam) {
  if (cap != nullptr) {
    if (cap->isOpened()) {
      cap->release();
    }
    delete cap;
    cap = nullptr;
  }
}

void Microscope::GetFrame(cv::Mat &dst) {
  openCam(SelectedCam);
  sleep(SelectedCam->delaytrigger);
  if (RunEnv == Arch::ARM) {
    for (uint32_t i = 0; i < 2; i++) {
      if (!cap->grab()) {
        throw Exception::CouldNotGrabImageException();
      }
      sleep(SelectedCam->delaytrigger);
    }
```

```
    cap->retrieve(dst);
  } else {
    for (uint32_t i = 0; i < 2; i++) {
      if (!cap->read(dst)) {
        throw Exception::CouldNotGrabImageException();
      }
    }
  }
}

void Microscope::GetHDRFrame(cv::Mat &dst, uint32_t noframes
    ) {
  // create the brightness steps
  Control_t *brightness = GetControl("Brightness");
  Control_t *contrast = GetControl("Contrast");

  uint32_t brightnessStep =
      (brightness->maximum - brightness->minimum) / noframes
          ;
  int8_t currentBrightness = brightness->current_value;
  int8_t currentContrast = contrast->current_value;
  contrast->current_value = contrast->maximum;

  cv::Mat currentImg;
  // take the shots at different brightness levels
  for (uint32_t i = 1; i <= noframes; i++) {
    brightness->current_value = brightness->minimum + (i *
        brightnessStep);
    GetFrame(currentImg);
    HDRframes.push_back(currentImg);
  }

  // Set the brightness and back to the previous used level
  brightness->current_value = currentBrightness;
  contrast->current_value = currentContrast;

  // Perform the exposure fusion
  cv::Mat fusion;
  cv::Ptr<cv::MergeMertens> merge_mertens = cv::
      createMergeMertens();
  merge_mertens->process(HDRframes, fusion);
  fusion *= 255;
  fusion.convertTo(dst, CV_8UC1);
}

Microscope::Control_t *Microscope::GetControl(const string
    name) {
  for (Controls_t::iterator it = SelectedCam->Controls.begin
      ();
       it != SelectedCam->Controls.end(); ++it) {
    if (name.compare(it->name) == 0) {
      return &*it;
    }
  }
  return nullptr;
}
```

```

void Microscope::SetControl(Control_t *control) {
  if ((SelectedCam->fd = open(SelectedCam->devString.c_str(), O_RDWR)) == -1) {
    throw Exception::MicroscopeException(EXCEPTION_NOCAMS, EXCEPTION_NOCAMS_NR);
  }

  struct v4l2_queryctrl queryctrl;
  struct v4l2_control controlctrl;

  memset(&queryctrl, 0, sizeof(queryctrl));
  queryctrl.id = control->ID;
  if (ioctl(SelectedCam->fd, VIDIOC_QUERYCTRL, &queryctrl) == -1) {
    if (errno != EINVAL) {
      close(SelectedCam->fd);
      throw Exception::MicroscopeException(EXCEPTION_QUERY, EXCEPTION_QUERY_NR);
    } else {
      close(SelectedCam->fd);
      throw Exception::MicroscopeException(EXCEPTION_CTRL_NOT_FOUND,
                                           EXCEPTION_CTRL_NOT_FOUND_NR);
    }
  } else if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED) {
    close(SelectedCam->fd);
    throw Exception::MicroscopeException(EXCEPTION_CTRL_NOT_FOUND,
                                         EXCEPTION_CTRL_NOT_FOUND_NR);
  } else {
    memset(&controlctrl, 0, sizeof(controlctrl));
    controlctrl.id = control->ID;
    controlctrl.value = control->current_value;

    if (ioctl(SelectedCam->fd, VIDIOC_S_CTRL, &controlctrl) == -1) {
      // Fails on auto white balance
      // throw Exception::MicroscopeException(EXCEPTION_CTRL_VALUE,
      //                                      EXCEPTION_CTRL_VALUE_NR);
    }
  }
  close(SelectedCam->fd);
}
}
```

D.0.2 Beaglebone Black Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class BBB
The core BeagleBone Black class used for all hardware related classes.
Consisting of universal used method, functions and variables. File operations,
polling and threading
*/

#pragma once

#define SLOTS \
  "/sys/devices/platform/bone_capemgr/slots" /*!< Beaglebone capemanager slots file*/

#include <fstream>
#include <sstream>
#include <string>
#include <sys/stat.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/epoll.h>
#include <fcntl.h>
#include <regex>
#include <stdexcept>

#include "GPIOReadException.h"
#include "FailedToCreateGPIOPollingThreadException.h"
#include "ValueOutOfBoundsException.h"

using namespace std;

namespace Hardware {
typedef int (*CallbackType)(
    int); /*!< CallbackType used to pass a function to a thread*/

class BBB {
public:
  int debounceTime; /*!< debounce time for a button in milliseconds*/

  BBB();
  ~BBB();
```

```

protected:
  bool threadRunning;              /*!< used to stop the thread*/
  pthread_t thread;                /*!< The thread*/
  CallbackType callbackFunction; /*!< the callbakcfunction*/

  bool DirectoryExist(const string &path);
  bool CapeLoaded(const string &shield);

  string Read(const string &path);
  void Write(const string &path, const string &value);

  /*! Converts a number to a string
  \param Number as typename
  \returns the number as a string
  */
  template <typename T> string NumberToString(T Number) {
    ostringstream ss;
    ss << Number;
    return ss.str();
  };

  /*! Converts a string to a number
  \param Text the string that needs to be converted
  \return the number as typename
  */
  template <typename T> T StringToNumber(string Text) {
    stringstream ss(Text);
    T result;
    return ss >> result ? result : 0;
  };
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "BBB.h"

namespace Hardware {
/*! Constructor*/
BBB::BBB() {
  threadRunning = false;
  callbackFunction = NULL;
  debounceTime = 0;
  thread = (pthread_t)NULL;
}

/*! De-constructor*/
```

```
BBB::~BBB() {}

/*! Reads the first line from a file
\param path constant string pointing towards the file
\returns this first line
*/
string BBB::Read(const string &path) {
  ifstream fs;
  fs.open(path.c_str());
  if (!fs.is_open()) {
    throw Exception::GPIOReadException(("Can't open: " +
        path).c_str());
  }
  string input;
  getline(fs, input);
  fs.close();
  return input;
}

/*! Writes a value to a file
\param path a constant string pointing towards the file
\param value a constant string which should be written in
    the file
*/
void BBB::Write(const string &path, const string &value) {
  ofstream fs;
  fs.open(path.c_str());
  if (!fs.is_open()) {
    throw Exception::GPIOReadException(("Can't open: " +
        path).c_str());
  }
  fs << value;
  fs.close();
}

/*! Checks if a directory exist
\returns true if the directory exists and false if not
*/
bool BBB::DirectoryExist(const string &path) {
  struct stat st;
  if (stat((char *)path.c_str(), &st) != 0) {
    return false;
  }
  return true;
}

/*! Checks if a cape is loaded in the file /sys/devices/
    bone_capemgr.9/slots
\param shield a const search string which is a (part) of the
    shield name
\return true if the search string is found otherwise false
*/
bool BBB::CapeLoaded(const string &shield) {
  bool shieldFound = false;

  ifstream fs;
```

```
  fs.open(SLOTS);
  if (!fs.is_open()) {
    throw Exception::GPIOReadException("Can't open SLOTS");
  }

  string line;
  while (getline(fs, line)) {
    if (line.find(shield) != string::npos) {
      shieldFound = true;
      break;
    }
  }
  fs.close();
  return shieldFound;
}
}
```

D.0.3 GPIO Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
     strictly prohibited
 * and only allowed with the written consent of the author (
     Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 * This code is based upon:
 * Derek Molloy, "Exploring BeagleBone: Tools and Techniques
      for Building
 * with Embedded Linux", Wiley, 2014, ISBN:9781118935125.
 * See: www.exploringbeaglebone.com
 */

#pragma once
#include "BBB.h"

#define EXPORT_PIN "/sys/class/gpio/export"
#define UNEXPORT_PIN "/sys/class/gpio/unexport"
#define GPIOS "/sys/class/gpio/gpio"
#define DIRECTION "/direction"
#define VALUE "/value"
#define EDGE "/edge"

using namespace std;

namespace Hardware {
class GPIO : public BBB {
public:
  enum Direction { Input, Output };
  enum Value { Low = 0, High = 1 };
  enum Edge { None, Rising, Falling, Both };

  int number; // Number of the pin

  int WaitForEdge();
  int WaitForEdge(CallbackType callback);
  void WaitForEdgeCancel() { this->threadRunning = false; }

  Value GetValue();
  void SetValue(Value value);

  Direction GetDirection();
  void SetDirection(Direction direction);

  Edge GetEdge();
  void SetEdge(Edge edge);

  GPIO(int number);
  ~GPIO();

private:
  string gpiopath;
  Direction direction;
```

```
  Edge edge;
  friend void *threadedPollGPIO(void *value);

  bool isExported(int number, Direction &dir, Edge &edge);
  bool ExportPin(int number);
  bool UnexportPin(int number);

  Direction ReadsDirection(const string &gpiopath);
  void WritesDirection(const string &gpiopath, Direction
      direction);

  Edge ReadsEdge(const string &gpiopath);
  void WritesEdge(const string &gpiopath, Edge edge);

  Value ReadsValue(const string &gpiopath);
  void WritesValue(const string &gpiopath, Value value);
};

void *threadedPollGPIO(void *value);
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
     strictly prohibited
 * and only allowed with the written consent of the author (
     Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "GPIO.h"

namespace Hardware {
GPIO::GPIO(int number) {

  this->number = number;
  gpiopath = GPIOS + NumberToString<int>(number);

  if (!isExported(number, direction, edge)) {
    ExportPin(number);
    direction = ReadsDirection(gpiopath);
    edge = ReadsEdge(gpiopath);
  }
  usleep(250000);
}

GPIO::~GPIO() { UnexportPin(number); }

int GPIO::WaitForEdge(CallbackType callback) {
  threadRunning = true;
  callbackFunction = callback;
  if (pthread_create(&this->thread, NULL, &threadedPollGPIO,
                     static_cast<void *>(this))) {
    threadRunning = false;
    throw Exception::
        FailedToCreateGPIOPollingThreadException();
```

```
  }
  return 0;
}

int GPIO::WaitForEdge() {
  if (direction == Output) {
    SetDirection(Input);
  }
  int fd, i, epollfd, count = 0;
  struct epoll_event ev;
  epollfd = epoll_create(1);
  if (epollfd == -1) {
    throw Exception::
        FailedToCreateGPIOPollingThreadException(
         "GPIO: Failed to create epollfd!");
  }
  if ((fd = open((gpiopath + VALUE).c_str(), O_RDONLY |
      O_NONBLOCK)) == -1) {
    throw Exception::GPIOReadException();
  }

  // read operation | edge triggered | urgent data
  ev.events = EPOLLIN | EPOLLET | EPOLLPRI;
  ev.data.fd = fd;

  if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
    throw Exception::
        FailedToCreateGPIOPollingThreadException(
         "GPIO: Failed to add control interface!");
  }

  while (count <= 1) {
    i = epoll_wait(epollfd, &ev, 1, -1);
    if (i == -1) {
      close(fd);
      return -1;
    } else {
      count++;
    }
  }
  close(fd);
  return 0;
}

GPIO::Value GPIO::GetValue() { return ReadsValue(gpiopath);
    }
void GPIO::SetValue(GPIO::Value value) { WritesValue(
    gpiopath, value); }

GPIO::Direction GPIO::GetDirection() { return direction; }
void GPIO::SetDirection(Direction direction) {
  this->direction = direction;
  WritesDirection(gpiopath, direction);
}

GPIO::Edge GPIO::GetEdge() { return edge; }
```

```
void GPIO::SetEdge(Edge edge) {
  this->edge = edge;
  WritesEdge(gpiopath, edge);
}

bool GPIO::isExported(int number __attribute__((unused)),
    Direction &dir,
                            Edge &edge) {
  // Checks if directory exist and therefore is exported
  if (!DirectoryExist(gpiopath)) {
    return false;
  }

  // Reads the data associated with the pin
  dir = ReadsDirection(gpiopath);
  edge = ReadsEdge(gpiopath);
  return true;
}

bool GPIO::ExportPin(int number) {
  switch (number) {
  case 7:
    system("config-pin P9.42 gpio");
    break;
  case 116:
    system("config-pin P9.91 gpio");
    break;
  case 112:
    system("config-pin P9.30 gpio");
    break;
  case 115:
    system("config-pin P9.27 gpio");
    break;
  case 14:
    system("config-pin P9.26 gpio");
    break;
  case 15:
    system("config-pin P9.24 gpio");
    break;
  case 49:
    system("config-pin P9.23 gpio");
    break;
  case 2:
    system("config-pin P9.22 gpio");
    break;
  case 3:
    system("config-pin P9.21 gpio");
    break;
  case 4:
    system("config-pin P9.18 gpio");
    break;
  case 5:
    system("config-pin P9.17 gpio");
    break;
  case 51:
    system("config-pin P9.16 gpio");
```

```
      break;
    case 48:
      system("config-pin P9.15 gpio");
      break;
    case 50:
      system("config-pin P9.14 gpio");
      break;
    case 31:
      system("config-pin P9.13 gpio");
      break;
    case 60:
      system("config-pin P9.12 gpio");
      break;
    case 30:
      system("config-pin P9.11 gpio");
      break;
    case 61:
      system("config-pin P8.26 gpio");
      break;
    case 22:
      system("config-pin P8.19 gpio");
      break;
    case 65:
      system("config-pin P8.18 gpio");
      break;
    case 27:
      system("config-pin P8.17 gpio");
      break;
    case 46:
      system("config-pin P8.16 gpio");
      break;
    case 47:
      system("config-pin P8.15 gpio");
      break;
    case 26:
      system("config-pin P8.14 gpio");
      break;
    case 23:
      system("config-pin P8.13 gpio");
      break;
    case 44:
      system("config-pin P8.12 gpio");
      break;
    case 45:
      system("config-pin P8.11 gpio");
      break;
    case 68:
      system("config-pin P8.10 gpio");
      break;
    case 69:
      system("config-pin P8.09 gpio");
      break;
    case 67:
      system("config-pin P8.08 gpio");
      break;
    case 66:
```

```
    system("config-pin P8.07 gpio");
    break;
  }
  usleep(250000);
}

bool GPIO::UnexportPin(int number) {
  //Write(UNEXPORT_PIN, NumberToString<int>(number));
}

GPIO::Direction GPIO::ReadsDirection(const string &gpiopath)
    {
  if (Read(gpiopath + DIRECTION) == "in") {
    return Input;
  } else {
    return Output;
  }
}

void GPIO::WritesDirection(const string &gpiopath, Direction
    direction) {
  switch (direction) {
  case Hardware::GPIO::Input:
    Write((gpiopath + DIRECTION), "in");
    break;
  case Hardware::GPIO::Output:
    Write((gpiopath + DIRECTION), "out");
    break;
  }
}

GPIO::Edge GPIO::ReadsEdge(const string &gpiopath) {
  string reader = Read(gpiopath + EDGE);
  if (reader == "none") {
    return None;
  } else if (reader == "rising") {
    return Rising;
  } else if (reader == "falling") {
    return Falling;
  } else {
    return Both;
  }
}

void GPIO::WritesEdge(const string &gpiopath, Edge edge) {
  switch (edge) {
  case Hardware::GPIO::None:
    Write((gpiopath + EDGE), "none");
    break;
  case Hardware::GPIO::Rising:
    Write((gpiopath + EDGE), "rising");
    break;
  case Hardware::GPIO::Falling:
    Write((gpiopath + EDGE), "falling");
    break;
  case Hardware::GPIO::Both:
```

```
      Write((gpiopath + EDGE), "both");
      break;
    default:
      break;
    }
}

GPIO::Value GPIO::ReadsValue(const string &gpiopath) {
    string path(gpiopath + VALUE);
    int res = StringToNumber<int>(Read(path));
    return (Value)res;
}

void GPIO::WritesValue(const string &gpiopath, Value value)
     {
    Write(gpiopath + VALUE, NumberToString<int>(value));
}

void *threadedPollGPIO(void *value) {
    GPIO *gpio = static_cast<GPIO *>(value);
    while (gpio->threadRunning) {
      gpio->callbackFunction(gpio->WaitForEdge());
      usleep(gpio->debounceTime * 1000);
    }
    return 0;
}
}
```

D.0.4 PWM Class

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#include "BBB.h"
#include <dirent.h>

#define OCP_PATH "/sys/class/pwm/"
#define PWM_CAPE "Override Board Name,00A0,Override Manuf,cape-universaln"

namespace Hardware {
class PWM : public BBB {
public:
  enum Pin // Four possible PWM pins
  { P8_13,
    P8_19,
    P9_14,
    P9_16 };
  enum Run // Signal generating
  { On = 1,
    Off = 0 };
  enum Polarity // Inverse duty polarity
  { Normal = 1,
    Inverted = 0 };

  Pin pin; // Current pin

  uint8_t GetPixelValue() { return pixelvalue; }
  void SetPixelValue(uint8_t value);

  float GetIntensity() { return intensity; };
  void SetIntensity(float value);

  int GetPeriod() { return period; };
  void SetPeriod(int value);

  int GetDuty() { return duty; };
  void SetDuty(int value);
  void SetIntensity();

  Run GetRun() { return run; };
  void SetRun(Run value);

  Polarity GetPolarity() { return polarity; };
  void SetPolarity(Polarity value);

  PWM(Pin pin);
```

```
  ~PWM();

private:
  int period;          // current period
  int duty;            // current duty
  float intensity;     // current intensity
  uint8_t pixelvalue; // current pixelvalue
  Run run;             // current run state
  Polarity polarity;  // current polaity

  string basepath;      // the basepath ocp
  string dutypath;      // base + duty path
  string periodpath;    // base + period path
  string runpath;       // base + run path
  string polaritypath; // base + polarity path

  void calcIntensity();
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "PWM.h"

namespace Hardware {
/// <summary>
/// Constructeur
/// </summary>
/// <param name="pin">Pin</param>
PWM::PWM(Pin pin) {
  this->pin = pin;

  // Check if PWM cape is loaded, if not load it
  if (!CapeLoaded(PWM_CAPE)) {
    Write(SLOTS, PWM_CAPE);
  }

  // Init the pin
  switch (pin) {
  case Hardware::PWM::P8_13:
    system("config-pin P8.13 pwm");
    basepath = OCP_PATH;
    basepath.append("pwmchip4/pwm1");
    break;
  case Hardware::PWM::P8_19:
    system("config-pin P8.19 pwm");
    basepath = OCP_PATH;
    basepath.append("pwmchip4/pwm0");
    break;
```

```
  case Hardware::PWM::P9_14:
    system("config-pin P9.14 pwm");
    basepath = OCP_PATH;
    basepath.append("pwmchip2/pwm0");
    break;
  case Hardware::PWM::P9_16:
    system("config-pin P9.16 pwm");
    basepath.append("pwmchip2/pwm1");
    break;
  }

  // Get the working paths
  dutypath = basepath + "/duty_cycle";
  periodpath = basepath + "/period";
  runpath = basepath + "/run";
  polaritypath = basepath + "/polarity";

  // Give Linux time to setup directory structure;
  usleep(250000);

  // Read current values
  period = StringToNumber<int>(Read(periodpath));
  duty = StringToNumber<int>(Read(dutypath));
  run = static_cast<Run>(StringToNumber<int>(Read(runpath)));
  polarity = static_cast<Polarity>(StringToNumber<int>(Read(polaritypath)));

  // calculate the current intensity
  calcIntensity();
}

PWM::~PWM() {}

/// <summary>
/// Calculate the current intensity
/// </summary>
void PWM::calcIntensity() {
  if (polarity == Normal) {
    if (duty == 0) {
      intensity = 0.0f;
    } else {
      intensity = (float)period / (float)duty;
    }
  } else {
    if (period == 0) {
      intensity = 0.0f;
    } else {
      intensity = (float)duty / (float)period;
    }
  }
}

/// <summary>
/// Set the intensity level as percentage
/// </summary>
```

```
/// <param name="value">floating value multipication factor </param>
void PWM::SetIntensity(float value) {
  if (polarity == Normal) {
    SetDuty(static_cast<int>((value * duty) + 0.5));
  } else {
    SetPeriod(static_cast<int>((value * period) + 0.5));
  }
}

/// <summary>
/// Set the output as a corresponding uint8_t value
/// </summary>
/// <param name="value">pixel value 0-255</param>
void PWM::SetPixelValue(uint8_t value) {
  if (period != 255) {
    SetPeriod(255);
  }
  SetDuty(255 - value);
  pixelvalue = value;
}

/// <summary>
/// Set the period of the signal
/// </summary>
/// <param name="value">period : int</param>
void PWM::SetPeriod(int value) {
  string valstr = NumberToString<int>(value);
  Write(periodpath, valstr);
  period = value;

  calcIntensity();
}

/// <summary>
/// Set the duty of the signal
/// </summary>
/// <param name="value">duty : int</param>
void PWM::SetDuty(int value) {
  string valstr = NumberToString<int>(value);
  Write(dutypath, valstr);
  duty = value;

  calcIntensity();
}

/// <summary>
/// Run the signal
/// </summary>
/// <param name="value">On or Off</param>
void PWM::SetRun(Run value) {
  int valInt = static_cast<int>(value);
  string valstr = NumberToString<int>(valInt);
  Write(runpath, valstr);
  run = value;
}
```

```

/// <summary>
/// Set the polarity
/// </summary>
/// <param name="value">Normal or Inverted signal</param>
void PWM::SetPolarity(Polarity value) {
  int valInt = static_cast<int>(value);
  string valstr = NumberToString<int>(valInt);
  Write(runpath, valstr);
  polarity = value;
}
}
```

D.0.5 ADC Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class ADC
Interaction with the beaglebone analogue pins
*/

#pragma once

#include "BBB.h"
#include "ADCReadException.h"

#define ADC0_PATH \
  "/sys/bus/iio/devices/iio:device0/in_voltage0_raw" /*!< path to analogue pin \
                                                        0*/
#define ADC1_PATH \
  "/sys/bus/iio/devices/iio:device0/in_voltage1_raw" /*!< path to analogue pin \
                                                        1*/
#define ADC2_PATH \
  "/sys/bus/iio/devices/iio:device0/in_voltage2_raw" /*!< path to analogue pin \
                                                        2*/
#define ADC3_PATH \
  "/sys/bus/iio/devices/iio:device0/in_voltage3_raw" /*!< path to analogue pin \
                                                        3*/
#define ADC4_PATH \
  "/sys/bus/iio/devices/iio:device0/in_voltage4_raw" /*!< path to analogue pin \
                                                        4*/
#define ADC5_PATH \
  "/sys/bus/iio/devices/iio:device0/in_voltage5_raw" /*!< path to analogue pin \
                                                        5*/
```

```
#define ADC6_PATH                                                          \
  "/sys/bus/iio/devices/iio:device0/in_voltage6_raw" /*!< path to analogue pin \
                                                                          6*/
#define ADC7_PATH                                                          \
  "/sys/bus/iio/devices/iio:device0/in_voltage7_raw" /*!< path to analogue pin \
                                                                          7*/

namespace Hardware {
class ADC : public BBB {
public:
  /*! Enumerator to indicate the analogue pin*/
  enum ADCPin {
    ADC0, /*!< AIN0 pin*/
    ADC1, /*!< AIN1 pin*/
    ADC2, /*!< AIN2 pin*/
    ADC3, /*!< AIN3 pin*/
    ADC4, /*!< AIN4 pin*/
    ADC5, /*!< AIN5 pin*/
    ADC6, /*!< AIN6 pin*/
    ADC7  /*!< AIN7 pin*/
  };

  ADCPin Pin; /*!< current pin*/

  ADC(ADCPin pin);
  ~ADC();

  int GetCurrentValue();
  float GetIntensity() { return Intensity; }
  int GetMinIntensity() { return MinIntensity; }
  int GetMaxIntensity() { return MaxIntensity; }

  void SetMinIntensity();
  void SetMaxIntensity();

  int WaitForValueChange();
  int WaitForValueChange(CallbackType callback);
  void WaitForValueChangeCancel() { this->threadRunning = false; }

private:
  string adcpath;    /*!< Path to analogue write file*/
  float Intensity;   /*!< Current intensity expressed as percentage*/
  int MinIntensity; /*!< Voltage level which represent 0 percentage*/
  int MaxIntensity; /*!< Voltage level which represent 100 percentage*/

```

```
  friend void *threadedPollADC(void *value); /*!< friend
     polling function*/
};

void *threadedPollADC(void *value);
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "ADC.h"

namespace Hardware {
/*! Constructor
\param pin and ADCPin type indicating which analogue pin to
   use
*/
ADC::ADC(ADCPin pin) {
  this->Pin = pin;
  switch (pin) {
  case Hardware::ADC::ADC0:
    adcpath = ADC0_PATH;
    break;
  case Hardware::ADC::ADC1:
    adcpath = ADC1_PATH;
    break;
  case Hardware::ADC::ADC2:
    adcpath = ADC2_PATH;
    break;
  case Hardware::ADC::ADC3:
    adcpath = ADC3_PATH;
    break;
  case Hardware::ADC::ADC4:
    adcpath = ADC4_PATH;
    break;
  case Hardware::ADC::ADC5:
    adcpath = ADC5_PATH;
    break;
  case Hardware::ADC::ADC6:
    adcpath = ADC6_PATH;
    break;
  case Hardware::ADC::ADC7:
    adcpath = ADC7_PATH;
    break;
  }

  MinIntensity = 0;
  MaxIntensity = 4096;
}

```

```
/*! De-constructor*/
ADC::~ADC() {}

/*! Reads the current voltage in the pin
\return an integer between 0 and 4096
*/
int ADC::GetCurrentValue() {
  int retVal = StringToNumber<int>(Read(adcpath));
  Intensity = (float)(retVal - MinIntensity) /
              (4096 - (MinIntensity + (4096 - MaxIntensity)));
  return retVal;
}

/*! Set the current voltage at the pin as the minimum voltage*/
void ADC::SetMinIntensity() {
  MinIntensity = StringToNumber<int>(Read(adcpath));
}

void ADC::SetMaxIntensity() {
  MaxIntensity = StringToNumber<int>(Read(adcpath));
}

/*! Threading enabled polling of the analogue pin
\param callback the function which should be called when polling indicates a
change CallbackType
\return 0
*/
int ADC::WaitForValueChange(CallbackType callback) {
  threadRunning = true;
  callbackFunction = callback;
  if (pthread_create(&thread, NULL, &threadedPollADC,
                     static_cast<void *>(this))) {
    threadRunning = false;
    throw Exception::FailedToCreateGPIOPollingThreadException();
  }
  return 0;
}

/*! Polling of the analogue pin
\return the current value
*/
int ADC::WaitForValueChange() {
  int fd, i, epollfd, count = 0;
  struct epoll_event ev;
  epollfd = epoll_create(1);
  if (epollfd == -1) {
    throw Exception::FailedToCreateGPIOPollingThreadException(
        "GPIO: Failed to create epollfd!");
  }
  if ((fd = open(adcpath.c_str(), O_RDONLY | O_NONBLOCK)) == -1) {
```

```
    throw Exception::ADCReadException();
  }
  ev.events = EPOLLIN;
  ev.data.fd = fd;

  if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
    throw Exception::FailedToCreateGPIOPollingThreadException(
        "ADC: Failed to add control interface!");
  }

  while (count <= 1) {
    i = epoll_wait(epollfd, &ev, 1, -1);
    if (i == -1) {
      close(fd);
      return -1;
    } else {
      count++;
    }
  }
  close(fd);
  return StringToNumber<int>(Read(adcpath));
}

/*! friendly function to start the threading*/
void *threadedPollADC(void *value) {
  ADC *adc = static_cast<ADC *>(value);
  while (adc->threadRunning) {
    adc->callbackFunction(adc->WaitForValueChange());
    usleep(200000);
  }
}
}
```

D.0.6 EC12P Class

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class EC12P
Interaction with the sparksfun RGB encoder
*/

#pragma once

#include "eqep.h"
#include "GPIO.h"
#include "FailedToCreateThreadException.h"

#include <pthread.h>

using namespace std;

namespace Hardware {
class EC12P {
public:
  EC12P();
  ~EC12P();

  /*! Enumerator indicating the color of the encoder shaft*/
  enum Color {
    Red,     /*!< Red*/
    Pink,    /*!< Pink*/
    Blue,    /*!< Blue*/
    SkyBlue, /*!< SkyBlue*/
    Green,   /*!< Green*/
    Yellow,  /*!< Yellow*/
    White,   /*!< White*/
    None     /*!< Off*/
  };

  void SetPixelColor(Color value);
  Color GetPixelColor() { return PixelColor; };

  void RainbowLoop(int sleepperiod);
  void StopRainbowLoop() { threadRunning = false; };

  eQEP Rotary{eQEP2, eQEP::eQEP_Mode_Absolute}; /*!< The encoder*/
  GPIO Button{68};                              /*!< The pushbutton*/

private:
  Color PixelColor; /*!< Current shaft color*/
```

```
  GPIO R{31}; /*!< Red LED*/
  GPIO B{48}; /*!< Blue LED*/
  GPIO G{51}; /*!< Green LED*/

  pthread_t thread;   /*!< the thread*/
  bool threadRunning; /*!< Bool used to stop the thread*/
  int sleepperiod;    /*!< Sleep period*/
  friend void *colorLoop(void *value);
};
void *colorLoop(void *value);
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "EC12P.h"

namespace Hardware {
/*! Constructor*/
EC12P::EC12P() {
  // Init Rotary button
  Button.SetDirection(GPIO::Input);
  Button.SetEdge(GPIO::Rising);

  // Init Encoder
  Rotary.set_period(100000000L);

  // Init Encoder color
  R.SetDirection(GPIO::Output);
  B.SetDirection(GPIO::Output);
  G.SetDirection(GPIO::Output);
  SetPixelColor(None);

  threadRunning = false;
}

/*! De-constructor*/
EC12P::~EC12P() {}

/*! Set the shaft color
\param value as Color enumerator
*/
void EC12P::SetPixelColor(Color value) {
  switch (value) {
  case Hardware::EC12P::Red:
    R.SetValue(GPIO::High);
    B.SetValue(GPIO::Low);
    G.SetValue(GPIO::Low);
    break;
```

```
  case Hardware::EC12P::Pink:
    R.SetValue(GPIO::High);
    B.SetValue(GPIO::High);
    G.SetValue(GPIO::Low);
    break;
  case Hardware::EC12P::Blue:
    R.SetValue(GPIO::Low);
    B.SetValue(GPIO::High);
    G.SetValue(GPIO::Low);
    break;
  case Hardware::EC12P::SkyBlue:
    R.SetValue(GPIO::Low);
    B.SetValue(GPIO::High);
    G.SetValue(GPIO::High);
    break;
  case Hardware::EC12P::Green:
    R.SetValue(GPIO::Low);
    B.SetValue(GPIO::Low);
    G.SetValue(GPIO::High);
    break;
  case Hardware::EC12P::Yellow:
    R.SetValue(GPIO::High);
    B.SetValue(GPIO::Low);
    G.SetValue(GPIO::High);
    break;
  case Hardware::EC12P::White:
    R.SetValue(GPIO::High);
    B.SetValue(GPIO::High);
    G.SetValue(GPIO::High);
    break;
  case Hardware::EC12P::None:
    R.SetValue(GPIO::Low);
    B.SetValue(GPIO::Low);
    G.SetValue(GPIO::Low);
    break;
  }
  PixelColor = value;
}

/*! Loops through all the colors except of as a thread  */
void EC12P::RainbowLoop(int sleepperiod) {
  this->sleepperiod = sleepperiod;
  this->threadRunning = true;
  if (pthread_create(&thread, NULL, colorLoop, this)) {
    throw Exception::FailedToCreateThreadException();
  }
}

/*! The thread function that runs trough all the colors*/
void *colorLoop(void *value) {
  int i = 0;
  EC12P *ec12p = static_cast<EC12P *>(value);
  EC12P::Color pcolor;
  while (ec12p->threadRunning) {
    pcolor = static_cast<EC12P::Color>(i);
    ec12p->SetPixelColor(pcolor);
```

```
      usleep(ec12p->sleepperiod);
      i++;
      if (i == 6) {
        i = 0;
      }
    }
    return ec12p;
  }
}
```

D.0.7 eQep Class

```
/*
* TI eQEP driver interface API
*
* Copyright (C) 2013 Nathaniel R. Lewis - http://nathanielrlewis.com/
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*
*
* This code is changed by Jelle Spijker (C) 2014.
* Introducing polling with threading.
*
*/

#pragma once

#include <iostream>
#include <stdint.h>
#include <string>
#include "BBB.h"

#define eQEP0 "/sys/devices/ocp.3/48300000.epwmss/48300180.eqep"
#define eQEP1 "/sys/devices/ocp.3/48302000.epwmss/48302180.eqep"
#define eQEP2 "/sys/devices/ocp.3/48304000.epwmss/48304180.eqep"

namespace Hardware {
// Class which defines an interface to my eQEP driver
class eQEP : public BBB {
  // Base path for the eQEP unit
  std::string path;

```

```
public:
  // Modes of operation for the eQEP hardware
  typedef enum {
    // Absolute positioning mode
    eQEP_Mode_Absolute = 0,

    // Relative positioning mode
    eQEP_Mode_Relative = 1,

    // Error flag
    eQEP_Mode_Error = 2,
  } eQEP_Mode;

  // Default constructor for the eQEP interface driver
  eQEP(std::string _path, eQEP_Mode _mode);

  // Reset the value of the encoder
  void set_position(int32_t position);

  // Get the position of the encoder, pass poll as true to poll the pin, whereas
  // passing false reads the immediate value
  int32_t get_position(bool _poll = true);

  // Thread of the poll
  int WaitForPositionChange(CallbackType callback);
  void WaitForPositionChangeCancel() { this->threadRunning = false; }

  // Set the polling period
  void set_period(long long unsigned int period);

  // Get the polling period of the encoder
  uint64_t get_period();

  // Set the mode of the eQEP hardware
  void set_mode(eQEP_Mode mode);

  // Get the mode of the eQEP hardware
  eQEP_Mode get_mode();

private:
  friend void *threadedPolleqep(void *value);
};

void *threadedPolleqep(void *value);
}
```

```
/*
* TI eQEP driver interface API
*
* Copyright (C) 2013 Nathaniel R. Lewis - http://nathanielrlewis.com/
*
* This program is free software; you can redistribute it and/or modify
```

```
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*
* This file is modified by Jelle Spijker 2014
* Added polling and threading capabilties
*
*/

// Pull in our eQEP driver definitions
#include "eqep.h"

// Language dependencies
#include <cstdint>
#include <cstdlib>
#include <cstdio>

// POSIX dependencies
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>
#include <sys/types.h>
#include <sys/stat.h>

namespace Hardware {
// Constructor for eQEP driver interface object
eQEP::eQEP(std::string _path, eQEP::eQEP_Mode _mode) : path(_path) {
  if (_path == eQEP0) {
    if (!CapeLoaded("bone_eqep0")) {
      Write(SLOTS, "bone_eqep0");
    }
  } else if (_path == eQEP1) {
    if (!CapeLoaded("bone_eqep1")) {
      Write(SLOTS, "bone_eqep1");
    }
  } else if (_path == eQEP2) {
    if (!CapeLoaded("bone_eqep2b")) {
      Write(SLOTS, "bone_eqep2b");
    }
```

```
  }

  // Set the mode of the hardware
  this->set_mode(_mode);

  // Reset the position
  this->set_position(0);
}

// Set the position of the eQEP hardware
void eQEP::set_position(int32_t position) {
  // Open the file representing the position
  FILE *fp = fopen((this->path + "/position").c_str(), "w");

  // Check that we opened the file correctly
  if (fp == NULL) {
    // Error, break out
    std::cerr << "[eQEP " << this->path << "] Unable to open position for write"
              << std::endl;
    return;
  }

  // Write the desired value to the file
  fprintf(fp, "%d\n", position);

  // Commit changes
  fclose(fp);
}

// Set the period of the eQEP hardware
void eQEP::set_period(long long unsigned int period) {
  // Open the file representing the position
  FILE *fp = fopen((this->path + "/period").c_str(), "w");

  // Check that we opened the file correctly
  if (fp == NULL) {
    // Error, break out
    std::cerr << "[eQEP " << this->path << "] Unable to open period for write"
              << std::endl;
    return;
  }

  // Write the desired value to the file
  fprintf(fp, "%llu\n", period);

  // Commit changes
  fclose(fp);
}

// Set the mode of the eQEP hardware
void eQEP::set_mode(eQEP::eQEP_Mode _mode) {
  // Open the file representing the position
  FILE *fp = fopen((this->path + "/mode").c_str(), "w");

```

```
  // Check that we opened the file correctly
  if (fp == NULL) {
    // Error, break out
    std::cerr << "[eQEP " << this->path << "] Unable to open mode for write"
              << std::endl;
    return;
  }

  // Write the desired value to the file
  fprintf(fp, "%u\n", _mode);

  // Commit changes
  fclose(fp);
}

int eQEP::WaitForPositionChange(CallbackType callback) {
  threadRunning = true;
  callbackFunction = callback;
  if (pthread_create(&this->thread, NULL, &threadedPolleqep,
                     static_cast<void *>(this))) {
    threadRunning = false;
    throw Exception::FailedToCreateGPIOPollingThreadException();
  }

  return 0;
}

// Get the position of the hardware
int32_t eQEP::get_position(bool _poll) {
  // Position temporary variable
  int32_t position;
  char dummy;
  struct pollfd ufd;

  // Do we want to poll?
  if (_poll) {
    // Open a connection to the attribute file.
    if ((ufd.fd = open((this->path + "/position").c_str(), O_RDWR)) < 0) {
      // Error, break out
      std::cerr << "[eQEP " << this->path
                << "] unable to open position for polling" << std::endl;
      return 0;
    }

    // Dummy read
    read(ufd.fd, &dummy, 1);

    // Poll the port
    ufd.events = (short)EPOLLET;
    if (poll(&ufd, 1, -1) < 0) {
      // Error, break out
      std::cerr << "[eQEP " << this->path << "] Error
```

```
                occurred whilst polling"
                    << std::endl;
            close(ufd.fd);
            return 0;
        }
    }

    // Read the position
    FILE *fp = fopen((this->path + "/position").c_str(), "r");

    // Check that we opened the file correctly
    if (fp == NULL) {
        // Error, break out
        std::cerr << "[eQEP " << this->path << "] Unable to open
            position for read"
                  << std::endl;
        close(ufd.fd);
        return 0;
    }

    // Write the desired value to the file
    fscanf(fp, "%d", &position);

    // Commit changes
    fclose(fp);

    // If we were polling, close the polling file
    if (_poll) {
        close(ufd.fd);
    }

    // Return the position
    return position;
}

// Get the period of the eQEP hardware
uint64_t eQEP::get_period() {
    // Open the file representing the position
    FILE *fp = fopen((this->path + "/period").c_str(), "r");

    // Check that we opened the file correctly
    if (fp == NULL) {
        // Error, break out
        std::cerr << "[eQEP " << this->path << "] Unable to open
            period for read"
                  << std::endl;
        return 0;
    }

    // Write the desired value to the file
    uint64_t period = 0;
    fscanf(fp, "%llu", &period);

    // Commit changes
    fclose(fp);

```

```
  // Return the period
  return period;
}

// Get the mode of the eQEP hardware
eQEP::eQEP_Mode eQEP::get_mode() {
  // Open the file representing the position
  FILE *fp = fopen((this->path + "/mode").c_str(), "r");

  // Check that we opened the file correctly
  if (fp == NULL) {
    // Error, break out
    std::cerr << "[eQEP " << this->path << "] Unable to open
        mode for read"
              << std::endl;
    return eQEP::eQEP_Mode_Error;
  }

  // Write the desired value to the file
  eQEP::eQEP_Mode mode;
  fscanf(fp, "%u", (unsigned int *)&mode);

  // Commit changes
  fclose(fp);

  // Return the mode
  return mode;
}

void *threadedPolleqep(void *value) {
  eQEP *eqep = static_cast<eQEP *>(value);
  while (eqep->threadRunning) {
    eqep->callbackFunction(eqep->get_position(true));
    usleep(eqep->debounceTime * 1000);
  }
  return 0;
}
}
```

D.0.8 SoilCape Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include "EC12P.h"
#include "GPIO.h"
#include "PWM.h"
#include "ADC.h"

namespace Hardware {
class SoilCape {
public:
  EC12P RGBEncoder;
  PWM MicroscopeLEDs{PWM::P9_14};
  ADC MicroscopeLDR{ADC::ADC0};

  SoilCape();
  ~SoilCape();
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "SoilCape.h"

namespace Hardware {
SoilCape::SoilCape() {}

SoilCape::~SoilCape() {}
}
```

D.0.9 USB Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
     strictly prohibited
 * and only allowed with the written consent of the author (
     Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>

#include <linux/usbdevice_fs.h>

namespace Hardware {
class USB {
public:
  USB();
  ~USB();
  void ResetUSB();
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
     strictly prohibited
 * and only allowed with the written consent of the author (
     Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "USB.h"

namespace Hardware {
USB::USB() {}

USB::~USB() {}

void USB::ResetUSB() {
  int fd, rc;

  fd = open("/dev/bus/usb/001/002", O_WRONLY);
  rc = ioctl(fd, USBDEVFS_RESET, 0);
  if (rc < 0) {
    throw - 1;
  }
  close(fd);
}
```

```
}
```

D.0.10 General project files

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-06-06T10:49:23
#
#-------------------------------------------------

QT       -= core gui

TARGET = SoilHardware
TEMPLATE = lib
VERSION = 0.9.1

DEFINES += SOILHARDWARE_LIBRARY
QMAKE_CXXFLAGS += -std=c++11 -pthread
unix:!macx: QMAKE_RPATHDIR += $$PWD/../../../build/install/

SOURCES += \
    USB.cpp \
    SoilCape.cpp \
    PWM.cpp \
    Microscope.cpp \
    GPIO.cpp \
    eqep.cpp \
    EC12P.cpp \
    BBB.cpp \
    ADC.cpp

HEADERS += \
    ValueOutOfBoundsException.h \
    USB.h \
    SoilCape.h \
    PWM.h \
    MicroscopeNotFoundException.h \
    Microscope.h \
    Hardware.h \
    GPIOReadException.h \
    GPIO.h \
    FailedToCreateThreadException.h \
    FailedToCreateGPIOPollingThreadException.h \
    eqep.h \
    EC12P.h \
    CouldNotGrabImageException.h \
    BBB.h \
    ADCReadException.h \
    ADC.h

#opencv
LIBS += -L/usr/local/lib -lopencv_core -lopencv_highgui -
   lopencv_photo -lopencv_imgcodecs -lopencv_videoio
INCLUDEPATH += /usr/local/include/opencv
INCLUDEPATH += /usr/local/include

#boost
DEFINES += BOOST_ALL_DYN_LINK
```

```
INCLUDEPATH += /usr/include/boost
LIBS += -L/usr/lib/x86_64-linux-gnu/ -lboost_filesystem -
    lboost_system

unix {
    target.path = $PWD/../../../build/install
    INSTALLS += target
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include "ADC.h"
#include "EC12P.h"
#include "eqep.h"
#include "GPIO.h"
#include "PWM.h"
#include "SoilCape.h"
#include "Microscope.h"
#include "CouldNotGrabImageException.h"
#include "ADCReadException.h"
#include "FailedToCreateGPIOPollingThreadException.h"
#include "FailedToCreateThreadException.h"
#include "GPIOReadException.h"
#include "MicroscopeNotFoundException.h"
#include "ValueOutOfBoundsException.h"
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class ValueOutOfBoundsException : public std::exception {
public:
```

```
  ValueOutOfBoundsException(string m = "Value out of bounds!") : msg(m){};
  ~ValueOutOfBoundsException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class ADCReadException : public std::exception {
public:
  ADCReadException(string m = "Can't read ADC data!") : msg(m){};
  ~ADCReadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <exception>
#include <string>
```

```

using namespace std;

namespace Hardware {
namespace Exception {
class FailedToCreateGPIOPollingThreadException : public std::exception {
public:
  FailedToCreateGPIOPollingThreadException(
      string m = "Failed to create GPIO polling thread!")
      : msg(m){};
  ~FailedToCreateGPIOPollingThreadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class FailedToCreateThreadException : public std::exception {
public:
  FailedToCreateThreadException(string m = "Couldn't create the thread!")
      : msg(m){};
  ~FailedToCreateThreadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */
#define EXCEPTION_OPENCAM "Exception could not open cam!"
#define EXCEPTION_OPENCAM_NR 0
#define EXCEPTION_NOCAMS "Exception no cam available!"
#define EXCEPTION_NOCAMS_NR 1
#define EXCEPTION_QUERY "Exception could not query device!"
#define EXCEPTION_QUERY_NR 3
#define EXCEPTION_FORMAT_RESOLUTION "Exception No supported formats and resolutions!"
#define EXCEPTION_FORMAT_RESOLUTION_NR 4
#define EXCEPTION_CTRL_NOT_FOUND "Control not found!"
#define EXCEPTION_CTRL_NOT_FOUND_NR 5
#define EXCEPTION_CTRL_VALUE "Control value not set!"
#define EXCEPTION_CTRL_VALUE_NR 5


#pragma once
#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class MicroscopeException : public std::exception {
public:
  MicroscopeException(string m = EXCEPTION_OPENCAM,
                      int n = EXCEPTION_OPENCAM_NR) : msg{m}, nr{n} {  }
  ~MicroscopeException() _GLIBCXX_USE_NOEXCEPT {}
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); }
  const int *id() const _GLIBCXX_USE_NOEXCEPT { return &nr; }

private:
  string msg;
  int nr;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
```

```
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class CouldNotGrabImageException : public std::exception {
public:
  CouldNotGrabImageException(string m = "Unable to grab the
      next image!")
      : msg(m){};
  ~CouldNotGrabImageException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return
      msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class GPIOReadException : public std::exception {
public:
  GPIOReadException(string m = "Can't read GPIO data!") :
      msg(m){};
  ~GPIOReadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return
      msg.c_str(); };

private:
  string msg;
};
}
```

```
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
     strictly prohibited
 * and only allowed with the written consent of the author (
     Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class GPIOReadException : public std::exception {
public:
  GPIOReadException(string m = "Can't read GPIO data!") :
      msg(m){};
  ~GPIOReadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return
      msg.c_str(); };

private:
  string msg;
};
}
}
```

E. Vision Library

E.0.1 Image processing Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
/*! Current class version*/
#define IMAGEPROCESSING_VERSION 1

/*! MACRO which sets the original pointer to the original image or a clone of
 * the earlier processed image */
#define CHAIN_PROCESS(chain, O, type)                                        \
  if (chain) {                                                               \
    TempImg = ProcessedImg.clone();                                          \
    O = (type *)TempImg.data;                                                \
  } else {                                                                   \
    O = (type *)OriginalImg.data;                                            \
  }
```

```
/*! MACRO which trows an EmtpyImageException if the matrix is empty*/
#define EMPTY_CHECK(img)                                                    \
  if (img.empty()) {                                                        \
    throw Exception::EmtpyImageException();                                 \
  }

#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

#include <stdint.h>
#include <cmath>
#include <vector>
#include <string>

#include <boost/signals2.hpp>
#include <boost/bind.hpp>

#include "EmptyImageException.h"
#include "WrongKernelSizeException.h"
#include "ChannelMismatchException.h"
#include "PixelValueOutOfBoundException.h"
#include "VisionDebug.h"

using namespace cv;

namespace Vision {
class ImageProcessing {
public:
  typedef boost::signals2::signal<void(float, std::string)> Progress_t;
  boost::signals2::connection
  connect_Progress(const Progress_t::slot_type &subscriber);

protected:
  uchar *GetNRow(int nData, int hKsize, int nCols, uint32_t totalRows);
  Mat TempImg;

  Progress_t prog_sig;

public:
  ImageProcessing();
  ~ImageProcessing();
  Mat OriginalImg;
  Mat ProcessedImg;

  static void getOrientented(Mat &BW, cv::Point_<double> &centroid,
                             double &theta, double &eccentricty);
```

```
static void RotateImg(Mat &src, Mat &dst, double &theta,
    cv::Point_<double> &Centroid, Rect &ROI);

double currentProg = 0.;
double ProgStep = 0.;

static std::vector<Mat> extractChannel(const Mat &src);

/*! Copy a matrix to a new matrix with a LUT mask
\param src the source image
\param *LUT type T with a LUT to filter out unwanted pixel
     values
\param cvType an in where you can pas CV_UC8C1 etc.
\return The new matrix
*/
template <typename T1, typename T2>
static Mat CopyMat(const Mat &src, T1 *LUT, int cvType) {
  Mat dst(src.size(), cvType);
  uint32_t nData = src.rows * src.cols * dst.step[1];
  if (cvType == 0 || cvType == 8 || cvType == 16 || cvType
       == 24) {
    for (uint32_t i = 0; i < nData; i += dst.step[1]) {
      dst.data[i] =
          static_cast<uint8_t>(LUT[*(T2 *)(src.data + (i *
              src.step[1]))]);
    }
  } else if (cvType == 1 || cvType == 9 || cvType == 17 ||
       cvType == 25) {
    for (uint32_t i = 0; i < nData; i += src.step[1]) {
      dst.data[i] =
          static_cast<int8_t>(LUT[*(T2 *)(src.data + (i *
             src.step[1]))]);
    }
  } else if (cvType == 2 || cvType == 10 || cvType == 18
      || cvType == 26) {
    for (uint32_t i = 0; i < nData; i += src.step[1]) {
      dst.data[i] =
          static_cast<uint16_t>(LUT[*(T2 *)(src.data + (i
             * src.step[1]))]);
    }
  } else if (cvType == 3 || cvType == 11 || cvType == 19
      || cvType == 27) {
    for (uint32_t i = 0; i < nData; i += src.step[1]) {
      dst.data[i] =
          static_cast<int16_t>(LUT[*(T2 *)(src.data + (i *
              src.step[1]))]);
    }
  } else if (cvType == 4 || cvType == 12 || cvType == 20
      || cvType == 28) {
    for (uint32_t i = 0; i < nData; i += src.step[1]) {
      dst.data[i] =
          static_cast<int32_t>(LUT[*(T2 *)(src.data + (i *
              src.step[1]))]);
    }
  }
  return dst;
```

```
}

/*! Copy a matrix to a new matrix with a mask
\param src the source image
\param *LUT type T with a LUT to filter out unwanted pixel values
\param cvType an in where you can pas CV_UC8C1 etc.
\return The new matrix
*/
template <typename T1>
static Mat CopyMat(const Mat &src, const Mat &mask, int cvType) {
  if (src.size != mask.size) {
    throw Exception::WrongKernelSizeException(
        "Mask not the same size as src Exception!");
  }
  if (mask.channels() != 1) {
    throw Exception::WrongKernelSizeException(
        "Mask has more then 1 channel Exception!");
  }
  Mat dst(src.size(), cvType);

  vector<Mat> exSrc = Vision::ImageProcessing::extractChannel(src);
  vector<Mat> exDst;

  int cvBaseType = cvType % 8;
  for_each(exSrc.begin(), exSrc.end(), [&](const Mat &sItem) {
    Mat dItem(src.size(), cvBaseType);
    std::transform(sItem.begin<T1>(), sItem.end<T1>(), mask.begin<T1>(),
                   dItem.begin<T1>(),
                   [](const T1 &s, const T1 &m) -> T1 { return s * m; });
    exDst.push_back(dItem);
  });

  merge(exDst, dst);

  return dst;
}

static cv::Mat WhiteBackground(const cv::Mat &src) {
  cv::Mat dst;
  cv::floodFill(src, dst, cv::Point(0, 0), cv::Scalar(255, 255, 255));
  return dst;
}

template <typename T1>
static void ShowDebugImg(cv::Mat img, T1 maxVal, std::string windowName,
                         bool scale = true) {
  if (img.rows > 0 && img.cols > 0) {
    cv::Mat tempImg(img.size(), img.type());
```

```
      if (scale == true) {
        std::vector<cv::Mat> exSrc = extractChannel(img);
        std::vector<cv::Mat> exDst;
        int cvBaseType = img.type() % 8;
        T1 MatMin = std::numeric_limits<T1>::max();
        T1 MatMax = std::numeric_limits<T1>::min();

        // Find the global max and min
        for_each(exSrc.begin(), exSrc.end(), [&](const Mat &sItem) {
          std::for_each(sItem.begin<T1>(), sItem.end<T1>(), [&](const T1 &s) {
            if (s > MatMax) {
              MatMax = s;
            } else if (s < MatMin) {
              MatMin = s;
            }
          });
        });

        int Range = MatMax - MatMin;
        if (Range < 1)
          Range = maxVal;

        // Convert the values
        for_each(exSrc.begin(), exSrc.end(), [&](const cv::Mat &sItem) {
          Mat dItem(img.size(), cvBaseType);
          std::transform(sItem.begin<T1>(), sItem.end<T1>(), dItem.begin<T1>(),
                         [&](const T1 &s) -> T1 {
                           return (T1)round(((s - MatMin) * maxVal) / Range);
                         });
          exDst.push_back(dItem);
        });

        merge(exDst, tempImg);
      } else {
        tempImg = img;
      }
      cv::namedWindow(windowName, cv::WINDOW_NORMAL);
      cv::imshow(windowName, tempImg);
      cv::waitKey(0);
      cv::destroyWindow(windowName);
    };
  };
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
```

```
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class ImageProcessing
\brief Core class of all the image classes
Core class of all the image classes with a few commonly shared functions and
variables
*/
#include "ImageProcessing.h"

namespace Vision {
/*! Constructor of the core class*/
ImageProcessing::ImageProcessing() {}

/*! De-constructor of the core class*/
ImageProcessing::~ImageProcessing() {}

/*! Create a LUT indicating which iteration variable i is the end of an row
\param nData an int indicating total pixels
\param hKsize int half the size of the kernel, if any. which acts as an offset
from the border pixels
\param nCols int number of columns in a row
\return array of uchars where a zero is a middle column and a 1 indicates an end
of an row minus the offset from half the kernel size
*/
uchar *ImageProcessing::GetNRow(int nData, int hKsize, int nCols,
                                uint32_t totalRows) {
  // Create LUT to determine when there is an new row
  uchar *nRow = new uchar[nData + 1]{};
  // int i = 0;
  int shift = nCols - hKsize - 1;
  for (uint32_t i = 0; i < totalRows; i++) {
    nRow[(i * nCols) + shift] = 1;
  }
  return nRow;
}

std::vector<Mat> ImageProcessing::extractChannel(const Mat &src) {
  vector<Mat> chans;
  split(src, chans);
  return chans;
}

void ImageProcessing::getOrientented(cv::Mat &BW, cv::Point_<double> &centroid,
                                     double &theta, double &eccentricty) {
  cv::Moments Mu = cv::moments(BW, true);

  centroid.x = Mu.m10 / Mu.m00;
```

```
  centroid.y = Mu.m01 / Mu.m00;

  theta = 0;
  double muPrime20 = (Mu.m20 / Mu.m00) - pow(centroid.x, 2);
  double muPrime02 = (Mu.m02 / Mu.m00) - pow(centroid.y, 2);
  double diffmuprime2 = muPrime20 - muPrime02;
  double muPrime11 = (Mu.m11 / Mu.m00) - (centroid.x *
      centroid.y);

  if (diffmuprime2 != 0) {
    theta = 0.5 * atan((2 * muPrime11) / diffmuprime2);
  }

  double term1 = (muPrime20 + muPrime02) /2;
  double term2 = sqrt(4 * pow(muPrime11, 2) + pow(
      diffmuprime2, 2)) / 2;
  eccentricty = sqrt(1-(term1 - term2)/ (term1 + term2));
}

void ImageProcessing::RotateImg(Mat &src, Mat &dst, double &
    theta,
                                cv::Point_<double> &Centroid
                                    , cv::Rect &ROI) {
  cv::Mat temp;
  temp.setTo(0);
  double alpha = cos(theta);
  double beta = sin(theta);
  double cx = src.cols / 2;
  double cy = src.rows / 2;
  double dx = cx - Centroid.x;
  double dy = cy - Centroid.y;
  double rotData[2][3]{{alpha, beta, alpha * dx + beta * dy
      + Centroid.x},
                       {-beta, alpha, alpha * dy + beta * dx
                           + Centroid.y}};
  cv::Mat totalrot(2, 3, CV_64FC1, rotData);

  cv::warpAffine(src, temp, totalrot, cv::Size(src.rows *
      2.5, src.cols * 2.5),
                 INTER_LINEAR);
  // determine the actual ROI
  cv::Point minP(0, 0);
  if (src.channels() == 1) {
    uchar *O = temp.data;
    uint32_t nData = temp.rows * temp.cols;
    minP.x = temp.rows;
    minP.y = temp.cols;
    cv::Point maxP(0, 0);
    int X, Y;
    for (uint32_t i = 0; i < nData; i++) {
      if (O[i] != 0) {
        Y = floor(i / temp.cols);
        X = (i % temp.cols);
        if (X < minP.x) {
          minP.x = X;
        }
```

```
        if (Y < minP.y) {
          minP.y = Y;
        }
        if (X > maxP.x) {
          maxP.x = X;
        }
        if (Y > maxP.y) {
          maxP.y = Y;
        }
      }
    }
    ROI = cv::Rect(minP, maxP);
  }

  if (src.channels() > 1) {
    Centroid.x -= cx;
    Centroid.y -= cy;

    double xnew = Centroid.x * alpha - Centroid.y * beta;
    double ynew = Centroid.x * beta - Centroid.y * alpha;

    Centroid.x = xnew + cx + minP.x;
    Centroid.y = ynew + cy + minP.y;
  }
  dst = temp(ROI).clone();
}

boost::signals2::connection
ImageProcessing::connect_Progress(const Progress_t::
    slot_type &subscriber) {
  return prog_sig.connect(subscriber);
}
}
```

E.0.2 Conversion Class

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#include "ImageProcessing.h"
#include "ConversionNotSupportedException.h"

namespace Vision {
class Conversion : public ImageProcessing {
public:
  /*! Enumerator which indicates the colorspace used*/
  enum ColorSpace {
    CIE_lab,    /*!< CIE La*b* colorspace */
    CIE_XYZ,    /*!< CIE XYZ colorspace */
    RI,         /*!< Redness Index colorspace */
    RGB,        /*!< RGB colorspace */
    Intensity, /*!< Grayscale colorspace */
    None        /*!< none */
  };
  ColorSpace OriginalColorSpace;  /*!< The original colorspace*/
  ColorSpace ProcessedColorSpace; /*!< The destination colorspace*/

  Conversion();
  Conversion(const Mat &src);
  Conversion(const Conversion &rhs);

  ~Conversion();

  Conversion &operator=(Conversion rhs);

  void Convert(ColorSpace convertFrom, ColorSpace convertTo,
               bool chain = false);
  void Convert(const Mat &src, Mat &dst, ColorSpace convertFrom,
               ColorSpace convertTo, bool chain = false);

private:
  /*!< Conversion matrix used in the conversion between RGB and CIE XYZ*/
  float XYZmat[3][3] = {{0.412453, 0.357580, 0.180423},
                        {0.212671, 0.715160, 0.072169},
                        {0.019334, 0.119194, 0.950227}};

  float whitePoint[3] = {
      0.9504, 1.0000, 1.0889}; /*!< Natural whitepoint in XYZ colorspace D65
```

```
                                            according to Matlab */
  // float whitePoint[3] = { 0.9642, 1.0000, 0.8251 }; /*!<
      Natural whitepoint
  // in XYZ colorspace D50 according to Matlab */

  void Lab2RI(float *O, float *P, int nData);
  void RGB2XYZ(uchar *O, float *P, int nData);
  void XYZ2Lab(float *O, float *P, int nData);
  void RGB2Intensity(uchar *O, uchar *P, int nData);
  inline float f_xyz2lab(float t);
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
     strictly prohibited
 * and only allowed with the written consent of the author (
     Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class Conversion
class which converts a cv::Mat image from one colorspace to
    the next colorspace
*/
#include "Conversion.h"
namespace Vision {
/*! Constructor of the class     */
Conversion::Conversion() {
  OriginalColorSpace = None;
  ProcessedColorSpace = None;
}

/*! Constructor of the class
\param src a cv::Mat object which is the source image
*/
Conversion::Conversion(const Mat &src) {
  OriginalColorSpace = None;
  ProcessedColorSpace = None;
  OriginalImg = src;
}

/*! Copy constructor*/
Conversion::Conversion(const Conversion &rhs) {
  this->OriginalColorSpace = rhs.OriginalColorSpace;
  this->OriginalImg = rhs.OriginalImg;
  this->ProcessedColorSpace = rhs.ProcessedColorSpace;
  this->ProcessedImg = rhs.ProcessedImg;
  this->TempImg = rhs.TempImg;
}

/*! De-constructor of the class*/
Conversion::~Conversion() {}

/*! Assignment operator*/
```

```
Conversion &Conversion::operator=(Conversion rhs) {
  if (&rhs != this) {
    this->OriginalColorSpace = rhs.OriginalColorSpace;
    this->OriginalImg = rhs.OriginalImg;
    this->ProcessedColorSpace = rhs.ProcessedColorSpace;
    this->ProcessedImg = rhs.ProcessedImg;
    this->TempImg = rhs.TempImg;
  }
  return *this;
}

/*! Convert the source image from one colorspace to a
   destination colorspace
- RGB 2 Intensity
- RGB 2 XYZ
- RGB 2 Lab
- RGB 2 Redness Index
- XYZ 2 Lab
- XYZ 2 Redness Index
- Lab 2 Redness Index
\param src a cv::Mat object which is the source image
\param dst a cv::Mat object which is the destination image
\param convertFrom the starting colorspace
\param convertTo the destination colorspace
\param chain use the results from the previous operation
   default value = false;
*/
void Conversion::Convert(const Mat &src, Mat &dst,
   ColorSpace convertFrom,
                         ColorSpace convertTo, bool chain) {
  OriginalImg = src;
  Convert(convertFrom, convertTo, chain);
  dst = ProcessedImg;
}

/*! Convert the source image from one colorspace to a
   destination colorspace
posibilities are:
- RGB 2 Intensity
- RGB 2 XYZ
- RGB 2 Lab
- RGB 2 Redness Index
- XYZ 2 Lab
- XYZ 2 Redness Index
- Lab 2 Redness Index
\param convertFrom the starting colorspace
\param convertTo the destination colorspace
\param chain use the results from the previous operation
   default value = false;
*/
void Conversion::Convert(ColorSpace convertFrom, ColorSpace
   convertTo,
                         bool chain) {
  OriginalColorSpace = convertFrom;
  ProcessedColorSpace = convertTo;

```

```
  // Exception handling
  EMPTY_CHECK(OriginalImg);
  currentProg = 0.;
  prog_sig(currentProg, "Converting colorspace");

  int nData = OriginalImg.rows * OriginalImg.cols;
  // uint32_t i, j;

  if (convertFrom == RGB && convertTo == Intensity) // RGB 2 Intensity
  {
    ProcessedImg.create(OriginalImg.size(), CV_8UC1);
    uchar *P = ProcessedImg.data;
    uchar *O;
    CHAIN_PROCESS(chain, O, uchar);

    prog_sig(currentProg, "RGB 2 Intensity conversion");
    RGB2Intensity(O, P, nData);
    currentProg += ProgStep;
    prog_sig(currentProg, "RGB 2 Intensity conversion Finished");
  } else if (convertFrom == RGB && convertTo == CIE_XYZ) // RGB 2 XYZ
  {
    ProcessedImg.create(OriginalImg.size(), CV_32FC3);
    float *P = (float *)ProcessedImg.data;
    uchar *O;
    CHAIN_PROCESS(chain, O, uchar);

    prog_sig(currentProg, "RGB 2 CIE XYZ conversion");
    RGB2XYZ(O, P, nData);
    currentProg += ProgStep;
    prog_sig(currentProg, "RGB 2 CIE XYZ conversion Finished");
  } else if (convertFrom == RGB && convertTo == CIE_lab) // RGB 2 Lab
  {
    ProcessedImg.create(OriginalImg.size(), CV_32FC3);
    float *P = (float *)ProcessedImg.data;
    uchar *O;
    CHAIN_PROCESS(chain, O, uchar);

    prog_sig(currentProg, "RGB 2 CIE XYZ conversion");
    RGB2XYZ(O, P, nData);
    currentProg += ProgStep;
    prog_sig(currentProg, "RGB 2 CIE XYZ conversion Finished");
    Convert(CIE_XYZ, CIE_lab, true);
  } else if (convertFrom == RGB && convertTo == RI) // RGB 2 RI
  {
    ProcessedImg.create(OriginalImg.size(), CV_32FC3);
    float *P = (float *)ProcessedImg.data;
    uchar *O;
    CHAIN_PROCESS(chain, O, uchar);

```

```
    prog_sig(currentProg, "RGB 2 CIE XYZ conversion");
    RGB2XYZ(O, P, nData);
    currentProg += ProgStep;
    prog_sig(currentProg, "RGB 2 CIE XYZ conversion Finished");
    Convert(CIE_XYZ, CIE_lab, true);
    Convert(CIE_lab, RI, true);
  } else if (convertFrom == CIE_XYZ && convertTo == CIE_lab) // XYZ 2 Lab
  {
    ProcessedImg.create(OriginalImg.size(), CV_32FC3);
    float *P = (float *)ProcessedImg.data;
    float *O;
    CHAIN_PROCESS(chain, O, float);

    prog_sig(currentProg, "CIE XYZ 2 CIE La*b* conversion");
    XYZ2Lab(O, P, nData);
    currentProg += ProgStep;
    prog_sig(currentProg, "CIE XYZ 2 CIE La*b* conversion Finished");
  } else if (convertFrom == CIE_XYZ && convertTo == RI) // XYZ 2 RI
  {
    ProcessedImg.create(OriginalImg.size(), CV_32FC3);
    float *P = (float *)ProcessedImg.data;
    float *O;
    CHAIN_PROCESS(chain, O, float);

    prog_sig(currentProg, "CIE XYZ 2 CIE La*b* conversion");
    XYZ2Lab(O, P, nData);
    currentProg += ProgStep;
    prog_sig(currentProg, "CIE XYZ 2 CIE La*b* conversion Finished");
    Convert(CIE_lab, RI, true);
  } else if (convertFrom == CIE_lab && convertTo == RI) // Lab 2 RI
  {
    ProcessedImg.create(OriginalImg.size(), CV_32FC1);
    float *P = (float *)ProcessedImg.data;
    float *O;
    CHAIN_PROCESS(chain, O, float);

    prog_sig(currentProg, "CIE La*b* 2 Redness Index conversion");
    Lab2RI(O, P, nData * 3);
    currentProg += ProgStep;
    prog_sig(currentProg, "CIE La*b* 2 Redness Index conversion Finsihed");
  } else {
    throw Exception::ConversionNotSupportedException();
  }
}

/*! Conversion from RGB to Intensity
\param O a uchar pointer to the source image
\param P a uchar pointer to the destination image
```

```
\param nData an int indicating the total number of pixels
*/
void Conversion::RGB2Intensity(uchar *O, uchar *P, int nData
    ) {
  uint32_t i;
  int j;
  i = 0;
  j = 0;
  while (j < nData) {
    P[j++] = (*(O + i + 2) * 0.2126 + *(O + i + 1) * 0.7152
        +
              *(O + i) * 0.0722); // Grey value
    i += 3;
  }
}

/*! Conversion from RGB to CIE XYZ
\param O a uchar pointer to the source image
\param P a uchar pointer to the destination image
\param nData an int indicating the total number of pixels
*/
void Conversion::RGB2XYZ(uchar *O, float *P, int nData) {
  uint32_t endData = nData * OriginalImg.step.buf[1];
  float R, G, B;
  for (uint32_t i = 0; i < endData; i += OriginalImg.step.
      buf[1]) {
    R = static_cast<float>(*(O + i + 2) / 255.0f);
    B = static_cast<float>(*(O + i + 1) / 255.0f);
    G = static_cast<float>(*(O + i) / 255.0f);
    P[i] = (XYZmat[0][0] * R) + (XYZmat[0][1] * B) + (XYZmat
        [0][2] * G); // X
    P[i + 1] = (XYZmat[1][0] * R) + (XYZmat[1][1] * B) + (
        XYZmat[1][2] * G); // Y
    P[i + 2] = (XYZmat[2][0] * R) + (XYZmat[2][1] * B) + (
        XYZmat[2][2] * G); // Z
  }
}

/*! Conversion from CIE XYZ to CIE La*b*
\param O a uchar pointer to the source image
\param P a uchar pointer to the destination image
\param nData an int indicating the total number of pixels
*/
void Conversion::XYZ2Lab(float *O, float *P, int nData) {
  uint32_t endData = nData * 3;
  float yy0, xx0, zz0;
  for (size_t i = 0; i < endData; i += 3) {
    xx0 = *(O + i) / whitePoint[0];
    yy0 = *(O + i + 1) / whitePoint[1];
    zz0 = *(O + i + 2) / whitePoint[2];

    if (yy0 > 0.008856) {
      P[i] = (116 * pow(yy0, 0.333f)) - 16; // L
    } else {
      P[i] = 903.3 * yy0; // L
    }
```

```

    P[i + 1] = 500 * (f_xyz2lab(xx0) - f_xyz2lab(yy0));
    P[i + 2] = 200 * (f_xyz2lab(yy0) - f_xyz2lab(zz0));
  }
}

inline float Conversion::f_xyz2lab(float t) {
  if (t > 0.008856) {
    return pow(t, 0.3333333333f);
  }
  return 7.787 * t + 0.137931034482759f;
}

/*! Conversion from CIE La*b* to Redness Index
\param O a uchar pointer to the source image
\param P a uchar pointer to the destination image
\param nData an int indicating the total number of pixels
*/
void Conversion::Lab2RI(float *O, float *P, int nData) {
  uint32_t j = 0;
  float L, a, b;
  for (int i = 0; i < nData; i += 3) {
    L = *(O + i);
    a = *(O + i + 1);
    b = *(O + i + 2);
    P[j++] =
        (L * (pow((pow(a, 2.0f) + pow(b, 2.0f)), 0.5f) * (
           pow(10, 8.2f)))) /
        (b * pow(L, 6.0f));
  }
}
}
```

E.0.3 Enhance Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#define ENHANCE_VERSION 1

#include "ImageProcessing.h"
#include "../SoilMath/SoilMath.h"

using namespace std;
using namespace SoilMath;

namespace Vision {
class Enhance : public ImageProcessing {
private:
  void CalculateSumOfNeighboringPixels(uchar *O, int i, int hKsize, int nCols,
                                        uint32_t &sum);
  float CalculateStdOfNeighboringPixels(uchar *O, int i, int hKsize, int nCols,
                                         int noNeighboursPix, float mean);

public:
  /*! Enumerator indicating the requested enhancement operation*/
  enum EnhanceOperation {
    _AdaptiveContrastStretch, /*!< custom adaptive contrast stretch operation*/
    _Blur,                    /*!< Blur operation*/
    _HistogramEqualization    /*!< Histogram equalization*/
  };

  Enhance();
  Enhance(const Mat &src);
  Enhance(const Mat &src, Mat &dst, uint8_t kernelsize = 9, float factor = 1.0,
          EnhanceOperation operation = _Blur);
  Enhance(const Enhance &rhs);

  ~Enhance();

  Enhance &operator=(Enhance rhs);

  void AdaptiveContrastStretch(uint8_t kernelsize, float factor,
                               bool chain = false);
```

```
  void AdaptiveContrastStretch(const Mat &src, Mat &dst, uint8_t kernelsize,
                               float factor);

  void Blur(uint8_t kernelsize, bool chain = false);
  void Blur(const Mat &src, Mat &dst, uint8_t kernelsize);

  void HistogramEqualization(bool chain = false);
  void HistogramEqualization(const Mat &src, Mat &dst);
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class Enhance
class which enhances a greyscale cv::Mat image
*/
#include "Enhance.h"

namespace Vision {
/*! Constructor*/
Enhance::Enhance() {}

/*! Constructor
\param src cv::Mat source image
*/
Enhance::Enhance(const Mat &src) {
  OriginalImg = src;
  ProcessedImg.create(OriginalImg.size(), CV_8UC1);
}

Enhance::Enhance(const Enhance &rhs) {
  this->OriginalImg = rhs.OriginalImg;
  this->ProcessedImg = rhs.OriginalImg;
  this->TempImg = rhs.TempImg;
}

/*! Constructor
\param src cv::Mat source image
\param dst cv::Mat destination image
\param kernelsize an uchar which represent the kernelsize should be an uneven
number higher than two
\param factor float which indicates the amount the effect should take place
standard value is 1.0 only used in the adaptive contrast stretch enhancement
\param operation enumerator EnhanceOperation which enhancement should be
```

```
performed
*/
Enhance::Enhance(const Mat &src, Mat &dst, uchar kernelsize, float factor,
                 EnhanceOperation operation) {
  OriginalImg = src;
  ProcessedImg.create(OriginalImg.size(), CV_8UC1);
  switch (operation) {
  case Vision::Enhance::_AdaptiveContrastStretch:
    AdaptiveContrastStretch(kernelsize, factor);
    break;
  case Vision::Enhance::_Blur:
    Blur(kernelsize);
    break;
  case Vision::Enhance::_HistogramEqualization:
    HistogramEqualization();
    break;
  }
  dst = ProcessedImg;
}

/*! Dec-constructor*/
Enhance::~Enhance() {}

Enhance &Enhance::operator=(Enhance rhs) {
  if (&rhs != this) {
    this->OriginalImg = rhs.OriginalImg;
    this->ProcessedImg = rhs.ProcessedImg;
    this->TempImg = rhs.ProcessedImg;
  }
  return *this;
}

/*! Calculate the standard deviation of the neighboring pixels
\param O uchar pointer to the current pixel of the original image
\param i current counter
\param hKsize half the kernelsize
\param nCols total number of columns
\param noNeighboursPix total number of neighboring pixels
\param mean mean value of the neighboring pixels
\return standard deviation
*/
float Enhance::CalculateStdOfNeighboringPixels(uchar *O, int i, int hKsize,
                                               int nCols, int noNeighboursPix,
                                               float mean) {
  uint32_t sum_dev = 0.0;
  float Std = 0.0;
  sum_dev = 0.0;
  Std = 0.0;
  for (int j = -hKsize; j < hKsize; j++) {
```

```
    for (int k = -hKsize; k < hKsize; k++) {
      // sum_dev += pow((O[i + j * nCols + k] - mean), 2);
      sum_dev += SoilMath::quick_pow2((O[i + j * nCols + k] - mean));
    }
  }
  // Std = sqrt(sum_dev / noNeighboursPix);
  Std = SoilMath::fastPow((static_cast<double>(sum_dev) / noNeighboursPix), 2);
  return Std;
}

/*! Calculate the sum of the neighboring pixels
\param O uchar pointer to the current pixel of the original image
\param i current counter
\param hKsize half the kernelsize
\param nCols total number of columns
\param sum Total sum of the neighboringpixels
*/
void Enhance::CalculateSumOfNeighboringPixels(uchar *O, int i, int hKsize,
                                              int nCols, uint32_t &sum) {
  for (int j = -hKsize; j < hKsize; j++) {
    for (int k = -hKsize; k < hKsize; k++) {
      sum += O[i + j * nCols + k];
    }
  }
}

/*! Homebrew AdaptiveContrastStretch function which calculate the mean and
standard deviation from the neighboring pixels if the current pixel is higher
then the mean the value is incremented with an given factor multiplied with the
standard deviation, and decreased if it's lower then the mean.
\param src cv::Mat source image
\param dst cv::Mat destination image
\param kernelsize an uchar which represent the kernelsize should be an uneven
number higher than two
\param factor float which indicates the amount the effect should take place
standard value is 1.0 only used in the adaptive contrast stretch enhancement
*/
void Enhance::AdaptiveContrastStretch(const Mat &src, Mat &dst,
                                      uchar kernelsize, float factor) {
  OriginalImg = src;
  ProcessedImg.create(OriginalImg.size(), CV_8UC1);
```

```
  AdaptiveContrastStretch(kernelsize, factor);
  dst = ProcessedImg;
}

/*! Homebrew AdaptiveContrastStretch function which calculate the mean and
standard deviation from the neighboring pixels if the current pixel is higher
then the mean the value is incremented with an given factor multiplied with the
standard deviation, and decreased if it's lower then the mean.
\param kernelsize an uchar which represent the kernelsize should be an uneven
number higher than two
\param factor float which indicates the amount the effect should take place
standard value is 1.0 only used in the adaptive contrast stretch enhancement
\param chain use the results from the previous operation default value = false;
*/
void Enhance::AdaptiveContrastStretch(uchar kernelsize, float factor,
                                      bool chain) {
  // Exception handling
  EMPTY_CHECK(OriginalImg);
  if (kernelsize < 3 || (kernelsize % 2) == 0) {
    throw Exception::WrongKernelSizeException();
  }
  CV_Assert(OriginalImg.depth() != sizeof(uchar));

  // Make the pointers to the Data
  uchar *O;
  CHAIN_PROCESS(chain, O, uchar);
  uchar *P = ProcessedImg.data;

  int i = 0;
  int hKsize = kernelsize / 2;
  int nCols = OriginalImg.cols;
  int pStart = (hKsize * nCols) + hKsize + 1;

  int nData = OriginalImg.rows * OriginalImg.cols;
  int pEnd = nData - pStart;
  uint32_t noNeighboursPix = kernelsize * kernelsize;
  uint32_t sum;
  float mean = 0.0;

  uchar *nRow = GetNRow(nData, hKsize, nCols, OriginalImg.rows);

  i = pStart;
  while (i++ < pEnd) {
    // Checks if pixel isn't a border pixel and progresses to the new row
    if (nRow[i] == 1) {
```

```
      i += kernelsize;
    }

    // Fill the neighboring pixel array
    sum = 0;
    mean = 0;

    // Calculate the statistics
    CalculateSumOfNeighboringPixels(O, i, hKsize, nCols, sum
        );
    mean = (float)(sum / noNeighboursPix);
    float Std = CalculateStdOfNeighboringPixels(O, i, hKsize
        , nCols,
                                                    noNeighboursPix
                                                        , mean);

    // Stretch

    if (O[i] > mean) {
      // int addValue = O[i] + (int)(round(factor * Std));
      int addValue = O[i] + static_cast<int>(round(factor *
          Std));
      if (addValue < 255) {
        P[i] = addValue;
      } else {
        P[i] = 255;
      }
    } else if (O[i] < mean) {
      // int subValue = O[i] - (int)(round(factor * Std));
      int subValue = O[i] - static_cast<int>(round(factor *
          Std));
      if (subValue > 0) {
        P[i] = subValue;
      } else {
        P[i] = 0;
      }
    } else {
      P[i] = O[i];
    }
  }

  // Stretch the image with an normal histogram equalization
  HistogramEqualization(true);

  delete[] nRow;
}

/*! Blurs the image with a NxN kernel
\param src cv::Mat source image
\param dst cv::Mat destination image
\param kernelsize an uchar which represent the kernelsize
    should be an uneven
number higher than two
*/
void Enhance::Blur(const Mat &src, Mat &dst, uchar
    kernelsize) {
```

```
  OriginalImg = src;
  ProcessedImg.create(OriginalImg.size(), CV_8UC1);
  Blur(kernelsize);
  dst = ProcessedImg;
}

/*! Blurs the image with a NxN kernel
\param kernelsize an uchar which represent the kernelsize should be an uneven
number higher than two
\param chain use the results from the previous operation default value = false;
*/
void Enhance::Blur(uchar kernelsize, bool chain) {
  // Exception handling
  EMPTY_CHECK(OriginalImg);
  if (kernelsize < 3 || (kernelsize % 2) == 0) {
    throw Exception::WrongKernelSizeException();
  }
  CV_Assert(OriginalImg.depth() != sizeof(uchar));

  // Make the pointers to the Data
  uchar *O;
  CHAIN_PROCESS(chain, O, uchar);
  uchar *P = ProcessedImg.data;

  int nData = OriginalImg.rows * OriginalImg.cols;
  int hKsize = kernelsize / 2;
  int nCols = OriginalImg.cols;
  int pStart = (hKsize * nCols) + hKsize + 1;
  int pEnd = nData - pStart;
  int noNeighboursPix = kernelsize * kernelsize;
  uint32_t sum;

  int i;
  uchar *nRow = GetNRow(nData, hKsize, nCols, OriginalImg.rows);
  i = pStart;
  while (i++ < pEnd) {
    // Checks if pixel isn't a border pixel and progresses to the new row
    if (nRow[i] == 1) {
      i += kernelsize;
    }

    // Calculate the sum of the kernel
    sum = 0;
    CalculateSumOfNeighboringPixels(O, i, hKsize, nCols, sum);

    P[i] = (uchar)(round(sum / noNeighboursPix));
  }

  delete[] nRow;
}
```

```
/*! Stretches the image using a histogram
\param chain use the results from the previous operation
    default value = false;
*/
void Enhance::HistogramEqualization(bool chain) {
  // Exception handling
  EMPTY_CHECK(OriginalImg);
  CV_Assert(OriginalImg.depth() != sizeof(uchar));

  // Make the pointers to the Data
  uchar *O;
  CHAIN_PROCESS(chain, O, uchar);
  uchar *P = ProcessedImg.data;

  // Calculate the statics of the whole image
  ucharStat_t imgStats(O, OriginalImg.rows, OriginalImg.cols
     );
  float sFact;
  if (imgStats.min != imgStats.max) {
    sFact = 255.0f / (imgStats.max - imgStats.min);
  } else {
    sFact = 1.0f;
  }

  uint32_t i = 256;
  uchar LUT_changeValue[256];
  while (i-- > 0) {
    LUT_changeValue[i] = (uchar)(((float)(i)*sFact) + 0.5f);
  }

  O = OriginalImg.data;

  i = OriginalImg.cols * OriginalImg.rows + 1;
  while (i-- > 0) {
    *P++ = LUT_changeValue[*O++ - imgStats.min];
  }
}
}
```

E.0.4 Morphological filter Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
     strictly prohibited
 * and only allowed with the written consent of the author (
     Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#define MORPHOLOGICALFILTER_VERSION 1

#include "ImageProcessing.h"

namespace Vision {
class MorphologicalFilter : public ImageProcessing {
public:
  enum FilterType { OPEN, CLOSE, ERODE, DILATE, NONE };

  MorphologicalFilter();
  MorphologicalFilter(FilterType filtertype);
  MorphologicalFilter(const Mat &src, FilterType filtertype
      = FilterType::NONE);
  MorphologicalFilter(const MorphologicalFilter &rhs);

  ~MorphologicalFilter();

  MorphologicalFilter &operator=(MorphologicalFilter &rhs);

  void Dilation(const Mat &mask, bool chain = false);
  void Erosion(const Mat &mask, bool chain = false);

  void Close(const Mat &mask, bool chain = false);
  void Open(const Mat &mask, bool chain = false);

private:
  void Filter(const Mat &mask, bool chain, uchar startVal,
      uchar newVal,
                 uchar switchVal);
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
     strictly prohibited
 * and only allowed with the written consent of the author (
     Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "MorphologicalFilter.h"

namespace Vision {
```

```
MorphologicalFilter::MorphologicalFilter() {}

MorphologicalFilter::MorphologicalFilter(FilterType
    filtertype) {
  switch (filtertype) {
  case FilterType::OPEN:
    Open(OriginalImg);
    break;
  case FilterType::CLOSE:
    Close(OriginalImg);
    break;
  case FilterType::ERODE:
    Erosion(OriginalImg);
    break;
  case FilterType::DILATE:
    Dilation(OriginalImg);
    break;
  case FilterType::NONE:
    break;
  }
}

MorphologicalFilter::MorphologicalFilter(const Mat &src,
                                         FilterType
                                             filtertype) {
  OriginalImg = src;
  ProcessedImg.create(OriginalImg.size(), CV_8UC1);
  switch (filtertype) {
  case FilterType::OPEN:
    Open(OriginalImg);
    break;
  case FilterType::CLOSE:
    Close(OriginalImg);
    break;
  case FilterType::ERODE:
    Erosion(OriginalImg);
    break;
  case FilterType::DILATE:
    Dilation(OriginalImg);
    break;
  case FilterType::NONE:
    break;
  }
}

MorphologicalFilter::MorphologicalFilter(const
    MorphologicalFilter &rhs) {
  this->OriginalImg = rhs.OriginalImg;
  this->ProcessedImg = rhs.ProcessedImg;
  this->TempImg = rhs.ProcessedImg;
}

MorphologicalFilter::~MorphologicalFilter() {}

MorphologicalFilter &MorphologicalFilter::operator=(
    MorphologicalFilter &rhs) {
```

```
  if (&rhs != this) {
    this->OriginalImg = rhs.OriginalImg;
    this->ProcessedImg = rhs.ProcessedImg;
    this->TempImg = rhs.TempImg;
  }
  return *this;
}

void MorphologicalFilter::Open(const Mat &mask, bool chain) {
  Erosion(mask, chain);
  Dilation(mask, true);
}

void MorphologicalFilter::Close(const Mat &mask, bool chain) {
  Dilation(mask, chain);
  Erosion(mask, true);
}

void MorphologicalFilter::Dilation(const Mat &mask, bool chain) {
  Filter(mask, chain, 0, 1, 1);
}

void MorphologicalFilter::Erosion(const Mat &mask, bool chain) {
  Filter(mask, chain, 1, 0, 0);
}

void MorphologicalFilter::Filter(const Mat &mask, bool chain, uchar startVal,
                                 uchar newVal, uchar switchVal) {
  // Exception handling
  CV_Assert(OriginalImg.depth() != sizeof(uchar));
  EMPTY_CHECK(OriginalImg);
  if (mask.cols % 2 == 0 || mask.cols < 3) {
    throw Exception::WrongKernelSizeException("Wrong Kernelsize columns!");
  }
  if (mask.rows % 2 == 0 || mask.rows < 3) {
    throw Exception::WrongKernelSizeException("Wrong Kernelsize rows!");
  }

  uint32_t hKsizeCol = (mask.cols / 2);
  uint32_t hKsizeRow = (mask.rows / 2);

  // make Pointers
  Mat workOrigImg(ProcessedImg.rows + mask.rows, ProcessedImg.cols + mask.cols,
                  CV_8UC1);
  workOrigImg.setTo(0);
  if (chain) {
    ProcessedImg.copyTo(workOrigImg(
```

```
          cv::Rect(hKsizeCol, hKsizeRow, ProcessedImg.cols,
             ProcessedImg.rows)));
      // workOrigImg(cv::Rect(hKsizeCol, hKsizeRow,
         ProcessedImg.cols,
      // ProcessedImg.rows)) = ProcessedImg.clone();
    } else {
      OriginalImg.copyTo(workOrigImg(
          cv::Rect(hKsizeCol, hKsizeRow, ProcessedImg.cols,
             ProcessedImg.rows)));
      // workOrigImg(cv::Rect(hKsizeCol, hKsizeRow,
         ProcessedImg.cols,
      // ProcessedImg.rows)) = OriginalImg.clone();
    }
    uchar *O = workOrigImg.data;

    Mat workProcImg(ProcessedImg.rows + mask.rows,
       ProcessedImg.cols + mask.cols,
                    CV_8UC1);
    uchar *P = workProcImg.data;

    // Init the relevant data
    //uint32_t nData = OriginalImg.cols * OriginalImg.rows;
    uint32_t nWData = workProcImg.cols * workProcImg.rows;
    uint32_t nWStart = (hKsizeRow * workProcImg.cols) +
       hKsizeRow;
    uint32_t nWEnd = nWData - hKsizeCol - hKsizeRow *
       workProcImg.cols - 1;
    uchar *nRow = GetNRow(nWData, hKsizeCol, workProcImg.cols,
        workProcImg.rows);
    int MaskPixel = 0, OPixel = 0;

    workProcImg.setTo(0);
    if (startVal != 0) {
      workProcImg(cv::Rect(hKsizeCol, hKsizeRow, ProcessedImg.
         cols,
                           ProcessedImg.rows)).setTo(startVal)
                              ;
    }
    SHOW_DEBUG_IMG(workOrigImg, uchar, 255, "workOrigImg
       Filter!", false);
    SHOW_DEBUG_IMG(mask, uchar, 255, "Filter mask", true);

    for (uint32_t i = nWStart; i < nWEnd; i++) {
      // Checks if pixel isn't a border pixel and progresses
         to the new row
      if (nRow[i] == 1) {
        i += mask.cols;
      }
      for (int r = 0; r < mask.rows; r++) {
        for (int c = 0; c < mask.cols; c++) {
          MaskPixel = c + r * mask.cols;
          OPixel = i - hKsizeCol + c + (r - hKsizeRow) *
             workProcImg.cols;
          if (mask.data[MaskPixel] == 1 && O[OPixel] ==
             switchVal) {
            P[i] = newVal;
```

```
          c = mask.cols;
          r = mask.rows;
        }
      }
    }
  }
  delete[] nRow;
  SHOW_DEBUG_IMG(workProcImg, uchar, 255, "workProcImg
     Filter!", true);
  ProcessedImg = workProcImg(Rect(hKsizeCol, hKsizeRow,
     ProcessedImg.cols,
                                   ProcessedImg.rows)).clone
                                      ();
  SHOW_DEBUG_IMG(ProcessedImg, uchar, 255, "Processed Image
     Filter!", true);
}
}
```

E.0.5 Segment Class

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <vector>
#include <queue>
#include <string>
#include <stdint.h>
#include <iostream>
#include <algorithm>
#include <utility>

#include <boost/range/adaptor/reversed.hpp>

#include "opencv2/imgproc/imgproc.hpp"

#include "ImageProcessing.h"
#include "MorphologicalFilter.h"
#include "../SoilMath/SoilMath.h"

namespace Vision {
class Segment : public ImageProcessing {
public:
  /*! Coordinates for the region of interest*/
  typedef struct Rect {
    uint16_t leftX;  /*!< Left X coordinate*/
    uint16_t leftY;  /*!< Left Y coordinate*/
    uint16_t rightX; /*!< Right X coordinate*/
    uint16_t rightY; /*!< Right Y coordinate*/
    Rect(uint16_t lx, uint16_t ly, uint16_t rx, uint16_t ry)
        : leftX(lx), leftY(ly), rightX(rx), rightY(ry){}
  } Rect_t;

  typedef std::vector<Vision::Segment::Rect_t> RectList_t;

  /*! Individual blob*/
  typedef struct Blob {
    uint16_t Label; /*!< ID of the blob*/
    cv::Mat Img;   /*!< BW image of the blob all the pixel belonging to the blob
                        are set to 1 others are 0*/
    cv::Rect ROI;  /*!< Coordinates for the blob in the original picture as a
                        cv::Rect*/
    uint32_t Area; /*!< Calculated stats of the blob*/
    cv::Point_<double> Centroid;
    double Theta;
```

```cpp
    Blob(uint16_t label, uint32_t area) : Label(label), Area(area){}
  } Blob_t;

  typedef std::vector<Blob_t> BlobList_t;
  BlobList_t BlobList; /*!< vector with all the individual blobs*/

  /*! Enumerator to indicate what kind of object to extract */
  enum TypeOfObjects {
    Bright, /*!< Enum value Bright object */
    Dark    /*!< Enum value Dark object. */
  };

  /*! Enumerator to indicate how the pixel correlate between each other in a
   * blob*/
  enum Connected {
    Four =
        2, /*!< Enum Four connected, relation between Center, North, East, South
          and West*/
    Eight =
        4 /*!< Enum Eight connected, relation between Center, North, NorthEast,
         East, SouthEast, South, SouthWest, West and NorthWest */
  };

  /*!< Enumerator which indicate which Segmentation technique should be used */
  enum SegmentationType {
    Normal, /*!< Segmentation looking at the intensity of an individual pixel */
    LabNeuralNet, /*!< Segmentation looking at the chromatic a* and b* of the
                     processed pixel and it's surrounding pixels, feeding it in
                     an Neural Net */
    GraphMinCut /*!< Segmentation using a graph function and the minimum cut */
  };

  cv::Mat LabelledImg;   /*!< Image with each individual blob labeled with a
                              individual number */
  uint16_t MaxLabel = 0; /*!< Maximum labels found in the labelled image*/
  uint16_t noOfFilteredBlobs =
      0; /*!< Total numbers of blobs that where filtered beacuse the where
            smaller than the minBlobArea*/

  ucharStat_t OriginalImgStats; /*!< Statistical data from the original image*/
```

```
    uint8_t ThresholdLevel = 0;    /*!< Current calculated threshold level*/

    float sigma = 2;
    uint32_t thresholdOffset = 4;

    Segment();
    Segment(const Mat &src);
    Segment(const Segment &rhs);

    ~Segment();

    Segment &operator=(Segment &rhs);

    void LoadOriginalImg(const Mat &src);

    void ConvertToBW(TypeOfObjects Typeobjects);
    void ConvertToBW(const Mat &src, Mat &dst, TypeOfObjects Typeobjects);

    void GetEdges(bool chain = false, Connected conn = Eight);
    void GetEdges(const Mat &src, Mat &dst, bool chain = false,
                  Connected conn = Eight);

    void GetEdgesEroding(bool chain = false);

    void GetBlobList(bool chain = false, Connected conn = Eight);

    void Threshold(uchar t, TypeOfObjects Typeobjects);

    void LabelBlobs(bool chain = false, uint16_t minBlobArea = 25,
                    Connected conn = Eight);

    void RemoveBorderBlobs(uint32_t border = 1, bool chain = false);

    void FillHoles(bool chain = false);

private:
    uint8_t GetThresholdLevel(TypeOfObjects TypeObject);
    void SetBorder(uchar *P, uchar setValue);
    void FloodFill(uchar *O, uchar *P, uint16_t x, uint16_t y, uchar fillValue,
                   uchar OldValue);
    void MakeConsecutive(uint16_t *valueArr, uint32_t noElem, uint16_t &maxlabel);
    void MakeConsecutive(uint16_t *valueArr, uint16_t *keyArr, uint16_t noElem,
                         uint16_t &maxlabel);
    void SortAdjacencyList(std::vector<std::vector<uint16_t>> &adj);
    void ConnectedBlobs(uchar *O, uint16_t *P,
```

```
                        std::vector<std::vector<uint16_t>> &
                           adj, uint32_t nCols,
                        uint32_t nRows, Connected conn);
  void InvertAdjacencyList(std::vector<std::vector<uint16_t
     >> &adj,
                             std::vector<std::vector<uint16_t
                                >> &adjInv);
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class  Segment
\brief Segmentation algorithms
With this class, various segmentation routines can be
   applied to a greyscale or
black and white source image.
*/
#include "Segment.h"

namespace Vision {
//! Constructor of the Segmentation class
Segment::Segment() {}

//! Constructor of the Segmentation class
Segment::Segment(const Mat &src) {
  OriginalImg = src;
  ProcessedImg.create(OriginalImg.size(), CV_8UC1);
  LabelledImg.create(OriginalImg.size(), CV_16UC1);
}

Segment::Segment(const Segment &rhs) {
  this->BlobList = rhs.BlobList;
  this->LabelledImg = rhs.LabelledImg;
  this->MaxLabel = rhs.MaxLabel;
  this->noOfFilteredBlobs = rhs.noOfFilteredBlobs;
  this->OriginalImg = rhs.OriginalImg;
  this->OriginalImgStats = rhs.OriginalImgStats;
  this->ProcessedImg = rhs.ProcessedImg;
  this->TempImg = rhs.TempImg;
  this->ThresholdLevel = rhs.ThresholdLevel;
}

//! De-constructor
Segment::~Segment() {}

Segment &Segment::operator=(Segment &rhs) {
  if (&rhs != this) {
    this->BlobList = rhs.BlobList;
```

```
    this->LabelledImg = rhs.LabelledImg;
    this->MaxLabel = rhs.MaxLabel;
    this->noOfFilteredBlobs = rhs.noOfFilteredBlobs;
    this->OriginalImg = rhs.OriginalImg;
    this->OriginalImgStats = rhs.OriginalImgStats;
    this->ProcessedImg = rhs.ProcessedImg;
    this->TempImg = rhs.TempImg;
    this->ThresholdLevel = rhs.ThresholdLevel;
  }
  return *this;
}

void Segment::LoadOriginalImg(const Mat &src) {
  OriginalImg = src;
  ProcessedImg.create(OriginalImg.size(), CV_8UC1);
  LabelledImg.create(OriginalImg.size(), CV_16UC1);
}

/*! Determine the threshold level by iteration, between two distribution,
presumably back- and foreground. It works towards the average of the two
averages and finally sets the threshold with two time the standard deviation
from the mean of the set object
\param TypeObject is an enumerator indicating if the bright or the dark pixels
are the object and should be set to one
\return The threshold level as an uint8_t  */
uint8_t Segment::GetThresholdLevel(TypeOfObjects TypeObject) {
  // Exception handling
  EMPTY_CHECK(OriginalImg);
  CV_Assert(OriginalImg.depth() != sizeof(uchar));

  // Calculate the statistics of the whole picture
  ucharStat_t OriginalImgStats(OriginalImg.data, OriginalImg.rows,
                               OriginalImg.cols);

  // Sets the initial threshold with the mean of the total picture
  pair<uchar, uchar> T;
  T.first = (uchar)(OriginalImgStats.Mean + 0.5);
  T.second = 0;

  uchar Rstd = 0;
  uchar Lstd = 0;
  uchar Rmean = 0;
  uchar Lmean = 0;

  // Iterate till optimum Threshold is found between back- & foreground
  while (T.first != T.second) {
    // Gets an array of the left part of the histogram
    uint32_t i = T.first;
```

```
    uint32_t *Left = new uint32_t[i]{};
    while (i-- > 0) {
      Left[i] = OriginalImgStats.bins[i];
    }

    // Gets an array of the right part of the histogram
    uint32_t rightEnd = 256 - T.first;
    uint32_t *Right = new uint32_t[rightEnd]{};
    i = rightEnd;
    while (i-- > 0) {
      Right[i] = OriginalImgStats.bins[i + T.first];
    }

    // Calculate the statistics of both histograms,
    // taking into account the current threshold
    ucharStat_t sLeft(Left, 0, T.first);
    ucharStat_t sRight(Right, T.first, 256);

    // Calculate the new threshold the mean of the means
    T.second = T.first;
    T.first = (uchar)(((sLeft.Mean + sRight.Mean) / 2) +
        0.5);

    Rmean = (uchar)(sRight.Mean + 0.5);
    Lmean = (uchar)(sLeft.Mean + 0.5);
    Rstd = (uchar)(sRight.Std + 0.5);
    Lstd = (uchar)(sLeft.Std + 0.5);
    delete[] Left;
    delete[] Right;
  }

  // Assumes the pixel value of the sought object lies
      between 2 sigma
  int val = 0;
  switch (TypeObject) {
  case Bright:
    val = Rmean - (sigma * Rstd) - thresholdOffset;
    if (val < 0) {
      val = 0;
    } else if (val > 255) {
      val = 255;
    }
    T.first = (uchar)val;
    break;
  case Dark:
    val = Lmean + (sigma * Lstd) + thresholdOffset;
    if (val < 0) {
      val = 0;
    } else if (val > 255) {
      val = 255;
    }
    T.first = (uchar)val;
    break;
  }

  return T.first;
```

```
}

/*! Convert a greyscale image to a BW using an automatic
    Threshold
\param src is the source image as a cv::Mat
\param dst destination image as a cv::Mat
\param TypeObject is an enumerator indicating if the bright
    or the dark pixels
are the object and should be set to one */
void Segment::ConvertToBW(const Mat &src, Mat &dst,
    TypeOfObjects Typeobjects) {
  OriginalImg = src;
  ProcessedImg.create(OriginalImg.size(), CV_8UC1);
  LabelledImg.create(OriginalImg.size(), CV_16UC1);
  ConvertToBW(Typeobjects);
  dst = ProcessedImg;
}

/*! Convert a greyscale image to a BW using an automatic
    Threshold
\param TypeObject is an enumerator indicating if the bright
    or the dark pixels
are the object and should be set to one     */
void Segment::ConvertToBW(TypeOfObjects Typeobjects) {
  // Determine the threshold
  uchar T = GetThresholdLevel(Typeobjects);

  // Threshold the picture
  Threshold(T, Typeobjects);
}

/*! Convert a greyscale image to a BW
\param t uchar set the value which is the tiping point
\param TypeObject is an enumerator indicating if the bright
    or the dark pixels
are the object and should be set to one     */
void Segment::Threshold(uchar t, TypeOfObjects Typeobjects)
    {
  // Exception handling
  EMPTY_CHECK(OriginalImg);
  CV_Assert(OriginalImg.depth() != sizeof(uchar) ||
            OriginalImg.depth() != sizeof(uint16_t));

  // Create LUT
  uchar LUT_newValue[256]{0};
  if (Typeobjects == Bright) {
    for (uint32_t i = t; i < 256; i++) {
      LUT_newValue[i] = 1;
    }
  } else {
    for (uint32_t i = 0; i <= t; i++) {
      LUT_newValue[i] = 1;
    }
  }

  // Create the pointers to the data
```

```
  uchar *P = ProcessedImg.data;
  uchar *O = OriginalImg.data;

  // Fills the ProcessedImg with either a 0 or 1
  for (int i = 0; i < OriginalImg.cols * OriginalImg.rows; i++) {
    P[i] = LUT_newValue[O[i]];
  }
}

/*! Set all the border pixels to a set value
\param *P uchar pointer to the Mat.data
\param setValue uchar the value which is written to the border pixels
*/
void Segment::SetBorder(uchar *P, uchar setValue) {
  // Exception handling
  EMPTY_CHECK(OriginalImg);
  CV_Assert(OriginalImg.depth() != sizeof(uchar) ||
            OriginalImg.depth() != sizeof(uint16_t));

  uint32_t nData = OriginalImg.cols * OriginalImg.rows;

  // Set borderPixels to 2
  uint32_t i = 0;
  uint32_t pEnd = OriginalImg.cols + 1;

  // Set the top row to value 2
  while (i < pEnd) {
    P[i++] = setValue;
  }

  // Set the bottom row to value 2
  i = nData + 1;
  pEnd = nData - OriginalImg.cols;
  while (i-- > pEnd) {
    P[i] = setValue;
  }

  // Sets the first and the last Column to 2
  i = 1;
  pEnd = OriginalImg.rows;
  while (i < pEnd) {
    P[(i * OriginalImg.cols) - 1] = setValue;
    P[(i++ * OriginalImg.cols)] = setValue;
  }
}

/*! Remove the blobs that are connected to the border
\param conn set the pixel connection eight or four
\param chain use the results from the previous operation default value = false;
*/
void Segment::RemoveBorderBlobs(uint32_t border, bool chain) {
  CV_Assert(OriginalImg.depth() != sizeof(uchar));
```

```
  EMPTY_CHECK(OriginalImg);
  // make Pointers
  uchar *O;
  CHAIN_PROCESS(chain, O, uchar);
  if (chain) {
    ProcessedImg = TempImg.clone();
  } else {
    ProcessedImg = OriginalImg.clone();
  }

  SHOW_DEBUG_IMG(OriginalImg, uchar, 255, "Original Image
      RemoverBorderBlobs!",
                 true);
  SHOW_DEBUG_IMG(TempImg, uchar, 255, "Temp Image
      RemoverBorderBlobs!", true);

  uchar *P = ProcessedImg.data;
  uint32_t cols = ProcessedImg.cols;
  uint32_t rows = ProcessedImg.rows;

  try {
    for (uint32_t i = 0; i < border; i++) {
      for (uint32_t j = 0; j < cols; j++) {
        if (O[(i * cols) + j] == 1 && P[(i * cols) + j] !=
            2) {
          cv::floodFill(ProcessedImg, cv::Point(j, i), (
              uchar)2);
        }
      }
    }

    for (uint32_t i = rows - border - 1; i < rows; i++) {
      for (uint32_t j = 0; j < cols; j++) {
        if (O[(i * cols) + j] == 1 && P[(i * cols) + j] !=
            2) {
          cv::floodFill(ProcessedImg, cv::Point(j, i), (
              uchar)2);
        }
      }
    }

    for (uint32_t i = border; i < rows - border; i++) {
      for (uint32_t j = 0; j < border; j++) {
        if (O[(i * cols) + j] == 1 && P[(i * cols) + j] !=
            2) {
          cv::floodFill(ProcessedImg, cv::Point(j, i), (
              uchar)2);
        }
        if (O[(i * cols) + (cols - j - 1)] == 1 &&
             P[(i * cols) + (cols - j - 1)] != 2) {
          cv::floodFill(ProcessedImg, cv::Point(cols - j -
              1, i), (uchar)2);
        }
      }
    }
  } catch (cv::Exception &e) {
```

```
  }
  SHOW_DEBUG_IMG(ProcessedImg, uchar, 255,
                 "Processed Image RemoverBorderBlobs before LUT!", true);

  // Change values 2 -> 0
  uchar LUT_newValue[3]{0, 1, 0};
  P = ProcessedImg.data;
  uint32_t nData = rows * cols;
  for (uint32_t i = 0; i < nData; i++) {
    P[i] = LUT_newValue[P[i]];
  }

  SHOW_DEBUG_IMG(ProcessedImg, uchar, 255,
                 "Processed Image RemoverBorderBlobs!", true);
}

/*! Label all the individual blobs in a BW source image. The result are written
to the labelledImg as an ushort
\param conn set the pixel connection eight or four
\param chain use the results from the previous operation default value = false;
\param minBlobArea minimum area when an artifact is considered a blob
*/
void Segment::LabelBlobs(bool chain, uint16_t minBlobArea, Connected conn) {
  // Exception handling
  CV_Assert(OriginalImg.depth() != sizeof(uchar));
  EMPTY_CHECK(OriginalImg);

  // make the Pointers to the data
  uchar *O;
  if (chain) {
    TempImg = ProcessedImg.clone();
    ProcessedImg = cv::Mat(OriginalImg.rows, OriginalImg.cols, CV_16UC1);
    O = (uchar *)TempImg.data;
  } else {
    O = (uchar *)OriginalImg.data;
  }
  uint16_t *P = (uint16_t *)LabelledImg.data;

  uint32_t nCols = OriginalImg.cols;
  uint32_t nRows = OriginalImg.rows;
  uint32_t nData = nCols * nRows;

  vector<vector<uint16_t>> CLdownstream;

  ConnectedBlobs(O, P, CLdownstream, nCols, nRows,
                 conn); // First loop through the image
  SortAdjacencyList(
      CLdownstream); // Sort all the adjacencylists and make unique,
```

```

  // identify all the lowest values in the adjacent list
  uint16_t *valueArr = new uint16_t[CLdownstream.size()];
  for (int i = CLdownstream.size() - 1; i >= 0; --i) {
    std::vector<uint16_t *> route;
    uint16_t minVal = i;

    for (uint32_t j = 0; j < CLdownstream[i].size(); j++) {

      // add the first node to the queue;
      route.push_back(&CLdownstream[i][j]);

      // itterate till the last node
      bool lastNodeReached = false;
      while (!lastNodeReached) {
        uint32_t nodesVisited = route.size() - 1;
        if (*route[nodesVisited] < minVal) {
          minVal = *route[nodesVisited];
        }
        route.push_back(&CLdownstream[*route[nodesVisited
            ]][0]);
        if (route[nodesVisited] == route[nodesVisited + 1])
            {
          route.pop_back();
          lastNodeReached = true;
        }
      }
      // Set all values to the lowest value
      for (uint32_t k = 0; k < route.size(); k++) {
        *route[k] = minVal;
      }
    }
    valueArr[i] = minVal;
  }

  // Make numbers consecutive
  MakeConsecutive(valueArr, CLdownstream.size(), MaxLabel);

  // Second loop through the pixels to give the values a
      final value
  for_each(P, P + nData, [&](uint16_t &V) { V = valueArr[V];
      });
  delete[] valueArr;
}

/*! Create a BW image with only edges from a BW image
\param src source image as a const cv::Mat
\param dst destination image as a cv::Mat
\param conn set the pixel connection eight or four
\param chain use the results from the previous operation
    default value = false;
*/
void Segment::GetEdges(const Mat &src, Mat &dst, bool chain,
     Connected conn) {
  OriginalImg = src;
  GetEdges(chain, conn);
```

```
  dst = ProcessedImg;
}

/*! Create a BW image with only edges from a BW image
\param conn set the pixel connection eight or four
\param chain use the results from the previous operation
    default value = false;
*/
void Segment::GetEdges(bool chain, Connected conn) {
  // Exception handling
  CV_Assert(OriginalImg.depth() != sizeof(uchar));
  EMPTY_CHECK(OriginalImg);

  // make Pointers
  uchar *O;
  CHAIN_PROCESS(chain, O, uchar);
  uchar *P = ProcessedImg.data;

  uint32_t nCols = OriginalImg.cols;
  uint32_t nRows = OriginalImg.rows;
  uint32_t nData = nCols * nRows;
  uint32_t pEnd = nData + 1;
  uint32_t i = 0;

  // Loop through the image and set each pixel which has a
      zero neighbor set it
  // to two.
  if (conn == Four) {
    // Loop through the picture
    while (i < pEnd) {
      // If current value = zero processed value = zero
      if (O[i] == 0) {
        P[i] = 0;
      }
      // If current value = 1 check North West, South and
          East and act
      // accordingly
      else if (O[i] == 1) {
        uchar *nPixels = new uchar[4];
        nPixels[0] = O[i - 1];
        nPixels[1] = O[i - nCols];
        nPixels[2] = O[i + 1];
        nPixels[3] = O[i + nCols];

        // Sort the neighbors for easier checking
        SoilMath::Sort::QuickSort<uchar>(nPixels, 4);
        if (nPixels[0] == 0) {
          P[i] = 1;
        } else {
          P[i] = 0;
        }
        delete[] nPixels;
      } else {
        throw Exception::PixelValueOutOfBoundException();
      }
      i++;
```

```
      }
    } else {
      // Loop through the picture
      while (i < pEnd) {
        // If current value = zero processed value = zero
        if (O[i] == 0) {
          P[i] = 0;
        }
        // If current value = 1 check North West, South and
            East and act
        // accordingly
        else if (O[i] == 1) {
          uchar *nPixels = new uchar[8];
          nPixels[0] = O[i - 1];
          nPixels[1] = O[i - nCols];
          nPixels[2] = O[i - nCols - 1];
          nPixels[3] = O[i - nCols + 1];
          nPixels[4] = O[i + 1];
          nPixels[5] = O[i + nCols + 1];
          nPixels[6] = O[i + nCols];
          nPixels[7] = O[i + nCols - 1];

          // Sort the neighbors for easier checking
          SoilMath::Sort::QuickSort<uchar>(nPixels, 8);

          if (nPixels[0] == 0) {
            P[i] = 1;
          } else {
            P[i] = 0;
          }
          delete[] nPixels;
        } else {
          throw Exception::PixelValueOutOfBoundException();
        }
        i++;
      }
    }
}

void Segment::GetEdgesEroding(bool chain) {
  // Exception handling
  CV_Assert(OriginalImg.depth() != sizeof(uchar));
  EMPTY_CHECK(OriginalImg);

  // make Pointers
  uchar *O;
  CHAIN_PROCESS(chain, O, uchar);
  uchar *P = ProcessedImg.data;

  uint32_t nCols = OriginalImg.cols;
  uint32_t nRows = OriginalImg.rows;
  uint32_t nData = nCols * nRows;

  // Setup the erosion
  MorphologicalFilter eroder;
  if (chain) {
```

```
      eroder.OriginalImg = TempImg;
    } else {
      eroder.OriginalImg = OriginalImg;
    }
    // Setup the processed image of the eroder
    eroder.ProcessedImg.create(OriginalImg.size(), CV_8UC1);
    eroder.ProcessedImg.setTo(0);
    // Setup the mask
    Mat mask(3, 3, CV_8UC1, 1);
    // Erode the image
    eroder.Erosion(mask, false);

    // Loop through the image and set the not eroded pixels to zero
    for (uint32_t i = 0; i < nData; i++) {
      if (O[i] != eroder.ProcessedImg.data[i]) {
        P[i] = 1;
      } else {
        P[i] = 0;
      }
    }

    // ProcessedImg = OriginalImg.clone() - eroder.ProcessedImg.clone();

    SHOW_DEBUG_IMG(eroder.ProcessedImg, uchar, 255, "Eroded img Processed Image!",
                   true);
    SHOW_DEBUG_IMG(ProcessedImg, uchar, 255, "GetEdgesEroding Processed Image!",
                   true);
  }

  /*! Create a BlobList subtracting each individual blob out of a Labelled image.
  If the labelled image is empty build a new one with a BW image.
  \param conn set the pixel connection eight or four
  \param chain use the results from the previous operation default value = false;
  */
  void Segment::GetBlobList(bool chain, Connected conn) {
    // Exception handling
    CV_Assert(OriginalImg.depth() != sizeof(uchar));
    EMPTY_CHECK(OriginalImg);

    // If there isn't a labelledImg make one
    if (MaxLabel < 1) {
      LabelBlobs(chain, 5, conn);
    }

    // Make an empty BlobList
    uint32_t nCols = OriginalImg.cols;
    uint32_t nRows = OriginalImg.rows;
    uint32_t nData = nCols * nRows;
    RectList_t rectList;
```

```

  // Calculate Stats the statistics
  uint16Stat_t LabelStats((uint16_t *)LabelledImg.data,
     LabelledImg.cols,
                          LabelledImg.rows, MaxLabel + 1, 0,
                              MaxLabel);

  BlobList.reserve(LabelStats.EndBin);
  rectList.reserve(LabelStats.EndBin);

  BlobList.push_back(Blob_t(0, 0));
  rectList.push_back(Rect_t(0, 0, 0, 0));

  for (uint32_t i = 1; i < LabelStats.EndBin; i++) {
    BlobList.push_back(Blob_t(i, LabelStats.bins[i]));
    rectList.push_back(Rect_t(nCols, nRows, 0, 0));
  }

  // make Pointers
  uint16_t *L = (uint16_t *)LabelledImg.data;

  uint32_t currentX, currentY;
  // uint16_t leftX, leftY, rightX, rightY;
  // Loop through the labeled image and extract the Blobs
  for (uint32_t i = 0; i < nData; i++) {
    if (L[i] != 0) {
      /* Determine the current x and y value of the current
         blob and
      checks if it is min/max */
      currentY = i / nCols;
      currentX = i % nCols;

      // Min value
      if (currentX < rectList[L[i]].leftX) {
        rectList[L[i]].leftX = currentX;
      }
      if (currentY < rectList[L[i]].leftY) {
        rectList[L[i]].leftY = currentY;
      }

      // Max value
      if (currentX > rectList[L[i]].rightX) {
        rectList[L[i]].rightX = currentX;
      }
      if (currentY > rectList[L[i]].rightY) {
        rectList[L[i]].rightY = currentY;
      }
    }
  }

  // Loop through the BlobList and finalize it
  uint8_t *LUT_filter = new uint8_t[MaxLabel + 1]{};
  for (uint32_t i = 1; i <= MaxLabel; i++) {
    LUT_filter[i] = 1;
    BlobList[i].ROI.y = rectList[i].leftY;
    BlobList[i].ROI.x = rectList[i].leftX;
```

```
    BlobList[i].ROI.height = rectList[i].rightY - rectList[i].leftY + 1;
    BlobList[i].ROI.width = rectList[i].rightX - rectList[i].leftX + 1;
    BlobList[i].Img = CopyMat<uint8_t, uint16_t>(
        LabelledImg(BlobList[i].ROI).clone(), LUT_filter, CV_8UC1);
    //SHOW_DEBUG_IMG(BlobList[i].Img, uchar, 255, "Blob", true);
    LUT_filter[i] = 0;
  }
  delete[] LUT_filter;

  // Remove background blob
  BlobList.erase(BlobList.begin());
}

void Segment::FillHoles(bool chain) {
  // Exception handling
  CV_Assert(OriginalImg.depth() != sizeof(uchar));
  EMPTY_CHECK(OriginalImg);

  // make Pointers
  uchar *O;
  CHAIN_PROCESS(chain, O, uchar);
  if (chain) {
    ProcessedImg = TempImg.clone();
  } else {
    ProcessedImg = OriginalImg.clone();
  }

  uchar *P = ProcessedImg.data;

  // Determine the starting point of the floodfill
  int itt = -1;
  while (P[++itt] != 0)
    ;
  uint16_t row = static_cast<uint16_t>(itt / OriginalImg.rows);
  uint16_t col = static_cast<uint16_t>(itt % OriginalImg.rows);

  // Fill the outside
  try {
    cv::floodFill(ProcessedImg, cv::Point(col, row), cv::Scalar(2));
  } catch (cv::Exception &e) {
  }

  // Set the unreached areas to 1 and the outside to 0;
  uchar LUT_newVal[3] = {1, 1, 0};
  uint32_t nData = OriginalImg.rows * OriginalImg.cols;
  uint32_t i = 0;
  while (i <= nData) {
    P[i] = LUT_newVal[P[i]];
    i++;
```

```
  }
}

/*!
 * \brief Segment::SortAdjacencyList Sort the the sub vectors
 * \param adj std::vector<std::vector<uint16_t>> &adj
 */
void Segment::SortAdjacencyList(std::vector<std::vector<uint16_t>> &adj) {
  uint32_t j = 0;
  for_each(adj.begin(), adj.end(), [&](std::vector<uint16_t> &L) {
    std::sort(L.begin(), L.end());
    std::vector<uint16_t>::iterator it;
    it = std::unique(L.begin(), L.end());
    L.resize(std::distance(L.begin(), it));
    if (L.size() > 1) {
      for (std::vector<uint16_t>::iterator iter = L.begin(); iter != L.end();
           ++iter) {
        if (*iter == j) {
          L.erase(iter);
          break;
        }
      }
    }
    j++;
  });
}

/*!
 * \brief Segment::ConnectedBlobs Connect all the blobs and created the
 * adjacency list
 * \param O
 * \param P
 * \param adj
 * \param nCols
 * \param nRows
 * \param conn
 */
void Segment::ConnectedBlobs(uchar *O, uint16_t *P,
                             std::vector<std::vector<uint16_t>> &adj,
                             uint32_t nCols, uint32_t nRows, Connected conn) {
  // Determine the size of the array for beginning and endrow and middle of a
  // row
  uint32_t noConn[3] = {static_cast<uint32_t>(conn),
                        (static_cast<uint32_t>(conn) / 2),
                        (static_cast<uint32_t>(conn) / 2) + 1};
  uint32_t lastConn[3] = {noConn[0] - 1, noConn[1] - 1, noConn[2] - 1};
```

```
  uint32_t nData = nCols * nRows;

  uint16_t currentlbl = 0;
  vector<uint16_t> zeroVector;
  zeroVector.push_back(currentlbl);
  adj.push_back(zeroVector);

  // Determine which borderpixels should be handled differently
  uchar *nRow = new uchar[nData]{};
  for (uint32_t i = nCols; i < nData; i += nCols) {
    nRow[i] = 1;
    nRow[i - 1] = 2;
  }

  // Set the first pixel
  if (O[0] == 0) {
    P[0] = 0;
  } else if (O[0] == 1) {
    P[0] = 1;
  } else {
    throw Exception::PixelValueOutOfBoundException();
  }

  // Walk through the toprow and determine if it's a new blob or it's connected
  // with previously determine blob
  for (uint32_t i = 1; i < nCols; i++) {
    if (O[i] == 0) {
      P[i] = 0;
    } else if (O[i] == 1) {
      // If West is zero assume this is a new blob
      if (P[i - 1] == 0) {
        P[i] = ++currentlbl;
        vector<uint16_t> cVector;
        cVector.push_back(currentlbl);
        adj.push_back(cVector);
      } else { // set as previous blob
        P[i] = P[i - 1];
      }
    } else { // Value of of bounds
      throw Exception::PixelValueOutOfBoundException();
    }
  }

  // walk through each pixel and determine if it's a new blob or it's connected
  // with previously determine blob
  for (uint32_t i = OriginalImg.cols; i < nData; i++) {
    if (O[i] == 0) { // Original pixel = 0
      P[i] = 0;
    } else if (O[i] == 1) {
      // Get an array of Neighboring Pixels
      uint16_t *nPixels = new uint16_t[noConn[nRow[i]]];
      if (nRow[i] != 1) {
        nPixels[0] = P[i - 1];
```

```
      }
      uint32_t j = i - nCols - ((nRow[i] == 1) ? 0 : ((conn == Four) ? 0 : 1));
      for_each(nPixels + ((nRow[i] != 1) ? 1 : 0), nPixels + noConn[nRow[i]],
               [&](uint16_t &N) { N = P[j++]; });

      // Sort the neighbors for easier checking
      SoilMath::Sort::QuickSort<uint16_t>(nPixels, noConn[nRow[i]]);

      // If all are zero assume this is a new blob
      if (nPixels[lastConn[nRow[i]]] == 0) {
        P[i] = ++currentlbl;
        vector<uint16_t> cVector;
        cVector.push_back(currentlbl);
        adj.push_back(cVector);
      } else {
        /* Sets the processed value to the smallest non-zero value and update
         * the connectedLabels */
        for (uint32_t j = 0; j < noConn[nRow[i]]; j++) {
          if (nPixels[j] > 0) {
            P[i] = nPixels[j];
            break;
          }
        }

        /* If previous blobs belong to different connected components set the
         * current processed value to the lowest value and remember that the
         * other values should be the lowest value*/
        if (P[i] != nPixels[lastConn[nRow[i]]]) {
          for (int j = lastConn[nRow[i]]; j >= 0; --j) {
            if (nPixels[j] <= P[i]) {
              break;
            } else {
              adj[nPixels[j]].push_back(P[i]);
            }
          }
        }
      }
      delete[] nPixels;
    } else {
      throw Exception::PixelValueOutOfBoundException();
    }
  }
  delete[] nRow;
}

/*!
 * \brief Segment::InvertAdjacencyList invert the adjecencylist for upstream
 * (unused)
 * \param adj
```

```
 * \param adjInv
 */
void Segment::InvertAdjacencyList(std::vector<std::vector<uint16_t>> &adj,
                                  std::vector<std::vector<uint16_t>> &adjInv) {
  // Build the inverted vector
  adjInv.resize(adj.size());
  uint16_t count = 0;
  for_each(adj.begin(), adj.end(), [&](std::vector<uint16_t> &V) {
    for_each(V.begin(), V.end(),
             [&](uint16_t &C) { adjInv[C].push_back(count); });
    count++;
  });
}

/*!
 * \brief Segment::MakeConsecutive make the valueArr consequative numbers
 * \param valueArr
 * \param noElem
 * \param maxLabel
 */
void Segment::MakeConsecutive(uint16_t *valueArr, uint32_t noElem,
                              uint16_t &maxLabel) {
  std::vector<std::vector<uint16_t>> conseq;
  conseq.resize(noElem);
  for (uint32_t i = 0; i < noElem; i++) {
    conseq[valueArr[i]].push_back(i);
  }
  uint32_t count = 1;
  for (uint32_t i = 1; i < noElem; i++) {
    if (conseq[i].size() > 0) {
      for (uint32_t j = 0; j < conseq[i].size(); j++) {
        valueArr[conseq[i][j]] = count;
      }
      count++;
    }
  }
  maxLabel = count - 1;
}

/*!
 * \brief Segment::MakeConsecutive probably a fault in this function. Don't use
 * \param valueArr
 * \param keyArr
 * \param noElem
 * \param maxlabel
 */
void Segment::MakeConsecutive(uint16_t *valueArr, uint16_t *keyArr,
                              uint16_t noElem, uint16_t &
```

```
                                               maxlabel) {
  SoilMath::Sort::QuickSort<uint16_t>(valueArr, keyArr,
      noElem);
  uint16_t count = 0;
  for (uint32_t i = 1; i < noElem; i++) {
    if (valueArr[i] != valueArr[i - 1]) {
      count++;
    }
    valueArr[i] = count;
  }
  SoilMath::Sort::QuickSort<uint16_t>(keyArr, valueArr,
      noElem);
  delete[] keyArr;
  maxlabel = count;
}
}
```

E.0.6 General project files

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-06-06T12:07:42
#
#-------------------------------------------------

QT       += core concurrent
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
QMAKE_CXXFLAGS += -std=c++11

TARGET = SoilVision
TEMPLATE = lib
VERSION = 0.9.2

DEFINES += SOILVISION_LIBRARY
unix:!macx: QMAKE_RPATHDIR += $$PWD/../../../build/install/

SOURCES += \
    Segment.cpp \
    MorphologicalFilter.cpp \
    ImageProcessing.cpp \
    Enhance.cpp \
    Conversion.cpp

HEADERS += \
    WrongKernelSizeException.h \
    VisionDebug.h \
    Vision.h \
    Segment.h \
    PixelValueOutOfBoundException.h \
    MorphologicalFilter.h \
    ImageProcessing.h \
    Enhance.h \
    EmptyImageException.h \
    ConversionNotSupportedException.h \
    Conversion.h \
    ChannelMismatchException.h

unix {
    target.path = $PWD/../../../build/install
    INSTALLS += target
}

#opencv
LIBS += -L/usr/local/lib -lopencv_core -lopencv_highgui -
   lopencv_imgproc
INCLUDEPATH += /usr/local/include/opencv
INCLUDEPATH += /usr/local/include

#boost
DEFINES += BOOST_ALL_DYN_LINK
INCLUDEPATH += /usr/include/boost

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilMath
```

```

INCLUDEPATH += $$PWD/../SoilMath
DEPENDPATH += $$PWD/../SoilMath
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! Collection header of all the basic Vision headers*/

#pragma once
#include "Conversion.h"
#include "Enhance.h"
#include "Segment.h"
#include "MorphologicalFilter.h"
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
// Debuging helper macros
#ifndef DEBUG
//#define DEBUG
#endif

#ifdef DEBUG
#include <limits>
#include <opencv2/highgui/highgui.hpp>
#include <vector>
#include "ImageProcessing.h"
#ifndef SHOW_DEBUG_IMG
#define SHOW_DEBUG_IMG(img, T1, maxVal, windowName, scale) \
  Vision::ImageProcessing::ShowDebugImg<T1>(img, maxVal, windowName, scale)
#endif // !SHOW_DEBUG_IMG
#else
#ifndef SHOW_DEBUG_IMG
#define SHOW_DEBUG_IMG(img, T1, maxVal, windowName, scale)
#endif // !SHOW_DEBUG_IMG
#endif
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
```

```
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class ChannelMismatchException
Exception class which is thrown when Extracted channel out of bounds exception
*/

#pragma once

#include <exception>
#include <string>

using namespace std;

namespace Vision {
namespace Exception {
class ChannelMismatchException : public std::exception {
public:
  ChannelMismatchException(
      string m = "Extracted channel out of bounds exception!")
      : msg(m){};
  ~ChannelMismatchException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class ConversionNotSupportedException
Exception class which is thrown when an illegal conversion is requested.
*/
#pragma once

#include <exception>
#include <string>

using namespace std;

```

```
namespace Vision {
namespace Exception {
class ConversionNotSupportedException : public std::exception {
public:
  ConversionNotSupportedException(
      string m = "Requested conversion is not supported!")
      : msg(m){};
  ~ConversionNotSupportedException() _GLIBCXX_USE_NOEXCEPT {};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class EmtpyImageException
Exception class which is thrown when operations are about to start on a empty
image.
*/

#pragma once

#include <exception>
#include <string>

using namespace std;

namespace Vision {
namespace Exception {
class EmtpyImageException : public std::exception {
public:
  EmtpyImageException(string m = "Empty Image!") : msg(m){};
  ~EmtpyImageException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class PixelValueOutOfBoundException
Exception class which is thrown when an unexpected pixel value has to be
computed
*/
#pragma once

#include <exception>
#include <string>

using namespace std;

namespace Vision {
namespace Exception {
class PixelValueOutOfBoundException : public std::exception {
public:
  PixelValueOutOfBoundException(string m = "Current pixel value out of bounds!")
      : msg(m){};
  ~PixelValueOutOfBoundException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class WrongKernelSizeException
Exception class which is thrown when a wrong kernelsize is requested
*/
#pragma once

#include <exception>
```

```
#include <string>

using namespace std;

namespace Vision {
namespace Exception {
class WrongKernelSizeException : public std::exception {
public:
  WrongKernelSizeException(string m = "Wrong kernel
     dimensions!") : msg(m){};
  ~WrongKernelSizeException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return
     msg.c_str(); };

private:
  string msg;
};
}
}
```

F. Analyzer Library

F.0.1 Analyzer Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#define STARTING_ESTIMATE_PROGRESS 300
#ifndef DEBUG
//#define DEBUG
#endif

#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <vector>
#include <cmath>

#include "sample.h"
#include "soilsettings.h"
#include "soilanalyzerexception.h"

#include "SoilMath.h"

#include <QtCore/QObject>
#include <QThread>
#include <QtConcurrent>

#include "Vision.h"
```

```

namespace SoilAnalyzer {
class Analyzer : public QObject {
  Q_OBJECT

public:
  bool PredictShape = true;
  float CurrentSIfactor = 0.0111915;
  bool SIfactorDet = false;
  struct Image_t {
    cv::Mat FrontLight;
    cv::Mat BackLight;
    float SIPixelFactor = 0.0111915;
  }; /*!< */

  typedef std::vector<Image_t> Images_t; /*!< */
  Images_t *Snapshots = nullptr;         /*!< */
  SoilSettings *Settings = nullptr;      /*!< */

  Sample *Results; /*!< */

  Analyzer(Images_t *snapshots, Sample *results,
      SoilSettings *settings);

  void Analyse();
  void Analyse(Images_t *snapshots, Sample *results,
      SoilSettings *settings);
  float CalibrateSI(float SI, cv::Mat &img);

  uint32_t MaxProgress = STARTING_ESTIMATE_PROGRESS; /*!< */

  SoilMath::NN NeuralNet; /*!< */

signals:
  void on_progressUpdate(int value);    /*!< */
  void on_maxProgressUpdate(int value); /*!< */
  void on_AnalysisFinished();           /*!< */

private:
  uint32_t currentProgress = 0;   /*!< */
  uint32_t currentParticleID = 0; /*!< */
  double BinRanges[15]{0.0,  0.038, 0.045, 0.063, 0.075,
      0.09, 0.125, 0.18,
                       0.25, 0.355, 0.5,   0.71,  1.0,   1.4,
                            2.0};

  SoilMath::FFT fft; /*!< */

  void CalcMaxProgress();
  void CalcMaxProgressAnalyze();
  void PrepImages();
  void GetBW(std::vector<cv::Mat> &images, std::vector<cv::
      Mat> &BWvector);
  void GetBW(cv::Mat &img, cv::Mat &BW);

  void GetEnhancedInt(Images_t *snapshots,
```

```
                            std::vector<cv::Mat> &intensityVector);
  void GetEnhancedInt(cv::Mat &img, cv::Mat &intensity);

  void GetParticles(std::vector<cv::Mat> &BW, Images_t *snapshots,
                            Particle::ParticleVector_t &partPopulation);
  void GetParticlesFromBlobList(Vision::Segment::BlobList_t &bloblist,
                                     Image_t *snapshot,
                                     Particle::ParticleVector_t &partPopulation);

  void CleanUpMatVector(std::vector<cv::Mat> &mv);
  void CleanUpMatVector(Images_t *mv);

  void GetFFD(Particle::ParticleVector_t &particalPopulation);

  void GetPrediction(Particle::ParticleVector_t &particlePopulation);
};
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "analyzer.h"

namespace SoilAnalyzer {

/*!
*\brief Analyzer::Analyzer
*\param snapshots
*\param results
*\param settings
*/
Analyzer::Analyzer(Images_t *snapshots, Sample *results,
                   SoilSettings *settings = nullptr) {
  this->Snapshots = snapshots;
  this->Results = results;
  if (settings == nullptr) {
    Settings = new SoilSettings;
  } else {
    this->Settings = settings;
  }
  NeuralNet.LoadState(Settings->NNlocation);
}

```

```
/*!
 * \brief Analyzer::PrepImages
 */
void Analyzer::PrepImages() {
  if (Snapshots == nullptr || Snapshots->size() == 0) {
    throw Exception::SoilAnalyzerException(
        EXCEPTION_NO_SNAPSHOTS,
                                           EXCEPTION_NO_SNAPSHOTS_NR
                                           );
  }

  std::vector<cv::Mat> intensityVector;
  GetEnhancedInt(Snapshots, intensityVector);

  std::vector<cv::Mat> BWVector;
  GetBW(intensityVector, BWVector);
  CleanUpMatVector(intensityVector);

  GetParticles(BWVector, Snapshots, Results->
      ParticlePopulation);

  CleanUpMatVector(BWVector);
  CleanUpMatVector(Snapshots);

  Results->isPreparedForAnalysis = true;
}

void Analyzer::Analyse(Images_t *snapshots, Sample *results,
                       SoilSettings *settings) {
  Snapshots = snapshots;
  Results = results;
  Settings = settings;
  Analyse();
}

/*!
 * \brief Analyzer::Analyse
 */
void Analyzer::Analyse() {
  CalcMaxProgress();
  if (!Results->isPreparedForAnalysis && !Results->
      IsLoadedFromDisk) {
    PrepImages();
  }
  GetFFD(Results->ParticlePopulation);
  if (PredictShape && Settings->PredictTheShape) {
    GetPrediction(Results->ParticlePopulation);
  }

  Results->Angularity =
      ucharStat_t(Results->GetAngularityVector()->data(),
                  Results->GetAngularityVector()->size(), 1,
                         7, 0, true);
  emit on_progressUpdate(currentProgress++);

  Results->Roundness =
```

```
      ucharStat_t(Results->GetRoundnessVector()->data(),
                  Results->GetRoundnessVector()->size(), 1,
                       5, 0, true);

  Results->PSD =
      SoilMath::PSD(Results->GetPSDVector()->data(),
                     Results->GetPSDVector()->size(),
                          BinRanges, 15, 14);
  emit on_progressUpdate(currentProgress++);

  emit on_AnalysisFinished();
}

void Analyzer::CleanUpMatVector(std::vector<Mat> &mv) {
  for_each(mv.begin(), mv.end(), [](cv::Mat &I) { I.release
      (); });
  mv.clear();
}

/*!
 * \brief Analyzer::CleanUpMatVector
 * \param mv
 */
void Analyzer::CleanUpMatVector(Images_t *mv) {
  for_each(mv->begin(), mv->end(), [](Image_t &I) {
    I.BackLight.release();
    I.FrontLight.release();
  });
  mv->clear();
}

/*!
 * \brief Analyzer::CalcMaxProgress
 */
void Analyzer::CalcMaxProgress() {
  // Static processing steps
  MaxProgress += Snapshots->size() * 5;

  // Optional processing steps
  if (Settings->useBlur) {
    MaxProgress += Snapshots->size();
  }
  if (Settings->useAdaptiveContrast) {
    MaxProgress += Snapshots->size();
  }
  if (Settings->fillHoles) {
    MaxProgress += Snapshots->size();
  }
  if (Settings->ignorePartialBorderParticles) {
    MaxProgress += Snapshots->size();
  }
  if (Settings->morphFilterType != Vision::
      MorphologicalFilter::NONE) {
    MaxProgress += Snapshots->size();
  }

```

```
  emit on_maxProgressUpdate(MaxProgress);
}

void Analyzer::CalcMaxProgressAnalyze() {
  MaxProgress -= STARTING_ESTIMATE_PROGRESS;
  MaxProgress += Results->ParticlePopulation.size() * 2;

  emit on_maxProgressUpdate(MaxProgress);
}

/*!
 * \brief Analyzer::GetEnhancedInt
 * \param snapshots
 * \param intensityVector
 */
void Analyzer::GetEnhancedInt(Images_t *snapshots,
                              std::vector<Mat> &
                                  intensityVector) {
  if (Settings->useBacklightProjection) {
   for_each(snapshots->begin(), snapshots->end(), [&](
       Image_t &I) {
      cv::Mat intensity;
      GetEnhancedInt(I.BackLight, intensity);
      intensityVector.push_back(intensity);
    });
  } else {
    for_each(snapshots->begin(), snapshots->end(), [&](
        Image_t &I) {
      cv::Mat intensity;
      GetEnhancedInt(I.FrontLight, intensity);
      intensityVector.push_back(intensity);
    });
  }
}

/*!
 * \brief Analyzer::GetEnhancedInt
 * \param img
 * \param intensity
 */
void Analyzer::GetEnhancedInt(Mat &img, Mat &intensity) {
  Vision::Conversion IntConvertor(img.clone());
  IntConvertor.Convert(Vision::Conversion::RGB, Vision::
      Conversion::Intensity);
  emit on_progressUpdate(currentProgress++);
  SHOW_DEBUG_IMG(IntConvertor.ProcessedImg, uchar, 255, "RGB
      2 Int", false);

  if (Settings->useBlur) {
    Vision::Enhance IntBlur(IntConvertor.ProcessedImg.clone
        ());
    IntBlur.Blur(Settings->blurKernelSize);
    emit on_progressUpdate(currentProgress++);
    uint32_t HBK = Settings->blurKernelSize / 2;
    uint32_t BK = Settings->blurKernelSize - 1;
    if (Settings->useAdaptiveContrast) {
```

```
      Vision::Enhance IntAdaptContrast(
          IntBlur.ProcessedImg(
                      cv::Rect(HBK, HBK, IntBlur.ProcessedImg.cols - BK,
                               IntBlur.ProcessedImg.rows - BK)).clone());
      IntAdaptContrast.AdaptiveContrastStretch(
          Settings->adaptContrastKernelSize,
          Settings->adaptContrastKernelFactor);
      emit on_progressUpdate(currentProgress++);
      uint32_t HAK = Settings->adaptContrastKernelSize / 2;
      uint32_t AK = Settings->adaptContrastKernelSize - 1;
      intensity = IntAdaptContrast.ProcessedImg(
          cv::Rect(HAK, HAK, IntAdaptContrast.ProcessedImg.cols - AK,
                   IntAdaptContrast.ProcessedImg.rows - AK));
    } else {
      intensity = IntBlur.ProcessedImg(
          cv::Rect(HBK, HBK, IntBlur.ProcessedImg.cols - BK,
                   IntBlur.ProcessedImg.rows - BK));
    }
  } else if (Settings->useAdaptiveContrast) {
    Vision::Enhance IntAdaptContrast(IntConvertor.ProcessedImg.clone());
    IntAdaptContrast.AdaptiveContrastStretch(
        Settings->adaptContrastKernelSize, Settings->adaptContrastKernelFactor);
    emit on_progressUpdate(currentProgress++);
    uint32_t HAK = Settings->adaptContrastKernelSize / 2;
    uint32_t AK = Settings->adaptContrastKernelSize - 1;
    intensity = IntAdaptContrast.ProcessedImg(
        cv::Rect(HAK, HAK, IntAdaptContrast.ProcessedImg.cols - AK,
                 IntAdaptContrast.ProcessedImg.rows - AK));
  } else {
    intensity = IntConvertor.ProcessedImg;
  }
  SHOW_DEBUG_IMG(intensity, uchar, 255, "Enhanced Int", false);
}

/*!
 * \brief Analyzer::GetBW
 * \param images
 * \param BWvector
 */
void Analyzer::GetBW(std::vector<cv::Mat> &images,
                     std::vector<cv::Mat> &BWvector) {
  for_each(images.begin(), images.end(), [&](cv::Mat &I) {
    cv::Mat BW;
    GetBW(I, BW);
    BWvector.push_back(BW);
  });
}
```

```
/*!
 * \brief Analyzer::GetBW
 * \param img
 * \param BW
 */
void Analyzer::GetBW(cv::Mat &img, cv::Mat &BW) {
  Vision::Segment SegBL(img.clone());
  SegBL.sigma = Settings->sigmaFactor;
  SegBL.thresholdOffset = Settings->thresholdOffsetValue;
  SegBL.ConvertToBW(Settings->typeOfObjectsSegmented);
  emit on_progressUpdate(currentProgress++);
  SHOW_DEBUG_IMG(SegBL.ProcessedImg, uchar, 255, "Segment", true);

  cv::Mat BWholes;
  if (Settings->fillHoles) {
    Vision::Segment Fillholes(SegBL.ProcessedImg);
    Fillholes.FillHoles();
    BWholes = Fillholes.ProcessedImg;
    emit on_progressUpdate(currentProgress++);
    SHOW_DEBUG_IMG(BWholes, uchar, 255, "Fillholes", true);
  } else {
    BWholes = SegBL.ProcessedImg;
  }

  cv::Mat BWborder;
  if (Settings->ignorePartialBorderParticles) {
    Vision::Segment RemoveBB(BWholes.clone());
    RemoveBB.RemoveBorderBlobs();
    BWborder = RemoveBB.ProcessedImg;
    emit on_progressUpdate(currentProgress++);
    SHOW_DEBUG_IMG(BWborder, uchar, 255, "RemoveBorderBlobs", true);
  } else {
    BWborder = BWholes;
  }

  if (Settings->morphFilterType != Vision::MorphologicalFilter::NONE) {
    Vision::MorphologicalFilter Morph(BWborder.clone());
    cv::Mat kernel = cv::Mat::zeros(Settings->filterMaskSize,
                                    Settings->filterMaskSize, CV_8UC1);
    uint32_t hMaskSize = Settings->filterMaskSize / 2;
    cv::circle(kernel, cv::Point(hMaskSize, hMaskSize), hMaskSize + 1, 1, -1);
    switch (Settings->morphFilterType) {
    case Vision::MorphologicalFilter::CLOSE:
      Morph.Close(kernel);
      break;
    case Vision::MorphologicalFilter::OPEN:
      Morph.Open(kernel);
      break;
    case Vision::MorphologicalFilter::DILATE:
      Morph.Dilation(kernel);
```

```
      break;
    case Vision::MorphologicalFilter::ERODE:
      Morph.Erosion(kernel);
      break;
    case Vision::MorphologicalFilter::NONE:
      Morph.ProcessedImg = Morph.OriginalImg;
      break;
    }
    BW = Morph.ProcessedImg;
    emit on_progressUpdate(currentProgress++);
    SHOW_DEBUG_IMG(BW, uchar, 255, "Morphological operation"
        , true);
  } else {
    BW = BWholes;
  }
}

/*!
 * \brief Analyzer::GetParticles
 * \param BW
 * \param snapshots
 * \param partPopulation
 */
void Analyzer::GetParticles(std::vector<Mat> &BW, Images_t *
    snapshots,
                            Particle::ParticleVector_t &
                                partPopulation) {
  for (uint32_t i = 0; i < snapshots->size(); i++) {
    Vision::Segment prepBW(BW[i]);
    prepBW.GetBlobList();
    emit on_progressUpdate(currentProgress++);
    GetParticlesFromBlobList(prepBW.BlobList, &(snapshots->
        at(i)),
                             partPopulation);
    emit on_progressUpdate(currentProgress++);
  }
}

/*!
 * \brief Analyzer::GetParticlesFromBlobList
 * \param bloblist
 * \param snapshot
 * \param edge
 * \param partPopulation
 */
void Analyzer::GetParticlesFromBlobList(
    Vision::Segment::BlobList_t &bloblist, Image_t *snapshot
        ,
    Particle::ParticleVector_t &partPopulation) {
  for_each(bloblist.begin(), bloblist.end(), [&](Vision::
      Segment::Blob_t &B) {
    Particle part;
    part.ID = currentParticleID++;
    part.PixelArea = B.Area;
    Vision::Segment::getOrientented(B.Img, B.Centroid, B.
        Theta,
```

```
                                        part.Eccentricty);
    cv::Mat RGB = Vision::Segment::CopyMat<uchar>(snapshot->
        FrontLight(B.ROI),
                                                      B.Img,
                                                          CV_8UC3
                                                          ).clone
                                                          ();
    cv::Rect ROI;
    Vision::Segment::RotateImg(B.Img, part.BW, B.Theta, B.
        Centroid, ROI);
    Vision::Segment::RotateImg(RGB, part.RGB, B.Theta, B.
        Centroid, ROI);
    Vision::Segment edgeSeg(part.BW);
    edgeSeg.GetEdgesEroding();
    part.Edge = edgeSeg.ProcessedImg.clone();
    part.SIPixelFactor = snapshot->SIPixelFactor;
    part.isPreparedForAnalysis = false;
    part.SetRoundness();
    partPopulation.push_back(part);
  });
}

/*!
 * \brief Analyzer::GetFFD
 * \param particalPopulation
 */
void Analyzer::GetFFD(Particle::ParticleVector_t &
    particalPopulation) {
  //for_each(particalPopulation.begin(), particalPopulation.
      end(),  [&](Particle &P) {
  QtConcurrent::blockingMap<Particle::ParticleVector_t>(
      particalPopulation, [&](Particle &P) {
        if (!P.isPreparedForAnalysis) {
          try {
            SoilMath::FFT fft;
            P.FFDescriptors = fft.GetDescriptors(P.Edge);
            P.isPreparedForAnalysis = true;
          } catch (SoilMath::Exception::MathException &e) {
            if (*e.id() == EXCEPTION_NO_CONTOUR_FOUND_NR) {
              P.isSmall = true;
            }
          }
          emit on_progressUpdate(currentProgress++);
        }
      });
}

/*!
 * \brief Analyzer::GetPrediction
 * \param particlePopulation
 */
void Analyzer::GetPrediction(Particle::ParticleVector_t &
    particlePopulation) {
  for_each(particlePopulation.begin(), particlePopulation.
      end(),
           [&](Particle &P) {
```

```
            if (P.isPreparedForAnalysis) {
              if (!P.isSmall) {
                ComplexVect_t usedFFDescr(P.FFDescriptors.
                    begin(),
                                          P.FFDescriptors.
                                              begin() +
                                              NeuralNet.
                                                  GetInputNeurons
                                                  ());
                P.Classification = NeuralNet.Predict(
                    usedFFDescr);
                P.isAnalysed = true;
              }
            }
          });
}

float Analyzer::CalibrateSI(float SI, Mat &img) {
  Vision::Conversion greyConv(img);
  greyConv.Convert(Vision::Conversion::RGB, Vision::
      Conversion::Intensity);
  Vision::Segment segment(greyConv.ProcessedImg);
  segment.ConvertToBW(Vision::Segment::Dark);
  segment.GetBlobList(true);
  uint32_t maxCircle = 0;
  for_each(segment.BlobList.begin(), segment.BlobList.end(),
          [&](Vision::Segment::Blob_t &B) {
            if (B.ROI.height > maxCircle) {
              maxCircle = B.ROI.height;
            }
            if (B.ROI.width > maxCircle) {
              maxCircle = B.ROI.width;
            }
          });
  qDebug() << "Maximum circle in pixels: " << maxCircle;
  CurrentSIfactor = SI / maxCircle;
  qDebug() << "Current SI factor : " << CurrentSIfactor;
  return CurrentSIfactor;
}
}
```

F.0.2 Sample Class

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
   strictly prohibited
 * and only allowed with the written consent of the author (
   Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include "stdint.h"
#include <vector>
#include <string>
#include "Stats.h"
#include "psd.h"
#include "particle.h"
#include <fstream>
#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/version.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/iostreams/filter/zlib.hpp>
#include <boost/iostreams/filtering_streambuf.hpp>
#include "zlib.h"
#include "soilanalyzertypes.h"

namespace SoilAnalyzer {
class Sample {
public:
  Sample();

  uint32_t ID;           /*!< The sample ID*/
  std::string Location; /*!< The Location where the sample
      was taken*/
  double Longtitude = 4.629618299999947;
  double Latitude = 51.8849149;
  double Depth = 0;
  std::string Date = "01-09-2015";
  std::string Name; /*!< The sample name identifier*/

  Particle::ParticleVector_t
      ParticlePopulation; /*!< the individual particles of
          the sample*/

  SoilMath::PSD PSD; /*!< The Particle Size Distribution*/
  ucharStat_t Roundness;
  ucharStat_t Angularity;
  floatStat_t RI; /*!< The statistical Redness Index data*/

  void Save(const std::string &filename);
  void Load(const std::string &filename);

```

```
  Particle::PSDVector_t *GetPSDVector();
  Particle::ClassVector_t *GetRoundnessVector();
  Particle::ClassVector_t *GetAngularityVector();
  Particle::doubleVector_t *GetCIELab_aVector();
  Particle::doubleVector_t *GetCIELab_bVector();

  bool isPreparedForAnalysis =
      false; /*!< is the sample ready for analysis, are all
         the particles
                extracted*/
  bool isAnalysed = false; /*!< is the sample analyzed*/

  bool ChangesSinceLastSave = false;
  bool ParticleChangedStatePSD = false;
  bool ParticleChangedStateClass = false;
  bool ParticleChangedStateRoundness = false;
  bool ParticleChangedStateAngularity = false;
  bool ColorChange = false;

  bool IsLoadedFromDisk = false;

private:
  Particle::PSDVector_t Diameter; /*!< The PSD raw data*/
  bool PSDGathered = false;       /*!< is the raw data
     gathered*/
  Particle::ClassVector_t RoundnessVec;
  bool RoundnessGathered = false;
  Particle::ClassVector_t AngularityVec;
  bool AngularityGathered = false;
  Particle::doubleVector_t CIELab_aVec;
  bool CIELab_aGathered = false;
  Particle::doubleVector_t CIELab_bVec;
  bool CIELab_bGathered = false;

  friend class boost::serialization::access;
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {
    ar &ID;
    ar &Location;
    ar &Name;
    ar &ParticlePopulation;
    ar &Diameter;
    ar &RoundnessVec;
    ar &AngularityVec;
    ar &PSD;
    ar &Roundness;
    ar &Angularity;
    ar &RI;
    ar &isPreparedForAnalysis;
    ar &isAnalysed;
    ar &ChangesSinceLastSave;
    ar &ParticleChangedStatePSD;
    ar &ParticleChangedStateClass;
    ar &ParticleChangedStateAngularity;
    ar &ParticleChangedStateRoundness;
    ar &PSDGathered;
```

```
    ar &RoundnessGathered;
    ar &AngularityGathered;
    ar &IsLoadedFromDisk;
    if (version > 0) {
      ar &Longtitude;
      ar &Latitude;
      ar &Date;
      ar &Depth;
      ar &AngularityVec;
      ar &AngularityGathered;
      ar &CIELab_aVec;
      ar &CIELab_aGathered;
      ar &CIELab_bVec;
      ar &CIELab_bGathered;
      ar &ColorChange;
    } else {
      Latitude = 51.8849149;
      Longtitude = 4.629618299999947;
      Date = "01-10-2015";
      Depth = 0;
      CIELab_aVec = Particle::doubleVector_t();
      CIELab_aGathered = false;
      CIELab_bVec = Particle::doubleVector_t();
      CIELab_bGathered = false;
      ColorChange = false;
    }
  }
};
}
BOOST_CLASS_VERSION(SoilAnalyzer::Sample, 1)
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "sample.h"
#include "particle.h"

namespace SoilAnalyzer {
namespace io = boost::iostreams;

/*!
 * \brief Sample::Sample
 */
Sample::Sample() {}

/*!
 * \brief Sample::Save
 * \param filename
 */
void Sample::Save(const std::string &filename) {
```

```
  std::ofstream ofs(filename.c_str(), std::ios::out | std::ios::binary);
  {
    io::filtering_streambuf<io::output> out;

    out.push(io::zlib_compressor(io::zlib::best_compression));
    out.push(ofs);
    {
      boost::archive::binary_oarchive oa(out);
      oa << boost::serialization::make_nvp("Sample", *this);
    }
  }
  ofs.close();
}

/*!
 * \brief Sample::Load
 * \param filename
 */
void Sample::Load(const std::string &filename) {
  std::ifstream ifs(filename.c_str(), std::ios::in | std::ios::binary);
  {
    io::filtering_streambuf<io::input> in;

    in.push(io::zlib_decompressor());
    in.push(ifs);
    {
      boost::archive::binary_iarchive ia(in);
      ia >> boost::serialization::make_nvp("Sample", *this);
    }
  }
  ifs.close();
}

/*!
 * \brief Sample::GetPSDVector
 * \return
 */
Particle::PSDVector_t *Sample::GetPSDVector() {
  if (!PSDGathered || ParticleChangedStatePSD) {
    Diameter.clear();
    for_each(ParticlePopulation.begin(), ParticlePopulation.end(),
             [&](Particle &P) { Diameter.push_back(P.GetSiDiameter()); });
    PSDGathered = true;
    ParticleChangedStatePSD = false;
  }
  return &Diameter;
}

Particle::ClassVector_t *Sample::GetAngularityVector() {
  if (!AngularityGathered || ParticleChangedStateAngularity) {
```

```
    AngularityVec.clear();
    for_each(ParticlePopulation.begin(), ParticlePopulation.end(),
             [&](Particle &P) { AngularityVec.push_back(P.GetAngularity()); });
    AngularityGathered = true;
    ParticleChangedStateAngularity = false;
  }
  return &AngularityVec;
}

Particle::ClassVector_t *Sample::GetRoundnessVector() {
  if (!RoundnessGathered || ParticleChangedStateRoundness) {
    RoundnessVec.clear();
    for_each(ParticlePopulation.begin(), ParticlePopulation.end(),
             [&](Particle &P) { RoundnessVec.push_back(P.GetRoundness()); });
    RoundnessGathered = true;
    ParticleChangedStateRoundness = false;
  }
  return &RoundnessVec;
}

Particle::doubleVector_t *Sample::GetCIELab_aVector() {
  if (!CIELab_aGathered || ColorChange) {
    CIELab_aVec.clear();
    for_each(ParticlePopulation.begin(), ParticlePopulation.end(),
             [&](Particle &P) { CIELab_aVec.push_back(P.getMeanLab().a); });
    CIELab_aGathered = true;
  }
  return &CIELab_aVec;
}

Particle::doubleVector_t *Sample::GetCIELab_bVector() {
  if (!CIELab_bGathered || ColorChange) {
    CIELab_bVec.clear();
    for_each(ParticlePopulation.begin(), ParticlePopulation.end(),
             [&](Particle &P) { CIELab_bVec.push_back(P.getMeanLab().b); });
    CIELab_bGathered = true;
  }
  return &CIELab_bVec;
}
}
```

F.0.3 Particle Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */
#pragma once

#include <opencv2/core.hpp>
#include <stdint.h>
#include <vector>
#include "SoilMath.h"
#include <fstream>
#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/version.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/iostreams/filter/zlib.hpp>
#include <boost/iostreams/filtering_streambuf.hpp>
#include "zlib.h"
#include "soilanalyzerexception.h"
#include "lab_t_archive.h"
#include "soilanalyzertypes.h"
#include "Vision.h"

namespace SoilAnalyzer {
class Particle {
public:
  typedef std::vector<Particle>
      ParticleVector_t; /*!< a vector consisting of individual particles*/
  typedef std::vector<double> PSDVector_t; /*!< a vector used in the PSD*/
  typedef std::vector<uint8_t>
      ClassVector_t; /*!< a vector used in the classification histogram*/
  typedef std::vector<float> floatVector_t;
  typedef std::vector<double> doubleVector_t;

  Particle();

  uint32_t ID; /*!< The particle ID*/

  cv::Mat BW;   /*!< The binary image of the particle*/
  cv::Mat Edge; /*!< The binary edge image of the particle*/
  cv::Mat RGB;  /*!< The RGB image of the particle*/

  Point_t Centroid = {0, 0};
  std::vector<Complex_t> FFDescriptors; /*!< The Fast Fourier Descriptors
```

```
                                                    describing the contour in the
                                                    Frequency domain */
  Predict_t Classification;              /*!< The classification prediction*/
  double SIPixelFactor = 0.0111915; /*!< The conversion factor from pixel to SI*/
  uint32_t PixelArea = 0;           /*!< The total area of the binary image*/
  double Eccentricty = 1;

  float GetSIVolume();
  float GetSiDiameter();
  uint8_t GetRoundness();
  uint8_t GetAngularity();
  float GetMeanRI();
  Lab_t getMeanLab();

  void SetRoundness();

  void Save(const std::string &filename);
  void Load(const std::string &filename);

  bool isPreparedForAnalysis = false; /*!< is the particle ready for analysis*/
  bool isAnalysed = false;            /*!< is the particle analyzed*/
  bool isSmall = false;

private:
  float SIVolume = 0.; /*!< The correspondening SI volume*/
  float SIDiameter = 0.;

  float meanRI = 0;
  Lab_t meanLab{0,0,0};
  cv::Mat LAB;

  void getLabImg();

  friend class boost::serialization::access;
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {

    ar &ID;
    ar &BW;
    ar &Edge;
    ar &RGB;
    ar &FFDescriptors;
    ar &Classification;
    ar &SIPixelFactor;
    ar &PixelArea;
    ar &SIVolume;
    ar &isPreparedForAnalysis;
    ar &isAnalysed;
    if (version > 0) {
```

```
      ar &isSmall;
      ar &SIDiameter;
      ar &Centroid.x;
      ar &Centroid.y;
      ar &Eccentricty;
    } else {
      isSmall = false;
      SIDiameter = GetSiDiameter();
      Centroid.x = 0;
      Centroid.y = 0;
      Eccentricty = 1;
    }
    if (version > 1) {
        ar &meanLab;
        ar &meanRI;
      }
    else {
        meanLab.L = 0;
        meanLab.a = 0;
        meanLab.b = 0;
      }
  }
};
}
BOOST_CLASS_VERSION(SoilAnalyzer::Particle, 2)
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "particle.h"

namespace SoilAnalyzer {
namespace io = boost::iostreams;

Particle::Particle() {}

/*!
 * \brief Particle::Save
 * \param filename
 */
void Particle::Save(const std::string &filename) {
  std::ofstream ofs(filename.c_str(), std::ios::out | std::ios::binary);
  {
    io::filtering_streambuf<io::output> out;

    out.push(io::zlib_compressor(io::zlib::best_compression));
    out.push(ofs);
    {
```

```
      boost::archive::binary_oarchive oa(out);
      oa << boost::serialization::make_nvp("Particle", *this);
    }
  }
  ofs.close();
}

/*!
 * \brief Particle::Load
 * \param filename
 */
void Particle::Load(const std::string &filename) {
  std::ifstream ifs(filename.c_str(), std::ios::in | std::ios::binary);
  {
    io::filtering_streambuf<io::input> in;

    in.push(io::zlib_decompressor());
    in.push(ifs);
    {
      boost::archive::binary_iarchive ia(in);
      ia >> boost::serialization::make_nvp("Particle", *this);
    }
  }
  ifs.close();
}

/*!
 * \brief Particle::GetSIVolume
 * \return
 */
float Particle::GetSIVolume() {
  if (SIVolume == 0.) {
    if (PixelArea == 0) {
      throw Exception::SoilAnalyzerException(
          EXCEPTION_PARTICLE_NOT_ANALYZED,
          EXCEPTION_PARTICLE_NOT_ANALYZED_NR);
    }
    SIVolume = SoilMath::calcVolume(PixelArea) *
      SIPixelFactor * (Eccentricty/2 + 0.5);
  }
  return SIVolume;
}

float Particle::GetSiDiameter() {
  if (SIDiameter == 0.) {
    if (PixelArea == 0) {
      throw Exception::SoilAnalyzerException(
          EXCEPTION_PARTICLE_NOT_ANALYZED,
          EXCEPTION_PARTICLE_NOT_ANALYZED_NR);
    }
    SIDiameter = SoilMath::calcDiameter(PixelArea) *
      SIPixelFactor  * (Eccentricty/2 + 0.5);
  }
```

```
  return SIDiameter;
}

uint8_t Particle::GetAngularity() {
  uint8_t angularity = ((Classification.Category - 1) % 6) +
      1;
  return angularity;
}

uint8_t Particle::GetRoundness() {
  uint8_t roundness = ((Classification.Category - 1) / 6) +
      1;
  return roundness;
}

void Particle::SetRoundness() {
  uint8_t ang = GetAngularity() - 1;
  Classification.Category +=
      ang + (static_cast<uint8_t>(floor(Eccentricty / 0.33))
          * 6);
  Classification.ManualSet = true;
}

Lab_t Particle::getMeanLab() {
  if (BW.empty() || RGB.empty()) {
    throw SoilAnalyzer::Exception::SoilAnalyzerException(
        EXCEPTION_NO_IMAGES_PRESENT,
            EXCEPTION_NO_IMAGES_PRESENT_NR);
  }
  if (meanLab.L == 0 && meanLab.a == 0 && meanLab.b == 0) {
    // convert to Lab
    if (LAB.empty()) {
      getLabImg();
    }
    std::vector<cv::Mat> LABvect = Vision::Conversion::
        extractChannel(LAB);
    std::vector<float> labvect;
    for_each(LABvect.begin(), LABvect.end(), [&](cv::Mat &I)
        {
      floatStat_t labStat((float *)I.data, I.rows, I.cols, (
          uchar *)BW.data, 1,
                          0, true);
      labvect.push_back(labStat.Mean);
    });
    meanLab.L = labvect[0];
    meanLab.a = labvect[1];
    meanLab.b = labvect[2];
  }
  return meanLab;
}

float Particle::GetMeanRI() {
  if (BW.empty() || RGB.empty()) {
    throw SoilAnalyzer::Exception::SoilAnalyzerException(
        EXCEPTION_NO_IMAGES_PRESENT,
            EXCEPTION_NO_IMAGES_PRESENT_NR);
```

```
  }
  if (meanRI == 0) {
    if (LAB.empty()) {
      getLabImg();
    }
    Vision::Conversion convertor(LAB);
    convertor.Convert(Vision::Conversion::CIE_lab, Vision::Conversion::RI);
    floatStat_t RIstat((float *)convertor.ProcessedImg.data, LAB.rows, LAB.cols,
                       (uchar *)BW.data, 1, 0, true);
    meanRI = RIstat.Mean;
  }
  return meanRI;
}

void Particle::getLabImg() {
  Vision::Conversion convertor(RGB);
  convertor.Convert(Vision::Conversion::RGB, Vision::Conversion::CIE_lab);
  LAB = convertor.ProcessedImg.clone();
}
}
```

F.0.4 Settings Class

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <string>
#include <fstream>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/serialization/version.hpp>
#include "../SoilVision/Vision.h"

namespace SoilAnalyzer {
/*!
 * \brief The SoilSettings class
 * \details A class with which the used settings can easily transfered to setup
 * the Sample class in one go. This class is also used in the GUI. and has a
 * possibility to saved to disk as a serialized object
 */
class SoilSettings {
public:
  SoilSettings();

  /*!
   * \brief SaveSettings a function to save the settings to disk
   * \param filename a string with the filename
   */
  void SaveSettings(std::string filename);

  /*!
   * \brief LoadSettings a function to load the settings from disk
   * \param filename a string with the filename
   */
  void LoadSettings(std::string filename);

  bool useAdaptiveContrast =
      false; /**< Should adaptive contrast stretch be used default is true*/
  uint32_t adaptContrastKernelSize =
      9; /**< The size of the adaptive contrast kernelsize*/
  float adaptContrastKernelFactor = 1.; /**< the factor with which to multiply
                                             the effect of the adaptive
```

```
                                                          contrast
                                                       stretch*/

    bool useBlur = false; /**< Should the mediaan blur be used
        during analsyis*/
    uint32_t blurKernelSize = 5; /**< the median blurkernel*/

    Vision::Segment::TypeOfObjects typeOfObjectsSegmented =
        Vision::Segment::Dark; /**< Which type of object
            should be segmented*/
    bool ignorePartialBorderParticles =
        true; /**< Indication of partial border particles
            should be used*/
    bool fillHoles = true; /**< should the holes be filled*/
    float sigmaFactor = 2; /**< The sigma factor or the
       bandwidth indicating which
                                  pixel intensity values count
                                     belong to an object*/
    int thresholdOffsetValue = 0; /**< an tweaking offset
       value*/

    Vision::MorphologicalFilter::FilterType morphFilterType =
        Vision::MorphologicalFilter::OPEN; /**< Indicating
            which type of
                                                       morhpological
                                                          filter should
                                                           be
                                                       used*/
    uint32_t filterMaskSize = 5;           /**< the filter
       mask*/

    uint32_t HDRframes =
        5; /**< The number of frames which should be used for
            the HDR image*/
    float lightLevel = 0.5; /**< The light level of the
       environmental case*/
    bool encInv = false;    /**< invert the values gained form
        the encoder*/
    bool enableRainbow =
        true; /**< run a rainbow loop on the RGB encoder
            during analysis*/
    bool useBacklightProjection = true;          /*!< use
       Projection*/
    bool useHDR = false;                         /*!< use HDR
       */
    std::string defaultWebcam = "USB Microscope"; /*!< The
       defaultWebcam string*/
    int Brightness_front = 0;   /*!< cam brightness setting
       front light*/
    int Brightness_proj = -10; /*!< cam brightness setting
       projected light*/
    int Contrast_front = 36;   /*!< cam contrast setting front
        light*/
    int Contrast_proj = 36;    /*!< cam contrast setting
       projected light*/
```

```
  int Saturation_front = 64; /*!< cam saturation setting front light*/
  int Saturation_proj = 0;   /*!< cam saturation setting projected light*/
  int Hue_front = 0;         /*!< cam hue setting front light*/
  int Hue_proj = -40;        /*!< cam hue setting projected light*/
  int Gamma_front = 100;     /*!< cam gamma setting front light*/
  int Gamma_proj = 200;      /*!< cam gamma setting projected light*/
  int PowerLineFrequency_front =
      1; /*!< cam powerline freq setting front light*/
  int PowerLineFrequency_proj =
      1;                    /*!< cam powerline freq setting projected light*/
  int Sharpness_front = 12; /*!< cam sharpness setting front light*/
  int Sharpness_proj = 25;  /*!< cam sharpness setting projected light*/
  int BackLightCompensation_front =
      1; /*!< cam backlight compensation setting front light */
  int BackLightCompensation_proj =
      1; /*!< cam backlight compensation setting projected light*/
  std::string NNlocation = "NeuralNet/Default.NN";
  bool useCUDA = false; /*!< CUDA enabled*/
  int selectedResolution = 0;
  std::string SampleFolder = "~/Samples";
  std::string SettingsFolder = "Settings";
  std::string NNFolder = "NeuralNet";
  std::string StandardSentTo = "j.spijker@ihcmerwede.com";
  std::string StandardPrinter = "PDF printer";
  uint32_t StandardNumberOfShots = 10;
  bool PredictTheShape = true;
  bool Revolution = true;
private:
  friend class boost::serialization::access;
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {
    if (version >= 0) {
      ar &BOOST_SERIALIZATION_NVP(useAdaptiveContrast);
      ar &BOOST_SERIALIZATION_NVP(adaptContrastKernelFactor);
      ar &BOOST_SERIALIZATION_NVP(adaptContrastKernelSize);
      ar &BOOST_SERIALIZATION_NVP(useBlur);
      ar &BOOST_SERIALIZATION_NVP(blurKernelSize);
      ar &BOOST_SERIALIZATION_NVP(typeOfObjectsSegmented);
      ar &BOOST_SERIALIZATION_NVP(
          ignorePartialBorderParticles);
      ar &BOOST_SERIALIZATION_NVP(fillHoles);
      ar &BOOST_SERIALIZATION_NVP(sigmaFactor);
      ar &BOOST_SERIALIZATION_NVP(morphFilterType);
      ar &BOOST_SERIALIZATION_NVP(filterMaskSize);
```

```
        ar &BOOST_SERIALIZATION_NVP(thresholdOffsetValue);
        ar &BOOST_SERIALIZATION_NVP(HDRframes);
        ar &BOOST_SERIALIZATION_NVP(lightLevel);
        ar &BOOST_SERIALIZATION_NVP(encInv);
        ar &BOOST_SERIALIZATION_NVP(enableRainbow);
        ar &BOOST_SERIALIZATION_NVP(useBacklightProjection);
        ar &BOOST_SERIALIZATION_NVP(useHDR);
        ar &BOOST_SERIALIZATION_NVP(defaultWebcam);
        ar &BOOST_SERIALIZATION_NVP(Brightness_front);
        ar &BOOST_SERIALIZATION_NVP(Brightness_proj);
        ar &BOOST_SERIALIZATION_NVP(Contrast_front);
        ar &BOOST_SERIALIZATION_NVP(Contrast_proj);
        ar &BOOST_SERIALIZATION_NVP(Saturation_front);
        ar &BOOST_SERIALIZATION_NVP(Saturation_proj);
        ar &BOOST_SERIALIZATION_NVP(Hue_front);
        ar &BOOST_SERIALIZATION_NVP(Hue_proj);
        ar &BOOST_SERIALIZATION_NVP(Gamma_front);
        ar &BOOST_SERIALIZATION_NVP(Gamma_proj);
        ar &BOOST_SERIALIZATION_NVP(PowerLineFrequency_front);
        ar &BOOST_SERIALIZATION_NVP(PowerLineFrequency_proj);
        ar &BOOST_SERIALIZATION_NVP(Sharpness_front);
        ar &BOOST_SERIALIZATION_NVP(Sharpness_proj);
        ar &BOOST_SERIALIZATION_NVP(
            BackLightCompensation_front);
        ar &BOOST_SERIALIZATION_NVP(BackLightCompensation_proj
            );
        ar &BOOST_SERIALIZATION_NVP(NNlocation);
        ar &BOOST_SERIALIZATION_NVP(useCUDA);
        ar &BOOST_SERIALIZATION_NVP(selectedResolution);
        ar &BOOST_SERIALIZATION_NVP(SampleFolder);
        ar &BOOST_SERIALIZATION_NVP(SettingsFolder);
        ar &BOOST_SERIALIZATION_NVP(NNFolder);
        ar &BOOST_SERIALIZATION_NVP(StandardSentTo);
        ar &BOOST_SERIALIZATION_NVP(StandardPrinter);
        ar &BOOST_SERIALIZATION_NVP(StandardNumberOfShots);
        ar &BOOST_SERIALIZATION_NVP(PredictTheShape);
        ar &BOOST_SERIALIZATION_NVP(Revolution);
      }
    }
};
}
BOOST_CLASS_VERSION(SoilAnalyzer::SoilSettings, 0)
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "soilsettings.h"

namespace SoilAnalyzer {
SoilSettings::SoilSettings() {}
```

```

void SoilSettings::LoadSettings(string filename) {
  std::ifstream ifs(filename.c_str());
  boost::archive::xml_iarchive ia(ifs);
  ia >> boost::serialization::make_nvp("SoilSettings", *this);
}

void SoilSettings::SaveSettings(string filename) {
  std::ofstream ofs(filename.c_str());
  boost::archive::xml_oarchive oa(ofs);
  oa << boost::serialization::make_nvp("SoilSettings", *this);
}
}
```

F.0.5 General project files

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-08-08T18:57:27
#
#-------------------------------------------------

QT       += core gui concurrent
QMAKE_CXXFLAGS += -std=c++11

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
@
CONFIG(release, debug|release):DEFINES += QT_NO_DEBUG_OUTPUT
@

TARGET = SoilAnalyzer
TEMPLATE = lib
VERSION = 0.9.96

DEFINES += SOILANALYZER_LIBRARY

SOURCES += \
    soilsettings.cpp \
    sample.cpp \
    particle.cpp \
    analyzer.cpp

HEADERS +=\
    soilsettings.h \
    sample.h \
    particle.h \
    analyzer.h \
    soilanalyzerexception.h \
    soilanalyzer.h \
    lab_t_archive.h \
    soilanalyzertypes.h

#opencv
LIBS += -L/usr/local/lib -lopencv_core -lopencv_highgui
INCLUDEPATH += /usr/local/include/opencv
INCLUDEPATH += /usr/local/include

#boost
DEFINES += BOOST_ALL_DYN_LINK
INCLUDEPATH += /usr/include/boost
LIBS += -L/usr/lib/x86_64-linux-gnu/ -lboost_serialization -
    lboost_iostreams

#Zlib
LIBS += -L/usr/local/lib -lz
INCLUDEPATH += /usr/local/include

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilMath
INCLUDEPATH += $$PWD/../SoilMath
DEPENDPATH += $$PWD/../SoilMath
```

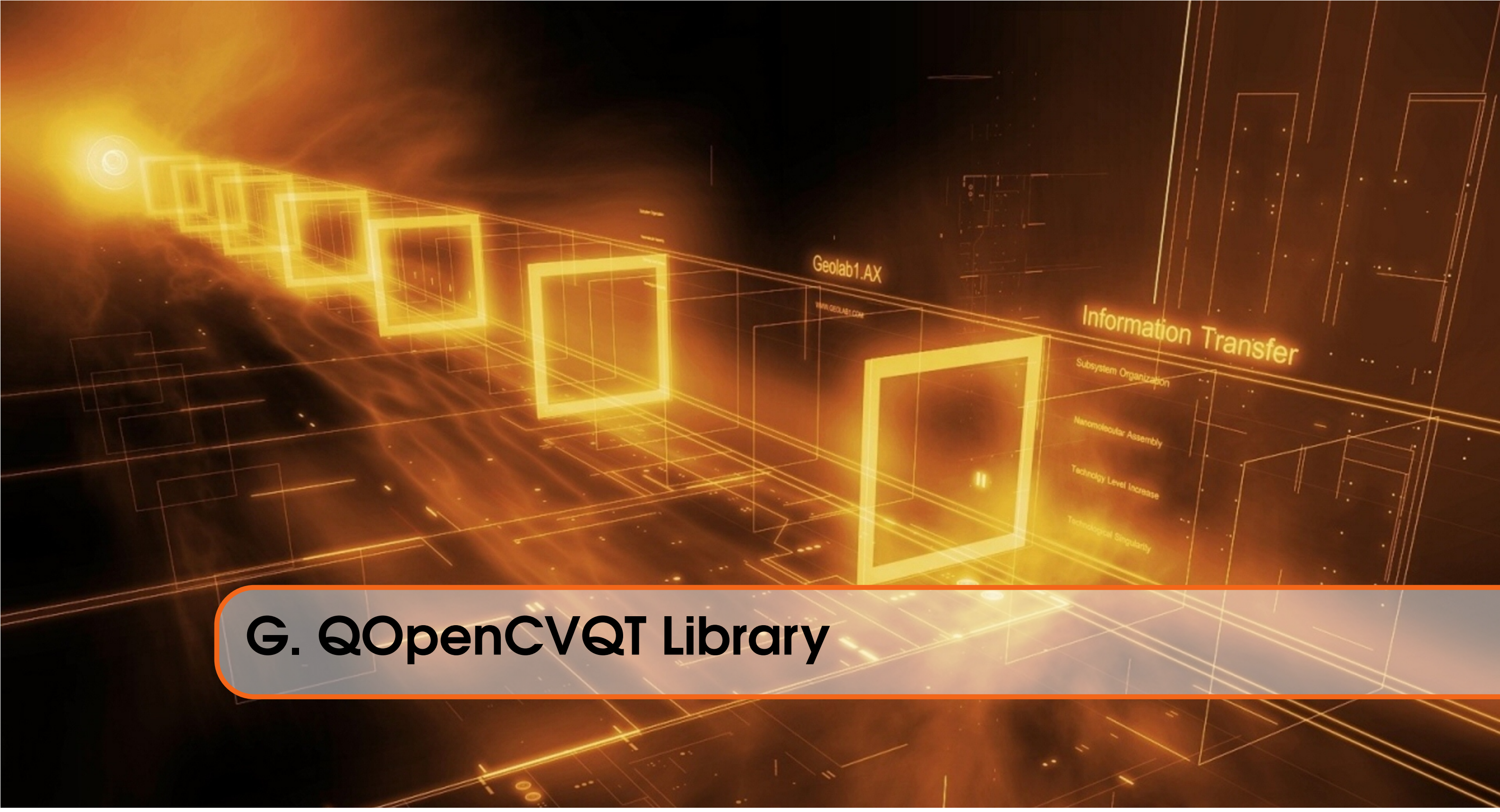
```

unix:!macx: LIBS += -L$$PWD/../../build/install/ -
    lSoilVision
INCLUDEPATH += $$PWD/../SoilVision
DEPENDPATH += $$PWD/../SoilVision

#MainLib

target.path = $PWD/../../../build/install
INSTALLS += target
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/access.hpp>
#include "soilanalyzertypes.h"

namespace boost {
namespace serialization {
/*!
 * \brief serialize Serialize the openCV mat to disk
 */
template <class Archive>
inline void serialize(Archive &ar, SoilAnalyzer::Lab_t &P,
    const unsigned int version __attribute__((unused))) {
  ar &P.L;
  ar &P.a;
  ar &P.b;
}
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
    strictly prohibited
 * and only allowed with the written consent of the author (
    Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */
#define EXCEPTION_PARTICLE_NOT_ANALYZED "Particle not
    analyzed Exception!"
#define EXCEPTION_PARTICLE_NOT_ANALYZED_NR 0
#define EXCEPTION_NO_SNAPSHOTS "No snapshots Exception!"
#define EXCEPTION_NO_SNAPSHOTS_NR 1
```

```
#define EXCEPTION_NO_IMAGES_PRESENT "No images to analyse
    exception!"
#define EXCEPTION_NO_IMAGES_PRESENT_NR 2

#pragma once
#include <exception>
#include <string>

namespace SoilAnalyzer {
  namespace Exception {
    class SoilAnalyzerException : public std::exception {
    public:
      SoilAnalyzerException(std::string m =
          EXCEPTION_PARTICLE_NOT_ANALYZED,
                            int n =
                                EXCEPTION_PARTICLE_NOT_ANALYZED_NR
                                ) : msg(m), nr(n) { }
      ~SoilAnalyzerException() _GLIBCXX_USE_NOEXCEPT {}
      const char *what() const _GLIBCXX_USE_NOEXCEPT {
          return msg.c_str(); }
      const int *id() const _GLIBCXX_USE_NOEXCEPT { return &
          nr; }

    private:
      std::string msg;
      int nr;
    };
  }
}
```

```
#ifndef SOILANALYZERTYPES
#define SOILANALYZERTYPES

namespace SoilAnalyzer {
  struct Point_t {
    double x;
    double y;
  };

  struct Lab_t {
    float L;
    float a;
    float b;
  };
}
#endif // SOILANALYZERTYPES
```



G. QOpenCVQT Library

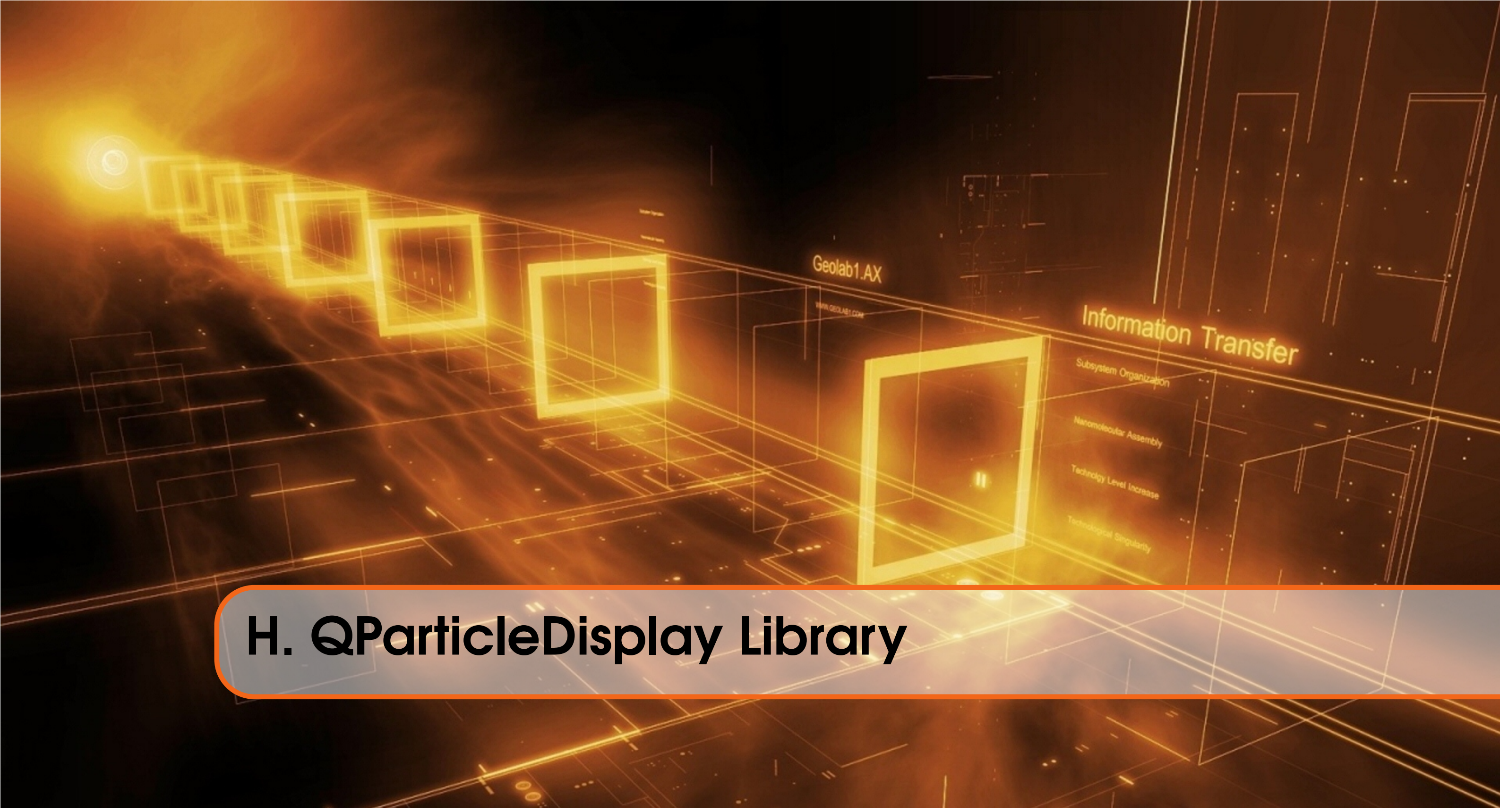
```
#----------------------------------------------------
#
# Project created by QtCreator 2015-08-08T08:11:34
#
#----------------------------------------------------

TARGET = QOpenCVQT
TEMPLATE = lib

QT += gui

DEFINES += QOPENCVQT_LIBRARY
VERSION = 1.1.0
CONFIG += shared

SOURCES += qopencvqt.cpp

HEADERS += qopencvqt.h

#opencv
LIBS += -L/usr/local/lib -lopencv_core
INCLUDEPATH += /usr/local/include/opencv
INCLUDEPATH += /usr/local/include

#MainLib
unix {
    target.path = $PWD/../../../build/install
    INSTALLS += target
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is
```

```
    strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#ifndef QOPENCVQT_H
#define QOPENCVQT_H

#include <QImage>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <vector>

class QOpenCVQT
{
public:
  QOpenCVQT();
  static cv::Mat WhiteBackground(const cv::Mat &src) {
    cv::Mat dst;
    cv::floodFill(src, dst, cv::Point(1,1), cv::Scalar_<uchar>(255,255,255));
    return dst;
  }

  static QImage Mat2QImage(const cv::Mat &src) {
    QImage dest;
    if (src.channels() == 1) {
      cv::Mat destRGB;
      std::vector<cv::Mat> grayRGB(3, src);
      cv::merge(grayRGB, destRGB);
      dest = QImage((uchar *)destRGB.data, destRGB.cols, destRGB.rows,
                    destRGB.step, QImage::Format_RGB888);
    } else {
      dest = QImage((uchar *)src.data, src.cols, src.rows, src.step,
                    QImage::Format_RGB888);
      dest = dest.rgbSwapped();
    }
    return dest;
  }
};

#endif // QOPENCVQT_H
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

```

```
#include "qopencvqt.h"


QOpenCVQT::QOpenCVQT()
{
}
```



H. QParticleDisplay Library

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-08-07T22:02:49
#
#-------------------------------------------------

QT       += core gui concurrent
QMAKE_CXXFLAGS += -std=c++11

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = QParticleDisplay
TEMPLATE = lib
CONFIG += shared
VERSION = 1.3.25

SOURCES += qparticledisplay.cpp

HEADERS  += qparticledisplay.h

FORMS    += qparticledisplay.ui

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lpictureflow-qt

INCLUDEPATH += $$PWD/../pictureflow-qt
DEPENDPATH += $$PWD/../pictureflow-qt

#MainLib
unix {
    target.path = $PWD/../../../build/install
    INSTALLS += target
```

```
}

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilAnalyzer

INCLUDEPATH += $$PWD/../SoilAnalyzer
DEPENDPATH += $$PWD/../SoilAnalyzer

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilMath

INCLUDEPATH += $$PWD/../SoilMath
DEPENDPATH += $$PWD/../SoilMath

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lQOpenCVQT

INCLUDEPATH += $$PWD/../QOpenCVQT
DEPENDPATH += $$PWD/../QOpenCVQT

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilVision
INCLUDEPATH += $$PWD/../SoilVision
DEPENDPATH += $$PWD/../SoilVision
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#include <QWidget>
#include <QImage>
#include <qopencvqt.h>
#include <QColor>
#include <QWheelEvent>

#include "soilanalyzer.h"

namespace Ui {
  class QParticleDisplay;
}

class QParticleDisplay : public QWidget
{
  Q_OBJECT

public:
  explicit QParticleDisplay(QWidget *parent = 0);
  ~QParticleDisplay();
  void SetSample(SoilAnalyzer::Sample *sample);
  SoilAnalyzer::Particle *SelectedParticle;
  void wheelEvent( QWheelEvent * event );
  void next();
```

```

signals:
  void particleChanged(int newValue);
  void shapeClassificationChanged(int newValue);
  void particleDeleted();

public slots:
  void setSelectedParticle(int newValue);

private slots:
  void on_selectedParticleChangedWidget(int value);
  void on_selectedParticleChangedSlider(int value);
  void on_pushButton_delete_clicked();

private:
  Ui::QParticleDisplay *ui;
  SoilAnalyzer::Sample *Sample;
  QVector<QImage> images;
  QImage ConvertParticleToQImage(SoilAnalyzer::Particle *particle);
  bool dontDoIt = false;
};
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "qparticledisplay.h"
#include "ui_qparticledisplay.h"

QParticleDisplay::QParticleDisplay(QWidget *parent)
    : QWidget(parent), ui(new Ui::QParticleDisplay) {
  ui->setupUi(this);
  ui->widget->setBackgroundColor(QColor("white"));
  ui->widget->setSlideSize(QSize(230, 230));
  connect(ui->widget, SIGNAL(centerIndexChanged(int)), this,
          SLOT(on_selectedParticleChangedWidget(int)));
  connect(ui->horizontalSlider, SIGNAL(valueChanged(int)), this,
          SLOT(on_selectedParticleChangedSlider(int)));
}

QParticleDisplay::~QParticleDisplay() {
  for (uint32_t i = 0; i < ui->widget->slideCount(); i++) {
    ui->widget->removeSlide(0);
  }
  delete ui->widget;
  delete ui;
}

void QParticleDisplay::setSelectedParticle(int newValue) {
```

```
  ui->widget->setCenterIndex(newValue);
  ui->horizontalSlider->setValue(newValue);
}

void QParticleDisplay::SetSample(SoilAnalyzer::Sample *
    sample) {
  this->Sample = sample;
  images.clear();
  ui->widget->clear();
  ui->horizontalSlider->setMaximum(this->Sample->
      ParticlePopulation.size() - 1);
  for (uint32_t i = 0; i < this->Sample->ParticlePopulation.
      size(); i++) {
    images.push_back(
        ConvertParticleToQImage(&Sample->ParticlePopulation.
            at(i)));
    ui->widget->addSlide(images[images.size() - 1]);
  }
  SelectedParticle = &Sample->ParticlePopulation[ui->widget
      ->centerIndex()];
  on_selectedParticleChangedSlider(0);
}

QImage
QParticleDisplay::ConvertParticleToQImage(SoilAnalyzer::
    Particle *particle) {
  QImage dst(particle->BW.cols + 10, particle->BW.rows + 10,
             QImage::Format_RGB32);
  uint32_t nData = particle->BW.cols * particle->BW.rows;
  uint32_t sData = ((dst.width() - 1) * 5) + 5;
  uchar *QDst = dst.bits();
  uchar *CVBW = particle->BW.data;
  uchar *CVRGB = particle->RGB.data;
  for (uint32_t i = 0; i < sData; i++) {
    *(QDst++) = 255;
    *(QDst++) = 255;
    *(QDst++) = 255;
    *(QDst++) = 0;
  }
  for (uint32_t i = 0; i < nData; i++) {
    if ((i % particle->BW.cols) == 0) {
      for (uint32_t j = 0; j < 10; j++) {
        *(QDst++) = 255;
        *(QDst++) = 255;
        *(QDst++) = 255;
        *(QDst++) = 0;
      }
    }
    if (CVBW[i]) {
      *(QDst++) = *(CVRGB);
      *(QDst++) = *(CVRGB + 1);
      *(QDst++) = *(CVRGB + 2);
      *(QDst++) = 0;
      CVRGB += 3;
    } else {
      *(QDst++) = 255;
```

```
      *(QDst++) = 255;
      *(QDst++) = 255;
      *(QDst++) = 0;
      CVRGB += 3;
    }
  }
  for (uint32_t i = 0; i < sData; i++) {
    *(QDst++) = 255;
    *(QDst++) = 255;
    *(QDst++) = 255;
    *(QDst++) = 0;
  }
  return dst;
}

void QParticleDisplay::on_pushButton_delete_clicked() {
  Sample->ParticlePopulation.erase(Sample->
      ParticlePopulation.begin() +
                                        ui->widget->centerIndex()
                                            );
  ui->widget->removeSlide(ui->widget->centerIndex());
  ui->horizontalSlider->setMaximum(this->Sample->
      ParticlePopulation.size() - 1);
  Sample->ParticleChangedStatePSD = true;
  Sample->ParticleChangedStateAngularity = true;
  Sample->ParticleChangedStateRoundness = true;
  Sample->ChangesSinceLastSave = true;
  Sample->ColorChange = true;
  SelectedParticle = &Sample->ParticlePopulation[ui->widget
      ->centerIndex()];
  emit particleDeleted();
}

void QParticleDisplay::on_selectedParticleChangedWidget(int
    value) {
  if (!dontDoIt) {
    dontDoIt = true;
    ui->horizontalSlider->setValue(value);
    SelectedParticle = &Sample->ParticlePopulation[ui->
        widget->centerIndex()];
    QString volume;
    volume.sprintf("%+06.2f", SelectedParticle->
        GetSiDiameter());
    ui->label_Volume->setText(volume);
    emit particleChanged(value);
    emit shapeClassificationChanged(SelectedParticle->
        Classification.Category);
    dontDoIt = false;
  }
}

void QParticleDisplay::on_selectedParticleChangedSlider(int
    value) {
  if (!dontDoIt) {
    dontDoIt = true;
    ui->widget->setCenterIndex(value);
```

```
      SelectedParticle = &Sample->ParticlePopulation[ui->
         widget->centerIndex()];
      QString volume;
      volume.sprintf("%+06.2f", SelectedParticle->
         GetSiDiameter());
      ui->label_Volume->setText(volume);
      emit particleChanged(value);
      emit shapeClassificationChanged(SelectedParticle->
         Classification.Category);
      dontDoIt = false;
    }
}

void QParticleDisplay::wheelEvent(QWheelEvent *event) {
  int i = ui->widget->centerIndex();
  i -= event->delta() / 120;
  if (i < 0) {
    i = ui->widget->slideCount() - abs(i) - 1;
  } else if (i >= ui->widget->slideCount()) {
    i = 0;
  }
  ui->widget->setCenterIndex(i);
  on_selectedParticleChangedWidget(i);
}

void QParticleDisplay::next() {
  int i = ui->widget->centerIndex();
  i++;
  if (i < 0) {
    i = ui->widget->slideCount() - abs(i) - 1;
  } else if (i >= ui->widget->slideCount()) {
    i = 0;
  }
  ui->widget->setCenterIndex(i);
  on_selectedParticleChangedWidget(i);
}
```

I. QParticleSelector Library

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-08-07T18:56:27
#
#-------------------------------------------------

QT       += core gui
QMAKE_CXXFLAGS += -std=c++11

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = QParticleSelector
TEMPLATE = lib
CONFIG += shared
VERSION = 0.1.11

SOURCES += qparticleselector.cpp

HEADERS  += qparticleselector.h

FORMS    += qparticleselector.ui

RESOURCES += \
    qparticleselector.qrc

#MainLib
unix {
    target.path = $PWD/../../../build/install
    INSTALLS += target
}

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilMath
```

```

INCLUDEPATH += $$PWD/../SoilMath
DEPENDPATH += $$PWD/../SoilMath
```

```
#ifndef QPARTICLESELECTOR_H
#define QPARTICLESELECTOR_H

#include <QWidget>
#include <QPushButton>

namespace Ui {
  class QParticleSelector;
}

class QParticleSelector : public QWidget
{
  Q_OBJECT

public:
  explicit QParticleSelector(QWidget *parent = 0);
  ~QParticleSelector();

  void setDisabled(bool value, int currentClass = 1);

signals:
  void valueChanged(int newValue);

public slots:
  void setValue(int newValue);

private slots:
  void on_pb_1_clicked(bool checked);

  void on_pb_2_clicked(bool checked);

  void on_pb_3_clicked(bool checked);

  void on_pb_4_clicked(bool checked);

  void on_pb_5_clicked(bool checked);

  void on_pb_6_clicked(bool checked);

  void on_pb_7_clicked(bool checked);

  void on_pb_8_clicked(bool checked);

  void on_pb_9_clicked(bool checked);

  void on_pb_10_clicked(bool checked);

  void on_pb_11_clicked(bool checked);

  void on_pb_12_clicked(bool checked);

  void on_pb_13_clicked(bool checked);
```

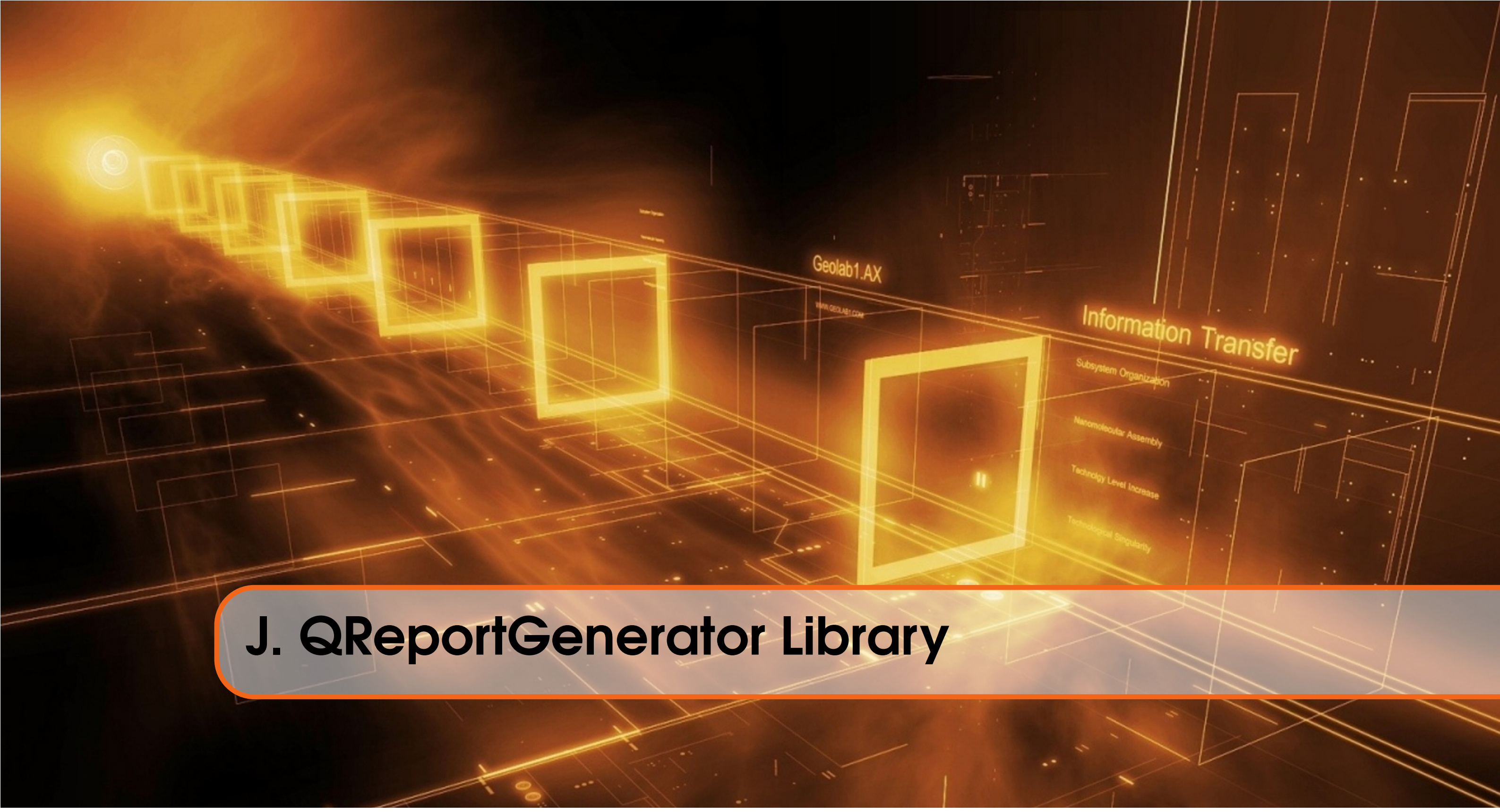
```

  void on_pb_14_clicked(bool checked);

  void on_pb_15_clicked(bool checked);

  void on_pb_16_clicked(bool checked);

  void on_pb_17_clicked(bool checked);

  void on_pb_18_clicked(bool checked);

private:
  QVector<QPushButton *> btns;
  Ui::QParticleSelector *ui;
};

#endif // QPARTICLESELECTOR_H
```

```
#include "qparticleselector.h"
#include "ui_qparticleselector.h"

QParticleSelector::QParticleSelector(QWidget *parent)
    : QWidget(parent), ui(new Ui::QParticleSelector) {
  ui->setupUi(this);
  btns.push_back(ui->pb_1);
  btns.push_back(ui->pb_2);
  btns.push_back(ui->pb_3);
  btns.push_back(ui->pb_4);
  btns.push_back(ui->pb_5);
  btns.push_back(ui->pb_6);
  btns.push_back(ui->pb_7);
  btns.push_back(ui->pb_8);
  btns.push_back(ui->pb_9);
  btns.push_back(ui->pb_10);
  btns.push_back(ui->pb_11);
  btns.push_back(ui->pb_12);
  btns.push_back(ui->pb_13);
  btns.push_back(ui->pb_14);
  btns.push_back(ui->pb_15);
  btns.push_back(ui->pb_16);
  btns.push_back(ui->pb_17);
  btns.push_back(ui->pb_18);
}

QParticleSelector::~QParticleSelector() {
  for (auto b : btns) {
    delete b;
  }
  btns.clear();
  delete ui;
}

void QParticleSelector::setValue(int newValue) {
  btns[newValue - 1]->setChecked(true);
}

```

```
void QParticleSelector::setDisabled(bool value, int currentClass) {
  for (auto b : btns) {
    b->setDisabled(value);
  }
  if (currentClass > 18 || currentClass < 1) {
    btns[0]->setChecked(true);
  } else {
    btns[currentClass - 1]->setChecked(true);
  }
}

void QParticleSelector::on_pb_1_clicked(bool checked) {
  if (checked) {
    emit valueChanged(1);
  }
}

void QParticleSelector::on_pb_2_clicked(bool checked) {
  if (checked) {
    emit valueChanged(2);
  }
}

void QParticleSelector::on_pb_3_clicked(bool checked) {
  if (checked) {
    emit valueChanged(3);
  }
}

void QParticleSelector::on_pb_4_clicked(bool checked) {
  if (checked) {
    emit valueChanged(4);
  }
}

void QParticleSelector::on_pb_5_clicked(bool checked) {
  if (checked) {
    emit valueChanged(5);
  }
}

void QParticleSelector::on_pb_6_clicked(bool checked) {
  if (checked) {
    emit valueChanged(6);
  }
}

void QParticleSelector::on_pb_7_clicked(bool checked) {
  if (checked) {
    emit valueChanged(7);
  }
}

void QParticleSelector::on_pb_8_clicked(bool checked) {
  if (checked) {
```

```
    emit valueChanged(8);
  }
}

void QParticleSelector::on_pb_9_clicked(bool checked) {
  if (checked) {
    emit valueChanged(9);
  }
}

void QParticleSelector::on_pb_10_clicked(bool checked) {
  if (checked) {
    emit valueChanged(10);
  }
}

void QParticleSelector::on_pb_11_clicked(bool checked) {
  if (checked) {
    emit valueChanged(11);
  }
}

void QParticleSelector::on_pb_12_clicked(bool checked) {
  if (checked) {
    emit valueChanged(12);
  }
}

void QParticleSelector::on_pb_13_clicked(bool checked) {
  if (checked) {
    emit valueChanged(13);
  }
}

void QParticleSelector::on_pb_14_clicked(bool checked) {
  if (checked) {
    emit valueChanged(14);
  }
}

void QParticleSelector::on_pb_15_clicked(bool checked) {
  if (checked) {
    emit valueChanged(15);
  }
}

void QParticleSelector::on_pb_16_clicked(bool checked) {
  if (checked) {
    emit valueChanged(16);
  }
}

void QParticleSelector::on_pb_17_clicked(bool checked) {
  if (checked) {
    emit valueChanged(17);
  }
```

```
}

void QParticleSelector::on_pb_18_clicked(bool checked) {
  if (checked) {
    emit valueChanged(18);
  }
}
```



J. QReportGenerator Library

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-08-20T08:46:42
#
#-------------------------------------------------

QT       += core gui concurrent network
QMAKE_CXXFLAGS += -std=c++11

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets printsupport multimedia multimediawidgets

@
CONFIG(release, debug|release):DEFINES += QT_NO_DEBUG_OUTPUT
@

unix:!macx: QMAKE_RPATHDIR += $$PWD/../../../build/install/

TARGET = QReportGenerator
TEMPLATE = lib
CONFIG += shared
VERSION = 0.1.00

SOURCES += \
    qreportgenerator.cpp \
    ../qcustomplot/examples/text-document-integration/qcpdocumentobject.cpp

HEADERS  += \
    qreportgenerator.h \
    ../qcustomplot/examples/text-document-integration/qcpdocumentobject.h
```

```

FORMS     += \
    qreportgenerator.ui

#MainLib
unix {
    target.path = $PWD/../../../build/install
    INSTALLS += target
}

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilMath
INCLUDEPATH += $$PWD/../SoilMath
DEPENDPATH += $$PWD/../SoilMath

DEFINES += QCUSTOMPLOT_USE_LIBRARY
unix:!macx: LIBS += -L$$PWD/../../build/install/ -lqcustomplot

INCLUDEPATH += $$PWD/../qcustomplot
DEPENDPATH += $$PWD/../qcustomplot

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilAnalyzer
INCLUDEPATH += $$PWD/../SoilAnalyzer
DEPENDPATH += $$PWD/../SoilAnalyzer

unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilVision
INCLUDEPATH += $$PWD/../SoilVision
DEPENDPATH += $$PWD/../SoilVision

RESOURCES += \
    qreportresources.qrc \
    ../VSA/vsa_resources.qrc

#maps
Mapstarget.path += $${OUT_PWD}/Maps
Mapstarget.files += $${PWD}/Maps/*
INSTALLS += Mapstarget
bMapstarget.path += $${PWD}/../../build/install/Maps
bMapstarget.files += $${PWD}/Maps/*
INSTALLS += bMapstarget
```

```
#ifndef QREPORTGENERATOR_H
#define QREPORTGENERATOR_H

#include <QMainWindow>
#include <QTextDocument>
#include <QDebug>
#include <QTextBlockFormat>
#include <QTextCharFormat>
#include <QTextBlock>
#include <QNetworkAccessManager>
#include <QNetworkReply>
#include <QTextDocumentWriter>
#include <QPrinter>
```

```
#include "soilanalyzer.h"
#include "SoilMath.h"

#include <qcustomplot.h>
#include "../qcustomplot/examples/text-document-integration/
    qcpdocumentobject.h"

namespace Ui {
  class QReportGenerator;
}

class QReportGenerator : public QMainWindow
{
  Q_OBJECT

public:
  QTextDocument *Report = nullptr;
  SoilAnalyzer::Sample *Sample = nullptr;
  SoilAnalyzer::SoilSettings *Settings = nullptr;
  QCustomPlot *PSD = nullptr;
  QCustomPlot *Roundness = nullptr;
  QCustomPlot *Angularity = nullptr;

  explicit QReportGenerator(QWidget *parent = 0,
      SoilAnalyzer::Sample *sample = nullptr, SoilAnalyzer::
      SoilSettings *settings = nullptr, QCustomPlot *psd =
      nullptr, QCustomPlot *roundness = nullptr, QCustomPlot
      *angularity = nullptr);
  ~QReportGenerator();

private slots:
  void on_locationImageDownloaded(QNetworkReply *reply);

  void on_actionSave_triggered();

  void on_actionExport_to_PDF_triggered();

private:
  Ui::QReportGenerator *ui;
  QCustomPlot *CIElabPlot = nullptr;

  void getLocationMap(double &latitude, double &longtitude);
  void SetupCIElabPLot();

  QImage *mapLocation = nullptr;

  QTextCursor rCurs;

  // Layout formats
  QTextBlockFormat TitleFormat;
  QTextBlockFormat HeaderFormat;
  QTextBlockFormat GeneralFormat;
  QTextBlockFormat ImageGraphFormat;

  QTextCharFormat TitleTextFormat;
```

```
  QTextCharFormat HeaderTextFormat;
  QTextCharFormat GtxtFormat;
  QTextCharFormat GFieldtxtFormat;

  QTextListFormat GeneralSampleList;
  QTextTableFormat GeneralTextTableFormat;


  QFont TitleFont;
  QFont HeaderFont;
  QFont GeneralFont;
  QFont FieldFont;
};

#endif // QREPORTGENERATOR_H
```

```
#include "qreportgenerator.h"
#include "ui_qreportgenerator.h"

QReportGenerator::QReportGenerator(QWidget *parent,
                                   SoilAnalyzer::Sample *sample,
                                   SoilAnalyzer::SoilSettings *settings,
                                   QCustomPlot *psd, QCustomPlot *roundness,
                                   QCustomPlot *angularity)
    : QMainWindow(parent), ui(new Ui::QReportGenerator) {
  ui->setupUi(this);
  if (settings == nullptr) {
    settings = new SoilAnalyzer::SoilSettings;
  }
  this->Settings = settings;
  if (sample == nullptr) {
    sample = new SoilAnalyzer::Sample;
  }
  this->Sample = sample;

  if (psd == nullptr) {
    psd = new QCustomPlot;
  }
  this->PSD = psd;

  if (roundness == nullptr) {
    roundness = new QCustomPlot;
  }
  this->Roundness = roundness;

  if (angularity == nullptr) {
    angularity = new QCustomPlot;
  }
  this->Angularity = angularity;

  Report = new QTextDocument(ui->textEdit);
```

```
ui->textEdit->setDocument(Report);
rCurs = QTextCursor(Report);

// Setup the layout
TitleFormat.setAlignment(Qt::AlignCenter);
TitleFont.setBold(true);
TitleFont.setPointSize(36);
TitleTextFormat.setFont(TitleFont);

HeaderFormat.setAlignment(Qt::AlignCenter);
HeaderFormat.setPageBreakPolicy(QTextFormat::
    PageBreak_AlwaysBefore);
HeaderFormat.setTopMargin(40);
HeaderFormat.setBottomMargin(10);
HeaderFont.setBold(true);
HeaderFont.setPointSize(18);
HeaderTextFormat.setFont(HeaderFont);

ImageGraphFormat.setAlignment(Qt::AlignCenter);
ImageGraphFormat.setTopMargin(10);
ImageGraphFormat.setBottomMargin(10);

GeneralFormat.setAlignment(Qt::AlignLeft);

GeneralFont.setPointSize(12);
GeneralFont.setBold(false);
GtxtFormat.setFont(GeneralFont);

FieldFont.setBold(true);
GFieldtxtFormat.setFont(FieldFont);

GeneralSampleList.setStyle(QTextListFormat::ListDisc);

GeneralTextTableFormat.setHeaderRowCount(1);
GeneralTextTableFormat.setBorderStyle(QTextFrameFormat::
    BorderStyle_None);
GeneralTextTableFormat.setWidth(
     QTextLength(QTextLength::PercentageLength, 90));
GeneralTextTableFormat.setAlignment(Qt::AlignCenter);

// Setup the Title
rCurs.setBlockFormat(TitleFormat);
rCurs.insertText("Soil Report", TitleTextFormat);
rCurs.insertBlock();

// Setup the general Text
rCurs.insertBlock(ImageGraphFormat);
QTextTable *mainTable = rCurs.insertTable(5, 2,
    GeneralTextTableFormat);
rCurs = mainTable->cellAt(0, 0).firstCursorPosition();
rCurs.insertText("Sample name:", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::fromStdString(Sample->Name),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

```

```
rCurs.insertText("Sample ID:", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->ID), GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Date:", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::fromStdString(Sample->Date),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Location:", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->Latitude),
    GtxtFormat);
rCurs.insertText(", ", GtxtFormat);
rCurs.insertText(QString::number(Sample->Longtitude),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Sample depth:", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->Depth),
    GtxtFormat);
rCurs.insertText(" [m]", GtxtFormat);
rCurs.movePosition(QTextCursor::NextBlock);
rCurs.insertBlock();

// Insert the Google map
getLocationMap(Sample->Latitude, Sample->Longtitude);

// Setup the QCustomplot handler
QCPDocumentObject *plotObjectHandler = new
    QCPDocumentObject(this);
ui->textEdit->document()->documentLayout()->
    registerHandler(
     QCPDocumentObject::PlotTextFormat, plotObjectHandler);

// Setup the Textdata for the PSD
rCurs.insertBlock(HeaderFormat, HeaderTextFormat);
rCurs.insertText("Particle Size Distribution");

rCurs.insertBlock(ImageGraphFormat);
QTextTable *PSDdescr = rCurs.insertTable(6, 2,
    GeneralTextTableFormat);
rCurs = PSDdescr->cellAt(0, 0).firstCursorPosition();
rCurs.insertText("No of particles:", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.n),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Mean: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.Mean),
    GtxtFormat);
```

```
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Minimum: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.min),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Maximum: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.max),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Range: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.Range),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Standard deviation: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.Std),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextBlock);

// Setup the PSD
rCurs.insertBlock(ImageGraphFormat);
rCurs.insertText(QString(QChar::ObjectReplacementCharacter
    ),
                 QCPDocumentObject::generatePlotFormat(PSD
                     , 600, 350));

rCurs.insertBlock(ImageGraphFormat);
QTextTable *PSDdata = rCurs.insertTable(16, 3,
    GeneralTextTableFormat);
rCurs.insertText("Mesh Size [mm]", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText("Cummulatief [%]", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText("Retained [-]", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText("2", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.CFD[14]),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.bins[14]),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText("1.4", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.CFD[13]),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->PSD.bins[13]),
```

```
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("1", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[12]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[12]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.71", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[11]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[11]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.5", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[10]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[10]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.355", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[9]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[9]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.25", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[8]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[8]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.18", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[7]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[7]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.125", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[6]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[6]),
```

```
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.09", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[5]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[5]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.075", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[4]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[4]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.063", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[3]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[3]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.045", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[2]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[2]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0.038", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[1]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[1]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText("0", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.CFD[0]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->PSD.bins[0]),
        GtxtFormat);
    rCurs.movePosition(QTextCursor::NextBlock);

    // Setup the Textdata for the Roundness
    rCurs.insertBlock(HeaderFormat, HeaderTextFormat);
    rCurs.insertText("Sphericity Classification");

    rCurs.insertBlock(ImageGraphFormat);
```

```
QTextTable *Rounddescr = rCurs.insertTable(6, 2,
    GeneralTextTableFormat);
rCurs = Rounddescr->cellAt(0, 0).firstCursorPosition();
rCurs.insertText("No of particles:", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->Roundness.n),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Mean: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->Roundness.Mean),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Minimum: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->Roundness.min),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Maximum: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->Roundness.max),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Range: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->Roundness.Range),
     GtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);

rCurs.insertText("Standard deviation: ", GFieldtxtFormat);
rCurs.movePosition(QTextCursor::NextCell);
rCurs.insertText(QString::number(Sample->Roundness.Std),
    GtxtFormat);
rCurs.movePosition(QTextCursor::NextBlock);

// Setup the Roundness Graph
rCurs.insertBlock(ImageGraphFormat);
rCurs.insertText(QString(QChar::ObjectReplacementCharacter
    ),
                 QCPDocumentObject::generatePlotFormat(
                    Roundness, 600, 400));

// Setup the Textdata for the Roundness
rCurs.insertBlock(HeaderFormat, HeaderTextFormat);
rCurs.insertText("Angularity Classification");

rCurs.insertBlock(ImageGraphFormat);
QTextTable *Angularitydescr = rCurs.insertTable(6, 2,
    GeneralTextTableFormat);
rCurs = Angularitydescr->cellAt(0, 0).firstCursorPosition
    ();
rCurs.insertText("No of particles:", GFieldtxtFormat);
```

```
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->Angularity.n), GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);

    rCurs.insertText("Mean: ", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->Angularity.Mean), GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);

    rCurs.insertText("Minimum: ", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->Angularity.min), GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);

    rCurs.insertText("Maximum: ", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->Angularity.max), GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);

    rCurs.insertText("Range: ", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->Angularity.Range), GtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);

    rCurs.insertText("Standard deviation: ", GFieldtxtFormat);
    rCurs.movePosition(QTextCursor::NextCell);
    rCurs.insertText(QString::number(Sample->Angularity.Std), GtxtFormat);
    rCurs.movePosition(QTextCursor::NextBlock);

    // Setup the Roundness Graph
    rCurs.insertBlock(ImageGraphFormat);
    rCurs.insertText(QString(QChar::ObjectReplacementCharacter),
                     QCPDocumentObject::generatePlotFormat(Angularity, 600, 400));

    // Setup the CIE La*b* graph
    // Setup the Textdata for the Roundness
    rCurs.insertBlock(HeaderFormat, HeaderTextFormat);
    rCurs.insertText("CIE La*b*");

    SetupCIElabPLot();
    rCurs.insertBlock(ImageGraphFormat);
    rCurs.insertText(QString(QChar::ObjectReplacementCharacter),
                     QCPDocumentObject::generatePlotFormat(CIElabPlot, 600, 400));

}

```

```
void QReportGenerator::getLocationMap(double &latitude, double &longtitude) {
  QNetworkAccessManager *manager = new QNetworkAccessManager;
  connect(manager, SIGNAL(finished(QNetworkReply *)), this,
          SLOT(on_locationImageDownloaded(QNetworkReply *)));
  QString locationURL("http://maps.googleapis.com/maps/api/staticmap?center=");
  locationURL.append(QString::number(latitude));
  locationURL.append(",");
  locationURL.append(QString::number(longtitude));
  locationURL.append("&zoom=17&size=600x750&maptype=hybrid&&format=png&visual_"
                     "refresh=true&markers=size:mid%7Ccolor:0xff0000%7Clabel:S%"
                     "7C");
  locationURL.append(QString::number(latitude));
  locationURL.append(",");
  locationURL.append(QString::number(longtitude));
  qDebug() << locationURL;
  QUrl googleStaticMapUrl(locationURL);
  manager->get(QNetworkRequest(googleStaticMapUrl));
}

void QReportGenerator::on_locationImageDownloaded(QNetworkReply *reply) {
  if (mapLocation == nullptr) {
    mapLocation = new QImage;
  }
  mapLocation->loadFromData(reply->readAll());

  if (mapLocation->isNull()) {
    mapLocation->load("Maps/SampleLocation.png");
  }

  QTextBlock location = Report->findBlockByNumber(15);
  QTextCursor insertMap(location);
  insertMap.setBlockFormat(ImageGraphFormat);
  insertMap.insertImage(*mapLocation);
  insertMap.insertBlock();
  insertMap.insertHtml("<br>");
}

QReportGenerator::~QReportGenerator()
{
  delete CIElabPlot;
  delete mapLocation;
  delete ui;
}

void QReportGenerator::on_actionSave_triggered() {
  QString fn = QFileDialog::getSaveFileName(
      this, tr("Save Report"), QString::fromStdString(Settings->SampleFolder),
      tr("Report (*.odf)"));
```

```
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".odf"))) {
      fn.append(tr(".odf"));
    }
    QTextDocumentWriter m_write;
    m_write.setFileName(fn);
    m_write.setFormat("odf");
    m_write.write(Report);
  }
}

void QReportGenerator::on_actionExport_to_PDF_triggered() {
  QString fn = QFileDialog::getSaveFileName(
      this, tr("Save Report"), QString::fromStdString(
          Settings->SampleFolder),
      tr("Report (*.pdf)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".pdf"))) {
      fn.append(tr(".pdf"));
    }
    QPrinter printer;
    printer.setOutputFormat(QPrinter::PdfFormat);
    printer.setOutputFileName(fn);
    Report->print(&printer);
  }
}

void QReportGenerator::SetupCIElabPLot() {
  if (CIElabPlot == nullptr) {
      CIElabPlot = new QCustomPlot();
    }

  QPen binPen;
  binPen.setColor((QColor("blue")));
  binPen.setStyle(Qt::SolidLine);
  binPen.setWidthF(1);

  // Setup the CIElabplot plot
  QCPPlotTitle *CIEtitle = new QCPPlotTitle(CIElabPlot);
  CIEtitle->setText("mean CIE Lab - a* vs. b*");
  CIEtitle->setFont(QFont("sans", 8, QFont::Bold));
  CIElabPlot->plotLayout()->insertRow(0);
  CIElabPlot->plotLayout()->addElement(0, 0, CIEtitle);

  CIElabPlot->addGraph(CIElabPlot->xAxis, CIElabPlot->yAxis)
     ;
  CIElabPlot->graph(0)
      ->setScatterStyle(QCPScatterStyle(QCPScatterStyle::
          ssCircle, 8));
  CIElabPlot->graph(0)->setPen(binPen);
  CIElabPlot->graph(0)->setName("a* vs. b*");
  CIElabPlot->graph(0)->setData(*Sample->GetCIELab_bVector()
     , *Sample->GetCIELab_aVector());
  CIElabPlot->graph(0)->setScatterStyle(QCPScatterStyle::
     ssCross);
  CIElabPlot->graph(0)->setLineStyle(QCPGraph::lsNone);
```

```
  CIElabPlot->xAxis->setLabel("mean chromatic b*");
  CIElabPlot->xAxis->setTickLabelFont(QFont("sans", 8, QFont
      ::Normal));
  CIElabPlot->xAxis->setScaleType(QCPAxis::stLinear);
  CIElabPlot->xAxis->setRange(-128,128);

  CIElabPlot->yAxis->setLabel("mean chromatic a*");
  CIElabPlot->yAxis->setTickLabelFont(QFont("sans", 8, QFont
      ::Normal));
  CIElabPlot->yAxis->setScaleType(QCPAxis::stLinear);
  CIElabPlot->yAxis->setRange(-128,128);
  CIElabPlot->replot();

}
```

K. Vision Soil Analyzer Program

K.0.1 General project files

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-08-07T16:50:24
#
#
#-------------------------------------------------

QT       += core gui concurrent
QMAKE_CXXFLAGS += -std=c++11

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets printsupport multimedia multimediawidgets

TARGET = VSA
TEMPLATE = app
VERSION = 0.9.7

unix:!macx: QMAKE_RPATHDIR += $$PWD/../../../build/install/

@
CONFIG(release, debug|release):DEFINES += QT_NO_DEBUG_OUTPUT
@

SOURCES += main.cpp\
        vsamainwindow.cpp \
    dialogsettings.cpp \
    dialognn.cpp

HEADERS  += vsamainwindow.h \
    dialogsettings.h \
    dialognn.h
```

```

FORMS    += vsamainwindow.ui \
    dialogsettings.ui \
    dialognn.ui

#opencv
LIBS += -L/usr/local/lib -lopencv_core -lopencv_highgui -lopencv_imgcodecs
INCLUDEPATH += /usr/local/include/opencv
INCLUDEPATH += /usr/local/include

#boost
DEFINES += BOOST_ALL_DYN_LINK
INCLUDEPATH += /usr/include/boost
LIBS += -L/usr/lib/x86_64-linux-gnu/ -lboost_filesystem -lboost_serialization -lboost_system -lboost_iostreams

#SoilMath lib
unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilMath
INCLUDEPATH += $$PWD/../SoilMath
DEPENDPATH += $$PWD/../SoilMath

#SoilHardware lib
unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilHardware
INCLUDEPATH += $$PWD/../SoilHardware
DEPENDPATH += $$PWD/../SoilHardware

#SoilVision lib
unix:!macx: LIBS += -L$$PWD/../../build/install/ -lSoilVision
INCLUDEPATH += $$PWD/../SoilVision
DEPENDPATH += $$PWD/../SoilVision

#QCustomplot lib
DEFINES += QCUSTOMPLOT_USE_LIBRARY
unix:!macx: LIBS += -L$$PWD/../../build/install/ -lqcustomplot
INCLUDEPATH += $$PWD/../qcustomplot
DEPENDPATH += $$PWD/../qcustomplot

#QParticleSelector
unix:!macx: LIBS += -L$$PWD/../../build/install/ -lQParticleSelector
INCLUDEPATH += $$PWD/../QParticleSelector
DEPENDPATH += $$PWD/../QParticleSelector

#QParticleDisplay
unix:!macx: LIBS += -L$$PWD/../../build/install/ -lQParticleDisplay
INCLUDEPATH += $$PWD/../QParticleDisplay
DEPENDPATH += $$PWD/../QParticleDisplay

#QOpenCVQT
unix:!macx: LIBS += -L$$PWD/../../build/install/ -lQOpenCVQT
INCLUDEPATH += $$PWD/../QOpenCVQT
```

```
DEPENDPATH += $$PWD/../QOpenCVQT

#QSoilAnalyzer
unix:!macx: LIBS += -L$$PWD/../../build/install/ -
    lSoilAnalyzer
INCLUDEPATH += $$PWD/../SoilAnalyzer
DEPENDPATH += $$PWD/../SoilAnalyzer

#QReportGenerator
unix:!macx: LIBS += -L$$PWD/../../build/install/ -
    lQReportGenerator
INCLUDEPATH += $$PWD/../QReportGenerator
DEPENDPATH += $$PWD/../QReportGenerator

#NeuralNetFiles
NNtarget.path += $${OUT_PWD}/NeuralNet
NNtarget.files += $${PWD}/NeuralNet/*.NN
INSTALLS += NNtarget
bNNtarget.path += $${PWD}/../../build/install/NeuralNet
bNNtarget.files += $${PWD}/NeuralNet/*.NN
INSTALLS += bNNtarget

#SettingFiles
INItarget.path += $${OUT_PWD}/Settings
INItarget.files += $${PWD}/Settings/*.ini
INSTALLS += INItarget
bINItarget.path += $$PWD/../../build/install/Settings
bINItarget.files += $$PWD/Settings/*.ini
INSTALLS += bINItarget

#SoilSamples
IMGtarget.path += $${OUT_PWD}/SoilSamples
IMGtarget.files += $${PWD}/SoilSamples/*.VSA
INSTALLS += IMGtarget
bIMGtarget.path += $${PWD}/../../build/install/SoilSamples
bIMGtarget.files += $${PWD}/SoilSamples/*.VSA
INSTALLS += bIMGtarget

#Images
Imgtarget.path += $${OUT_PWD}/Images
Imgtarget.files += $${PWD}/Images/*
INSTALLS += Imgtarget
bImgtarget.path += $${PWD}/../../build/install/Images
bImgtarget.files += $${PWD}/Images/*
INSTALLS += bImgtarget

#TestedSample
TestedSamplesTarget.path += $${OUT_PWD}/TestedSamples
TestedSamplesTarget.files += $${PWD}/TestedSamples/*
INSTALLS += Imgtarget
bTestedSamplesTarget.path += $${PWD}/../../build/install/
    TestedSamples
bTestedSamplesTarget.files += $${PWD}/TestedSamples/*
INSTALLS += bImgtarget

RESOURCES += \
```

```
    vsa_resources.qrc

#MainProg
unix {
    target.path = $PWD/../../../build/install
    INSTALLS += target
}

DISTFILES += \
    Settings/Default.ini \
    NeuralNet/Default.NN \
    Settings/User.ini \
    SoilSamples/Eurogrit_B3_01__Cat.VSA \
    SoilSamples/Gran_K1_0.5_2.5__01_Cat.VSA \
    TestedSamples/Filterzand_0.2_1.6.csv \
    TestedSamples/Magro_dol.csv \
    TestedSamples/Gran_K1.csv \
    TestedSamples/GL70.csv \
    TestedSamples/Gannet_20_40.csv \
    TestedSamples/Eurogrit.csv \
    TestedSamples/0.8_1.25.csv
```

```
#include "vsamainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  VSAMainWindow w;
  w.show();

  return a.exec();
}
```

K.0.2 Main window Class

```cpp
#ifndef VSAMAINWINDOW_H
#define VSAMAINWINDOW_H

#include <QDebug>
#include <QMainWindow>
#include <QErrorMessage>
#include <QMessageBox>
#include <QProgressBar>
#include <QBrush>

#include <stdint.h>

#include <qcustomplot.h>

#include "soilanalyzer.h"
#include "Hardware.h"

#include "dialognn.h"
#include "dialogsettings.h"
#include "qparticleselector.h"
#include "qreportgenerator.h"

namespace Ui {
class VSAMainWindow;
}

class VSAMainWindow : public QMainWindow {
  Q_OBJECT

public:
  explicit VSAMainWindow(QWidget *parent = 0);
  ~VSAMainWindow();

private slots:
  void on_actionSettings_triggered();

  void on_analyzer_finished();

  void on_actionNeuralNet_triggered();

  void on_actionNewSample_triggered();

  void on_actionSaveSample_triggered();

  void on_actionLoadSample_triggered();

  void on_actionUseLearning_toggled(bool arg1);

  void on_actionCalibrate_triggered();

  void on_Classification_changed(int newValue);

  void on_particle_deleted();

```

```cpp
  void on_actionAutomatic_Shape_Pediction_triggered(bool checked);

  void on_reset_graph(QMouseEvent * e);

  void on_actionReport_Generator_triggered();

  void on_particleChanged(int newPart);

  void on_PSD_contextMenuRequest(QPoint point);

  void on_compare_against();

  void on_restore_PSD();

private:
  Ui::VSAMainWindow *ui;
  DialogSettings *settingsWindow = nullptr;
  DialogNN *nnWindow = nullptr;
  QProgressBar *Progress;
  QErrorMessage *CamError = nullptr;
  QMessageBox *SaveMeMessage = nullptr;
  QMessageBox *BacklightMessage = nullptr;
  QMessageBox *ShakeItBabyMessage = nullptr;
  QReportGenerator *ReportGenWindow = nullptr;

  SoilAnalyzer::SoilSettings *Settings = nullptr;
  Hardware::Microscope *Microscope = nullptr;
  SoilAnalyzer::Sample *Sample = nullptr;
  SoilAnalyzer::Analyzer *Analyzer = nullptr;
  SoilAnalyzer::Analyzer::Images_t *Images = nullptr;
  QCPBars *RoundnessBars = nullptr;
  QCPBars *AngularityBars = nullptr;
  std::vector<double> PSDTicks = {0.0,  0.038, 0.045, 0.063, 0.075,
                                  0.09, 0.125, 0.18,  0.25, 0.355,
                                  0.5,  0.71,  1.0,   1.4, 2.0};
  QVector<QString> RoundnessCat = {"High", "Medium", "Low"};
  std::vector<double> RoundnessTicks = {1, 2, 3};
  QVector<QString> AngularityCat = {"Very Angular", "Angular", "Sub Angular",
                                    "Sub Rounded",  "Rounded", "Well Rounded"};
  std::vector<double> AngularityTicks = {1, 2, 3, 4, 5, 6};

  bool ParticleDisplayerFilled = false;

  void SetPSDgraph();
  void setRoundnessHistogram();
  void setAngularityHistogram();
  void setAmpgraph();
  void TakeSnapShots();
};
```

```
#endif // VSAMAINWINDOW_H
```

```
#include "vsamainwindow.h"
#include "ui_vsamainwindow.h"

VSAMainWindow::VSAMainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::VSAMainWindow) {
  ui->setupUi(this);

  // Load the usersettings
  Settings = new SoilAnalyzer::SoilSettings;
  Settings->LoadSettings("Settings/User.ini");

  // Set the message windows
  CamError = new QErrorMessage(this);
  SaveMeMessage = new QMessageBox(this);
  SaveMeMessage->setText(tr("Sample is not saved, Save sample?"));
  SaveMeMessage->addButton(QMessageBox::Abort);
  SaveMeMessage->addButton(QMessageBox::Close);

  BacklightMessage = new QMessageBox(this);
  BacklightMessage->setText("Turn off Frontlight! Turn on Backlight!");
  ShakeItBabyMessage = new QMessageBox(this);

  // Load the Microscope
  Microscope = new Hardware::Microscope;
  try {
    Microscope->FindCam(Settings->defaultWebcam)->SelectedResolution =
        &Microscope->FindCam(Settings->defaultWebcam)
             ->Resolutions[Settings->selectedResolution];
  } catch (exception &e) {
    Microscope->FindCam(0)->SelectedResolution =
        &Microscope->FindCam(0)->Resolutions[Settings->selectedResolution];
  }
  try {
    if (!Microscope->openCam(Settings->defaultWebcam)) {
      int defaultCam = 0;
      Microscope->openCam(defaultCam);
      Settings->defaultWebcam = Microscope->SelectedCam->Name;
    }
  } catch (Hardware::Exception::MicroscopeException &e) {
    if (*e.id() == EXCEPTION_OPENCAM_NR) {
      try {
        int defaultCam = 0;
        Microscope->openCam(defaultCam);
        Settings->defaultWebcam = Microscope->SelectedCam->Name;
      } catch (Hardware::Exception::MicroscopeException &e) {
        if (*e.id() == EXCEPTION_NOCAMS_NR) {
          CamError->showMessage(
```

```
            tr("No cams found! Connect the cam and set the default"));
        settingsWindow = new DialogSettings(this, Settings, Microscope);
      }
    }
  }
}

// Setup the sample
Sample = new SoilAnalyzer::Sample;
Images = new SoilAnalyzer::Analyzer::Images_t;
Analyzer = new SoilAnalyzer::Analyzer(Images, Sample, Settings);

// Setup the setting Window
if (settingsWindow == nullptr) {
  settingsWindow =
      new DialogSettings(this, Settings, Microscope, &Analyzer->NeuralNet);
}

// Setup the NN window
if (nnWindow == nullptr) {
  nnWindow =
      new DialogNN(this, &Analyzer->NeuralNet, Settings, settingsWindow);
}

// Setup the progresbar and connect it to the Analyzer
Progress = new QProgressBar(ui->statusBar);
Progress->setMaximum(Analyzer->MaxProgress);
Progress->setValue(0);
Progress->setAlignment(Qt::AlignLeft);
Progress->setMinimumSize(750, 19);
ui->statusBar->addWidget(Progress);
connect(Analyzer, SIGNAL(on_progressUpdate(int)), Progress,
        SLOT(setValue(int)));
connect(Analyzer, SIGNAL(on_progressUpdate(int)), Progress,
        SLOT(setMaximum(int)));
connect(Analyzer, SIGNAL(on_AnalysisFinished()), this,
        SLOT(on_analyzer_finished()));
// Setup the plot linestyles;
QPen pdfPen;
pdfPen.setColor(QColor("gray"));
pdfPen.setStyle(Qt::DashDotDotLine);
pdfPen.setWidthF(1);

QPen meanPen;
meanPen.setColor(QColor("darkBlue"));
meanPen.setStyle(Qt::DashLine);
meanPen.setWidthF(1);

QPen binPen;
```

```
binPen.setColor((QColor("blue")));
binPen.setStyle(Qt::SolidLine);
binPen.setWidthF(2);

// Setup the PSD plot
QCPPlotTitle *PSDtitle = new QCPPlotTitle(ui->Qplot_PSD);
PSDtitle->setText("Particle Size Distribution");
PSDtitle->setFont(QFont("sans", 8, QFont::Bold));
ui->Qplot_PSD->plotLayout()->insertRow(0);
ui->Qplot_PSD->plotLayout()->addElement(0, 0, PSDtitle);

ui->Qplot_PSD->addGraph(ui->Qplot_PSD->xAxis, ui->Qplot_PSD->yAxis);
ui->Qplot_PSD->graph(0)
     ->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ssCircle, 8));
ui->Qplot_PSD->graph(0)->setPen(binPen);
ui->Qplot_PSD->graph(0)->setName("Particle Size Distribution");
ui->Qplot_PSD->graph(0)->addToLegend();

ui->Qplot_PSD->xAxis->setLabel("Particle size [mm]");
ui->Qplot_PSD->xAxis->setRange(0.01, 10);
ui->Qplot_PSD->xAxis->setAutoTicks(false);
ui->Qplot_PSD->xAxis->setTickVector(QVector<double>::fromStdVector(PSDTicks));
ui->Qplot_PSD->xAxis->setTickLabelRotation(30);
ui->Qplot_PSD->xAxis->setTickLabelFont(QFont("sans", 8, QFont::Normal));
ui->Qplot_PSD->xAxis->setScaleType(QCPAxis::stLogarithmic);

QFont legendfont;
legendfont.setPointSize(10);
ui->Qplot_PSD->legend->setFont(legendfont);
ui->Qplot_PSD->legend->setSelectedFont(legendfont);
ui->Qplot_PSD->legend->setVisible(true);
ui->Qplot_PSD->axisRect()->insetLayout()->setInsetAlignment(
     0, Qt::AlignTop | Qt::AlignLeft);

ui->Qplot_PSD->yAxis->setLabel("Percentage [%]");
ui->Qplot_PSD->yAxis->setRange(0, 100);
ui->Qplot_PSD->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom);
ui->Qplot_PSD->yAxis->grid()->setSubGridVisible(true);

connect(ui->Qplot_PSD, SIGNAL(mouseDoubleClick(QMouseEvent *)), this,
          SLOT(on_reset_graph(QMouseEvent *)));
ui->Qplot_PSD->setContextMenuPolicy(Qt::CustomContextMenu);
connect(ui->Qplot_PSD, SIGNAL(customContextMenuRequested(QPoint)), this,
          SLOT(on_PSD_contextMenuRequest(QPoint)));

```

```
    // Setup the Roundness plot
    QCPPlotTitle *Roundnesstitle = new QCPPlotTitle(ui->QPlot_Roudness);
    Roundnesstitle->setText("Sphericity Histogram");
    Roundnesstitle->setFont(QFont("sans", 8, QFont::Bold));
    ui->QPlot_Roudness->plotLayout()->insertRow(0);
    ui->QPlot_Roudness->plotLayout()->addElement(0, 0, Roundnesstitle);

    ui->QPlot_Roudness->addGraph(ui->QPlot_Roudness->xAxis,
                                 ui->QPlot_Roudness->yAxis2);
    ui->QPlot_Roudness->addGraph(ui->QPlot_Roudness->xAxis,
                                 ui->QPlot_Roudness->yAxis2);
    ui->QPlot_Roudness->graph(0)->setPen(pdfPen);
    ui->QPlot_Roudness->graph(1)->setPen(meanPen);

    RoundnessBars =
        new QCPBars(ui->QPlot_Roudness->xAxis, ui->QPlot_Roudness->yAxis);
    ui->QPlot_Roudness->addPlottable(RoundnessBars);
    RoundnessBars->setPen(binPen);

    ui->QPlot_Roudness->xAxis->setAutoTicks(false);
    ui->QPlot_Roudness->xAxis->setAutoTickLabels(false);
    ui->QPlot_Roudness->xAxis->setTickVector(
        QVector<double>::fromStdVector(RoundnessTicks));
    ui->QPlot_Roudness->xAxis->setTickVectorLabels(RoundnessCat);
    ui->QPlot_Roudness->xAxis->setTickLabelRotation(30);
    ui->QPlot_Roudness->xAxis->setSubTickCount(0);
    ui->QPlot_Roudness->xAxis->setTickLength(0, 4);
    ui->QPlot_Roudness->xAxis->grid()->setVisible(true);
    ui->QPlot_Roudness->xAxis->setRange(0, 4);
    ui->QPlot_Roudness->xAxis->setLabel("Count [-]");
    ui->QPlot_Roudness->xAxis->setLabelFont(QFont("sans", 8, QFont::Bold));
    ui->QPlot_Roudness->xAxis->setTickLabelFont(QFont("sans", 8, QFont::Normal));
    ui->QPlot_Roudness->xAxis->setPadding(25);
    ui->QPlot_Roudness->yAxis->setLabel("Sphericity [-]");
    ui->QPlot_Roudness->yAxis->setLabelFont(QFont("sans", 8, QFont::Bold));

    // Setup the angularity plot
    QCPPlotTitle *Angularitytitle = new QCPPlotTitle(ui->QPlot_Angularity);
    Angularitytitle->setText("Angularity Histogram");
    Angularitytitle->setFont(QFont("sans", 8, QFont::Bold));
    ui->QPlot_Angularity->plotLayout()->insertRow(0);
    ui->QPlot_Angularity->plotLayout()->addElement(0, 0, Angularitytitle);

    ui->QPlot_Angularity->addGraph(ui->QPlot_Angularity->xAxis,
                                   ui->QPlot_Angularity->yAxis2);
```

```
    ui->QPlot_Angularity->addGraph(ui->QPlot_Angularity->xAxis,
                                   ui->QPlot_Angularity->yAxis2);
    AngularityBars =
        new QCPBars(ui->QPlot_Angularity->xAxis, ui->QPlot_Angularity->yAxis);
    ui->QPlot_Angularity->addPlottable(AngularityBars);
    AngularityBars->setPen(binPen);

    ui->QPlot_Angularity->xAxis->setAutoTicks(false);
    ui->QPlot_Angularity->xAxis->setAutoTickLabels(false);
    ui->QPlot_Angularity->xAxis->setTickVector(
        QVector<double>::fromStdVector(AngularityTicks));
    ui->QPlot_Angularity->xAxis->setTickVectorLabels(AngularityCat);
    ui->QPlot_Angularity->xAxis->setTickLabelRotation(30);
    ui->QPlot_Angularity->xAxis->setSubTickCount(0);
    ui->QPlot_Angularity->xAxis->setTickLength(0, 4);
    ui->QPlot_Angularity->xAxis->grid()->setVisible(true);
    ui->QPlot_Angularity->xAxis->setRange(0, 7);
    ui->QPlot_Angularity->xAxis->setLabel("Count [-]");
    ui->QPlot_Angularity->xAxis->setLabelFont(QFont("sans", 8, QFont::Bold));
    ui->QPlot_Angularity->xAxis->setTickLabelFont(
        QFont("sans", 8, QFont::Normal));
    ui->QPlot_Angularity->yAxis->setLabel("Sphericity [-]");
    ui->QPlot_Angularity->yAxis->setLabelFont(QFont("sans", 8, QFont::Bold));
    ui->QPlot_Angularity->graph(0)->setPen(pdfPen);
    ui->QPlot_Angularity->graph(1)->setPen(meanPen);

    // Setup the Amplitude diagram
    QCPPlotTitle *Amptitle = new QCPPlotTitle(ui->QPlot_Amp);
    Amptitle->setText("Fast Fourier Amplitude for the current particle");
    Amptitle->setFont(QFont("sans", 8, QFont::Bold));
    ui->QPlot_Amp->plotLayout()->insertRow(0);
    ui->QPlot_Amp->plotLayout()->addElement(0, 0, Amptitle);

    ui->QPlot_Amp->addGraph(ui->QPlot_Amp->xAxis, ui->QPlot_Amp->yAxis);

    ui->QPlot_Amp->xAxis->setTickLabelRotation(30);
    ui->QPlot_Amp->xAxis->setSubTickCount(0);
    ui->QPlot_Amp->xAxis->setTickLength(0, 4);
    ui->QPlot_Amp->xAxis->grid()->setVisible(true);
    ui->QPlot_Amp->xAxis->setRange(0, 512);
    ui->QPlot_Amp->xAxis->setLabel("Frequency [-]");
    ui->QPlot_Amp->xAxis->setLabelFont(QFont("sans", 8, QFont::Bold));
    ui->QPlot_Amp->xAxis->setTickLabelFont(QFont("sans", 8, QFont::Normal));
    ui->QPlot_Amp->yAxis->setLabel("Amplitude [-]");
    ui->QPlot_Amp->yAxis->setLabelFont(QFont("sans", 8, QFont::Bold));
```

```
  ui->QPlot_Amp->yAxis->setScaleType(QCPAxis::stLogarithmic);
  ui->QPlot_Amp->graph()->setPen(binPen);
  ui->QPlot_Amp->graph()->setLineStyle(QCPGraph::lsLine);
  ui->QPlot_Amp->graph()->setBrush(QBrush(QColor(50,50,200,40)));

  // Connect the Particle display and Selector
  connect(ui->widget_ParticleSelector, SIGNAL(valueChanged(int)), this,
          SLOT(on_Classification_changed(int)));
  connect(ui->widget_ParticleDisplay, SIGNAL(shapeClassificationChanged(int)),
          ui->widget_ParticleSelector, SLOT(setValue(int)));
  connect(ui->widget_ParticleDisplay, SIGNAL(particleDeleted()), this,
          SLOT(on_particle_deleted()));
  connect(ui->widget_ParticleDisplay, SIGNAL(particleChanged(int)), this,
          SLOT(on_particleChanged(int)));

  // Setup the bar
  ui->actionUseLearning->setChecked(Settings->PredictTheShape);

  // Setup the widgets
  ui->widget_ParticleSelector->setDisabled(true);
}

VSAMainWindow::~VSAMainWindow() {
  delete Settings;
  delete Microscope;
  delete Analyzer;
  delete Sample;
  delete Images;

  delete settingsWindow;
  delete nnWindow;
  delete CamError;
  delete SaveMeMessage;
  delete BacklightMessage;
  delete ShakeItBabyMessage;
  delete ui;
}

void VSAMainWindow::on_actionSettings_triggered() {
  settingsWindow->openTab(0);
  settingsWindow->show();
}

void VSAMainWindow::on_analyzer_finished() {
  if (!ParticleDisplayerFilled && Sample->ParticlePopulation.size() > 0) {
    ui->widget_ParticleDisplay->SetSample(Sample);
  }
  SetPSDgraph();
```

```
  setRoundnessHistogram();
  setAngularityHistogram();
  ParticleDisplayerFilled = true;
}

void VSAMainWindow::SetPSDgraph() {
  std::vector<double> stdPSDvalue(Sample->PSD.CFD, Sample->
      PSD.CFD + 15);
  ui->Qplot_PSD->graph(0)->setData(PSDTicks, stdPSDvalue);
  ui->Qplot_PSD->replot();
}

void VSAMainWindow::setRoundnessHistogram() {
  // Setup the Histogram bins
  std::vector<double> stdValues(Sample->Roundness.bins + 1,
                                Sample->Roundness.bins + 4);

  ui->QPlot_Roudness->yAxis->setRange(
      0, static_cast<double>(Sample->Roundness.
          HighestFrequency()));
  RoundnessBars->setData(RoundnessTicks, stdValues);

  // Setup the Prediction Density Function
  std::vector<double> stdPDFkey, stdPDFvalues;
  Sample->Roundness.GetPDFfunction(stdPDFkey, stdPDFvalues,
      0.2, 0, 4);
  ui->QPlot_Roudness->graph(0)->setData(stdPDFkey,
      stdPDFvalues);
  ui->QPlot_Roudness->yAxis2->setRange(0, Sample->Roundness.
      HighestPDF);

  // Setup the mean Vector
  QVector<double> meanKey(2, static_cast<double>(Sample->
      Roundness.Mean));
  QVector<double> meanValue(2);
  meanValue[0] = 0;
  meanValue[1] = Sample->Roundness.HighestPDF;
  ui->QPlot_Roudness->graph(1)->setData(meanKey, meanValue);
  ui->QPlot_Roudness->replot();
}

void VSAMainWindow::setAngularityHistogram() {
  // Setup the Histogram bins
  std::vector<double> stdValues(Sample->Angularity.bins + 1,
                                Sample->Angularity.bins + 7)
                                    ;

  ui->QPlot_Angularity->yAxis->setRange(
      0, static_cast<double>(Sample->Angularity.
          HighestFrequency()));
  AngularityBars->setData(AngularityTicks, stdValues);

  // Setup the Prediction Density Function
  std::vector<double> stdPDFkey, stdPDFvalues;
  Sample->Angularity.GetPDFfunction(stdPDFkey, stdPDFvalues,
       0.2, 0, 7);
```

```
  ui->QPlot_Angularity->graph(0)->setData(stdPDFkey,
      stdPDFvalues);
  ui->QPlot_Angularity->yAxis2->setRange(0, Sample->
      Angularity.HighestPDF);

  // Setup the mean Vector
  QVector<double> meanKey(2, static_cast<double>(Sample->
      Angularity.Mean));
  QVector<double> meanValue(2);
  meanValue[0] = 0;
  meanValue[1] = Sample->Angularity.HighestPDF;
  ui->QPlot_Angularity->graph(1)->setData(meanKey, meanValue
      );
  ui->QPlot_Angularity->replot();
}

void VSAMainWindow::setAmpgraph() {
  ui->QPlot_Amp->graph(0)->clearData();
  ComplexVect_t *comp =
      &ui->widget_ParticleDisplay->SelectedParticle->
          FFDescriptors;
  uint32_t count = (comp->size() > 64) ? 64 : comp->size();
  for (uint32_t i = 0; i < count; i++) {
    ui->QPlot_Amp->graph(0)->addData(i, abs(comp->at(i)));
  }
  ui->QPlot_Amp->rescaleAxes();
  ui->QPlot_Amp->replot();
}

void VSAMainWindow::on_particleChanged(int newPart) {
    setAmpgraph(); }

void VSAMainWindow::on_actionNeuralNet_triggered() {
  if (nnWindow != nullptr) {
    nnWindow =
        new DialogNN(this, &Analyzer->NeuralNet, Settings,
            settingsWindow);
  }
  nnWindow->show();
}

void VSAMainWindow::on_actionNewSample_triggered() {
  if (Sample->ChangesSinceLastSave) {
    if (SaveMeMessage->exec() == QMessageBox::Abort) {
      return;
    }
  }
  delete Sample;
  Sample = nullptr;
  delete Images;
  Images = nullptr;
  Sample = new SoilAnalyzer::Sample;
  Images = new SoilAnalyzer::Analyzer::Images_t;
  TakeSnapShots();
  try {
    Analyzer->Analyse(Images, Sample, Settings);
```

```
  } catch (SoilAnalyzer::Exception::SoilAnalyzerException &e
     ) {
    if (*e.id() == EXCEPTION_NO_SNAPSHOTS_NR) {
      CamError->showMessage(
          "No images acquired! Check you microscope settings
              ");
      return;
    }
  }
  Sample->ChangesSinceLastSave = true;
  if (Sample->ParticlePopulation.size() > 0) {
    ui->widget_ParticleSelector->setDisabled(
        false,
        ui->widget_ParticleDisplay->SelectedParticle->
            Classification.Category);
  }
}

void VSAMainWindow::TakeSnapShots() {
  Analyzer->SIfactorDet = true; // remeber to remove
  if (!Analyzer->SIfactorDet) {
    QMessageBox *DetSIFactor = new QMessageBox(this);
    DetSIFactor->setText("Put calibration Disc under the
        microscope");
    DetSIFactor->exec();
    on_actionCalibrate_triggered();
    DetSIFactor->setText("Place sample under the microscope"
        );
    DetSIFactor->exec();
  }
  if (Settings->useBacklightProjection && !Settings->useHDR)
      {
    for (uint32_t i = 0; i < Settings->StandardNumberOfShots
        ; i++) {
      SoilAnalyzer::Analyzer::Image_t newShot;
      newShot.SIPixelFactor = Analyzer->CurrentSIfactor;
      Microscope->GetFrame(newShot.FrontLight);
      BacklightMessage->exec();
      Microscope->GetFrame(newShot.BackLight);
      Images->push_back(newShot);
      QString ShakeMsg = "Shake it baby! ";
      int number = Settings->StandardNumberOfShots - i;
      ShakeMsg.append(QString::number(number));
      ShakeMsg.append(" to go!");
      ShakeItBabyMessage->setText(ShakeMsg);
      ShakeItBabyMessage->exec();
    }
  } else if (Settings->useBacklightProjection && Settings->
     useHDR) {
    for (uint32_t i = 0; i < Settings->StandardNumberOfShots
        ; i++) {
      SoilAnalyzer::Analyzer::Image_t newShot;
      newShot.SIPixelFactor = Analyzer->CurrentSIfactor;
      Microscope->GetHDRFrame(newShot.FrontLight, Settings->
          HDRframes);
      BacklightMessage->exec();
```

```
      Microscope->GetFrame(newShot.BackLight);
      Images->push_back(newShot);
      QString ShakeMsg = "Shake it baby! ";
      int number = Settings->StandardNumberOfShots - i - 1;
      ShakeMsg.append(QString::number(number));
      ShakeMsg.append(" to go!");
      ShakeItBabyMessage->setText(ShakeMsg);
      ShakeItBabyMessage->exec();
    }
  } else if (!Settings->useBacklightProjection && Settings->useHDR) {
    for (uint32_t i = 0; i < Settings->StandardNumberOfShots; i++) {
      SoilAnalyzer::Analyzer::Image_t newShot;
      newShot.SIPixelFactor = Analyzer->CurrentSIfactor;
      Microscope->GetHDRFrame(newShot.FrontLight, Settings->HDRframes);
      Images->push_back(newShot);
      QString ShakeMsg = "Shake it baby! ";
      int number = Settings->StandardNumberOfShots - i - 1;
      ShakeMsg.append(QString::number(number));
      ShakeMsg.append(" to go!");
      ShakeItBabyMessage->setText(ShakeMsg);
      ShakeItBabyMessage->exec();
    }
  } else if (!Settings->useBacklightProjection && !Settings->useHDR) {
    for (uint32_t i = 0; i < Settings->StandardNumberOfShots; i++) {
      SoilAnalyzer::Analyzer::Image_t newShot;
      newShot.SIPixelFactor = Analyzer->CurrentSIfactor;
      Microscope->GetFrame(newShot.FrontLight);
      Images->push_back(newShot);
      QString ShakeMsg = "Shake it baby! ";
      int number = Settings->StandardNumberOfShots - i - 1;
      ShakeMsg.append(QString::number(number));
      ShakeMsg.append(" to go!");
      ShakeItBabyMessage->setText(ShakeMsg);
      ShakeItBabyMessage->exec();
    }
  }
}

void VSAMainWindow::on_actionSaveSample_triggered() {
  QString fn = QFileDialog::getSaveFileName(
      this, tr("Save Sample"), QString::fromStdString(Settings->SampleFolder),
      tr("Sample (*.VSA)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".VSA"))) {
      fn.append(tr(".VSA"));
    }
    Sample->IsLoadedFromDisk = true;
    Sample->ChangesSinceLastSave = false;
    Sample->Save(fn.toStdString());
    qDebug() << "Saving finished";
```

```
  }
}

void VSAMainWindow::on_actionLoadSample_triggered() {
  if (Sample->ChangesSinceLastSave) {
    if (SaveMeMessage->exec() == QMessageBox::Abort) {
      return;
    }
  }

  QString fn = QFileDialog::getOpenFileName(
      this, tr("Open Sample"), QString::fromStdString(Settings->SampleFolder),
      tr("Sample (*.VSA)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".VSA"))) {
      fn.append(tr(".VSA"));
    }
    delete Sample;
    Sample = nullptr;
    delete Images;
    Images = nullptr;
    Sample = new SoilAnalyzer::Sample;
    Images = new SoilAnalyzer::Analyzer::Images_t;
    try {
      Sample->Load(fn.toStdString());
    } catch (boost::archive::archive_exception &e) {
      // qDebug() << *e.what();
    }
    ParticleDisplayerFilled = false;
    Sample->Angularity.Data = Sample->GetAngularityVector()->data();
    Sample->Roundness.Data = Sample->GetRoundnessVector()->data();
    Sample->PSD.Data = Sample->GetPSDVector()->data();
    Analyzer->Results = Sample;
    on_analyzer_finished();
    ui->widget_ParticleSelector->setDisabled(
        false,
        ui->widget_ParticleDisplay->SelectedParticle->Classification.Category);
  }
}

void VSAMainWindow::on_actionUseLearning_toggled(bool arg1) {
  Analyzer->PredictShape = !arg1;
}

void VSAMainWindow::on_actionCalibrate_triggered() {
  cv::Mat calib;
  Microscope->GetFrame(calib);
  Analyzer->CalibrateSI(16.25, calib);
}

void VSAMainWindow::on_Classification_changed(int newValue)
```

```
    {
  uint8_t *Cat =
      &ui->widget_ParticleDisplay->SelectedParticle->Classification.Category;
  if ((*Cat - 1) % 6 != (newValue - 1) % 6) {
    Sample->ParticleChangedStateAngularity = true;
  }
  if ((*Cat - 1) / 6 != (newValue - 1) / 6) {
    Sample->ParticleChangedStateRoundness = true;
  }
  ui->widget_ParticleDisplay->SelectedParticle->Classification.Category =
      newValue;
  ui->widget_ParticleDisplay->SelectedParticle->Classification.ManualSet = true;
  Sample->ChangesSinceLastSave = true;
  Analyzer->Analyse();
  ui->widget_ParticleDisplay->next();
}

void VSAMainWindow::on_particle_deleted() { Analyzer->Analyse(); }

void VSAMainWindow::on_actionAutomatic_Shape_Pediction_triggered(bool checked) {
  Settings->PredictTheShape = checked;
}

void VSAMainWindow::on_reset_graph(QMouseEvent *e) {
  ui->Qplot_PSD->xAxis->setRange(0, 10);
  ui->Qplot_PSD->yAxis->setRange(0, 100);
  ui->Qplot_PSD->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom);
  ui->Qplot_PSD->replot();
}

void VSAMainWindow::on_actionReport_Generator_triggered() {
  if (ReportGenWindow == nullptr) {
    ReportGenWindow =
        new QReportGenerator(this, Sample, Settings, ui->Qplot_PSD,
                                 ui->QPlot_Roudness, ui->QPlot_Angularity);
  }
  ReportGenWindow->show();
}

void VSAMainWindow::on_PSD_contextMenuRequest(QPoint point) {
  QMenu *menu = new QMenu(this);
  menu->setAttribute(Qt::WA_DeleteOnClose);

  menu->addAction("Compare against...", this, SLOT(on_compare_against()));
  menu->addAction("Restore", this, SLOT(on_restore_PSD()));
```

```
  menu->popup(ui->Qplot_PSD->mapToGlobal(point));
}

void VSAMainWindow::on_compare_against() {
  QString fn = QFileDialog::getOpenFileName(
      this, tr("Open CSV"), QString::fromStdString(Settings->SampleFolder),
      tr("Comma Seperated Value (*.csv)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".csv"))) {
      fn.append(tr(".csv"));
    }

    if (ui->Qplot_PSD->graphCount() > 1) {
      ui->Qplot_PSD->legend->removeItem(1);
      ui->Qplot_PSD->removeGraph(1);
    }

    QStringList rows;
    QStringList cellValues;

    QFile f(fn);
    if (f.open(QIODevice::ReadOnly)) {
      QString data;
      data = f.readAll();
      rows = data.split('\n');
      f.close();
      for (uint32_t i = 0; i < rows.size(); i++) {
        QStringList cols = rows[i].split(',');
        for (uint32_t j = 0; j < cols.size(); j++) {
          cellValues.append(cols[j]);
        }
      }
      cellValues.removeLast();

      std::vector<double> compValues(15);
      for (uint32_t i = 0; i < cellValues.size(); i += 4) {
        bool conversionSucces = false;
        double binValue = cellValues[i].toDouble(&conversionSucces);
        qDebug() << cellValues[i + 3];
        if (conversionSucces) {
          for (uint32_t j = 0; j < 15; j++) {
            if (binValue == PSDTicks[j]) {
              compValues[j] = cellValues[i + 3].toDouble();
            }
          }
        }
      }
      ui->Qplot_PSD->addGraph(ui->Qplot_PSD->xAxis, ui->Qplot_PSD->yAxis);
      ui->Qplot_PSD->graph(1)->setData(PSDTicks, compValues);
      QPen compPen;
      compPen.setColor(QColor("darkBlue"));
      compPen.setStyle(Qt::DashLine);
```

```
      compPen.setWidthF(1);
      ui->Qplot_PSD->graph(1)->setPen(compPen);
      ui->Qplot_PSD->graph(1)->setName("Compared Particle
         Size Distribution");
      ui->Qplot_PSD->graph(1)->addToLegend();
      ui->Qplot_PSD->replot();
    }
  }
}

void VSAMainWindow::on_restore_PSD() {
  if (ui->Qplot_PSD->graphCount() > 1) {
    ui->Qplot_PSD->legend->removeItem(1);
    ui->Qplot_PSD->removeGraph(1);
  }
  on_reset_graph(nullptr);
}
```

K.0.3 Dialog window Class

```
#ifndef DIALOGSETTINGS_H
#define DIALOGSETTINGS_H

#include <QDialog>
#include <soilsettings.h>
#include <QFileDialog>
#include <QString>
#include <QDir>
#include <QSlider>
#include "Hardware.h"

namespace Ui {
class DialogSettings;
}

class DialogSettings : public QDialog {
  Q_OBJECT

public:
  SoilAnalyzer::SoilSettings *Settings = nullptr;
  explicit DialogSettings(QWidget *parent = 0,
                          SoilAnalyzer::SoilSettings *settings = nullptr,
                          Hardware::Microscope *microscope = nullptr,
                          SoilMath::NN *nn = nullptr, bool openNN = false);
  ~DialogSettings();

  void openTab(int newValue);
private slots:

  void on_pushButton_RestoreDefault_clicked();

  void on_pushButton_Open_clicked();

  void on_pushButton_Save_clicked();

  void on_checkBox_Backlight_clicked(bool checked);

  void on_comboBox_Microscopes_currentIndexChanged(const QString &arg1);

  void on_comboBox_Resolution_currentIndexChanged(int index);

  void on_checkBox_useHDR_clicked(bool checked);

  void on_spinBox_NoFrames_editingFinished();

  void on_doubleSpinBox_LightLevel_editingFinished();

  void on_checkBox_useRainbow_clicked(bool checked);

```

```
    void on_checkBox_InvertEncoder_clicked(bool checked);

    void on_checkBox_useCUDA_clicked(bool checked);

    void on_horizontalSlider_BrightFront_valueChanged(int value);

    void on_horizontalSlider_ContrastFront_valueChanged(int value);

    void on_horizontalSlider_SaturationFront_valueChanged(int value);

    void on_horizontalSlider_HueFront_valueChanged(int value);

    void on_horizontalSlider_SharpnessFront_valueChanged(int value);

    void on_horizontalSlider_BrightProj_valueChanged(int value);

    void on_horizontalSlider_ContrastProj_valueChanged(int value);

    void on_horizontalSlider_SaturationProj_valueChanged(int value);

    void on_horizontalSlider_HueProj_valueChanged(int value);

    void on_horizontalSlider_SharpnessProj_valueChanged(int value);

    void on_cb_use_adaptContrast_3_clicked(bool checked);

    void on_cb_useBlur_3_clicked(bool checked);

    void on_rb_useDark_3_toggled(bool checked);

    void on_cb_ignoreBorder_3_clicked(bool checked);

    void on_cb_fillHoles_3_clicked(bool checked);

    void on_sb_sigmaFactor_3_editingFinished();

    void on_rb_useOpen_3_clicked(bool checked);

    void on_rb_useClose_3_clicked(bool checked);

    void on_rb_useErode_3_clicked(bool checked);

    void on_rb_useDilate_3_clicked(bool checked);

    void on_sb_morphMask_3_editingFinished();

    void on_spinBox_MaxGen_editingFinished();

```

```
  void on_spinBox_PopSize_editingFinished();

  void on_doubleSpinBox_MutationRate_editingFinished();

  void on_spinBox_Elitisme_editingFinished();

  void on_doubleSpinBox_endError_editingFinished();

  void on_doubleSpinBox_maxWeight_editingFinished();

  void on_doubleSpinBox_MinWeight_editingFinished();

  void on_doubleSpinBox_Beta_editingFinished();

  void on_spinBox_InputNeurons_editingFinished();

  void on_spinBox_HiddenNeurons_editingFinished();

  void on_spinBox_OutputNeurons_editingFinished();

  void on_pushButton_selectSampleFolder_clicked();

  void on_pushButton_SelectSettingFolder_clicked();

  void on_pushButton_SelectNNFolder_clicked();

  void on_pushButton_SelectNN_clicked();

  void on_spinBox_NoShots_editingFinished();

  void on_checkBox_PredictShape_clicked(bool checked);

  void on_checkBox_revolt_clicked(bool checked);

private:
  Ui::DialogSettings *ui;
  Hardware::Microscope *Microscope;
  SoilMath::NN *NN;
  bool initfase = true;
  void SetCamControl(Hardware::Microscope::Cam_t *
      selectedCam,
                     QSlider *Brightness, QSlider *Contrast,
                     QSlider *Saturation, QSlider *Hue,
                        QSlider *Sharpness);
};

#endif // DIALOGSETTINGS_H
```

```
#include "dialogsettings.h"
#include "ui_dialogsettings.h"
#include <opencv2/core.hpp>

DialogSettings::DialogSettings(QWidget *parent,
                               SoilAnalyzer::SoilSettings *
                                   settings,
                               Hardware::Microscope *
```

```
                                      microscope,
                                  SoilMath::NN *nn, bool openNN
                                      )
    : QDialog(parent), ui(new Ui::DialogSettings) {
  ui->setupUi(this);
  if (settings == nullptr) {
    settings = new SoilAnalyzer::SoilSettings;
  }
  Settings = settings;
  if (microscope == nullptr) {
    microscope = new Hardware::Microscope;
  }
  if (nn == nullptr) {
    nn = new SoilMath::NN;
  }

  // Setup the Hardware tab
  Microscope = microscope;
  QStringList Cams;
  for (uint32_t i = 0; i < Microscope->AvailableCams.size();
       i++) {
    Cams << Microscope->AvailableCams[i].Name.c_str();
  }
  ui->comboBox_Microscopes->addItems(Cams);
  ui->comboBox_Microscopes->setCurrentIndex(Microscope->
      SelectedCam->ID);

  QStringList Resolutions;
  for (uint32_t i = 0; i < Microscope->SelectedCam->
      Resolutions.size(); i++) {
    Resolutions << Microscope->SelectedCam->Resolutions[i].
        to_string().c_str();
  }
  ui->comboBox_Resolution->addItems(Resolutions);
  ui->comboBox_Resolution->setCurrentIndex(
       Microscope->SelectedCam->SelectedResolution->ID);

  ui->spinBox_NoShots->setValue(Settings->
      StandardNumberOfShots);

  ui->spinBox_NoFrames->setValue(Settings->HDRframes);
  ui->spinBox_NoFrames->setDisabled(true);
  ui->label_nf->setDisabled(true);

  ui->checkBox_Backlight->setChecked(Settings->
      useBacklightProjection);
  ui->tabWidget_Hardware->setTabEnabled(2, Settings->
      useBacklightProjection);

  ui->checkBox_InvertEncoder->setChecked(Settings->encInv);
  ui->checkBox_useCUDA->setChecked(Settings->useCUDA);

  Settings->useCUDA = false;
  ui->checkBox_useCUDA->setDisabled(true);

  ui->checkBox_useHDR->setChecked(Settings->useHDR);
```

```
ui->checkBox_useRainbow->setChecked(Settings->enableRainbow);

// Get system info
struct utsname unameData;
uname(&unameData);

ui->label_machinename->setText(tr(unameData.machine));
ui->label_nodename->setText(tr(unameData.nodename));
ui->label_releasename->setText(tr(unameData.release));
ui->label_systemname->setText(tr(unameData.sysname));
ui->label_versioname->setText(tr(unameData.version));
if (Microscope->RunEnv == Hardware::Microscope::X64) {
  ui->checkBox_useRainbow->setDisabled(true);
  ui->checkBox_InvertEncoder->setDisabled(true);
  ui->doubleSpinBox_LightLevel->setDisabled(true);
  ui->label_ll->setDisabled(true);
}

SetCamControl(
    Microscope->SelectedCam, ui->horizontalSlider_BrightFront,
    ui->horizontalSlider_ContrastFront, ui->horizontalSlider_SaturationFront,
    ui->horizontalSlider_HueFront, ui->horizontalSlider_SharpnessFront);
ui->horizontalSlider_BrightFront->setValue(Settings->Brightness_front);
ui->horizontalSlider_ContrastFront->setValue(Settings->Contrast_front);
ui->horizontalSlider_HueFront->setValue(Settings->Hue_front);
ui->horizontalSlider_SaturationFront->setValue(Settings->Saturation_front);
ui->horizontalSlider_SharpnessFront->setValue(Settings->Sharpness_front);

SetCamControl(
    Microscope->SelectedCam, ui->horizontalSlider_BrightProj,
    ui->horizontalSlider_ContrastProj, ui->horizontalSlider_SaturationProj,
    ui->horizontalSlider_HueProj, ui->horizontalSlider_SharpnessProj);
ui->horizontalSlider_BrightProj->setValue(Settings->Brightness_proj);
ui->horizontalSlider_ContrastProj->setValue(Settings->Contrast_proj);
ui->horizontalSlider_HueProj->setValue(Settings->Hue_proj);
ui->horizontalSlider_SaturationProj->setValue(Settings->Saturation_proj);
ui->horizontalSlider_SharpnessProj->setValue(Settings->Sharpness_proj);

// Setup the Vision tab
```

```
  ui->cb_fillHoles_3->setChecked(Settings->fillHoles);
  ui->cb_ignoreBorder_3->setChecked(Settings->ignorePartialBorderParticles);
  ui->cb_useBlur_3->setChecked(Settings->useBlur);
  if (!Settings->useBlur) {
    ui->sb_blurMask_3->setEnabled(false);
  }
  ui->cb_use_adaptContrast_3->setChecked(Settings->useAdaptiveContrast);
  if (!Settings->useAdaptiveContrast) {
    ui->sb_adaptContrastFactor_3->setEnabled(false);
    ui->sb_adaptContrKernel_3->setEnabled(false);
  }
  switch (Settings->typeOfObjectsSegmented) {
  case Vision::Segment::Bright:
    ui->rb_useDark_3->setChecked(false);
    ui->rb_useLight_3->setChecked(true);
    break;
  case Vision::Segment::Dark:
    ui->rb_useDark_3->setChecked(true);
    ui->rb_useLight_3->setChecked(false);
    break;
  }
  switch (Settings->morphFilterType) {
  case Vision::MorphologicalFilter::CLOSE:
    ui->rb_useClose_3->setChecked(true);
    ui->rb_useDilate_3->setChecked(false);
    ui->rb_useErode_3->setChecked(false);
    ui->rb_useOpen_3->setChecked(false);
    break;
  case Vision::MorphologicalFilter::OPEN:
    ui->rb_useClose_3->setChecked(false);
    ui->rb_useDilate_3->setChecked(false);
    ui->rb_useErode_3->setChecked(false);
    ui->rb_useOpen_3->setChecked(true);
    break;
  case Vision::MorphologicalFilter::ERODE:
    ui->rb_useClose_3->setChecked(false);
    ui->rb_useDilate_3->setChecked(false);
    ui->rb_useErode_3->setChecked(true);
    ui->rb_useOpen_3->setChecked(false);
    break;
  case Vision::MorphologicalFilter::DILATE:
    ui->rb_useClose_3->setChecked(false);
    ui->rb_useDilate_3->setChecked(true);
    ui->rb_useErode_3->setChecked(false);
    ui->rb_useOpen_3->setChecked(false);
    break;
  }

  ui->sb_adaptContrastFactor_3->setValue(Settings->adaptContrastKernelFactor);
  ui->sb_adaptContrKernel_3->setValue(Settings->adaptContrastKernelSize);
  ui->sb_blurMask_3->setValue(Settings->blurKernelSize);
  ui->sb_morphMask_3->setValue(Settings->filterMaskSize);
```

```
  ui->sb_sigmaFactor_3->setValue(Settings->sigmaFactor);

  // Setup the neural Network tab
  NN = nn;
  QPixmap NNpix("Images/feedforwardnetwork2.png");
  ui->label_NNimage->setPixmap(NNpix);
  ui->label_NNimage->setScaledContents(true);

  ui->spinBox_InputNeurons->setValue(NN->GetInputNeurons());
  ui->spinBox_HiddenNeurons->setValue(NN->GetHiddenNeurons()
      );
  ui->spinBox_OutputNeurons->setValue(NN->GetOutputNeurons()
      );
  ui->spinBox_Elitisme->setValue(NN->ElitismeUsedByGA);
  ui->spinBox_MaxGen->setValue(NN->MaxGenUsedByGA);
  ui->spinBox_PopSize->setValue(NN->PopulationSizeUsedByGA);
  ui->doubleSpinBox_endError->setValue(NN->EndErrorUsedByGA)
      ;
  ui->doubleSpinBox_MutationRate->setValue(NN->
      MutationrateUsedByGA);
  ui->doubleSpinBox_Beta->setValue(NN->GetBeta());
  ui->doubleSpinBox_maxWeight->setValue(NN->
      MaxWeightUsedByGA);
  ui->doubleSpinBox_MinWeight->setValue(NN->
      MinWeightUSedByGa);
  ui->checkBox_PredictShape->setChecked(Settings->
      PredictTheShape);
  ui->checkBox_revolt->setChecked(Settings->Revolution);

  // Setup the preference tab
  ui->lineEdit_NeuralNetFolder->setText(
       QString::fromStdString(Settings->NNFolder));
  ui->lineEdit_Printer->setText(
       QString::fromStdString(Settings->StandardPrinter));
  ui->lineEdit_Samplefolder->setText(
       QString::fromStdString(Settings->SampleFolder));
  ui->lineEdit_SendTo->setText(
       (QString::fromStdString(Settings->StandardSentTo)));
  ui->lineEdit_SettingFolder->setText(
       QString::fromStdString(Settings->SettingsFolder));
  ui->lineEdit__NeuralNet->setText(
       QString::fromStdString(Settings->NNlocation));

  if (openNN) {
    ui->tabWidget->setCurrentIndex(3);
  }
  initfase = false;
}

DialogSettings::~DialogSettings() { delete ui; }

void DialogSettings::openTab(int newValue) {
  if (newValue > ui->tabWidget->count()) {
    ui->tabWidget->setCurrentIndex(newValue);
  }
}
```

```

void DialogSettings::on_pushButton_RestoreDefault_clicked()
    {
  Settings->LoadSettings("Settings/Default.ini");
}

void DialogSettings::on_pushButton_Open_clicked() {
  QString fn = QFileDialog::getOpenFileName(
      this, tr("Open Settings"), QDir::homePath(), tr("
          Settings (*.ini)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".ini"))) {
      fn.append(tr(".ini"));
    }
    Settings->LoadSettings(fn.toStdString());
  }
}

void DialogSettings::on_pushButton_Save_clicked() {
  QString fn = QFileDialog::getSaveFileName(
      this, tr("Save Settings"), QDir::homePath(), tr("
          Settings (*.ini)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".ini"))) {
      fn.append(tr(".ini"));
    }
    Settings->SaveSettings(fn.toStdString());
  }
}

void DialogSettings::on_checkBox_Backlight_clicked(bool
    checked) {
  ui->tabWidget_Hardware->setTabEnabled(2, checked);
  Settings->useBacklightProjection = checked;
}

void DialogSettings::
    on_comboBox_Microscopes_currentIndexChanged(
      const QString &arg1) {

  if (!initfase) {
    std::string selectedCam = arg1.toStdString();
    Microscope->openCam(selectedCam);
    Settings->defaultWebcam = selectedCam;

    ui->comboBox_Resolution->clear();
    QStringList Resolutions;
    for (uint32_t i = 0; i < Microscope->SelectedCam->
        Resolutions.size(); i++) {
      Resolutions
          << Microscope->SelectedCam->Resolutions[i].
              to_string().c_str();
    }
    ui->comboBox_Resolution->addItems(Resolutions);
    ui->comboBox_Resolution->setCurrentIndex(
        Microscope->SelectedCam->SelectedResolution->ID);
```

```
  }
}

void DialogSettings::on_comboBox_Resolution_currentIndexChanged(int index) {
  if (!initfase) {
    Microscope->SelectedCam->SelectedResolution = &Microscope->SelectedCam->Resolutions[index];
    Settings->selectedResolution = index;
  }
}

void DialogSettings::on_checkBox_useHDR_clicked(bool checked) {
  ui->spinBox_NoFrames->setDisabled(!checked);
  ui->label_nf->setDisabled(!checked);
  Settings->useHDR = checked;
}

void DialogSettings::SetCamControl(Hardware::Microscope::Cam_t *selectedCam,
                                   QSlider *Brightness, QSlider *Contrast,
                                   QSlider *Saturation, QSlider *Hue,
                                   QSlider *Sharpness) {
  for (uint32_t i = 0; i < selectedCam->Controls.size(); i++) {
    if (selectedCam->Controls[i].name.compare("Brightness") == 0) {
      Brightness->setMinimum(selectedCam->Controls[i].minimum);
      Brightness->setMaximum(selectedCam->Controls[i].maximum);
    } else if (selectedCam->Controls[i].name.compare("Contrast") == 0) {
      Contrast->setMinimum(selectedCam->Controls[i].minimum);
      Contrast->setMaximum(selectedCam->Controls[i].maximum);
    } else if (selectedCam->Controls[i].name.compare("Saturation") == 0) {
      Saturation->setMinimum(selectedCam->Controls[i].minimum);
      Saturation->setMaximum(selectedCam->Controls[i].maximum);
    } else if (selectedCam->Controls[i].name.compare("Hue") == 0) {
      Hue->setMinimum(selectedCam->Controls[i].minimum);
      Hue->setMaximum(selectedCam->Controls[i].maximum);
    } else if (selectedCam->Controls[i].name.compare("Sharpness") == 0) {
      Sharpness->setMinimum(selectedCam->Controls[i].minimum);
      Sharpness->setMaximum(selectedCam->Controls[i].maximum);
```

```
    }
  }
}

void DialogSettings::on_spinBox_NoFrames_editingFinished() {
  Settings->HDRframes = ui->spinBox_NoFrames->value();
}

void DialogSettings::on_doubleSpinBox_LightLevel_editingFinished() {
  Settings->lightLevel =
      static_cast<float>(ui->doubleSpinBox_LightLevel->value());
}

void DialogSettings::on_checkBox_useRainbow_clicked(bool checked) {
  Settings->enableRainbow = checked;
}

void DialogSettings::on_checkBox_InvertEncoder_clicked(bool checked) {
  Settings->encInv = checked;
}

void DialogSettings::on_checkBox_useCUDA_clicked(bool checked) {
  Settings->useCUDA = checked;
}

void DialogSettings::on_horizontalSlider_BrightFront_valueChanged(int value) {
  if (!initfase) {
    Settings->Brightness_front = value;
  }
}

void DialogSettings::on_horizontalSlider_ContrastFront_valueChanged(int value) {
  if (!initfase) {
    Settings->Contrast_front = value;
  }
}

void DialogSettings::on_horizontalSlider_SaturationFront_valueChanged(int value) {
  if (!initfase) {
    Settings->Saturation_front = value;
  }
}

void DialogSettings::on_horizontalSlider_HueFront_valueChanged(int value) {
  if (!initfase) {
```

```
    Settings->Hue_front = value;
  }
}

void DialogSettings::
    on_horizontalSlider_SharpnessFront_valueChanged(
    int value) {
  if (!initfase) {
    Settings->Sharpness_front = value;
  }
}

void DialogSettings::
    on_horizontalSlider_BrightProj_valueChanged(int value) {
  if (!initfase) {
    Settings->Brightness_proj = value;
  }
}

void DialogSettings::
    on_horizontalSlider_ContrastProj_valueChanged(int value)
    {
  if (!initfase) {
    Settings->Contrast_proj = value;
  }
}

void DialogSettings::
    on_horizontalSlider_SaturationProj_valueChanged(
    int value) {
  if (!initfase) {
    Settings->Saturation_proj = value;
  }
}

void DialogSettings::
    on_horizontalSlider_HueProj_valueChanged(int value) {
  if (!initfase) {
    Settings->Hue_proj = value;
  }
}

void DialogSettings::
    on_horizontalSlider_SharpnessProj_valueChanged(int value)
    {
  if (!initfase) {
    Settings->Sharpness_proj = value;
  }
}

void DialogSettings::on_cb_use_adaptContrast_3_clicked(bool
    checked) {
  Settings->useAdaptiveContrast = checked;
  ui->sb_adaptContrastFactor_3->setDisabled(!checked);
  ui->sb_adaptContrKernel_3->setDisabled(!checked);
}
```

```

void DialogSettings::on_cb_useBlur_3_clicked(bool checked) {
  Settings->useBlur = checked;
  ui->sb_blurMask_3->setDisabled(!checked);
}

void DialogSettings::on_rb_useDark_3_toggled(bool checked) {
  if (checked) {
    Settings->typeOfObjectsSegmented = Vision::Segment::Dark;
  } else {
    Settings->typeOfObjectsSegmented = Vision::Segment::Bright;
  }
}

void DialogSettings::on_cb_ignoreBorder_3_clicked(bool checked) {
  Settings->ignorePartialBorderParticles = checked;
}

void DialogSettings::on_cb_fillHoles_3_clicked(bool checked) {
  Settings->fillHoles = checked;
}

void DialogSettings::on_sb_sigmaFactor_3_editingFinished() {
  Settings->sigmaFactor = ui->sb_sigmaFactor_3->value();
}

void DialogSettings::on_rb_useOpen_3_clicked(bool checked) {
  Settings->morphFilterType = Vision::MorphologicalFilter::OPEN;
}

void DialogSettings::on_rb_useClose_3_clicked(bool checked) {
  Settings->morphFilterType = Vision::MorphologicalFilter::CLOSE;
}

void DialogSettings::on_rb_useErode_3_clicked(bool checked) {
  Settings->morphFilterType = Vision::MorphologicalFilter::ERODE;
}

void DialogSettings::on_rb_useDilate_3_clicked(bool checked) {
  Settings->morphFilterType = Vision::MorphologicalFilter::DILATE;
}

void DialogSettings::on_sb_morphMask_3_editingFinished() {
  Settings->filterMaskSize = ui->sb_morphMask_3->value();
}
```

```

void DialogSettings::on_spinBox_MaxGen_editingFinished() {
  NN->MaxGenUsedByGA = ui->spinBox_MaxGen->value();
}

void DialogSettings::on_spinBox_PopSize_editingFinished() {
  NN->PopulationSizeUsedByGA = ui->spinBox_PopSize->value();
}

void DialogSettings::on_doubleSpinBox_MutationRate_editingFinished() {
  NN->MutationrateUsedByGA = ui->doubleSpinBox_MutationRate->value();
}

void DialogSettings::on_spinBox_Elitisme_editingFinished() {
  NN->ElitismeUsedByGA = ui->spinBox_Elitisme->value();
}

void DialogSettings::on_doubleSpinBox_endError_editingFinished() {
  NN->EndErrorUsedByGA = ui->doubleSpinBox_endError->value();
}

void DialogSettings::on_doubleSpinBox_maxWeight_editingFinished() {
  NN->MaxWeightUsedByGA = ui->doubleSpinBox_maxWeight->value();
}

void DialogSettings::on_doubleSpinBox_MinWeight_editingFinished() {
  NN->MinWeightUSedByGa = ui->doubleSpinBox_MinWeight->value();
}

void DialogSettings::on_doubleSpinBox_Beta_editingFinished() {
  NN->SetBeta(ui->doubleSpinBox_Beta->value());
}

void DialogSettings::on_spinBox_InputNeurons_editingFinished() {
  NN->SetInputNeurons(ui->spinBox_InputNeurons->value());
}

void DialogSettings::on_spinBox_HiddenNeurons_editingFinished() {
  NN->SetHiddenNeurons(ui->spinBox_HiddenNeurons->value());
}

void DialogSettings::on_spinBox_OutputNeurons_editingFinished() {
  NN->SetOutputNeurons(ui->spinBox_OutputNeurons->value());
}
```

```

void DialogSettings::
    on_pushButton_selectSampleFolder_clicked() {
  QString fn = QFileDialog::getExistingDirectory(
      this, tr("Select the Sample Directory"),
      QString::fromStdString(Settings->SampleFolder),
      QFileDialog::ShowDirsOnly | QFileDialog::
          DontResolveSymlinks);
  if (!fn.isEmpty()) {
    ui->lineEdit_Samplefolder->setText(fn);
    Settings->SampleFolder = fn.toStdString();
  }
}

void DialogSettings::
    on_pushButton_SelectSettingFolder_clicked() {
  QString fn = QFileDialog::getExistingDirectory(
      this, tr("Select the Setting Directory"),
      QString::fromStdString(Settings->SettingsFolder),
      QFileDialog::ShowDirsOnly | QFileDialog::
          DontResolveSymlinks);
  if (!fn.isEmpty()) {
    ui->lineEdit_SettingFolder->setText(fn);
    Settings->SettingsFolder = fn.toStdString();
  }
}

void DialogSettings::on_pushButton_SelectNNFolder_clicked()
    {
  QString fn = QFileDialog::getExistingDirectory(
      this, tr("Select the NeuralNet Directory"),
      QString::fromStdString(Settings->NNFolder),
      QFileDialog::ShowDirsOnly | QFileDialog::
          DontResolveSymlinks);
  if (!fn.isEmpty()) {
    ui->lineEdit_NeuralNetFolder->setText(fn);
    Settings->NNFolder = fn.toStdString();
  }
}

void DialogSettings::on_pushButton_SelectNN_clicked() {
  QString fn =
      QFileDialog::getOpenFileName(this, tr("Select the
          standard Neural Net"),
                                   QDir::homePath(), tr("
                                       NeuralNet (*.NN)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".NN"))) {
      fn.append(tr(".NN"));
    }
    Settings->NNlocation = fn.toStdString();
    ui->lineEdit__NeuralNet->setText(fn);
  }
}

void DialogSettings::on_spinBox_NoShots_editingFinished() {
```

```
  Settings->StandardNumberOfShots = ui->spinBox_NoShots->
     value();
}

void DialogSettings::on_checkBox_PredictShape_clicked(bool
   checked) {
  Settings->PredictTheShape = checked;
}

void DialogSettings::on_checkBox_revolt_clicked(bool checked
   )
{
    Settings->Revolution = checked;
}
```

K.0.4 Dialog Neural Network Class

```
#ifndef DIALOGNN_H
#define DIALOGNN_H

#include <QDialog>
#include "SoilMath.h"
#include "soilanalyzer.h"
#include "dialogsettings.h"
#include <qcustomplot.h>
#include <QDebug>

namespace Ui {
  class DialogNN;
}

class DialogNN : public QDialog
{
  Q_OBJECT

public:
  explicit DialogNN(QWidget *parent = 0, SoilMath::NN *
     neuralnet = nullptr, SoilAnalyzer::SoilSettings *
     settings = nullptr, DialogSettings *settingWindow =
     nullptr);
  ~DialogNN();

private slots:

  void on_pushButton_Settings_clicked();

  void on_learnErrorUpdate(double newError);

  void on_pushButton_SelectSamples_clicked();

  void on_pushButton_Learn_clicked();

  void on_pushButton_SaveNN_clicked();

  void on_pushButton_OpenNN_clicked();

  void on_actionAbort_triggered();

private:
  Ui::DialogNN *ui;
  DialogSettings *SettingsWindow = nullptr;
  SoilMath::NN *NeuralNet = nullptr;
  SoilAnalyzer::SoilSettings *Settings = nullptr;

  void setupErrorGraph();
  void makeLearnVectors(InputLearnVector_t &input,
     OutputLearnVector_t &output);

  QVector<double> currentError;
  QVector<double> errorTicks;
  double currentGeneration = 0;
```

```
  QStringList fn;
};

#endif // DIALOGNN_H
```

```
#include "dialognn.h"
#include "ui_dialognn.h"

DialogNN::DialogNN(QWidget *parent, SoilMath::NN *neuralnet,
                   SoilAnalyzer::SoilSettings *settings,
                   DialogSettings *settingsWindow)
    : QDialog(parent), ui(new Ui::DialogNN) {
  ui->setupUi(this);

  if (neuralnet == nullptr) {
    neuralnet = new SoilMath::NN;
  }
  NeuralNet = neuralnet;
  if (settings == nullptr) {
    settings = new SoilAnalyzer::SoilSettings;
  }
  Settings = settings;
  if (settingsWindow == nullptr) {
    settingsWindow = new DialogSettings;
  }
  SettingsWindow = settingsWindow;

  // Setup the Qplots
  ui->widget_NNError->addGraph();
  ui->widget_NNError->addGraph();

  ui->widget_NNError->xAxis->setLabel("Generation [-]");
  ui->widget_NNError->yAxis->setLabel("Error [%]");
  QCPPlotTitle *widget_NNErrorTitle = new QCPPlotTitle(ui->
      widget_NNError);
  widget_NNErrorTitle->setText("Learning error");
  widget_NNErrorTitle->setFont(QFont("sans", 10, QFont::Bold
      ));
  ui->widget_NNError->plotLayout()->insertRow(0);
  ui->widget_NNError->plotLayout()->addElement(0, 0,
      widget_NNErrorTitle);

  setupErrorGraph();

  // Connect the NN learn error
  connect(NeuralNet, SIGNAL(learnErrorUpdate(double)), this,
          SLOT(on_learnErrorUpdate(double)));
}

DialogNN::~DialogNN() { delete ui; }

void DialogNN::on_pushButton_Settings_clicked() {
  SettingsWindow->openTab(2);
  SettingsWindow->show();
  setupErrorGraph();
}
```

```

void DialogNN::on_learnErrorUpdate(double newError) {
  ui->widget_NNError->graph(0)->addData(currentGeneration,
      newError);
  currentGeneration += 1;
  ui->widget_NNError->yAxis->rescale();
  //ui->widget_NNError->yAxis->setRange(0, 20);
  ui->widget_NNError->replot();
}

void DialogNN::setupErrorGraph() {
  errorTicks.clear();
  for (uint32_t i = 0; i < NeuralNet->MaxGenUsedByGA; i++) {
    errorTicks.push_back(i);
  }
  ui->widget_NNError->xAxis->setRange(0, NeuralNet->
      MaxGenUsedByGA);
  QVector<double> endErrorValue(2, NeuralNet->
      EndErrorUsedByGA);
  QVector<double> endErrorKey(2, 0);
  endErrorKey[1] = NeuralNet->MaxGenUsedByGA;
  ui->widget_NNError->graph(1)->setData(endErrorKey,
      endErrorValue);
  ui->widget_NNError->xAxis->setAutoTicks(false);
  ui->widget_NNError->xAxis->setTickVector(errorTicks);
  ui->widget_NNError->xAxis->setTickLabels(false);
  //ui->widget_NNError->yAxis->setScaleType(QCPAxis::
      stLogarithmic);
  ui->widget_NNError->replot();
}

void DialogNN::on_pushButton_SelectSamples_clicked() {
  fn = QFileDialog::getOpenFileNames(
      this, tr("Open Samples"), QString::fromStdString(
          Settings->SampleFolder),
      tr("Samples (*.VSA)"));
  for_each(fn.begin(), fn.end(), [](QString &f) {
    if (!f.contains(tr(".VSA"))) {
      f.append(tr(".VSA"));
    }
  });
}

void DialogNN::on_pushButton_Learn_clicked() {
  if (fn.size() < 1) {
    return;
  }
  InputLearnVector_t InputVec;
  OutputLearnVector_t OutputVec;
  makeLearnVectors(InputVec, OutputVec);
  NeuralNet->Learn(InputVec, OutputVec, NeuralNet->
      GetInputNeurons());
  setupErrorGraph();
}

void DialogNN::makeLearnVectors(InputLearnVector_t &input,
```

```
                                OutputLearnVector_t &output)
                                    {
  for (uint32_t i = 0; i < fn.size(); i++) {
    SoilAnalyzer::Sample sample;
    sample.Load(fn[i].toStdString());
    for_each(sample.ParticlePopulation.begin(), sample.
        ParticlePopulation.end(),
             [&](SoilAnalyzer::Particle &P) {
               if (P.FFDescriptors.size() >= NeuralNet->
                   GetInputNeurons()) {
                 ComplexVect_t ffdesc;
                 for (uint32_t j = 0; j < NeuralNet->
                     GetInputNeurons(); j++) {
                   ffdesc.push_back(P.FFDescriptors[j]);
                 }
                 input.push_back(ffdesc);
                 Predict_t predict = P.Classification;
                 predict.OutputNeurons = SoilMath::
                     makeOutput(P.GetAngularity(), NeuralNet
                     ->GetOutputNeurons());
                 output.push_back(predict);
               }
             });
  }
}

void DialogNN::on_pushButton_SaveNN_clicked() {
  QString fn = QFileDialog::getSaveFileName(
      this, tr("Save NeuralNet"), QString::fromStdString(
          Settings->NNFolder),
      tr("NeuralNet (*.NN)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".NN"))) {
      fn.append(tr(".NN"));
    }
    NeuralNet->SaveState(fn.toStdString());
  }
}

void DialogNN::on_pushButton_OpenNN_clicked() {
  QString fn = QFileDialog::getOpenFileName(
      this, tr("Open NeuralNet"),
      QString::fromStdString(Settings->SampleFolder), tr("
          NeuralNet (*.NN)"));
  if (!fn.isEmpty()) {
    if (!fn.contains(tr(".NN"))) {
      fn.append(tr(".NN"));
    }
    if (NeuralNet != nullptr) {
      delete NeuralNet;
    }
    NeuralNet->LoadState(fn.toStdString());
    connect(NeuralNet, SIGNAL(learnErrorUpdate(double)),
        this,
            SLOT(on_learnErrorUpdate(double)));
  }
```

```
}

void DialogNN::on_actionAbort_triggered()
{
    NeuralNet->EndErrorUsedByGA = ui->widget_NNError->graph
        (0)->data()->lastKey();
}
```